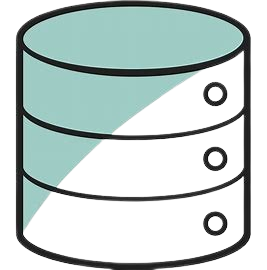


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Programme | : | B. Tech (CSE) | Semester | : | Winter Semester 2023-2024 |
| Course Code &  Course Title | : | BCSE302L & Database Systems | Slot (s) | : | A2+TA2 |
| Faculty | : | Dr. M. Premalatha | Marks | : | 10 |

**DIGITAL ASSIGNMENT – PROJECT**

**Connection Of Front-End Scripting Languages with Databases**

**DATABASE SIMPLIFIER**



**By**

**Register Number: 22BCE5007**

**Name: Rohit Sinha**

* **Introduction**

In the realm of modern technology, data serves as the lifeblood of countless applications and systems, making efficient interaction with databases paramount. The seamless connection between front-end scripting languages and databases is pivotal in shaping user experiences and streamlining data management processes. This project endeavors to bridge the gap between front-end scripting languages and databases, enhancing user-friendliness and accessibility in data interaction.

The project addresses the challenge of facilitating smooth communication between front-end scripts and databases, which can often be complex and cumbersome, requiring intricate handling of queries, data retrieval, and presentation. By developing a robust solution that fosters intuitive and efficient data exchange, this project aims to mitigate these complexities.

At its core, the project recognizes that modern web applications rely heavily on dynamic data exchange, with users expecting real-time access to information, seamless updates, and intuitive interactions with underlying data repositories. However, traditional methods of interfacing front-end scripts with databases often introduce bottlenecks, security vulnerabilities, and scalability issues.

The proposed solution introduces a comprehensive framework that streamlines communication between front-end scripting languages and databases. By leveraging cutting-edge technologies and adhering to industry best practices, the project aims to create a unified, standardized approach that abstracts away complexities, enabling developers to focus on delivering rich user experiences.

Key objectives include enhancing security through robust authentication mechanisms, input validation, and encrypted communication channels, as well as addressing scalability and performance optimization through advanced caching, query optimization, and load balancing strategies.

Furthermore, the project emphasizes user-friendliness and accessibility, providing intuitive interfaces and comprehensive documentation to empower developers of varying skill levels to leverage the power of database interactions.

Ultimately, this project represents a significant step forward in bridging the gap between front-end scripting languages and databases, paving the way for more robust, efficient, and user-friendly applications that harness the full potential of data-driven experiences.

* **Project Overview:**

The project represents a comprehensive endeavour aimed at revolutionizing the interaction between front-end scripting languages and databases by seamlessly integrating Java Swing for the user interface (UI) alongside a Flask server and MySQL database. This multifaceted approach encompasses a range of functionalities, from query submission to result presentation, ensuring a cohesive and intuitive user experience throughout the data management process.

* **Key Components:**

1. **Flask Server:** Positioned as the central communication conduit, the Flask server orchestrates the flow of data between the Java Swing UI and the MySQL database. Leveraging Flask's lightweight and adaptable framework, the server adeptly handles incoming requests, processes queries, and facilitates interactions with the database layer. Its role extends beyond mere data transmission to encompass query optimization, error handling, and performance tuning, thereby ensuring optimal functionality and reliability.
2. **MySQL Database:** Serving as the repository for structured data, the MySQL database lies at the core of the system, storing and managing information accessed and manipulated by the front-end UI. The integration with the Flask server enables seamless execution of SQL queries, data retrieval, and manipulation, fostering a robust and scalable data management solution. With features such as transaction support, data indexing, and concurrency control, the MySQL database ensures data integrity and consistency, crucial for mission-critical applications across various domains.
3. **Java Swing UI:** The incorporation of Java Swing for the UI component elevates the project's user experience to new heights, offering a rich and interactive interface for users to engage with the system. From intuitive form inputs to dynamic data visualization, the Swing framework empowers developers to design elegant and user-friendly interfaces that resonate with modern design principles. Leveraging Swing's extensive widget library and event-driven architecture, the UI facilitates seamless navigation, query submission, and result presentation, enhancing usability and accessibility for users of all proficiency levels.

