INVENTORY MANAGEMENT SYSTEM

Mugil M R (21z322)
Sanjay P (21z344)
Vishnu Dharsan A S (21z370)

19Z512 - SOFTWARE PACKAGE DEVELOPMENT

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

Of Anna University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

INDEX

PROBLEM STATEMENT

The current industry management system is outdated and inefficient. It is difficult to use, and it does not provide the insights that are needed to make informed decisions. This is leading to problems such as:

i. Increased costs: The current system is not very efficient, which is leading to increased costs. For example, it is difficult to track inventory levels, which can lead to stockouts and lost sales.
ii. Poor decision-making: The current system does not provide the insights that are needed to make informed decisions. This is leading to poor decision-making, which is impacting the bottom line.
iii. Wasted time: The current system is difficult to use, which is wasting time for employees. This time could be better spent on other tasks, such as generating new leads or improving customer service. Inefficient workflows: The outdated system lacks streamlined workflows, resulting in inefficient processes and delays in production, supply chain management, and other critical operations. This inefficiency leads to reduced productivity and customer dissatisfaction.
iv. Limited scalability: The current system may struggle to accommodate the organization's growth and changing requirements. It may lack the flexibility to adapt to new technologies, industry trends, or expanding operations, hindering scalability and hindering business expansion.
v. Lack of integration: The outdated system may not integrate well with other software applications or systems used within the organization. This lack of integration leads to data silos, manual data entry, and challenges in sharing information across departments, hindering collaboration and efficiency.
vi. Compliance risks: The outdated system may not adequately support regulatory compliance requirements specific to the industry. This exposes the organization to compliance risks, such as non-compliance with safety standards, data protection regulations, or environmental regulations.
vii. Insufficient reporting and analytics: The current system may lack robust reporting and analytics capabilities, making it challenging to derive meaningful insights from the data. This hampers the organization's ability to track key performance indicators, identify trends, and make data-driven decisions.
viii. Inadequate customer relationship management: The outdated system may not effectively support customer relationship management processes. This can result in poor customer service, difficulty in managing customer information, and missed opportunities for upselling or cross-selling.

ix. Predictive Analytics: The system should incorporate predictive analytics capabilities

to anticipate future trends, demands, and potential issues. By analyzing historical data and utilizing advanced algorithms, the system can provide forecasts for production needs, inventory requirements, maintenance schedules, and market demand. This feature empowers organizations to make proactive decisions, optimize resource allocation, minimize risks, and seize opportunities ahead of time.

x. Supplier Performance Management: The system should include a module for supplier performance management. This feature allows organizations to assess and track the performance of their suppliers based on predefined metrics such as delivery timeliness, quality standards, pricing, and responsiveness. It enables organizations to identify top-performing suppliers, mitigate risks associated with underperforming suppliers, and foster strategic relationships that contribute to operational excellence and customer satisfaction.

To address these problems, a new industry management system should be implemented with modern features and capabilities that streamline processes, improve decision-making, enhance user experience, provide real-time insights, and ensure compliance with industry standards. The upgraded industry management system will provide organizations with even greater capabilities to optimize operations, enhance decision-making, and effectively manage their supplier relationships.

SOFTWARE REQUIREMENTS SPECIFICATION

ABSTRACT:

The Industry Management System (IMS) is a comprehensive software solution designed to streamline and optimize operations within industrial settings. This abstract presents an overview of the key features, benefits, and functionalities of IMS, an innovative and robust platform tailored to enhance efficiency, productivity, and decision-making in various industrial sectors.IMS integrates cutting-edge technologies and intelligent algorithms to facilitate seamless data collection, analysis, and visualization. The system can efficiently manage diverse aspects of industrial processes, including production planning, resource allocation, inventory management, quality control, and workforce scheduling. By centralizing essential data and automating routine tasks, IMS empowers organizations to minimize manual errors, reduce downtime, and enhance overall operational performance. The main application of this system in brief is that if the stocks in the industry is over this system notifies and helps in reordering. It also helps in inventory tracking, that it gives all information regarding the product, product sales etc. If any customer orders a product then all the details of the ordered product is sent to the customer itself directly including the tracking of the product.

INTRODUCTION:

PURPOSE:

The purpose of this document is to outline the requirements for the development of an industrial management system. The system will provide functionality for managing various industrial processes, including production, inventory, tracking, reordering and reporting.

SCOPE:

The system will be designed to support industrial organizations across different sectors, such as manufacturing, logistics, and warehousing. It will enable efficient management of resources, data, and workflows to optimize operations and enhance productivity.

SYSTEM FEATURES:

1. **Inventory management:** enables tracking and management of inventory.
2. **Real time inventory management:** includes tracking incoming stock, outgoing orders, and current stock levels in the warehouse or store.
3. **Reordering of Inventory:** helps companies avoid running out of products or tying up too much capital in inventory.
4. **Email notification:** helps in sending email notification of their order status send to them periodically.
5. **Product Management:** Managing the various products manufactured and storing product information.

FUNCTIONAL REQUIREMENTS:

1. **User Management:** The system should provide user authentication, authorization, and user role management to control access to different features and functionalities based on user roles and permissions.

2. **Inventory Management:** The system should enable tracking and management of inventory, including real-time stock levels, stock movements, purchase orders, sales orders, and product information.

2.1). **Real-time Inventory Tracking:** One of the primary functionalities of an inventory management system is to provide real-time tracking of inventory levels. This includes tracking incoming stock, outgoing orders, and current stock levels in the warehouse or store. With real-time tracking, businesses can always have an accurate view of their inventory, reducing the risk of stockouts and overstocking. It allows businesses to make informed decisions on purchasing, sales, and restocking, optimizing inventory turnover and minimizing holding costs.

2.2). **Reordering of Inventory:** Inventory reaches a specific threshold; our Inventory Management System can be programmed to tell managers to reorder that product.

This helps companies avoid running out of products or tying up too much capital in inventory. This is very good feature and add extra advantages to our system. Due to less involvement of human chances of error has been reduced exponentially.

2.3). **Email notification:** Whenever customer books any order in our company an email alert has been sent to him/her as confirmation and tracking id and email notification of their order status send to them periodically.

3.. **<u>Sales and Customer Relationship Management (CRM)</u>:** The system should enable managing customer information, tracking sales leads, managing customer orders, generating invoices, and maintaining customer communication history.

NON-FUNCTIONAL REQUIREMENTS:

1. **<u>Performance</u>:** Specifies the system's response time, throughput, and resource utilization under different workloads to ensure efficient and timely processing of industry-related tasks.
2. **<u>Reliability</u>:** Defines the system's ability to perform consistently and reliably over a specified period, ensuring minimal downtime, data loss, or system failures.
3. **<u>Security</u>:** Specifies the measures to protect the system, data, and user information from unauthorized access, breaches, or malicious activities. It includes authentication, authorization, encryption, and auditing mechanisms.
4. **<u>Usability</u>:** Defines the system's ease of use, intuitiveness, and user-friendliness to ensure that industry professionals can navigate and interact with the system efficiently, even with varying levels of technical expertise.
5. **<u>Scalability</u>:** Specifies the system's ability to handle increased workloads, accommodate a growing number of users, and scale its resources (e.g., processing power, storage) without significant performance degradation.
6. **<u>Maintainability</u>:** Defines the ease with which the system can be maintained, modified, and enhanced over time. It includes factors such as code readability, documentation, modularity, and adherence to software engineering best practices.
7. **<u>Compliance</u>:** Addresses the system's adherence to industry-specific regulations, standards, and legal requirements, ensuring that the system meets necessary compliance criteria, such as data privacy laws or industry-specific regulations.
8. **<u>Availability</u>:** Specifies the system's uptime requirements, including backup and disaster recovery mechanisms, to ensure the system remains

accessible and operational as per the defined service level agreements (SLAs).

9. **<u>Performance Efficiency</u>:** Defines the system's resource utilization and efficiency, such as CPU, memory, and network utilization, to ensure optimal performance and minimize wastage of system resources.

10. **<u>Interoperability</u>:** Specifies the system's ability to integrate and interact with other existing systems, software, or hardware components used in the industry, facilitating seamless data exchange and workflow integration.

SYSTEM CONFIGURATION:

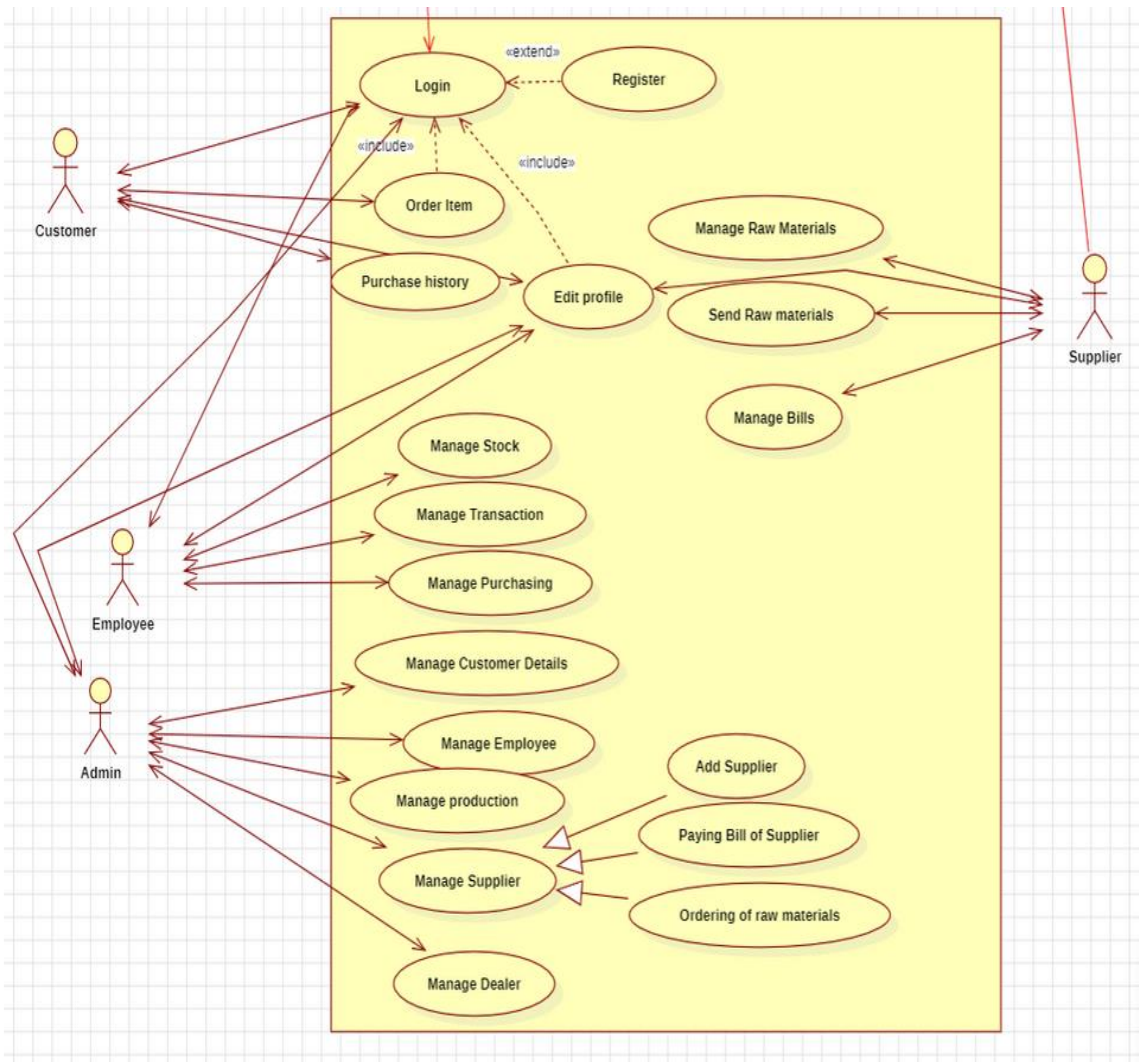
HARDWARE INTERFACE:

- Processor: Intel i3 or AMD Ryzen 3 or Higher
- RAM: 2 GB or Higher
- Graphic Card: GeForce 710 or Higher
- The two parties should be connected by LAN or WAN for the communication purpose.

