Software Requirements Specification

for

Stock Inventory Management

Version 1.0 approved

Prepared by

Ranjith T
Surendhar
Jaswin Kumar N R
Mohammed Thalha
Navamohan M

16.08.2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The Stock Inventory Management System is designed to streamline the process of tracking and managing inventory across various locations, ensuring accurate stock levels, minimizing losses, and optimizing supply chain operations. The system will enable real-time monitoring of stock quantities, automate reordering processes, and provide detailed reporting on stock movement, usage, and trends. It is intended to reduce human error, enhance operational efficiency, and support decision-making by providing actionable insights into inventory data. Additionally, this system will also offer an integrated billing service, which will automatically manage the stock database when items are sold. The system will cater to the needs of businesses of all sizes, offering scalable solutions that can be customized according to specific industry requirements.

1.2 Document Conventions

The standard document font is Times New Roman, with a font size of 12pt. Subheadings are emboldened with a font size of 14pt, while section headings are emboldened with a font size of 18pt. Lists are depicted with ordered or unordered bullet points depending on priority of the list items. Other important keywords are also emboldened.

1.3 Intended Audience and Reading Suggestions

The intended audience for this Software Requirements Specification (SRS) document includes project stakeholders, software developers, quality assurance teams, system analysts, and end-users who are involved in or impacted by the development and deployment of the Stock Inventory Management System. Stakeholders will gain an understanding of the system’s capabilities, while developers and analysts will use the document as a guide for implementation and design. Quality assurance teams will reference the SRS to ensure the system meets the specified requirements, and end-users will find it useful for understanding the system's functionality and benefits. It is recommended that readers familiarize themselves with the business processes related to inventory management, as well as basic concepts of software development and system architecture, to fully comprehend the contents of this document.

1.4 Product Scope

The primary purpose of the system is to automate and streamline inventory processes, including stock level monitoring, order management, and reporting. By reducing manual errors, and improving real-time visibility into inventory status, the system aims to minimize overstocking and stockouts, thereby reducing costs and improving customer satisfaction. The system aligns with corporate goals of operational excellence and cost efficiency, contributing to better resource management and supporting strategic business growth.

1.5 References

Bootstrap Documentation: https://getbootstrap.com/docs/5.3/getting-started/introduction/
Firebase Documentation: https://firebase.google.com/docs
Ruby on Rails: https://rubyonrails.org
Existing Inventory Management System Reference: https://www.zoho.com/in/inventory/free-inventory-management-software/

2. Overall Description

2.1 Product Perspective

The Stock Inventory Management System is designed as an automated solution for managing shop inventory and integrating billing processes. This system will replace the traditional manual methods of tracking inventory and generating bills, offering a more efficient, secure, and user-friendly approach to managing stock levels, sales data, and customer billing. The system is envisioned as a web-based application, accessible from any modern desktop with an internet connection, and utilizes a cloud-based backend for real-time data synchronization.

2.2 Product Functions

The system provides several core functions:

- **Stock In:** Allows users to add new items to the inventory by updating the database with item details.
- **Stock Out:** Tracks the sale of items, updating inventory data and providing details of the transactions.
- **Sales Charts:** Visualizes sales data over specific periods to help users track sales trends and profits.
- **Billing:** Generates customer bills and invoices, automatically updating the inventory after sales are completed.
- **Authentication:** Ensures secure access for different user roles, such as cashiers and managers, with appropriate access permissions.

2.3 User Classes and Characteristics

The Stock Inventory Management System is designed to be used by multiple user classes, each with distinct characteristics and needs:

i. Administrators:

Description: Administrators have the highest level of access within the system, responsible for overall system management, user permissions, and configuration settings.

Characteristics: Typically, they possess a strong technical background and an in-depth understanding of the system's functionalities. Administrators are the most frequent users, with responsibilities including user management, system customization, and overseeing security protocols.

Importance: Critical, as they maintain the system's integrity and ensure that it operates smoothly.

ii. Inventory Managers:

Description: Inventory managers are responsible for overseeing stock levels, managing reorder points, and ensuring that inventory data is accurate and up to date.

Characteristics: They have a deep understanding of inventory processes and are proficient in using the system's core features such as stock level monitoring, reporting, and order management. They use the system regularly to perform day-to-day tasks.

Importance: High, as they directly impact the efficiency and effectiveness of inventory management within the organization.

iii. Warehouse Staff:

Description: Warehouse staff interact with the system to update inventory records, process shipments, and manage physical stock movements.

Characteristics: These users may have varying levels of technical expertise. The system's interface for this class should be intuitive and easy to use, focusing on simplifying tasks such as scanning barcodes, updating stock levels, and processing orders.

Importance: Medium, as their accurate data input is essential for maintaining up-to-date inventory records.

iv. Procurement Officers:

Description: Procurement officers use the system to manage supplier relationships, place orders, and track order statuses.

Characteristics: They require access to reporting and analysis features to make informed purchasing decisions. Their technical expertise varies, so the system should offer easy-to-understand dashboards and reports.

Importance: High, as they ensure that the organization maintains optimal stock levels and avoids overstocking or stockouts.

v. Sales Personnel:

Description: Sales personnel may use the system to check stock availability, track product movement, and generate sales reports.

Characteristics: Typically, they have a moderate level of technical expertise. They need quick access to inventory data to respond to customer inquiries and manage sales orders effectively.

Importance: Medium, as their interaction with the system supports customer satisfaction and sales performance.

vi. Executives/Management:

Description: Executives and management use the system to gain insights into inventory performance, trends, and overall efficiency.

Characteristics: They require high-level, summarized reports and analytics rather than detailed operational features. Their technical expertise is often focused on data interpretation rather than system operation.

Importance: High, as their decisions, based on the insights provided by the system, shape the strategic direction of the organization.

