**Final Report: KU Grocery Shop Application**

**1. Introduction**

In response to the increasing demand for online grocery shopping, the KU Grocery Shop Application aims to provide a seamless and convenient platform for customers to purchase groceries from the comfort of their homes. This report outlines the design and implementation of the Java-based application developed for the ECE230 course at Khalifa University.

**2. Objective**

The primary objective of this project is to develop a robust and user-friendly application that caters to the specific needs of KU Grocery. The application must facilitate efficient inventory management, provide multiple levels of authentication for administrators and cashiers, and offer a smooth shopping experience for customers. Additionally, the project aims to apply object-oriented programming concepts learned in the course, including inheritance, interfaces, polymorphism, and data persistence.

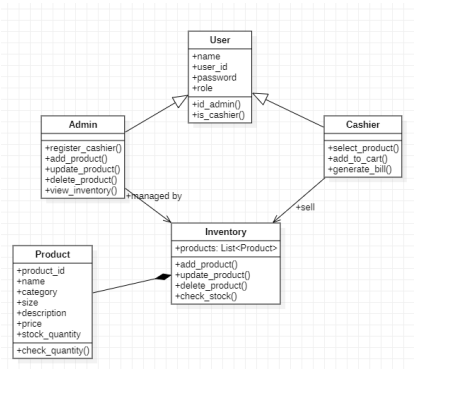
**3. Description of the Problem**

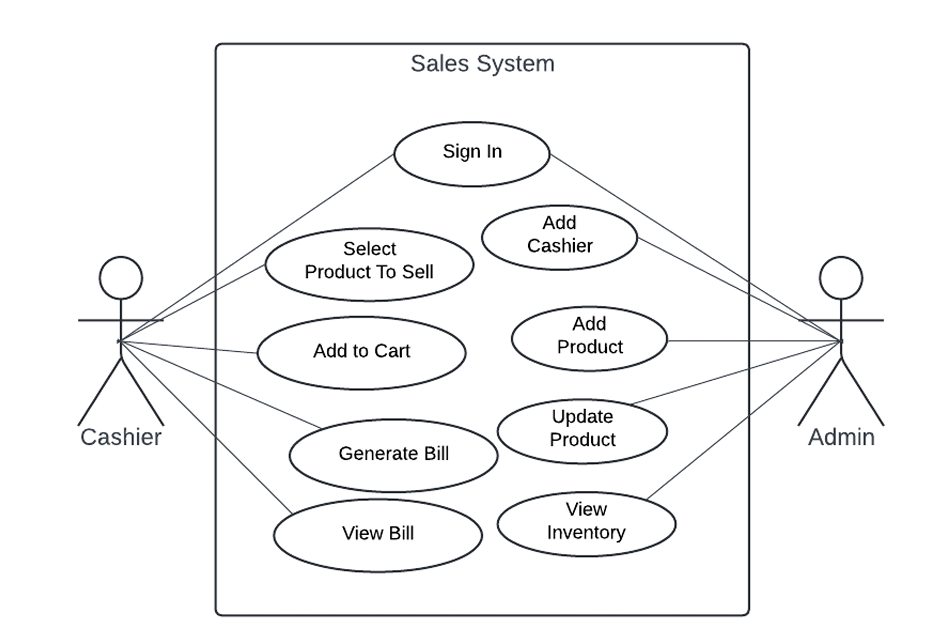
With the growing popularity of online grocery shopping, KU Grocery recognizes the need for a sophisticated application that can handle various tasks efficiently. The key challenges addressed by the KU Grocery Shop Application include:

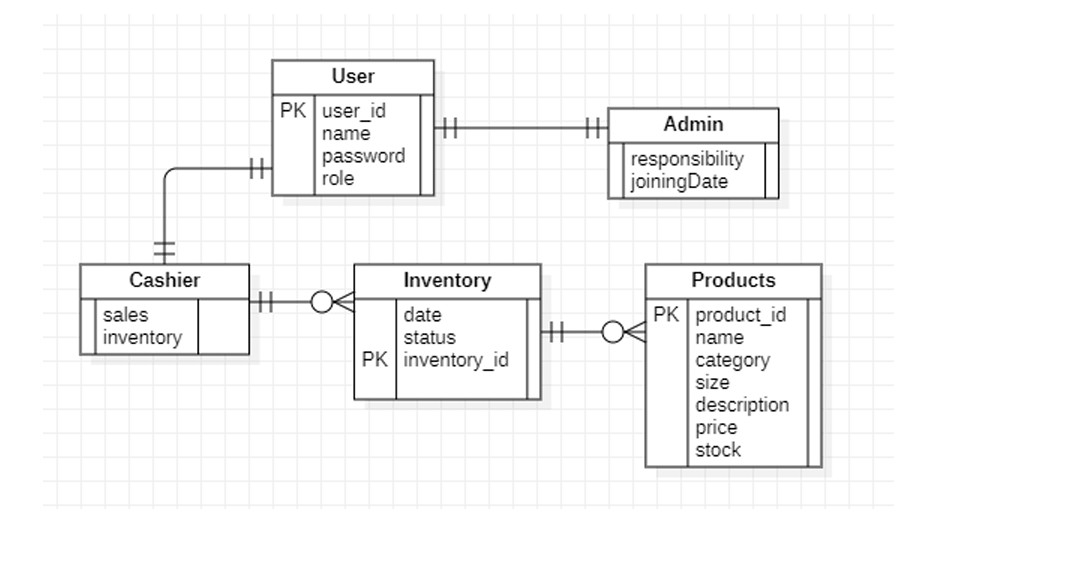
* **Authentication:** Implementing two levels of authentication for administrators and cashiers to ensure secure access to the system.
* **Inventory Management:** Developing functionalities for adding, updating, and deleting products, as well as tracking inventory levels and generating reports.
* **User Interface:** Designing an intuitive and visually appealing interface for administrators and cashiers to navigate seamlessly.
* **Data Persistence:** Ensuring that user and product information is stored persistently in a database or files to maintain data integrity and facilitate future transactions.
* **Functionalities:** Incorporating essential features such as product categorization, item selection, quantity management, and bill generation to streamline the shopping process for customers.

By addressing these challenges, the KU Grocery Shop Application aims to enhance operational efficiency, improve customer satisfaction, and capitalize on the growing trend of online grocery shopping.

1. **Class Diagram:**

****





## 5.explanation of each class and method.

### User Class:

* **Attributes**:
  + name: Represents the name of the user.
  + userId: Represents the unique identifier for the user.
  + password: Represents the password associated with the user’s account.
* **Methods**:
  + getName(): Returns the name of the user.
  + setName(String name): Sets the name of the user.
  + getUserId(): Returns the user ID.
  + setUserId(String userId): Sets the user ID.
  + getPassword(): Returns the user’s password.
  + setPassword(String password): Sets the user’s password.

### Admin Class:

* **Methods**:
  + signUpCashier(Cashier cashier): Allows the admin to sign up a new cashier by taking a Cashier object as a parameter.

