GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

AAPP013-4-2-OOP-L-6

UCDF2309ICT(SE)

Member 1 : Heng Xin Hui

Member 2 : Phang Shea Wen

HAND OUT DATE	:	**Thursday, 6th February 2025**
HAND IN DATE	:	**Thursday, 20th March 2025 @ 11:59PM**
WEIGHTAGE	:	**100%**

INSTRUCTIONS TO CANDIDATES:

1. Submit your assignment online in Moodle Folder unless advised otherwise
2. Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld
3. Cases of plagiarism will be penalized
4. You must obtain at least 50% in each component to pass this module

Table of Contents

1.0 Introduction to PPE Inventory Management System 3

1.1 Assumptions on PPE Inventory Management System..... 4

2.0 System Design and Implementation 7

2.1 Object-Oriented Concepts..... 8

3.0 Graphical User Interface (GUI) 15

4.0 Conclusion 39

5.0 References..... 40

1.0 Introduction to PPE Inventory Management System

Personal Protective Equipment (PPE) Inventory Management System is a specialized computer program designated to structurally manage inventory of PPE items stored in a warehouse, seamlessly handling incoming PPE item stocks from multiple suppliers and outgoing PPE item distribution to hospitals. Three primary users are operating this system, including system administrators, warehouse managers, and warehouse staff. Each system user played a significant role in ensuring hospitals receive the necessary PPE items on time while maintaining accurate stock levels:

- Warehouse staff are obliged to actively update inventory stock levels when PPE stocks are received from suppliers or dispatched to hospitals.
- Warehouse managers are obliged to oversee PPE stock levels, review low stock resolution status, generate reports and monitor inventory performance in the PPE warehouse. They may perform CRUD operations on only warehouse staff users. To supervise operational efficiency, managers may use the dedicated reporting functionalities to track item inventory flexibly.
- System administrators are obliged to manage user roles, permissions, and system settings to enforce role-based access control. There is only one super administrator. The super administrator holds the highest level of access control over the PPE Inventory Management System. The super administrator can perform CRUD operations on all users.

Basic functionalities such as user management and initial inventory creation, item inventory update, item inventory tracking, search functionalities, and reporting functionalities basically form the foundation of the system.

1.1 Assumptions on PPE Inventory Management System

In developing the PPE Inventory Management System, several assumptions were made to handle aspects that are not explicitly defined. Based on industry standards and logical inferences, the following assumptions were established:

(I) System Administration

- Exactly one system administrator user is created with the system, with predefined account credentials (ID, password) stored in a separate text file (superAdmin.txt).
- The system administrator performs role-based access control, allowing and restricting permissions to perform certain functions, such as CRUD operations on warehouse manager user accounts and warehouse staff accounts, reset inventory, and reset system.
- The system administrator can perform CRUD operations on other user accounts.
- The system administrator can delete the inventory, reset inventory which clears and initialize the inventory; and reset the system which permanently deletes the inventory, supplier, and hospital data.
- Upon full system reset, a backup of each files will be created and stored.

(II) Role-Based Access Control (RBAC)

- By default, only the system administrator can conduct all the operations, CRUD operation on manager, CRUD on staff, reset inventory, delete inventory, and reset system; the warehouse manager is only permitted to conduct CRUD operations on warehouse staff; while warehouse staff can only perform inventory management operations, does not have the permission to conduct system-level operations.
- The user permissions may be modified at any time by the system administrator.

Types of Users	CRUD on Manager	CRUD on Staff	Reset Inventory	Delete Inventory	Reset System
Super Admin	Allow	Allow	Allow	Allow	Allow
Warehouse Manager	Deny	Allow	Deny	Deny	Deny
Warehouse Staff	Deny	Deny	Deny	Deny	Deny

Table 1.1.1 Default System User Permissions

(III) User Management

- The system has three main users, which are system administrator, warehouse manager, and warehouse staff.
- Each user roles have specific permissions and restrictions by enforcing role-based control, preventing unauthorized activities.

(IV) Inventory Management and Stock Handling

- The PPE inventory file is empty at the first start of the program; the PPE inventory file needs to be initialized with at least one supplier stored into the system before PPE items can be added and initialized.
- Warehouse staff must manually update PPE stock levels upon receiving from suppliers or distributing to hospitals.
- PPE item cannot be distributed if the stock level is below the predefined stock threshold quantity.
- If an PPE item falls below the minimum stock threshold quantity (25 boxes), a Low Stock Alert is triggered.
- Transactions of PPE item received from suppliers or dispatched to hospitals are recorded.

(V) Supplier and Hospital Data Management

- Each supplier and hospital stored into the system are assigned with a sequential code that are unique, refraining duplicate records.
- Each PPE item is supplied by precisely one supplier, but one supplier may supply more than one type of PPE item.
- Only hospitals that are stored into the system can receive PPE distributions.

(VI) System Logging and Security

- All user activities including login attempts, stock updates, PPE transactions, etc are logged for tracking and accountability.
- Failed attempts including failed login attempts, failed user account modifications, permissions denied, etc are logged into an error log for traceability and debugging purposes.
- User authentication using unique user account ID and password is required to access the system.

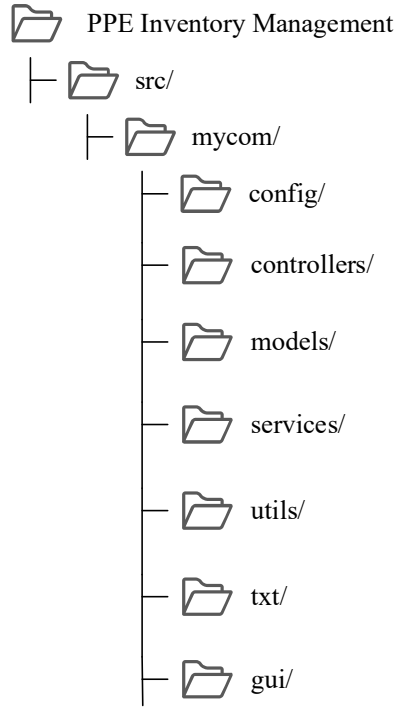
- A strong password is mandated to be set, following common password principles of minimum 8 characters, combinations of uppercase and lowercase letters, digits, and special characters.

(VII) Report Generation & Data Retrieval

- The system supports searching functionalities for users, PPE items, suppliers, and hospitals using keywords of either the unique ID or code assigned or the name.
- Warehouse managers can generate 4 main types of reports, which are PPE inventory report, low stock report, PPE items Received report, and PPE items dispatched report
- Reports may be exported to PDF for documentation and review.

2.0 System Design and Implementation

The development of the PPE Inventory Management System revolves around using Object-Oriented Programming (OOP), providing modularity, maintainability, and scalability. The system design and implementation mainly include justification on how the application of the 6 basic Object-oriented concepts efficaciously structure the system, what file handling approach is utilized to ensure seamless data persistence, and the key functionalities of the system. The system is structurally orchestrated in a layered architecture, categorizing program files into system configurations, controllers, models, services, GUI, reports, and text files, explicitly distinguishing the responsibility of each component.



Package	Responsibility
config/	Holds the system settings.
controllers/	Holds classes that handle user actions and call services methods.
models/	Holds classes representing data entities (User, Suppliers, etc.).
services/	Holds classes that manage business logic (user management, inventory management, etc.).
utils/	Hold utility classes that provide utility methods used across the system.
txt/	Stores text files for data persistence.
gui/	Holds the GUI components for system interaction.

Diagram 2.1 Package-Level-01 Directory Structure of PPE Inventory Management

2.1 Object-Oriented Concepts

The main 6 basic Object-Oriented concepts include object, class, abstraction, encapsulation, polymorphism, and inheritance. These Object-Oriented concepts serve as the bone structure of the PPE Inventory System.

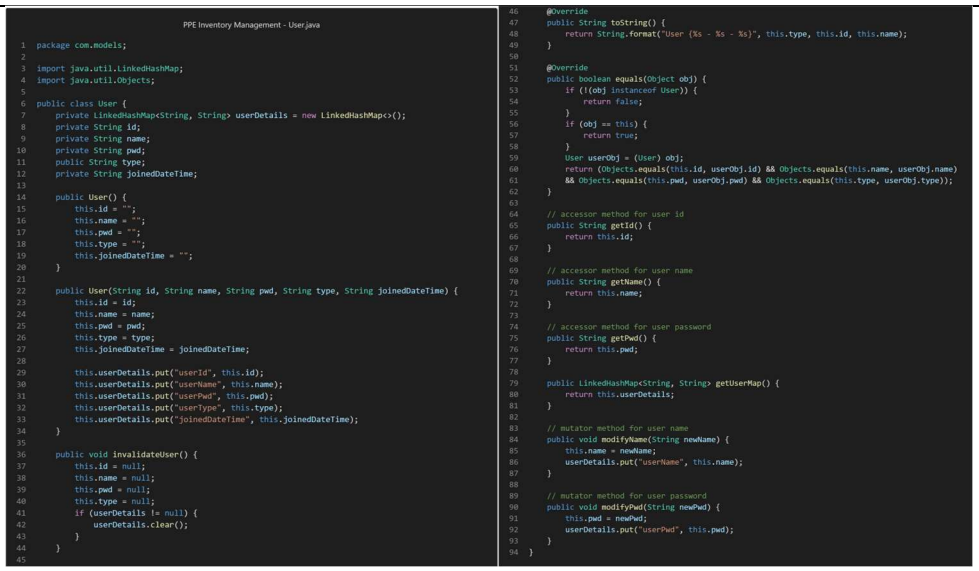
Table 2.1 Object-Oriented Concepts

OOP Concepts	Justification
Object	An object is an instance of a class, holding specific values for the attributes defined within the class. The attributes represent the state, and the methods represent the behavior of the object. Application: • When a user logs in, a **'User'** object is instantiated with user credentials, and user role. A **'UserManagement'** object is also created by passing the **'User'** object as parameter, for the purpose of running user management operations, including assigning the user permissions. • When a new PPE item is added to the inventory, a **'PPEItem'** object is instantiated with an item code, item name, supplier, and stock quantity. A **'PPEManagement'** object is also created to run inventory management operations, including receiving and distributing stocks.

PPEInventory Management System - UserController.java

```
// Login & Logout
    public boolean userLogin(String userId, String pwd) {
        List<User> allUsers = UserManagement.getAllUsers();
        for (User user : allUsers) {
            if (user.getId().equals(userId) && user.getPwd().equals(pwd)) {
                User loginUser = user;
                this.activeUser = loginUser;
                this.userServices = new UserManagement(this.activeUser);
                this.transactionController = new TransactionController(this.activeUser);
                this.ppeController = new PPEController(this.activeUser);
                this.supplierController = new SupplierController(this.activeUser);
                this.hospitalController = new HospitalController(this.activeUser);
                Logger.log(String.format("User {%s - %s - %s} logged in", loginUser.type, userId, loginUser.getName()));
                return true;
            }
        }
        if (UserManagement.getSuperAdmin().getId().equals(userId) && UserManagement.getSuperAdmin().getPwd().equals(pwd)) {
            this.activeUser = UserManagement.getSuperAdmin();
            this.userServices = new UserManagement(this.activeUser);
            this.transactionController = new TransactionController(this.activeUser);
            this.ppeController = new PPEController(this.activeUser);
            this.supplierController = new SupplierController(this.activeUser);
            this.hospitalController = new HospitalController(this.activeUser);
            Logger.log(String.format("User {%s - %s - %s} logged in", this.activeUser.type, userId, this.activeUser.getName()));
            return true;
        }
        Logger.errorLog(String.format("User with ID '%s' failed to logged in\n\tErrorMsg: %s", userId, "Incorrect Login Credentials"));
        return false;
    }
```

Figure 2.1.1 ***UserController*** *controller class –* ***userLogin*** *method*

Class	 *Figure 2.1.2* ***User*** *model class*
Abstraction	Abstraction is the process of hiding the implementation details from the user, showing only the necessary functionalities or features to them. Notably, abstract classes cannot be instantiated, indicating objects cannot be created from an abstract class. • Common data structure and behaviors (attributes and methods) are declared in a parent class of type abstract. • The parent class served as a base class for other sub-classes to inherit from, allowing sub-classes to fully implement its functionalities (attributes and methods). Application: 1. **'Item'** class is declared as abstract, providing common attributes and methods to sub-classes. The sub-classes include the model classes such as **'PPEItem', 'Supplier',** and **'Hospital'** classes, which shares common attributes of 'code' and 'name'; and common accessor methods for 'code' and 'name'.

PPE Inventory Management - Item.java

```java
package com.models;

public abstract class Item {
    protected String code;
    protected String name;

    public Item(String code, String name) {
        this.code = code;
        this.name = name;
    }

    public String getCode() {
        return this.code;
    }

    public String getName() {
        return this.name;
    }
}
```

Figure 2.1.3 ***Item*** *model class*

2. **'PPETransaction'** class is declared as an interface to establish common transaction functionalities, which are adding stock and subtracting stock for **'PPEItem'** objects. The common methods are 'addStock()', and 'subtractStock()'. The system hides the internal processing of updating stock quantity, exposing only the interface of the method (access modifiers, return type, method name, parameters).

PPEInventory Management System - PPETransaction.java

```java
package mycom.models;

public interface PPETransaction {
    boolean addStock(int newStockQuantity);
    boolean subtractStock(int newStockQuantity);
}
```

Figure 2.1.4 ***PPETransaction*** *model class*

Encapsulation	Encapsulation includes specifying the access to a specific data structure or behaviors, enforcing visibility control. Encapsulation provides a layer of protection by restricting access to data and methods. • Access modifier of the attributes and methods needs to be defined (public, protected, private), if unspecified, the access will be at a package-level, only accessible within the same package.