2.4 Operating Environment

The system is designed to operate on any modern desktop system with at least a Quad-Core Intel / AMD processor, 4GB of RAM, and an internet connection. The backend system will be hosted on a cloud service, such as Firebase, ensuring that the system is accessible from anywhere and can scale according to demand.

2.5 Design and Implementation Constraints

Frontend Framework: The user interface will be developed using Bootstrap 5, ensuring responsive and mobile-friendly design.

Backend Framework: Ruby on Rails will be used for the backend, facilitating a dynamic and efficient web application development process.

Database: The system will use Firebase's Cloud Firestore for inventory data and Firebase Realtime Database for transaction details, ensuring real-time updates.

Security: The application relies on Firebase Authentication to manage user access securely, with different privileges assigned to different user roles.

2.6 User Documentation

Comprehensive documentation will be provided to users, detailing the following:

- System Installation and Setup: Step-by-step instructions for installing and configuring the application.

- User Guide: Detailed instructions on how to use each function of the system, including screenshots and examples.

- Troubleshooting Guide: Common issues and their solutions to help users resolve problems independently.

2.7 Assumptions and Dependencies

- The system assumes that users have a stable internet connection for accessing the cloud-hosted application.

- The application depends on the availability of Firebase services for database management and authentication.

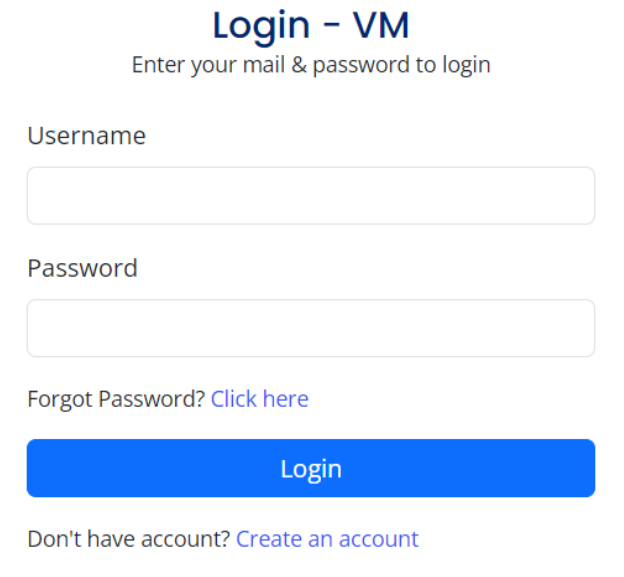
- The system assumes that the hardware requirements are met by the users for optimal performance.

- Dependencies also include the specific versions of Bootstrap and Ruby on Rails mentioned, ensuring compatibility and access to the necessary features.

