****

**UNIVERSITY OF SRI JAYEWARDENEPURA**

**Faculty of Technology**

**Department of Information and Communication Technology**

**ITS3552 – Software Engineering II**

**Assignment 02**

**GROUP MEMBERS:**

**Inventory Management System (IMS) Software Requirements Specification**

**Table of Contents**

**1.Introduction**

1.1. Purpose

1.2.Scope

1.3.References

1.4.System Overview

**2.System Requirements**

2.1.Functional Requirements

2.2. Non-Functional Requirements

2.3 User Roles and Permissions

Administrator(Manager)

Employee

2.4 Use Cases

Create and Update Products

Track Sales and Inventory

Predict Supply and Demand

**3.Database Design**

3.1 Database Schema

3.2 Data Models

3.3Data Security Measures

Encryption

Access Control

**4.System Architecture**

4.1 Architecture pattern

4.2 Design Patterns

4.3 high level architecture diagram

**5.System Design**

5.1 user interface design

5.2 User Interface mockups

5.3 User Interface Patterns

**6.development procedure**

6.1 Code Best Practices

Coding Standards (e.g., naming conventions, indentation)

6.2.Code Documentation Guidelines

6.3.Technologies

6.4.Coding Guidelines

Version Control (e.g., Git)

6.5.Collaboration

Team Collaboration Tools (e.g., Slack, JIRA)

Collaboration Workflow

**7.Testing**

7.1 Testing Types

Unit Testing

Integration Testing

Functional Testing

Performance Testing

7.2 Test Cases and Scenarios

7.3 Test Data

7.4 Test Results and Reporting

**8.Security**

8.1 Code-Level Security Principles

Input Validation

Authentication and Authorization

8.2 Data Security

Encryption

Backup and Recovery Procedures

**10.Maintenance and Support**

10.1 Bug Tracking and Resolution

10.2 Software Updates and Enhancements

10.3 User Support

**11. Constraints**

**12. Assumptions**

**13.Appendices**

13.1 Glossary

13.2 Acronyms

13.3 Document History

**ACKNOWLEDGEMENT**

As a team, we would like to express our deepest appreciation to all who provided us the

possibility to complete this ‘inventory Management System’ Software project.

successfully. A special gratitude, we would like to give to MRS. Sankani , Our lecturer for

giving us the valuable Guidance, Advice, Clearing all our Doubts at any time without any

hesitation and for all the support shown throughout the period to complete our project.

Finally, we would like to take this opportunity to express our gratitude to all my group members( Mitra Jain, Suresh Chandra, Mahesh Shah, Kritika Mittal, and V S Athira). The project would not have been successful without their cooperation and input.

**INTRODUCTION**

An inventory management system is a critical component of modern businesses, designed to efficiently track, manage, and optimize the flow of goods and products within an organization. In software engineering, an inventory management system is a comprehensive software solution that integrates various modules and functionalities to streamline inventory-related operations. These systems help businesses maintain accurate inventory records, forecast demand, minimize stockouts, reduce carrying costs, and ultimately enhance overall operational efficiency. By leveraging technology, businesses can gain real-time visibility into their inventory, make informed decisions, and ensure that products are readily available to meet customer demands while minimizing excess stock. Effective software engineering in this context involves designing, developing, and maintaining a robust, user-friendly, and scalable inventory management system tailored to the specific needs of the organization.

**1.1 Purpose**

inventory management is key to maintaining a profitable, organized, and productive business. For some companies, practicing inventory management is simple: they take inventory every week or so by walking through a storage closet and checking to see if they’re low on anything. But other companies must take inventory management quite seriously, tracking every item the minute it arrives, moves, or is used up.

The main purpose of inventory management is to help businesses easily and efficiently manage the ordering, stocking, storing, and use of inventory. By effectively managing your inventory, you’ll always know what items are in stock, how many of them there are, and where they are located.

Plus, practicing strong inventory management allows you to understand how you use your inventory–and how demand changes for it–over time. You can zero in on exactly what you need, what’s not so important, and what’s just a waste of money. That’s using inventory management to practice inventory control. By the way, inventory control is the balancing act of always having enough stock to meet demand, while spending as little as possible on ordering and carrying inventory.