### Cashier Class:

* **Methods**:
  + login(): Represents the login functionality for a cashier.
  + logout(): Represents the logout functionality for a cashier.
  + updateProductComboBox(): Updates the product combo box based on the selected category.
  + updateSizeAndProductId(): Updates the size combo box and product ID field based on the selected product.
  + loadProductData(): Loads product data from a CSV file into the product list.
  + updateProductDetails(): Updates the product details area based on the selected product.
  + addProductToCart(): Adds a selected product to the cart with the specified quantity.
  + generateBill(): Generates a bill for the items in the cart, including total price, VAT, and final amount.

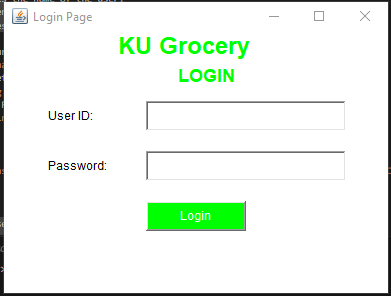
### Product Class:

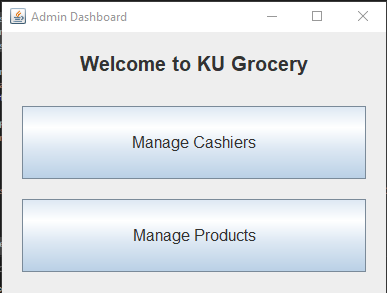
* **Attributes**:
  + name: Represents the name of the product.
  + productId: Represents the unique identifier for the product.
  + category: Represents the category of the product.
  + size: Represents the size of the product.
  + description: Represents the description of the product.
  + price: Represents the price of the product.
  + stockQuantity: Represents the quantity of the product in stock.
* **Methods**:
  + getName(): Returns the name of the product.
  + setName(String name): Sets the name of the product.
  + getProductId(): Returns the product ID.
  + setProductId(String productId): Sets the product ID.
  + getCategory(): Returns the category of the product.
  + setCategory(String category): Sets the category of the product.
  + getSize(): Returns the size of the product.
  + setSize(String size): Sets the size of the product.
  + getDescription(): Returns the description of the product.
  + setDescription(String description): Sets the description of the product.
  + getPrice(): Returns the price of the product.
  + setPrice(double price): Sets the price of the product.
  + getStockQuantity(): Returns the stock quantity of the product.
  + setStockQuantity(int stockQuantity): Sets the stock quantity of the product.
  + saveToFile(String filename): Saves the product details to a file.
  + updateProduct(String filename, String productId, Product updatedProduct): Updates an existing product in the file.

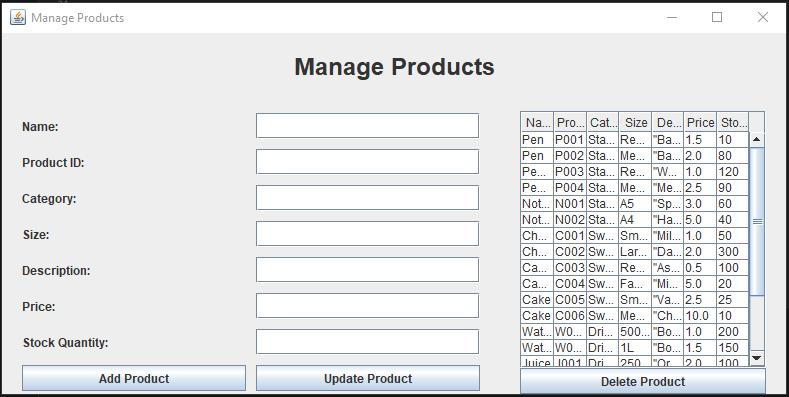
These classes and methods form the backbone of the grocery shop application, providing functionalities such as user management, product management, and authentication. Each class encapsulates related attributes and behaviors, promoting modularity and maintainability in the codebase.

