package chefdonburi;

import javax.swing.\*;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.swing.table.JTableHeader;

public final class Inventory implements ActionListener {

private static final Inventory instance = null;

private JFrame frmInventoryManagement;

private JTable inventoryTable, inventory2Table;

private DefaultTableModel model, model2;

private JButton btnAdd, btnEdit, btnDelete, btnRefresh, btnAdd2, btnEdit2, btnDelete2, btnRefresh2, btnNextDay, btnAllNextDay, btnNextDay2, btnAllNextDay2;

private JLabel lblDate, lblSearch;

private JTextField txtSearch;

private JComboBox<String> categoryComboBox;

private Connection connection;

private PreparedStatement ps;

private ResultSet rs;

Inventory() {

init();

}

public Inventory getInstance() {

return instance;

}

void init() {

frmInventoryManagement = new JFrame("Inventory Management");

ImageIcon frameicon = new ImageIcon("src\\Images\\jframeicon.jpg");

Image frame = frameicon.getImage().getScaledInstance(100, 100, Image.SCALE\_SMOOTH);

frmInventoryManagement.setIconImage(frame);

frmInventoryManagement.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

frmInventoryManagement.setSize(1100, 700);

frmInventoryManagement.setLocationRelativeTo(null);

// Top Panel

JPanel topPanel = new JPanel(new BorderLayout());

// Initialize lblDate

lblDate = new JLabel(getCurrentDate(), SwingConstants.RIGHT);

lblDate.setFont(new Font("Arial", Font.BOLD, 14)); // Unified Arial font

lblDate.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 10));

topPanel.add(lblDate, BorderLayout.EAST);

// Search Panel

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

lblSearch = new JLabel("Search Items/Category:");

lblSearch.setFont(new Font("Arial", Font.BOLD, 14));

txtSearch = new JTextField(15);

txtSearch.addActionListener(e -> searchInventory());

categoryComboBox = new JComboBox<>(new String[]{"All", "Frozen Goods", "Sushi", "Vegetables", "Rice", "Dry Ingredients", "Dairy", "Wrapper", "Noodles", "Sauce", "Condiments", "Fruits", "Others", "Essentials"});

categoryComboBox.addActionListener(e -> {

txtSearch.setText("");

searchInventory();

});

// Initialize Category Label

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

categoryLabel.setFont(new Font("Arial", Font.BOLD, 14)); // Unified Arial font

searchPanel.add(lblSearch);

searchPanel.add(txtSearch);

searchPanel.add(categoryLabel);

searchPanel.add(categoryComboBox);

topPanel.add(searchPanel, BorderLayout.WEST);

frmInventoryManagement.add(topPanel, BorderLayout.NORTH);

// Inventory Table

model = new DefaultTableModel(new String[]{"ID", "Category", "Items", "Unit", "Price", "Beginning", "In", "Out", "Scrap", "Spoilage", "Ending"}, 0);

inventoryTable = new JTable(model);

inventoryTable.setRowHeight(30);

// Center align table data

DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();

centerRenderer.setHorizontalAlignment(SwingConstants.CENTER);

for (int i = 0; i < inventoryTable.getColumnCount(); i++) {

inventoryTable.getColumnModel().getColumn(i).setCellRenderer(centerRenderer);

}

JTableHeader header1 = inventoryTable.getTableHeader();

header1.setPreferredSize(new Dimension(header1.getPreferredSize().width, 30));

header1.setFont(new Font("Arial", Font.BOLD, 14)); // Bold Arial for header

header1.setBackground(new Color(223, 49, 42));

header1.setForeground(new Color(242, 245, 224));

JPanel inventoryPanel = new JPanel(new BorderLayout());

inventoryPanel.add(new JScrollPane(inventoryTable), BorderLayout.CENTER);

JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));

btnAdd = createButton("Add Item", btnPanel);

btnEdit = createButton("Edit Item", btnPanel);

btnDelete = createButton("Delete Item", btnPanel);

btnRefresh = createButton("Refresh", btnPanel);

btnNextDay = createButton("NextDay", btnPanel);

btnNextDay.addActionListener(e -> carryOverEndingStock(inventoryTable, model, "inventory"));

btnAllNextDay = createButton("AllNextDay", btnPanel);

btnAllNextDay.addActionListener(e -> carryOverAllEndingStock(inventoryTable, model, "inventory"));

inventoryPanel.add(btnPanel, BorderLayout.SOUTH);

// Inventory2 Table

model2 = new DefaultTableModel(new String[]{"ID", "Category", "Item", "Price", "SF", "Beginning", "In", "Out", "Spoilage", "Ending"}, 0);

inventory2Table = new JTable(model2);

inventory2Table.setRowHeight(30);

for (int i = 0; i < inventory2Table.getColumnCount(); i++) {

inventory2Table.getColumnModel().getColumn(i).setCellRenderer(centerRenderer);

