package m6.ui;

import m6.ConnectionManager;

import m6.UserLoginInfo;

import m6.components.StyledButton;

import m6.table\_model.TranscationTableModel;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.sql.\*;

public class TransactionViewerUI extends UserBaseUI {

private UserLoginInfo userLoginInfo;

private String username;

private String accountNumber;

private StyledButton depositBtn, withdrawBtn, transferBtn;

private JTable table;

private JScrollPane scrollPane;

public TransactionViewerUI(UserLoginInfo userLoginInfo, String username) {

super(userLoginInfo);

this.userLoginInfo = userLoginInfo;

this.username = username;

this.accountNumber = getAccountNumber(username);

setPageTitle("View Transactions | " + username);

backButton.setVisible(true);

backButton.addActionListener(this);

initUI();

bind();

}

private void initUI() {

table = new JTable(new TranscationTableModel(this, username));

table.setPreferredScrollableViewportSize(new Dimension(500, 70));

table.setCellSelectionEnabled(false);

table.setRowSelectionAllowed(false);

table.setColumnSelectionAllowed(false);

// table.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

scrollPane = new JScrollPane(table);

scrollPane.setBounds(10, 115, 775, 345);

depositBtn = new StyledButton("Deposit");

depositBtn.setBounds(10, 72, 110, 35);

depositBtn.setBackground(new Color(0xDCBD7E));

withdrawBtn = new StyledButton("Withdraw");

withdrawBtn.setBounds(135, 72, 110, 35);

withdrawBtn.setBackground(new Color(0xDCBD7E));

transferBtn = new StyledButton("Transfer");

transferBtn.setBounds(260, 72, 110, 35);

transferBtn.setBackground(new Color(0xDCBD7E));

if (!userLoginInfo.status.equals("employee")) {

depositBtn.setVisible(false);

withdrawBtn.setVisible(false);

transferBtn.setVisible(false);

}

mainPanel.add(depositBtn);

mainPanel.add(withdrawBtn);

mainPanel.add(transferBtn);

mainPanel.add(scrollPane);

}

private void bind() {

depositBtn.addActionListener(this);

withdrawBtn.addActionListener(this);

transferBtn.addActionListener(this);

}

public void actionPerformed(ActionEvent e) {

super.actionPerformed(e);

Object src = e.getSource();

try {

if (src == backButton) {

if (userLoginInfo.status.equals("employee")) {

new CustomerViewerUI(userLoginInfo).setVisible(true);

} else {

new CustomerUI(userLoginInfo).setVisible(true);

}

setVisible(false);

dispose();

} else if (src == depositBtn) {

deposit();

} else if (src == withdrawBtn) {

withdraw();

} else if (src == transferBtn) {

String to = JOptionPane.showInputDialog("Recipient account number");

if (verifyAccountNumber(to)) {

transfer(to);

} else {

JOptionPane.showMessageDialog(this, "Account number not found");

}

}

} catch (Exception er) {

er.printStackTrace();

JOptionPane.showMessageDialog(this, "Failed to perform the action");

}

}

private boolean verifyAccountNumber(String accountNumber) {

try {

Connection conn = ConnectionManager.getInstance().getConnection();

PreparedStatement ps = conn.prepareStatement(

"SELECT accountNumber FROM account WHERE accountNumber=?"

);

ps.setString(1, accountNumber);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

return true;

}

} catch (Exception er) {

er.printStackTrace();

JOptionPane.showMessageDialog(this, "Failed to verify account.");

}

return false;

}

private void transfer(String to) {

try {

double depositAmount = Double.parseDouble(

JOptionPane.showInputDialog("Transfer amount")

);

if (depositAmount <= 0 || depositAmount > getAccountBalance()) {

JOptionPane.showMessageDialog(this, "Wrong amount.");

return;

}

Connection conn = ConnectionManager.getInstance().getConnection();

PreparedStatement ps = conn.prepareStatement(

"INSERT INTO transaction(accountNumber, type, amount, date) VALUES(?, ?, ?, ?)"