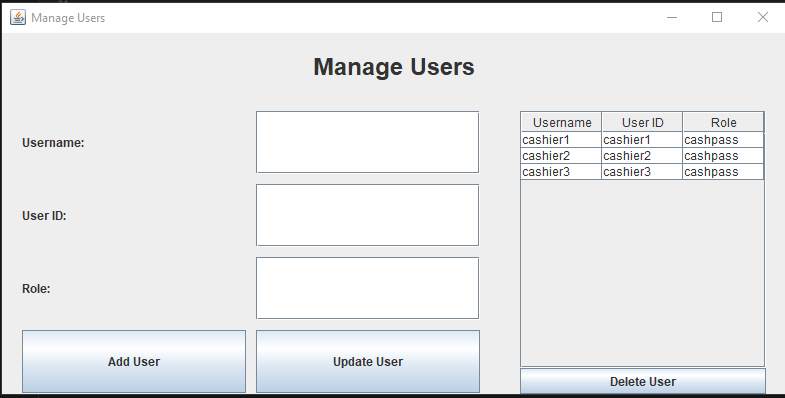
1. **Admin Side ScreenShort :**

**AdminSide**

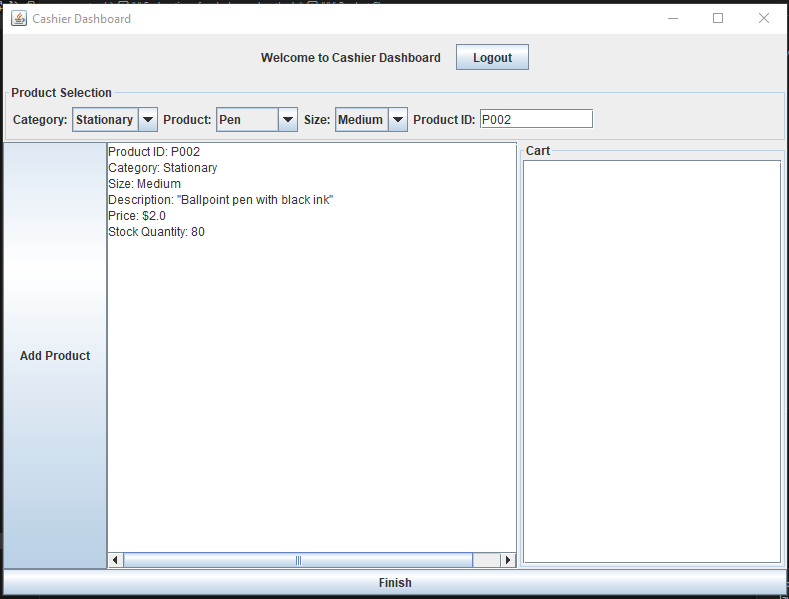
****

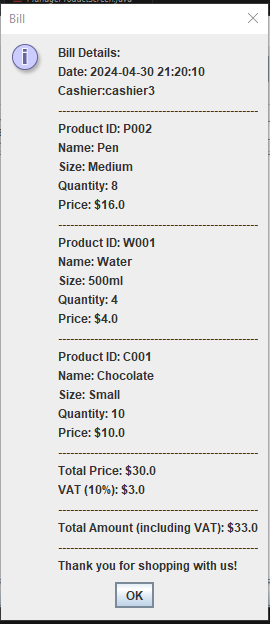
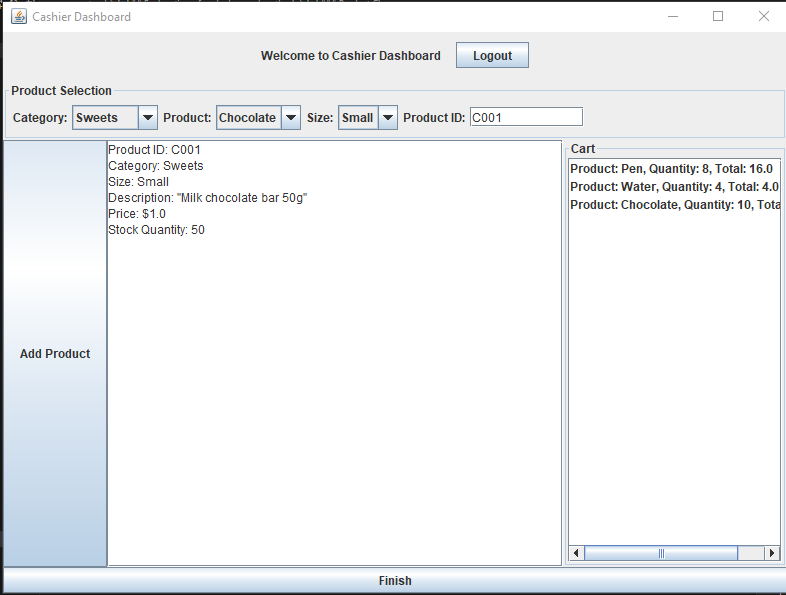
****

****

****

**User Side**

****

****

**Code:**

**User.java**

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

import javax.swing.JOptionPane;

public class User {

    private String name;

    private String userId;

    private String password;

    public User(String name, String userId, String password) {

        this.name = name;

        this.userId = userId;

        this.password = password;

    }

    // Getters and Setters for User attributes

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getUserId() {

        return userId;

    }

    public void setUserId(String userId) {

        this.userId = userId;

    }

    public String getPassword() {

        return password;

    }

    public void setPassword(String password) {

        this.password = password;

    }

    // Method to save user details to a CSV file

    public void saveToFile(String filename) {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, true))) {

            String userLine = String.format("%s,%s,%s\n", name, userId, password);

            writer.write(userLine);

            JOptionPane.showMessageDialog(null, "User details saved to file: " + filename);

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error occurred while saving user details to file: " + e.getMessage(),

                    "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    // Method to update an existing user in the CSV file

    public static void updateUser(String filename, String userId, User updatedUser) {

        List<User> users = new ArrayList<>();

        // Read existing users from the CSV file

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

            String line;

            while ((line = reader.readLine()) != null) {

                String[] parts = line.split(",");

                if (parts.length == 3) {

                    String name = parts[0];

                    String existingUserId = parts[1];

                    String password = parts[2];

                    // Create User object from CSV data

                    User user = new User(name, existingUserId, password);

                    users.add(user);

                }

            }

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error occurred while reading user details from file: " + e.getMessage(),

                    "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Update the user with the specified userId

        boolean found = false;

        for (User user : users) {

            if (user.getUserId().equals(userId)) {

                // Update user attributes

                user.setName(updatedUser.getName());

                user.setPassword(updatedUser.getPassword());

                found = true;

                break;

            }

        }

        if (!found) {

            JOptionPane.showMessageDialog(null, "User with userId '" + userId + "' not found.",

                    "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Write updated users back to the CSV file

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {

            for (User user : users) {

                String userLine = String.format("%s,%s,%s\n", user.getName(), user.getUserId(), user.getPassword());

                writer.write(userLine);

            }

            JOptionPane.showMessageDialog(null, "User with userId '" + userId + "' updated successfully.");

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error occurred while writing updated user details to file: " + e.getMessage(),

                    "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    // Example usage:

    public static void main(String[] args) {

        try (Scanner scanner = new Scanner(System.in)) {

            System.out.println("Enter your username: ");

            String username = scanner.nextLine();

            System.out.println("Enter your userID: ");

            String userId = scanner.nextLine();

            System.out.println("Enter your password: ");

            String password = scanner.nextLine();

            User newUser = new User(username,userId,password);

            newUser.saveToFile("users.csv");

        }

    }

}

**Product.java:**

import java.io.\*;

import java.util.\*;

import javax.swing.JOptionPane;

public class Product {

    private String name;

    private String productId;

    private String category;

    private String size;

    private String description;

    private double price;

    private int stockQuantity;

    public Product(String name, String productId, String category, String size, String description, double price, int stockQuantity) {

        this.name = name;

        this.productId = productId;

        this.category = category;

        this.size = size;

        this.description = description;

        this.price = price;

        this.stockQuantity = stockQuantity;

    }

    //add getter and

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getProductId() {

        return productId;

    }

    public void setProductId(String productId) {

        this.productId = productId;

    }

    public String getCategory() {

        return category;

    }

    public void setCategory(String category) {

        this.category = category;

    }

    public String getSize() {

        return size;

    }

    public void setSize(String size) {

        this.size = size;

    }

    public String getDescription() {

        return description;

    }

    public void setDescription(String description) {

        this.description = description;

    }

    public double getPrice() {

        return price;

    }

    public void setPrice(double price) {

        this.price = price;

    }

    public int getStockQuantity() {

        return stockQuantity;

    }

    public void setStockQuantity(int stockQuantity) {

        this.stockQuantity = stockQuantity;

    }

    public void saveToFile(String filename) {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, true))) {

            String productLine = String.format("%s,%s,%s,%s,%s,%.2f,%d\n",

                    name, productId, category, size, description, price, stockQuantity);

            writer.write(productLine);

            JOptionPane.showMessageDialog(null, "Product details saved to file: " + filename);

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error occurred while saving product details to file: " + e.getMessage(),

                    "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    // Method to update an existing product in the CSV file

    public static void updateProduct(String filename, String productId, Product updatedProduct) {

        List<Product> products = new ArrayList<>();

        // Read existing products from the CSV file

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

            String line;

            while ((line = reader.readLine()) != null) {

                String[] parts = line.split(",");

                if (parts.length == 7) {

                    String name = parts[0];

                    String existingProductId = parts[1];

                    String category = parts[2];

                    String size = parts[3];

                    String description = parts[4];

                    double price = Double.parseDouble(parts[5]);

                    int stockQuantity = Integer.parseInt(parts[6]);

                    // Create Product object from CSV data

                    Product product = new Product(name, existingProductId, category, size, description, price, stockQuantity);

                    products.add(product);

                }

            }

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error occurred while reading product details from file: " + e.getMessage(),

                    "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Update the product with the specified productId

        boolean found = false;

        for (Product product : products) {

            if (product.getProductId().equals(productId)) {

                // Update product attributes

                product.setName(updatedProduct.getName());

                product.setCategory(updatedProduct.getCategory());

                product.setSize(updatedProduct.getSize());

                product.setDescription(updatedProduct.getDescription());

                product.setPrice(updatedProduct.getPrice());

                product.setStockQuantity(updatedProduct.getStockQuantity());

                found = true;

                break;

            }

        }

        if (!found) {

            JOptionPane.showMessageDialog(null, "Product with productId '" + productId + "' not found.",