**1.2 Scope**

Our Inventory Management System will feature an intuitive and user-friendly interface, ensuring that users, including small business owners, can efficiently operate the system without the need for advanced training or extensive documentation. The system will be built upon a reliable and secure database, ensuring the accuracy of real-time data, which is essential for effective inventory monitoring and control. Performance optimization will be a top priority, enabling fast access to actionable inventory information. Additionally, the system will be designed to support scalability, allowing administrators to easily integrate new software modules with minimal configuration, ensuring that the system can adapt to evolving business needs.

**1.3.References**

**1.4.System Overview**

The Inventory Management System (IMS) is a software solution tailored for small businesses to efficiently manage their inventory and meet customer demands. It is designed to provide:

1. **User-Friendly Interface**: The IMS offers an intuitive and easy-to-use interface that requires no advanced training or extensive documentation, ensuring accessibility for all users.
2. **Reliable Database**: The system is built on a secure and reliable database that ensures the accuracy and real-time availability of inventory data.
3. **High Performance**: IMS delivers fast and actionable inventory monitoring and control capabilities, enhancing efficiency in managing inventory levels.
4. **Scalability**: Administrators can effortlessly expand the system's functionality by adding software modules with minimal configuration, allowing for future growth and adaptation to changing business needs.

**2.System Requirements**

2.1.Functional Requirements

1. **User Authentication and Authorization:**
   * Users must log in with unique credentials.
   * Before logging in, the user should register into system.
2. **Product Management:**
   * Add new products with details .
   * Edit and update product information.
   * Delete or discontinued products.
   * Print report
3. **Inventory Tracking:**
   * Display real-time inventory levels for each product.
   * Automatically update inventory quantities after each sale.
4. **Sales Record Management:**
   * Record sales transactions, including date, time, product sold, and quantity.
   * Generate sales reports and analytics.
   * About update & sales.
5. **Reporting and Analytics:**
   * Generate reports.
   * Allow users to customize report parameters and export data.
6. **Low Stock Alerts:**
   * Send notifications (email or in-app) to authorized users when inventory levels are critically low.
   * Allow users to set custom alert thresholds.

.

2.2. Non-Functional Requirements

1. **Performance:**
   * The system should provide fast response times, even with a large volume of data.
2. **Scalability:**
   * The system must support easy expansion by allowing administrators to add new features or modules without major disruptions.
3. **Reliability:**
   * The database should be highly reliable, with minimal downtime for maintenance.
   * The system must prevent data corruption and provide data recovery mechanisms.
4. **Security:**
   * Implement robust security measures to protect user data and prevent unauthorized access.
   * Regularly update and patch the system to address security vulnerabilities.
   * Implement user authentication and session management.
   * Regularly back up data to prevent loss in case of system failure.
5. **Usability:**
   * The user interface should be intuitive and require minimal training.
   * Response in real time
6. **Availability:**
   * Ensure that the system is available 24/7, with scheduled maintenance communicated in advance.
   * Implement redundancy and failover mechanisms to minimize downtime.
7. **Data Backup and Recovery:**
   * Regularly back up the database to prevent data loss.
   * Implement a robust data recovery plan in case of system failures or disasters.
8. **Compliance:**
   * The system should comply with relevant data protection and privacy regulations (e.g., GDPR, HIPAA).

**7.Security**

7.1 Code-Level Security Principles

1. **Input Validation:**
   * Always validate user inputs to prevent SQL injection, cross-site scripting (XSS), and other injection attacks.
   * Use parameterized queries or prepared statements when interacting with the database to prevent SQL injection.
2. **Authentication and Authorization:**
   * Implement strong authentication mechanisms to verify the identity of users.
   * Enforce proper authorization checks to ensure that users have the appropriate permissions to access specific resources or perform actions.
3. **Password Security:**

* Enforce strong password policies, including complex requirements and regular password expiration.

1. **User-Friendly Interface:**
   * Ensure an intuitive and easy-to-navigate user interface.
   * Implement user-friendly forms for data entry and retrieval.
   * Provide tooltips and help sections for user assistance.

2.3 User Roles and Permissions

Administrator(Manager)

Administrator (Manager):

The Administrator, often referred to as the Manager, is a user with the highest level of authority and control within the inventory management system.

Their primary responsibilities include overseeing and managing the entire system, making critical decisions, and ensuring the system functions correctly.