SOFTWARE INTERFACE:

This software is compatible with Windows, Linux and Mac operating systems. Software is web based so that it requires browser and internet connection.

USE CASE MODEL



DOMAIN CLASSES

- Supplier
- Customer
- Admin
- Employee
- Transaction
- Inventory
- System
- Email Receipt
- Software
- Industry
- Bank
- Bank Employee
- Restock
- Alert

People:

1. Supplier
2. Customer
3. Admin
4. Employee
5. Bank Employee

Places

1. Inventory

Things

1. Email Receipt
2. System

Organization

1. Industry
2. Bank

Concepts

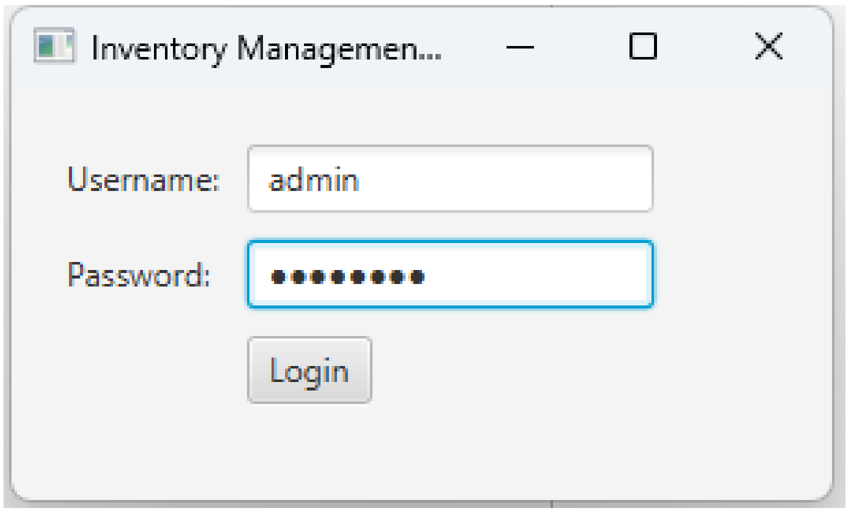
1. Software

Events

1. Transaction
2. Restock
3. Alert

GRAPHICAL USER INTERFACE

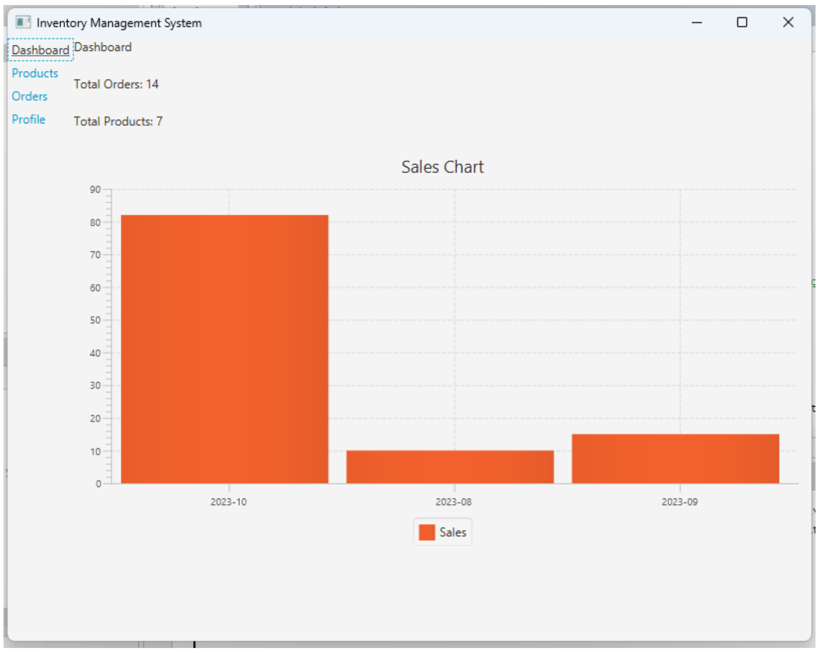
1). Login


Inventory Managemen...

Username: admin

Password: ••••••••

2). Dashboard

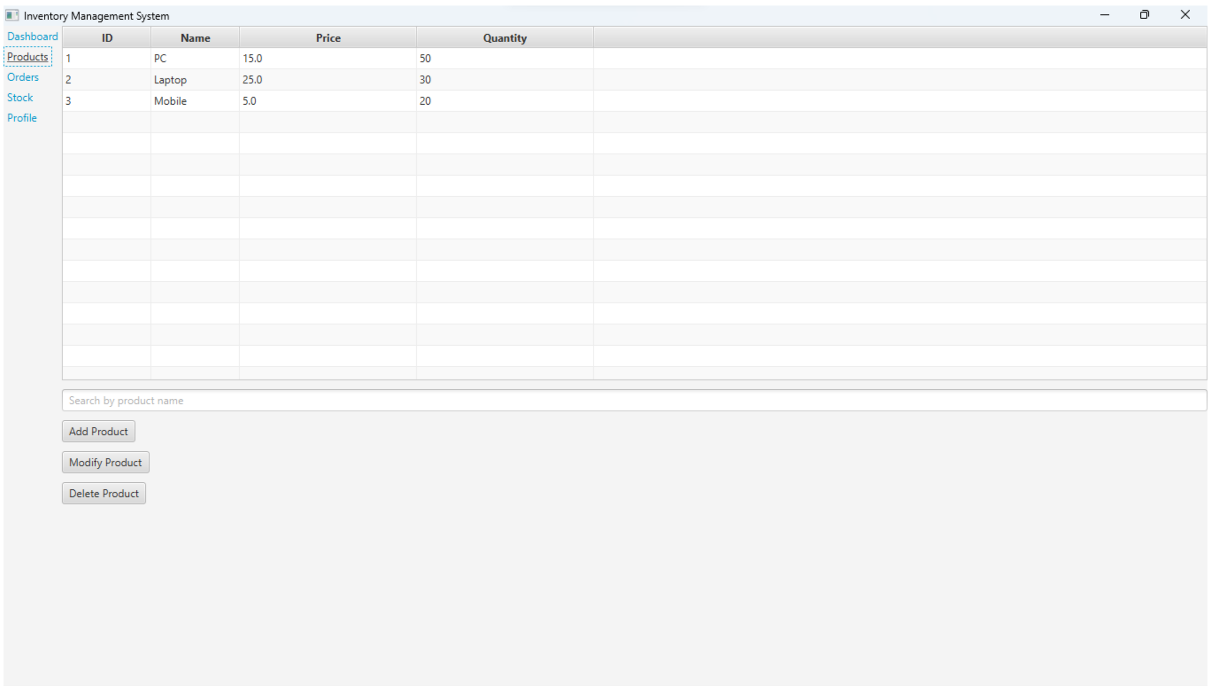

Inventory Management System

Dashboard

Total Orders: 14

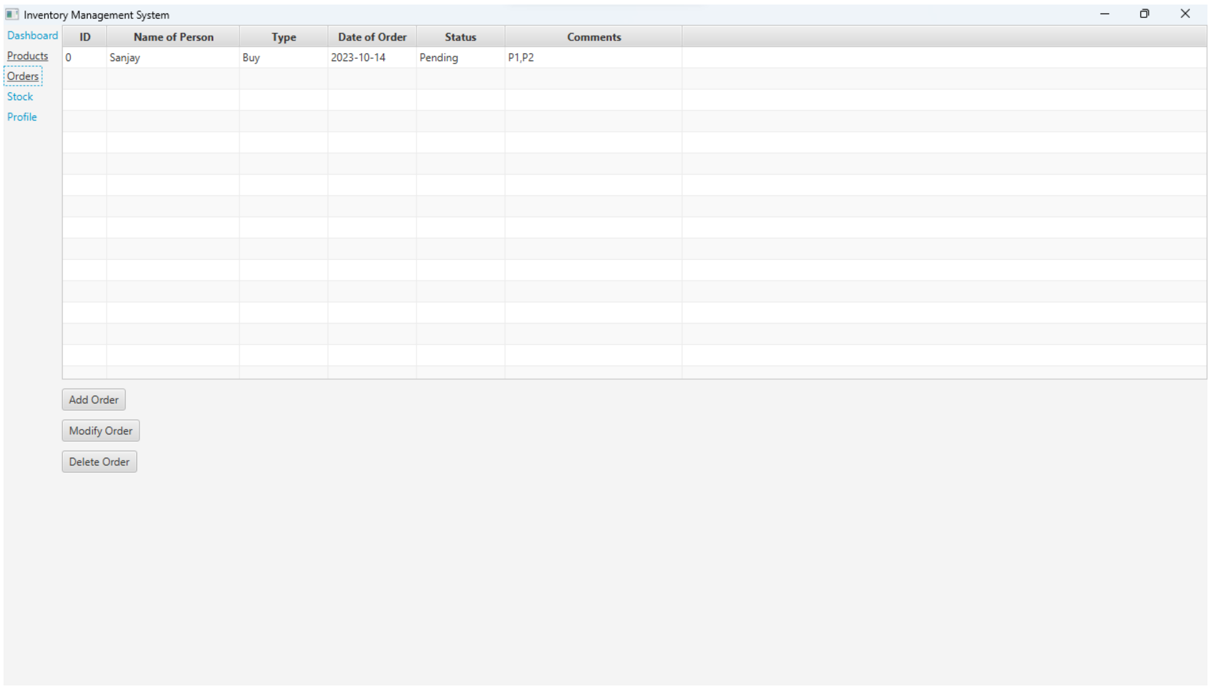
Total Products: 7

3). Products


Inventory Management System

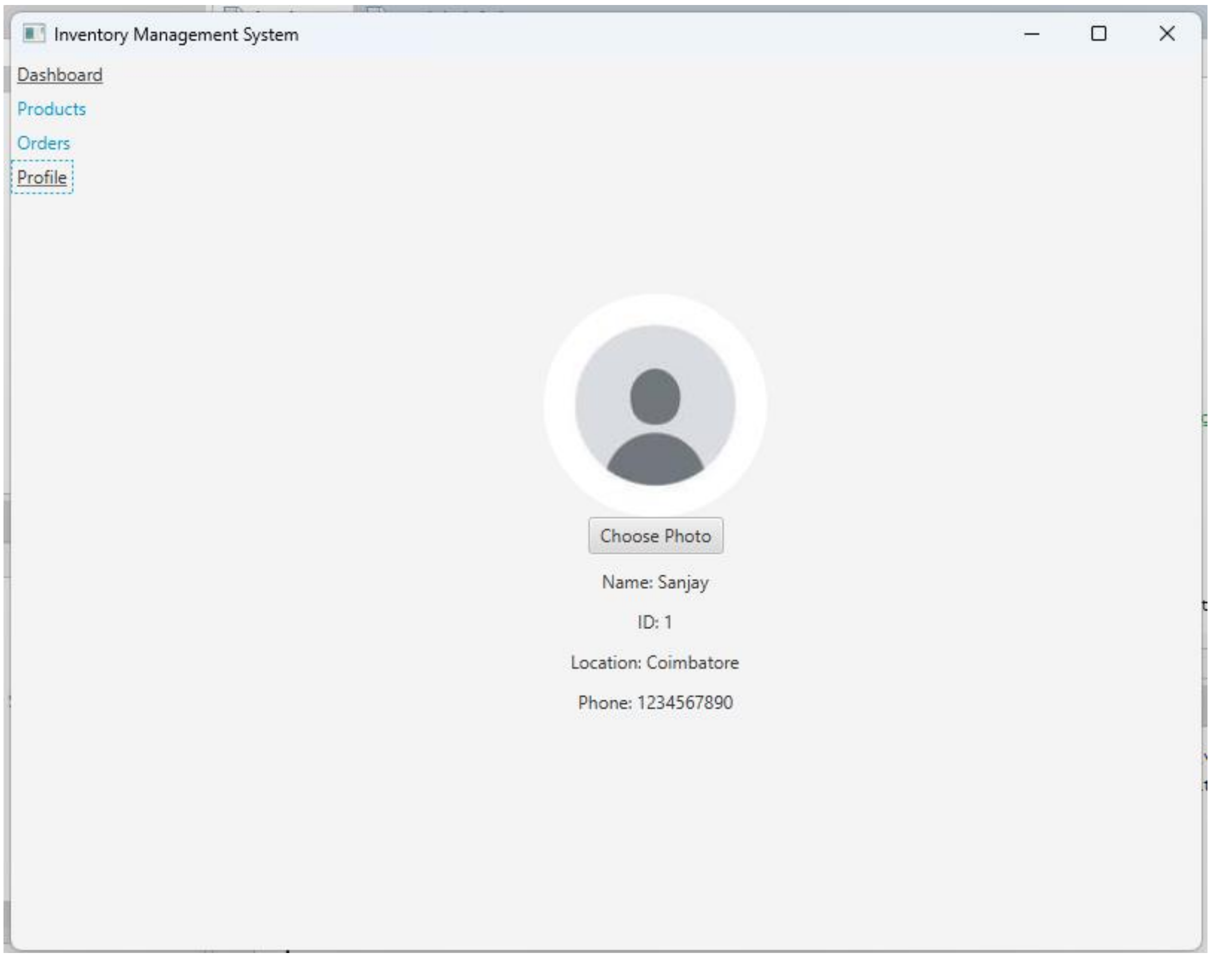
ID	Name	Price	Quantity
1	PC	15.0	50
2	Laptop	25.0	30
3	Mobile	5.0	20