                    "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Write updated products back to the CSV file

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {

            for (Product product : products) {

                String productLine = String.format("%s,%s,%s,%s,%s,%.2f,%d\n",

                        product.getName(), product.getProductId(), product.getCategory(),

                        product.getSize(), product.getDescription(), product.getPrice(), product.getStockQuantity());

                writer.write(productLine);

            }

            JOptionPane.showMessageDialog(null, "Product with productId '" + productId + "' updated successfully.");

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error occurred while writing updated product details to file: " + e.getMessage(),

                    "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    // Example usage:

    public static void main(String[] args) {

        try (Scanner scanner = new Scanner(System.in)) {

            String filename = "products.csv";

            System.out.println("How many products do you want to add?");

            int count = Integer.parseInt(scanner.nextLine());

            for (int i = 0; i < count; i++) {

                System.out.println("Adding product " + (i + 1));

                System.out.println("Enter product name:");

                String name = scanner.nextLine();

                System.out.println("Enter product ID:");

                String productId = scanner.nextLine();

                System.out.println("Choose category (1. Stationary, 2. Sweets, 3. Drinks):");

                String category = getCategoryFromChoice(scanner.nextLine());

                System.out.println("Enter size:");

                String size = scanner.nextLine();

                System.out.println("Enter description:");

                String description = scanner.nextLine();

                System.out.println("Enter price:");

                double price = Double.parseDouble(scanner.nextLine());

                System.out.println("Enter quantity:");

                int quantity = Integer.parseInt(scanner.nextLine());

                Product product = new Product(name, productId, category, size, description, price, quantity);

                product.saveToFile(filename);

            }

        }

    }

    private static String getCategoryFromChoice(String choice) {

        switch (choice) {

            case "1" -> {

                return "Stationary";

            }

            case "2" -> {

                return "Sweets";

            }

            case "3" -> {

                return "Drinks";

            }

            default -> {

                System.out.println("Invalid choice, defaulting to Stationary");

                return "Stationary";

            }

        }

    }

}

**LoginPage.java:**

import java.awt.\*;

import java.awt.event.\*;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class LoginPage extends Frame {

    private Label userIdLabel, passwordLabel;

    private TextField userIdField;

    private TextField passwordField;

    private Button loginButton;

    public LoginPage() {

        setTitle("Login Page");

        setSize(400, 300);

        setLayout(null); // Using absolute positioning

        // Main heading for KU Grocery

        Label mainHeading = new Label("KU Grocery");

        mainHeading.setFont(new Font("Arial", Font.BOLD, 24));

        mainHeading.setForeground(Color.GREEN);

        mainHeading.setBounds(120, 30, 200, 30);

        add(mainHeading);

        // Subheading for LOGIN

        Label loginHeading = new Label("LOGIN");

        loginHeading.setFont(new Font("Arial", Font.BOLD, 18));

        loginHeading.setForeground(Color.GREEN);

        loginHeading.setBounds(180, 60, 80, 30);

        add(loginHeading);

        // Create labels

        userIdLabel = new Label("User ID:");

        userIdLabel.setBounds(50, 100, 80, 30);

        add(userIdLabel);

        passwordLabel = new Label("Password:");

        passwordLabel.setBounds(50, 150, 80, 30);

        add(passwordLabel);

        // Create text fields

        userIdField = new TextField();

        userIdField.setBounds(150, 100, 200, 30);

        add(userIdField);

        passwordField = new TextField();

        passwordField.setBounds(150, 150, 200, 30);

        passwordField.setEchoChar('\*');

        add(passwordField);

        // Create login button

        loginButton = new Button("Login");

        loginButton.setBounds(150, 200, 100, 30);

        loginButton.setBackground(Color.GREEN);

        loginButton.setForeground(Color.WHITE);

        loginButton.addActionListener((ActionEvent e) -> {

            String userId = userIdField.getText();

            String password = passwordField.getText();

            if (userId.equals("Salim") && password.equals("Salim1234")) {

                // Admin login

                new AdminDashboard().setVisible(true);

                setVisible(false);

                return;

            }

            boolean authenticated = authenticateUser(userId, password);

            if (authenticated) {

                // Cashier login

                new CashierDashboard(LoginPage.this, userId, password).setVisible(true);

                setVisible(false);

            } else {

                showMessageDialog("Invalid User ID or Password!");

            }

        });

        add(loginButton);

        setBackground(Color.WHITE);

        // Center the frame on screen

        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

        setLocation((screenSize.width - getWidth()) / 2, (screenSize.height - getHeight()) / 2);

        // Handle window close event

        addWindowListener(new WindowAdapter() {

            @Override

            public void windowClosing(WindowEvent e) {

                dispose(); // Close the window

            }

        });

        setVisible(true); // Display the frame

    }

    private boolean authenticateUser(String userId, String password) {

        String csvFile = "users.csv";

        String line;

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {

            while ((line = br.readLine()) != null) {

                String[] userData = line.split(",");

                if (userData.length == 3) {

                    String csvUserId = userData[1].trim();

                    String csvPassword = userData[2].trim();

                    if (csvUserId.equals(userId) && csvPassword.equals(password)) {

                        return true; // Match found, authentication successful

                    }

                }

            }

        } catch (IOException ex) {

            // Handle file IO exception

        }

        return false; // No matching user found or error occurred

    }

    private void showMessageDialog(String message) {

        // Display a message dialog

        Dialog dialog = new Dialog(this, "Message", true);

        dialog.setLayout(new FlowLayout());

        dialog.setSize(300, 100);

        Label label = new Label(message);

        dialog.add(label);

        dialog.setLocationRelativeTo(this);

        Button okButton = new Button("OK");

        okButton.addActionListener((ActionEvent e) -> {

            dialog.dispose(); // Close the dialog

        });

        okButton.setBackground(Color.GREEN);

        okButton.setForeground(Color.WHITE);

        dialog.add(okButton);

        dialog.setVisible(true);

    }

    public static void main(String[] args) {

        new LoginPage();

    }

}

**AdminDashboard.java**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class AdminDashboard extends JFrame {

    public AdminDashboard() {

        setTitle("Admin Dashboard");

        setSize(400, 300);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null);

        JLabel welcomeLabel = new JLabel("Welcome to KU Grocery");

        welcomeLabel.setFont(new Font("Arial", Font.BOLD, 20));

        welcomeLabel.setHorizontalAlignment(SwingConstants.CENTER);

        welcomeLabel.setBorder(BorderFactory.createEmptyBorder(20, 0, 30, 0));

        add(welcomeLabel, BorderLayout.NORTH);

        JPanel buttonPanel = new JPanel(new GridLayout(2, 1, 0, 20));

        buttonPanel.setBorder(BorderFactory.createEmptyBorder(0, 20, 20, 20));

        JButton manageCashiersButton = new JButton("Manage Cashiers");

        manageCashiersButton.setFont(new Font("Arial", Font.PLAIN, 16));

        manageCashiersButton.addActionListener((ActionEvent e) -> {

            new ManageUserScreen().setVisible(true);

        });

        buttonPanel.add(manageCashiersButton);