- To access and modify the attribute, an accessor and mutator method is necessary to be declared.

Application:

- model classes, accessor and mutator methods are defined to get or modify the value of attributes such as user details, PPE item details, hospital details, and supplier details that have their access modifier set as private.

PPEInventory Management System - Hospital.java

```java
public class Hospital extends Item {
    // hospitalCode, hospitalName, hospitalContact, hospitalAddress
    public static final String filePath = SystemConfig.hospitalFilePath;
    private String contact, address;
    private LinkedHashMap<String, String> hospitalDetails = new LinkedHashMap<>();

    public String getCode() {
        return this.code;
    }

    public String getName() {
        return this.name;
    }

    public String getContact() {
        return this.contact;
    }

    public String getAddress() {
        return this.address;
    }

    public LinkedHashMap<String, String> getHospitalMap() {
        return this.hospitalDetails;
    }

    // mutator method for hospital name
    public void modifyName(String newName) {
        this.name = newName;
        hospitalDetails.put("hospitalName", this.name);
    }

    // mutator method for hospital contact
    public void modifyContact(String newContact) {
        this.contact = newContact;
        hospitalDetails.put("hospitalContact", this.contact);
    }

    // mutator method for hospital address
    public void modifyAddress(String newAddress) {
        this.address = newAddress;
        hospitalDetails.put("hospitalAddress", this.address);
    }
}
```

*Figure 2.1.5 Accessor and Mutator methods in **Hospital** model class*

- services classes, accessor and mutator methods are defined to get or modify the value of attributes such as all hospitals, all suppliers, all PPE items, and low stock items that have their access modifiers set as private.

PPEInventory Management System - PPEManagement.java

```java
public class PPEManagement {
    private static FileHandler ppehandler = new FileHandler(SystemConfig.ppeItemFilePath);
    private static FileHandler supplierHandler = new FileHandler(SystemConfig.supplierFilePath);
    private static FileHandler transactionHandler = new FileHandler(SystemConfig.transactionFilePath);
    private static FileHandler lowStockHandler = new FileHandler(SystemConfig.lowStockFilePath);
    private static List<PPEItem> allItems = new ArrayList<>();
    private static List<Supplier> allSuppliers = new ArrayList<>();
    private static ArrayList<LinkedHashMap<String, String>> lowStockItems = new ArrayList<>();

    public PPEManagement() {
        loadPPEItems();
        loadSuppliers();
        updateLowStockItems();
    }

    //

    public static List<PPEItem> getAllItems() {
        return allItems;
    }

    public static ArrayList<LinkedHashMap<String, String>> getLowStockItems() {
        updateLowStockItems();
        loadLowStockItems();
        return lowStockItems;
    }

    //

}
```

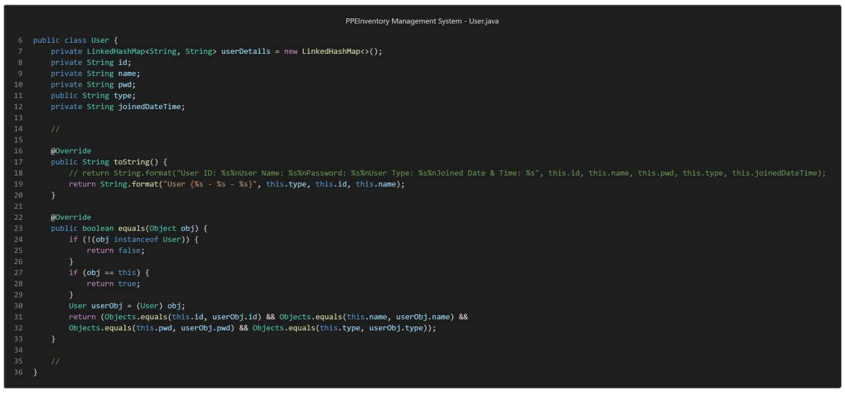
*Figure 2.1.6 Accessor methods in **PPEManagement** services class*

Polymorphism	Polymorphism in Java refers to the ability to define the same methods in multiple varied forms. • Involves method overloading and method overriding. • Method overloading refers to having multiple methods of the same name but with different signatures, where the parameter list varies. • Method overriding refers to having multiple methods of the same name and of the same signature, where both the method name and parameter list are the same, but differ in the implementation statement. • Polymorphism in most cases is closely tied to inheritance, where methods defined in a sub-class are also defined in the super-class. Application: (I) Constructor Overriding and Overloading • No-argument constructor of classes is defined, overriding the default constructor provided by the java virtual machine (jvm) by applying specific attribute initialization upon object instantiation. • Argument constructor of classes is defined, overloading the other constructors defined within the class scope by declaring the parameters.

PPEInventory Management System - Supplier.java

```
9  public class Supplier extends Item {
10     // supplierCode, supplierName, supplierContact, supplierAddress
11     public static final String filePath = SystemConfig.supplierFilePath;
12     private String contact, address;
13     private LinkedHashMap<String, String> supplierDetails = new LinkedHashMap<>();
14
15     public Supplier() {
16         super("", "");
17         this.contact = "";
18         this.address = "";
19     }
20
21     public Supplier(String code, String name, String contact, String address) {
22         super(code, name);
23         this.contact = contact;
24         this.address = address;
25         supplierDetails.put("supplierCode", getCode());
26         supplierDetails.put("supplierName", getName());
27         supplierDetails.put("supplierContact", this.contact);
28         supplierDetails.put("supplierAddress", this.address);
29     }
30     //
31 }
```

*Figure 2.1.7 No-argument constructure and Argument constructor in **Supplier** model class*

	(II) Other Method Overriding • **'PPEItem'** class implements the **'PPETransaction'** interface and overrides the 'addStock()' and 'subtractStock()' method to specifically modify stock quantity.' • model classes such as **'User', 'PPEItem', 'Supplier' and 'Hospital'** override the default 'toString()' method, producing custom string representations.  PPEInventory Management System - PPEItem.java *Figure 2.1.8 addStock() and subtractStock() method overriding in **PPEItem** model class*  PPEInventory Management System - User.java *Figure 2.1.9 toString() and equals() method overriding in **User** model class*
Inheritance	Inheritance is the ability to derive new classes from existing classes. The derived classes can inherit the attributed and methods of the super class. Inheritance allows polymorphism by defining methods of the same name with existing methods in the superclass, allowing method overriding and method

overloading. The child classes may add more specific data structure and behavior on top of the super class.

Application:

- Custom exception classes such as **'UserException', 'PPEItemException', 'SupplierException',** and **'HospitalException'** extend the java virtual machine (jvm) provided **'Exception'** class.
- The child classes derived from the **'Exception'** superclass makes error handling more structured, readable, and maintainability. Having specific exception classes allows capturing of different types of errors independently.
- Hybrid inheritance, such as **'PPEItem'** class extends **'Item'** class and implements **'PPETransaction'** interface, implementing multiple inheritance that is not supported by the java virtual machine (jvm) by default.

PPEInventory Management System - UserException.java

```
3  public class UserException extends Exception {
4      public UserException(String errorMsg) {
5          super("User Handling Error: " + errorMsg);
6      }
7  }
```

Figure 2.1.10 ***UserException*** *utils class inheriting* ***Exception*** *class*

PPEInventory Management System - PPEItem.java

```
8  public class PPEItem extends Item implements PPETransaction {
9      // itemCode, itemName, supplierCode, stockQuantity
10     public static final String filePath = SystemConfig.ppeItemFilePath;
11     private String supplier, lastRestockDateTime;
12     private int quantity;
13     private LinkedHashMap<String, String> details = new LinkedHashMap<>();
14
15     public PPEItem() {
16         super("", "");
17         this.supplier = "";
18     }
19
20     public PPEItem(String code, String itemName, String lastRestockDateTime) {
21             super(code, itemName);
22             this.supplier = "";
23             this.quantity = 100;
24             this.lastRestockDateTime = lastRestockDateTime;
25             details.put("itemCode", getCode());
26             details.put("itemName", getName());
27             details.put("supplierCode", this.supplier);
28             details.put("stockQuantity", String.valueOf(this.quantity));
29             details.put("lastRestockDateTime", this.lastRestockDateTime);
30     }
31
32     //
33
34     @Override
35     public boolean addStock(int newStockQuantity) {
36         if (newStockQuantity > 0) {
37             this.quantity += newStockQuantity;
38             details.put("stockQuantity", String.valueOf(this.quantity));
39             return true;
40         } else {
41             return false;
42         }
43     }
44
45     @Override
46     public boolean subtractStock(int newStockQuantity) {
47         if (newStockQuantity > 0 && this.quantity >= newStockQuantity) {
48             this.quantity -= newStockQuantity;
49             details.put("stockQuantity", String.valueOf(this.quantity));
50             return true;
51         } else {
52             return false;
53         }
54     }
55
56     //
57 }
```

Figure 2.1.10 ***PPEItem*** *model class inheriting* ***Item*** *and* ***PPETransaction*** *model class*

3.0 Graphical User Interface (GUI)

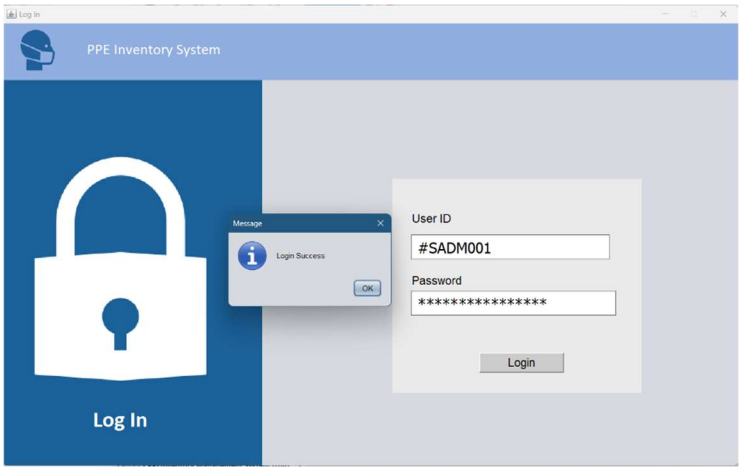


Diagram 3.1: Confirmation message displayed when the Super Admin successfully logs in

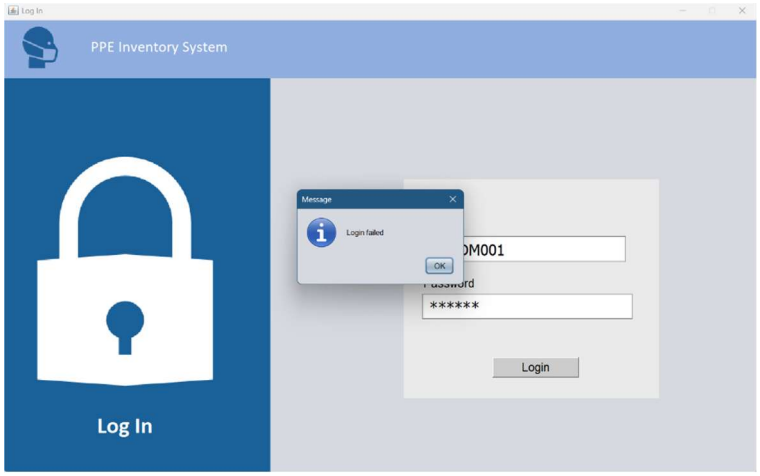


Diagram 3.2: Error message output when Super Admin inputs invalid User ID or password

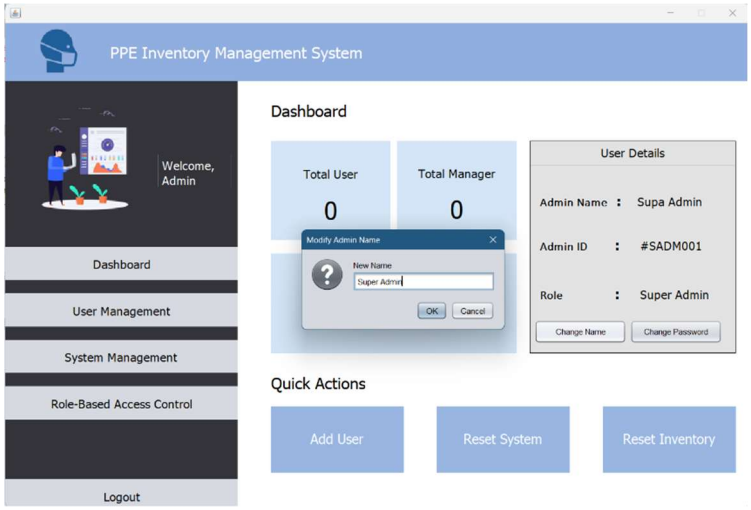


Diagram 3.3: Input field message when the Super Admin changes the name

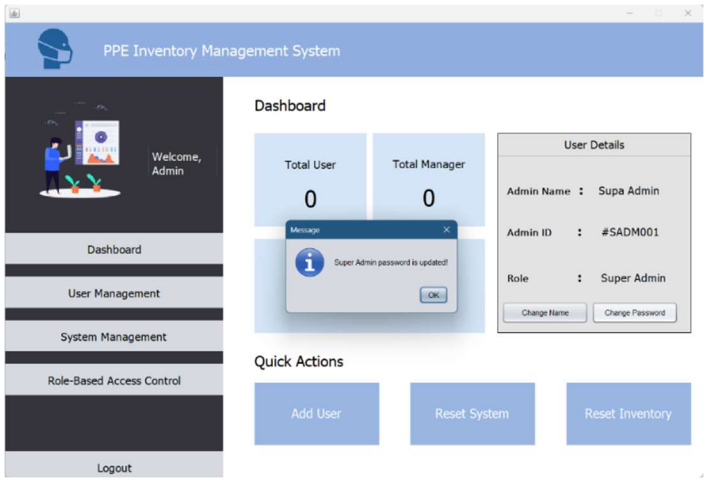


Diagram 3.4: Confirmation message output when the Super Admin successfully updated the password