4). Orders


Inventory Management System

ID	Name of Person	Type	Date of Order	Status	Comments
0	Sanjay	Buy	2023-10-14	Pending	P1,P2

5). Profile



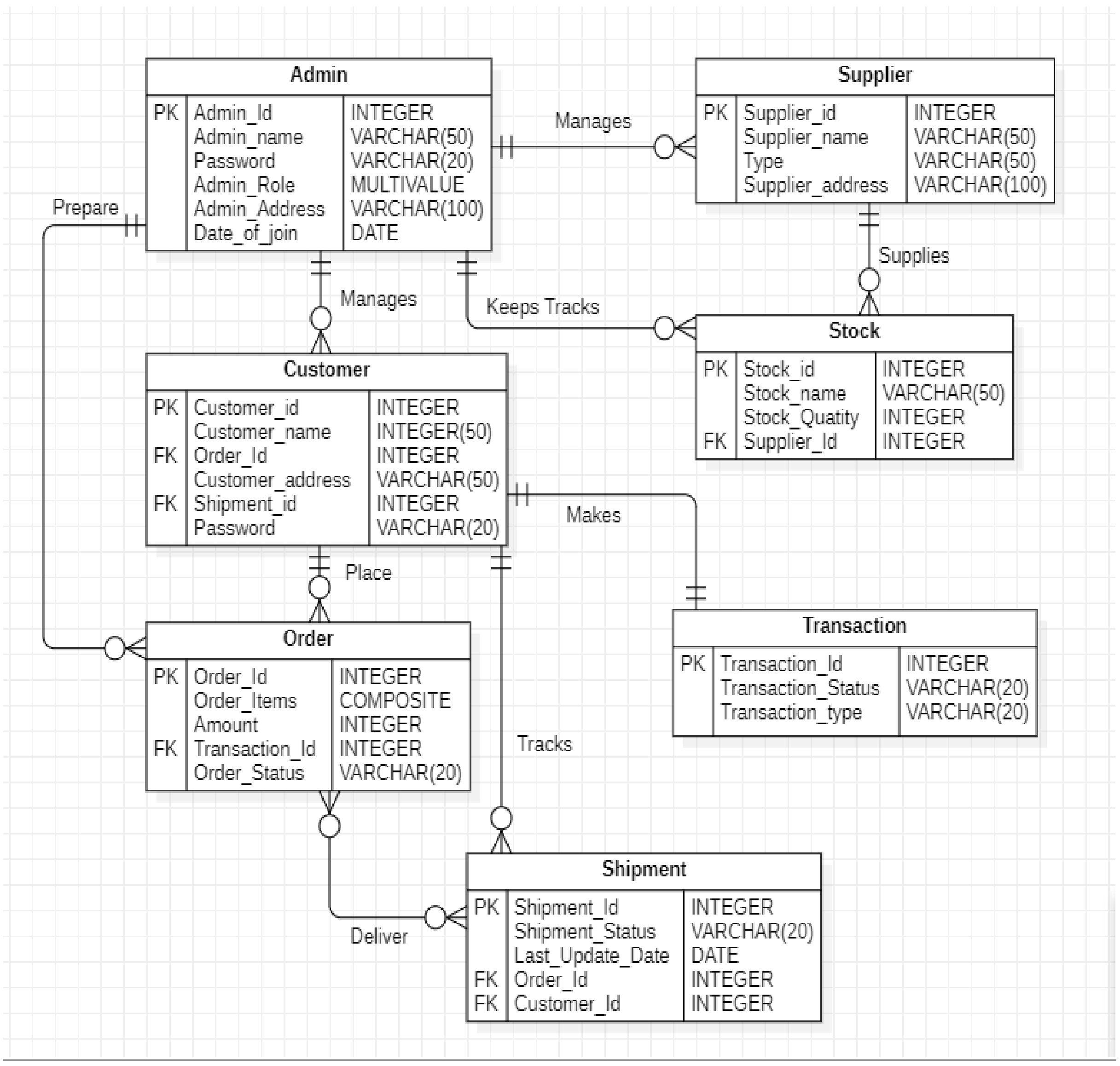
Name: Sanjay

ID: 1

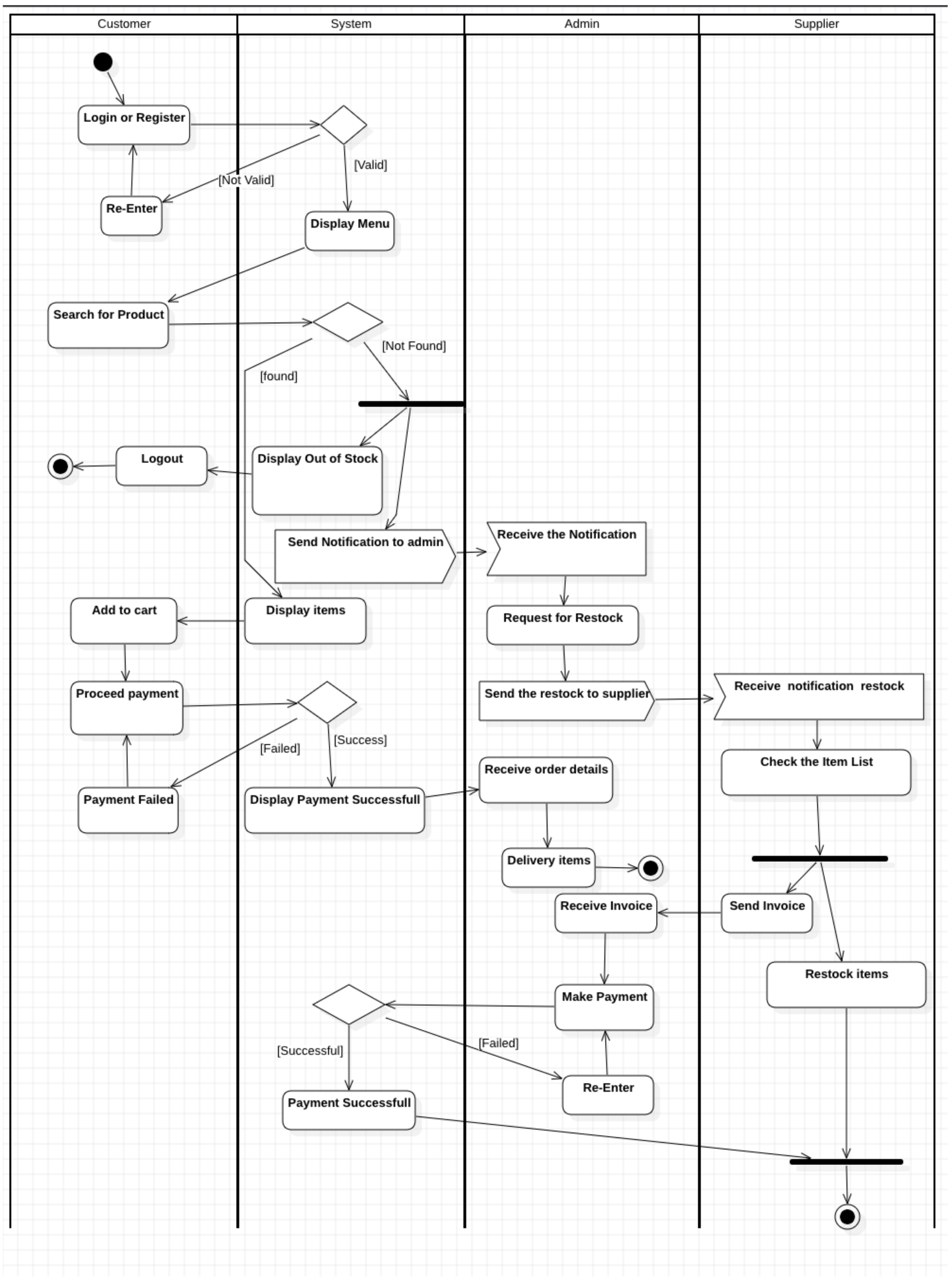
Location: Coimbatore

Phone: 1234567890

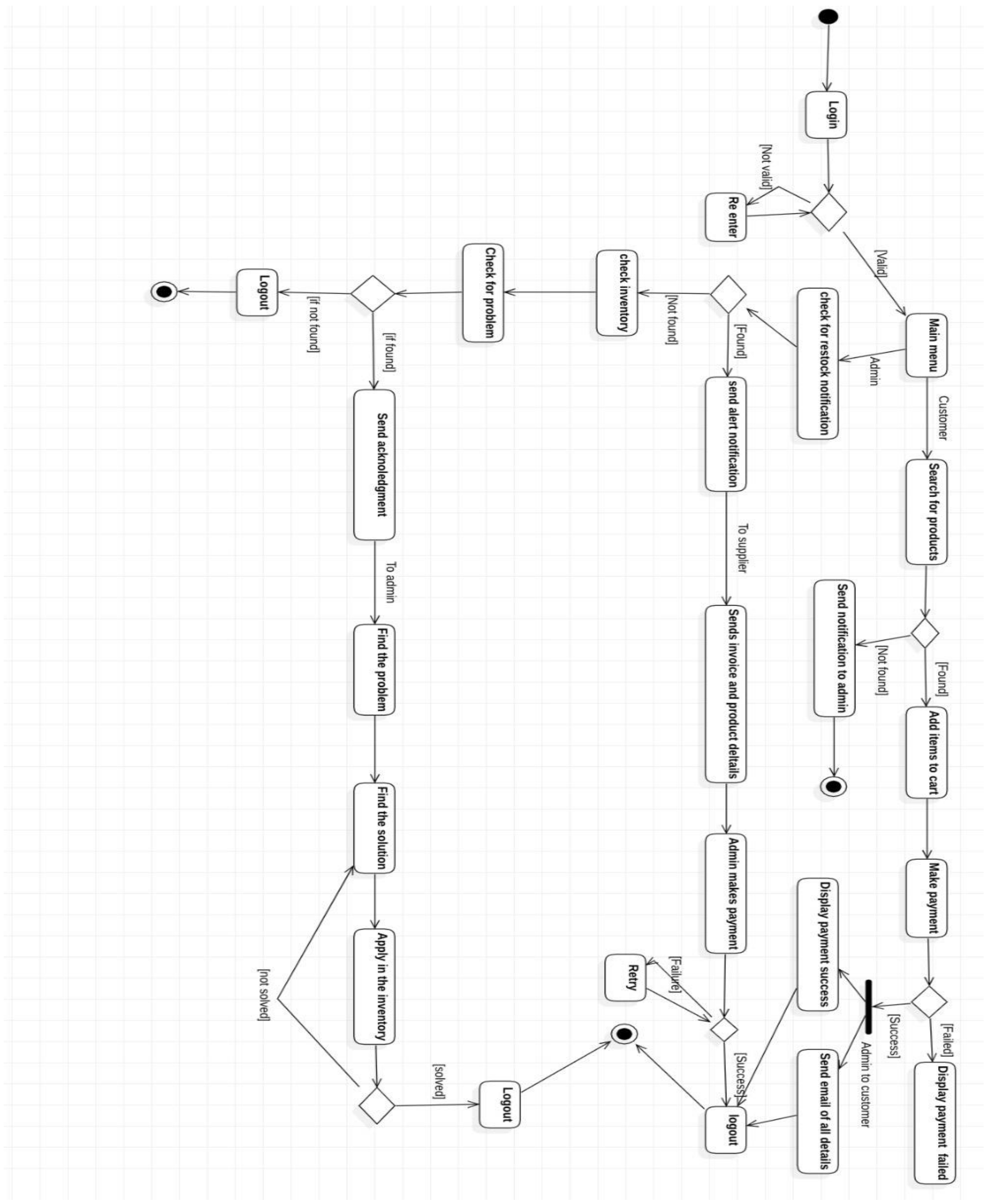
ER DIAGRAM


Admin
PK Admin_Id INTEGER
Admin_name VARCHAR(50)
Password VARCHAR(20)
Admin_Role MULTIVALUE
Admin_Address VARCHAR(100)
Date_of_join DATE
Manages
Supplier
PK Supplier_id INTEGER
Supplier_name VARCHAR(50)
Type VARCHAR(50)
Supplier_address VARCHAR(100)
Supplies
Prepare
Manages
Keeps Tracks
Stock
PK Stock_id INTEGER
Stock_name VARCHAR(50)
Stock_Quatity INTEGER
FK Supplier_Id INTEGER
Customer