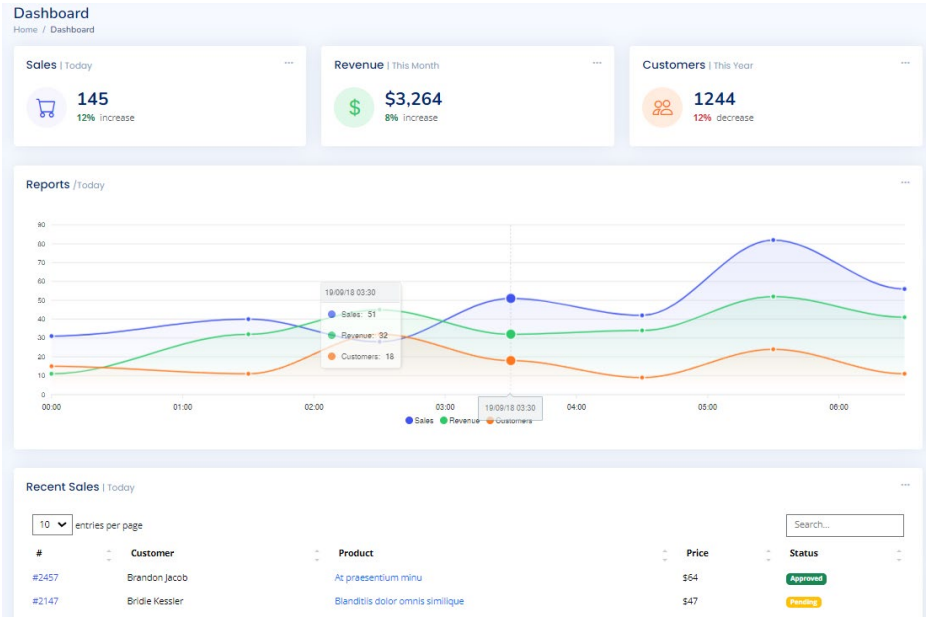
3. External Interface Requirements

3.1 User Interfaces

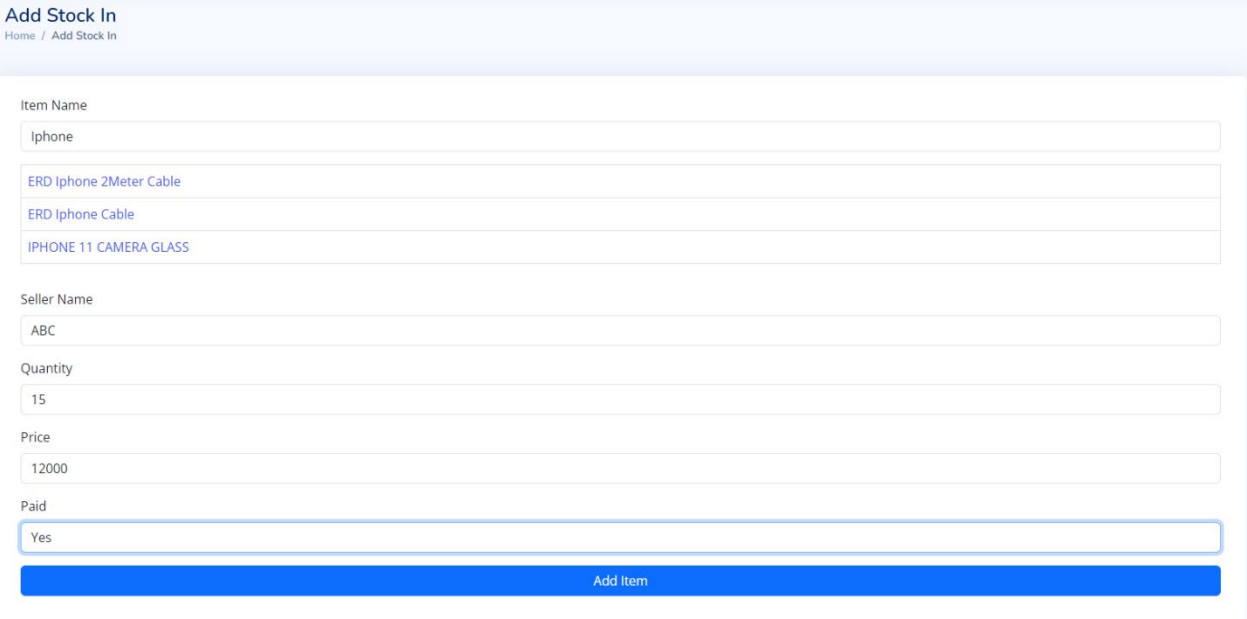
1. The Login Page: The first step will be authentication. The login system will authenticate the user who is logging in to the app. Be it the manager, cashier, or a procurement officer, they can login with their account and be granted access to the services they specifically require.



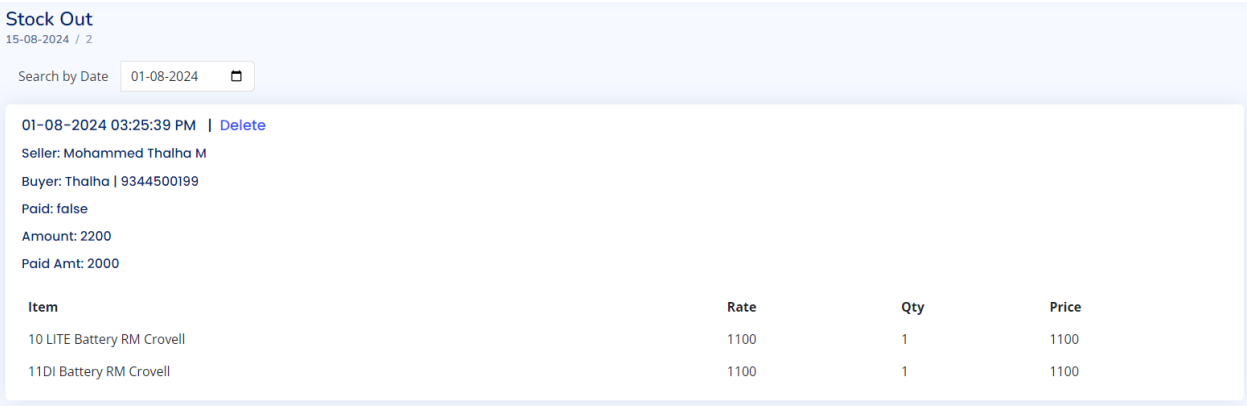
2. The Dashboard: The Sales Dashboard in the Stock Inventory Management System will focus on providing a detailed overview of the number of items sold across various categories and time periods. It will display key metrics such as total units sold, best-selling products, and sales volume by product category, region, and sales channel. The dashboard will feature visualizations like bar charts and line graphs to track the quantity of items sold over time, highlighting trends and fluctuations in sales performance. Users can drill down into the data to see the number of units sold for specific products or compare sales figures across different periods. This dashboard will be an essential tool for sales personnel, managers, and executives to monitor product movement, identify high-demand items, and make data-driven decisions to optimize inventory and sales strategies.



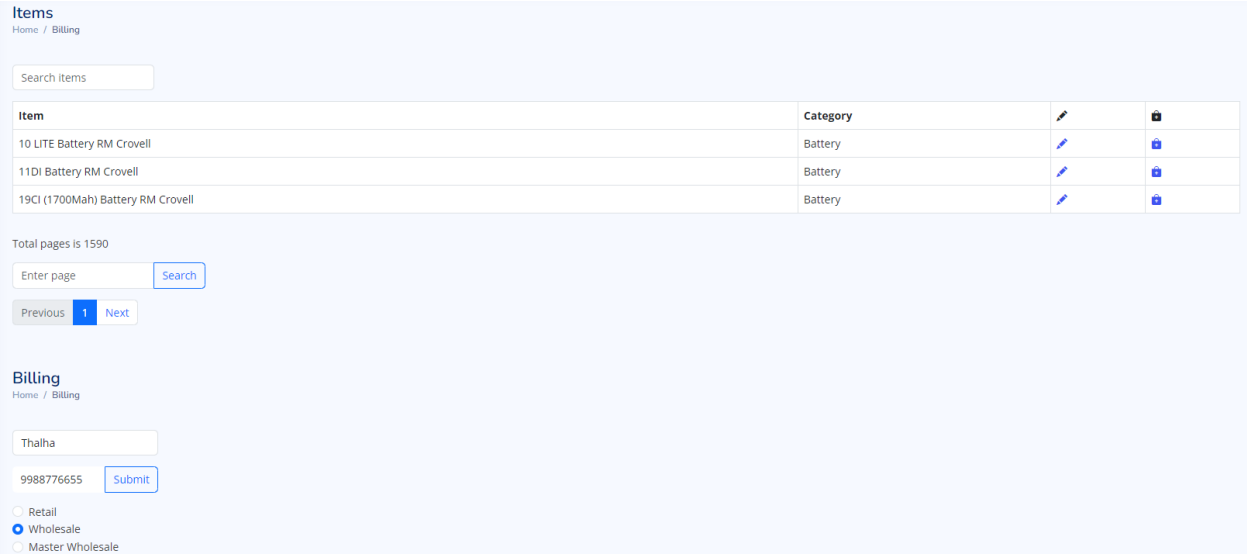
3. Item Management Interfaces: For example, the Stock - In will show the items added to the inventory by date. It will also provide an interface for adding new items to the inventory. The interface will be user friendly and convenient for anyone to work with.

