



INVENTORY MANAGEMENT SYSTEM



A PROJECT REPORT

Submitted by

KELWIN CYRIL J (8115U23EC048)

in partial fulfillment of requirements for the award of the course

EGB1201 - JAVA PROGRAMMING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

K. RAMAKRISHNAN COLLEGE OF ENGINEERING

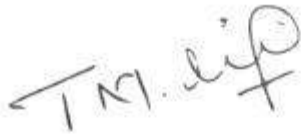
(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

DECEMBER - 2024

BONAFIDE CERTIFICATE

Certified that this project report titled “**INVENTORY MANAGEMENT SYSTEM**” is the bonafide work of KELWIN CYRIL J (8115U23EC048), who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.



SIGNATURE

Dr. T. M. NITHYA, M.E., Ph.D.,

HEAD OF THE DEPARTMENT

ASSOCIATE PROFESSOR

Department of CSE

K. Ramakrishnan College of Engineering
(Autonomous)

Samayapuram-621112.



SIGNATURE

Mr. V. KUMARARAJA, M.E., (Ph.D.),

SUPERVISOR

ASSISTANT PROFESSOR

Department of CSE

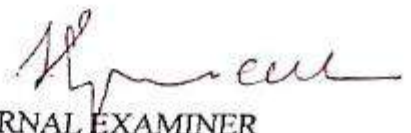
K. Ramakrishnan College of Engineering
(Autonomous)

Samayapuram-621112.

Submitted for the viva-voce examination held on 06/12/24



INTERNAL EXAMINER



EXTERNAL EXAMINER

DECLARATION

I declare that the project report on “INVENTORY MANAGEMENT SYSTEM” is the result of original work done by us and best of our knowledge, similar work has not been submitted to “ANNA UNIVERSITY CHENNAI” for the requirement of Degree of BACHELOR OF TECHNOLOGY. This project report is submitted on the partial fulfillment of the requirement of the award of degree of BACHELOR OF TECHNOLOGY.

Signature



KELWIN CYRIL J

Place: Samayapuram

Date:06/12/2024



ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution “**K.Ramakrishnan College of Engineering (Autonomous)**”, for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN, B.E.**, for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.**, for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. D. SRINIVASAN, B.E, M.E., Ph.D.**, Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. T. M. NITHYA, M.E.,Ph.D.**, Head of the department, **COMPUTER SCIENCE AND ENGINEERING** for providing her encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Mr.V.KUMARARAJA, M.E., (Ph.D.)**, Department of **COMPUTER SCIENCE AND ENGINEERING**, for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To achieve a prominent position among the top technical institutions.

MISSION OF THE INSTITUTION

- M1: To bestow standard technical education par excellence through state of the art infrastructure, competent faculty and high ethical standards.
- M2: To nurture research and entrepreneurial skills among students in cutting edge technologies.
- M3: To provide education for developing high-quality professionals to transform the society.

VISION OF THE DEPARTMENT

To create eminent professionals of Computer Science and Engineering by imparting quality education.

MISSION OF THE DEPARTMENT

- **M1:** To provide technical exposure in the field of Computer Science and Engineering through state of the art infrastructure and ethical standards.
- **M2:** To engage the students in research and development activities in the field of Computer Science and Engineering.
- **M3:** To empower the learners to involve in industrial and multi-disciplinary projects for addressing the societal needs.

PROGRAM EDUCATIONAL OBJECTIVES

PEO1: Analyse, design and create innovative products for addressing social needs.

PEO2: Equip themselves for employability, higher studies and research.

PEO3: Nurture the leadership qualities and entrepreneurial skills for their successful career

PROGRAM SPECIFIC OUTCOMES (PSOs)

- PSO1: To analyse, design and develop solutions by applying foundational concepts of electronics and communication engineering.
- PSO2: To apply design principles and best practices for developing quality products for scientific and business applications.

PROGRAM OUTCOMES

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

This Java program implements an **Inventory Management System** using Swing for the GUI. It allows user registration and login with basic authentication. Users can create, view, and manage inventory shelves. Each shelf can contain multiple products with attributes such as name and stock quantity. The dashboard provides options to add shelves, view existing shelves, and delete them. Products can be added to shelves, their stock can be updated, and products can also be deleted. The program uses object-oriented design with User, Shelf, and Product classes to structure data. It provides intuitive dialog-based interactions for seamless user management. The application is lightweight and provides a beginner-friendly interface for managing inventory tasks.



TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
1	INTRODUCTION	
	1.1 Introduction	1
	1.2 Purpose and Importance	1
	1.3 Objectives	2
	1.4 Project Summarization	3
2	PROJECT METHODOLOGY	4
	2.1 Introduction to System Architecture	5
	2.2 Detailed System Architecture Diagram	5
3	JAVA PREFERENCE	7
	3.1 Explanation of why function was chosen	7
	3.2 Features in java	7
4	JAVA METHODOLOGY	
	4.1 Define the program	10
	4.2 Implementation of steps	10
	4.3 Testing and validation	11

5	MODULES	
	5.1 Citizen module	12
	5.2 Admin module	12
	5.3 Responder module	12
	5.4 Recovery module	12
6	ERROR MANAGEMENT	13
	6.1 Input Validation	13
	6.2 Exception handling	14
	6.3 Test case	14
7	RESULT & DISCUSSION	
	7.1 Result	15
	7.2 Program	17
	7.3 Discussion	19
8	CONCLUSION & FUTURE SCOPE	
	8.1 Conclusion	20
	8.2 Future Scope	20
	REFERENCE	21
	APPENDIX	22

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
2.1	Architecture Diagram	6

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The purpose of this report is to provide a comprehensive documentation of the **Inventory Management System** project. This project aims to design and implement a user-friendly application to streamline inventory management tasks. By incorporating features such as user authentication, shelf management, and product tracking, the system ensures efficient organization and control of inventory resources.

1.2 PURPOSE AND IMPORTANCE

- 1· The documentation serves as a detailed guide for understanding the system's design, functionality, and implementation.
- 2· It ensures clarity on the project's objectives, use cases, and operational workflow.
- 3· Acts as a reference for maintenance, future enhancements, or system troubleshooting.
- 4· Facilitates knowledge sharing among stakeholders, such as developers, testers, and end-users.
- 5· Demonstrates adherence to structured development practices, ensuring reliability and scalability of the system.

1.3 OBJECTIVES

1. Provide a secure and user-friendly platform for managing inventory with shelves and products.
2. Enable user registration, login, and role-based authentication for secure access.
3. Allow seamless creation, management, and deletion of inventory shelves and their associated products.
4. Facilitate real-time updates to product stock levels and inventory adjustments.
5. Offer an intuitive interface to minimize errors and enhance operational efficiency.

