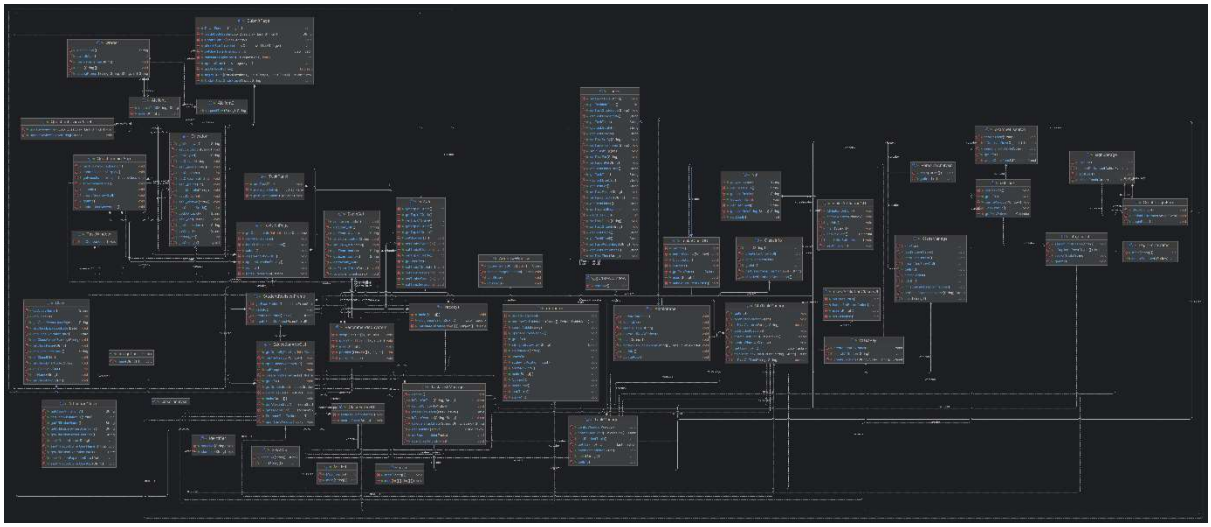
Criterion C: Development

UML diagram

Full image: https://imgur.com/ng2gl50



Full image: https://imgur.com/pAE4Dil

Contents

Graphical User Interface (GUI)....4

HUB....4

Task manager....4

Task app....5

Revision Page (student), Review Page (Class Owner) 5

External Libraries....7

Web API and auto grading....8

Encapsulation....10

Inheritance....11

Complex Methods to customise JTable....12

Security (Hashing password) 14

Web database server....15

2D Array....17

KNN algorithm for recommend system....18

Hash table....20