* **Project Workflow:**

The project represents a comprehensive endeavour aimed at revolutionizing the interaction between front-end scripting languages and databases by seamlessly integrating Java Swing for the user interface (UI) alongside a Flask server and MySQL database. This multifaceted approach encompasses a range of functionalities, from query submission to result presentation, ensuring a cohesive and intuitive user experience throughout the data management process.

* **User Authentication and Database Connection:**

1. The user opens the application and is prompted to login using their database credentials.
2. The Flask server receives the login request and validates the provided credentials.
3. Upon successful validation, the Flask server establishes a connection to the database using a dedicated module.
4. The connection status is relayed back to the Java client, indicating whether the connection was successful or if any errors occurred.
5. If successful, the user is directed to the query page; otherwise, an error message is displayed, prompting the user to retry the login process.

* **Query Submission and Validation:**

1. On the query page, the user inputs the desired SQL query into the designated field.
2. The Java client sends the query to the Flask server for validation.
3. The Flask server verifies the query syntax and semantics to ensure its compatibility with the database.
4. If the query is validated successfully, the Flask server returns a response code of 200 to the Java client, indicating readiness for execution.
5. In case of validation failure, a response code of 400 is returned, prompting the user to correct the query.

* **Query Execution and Result Handling:**

1. Upon receiving confirmation of query validity, the Java client sends the validated query to the Flask server for execution.
2. The Flask server executes the query against the connected database and retrieves the result set, if any.
3. If a result set is obtained, it is serialized into JSON format by the Flask server and transmitted back to the Java client.
4. The Java client parses the JSON result set using a custom parser and populates a JTable component with the retrieved data.
5. If the query does not return a result set, the Java client proceeds without displaying a results page.

* **Displaying Results and Exporting Data:**

1. If a result set is received, the Java client displays the results in a new page, termed the "Results Page," using the populated JTable component.
2. The Results Page allows users to visualize and interact with the queried data in a tabular format.
3. Additionally, the Results Page features an "Export to CSV" button, enabling users to save the current result set as a CSV file for further analysis or archival purposes.

