ACKNOWLEDGEMENT

We express our sincere thanks to the management and **Dr. J. JANET M.E., Ph.D.,** Principal, Sri Krishna College of Engineering and Technology, Coimbatore for providing us the facilities to carry out this mini project work.

We are thankful to **Dr. K. SASIKALARANI M.E, Ph.D.,** Professor and Head, Department of Artificial Intelligence and Data Science, for her continuous evaluation and comments given during the course of the mini project work.

We express our deep sense of gratitude to our supervisor **Dr. M.ROHINI M.E, Ph.D.,** Assistant Professor, Department of Computer science and Engineering for her valuable advice, guidance and support during the course of our mini project work.

We express our heartfelt sense of gratitude and thanks to our beloved parents, family and friends who have helped during the mini project course.

ABSTRACT

Effective warehouse inventory management is essential for optimizing operations and ensuring the smooth functioning of supply chains in today's competitive business landscape. This work provides an overview of key elements and strategies involved in warehouse inventory management, highlighting the significance of technology, data analytics, and process optimization. The primary objective of warehouse inventory management is to maintain optimal stock levels while minimizing costs and maximizing customer satisfaction. To achieve this, modern warehouses have shifted from traditional manual methods to technology-driven solutions. This work discusses the importance of using cutting-edge inventory management software, RFID technology, and automated systems to streamline inventory control processes. Furthermore, data analytics plays a crucial role in warehouse inventory management. By harnessing big data and analytics tools, organizations can gain valuable insights into demand forecasting, stock rotation, and order fulfillment. This work explores the benefits of predictive analytics and data-driven decision-making in achieving inventory optimization. In conclusion, underscores the significance of adopting a holistic approach to warehouse inventory management that combines technology, data analytics, and process optimization. Implementing these strategies can lead to improved inventory control, reduced operational costs, and increased customer satisfaction, ultimately contributing to the success and competitiveness of any organization in the modern business environment.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	FULL FORM
APS	ADVANCED PLANNING AND SCHEDULING
ASN	ADVANCED SHIPPING NOTICE
ASRS	AUTOMATED STORAGE AND RETRIEVAL SYSTEM
B2B	BUSINESS TO BUSSINESS
B2C	BUSINESS TO CONSUMER
RAM	RANDOM ACCESS MEMORY
ROM	READ ONLY MEMORY
UI	USER INTERFACE
JVM	JAVA VIRTUAL MACHINE
OOP	OBJECT - ORIENTED PROGRAMMING
API	APPLICATION PROGRAMMING INTERFACE
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
CI/CD	CONTINUOUS INTEGRATION / CONTINUOUS DEVELOPMENT
AI	ARTIFICIAL INTELLIGENCE
ML	MACHINE LEARNING
BOM	BILL OF MATERIAL

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF AIRLINE RESERVATION

A Warehouse Inventory Management System (WIMS) is a comprehensive software solution designed to efficiently oversee and optimize the storage, tracking, and control of goods and materials within a warehouse or distribution center. Unlike airline reservation systems (ARS), which primarily focus on passenger bookings, WIMS is dedicated to enhancing the logistical and operational aspects of inventory management. The primary objective of this software application is to ensure precise control over inventory, minimize holding costs, and meet customer demand effectively.

WIMS encompasses various critical functions, including inventory tracking, order fulfillment, demand forecasting, and space utilization. Real-time tracking technologies such as barcode scanning and RFID are used to maintain accurate information about item availability and location. The system facilitates efficient order picking strategies to enhance productivity within the warehouse.

Accurate demand forecasting plays a pivotal role in anticipating future inventory requirements, reducing the risks of overstocking or stockouts. By analyzing historical data and market trends, WIMS helps organizations align stock levels with projected demand. Moreover, it aids in the maintenance of safety stock levels as a buffer to mitigate unexpected demand spikes or supply chain disruptions.

Strong supplier relationships are essential in timely replenishment and maintaining consistent inventory levels. Concepts such as Just-in-Time (JIT) and Vendor-Managed Inventory (VMI) systems are integrated into WIMS to optimize inventory management. Additionally, data analytics tools provide valuable insights into inventory trends and operational efficiencies, enabling data-driven decision-making.

Continuous improvement efforts, including regular audits and process optimization, are essential for ensuring that the warehouse operates at its best.

1.2 PROJECT OVERVIEW

The Warehouse Inventory Management System is a software project aimed at creating a comprehensive web-based platform designed to efficiently manage and optimize all aspects of inventory control within a warehouse or distribution center. Its primary objective is to streamline warehouse operations, reduce costs, and enhance inventory management efficiency. This project also includes features for user access control, allowing different roles such as administrators and warehouse staff to access and manage inventory data.

1. Inventory Tracking: The core functionality of the system is to enable users to track and manage inventory items efficiently. Warehouse staff can input and update information about items, including item name, quantity, location within the warehouse, and other relevant details. The system provides real-time visibility into inventory levels.

2. Order Processing: In addition to tracking inventory, the system supports order processing. It allows users to create and manage customer orders, allocate items from inventory, and generate packing lists or invoices for shipments. The system also automates order fulfillment processes.

3. Database Management: The system relies on a robust database to store and retrieve information about inventory items, orders, suppliers, and other related data. It should efficiently handle data to ensure accurate and up-to-date information at all times.

4. Supplier Management: Warehouse Inventory Management includes features for managing relationships with suppliers. Users can track supplier information, manage purchase orders, and handle the replenishment of inventory stock.

5. Reporting And Analytics: The system offers reporting and analytics tools to provide insights into inventory trends, turnover rates, stock levels, and more. This data-driven approach helps organizations make informed decisions about inventory management.

6. User Access Control: Access to the system is role-based, with different permissions for administrators, warehouse staff, and other users. This ensures that only authorized personnel can perform specific actions within the system.

7. Security: Security is a top priority, as the system deals with sensitive inventory and business data. Robust security measures are implemented to protect data integrity and prevent unauthorized access.

1.3 OBJECTIVES

- To introduce automation into warehouse inventory management processes.
- To enhance the efficiency of warehouse operations and logistics.
- To offer added value to warehouse management for improved customer satisfaction..
- To demonstrate the organization's commitment to embracing and leveraging modern technology in inventory control and management.

1.4 SCOPE OF THE PROJECT

The Warehouse Inventory Management System establishes a user-friendly interface that facilitates seamless communication between warehouse staff and administrators through software. This project focuses on automating and optimizing the inventory control processes within a warehouse or distribution center. The system offers comprehensive features, including real-time inventory tracking, order processing, supplier management, and reporting.

For data storage and retrieval, the project utilizes the Structured Query Language (SQL), specifically MySQL, to efficiently manage and maintain the inventory database. This approach allows for the storage of a vast number of inventory records, although it is important to note that the retrieval process may experience some latency, particularly when searching for specific records within the database due to the sequential nature of file handling.

CHAPTER - 2

REQUIREMENTS

2.1 FUNCTIONAL REQUIREMENTS

2.1.1 INVENTORY TRACKING

Functional requirements for inventory tracking in Warehouse Inventory Management are crucial to ensuring the system's effectiveness in managing and optimizing inventory control processes. These requirements encompass several key aspects. Firstly, the system should support real-time tracking of inventory items, allowing for the immediate recording and updating of item information, such as quantity, location, and status. Secondly, it should enable efficient data input methods, including barcode scanning or RFID technology, to streamline the tracking process. Thirdly, the system should provide search and filtering capabilities, enabling users to quickly locate specific items within the warehouse. Moreover, it should support batch operations, allowing for the simultaneous tracking of multiple items or bulk updates.

2.1.2 ITEM MANAGEMENT

Functional requirements for item management within a Warehouse Inventory Management System are crucial for the efficient control and oversight of inventory items. The system should enable users to create, track, and manage items seamlessly. Users must have the capability to add new items with detailed information, such as item names, descriptions, SKUs (Stock Keeping Units), and categorizations. Real-time item tracking is essential, allowing users to monitor the movement, location, and quantity of items as they enter or exit the warehouse. To enhance accuracy, unique identification methods like barcodes, QR codes, or RFID tags should be supported. Users should be able to manage item quantities, update item statuses, and specify item conditions when necessary. Efficient location management is critical, enabling users to assign and modify physical storage locations within the warehouse.

2.1.3 ORDER PROCESSING

In Warehouse Inventory Management System are pivotal for the smooth and precise handling of customer orders. The system must enable users to create, edit, and manage orders efficiently. It should support various order statuses to monitor order progress and automatically

allocate inventory items to fulfill orders accurately. Handling backorders when items are temporarily unavailable is crucial. Users should have the flexibility to specify multiple shipping addresses within a single order and select different shipping methods. The system must generate order confirmations and send them to customers via email while updating inventory levels as orders are processed. Integration with a secure payment gateway is essential for secure online payments. Order history and tracking features ensure transparency and customer satisfaction, while support for returns and refunds streamlines post-purchase processes

2.1.4 REPORT GENERATION

Functional requirements for report generation in an Inventory Management System play a pivotal role in providing valuable insights and supporting data-driven decision-making. The system should offer the capability to generate a variety of reports, including inventory status reports, order fulfillment reports, stock turnover reports, and supplier performance reports. Users must be able to select specific criteria for report generation, such as date ranges, categories, or item attributes, to tailor reports to their needs.