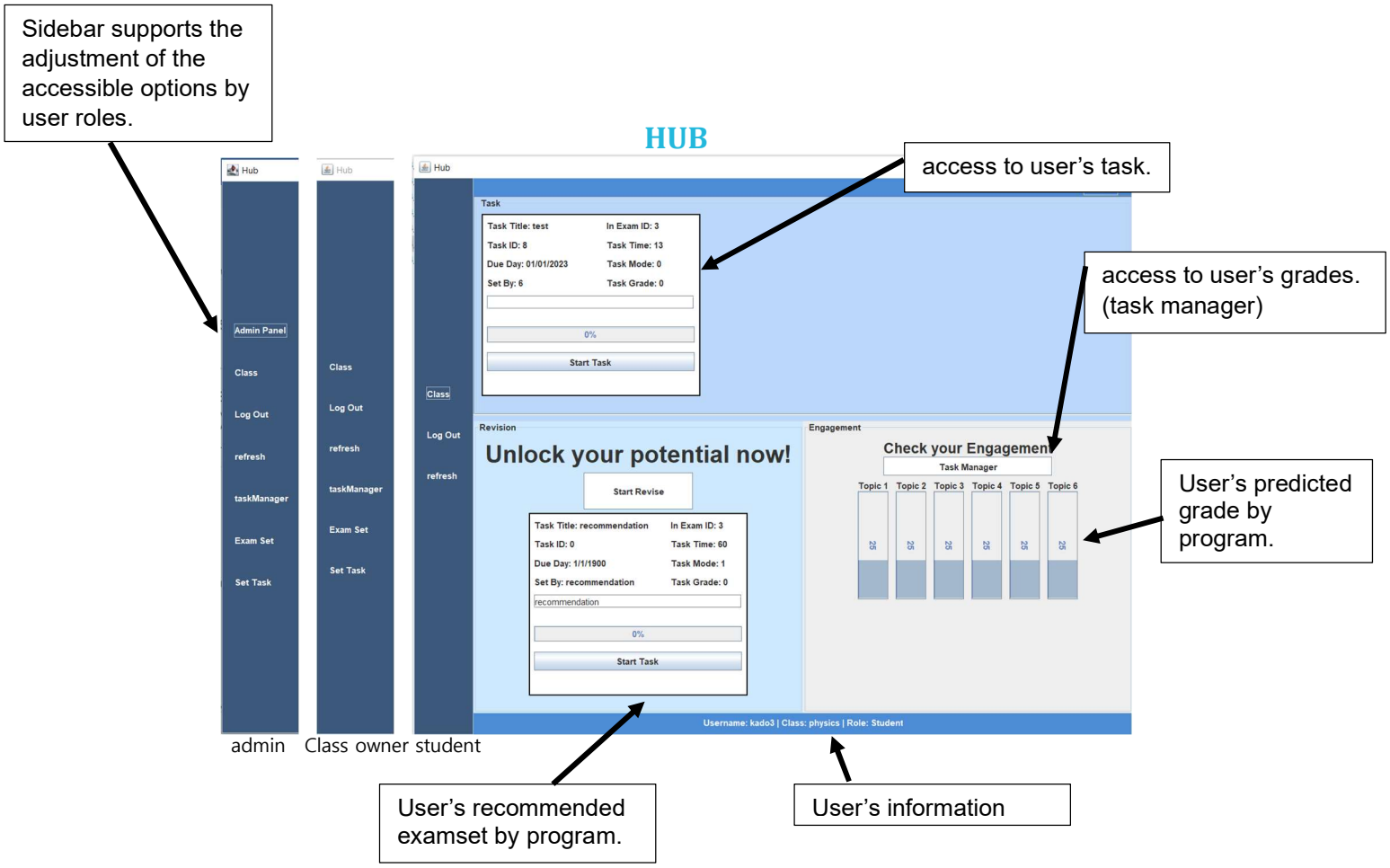
Recursion....21

Error handling....21

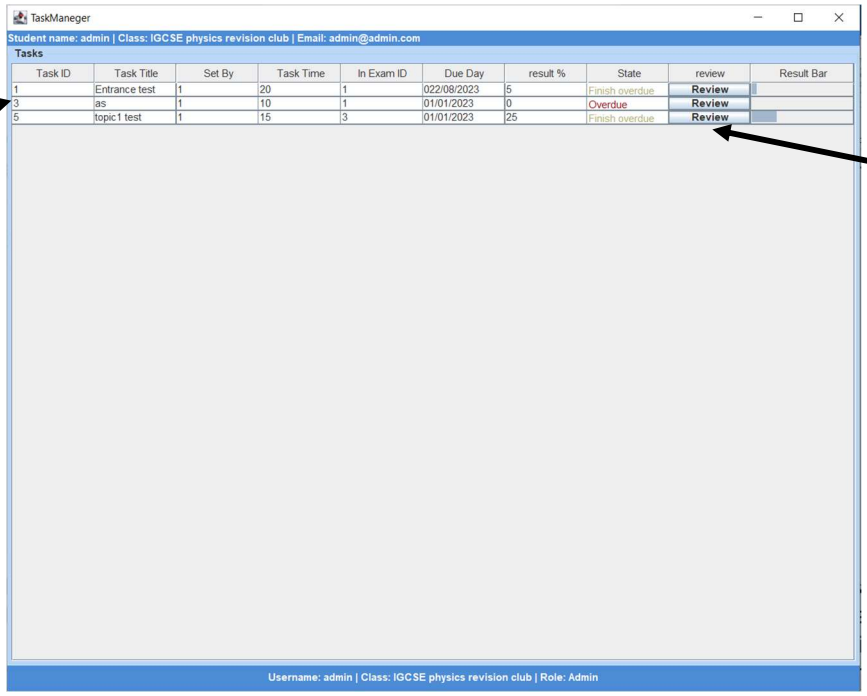
Lambda expression....22

Graphical User Interface (GUI)

Users get access to their work, grades, and participation metrics and may engage with this GUI. data are dynamically imported and shown via the GUI. This also supports the adjustment of the accessible options by user roles such as students, instructors, administrators, etc. The program provides a distinct separation between user interface components and fundamental functionality by using the Model-View-Controller (MVC) design pattern.