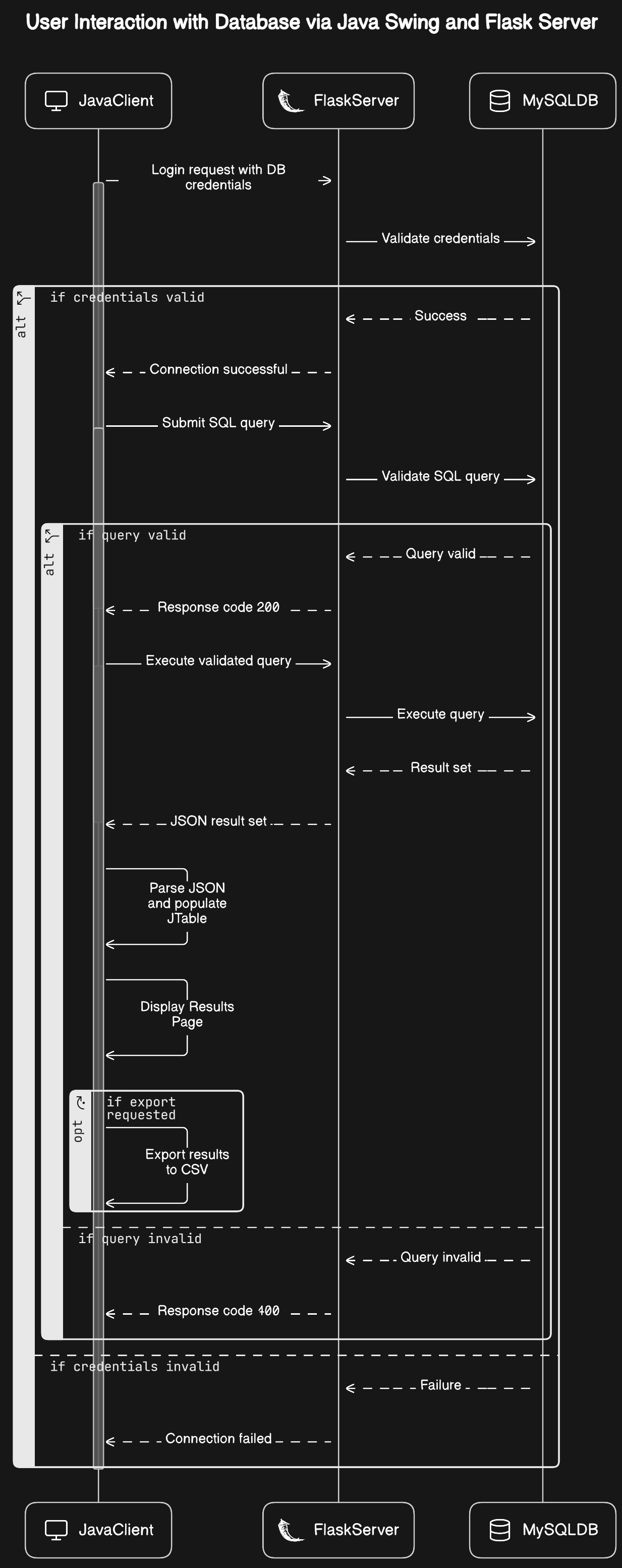
* **User Interaction and Navigation:**

1. Throughout the workflow, users interact with the Java Swing UI to input queries, navigate between pages, and interact with the displayed data.
2. Clear and intuitive navigation elements guide users through the application's functionalities, ensuring a seamless and user-friendly experience.

* **Error Handling and Feedback:**

1. The system provides informative error messages and feedback at each stage of the workflow, guiding users in resolving issues and completing tasks effectively.
2. Error handling mechanisms in both the Java client and Flask server ensure robustness and reliability, minimizing disruptions to the user experience and facilitating smooth operation under varying conditions.

*PTO*→



* **Source Code:**
* **GitHub –** [DatabaseSimplifier.git](https://github.com/Azaken1248/easyDBMSLAB)
* **Zip File -** [DatabaseSimplifier.zip](https://drive.google.com/file/d/1RwqMKp995Cz-yWIFlVmBnmi-n_Oa6d18/view?usp=sharing)
* **Java Front End =>**

1. **SQLApp.java**

public class SQLApp {

    public static void main(String[] args) {

        new ConnectionPage();

    }

}

1. **ConnectionPage.java**

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import javax.swing.border.LineBorder;

import java.awt.\*;

import java.awt.event.\*;

import Connectionclient.Client;

public class ConnectionPage implements ActionListener, MouseListener {

        Dimension screenSize;

        ImageIcon icon;

        JFrame window;

        JPanel header, body, footer;

        JLabel welcomeLabel, descriptorLabel, hostLabel, userLabel, passLabel, conLabel;

        JTextField userField, hostField;

        JPasswordField passField;

        JButton connectButton;

        ConnectionPage() {

                icon = new ImageIcon("./Assets/icon.png");

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

                int screenWidth = (int) screenSize.getWidth();

                int screenHeight = (int) screenSize.getHeight();

                window = new JFrame("SQL App");

                window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

                window.setIconImage(icon.getImage());

                window.setSize(screenWidth / 2, screenHeight / 2);

                window.setResizable(false);

                window.getContentPane().setBackground(Color.black);

                window.setLayout(new BorderLayout());

                window.setLocationRelativeTo(null);

                header = new JPanel();

                header.setBackground(Color.black);

                header.setBorder(BorderFactory.createEmptyBorder(30, 50, 30, 50));

                header.setLayout(new FlowLayout());

                body = new JPanel();

                body.setBackground(Color.black);

                body.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 90, 10, 90),

                                new LineBorder(Color.white, 3, true)));

                body.setLayout(new FlowLayout());

                footer = new JPanel();

                footer.setBackground(Color.black);

                footer.setLayout(new FlowLayout());

                welcomeLabel = new JLabel("Welcome To SQL Connector");

                welcomeLabel.setBackground(Color.black);

                welcomeLabel.setForeground(Color.white);

                welcomeLabel.setFont(new Font("Calibri", 0, 32));

                descriptorLabel = new JLabel(

                                "                                Enter Connection Detail                                ");

                descriptorLabel.setBackground(Color.black);

                descriptorLabel.setForeground(Color.white);

                descriptorLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                descriptorLabel.setFont(new Font("Calibri", 0, 24));

                hostLabel = new JLabel("Hostname : ");

                hostLabel.setBackground(Color.black);

                hostLabel.setForeground(Color.white);

                hostLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                hostLabel.setFont(new Font("Calibri", 0, 20));

                userLabel = new JLabel("Username : ");