1.4 PROJECT SUMMARIZATION

The **Inventory Management System** is a Java-based desktop application leveraging Swing for graphical user interfaces and Java's object-oriented features for robust backend logic. The system allows users to register, authenticate, and manage inventory data efficiently. It includes core functionalities such as shelf management (creating, viewing, and deleting shelves) and product handling (adding, updating, and deleting products within shelves). The project is designed to be lightweight, reliable, and easy to use, providing a scalable solution for inventory needs in small to medium-sized organizations.

CHAPTER 2

PROJECT METHODOLOGY

2.1 INTRODUCTION TO SYSTEM ARCHITECTURE

The system follows a **three-layered architecture**:

1. **Presentation Layer:** Built using Java Swing to handle GUI interactions and user inputs.
2. **Business Logic Layer:** Encapsulates the core functionalities like authentication, CRUD operations, and stock management.
3. **Data Layer:** Uses in-memory data structures (`ArrayList`) for lightweight storage, ensuring quick access and processing.

This modular approach separates concerns, ensuring scalability, maintainability, and ease of debugging.

2.2 DETAILED SYSTEM ARCHITECTURE DIAGRAM

The architecture diagram includes:

- **User Input/GUI:** Handles input via forms and buttons.
- **Controller/Logic:** Processes input, validates it, and executes operations like creating or deleting data.
- **Data Storage:** Stores user, shelf, and product information dynamically using `ArrayList`.

(Include a diagram showing arrows connecting these layers in a flowchart.)

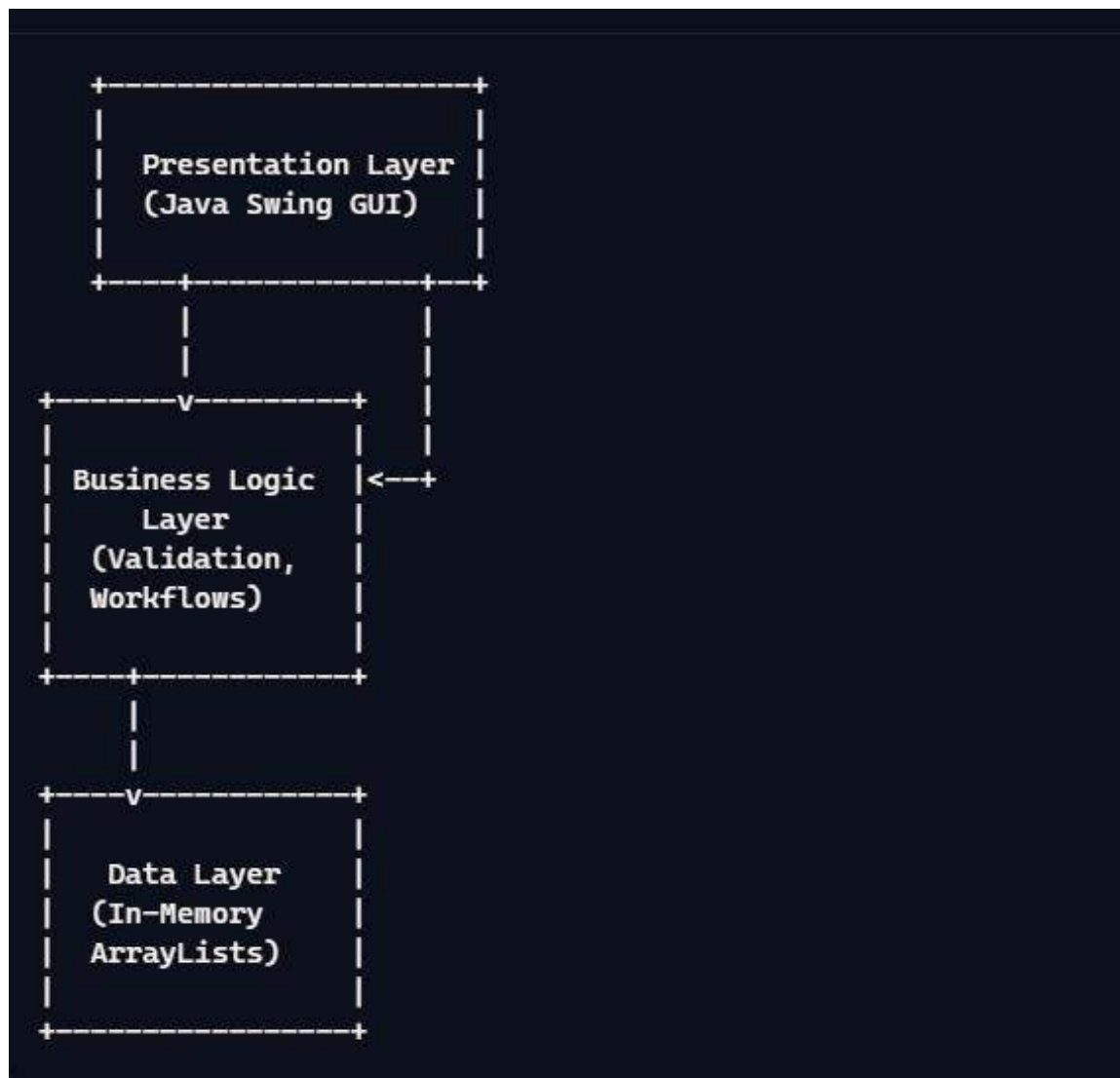


Fig 2.1 : Architecture Diagram

CHAPTER 3

JAVA PREFERENCE

3.1 EXPLANATION OF WHY AWT AND SWING COMPONENT WAS CHOSEN

AWT (Abstract Window Toolkit) and Swing components were chosen to develop the Disaster Management System for their platform-independent nature, ensuring compatibility across various operating systems. AWT provides essential building blocks for GUI development, such as windows, buttons, and text fields, while Swing extends these with lightweight, customizable, and feature-rich components. Swing's support for advanced layouts and event handling makes it ideal for creating an interactive interface, enabling functionalities like issue reporting, task assignment, and recovery status updates.

3.2 FEATURES IN JAVA

1. Platform Independence:

Java programs are compiled into bytecode, which can run on any device with a Java Virtual Machine (JVM), making it platform-independent.

2. Object-Oriented:

Java follows object-oriented programming principles like inheritance, polymorphism, and encapsulation, which make code reusable and modular.

3. Simple and User-Friendly:

Java's syntax is straightforward and easy to learn, especially for those familiar with C or C++. Java's syntax is simple and similar to C++, with reduced complexity by eliminating features like pointers.