Administrator permissions typically include:

* Product management: Adding, Updating, or Deleting products from the inventory.
* Inventory tracking: Accessing real-time inventory levels, adjusting, and setting alerts.
* Sales record management: generating reports, and analytics.
* System configuration: Modifying system settings, adding modules, and customizing the system to meet the organization's evolving needs.
* Security management: Configuring and maintaining security measures, including user authentication, authorization, and password policies.
* Data backup and recovery: Managing data backups and implementing data recovery plans.
* Compliance: Ensuring that the system complies with relevant data protection and privacy regulations.

Employee:

* Employees are users with limited access and responsibilities within the inventory management system.
* Their role typically involves day-to-day tasks related to inventory management but does not include administrative functions.
* Employee permissions often include:
* User authentication: Logging into the system with unique credentials.
* Inventory tracking: Viewing real-time inventory levels for products.
* Generating basic reports related to their daily tasks.
* Product management (partial access): Some employees may have permission to update product information or mark products as "out of stock" but may not be allowed to add or delete products.
* Low stock alerts (partial access): Depending on their responsibilities, employees may receive notifications when inventory levels are critically low or be able to set custom alert thresholds for specific products.

2.4 Use Cases:

* Create, delete and Update Products:
* Description
  + This use case involves the ability to create new product entries in the inventory management system, delete damaged and expired products and update existing product information as needed.
* Functionality
  + Creating Products: administrators can add new products to the system. This includes providing details such as product name, description, SKU (Stock Keeping Unit), price, and initial stock quantity.
  + Delete product: administrators can delete products from the system.
  + Updating Products: admin and employee both Users can modify product information when necessary. This could involve changing product prices, descriptions, or other attributes.
  + Purpose: This use case ensures that the inventory database is up to date with the latest product information, enabling accurate tracking and management of inventory levels.
* Track Sales and Inventory:
* Description: This use case involves monitoring the sales transactions and real-time inventory levels within the inventory management system.
* Functionality:
* Inventory Monitoring: The system continuously updates and displays real-time inventory levels for each product. This allows users to see the current stock status, identify low-stock situations, and take appropriate actions.
* Purpose: Tracking sales and inventory levels is essential for maintaining accurate records, preventing stockouts, and ensuring products are available to meet customer demand. It also helps identify trends and patterns in sales.
* Predict Supply and Demand:
* Description: This use case involves using historical sales and inventory data to predict future supply and demand patterns for products.
* Functionality:
* Data Analysis: The system may include analytics tools that analyze historical sales data, seasonal trends, and other factors to make predictions about which products will be in high demand in the future.
* Demand Forecasting: Based on the analysis, the system can generate demand forecasts, helping businesses plan their inventory levels and order products in advance.
* Supply Planning: Businesses can use these predictions to optimize their supply chain, ensuring they have the right amount of inventory on hand to meet expected demand.
* Purpose: Predicting supply and demand helps businesses reduce excess stock, prevent stockouts, and make informed decisions about inventory replenishment. It contributes to cost savings and efficient inventory management.

**4.1.System Architecture**

The following outlines the application of the Model-View-Controller (MVC) architectural pattern to our Inventory Management System (IMS). MVC is a well-established pattern that separates the concerns of data management, user interface, and application logic, contributing to better maintainability and scalability.

**1. Model (Data and Business Logic):**

Description: The Model component of the IMS is responsible for managing data, implementing business logic, and ensuring data integrity.

Responsibilities:

Maintain a secure and reliable database for storing product information, sales records, and inventory data.

Implement algorithms for demand forecasting and inventory control to optimize stock levels.

Handle data validation, ensuring that data adheres to predefined business rules.

Purpose: The Model component ensures that data is managed consistently and provides the foundation for accurate inventory management.

**2. View (User Interface):**

Description: The View component of the IMS is responsible for presenting data to users through a user-friendly interface.

Responsibilities:

Render product listings, sales reports, and real-time inventory information to users.

Provide user-friendly interface for creating, deleting, and updating, and viewing products and sales records.

Display notifications and alerts, such as low stock warnings, in a clear and understandable manner.

Purpose: The View component ensures that users can interact with the system efficiently and receive information in a user-friendly format.

