Kevin Cain

ITMD 411

Final Project

Java class for "TroubleTickets" allows users to create and view support tickets, and has options for administrators to update or delete tickets.

The class extends the JFrame class, indicating that it is a window that can be displayed on the screen. It implements the ActionListener interface, which means it can handle events such as button clicks.

The class defines several member objects, including a DAO (Data Access Object) for performing CRUD (Create, Read, Update, Delete) operations on the database. It also defines JMenu and JMenuItem objects for creating a menu bar, with sub-menu items for opening, viewing, updating, and deleting support tickets.

The constructor for the Tickets class takes a boolean value indicating whether the user is an administrator or not, and calls the createMenu() and prepareGUI() methods to initialize the menu bar and window components.

The createMenu() method initializes the sub-menu items for each main menu item, adds action listeners for each menu item, and sets the font and color for the menu items.

The prepareGUI() method sets the look and feel of the application, creates a JMenuBar object, adds the main menu items to the menu bar, sets the window size, background color, and location, and makes the window visible.

The actionPerformed() method handles the actions for each sub-menu item. If the "Open Ticket" item is selected, it prompts the user for their name and a ticket description, inserts the information into the database, and displays a message box indicating success or failure. If the "View Ticket" item is selected, it retrieves all ticket details from the database and displays them in a JTable object.

The Dao class connects to a MySQL database and allows users to perform CRUD operations on the tickets in the system. It also can create tables in the database, add users to the user table, and close tickets.

The Dao class contains methods for interacting with the database. The getConnection() method sets up the connection with the database. The createTables() method creates the tables in the database. The addUsers() method adds users to the user table. The insertRecords() method inserts a new ticket into the ticket table. The readRecords() method retrieves all tickets from the ticket table. The deleteTicket() method deletes a ticket from the ticket table. The closeTicket() method changes the status of a ticket to closed and adds an end date to the ticket.

The program reads user data from a the CSV file and adds it to the user table in the database. The program also uses prepared statements to prevent SQL injection attacks.

the login class functionality for the gui Troubletickets, login. There is a form with two input fields for username and password, and two buttons for submitting the login credentials or exiting the application. The program uses a database to store user credentials and verify them during the login process. If the user enters valid credentials, the program opens a new window with the help desk application's main interface. If the user enters invalid credentials, the program displays an error message and allows the user to retry up to three times. The program uses the DAO pattern to interact with the database, and it handles SQL exceptions using try-with-resources blocks.

The Java class ticketsJTable provides a method buildTableModel that takes a ResultSet object and returns a DefaultTableModel object.

The buildTableModel method uses the ResultSet metadata to retrieve the column names and number of columns. It then loops through the ResultSet data and retrieves the data for each column. The retrieved data is then stored in a Vector of Vectors, where each Vector represents a row of data. Finally, the DefaultTableModel is created with the Vector of Vectors and the column names and returned.

This method is used for building a JTable object in a Swing GUI application that displays data from a database.

CODE:

package troubleTickets;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Dao {

// instance fields

static Connection connect = null;

Statement statement = null;

// constructor

public Dao() {

}

public Connection getConnection() {

// Setup the connection with the DB

try {

connect = DriverManager

.getConnection("jdbc:mysql://www.papademas.net:3307/tickets?autoReconnect=true&useSSL=false"

+ "&user=fp411&password=411");

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return connect;

}

// CRUD implementation

public void createTables() {

// variables for SQL Query table creations

final String createTicketsTable = "CREATE TABLE kcain\_tickets(ticket\_id INT AUTO\_INCREMENT PRIMARY KEY, ticket\_issuer VARCHAR(30), ticket\_description VARCHAR(200))";

final String createUsersTable = "CREATE TABLE kcain\_users(uid INT AUTO\_INCREMENT PRIMARY KEY, uname VARCHAR(30), upass VARCHAR(30), admin int)";

try {

// execute queries to create tables

statement = getConnection().createStatement();

statement.executeUpdate(createTicketsTable);

statement.executeUpdate(createUsersTable);

System.out.println("Created tables in given database...");

// end create table

// close connection/statement object

statement.close();

connect.close();

} catch (Exception e) {

System.out.println(e.getMessage());

}

// add users to user table

addUsers();

}

public void addUsers() {

// add list of users from userlist.csv file to users table

// variables for SQL Query inserts

String sql;

Statement statement;

BufferedReader br;

List<List<String>> array = new ArrayList<>(); // list to hold (rows & cols)