}

JTableHeader header2 = inventory2Table.getTableHeader();

header2.setPreferredSize(new Dimension(header2.getPreferredSize().width, 30));

header2.setFont(new Font("Arial", Font.BOLD, 14)); // Bold Arial for header

header2.setBackground(new Color(223, 49, 42));

header2.setForeground(new Color(242, 245, 224));

JPanel inventory2Panel = new JPanel(new BorderLayout());

inventory2Panel.setBorder(BorderFactory.createEmptyBorder(10, 0, 0, 0));

inventory2Panel.add(new JScrollPane(inventory2Table), BorderLayout.CENTER);

JPanel btnPanel2 = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));

btnAdd2 = createButton("Add Item2", btnPanel2);

btnEdit2 = createButton("Edit Item2", btnPanel2);

btnDelete2 = createButton("Delete Item2", btnPanel2);

btnRefresh2 = createButton("Refresh2", btnPanel2);

btnNextDay2 = createButton("NextDay2", btnPanel2);

btnNextDay2.addActionListener(e -> carryOverEndingStock2(inventory2Table, model2, "inventory2"));

btnAllNextDay2 = createButton("AllNextDay2", btnPanel2);

btnAllNextDay2.addActionListener(e -> carryOverAllEndingStock2(inventory2Table, model2, "inventory2"));

inventory2Panel.add(btnPanel2, BorderLayout.SOUTH);

// SplitPane

JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL\_SPLIT, inventoryPanel, inventory2Panel);

splitPane.setResizeWeight(0.7);

splitPane.setDividerSize(10);

frmInventoryManagement.add(splitPane, BorderLayout.CENTER);

loadInventoryTable();

loadInventory2Table();

frmInventoryManagement.setVisible(true);

}

private JButton createButton(String text, JPanel panel) {

JButton button = new JButton(text);

button.setForeground(new Color(242, 245, 224));

button.setBackground(new Color(223, 49, 42));

button.setFont(new Font("Arial", Font.BOLD, 14));

button.addActionListener(this);

panel.add(button);

return button;

}

private String getCurrentDate() {

return new java.text.SimpleDateFormat("MMMM d, yyyy").format(new java.util.Date());

}

@Override

public void actionPerformed(ActionEvent e) {

if (e.getSource() == btnAdd) {

addItem();

} else if (e.getSource() == btnAdd2) {

addItem2();

} else if (e.getSource() == btnEdit) {

editItem();

} else if (e.getSource() == btnEdit2) {

editItem2();

} else if (e.getSource() == btnDelete) {

deleteItem();

} else if (e.getSource() == btnDelete2) {

deleteItem2();

} else if (e.getSource() == btnRefresh) {

loadInventoryTable();

loadInventory2Table();

}

}

private void searchInventory() {

String searchText = txtSearch.getText().trim();

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

model.setRowCount(0);

try {

connection = new Database().getConnection();

String query = "SELECT \* FROM inventory WHERE (ITEMS LIKE ? OR CATEGORY LIKE ?)";

if (!selectedCategory.equals("All")) query += " AND CATEGORY = ?";

ps = connection.prepareStatement(query);

ps.setString(1, "%" + searchText + "%");

ps.setString(2, "%" + searchText + "%");

if (!selectedCategory.equals("All")) ps.setString(3, selectedCategory);

rs = ps.executeQuery();

while (rs.next()) {

model.addRow(new Object[]{

rs.getInt("ID"),

rs.getString("CATEGORY"),

rs.getString("ITEMS"),

rs.getString("UNIT"),

rs.getDouble("PRICE"),

rs.getString("BEGINNING"),

rs.getString("QUANTITY\_IN"),

rs.getString("QUANTITY\_OUT"),

rs.getString("SCRAP"),

rs.getString("SPOILAGE"),

rs.getString("ACTUAL")

});

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error searching inventory: " + e.getMessage());

}

model2.setRowCount(0);

try {

String query2 = "SELECT \* FROM inventory2 WHERE (ITEM LIKE ? OR CATEGORY LIKE ?)";

if (!selectedCategory.equals("All")) query2 += " AND CATEGORY = ?";

ps = connection.prepareStatement(query2);

ps.setString(1, "%" + searchText + "%");

ps.setString(2, "%" + searchText + "%");

if (!selectedCategory.equals("All")) ps.setString(3, selectedCategory);

rs = ps.executeQuery();

while (rs.next()) {

model2.addRow(new Object[]{

rs.getInt("ID"),

rs.getString("CATEGORY"),

rs.getString("ITEM"),

rs.getDouble("PRICE"),

rs.getDouble("SF"),

rs.getString("BEGINNING"),

rs.getString("QUANTITY\_IN"),

rs.getString("QUANTITY\_OUT"),

rs.getString("SPOILAGE"),

rs.getString("ACTUAL")

