package chefdonburi;

import javax.swing.\*;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.JTableHeader;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.sql.\*;

import java.text.SimpleDateFormat;

import java.text.ParseException;

import java.util.Date;

public final class Expenses implements ActionListener {

JFrame frmExpenses;

JTable expensesTable;

DefaultTableModel model;

JButton btnAdd, btnEdit, btnDelete, btnRefresh;

JTextField txtSearch;

JLabel lblDate, lblTotalPrice;

private Connection connection;

private PreparedStatement ps;

private ResultSet rs;

public Expenses() {

init();

}

public void init() {

frmExpenses = new JFrame("Expenses Management");

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

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

frmExpenses.setIconImage(frame);

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

frmExpenses.setSize(1100, 700);

frmExpenses.setLocationRelativeTo(null);

frmExpenses.setLayout(new BorderLayout(10, 10));

// Top panel with search bar and date

JPanel topPanel = new JPanel(new BorderLayout(10, 10));

JPanel searchPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 10));

JLabel lblSearch = new JLabel("Search Item Name:");

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

txtSearch = new JTextField(20);

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

searchPanel.add(lblSearch);

searchPanel.add(txtSearch);

topPanel.add(searchPanel, BorderLayout.WEST);

JPanel datePanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 10));

lblDate = new JLabel();

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

updateDateLabel();

datePanel.add(lblDate);

topPanel.add(datePanel, BorderLayout.EAST);

frmExpenses.add(topPanel, BorderLayout.NORTH);

// Table setup

model = new DefaultTableModel(

new String[]{"ID", "Item Name", "Item Price", "Number/Unit", "Source of Purchase", "Mode of Payment", "Date & Time"},

0

);

expensesTable = new JTable(model);

expensesTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

expensesTable.setFont(new Font("Arial", Font.PLAIN, 14));

expensesTable.setRowHeight(30);

DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();

centerRenderer.setHorizontalAlignment(JLabel.CENTER);

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

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

}

JTableHeader tableHeader = expensesTable.getTableHeader();

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

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

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

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

JScrollPane scrollPane = new JScrollPane(expensesTable);

scrollPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

frmExpenses.add(scrollPane, BorderLayout.CENTER);

// Button panel

JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 20));

Dimension buttonSize = new Dimension(150, 30);

btnAdd = createButton("Add Expense", buttonSize, this);

btnPanel.add(btnAdd);

btnEdit = createButton("Edit Expense", buttonSize, this);

btnPanel.add(btnEdit);

btnDelete = createButton("Delete Expense", buttonSize, this);

btnPanel.add(btnDelete);

btnRefresh = createButton("Refresh", buttonSize, this);

btnPanel.add(btnRefresh);

JPanel centerPanel = new JPanel(new BorderLayout(10, 10));

centerPanel.add(scrollPane, BorderLayout.CENTER);

centerPanel.add(btnPanel, BorderLayout.SOUTH);

frmExpenses.add(centerPanel, BorderLayout.CENTER);

// Bottom panel for total price

JPanel bottomPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 10));

lblTotalPrice = new JLabel("Total Price: 0.00");

lblTotalPrice.setFont(new Font("Arial", Font.BOLD, 16));

bottomPanel.add(lblTotalPrice);

frmExpenses.add(bottomPanel, BorderLayout.SOUTH);

loadExpensesTable();

txtSearch.addKeyListener(new KeyAdapter() {

@Override

public void keyPressed(KeyEvent e) {

if (e.getKeyCode() == KeyEvent.VK\_ENTER) {

searchItem();

}

}

});

frmExpenses.setVisible(true);

}

private JButton createButton(String text, Dimension size, ActionListener listener) {

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, 15));

button.setPreferredSize(size);

button.addActionListener(listener);

return button;

}

@Override

public void actionPerformed(ActionEvent e) {

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

addExpense();

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

editExpense();

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

deleteExpense();

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

loadExpensesTable();

}

}

private void searchItem() {

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

model.setRowCount(0);

if (searchText.isEmpty()) {

loadExpensesTable();

return;

}

try {

connection = new Database().getConnection();

ps = connection.prepareStatement("SELECT \* FROM expenses WHERE LOWER(ITEM\_NAME) LIKE ?");

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

rs = ps.executeQuery();

while (rs.next()) {

addRowToModel(rs);