PK Customer_id INTEGER
Customer_name INTEGER(50)
FK Order_Id INTEGER
Customer_address VARCHAR(50)
FK Shipment_id INTEGER
Password VARCHAR(20)
Makes
Place
Transaction
PK Transaction_Id INTEGER
Transaction_Status VARCHAR(20)
Transaction_type VARCHAR(20)
Order
PK Order_Id INTEGER
Order_Items COMPOSITE
Amount INTEGER
FK Transaction_Id INTEGER
Order_Status VARCHAR(20)
Tracks
Shipment
PK Shipment_Id INTEGER
Shipment_Status VARCHAR(20)
Last_Update_Date DATE
FK Order_Id INTEGER
FK Customer_Id INTEGER
Deliver

ACTIVITY DIAGRAM

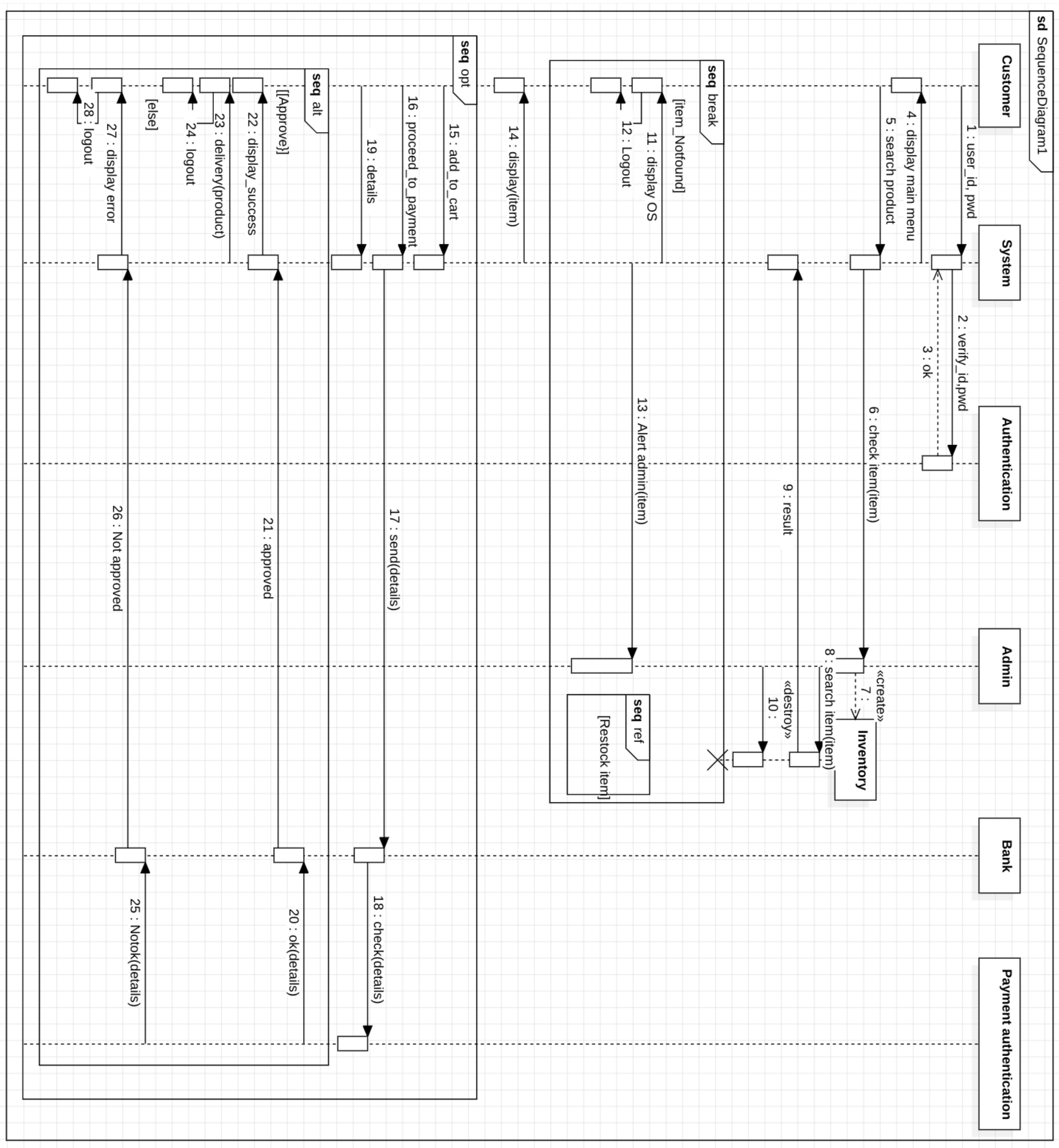


STATE DIAGRAM



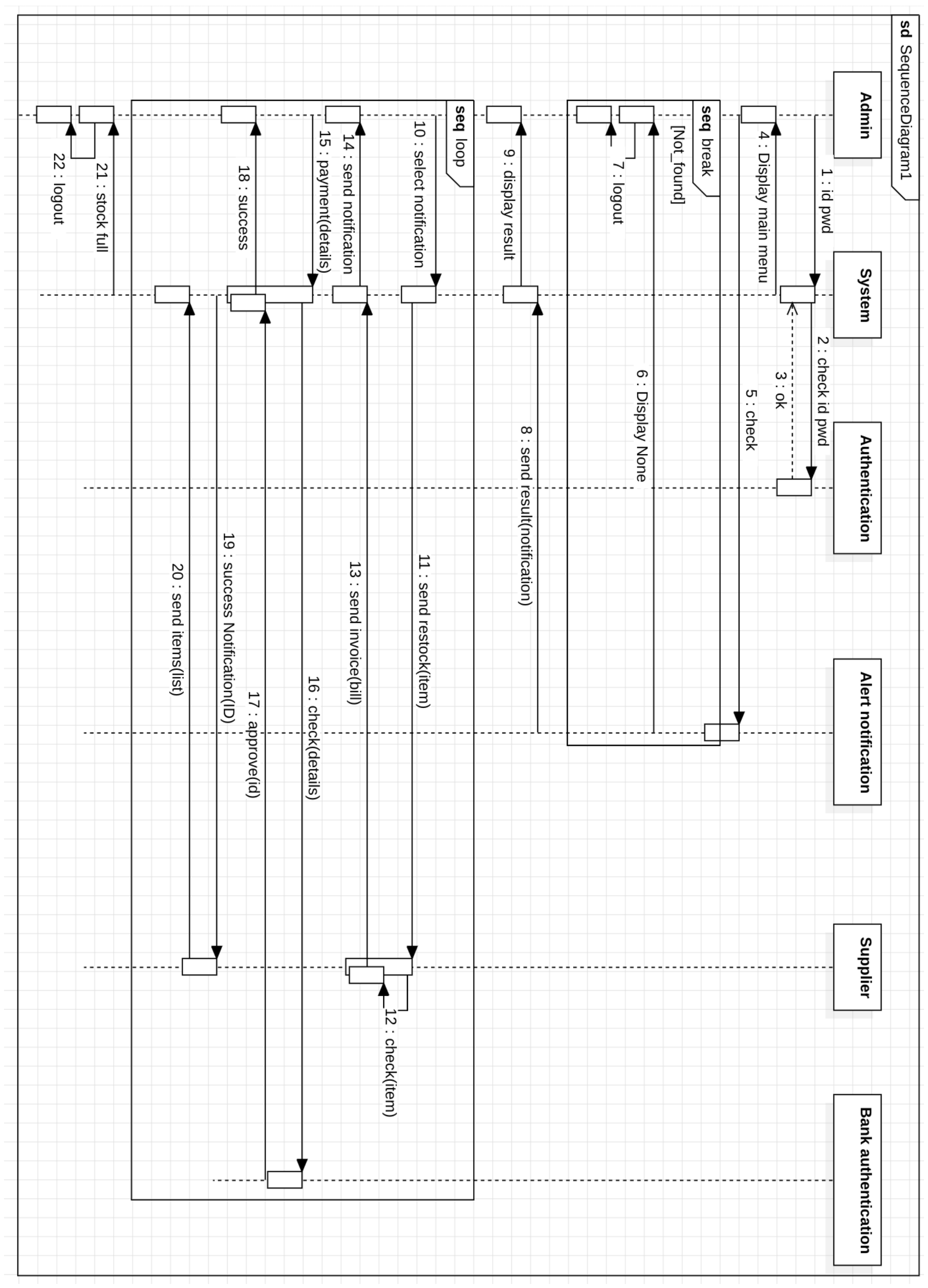
SEQUENCE DIAGRAM

(Ordering of item)

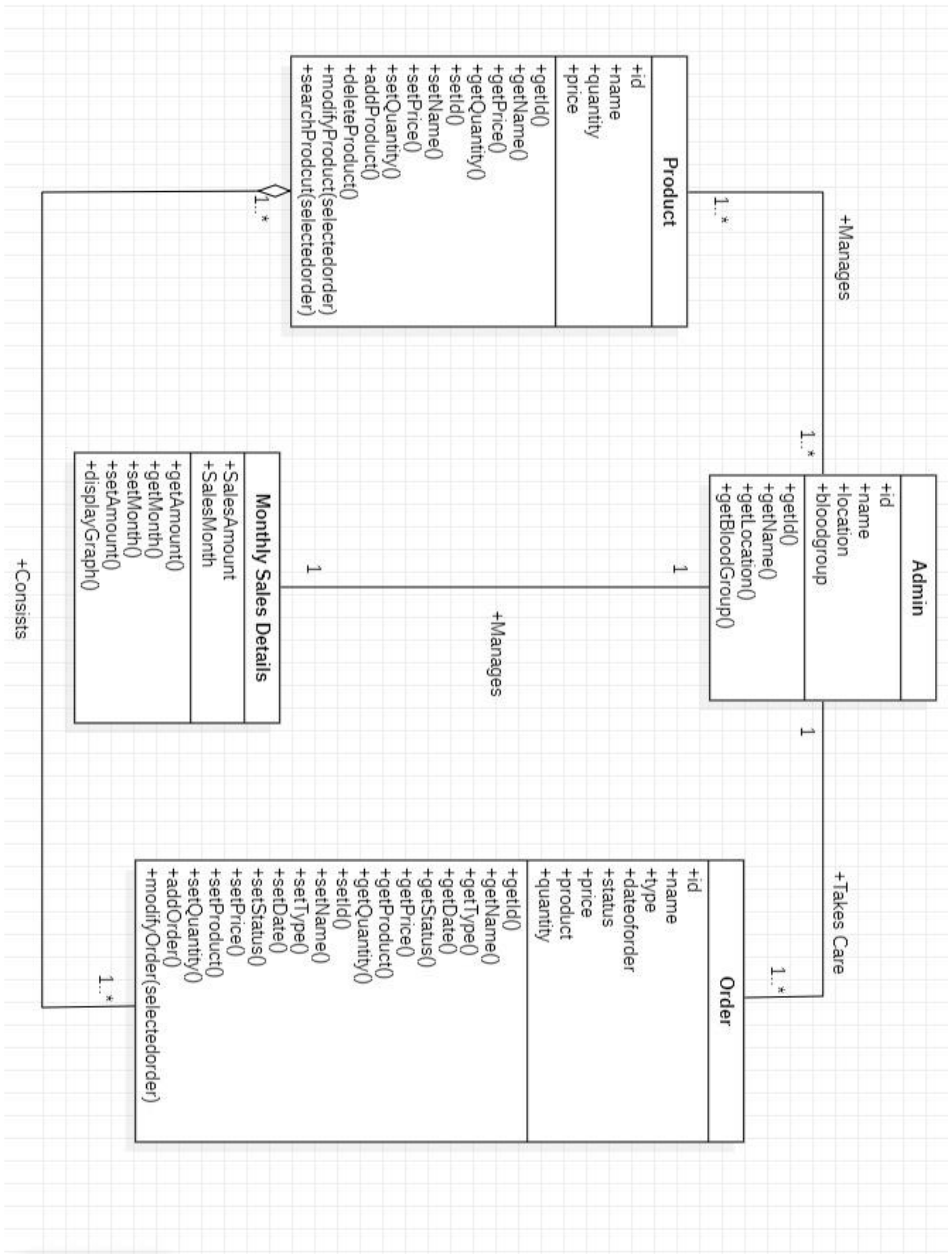


SEQUENCE DIAGRAM

(Restock of item)