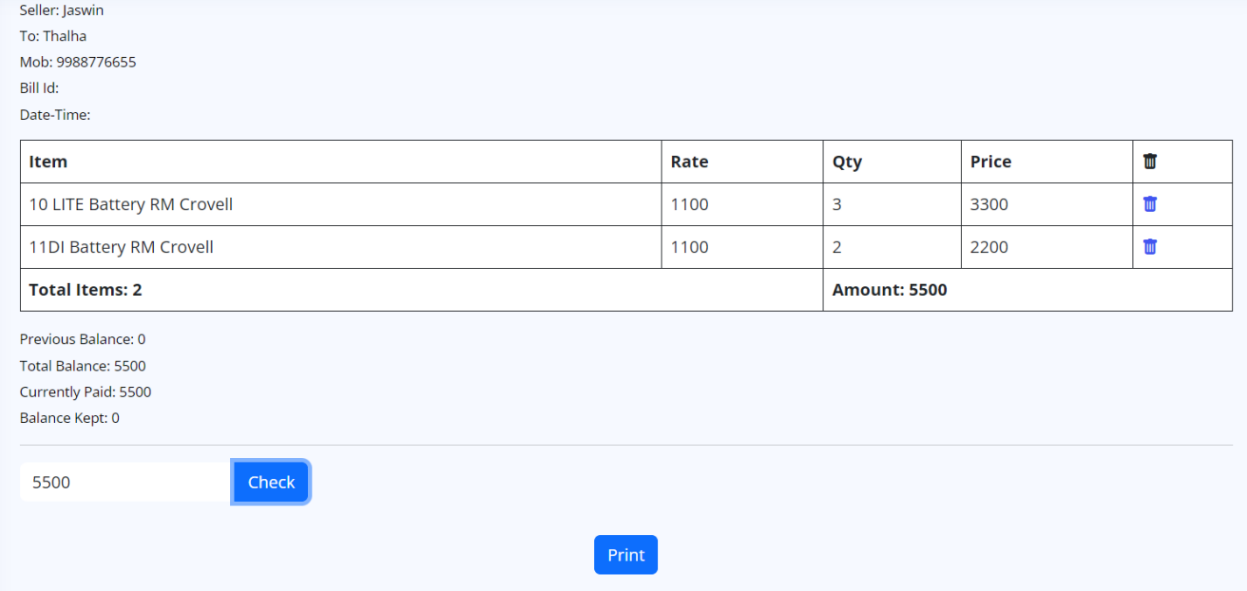


4. Detailed Cards: Pages such as the Stock – Out page will provide details by date about the items sold, along with the customer details and billing information. It will help track the items which were taken out of the inventory by sales.

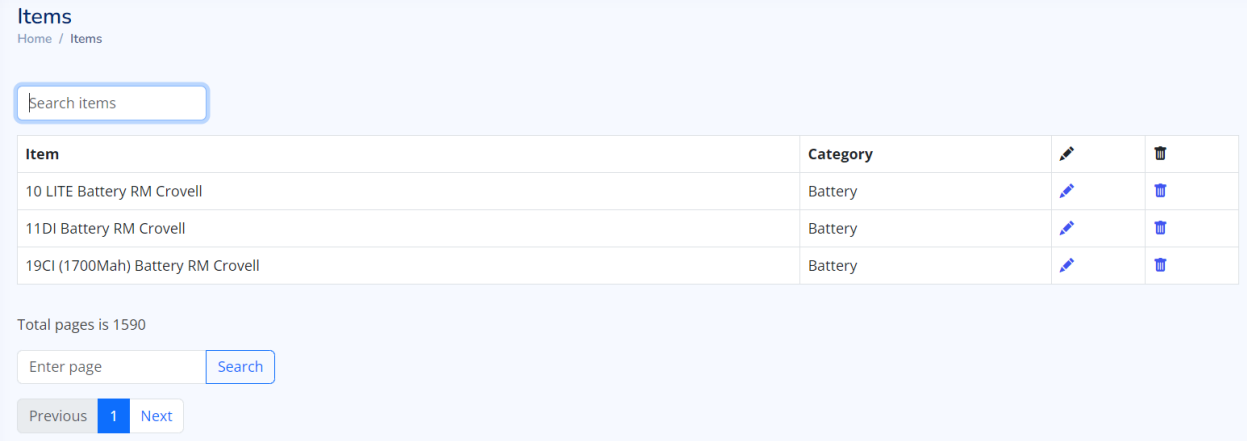


5. Billing Interface: The billing section will provide an interface to add items to the bill, enter customer details and print the bill. This will be an easily accessible interface for the cashier / salesperson who will handle sale of items while also having to update the stock out database, which will be done automatically.