        JButton manageProductsButton = new JButton("Manage Products");

        manageProductsButton.setFont(new Font("Arial", Font.PLAIN, 16));

        manageProductsButton.addActionListener((ActionEvent e) -> {

            new ManageProductScreen().setVisible(true);

        });

        buttonPanel.add(manageProductsButton);

        add(buttonPanel, BorderLayout.CENTER);

        setVisible(true);

    }

    public static void main(String[] args) {

        new AdminDashboard();

    }

}

**CashierDashboard.java:**

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import javax.swing.\*;

public class CashierDashboard extends JFrame {

    private final JComboBox<String> categoryComboBox;

    private final JComboBox<String> productComboBox;

    private final JComboBox<String> sizeComboBox;

    private final JTextField productIdField;

    private final JTextArea productDetailsArea;

    private final JSpinner quantitySpinner;

    private final LoginPage loginPage;

    private final String userId;

    private final List<Product> productList;

    private final List<Product> cart;

    private final DefaultListModel<String> cartListModel;

    //  private JScrollPane cartScrollPane;

    public CashierDashboard(LoginPage loginPage, String userId, String password) {

        this.userId = userId;

        this.loginPage = loginPage;

        setTitle("Cashier Dashboard");

        setSize(800, 600);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null);

        JPanel mainPanel = new JPanel(new BorderLayout());

        // Product Selection Panel

        JPanel selectionPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

        selectionPanel.setBorder(BorderFactory.createTitledBorder("Product Selection"));

        // Category ComboBox

        JLabel categoryLabel = new JLabel("Category:");

        categoryComboBox = new JComboBox<>();

        categoryComboBox.addItem("Stationary");

        categoryComboBox.addItem("Sweets");

        categoryComboBox.addItem("Drinks");

        categoryComboBox.addActionListener((ActionEvent e) -> {

            updateProductComboBox();

        });

        selectionPanel.add(categoryLabel);

        selectionPanel.add(categoryComboBox);

        // Product ComboBox

        JLabel productLabel = new JLabel("Product:");

        productComboBox = new JComboBox<>();

        productComboBox.addActionListener((ActionEvent e) -> {

            updateProductDetails();

        });

        selectionPanel.add(productLabel);

        selectionPanel.add(productComboBox);

        // Size ComboBox

        JLabel sizeLabel = new JLabel("Size:");

        sizeComboBox = new JComboBox<>();

        selectionPanel.add(sizeLabel);

        selectionPanel.add(sizeComboBox);

        // Product ID Field

        JLabel productIdLabel = new JLabel("Product ID:");

        productIdField = new JTextField(10);

        selectionPanel.add(productIdLabel);

        selectionPanel.add(productIdField);

        // Product Details Area

        productDetailsArea = new JTextArea(10, 40);

        productDetailsArea.setEditable(false);

        JScrollPane detailsScrollPane = new JScrollPane(productDetailsArea);

        // Quantity Spinner

        JLabel quantityLabel = new JLabel("Quantity:");

        SpinnerModel quantityModel = new SpinnerNumberModel(1, 1, 100, 1);

        quantitySpinner = new JSpinner(quantityModel);

        // Add Product Button

        JButton addButton = new JButton("Add Product");

        addButton.addActionListener((ActionEvent e) -> {

            addProductToCart();

        });

        // Cart Panel

        JPanel cartPanel = new JPanel(new BorderLayout());

        cartPanel.setBorder(BorderFactory.createTitledBorder("Cart"));

        cartListModel = new DefaultListModel<>();

        JList<String> cartList = new JList<>(cartListModel);

        JScrollPane cartScrollPane = new JScrollPane(cartList);

        cartPanel.add(cartScrollPane, BorderLayout.CENTER);

        // DefaultListModel<String> cartListModel = new DefaultListModel<>();

        // JList<String> cartList = new JList<>(cartListModel);

        // JScrollPane cartScrollPane = new JScrollPane(cartList);

        // cartPanel.add(cartScrollPane, BorderLayout.CENTER);

        // Inside the constructor

        // cartScrollPane = new JScrollPane();

        // Finish Button

        JButton finishButton = new JButton("Finish");

        finishButton.addActionListener((ActionEvent e) -> {

            generateBill();

        });

        // North panel

        JPanel northPanel = new JPanel();

        northPanel.setPreferredSize(new Dimension(800, 50)); // Expl

        JPanel leftPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

        JLabel welcomeLabel = new JLabel("Welcome to Cashier Dashboard");

        leftPanel.add(welcomeLabel);

        JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

        JButton logoutButton = new JButton("Logout");

        logoutButton.addActionListener((ActionEvent e) -> {

            logout();

        });

        rightPanel.add(logoutButton);

        // Add sub-panels to northPanel

        northPanel.add(leftPanel, BorderLayout.WEST);

        northPanel.add(rightPanel, BorderLayout.EAST);

        JPanel combinedNorthPanel = new JPanel();

        combinedNorthPanel.setLayout(new BoxLayout(combinedNorthPanel, BoxLayout.Y\_AXIS));

        combinedNorthPanel.add(northPanel);

        combinedNorthPanel.add(selectionPanel);

        // Add Components to Main Panel

        mainPanel.add(detailsScrollPane, BorderLayout.CENTER);

        mainPanel.add(quantityLabel, BorderLayout.WEST);

        mainPanel.add(quantitySpinner, BorderLayout.WEST);

        mainPanel.add(addButton, BorderLayout.WEST);

        mainPanel.add(cartPanel, BorderLayout.EAST);

        mainPanel.add(finishButton, BorderLayout.SOUTH);

        mainPanel.add(combinedNorthPanel, BorderLayout.NORTH);

        add(mainPanel);

        validate();

        repaint();

        setVisible(true);

        // Initialize product list and cart

        productList = new ArrayList<>();

        cart = new ArrayList<>();

        productComboBox.addActionListener(e -> {

            updateProductDetails();  // Update details based on selection

            updateSizeAndProductId();  // New method to update size and product ID

        });

        loadProductData();

    }

    private void logout() {

        this.setVisible(false); // Hide the CashierDashboard

        loginPage.setVisible(true); // Show the LoginPage again

    }

    private void updateProductComboBox() {

        String selectedCategory = (String) categoryComboBox.getSelectedItem();

        productComboBox.removeAllItems();

        sizeComboBox.removeAllItems();

        productIdField.setText("");

        for (Product product : productList) {

            if (product.getCategory().equals(selectedCategory)) {

                productComboBox.addItem(product.getName());

            }

        }

    }

    private void updateSizeAndProductId() {

        sizeComboBox.removeAllItems();

        String selectedProductName = (String) productComboBox.getSelectedItem();

        for (Product product : productList) {

            if (product.getName().equals(selectedProductName)) {

                sizeComboBox.addItem(product.getSize());  // Populate size combo box

                productIdField.setText(product.getProductId());  // Set product ID once for selected product

                break;  // Assuming only one product with a unique name and size, break after found

            }

        }

    }

    private void loadProductData() {

        String csvFile = "products.csv";

        String line;

        productList.clear(); // Clear existing product list

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {

            // Skip the header line if your CSV has headers

            br.readLine();

            while ((line = br.readLine()) != null) {

                String[] data = line.split(",");