Moreover, reports should include visual elements like charts and graphs to enhance data visualization and make it easier to identify trends and patterns. Integration with other system modules, such as item management and order processing, ensures that reports are based on real-time and accurate data.

2.2 NON FUNCTIONAL REQUIREMENTS

2.2.1 PERFORMANCE

Non-functional requirements outline the critical quality attributes and characteristics that a Warehouse Inventory Management System should embody. These specifications are centered on how the system should perform, emphasizing the user experience and system scalability. The system should respond promptly to user interactions, encompassing activities like item tracking, order processing, and report generation, all delivered with minimal latency to ensure a seamless user experience. Furthermore, the Warehouse Inventory Management System must be architected for both horizontal and vertical scalability, enabling it to gracefully handle increased workloads during periods of heightened activity, such as inventory restocking or surges in order processing, without compromising performance. To uphold these

performance standards, routine performance testing should be undertaken to replicate demanding scenarios and verify the system's capacity to sustain optimal performance during peak usage, thus guaranteeing efficient and responsive inventory management operations

2.2.2 USER INTERFACE

In the context of Warehouse Inventory Management, the design of the user interface (UI) holds significant importance in facilitating effective interactions between users, be they warehouse staff or administrators, and the system. The UI should prioritize user-friendliness and intuitiveness, catering to individuals with varying levels of experience. It should allow users to navigate the system seamlessly, eliminating the need for extensive training. Clear and familiar design patterns should be employed, ensuring that users can easily understand and interact with the console-based application. The provision of clear and concise instructions within the interface is crucial to guide users in efficiently utilizing the system. These non-functional requirements play a pivotal role in ensuring that the Warehouse Inventory Management System operates reliably, securely, and efficiently, ultimately enhancing its usability and overall reliability for warehouse personnel and administrators.

2.3 SYSTEM REQUIREMENTS

2.3.1 HARDWARE SPECIFICATION

- Processor Type : Intel Core i5
- Speed : 3.40 GHz
- RAM : 4 GB
- Hard Disk : 500 GB
- Keyboard : 101/102 Standard Keys
- Mouse : Optical Mouse

2.3.2 SOFTWARE SPECIFICATION

- Operating System : Supports Windows and Linux Operating Systems
- Technologies used : Core JAVA
- Database : MySQL

2.4 TECHNICAL REQUIREMENT DESCRIPTION

2.4.1 TECHNOLOGIES USED: CORE JAVA

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is intended to let application developers write once, and run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. It was developed by James Gosling and his team at Sun Microsystems (now owned by Oracle Corporation) and released in 1995. Java has gained immense popularity for a variety of reasons, including its rich feature set.

FEATURES OF JAVA

- Platform Independence (Write Once, Run Anywhere): One of Java's standout features is its platform independence. Java programs are compiled into bytecode, a platform-neutral intermediate form. This bytecode can be executed on any system with a Java Virtual Machine (JVM), allowing developers to write code once and run it on multiple platforms. This "write once, run anywhere" capability is instrumental in making Java a versatile and widely adopted programming language for cross-platform development, from desktop applications to web services.
- Object-Oriented Programming (OOP): Java is firmly rooted in the principles of Object-Oriented Programming (OOP). It encourages the organization of code into reusable, self-contained objects that encapsulate both data and behavior. This approach fosters modularity, code reusability, and easier maintenance, as well as supporting the development of complex software systems by breaking them down into manageable components..
- Robustness and Memory Management: Java places a strong emphasis on robustness and reliability. It includes features like automatic memory management, where the Java Virtual Machine (JVM) takes care of memory allocation and garbage collection, reducing the risk of memory-related errors, such as memory leaks and dangling pointers. Additionally, Java's comprehensive exception handling mechanism enables developers to gracefully handle and recover from runtime errors, enhancing application resilience.
- Multithreading Support: Java provides built-in support for multithreading, enabling developers to create applications that execute multiple threads concurrently.

Multithreading is crucial for building responsive and scalable software, as it allows tasks to run in parallel, making the most of modern multi-core processors. Java's thread management features, like synchronization, ensure safe and controlled access to shared resources in multithreaded environments.

- Rich Standard Library (API): Java comes with an extensive standard library, commonly referred to as the Java API (Application Programming Interface). This library provides a wide range of pre-built classes and methods for performing various tasks, from basic file I/O to advanced networking and graphical user interface development. The Java API simplifies and accelerates development by offering a comprehensive set of tools, reducing the need to reinvent the wheel when building applications. This rich ecosystem of classes and packages is a significant asset for Java developers, saving time and effort in application development.

2.4.2 DATABASE : MYSQL

A database is an application that stores the organized collection of records. It can be accessed and manage by the user very easily. It allows us to organize data into tables, rows, columns, and indexes to find the relevant information very quickly. Each database contains distinct API for performing database operations such as creating, managing, accessing, and searching the data it stores. MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. It is a popular choice for building database-driven applications, websites, and services.

FEATURES OF MYSQL

- High Performance: MySQL is known for its fast data retrieval and processing capabilities, making it suitable for high-traffic websites and applications. MySQL can handle small to large-scale applications and can be scaled vertically (by adding more resources to a single server) or horizontally (by distributing data across multiple servers).
- SQL (Structured Query Language): MySQL uses SQL for querying and managing data. Developers can use SQL statements to create, read, update, and delete data, as well as perform complex database operations. It allows for the creation of complex data structures and relationships between tables.

- Cross-Platform: MySQL is compatible with various operating systems, including Windows, Linux, macOS, and more. This cross-platform support makes it versatile for different development environments.
- Community and Support: MySQL has a large and active community, as well as commercial support options provided by Oracle, making it suitable for both small projects and enterprise-level applications

CHAPTER – 3

MODULE DESCRIPTION

3.1 PROPOSED SYSTEM

In the context of Warehouse Inventory Management, the proposed system entails the development of a robust console-based application with a text-based interface for efficient user interactions. The architecture of our proposed Warehouse Inventory Management System includes the following key functionalities:

1. Item Tracking : The system should provide users with the ability to search for inventory items. It displays detailed information about items, including item names, descriptions, quantities, and locations within the warehouse.

2. View Inventory: Users can access an overview of the entire inventory, showcasing item details, including item names, quantities, and storage locations.

3. Add Items: Users should be able to add new items to the inventory, providing essential item information such as names, descriptions, and quantities. The system assigns unique identifiers (e.g., SKU) to newly added items.

4. Update Items: Users can modify existing item details, including descriptions, quantities, and storage locations. The system ensures real-time updates to maintain accurate inventory records.

3.2 MODULE DESCRIPTION

3.2.1 INVENTORY MANAGEMENT

The Item Management Module facilitates the addition, modification, and removal of items, along with detailed tracking of item information. Meanwhile, the Inventory Tracking Module enables precise monitoring of item movements and storage locations, issuing alerts when stock levels are low. The Order Processing Module manages the end-to-end order lifecycle, from order creation and editing to shipping and delivery.

User access and authentication are governed by the User Authentication and

Authorization Module, ensuring role-based access control. The Reporting and Analytics Module empowers data-driven decision-making with custom report generation and data analysis capabilities.

In the context of Warehouse Inventory Management, the inventory management module plays a central role in overseeing the entire inventory control process. This module includes several key functionalities:

- Item Management:

  - Users can add new items to the warehouse inventory, specifying essential item details such as name, description, quantity, and storage location. Additionally, they can update existing item information and remove items when necessary.

- Inventory Tracking:

  -This function allows users to track the movement and storage locations of items within the warehouse. It provides real-time visibility into item availability, aiding in efficient retrieval and management.

- Stock Alerts:

  -The system can issue alerts when inventory levels fall below specified thresholds, ensuring timely restocking and preventing stockouts.

- Location Management:

  -Users have the ability to assign, update, or modify physical storage locations within the warehouse. Effective location management optimizes item organization and retrieval.

- Order Processing:

  -This module handles the creation and management of customer orders. Users can create new orders, edit existing ones, and track the status of each order from processing to delivery.

- Report And Analytics:

  -Robust reporting capabilities enable users to generate reports on inventory status, item turnover, and other critical metrics. These reports facilitate data-driven decision-making and strategic planning.

- Security And Audit Trail:

  -Security measures are implemented to protect sensitive inventory data, and an audit trail records and tracks user actions within the system for accountability and auditing purposes.

3.2.2 ITEM TRACKING

In the context of Warehouse Inventory Management, the Item Tracking module serves as the backbone for maintaining accurate and real-time information about inventory items within the warehouse. This module encompasses several crucial functionalities:

- Item Records Creation:

  -Users can create detailed records for each item in the inventory, including essential information such as item name, description, SKU (Stock Keeping Unit), quantity, unit of measurement, and purchase/selling prices.

- Movement History:

  -Item tracking records the history of item movements within the warehouse. This includes details such as when an item was received, moved between storage locations, or shipped out for delivery.

- Report And Analytics:

  -Robust reporting capabilities enable users to generate reports on item quantities, movement history, and stock status. These reports support data-driven decision-making and inventory optimization.

3.2.3 ORDER MANAGEMENT

In the context of Warehouse Inventory Management, the Order Management module is central to overseeing the entire order lifecycle, from creation to fulfillment. This module encompasses several key functionalities

- Order Creation:

  -Users can initiate the creation of new orders, specifying details such as the customer's name, contact information, shipping address, and order items. Items can be added from the inventory, and quantities are recorded.