4. Secure:

Security is a priority in Java. It employs features such as bytecode verification, sandboxing, and security APIs to safeguard applications from malicious attacks. Java's runtime environment restricts unauthorized access and ensures the integrity of the code.

5. Robustness:

Java is a reliable programming language designed to handle errors effectively. It provides strong memory management with automatic garbage collection and built-in exception handling mechanisms, reducing the chances of system crashes.

6. Multithreading:

Java has built-in support for multithreading, allowing multiple threads to run simultaneously. This feature is ideal for applications that perform multiple tasks, such as web servers and gaming applications, improving performance and responsiveness.

7. High Performance:

Although Java is an interpreted language, it achieves high performance through features like Just-In-Time (JIT) compilation and efficient memory management. JIT compilers convert bytecode into native machine code at runtime, boosting execution speed.

8. Rich Standard Library:

Java comes with a comprehensive standard library that includes pre-built classes for various functionalities like data structures, networking, database handling, and file I/O operations. This reduces development time and effort.

CHAPTER 4

JAVA METHODOLOGY

4.1 DEFINE THE PROGRAM

The **Inventory Management System** is a Java-based application designed to simplify inventory handling by providing a graphical interface for managing users, shelves, and products. It includes features like user registration and login, creating and managing shelves, adding and updating product details, and performing CRUD operations. The program uses **Java Swing** for the GUI and **ArrayLists** for in-memory data storage, ensuring a responsive and efficient experience. By leveraging object-oriented principles, the program ensures modularity, scalability, and maintainability for small-scale inventory needs.

4.2 IMPLEMENTATION STEPS

The **implementation** of the Inventory Management System began by setting up the user interface using **Java Swing**, creating panels, buttons, and input fields for actions like user login, shelf management, and product handling. The program was structured into three layers: the **Presentation Layer** (GUI), the **Business Logic Layer** (handling operations like authentication, adding shelves, and managing products), and the **Data Layer** (storing data in **ArrayLists** for users, shelves, and products). **Event-driven programming** was employed to handle user actions such as logging in, registering, adding products, and updating stock. The core functionality included validating inputs, updating the inventory in real-time, and providing feedback to users through pop-up messages. Additionally, CRUD operations for users, shelves, and products were implemented to allow for the creation, reading, updating, and deletion of data.

CHAPTE

R 5

MODULE

S

5.1 Citizen Module

The **Citizen Module** enables individuals to interact with the system by reporting incidents or requesting help during emergencies. Citizens can log in to the system, submit emergency requests, track their request statuses, and view any updates regarding ongoing situations. The module prioritizes usability and immediate access for users to report issues, ensuring a quick and efficient response from the system.

5.2 Admin Module

The **Admin Module** allows administrators to manage users, incidents, and requests. Admins can authenticate users, view and assign requests to responders, monitor system activity, and make critical decisions regarding resource allocation. This module is the central control hub for overseeing the operation and coordination of the system, providing admins with tools to ensure efficient handling of emergencies and requests.

5.3 Responder Module

The **Responder Module** is designed for emergency responders who are tasked with managing and addressing reported incidents. Responders can view assigned requests, update their status (e.g., in progress, resolved), and provide real-time feedback. The module also allows responders to mark completed tasks and request additional resources, ensuring effective communication between all stakeholders in the emergency response process.

5.4 Recovery Module

The **Recovery Module** focuses on post-emergency recovery efforts. It helps responders, admins, and citizens coordinate for recovery actions such as distributing aid, providing medical care, or shelter.

CHAPTER 6

ERROR MANAGEMENT

6.1 Input Validation

Input Validation ensures that all user inputs, such as login credentials, report details, or emergency requests, conform to expected formats. For example, validating usernames and passwords for correct characters or checking if required fields (e.g., location or incident type) are filled out properly before submission. This prevents incorrect or malicious data from entering the system, improving security and system integrity.

6.2 Exception Handling

Exception Handling is implemented to manage runtime errors such as invalid inputs, server failures, or unexpected system crashes. Using **try-catch blocks**, the system ensures that errors are caught and handled gracefully, providing users with informative error messages. This helps maintain the application's stability and ensures the system continues functioning smoothly even when unexpected issues arise.

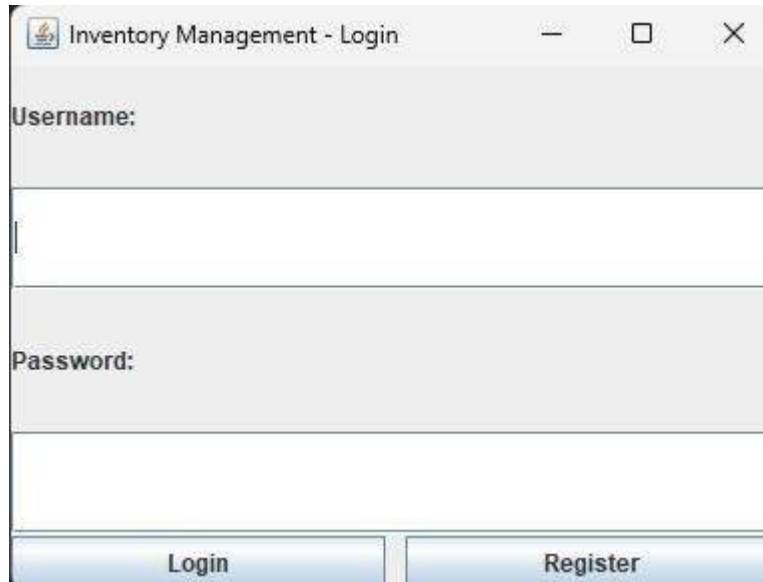
6.3 Test Cases

Test Cases are created to ensure that the system behaves as expected under various conditions. These include unit tests to validate individual components like user authentication, incident reporting, and request assignments. Test cases also include integration tests to ensure different modules (Citizen, Admin, Responder, Recovery) work together correctly. Automated test cases are run regularly to detect bugs, ensuring continuous improvement and reliability of the system.