});

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error searching inventory2: " + e.getMessage());

} finally {

closeConnections();

}

}

private void loadInventoryTable() {

model.setRowCount(0);

try {

connection = new Database().getConnection();

ps = connection.prepareStatement("SELECT \* FROM inventory");

rs = ps.executeQuery();

while (rs.next()) {

model.addRow(new Object[]{

rs.getInt("ID"),

rs.getString("CATEGORY"),

rs.getString("ITEMS"),

rs.getString("UNIT"),

rs.getDouble("PRICE"),

rs.getString("BEGINNING"),

rs.getString("QUANTITY\_IN"),

rs.getString("QUANTITY\_OUT"),

rs.getString("SCRAP"),

rs.getString("SPOILAGE"),

rs.getString("ACTUAL")

});

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error loading inventory: " + e.getMessage());

} finally {

closeConnections();

}

}

private void loadInventory2Table() {

model2.setRowCount(0);

try {

connection = new Database().getConnection();

ps = connection.prepareStatement("SELECT \* FROM inventory2");

rs = ps.executeQuery();

while (rs.next()) {

model2.addRow(new Object[]{

rs.getInt("ID"),

rs.getString("CATEGORY"),

rs.getString("ITEM"),

rs.getDouble("PRICE"),

rs.getDouble("SF"),

rs.getString("BEGINNING"),

rs.getString("QUANTITY\_IN"),

rs.getString("QUANTITY\_OUT"),

rs.getString("SPOILAGE"),

rs.getString("ACTUAL")

});

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error loading inventory2: " + e.getMessage());

} finally {

closeConnections();

}

}

private void carryOverEndingStock(JTable inventoryTable1, DefaultTableModel model1, String inventory) {

int selectedRow = inventoryTable1.getSelectedRow();

if (selectedRow == -1) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please select an item to update.", "No Item Selected", JOptionPane.WARNING\_MESSAGE);

return;

}

String endingStock = (String) model1.getValueAt(selectedRow, 10); // 10th column for "Ending"

String numericEndingStock = endingStock.replaceAll("[^0-9.]", ""); // Extract numbers only

int choice = JOptionPane.showConfirmDialog(frmInventoryManagement,

"Are you sure you want to update the beginning stock with the current ending stock (" + numericEndingStock + ")?",

"Confirm Update", JOptionPane.YES\_NO\_OPTION);

if (choice == JOptionPane.YES\_OPTION) {

model1.setValueAt(numericEndingStock, selectedRow, 5); // Update "Beginning" with numeric value

for (int i = 6; i <= 9; i++) model1.setValueAt("", selectedRow, i); // Clear In, Out, Scrap, Spoilage

int id = (int) model1.getValueAt(selectedRow, 0);

try {

connection = new Database().getConnection();

String query = "UPDATE inventory SET BEGINNING = ?, QUANTITY\_IN = ?, QUANTITY\_OUT = ?, SCRAP = ?, SPOILAGE = ? WHERE ID = ?";

ps = connection.prepareStatement(query);

ps.setString(1, numericEndingStock);

for (int i = 2; i <= 5; i++) ps.setString(i, "");

ps.setInt(6, id);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Beginning stock updated successfully.");

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error updating stock: " + e.getMessage());

} finally {

closeConnections();

}

}

}

private void carryOverAllEndingStock(JTable inventoryTable1, DefaultTableModel model1, String inventory) {

int choice = JOptionPane.showConfirmDialog(frmInventoryManagement,

"Are you sure you want to update the beginning stock for all items with their current ending stock?",

"Confirm Update for All Items", JOptionPane.YES\_NO\_OPTION);

if (choice == JOptionPane.YES\_OPTION) {

try {

connection = new Database().getConnection();

String query = "UPDATE inventory SET BEGINNING = ?, QUANTITY\_IN = ?, QUANTITY\_OUT = ?, SCRAP = ?, SPOILAGE = ? WHERE ID = ?";

for (int row = 0; row < model1.getRowCount(); row++) {

String endingStock = (String) model1.getValueAt(row, 10); // "Ending" column

String numericEndingStock = endingStock.replaceAll("[^0-9.]", ""); // Extract numbers only

int id = (int) model1.getValueAt(row, 0);

model1.setValueAt(numericEndingStock, row, 5); // Set "Beginning" to numeric "Ending"

for (int i = 6; i <= 9; i++) model1.setValueAt("", row, i); // Clear In, Out, Scrap, Spoilage

ps = connection.prepareStatement(query);

ps.setString(1, numericEndingStock);

for (int i = 2; i <= 5; i++) ps.setString(i, "");

ps.setInt(6, id);

ps.executeUpdate();

}

JOptionPane.showMessageDialog(frmInventoryManagement, "Beginning stock for all items updated successfully.");