                userLabel.setBackground(Color.black);

                userLabel.setForeground(Color.white);

                userLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                userLabel.setFont(new Font("Calibri", 0, 20));

                passLabel = new JLabel("Password :  ");

                passLabel.setBackground(Color.black);

                passLabel.setForeground(Color.white);

                passLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                passLabel.setFont(new Font("Calibri", 0, 20));

                conLabel = new JLabel("");

                conLabel.setBackground(Color.black);

                conLabel.setForeground(Color.white);

                conLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                conLabel.setFont(new Font("Calibri", 0, 20));

                hostField = new JTextField(30);

                hostField.setBackground(Color.black);

                hostField.setForeground(Color.white);

                hostField.setCaretColor(Color.white);

                hostField.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 10, 10, 10),

                                new LineBorder(Color.white, 3, true)));

                hostField.setFont(new Font("Calibri", 0, 20));

                userField = new JTextField(30);

                userField.setBackground(Color.black);

                userField.setForeground(Color.white);

                userField.setCaretColor(Color.white);

                userField.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 10, 10, 10),

                                new LineBorder(Color.white, 3, true)));

                userField.setFont(new Font("Calibri", 0, 20));

                passField = new JPasswordField(30);

                passField.setBackground(Color.black);

                passField.setForeground(Color.white);

                passField.setCaretColor(Color.white);

                passField.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 10, 30, 10),

                                new LineBorder(Color.white, 3, true)));

                passField.setFont(new Font("Calibri", 0, 20));

                connectButton = new JButton("           Connect           ");

                connectButton.setFocusable(false);

                connectButton.setBackground(Color.white);

                connectButton.setForeground(Color.black);

                connectButton.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 10, 0, 10),

                                new LineBorder(Color.white, 2, true)));

                connectButton.setFont(new Font("Calibri", 0, 20));

                // connectButton.setSize(20, 20);

                connectButton.addMouseListener(this);

                connectButton.addActionListener(this);

                header.add(welcomeLabel);

                body.add(descriptorLabel);

                body.add(hostLabel);

                body.add(hostField);

                body.add(userLabel);

                body.add(userField);

                body.add(passLabel);

                body.add(passField);

                body.add(connectButton);

                footer.add(conLabel);

                window.add(header, BorderLayout.NORTH);

                window.add(body, BorderLayout.CENTER);

                window.add(footer, BorderLayout.SOUTH);

                window.setVisible(true);

        }

        @Override

        public void actionPerformed(ActionEvent e) {

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

                        conLabel.setText("Connecting...");

                        String host, username, password;

                        host = hostField.getText();

                        username = userField.getText();

                        password = String.valueOf(passField.getPassword());

                        System.out.println("host: " + host + "\nusername: " + username + "\npassword: " + password);

                        int code = Client.connectToDB(host, username, password);

                        if (code != 200) {

                                conLabel.setText("Falied to Establish Connection (" + code + ")");

                                conLabel.setForeground(Color.red);

                        } else {

                                conLabel.setText("Connection Successful!");

                                conLabel.setForeground(Color.green);

                                new QueryPage();

                                window.dispose();

                        }

                }

        }

        @Override

        public void mouseClicked(MouseEvent e) {

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

                        connectButton.setBackground(Color.white);

                        connectButton.setForeground(Color.black);

                }

        }

        @Override

        public void mousePressed(MouseEvent e) {

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

                        connectButton.setBackground(Color.white);

                        connectButton.setForeground(Color.black);

                }

        }

        @Override

        public void mouseReleased(MouseEvent e) {

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

                        connectButton.setBackground(Color.white);

                        connectButton.setForeground(Color.black);

                }

        }

        @Override

        public void mouseEntered(MouseEvent e) {

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

                        connectButton.setBackground(Color.white);

                        connectButton.setForeground(Color.black);

                }

        }

        @Override

        public void mouseExited(MouseEvent e) {

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

                        connectButton.setBackground(Color.white);

                        connectButton.setForeground(Color.black);

                }

        }

}

1. **Cilent.java**

package Connectionclient;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.net.HttpURLConnection;

import java.net.URL;

public class Client {

    /\*

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

     \* try {

     \* // Send data to the server

     \* sendDataToServer("{\"query\": \"show databases;\"}");

     \*

     \* // Receive data from the server and store it as a file

     \* receiveDataFromServerAndStoreAsFile();