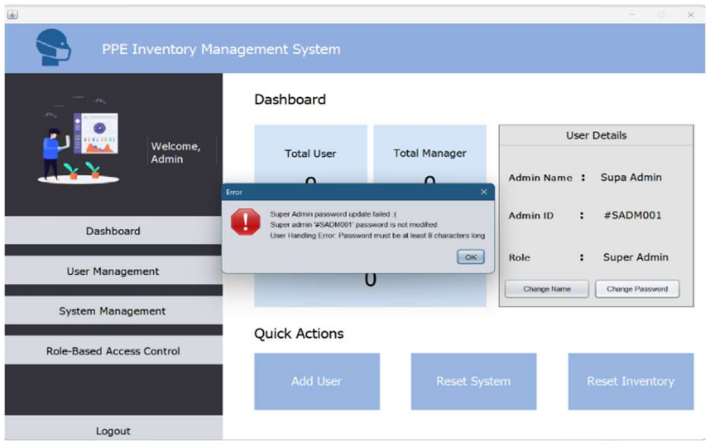


Diagram 3.5: Error message when the Super Admin changes the password that does not meet the requirements

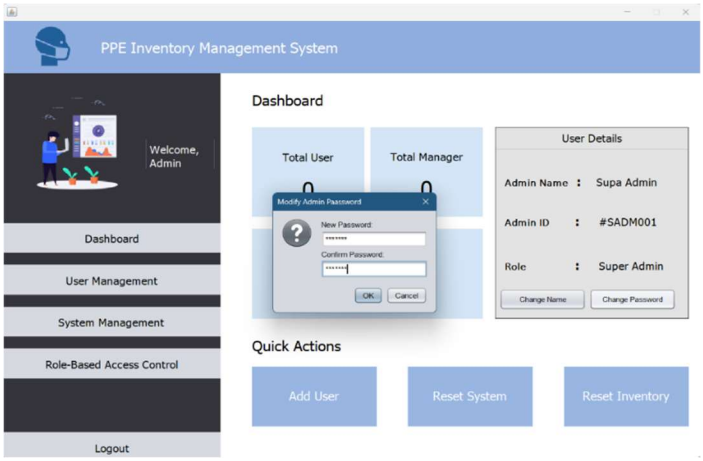


Diagram 3.6: Input field message when the Super Admin change password

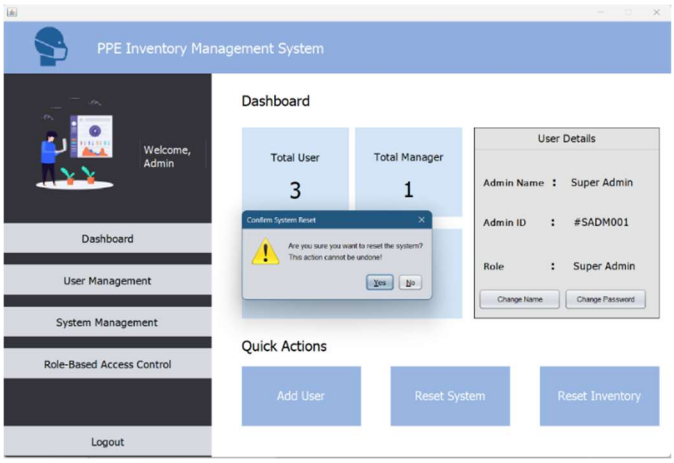


Diagram 3.7: Confirmation message asking the Super Admin to confirm the system reset action.

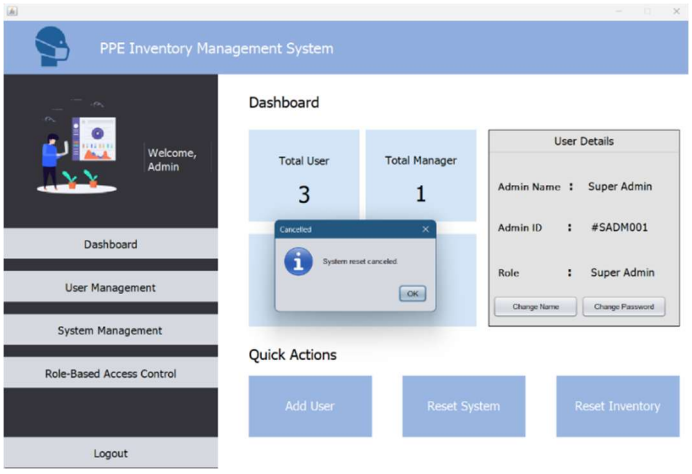


Diagram 3.8: System reset cancellation confirmation message

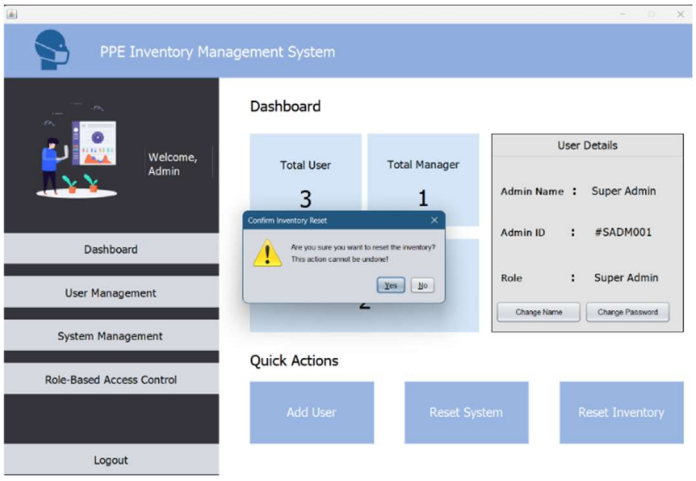


Diagram 3.9: Confirmation message asking the Super Admin to confirm the Inventory reset action.

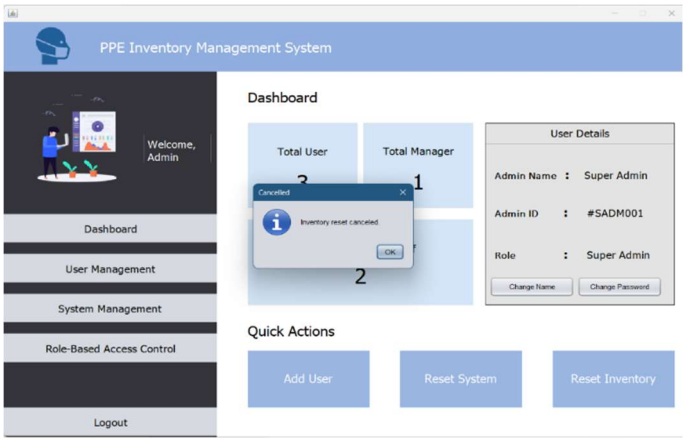


Diagram 3.10: Inventory reset cancellation confirmation message

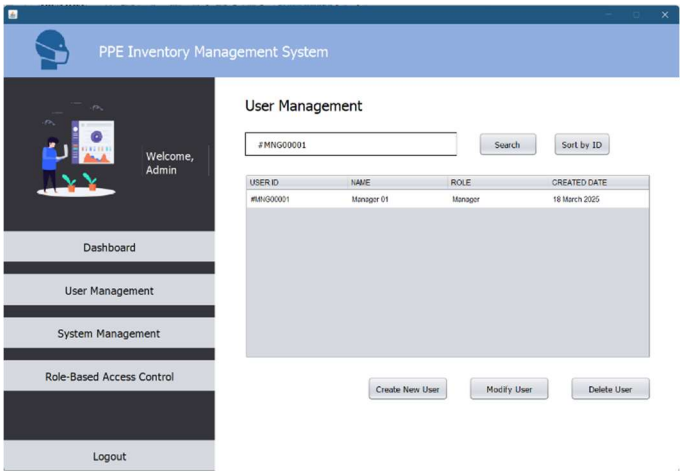


Diagram 3.11: Input field message when Super Admin searches by user ID

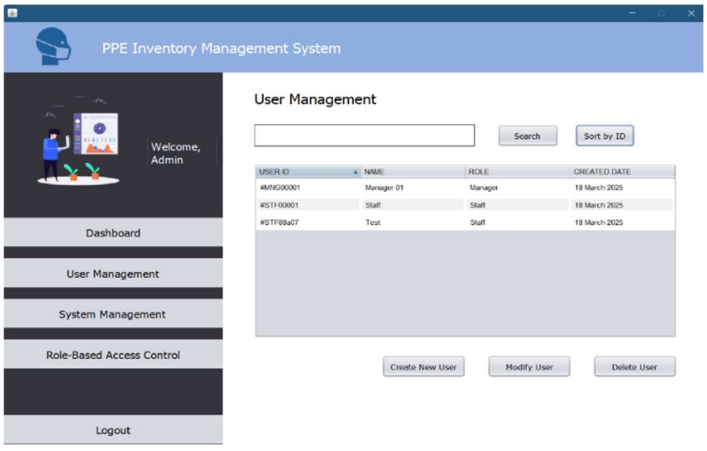


Diagram 3.12: Sort users by clicking the 'Sort by ID' button

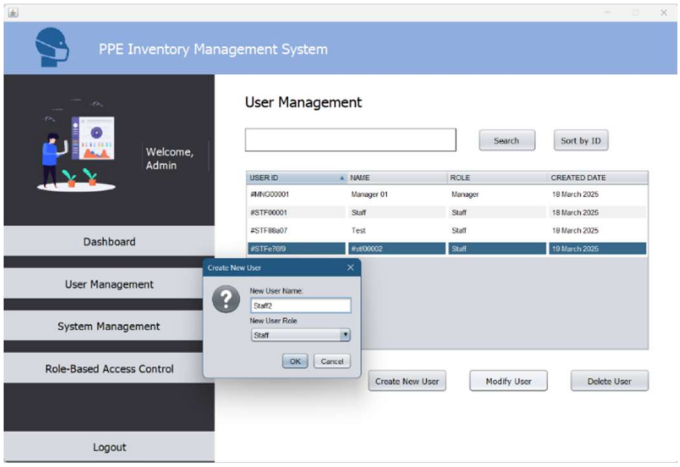


Diagram 3.13: Create New User input field message

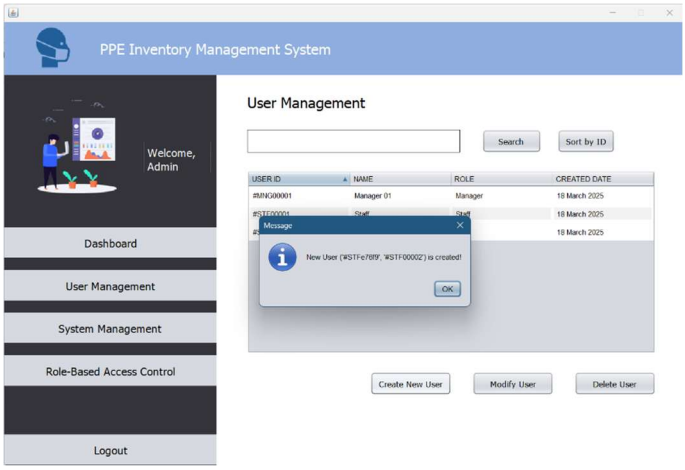


Diagram 3.14: Confirmation message of Creating New User

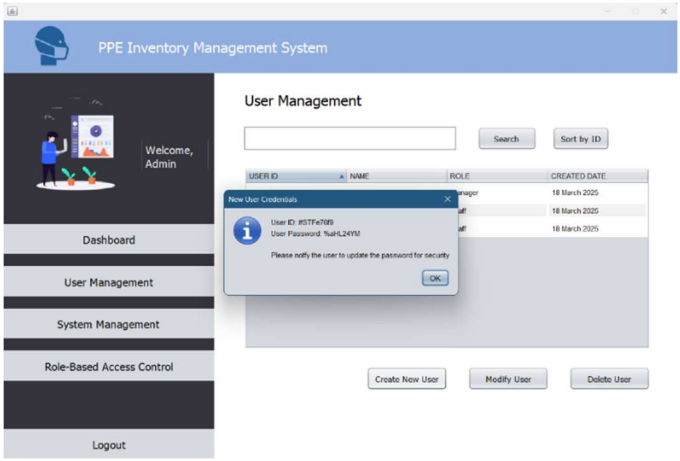


Diagram 3.15: Confirmation message showing the newly created User ID and password