);

ps.setString(1, accountNumber);

ps.setString(2, "withdraw");

ps.setDouble(3, depositAmount);

ps.setDate(4, new Date(System.currentTimeMillis()));

ps.executeUpdate();

System.out.println(ps);

updateBalance(getAccountBalance() - depositAmount);

ps = conn.prepareStatement(

"INSERT INTO transaction(accountNumber, type, amount, date) VALUES(?, ?, ?, ?)"

);

ps.setString(1, to);

ps.setString(2, "deposit");

ps.setDouble(3, depositAmount);

ps.setDate(4, new Date(System.currentTimeMillis()));

ps.executeUpdate();

System.out.println(ps);

ps = conn.prepareStatement(

"UPDATE account SET balance=balance+? WHERE accountNumber=?"

);

ps.setDouble(1, depositAmount);

ps.setString(2, to);

ps.executeUpdate();

System.out.println(ps);

table.setModel(new TranscationTableModel(this, username));

} catch (Exception e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Failed to transfer balance.");

}

}

private void updateBalance(double newBalance) {

try {

Connection conn = ConnectionManager.getInstance().getConnection();

PreparedStatement ps = conn.prepareStatement(

"UPDATE account SET balance=? WHERE accountNumber=?"

);

ps.setDouble(1, newBalance);

ps.setString(2, accountNumber);

ps.executeUpdate();

System.out.println(ps);

table.setModel(new TranscationTableModel(this, username));

} catch (Exception e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Failed to update balance.");

}

}

private void withdraw() {

try {

double amount = Double.parseDouble(

JOptionPane.showInputDialog("Withdraw amount")

);

if (amount <= 0 || amount > getAccountBalance()) {

JOptionPane.showMessageDialog(this, "Wrong amount.");

return;

}

Connection conn = ConnectionManager.getInstance().getConnection();

PreparedStatement ps = conn.prepareStatement(

"INSERT INTO transaction(accountNumber, type, amount, date) VALUES(?, ?, ?, ?)"

);

ps.setString(1, accountNumber);

ps.setString(2, "withdraw");

ps.setDouble(3, amount);

ps.setDate(4, new Date(System.currentTimeMillis()));

ps.executeUpdate();

System.out.println(ps);

updateBalance(getAccountBalance() - amount);

table.setModel(new TranscationTableModel(this, username));

} catch (Exception e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Failed to withdraw");

}

}

private double getAccountBalance() {

try {

Connection conn = ConnectionManager.getInstance().getConnection();

PreparedStatement ps = conn.prepareStatement(

"SELECT account.balance FROM account, customer WHERE customer.username=? AND account.accountNumber=customer.accountNumber"

);

ps.setString(1, username);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

double balance = rs.getDouble("account.balance");

return balance;

}

} catch (SQLException e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Error! Failed to fetch data.");

}

return 0;

}

private void deposit() {

try {

double depositAmount = Double.parseDouble(

JOptionPane.showInputDialog("Deposit amount")

);

if (depositAmount <= 0) {

JOptionPane.showMessageDialog(this, "Amount must be positive.");

return;

}

Connection conn = ConnectionManager.getInstance().getConnection();

PreparedStatement ps = conn.prepareStatement(

"INSERT INTO transaction(accountNumber, type, amount, date) VALUES(?, ?, ?, ?)"

);

ps.setString(1, accountNumber);

ps.setString(2, "deposit");

ps.setDouble(3, depositAmount);

ps.setDate(4, new Date(System.currentTimeMillis()));

ps.executeUpdate();

System.out.println(ps);

updateBalance(getAccountBalance() + depositAmount);

table.setModel(new TranscationTableModel(this, username));

} catch (Exception e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Failed to deposit.");

}

}

private String getAccountNumber(String username) {

Connection conn = ConnectionManager.getInstance().getConnection();

try {

PreparedStatement ps = conn.prepareStatement(

"SELECT accountNumber FROM customer WHERE username=?"

);

ps.setString(1, username);

System.out.println(ps);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

return rs.getString("accountNumber");

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

}