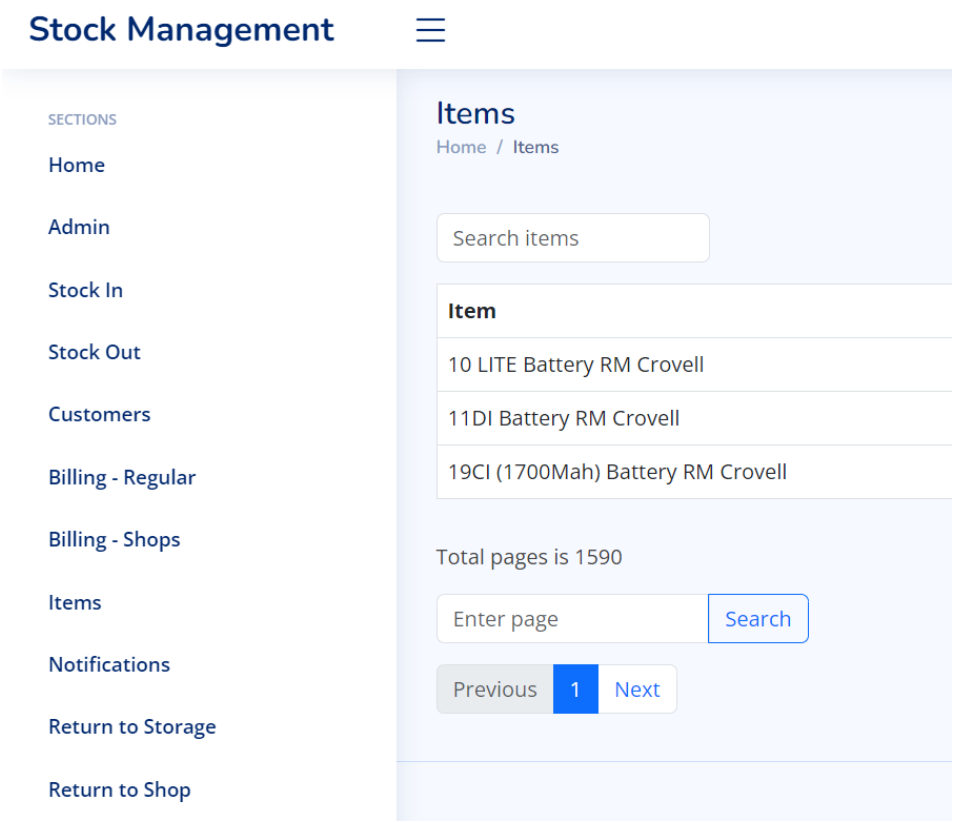




6. The Items List: This provides a list of all items in the inventory in a tabulated and paginated format. It will also provide a search function which can filter the required items to make finding items easier. This page helps keep track of the total available items in the store.



7. The Navigation Menu: It provides an interface to navigate to different parts of the application. It will be easily accessible by admins and users alike. It should be the prominent feature of every page of the application.



3.2 Hardware Interfaces

The Stock Inventory Management System will interface with various hardware components to facilitate

seamless data collection, processing, and control operations. The key interfaces include printers, and computing devices such as desktop computers, tablets, and mobile phones.

1. Printers:

Logical Characteristics: The system will generate labels, invoices, and reports, which will be sent to printers for physical output. The software will format the print data according to the specific requirements of the document type, ensuring compatibility with various printer models.

Physical Characteristics: Printers will interface with the system via USB, Ethernet, or Wi-Fi connections. The system will support standard printing protocols such as IPP (Internet Printing Protocol) and LPR/LPD (Line Printer Remote/Daemon) to ensure compatibility with a wide range of printers, including thermal label printers and standard office printers.

2. Computing Devices (Desktops, Tablets, Mobile Phones):

Logical Characteristics: The system will be accessible from various computing devices, offering a responsive interface that adapts to different screen sizes and input methods (mouse, keyboard, touch). Data interactions include user inputs for inventory management tasks, retrieval of reports, and real-time updates to inventory records.

Physical Characteristics: These devices will connect to the system through wired (Ethernet) or wireless (Wi-Fi, cellular data) networks. The system will utilize standard communication protocols such as HTTP/HTTPS for web-based interactions and APIs to facilitate data exchange. The software will be compatible with major operating systems like Windows, macOS, iOS, and Android, ensuring broad accessibility across different device types.

The integration of these hardware components with the Stock Inventory Management System will be designed to ensure reliable, efficient, and secure data flow, enabling the system to function effectively within the operational environment.

3.3 Software Interfaces

The Stock Inventory Management System will interface with several software components to deliver a seamless, integrated solution. These components include the frontend framework, backend environment, databases, and authentication services.

1. Frontend (Bootstrap 5):

Connection: The frontend is built using Bootstrap 5, providing a responsive and user-friendly interface. It communicates with the backend via RESTful APIs, transmitting user inputs, such as inventory updates, queries, and form submissions, to the server.

Data Flow: Data items include user actions, such as adding or removing stock, and requests for reports or analytics. The frontend receives data from the backend in JSON format, rendering it in a structured and visually appealing manner using Bootstrap components.

Purpose: The purpose of these interactions is to present users with an intuitive interface that facilitates the efficient management of inventory tasks.

2. Backend (JavaScript, Ruby on Rails):

Connection: The backend, developed with JavaScript and Ruby on Rails, serves as the core logic and processing engine of the system. It processes requests from the frontend, interacts with the database, and returns the required data or performs the necessary operations.

Data Flow: The backend receives API requests, processes them, and sends back responses. For example, when a user requests stock levels, the backend queries the Firebase databases and returns the results to the frontend.

Purpose: The backend manages business logic, data validation, and ensures secure and efficient operations within the system.

3. Database (Firebase Realtime Database and Firebase Cloud Firestore):

Connection: The system utilizes Firebase Realtime Database and Firebase Cloud Firestore for data storage and retrieval. The backend communicates with these databases via Firebase SDKs to store inventory data, user information, and transaction records.

