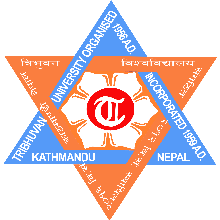
**Tribhuvan University**

**Institute of Science and Technology**

**Amrit Campus**



A

Minor Project on  
“Inventory Management System”

[Course Code: BIT 352]

Submitted By:

Apil Adhikari (207/207)

Dipesh Ghimire (199/207)

Prateek Bastola (209/077)

Rabin Pant (210/077)

Ujjwal Aryal (229/077)

Submitted To:

Department of Bachelor in Information Technology

Lecturer: Janak Raj Joshi

July, 2024

Certificate of Approval

This is to clarify that the project prepared by Apil Adhikari, Dipesh Ghimire, Prateek Bastola, Rabin Pant and Ujjwal Aryal entitled “Inventory Management System” in the partial fulfillment of the requirements for the degree of Bachelors in Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree. Finally, Department of BIT, Amrit Campus approves this project for final evaluation.

………………….

Janak Raj Joshi

Lecturer of Management Information System

…………………..

External Examiner

Acknowledgements

We would like to express our deepest gratitude to all those who contributed to the successful completion of this Inventory Management System project. First and foremost, we would like to thank our supervisor and mentor, Janak Raj Joshi, for their invaluable guidance, insightful feedback, and constant support throughout the development of this project. Their expertise and encouragement have been instrumental in navigating challenges and achieving our project goals.

We extend our sincere appreciation to the faculty and staff of Amrit Campus for providing the resources and environment necessary for this project. Special thanks to Binod Thapa for his support and assistance.

We are grateful to our colleagues and peers for their collaboration, constructive criticism, and encouragement. Their camaraderie and exchange of ideas have enriched this project and made the process enjoyable. Finally, we would like to express our heartfelt gratitude to our families and friends for their unwavering support, patience, and understanding during the course of this project.

Thank you to everyone who contributed to the success of this project. Your efforts and encouragement have been invaluable.

Thanking You,

Apil Adhikari (207/077)

Dipesh Ghimire (199/077)

Prateek Bastola (209/077)

Rabin Pant (210/077)

Ujjwal Aryal (229/077)

List of Figures and Tables

[Figure 1: Gantt Chart 18](#_Toc172638548)

[Figure 2: Entity Relationship (ER) Diagram for Inventory Management System 21](#_Toc172638549)

[Figure 3: Activity Diagram for Inventory Management System 22](#_Toc172638550)

[Figure 4: Use-Case Diagram for Inventory Management System 23](#_Toc172638551)

[Figure 5: Sequence Diagram for Inventory Management System 23](#_Toc172638552)

[Table 1: Unit Testing 30](#_Toc172635917)

# List of Abbreviations and Acronyms

AJAX Asynchronous JavaScript and XMS

API Application Programming Interface

CSS Cascading Style Sheets

DFD Data Flow Diagram

ERD Entity Relationship Diagram

HTML Hypertext Markup Language

IDE Integrated Development Environment

MVC Model View Controller

Table of Contents

[Cover Page i](#_Toc172641122)

[Certificate of Approval ii](#_Toc172642209)

[Acknowledgements iii](#_Toc172642210)

[List of Figures and Tables iv](#_Toc172642211)

[List of Abbreviations and Acronyms v](#_Toc172642212)

[Table of Contents vi](#_Toc172642213)

[Abstract viii](#_Toc172642214)

[Chapter 1: Introduction 1](#_Toc172642215)

[1.1 Introduction 1](#_Toc172642216)

[1.2 Problem Statement 1](#_Toc172642217)

[1.3 Objectives 1](#_Toc172642218)

[1.4 Scope 2](#_Toc172642219)

[1.5 Limitations 2](#_Toc172642220)

[Chapter 2: Background Study and Literature Review 4](#_Toc172642221)

[2.1 Background Study 4](#_Toc172642222)

[2.2 Literature Review 4](#_Toc172642223)

[2.1.1 Inventory Management Theories 4](#_Toc172642224)

[2.1.2 Inventory Control Techniques 5](#_Toc172642225)

[2.1.3 Technological Advancements 5](#_Toc172642226)

[2.1.4 User Roles and System Design 5](#_Toc172642227)

[2.1.5 Case Studies and Practical Implementations 6](#_Toc172642228)

[Chapter 3: System Analysis 7](#_Toc172642229)

[3.1 System Analysis 7](#_Toc172642230)

[3.1.1 Requirement Analysis 7](#_Toc172642231)

[3.1.2 Feasibility Analysis 7](#_Toc172642232)

[3.1.3 Methods for Determining System Requirements 9](#_Toc172642233)

[3.1.4 Gantt Chart 11](#_Toc172642234)

[Chapter 4: System Design and Development 12](#_Toc172642235)

[4.1 System Design 12](#_Toc172642236)

[4.1.1 Architectural Design 12](#_Toc172642237)

[4.1.2 Logical Design 13](#_Toc172642238)

[4.1.3 Physical Design 13](#_Toc172642239)

[4.1.4 Tools Used for System Design 14](#_Toc172642240)

[4.2 System Development 17](#_Toc172642241)

[4.2.1 Software Methodology 17](#_Toc172642242)

[4.2.2 Tools 18](#_Toc172642243)

[4.2.3 Development Method 18](#_Toc172642244)

[4.2.4 Hardware Requirements 18](#_Toc172642245)

[4.2.5 Software Requirements 18](#_Toc172642246)

[4.2.6 Programming Languages 19](#_Toc172642247)

[Chapter 5: Implementation and Testing 20](#_Toc172642248)

[5.1 Implementation 20](#_Toc172642249)

[5.1.1 Tools Used 20](#_Toc172642250)

[5.1.2 Technologies 21](#_Toc172642251)

[5.1.3 Implementation Steps 23](#_Toc172642252)

[5.2 Testing 24](#_Toc172642253)

[5.2.1 Unit Testing 24](#_Toc172642254)

[5.2.2 System Testing 24](#_Toc172642255)

[5.2.3 User Acceptance Testing 25](#_Toc172642256)

[5.3 Result Analysis 26](#_Toc172642257)

[5.3.1 Functional Testing Outcomes 26](#_Toc172642258)

[5.3.2 Performance and Reliability 26](#_Toc172642259)

[5.3.3 Security and Compliance 26](#_Toc172642260)

[5.3.4 User Acceptance 26](#_Toc172642261)

[5.3.5 Issues and Resolutions 26](#_Toc172642262)

[Chapter 6: Conclusion and Future Recommendations 28](#_Toc172642263)

[6.1 Conclusion 28](#_Toc172642264)

[6.2 Future Recommendations 28](#_Toc172642265)

[References 30](#_Toc172642266)

[Appendix 31](#_Toc172642267)

# Abstract

Effective inventory management is crucial for businesses aiming to maintain optimal stock levels, reduce costs, and enhance operational efficiency. This project presents a comprehensive Inventory Management System (IMS) designed to address these challenges by providing a robust framework for managing inventory across various business processes. The system integrates advanced features such as real-time tracking of inventory levels, automated reordering, and comprehensive reporting tools. Our IMS utilizes cutting-edge technologies, to ensure accurate and efficient inventory tracking and management. By leveraging a multi-tier architecture and repository pattern, the system enhances data integrity and scalability. The user-friendly interface allows for seamless interaction with the system, enabling users to manage inventory, suppliers, and transactions effectively. This project also explores the integration of the IMS with enterprise resource planning (ERP) systems, which further streamlines operations and decision-making. The system was subjected to rigorous testing, including unit and system testing, to ensure reliability and performance. The results indicate significant improvements in inventory accuracy, reduced operational costs, and enhanced user satisfaction. The study concludes with recommendations for future enhancements, including the integration of machine learning algorithms for predictive analytics and the incorporation of cloud-based solutions to facilitate remote access and scalability.

**Keywords**: Automated Reordering, Data integrity, Enterprise Planning (EPR) integration, Inventory Management System, Operational efficiency, Real-time tracking, multi-tier architecture

Chapter 1: Introduction

1.1 Introduction

In today’s dynamic business environment, efficient inventory management is crucial for operational success and customer satisfaction. An Inventory Management System (IMS) is designed to automate and optimize the process of tracking and managing inventory. This system ensures accurate monitoring of stock levels, order status, and inventory movements, thus enabling businesses to streamline their operations and make informed decisions.

The Inventory Management System developed for this project is a comprehensive application designed to handle various aspects of inventory management. It provides a user-friendly interface and robust functionality, addressing the critical needs of businesses in managing their inventory effectively. By automating routine tasks and offering real-time insights, the system aims to enhance operational efficiency and reduce manual errors.

1.2 Problem Statement

Traditional inventory management methods often involve manual tracking, which can lead to several challenges, including:

* Overstocking: Excess inventory can lead to high holding costs and wasted resources.
* Stockouts: Insufficient stock can result in missed sales opportunities and customer dissatisfaction.
* Inaccurate Tracking: Manual records are prone to errors, leading to discrepancies in stock levels and financial reporting.
* Inefficient Processes: Time-consuming manual processes can affect overall productivity and operational efficiency.

To address these challenges, there is a need for an automated solution that provides real-time tracking, reduces manual intervention, and offers comprehensive reporting capabilities. The Inventory Management System is designed to solve these problems by integrating various functionalities into a single platform.

1.3 Objectives

The primary objectives of the Inventory Management System are:

* **Automation**: To automate inventory tracking and management processes, reducing the need for manual intervention and minimizing errors.
* **Real-Time Data**: To provide real-time updates on inventory levels, transactions, and order statuses, facilitating better decision-making.
* **Cost Efficiency**: To optimize inventory levels and reduce costs associated with overstocking and stockouts.
* **Enhanced Customer Satisfaction**: To ensure product availability and timely order fulfillment, improving customer satisfaction.

1.4 Scope

* **System Overview:** The Inventory Management System is designed to handle essential inventory tasks, including managing product categories, units of measurement, storage racks, suppliers, customers, products, purchases, and transactions. It features reporting capabilities such as custom reports, detailed reports, and stock level summaries.
* **User Roles and Permissions:** The system supports three distinct user roles:
  + **Super Admin**: Manages user accounts and has full access to all features.
  + **Admin**: Executes all operations, including CRUD (Create, Read, Update, Delete) actions and report generation.
  + **Normal User**: Limited to basic functions like billing and viewing product lists.
* **User Interface:** Designed with an intuitive and user-friendly interface to ensure ease of use across different functionalities, aiming to streamline inventory management processes.

1.5 Limitations

While we strive to deliver a robust and feature-rich "Inventory Management System" certain limitations should be acknowledged:

* **Financial Management**: The system does not encompass detailed financial accounting features such as cost analysis, profit and loss statements, or comprehensive financial reports. It focuses primarily on inventory management rather than full financial oversight.
* **External Integration**: The system does not offer integration capabilities with external software or third-party applications, limiting its functionality to internal operations only.
* **Scalability Constraints**: While effective for standard inventory tasks, the system may require additional customization to handle larger-scale operations or more complex inventory needs.
* **Advanced Analytics**: Reporting features are basic and do not include advanced analytics or predictive tools. Users seeking in-depth data analysis may need to use additional software.
* **Training Requirements**: Effective use of the system necessitates proper training. Although the system interface is user-friendly, training is essential to ensure users can fully utilize all features.
* **Customization Limits**: The system’s customization options are confined to built-in functionalities. Users with specialized needs may require further development to adapt the system to unique requirements.