                // Check if the data has the expected number of elements

                if (data.length == 7) {

                    String name = data[0];

                    String productId = data[1];

                    String category = data[2];

                    String size = data[3];

                    String description = data[4];

                    double price = Double.parseDouble(data[5]);

                    int stockQuantity = Integer.parseInt(data[6]);

                    // Add product to the list

                    productList.add(new Product(name, productId, category, size, description, price, stockQuantity));

                } else {

                    // Log or display a message for unexpected data

                    System.err.println("Skipping invalid data: " + line);

                }

            }

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error loading product data from file: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

        // After loading, you might need to update your product combo boxes

        updateProductComboBox();

        if (!productList.isEmpty()) {

            productComboBox.setSelectedIndex(0);

            updateProductDetails();

        }

    }

    private void updateProductDetails() {

        String selectedProductName = (String) productComboBox.getSelectedItem();

        if (selectedProductName != null) {

            for (Product product : productList) {

                if (product.getName().equals(selectedProductName)) {

                    String details = buildProductDetails(product);

                    productDetailsArea.setText(details);

                    return;

                }

            }

        }

        productDetailsArea.setText("Product not found.");

    }

    private String buildProductDetails(Product product) {

        return "Product ID: " + product.getProductId() + "\n"

                + "Category: " + product.getCategory() + "\n"

                + "Size: " + product.getSize() + "\n"

                + "Description: " + product.getDescription() + "\n"

                + "Price: $" + product.getPrice() + "\n"

                + "Stock Quantity: " + product.getStockQuantity();

    }

    private void addProductToCart() {

        String selectedProductName = (String) productComboBox.getSelectedItem();

        String selectedCategory = (String) categoryComboBox.getSelectedItem();

        String selectedSize = (String) sizeComboBox.getSelectedItem();

        String selectedProductId = productIdField.getText().trim();

        int quantity = (int) quantitySpinner.getValue();

        // Check if the product already exists in the cart

        for (Product cartProduct : cart) {

            if (cartProduct.getCategory().equals(selectedCategory)

                    && cartProduct.getName().equals(selectedProductName)

                    && cartProduct.getSize().equals(selectedSize)

                    && cartProduct.getProductId().equals(selectedProductId)) {

                // Update the quantity of the existing product in the cart

                int newQuantity = cartProduct.getStockQuantity() + quantity;

                cartProduct.setStockQuantity(newQuantity);

                // Update the quantity in the cart list model

                int index = -1;

                for (int i = 0; i < cartListModel.size(); i++) {

                    if (cartListModel.getElementAt(i).startsWith("Product: " + cartProduct.getName())) {

                        index = i;

                        break;

                    }

                }

                // int index = cartListModel.indexOf("Product: " + cartProduct.getName());

                if (index != -1) {

                    cartListModel.set(index, "Product: " + cartProduct.getName() + ", Quantity: " + cartProduct.getStockQuantity() + ", Total: " + cartProduct.getPrice() \* cartProduct.getStockQuantity());

                }

                // Update the product details area

                updateProductDetails();

                return;

            }

        }

        // If the product does not exist in the cart, add it to the cart list model

        for (Product product : productList) {

            if (product.getCategory().equals(selectedCategory)

                    && product.getName().equals(selectedProductName)

                    && product.getSize().equals(selectedSize)

                    && product.getProductId().equals(selectedProductId)) {

                Product cartProduct = new Product(product.getName(), product.getProductId(), product.getCategory(),

                        product.getSize(), product.getDescription(), product.getPrice(), quantity);

                cart.add(cartProduct);

                cartListModel.addElement("Product: " + cartProduct.getName() + ", Quantity: " + cartProduct.getStockQuantity() + ", Total: " + cartProduct.getPrice() \* quantity);

                return;

            }

        }

        // If the product is not found in the productList, display an error message

        productDetailsArea.setText("Product not found.");

    }

    private void generateBill() {

        double totalAmount = 0;

        StringBuilder billDetails = new StringBuilder();

        billDetails.append("Bill Details:\n");

        billDetails.append("Date: ").append(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date())).append("\n");

        billDetails.append("Cashier:").append(userId).append("\n");

        billDetails.append("--------------------------------------------------\n");

        for (Product product : cart) {

            billDetails.append("Product ID: ").append(product.getProductId()).append("\n");

            billDetails.append("Name: ").append(product.getName()).append("\n");

            billDetails.append("Size: ").append(product.getSize()).append("\n");

            billDetails.append("Quantity: ").append(product.getStockQuantity()).append("\n");

            double totalPrice = product.getPrice() \* product.getStockQuantity();

            billDetails.append("Price: $").append(totalPrice).append("\n");

            billDetails.append("--------------------------------------------------\n");

            totalAmount += totalPrice;

        }

        double vat = totalAmount \* 0.1; // Assuming 10% VAT

        double finalAmount = totalAmount + vat;

        billDetails.append("Total Price: $").append(totalAmount).append("\n");

        billDetails.append("VAT (10%): $").append(vat).append("\n");

        billDetails.append("--------------------------------------------------\n");

        billDetails.append("Total Amount (including VAT): $").append(finalAmount).append("\n");

        billDetails.append("--------------------------------------------------\n");

        billDetails.append("Thank you for shopping with us!");

        JOptionPane.showMessageDialog(this, billDetails.toString(), "Bill", JOptionPane.INFORMATION\_MESSAGE);

        // Clear cart after generating bill

        cart.clear();

        cartListModel.clear(); // Use direct reference to clear the model

    }

}

**ManageUser.java:**

import javax.swing.\*;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.\*;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class ManageUserScreen extends JFrame {

    private JTextField usernameField, userIdField, roleField;

    private JButton addButton, updateButton, deleteButton;

    private JTable userTable;

    private DefaultTableModel tableModel;

    public ManageUserScreen() {

        setTitle("Manage Users");

        setSize(800, 400);

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the frame on the screen

        // Heading label for Manage Users

        JLabel headingLabel = new JLabel("Manage Users");

        headingLabel.setFont(new Font("Arial", Font.BOLD, 24));

        headingLabel.setHorizontalAlignment(SwingConstants.CENTER);

        headingLabel.setBorder(BorderFactory.createEmptyBorder(20, 0, 30, 0));

        add(headingLabel, BorderLayout.NORTH);

        // Panel to hold the user form and buttons

        JPanel leftPanel = new JPanel();

        leftPanel.setLayout(new GridLayout(4, 2, 10, 10)); // 4 rows, 2 columns with gaps

        leftPanel.setBorder(BorderFactory.createEmptyBorder(0, 20, 0, 20));

        // User form fields

        usernameField = new JTextField(20);

        userIdField = new JTextField(20);

        roleField = new JTextField(20);

        // Add form fields to the panel

        leftPanel.add(new JLabel("Username:"));

        leftPanel.add(usernameField);

        leftPanel.add(new JLabel("User ID:"));

        leftPanel.add(userIdField);

        leftPanel.add(new JLabel("Role:"));

        leftPanel.add(roleField);

        // Buttons for add, update, delete operations

        addButton = new JButton("Add User");

        addButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                addUser();