     \* } catch (Exception e) {

     \* e.printStackTrace();

     \* }

     \* }

     \*/

    public static int sendDataToServer(String jsonData) throws Exception {

        URL url = new URL("http://localhost:5000/data");

        HttpURLConnection conn = (HttpURLConnection) url.openConnection();

        conn.setRequestMethod("POST");

        conn.setRequestProperty("Content-Type", "application/json");

        conn.setDoOutput(true);

        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));

        out.write(jsonData);

        out.flush();

        out.close();

        int responseCode = conn.getResponseCode();

        // System.out.println("Response Code from Server (Sending Data): " +

        // responseCode);

        conn.disconnect();

        return responseCode;

    }

    public static String receiveDataFromServerAndStoreAsFile() throws Exception {

        URL url = new URL("http://localhost:5000/data"); // Corrected URL

        HttpURLConnection conn = (HttpURLConnection) url.openConnection();

        conn.setRequestMethod("GET");

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));

        String inputLine;

        StringBuffer content = new StringBuffer();

        while ((inputLine = in.readLine()) != null) {

            content.append(inputLine);

        }

        in.close();

        conn.disconnect();

        // Store received data as a file

        String receivedData = content.toString();

        BufferedWriter writer = new BufferedWriter(new FileWriter("received\_data.json"));

        writer.write(receivedData);

        writer.close();

        System.out.println("Received data has been stored as 'received\_data.txt'");

        return receivedData;

    }

    public static int connectToDB(String host, String user, String pass) {

        String json\_data = "{\"host\": \"" + host + "\", \"user\": \"" + user + "\", \"password\": \"" + pass + "\"}";

        try {

            int response = sendDataToServer(json\_data);

            return response;

        } catch (Exception e) {

            return -1;

        }

    }

    public static int sendQuery(String query) {

        String query\_data = "{\"query\":\"" + query + "\"}";

        try {

            int response = sendDataToServer(query\_data);

            return response;

        } catch (Exception e) {

            return -1;

        }

    }

    public static String retrieveResults() {

        try {

            return receiveDataFromServerAndStoreAsFile();

        } catch (Exception e) {

            return "";

        }

    }

}

1. **QueryPage.java**

import java.util.Set;

import java.util.List;

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import javax.swing.border.LineBorder;

import javax.swing.table.DefaultTableModel;

import Connectionclient.Client;

import java.awt.\*;

import java.awt.event.\*;

//import Connectionclient.Client;

public class QueryPage implements ActionListener, MouseListener {

        Dimension screenSize;

        ImageIcon icon;

        JFrame window;

        JPanel header, body, footer;

        JLabel welcomeLabel, descriptorLabel, queryLabel, displayLabel;

        JTextField queryField;

        JButton executeButton;

        JTable resultsTable;

        DefaultTableModel tableModel;

        public static String[] getCols() {

                Object[] cols;

                String[] ret\_cols;

                JSONReader reader = new JSONReader();

                String data = reader.readJsonFromFile("received\_data.json");

                Set<String> cols\_set = reader.getKeysFromJSON(data);

                cols = cols\_set.toArray();

                ret\_cols = new String[cols.length];

                for (int i = 0; i < cols.length; i++) {

                        String col = cols[i].toString();

                        ret\_cols[i] = col;

                }

                return ret\_cols;

        }

        public static Object[][] getData(String[] cols) {

                Object[][] dataVals = null;

                JSONReader reader = new JSONReader();

                String data = reader.readJsonFromFile("received\_data.json");

                List<Object> vals = reader.getArrayValues(data, cols[0]);

                dataVals = new Object[cols.length][vals.size()];

                for (int i = 0; i < cols.length; i++) {

                        vals = reader.getArrayValues(data, cols[i]);

                        for (int j = 0; j < vals.size(); j++) {

                                String check = vals.get(j).toString();

                                check = check.replace("[", "").replace("]", "");

                                check = check.replace("\"", "");

                                check = check.replace(" ", "");

                                dataVals[i][j] = (Object) check;

                        }

                }

                return dataVals;

        }

        public static Object[][] transposeArray(Object[][] array) {

                int rows = array.length;

                int cols = array[0].length;

                Object[][] transposedArray = new Object[cols][rows];

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

                        for (int j = 0; j < cols; j++) {

                                transposedArray[j][i] = array[i][j];