- Order Editing:

  - Users have the flexibility to edit existing orders, allowing modifications to items, quantities, and customer information before the order is processed.

- Inventory Adjustment:

  - When an order is placed, the system automatically adjusts the inventory by reducing the quantities of the ordered items. This ensures accurate stock levels.

- Shipping And Delivery Management:

  -Users can manage the shipping and delivery process, including selecting shipping carriers, generating shipping labels, and tracking shipments until they reach their destination.

- Payment Processing:

  -The module includes secure payment processing capabilities, allowing users to record payments made by customers for their orders.

- Order Confirmation:

  -Upon order placement, customers receive order confirmation notifications containing order details, estimated delivery dates, and payment receipts.

- Return And Refund Handling:

  -In cases of returns or order discrepancies, users can initiate return requests and process refunds to maintain customer satisfaction.

3.2.4 DATABASE

The Warehouse Inventory Management System relies on a robust database to efficiently store and manage data related to inventory, products, suppliers, orders, and transactions. The database is a critical component that ensures the accuracy and integrity of inventory operations. In this context, let's outline the key entities and their attributes within the Inventory Database:

1. Products:

- Product_ID (Primary Key): A unique identifier for each product.
- Product_Name: The name or description of the product.
- Category: The category or type to which the product belongs.
- Unit_Price: The price of a single unit of the product.
- Quantity_In_Stock: The current quantity of the product available in the warehouse.

2. Suppliers:

- Supplier_ID (Primary Key): A unique identifier for each supplier.
- Supplier_Name: The name of the supplier or vendor.
- Contact_Info: Contact information for the supplier, such as email and phone number.
- Address: The supplier's business address..

3. Orders:

- Order_ID (Primary Key): Unique identifier for each reservation.
- Order_DateTime: References the passenger associated with the reservation.
- Product_ID: References the flight for which the reservation is made.
- Quantity_Ordered: A unique code generated for each reservation.
- Total_Price: Indicates the status of the reservation (Confirmed, Pending, Canceled).

4. Transactions:

- Transaction_ID(Primary Key): A unique identifier for each transaction.
- Product_ID: References the product involved in the transaction.
- Quantity: The quantity of products involved in the transaction.

5. Order Items:

- Order_ID(Primary Key): A unique identifier for each order item.
- Product_ID: References the product being ordered.
- Quantity_Ordered: The quantity of the product ordered in this item.
- Total_Price: The total price for the quantity of products ordered in this item.

6. Users:

- User_ID (Primary Key): A unique identifier for each user.
- Username: The user's login username.
- User_Role: Defines the role or permissions of the user (e.g., Admin, Warehouse Staff).
- Password: The user's encrypted password for authentication.

CHAPTER – 4

DESIGN OF PROPOSED SYSTEM

4.1 SYSTEM DESIGN

4.1.1 INPUT DESIGN

In Warehouse Inventory Management, the input design is tailored to emphasize user-friendliness and operational efficiency. The interface is adept at capturing critical inventory data while incorporating error handling mechanisms and confirmation steps to provide users with guidance and ensure data accuracy. Our primary objective is to deliver a seamless, secure, and efficient process for managing warehouse inventory operations.

4.2 SOFTWARE ENGINEERING APPROACH

The field of software engineering is related to the development software in systematic manner unlike simple programs which can be developed in isolation and there may not be any systematic approach being followed. For a mature process, itshould be possible to determine in advance how much time and effort will be required to produce the final product.

The model used is **Waterfall Model or Classic Life Cycle**. In this model first of all the existing system is observed. Then customer requirements are taken in consideration then planning, modeling, construction and finally deployment.