}

if (model.getRowCount() == 0) {

JOptionPane.showMessageDialog(frmExpenses, "No matching items found.", "Search Result", JOptionPane.INFORMATION\_MESSAGE);

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmExpenses, "Database error: " + e.getMessage(), "Database Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

updateTotalPrice();

}

private void updateDateLabel() {

SimpleDateFormat sdf = new SimpleDateFormat("MMMM d, yyyy");

lblDate.setText(sdf.format(new Date()));

}

private void loadExpensesTable() {

model.setRowCount(0);

try {

connection = new Database().getConnection();

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

rs = ps.executeQuery();

while (rs.next()) {

addRowToModel(rs);

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmExpenses, "Database error: " + e.getMessage(), "Database Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

updateTotalPrice();

}

private String formatDate(String dateTime, String inputPattern, String outputPattern) {

try {

SimpleDateFormat inputFormat = new SimpleDateFormat(inputPattern);

SimpleDateFormat outputFormat = new SimpleDateFormat(outputPattern);

Date date = inputFormat.parse(dateTime);

return outputFormat.format(date);

} catch (ParseException e) {

return dateTime;

}

}

private void addRowToModel(ResultSet rs) throws SQLException {

int id = rs.getInt("ID");

String itemName = rs.getString("ITEM\_NAME");

double itemPrice = rs.getDouble("ITEM\_PRICE");

String numberUnit = rs.getString("NUMBER\_UNIT");

String source = rs.getString("SOURCE");

String modeOfPayment = rs.getString("MODE\_OF\_PAYMENT");

String dateTime = rs.getString("DATE\_TIME");

String formattedDate = formatDate(dateTime, "yyyy-MM-dd HH:mm:ss", "MMMM d, yyyy HH:mm:ss");

model.addRow(new Object[]{id, itemName, itemPrice, numberUnit, source, modeOfPayment, formattedDate});

}

private void addExpense() {

JTextField txtItemName = new JTextField();

JTextField txtItemPrice = new JTextField();

JTextField txtNumberUnit = new JTextField();

JTextField txtSource = new JTextField();

JTextField txtModeOfPayment = new JTextField();

boolean inputValid = false;

while (!inputValid) {

Object[] message = {

"Item Name:", txtItemName,

"Item Price:", txtItemPrice,

"Number/Unit:", txtNumberUnit,

"Source of Purchase:", txtSource,

"Mode of Payment:", txtModeOfPayment

};

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

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

String itemName = txtItemName.getText().trim();

String itemPriceText = txtItemPrice.getText().trim();

String numberUnit = txtNumberUnit.getText().trim();

String source = txtSource.getText().trim();

String modeOfPayment = txtModeOfPayment.getText().trim();

if (itemName.isEmpty() || itemPriceText.isEmpty() || numberUnit.isEmpty() || source.isEmpty() || modeOfPayment.isEmpty()) {

JOptionPane.showMessageDialog(frmExpenses, "All fields must be filled out.", "Input Error", JOptionPane.ERROR\_MESSAGE);

continue;

}

try {

double itemPrice = Double.parseDouble(itemPriceText);

connection = new Database().getConnection();

ps = connection.prepareStatement(

"INSERT INTO expenses (ITEM\_NAME, ITEM\_PRICE, NUMBER\_UNIT, SOURCE, MODE\_OF\_PAYMENT, DATE\_TIME) VALUES (?, ?, ?, ?, ?, NOW())"

);

ps.setString(1, itemName);

ps.setDouble(2, itemPrice);

ps.setString(3, numberUnit);

ps.setString(4, source);

ps.setString(5, modeOfPayment);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmExpenses, "Expense added successfully!");

loadExpensesTable(); // Refresh table

inputValid = true;

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(frmExpenses, "Invalid price format.", "Input Error", JOptionPane.ERROR\_MESSAGE);

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmExpenses, "Database error: " + e.getMessage(), "Database Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

} else {

break; // Exit loop if Cancel is clicked

}

}

}

private void editExpense() {

int selectedRow = expensesTable.getSelectedRow();

if (selectedRow >= 0) {

// Retrieve original values

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

String originalItemName = model.getValueAt(selectedRow, 1).toString();

String originalItemPrice = model.getValueAt(selectedRow, 2).toString();

String originalNumberUnit = model.getValueAt(selectedRow, 3).toString();