Chapter 2: Background Study and Literature Review

2.1 Background Study

The background study for the Inventory Management System (IMS) explores foundational concepts crucial for effective inventory control. Central to inventory management are models such as Economic Order Quantity (EOQ), which optimizes the order quantity to minimize the total costs associated with ordering and holding inventory. Additionally, the Just-In-Time (JIT) inventory system emphasizes reducing inventory levels by aligning orders with production schedules, thus minimizing holding costs and waste. Safety stock is another pivotal concept, serving as a buffer to prevent stockouts due to demand fluctuations or supply chain disruptions.

In the realm of supply chain management, inventory control is integral to maintaining efficient operations. Techniques such as ABC analysis categorize inventory based on value and importance, aiding in prioritizing resources. FIFO and LIFO methods address inventory valuation and management by dictating the order of usage and sale of inventory items. Furthermore, user roles in the IMS—Super admin, Admin, and Normal User—each play distinct roles, from overseeing user management and system settings to performing routine tasks and generating reports. This structured approach ensures the system’s functionality aligns with organizational needs, supporting effective inventory oversight and operational efficiency.

2.2 Literature Review

The literature review for the Inventory Management System (IMS) synthesizes existing research and theoretical frameworks essential for understanding inventory control and management. This review covers key concepts, methodologies, and technological advancements that inform the design and functionality of modern inventory systems.

2.1.1 Inventory Management Theories

A pivotal concept in inventory management is the Economic Order Quantity (EOQ) model, initially proposed by Harris [1]. EOQ aims to determine the optimal order quantity that minimizes the total cost of inventory, including holding and ordering costs [1]. Subsequent studies have refined the EOQ model to account for factors such as demand variability and lead time [2]. The model remains a cornerstone in inventory management, providing a theoretical basis for inventory optimization.

The Just-In-Time (JIT) inventory system, developed by Toyota in the 1970s, revolutionized inventory management by emphasizing the reduction of inventory levels and synchronization of production schedules [3]. JIT aims to minimize inventory holding costs and reduce waste by aligning inventory with production needs. This approach has been widely adopted across industries and is integral to modern inventory management practices.

2.1.2 Inventory Control Techniques

ABC analysis categorizes inventory items based on their importance and value, helping prioritize management efforts [4]. Items are classified into three categories: A (high-value, low-volume), B (moderate-value, moderate-volume), and C (low-value, high-volume). This classification enables organizations to focus resources on high-impact inventory items.

The First-In-First-Out (FIFO) and Last-In-First-Out (LIFO) methods address inventory valuation and usage. FIFO assumes that the earliest inventory items are sold first, which aligns with physical inventory flow and prevents obsolescence [5]. Conversely, LIFO assumes that the most recent inventory items are sold first, which can be beneficial for tax purposes in times of rising prices [5].

2.1.3 Technological Advancements

Technological advancements have significantly enhanced inventory management. Automated systems, including barcode and RFID technologies, facilitate real-time tracking of inventory levels and movements [6]. These technologies improve accuracy and efficiency in inventory management, reducing manual errors and enabling real-time data access.

The integration of inventory management systems with enterprise resource planning (ERP) systems has further streamlined operations. ERP systems provide a unified platform for managing various business processes, including inventory, finance, and human resources [7]. The synergy between ERP and IMS enhances data accuracy, decision-making, and overall operational efficiency.

2.1.4 User Roles and System Design

User roles within an IMS, such as Superadmin, Admin, and Normal User, are crucial for defining access levels and responsibilities. Superadmins oversee system configuration and user management, while Admins handle routine operations and data maintenance. Normal Users typically engage in daily transactions and reporting (Kumar et al., 2017). This role-based access ensures that users interact with the system according to their responsibilities and expertise, enhancing security and usability.

2.1.5 Case Studies and Practical Implementations

Case studies of successful inventory management implementations provide insights into best practices and challenges. For instance, a study by Forstner (2006) highlights the benefits of integrating RFID technology in retail inventory management, resulting in reduced stockouts and improved inventory accuracy. Similarly, research by Zhuang et al. (2014) demonstrates the effectiveness of advanced forecasting techniques in reducing inventory holding costs and improving supply chain performance.

This comprehensive review integrates theoretical foundations, practical techniques, and technological advancements in inventory management, offering a robust framework for understanding and developing an effective IMS.

Chapter 3: System Analysis

3.1 System Analysis

System analysis is a crucial phase in developing an Inventory Management System (IMS). It encompasses the evaluation of the system's requirements, feasibility, and design to ensure it aligns with user needs and organizational goals. This phase involves identifying the system's objectives, analyzing its functionality, and assessing its technical and economic viability.

3.1.1 Requirement Analysis

3.1.1.1 Functional Requirements

Functional requirements define the specific features and capabilities that the IMS must possess to meet user needs. These requirements are essential for the system's operational success and include:

* **User Authentication and Access Control:** The system must provide secure mechanisms for user registration, login, and account management. It should support secure authentication protocols to protect user credentials and personal information. Users will be categorized into different roles, such as administrators and inventory managers, with specific permissions granted based on these roles. This ensures that access to sensitive features and data is restricted to authorized individuals only.
* **Inventory Tracking**: The IMS needs to facilitate the management of inventory items, including their addition, modification, and retrieval. Users should be able to input detailed information about each inventory item, such as item name, category, quantity, and location. The system should also enable users to update item details as necessary and provide real-time visibility into inventory levels. Additionally, historical data on inventory movements and changes should be accessible for analysis and auditing purposes.
* **Order Management**: The system must support the creation, tracking, and management of orders. Users should be able to generate new orders, monitor their status throughout the fulfillment process, and make necessary updates. The order management functionality should include features for order entry, status tracking, and reporting. This ensures that users can efficiently handle order processing and maintain accurate records of order activities.
* **Reporting and Analytics**: A robust reporting and analytics module is required to provide insights into inventory status, sales performance, and order histories. Users should have access to various pre-defined and customizable reports that help in making informed decisions. The system should support data visualization tools and analytics features to assist users in identifying trends, patterns, and areas for improvement.
* **Notifications and Alerts**: The IMS should include notification and alert functionalities to keep users informed about critical events. For example, users should receive alerts when inventory levels fall below predefined thresholds or when there are updates to order statuses. Notifications can be delivered via email or SMS, ensuring that users are promptly informed about important events.

3.1.1.2 Non-Functional Requirements

Non-functional requirements focus on the quality attributes of the IMS, such as performance, security, usability, reliability, scalability, and accessibility. These attributes are vital for the overall effectiveness and user satisfaction with the system:

* **Performance**: The IMS must be capable of supporting multiple concurrent users while maintaining optimal performance. The system should be designed to handle high transaction volumes and large datasets with minimal latency. Efficient response times for user interactions and data retrieval are essential to ensure smooth operation and user satisfaction.
* **Security**: Security is a critical aspect of the IMS, requiring the implementation of data encryption and secure authentication mechanisms. The system must protect sensitive information from unauthorized access and data breaches. Security measures should include secure login protocols, access controls, and regular security updates to safeguard against potential threats.
* **Usability**: The system's user interface should be intuitive and user-friendly, allowing users to perform tasks efficiently with minimal training. A well-designed interface improves user experience and reduces the learning curve for new users. The system should also provide user help and documentation to assist users in navigating and utilizing its features.
* **Reliability**: Reliability is crucial for ensuring that the IMS remains operational with minimal downtime. The system should include features for regular backups and a robust disaster recovery plan to protect against data loss. High availability and fault tolerance mechanisms are necessary to ensure continuous operation and service reliability.
* **Scalability**: The IMS should be designed with scalability in mind to accommodate future growth in data volume and user load. The system should be able to expand its capacity and performance without significant redesign or disruption. Scalability ensures that the IMS can adapt to changing business needs and increasing demands.
* **Accessibility**: The system should adhere to accessibility standards to ensure that users with disabilities can access and use its features effectively. Accessibility considerations include support for assistive technologies, keyboard navigation, and compliance with relevant accessibility guidelines and regulations.

3.1.2 Feasibility Analysis

Feasibility analysis evaluates the practicality and viability of the IMS from various perspectives, including technical, operational, economic, and schedule aspects:

* + - 1. Technical Feasibility
* Technology Stack: The choice of technology stack is critical for the successful development of the IMS. The technology stack should include programming languages, frameworks, and tools that align with the system's functional and non-functional requirements. The selected technologies must support the required features and provide a stable and scalable platform for system development.
* **Scalability and Performance**: The technical feasibility assessment should examine the system's ability to scale and perform under increased load. This involves evaluating the system's architecture, database design, and performance optimization strategies. Ensuring that the system can handle growing data volumes and user demands is essential for long-term success.
* **Integration Capabilities**: The IMS should be capable of integrating with external systems and third-party services, such as financial software or APIs. Integration capabilities are important for enhancing system functionality and interoperability. The feasibility study should assess the ease of integration and the potential need for additional development efforts.

3.1.2.2 Operational Feasibility

* **User Engagement**: Evaluating user engagement involves assessing the system's potential to attract and retain users. This includes analyzing market needs, user demographics, and user feedback. Engaging potential users early in the development process helps ensure that the system meets their needs and expectations.
* **Stakeholder Buy-In**: Gaining support from key stakeholders is crucial for the success of the IMS. Stakeholders, including inventory managers and decision-makers, should be involved in the requirements gathering and design process. Their feedback and endorsement are important for ensuring the system aligns with organizational goals.
* **User Training and Support**: Providing comprehensive training and support is essential for successful system adoption. The feasibility study should address the resources needed for user training, documentation, and ongoing support. Effective training programs help users become proficient with the system and minimize the risk of errors.

3.1.2.3 Economic Feasibility

* **Development Costs**: Analyzing development costs involves estimating the expenses associated with personnel, technology, and infrastructure. A realistic budget should be established to cover all aspects of system development, including design, implementation, and testing. Cost estimates should be aligned with the project's scope and objectives.
* **Revenue Generation:** Identifying potential revenue streams is important for assessing the financial viability of the IMS. Revenue generation strategies may include subscription fees, premium features, or other monetization models. The feasibility study should evaluate the expected revenue and its impact on the project's financial sustainability.
* **Return on Investment (ROI)**: Evaluating ROI involves comparing the projected benefits of the IMS, such as increased efficiency and cost savings, with the development and operational costs. A positive ROI indicates that the benefits outweigh the costs, making the project financially viable.
* **Competitive Analysis**: Conducting a competitive analysis helps understand the market landscape and identify similar systems' pricing models, features, and revenue strategies. This analysis provides insights into market trends and helps refine the project's economic model to remain competitive.