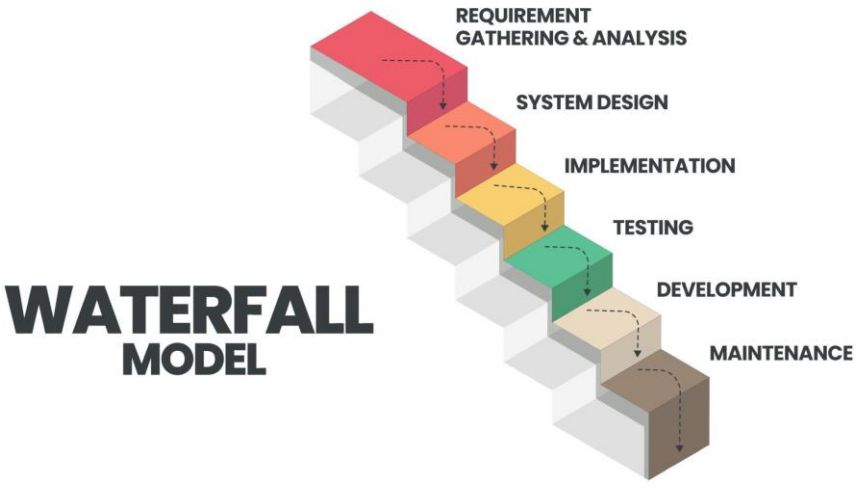


Fig 4.2 : WATERFALL MODEL

4.3 UML DIAGRAMS

4.3.1 CLASS DIAGRAM

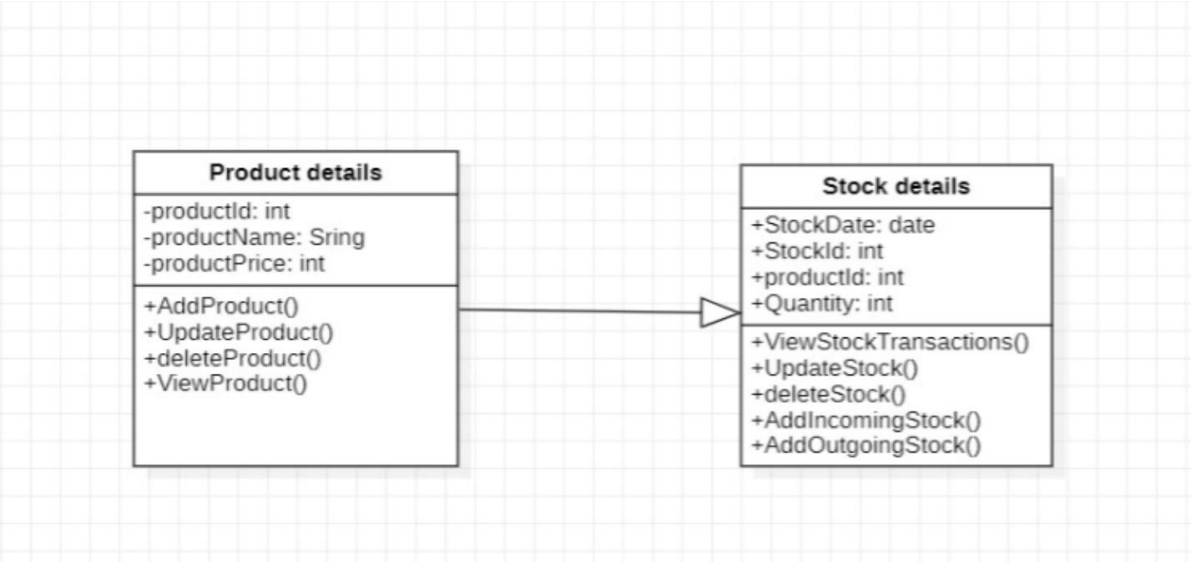


Fig 4.3.1 : CLASS DIAGRAM

4.3.2 ENHANCED ER DIAGRAM

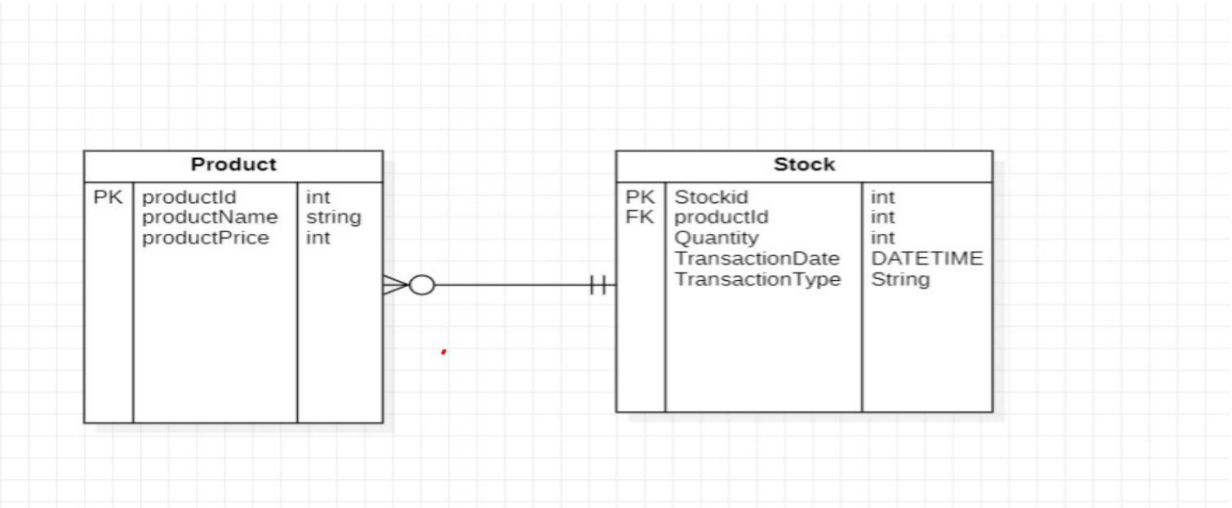


Fig 4.3.2 : ENHANCED ER DIAGRAM

4.3.3 STATE ACTIVITY DIAGRAM

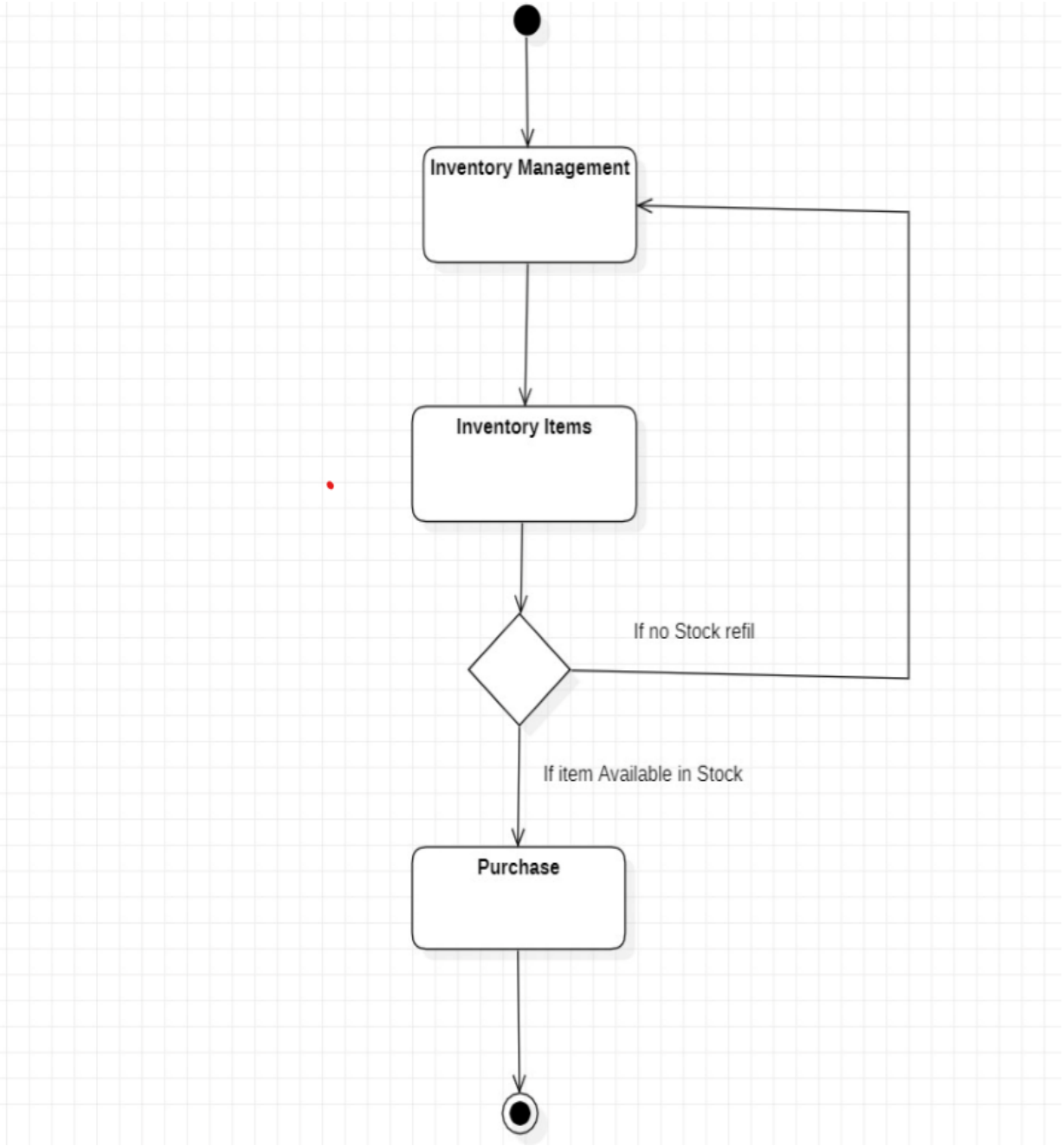


Fig 4.3.3 : STATE ACTIVITY DIAGRAM

4.3.4 SEQUENCE DIAGRAM

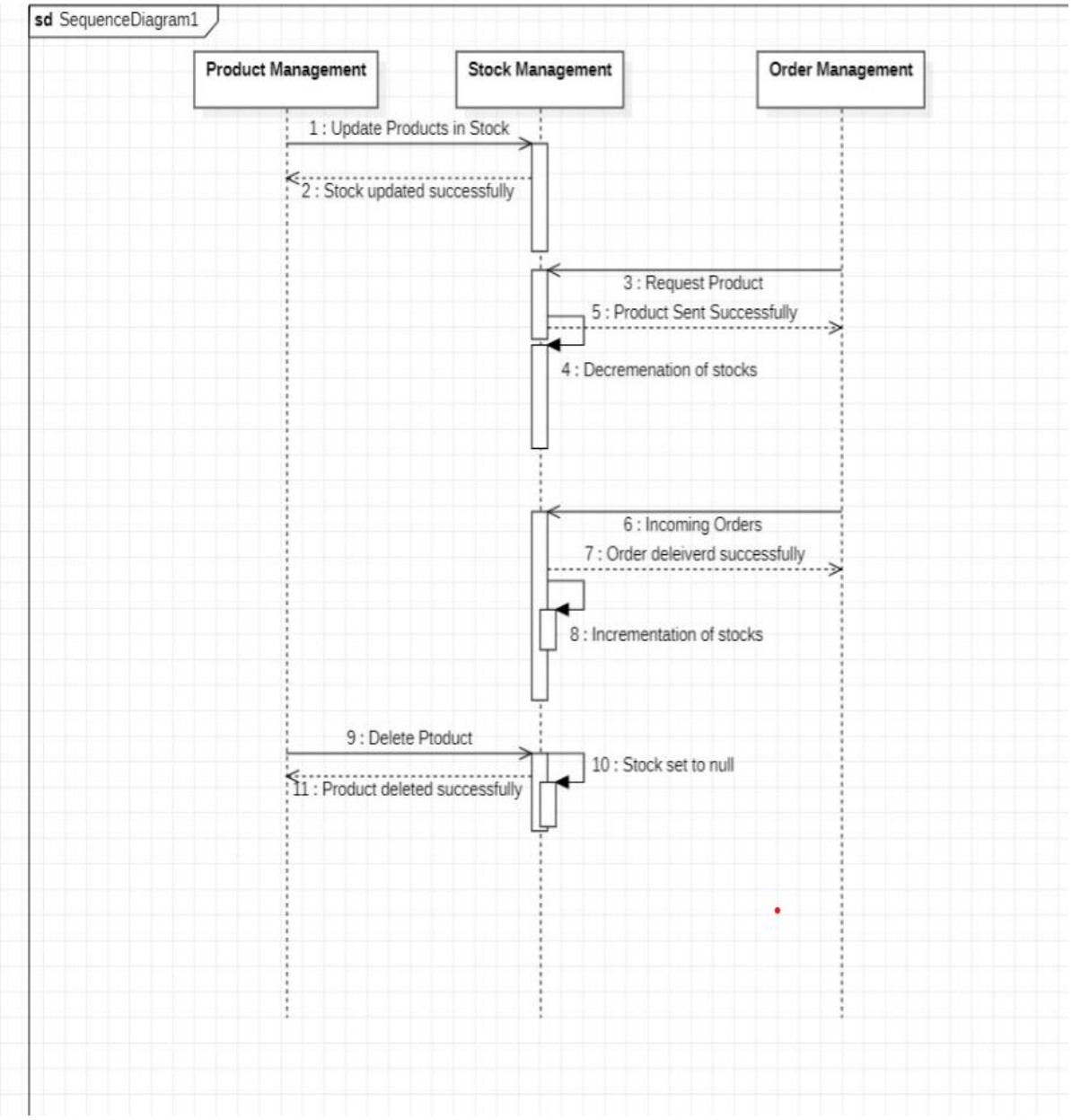


Fig 4.3.4 : SEQUENCE DIAGRAM

4.3.5 USE CASE DIAGRAM

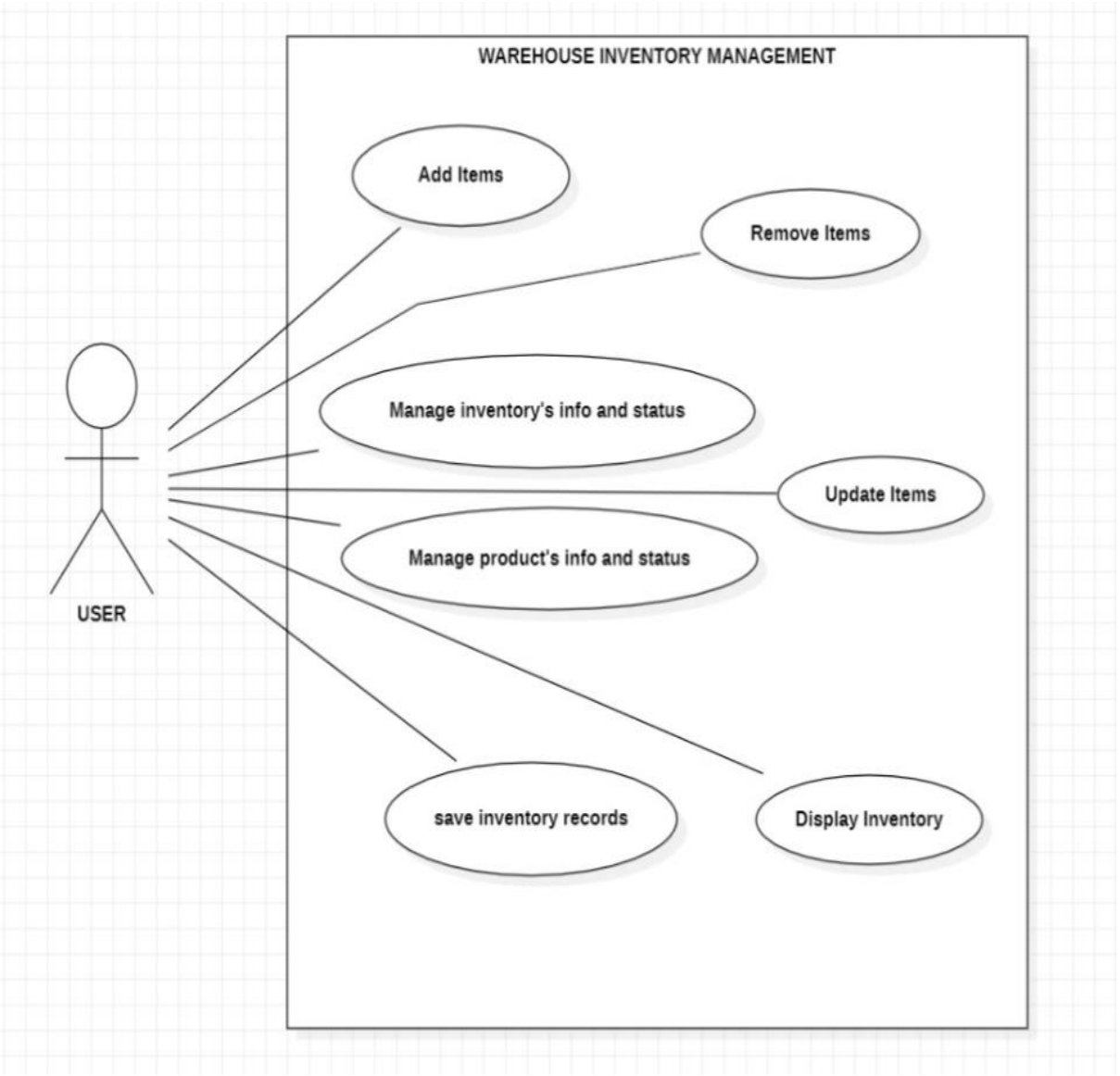


Fig 4.3.5 : USE CASE DIAGRAM

CHAPTER – 5

IMPLEMENTATION OF PROPOSED SYSTEM

5.1 INTRODUCTION

The implementation of the proposed Warehouse Inventory Management System is a multifaceted process that involves meticulous planning and execution. It commences with the establishment of a robust development environment, encompassing the selection of appropriate programming languages, frameworks, and tools. Simultaneously, the configuration of the database structure and connections is set up. Throughout the development phase, paramount importance is given to adhering to stringent security measures and ensuring compliance with data protection regulations.

Testing constitutes a pivotal aspect of the implementation process, encompassing various levels, including unit testing, integration testing, and user acceptance testing. This rigorous testing regimen aims to pinpoint and rectify any discrepancies or performance bottlenecks that may arise. Comprehensive documentation, training modules, and compliance checks are integral components of the implementation strategy, ensuring that the system not only functions robustly but also maintains user-friendliness and complies with industry standards and legal prerequisites.

Post-implementation, the deployment phase takes precedence, necessitating the hosting of the system on production servers or cloud-based platforms to guarantee security and efficiency in a live operational environment. Ongoing support and maintenance procedures follow, encompassing regular software updates, bug fixes, and responsive customer support to sustain the system's robustness and reliability over time.

Integration with external systems, such as suppliers' networks and financial processors, facilitates seamless warehouse operations, while proactive monitoring and error tracking mechanisms swiftly identify and address any emerging issues. Concurrently, the system must maintain strict compliance with industry regulations and uphold data privacy standards.

5.2 TOOLS AND FRAMEWORKS INVOLVED IN THE SYSTEM

5.2.1 JAVA

Developing an Airline Reservation System (ARS) as a Java console application involves creating a text-based user interface that allows users to interact with the system through a command-line interface. This interface should be designed to be clear, intuitive, and responsive, enabling users to perform essential tasks such as flight searches, reservations, and user authentication. The application's main menu serves as the central hub, providing users with options to access various features of the ARS. These features include flight search functionality, enabling users to find available flights by specifying criteria like departure and destination airports, travel dates, and the number of passengers. The system also incorporates user authentication, requiring users to log in with their credentials before making reservations or accessing personal information. Additionally, reservation management capabilities allow users to view, modify, or cancel their bookings, with support for handling special requests.

To ensure robustness and data management, the application leverages data structures like arrays or collections to store information about flights, passengers, and reservations. Error handling and input validation are essential components, ensuring that users receive clear error messages when input issues occur and preventing data entry errors or security vulnerabilities. Logging functionality helps in recording important events and transactions for auditing and debugging purposes.

5.2.2 MySQL

Integrating MySQL into a Java-based console Airline Reservation System (ARS) is a crucial step in building a robust and data-driven application. First, set up and configure MySQL database, ensuring that it is appropriately installed and a dedicated database is created to store ARS-related data. Next, the MySQL Connector should be added to your Java project to facilitate communication between the console application and the MySQL database. This library provides the necessary tools for establishing and managing database connections.