Data Flow: Incoming data includes stock entries, user activity logs, and transaction details. Outgoing data includes reports, analytics, and real-time updates on inventory status. The system uses Firestore for structured data and the Realtime Database for real-time synchronization of inventory levels.

Purpose: These databases store and manage all inventory-related data, ensuring that the system has quick access to up-to-date information. Firestore is particularly used for more complex queries and data aggregation, while the Realtime Database supports real-time updates.

4. Authentication (Firebase Auth):

Connection: Firebase Auth handles user authentication and authorization. It integrates with the backend to manage user sign-in, sign-up, password management, and session control.

Data Flow: Data items include user credentials, authentication tokens, and session data. Upon successful authentication, Firebase Auth generates a token that is used by the backend to validate user sessions and determine access levels.

Purpose: The purpose of this component is to secure access to the system, ensuring that only authorized users can perform specific actions based on their roles.

5. Communication and Data Sharing:

Communication Protocols: The system uses HTTPS for secure communication between the frontend, backend, and Firebase services. RESTful APIs facilitate data exchange between the frontend and backend.

Shared Data: Data shared across components include user credentials, inventory records, and transaction logs. The sharing mechanism is implemented using Firebase's real-time capabilities for instant updates and Firestore for data persistence.

Implementation Constraints: Given that Firebase Auth is used for authentication, all secure operations are tied to Firebase's authentication tokens, ensuring that each request to the backend is validated. The system must also handle data consistency across the Realtime Database and Firestore, ensuring that changes in one are reflected in the other as needed.

3.4 Communications Interfaces

The Stock Inventory Management System requires several communications functions to ensure seamless interaction between users, the software, and external systems. These include web browser interactions, network communications, and secure data transmission.

1. Web Browser Communications:

Requirements: The system must support modern web browsers (e.g., Chrome, Firefox, Safari, Edge) to provide a consistent user experience across different platforms. It will use HTTP/HTTPS protocols for communication between the client-side (browser) and server-side (backend).

Message Formatting: Data exchanged between the frontend and backend will be formatted in JSON, ensuring compatibility and ease of parsing across different environments.

Communication Standards: The system will rely on HTTPS to secure communications, ensuring data integrity and confidentiality during transmission.

2. Network Server Communications:

Requirements: The backend will interact with Firebase services, requiring support for secure API calls over the internet. The system will need reliable connectivity to the Firebase Realtime Database and Firestore for data storage and retrieval.

Communication Protocols: The system will use RESTful APIs over HTTPS for secure communication with Firebase. WebSockets may be employed for real-time updates, especially when using the Firebase Realtime Database.

Data Transfer Rates: The system must handle varying data transfer rates, ensuring that large data sets are transmitted efficiently without overwhelming network resources. This is particularly important for syncing large volumes of inventory data.

3. E-mail Communications:

Requirements: The system will send automated email notifications for specific events, such as low stock alerts, order confirmations, and user registration confirmations. These emails will be sent through an SMTP server integrated with the backend.

Message Formatting: Emails will be formatted in HTML for readability and will include relevant data such as product details, order summaries, and links to the system.

Communication Standards: The system will use standard SMTP for sending emails, with SSL/TLS encryption to secure email communications.

4. Communication Security and Encryption:

Requirements: All communications, whether between the client and server or the backend and external

services (e.g., Firebase), must be encrypted to protect sensitive data. This includes using HTTPS for web traffic and secure API calls.

Encryption Standards: The system will use TLS (Transport Layer Security) for encrypting HTTPS communications, ensuring that data such as user credentials, inventory details, and transaction records are transmitted securely.

Authentication Tokens: Firebase Auth will generate and manage authentication tokens, which will be passed with each request to the backend to ensure that only authorized communications are processed.

5. Data Transfer Rates and Synchronization Mechanisms:

Requirements: The system must efficiently manage data transfer rates to ensure smooth operation, particularly during peak usage times or when handling large datasets. It will also need to synchronize data between the Realtime Database and Firestore to maintain consistency.

Synchronization Mechanisms: The system will employ Firebase's built-in synchronization capabilities, ensuring real-time updates across devices and users. For larger data transfers, the system will implement batch processing to optimize performance and reduce latency.

6. Web Forms:

Requirements: Users will interact with the system through electronic forms for tasks such as adding inventory, updating stock levels, and generating reports. These forms will be submitted via HTTPS, ensuring secure transmission.

Message Formatting: Form data will be structured in JSON when sent to the backend, allowing for efficient processing and validation before being stored in the database.

4. System Features

4.1 Authentication:

4.1.1 Description and Priority:

The Login Page is the gateway to the Stock Inventory Management System, allowing users to securely access their accounts based on their roles (e.g., Administrator, Inventory Manager, Warehouse Staff). It is essential for ensuring that only authorized personnel can interact with the system.

Priority: High
Priority Component Ratings:
Benefit: High
Penalty: High (Without proper authentication, unauthorized access could lead to data breaches)
Cost: Medium
Risk: High

4.1.2 Stimulus/Response Sequences:

Stimulus: The user navigates to the Login Page and enters their username and password.
Response: The system validates the credentials against Firebase Auth.

Stimulus: The system detects valid credentials.
Response: The user is redirected to the Dashboard.

Stimulus: The system detects invalid credentials.
Response: The user is presented with an error message and prompted to retry.

4.1.3 Functional Requirements:

1. The system must validate user credentials against Firebase Auth.
2. The system must prevent login attempts after a specified number of failed attempts and prompt the user to reset their password.
3. The system must provide a "Forgot Password" feature that allows users to reset their password via email.
4. The system must log unsuccessful login attempts and notify administrators if suspicious activity is detected.
5. The system should support multi-factor authentication (MFA) for enhanced security.