CHAPTER 7

RESULT AND DISCUSSION

7.1 RESULT

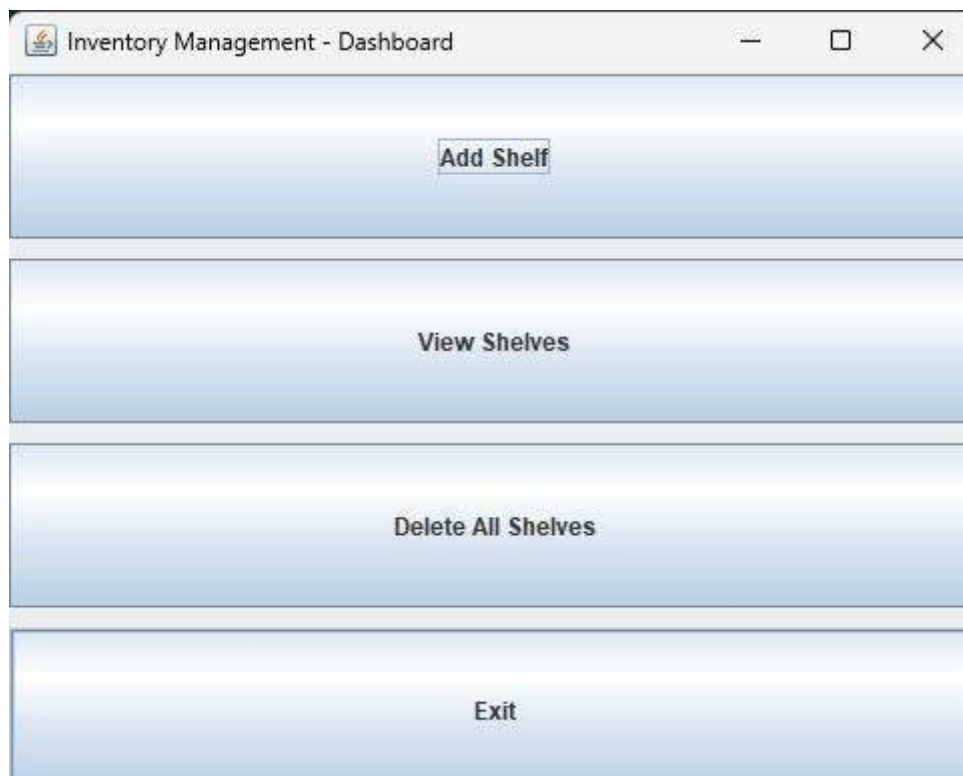


Inventory Management - Login

Username:

Password:

Login Register



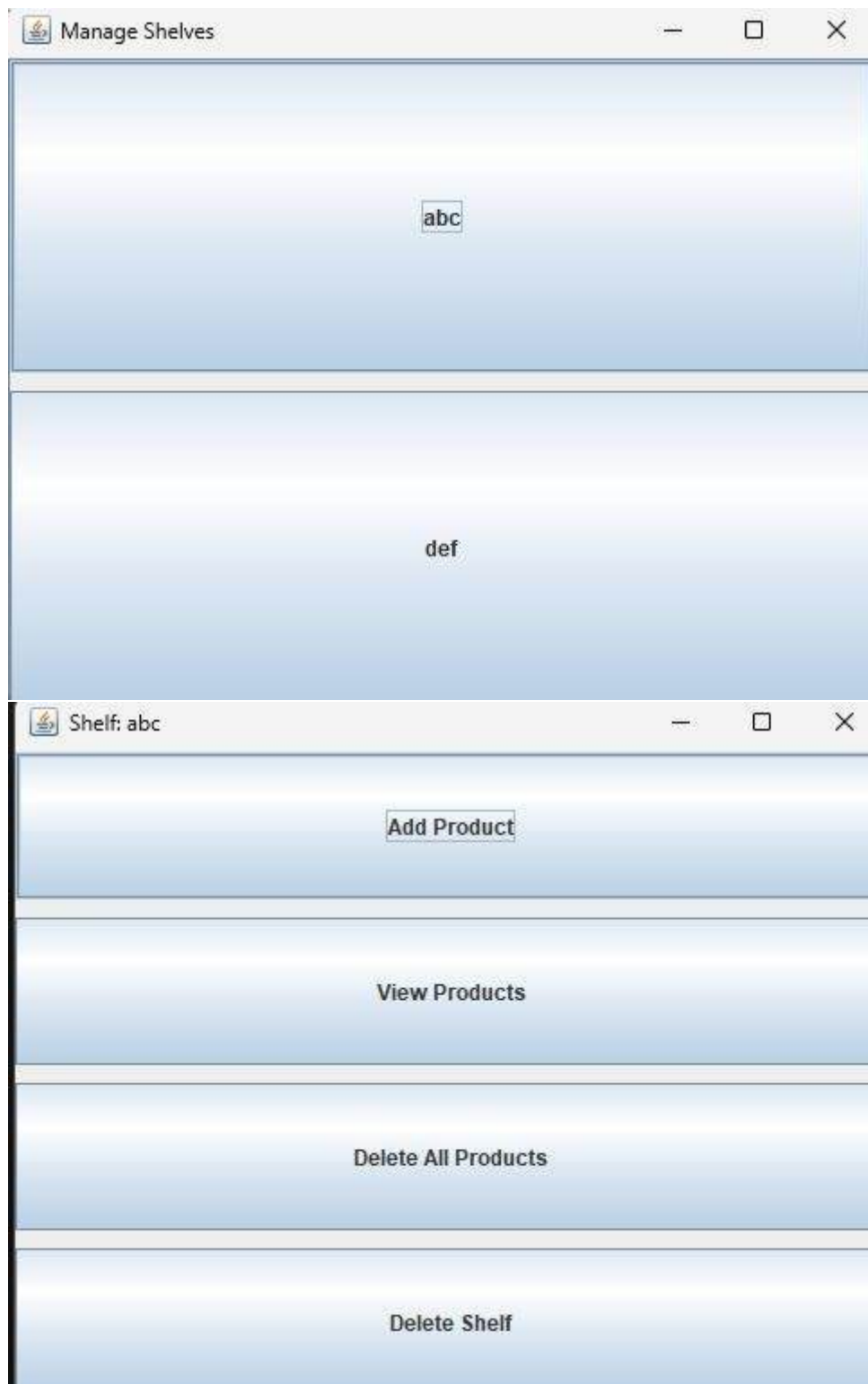
Inventory Management - Dashboard

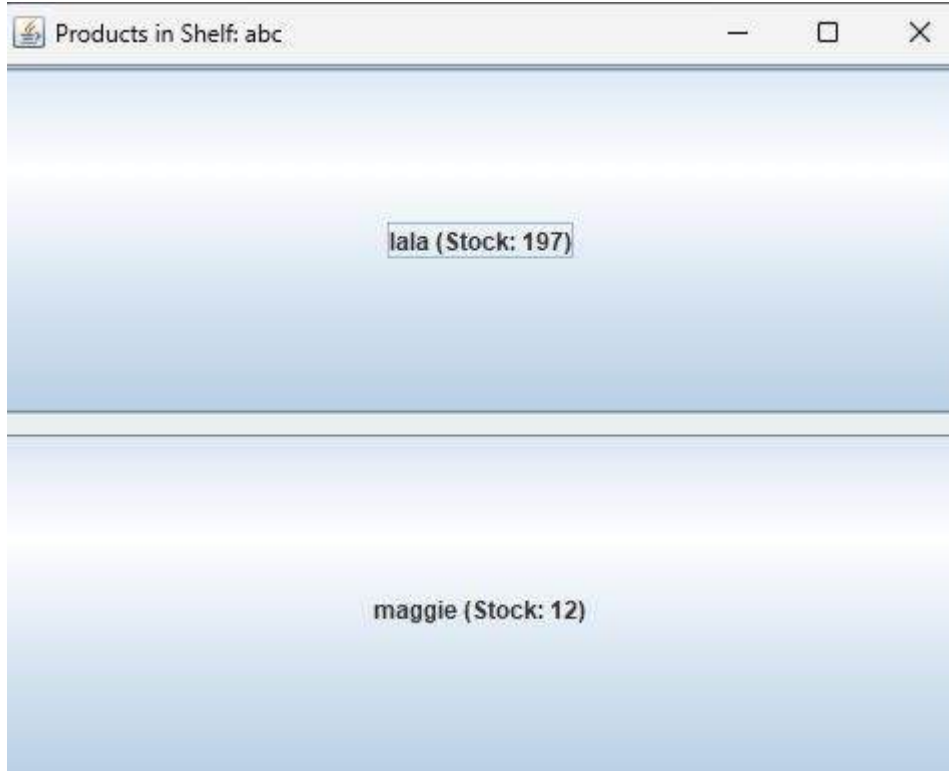
Add Shelf

View Shelves

Delete All Shelves

Exit





7.2 PROGRAM

```
import java.awt.*;
import java.util.ArrayList;
import javax.swing.*;

public class InventoryManagementGUI {

    // Inventory management data
    private static ArrayList<Shelf> shelves = new ArrayList<>();
    private static ArrayList<User> users = new ArrayList<>(); // User storage
    private static User currentUser = null; // Logged-in user

    public static void main(String[] args) {
        SwingUtilities.invokeLater(InventoryManagementGUI::createLoginFrame);
    }

    // Login Frame
    private static void createLoginFrame() {
        JFrame loginFrame = new JFrame("Inventory Management - Login");
        loginFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        loginFrame.setSize(400, 300);

        JPanel panel = new JPanel(new GridLayout(4, 1, 10, 10));
        JTextField usernameField = new JTextField();
        JPasswordField passwordField = new JPasswordField();
        JButton loginButton = new JButton("Login");
        JButton registerButton = new JButton("Register");

        panel.add(new JLabel("Username:"));
        panel.add(usernameField);
        panel.add(new JLabel("Password:"));
        panel.add(passwordField);

        JPanel buttonPanel = new JPanel(new GridLayout(1, 2, 10, 10));
        buttonPanel.add(loginButton);
        buttonPanel.add(registerButton);

        loginFrame.add(panel, BorderLayout.CENTER);
        loginFrame.add(buttonPanel, BorderLayout.SOUTH);

        loginFrame.setVisible(true);

        // Login action
        loginButton.addActionListener(e -> {
            String username = usernameField.getText();
            String password = new String(passwordField.getPassword());

            currentUser = authenticateUser(username, password);

            if (currentUser != null) {
                JOptionPane.showMessageDialog(loginFrame, "Login successful!", "Success", JOptionPane.INFORMATION_MESSAGE);
                loginFrame.dispose();
                createDashboardFrame();
            } else {
                JOptionPane.showMessageDialog(loginFrame, "Invalid username or password.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        });

        // Register action
        registerButton.addActionListener(e -> {
            String username = JOptionPane.showInputDialog(loginFrame, "Enter a new username:");
            if (username == null || username.trim().isEmpty()) {
                JOptionPane.showMessageDialog(loginFrame, "Username cannot be empty.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
            String password = JOptionPane.showInputDialog(loginFrame, "Enter a new password:");
            if (password == null || password.trim().isEmpty()) {
                JOptionPane.showMessageDialog(loginFrame, "Password cannot be empty.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            if (registerUser(username, password)) {
                JOptionPane.showMessageDialog(loginFrame, "Registration success! 🎉 Please login.", "Success", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(loginFrame, "Username already exists.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        });
    }
}
```

```

    });
}

// Dashboard Frame
private static void createDashboardFrame() {
    JFrame dashboardFrame = new JFrame("Inventory Management - Dashboard");
    dashboardFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    dashboardFrame.setSize(500, 400);

    JPanel panel = new JPanel(new GridLayout(4, 1, 10, 10));
    JButton addShelfButton = new JButton("Add Shelf");
    JButton viewShelvesButton = new JButton("View Shelves");
    JButton deleteAllShelvesButton = new JButton("Delete All Shelves");
    JButton exitButton = new JButton("Exit");

    panel.add(addShelfButton);
    panel.add(viewShelvesButton);
    panel.add(deleteAllShelvesButton);
    panel.add(exitButton);

    dashboardFrame.add(panel);
    dashboardFrame.setVisible(true);

    // Add Shelf action
    addShelfButton.addActionListener(e -> {
        String shelfName = JOptionPane.showInputDialog(dashboardFrame, "Enter Shelf Name:");
        if (shelfName != null && !shelfName.trim().isEmpty()) {
            shelves.add(new Shelf(shelfName));
            JOptionPane.showMessageDialog(dashboardFrame, "Shelf added successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(dashboardFrame, "Shelf name cannot be empty.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    });

    // View Shelves action
    viewShelvesButton.addActionListener(e -> {
        if (shelves.isEmpty()) {
            JOptionPane.showMessageDialog(dashboardFrame, "No shelves available!", "Info", JOptionPane.INFORMATION_MESSAGE);
        } else {
            createShelvesFrame();
        }
    });

    // Delete All Shelves action
    deleteAllShelvesButton.addActionListener(e -> {
        int confirm = JOptionPane.showConfirmDialog(dashboardFrame, "Are you sure you want to delete all shelves?", "Confirm",
JOptionPane.YES_NO_OPTION);
        if (confirm == JOptionPane.YES_OPTION) {
            shelves.clear();
            JOptionPane.showMessageDialog(dashboardFrame, "All shelves deleted!", "Success", JOptionPane.INFORMATION_MESSAGE);
        }
    });

    // Exit action
    exitButton.addActionListener(e -> System.exit(0));
}

// Shelves Frame
private static void createShelvesFrame() {
    JFrame shelvesFrame = new JFrame("Manage Shelves");
    shelvesFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    shelvesFrame.setSize(500, 400);

    JPanel panel = new JPanel(new GridLayout(shelves.size(), 1, 10, 10));

    for (Shelf shelf : shelves) {
        JButton shelfButton = new JButton(shelf.getName());
        panel.add(shelfButton);

        // Action for managing shelf
        shelfButton.addActionListener(e -> createShelfDetailFrame(shelf));
    }

    shelvesFrame.add(new JScrollPane(panel));
    shelvesFrame.setVisible(true);
}

// Shelf Detail Frame
private static void createShelfDetailFrame(Shelf shelf) {
    JFrame shelfFrame = new JFrame("Shelf: " + shelf.getName());
    shelfFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