String originalSource = model.getValueAt(selectedRow, 4).toString();

String originalModeOfPayment = model.getValueAt(selectedRow, 5).toString();

// Create editable text fields pre-filled with original values

JTextField txtItemName = new JTextField(originalItemName);

JTextField txtItemPrice = new JTextField(originalItemPrice);

JTextField txtNumberUnit = new JTextField(originalNumberUnit);

JTextField txtSource = new JTextField(originalSource);

JTextField txtModeOfPayment = new JTextField(originalModeOfPayment);

// Define the input fields and labels for the dialog

Object[] message = {

"Item Name:", txtItemName,

"Item Price:", txtItemPrice,

"Number/Unit:", txtNumberUnit,

"Source of Purchase:", txtSource,

"Mode of Payment:", txtModeOfPayment

};

while (true) {

// Display the dialog

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

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

// Get user input

String itemName = txtItemName.getText().trim();

String itemPriceText = txtItemPrice.getText().trim();

String numberUnit = txtNumberUnit.getText().trim();

String source = txtSource.getText().trim();

String modeOfPayment = txtModeOfPayment.getText().trim();

// Check for empty fields

if (itemName.isEmpty() || itemPriceText.isEmpty() || numberUnit.isEmpty() || source.isEmpty() || modeOfPayment.isEmpty()) {

JOptionPane.showMessageDialog(frmExpenses, "All fields must be filled out.", "Input Error", JOptionPane.ERROR\_MESSAGE);

continue; // Keep the dialog open

}

// Check for no changes

if (itemName.equals(originalItemName) &&

itemPriceText.equals(originalItemPrice) &&

numberUnit.equals(originalNumberUnit) &&

source.equals(originalSource) &&

modeOfPayment.equals(originalModeOfPayment)) {

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

continue; // Allow user to continue editing

}

// Validate and save changes

try {

double itemPrice = Double.parseDouble(itemPriceText); // Validate price format

// Update the database

connection = new Database().getConnection();

ps = connection.prepareStatement(

"UPDATE expenses SET ITEM\_NAME=?, ITEM\_PRICE=?, NUMBER\_UNIT=?, SOURCE=?, MODE\_OF\_PAYMENT=?, DATE\_TIME=NOW() WHERE ID=?"

);

ps.setString(1, itemName);

ps.setDouble(2, itemPrice);

ps.setString(3, numberUnit);

ps.setString(4, source);

ps.setString(5, modeOfPayment);

ps.setInt(6, id);

ps.executeUpdate();

// Refresh the table

loadExpensesTable();

JOptionPane.showMessageDialog(frmExpenses, "Expense updated successfully!");

break; // Exit the loop when successful

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(frmExpenses, "Invalid price format.", "Input Error", JOptionPane.ERROR\_MESSAGE);

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmExpenses, "Database error: " + e.getMessage(), "Database Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

} else {

break; // Exit the loop if the user clicks Cancel

}

}

} else {

JOptionPane.showMessageDialog(frmExpenses, "Please select an expense to edit.", "Selection Error", JOptionPane.WARNING\_MESSAGE);

}

}

private void deleteExpense() {

int selectedRow = expensesTable.getSelectedRow();

if (selectedRow >= 0) {

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

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

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

try {

connection = new Database().getConnection();

ps = connection.prepareStatement("DELETE FROM expenses WHERE ID=?");

ps.setInt(1, id);

ps.executeUpdate();

JOptionPane.showMessageDialog(frmExpenses, "Expense deleted successfully!");

loadExpensesTable(); // Refresh table

} catch (SQLException e) {

JOptionPane.showMessageDialog(frmExpenses, "Database error: " + e.getMessage(), "Database Error", JOptionPane.ERROR\_MESSAGE);

} finally {

closeConnections();

}

}

} else {

JOptionPane.showMessageDialog(frmExpenses, "Please select an expense to delete.", "Selection Error", JOptionPane.WARNING\_MESSAGE);

}

}

private void updateTotalPrice() {

double totalPrice = 0;

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

totalPrice += (double) model.getValueAt(i, 2);

}

lblTotalPrice.setText("Total Price: " + String.format("%.2f", totalPrice));

}

private void closeConnections() {

try {

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

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

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

} catch (SQLException e) {

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

}

}

public static void main(String[] args) {

Expenses expenses = new Expenses();

}

}