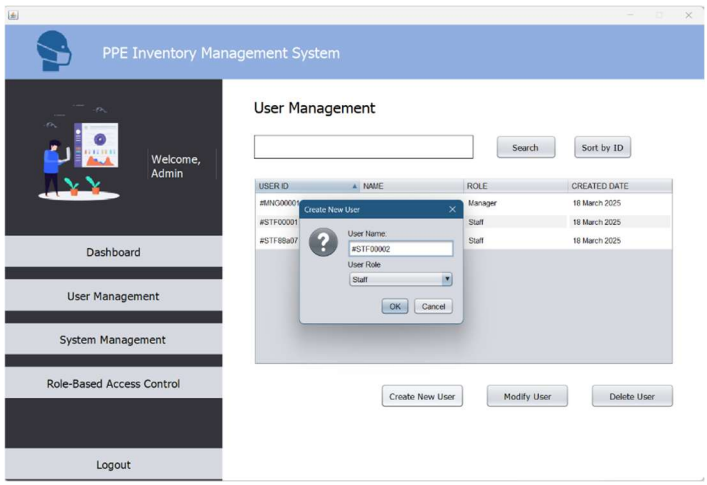


Diagram 3.16: Input field message when Super Admin modifies user

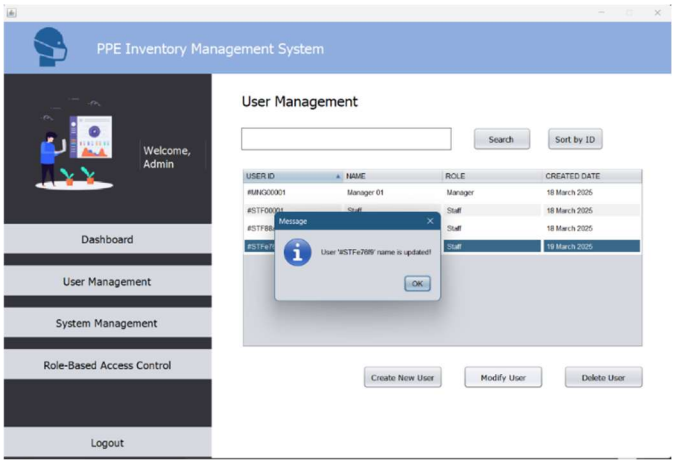


Diagram 3.17: Confirmation message of Modify User

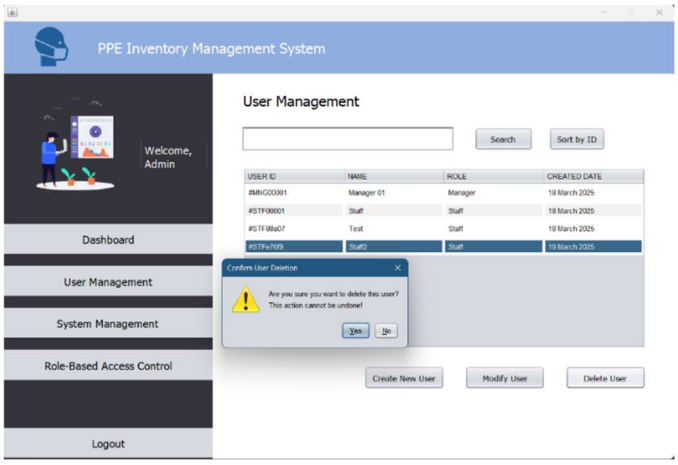


Diagram 3.18: Confirmation message asking Super Admin whether to delete the user

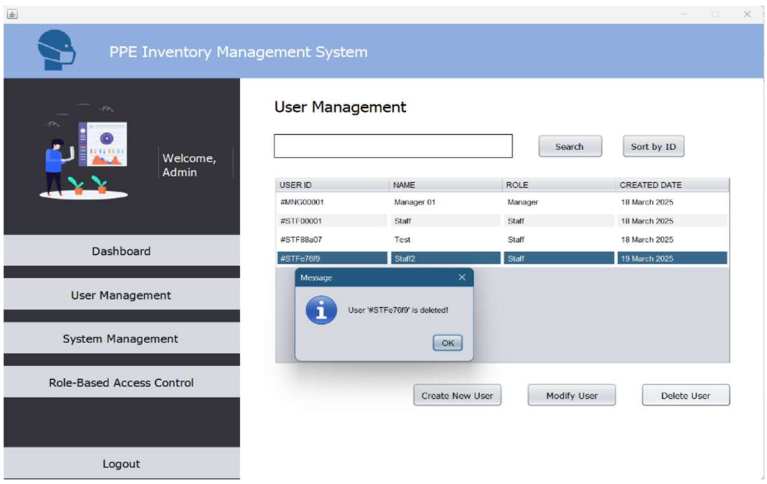


Diagram 3.19: Confirmation message of Delete User

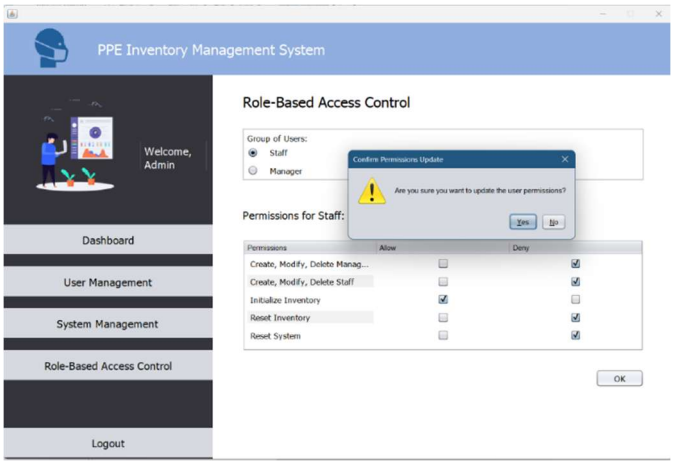


Diagram 3.20: Confirmation message of update user permissions

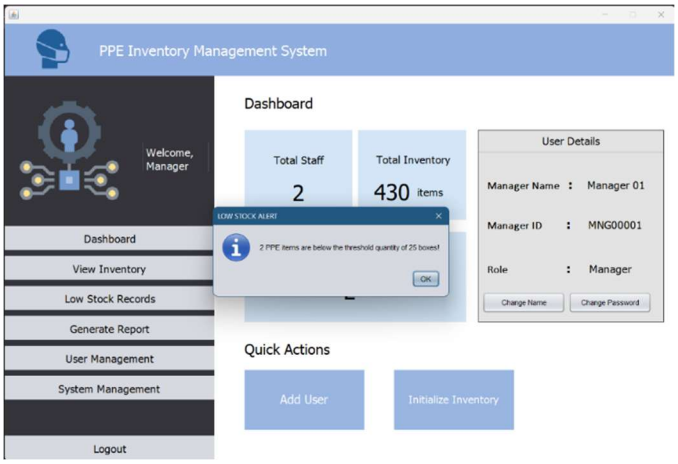


Diagram 3.21: Low stock alert reminder message when the manager logs in

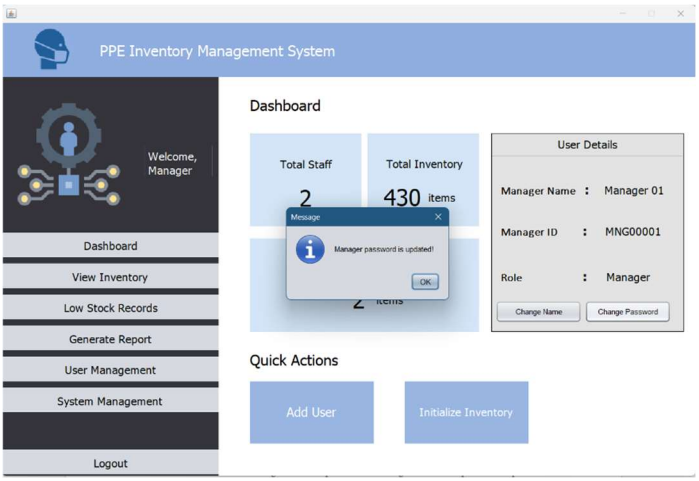


Diagram 3.22: Confirmation message output when the Manager successfully updated the password

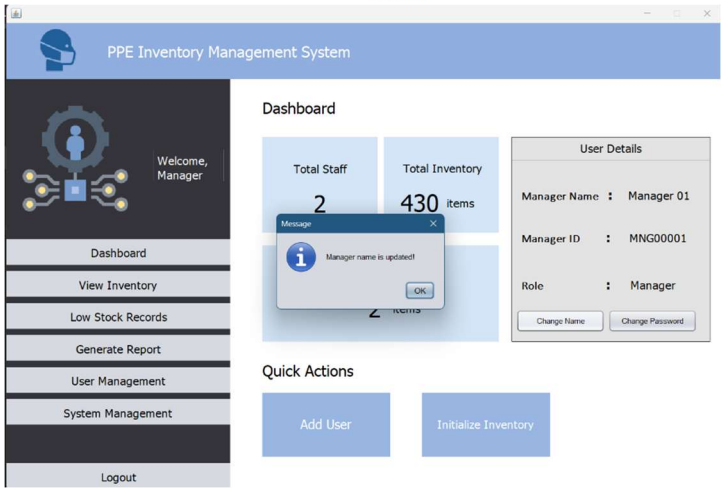


Diagram 3.23: Confirmation message displayed when the Manager successfully changed the name

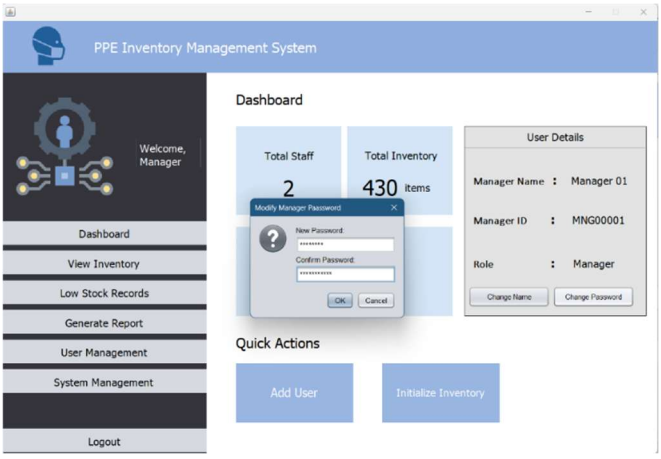


Diagram 3.24: Input field message when the Manager changes the password

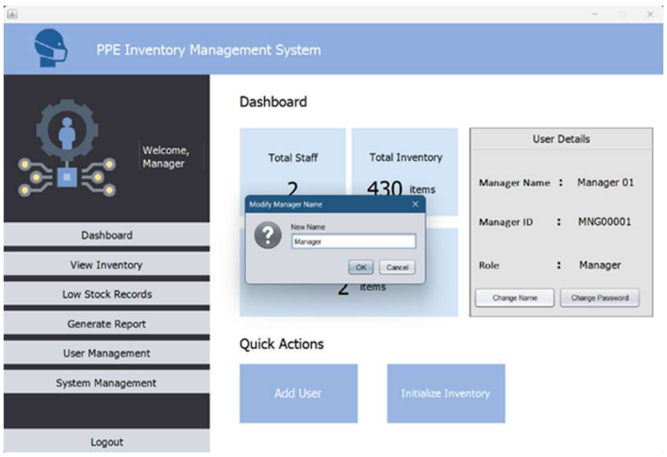


Diagram 3.25: Input field message when modifying the manager's name

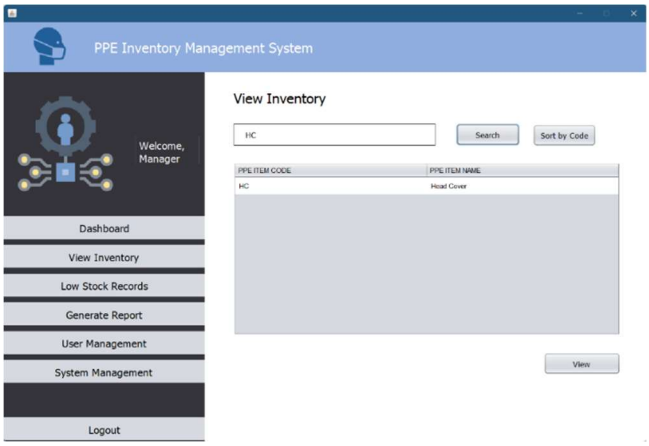


Diagram 3.26: Message output when the manager searches inventory by entering a PPE item code

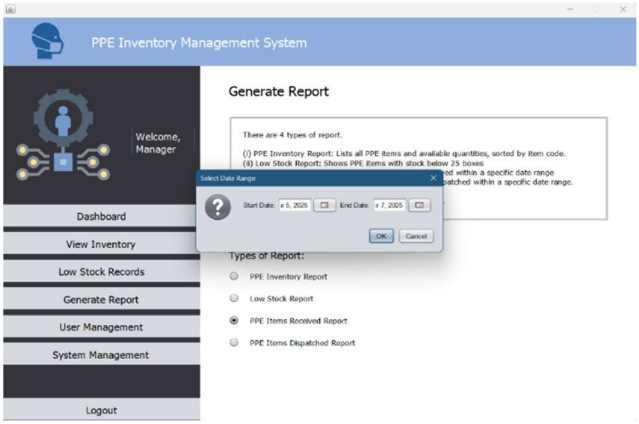


Diagram 3.27: Select Date Range field for generating the PPE Items Received Report by the manager