                        }

                }

                return transposedArray;

        }

        QueryPage() {

                icon = new ImageIcon("./Assets/icon.png");

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

                int screenWidth = (int) screenSize.getWidth();

                int screenHeight = (int) screenSize.getHeight();

                window = new JFrame("SQL App");

                window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

                window.setIconImage(icon.getImage());

                window.setSize(screenWidth / 2, screenHeight / 2);

                window.setResizable(false);

                window.getContentPane().setBackground(Color.black);

                window.setLayout(new BorderLayout());

                window.setLocationRelativeTo(null);

                header = new JPanel();

                header.setBackground(Color.black);

                header.setBorder(BorderFactory.createEmptyBorder(30, 50, 30, 50));

                header.setLayout(new FlowLayout());

                body = new JPanel();

                body.setBackground(Color.black);

                body.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 90, 10, 90),

                                new LineBorder(Color.white, 3, true)));

                body.setLayout(new FlowLayout());

                footer = new JPanel();

                footer.setBackground(Color.black);

                footer.setBorder(BorderFactory.createEmptyBorder(120, 90, 10, 90));

                footer.setLayout(new FlowLayout());

                welcomeLabel = new JLabel("Welcome To SQL Connector");

                welcomeLabel.setBackground(Color.black);

                welcomeLabel.setForeground(Color.white);

                welcomeLabel.setFont(new Font("Calibri", 0, 32));

                descriptorLabel = new JLabel(

                                "                                    Enter Query                                  ");

                descriptorLabel.setBackground(Color.black);

                descriptorLabel.setForeground(Color.white);

                descriptorLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                descriptorLabel.setFont(new Font("Calibri", 0, 24));

                queryLabel = new JLabel("Enter Query : ");

                queryLabel.setBackground(Color.black);

                queryLabel.setForeground(Color.white);

                queryLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                queryLabel.setFont(new Font("Calibri", 0, 20));

                displayLabel = new JLabel("");

                displayLabel.setBackground(Color.black);

                displayLabel.setForeground(Color.white);

                displayLabel.setBorder(new EmptyBorder(10, 10, 10, 10));

                displayLabel.setFont(new Font("Calibri", 0, 20));

                queryField = new JTextField(30);

                queryField.setBackground(Color.black);

                queryField.setForeground(Color.white);

                queryField.setCaretColor(Color.white);

                queryField.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 10, 0, 10),

                                new LineBorder(Color.white, 3, true)));

                queryField.setFont(new Font("Calibri", 0, 20));

                executeButton = new JButton("           Execute           ");

                executeButton.setFocusable(false);

                executeButton.setBackground(Color.black);

                executeButton.setForeground(Color.white);

                executeButton.setSize(new Dimension(100, 100));

                executeButton.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 10, 10, 10),

                                new LineBorder(Color.white, 2, true)));

                executeButton.setFont(new Font("Calibri", 0, 20));

                // executeButton.setSize(20, 20);

                executeButton.addMouseListener(this);

                executeButton.addActionListener(this);

                header.add(welcomeLabel);

                body.add(descriptorLabel);

                body.add(queryLabel);

                body.add(queryField);

                body.add(executeButton);

                footer.add(displayLabel);

                window.add(header, BorderLayout.NORTH);

                window.add(body, BorderLayout.CENTER);

                window.add(footer, BorderLayout.SOUTH);

                window.setVisible(true);

        }

        @Override

        public void actionPerformed(ActionEvent e) {

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

                        displayLabel.setText("Executing...");

                        String query = queryField.getText();

                        int code = Client.sendQuery(query);

                        if (code != 200) {

                                displayLabel.setForeground(Color.red);

                                displayLabel.setText("Query execution Failed!! (CODE: " + code + ")");

                        } else {

                                displayLabel.setForeground(Color.green);

                                displayLabel.setText("Query Sent Succesfully!");

                        }

                        String results = Client.retrieveResults();

                        if (!(results.equals(""))) {

                                displayLabel.setForeground(Color.green);

                                displayLabel.setText("Query Executed Succesfully!");

                                String[] cols = getCols();

                                Object[][] data = getData(cols);

                                data = transposeArray(data);

                                new ResultsWindow(cols, data);

                        }

                }

        }

        @Override

        public void mouseClicked(MouseEvent e) {

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

                        executeButton.setBackground(Color.white);