CLASS DIAGRAM



TESTING

Test Case Specifications for System Testing

S.NO	Test Case	Condition being Checked	Expected Output
1.	User Login with Invalid Credential	Username and password checked in the database.	It shows the error alert asking user to enter correct credential.
2.	User Login with incomplete field	If username or password field or incomplete	It shows error alert that username field or password field is empty.
3.	User Login with correct Credential	Username and password checked in database	It shows dashboard page.
4.	Add product with incomplete field	Checks any field is incomplete	Shows error message
5.	Add product with already existing product	Checks product in inventory	Shows add product message
6.	Add product new item	Checks product in inventory	Shows add product message
7.	Modify product without get selected	Checks if product is selected	Shows error message
8.	Modify product	Checks if product enters correctly	Shows product modified
9.	Delete product without get selected	Checks if product is selected	Show error message
10.	Search product	Checks if product available	Shows the products
11.	Add orders with incomplete field	Checks all field are entered	Shows error message
12.	Add orders with proper details	Checks inventory if product out of stock	Shows order placed

13.	Add orders with proper details but product is out of stock	Checks inventory if product is out of stock	Shows errors message
14.	Modify order without selecting the order	Check if order is selected	Shows error message
15.	Modify order	Check if order details entered properly	Shows order placed message
16.	Incorrect email	Checks if email address is correct.	Shows can’t send email message should display