// read data from file

try {

br = new BufferedReader(new FileReader(new File("./userlist.csv")));

String line;

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

array.add(Arrays.asList(line.split(",")));

}

} catch (Exception e) {

System.out.println("There was a problem loading the file");

}

try {

// Setup the connection with the DB

statement = getConnection().createStatement();

for (List<String> rowData : array) {

sql = "insert into kcain\_users(uname,upass,admin) " + "values('" + rowData.get(0) + "'," + " '"

+ rowData.get(1) + "','" + rowData.get(2) + "');";

statement.executeUpdate(sql);

}

System.out.println("Inserts completed in the given database...");

// close statement object

statement.close();

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

public int insertRecords(String ticketName, String ticketDesc) {

int id = 0;

try {

statement = getConnection().createStatement();

statement.executeUpdate("Insert into kcain\_tickets" + "(ticket\_issuer, ticket\_description) values(" + " '"

+ ticketName + "','" + ticketDesc + "')", Statement.RETURN\_GENERATED\_KEYS);

// retrieve ticket id number newly auto generated upon record insertion

ResultSet resultSet = null;

resultSet = statement.getGeneratedKeys();

if (resultSet.next()) {

// retrieve first field in table

id = resultSet.getInt(1);

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return id;

}

public ResultSet readRecords() {

ResultSet results = null;

try {

statement = connect.createStatement();

results = statement.executeQuery("SELECT \* FROM kcain\_tickets");

//connect.close();

} catch (SQLException e1) {

e1.printStackTrace();

}

return results;

}

public void deleteTicket(int id) {

try {

Statement preparedStatement = null;

String sql = "DELETE FROM kcain\_tickets " + "WHERE tid = ?";

preparedStatement = connect.prepareStatement(sql);

((PreparedStatement) preparedStatement).setInt(1, id);

preparedStatement.executeUpdate(sql);

}

catch (SQLException se) {

se.printStackTrace();

}

}

/\*\*

\* closeTicket() takes in the id of the ticket and changes the ticket's status from open to close and adds a end date to signify the time stamp of when the ticket is closed.

\* @param id the id of the ticket that the user wants to close

\*/

public void closeTicket(int id) {

java.sql.Timestamp date2 = new java.sql.Timestamp(new java.util.Date().getTime());

try {

PreparedStatement preparedStatement = null;

String sql = "UPDATE kcain\_tickets " + "SET status = ?, end\_date = ? WHERE tid = ?";

preparedStatement = connect.prepareStatement(sql);

preparedStatement.setString(1, "close");

preparedStatement.setTimestamp(2, date2);

preparedStatement.setInt(3, id);

preparedStatement.executeUpdate();

}

catch (SQLException se) {

se.printStackTrace();

}

}

/\*\*

\* updateTicket takes the id of the ticket and a String that holds what the user wants to update, with timestap

\* @param id the id of the ticket that the user wants to update

\* @param updates the things that the user wants to update to the ticket\_desc

\*/

public void updateTicket(int id, String updates) {

java.sql.Timestamp date2 = new java.sql.Timestamp(new java.util.Date().getTime());

try {

statement = connect.createStatement();

String sql = "SELECT ticket\_desc FROM lchen\_finalTicketSystem WHERE tid = '"+id+"'";

ResultSet rs = statement.executeQuery(sql);

String oldDesc = "";

if(rs.next()) {

oldDesc = rs.getString(1);

}

String newDesc = oldDesc + " \n\n" + updates + "\n" + "Updates as of " + date2;

PreparedStatement preparedStatement = null;

sql = "UPDATE kcain\_tickets " + "SET ticket\_desc = ? WHERE tid = ?";

preparedStatement = connect.prepareStatement(sql);

preparedStatement.setString(1, newDesc);

preparedStatement.setInt(2, id);

preparedStatement.executeUpdate();

}

catch (SQLException se) {

se.printStackTrace();

}

}

}

package troubleTickets;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.sql.SQLException;

import javax.swing.JFrame;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.UIManager;

@SuppressWarnings("serial")

public class Tickets extends JFrame implements ActionListener {

// class level member objects

Dao dao = new Dao(); // for CRUD operations

Boolean chkIfAdmin = null;

// Main menu object items

private JMenu mnuFile = new JMenu("File");

private JMenu mnuAdmin = new JMenu("Admin");

private JMenu mnuTickets = new JMenu("Tickets");