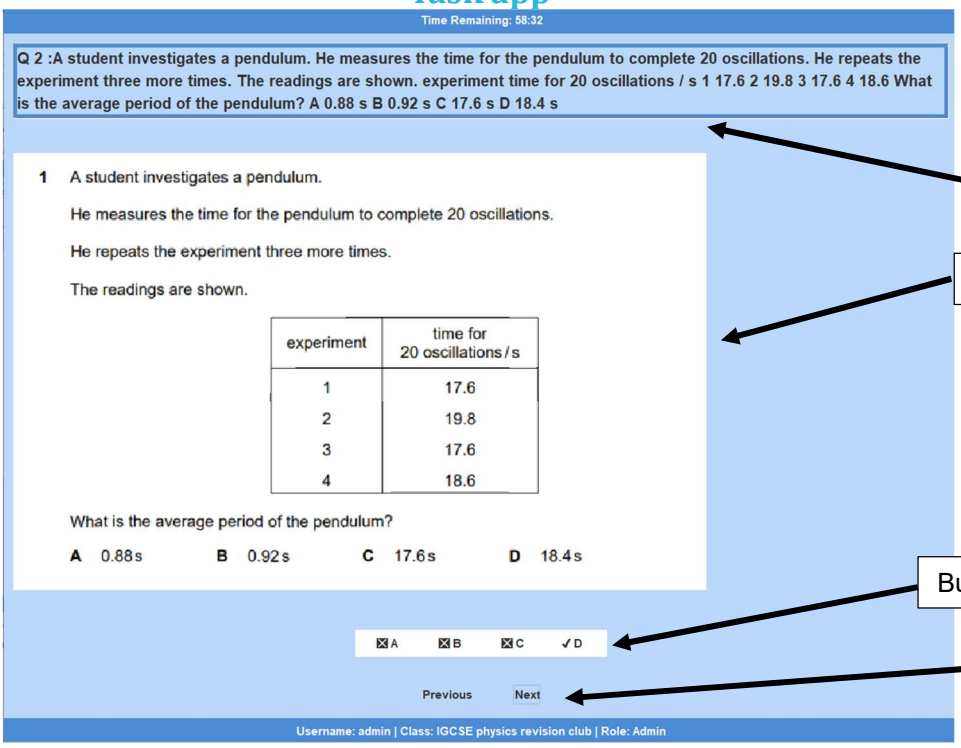
Task manager



filled with task information imported from the Online MySQL database.

Button to review all questions.

Task app



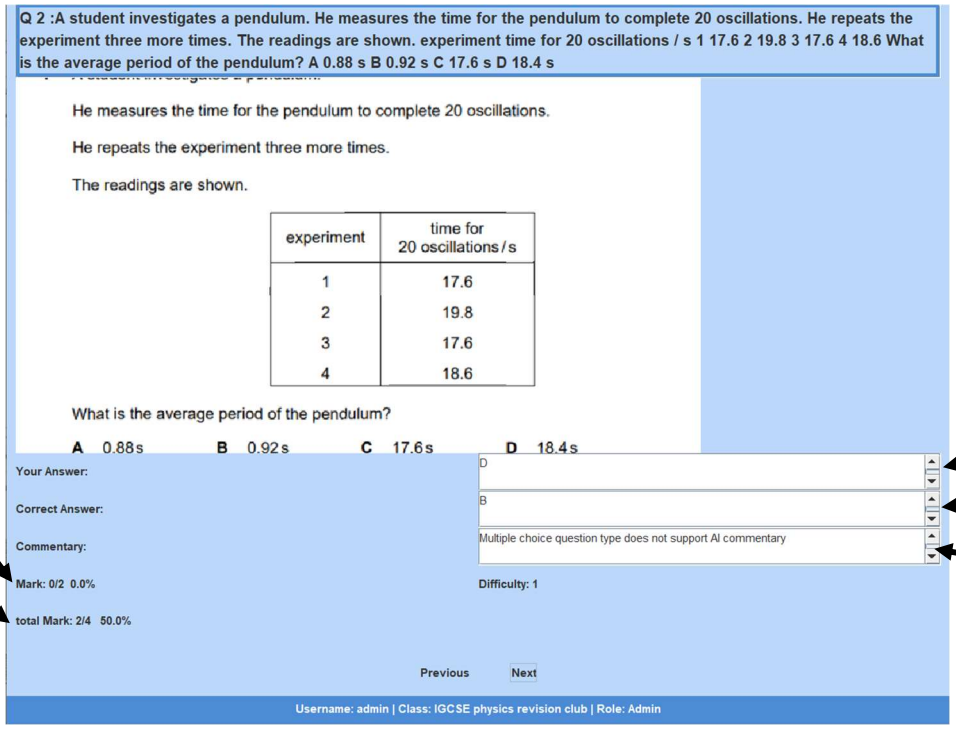
Show question (text).