APPENDIX

```
package com.mycompany.ims;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import java.util.Optional;
import javafx.scene.control.*;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.stage.Modality;
import javafx.stage.StageStyle;
import javafx.beans.property.*;
import java.time.LocalDate;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import java.io.*;
import javax.mail.*;
import javafx.scene.shape.Circle;
import javafx.stage.FileChooser;
import javafx.geometry.Pos;
import java.sql.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.util.Properties;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.chart.XYChart.Series;

public class App extends Application {
    private Stage primaryStage;
    private BorderPane adminLayout;
    private StackPane contentArea;
    private String username;
    private final ObservableList<Product> products = FXCollections.observableArrayList();
    private final ObservableList<Order> orders = FXCollections.observableArrayList();
    TableView<Product> productTable=createProductTable();
    TableView<Order> ordersTable;

    private static final String DB_URL = "jdbc:mysql://localhost:3306/inventory";
    private static final String DB_USER = "root";
    private static final String DB_PASSWORD = "Sanjay@1234567890";

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        try {
        this.primaryStage = primaryStage;
        primaryStage.setTitle("Inventory Management System");
        GridPane loginGrid = createLoginGrid();
        String css = "body { background-color: #333333; }\n" +
                ".sidebar { background-color: #444444; color: white; padding: 10px; width: 200px; }\n" +
                ".sidebar .hyperlink { text-decoration: none; color: white; }\n" +
                ".sidebar .hyperlink:hover { text-decoration: underline; }";
        Scene loginScene = new Scene(loginGrid, 300, 150);
        loginScene.getStylesheets().add("data:text/css," + css);
        primaryStage.setScene(loginScene);
        primaryStage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }
    }

    private GridPane createLoginGrid() {
    GridPane grid = new GridPane();
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(20, 20, 20, 20));

    Label usernameLabel = new Label("Username:");
    TextField usernameField = new TextField();
    Label passwordLabel = new Label("Password:");
    PasswordField passwordField = new PasswordField();
    Button loginButton = new Button("Login");

    grid.add(usernameLabel, 0, 0);
    grid.add(usernameField, 1, 0);
    grid.add(passwordLabel, 0, 1);
    grid.add(passwordField, 1, 1);
    grid.add(loginButton, 1, 2);

    loginButton.setOnAction(e -> {
        String u = usernameField.getText();
        String password = passwordField.getText();
        String name = null;
        username = u;

        // Check if either field is empty
        if (u.isEmpty() && password.isEmpty()) {
```

```
    showAlert("Login Failed", "Please enter both username and password.");
    return; // Stop further processing
}
if (u.isEmpty()) {
    showAlert("Login Failed", "Please enter  username.");
    return; // Stop further processing
}
if (u.isEmpty()) {
    showAlert("Login Failed", "Please enter password.");
    return; // Stop further processing
}
// Connect to the database and perform login
try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
    if (connection != null) {
        // Create a SQL query to check the user's credentials
        String sql = "SELECT * FROM users WHERE username = ? AND password = ?";

        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
            preparedStatement.setString(1, u);
            preparedStatement.setString(2, password);
            ResultSet resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                // User credentials are valid
                name = resultSet.getString("name");
                showAlert("Login Successful", "Welcome, " + name + "!");
                switchToAdminDashboard();
            } else {
                // User credentials are invalid
                showAlert("Login Failed", "Please check your credentials.");
            }
        }
    }
} catch (SQLException ex) {
    ex.printStackTrace();
}
```

```
    });

    return grid;
}

    private void showAlert(String title, String message) {
    Alert alert = new Alert(AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}
    private void switchToAdminDashboard() {
        adminLayout = new BorderPane();

        // Create the sidebar with links to dashboard, products, orders, stock, and profile
        VBox sidebar = new VBox();
        Hyperlink dashboardLink = new Hyperlink("Dashboard");
        Hyperlink productsLink = new Hyperlink("Products");
        Hyperlink ordersLink = new Hyperlink("Orders");
        Hyperlink profileLink = new Hyperlink("Profile");

        dashboardLink.getStyleClass().add("hyperlink");  // Add a CSS class to the hyperlinks
        productsLink.getStyleClass().add("hyperlink");
        ordersLink.getStyleClass().add("hyperlink");
        profileLink.getStyleClass().add("hyperlink");

        sidebar.getChildren().addAll(dashboardLink, productsLink, ordersLink, profileLink);
        adminLayout.setLeft(sidebar);

        contentArea = new StackPane();
        adminLayout.setCenter(contentArea);

        // Add event handlers for the sidebar links to switch content
        dashboardLink.setOnAction(e -> showDashboard());
        productsLink.setOnAction(e -> showProducts());
        ordersLink.setOnAction(e -> showOrders());
        profileLink.setOnAction(e -> showProfile());

        // Display the dashboard by default
        showDashboard();

        Scene adminScene = new Scene(adminLayout, 800, 600);
        primaryStage.setScene(adminScene);
    }
    private void showDashboard() {
        VBox dashboardContent = new VBox();
        dashboardContent.setSpacing(20);

        Label titleLabel = new Label("Dashboard");
        titleLabel.getStyleClass().add("header-label");

        int totalOrders = fetchTotalOrdersFromDB();
        int totalProducts = fetchTotalProductsFromDB();
        // Display total orders and total products
        Label totalOrdersLabel = new Label("Total Orders: " + totalOrders);
        Label totalProductsLabel = new Label("Total Products: " + totalProducts);

        // Create and display the sales graph
        BarChart<String, Number> salesChart = createSalesChartFromDatabase();

        dashboardContent.getChildren().addAll(titleLabel, totalOrdersLabel, totalProductsLabel, salesChart);
        contentArea.getChildren().clear();
        contentArea.getChildren().add(dashboardContent);
    }
    private int fetchTotalOrdersFromDB() {
    int totalOrders = 0;

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "SELECT COUNT(*) FROM orders"; // Modify this query as needed
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                ResultSet resultSet = preparedStatement.executeQuery();

                if (resultSet.next()) {
                    totalOrders = resultSet.getInt(1);
                }
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return totalOrders;
}
    private int fetchTotalProductsFromDB() {
    int totalProducts = 0;
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "SELECT COUNT(*) FROM products"; // Modify this query as needed
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                ResultSet resultSet = preparedStatement.executeQuery();

                if (resultSet.next()) {
                    totalProducts = resultSet.getInt(1);
                }
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return totalProducts;
}
    private BarChart<String, Number> createSalesChartFromDatabase() {
    // Define the axes
    CategoryAxis xAxis = new CategoryAxis(); // X-axis for months
    NumberAxis yAxis = new NumberAxis();     // Y-axis for sales

    // Create the chart
```

```
        BarChart<String, Number> barChart = new BarChart<>(xAxis, yAxis);
        barChart.setTitle("Sales Chart");

        // Fetch sales data from the database
        List<MonthlySalesData> monthlySalesData = fetchMonthlySalesDataFromDB();

        // Create a series for sales data
        Series<String, Number> salesData = new Series<>();
        salesData.setName("Sales");

        // Add data points for each month
        for (MonthlySalesData data : monthlySalesData) {
            salesData.getData().add(new XYChart.Data<>(data.getMonth(), data.getSalesAmount()));
        }

        // Add the sales data series to the chart
        barChart.getData().add(salesData);

        return barChart;
    }
        private List<MonthlySalesData> fetchMonthlySalesDataFromDB() {
        List<MonthlySalesData> monthlySalesData = new ArrayList<>();
        Map<String, Integer> salesByMonth = new HashMap<>();
        try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
            if (connection != null) {
                String sql = "SELECT order_type, order_date, quantity FROM orders"; // Modify this query as needed
                try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                    ResultSet resultSet = preparedStatement.executeQuery();

                    // Map to group sales by month


                    while (resultSet.next()) {
                        Date saleDate = resultSet.getDate("order_date");
                        int orderQuantity = resultSet.getInt("quantity");
                        String type = resultSet.getString("order_type");
                // Format the date to 'YYYY-MM' (e.g., '2023-10')
                String monthYear = new SimpleDateFormat("yyyy-MM").format(saleDate);

                // Add or accumulate sales for the month, considering order quantity
                if("Sell".equals(type)){
                    salesByMonth.put(monthYear, salesByMonth.getOrDefault(monthYear, 0) + orderQuantity);
                }
            }
        }
    }
} catch (SQLException ex) {
    ex.printStackTrace();
}

// Convert the map to a list of MonthlySalesData
for (Map.Entry<String, Integer> entry : salesByMonth.entrySet()) {
    monthlySalesData.add(new MonthlySalesData(entry.getKey(), entry.getValue()));
}

return monthlySalesData;
}

class MonthlySalesData {
private String month;
private double salesAmount;

public MonthlySalesData(String month, double salesAmount) {
    this.month = month;
    this.salesAmount = salesAmount;
}

public String getMonth() {
    return month;
} public double getSalesAmount() {
    return salesAmount;
}}
```

```
    private void filterProducts(String searchQuery) {
    ObservableList<Product> filteredProducts = FXCollections.observableArrayList();

    for (Product product : products) {
        if (product.getName().toLowerCase().contains(searchQuery.toLowerCase())) {
            filteredProducts.add(product);
        }
    }

    // Update the TableView with the filtered products
    productTable.setItems(filteredProducts);
}
    private void showProducts() {
    productTable.getItems().clear();
    Button addButton = new Button("Add Product");
    Button modifyButton = new Button("Modify Product");
    Button deleteButton = new Button("Delete Product");
    productTable.setEditable(true);

    // Initialize an ObservableList to store product data
//    ObservableList<Product> products = FXCollections.observableArrayList();

    // Retrieve product data from the database and populate the 'products' list
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "SELECT * FROM products"; // Modify this query as needed
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                ResultSet resultSet = preparedStatement.executeQuery();

                while (resultSet.next()) {
                    int productId = resultSet.getInt("pid");
                    String productName = resultSet.getString("productname");
                    double productPrice = resultSet.getDouble("sellprice");
                    int productquantity = resultSet.getInt("quantity");

                    // Create a Product instance and add it to the 'products' list
                Product product = new Product(productId, productName, productPrice, productquantity);
                products.add(product);
            }
        }
    }
} catch (SQLException ex) {
    ex.printStackTrace();
}
productTable.setItems(products);
TextField searchField = new TextField();
searchField.setPromptText("Search by product name");

// Add an event handler to filter products based on the search query
searchField.textProperty().addListener((observable, oldValue, newValue) -> {
    if (newValue.isEmpty()) {
        productTable.setItems(products); // Display all products
    } else {
        filterProducts(newValue); // Filter based on search query
    }
});

addButton.setOnAction(e -> addProduct());
modifyButton.setOnAction(e -> {
Product selectedProduct = productTable.getSelectionModel().getSelectedItem();
if (selectedProduct != null) {
    modifyProduct(selectedProduct);
} else {
    showAlert("No Product Selected", "Please select a product to modify.");
}
});
deleteButton.setOnAction(e -> {
Product selectedProduct = productTable.getSelectionModel().getSelectedItem();
if (selectedProduct != null) {
    deleteProduct(selectedProduct);
} else {
    showAlert("No Product Selected", "Please select a product to delete.");
```

```
        }
    });

        VBox productsContent = new VBox();
        productsContent.setSpacing(10);
        productsContent.getChildren().addAll(productTable, searchField, addButton, modifyButton, deleteButton);

        contentArea.getChildren().clear();
        contentArea.getChildren().add(productsContent);
    }



        private TableView<Product> createProductTable() {
            TableView<Product> table = new TableView<>();
            table.setEditable(true);

            TableColumn idColumn = new TableColumn("ID");
            idColumn.setMinWidth(100);
            idColumn.setCellValueFactory(
                    new PropertyValueFactory<Product, Integer>("id"));

            TableColumn nameColumn = new TableColumn("Name");
            nameColumn.setMinWidth(100);
            nameColumn.setCellValueFactory(
                    new PropertyValueFactory<Product, String>("name"));

            TableColumn priceColumn = new TableColumn("SellPrice");
            priceColumn.setMinWidth(200);
            priceColumn.setCellValueFactory(
                    new PropertyValueFactory<Product, Double>("price"));
                  TableColumn quantityColumn = new TableColumn("Quantity");
            quantityColumn.setMinWidth(200);
            quantityColumn.setCellValueFactory(
                    new PropertyValueFactory<Product, Integer>("Quantity"));
            table.getColumns().addAll(idColumn,nameColumn,priceColumn,quantityColumn);
            return table;
    }


private void addProduct() {
    // Create a new product dialog
    Dialog<Product> dialog = new Dialog<>();
    dialog.initStyle(StageStyle.UTILITY);
    dialog.initModality(Modality.APPLICATION_MODAL);

    dialog.setTitle("Add Product");
    dialog.setHeaderText("Enter product details:");

    // Create and configure the dialog content
    VBox dialogContent = new VBox();
    dialogContent.setSpacing(10);

    TextField idField = new TextField();
    idField.setPromptText("ID");
    TextField nameField = new TextField();
    nameField.setPromptText("Name");
    TextField priceField = new TextField();
    priceField.setPromptText("Price");
    TextField quantityField = new TextField();
    quantityField.setPromptText("Quantity");

    dialogContent.getChildren().addAll(idField, nameField, priceField, quantityField);
    dialog.getDialogPane().setContent(dialogContent);

    // Create OK and Cancel buttons
    ButtonType addButton = new ButtonType("Add", ButtonBar.ButtonData.OK_DONE);
    dialog.getDialogPane().getButtonTypes().addAll(addButton, ButtonType.CANCEL);

    // Set the result converter for the dialog
    dialog.setResultConverter(dialogButton -> {
        if (dialogButton == addButton) {
            String idText = idField.getText();
            String nameText = nameField.getText();
            String priceText = priceField.getText();
            String quantityText = quantityField.getText();
            if (quantityText.isEmpty() || idText.isEmpty() || nameText.isEmpty() || priceText.isEmpty()) {
            showAlert(Alert.AlertType.ERROR, "Error", "All fields must be filled.");
            return null;
        }
            try {
                int id = Integer.parseInt(idField.getText());
                String name = nameField.getText();
                double price = Double.parseDouble(priceField.getText());
                int quantity = Integer.parseInt(quantityField.getText());
                // Create a new Product instance
                Product newProduct = new Product(id, name, price, quantity);

                // Update the database with the new product data (insert into the database)
                insertDatabase(newProduct);

                // Add the new product to the observable list
                products.add(newProduct);

                return newProduct;
            } catch (NumberFormatException e) {
                // Handle parsing errors
                return null;
            }
        }
        return null;
    });

    // Show the dialog and handle the result
    Optional<Product> result = dialog.showAndWait();
    result.ifPresent(newProduct -> {
        System.out.println("Product added: " + newProduct.getName());
        showAlert(Alert.AlertType.INFORMATION, "Success", "Product added successfully.");
    });
```

```
}
private void insertDatabase(Product newProduct) {
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "INSERT INTO products (pid, productname, sellprice, quantity) " +
                    "VALUES (?, ?, ?, ?)";
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                // Set the values for the parameters in the SQL query
                preparedStatement.setInt(1, newProduct.getId());
                preparedStatement.setString(2, newProduct.getName());
                preparedStatement.setDouble(3, newProduct.getPrice());
                preparedStatement.setInt(4, newProduct.getQuantity());
                // Execute the INSERT query
                preparedStatement.executeUpdate();
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to add the product to the database.");