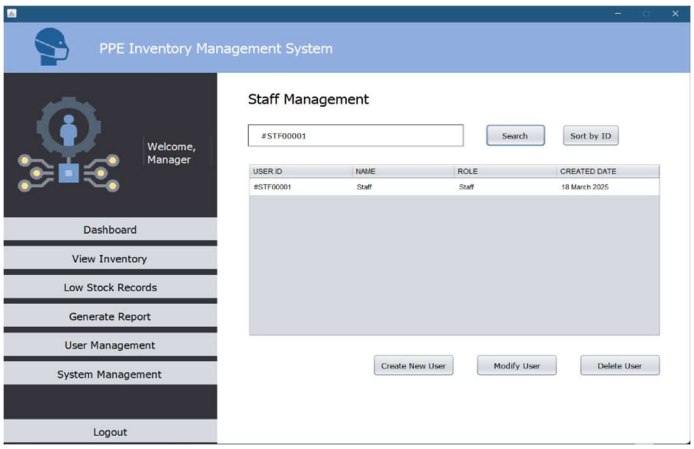


Diagram 3.28: Input field message when Manager searches by user ID

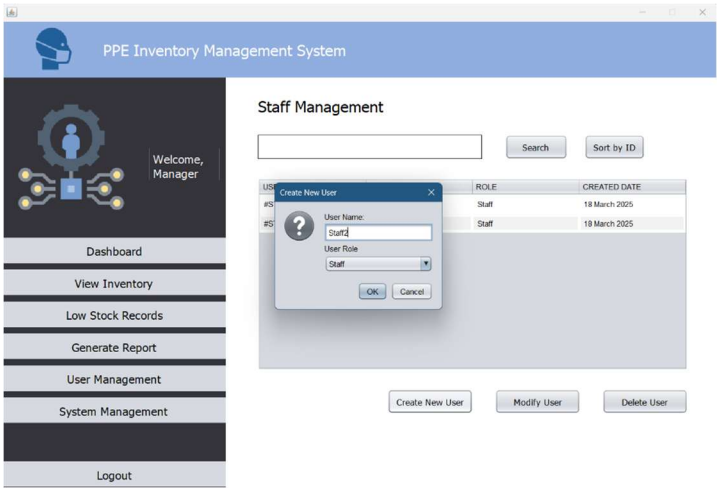


Diagram 3.29: Input field message when Manager create new user

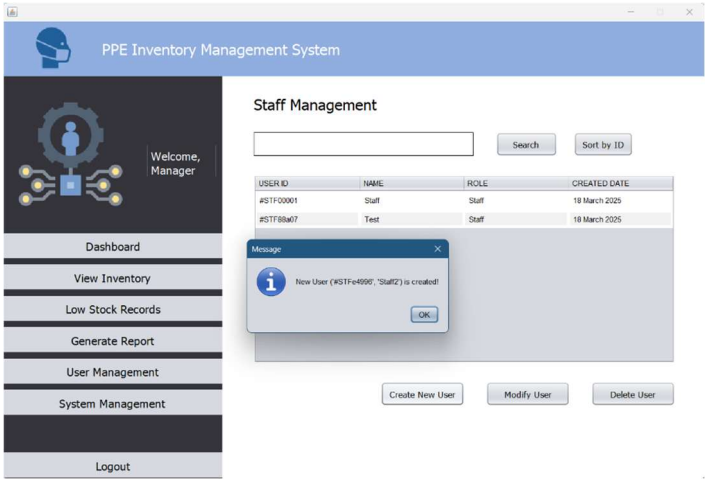


Diagram3.30: Confirmation message of creating new user

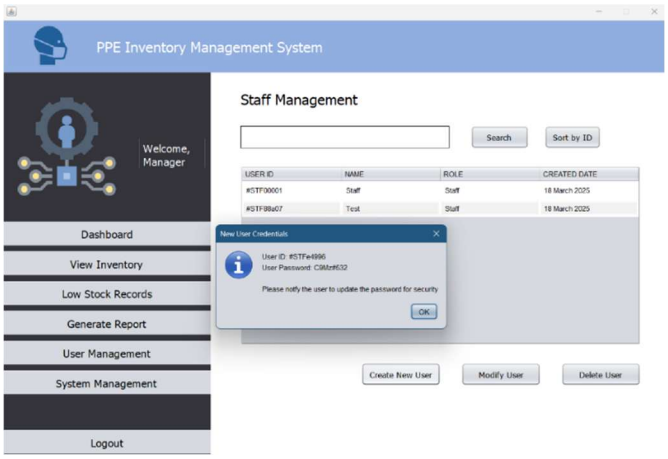


Diagram 3.31: Confirmation message showing the newly created User ID and password

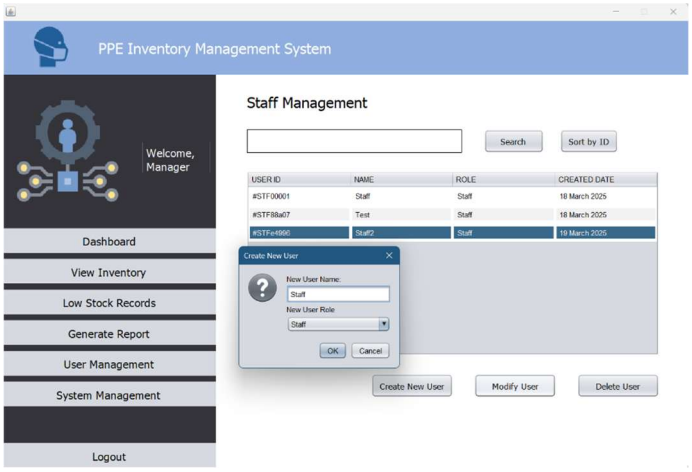


Diagram 3.32: Input field message when Manager modify user

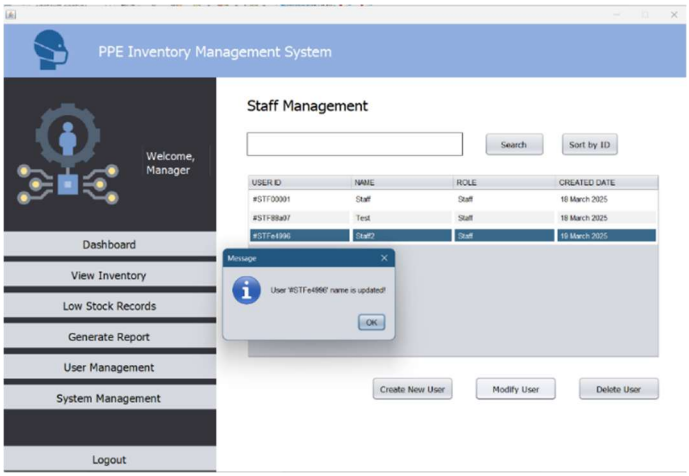


Diagram 3.33: Confirmation message of Modify User

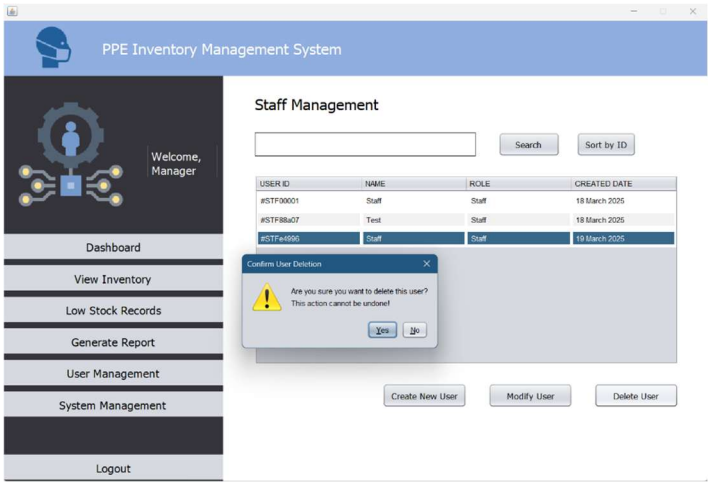


Diagram 3.34 Confirmation message asking Manager whether to delete the user

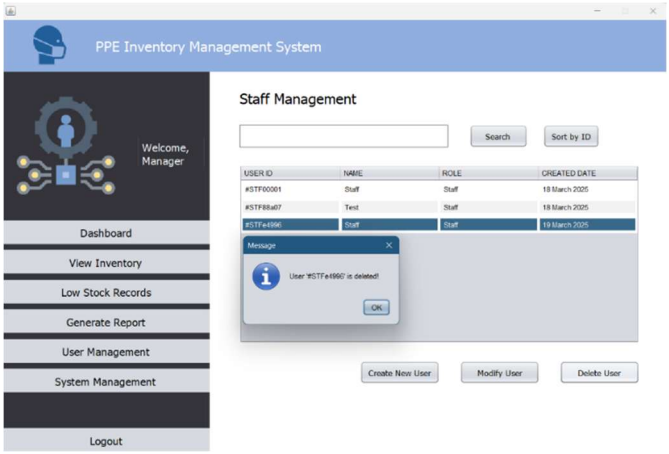


Diagram 3.35: Confirmation message of Delete User

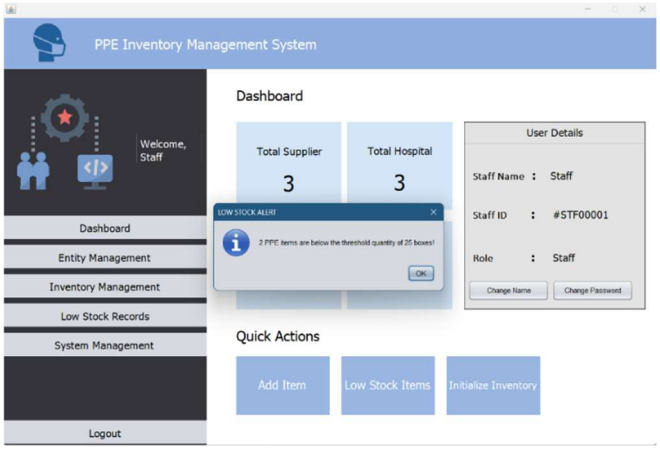


Diagram 3.36: Low stock alert reminder message when the staff logs in

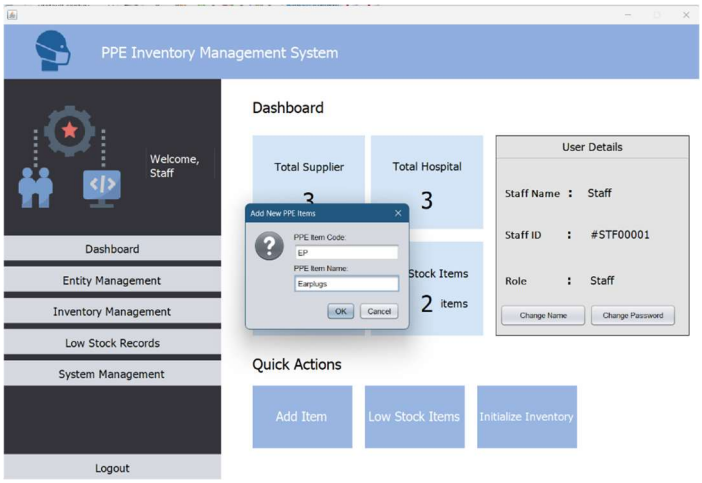


Diagram 3.37: Input field for adding items after clicking the Quick Actions button