Show question (image).

Button to choose answer.

Button to change multiple question.

Revision Page (student), Review Page (Class Owner)



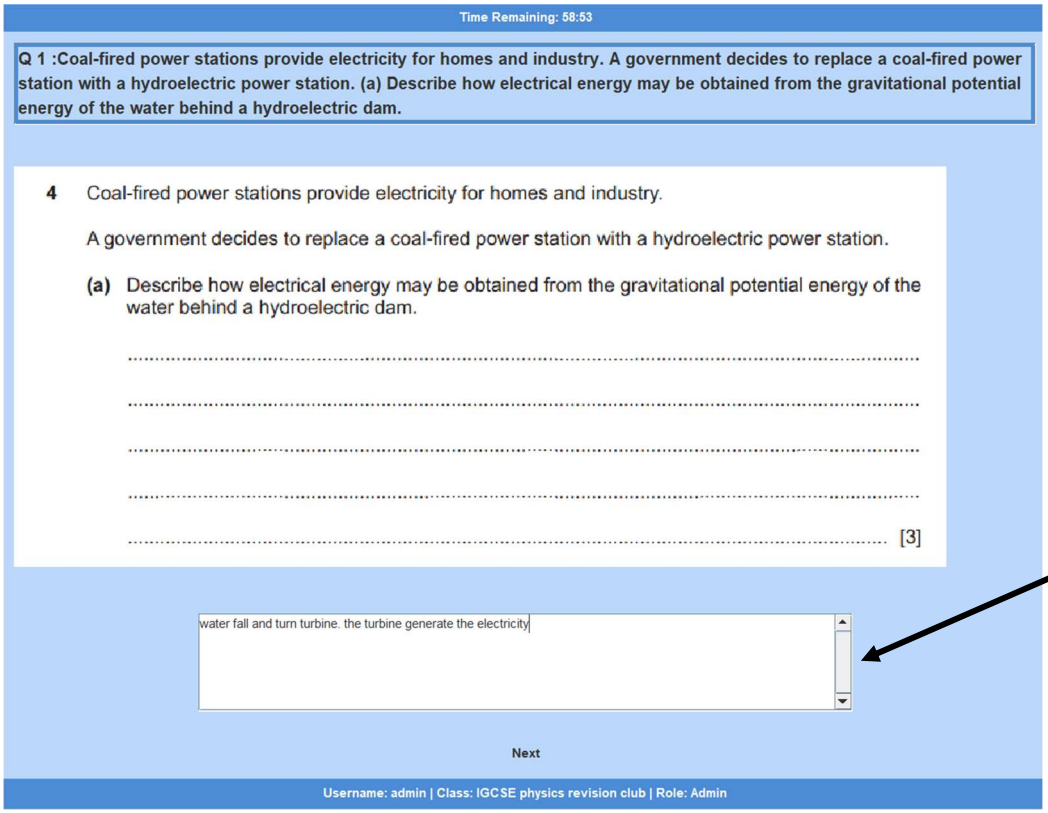
Question marks.

Total task mark

User's answer.

correct answer.

Chat GPT's justification.



Text field to answer writing question.