Once the setup is complete, design the database schema to accommodate ARS data entities such as flights, passengers, reservations, and user credentials. Develop Java classes and methods to interact with the database, performing operations like inserting new flight records, retrieving passenger information, and updating reservation details. Proper resource management and exception handling are vital to ensure that database connections and resources are used efficiently and securely. This integration empowers your console ARS to store and retrieve data seamlessly, allowing users to search for flights, make reservations, and manage their bookings effectively while maintaining the integrity and security of the stored data.

5.2.3 VISUAL STUDIO CODE

Visual Studio Code (VSCode) is a versatile and lightweight integrated development environment (IDE) that can be an excellent choice for developing a Java-based console Airline Reservation System (ARS). With its extensive range of extensions and features, VSCode streamlines Java coding and debugging tasks. VSCode offers a powerful code editor with syntax highlighting, code completion, and built-in version control support through Git. Developers can customize the environment with extensions such as Java Extension Pack, which provides advanced Java language support, debugging capabilities, and integrated build tools. These features enable efficient coding and facilitate collaboration within development teams.

Additionally, VSCode's seamless integration with terminal and command-line tools simplifies interactions with the console-based ARS. Developers can conveniently run Java applications, execute database queries, and manage project dependencies directly from the integrated terminal. The lightweight nature of VSCode ensures that it doesn't consume excessive system resources, making it a responsive and agile choice for developing and testing the ARS. Furthermore, VSCode's extensibility allows developers to tailor the IDE to their specific project needs, making it a flexible and adaptable tool for building a robust Java-based console ARS efficiently.

5.2.4 GITHUB

GitHub plays a pivotal role in the development and collaboration of a Java-based console Airline Reservation System (ARS). It serves as a centralized repository for storing and managing the project's source code, allowing multiple developers to work collaboratively on different aspects of the application. Each developer can clone the project's repository, make changes or additions to the code, and then push those changes back to the repository. This ensures version control and keeps a historical record of all code modifications, making it easy to track changes, resolve conflicts, and maintain code integrity. Additionally, GitHub's branching and merging features facilitate the implementation of new features or bug fixes in isolation, preventing disruption to the main codebase until changes are thoroughly tested and ready for integration. GitHub's issue tracking system provides a structured way to manage tasks, enhancements, and bugs, allowing developers to prioritize and assign work efficiently. Furthermore, it promotes collaboration among team members by providing tools for code reviews and discussions, ensuring code quality and consistency.

GitHub also extends beyond version control to offer robust deployment capabilities. It seamlessly integrates with Continuous Integration/Continuous Deployment (CI/CD) pipelines, enabling automated builds, testing, and deployment of the console ARS. This integration helps streamline the development lifecycle, ensuring that code changes are automatically tested and deployed to production or staging environments, reducing the risk of human error and enhancing the overall development workflow. GitHub Actions, a built-in CI/CD platform, allows developers to define custom workflows for tasks such as running tests, building executable JAR files, and deploying the console ARS to a server or cloud platform. By leveraging GitHub's comprehensive ecosystem of collaboration, version control, and CI/CD tools, the development of a Java-based console ARS becomes not only efficient but also conducive to maintaining code quality, stability, and scalability throughout the project's lifecycle.