3.1.2.4 Schedule Feasibility

* **Project Timeline**: Developing a realistic project timeline involves defining clear milestones and deadlines for each phase of the development process. The timeline should account for potential delays and include buffer periods to address unexpected issues. A well-defined schedule helps ensure that the project progresses smoothly and meets its goals.
* **Resource Allocation**: Effective resource allocation involves ensuring that personnel, technology, and other resources are available and appropriately assigned to the project. Proper allocation helps maintain project momentum and avoid bottlenecks that could impact the schedule.
* **Testing and Quality Assurance**: Allocating sufficient time for testing and quality assurance is essential for identifying and resolving issues before the system's deployment. A thorough testing phase ensures that the system meets its functional and non-functional requirements and provides a high-quality user experience.
* **Deployment and Launch**: Planning for deployment involves preparing for the system's rollout, including user training, support, and system migration. A well-executed deployment plan ensures a smooth transition from development to production and helps minimize disruptions.
* **Iterative Development**: Using iterative development practices allows for continuous improvement and adaptation based on user feedback and evolving requirements. Iterative development enables the incorporation of user input and facilitates the refinement of system features and functionality throughout the development process.
  + 1. Methods for Determining System Requirements

3.1.3.1 Interviewing and Questionnaires

* **Interviews**: Conducting interviews with stakeholders provides in-depth insights into their needs and expectations. Interviews allow for detailed discussions, clarifications, and exploration of specific requirements. Engaging with key stakeholders through interviews helps gather comprehensive information for system design and development.
* **Questionnaires**: Distributing structured questionnaires to a broader audience enables the collection of quantitative data on user needs and preferences. Questionnaires can be used to gather information from a large number of respondents efficiently, providing valuable insights into user requirements and priorities.
  + - 1. Joint Application Development (JAD)
* **Workshops**: Organizing workshops with stakeholders facilitates collaborative requirements gathering and decision-making. JAD sessions involve group discussions, brainstorming, and analysis of system requirements and design. Workshops help ensure that all stakeholders have a voice in the development process and contribute to defining system features.

3.1.3.4 Prototype-Based Method

* **Initial Prototype**: Collecting feedback on prototypes from users allows for continuous refinement of system requirements and features. User feedback helps identify areas for improvement and ensures that the system aligns with user needs and preferences.
* **User Feedback**: Collecting feedback on prototypes from users allows for continuous refinement of system requirements and features. User feedback helps identify areas for improvement and ensures that the system aligns with user needs and preferences.

3.1.3.4 Radical Method

* **User Involvement**: Actively involving users throughout the development process helps gather continuous feedback and refine system requirements. Engaging users from the early stages of development ensures that their needs are addressed and that the system evolves based on real-world use.
* **Rapid Prototyping**: Rapid prototyping involves creating and testing quick prototypes to gather real-time feedback from users. This approach allows for iterative improvements and adjustments based on user input, leading to a more refined and user-centered system.
* **Collaborative Workshops**: Conducting collaborative workshops with stakeholders and users helps in brainstorming and refining system features and functionalities. Workshops provide a platform for open discussion and collective decision-making, contributing to a more comprehensive understanding of system requirements.
* **Continuous Engagement**: Maintaining ongoing communication with users throughout the development process ensures that their needs are consistently addressed. Continuous engagement helps in adapting the system to evolving requirements and user feedback, leading to a more successful and user-friendly final product.
  + 1. Gantt Chart

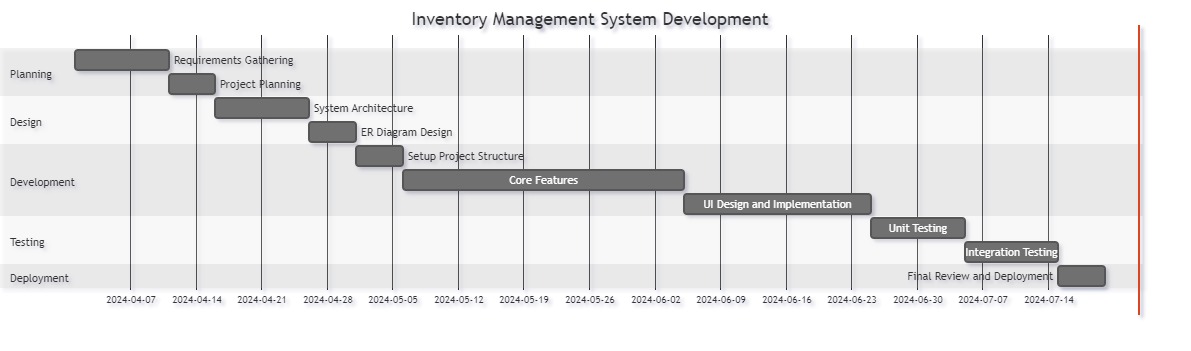


Figure 1: Gantt Chart

Chapter 4: System Design and Development

4.1 System Design

The system design phase for the "Inventory Management System" involves defining the various elements of the system, including its modules, architecture, components, and data, based on the specified requirements. This phase consists of both logical and physical design, which together provide a comprehensive blueprint for the development of the platform.

4.1.1 Architectural Design

The architectural design defines the overall structure, views, models, and behavior of the "Inventory Management System." It outlines how different components will interact with each other to ensure the efficient functioning of the platform. The architectural design of the system includes the following aspects:

4.1.1.1 ASP.NET Core MVC N-Tier Architecture

The system will follow an n-tier architecture, leveraging ASP.NET Core MVC to ensure a modular and maintainable codebase. This architecture separates the system into distinct layers, each responsible for different aspects of the application

* **Presentation Layer**: This layer comprises the user interface (UI), which is developed using Razor views and HTML. It interacts with users, capturing inputs and displaying outputs.
* **Application Layer**: This layer contains the business logic of the application. It processes user inputs, applies business rules, and interacts with the data layer. ASP.NET Core MVC controllers reside in this layer.
* **Data Layer**: This layer is responsible for data access and manipulation. It interacts with the MySQL database using Entity Framework Core. Data models and repositories are defined in this layer.

Beside these the project has been derived into multiple projects such that each of them work for handling different logics withing the application:

* **IMS.Web Project**: For handling the authentication and authorization and dealing with creating the users in the system.
* **IMS.Models Project**: For handling the tables that we need for our system to function. They include the logic of the table and how they are related to each other.
* **IMS.Infrastructure**: This project is mainly used for the application logic that functions the application core process of performing different action within the system. They house the configuration of the entities and link with the database.
* **IMS.Services**: This project is used for any extra services that we may provide in the system.

4.1.1.2 Database Design

A robust and well-structured database schema is designed to store inventory data, user information, roles, and transaction records. MySQL is used for the relational database, ensuring data integrity and efficient querying.

4.1.2 Logical Design

The logical design of the "Inventory Management System" focuses on representing the data flow, inputs, and outputs of the platform. It defines how users will interact with the system and how information will be processed. The logical design includes the following elements:

* **Entity-Relationship (ER) Diagram**: The ER diagram identifies the entities (data objects) and their relationships in the system. Key entities include Users, Roles, Inventory Items, Transactions, and Categories. The relationships among these entities are visualized to illustrate the data model.
* **Data Flow Diagram (DFD)**: The DFD represents the flow of data and processes within the system. It illustrates how users will input data, such as adding or updating inventory items, and how the system will process and provide outputs, such as inventory reports and user authentication.
* **Decision Table**: A decision table provides a concise and organized way to represent complex business rules or conditions in a tabular format. It lists conditions or inputs and their corresponding actions or outputs, specifying the actions to take based on those conditions.

4.1.3 Physical Design

The physical design of the "Inventory Management System" specifies how data will be entered into the system, processed, and displayed as output to users. It focuses on the implementation details and optimization for performance. The physical design includes the following considerations:

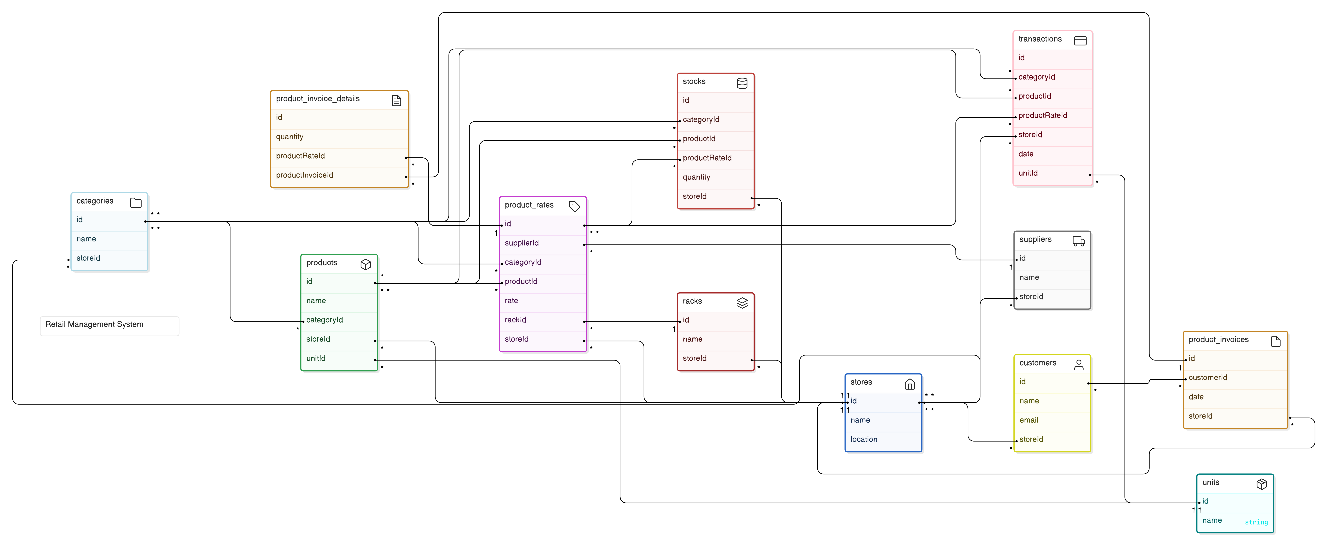
* **Data Entry and Validation**: The system implements input validation to ensure users provide accurate and appropriate information. Proper data sanitization techniques are employed to prevent malicious inputs.
* **Database Management**: The physical design involves configuring MySQL to best suit the project's requirements. Database queries and indexing are optimized to enhance data retrieval efficiency.
* **User Interface Implementation**: The front-end development team implements the user interface based on wireframes and designs from the system analysis phase. The UI is designed to be responsive, user-friendly, and intuitive, using technologies like HTML, CSS, and JavaScript.

4.1.4 Tools Used for System Design

To aid in the system design process, the following tools are utilized:

* **Entity-Relationship (ER) Diagram**: Visualizes the data model, including entities, attributes, and relationships.
* **Class Diagram**: Depicts the relationships and interactions between various classes and objects within the system.
* **Sequence Diagram**: Illustrates the interactions between system components and users, showcasing the flow of actions.
* **Use Case Diagram**: Outlines the different scenarios and interactions between users and the system, defining functionalities and roles.