                        executeButton.setForeground(Color.black);

                }

        }

        @Override

        public void mousePressed(MouseEvent e) {

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

                        executeButton.setBackground(Color.white);

                        executeButton.setForeground(Color.black);

                }

        }

        @Override

        public void mouseReleased(MouseEvent e) {

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

                        executeButton.setBackground(Color.white);

                        executeButton.setForeground(Color.black);

                }

        }

        @Override

        public void mouseEntered(MouseEvent e) {

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

                        executeButton.setBackground(Color.white);

                        executeButton.setForeground(Color.black);

                }

        }

        @Override

        public void mouseExited(MouseEvent e) {

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

                        executeButton.setBackground(Color.black);

                        executeButton.setForeground(Color.white);

                }

        }

}

1. **JSONReader.java**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.\*;

public class JSONReader {

    /\*

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

     \* // Example usage:

     \* JSONReader jsonReader = new JSONReader();

     \* String json = jsonReader.readJsonFromFile("received\_data.json");

     \*

     \* Set<String> keys = jsonReader.getKeysFromJSON(json);

     \* System.out.println("Keys with array values: " + keys);

     \*

     \* for (String key : keys) {

     \* List<Object> values = jsonReader.getArrayValues(json, key);

     \* System.out.println("Array values for key '" + key + "': " + values);

     \* }

     \* }

     \*/

    public String readJsonFromFile(String filename) {

        StringBuilder jsonBuilder = new StringBuilder();

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

            String line;

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

                jsonBuilder.append(line);

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

        return jsonBuilder.toString();

    }

    public Set<String> getKeysFromJSON(String json) {

        Set<String> keys = new HashSet<>();

        // Find keys with array values

        int start = json.indexOf('[');

        while (start != -1) {

            // Find the key preceding this array value

            int keyStart = json.lastIndexOf('"', start - 1);

            int keyEnd = json.lastIndexOf('"', keyStart - 1);

            String key = json.substring(keyEnd + 1, keyStart);

            keys.add(key);

            // Move to the next array value

            start = json.indexOf('[', start + 1);

        }

        return keys;

    }

    public List<Object> getArrayValues(String json, String key) {

        List<Object> values = new ArrayList<>();

        // Find the starting index of the array for the given key

        int startIndex = json.indexOf("\"" + key + "\":") + key.length() + 3;

        if (startIndex == -1) {

            return values;

        }

        // Find the ending index of the array for the given key

        int endIndex = json.indexOf("]", startIndex);

        if (endIndex == -1) {

            return values;

        }

        // Extract the array substring

        String arrayStr = json.substring(startIndex, endIndex);

        // Split the array string by commas to get individual values

        String[] valueArr = arrayStr.split(",");

        for (String value : valueArr) {

            // Remove leading and trailing spaces, and remove quotes if present

            value = value.trim().replaceAll("^\"|\"$", "");

            // Check if it's a number

            if (isNumeric(value)) {

                // Parse as double

                values.add(Double.parseDouble(value));

            } else {

                values.add(value); // Treat as string

            }

        }

        return values;

    }

    // Helper method to check if a string is numeric

    private boolean isNumeric(String str) {

        return str.matches("-?\\d+(\\.\\d+)?");

    }

}

1. **ResultsWindow.java**

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import javax.swing.border.LineBorder;

import javax.swing.table.TableModel;

import java.awt.\*;

import java.awt.event.\*;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

public class ResultsWindow implements ActionListener, MouseListener {

    /\*

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

     \* JSONReader reader = new JSONReader();

     \* String data = reader.readJsonFromFile("received\_data.json");

     \* // System.out.println(data);

     \* Set<String> d = reader.getKeysFromJSON(data);

     \* List<Object> v = reader.getArrayValues(data, "Database");

     \* System.out.println(d.toString());

     \* System.out.println(v.toString());

     \* }

     \*/

    Dimension screenSize;

    ImageIcon icon;

    JFrame window;

    JPanel headerPanel, tablePanel, exportPanel;

    JLabel headerLabel;

    JScrollPane tablePane;

    JTable resultsTable;

    JButton exportButton;

    private void writeTableDataToCSV(JTable table, File file) throws IOException {

        TableModel model = table.getModel();