// Sub menu item objects for all Main menu item objects

JMenuItem mnuItemExit;

JMenuItem mnuItemUpdate;

JMenuItem mnuItemDelete;

JMenuItem mnuItemOpenTicket;

JMenuItem mnuItemViewTicket;

public Tickets(Boolean isAdmin) {

chkIfAdmin = isAdmin;

createMenu();

prepareGUI();

}

private void createMenu() {

// Initialize sub menu items for File main menu

mnuItemExit = new JMenuItem("Exit");

mnuFile.add(mnuItemExit);

// Initialize sub menu items for Tickets main menu

mnuItemOpenTicket = new JMenuItem("Open Ticket");

mnuTickets.add(mnuItemOpenTicket);

mnuItemViewTicket = new JMenuItem("View Ticket");

mnuTickets.add(mnuItemViewTicket);

// Add action listeners for each desired menu item

mnuItemExit.addActionListener(this);

mnuItemOpenTicket.addActionListener(this);

mnuItemViewTicket.addActionListener(this);

// Set font and foreground color for menu items

Font font = new Font("Arial", Font.PLAIN, 14);

Color color = new Color(0, 102, 204);

mnuItemExit.setFont(font);

mnuItemOpenTicket.setFont(font);

mnuItemViewTicket.setFont(font);

mnuItemExit.setForeground(color);

mnuItemOpenTicket.setForeground(color);

mnuItemViewTicket.setForeground(color);

// Only show the Admin menu if the user is an admin

if (chkIfAdmin != null && chkIfAdmin) {

// Initialize sub menu items for Admin main menu

mnuItemUpdate = new JMenuItem("Update Ticket");

mnuAdmin.add(mnuItemUpdate);

mnuItemDelete = new JMenuItem("Delete Ticket");

mnuAdmin.add(mnuItemDelete);

// Add action listeners for each desired menu item

mnuItemUpdate.addActionListener(this);

mnuItemDelete.addActionListener(this);

// Set font and foreground color for menu items

mnuItemUpdate.setFont(font);

mnuItemDelete.setFont(font);

mnuItemUpdate.setForeground(color);

mnuItemDelete.setForeground(color);

}

}

private void prepareGUI() {

try {

// Nimbus look and feel

UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");

} catch (Exception e) {

// If Nimbus is not available, set the system look and feel.

e.printStackTrace();

}

// create JMenu bar

JMenuBar bar = new JMenuBar();

bar.add(mnuFile); // add main menu items in order, to JMenuBar

bar.add(mnuAdmin);

bar.add(mnuTickets);

// add menu bar components to frame

setJMenuBar(bar);

addWindowListener(new WindowAdapter() {

// define a window close operation

public void windowClosing(WindowEvent wE) {

System.exit(0);

}

});

// set frame options

setSize(400, 400);

getContentPane().setBackground(Color.LIGHT\_GRAY);

setLocationRelativeTo(null);

setVisible(true);

}

@Override

public void actionPerformed(ActionEvent e) {

// implement actions for sub menu items

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

System.exit(0);

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

// get ticket information

String ticketName = JOptionPane.showInputDialog(null, "Enter your name");

String ticketDesc = JOptionPane.showInputDialog(null, "Enter a ticket description");

// insert ticket information to database

int id = dao.insertRecords(ticketName, ticketDesc);

// display results if successful or not to console / dialog box

if (id != 0) {

System.out.println("Ticket ID : " + id + " created successfully!!!");

JOptionPane.showMessageDialog(null, "Ticket id: " + id + " created");

} else

System.out.println("Ticket cannot be created!!!");

}

else if (e.getSource() == mnuItemViewTicket) {

// retrieve all tickets details for viewing in JTable

try {

JTable jt = new JTable(ticketsJTable.buildTableModel(dao.readRecords()));

jt.setBounds(30, 40, 200, 400);

JScrollPane sp = new JScrollPane(jt);

add(sp);

setVisible(true); // refreshes or repaints frame on screen

} catch (SQLException e1) {

e1.printStackTrace();

}

}

}

}

**package** troubleTickets;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**import** javax.swing.JPasswordField;

**import** javax.swing.JTextField;

@SuppressWarnings("serial")

**public** **class** Login **extends** JFrame {

Dao conn;

**public** Login() {

**super**("My Help Desk Application Login");

conn = **new** Dao();

conn.createTables();

setSize(500, 310);

setLocationRelativeTo(**null**); // centers window

// SET UP CONTROLS

JLabel lblUsername = **new** JLabel("Username:", JLabel.***RIGHT***);