CHAPTER - 6

TESTING

6.1 INTRODUCTION

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive. A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

6.2 STRATEGIC APPROACH TO SOFTWARE TESTING

The software engineering process can be viewed as a spiral.Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behaviour , performance, constraints and validation criteria for software are established.Moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decrease the level of abstraction on each turn. A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.

6.3 UNIT TESTING

Unit testing is a cornerstone of software development, honing in on the smallest units of code, often individual modules or functions. In our context, unit testing takes a white box approach, intimately scrutinizing the internal workings of these modules. This method empowers us to ensure the correctness and reliability of each component, identifying issues related to code complexity and logic errors.

6.3.1 WHITE BOX TESTING

To follow the concept of white box testing we have tested each form. We have created independently to verify that Data flow is correct and all conditions are exercised to check their validity.

- All loops are executed on their boundaries. This type of testing ensures that all independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity paths.

6.3.2 CONDITIONAL TESTING

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested so that each path that may generate on a particular condition is traced to uncover any possible errors.

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variable were declared. The definition-use chain method was used in this type of testing.

6.3.3 DATA FLOW TESTING

Data Flow Testing is a critical testing technique that focuses on tracing the flow of data through a program based on the definitions and uses of variables. It's particularly relevant when dealing with local variables that have limited scope within specific sections of the program. This approach relies on the definition-use chain method, systematically analyzing how data is defined, manipulated, and utilized throughout the code. By following this data flow, testers can uncover potential issues such as data inconsistencies, uninitialized variables, or data misuse.

6.3.4 LOOP TESTING

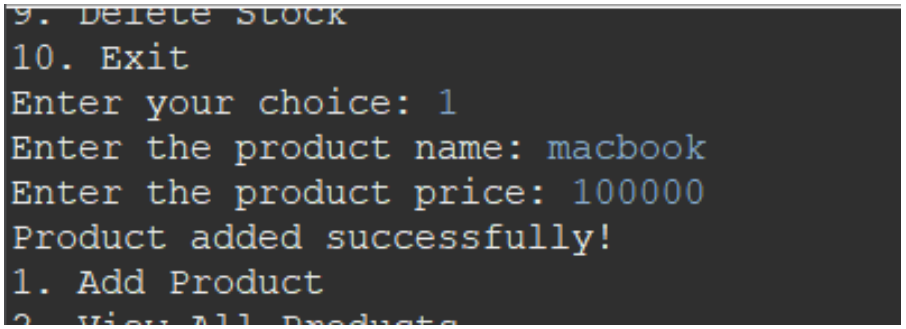
In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

- All the loops were tested at their limits, just above them and just below them. All the loops were skipped at least once.
- For nested loops test the inner most loop first and then work outwards. For concatenated loops the values of dependent loops were set with the help of connected loop.
- Unstructured loops were resolved into nested loops or concatenated loops and tested as above.
- Each unit has been separately tested by the development team itself and all the input have been validated.

6.4 TEST CASE

A test case, in software engineering, is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do. It may take many test cases to determine that a software program or system is considered.sufficiently scrutinized to be released. Test cases are often referred to as test scripts, particularly when written - when they are usually collected into test suites.

6.4.1 TEST CASE I



EXPECTED : A new product should be added in the database.

ACTUAL : A new product is added in the database.

6.4.2 TEST CASE II

```
Enter your choice: 6
Enter the product ID to update: 4
Enter the new product name: macbook m1
Enter the new product price: 120000
Product updated successfully!
```

```
Enter your choice: 2
All Products:
3 - iphone red - 19229
4 - macbook m1 - 120000
```

EXPECTED : Product details should be updated.

ACTUAL : Product details are updates successfully.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

In conclusion, the development of a comprehensive Warehouse Inventory Management System represents a significant advancement in addressing the evolving needs of modern inventory management and supply chain operations. This project was undertaken to meet the increasing demand for efficient, user-friendly, and technologically advanced inventory tracking and management solutions.

The Warehouse Inventory Management System presented here has demonstrated its capacity to streamline inventory processes, improve operational efficiency, and equip organizations with valuable tools for managing their inventory, suppliers, and order processing. Through the implementation of robust technologies and a well-structured framework, we have created a flexible and adaptable system capable of meeting the diverse needs of warehouses and inventory-centric businesses.

The advantages of this system are evident: enhanced inventory accuracy, reduced manual labor, mitigation of human errors, and overall operational efficiency. Its modular and scalable architecture opens the door to future enhancements, making it a forward-thinking investment for businesses looking to remain competitive in the dynamic landscape of inventory management.

In an era where inventory management plays a pivotal role in supply chain optimization and customer satisfaction, this Warehouse Inventory Management System stands as a significant leap toward harnessing technology to meet the demands of modern business operations.

7.2 FUTURE WORK:

In the realm of Warehouse Inventory Management, several exciting avenues for future work beckon. Embracing advanced data analytics and machine learning promises to unlock deeper insights into inventory dynamics, enabling precise demand forecasting and optimization. The integration of IoT technologies offers real-time visibility into inventory conditions, ensuring product quality and reducing errors. Automation and robotics can revolutionize warehousing by introducing autonomous systems for tasks like picking and packing. Blockchain's potential for transparency and traceability in the supply chain could become a game-changer.

In the realm of Warehouse Inventory Management, the potential for future advancements is substantial, offering opportunities to further optimize and refine inventory tracking and control processes.

The Internet of Things (IoT) can be seamlessly integrated into the system to enable real-time monitoring of inventory using sensors and smart devices. This would provide accurate and up-to-the-minute data on inventory levels, reducing the risk of stockouts or overstocking. Implementing advanced predictive analytics and machine learning algorithms can enhance demand forecasting accuracy. By analyzing historical data and market trends, the system can make more informed suggestions for reordering and inventory replenishment.

Developing a mobile application for warehouse inventory management would offer greater flexibility and accessibility for warehouse staff. Mobile apps can streamline tasks such as barcode scanning, order processing, and inventory adjustments while on the go.Blockchain technology can be explored to enhance supply chain transparency and traceability. It can provide an immutable ledger of all inventory-related transactions, reducing the risk of fraud and errors.

Implement sustainable inventory practices, such as optimizing energy consumption, reducing waste, and using eco-friendly packaging. This aligns with growing environmental concerns and can enhance the company's reputation. Explore voice recognition and artificial intelligence (AI) for voice-activated inventory management tasks. AI can assist in inventory categorization, demand prediction, and order optimization.

Ensure that the system is scalable to accommodate growing inventory needs.As businesses expand, the inventory management system should seamlessly adapt to handle larger volumes

REFERENCES

[1] Bartholdi III, J.J. & Hackman, S.T. (2014, August 19). *Warehouse and distribution science* (Release 0.96). The Supply Chain and Logistics Institute School of Industrial and Systems Engineering Georgia Institute of Technology Atlanta, GA 30332-0205 USA.

[2] Chalotra, V. (2013). Inventory management and small firms growth: An analytical study in supply chain. *Vision: The Journal of Vision Perspective*, *17*(3): 213-222. https://doi.org/10.1177/0972262913496726.

[3] Fellows, P. & Rottger, A. (n.d.). *Business management for small-scale agro-processors* [Working Document]. Food and Agriculture Organization of the United Nations,Rome. https://www.fao.org/3/j7790e/j7790e.pdf?msclkid=07d8911bcee611ecbad3109427dc6047.

[4] Popovic, V.,Kilibarda, M., Andrejic, M., Jereb, B., & Dragan, D. (2021, Feb. 13). A new sustainable warehouse management approach for workforce and activities scheduling. *Sustainability*, *13*(4). https://doi.org/10.3390/su13042021.

[5] *Saylor Academy (2019 , April Inventory management System*. https://learn.saylor.org/mod/page/view.php?id=9328%23%3a%7e%3atext%3dBasic+economic+order+quantity+model+(EOQ)+Used+to%2ccosts+of+holding+inventory+and+ordering+inventory+Assumptions%3a&msclkid=93111cd4ced

[6] LINCS in Supply Chain Management Consortium. (2017, March 3). *Warehousing operations certification track* (Version v2.28). https://www.skillscommons.org/bitstream/handle/taaccct/14296/LINCS%20Warehousing%20Operations%20Content.pdf?sequence=1&isAllowed=y. CC BY 4.0.

[7] Klumpp, M. & Heragu, S. (2019). Outbound logistics and distribution management. In Zijm, H., Klumpp, M., Regattieri, A. & Heragu, S. (Eds.), *Operations, Logistics and Supply Chain Management* (1st ed., pp. 305-330). Springer.

[8] Ting, S. and Cho, D.I. (2008). An integrated approach for supplier selection and purchasing decisions. *Supply Chain Management*, *13*(2) pp. 116-127. https://doi.org/10.1108/13598540810860958.