            }

        });

        leftPanel.add(addButton);

        updateButton = new JButton("Update User");

        updateButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                updateUser();

            }

        });

        leftPanel.add(updateButton);

        // Panel to hold the user table and delete button

        JPanel rightPanel = new JPanel(new BorderLayout());

        rightPanel.setBorder(BorderFactory.createEmptyBorder(0, 20, 0, 20));

        // Table model for user data

        tableModel = new DefaultTableModel();

        userTable = new JTable(tableModel);

        JScrollPane tableScrollPane = new JScrollPane(userTable);

        rightPanel.add(tableScrollPane, BorderLayout.CENTER);

        deleteButton = new JButton("Delete User");

        deleteButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                deleteUser();

            }

        });

        rightPanel.add(deleteButton, BorderLayout.SOUTH);

        // Add panels to the frame

        add(leftPanel, BorderLayout.WEST);

        add(rightPanel, BorderLayout.CENTER);

        // Load initial user data into the table

        loadUserData();

        // Add list selection listener to the user table

        userTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

            public void valueChanged(ListSelectionEvent event) {

                if (!event.getValueIsAdjusting()) {

                    int selectedRow = userTable.getSelectedRow();

                    if (selectedRow != -1) {

                        displaySelectedUser(selectedRow);

                    }

                }

            }

        });

        setVisible(true); // Display the frame

    }

    private void loadUserData() {

        String csvFile = "users.csv";

        String line;

        String[] headers = {"Username", "User ID", "Role"};

        // Set table headers

        tableModel.setColumnIdentifiers(headers);

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {

            while ((line = br.readLine()) != null) {

                String[] data = line.split(",");

                if (data.length == 3) {

                    tableModel.addRow(data);

                }

            }

        } catch (IOException e) {

            e.printStackTrace();

            JOptionPane.showMessageDialog(null, "Error loading user data from file: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private void displaySelectedUser(int rowIndex) {

        usernameField.setText(tableModel.getValueAt(rowIndex, 0).toString());

        userIdField.setText(tableModel.getValueAt(rowIndex, 1).toString());

        roleField.setText(tableModel.getValueAt(rowIndex, 2).toString());

    }

    private void addUser() {

        String username = usernameField.getText();

        String userId = userIdField.getText();

        String role = roleField.getText();

        User user = new User(username, userId, role);

        user.saveToFile("users.csv");

        tableModel.addRow(new Object[]{username, userId, role});

        clearFields();

    }

        private void updateUser() {

            int selectedRow = userTable.getSelectedRow();

            if (selectedRow != -1) {

                String username = usernameField.getText();

                String userId = userIdField.getText();

                String role = roleField.getText();

                tableModel.setValueAt(username, selectedRow, 0);

                tableModel.setValueAt(userId, selectedRow, 1);

                tableModel.setValueAt(role, selectedRow, 2);

                try {

                    FileWriter writer = new FileWriter("users.csv");

                    for (int i = 0; i < tableModel.getRowCount(); i++) {

                        for (int j = 0; j < tableModel.getColumnCount(); j++) {

                            writer.append(tableModel.getValueAt(i, j).toString());

                            if (j != tableModel.getColumnCount() - 1) {

                                writer.append(",");

                            }

                        }

                        writer.append("\n");

                    }

                    writer.close();

                    clearFields();

                } catch (IOException ex) {

                    ex.printStackTrace();

                    JOptionPane.showMessageDialog(null, "Error updating user: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

                }

            } else {

                JOptionPane.showMessageDialog(null, "Please select a user to update.", "Error", JOptionPane.ERROR\_MESSAGE);

            }

        }

    private void deleteUser() {

        int selectedRow = userTable.getSelectedRow();

        if (selectedRow != -1) {

            tableModel.removeRow(selectedRow);

            try {

                FileWriter writer = new FileWriter("users.csv");

                for (int i = 0; i < tableModel.getRowCount(); i++) {

                    for (int j = 0; j < tableModel.getColumnCount(); j++) {

                        writer.append(tableModel.getValueAt(i, j).toString());

                        if (j != tableModel.getColumnCount() - 1) {

                            writer.append(",");

                        }

                    }

                    writer.append("\n");

                }

                writer.close();

            } catch (IOException ex) {

                ex.printStackTrace();

                JOptionPane.showMessageDialog(null, "Error deleting user: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

            }

        } else {

            JOptionPane.showMessageDialog(null, "Please select a user to delete.", "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private void clearFields() {

        usernameField.setText("");

        userIdField.setText("");

        roleField.setText("");

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                new ManageUserScreen();

            }

        });

    }

}

**ManageProduct.java:**

import java.awt.\*;

import java.awt.event.\*;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import javax.swing.\*;

import javax.swing.event.ListSelectionEvent;

import javax.swing.table.DefaultTableModel;

public class ManageProductScreen extends JFrame {

    private final JTextField nameField, productIdField, categoryField, sizeField, descriptionField, priceField, stockQuantityField;

    private final JButton addButton, updateButton, deleteButton;

    private JTable productTable;

    private final DefaultTableModel tableModel;

    public ManageProductScreen() {

        setTitle("Manage Products");

        setSize(800, 400);

        setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

        setLocationRelativeTo(null); // Center the frame on the screen

        // Heading label for Manage Products

        JLabel headingLabel = new JLabel("Manage Products");

        headingLabel.setFont(new Font("Arial", Font.BOLD, 24));

        headingLabel.setHorizontalAlignment(SwingConstants.CENTER);

        headingLabel.setBorder(BorderFactory.createEmptyBorder(20, 0, 30, 0));

        add(headingLabel, BorderLayout.NORTH);

        // Panel to hold the product form and buttons

        JPanel leftPanel = new JPanel();

        leftPanel.setLayout(new GridLayout(8, 2, 10, 10)); // 8 rows, 2 columns with gaps

        leftPanel.setBorder(BorderFactory.createEmptyBorder(0, 20, 0, 20));

        // Product form fields

        nameField = new JTextField(20);

        productIdField = new JTextField(20);

        categoryField = new JTextField(20);

        sizeField = new JTextField(20);

        descriptionField = new JTextField(20);

        priceField = new JTextField(20);

        stockQuantityField = new JTextField(20);

        // Add form fields to the panel

        leftPanel.add(new JLabel("Name:"));

        leftPanel.add(nameField);

        leftPanel.add(new JLabel("Product ID:"));

        leftPanel.add(productIdField);

        leftPanel.add(new JLabel("Category:"));

        leftPanel.add(categoryField);

        leftPanel.add(new JLabel("Size:"));

        leftPanel.add(sizeField);

        leftPanel.add(new JLabel("Description:"));

        leftPanel.add(descriptionField);

        leftPanel.add(new JLabel("Price:"));

        leftPanel.add(priceField);

        leftPanel.add(new JLabel("Stock Quantity:"));

        leftPanel.add(stockQuantityField);

        // Buttons for add, update, delete operations

        addButton = new JButton("Add Product");

        addButton.addActionListener((ActionEvent e) -> {

            addProduct();

        });

        leftPanel.add(addButton);

        updateButton = new JButton("Update Product");

        updateButton.addActionListener((ActionEvent e) -> {

            updateProduct();