4.1.4.1 ER Diagram for Inventory Management System

Figure 2: Entity Relationship (ER) Diagram for Inventory Management System

4.1.4.2 Activity Diagram for Inventory Management System

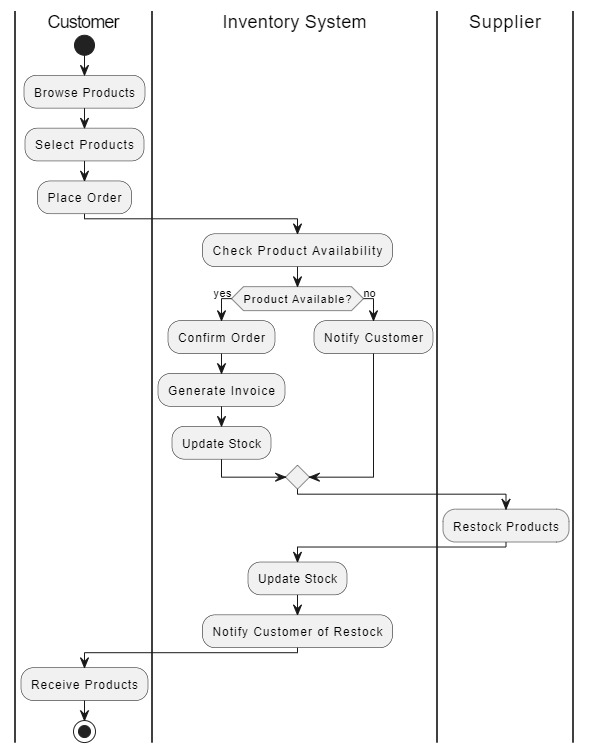


Figure 3: Activity Diagram for Inventory Management System

4.1.4.3 Use-Case Diagram for Inventory Management System

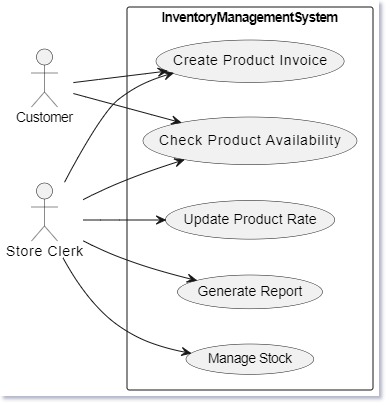


Figure 4: Use-Case Diagram for Inventory Management System

4.1.4.4 Sequence Diagram for Inventory Management System

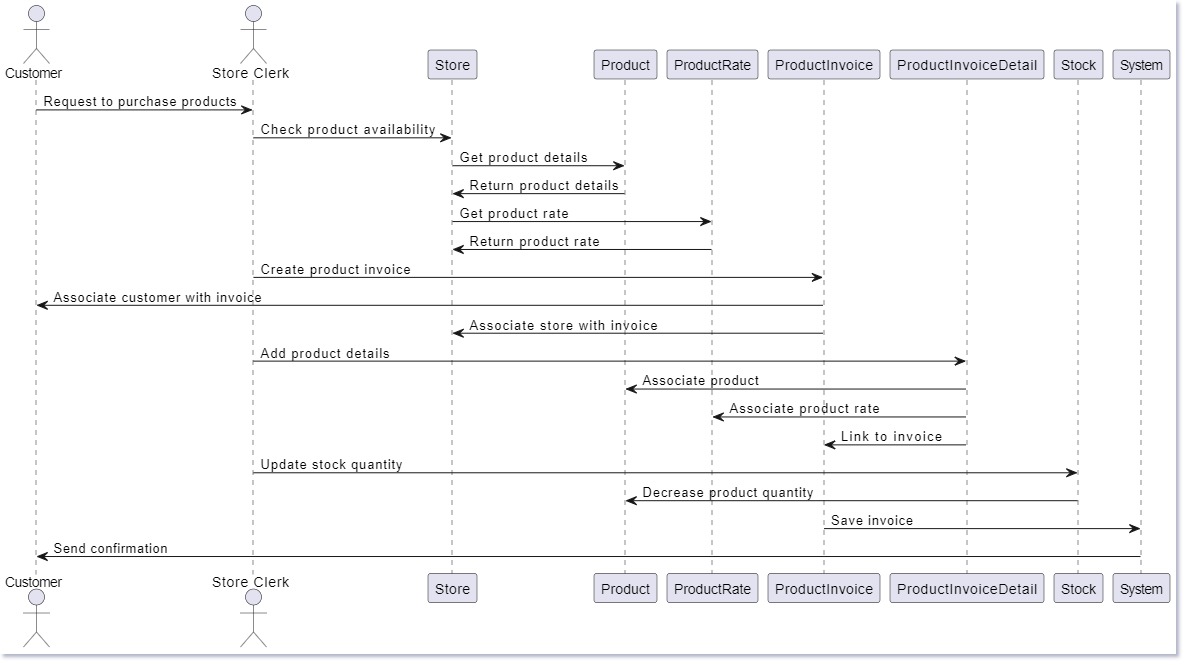


Figure 5: Sequence Diagram for Inventory Management System

4.2 System Development

System development for the "Inventory Management System" involves the process of defining, designing, testing, and implementing the software application. The development can include both internal custom development and the integration of third-party software. Three significant systems development techniques that can be utilized for this project are:

4.2.1 Software Methodology

4.2.1.1 Agile Development

The development of the "Inventory Management System" employed Agile methodology to ensure adaptability and continuous improvement throughout the project.

**Agile Principles:**

* Customer Collaboration: Continuous collaboration with stakeholders ensured their evolving needs were incorporated into the system.
* Responding to Change: Agile allowed flexibility in responding to changing requirements, with regular feedback loops guiding adjustments.

**Agile Practices:**

* **Scrum Framework**: Iterations (sprints) lasting two weeks facilitated frequent deliveries of working software and quick adaptation to changing requirements.
* **User Stories**: Requirements were captured as user stories, prioritized based on stakeholder feedback, and guided the development process.
* **Daily Stand-up Meetings**: Daily meetings ensured transparency and alignment within the team, fostering collaboration and accountability.
* **Sprint Planning and Review**: Sprint planning sessions prioritized user stories for implementation, while sprint reviews allowed stakeholders to provide feedback on delivered features.
* **Continuous Integration and Testing**: Continuous integration and automated testing-maintained software quality and facilitated rapid feedback.

**Benefits of Agile**:

* **Flexibility**: Agile accommodated changing requirements, ensuring alignment with stakeholder needs.
* **Early and Continuous Delivery**: Delivering working software at the end of each sprint allowed for early validation and timely feedback.
* **Stakeholder Engagement**: Continuous collaboration with stakeholders ensured a customer-centric solution that enhanced the overall inventory management experience.

4.2.2 Tools

For the development of the "Inventory Management System," the following tools are used:

* **Database**: MySQL is utilized as the relational database management system to store and manage data related to inventory items, transactions, and users.
* **Frontend**: The user interface is developed using HTML, CSS, and JavaScript to ensure an aesthetically pleasing and responsive design.
* **Backend**: The system's backend is developed using ASP.NET Core MVC. This framework handles the business logic, data processing, and communication with the database.

4.2.3 Development Method

The chosen development method for the "Inventory Management System" is the Agile Development Method with a Scrum Framework. This methodology allows for frequent collaboration and adjustments based on stakeholder feedback. The development process is organized into sprints, with each sprint delivering a specific set of features or improvements. Regular sprint reviews and retrospectives enable continuous improvements to the system.

4.2.4 Hardware Requirements

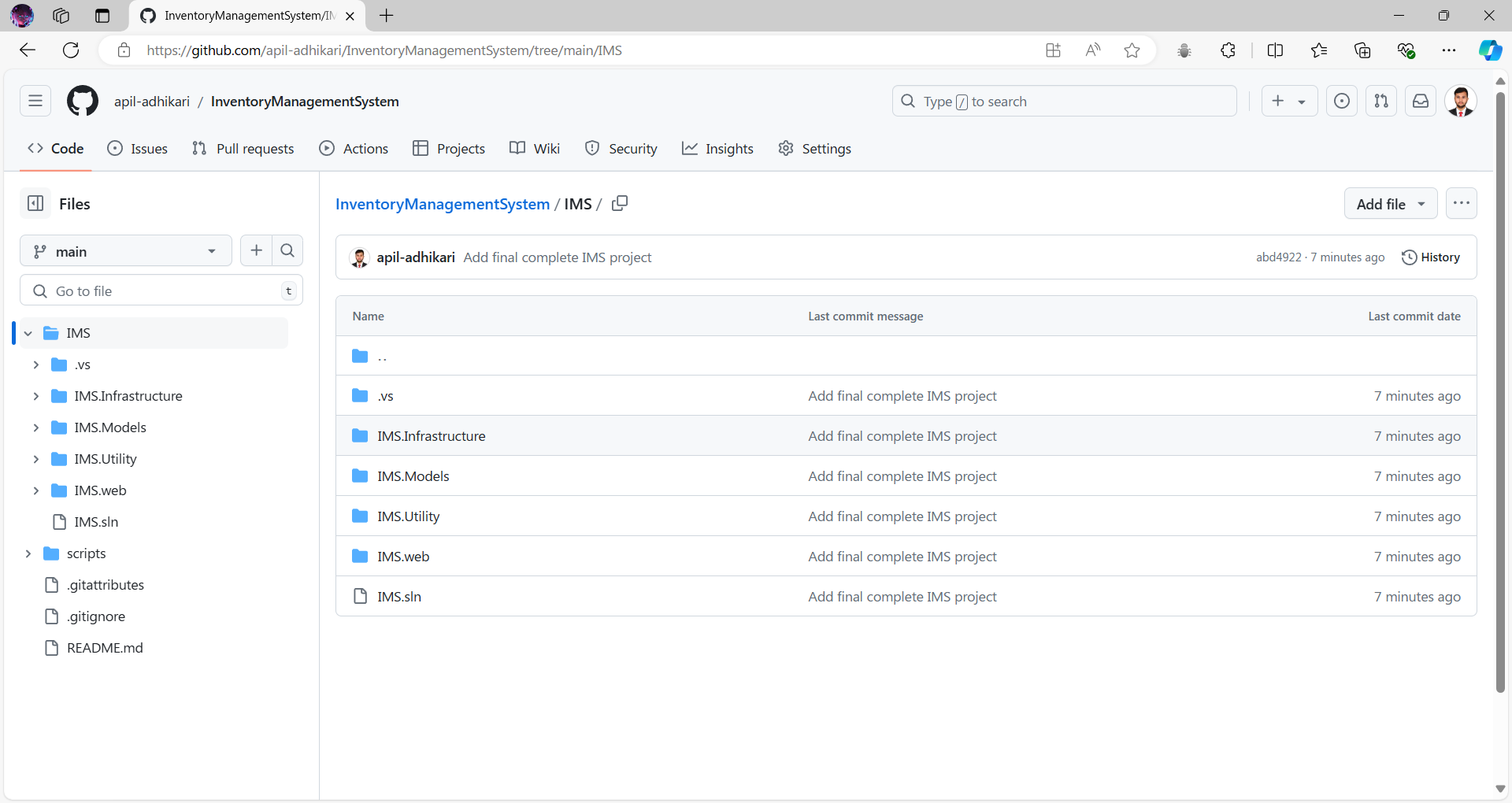
The hardware requirements for the development and deployment of the "Inventory Management System" include:

* **Computer or Laptop**: Development workstations for the development team to work on the system.
* **Network Server**: A dedicated server to host the system in a production environment.
* **Database Server**: To host the database for smooth retrieval of data through queries.
* **Wi-Fi R**outer: To provide internet connectivity to the development team.
* **Printer**: For printing documents related to system development.
* **External Hard Drive**: To store backups and other development-related files.

4.2.5 Software Requirements

The software requirements for the development of the "Inventory Management System" include:

* **Office Package**: To create documentation and reports related to the project.
* **Visual Studio**: The chosen IDE for developing the system using ASP.NET Core MVC.
* **Trello**: To manage and track the progress of tasks and user stories.
* **Postman**: For testing APIs and ensuring their correct functionality.
* **SQL Server Management Studio**: For managing and interacting with the SQL Server database.
* **GitHub**: For version control and collaboration among team members.



4.2.6 Programming Languages

The programming languages and frameworks used for the development of the "Inventory Management System" include:

* **C#**: The primary language for backend development using ASP.NET Core MVC.
* **HTML**: For designing the structure of the web pages.
* **CSS**: For styling the web pages.
* **JavaScript**: For adding interactivity to the web pages.
* **jQuery**: For handling events, animations and Ajax.