4.2 Stock In Page:

4.2.1 Description and Priority:

The Stock In Page allows users to add new inventory to the system and view items added on a specific date. This feature is crucial for maintaining accurate and up-to-date inventory records.

Priority: High
Priority Component Ratings:
Benefit: High
Penalty: High (Inaccurate stock records could lead to supply chain disruptions)
Cost: Medium
Risk: Medium

4.2.2 Stimulus/Response Sequences:

Stimulus: The user navigates to the Stock In Page.
Response: The system displays a form for adding stock items, including fields for item name, quantity, supplier, and date.
Stimulus: The user fills in the form and submits the stock addition.
Response: The system validates the inputs and updates the inventory records in the Firebase database.

Stimulus: The user selects a date to view items added on that day.
Response: The system retrieves and displays a list of items added on the selected date.

4.2.3 Functional Requirements:

1. The system must allow users to add new stock items, specifying details such as item name, quantity, supplier, and date.
2. The system must validate inputs, ensuring that required fields are completed and that quantities are positive integers.
3. The system must update the inventory records in Firebase Realtime Database and Firestore upon successful submission.
4. The system must provide a date filter to allow users to view stock items added on a specific date.
5. The system must handle input errors gracefully, displaying appropriate error messages.

4.3 Stock Out Page:

4.3.1 Description and Priority:

The Stock Out Page allows users to record items that have been sold, specifying the date of sale and the customer. It also provides a view of items sold on a particular date, supporting inventory tracking and sales reporting.

Priority: High
Priority Component Ratings:
Benefit: High
Penalty: High (Inaccurate sales records could lead to stock discrepancies and loss of revenue)
Cost: Medium
Risk: Medium

4.3.2 Stimulus/Response Sequences:

Stimulus: The user navigates to the Stock Out Page.
Response: The system displays a form for recording sold items, including fields for item name, quantity, customer, and date of sale.

Stimulus: The user fills in the form and submits the stock out record.
Response: The system validates the inputs and updates the inventory records, reducing the stock levels accordingly.

Stimulus: The user selects a date to view items sold on that day.
Response: The system retrieves and displays a list of items sold on the selected date, along with customer details.

4.3.3 Functional Requirements:

1. The system must allow users to record stock out transactions, specifying details such as item name, quantity, customer, and date of sale.

2. The system must validate inputs, ensuring that required fields are completed and that quantities do not exceed available stock levels.
3. The system must update the inventory records in Firebase Realtime Database and Firestore upon successful submission, reducing stock levels.
4. The system must provide a date filter to allow users to view items sold on a specific date.
5. The system must display customer details alongside the items sold, allowing for full traceability of transactions.

4.4 Billing System:

4.4.1 Description and Priority:

The Billing System is responsible for generating invoices for sales transactions, calculating totals based on items sold, and applying taxes or discounts. This feature is vital for ensuring accurate financial records and facilitating customer transactions.

Priority: High
Priority Component Ratings:
Benefit: High
Penalty: High (Errors in billing could lead to financial discrepancies and customer dissatisfaction)
Cost: Medium
Risk: Medium

4.4.2 Stimulus/Response Sequences:

Stimulus: The user completes a sale on the Stock Out Page.
Response: The system automatically triggers the billing process, calculating the total cost of the items sold, including any taxes or discounts.

Stimulus: The user views the generated invoice.
Response: The system displays the invoice, detailing items sold, quantities, prices, taxes, discounts, and the total amount due.

Stimulus: The user opts to print or email the invoice.
Response: The system formats the invoice for printing or generates a PDF to be sent via email.

4.4.3 Functional Requirements:

1. The system must automatically generate invoices upon the completion of a sale, detailing each item sold, its quantity, price, and any applicable taxes or discounts.
2. The system must calculate totals accurately, including subtotals, taxes, discounts, and the final amount due.
3. The system must allow users to print invoices or send them via email in PDF format.
4. The system must store copies of all invoices in the database for future reference and auditing purposes.

5. The system must handle any errors in the billing process, such as incorrect item prices, and provide mechanisms for correction.

4.5 Dashboard:

4.5.1 Description and Priority:

The Dashboard provides a comprehensive overview of sales and inventory activities, including data on items added, items sold, and overall stock levels. It presents this information through charts, graphs, and summaries, offering users quick insights into the system's performance.

Priority: High
Priority Component Ratings:
Benefit: High
Penalty: Medium (Without a dashboard, users would lack a quick reference to key metrics, leading to inefficiencies)
Cost: Medium
Risk: Low

4.5.2 Stimulus/Response Sequences:

Stimulus: The user logs into the system and navigates to the Dashboard.
Response: The system retrieves and displays key metrics, including total items added, items sold, current stock levels, and recent transactions.

Stimulus: The user selects specific filters or time periods.
Response: The system updates the displayed data to reflect the selected criteria, showing trends and comparisons over time.

Stimulus: The user clicks on a specific metric or chart.
Response: The system drills down into the detailed data, providing more granular insights into the selected metric.

4.5.3 Functional Requirements:

1. The system must display key metrics related to inventory and sales, including total items added, items sold, and current stock levels.
2. The system must allow users to filter data by date, product category, or other relevant criteria.
3. The system must present data visually through charts, graphs, and summaries, providing quick insights into performance.
4. The system must allow users to drill down into specific metrics to view more detailed data and trends.
5. The system must update dashboard data in real-time or near-real-time, ensuring that users have access to the most current information.

5. Other Nonfunctional Requirements

5.1 Performance

- Response Time: The system should provide real-time updates to the inventory when stock is added or sold, with a response time of under 2 seconds.
- Throughput: The system should be capable of handling at least 1000 transactions per hour without performance degradation.
- Scalability: The system should be scalable to handle an increasing number of users and transactions as the shop grows. This includes the ability to scale the backend to handle higher loads, both in terms of database queries and user requests.