APPENDICES

APPENDIX - I

SOURCE CODE

Main.java

```
package Inventory;

import java.util.List;
import java.util.Scanner;
import java.sql.*;
import java.util.Date;

public class Main {
    private static final String DB_URL = "jdbc:mysql://localhost:3306/inventory";
    private static final String DB_USERNAME = "root";
    private static final String DB_PASSWORD = "suhaas";

    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver"); // Register the MySQL JDBC driver

            dbManager.createTables(); // Create tables if they don't exist
            dbManager.createDatabaseIfNotExists();

            System.out.println("Welcome to the Inventory Management System!");

            while (true) {
                System.out.println("1. Add Product");
                System.out.println("2. View All Products");
                System.out.println("3. Add Incoming Stock");
                System.out.println("4. Add Outgoing Stock");
                System.out.println("5. View Stock Transactions");
                System.out.println("6. Update Product");
                System.out.println("7. Delete Product");
                System.out.println("8. Update Stock");
```

```
        System.out.println("9. Delete Stock");
        System.out.println("10. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                addProduct();
                break;
            case 2:
                viewAllProducts();
                break;
            case 3:
                addIncomingStock();
                break;
            case 4:
                addOutgoingStock();
                break;
            case 5:
                viewStockTransactions();
                break;
            case 6:
                updateProduct();
                break;
            case 7:
                deleteProduct();
                break;
            case 8:
                updateStock();
                break;
            case 9:
                deleteStock();
                break;
            case 10:
                System.out.println("Goodbye!");
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
```

```
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.err.println("MySQL JDBC driver not found. Make sure you

 have added it to your project's classpath.");
        }
    }

    private static void addProduct() {
        scanner.nextLine(); // Consume the newline character after reading the integer
        System.out.print("Enter the product name: ");
        String productName = scanner.nextLine();

        System.out.print("Enter the product price: ");
        int productPrice = scanner.nextInt();

        Product product = new Product(productName);
        product.setProductPrice(productPrice);
        dbManager.addProduct(product);

        System.out.println("Product added successfully!");
    }

    private static void viewAllProducts() {
        List<Product> products = dbManager.getAllProducts();

        System.out.println("All Products:");
        for (Product product : products) {
            System.out.println(product.getProductId() + " - " + product.getProductName()
+ " - " + product.getProductPrice());
        }
    }

    private static void addIncomingStock() {
        System.out.print("Enter the product ID: ");
        int productId = scanner.nextInt();

        System.out.print("Enter the quantity of incoming stock: ");
        int quantity = scanner.nextInt();
```

```
        Stock stock = new Stock(productId, quantity, "Incoming", new Date());
        dbManager.addStock(stock);

        System.out.println("Incoming stock added successfully!");
    }

    private static void addOutgoingStock() {
        System.out.print("Enter the product ID: ");
        int productId = scanner.nextInt();

        System.out.print("Enter the quantity of outgoing stock: ");
        int quantity = scanner.nextInt();

        Stock stock = new Stock(productId, quantity, "Outgoing", new Date());
        dbManager.addStock(stock);

        System.out.println("Outgoing stock added successfully!");
    }

    private static void viewStockTransactions() {
        List<Stock> stockTransactions = dbManager.getAllStockTransactions();

        System.out.println("All Stock Transactions:");
        for (Stock stock : stockTransactions) {
            System.out.println(stock.getStockId() + " - Product ID: " + stock.getProductId()
+ " - Product Name: " + stock.getProductName() +
                    " - Quantity: " + stock.getQuantity() + " - Transaction Type: "
 + stock.getTransactionType() +
                    " - Transaction Date: " + stock.getTransactionDate());
        }
    }

    private static void updateProduct() {
        System.out.print("Enter the product ID to update: ");
        int productId = scanner.nextInt();

        scanner.nextLine(); // Consume the newline character after reading the integer
        System.out.print("Enter the new product name: ");
        String productName = scanner.nextLine();
```

```
    System.out.print("Enter the new product price: ");
    int productPrice = scanner.nextInt();

    Product product = new Product(productName);
    product.setProductId(productId);
    product.setProductPrice(productPrice);
    dbManager.updateProduct(product);

    System.out.println("Product updated successfully!");
}

private static void deleteProduct() {
    System.out.print("Enter the product ID to delete: ");
    int productId = scanner.nextInt();

    dbManager.deleteProduct(productId);

    System.out.println("Product deleted successfully!");
}

private static void updateStock() {
    System.out.print("Enter the stock ID to update: ");
    int stockId = scanner.nextInt();

    System.out.print("Enter the new product ID: ");
    int productId = scanner.nextInt();

    System.out.print("Enter the new quantity: ");
    int quantity = scanner.nextInt();

    scanner.nextLine(); // Consume the newline character after reading the integer
    System.out.print("Enter the new transaction type (Incoming/Outgoing): ");
    String transactionType = scanner.nextLine();

    Stock stock = new Stock(productId, quantity, transactionType, new Date());
    stock.setStockId(stockId);
    dbManager.updateStock(stock);

    System.out.println("Stock updated successfully!");
}
```

```
    private static void deleteStock() {
        System.out.print("Enter the stock ID to delete: ");
        int stockId = scanner.nextInt();

        dbManager.deleteStock(stockId);

        System.out.println("Stock deleted successfully!");
    }
}
```

==========

DatabaseManagerService.java

```
package Inventory;

import java.util.List;

public interface DatabaseManagerService {

        void createDatabaseIfNotExists();
        void createTables();
        void addProduct(Product product);
        List<Product> getAllProducts() ;
        void addStock(Stock stock);
        List<Stock> getAllStockTransactions() ;
        void updateProduct(Product product);
        void deleteProduct(int productId);
        void updateStock(Stock stock);
        void deleteStock(int stockId);


}
```