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error updating stock: " + e.getMessage());

} finally {

closeConnections();

}

}

}

private void carryOverEndingStock2(JTable table, DefaultTableModel tableModel, String tableName) {

int selectedRow = table.getSelectedRow();

if (selectedRow == -1) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please select an item in " + tableName + " to update.", "No Item Selected", JOptionPane.WARNING\_MESSAGE);

return;

}

String endingStock = (String) tableModel.getValueAt(selectedRow, 9);

int choice2 = JOptionPane.showConfirmDialog(frmInventoryManagement,

"Are you sure you want to update the beginning stock with the current ending stock (" + endingStock + ")?",

"Confirm Update", JOptionPane.YES\_NO\_OPTION);

if (choice2 == JOptionPane.YES\_OPTION) {

tableModel.setValueAt(endingStock, selectedRow, 5); // 5th column for "Beginning"

for (int i = 6; i <= 8; i++) tableModel.setValueAt("", selectedRow, i); // Clear columns for In, Out, Spoilage

int id = (int) tableModel.getValueAt(selectedRow, 0);

try {

connection = new Database().getConnection();

String query = String.format("UPDATE %s SET BEGINNING = ?, QUANTITY\_IN = ?, QUANTITY\_OUT = ?, SPOILAGE = ? WHERE ID = ?", tableName);

ps = connection.prepareStatement(query);

ps.setString(1, endingStock); // BEGINNING column

for (int i = 2; i <= 4; i++) ps.setString(i, ""); // Clear QUANTITY\_IN, QUANTITY\_OUT, SPOILAGE

ps.setInt(5, id); // Set ID

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Beginning stock updated successfully in " + tableName + ".");

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error updating stock in " + tableName + ": " + e.getMessage());

} finally {

closeConnections();

}

}

}

private void carryOverAllEndingStock2(JTable table, DefaultTableModel tableModel, String tableName) {

int choice2 = JOptionPane.showConfirmDialog(frmInventoryManagement,

"Are you sure you want to update the beginning stock for all items with their current ending stock?",

"Confirm Update for All Items", JOptionPane.YES\_NO\_OPTION);

if (choice2 == JOptionPane.YES\_OPTION) {

try {

connection = new Database().getConnection();

String query = String.format("UPDATE %s SET BEGINNING = ?, QUANTITY\_IN = ?, QUANTITY\_OUT = ?, SPOILAGE = ? WHERE ID = ?", tableName);

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

String endingStock = (String) tableModel.getValueAt(row, 9); // "Ending" column

int id = (int) tableModel.getValueAt(row, 0);

tableModel.setValueAt(endingStock, row, 5); // Set "Beginning" to "Ending"

for (int i = 6; i <= 8; i++) tableModel.setValueAt("", row, i); // Clear columns for In, Out, Spoilage

ps = connection.prepareStatement(query);

ps.setString(1, endingStock); // BEGINNING column

for (int i = 2; i <= 4; i++) ps.setString(i, ""); // Clear QUANTITY\_IN, QUANTITY\_OUT, SPOILAGE

ps.setInt(5, id); // Set ID

ps.executeUpdate();

}

JOptionPane.showMessageDialog(frmInventoryManagement, "Beginning stock for all items updated successfully in " + tableName + ".");

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error updating stock in " + tableName + ": " + e.getMessage());

} finally {

closeConnections();

}

}

}

private void addItem() {

String[] categories = {"Frozen Goods", "Sushi", "Vegetables", "Rice", "Dry Ingredients", "Dairy", "Wrapper", "Noodles", "Sauce", "Condiments", "Fruits", "Others", "Essentials"};

JComboBox<String> categoryCombobox = new JComboBox<>(categories);

JTextField txtItem = new JTextField();

JTextField txtUnit = new JTextField();

JTextField txtPrice = new JTextField();

JTextField txtBeginning = new JTextField();

JTextField txtIn = new JTextField();

JTextField txtOut = new JTextField();

JTextField txtScrap = new JTextField();

JTextField txtSpoilage = new JTextField();

JTextField txtActual = new JTextField();

txtActual.setEditable(false); // Make ending stock read-only

Object[] message = {

"Category:", categoryCombobox,

"Item:", txtItem,

"Unit:", txtUnit,

"Price:", txtPrice,

"Beginning:", txtBeginning,

"Quantity In:", txtIn,

"Quantity Out:", txtOut,

"Scrap:", txtScrap,

"Spoilage:", txtSpoilage,

"Ending (calculated):", txtActual

};

boolean isInputValid;

do {

isInputValid = true;

int option = JOptionPane.showConfirmDialog(frmInventoryManagement, message, "Add New Item", JOptionPane.OK\_CANCEL\_OPTION);