5.2 Security

- Authentication and Authorization: The system must implement Firebase Authentication to ensure secure access to the application. Only authorized personnel should be able to access sensitive features such as adding or removing stock items.
- Data Encryption: All sensitive data, including user credentials and transaction details, should be encrypted during transmission and storage. This includes using HTTPS for data transmission and encryption standards like AES for storing data.
- Access Control: Role-based access control (RBAC) should be enforced to ensure that users have access only to the functionalities relevant to their roles (e.g., cashiers vs. managers).
- Data Integrity: The system must ensure data integrity, preventing unauthorized modification of inventory data or transaction records.

5.3 Reliability

- Availability: The system should be available 99.9% of the time, ensuring minimal downtime. Any planned maintenance should be scheduled during off-peak hours to minimize disruption.
- Fault Tolerance: The system should be able to handle unexpected failures without losing data. For example, if the server crashes, the system should recover without data loss.
- Backup and Recovery: Regular backups of the database should be taken to ensure data can be recovered in case of system failure. The recovery process should restore data within 30 minutes of a failure.

5.4 Usability

- User Interface: The user interface should be intuitive and easy to navigate, with minimal training required for new users. Bootstrap's responsive design should ensure a consistent user experience across different devices (e.g., desktops, tablets, smartphones).
- Accessibility: The system should be accessible to users with disabilities, following the Web Content Accessibility Guidelines (WCAG) 2.1 Level AA standards.
- Internationalization: The system should support multiple languages if the shop caters to a diverse

customer base.

5.5 Maintainability

- Code Quality: The code should be modular, well-documented, and follow best practices to facilitate easy maintenance and future updates. The use of Ruby on Rails will ensure a clear MVC architecture, promoting separation of concerns.
- Version Control: All code changes should be tracked using a version control system like Git, ensuring that any issues can be quickly identified and rolled back if necessary.
- Update Process: The system should allow for seamless updates with minimal downtime, ensuring that new features or bug fixes can be deployed without disrupting the service.

5.6 Compatibility

- Browser Compatibility: The system should be compatible with all major web browsers, including Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. It should also degrade gracefully on older browser versions.
- Device Compatibility: The application should work smoothly on various devices, including desktops, tablets, and smartphones. The responsive design provided by Bootstrap 5 will ensure that the user interface adapts to different screen sizes.
- Integration with External Systems: The system should integrate seamlessly with external systems, such as payment gateways or third-party APIs, if needed in the future.

5.7 Efficiency

- Resource Utilization: The system should be optimized to minimize resource consumption, such as CPU, memory, and network bandwidth. This includes optimizing database queries and reducing the size of frontend assets (e.g., images, scripts).
- Energy Efficiency: The system should be designed to minimize energy consumption, especially in cloud-hosted environments where resource usage directly impacts costs.

5.8 Legal and Regulatory Compliance

- Data Privacy: The system must comply with data privacy regulations such as the General Data Protection Regulation (GDPR) or any other relevant laws. This includes obtaining user consent for data collection and ensuring data is stored and processed in compliance with legal requirements.
- Auditability: The system should maintain an audit trail of all transactions and changes to inventory data, allowing for easy traceability and accountability in case of discrepancies.

5.9 Supportability

- Technical Support: The system should include clear documentation for users and administrators, covering installation, configuration, usage, and troubleshooting. Additionally, a support channel (e.g., email or chat) should be available for addressing user issues.
- Community Support: Given the use of popular frameworks like Ruby on Rails and Bootstrap, the

system should benefit from the large community support available for these technologies

6. Other Requirements

Database Requirements

- The database schema, including tables, columns, data types, and relationships, will be defined.
- Database normalization will be implemented to optimize data integrity and efficiency.
- Data validation and constraints will be enforced to ensure data accuracy and consistency.
- Database triggers or constraints will be used to enforce business rules.
- Appropriate indexes will be created for frequently queried columns.
- Database partitioning will be considered for large datasets.
- Data backup and recovery strategies and procedures will be defined and implemented.
- Disaster recovery plans will be implemented.
- Data encryption and access controls will be enforced to protect sensitive information.

Reuse Objectives

- Potential reusable components (e.g., modules, libraries) will be identified.
- The system will be designed with modularity and maintainability in mind.
- The choice of Bootstrap, JavaScript, Ruby on Rails, and Firebase will be justified based on their suitability for the project.
- Integration with existing systems or services (e.g., payment gateways, shipping providers) will be considered.

Firebase Specific Requirements

- User roles and permissions (e.g., admin, user, guest) will be defined.
- Secure authentication mechanisms (e.g., email/password, social login) will be implemented.
- The appropriate Firebase database (Realtime Database or Cloud Firestore) will be selected based on project needs.
- Data structure and indexing rules will be defined.
- Data synchronization and offline capabilities will be implemented.
- Firebase Security Rules will be created to protect data access.

Appendix A: Glossary

CRUD: Create, Read, Update, Delete

API: Application Programming Interface

SKU: Stock Keeping Unit

Firebase: A platform by Google that provides backend services for web and mobile applications

Authentication: The process of verifying the identity of a user

Authorization: The process of determining what a user is allowed to do

Inventory: A detailed list of items in stock

Stock: A supply of goods kept on hand for sale or use

Product: An item or service offered for sale

Supplier: A person or company that provides goods or services

Order: A request for goods or services

User: An individual who interacts with the system

Admin: A user with administrative privileges

SQL: Structured Query Language

HTML: HyperText Markup Language

CSS: Cascading Style Sheets

JS: JavaScript

ROR: Ruby on Rails

Appendix B: Flow Diagram