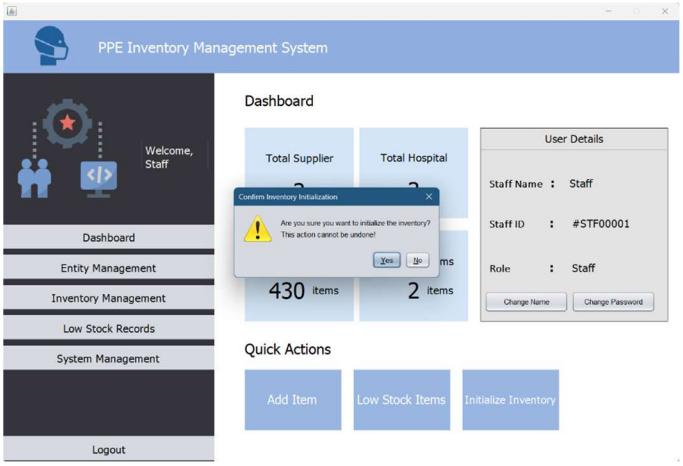


Diagram 3.38: Confirmation message of Initialize Inventory

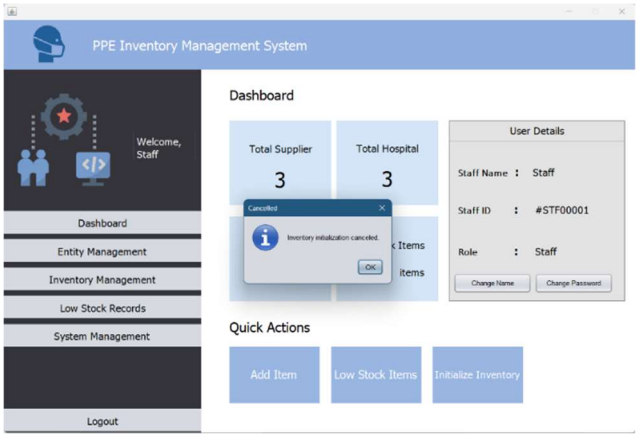


Diagram 3.39: Initialize Inventory cancellation confirmation message

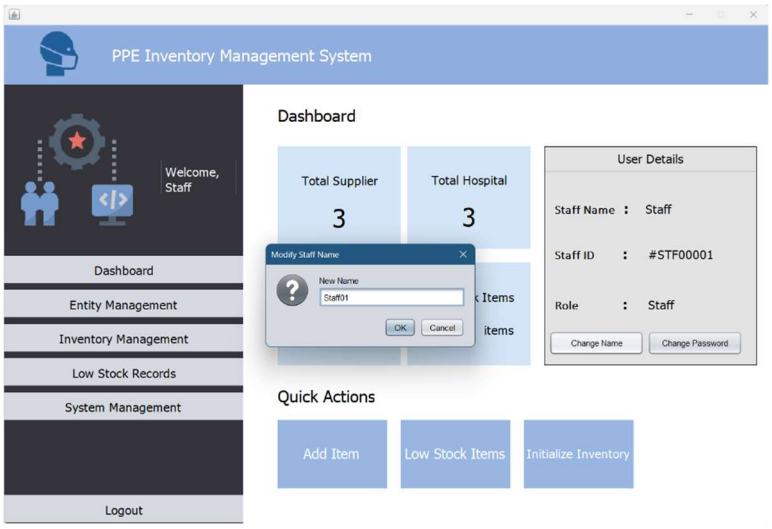


Diagram 3.40: Input field of Modify Staff Name

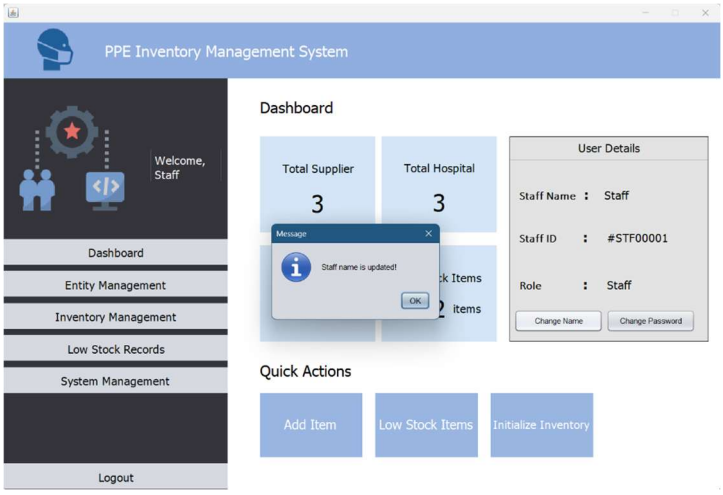


Diagram 3.41: Confirmation message displayed when the Staff successfully changed the name

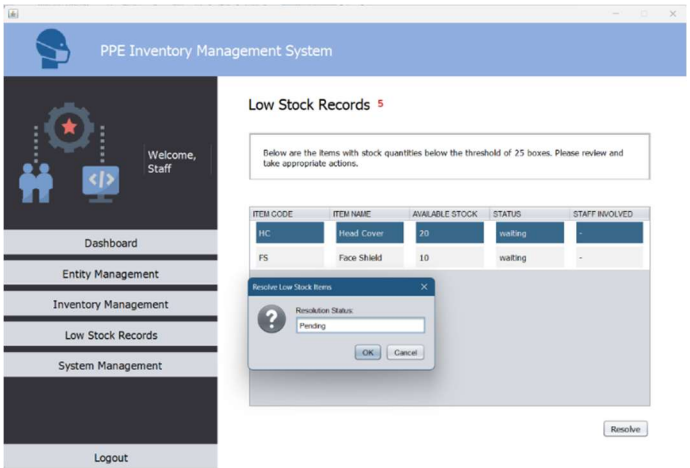


Diagram 3.42: Input field of Resolve Low Stock Items

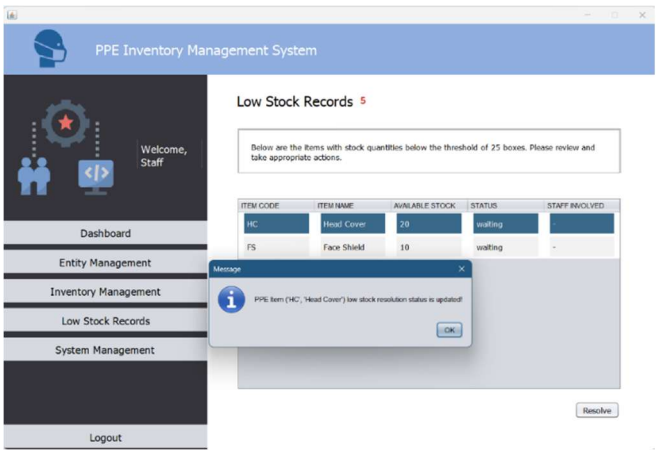


Diagram 3.43: Confirmation message for updating the low stock resolution status.

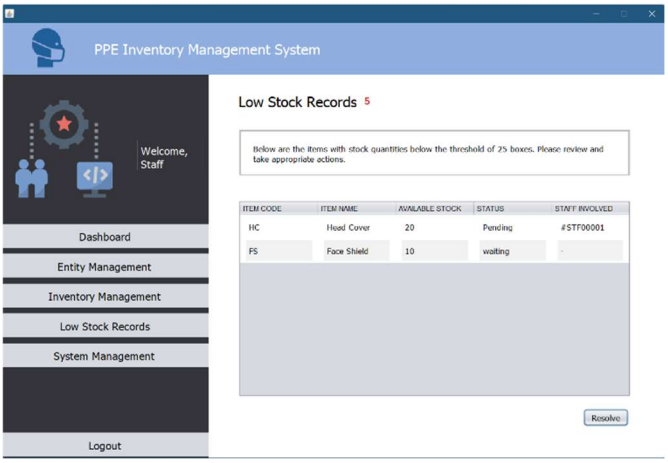


Diagram 3.44: Output of the updated status after completing the update action

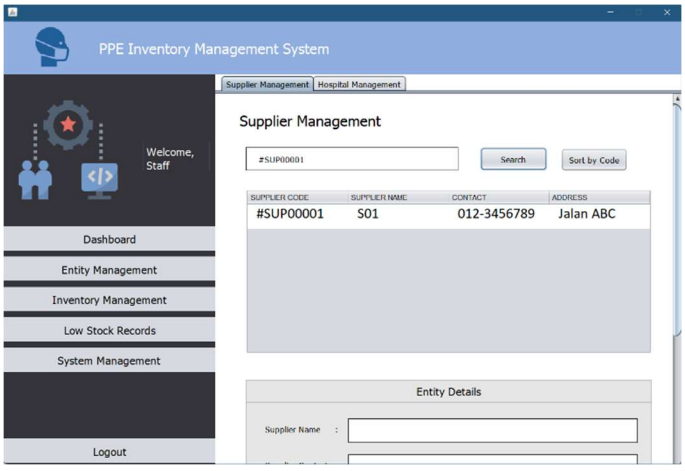


Diagram 3.45: Result message displayed when staff searches by Supplier Code

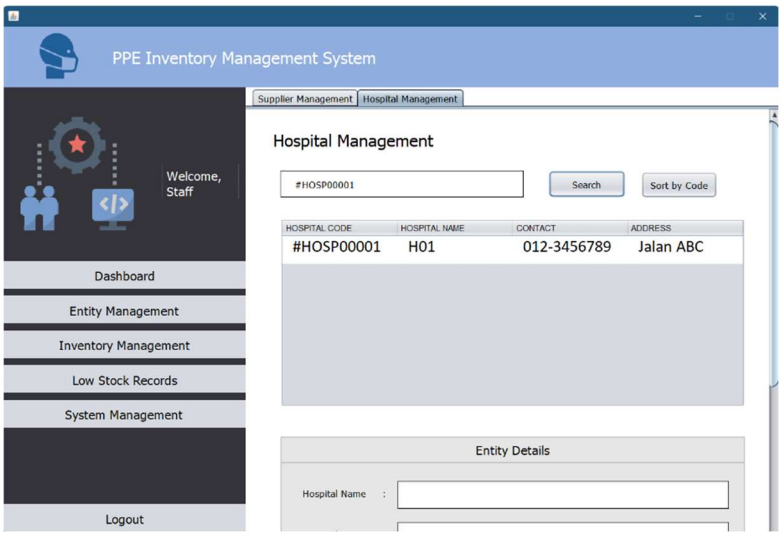


Diagram 3.46: Result message displayed when staff searches by Hospital Code

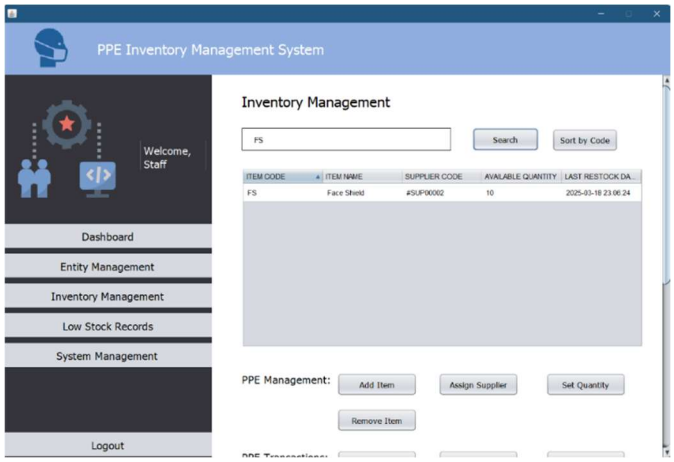


Diagram 3.47: Result message displayed when staff searches by Item Code

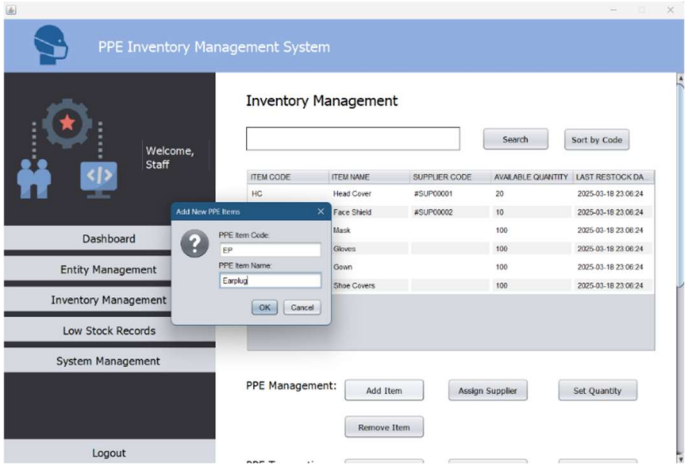


Diagram 3.48: Add New PPE Items input field displayed when clicking the Add Item button

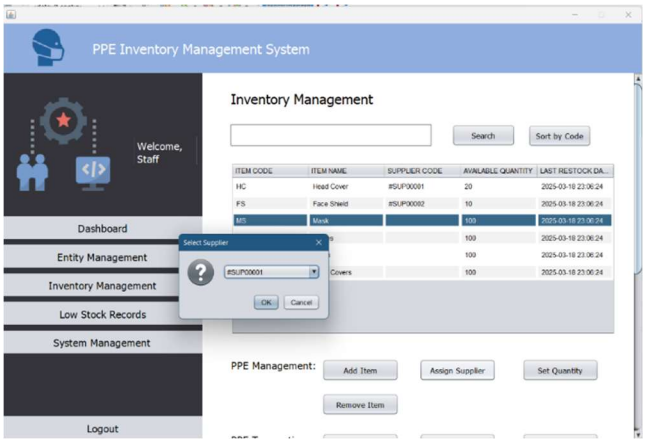


Diagram 3.49: Select Supplier message displayed when clicking the Assign Supplier button

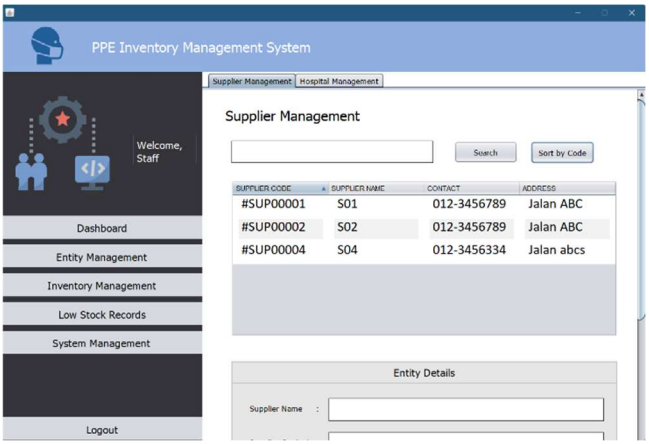


Diagram 3.50: Output after clicking the Sort by Code button on the Supplier Management page

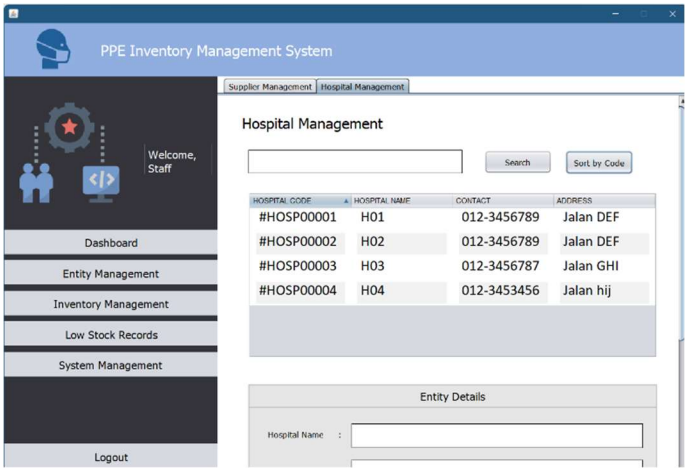


Diagram 3.51: Output after clicking the Sort by Code button on the Hospital Management page