External Libraries

Jcalander.jar provides the capability to conveniently process date selection and display. json.jar is used to handle JSON data, mysql-connector.jar supports efficient communication between Java applications and MySQL databases, and junit-platform-console-standalone.jar is used to verify the accuracy and quality of code.

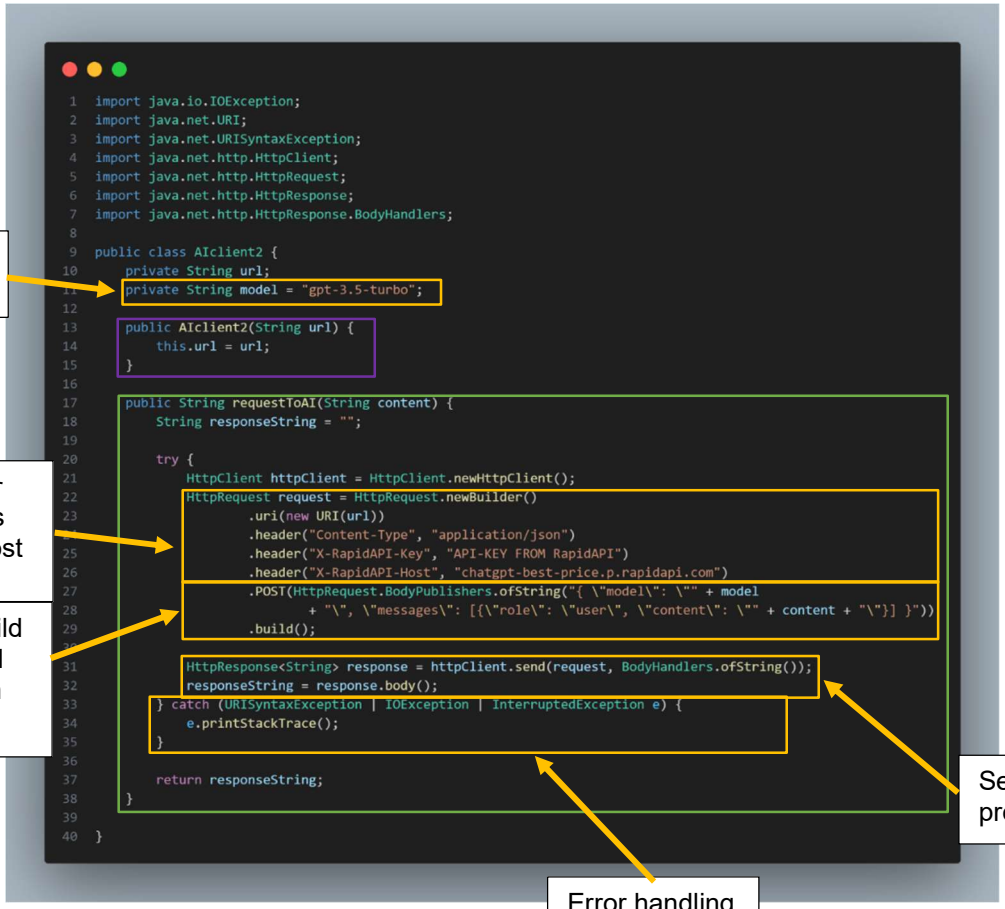

```
1  import org.json.JSONObject;
```

```
1  import com.toedter.calendar.JCalendar;
2  import com.toedter.calendar.JDateChooser;
```

```
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.PreparedStatement;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6  import java.util.ArrayList;
7  import java.util.List;
```