JLabel lblPassword = **new** JLabel("Password:", JLabel.***RIGHT***);

JLabel lblStatus = **new** JLabel(" ", JLabel.***CENTER***);

JTextField txtUname = **new** JTextField(10);

JPasswordField txtPassword = **new** JPasswordField();

JButton btnSubmit = **new** JButton("Login");

JButton btnExit = **new** JButton("Exit");

// SET FONTS

Font labelFont = **new** Font(Font.***SANS\_SERIF***, Font.***BOLD***, 16);

Font buttonFont = **new** Font(Font.***SANS\_SERIF***, Font.***PLAIN***, 14);

lblUsername.setFont(labelFont);

lblPassword.setFont(labelFont);

lblStatus.setFont(labelFont);

btnSubmit.setFont(buttonFont);

btnExit.setFont(buttonFont);

// SET COLORS

Color bgColor = **new** Color(180, 180, 180);

Color textColor = **new** Color(40, 40, 40);

lblUsername.setForeground(textColor);

lblPassword.setForeground(textColor);

lblStatus.setForeground(textColor);

getContentPane().setBackground(bgColor);

// SET BUTTON SIZE

btnSubmit.setPreferredSize(**new** java.awt.Dimension(100, 35));

btnExit.setPreferredSize(**new** java.awt.Dimension(100, 35));

// ADD OBJECTS TO FRAME

setLayout(**new** GridLayout(4, 2));

add(lblUsername); // 1st row filler

add(txtUname);

add(lblPassword); // 2nd row

add(txtPassword);

add(btnSubmit); // 3rd row

add(btnExit);

add(lblStatus); // 4th row

btnSubmit.addActionListener(**new** ActionListener() {

**int** count = 0; // count agent

@Override

**public** **void** actionPerformed(ActionEvent e) {

**boolean** admin = **false**;

count = count + 1;

// verify credentials of user (MAKE SURE TO CHANGE TO YOUR TABLE NAME BELOW)

String query = "SELECT \* FROM kcain\_users WHERE uname = ? and upass = ?;";

**try** (PreparedStatement stmt = conn.getConnection().prepareStatement(query)) {

stmt.setString(1, txtUname.getText());

stmt.setString(2, txtPassword.~~getText~~());

ResultSet rs = stmt.executeQuery();

**if** (rs.next()) {

admin = rs.getBoolean("admin"); // get table column value

**new** Tickets(admin); //open Tickets file / GUI interface

setVisible(**false**); // HIDE THE FRAME

dispose(); // CLOSE OUT THE WINDOW

} **else**

lblStatus.setText("Try again! " + (3 - count) + " / 3 attempt(s) left");

} **catch** (SQLException ex) {

ex.printStackTrace();

}

}

});

btnExit.addActionListener(e -> System.*exit*(0));

setVisible(**true**); // SHOW THE FRAME

}

**public** **static** **void** main(String[] args) {

**new** Login();

}

}

package troubleTickets;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import java.util.Vector;

import javax.swing.table.DefaultTableModel;

public class ticketsJTable {

@SuppressWarnings("unused")

private final DefaultTableModel tableModel = new DefaultTableModel();

public static DefaultTableModel buildTableModel(ResultSet rs) throws SQLException {

ResultSetMetaData metaData = rs.getMetaData();

// names of columns

Vector<String> columnNames = new Vector<String>();

int columnCount = metaData.getColumnCount();

for (int column = 1; column <= columnCount; column++) {

columnNames.add(metaData.getColumnName(column));

}

// data of the table

Vector<Vector<Object>> data = new Vector<Vector<Object>>();

while (rs.next()) {

Vector<Object> vector = new Vector<Object>();

for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {

vector.add(rs.getObject(columnIndex));

}

data.add(vector);

}

// return data/col.names for JTable

return new DefaultTableModel(data, columnNames);

}

}

Screenshots:

Graphical user interface, application

Description automatically generated![Graphical user interface, text, application

Description automatically generated]()![Graphical user interface, text

Description automatically generated]()![Graphical user interface, text, application

Description automatically generated]()![A picture containing text, stationary, writing implement, screenshot

Description automatically generated]()![Table

Description automatically generated]()![Graphical user interface, application, table, Excel

Description automatically generated]()Graphical user interface, application

Description automatically generated![Graphical user interface, text, application, email

Description automatically generated]()