**3. Controller (Application Logic):**

Description: The Controller component acts as an intermediary between the Model and the View, managing user input and application logic.

Responsibilities:

Receive and process user requests for actions such as adding new products, delete and update products when necessary, recording sales, or generating reports.

Validate user input and ensure it complies with business rules.

Communicate with the View to display updated information to users.

Purpose: The Controller component separates the user interface from the application logic, promoting modularity and maintainability, and supporting multiple views for different user interfaces.

**Why MVC?**

Implementing the Model-View-Controller (MVC) architectural pattern in our Inventory Management System (IMS) is a strategic choice. This approach ensures a clear separation of concerns, allowing us to manage data, user interfaces, and application logic independently. With MVC, we can enhance the system's maintainability, scalability, and user experience. The Model handles data and business logic, the View delivers a user-friendly interface, and the Controller manages user interactions and application flow. This separation empowers us to adapt and extend the IMS efficiently as our inventory management needs evolve, ultimately providing a robust and user-centric solution.

**Advantages of Using the MVC Pattern:**

* Modularity: MVC divides the system into distinct components, making it easier to develop, test, and maintain each component independently.
* Scalability: The system can be scaled by adding more Views or Controllers without major changes to the underlying Model.
* Reusability: Components like the Model and Controller can be reused across different parts of the system or in future projects.
* Maintainability: Changes to one component do not necessitate modifications to others, reducing the risk of introducing errors.
* Enhanced User Experience: The View can be customized to provide an intuitive and user-friendly interface, improving user satisfaction.

**Limitations:**

* Testing Challenges: Writing unit tests for MVC can be complex, especially when dealing with the user interface components in View.
* Increased File Count: Implementing MVC often results in a larger number of files and directories, which can make the project directory structure more complex.

**4.2 Design Patterns**

Design Patterns in the MVC Architecture of our Inventory Management System (IMS):

* Singleton Pattern (Model):

We apply the Singleton pattern to our model component to ensure that there is only one instance of the Model throughout the application's lifespan. This guarantees data consistency and integrity by preventing multiple instances with potentially conflicting data.

It restricts the instantiation of the Model class to a single instance. Any part of the application that requires access to the Model interacts with this single instance. This ensures that all data and business logic are centralized, reducing the risk of data inconsistencies.

* Observer Pattern (View):

Within the View component, we use the Observer pattern to address the requirement for real-time updates of inventory and sales information displayed to users.

Views register themselves as observers of the Model. When changes occur in the Model's data, such as inventory updates, the Model notifies all registered observers (Views). This mechanism ensures that Views automatically update their displays to reflect the most current data, enhancing the user experience with real-time information.

* Factory Method Pattern (Controller):

Within the Controller component, we use the Factory Method pattern to decouple the creation of specific controllers for different user interactions.

The Factory Method pattern defines an interface for creating Controllers, but the concrete subclasses of the Factory determine which specific Controller to instantiate based on the user's request. This promotes flexibility by allowing the application to create the appropriate Controller dynamically, based on the user's interaction, without tightly coupling the Controller creation logic to the user interface.

**6.3.Technologies**

* C# (Programming Language):

We have chosen C# as the primary programming language for building our IMS. C# is a versatile and widely adopted language known for its strong typing, modern features, and compatibility with various development platforms, making it an excellent choice for creating scalable and maintainable software solutions. C# is used to implement business logic, data processing, user interface, and various functionalities within the IMS.

* .NET Framework:

provides a comprehensive set of tools and libraries for building Windows desktop applications with a rich user interface.it is best for windows-based applications. WinForms is a graphical user interface (GUI) library that allows you to create windows-based applications with a traditional, native Windows look and feel.

* MySQL (Database Management System):

MySQL is our chosen database management system (DBMS) for storing and managing the IMS's critical data, including product information, sales records, and inventory levels. MySQL is an open-source, relational database that offers excellent performance and scalability.

MySQL serves as the backend database for our IMS, ensuring data reliability, integrity, and efficient data retrieval and storage.

These technologies, C#, and MySQL, complement each other in our IMS, enabling us to build a user-friendly and scalable inventory management solution while ensuring data accuracy and accessibility. Our choice of these technologies reflects our commitment to delivering a robust and reliable IMS that can adapt to the evolving needs of our business.