if (option == JOptionPane.OK\_OPTION) {

if (txtItem.getText().trim().isEmpty()) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Item name is required!", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

double price = 0.0;

try {

String priceText = txtPrice.getText().trim();

if (!priceText.isEmpty()) {

price = Double.parseDouble(priceText);

if (price <= 0) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please enter a positive price.", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Invalid number format for price.", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

double beginning = parseQuantity(txtBeginning.getText(), "Beginning");

double in = parseQuantity(txtIn.getText(), "Quantity In");

double out = parseQuantity(txtOut.getText(), "Quantity Out");

double scrap = parseQuantity(txtScrap.getText(), "Scrap");

double spoilage = parseQuantity(txtSpoilage.getText(), "Spoilage");

double totalAvailable = beginning + in;

if (out + scrap + spoilage > totalAvailable) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Out, Scrap, and Spoilage cannot exceed Beginning + Quantity In.", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

double ending = totalAvailable - out - scrap - spoilage;

String unit = txtUnit.getText().trim().toLowerCase();

String endingWithUnit = formatEndingWithUnit(ending, unit);

txtActual.setText(endingWithUnit);

if (isInputValid) {

try {

connection = new Database().getConnection();

String query = "INSERT INTO inventory (CATEGORY, ITEMS, UNIT, PRICE, BEGINNING, QUANTITY\_IN, QUANTITY\_OUT, SCRAP, SPOILAGE, ACTUAL) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

ps = connection.prepareStatement(query);

ps.setString(1, (String) categoryCombobox.getSelectedItem());

ps.setString(2, txtItem.getText().trim());

ps.setString(3, unit);

ps.setDouble(4, price);

ps.setDouble(5, beginning);

ps.setDouble(6, in);

ps.setDouble(7, out);

ps.setDouble(8, scrap);

ps.setDouble(9, spoilage);

ps.setString(10, endingWithUnit);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Item added successfully!");

loadInventoryTable();

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error adding item: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

}

} else {

break;

}

} while (!isInputValid);

}

// Method to add item to inventory2Table

private void addItem2() {

String[] categories = {"Frozen Goods", "Sushi", "Vegetables", "Rice", "Dry Ingredients", "Dairy", "Wrapper", "Noodles", "Sauce", "Condiments", "Fruits", "Others", "Essentials"};

JComboBox<String> categoryCombobox = new JComboBox<>(categories);

JTextField txtItem = new JTextField();

JTextField txtPrice = new JTextField();

JTextField txtSF = new JTextField(); // SF (Stock Factor or similar)

JTextField txtBeginning = new JTextField();

JTextField txtIn = new JTextField();

JTextField txtOut = new JTextField();

JTextField txtSpoilage = new JTextField();

JTextField txtActual = new JTextField();

txtActual.setEditable(false); // Make ending stock read-only

Object[] message = {

"Category:", categoryCombobox,

"Item:", txtItem,

"Price:", txtPrice,

"SF:", txtSF,

"Beginning:", txtBeginning,

"Quantity In:", txtIn,

"Quantity Out:", txtOut,

"Spoilage:", txtSpoilage,

"Ending (calculated):", txtActual

};

boolean isInputValid;

do {

isInputValid = true;

int option = JOptionPane.showConfirmDialog(frmInventoryManagement, message, "Add New Item to Inventory2", JOptionPane.OK\_CANCEL\_OPTION);

if (option == JOptionPane.OK\_OPTION) {

if (txtItem.getText().trim().isEmpty()) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Item name is required!", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

double price = 0.0;

double sf = 0.0;

try {

String priceText = txtPrice.getText().trim();

if (!priceText.isEmpty()) {

price = Double.parseDouble(priceText);

if (price <= 0) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please enter a positive price.", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

}

String sfText = txtSF.getText().trim();

if (!sfText.isEmpty()) {

sf = Double.parseDouble(sfText);

if (sf <= 0) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please enter a positive SF value.", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

}

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Invalid number format for price or SF.", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

double beginning = parseQuantity(txtBeginning.getText(), "Beginning");

double in = parseQuantity(txtIn.getText(), "Quantity In");

double out = parseQuantity(txtOut.getText(), "Quantity Out");

double spoilage = parseQuantity(txtSpoilage.getText(), "Spoilage");

double totalAvailable = beginning + in;

if (out + spoilage > totalAvailable) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Out and Spoilage cannot exceed Beginning + Quantity In.", "Error", JOptionPane.ERROR\_MESSAGE);

isInputValid = false;

continue;

}

double ending = totalAvailable - out - spoilage;

txtActual.setText(formatEndingWithUnit(ending, "pcs")); // Assuming units are in pieces (adjust if needed)

if (isInputValid) {

try {

connection = new Database().getConnection();