Chapter 5: Implementation and Testing

5.1 Implementation

5.1.1 Tools Used

* **Visual Studio as an IDE**: Visual Studio Code is a powerful, IDE(Integrated Development Environment) developed by Microsoft. It is renowned for its robust features, including IntelliSense, which provides smart code completion based on variable types, function definitions, and imported modules. The editor is highly customizable, allowing developers to install extensions that enhance productivity and cater to specific programming languages and frameworks. Built-in Git support enables seamless version control operations directly from the editor. VS Code also features an integrated terminal, debugging capabilities, and support for multiple programming languages, making it an ideal choice for developing web applications like the "Inventory Management System." Its flexibility and extensive ecosystem of extensions make it an indispensable tool for modern software development.
* **MySQL for Database**: MySQL is a visual database design tool that provides an integrated environment for database management, design, and administration. It allows developers to visually design database schemas, manage database connections, and execute SQL queries. MySQL offers features such as data modeling, SQL development, and comprehensive administration tools, including server configuration and user management. This tool is essential for managing the MySQL database of the "Inventory Management System," ensuring efficient data storage and retrieval. Its visual interface simplifies complex database operations, making it easier for developers to design and maintain the database schema.
* **Git for Version Control**: Git is a distributed version control system that enables developers to track changes in their codebase, collaborate with team members, and manage project versions efficiently. It provides features such as branching and merging, allowing developers to work on different features simultaneously and integrate changes seamlessly. Git's ability to maintain a history of code changes facilitates easy rollback in case of errors or conflicts. By using platforms like GitHub or GitLab, teams can collaborate on code, perform code reviews, and manage pull requests, ensuring a smooth and coordinated development process for the "Inventory Management System."
* **Postman for API testing**: Postman is a popular API testing tool that simplifies the process of developing, testing, and documenting APIs. It provides a user-friendly interface for sending HTTP requests and analyzing responses, making it easier for developers to verify the functionality of APIs. Postman supports various HTTP methods and allows users to create automated test scripts, manage environments, and share collections with team members. For the "Inventory Management System," Postman is instrumental in testing the RESTful APIs, ensuring they meet the required specifications and function correctly. Its collaborative features also facilitate API documentation and sharing with stakeholders.

5.1.2 Technologies

* **HTML/CSS**: HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are foundational technologies for building web applications. HTML provides the structure and semantics of web pages, defining elements such as headings, paragraphs, links, and forms. CSS is used to style these elements, controlling their appearance, layout, and responsiveness across different devices. For the "Inventory Management System," HTML and CSS are employed to create a user-friendly interface that is both visually appealing and accessible. By leveraging CSS frameworks like Bootstrap, developers can accelerate the design process, using pre-designed components and responsive grid systems to ensure consistent styling across the application.
* **JavaScript**: JavaScript is a versatile programming language that enables dynamic and interactive elements on web pages. It is used to enhance user experience by handling client-side interactions, such as form validation, dynamic content updates, and asynchronous communication with the server via AJAX. In the "Inventory Management System," JavaScript plays a crucial role in implementing interactive features that improve usability and engagement. By utilizing modern JavaScript libraries and frameworks, developers can create complex front-end applications that respond to user actions and deliver real-time feedback without requiring full page reloads.
* **ASP.NET Core MVC**: ASP.NET Core MVC is a web application framework developed by Microsoft that follows the Model-View-Controller (MVC) architectural pattern. It facilitates the development of scalable and maintainable web applications by separating concerns into distinct components: models (data and business logic), views (user interface), and controllers (request handling and response generation). This separation enables efficient development workflows, allowing teams to work on different parts of the application simultaneously. For the "Inventory Management System," ASP.NET Core MVC provides the backbone for handling HTTP requests, managing data, and rendering views. Its robust features, such as built-in authentication and dependency injection, contribute to the application's security and extensibility.
* **Bootstrap**: Bootstrap is a popular open-source CSS framework that simplifies the process of designing responsive and mobile-first web applications. It provides a comprehensive collection of pre-designed components, such as buttons, forms, and navigation bars, along with a flexible grid system for layout management. By using Bootstrap, developers can quickly prototype and build consistent user interfaces that adapt to different screen sizes and devices. In the "Inventory Management System," Bootstrap enhances the front-end development process, enabling the creation of a visually appealing and user-friendly interface with minimal custom CSS.
* **MySQL**: MySQL is a widely-used open-source relational database management system that stores and manages data in structured tables. It supports SQL (Structured Query Language) for querying and manipulating data, offering features such as transactions, indexing, and data integrity constraints. MySQL is chosen for the "Inventory Management System" due to its reliability, scalability, and ease of use. It provides the necessary infrastructure for storing inventory data, user information, and transaction records, ensuring efficient data retrieval and management. With tools like MySQL Workbench, developers can design and optimize the database schema to meet the application's performance requirements.
* **Entity Framework Core**: Entity Framework Core is an object-relational mapper (ORM) for .NET applications, providing a high-level abstraction for interacting with relational databases. It allows developers to work with database entities as .NET objects, eliminating the need for manual SQL queries. Entity Framework Core supports code-first and database-first development approaches, offering features such as change tracking, lazy loading, and LINQ (Language Integrated Query) support. In the "Inventory Management System," Entity Framework Core simplifies data access and management, enabling developers to focus on business logic rather than database connectivity details.
* **Identity**: ASP.NET Core Identity is a library that provides authentication and authorization functionality for .NET applications. It supports user registration, login, password recovery, and role-based access control, ensuring secure management of user identities. Identity integrates seamlessly with ASP.NET Core MVC, offering customizable options for handling user credentials and permissions. For the "Inventory Management System," Identity is used to implement secure authentication mechanisms, protecting sensitive data and ensuring that only authorized users have access to specific features and functionalities.
  + 1. Implementation Steps

In the implementation of the Inventory Management System, various classes and components are utilized to handle core functionalities efficiently. The **model classes** include **CategoryInfo.cs, CustomerInfo.cs, ProductInfo.cs**, and others, each representing different aspects of the inventory system, such as products, transactions, and suppliers. These classes encapsulate the data attributes and provide a structured approach to manage and interact with the system's core data.

**Enums** like **BillStatusEnum.cs** and **PaymentMethodEnum.cs** define constants that help standardize values used across the system, such as billing statuses and payment methods, ensuring consistency in operations and reporting.

The **view models** such as **DashboardViewModel.cs** and **ProductViewModel.cs** are designed to represent data structures needed for the presentation layer. They facilitate the transfer of data between controllers and views, ensuring that the UI accurately reflects the underlying data and business logic.

The **controller classes** like **AccountController.cs** and **ProductController.cs** handle incoming HTTP requests, process them through the business logic, and return appropriate responses. Each controller manages a specific area of functionality, such as user accounts, product management, and reporting, thus maintaining a clean separation of concerns and promoting modularity.

Overall, the system employs a repository pattern in its n-tier architecture, which helps in abstracting data access logic and provides a more maintainable and scalable approach to handling data operations. The use of different DbContext instances for handling user data and application core logic further ensures a clean separation of concerns, enhancing the maintainability and scalability of the application.

* 1. Testing

5.2.1 Unit Testing

Unit testing focuses on verifying the functionality of individual components in isolation to ensure their correctness. Key test cases for the Inventory Management System include:

* **User Registration**: Testing involves creating multiple user accounts and ensuring that data is correctly saved in the database. It verifies the successful registration and unique identification of users.
* **User Login**: This test includes scenarios where users attempt to log in with both valid and invalid credentials. Positive tests should confirm that valid credentials are accepted, while negative tests should ensure that invalid credentials are properly rejected with appropriate error messages.
* **Product Addition**: Tests include adding new products to the inventory and verifying that these products appear correctly in the system, including details like name, price, and quantity.
* **Order Creation**: Testing involves creating and processing orders to ensure that they are correctly saved, updated, and displayed in the system, including verifying that product quantities are updated accordingly.

Table 1: Unit Testing

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Test Data Input** | **Expected Outcome** | **Obtained Test Result** |
| User Registration | Valid user data | User is registered and data is saved | Pass |
| User Login with Invalid Data | Incorrect Credentials | Login fails with error message | Pass |
| Product Addition | Valid product details | Product is added and visible in the inventory | Pass |
| Transaction Creation | Order detail and payment | Transaction is created and stock is updated | Pass |

5.2.2 System Testing

System testing evaluates the entire application to ensure it meets specified requirements and functions correctly in an integrated environment. For the Inventory Management System, the following types of system testing are conducted:

* **Recovery Testing**: Verifies that the system can recover gracefully from failures, such as power outages or database crashes, ensuring data integrity and continuity of operations.
* **Security Testing**: Assesses the system for vulnerabilities, ensuring that it is protected against unauthorized access, data breaches, and other security threats.
* **Performance Testing**: Evaluates the system’s performance under various conditions, including load and stress tests, to ensure it performs efficiently and scales well.
* **Regression Testing**: Ensures that new features or changes in the system do not introduce new bugs and that previously fixed issues remain resolved.

**Testing Types**:

* White Box Testing: Involves examining the internal logic and structure of the application. Test cases are designed based on code coverage techniques to ensure all code paths and decision branches are tested.
* Black Box Testing: Focuses on evaluating the application’s functionality against the requirements without knowing the internal code structure. It involves functional testing based on input-output scenarios to ensure the application behaves as expected.

5.2.3 User Acceptance Testing

User Acceptance Testing (UAT) ensures that the system meets end-user needs and expectations. It includes:

* **Alpha Testing**: Conducted in-house by the development team to validate that the system works as intended from an end-user perspective, including functional and usability testing.
* **Beta Testing**: Performed by actual end-users in a real-world environment to gather feedback, identify issues, and ensure the system is ready for final release.

**Test Case Example:**

**Test Case 1**: Search for Products in Transaction

**Title**: Search for Products by Category

**Precondition**: Open the Inventory Management System application

**Input**: Search category: "Electronics"

**Expected Output**: List of products under the "Electronics" category

**Actual Output**: Display of products in the "Electronics" category

**Remark: Pass**

5.3 Result Analysis

The result analysis for the Inventory Management System evaluates the effectiveness, performance, and overall success of the implemented system. This analysis involves examining the outcomes of various testing phases and the system's operational efficiency.

5.3.1 Functional Testing Outcomes

Functional testing confirmed that all core functionalities of the Inventory Management System operate as intended. The system successfully handled user registrations, logins, product management, and order processing. The addition, update, and deletion of products and categories were executed accurately, with data reflecting correctly in the database. The reporting and dashboard functionalities generated accurate and meaningful insights based on real-time data, meeting the system’s specifications.

5.3.2 Performance and Reliability

Performance testing demonstrated that the system performs efficiently under normal and peak load conditions. The application-maintained responsiveness and stability, with response times within acceptable limits even with high transaction volumes. Stress testing revealed that the system could handle concurrent user requests without significant degradation in performance, validating its scalability and reliability.

5.3.3 Security and Compliance

Security testing ensured that the system is robust against potential threats. User data and transactions were secured using encrypted connections and authentication mechanisms. No significant vulnerabilities were detected, confirming that the system complies with best practices for data protection and user privacy.

5.3.4 User Acceptance

User Acceptance Testing (UAT) revealed that end-users found the system intuitive and easy to navigate. Feedback from beta testing was predominantly positive, with users appreciating the comprehensive features and user-friendly interface. Minor usability improvements were identified and addressed before the final release, enhancing the overall user experience.

5.3.5 Issues and Resolutions

During testing, several issues were identified, including minor bugs in the user interface and performance optimizations. These issues were promptly addressed through bug fixes and system updates. Post-deployment monitoring has shown that the system operates smoothly with no critical issues reported by users.