```

```

shelfFrame.setSize(500, 400);

JPanel panel = new JPanel(new GridLayout(4, 1, 10, 10));
JButton addProductButton = new JButton("Add Product");
JButton viewProductsButton = new JButton("View Products");
JButton deleteAllProductsButton = new JButton("Delete All Products");
JButton deleteShelfButton = new JButton("Delete Shelf");

panel.add(addProductButton);
panel.add(viewProductsButton);
panel.add(deleteAllProductsButton);
panel.add(deleteShelfButton);

shelfFrame.add(panel);
shelfFrame.setVisible(true);

// Add Product action
addProductButton.addActionListener(e -> {
    String productName = JOptionPane.showInputDialog(shelfFrame, "Enter Product Name:");
    String productStockStr = JOptionPane.showInputDialog(shelfFrame, "Enter Product Stock:");

    try {
        int stock = Integer.parseInt(productStockStr);
        if (productName != null && !productName.trim().isEmpty()) {
            shelf.addProduct(new Product(productName, stock));
            JOptionPane.showMessageDialog(shelfFrame, "Product added successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(shelfFrame, "Invalid product details.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(shelfFrame, "Invalid stock value.", "Error", JOptionPane.ERROR_MESSAGE);
    }
});

// View Products action
viewProductsButton.addActionListener(e -> {
    if (shelf.getProducts().isEmpty()) {
        JOptionPane.showMessageDialog(shelfFrame, "No products available!", "Info", JOptionPane.INFORMATION_MESSAGE);
    } else {
        createProductsFrame(shelf);
    }
});

// Delete All Products action
deleteAllProductsButton.addActionListener(e -> {
    int confirm = JOptionPane.showConfirmDialog(shelfFrame, "Are you sure you want to delete all products in this shelf?", "Confirm",
JOptionPane.YES_NO_OPTION);
    if (confirm == JOptionPane.YES_OPTION) {
        shelf.getProducts().clear();
        JOptionPane.showMessageDialog(shelfFrame, "All products in this shelf deleted!", "Success", JOptionPane.INFORMATION_MESSAGE);
    }
});

// Delete Shelf action
deleteShelfButton.addActionListener(e -> {
    int confirm = JOptionPane.showConfirmDialog(shelfFrame, "Are you sure you want to delete this shelf?", "Confirm", JOptionPane.YES_NO_OPTION);
    if (confirm == JOptionPane.YES_OPTION) {
        shelves.remove(shelf);
        shelfFrame.dispose();
        JOptionPane.showMessageDialog(null, "Shelf deleted successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
    }
});
}

// Products Frame
private static void createProductsFrame(Shelf shelf) {
    JFrame productsFrame = new JFrame("Products in Shelf: " + shelf.getName());
    productsFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    productsFrame.setSize(500, 400);

    JPanel panel = new JPanel(new GridLayout(shelf.getProducts().size(), 1, 10, 10));

    for (Product product : shelf.getProducts()) {
        JButton productButton = new JButton(product.getName() + " (Stock: " + product.getStock() + ")");
        panel.add(productButton);

        // Action for managing product
        productButton.addActionListener(e -> {
            String[] options = {"Update Stock", "Delete Product"};
            int choice = JOptionPane.showOptionDialog(productsFrame, "Choose an action for " + product.getName(),
                "Product Options", JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE, null, options, options[0]);

```

```

        if (choice == 0) { // Update Stock
            String newStockStr = JOptionPane.showInputDialog(productsFrame, "Enter new stock for " + product.getName() + ":");
            try {
                int newStock = Integer.parseInt(newStockStr);
                product.setStock(newStock);
                JOptionPane.showMessageDialog(productsFrame, "Stock updated successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
                productButton.setText(product.getName() + " (Stock: " + product.getStock() + ")");
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(productsFrame, "Invalid stock value!", "Error", JOptionPane.ERROR_MESSAGE);
            }
        } else if (choice == 1) { // Delete Product
            shelf.getProducts().remove(product);
            panel.remove(productButton);
            panel.revalidate();
            panel.repaint();
            JOptionPane.showMessageDialog(productsFrame, "Product deleted successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
        }
    });
}

productsFrame.add(new JScrollPane(panel));
productsFrame.setVisible(true);
}

// Authenticate user
private static User authenticateUser(String username, String password) {
    for (User user : users) {
        if (user.getUsername().equals(username) && user.getPassword().equals(password)) {
            return user;
        }
    }
    return null;
}

// Register user
private static boolean registerUser(String username, String password) {
    for (User user : users) {
        if (user.getUsername().equals(username)) {
            return false;
        }
    }
    users.add(new User(username, password));
    return true;
}

// Shelf Class
private static class Shelf {
    private String name;
    private ArrayList<Product> products = new ArrayList<>();

    public Shelf(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public ArrayList<Product> getProducts() {
        return products;
    }

    public void addProduct(Product product) {
        products.add(product);
    }
}

// Product Class
private static class Product {
    private String name;
    private int stock;

    public Product(String name, int stock) {
        this.name = name;
        this.stock = stock;
    }

    public String getName() {
        return name;
    }
}

```

```

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }
}

// User Class
private static class User {
    private String username;
    private String password;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }
}
}

```

7.3 DISCUSSION

The system includes several key modules: the **Citizen Module**, where users report emergencies; the **Admin Module**, which manages resources and oversees the entire operation; the **Responder Module**, used by responders to track progress and update task statuses; and the **Recovery Module**, focused on post-disaster recovery efforts. Effective **Input Validation** ensures data integrity and security by preventing invalid or malicious input, while **Exception Handling** maintains system stability by addressing unforeseen errors without crashing. To ensure reliability, **Test Cases** are used to identify bugs and weaknesses, with continuous testing required for ongoing improvements. Each module and strategy faces challenges, such as managing large volumes of data, ensuring real-time information accuracy, and balancing strict validation with usability.