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(file))) {

            // Write column headers

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

                bw.write(model.getColumnName(i));

                if (i < model.getColumnCount() - 1) {

                    bw.write(",");

                }

            }

            bw.newLine();

            // Write rows

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

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

                    bw.write(model.getValueAt(i, j).toString());

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

                        bw.write(",");

                    }

                }

                bw.newLine();

            }

        }

    }

    ResultsWindow(String[] cols, Object[][] data) {

        icon = new ImageIcon("./assets/icon.png");

        window = new JFrame("Results Fetched");

        window.setIconImage(icon.getImage());

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

        window.getContentPane().setBackground(Color.black);

        window.setLayout(new BorderLayout());

        window.setResizable(false);

        headerPanel = new JPanel();

        headerPanel.setBackground(Color.black);

        headerPanel.setBorder(BorderFactory.createEmptyBorder(30, 50, 20, 50));

        headerPanel.setLayout(new FlowLayout());

        tablePanel = new JPanel();

        tablePanel.setBackground(Color.black);

        tablePanel.setBorder(new EmptyBorder(0, 90, 10, 90));

        tablePanel.setLayout(new FlowLayout());

        exportPanel = new JPanel();

        exportPanel.setBackground(Color.black);

        exportPanel.setBorder(BorderFactory.createEmptyBorder(10, 90, 10, 90));

        exportPanel.setLayout(new FlowLayout());

        headerLabel = new JLabel("Results Fetched");

        headerLabel.setBackground(Color.black);

        headerLabel.setForeground(Color.white);

        headerLabel.setFont(new Font("Calibri", 0, 32));

        resultsTable = new JTable(data, cols);

        resultsTable.setBackground(Color.black);

        resultsTable.setForeground(Color.white);

        resultsTable.setEnabled(false);

        tablePane = new JScrollPane(resultsTable);

        tablePane.getViewport().setView(resultsTable);

        tablePane.getViewport().setBackground(null);

        exportButton = new JButton("                      Export  To CSV                      ");

        exportButton.setFocusable(false);

        exportButton.setBackground(Color.black);

        exportButton.setForeground(Color.white);

        exportButton.setSize(new Dimension(100, 100));

        exportButton.setBorder(BorderFactory.createCompoundBorder(new EmptyBorder(10, 10, 10, 10),

                new LineBorder(Color.white, 2, true)));

        exportButton.setFont(new Font("Calibri", 0, 20));

        exportButton.addActionListener(this);

        exportButton.addMouseListener(this);

        tablePanel.add(tablePane);

        exportPanel.add(exportButton);

        window.add(headerPanel, BorderLayout.NORTH);

        window.add(tablePanel, BorderLayout.CENTER);

        window.add(exportPanel, BorderLayout.SOUTH);

        headerPanel.add(headerLabel);

        window.pack();

        window.setVisible(true);

    }

    @Override

    public void actionPerformed(ActionEvent e) {

        JFileChooser fileChooser = new JFileChooser();

        int returnValue = fileChooser.showSaveDialog(null);

        if (returnValue == JFileChooser.APPROVE\_OPTION) {

            File selectedFile = fileChooser.getSelectedFile();

            try {

                writeTableDataToCSV(resultsTable, selectedFile);

                JOptionPane.showMessageDialog(null, "Exported successfully!");

            } catch (IOException ex) {

                ex.printStackTrace();

                JOptionPane.showMessageDialog(null, "An error occurred during the export!");

            }

        }

    }

    @Override

    public void mouseClicked(MouseEvent e) {

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

            exportButton.setBackground(Color.white);

            exportButton.setForeground(Color.black);

        }

    }

    @Override

    public void mousePressed(MouseEvent e) {

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

            exportButton.setBackground(Color.white);

            exportButton.setForeground(Color.black);

        }

    }

    @Override

    public void mouseReleased(MouseEvent e) {

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

            exportButton.setBackground(Color.white);

            exportButton.setForeground(Color.black);

        }

    }

    @Override

    public void mouseEntered(MouseEvent e) {

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

            exportButton.setBackground(Color.white);

            exportButton.setForeground(Color.black);

        }

    }

    @Override

    public void mouseExited(MouseEvent e) {

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

            exportButton.setBackground(Color.black);

            exportButton.setForeground(Color.white);

        }

    }

    /\*

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

     \* new ResultsWindow("", null);

     \* }

     \*/

}