**6.4 Coding Guidelines**

In the development of our Inventory Management System (IMS), adhering to consistent coding guidelines is crucial to ensure code readability, maintainability, and collaboration among our development team. We follow these coding guidelines:

Version Control: Frequent commits with descriptive commit messages in Git to track changes, resolve issues, and maintain version history. Often referred to as source code or revision control, is a systematic way to track changes to your software codebase over time. It allows multiple developers to collaborate on a project, manage code changes, and keep a detailed history of modifications. For our IMS development, we rely on Git, one of the most widely used version control systems. Here's how version control benefits our project:

History Tracking: Git maintains a chronological history of all code changes, known as commits. Each commit includes details such as who made the change, when it was made, and why. This history is invaluable for debugging, understanding the evolution of the codebase, and identifying the source of issues.

Branching: Git allows developers to create branches, which are separate lines of development. We use branches to work on new features, bug fixes, or experiments without affecting the main or "master" branch. This isolation is crucial for parallel development efforts.

Collaboration: Multiple team members can collaborate simultaneously on different branches. Git facilitates merging changes from one branch into another, enabling seamless teamwork and code integration.

Code Review: Code reviews are an integral part of our development process. Git enables us to create pull requests (PRs) for proposed code changes. Team members can review the changes, provide feedback, and discuss improvements within the PR before merging it into the main branch.

Conflict Resolution: When two or more developers make changes to the same code simultaneously, Git detects and highlights conflicts. Resolving conflicts is a structured process, ensuring that code modifications are harmoniously integrated.

Rollback and Revert: Git makes it straightforward to roll back to a previous commit or revert specific changes. This feature is valuable for mitigating issues and ensuring that the codebase remains stable.

Backup and Disaster Recovery: By using Git repositories hosted on remote platforms like GitHub we have robust backups of our codebase. In case of data loss or hardware failures, we can recover the code from the remote repository.

Tracking Features and Bug Fixes: Git commits are typically tagged with descriptive messages that link changes to specific features, bug fixes, or issues. This clear association aids in project management and release planning.

Version Tagging: We use Git tags to mark significant milestones or releases in our project. This allows us to easily identify and access specific versions of the IMS codebase.

**6.5 Collaboration**

Collaboration is essential in the development of our Inventory Management System (IMS). Effective communication and the use of collaboration tools are key to our success. Here's how we facilitate collaboration within our development team:

Version Control : We utilize Git as our version control system, enabling us to track changes, manage branches, and collaborate on code development efficiently. Our Git repository is hosted on a platform that facilitates collaboration and code review.

Team Collaboration Tools: We leverage various collaboration tools to streamline communication and project management:

Git and GitHub:

Purpose: Git is a version control system, while GitHub is web-based platforms for hosting Git repositories and managing collaborative development.

Why We Use It: Git and GitHub are essential for version control, code collaboration, and code review. They enable multiple developers to work on the IMS codebase concurrently, manage code changes, and conduct code reviews.

Zoom or Microsoft Teams (for Meetings):

Purpose: These platforms provide video conferencing, screen sharing, and virtual meeting capabilities.

Why We Use It: Video conferencing tools are valuable for team meetings, discussions, and collaboration, especially when team members are geographically dispersed.

Email (for Formal Communication):

Purpose: Email is a traditional communication tool used for sending formal messages, documentation, and reports.

Why We Use It: E-mail is used for more formal communication, it help us to contact our lecturer and demonstrators to ask doubts about our project.

Google Workspace or Microsoft Office 365 :

Purpose: These suites offer cloud-based document creation and collaboration tools, including word processing, spreadsheets, and presentation software.

Why We Use It: Google Workspace or Office 365 enables collaborative document editing and sharing, ensuring that team members can collaborate on project-related documents in real-time.

**Collaboration Workflow:**

Code Reviews: We conduct thorough code reviews to maintain code quality, identify potential issues, and share knowledge within the team.

Sprints and Agile Practices: We follow agile methodologies, conducting regular sprint planning, stand-up meetings, and retrospectives to ensure that development aligns with project goals and priorities.

Task Assignment: Tasks are assigned based on team members' expertise, and progress is tracked using our project management tools.

By employing these collaboration practices and tools, we create a collaborative and efficient development environment that fosters teamwork, transparency, and the successful development of our IMS.