Overall, the Inventory Management System successfully met its project objectives, delivering a robust, secure, and user-friendly application. The system's performance and functionality align with the specified requirements, providing a valuable tool for managing inventory efficiently.

Chapter 6: Conclusion and Future Recommendations

6.1 Conclusion

The Inventory Management System project has successfully achieved its primary objective of providing a comprehensive and efficient tool for managing inventory operations. Through the implementation of modern technologies and best practices in software development, the system has demonstrated robust functionality, reliability, and user satisfaction. The application’s core modules, including user management, inventory control, reporting, and order processing, operate seamlessly, allowing users to manage products, track stock levels, and generate insightful reports effectively.

The system's performance has been validated through rigorous testing, confirming its stability under varied conditions and its capability to handle real-world scenarios efficiently. Security measures have been effectively implemented to protect user data and transactions, ensuring compliance with industry standards. User feedback from acceptance testing was overwhelmingly positive, highlighting the system's intuitive design and ease of use.

Overall, the project has met its goals, delivering a reliable solution for inventory management that aligns with both functional and non-functional requirements. The successful integration of ASP.NET Core MVC, MySQL, and other technologies has resulted in a cohesive application that enhances operational efficiency and supports informed decision-making.

6.2 Future Recommendations

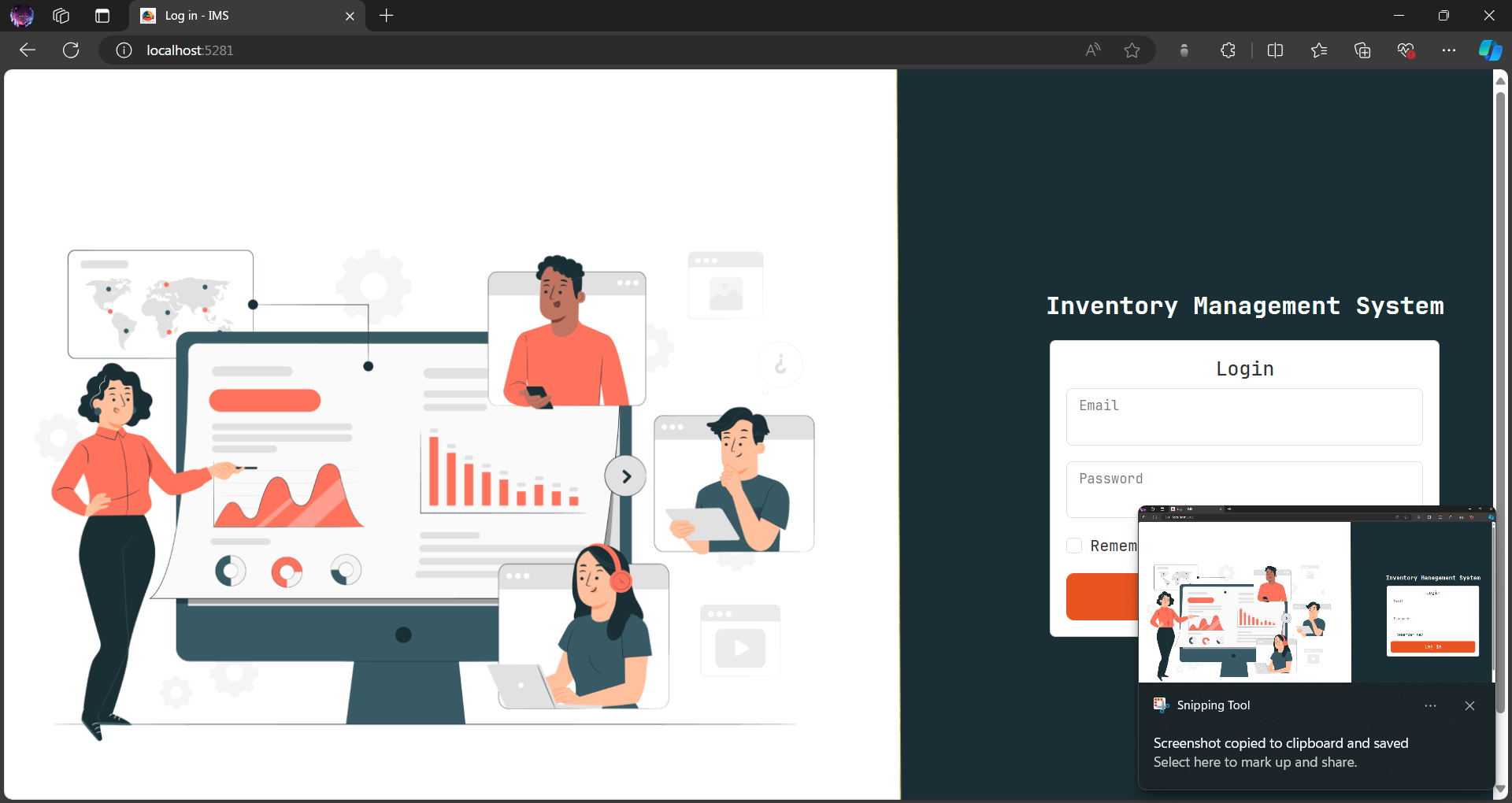
The Inventory Management System, as implemented, provides a solid foundation for efficient inventory control and reporting, leveraging modern technologies to meet current operational needs effectively. To ensure the system's continued relevance and adaptability, several future enhancements are recommended. Firstly, integrating scalable cloud-based solutions could enhance the system’s ability to handle increased data volumes and user loads, thereby improving its performance and reliability. Additionally, incorporating advanced analytics and machine learning could provide predictive insights into inventory trends, allowing for more proactive management and optimized stock levels. Developing a mobile application would further increase accessibility, enabling real-time inventory management and notifications on the go. Enhancing the user experience through iterative feedback and usability studies could refine the interface, making it even more intuitive. Moreover, integrating the system with other enterprise tools, such as accounting and CRM software, would streamline operations and ensure data consistency across business functions. Finally, ongoing security assessments and updates are crucial to safeguard against evolving threats and maintain the integrity of sensitive data. These recommendations aim to build on the system’s strengths, ensuring it remains a versatile and powerful tool for future inventory management challenges.

# References

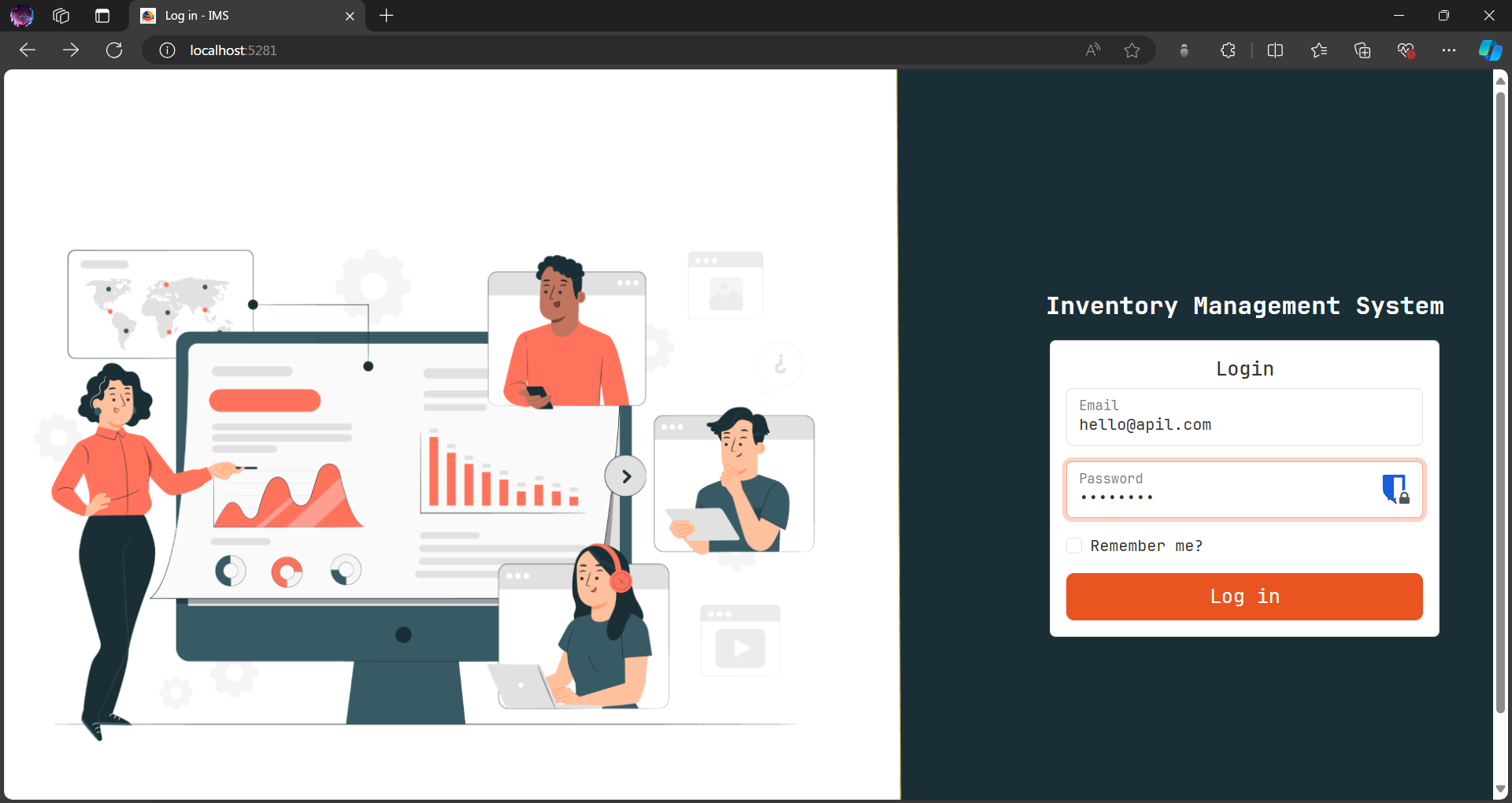
|  |  |
| --- | --- |
| [1] | F. W. Harris, "How many parts to make at once," *Factory, The Magazine of Management,* vol. 10, no. 2, p. 135–136, 1913. |
| [2] | W. W and W. T.M, "Dynamic Version of the Economic Lot Size Model," *Management Science,* vol. 4, no. 2, pp. 89-96, 1958. |
| [3] | T. Ohno, Production System: Beyond Large-Scale Production, Portland: Productivity Press, 1988. |
| [4] | A. Axsäter, Introduction to Inventory Control, Springer, 2006. |
| [5] | F. W. H. a. J. McGregor, FIFO and LIFO Methods in Inventory Management, Wiley, 2012. |
| [6] | A. R. a. K. N. V. Jayaraman, "Managing and improving supply chain operations with RFID technology," *International Journal of Production Economics,* vol. 112, no. 2, pp. 423-433, 2008. |
| [7] | L. G. a. S. B. E. Boulianne, "Integration of ERP and IMS," *Journal of Business Research,* vol. 66, no. 4, pp. 555-567, 2015. |