String query = "INSERT INTO inventory2 (CATEGORY, ITEM, PRICE, SF, BEGINNING, QUANTITY\_IN, QUANTITY\_OUT, SPOILAGE, ACTUAL) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";

ps = connection.prepareStatement(query);

ps.setString(1, (String) categoryCombobox.getSelectedItem());

ps.setString(2, txtItem.getText().trim());

ps.setDouble(3, price);

ps.setDouble(4, sf);

ps.setDouble(5, beginning);

ps.setDouble(6, in);

ps.setDouble(7, out);

ps.setDouble(8, spoilage);

ps.setDouble(9, ending); // Store the calculated ending value

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Item added to Inventory2 successfully!");

loadInventory2Table();

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error adding item to Inventory2: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

}

} else {

break;

}

} while (!isInputValid);

}

private double parseQuantity(String text, String fieldName) {

try {

if (text.trim().isEmpty()) {

return 0.0;

}

String numericPart = text.replaceAll("[^0-9.]", "").trim();

String unitPart = text.replaceAll("[0-9.]", "").trim().toLowerCase();

double quantity = Double.parseDouble(numericPart);

if (unitPart.equals("g")) {

quantity /= 1000.0;

} else if (unitPart.equals("kg")) {

// Already in kilograms

} else if (unitPart.equals("pcs")) {

// No conversion needed

} else if (!unitPart.isEmpty()) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Unsupported unit for " + fieldName + ": " + unitPart, "Error", JOptionPane.ERROR\_MESSAGE);

throw new IllegalArgumentException("Unsupported unit: " + unitPart);

}

return quantity;

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Invalid number format for " + fieldName, "Error", JOptionPane.ERROR\_MESSAGE);

throw e;

}

}

private String formatEndingWithUnit(double ending, String unit) {

switch (unit) {

case "g":

// Convert grams to kilograms for display if ending weight is less than 1 kg

double kilograms = ending / 1000.0;

// Display in kg with 1 to 3 decimals based on need; e.g., 0.1 kg or 0.123 kg

return kilograms < 1.0

? String.format("%.3f", kilograms).replaceAll("0{1,2}$", "") + " kg"

: String.format("%.1f kg", kilograms);

case "kg":

// Format for kilograms, removing unnecessary trailing zeros

return String.format("%.3f", ending).replaceAll("0{1,2}$", "") + " kg";

case "pcs":

// Display pieces as a whole number

return String.format("%.0f pcs", ending);

default:

// Default format with 3 decimals

return String.format("%.3f", ending);

}

}

private void editItem() {

int selectedRow = inventoryTable.getSelectedRow();

if (selectedRow == -1) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please select an item to edit!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int itemId = (int) model.getValueAt(selectedRow, 0);

String category = (String) model.getValueAt(selectedRow, 1);

String item = (String) model.getValueAt(selectedRow, 2);

String unit = (String) model.getValueAt(selectedRow, 3);

double price = (double) model.getValueAt(selectedRow, 4);

String beginning = (String) model.getValueAt(selectedRow, 5);

String quantityIn = (String) model.getValueAt(selectedRow, 6);

String quantityOut = (String) model.getValueAt(selectedRow, 7);

String scrap = (String) model.getValueAt(selectedRow, 8);

String spoilage = (String) model.getValueAt(selectedRow, 9);

String actual = (String) model.getValueAt(selectedRow, 10);

JComboBox<String> categoryCombobox = new JComboBox<>(new String[]{"Frozen Goods", "Sushi", "Vegetables", "Rice", "Dry Ingredients", "Dairy", "Wrapper", "Noodles", "Sauce", "Condiments", "Fruits", "Others", "Essentials"});

categoryCombobox.setSelectedItem(category);

JTextField txtItem = new JTextField(item);

JTextField txtUnit = new JTextField(unit);

JTextField txtPrice = new JTextField(String.valueOf(price));

JTextField txtBeginning = new JTextField(beginning);

JTextField txtIn = new JTextField(quantityIn);

JTextField txtOut = new JTextField(quantityOut);

JTextField txtScrap = new JTextField(scrap);

JTextField txtSpoilage = new JTextField(spoilage);

JTextField txtActual = new JTextField(actual);

txtActual.setEditable(false);

while (true) { // Keep the dialog open if no changes are made

Object[] message = {

"Category:", categoryCombobox,

"Item:", txtItem,

"Unit:", txtUnit,

"Price:", txtPrice,

"Beginning:", txtBeginning,

"Quantity In:", txtIn,

"Quantity Out:", txtOut,

"Scrap:", txtScrap,

"Spoilage:", txtSpoilage,

"Ending (calculated):", txtActual

};

int option = JOptionPane.showConfirmDialog(frmInventoryManagement, message, "Edit Item", JOptionPane.OK\_CANCEL\_OPTION);

if (option == JOptionPane.CANCEL\_OPTION || option == JOptionPane.CLOSED\_OPTION) {