    }
}

    private void showAlert(Alert.AlertType type, String title, String content) {
    Alert alert = new Alert(type);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(content);
    alert.showAndWait();
}
    private void updateDatabase(Product updatedProduct) {
    // Define your database connection details

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "UPDATE products SET productname = ?, sellprice = ?, quantity = ? WHERE pid = ?";
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                // Set the values for the parameters in the SQL query
                preparedStatement.setString(1, updatedProduct.getName());
                preparedStatement.setDouble(2, updatedProduct.getPrice());
                preparedStatement.setInt(3, updatedProduct.getQuantity());
                preparedStatement.setInt(4, updatedProduct.getId());

                // Execute the UPDATE query
                preparedStatement.executeUpdate();
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to update the product in the database.");
    }
}

private void modifyProduct(Product selectedProduct) {
    if (selectedProduct == null) {
        // No product selected, show a message or handle this case as needed
        return;
    }

    // Create a dialog to edit the product
    Dialog<Product> dialog = new Dialog<>();
    dialog.initStyle(StageStyle.UTILITY);
    dialog.initModality(Modality.APPLICATION_MODAL);

    dialog.setTitle("Modify Product");
    dialog.setHeaderText("Edit product details:");

    // Create and configure the dialog content
    VBox dialogContent = new VBox();
    dialogContent.setSpacing(10);

    TextField idField = new TextField(Integer.toString(selectedProduct.getId()));
    idField.setPromptText("ID");
```

```
TextField nameField = new TextField(selectedProduct.getName());
nameField.setPromptText("Name");
TextField priceField = new TextField(Double.toString(selectedProduct.getPrice()));
priceField.setPromptText("Price");
TextField quantityField = new TextField(Integer.toString(selectedProduct.getQuantity()));
quantityField.setPromptText("Quantity");

dialogContent.getChildren().addAll(idField, nameField, priceField, quantityField);
dialog.getDialogPane().setContent(dialogContent);

// Create OK and Cancel buttons
ButtonType modifyButton = new ButtonType("Modify", ButtonBar.ButtonData.OK_DONE);
dialog.getDialogPane().getButtonTypes().addAll(modifyButton, ButtonType.CANCEL);

// Set the result converter for the dialog
dialog.setResultConverter(buttonType -> {
    if (buttonType == modifyButton) {
        try {
            int id = Integer.parseInt(idField.getText());
            String name = nameField.getText();
            double price = Double.parseDouble(priceField.getText());
            int quantity = Integer.parseInt(quantityField.getText());

            // Update the selected product with the new values
            selectedProduct.setId(id);
            selectedProduct.setName(name);
            selectedProduct.setPrice(price);
            selectedProduct.setQuantity(quantity);

            // Update the product in the database
            updateDatabase(selectedProduct);

            return selectedProduct;
        } catch (NumberFormatException e) {
            // Handle parsing errors
            return null;
            }
        }
        return null;
    });

    // Show the dialog and handle the result
    Optional<Product> result = dialog.showAndWait();
    result.ifPresent(updatedProduct -> {
        // Handle the modified product data
        productTable.refresh(); // Refresh the table to reflect changes
    });
}


    private void deleteProduct(Product selectedProduct) {
    if (selectedProduct == null) {
        // No product selected, show a message or handle this case as needed
        return;
    }

    Alert confirmationAlert = new Alert(Alert.AlertType.CONFIRMATION);
    confirmationAlert.setTitle("Confirm Deletion");
    confirmationAlert.setHeaderText("Are you sure you want to delete this product?");
    confirmationAlert.setContentText("Product ID: " + selectedProduct.getId() + "\nProduct Name: " +
selectedProduct.getName());

    ButtonType deleteButtonType = new ButtonType("Delete", ButtonBar.ButtonData.OK_DONE);
    confirmationAlert.getButtonTypes().setAll(deleteButtonType, ButtonType.CANCEL);

    Optional<ButtonType> result = confirmationAlert.showAndWait();

    if (result.isPresent() && result.get() == deleteButtonType) {
        // User confirmed the deletion
        products.remove(selectedProduct); // Remove the product from the list
        deleteProductFromDatabase(selectedProduct);
    }
```

```
}
    private void deleteProductFromDatabase(Product productToDelete) {
        try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "DELETE FROM products WHERE pid = ?";
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                // Set the product ID for the DELETE operation
                preparedStatement.setInt(1, productToDelete.getId());

                // Execute the DELETE query
                preparedStatement.executeUpdate();
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to delete the product from the database.");
    }
}



//    private void setContent(String content) {
//        Label contentLabel = new Label(content);
//        contentArea.getChildren().clear();
//        contentArea.getChildren().add(contentLabel);
//    }
    public class Product {
    private final SimpleIntegerProperty id;
    private final SimpleStringProperty name;
    private final SimpleDoubleProperty price;
    private final SimpleIntegerProperty quantity;

    public Product(int id, String name, double price, int quantity) {
        this.id = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.price = new SimpleDoubleProperty(price);
        this.quantity = new SimpleIntegerProperty(quantity);
    }

    // Getters and setters for the fields (id, name, price, quantity)

    public int getId() {
        return id.get();
    }

    public void setId(int id) {
        this.id.set(id);
    }

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

    public void setName(String name) {
        this.name.set(name);
    }

    public double getPrice() {
        return price.get();
    }

    public void setPrice(double price) {
        this.price.set(price);
    }

    public int getQuantity() {
        return quantity.get();
    }

    public void setQuantity(int quantity) {
        this.quantity.set(quantity);
    }
}
```

```
    private void showOrders() {
    // Retrieve orders from the database and populate the 'orders' list
    retrieveOrdersFromDatabase();

    ordersTable = createOrdersTable();
    Button addOrderButton = new Button("Add Order");
    Button modifyOrderButton = new Button("Modify Order");

    ordersTable.setEditable(true);

    addOrderButton.setOnAction(e -> addOrder(ordersTable));
    modifyOrderButton.setOnAction(e -> {
        Order selectedOrder = ordersTable.getSelectionModel().getSelectedItem();
        if (selectedOrder != null) {
            modifyOrder(selectedOrder);
        } else {
            showAlert(Alert.AlertType.ERROR, "Error", "No order selected. Please select an order to modify.");
        }
    });

    checkLowStock();

    VBox ordersContent = new VBox();
    ordersContent.setSpacing(10);
    ordersContent.getChildren().addAll(ordersTable, addOrderButton, modifyOrderButton);

    contentArea.getChildren().clear();
    contentArea.getChildren().add(ordersContent);
}
private void checkLowStock() {
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "SELECT * FROM products WHERE quantity <= ?"; // Define your threshold

            // Set your low stock threshold here
            int lowStockThreshold = 10; // You can adjust this as needed
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                preparedStatement.setInt(1, lowStockThreshold);

                ResultSet resultSet = preparedStatement.executeQuery();

                while (resultSet.next()) {
                    String productName = resultSet.getString("productname");
                    int productQuantity = resultSet.getInt("quantity");

                    showAlert(Alert.AlertType.WARNING, "Low Stock Alert",
                            "Product " + productName + " has low stock. Quantity: " + productQuantity);
                }
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

private void retrieveOrdersFromDatabase() {
    // Clear the existing 'orders' list
    orders.clear();

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "SELECT * FROM orders"; // Modify this query as needed
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                ResultSet resultSet = preparedStatement.executeQuery();

                while (resultSet.next()) {
                    int id = resultSet.getInt("id");
                    String nameOfPerson = resultSet.getString("person_name");
                    String type = resultSet.getString("order_type");
                    LocalDate dateOfOrder = resultSet.getDate("order_date").toLocalDate();
                    String status = resultSet.getString("order_status");
                    double price = resultSet.getDouble("order_price");
                    String comments = resultSet.getString("productname");
                    int quantity = resultSet.getInt("quantity");

                    // Create an Order instance and add it to the 'orders' list
                    Order order = new Order(id, nameOfPerson, type, dateOfOrder, status, price, comments, quantity);
                    orders.add(order);
                }
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to retrieve orders from the database.");
    }
}

private TableView<Order> createOrdersTable() {
    TableView<Order> table = new TableView<>();

    // Define columns
    TableColumn<Order, Integer> idColumn = new TableColumn<>("ID");
    idColumn.setMinWidth(50);
    idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));

    TableColumn<Order, String> nameColumn = new TableColumn<>("Name of Person");
    nameColumn.setMinWidth(150);
    nameColumn.setCellValueFactory(new PropertyValueFactory<>("nameOfPerson"));

    TableColumn<Order, String> typeColumn = new TableColumn<>("Type");
    typeColumn.setMinWidth(100);
    typeColumn.setCellValueFactory(new PropertyValueFactory<>("type"));

    TableColumn<Order, String> dateColumn = new TableColumn<>("Date of Order");
    dateColumn.setMinWidth(100);
    dateColumn.setCellValueFactory(new PropertyValueFactory<>("dateOfOrder"));
    TableColumn<Order, String> statusColumn = new TableColumn<>("Status");
    statusColumn.setMinWidth(100);
    statusColumn.setCellValueFactory(new PropertyValueFactory<>("status"));

    TableColumn<Order, Double> priceColumn = new TableColumn<>("Price");
    priceColumn.setMinWidth(100);
    priceColumn.setCellValueFactory(new PropertyValueFactory<>("price"));

    TableColumn<Order, String> commentsColumn = new TableColumn<>("Product Name");
    commentsColumn.setMinWidth(100);
    commentsColumn.setCellValueFactory(new PropertyValueFactory<>("comments"));

    TableColumn<Order, String> quantityColumn = new TableColumn<>("Quantity");
    quantityColumn.setMinWidth(100);
    quantityColumn.setCellValueFactory(new PropertyValueFactory<>("quantity"));

    // Set the items and columns
    table.setItems(orders);
    table.getColumns().addAll(idColumn, nameColumn, typeColumn, dateColumn, statusColumn, priceColumn,
commentsColumn, quantityColumn);

    return table;
}

private void addOrder(TableView<Order> orderTable) {
    // Create a new order dialog
    Dialog<Order> dialog = new Dialog<>();
    dialog.initStyle(StageStyle.UTILITY);
    dialog.initModality(Modality.APPLICATION_MODAL);

    dialog.setTitle("Add Order");
    dialog.setHeaderText("Enter order details:");

    VBox dialogContent = new VBox();
    dialogContent.setSpacing(10);
    TextField idField = new TextField();
    idField.setPromptText("ID");
    TextField nameOfPersonField = new TextField();
    nameOfPersonField.setPromptText("Name of Person");
    TextField typeField = new TextField();
    typeField.setPromptText("Type (Buy/Sell)");
    DatePicker dateField = new DatePicker();
    dateField.setPromptText("Date of Order");
    ComboBox<String> statusComboBox = new ComboBox<>(FXCollections.observableArrayList("Pending",
"Processed"));
    statusComboBox.setPromptText("Status");
    TextField priceField = new TextField();
    priceField.setPromptText("Price");
    TextField commentsField = new TextField();
    commentsField.setPromptText("Product Name");
    TextField emailField = new TextField();
    emailField.setPromptText("Email");
    TextField quantityField = new TextField();
    quantityField.setPromptText("Quantity");
    dialogContent.getChildren().addAll(idField, nameOfPersonField, typeField, dateField, statusComboBox,
priceField, commentsField, emailField, quantityField);
    dialog.getDialogPane().setContent(dialogContent);

    ButtonType addButton = new ButtonType("Add", ButtonBar.ButtonData.OK_DONE);
    dialog.getDialogPane().getButtonTypes().addAll(addButton, ButtonType.CANCEL);

    dialog.setResultConverter(buttonType -> {
        if (buttonType == addButton) {
            if (isEmptyField(idField) || isEmptyField(nameOfPersonField) || isEmptyField(typeField)
                    || dateField.getValue() == null || statusComboBox.getValue() == null
                    || isEmptyField(priceField) || isEmptyField(commentsField)
                    || isEmptyField(emailField) || isEmptyField(quantityField)) {
                showAlert(Alert.AlertType.ERROR, "Missing Fields", "Please fill in all required fields.");
                return null; // Prevent adding the order
            }
            try {
                int id = Integer.parseInt(idField.getText());
                String nameOfPerson = nameOfPersonField.getText();
                String type = typeField.getText();
                LocalDate date = dateField.getValue();
                String status = statusComboBox.getValue();
                Double price = Double.valueOf(priceField.getText());
                String comments = commentsField.getText();
                String email = emailField.getText();
                int quantity = Integer.parseInt(quantityField.getText());
                if ("Sell".equals(type)) {
                    // Check if the product quantity is sufficient
                    boolean quantitySufficient = checkProductQuantitySufficient(comments, quantity);

                    if (!quantitySufficient) {
                        showAlert(Alert.AlertType.WARNING, "Insufficient Quantity", "The product quantity is not
sufficient to place this order.");
                        return null; // Prevent adding the order
                    }
                }
                boolean productExists = checkProductExists(comments);

                if (!productExists && "Sell".equals(type)) {
                    showAlert(Alert.AlertType.WARNING, "Product Not Found", "The product does not exist in the
database. Please check the product name and retry.");
                    return null; // Prevent adding the order
                }

                if ("Buy".equals(type)) {
                    // Increase product quantity for 'Buy' orders
                    if(!productExists){
                        Product newProduct = new Product(id, comments, (price/(double)quantity) , quantity);
                        insertDatabase(newProduct);
                    }
                    updateProductQuantity(comments, quantity,"Sell");
                } else if ("Sell".equals(type)) {
                    // Decrease product quantity for 'Sell' orders
                    updateProductQuantity(comments, quantity,"Buy");
                }
```

```
                if ("Sell".equals(type)) {
                    // Send an email with order details
                    String subject = "Order Details";
                    String body = "Order ID: " + id + "\n"
                            + "Name of Person: " + nameOfPerson + "\n"
                            + "Type: " + type + "\n"
                            + "Date of Order: " + date.toString() + "\n"
                            + "Status: " + status + "\n"
                            + "Price: " + price + "\n"
                            + "Product Name: " + comments + "\n"
                            + "Quantity: " + quantity;
                    sendEmail(email, subject, body);
                }

                Order newOrder = new Order(id, nameOfPerson, type, date, status, price, comments, quantity);
                orders.add(newOrder);

                // Insert the new order into the database
                insertOrderIntoDatabase(newOrder,email);

                return newOrder;
            } catch (NumberFormatException e) {
                // Handle parsing errors
                return null;
            }
        }
        return null;
    });

    Optional<Order> result = dialog.showAndWait();
    result.ifPresent(newOrder -> {
        ordersTable.refresh();
        showAlert(Alert.AlertType.INFORMATION, "Success", "Order added successfully.");
    });
}
```

```
private boolean checkProductQuantitySufficient(String productName, int orderedQuantity) {
    // Retrieve the product quantity from the database using productName
    int availableQuantity = retrieveProductQuantity(productName);

    // Check if the available quantity is sufficient for the ordered quantity
    return availableQuantity >= orderedQuantity;
}

// Helper method to retrieve product quantity from the database
private int retrieveProductQuantity(String productName) {
    // Query the database to retrieve the quantity of the product with the given name
    // You can add the database retrieval logic here and return the quantity
    // For example, you can connect to the database and run a SQL query.
    // This code may vary depending on your database setup and data access method.
    int availableQuantity = 0; // Initialize with a default value

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String sql = "SELECT quantity FROM products WHERE productname = ?";
            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
                preparedStatement.setString(1, productName);
                ResultSet resultSet = preparedStatement.executeQuery();

                if (resultSet.next()) {