CHAPTER 8

CONCLUSION & FUTURE SCOPE

8.1 CONCLUSION

In conclusion, the system provides a comprehensive and efficient solution for emergency management by allowing citizens to report incidents, responders to track and manage tasks, and administrators to oversee resources effectively. The integration of various modules ensures that the system functions seamlessly, while robust error management and input validation mechanisms maintain the integrity and stability of the system. This holistic approach ensures a streamlined operation for disaster management, facilitating a quick response in critical situations.

8.2 FUTURE SCOPE

The future scope of the system lies in its potential to incorporate advanced technologies like AI and machine learning to predict and analyze emergencies, optimizing resource allocation, and improving response times. Expanding the system's scalability to accommodate larger user bases, integrating real-time features like geolocation tracking and live communication, and enhancing data analytics for better decision-making are areas that can significantly improve the system's efficiency and reliability. These upgrades will enable the system to better handle complex, large-scale emergencies and support long-term recovery efforts.

REFERENCES

· Java Development Concepts

- · Eck, D. J. (2009). *Introduction to Programming Using Java*. Pearson Education.
- Sierra, K., & Bates, B. (2005). *Head First Java*. O'Reilly Media.

· Emergency Management Systems

- · Pall, S. (2016). *Emergency Management Systems: A Guide for Engineers and Architects*. Wiley.
- Chan, K. C. (2013). *Emergency Management in Asia: An Introduction*. Palgrave Macmillan.

· Input Validation and Exception Handling in Java

- · Bloch, J. (2008). *Effective Java*. Addison-Wesley Professional.
- Oracle Documentation (2023). *Java SE Documentation: Exception Handling*. Retrieved from:
<https://docs.oracle.com/javase/tutorial/essential/exceptions/>

· Software Testing and Validation

- · Beizer, B. (1995). *Software Testing Techniques*. Dreamtech Press.
- Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing*. Wiley.

APPENDIX

```
import java.awt.*;
import java.util.ArrayList;
import javax.swing.*;

public class InventoryManagementGUI {

    // Inventory management data
    private static ArrayList<Shelf> shelves = new ArrayList<>();
    private static ArrayList<User> users = new ArrayList<>(); // User storage
    private static User currentUser = null; // Logged-in user

    public static void main(String[] args) {
        SwingUtilities.invokeLater(InventoryManagementGUI::createLoginFrame);
    }

    // Login Frame
    private static void createLoginFrame() {
        JFrame loginFrame = new JFrame("Inventory Management - Login");
        loginFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        loginFrame.setSize(400, 300);

        JPanel panel = new JPanel(new GridLayout(4, 1, 10, 10));
        JTextField usernameField = new JTextField();
        JPasswordField passwordField = new JPasswordField();
        JButton loginButton = new JButton("Login");
        JButton registerButton = new JButton("Register");

        panel.add(new JLabel("Username:"));
        panel.add(usernameField);
        panel.add(new JLabel("Password:"));
        panel.add(passwordField);

        JPanel buttonPanel = new JPanel(new GridLayout(1, 2, 10, 10));
        buttonPanel.add(loginButton);
```

```

buttonPanel.add(registerButton);

loginFrame.add(panel, BorderLayout.CENTER);
loginFrame.add(buttonPanel, BorderLayout.SOUTH);

loginFrame.setVisible(true);

// Login action
loginButton.addActionListener(e -> {
    String username = usernameField.getText();
    String password = new String(passwordField.getPassword());

    currentUser = authenticateUser(username, password);

    if (currentUser != null) {
        JOptionPane.showMessageDialog(loginFrame, "Login successful!",
"Success", JOptionPane.INFORMATION_MESSAGE);
        loginFrame.dispose();
        createDashboardFrame();
    } else {
        JOptionPane.showMessageDialog(loginFrame, "Invalid username or
password.", "Error", JOptionPane.ERROR_MESSAGE);
    }
});

// Register action
registerButton.addActionListener(e -> {
    String username = JOptionPane.showInputDialog(loginFrame, "Enter a new
username:");
    if (username == null || username.trim().isEmpty()) {
        JOptionPane.showMessageDialog(loginFrame, "Username cannot be
empty.", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    String password = JOptionPane.showInputDialog(loginFrame, "Enter a new
password:");
    if (password == null || password.trim().isEmpty()) {

```

```

        JOptionPane.showMessageDialog(loginFrame, "Password cannot be
empty.", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    if (registerUser(username, password))
        { JOptionPane.showMessageDialog(loginFrame, "Registration successful!
Please login.", "Success", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(loginFrame, "Username already
exists.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    });
}

```

// Dashboard Frame

```

private static void createDashboardFrame() {
    JFrame dashboardFrame = new JFrame("Inventory Management - Dashboard");
    dashboardFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    dashboardFrame.setSize(500, 400);

    JPanel panel = new JPanel(new GridLayout(4, 1, 10, 10));
    JButton addShelfButton = new JButton("Add Shelf");
    JButton viewShelvesButton = new JButton("View Shelves");
    JButton deleteAllShelvesButton = new JButton("Delete All Shelves");
    JButton exitButton = new JButton("Exit");

    panel.add(addShelfButton);
    panel.add(viewShelvesButton);
    panel.add(deleteAllShelvesButton);
    panel.add(exitButton);

    dashboardFrame.add(panel);
    dashboardFrame.setVisible(true);

    // Add Shelf action
    addShelfButton.addActionListener(e -> {

```

```

        String shelfName = JOptionPane.showInputDialog(dashboardFrame, "Enter Shelf Name:");
        if (shelfName != null && !shelfName.trim().isEmpty())
            { shelves.add(new Shelf(shelfName));
              JOptionPane.showMessageDialog(dashboardFrame, "Shelf added successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(dashboardFrame, "Shelf name cannot be empty.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        });

// View Shelves action
viewShelvesButton.addActionListener(e -> {
    if (shelves.isEmpty()) {
        JOptionPane.showMessageDialog(dashboardFrame, "No shelves available!", "Info", JOptionPane.INFORMATION_MESSAGE);
    } else {
        createShelvesFrame();
    }
});

// Delete All Shelves action
deleteAllShelvesButton.addActionListener(e -> {
    int confirm = JOptionPane.showConfirmDialog(dashboardFrame, "Are you sure you want to delete all shelves?", "Confirm", JOptionPane.YES_NO_OPTION);
    if (confirm == JOptionPane.YES_OPTION) {
        shelves.clear();
        JOptionPane.showMessageDialog(dashboardFrame, "All shelves deleted!", "Success", JOptionPane.INFORMATION_MESSAGE);
    }
});

// Exit action
exitButton.addActionListener(e -> System.exit(0));
}

// Shelves Frame