return; // Exit the method if user cancels

}

try {

double newPrice = Double.parseDouble(txtPrice.getText());

double beginningQty = parseQuantity(txtBeginning.getText(), "Beginning");

double inQty = parseQuantity(txtIn.getText(), "Quantity In");

double outQty = parseQuantity(txtOut.getText(), "Quantity Out");

double scrapQty = parseQuantity(txtScrap.getText(), "Scrap");

double spoilageQty = parseQuantity(txtSpoilage.getText(), "Spoilage");

double totalAvailable = beginningQty + inQty;

if (outQty + scrapQty + spoilageQty > totalAvailable) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Out, Scrap, and Spoilage cannot exceed Beginning + Quantity In.", "Error", JOptionPane.ERROR\_MESSAGE);

continue; // Keep the dialog open if validation fails

}

double endingQty = totalAvailable - outQty - scrapQty - spoilageQty;

String formattedEnding = formatEndingWithUnit(endingQty, txtUnit.getText());

// Check if any changes were made

if (category.equals(categoryCombobox.getSelectedItem()) &&

item.equals(txtItem.getText()) &&

unit.equals(txtUnit.getText()) &&

price == newPrice &&

Double.parseDouble(beginning) == beginningQty &&

Double.parseDouble(quantityIn) == inQty &&

Double.parseDouble(quantityOut) == outQty &&

Double.parseDouble(scrap) == scrapQty &&

Double.parseDouble(spoilage) == spoilageQty &&

actual.equals(formattedEnding)) {

JOptionPane.showMessageDialog(frmInventoryManagement, "No changes were made.", "No Changes", JOptionPane.INFORMATION\_MESSAGE);

continue; // Keep the dialog open after showing the message

}

// Update the database

connection = new Database().getConnection();

String query = "UPDATE inventory SET CATEGORY = ?, ITEMS = ?, UNIT = ?, PRICE = ?, BEGINNING = ?, QUANTITY\_IN = ?, QUANTITY\_OUT = ?, SCRAP = ?, SPOILAGE = ?, ACTUAL = ? WHERE ID = ?";

ps = connection.prepareStatement(query);

ps.setString(1, (String) categoryCombobox.getSelectedItem());

ps.setString(2, txtItem.getText());

ps.setString(3, txtUnit.getText());

ps.setDouble(4, newPrice);

ps.setDouble(5, beginningQty);

ps.setDouble(6, inQty);

ps.setDouble(7, outQty);

ps.setDouble(8, scrapQty);

ps.setDouble(9, spoilageQty);

ps.setString(10, formattedEnding);

ps.setInt(11, itemId);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Item updated successfully!");

loadInventoryTable();

return; // Exit after successful update

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error updating item: " + e.getMessage());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Invalid number format for price or quantity.", "Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

}

}

private void editItem2() {

int selectedRow = inventory2Table.getSelectedRow();

if (selectedRow == -1) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please select an item in Inventory2 to edit!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int itemId = (int) model2.getValueAt(selectedRow, 0);

String category = (String) model2.getValueAt(selectedRow, 1);

String item = (String) model2.getValueAt(selectedRow, 2);

double price = (double) model2.getValueAt(selectedRow, 3);

double sf = (double) model2.getValueAt(selectedRow, 4);

String beginning = (String) model2.getValueAt(selectedRow, 5);

String quantityIn = (String) model2.getValueAt(selectedRow, 6);

String quantityOut = (String) model2.getValueAt(selectedRow, 7);

String spoilage = (String) model2.getValueAt(selectedRow, 8);

String actual = (String) model2.getValueAt(selectedRow, 9);

JComboBox<String> categoryCombobox = new JComboBox<>(new String[]{"Frozen Goods", "Sushi", "Vegetables", "Rice", "Dry Ingredients", "Dairy", "Wrapper", "Noodles", "Sauce", "Condiments", "Fruits", "Others", "Essentials"});

categoryCombobox.setSelectedItem(category);

JTextField txtItem = new JTextField(item);

JTextField txtPrice = new JTextField(String.valueOf(price));

JTextField txtSF = new JTextField(String.valueOf(sf));

JTextField txtBeginning = new JTextField(beginning);

JTextField txtIn = new JTextField(quantityIn);

JTextField txtOut = new JTextField(quantityOut);

JTextField txtSpoilage = new JTextField(spoilage);