                    availableQuantity = resultSet.getInt("quantity");
                }
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    return availableQuantity;
}
private boolean isEmptyField(TextField textField) {
    return textField.getText().trim().isEmpty();
```

```
}
private void sendEmail(String recipientEmail, String subject, String body) {
    final String username = "comardecomrade@gmail.com"; // Your Gmail email address
    final String password = "ovgnzmkumzeixbcl"; // Your Gmail password

    Properties props = new Properties();
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.host", "smtp.gmail.com");
    props.put("mail.smtp.port", "587");

    Session session = Session.getInstance(props, new javax.mail.Authenticator() {
        protected javax.mail.PasswordAuthentication getPasswordAuthentication() {
            return new javax.mail.PasswordAuthentication(username, password);
        }
    });

    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(username));
        message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(recipientEmail));
        message.setSubject(subject);
        message.setText(body);

        Transport.send(message);
        showAlert("Email sent successfully!", "Success");
    } catch (MessagingException e) {
        e.printStackTrace();
        showAlert("Failed to send the email. Check your SMTP server settings and credentials.", "Error");
    }
}

private boolean checkProductExists(String productName) {
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
         PreparedStatement preparedStatement = connection.prepareStatement("SELECT COUNT(*) FROM
products WHERE productname = ?")) {
        preparedStatement.setString(1, productName);
        ResultSet resultSet = preparedStatement.executeQuery();
        resultSet.next();
        int count = resultSet.getInt(1);
        return count > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false; // In case of an error, assume the product doesn't exist
    }
}
private void updateProductQuantity(String productName, int quantityChange, String type) {
    String updateSql;
    if("Buy".equals(type)){
        updateSql="UPDATE products SET quantity = quantity - ? WHERE productname = ?";
    }
    else
    {
        updateSql="UPDATE products SET quantity = quantity + ? WHERE productname = ?";
    }
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
         PreparedStatement preparedStatement = connection.prepareStatement(updateSql)) {
        preparedStatement.setInt(1, Math.abs(quantityChange));
        preparedStatement.setString(2, productName);
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to update product quantity.");
    }
}
private void insertOrderIntoDatabase(Order order,String email) {

    String insertSql = "INSERT INTO orders (id, person_name, order_type, order_date, order_status,
order_price, productname, email, quantity) " +
                    "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        try (PreparedStatement preparedStatement = connection.prepareStatement(insertSql)) {
            preparedStatement.setInt(1, order.getId());
            preparedStatement.setString(2, order.getNameOfPerson());
            preparedStatement.setString(3, order.getType());
            preparedStatement.setDate(4, java.sql.Date.valueOf(order.getDateOfOrder())); // Convert LocalDate to
java.sql.Date
            preparedStatement.setString(5, order.getStatus());
            preparedStatement.setDouble(6, order.getPrice());
            preparedStatement.setString(7, order.getComments());
            preparedStatement.setString(8, email);
            preparedStatement.setInt(9,order.getQuantity());


            preparedStatement.executeUpdate();
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to insert the order into the database.");
    }
}


private String getEmail(Order selectedOrder) {
    String email = null;
    String sql = "SELECT email FROM orders WHERE id = ?";
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
            preparedStatement.setInt(1, selectedOrder.getId());
            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                email = resultSet.getString("email");
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return email;
}
```

```
private void modifyOrder(Order selectedOrder) {
    if (selectedOrder == null) {
        // No order selected, show a message or handle this case as needed
        return;
    }

    // Create a dialog to edit the order
    Dialog<Order> dialog = new Dialog<>();
    dialog.initStyle(StageStyle.UTILITY);
    dialog.initModality(Modality.APPLICATION_MODAL);

    dialog.setTitle("Modify Order");
    dialog.setHeaderText("Edit order details:");

    VBox dialogContent = new VBox();
    dialogContent.setSpacing(10);

    TextField idField = new TextField(Integer.toString(selectedOrder.getId()));
    idField.setPromptText("ID");
    TextField nameOfPersonField = new TextField(selectedOrder.getNameOfPerson());
    nameOfPersonField.setPromptText("Name of Person");
    TextField typeField = new TextField(selectedOrder.getType());
    typeField.setPromptText("Type (Buy/Sell)");
    DatePicker dateField = new DatePicker(selectedOrder.getDateOfOrder());
    dateField.setPromptText("Date of Order");
    ComboBox<String> statusComboBox = new ComboBox<>(FXCollections.observableArrayList("Pending",
"Processed"));
    statusComboBox.setPromptText("Status");
    statusComboBox.setValue(selectedOrder.getStatus());
    TextField priceField = new TextField(Double.toString(selectedOrder.getPrice()));
    priceField.setPromptText("Price");
    TextField commentsField = new TextField(selectedOrder.getComments());
    commentsField.setPromptText("Product Name");
    String email = getEmail(selectedOrder);
    TextField emailField = new TextField(email);
    TextField quantityField = new TextField(Integer.toString(selectedOrder.getQuantity()));
    idField.setPromptText("Quantity");

    dialogContent.getChildren().addAll(idField, nameOfPersonField, typeField, dateField, statusComboBox,
priceField, commentsField, emailField, quantityField);
    dialog.getDialogPane().setContent(dialogContent);

    ButtonType modifyButton = new ButtonType("Modify", ButtonBar.ButtonData.OK_DONE);
    dialog.getDialogPane().getButtonTypes().addAll(modifyButton, ButtonType.CANCEL);

    dialog.setResultConverter(buttonType -> {
        if (buttonType == modifyButton) {
            try {
                int id = Integer.parseInt(idField.getText());
                String nameOfPerson = nameOfPersonField.getText();
                String newType = typeField.getText(); // Get the new order type
                LocalDate date = dateField.getValue();
                String status = statusComboBox.getValue();
                Double price = Double.valueOf(priceField.getText());
                String comments = commentsField.getText();
                String emailf = emailField.getText();
                int newQuantity = Integer.parseInt(quantityField.getText());

                int quantityChange = 0;
                if (!selectedOrder.getType().equals(newType)) {
                    // Order type changed, adjust the quantity
                    if ("Buy".equals(selectedOrder.getType())) {
                        // Changed from Buy to Sell
                        quantityChange = selectedOrder.getQuantity();
                        updateProductQuantity(comments, quantityChange, "Buy");
                    } else {
                        // Changed from Sell to Buy
                        quantityChange = selectedOrder.getQuantity();
                        updateProductQuantity(comments, quantityChange, "Sell");
                    }
                }
                boolean productexist = checkProductExists(comments,newQuantity);

                if (!productexist && "Sell".equals(newType)) {
                    showAlert(Alert.AlertType.WARNING, "Product Not Found", "The product does not exist in the database. Please check the product name and retry.");
                    return null; // Prevent adding the order
                }

                if ("Buy".equals(newType)) {
                    // Increase product quantity for 'Buy' orders
                   if(!productexist){
                        Product newProduct = new Product(id, comments, (price/(double)newQuantity) , newQuantity);
                        insertDatabase(newProduct);
                    }
                    updateProductQuantity(comments, newQuantity,"Sell");
                } else if ("Sell".equals(newType)) {
                    // Decrease product quantity for 'Sell' orders
                    updateProductQuantity(comments, newQuantity,"Buy");
                }
                selectedOrder.setId(id);
                selectedOrder.setNameOfPerson(nameOfPerson);
                selectedOrder.setType(newType); // Update the order type
                selectedOrder.setDateOfOrder(date);
                selectedOrder.setStatus(status);
                selectedOrder.setPrice(price);
                selectedOrder.setComments(comments);
                selectedOrder.setQuantity(newQuantity);

                // Update the order in the database
                updateOrderInDatabase(selectedOrder, emailf);

                return selectedOrder;
            } catch (NumberFormatException e) {
                // Handle parsing errors
                return null;
```

```
            }
        }
        return null;
    });

    Optional<Order> result = dialog.showAndWait();
    result.ifPresent(updatedOrder -> {
        ordersTable.refresh();
        showAlert(Alert.AlertType.INFORMATION, "Success", "Order modified successfully.");
    });
}
private boolean checkProductExists(String productName,int Quantity) {
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
         PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM products
WHERE productname = ?")) {
        preparedStatement.setString(1, productName);
        ResultSet resultSet = preparedStatement.executeQuery();
        resultSet.next();
        int count=resultSet.getInt("quantity")-Quantity;
        return count >= 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false; // In case of an error, assume the product doesn't exist
    }
}

private void updateOrderInDatabase(Order updatedOrder,String email) {
    String updateSql = "UPDATE orders SET person_name=?, order_type=?, order_date=?, order_status=?,
order_price=?, productname=?, email=?, quantity=? WHERE id=?";

    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        try (PreparedStatement preparedStatement = connection.prepareStatement(updateSql)) {
            preparedStatement.setString(1, updatedOrder.getNameOfPerson());
            preparedStatement.setString(2, updatedOrder.getType());
            preparedStatement.setDate(3, java.sql.Date.valueOf(updatedOrder.getDateOfOrder()));
            preparedStatement.setString(4, updatedOrder.getStatus());
            preparedStatement.setDouble(5, updatedOrder.getPrice());
            preparedStatement.setString(6, updatedOrder.getComments());
            preparedStatement.setInt(9, updatedOrder.getId());
            preparedStatement.setString(7, email);
            preparedStatement.setInt(8,updatedOrder.getQuantity());

            preparedStatement.executeUpdate();
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        showAlert(Alert.AlertType.ERROR, "Error", "Failed to update the order in the database.");
    }
}



// Define the Order class with appropriate getters and setters
public class Order {
    private final SimpleIntegerProperty id;
    private final SimpleStringProperty nameOfPerson;
    private final SimpleStringProperty type;
    private final SimpleObjectProperty<LocalDate> dateOfOrder;
    private final SimpleStringProperty status;
    private final SimpleDoubleProperty price;
    private final SimpleStringProperty comments;
    private final SimpleIntegerProperty quantity;

    public Order(int id, String nameOfPerson, String type, LocalDate dateOfOrder, String status, Double price,
String comments, int quantity) {
        this.id = new SimpleIntegerProperty(id);
        this.nameOfPerson = new SimpleStringProperty(nameOfPerson);
        this.type = new SimpleStringProperty(type);
        this.dateOfOrder = new SimpleObjectProperty<>(dateOfOrder);
        this.status = new SimpleStringProperty(status);
        this.price=new SimpleDoubleProperty(price);
        this.comments = new SimpleStringProperty(comments);
        this.quantity = new SimpleIntegerProperty(quantity);
    }
```

```
// Define getters and setters for the Order class
public int getId() {
    return id.get();
}

public String getNameOfPerson() {
    return nameOfPerson.get();
}

public String getType() {
    return type.get();
}

public LocalDate getDateOfOrder() {
    return dateOfOrder.get();
}

public String getStatus() {
    return status.get();
}
public Double getPrice() {
    return price.get();
}
public String getComments() {
    return comments.get();
}
public int getQuantity() {
    return quantity.get();
}
// Setters
public void setId(int id) {
    this.id.set(id);
}

public void setNameOfPerson(String nameOfPerson) {
        this.nameOfPerson.set(nameOfPerson);
    }

    public void setType(String type) {
        this.type.set(type);
    }

    public void setDateOfOrder(LocalDate dateOfOrder) {
        this.dateOfOrder.set(dateOfOrder);
    }

    public void setStatus(String status) {
        this.status.set(status);
    }
    public void setPrice(Double price) {
        this.price.set(price);
    }
    public void setComments(String comments) {
        this.comments.set(comments);
    }
    public void setQuantity(int quantity) {
        this.quantity.set(quantity);
    }
}
private void showProfile() {
    VBox profileContent = new VBox(10);
    profileContent.setAlignment(Pos.CENTER);

    // Create a circular clip for the profile photo
    Circle clip = new Circle(75); // Radius of the circle
    clip.setCenterX(75); // X position of the circle center
    clip.setCenterY(75); // Y position of the circle center

    // Create labels for user information
    Label nameLabel = new Label();
    Label locationLabel = new Label();
    Label phoneLabel = new Label();
    Label idLabel = new Label();

    // Create a button to choose a new profile photo
    Button choosePhotoButton = new Button("Choose Photo");

    // Load the default profile photo
    Image defaultPhoto = new Image("file:E:/5 sem/SPD/Project/download.jpeg");
    ImageView profilePhoto = new ImageView(defaultPhoto);
    profilePhoto.setFitWidth(150);
    profilePhoto.setFitHeight(150);
    profilePhoto.setClip(clip);
    VBox profilePhotoContainer = new VBox();
    profilePhotoContainer.setAlignment(Pos.CENTER);
    profilePhotoContainer.getChildren().addAll(profilePhoto, choosePhotoButton);

    // Retrieve user information from the database and create an Admin object
    Admin admin = fetchAdminDetailsFromDatabase();

    nameLabel.setText("Name: " + admin.getName());
    locationLabel.setText("Location: " + admin.getLocation());
    phoneLabel.setText("Phone: " + admin.getPhone());
    idLabel.setText("ID: " + admin.getId());

    // Add the action to choose a new profile photo
    choosePhotoButton.setOnAction((var e) -> {
        FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Image Files", "*.jpg", "*.png",
"*.jpeg"));

        // Show the file dialog and get the selected file
        File selectedFile = fileChooser.showOpenDialog(primaryStage);

        if (selectedFile != null) {
            Image newProfilePhoto = new Image(selectedFile.toURI().toString());
            profilePhoto.setImage(newProfilePhoto);
            profilePhoto.setClip(clip);
        }
    });

    // Add user information and photo container to the profile content
    profileContent.getChildren().addAll(profilePhotoContainer, nameLabel, idLabel, locationLabel, phoneLabel);

    // Clear the existing content and add the profile content
    contentArea.getChildren().clear();
    contentArea.getChildren().add(profileContent);
}

// Method to retrieve user information from the database and create an Admin object
private Admin fetchAdminDetailsFromDatabase() {
    try (Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD)) {
        if (connection != null) {
            String selectUserSql = "SELECT * FROM users WHERE username = ?";

            try (PreparedStatement preparedStatement = connection.prepareStatement(selectUserSql)) {
                preparedStatement.setString(1, username);
                ResultSet resultSet = preparedStatement.executeQuery();

                if (resultSet.next()) {
                    // Retrieve user information from the database
                    String name = resultSet.getString("name");
                    String location = resultSet.getString("location");
                    String phone = resultSet.getString("phone");
                    String id = resultSet.getString("id");

                    return new Admin(name, location, phone, id);
                }
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
```

```
        // Return an empty Admin object if no data was retrieved
        return new Admin("", "", "", "");
    }

    public class Admin {
        private final String name;
        private final String location;
        private final String phone;
        private final String id;

        public Admin(String name, String location, String phone, String id) {
            this.name = name;
            this.location = location;
            this.phone = phone;
            this.id = id;
        }

        // Getters for the profile details
        public String getName() {
            return name;
        }

        public String getLocation() {
            return location;
        }

        public String getPhone() {
            return phone;
        }

        public String getId() {
            return id;    }}}
```

CONCLUSION

An effective inventory management system is a vital component of any business, regardless of its size or industry. It plays a crucial role in optimizing operations, tracking the entire system, and ensuring customer satisfaction. A well-designed inventory management system provides several benefits, including cost control, Increased efficiency, enhanced customer satisfaction, reduced errors, better supplier relationships, sending email notification and restock notification.