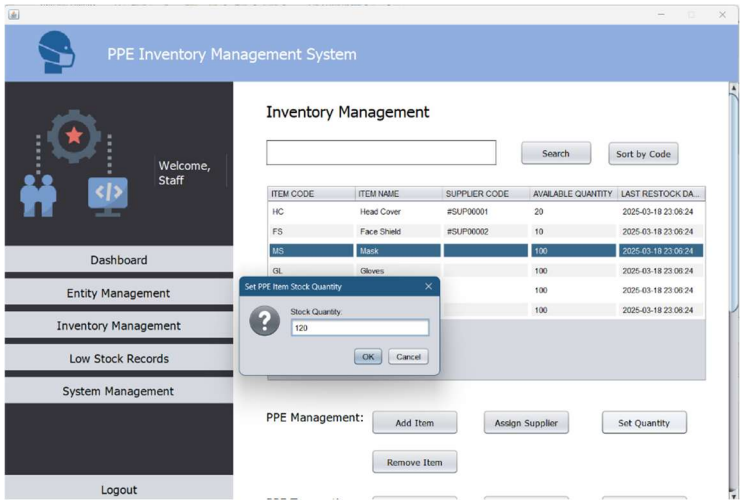


Diagram 3.52: Select Supplier message displayed when clicking the Assign Supplier button