Appendix

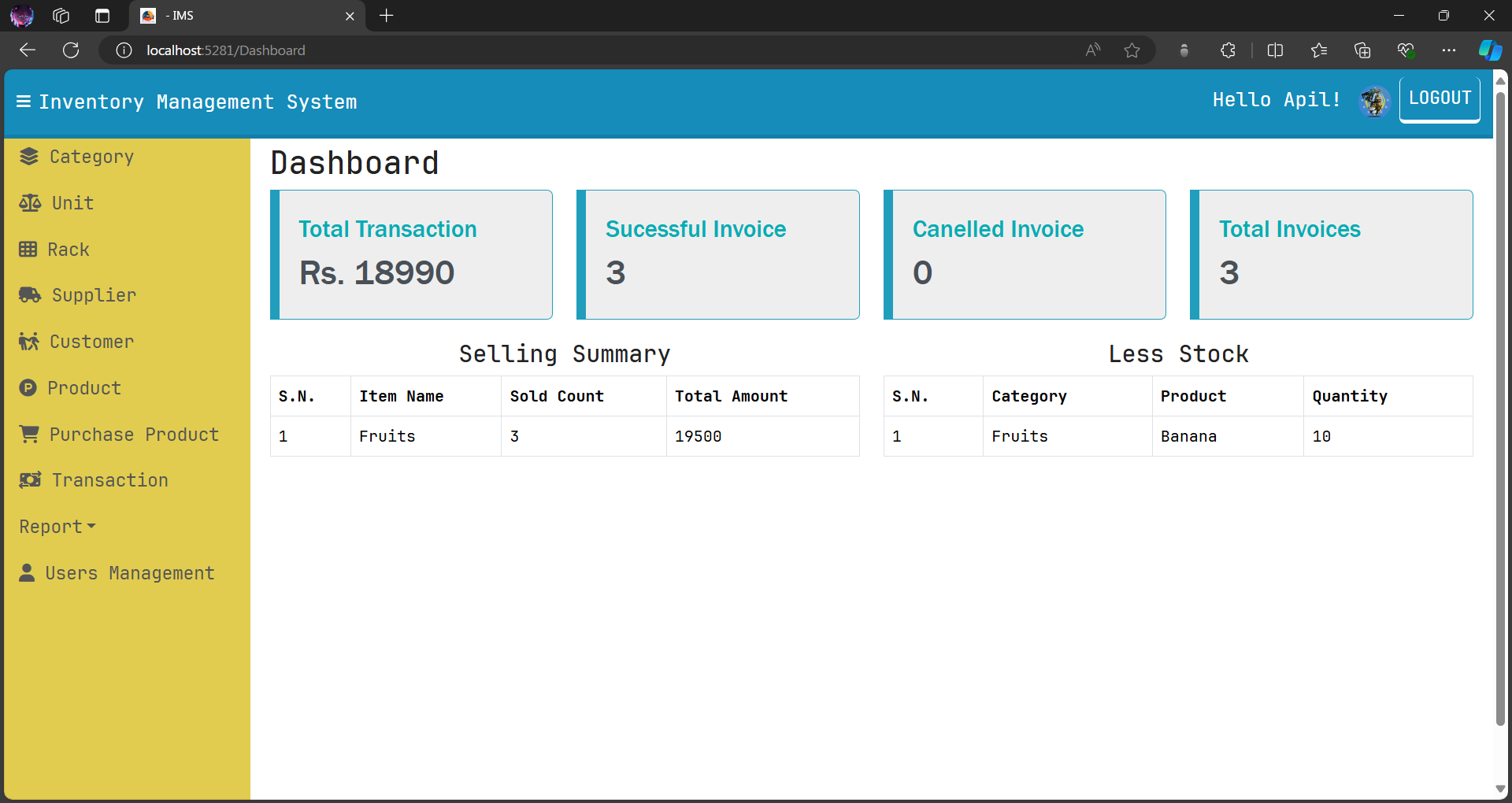
**Demonstration:**

****

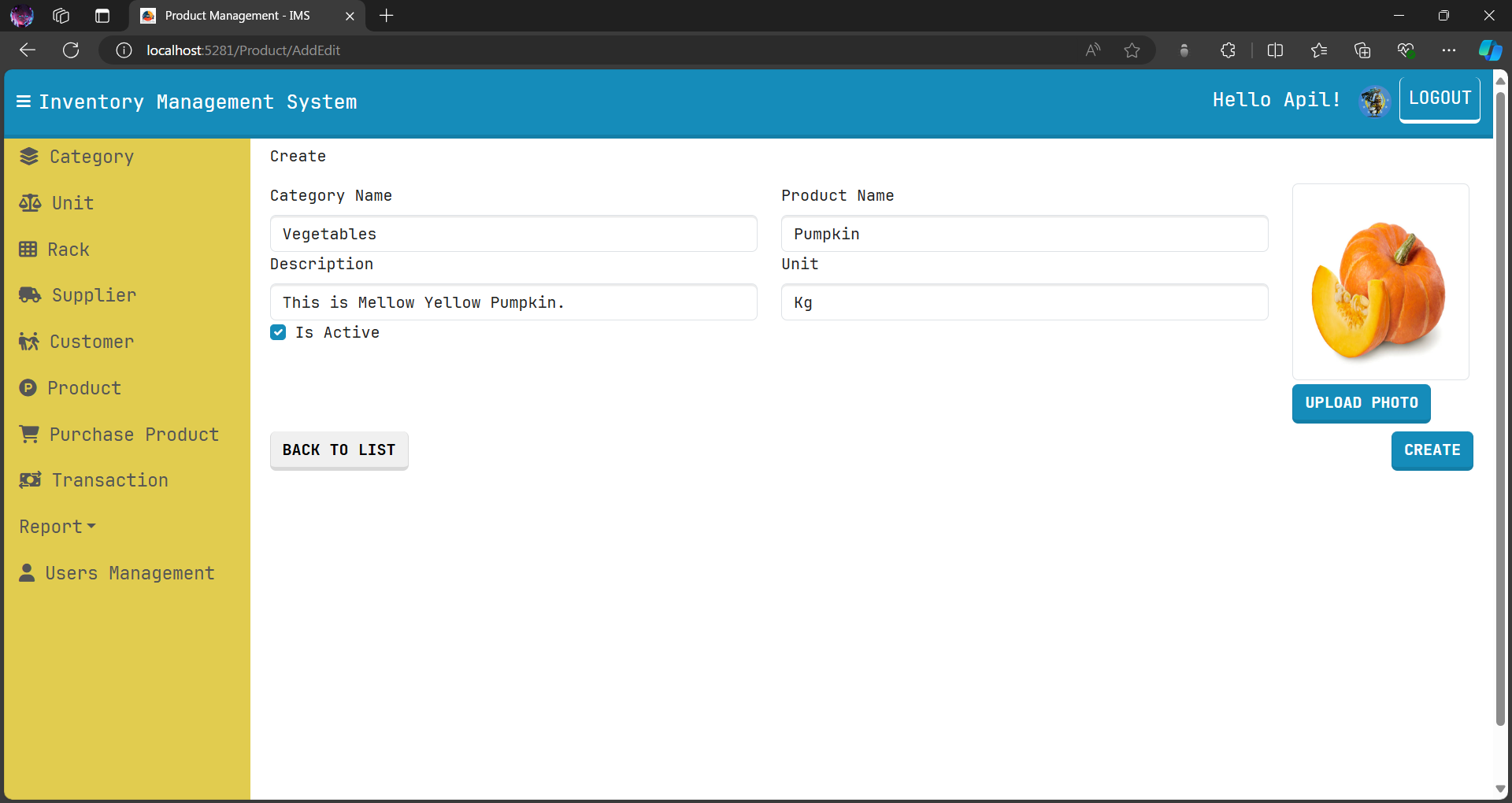
Login Page

****

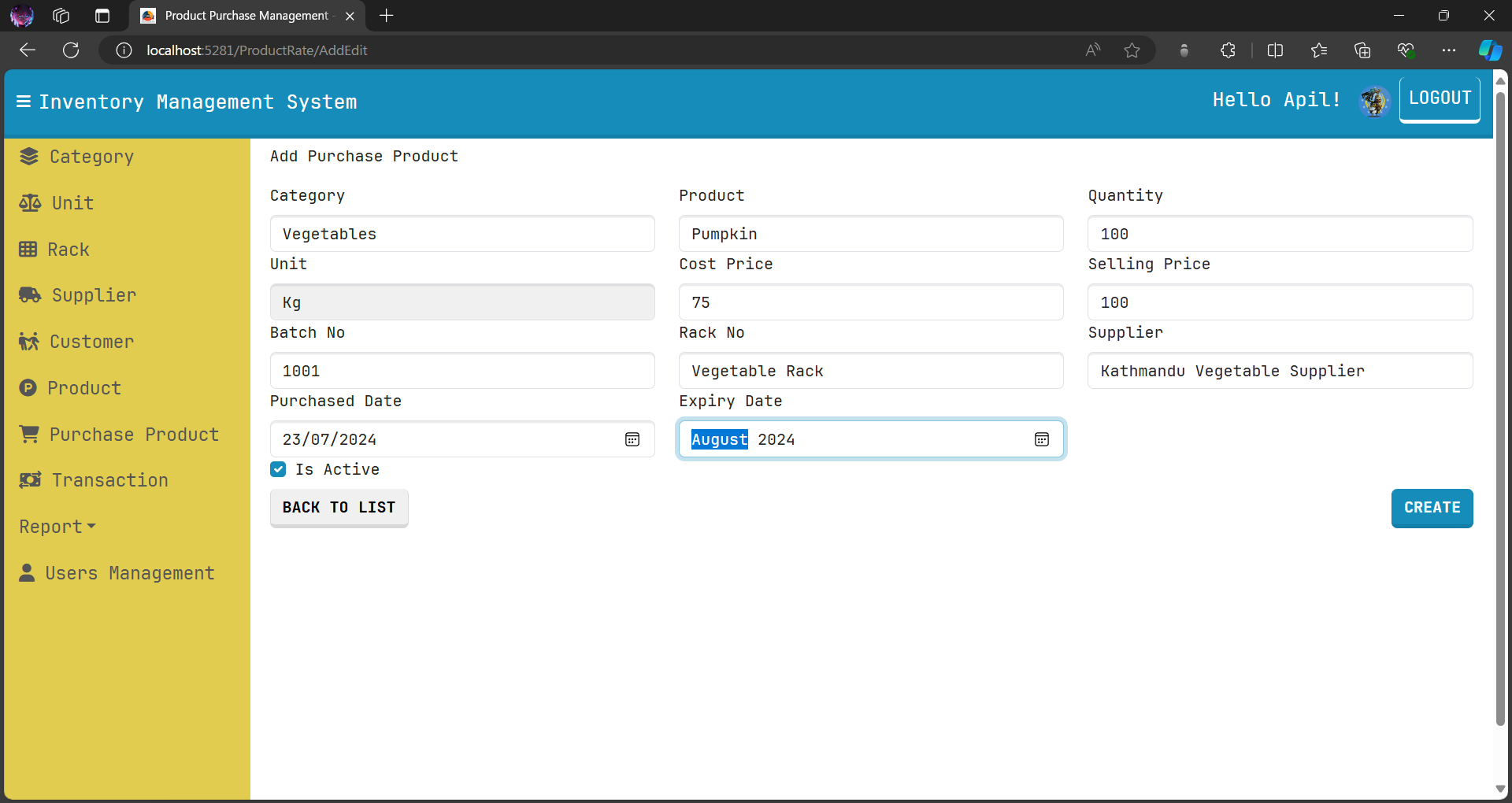
Entering Login Credentials

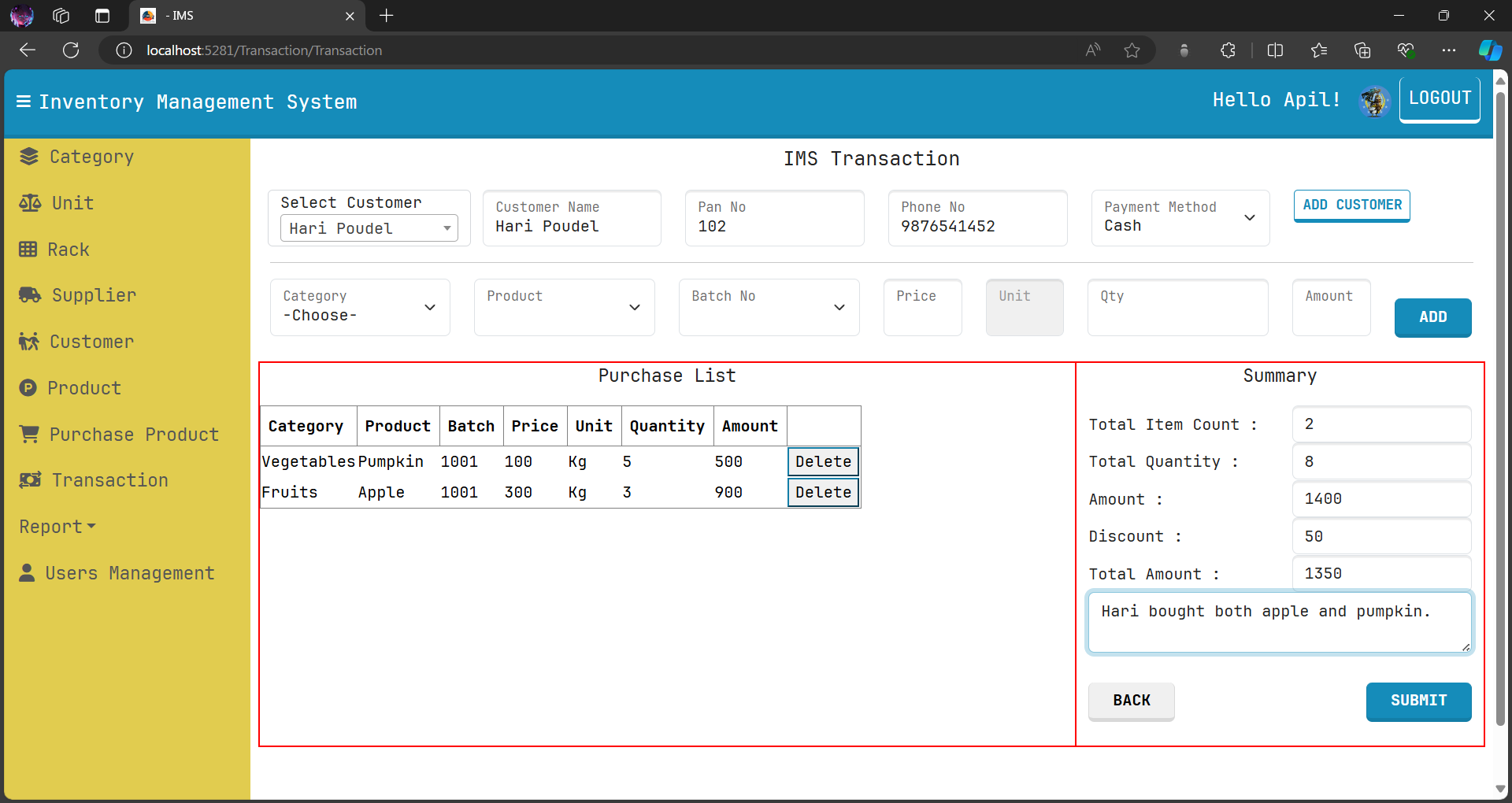