==========

DatabaseManager.java

```
package Inventory;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;


public class DatabaseManager implements DatabaseManagerService {
    private final String url;
    private final String username;
```

```
    private final String password;

    public DatabaseManager(String url, String username, String password) {
        this.url = url;
        this.username = username;
        this.password = password;
    }
    @Override
    public void createDatabaseIfNotExists() {
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String createDatabaseQuery = "CREATE DATABASE IF NOT EXISTS inventory";
            PreparedStatement createDatabaseStatement = conn.prepareStatemen
t(createDatabaseQuery);
            createDatabaseStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void createTables() {
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String createProductTableQuery = "CREATE TABLE IF NOT EXISTS products ("
                    + "productId INT AUTO_INCREMENT PRIMARY KEY, "
                    + "productName VARCHAR(255), "
                    + "productPrice INT)";
            PreparedStatement createProductTableStatement =
conn.prepareStatement(createProductTableQuery);
            createProductTableStatement.executeUpdate();

            String createStockTableQuery = "CREATE TABLE IF NOT EXISTS stock ("
                    + "stockId INT AUTO_INCREMENT PRIMARY KEY, "
                    + "productId INT NOT NULL, "
                    + "quantity INT NOT NULL, "
                    + "transactionType VARCHAR(10) NOT NULL, "
                    + "transactionDate DATE NOT NULL)";
            PreparedStatement createStockTableStatement = conn.prepareStatement
(createStockTableQuery);
            createStockTableStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void addProduct(Product product) {
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String insertQuery = "INSERT INTO products (productName,productPrice)
VALUES (?,?)";
            PreparedStatement preparedStatement = conn.prepareStatement(insertQuery,
Statement.RETURN_GENERATED_KEYS);
            preparedStatement.setString(1, product.getProductName());
            preparedStatement.setInt(2, product.getProductPrice());
            preparedStatement.executeUpdate();
```

```
            ResultSet generatedKeys = preparedStatement.getGeneratedKeys();
            if (generatedKeys.next()) {
                product.setProductId(generatedKeys.getInt(1));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public List<Product> getAllProducts() {
        List<Product> products = new ArrayList<>();
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String query = "SELECT * FROM products";
            Statement statement = conn.createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            while (resultSet.next()) {
                int productId = resultSet.getInt("productId");
                String productName = resultSet.getString("productName");
                int productPrice =resultSet.getInt("productPrice");
                Product product = new Product(productName);
                product.setProductPrice(productPrice);
                product.setProductId(productId);
                products.add(product);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return products;
    }
    @Override
    public void addStock(Stock stock) {
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String insertQuery = "INSERT INTO stock (productId, quantity,
 transactionType, transactionDate) VALUES (?, ?, ?, ?)";
            PreparedStatement preparedStatement = conn.prepareStatement(insertQuery,
Statement.RETURN_GENERATED_KEYS);
            preparedStatement.setInt(1, stock.getProductId());
            preparedStatement.setInt(2, stock.getQuantity());
            preparedStatement.setString(3, stock.getTransactionType());
            preparedStatement.setDate(4, new java.sql.Date(stock.getTransactionDate().
getTime()));
            preparedStatement.executeUpdate();

            ResultSet generatedKeys = preparedStatement.getGeneratedKeys();
            if (generatedKeys.next()) {
                stock.setStockId(generatedKeys.getInt(1));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public List<Stock> getAllStockTransactions() {
```

```
        List<Stock> stockTransactions = new ArrayList<>();
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String query = "SELECT s.*,p.productName FROM stock s join products p on
p.productId=s.productId";
            Statement statement = conn.createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            while (resultSet.next()) {
                int stockId = resultSet.getInt("stockId");
                int productId = resultSet.getInt("productId");
                String productName =resultSet.getString("productName");
                int quantity = resultSet.getInt("quantity");
                String transactionType = resultSet.getString("transactionType");
                Date transactionDate = resultSet.getDate("transactionDate");
                Stock stock = new Stock(productId, quantity, transactionType, transactionDate);
                stock.setStockId(stockId);
                stock.setProductName(productName);
                stockTransactions.add(stock);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return stockTransactions;
    }

    // Add other methods for updating and deleting products and stock
    @Override
    public void updateProduct(Product product) {
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String query = "UPDATE products SET productName=?, productPrice=?
WHERE productId=?";
            PreparedStatement preparedStatement = conn.prepareStatement(query);
            preparedStatement.setString(1, product.getProductName());
            preparedStatement.setInt(2, product.getProductPrice());
            preparedStatement.setInt(3, product.getProductId());
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void deleteProduct(int productId) {
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String query = "DELETE FROM products WHERE productId=?";
            PreparedStatement preparedStatement = conn.prepareStatement(query);
            preparedStatement.setInt(1, productId);
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void updateStock(Stock stock) {
```

```
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String query = "UPDATE stock SET productId=?, quantity=?,
 transactionType=?, transactionDate=? WHERE stockId=?";
            PreparedStatement preparedStatement = conn.prepareStatement(query);
            preparedStatement.setInt(1, stock.getProductId());
            preparedStatement.setInt(2, stock.getQuantity());
            preparedStatement.setString(3, stock.getTransactionType());
            preparedStatement.setDate(4, new java.sql.Date(stock.getTransactionDate()
.getTime()));
            preparedStatement.setInt(5, stock.getStockId());
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void deleteStock(int stockId) {
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            String query = "DELETE FROM stock WHERE stockId=?";
            PreparedStatement preparedStatement = conn.prepareStatement(query);
            preparedStatement.setInt(1, stockId);
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

APPENDIX - II

SCREENSHOTS

7.1 ADDING PRODUCT

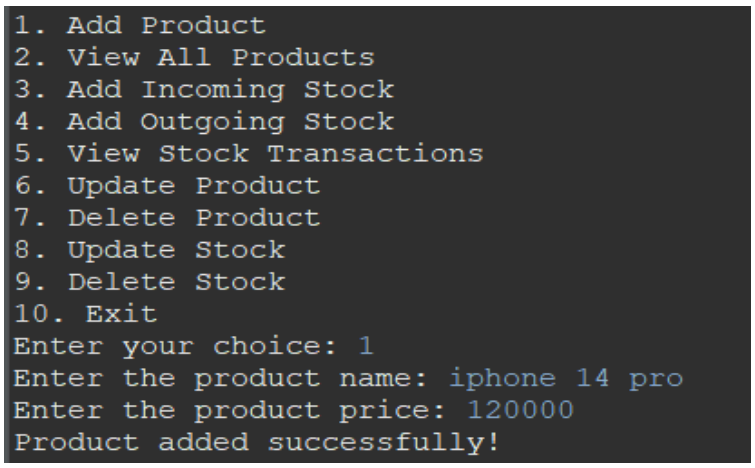


Fig 7.1 : ADDING PRODUCT

7.2 VIEW ALL PRODUCTS

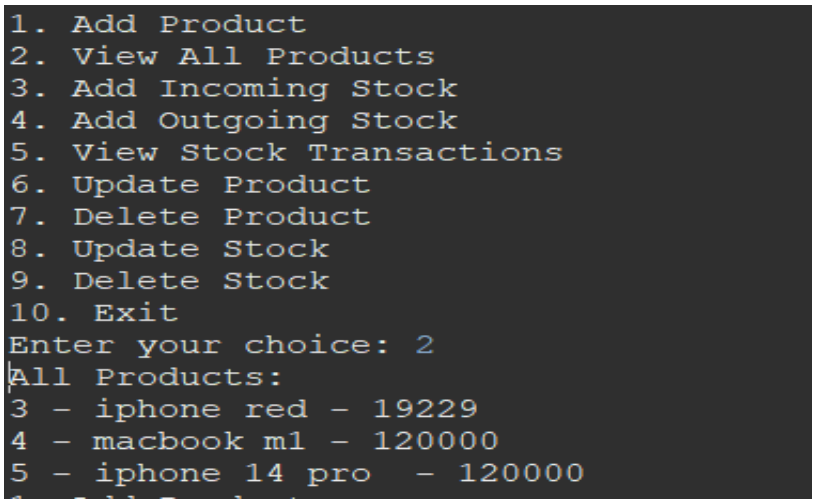


Fig 7.2 : VIEW ALL PRODUCTS

7.3 ADD INCOMING STOCK

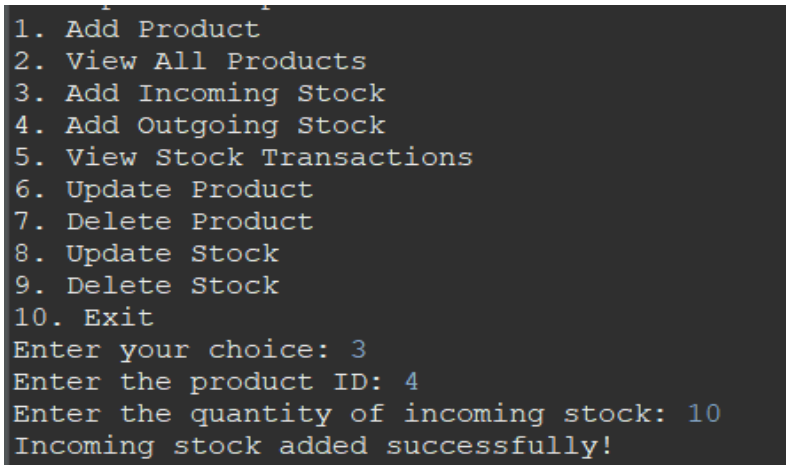


Fig 7.3 :ADD INCOMING STOCK

7.4 VIEW STOCK TRANSACTION

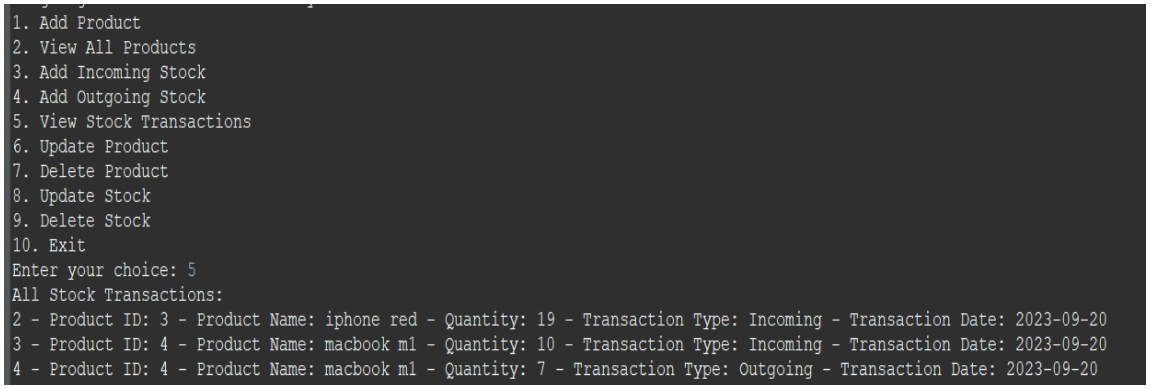


Fig 7.4 : STOCK TRANSACTION

7.5 UPDATE PRODUCT

```
Enter your choice: 6
Enter the product ID to update: 4
Enter the new product name: iphone 14 pro max
Enter the new product price: 150000
Product updated successfully!
1. Add Product
2. View All Products
3. Add Incoming Stock
4. Add Outgoing Stock
5. View Stock Transactions
6. Update Product
7. Delete Product
8. Update Stock
9. Delete Stock
10. Exit
Enter your choice: 2
All Products:
3 - iphone red - 19229
4 - iphone 14 pro max - 150000
5 - iphone 14 pro  - 120000
```

Fig 7.5 : UPDATE PRODUCT

7.6 DELETE PRODUCT

```
Enter your choice: 7
Enter the product ID to delete: 5
Product deleted successfully!
1. Add Product
2. View All Products
3. Add Incoming Stock
4. Add Outgoing Stock
5. View Stock Transactions
6. Update Product
7. Delete Product
8. Update Stock
9. Delete Stock
10. Exit
Enter your choice: 2
All Products:
3 - iphone red - 19229
4 - iphone 14 pro max - 150000
```

Fig 7.6 : DELETE PRODUCT

7.7 ADD OUTGOING STOCK

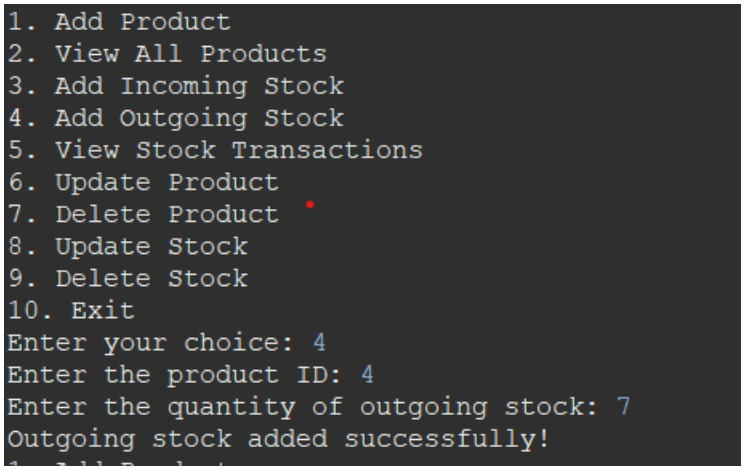


Fig 7.7 : ADD OUTGOING STOCK