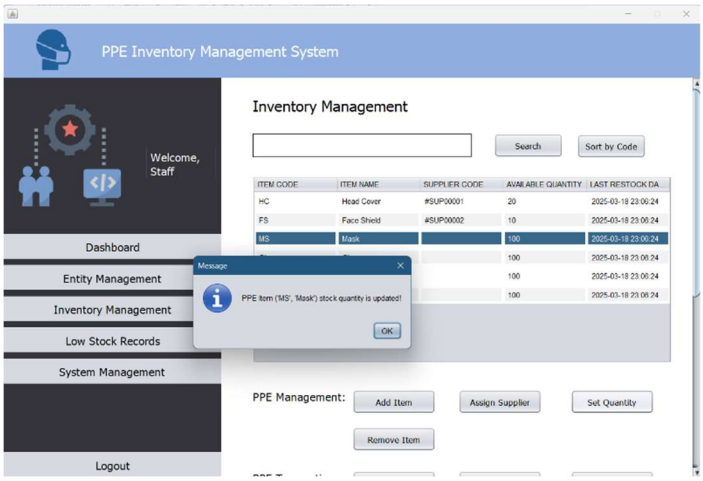


Diagram 3.53: Confirmation message of set quantity

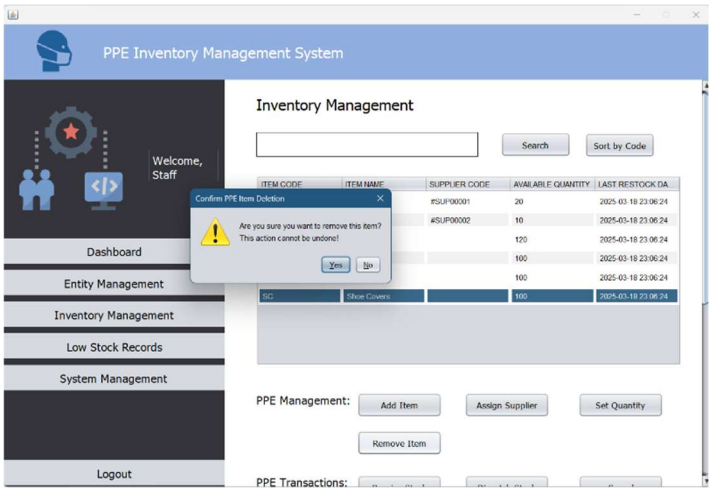


Diagram 3.54: Confirmation message of Remove PPE Item

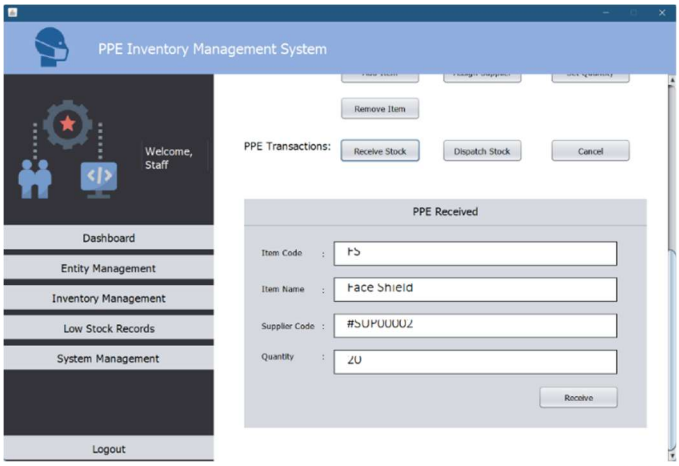


Diagram 3.55: Input of PPE receive quantity

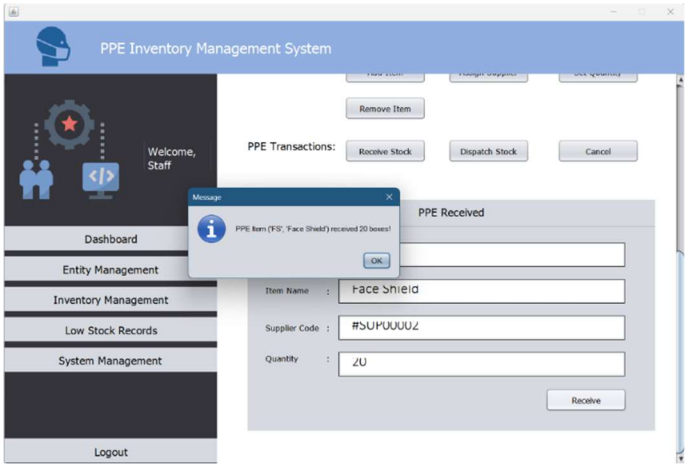


Diagram 3.56: Confirmation message for received PPE item quantity

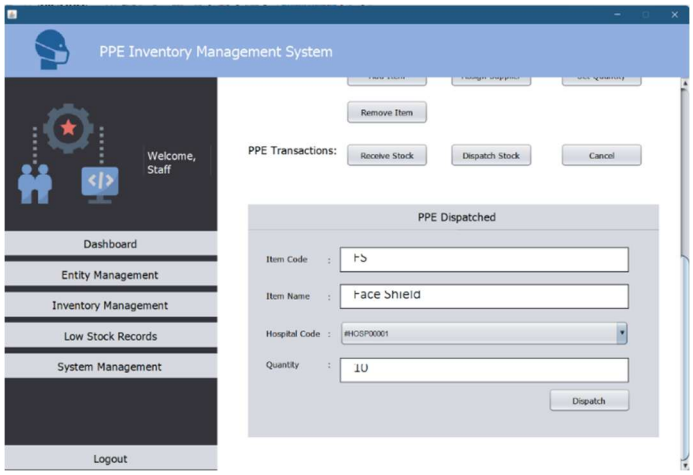


Diagram 3.57: Input of PPE Dispatched quantity

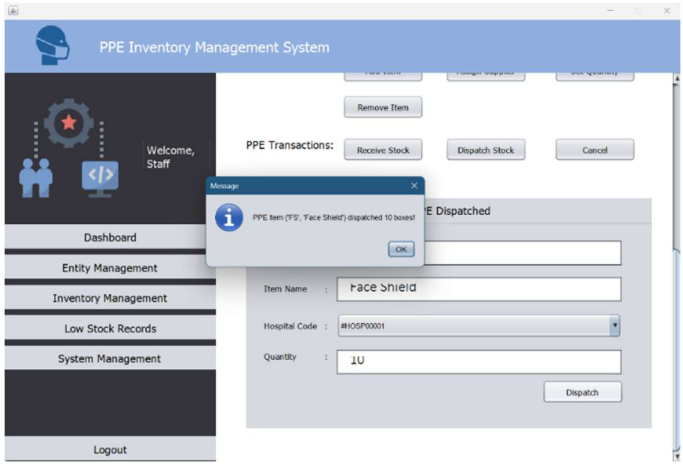


Diagram 3.58: Confirmation message for dispatched PPE item quantity

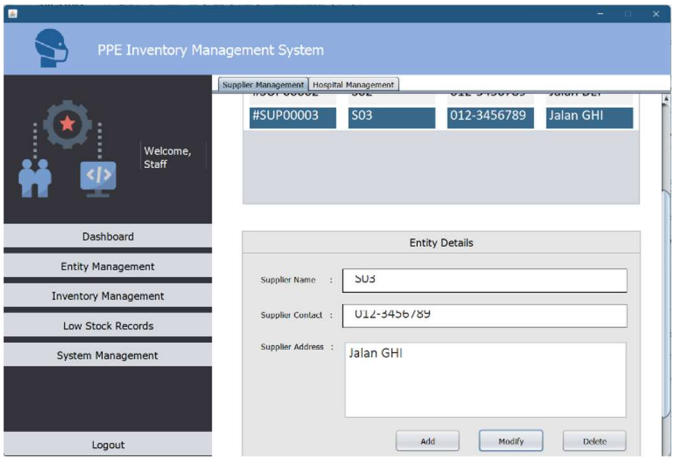


Diagram 3.59: Input Fields to Modify Supplier Details

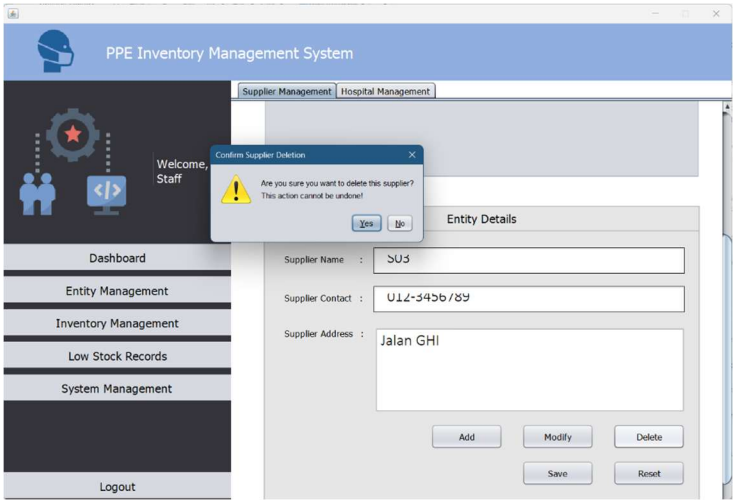


Diagram 3.60: Confirmation Message to Delete Supplier

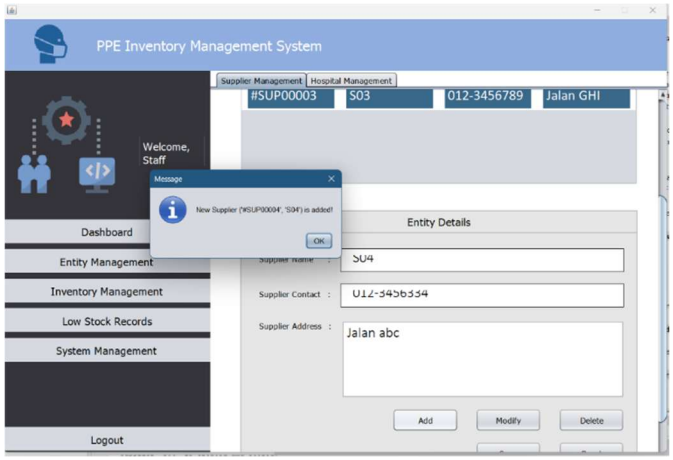


Diagram 3.61: Confirmation Message for Adding a New Supplier

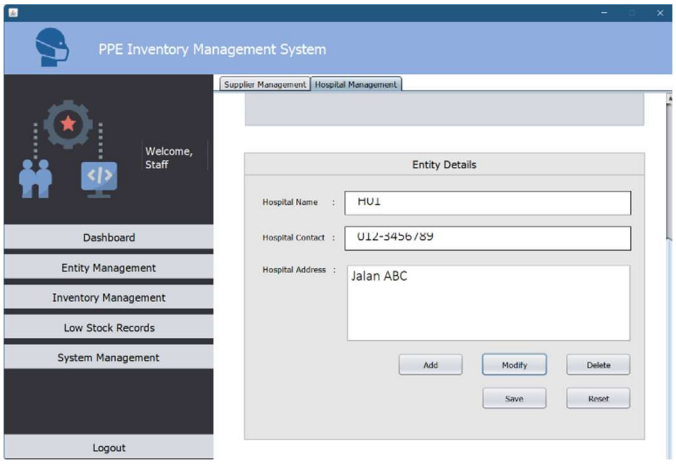


Diagram 3.62: Input Fields to Modify Hospital Details

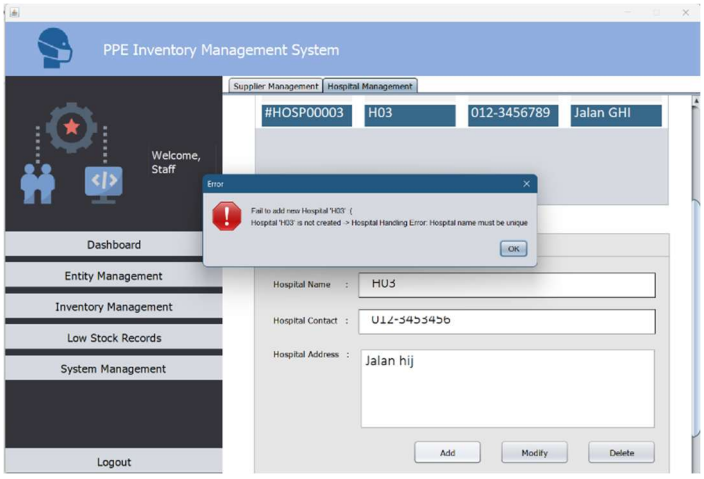


Diagram 3.63: Output of Error Message for Duplicate Hospital Name

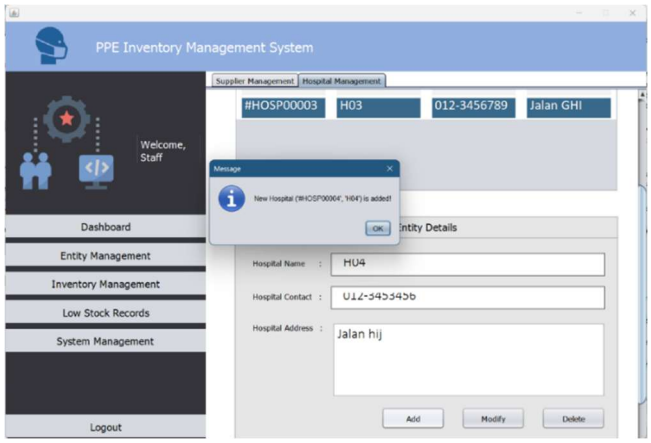


Diagram 3.64: Output of Error Message for Duplicate Hospital Name

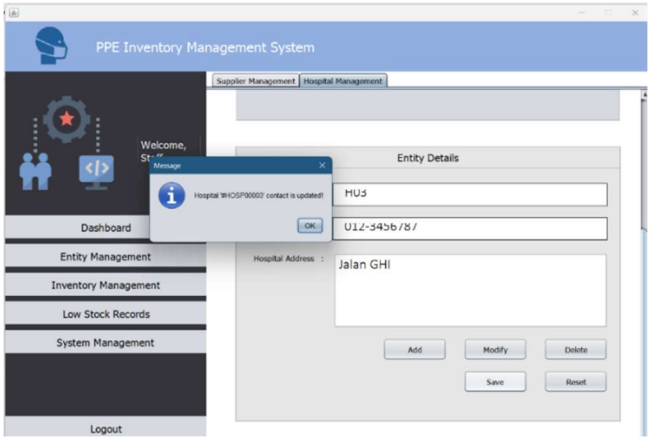


Diagram 3.65: Confirmation Message for Successfully Updating Hospital Contact

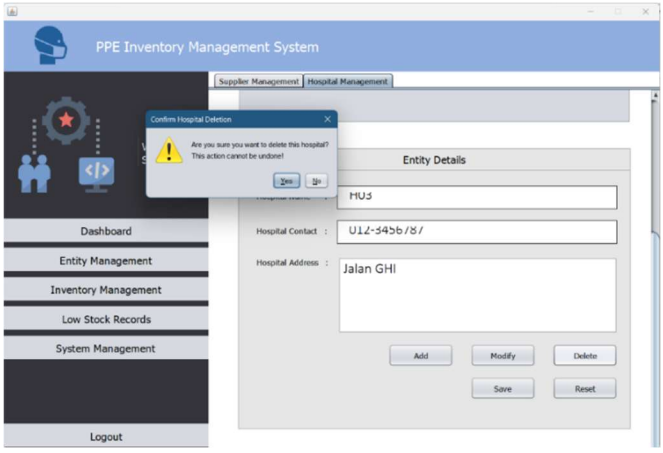


Diagram 3.66: Confirmation Message for Hospital Deletion

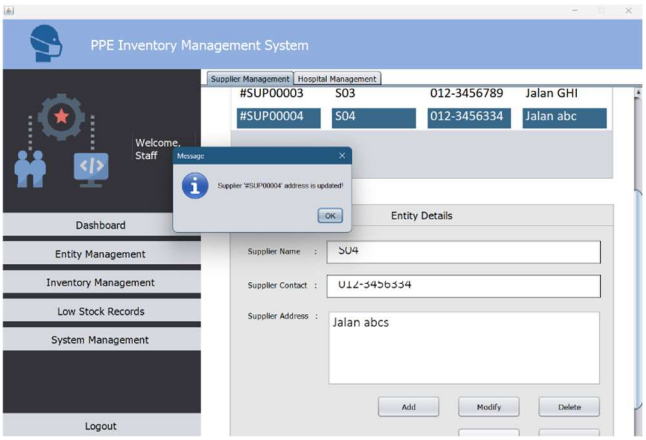


Diagram 3.67: Confirmation Message for Update Supplier

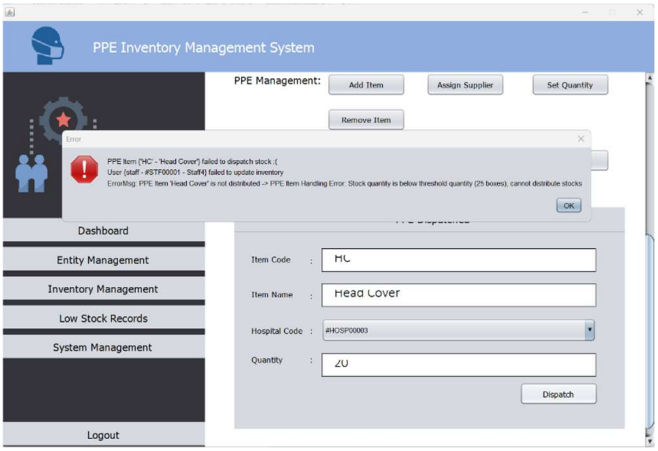


Diagram 3.68: Error Message Displayed When PPE Dispatch Fails Because Stock Is Below Threshold Quantity

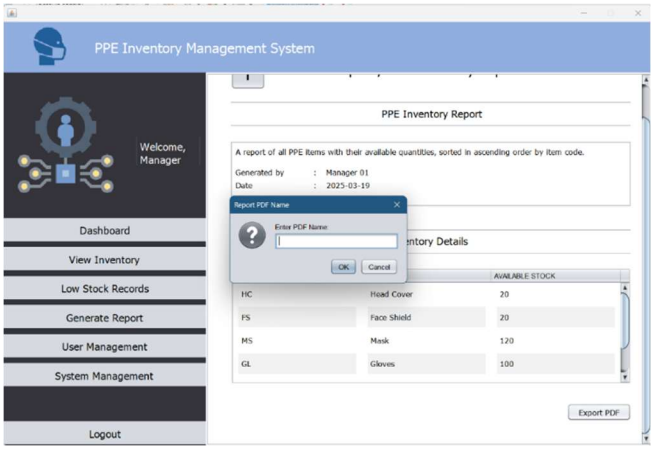


Diagram 3.69: Input Field to Enter PDF Name When Clicking the Export PDF Button

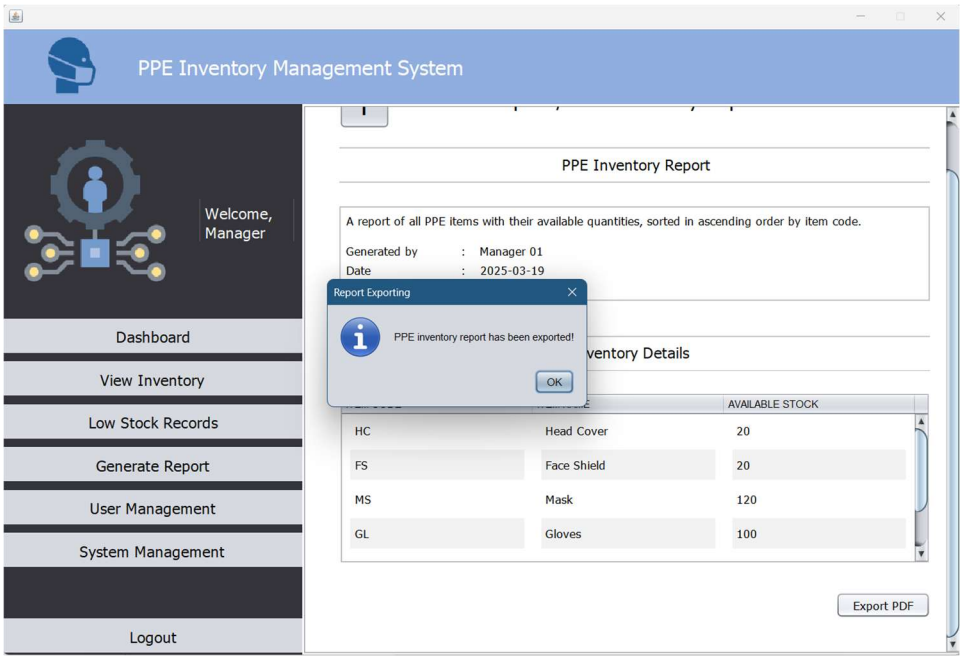


Diagram 3.70: Confirmation Message of Exporting Report

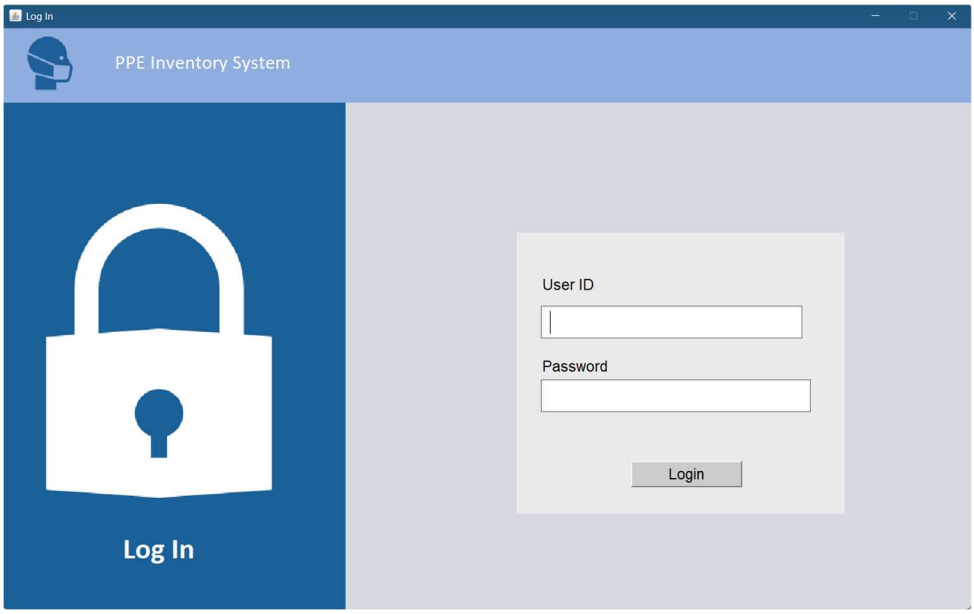


Diagram 3.71: Output When Clicking the Log Out Button

4.0 Conclusion

In conclusion, the PPE Inventory Management System highlights how OOP principles can be effectively applied in Java. With features such as user management, inventory tracking, and search capabilities, it simplifies the management of PPE items, suppliers, and hospitals. A user-friendly Graphical User Interface (GUI) was also designed to enhance navigation and improve the overall user experience. Data is stored in text files such as users.txt, ppe.txt and transactions.txt to facilitate efficient retrieval and maintain a lightweight system. The role-based access controls can also simplify the operations and clearly define user responsibilities.

At the same time, the team also gained a deeper understanding of OOP and practical experience in system design and implementation. Its scalable structure makes it easy to expand in the future by adding more hospitals, suppliers, PPE items, or new features. These improvements not only enhance the system's reliability and efficiency but also lay strong groundwork for continuous improvement and adaptation to evolving needs.

5.0 References

(2025). Youtu.be. https://youtu.be/L-UPOw1nHCI?si=W1qljiZoe4WKh1ui

(2025). Youtu.be. https://youtu.be/OPmYIjpuCFQ?si=5xzGW61nbsziBE6y

File exists() method in Java with examples. (2019, January 25). GeeksforGeeks. https://www.geeksforgeeks.org/file-exists-method-in-java-with-examples/

GeeksforGeeks. (2016, October 26). *Java LinkedHashMap*. GeeksforGeeks. http://geeksforgeeks.org/linkedhashmap-class-in-java/

GeeksforGeeks. (2017, November 14). *Abstraction in Java*. GeeksforGeeks. https://www.geeksforgeeks.org/abstraction-in-java-2/

iText Tutorial - Tutorialspoint. (n.d.). Www.tutorialspoint.com. https://www.tutorialspoint.com/itext/index.htm

Java Date and Time. (2019). W3schools.com. https://www.w3schools.com/java/java_date.asp

Java Inheritance (With Examples). (n.d.). Www.programiz.com. https://www.programiz.com/java-programming/inheritance

JCalendar. (n.d.). Toedter. https://toedter.com/jcalendar/

LocalDateTime (Java Platform SE 8). (n.d.). Docs.oracle.com. https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html

Pei, D. (2024, October 9). *Role-Based Access Control: Simplifying Access Security Management*. Linford & Company LLP. https://linfordco.com/blog/role-based-access-control-rbac/

w3schools. (2019). *Java Encapsulation and Getters and Setters*. W3schools.com. https://www.w3schools.com/java/java_encapsulation.asp

W3schools. (2019). *Java Polymorphism*. W3schools.com. https://www.w3schools.com/java/java_polymorphism.asp

W3schools. (n.d.). *Java Read Files*. Www.w3schools.com. https://www.w3schools.com/java/java_files_read.asp