        });

        leftPanel.add(updateButton);

        // Panel to hold the product table and delete button

        JPanel rightPanel = new JPanel(new BorderLayout());

        rightPanel.setBorder(BorderFactory.createEmptyBorder(0, 20, 0, 20));

        // Table model for product data

        tableModel = new DefaultTableModel();

        productTable = new JTable(tableModel);

        JScrollPane tableScrollPane = new JScrollPane(productTable);

        rightPanel.add(tableScrollPane, BorderLayout.CENTER);

        deleteButton = new JButton("Delete Product");

        deleteButton.addActionListener((ActionEvent e) -> {

            deleteProduct();

        });

        rightPanel.add(deleteButton, BorderLayout.SOUTH);

        // Add panels to the frame

        add(leftPanel, BorderLayout.WEST);

        add(rightPanel, BorderLayout.CENTER);

        // Load initial product data into the table

        loadProductData();

        // Add list selection listener to the product table

        productTable.getSelectionModel().addListSelectionListener((ListSelectionEvent event) -> {

            if (!event.getValueIsAdjusting()) {

                int selectedRow = productTable.getSelectedRow();

                if (selectedRow != -1) {

                    displaySelectedProduct(selectedRow);

                }

            }

        });

        setVisible(true); // Display the frame

    }

    private void loadProductData() {

        String csvFile = "products.csv";

        String line;

        String[] headers = {"Name", "Product ID", "Category", "Size", "Description", "Price", "Stock Quantity"};

        // Clear existing data in the table model

        tableModel.setRowCount(0); // Clear existing rows

        // Set table headers

        tableModel.setColumnIdentifiers(headers);

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {

            while ((line = br.readLine()) != null) {

                String[] data = line.split(",");

                // Check if the data has the expected number of elements

                if (data.length == headers.length) {

                    tableModel.addRow(data); // Add row to the table model

                } else {

                    // Log or display a message for unexpected data

                    System.err.println("Skipping invalid data: " + line);

                }

            }

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error loading product data from file: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private void displaySelectedProduct(int rowIndex) {

        nameField.setText(tableModel.getValueAt(rowIndex, 0).toString());

        productIdField.setText(tableModel.getValueAt(rowIndex, 1).toString());

        categoryField.setText(tableModel.getValueAt(rowIndex, 2).toString());

        sizeField.setText(tableModel.getValueAt(rowIndex, 3).toString());

        descriptionField.setText(tableModel.getValueAt(rowIndex, 4).toString());

        priceField.setText(tableModel.getValueAt(rowIndex, 5).toString());

        stockQuantityField.setText(tableModel.getValueAt(rowIndex, 6).toString());

    }

    private void addProduct() {

        String name = nameField.getText();

        String productId = productIdField.getText();

        String category = categoryField.getText();

        String size = sizeField.getText();

        String description = descriptionField.getText();

        String priceText = priceField.getText();

        String stockQuantityText = stockQuantityField.getText();

        // Validate input fields

        if (name.isEmpty() || productId.isEmpty() || category.isEmpty() || size.isEmpty() || description.isEmpty() || priceText.isEmpty() || stockQuantityText.isEmpty() || !isDouble(priceText) || !isInteger(stockQuantityText)) {

            JOptionPane.showMessageDialog(null, "Please fill in all the fields with valid values.", "Error", JOptionPane.ERROR\_MESSAGE);

            return;

        }

        // Add product to table

        String[] rowData = {name, productId, category, size, description, priceText, stockQuantityText};

        tableModel.addRow(rowData);

        // Clear input fields after adding

        clearFields();

        // Save data to file

        saveDataToFile();

    }

// Helper method to check if a string can be parsed as a double

    private boolean isDouble(String str) {

        try {

            Double.valueOf(str);

            return true;

        } catch (NumberFormatException e) {

            return false;

        }

    }

// Helper method to check if a string can be parsed as an integer

    private boolean isInteger(String str) {

        try {

            Integer.valueOf(str);

            return true;

        } catch (NumberFormatException e) {

            return false;

        }

    }

    private void updateProduct() {

        int selectedRow = productTable.getSelectedRow();

        if (selectedRow != -1) {

            String name = nameField.getText();

            String productId = productIdField.getText();

            String category = categoryField.getText();

            String size = sizeField.getText();

            String description = descriptionField.getText();

            String priceText = priceField.getText();

            String stockQuantityText = stockQuantityField.getText();

            // Check if any field is empty

            if (name.isEmpty() || productId.isEmpty() || category.isEmpty() || size.isEmpty() || description.isEmpty() || priceText.isEmpty() || stockQuantityText.isEmpty()) {

                JOptionPane.showMessageDialog(null, "Please fill in all the fields.", "Error", JOptionPane.ERROR\_MESSAGE);

                return;

            }

            try {

                double price = Double.parseDouble(priceText);

                int stockQuantity = Integer.parseInt(stockQuantityText);

                // Update selected row in table

                tableModel.setValueAt(name, selectedRow, 0);

                tableModel.setValueAt(productId, selectedRow, 1);

                tableModel.setValueAt(category, selectedRow, 2);

                tableModel.setValueAt(size, selectedRow, 3);

                tableModel.setValueAt(description, selectedRow, 4);

                tableModel.setValueAt(price, selectedRow, 5);

                tableModel.setValueAt(stockQuantity, selectedRow, 6);

                // Save data to file

                saveDataToFile();

                // Clear input fields after updating

                clearFields();

                // Show success message

                JOptionPane.showMessageDialog(null, "Product updated successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

            } catch (NumberFormatException ex) {

                JOptionPane.showMessageDialog(null, "Please enter valid values for price and stock quantity.", "Error", JOptionPane.ERROR\_MESSAGE);

            }

        } else {

            JOptionPane.showMessageDialog(null, "Please select a product to edit.", "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private boolean validateFields(String name, String productId, String category, String size, String description, String priceText, String stockQuantityText) {

        return !name.isEmpty() && !productId.isEmpty() && !category.isEmpty() && !size.isEmpty() && !description.isEmpty() && !priceText.isEmpty() && !stockQuantityText.isEmpty();

    }

    private void deleteProduct() {

        int selectedRow = productTable.getSelectedRow();

        if (selectedRow != -1) {

            tableModel.removeRow(selectedRow);

            // Save data to file

            saveDataToFile();

        } else {

            JOptionPane.showMessageDialog(null, "Please select a product to delete.", "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private void saveDataToFile() {

        String csvFile = "products.csv";

        try (FileWriter writer = new FileWriter(csvFile)) {

            for (int i = 0; i < tableModel.getRowCount(); i++) {

                for (int j = 0; j < tableModel.getColumnCount(); j++) {

                    writer.append(tableModel.getValueAt(i, j).toString());

                    if (j < tableModel.getColumnCount() - 1) {

                        writer.append(",");

                    }

                }

                writer.append("\n");

            }

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error saving product data to file: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private void clearFields() {

        nameField.setText("");

        productIdField.setText("");

        categoryField.setText("");

        sizeField.setText("");

        descriptionField.setText("");

        priceField.setText("");

        stockQuantityField.setText("");

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> new ManageProductScreen());

    }

}