****

Inventory Management System Dashboard

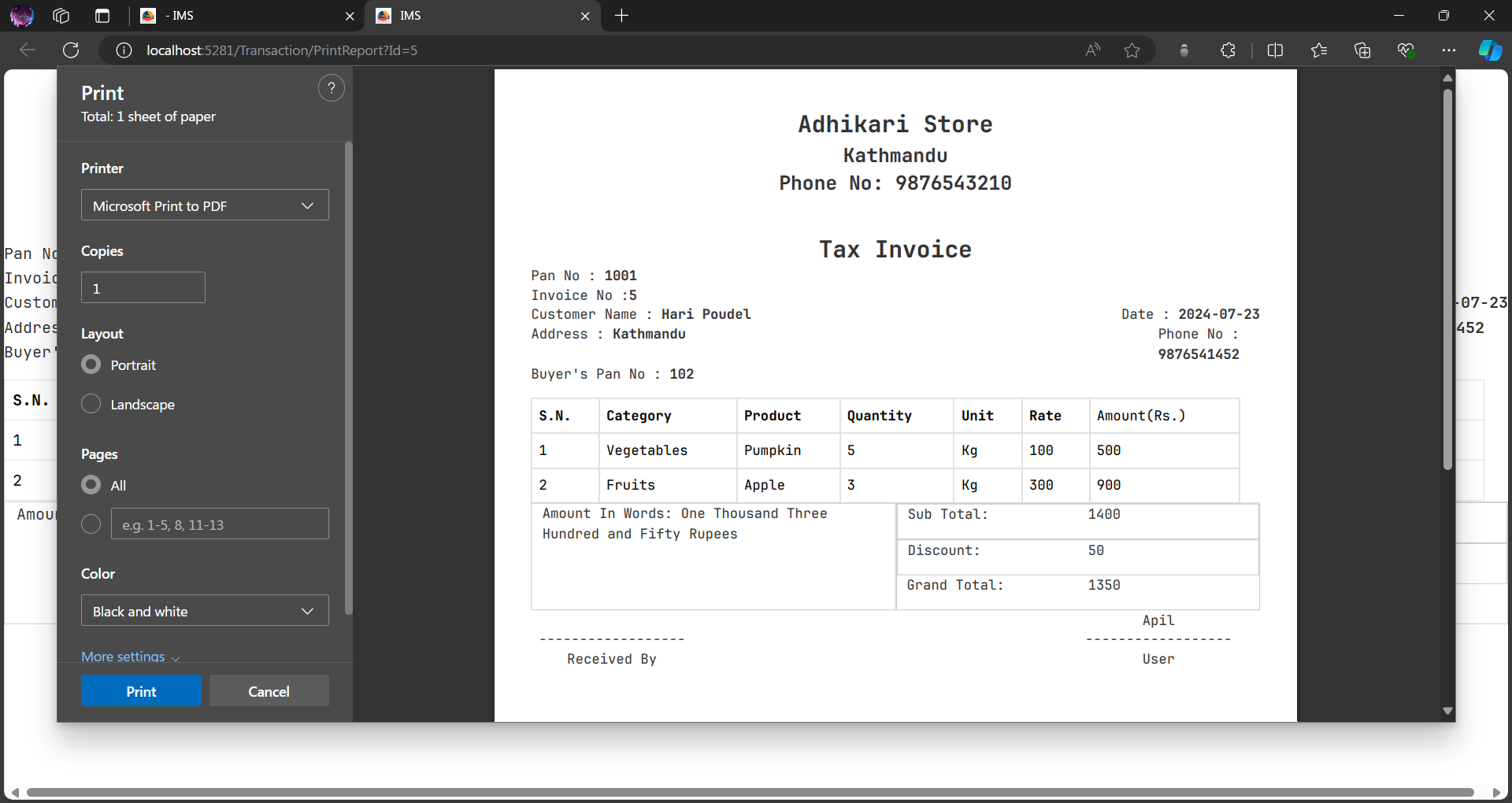


Adding Product

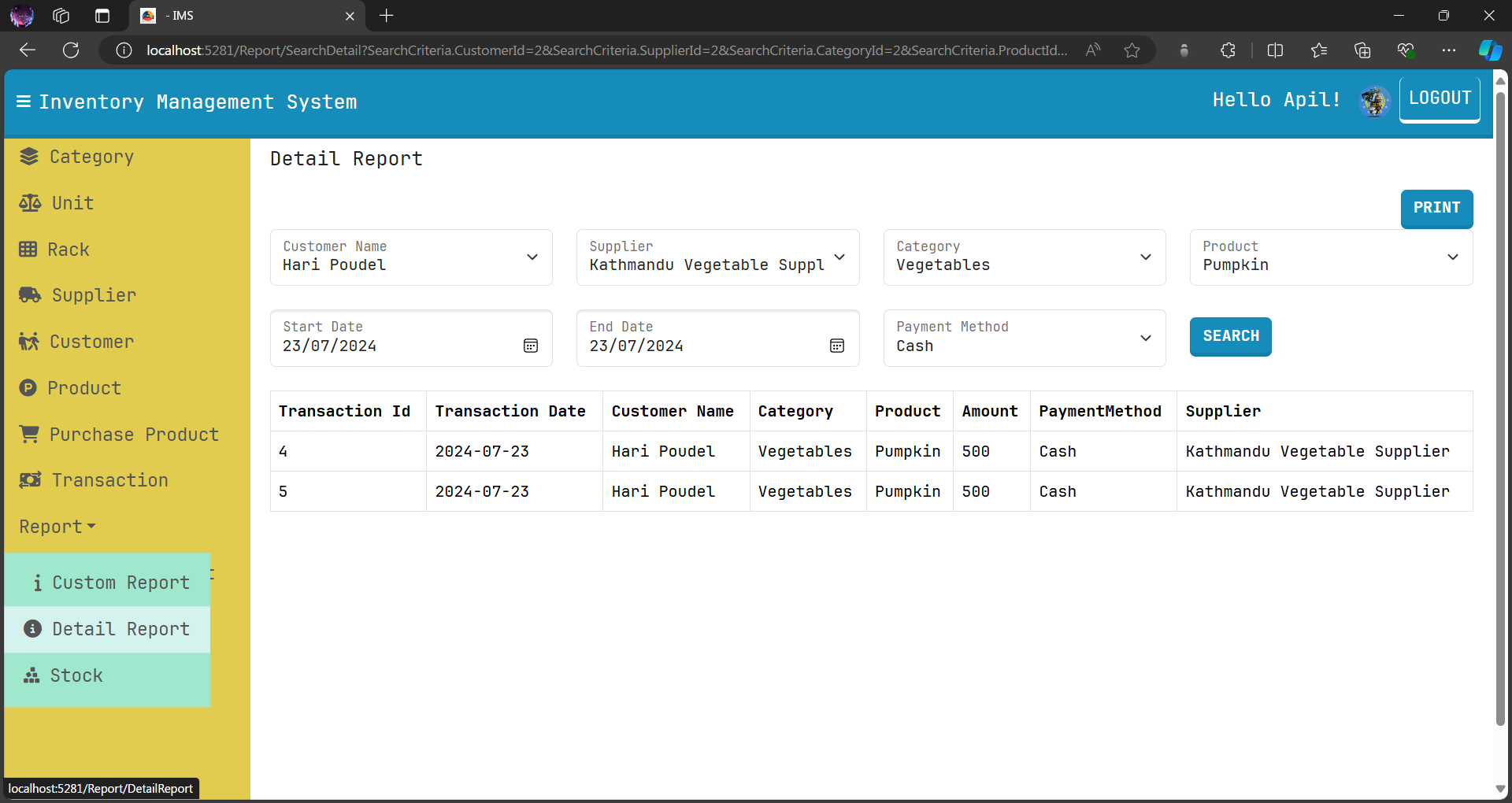
Purchasing Product



Selling Product



Printing Bill

Detailed Report