JTextField txtActual = new JTextField(actual);

txtActual.setEditable(false);

while (true) { // Keep the dialog open if no changes are made

Object[] message = {

"Category:", categoryCombobox,

"Item:", txtItem,

"Price:", txtPrice,

"SF:", txtSF,

"Beginning:", txtBeginning,

"Quantity In:", txtIn,

"Quantity Out:", txtOut,

"Spoilage:", txtSpoilage,

"Ending (calculated):", txtActual

};

int option = JOptionPane.showConfirmDialog(frmInventoryManagement, message, "Edit Item in Inventory2", JOptionPane.OK\_CANCEL\_OPTION);

if (option == JOptionPane.CANCEL\_OPTION || option == JOptionPane.CLOSED\_OPTION) {

return; // Exit the method if user cancels

}

try {

double newPrice = Double.parseDouble(txtPrice.getText());

double newSF = Double.parseDouble(txtSF.getText());

double beginningQty = parseQuantity(txtBeginning.getText(), "Beginning");

double inQty = parseQuantity(txtIn.getText(), "Quantity In");

double outQty = parseQuantity(txtOut.getText(), "Quantity Out");

double spoilageQty = parseQuantity(txtSpoilage.getText(), "Spoilage");

double totalAvailable = beginningQty + inQty;

if (outQty + spoilageQty > totalAvailable) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Out and Spoilage cannot exceed Beginning + Quantity In.", "Error", JOptionPane.ERROR\_MESSAGE);

continue; // Keep the dialog open if validation fails

}

double endingQty = totalAvailable - outQty - spoilageQty;

String formattedEnding = String.format("%.2f", endingQty);

// Check if any changes were made

if (category.equals(categoryCombobox.getSelectedItem()) &&

item.equals(txtItem.getText()) &&

price == newPrice &&

sf == newSF &&

Double.parseDouble(beginning) == beginningQty &&

Double.parseDouble(quantityIn) == inQty &&

Double.parseDouble(quantityOut) == outQty &&

Double.parseDouble(spoilage) == spoilageQty &&

actual.equals(formattedEnding)) {

JOptionPane.showMessageDialog(frmInventoryManagement, "No changes were made.", "No Changes", JOptionPane.INFORMATION\_MESSAGE);

continue; // Keep the dialog open after showing the message

}

// Update the database

connection = new Database().getConnection();

String query = "UPDATE inventory2 SET CATEGORY = ?, ITEM = ?, PRICE = ?, SF = ?, BEGINNING = ?, QUANTITY\_IN = ?, QUANTITY\_OUT = ?, SPOILAGE = ?, ACTUAL = ? WHERE ID = ?";

ps = connection.prepareStatement(query);

ps.setString(1, (String) categoryCombobox.getSelectedItem());

ps.setString(2, txtItem.getText());

ps.setDouble(3, newPrice);

ps.setDouble(4, newSF);

ps.setDouble(5, beginningQty);

ps.setDouble(6, inQty);

ps.setDouble(7, outQty);

ps.setDouble(8, spoilageQty);

ps.setString(9, formattedEnding);

ps.setInt(10, itemId);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Item in Inventory2 updated successfully!");

loadInventory2Table();

return; // Exit after successful update

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error updating item in Inventory2: " + e.getMessage());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Invalid number format for price, SF, or quantity.", "Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

}

}

private void deleteItem() {

int selectedRow = inventoryTable.getSelectedRow();

if (selectedRow == -1) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please select an item to delete!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int itemId = (int) model.getValueAt(selectedRow, 0);

int option = JOptionPane.showConfirmDialog(frmInventoryManagement, "Are you sure you want to delete this item?", "Confirm Delete", JOptionPane.YES\_NO\_OPTION);

if (option == JOptionPane.YES\_OPTION) {

try {

connection = new Database().getConnection();

String query = "DELETE FROM inventory WHERE ID = ?";

ps = connection.prepareStatement(query);

ps.setInt(1, itemId);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Item deleted successfully!");

loadInventoryTable();

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error deleting item: " + e.getMessage());

} finally {

closeConnections();

}

}

}

private void deleteItem2() {

int selectedRow = inventory2Table.getSelectedRow();

if (selectedRow == -1) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Please select an item in Inventory2 to delete!", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int itemId = (int) model2.getValueAt(selectedRow, 0);

int option = JOptionPane.showConfirmDialog(frmInventoryManagement, "Are you sure you want to delete this item from Inventory2?", "Confirm Delete", JOptionPane.YES\_NO\_OPTION);

if (option == JOptionPane.YES\_OPTION) {

try {

connection = new Database().getConnection();

String query = "DELETE FROM inventory2 WHERE ID = ?";

ps = connection.prepareStatement(query);

ps.setInt(1, itemId);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmInventoryManagement, "Item deleted successfully from Inventory2!");

loadInventory2Table();

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error deleting item from Inventory2: " + e.getMessage());

} finally {

closeConnections();

}

}

}

private void closeConnections() {

try {

if (rs != null) rs.close();

if (ps != null) ps.close();

if (connection != null) connection.close();

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmInventoryManagement, "Error closing connections: " + e.getMessage());

}

}

public static void main(String[] args) {

Inventory inventory = new Inventory();

}

}