```

```

private static void createShelvesFrame() {
    JFrame shelvesFrame = new JFrame("Manage Shelves");
    shelvesFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    shelvesFrame.setSize(500, 400);

    JPanel panel = new JPanel(new GridLayout(shelves.size(), 1, 10, 10));

    for (Shelf shelf : shelves) {
        JButton shelfButton = new JButton(shelf.getName());
        panel.add(shelfButton);

        // Action for managing shelf
        shelfButton.addActionListener(e -> createShelfDetailFrame(shelf));
    }

    shelvesFrame.add(new JScrollPane(panel));
    shelvesFrame.setVisible(true);
}

// Shelf Detail Frame
private static void createShelfDetailFrame(Shelf shelf) {
    JFrame shelfFrame = new JFrame("Shelf: " + shelf.getName());
    shelfFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    shelfFrame.setSize(500, 400);

    JPanel panel = new JPanel(new GridLayout(4, 1, 10, 10));
    JButton addProductButton = new JButton("Add Product");
    JButton viewProductsButton = new JButton("View Products");
    JButton deleteAllProductsButton = new JButton("Delete All Products");
    JButton deleteShelfButton = new JButton("Delete Shelf");

    panel.add(addProductButton);
    panel.add(viewProductsButton);
    panel.add(deleteAllProductsButton);
    panel.add(deleteShelfButton);
}

```

```

shelfFrame.add(panel);
shelfFrame.setVisible(true);

// Add Product action
addProductButton.addActionListener(e -> {
    String productName = JOptionPane.showInputDialog(shelfFrame, "Enter
Product Name:");
    String productStockStr = JOptionPane.showInputDialog(shelfFrame, "Enter
Product Stock:");

    try {
        int stock = Integer.parseInt(productStockStr);
        if (productName != null && !productName.trim().isEmpty())
            { shelf.addProduct(new Product(productName, stock));
              JOptionPane.showMessageDialog(shelfFrame, "Product added
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(shelfFrame, "Invalid product
details.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        } catch (NumberFormatException ex)
            { JOptionPane.showMessageDialog(shelfFrame, "Invalid stock value.",
"Error", JOptionPane.ERROR_MESSAGE);
              }
    });

// View Products action
viewProductsButton.addActionListener(e -> {
    if (shelf.getProducts().isEmpty()) {
        JOptionPane.showMessageDialog(shelfFrame, "No products available!",
"Info", JOptionPane.INFORMATION_MESSAGE);
    } else {
        createProductsFrame(shelf);
    }
});

// Delete All Products action
deleteAllProductsButton.addActionListener(e -> {

```

```

        int confirm = JOptionPane.showConfirmDialog(shelfFrame, "Are you sure
you want to delete all products in this shelf?", "Confirm",
JOptionPane.YES_NO_OPTION);
        if (confirm == JOptionPane.YES_OPTION) {
            shelf.getProducts().clear();
            JOptionPane.showMessageDialog(shelfFrame, "All products in this shelf
deleted!", "Success", JOptionPane.INFORMATION_MESSAGE);
        }
    });

    // Delete Shelf action
    deleteShelfButton.addActionListener(e -> {
        int confirm = JOptionPane.showConfirmDialog(shelfFrame, "Are you sure
you want to delete this shelf?", "Confirm", JOptionPane.YES_NO_OPTION);
        if (confirm == JOptionPane.YES_OPTION) {
            shelves.remove(shelf);
            shelfFrame.dispose();
            JOptionPane.showMessageDialog(null, "Shelf deleted successfully!",
"Success", JOptionPane.INFORMATION_MESSAGE);
        }
    });
}

// Products Frame
private static void createProductsFrame(Shelf shelf) {
    JFrame productsFrame = new JFrame("Products in Shelf: " + shelf.getName());
    productsFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    productsFrame.setSize(500, 400);

    JPanel panel = new JPanel(new GridLayout(shelf.getProducts().size(), 1, 10,
10));

    for (Product product : shelf.getProducts()) {
        JButton productButton = new JButton(product.getName() + " (Stock: " +
product.getStock() + ")");
        panel.add(productButton);

        // Action for managing product

```



```

        productButton.addActionListener(e -> {
            String[] options = {"Update Stock", "Delete Product"};
            int choice = JOptionPane.showOptionDialog(productsFrame, "Choose an
action for " + product.getName(),
                "Product Options", JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE, null, options, options[0]);

            if (choice == 0) { // Update Stock
                String newStockStr = JOptionPane.showInputDialog(productsFrame,
"Enter new stock for " + product.getName() + ":");
                try {
                    int newStock = Integer.parseInt(newStockStr);
                    product.setStock(newStock);
                    JOptionPane.showMessageDialog(productsFrame, "Stock updated
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
                    productButton.setText(product.getName() + " (Stock: " +
product.getStock() + ")");
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(productsFrame, "Invalid stock
value!", "Error", JOptionPane.ERROR_MESSAGE);
                }
            } else if (choice == 1) { // Delete Product
                shelf.getProducts().remove(product);
                panel.remove(productButton);
                panel.revalidate();
                panel.repaint();
                JOptionPane.showMessageDialog(productsFrame, "Product deleted
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
            }
        });
    }

    productsFrame.add(new JScrollPane(panel));
    productsFrame.setVisible(true);
}

// Authenticate user
private static User authenticateUser(String username, String password) {

```

```

        for (User user : users) {
            if (user.getUsername().equals(username) &&
user.getPassword().equals(password)) {
                return user;
            }
        }
        return null;
    }

```

// Register user

```

private static boolean registerUser(String username, String password)
{
    for (User user : users) {
        if (user.getUsername().equals(username))
            { return false;
        }
    }
    users.add(new User(username, password));
    return true;
}

```

// Shelf Class

```

private static class Shelf
{
    private String name;
    private ArrayList<Product> products = new ArrayList<>();

    public Shelf(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }

    public ArrayList<Product> getProducts()
    {
        return products;
    }
}

```

```
        public void addProduct(Product product) {  
            products.add(product);  
        }  
    }  
}
```

// Product Class

```
private static class Product  
{ private String name;  
  private int stock;  
  
  public Product(String name, int stock) {  
      this.name = name;  
      this.stock = stock;  
  }  
  
  public String getName()  
  { return name;  
  }  
  
  public int getStock()  
  { return stock;  
  }  
  
  public void setStock(int stock)  
  { this.stock = stock;  
  }  
}
```

// User Class

```
private static class User  
{ private String  
  username; private String  
  password;  
  
  public User(String username, String password)  
  { this.username = username;  
    this.password = password;
```

```
}
```

```
public String getUsername() {  
    return username;  
}
```

```
public String getPassword()  
    { return password;  
}
```

```
}
```

```
}
```