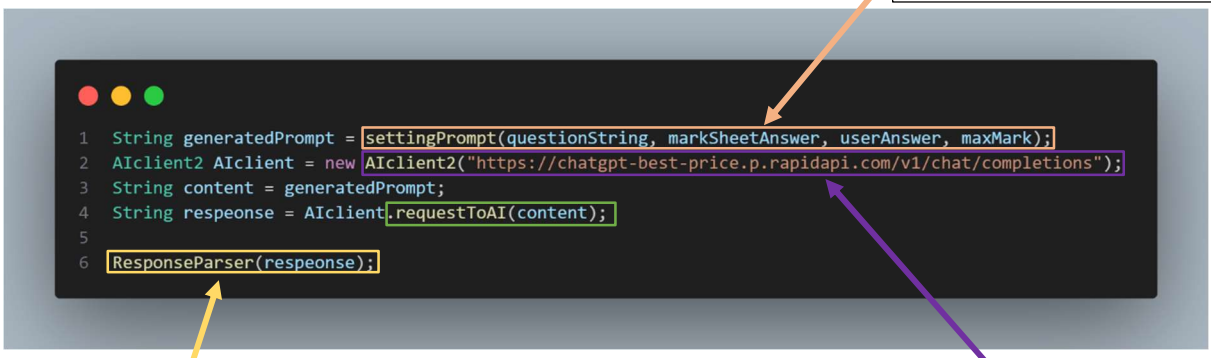
Web API and auto grading

this HTTP client class('AIclient2') allows us to mark the essay question with ChatGPT by using Web API that is "ChatGPT Best Price" provided by "rapid Api". This returns a response in a JSON format. This plays an important role in reducing the workload of running an IGCSE physics club, which the client already has a heavy workload with his job as a teacher.




```java
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpResponse.BodyHandlers;

public class AIclient2 {
    private String url;
    private String model = "gpt-3.5-turbo";

    public AIclient2(String url) {
        this.url = url;
    }

    public String requestToAI(String content) {
        String responseString = "";

        try {
            HttpClient httpClient = HttpClient.newHttpClient();
            HttpRequest request = HttpRequest.newBuilder()
                    .uri(new URI(url))
                    .header("Content-Type", "application/json")
                    .header("X-RapidAPI-Key", "API-KEY FROM RapidAPI")
                    .header("X-RapidAPI-Host", "chatgpt-best-price.p.rapidapi.com")
                    .POST(HttpRequest.BodyPublishers.ofString("{ \"model\": \"" + model
                            + "\", \"messages\": [{\"role\": \"user\", \"content\": \"" + content + "\"}] }"))
                    .build();

            HttpResponse<String> response = httpClient.send(request, BodyHandlers.ofString());
            responseString = response.body();
        } catch (URISyntaxException | IOException | InterruptedException e) {
            e.printStackTrace();
        }

        return responseString;
    }
}
```

Use the 'AIclient2' class:




```java
String generatedPrompt = settingPrompt(questionString, markSheetAnswer, userAnswer, maxMark);
AIclient2 AIclient = new AIclient2("https://chatgpt-best-price.p.rapidapi.com/v1/chat/completions");
String content = generatedPrompt;
String respeonse = AIclient.requestToAI(content);

ResponseParser(respeonse);
```

A prompt is created using the 'settingPrompt()' method.

```
public static String settingPrompt(String question, String markSheet, String userAnswer, int maxMark) {
    prompt = String.format(
            "you are grader, the IGCSE student answered determin awardmark and justfication     " +
                    "Question is " + question + "    " +
                    "marksheet is " + markSheet + "     " +
                    "useranswer is " + userAnswer + "    " +
                    "maxMark is " + maxMark + "     " + "    " +
                    "Please answer in the following format with the above information.      " +
                    "     " +
                    "Award Mark:            " +
                    "Justification:"

    );
    return prompt;
}
```

Use ResponseParser() method to parse the JSON object.

```
public static void ResponseParser(String jsonResponse) {
    JSONObject jsonObject = new JSONObject(jsonResponse);
    JSONObject messageObject = jsonObject.getJSONArray("choices")
            .getJSONObject(0)
            .getJSONObject("message");

    String content = messageObject.getString("content");
    int startIndex = content.indexOf("Award Mark: ") + 12;
    int endIndex = content.indexOf("\n", startIndex);

    awardMark = Integer.parseInt(content.substring(startIndex, endIndex).trim());

    startIndex = content.indexOf("Justification: ") + 15;
    justification = content.substring(startIndex).trim();

    System.out.println(awardMark);
    System.out.println("\n");
    System.out.println(justification);
}
```

Declare JSON object.

In the content, get useful information such as award Mark, justification.

Encapsulation

I utilized encapsulation throughout the program to ensure that some variables could not be accessed directly without the use of getters and setters. I had to utilize these methods to access these variables consistently, which simplified and improved the code. This is also a common OOP feature, which makes my software simpler to comprehend and maintain. Additionally, by afterwards placing these objects in the array list after creating them in this manner, the code became even clearer.

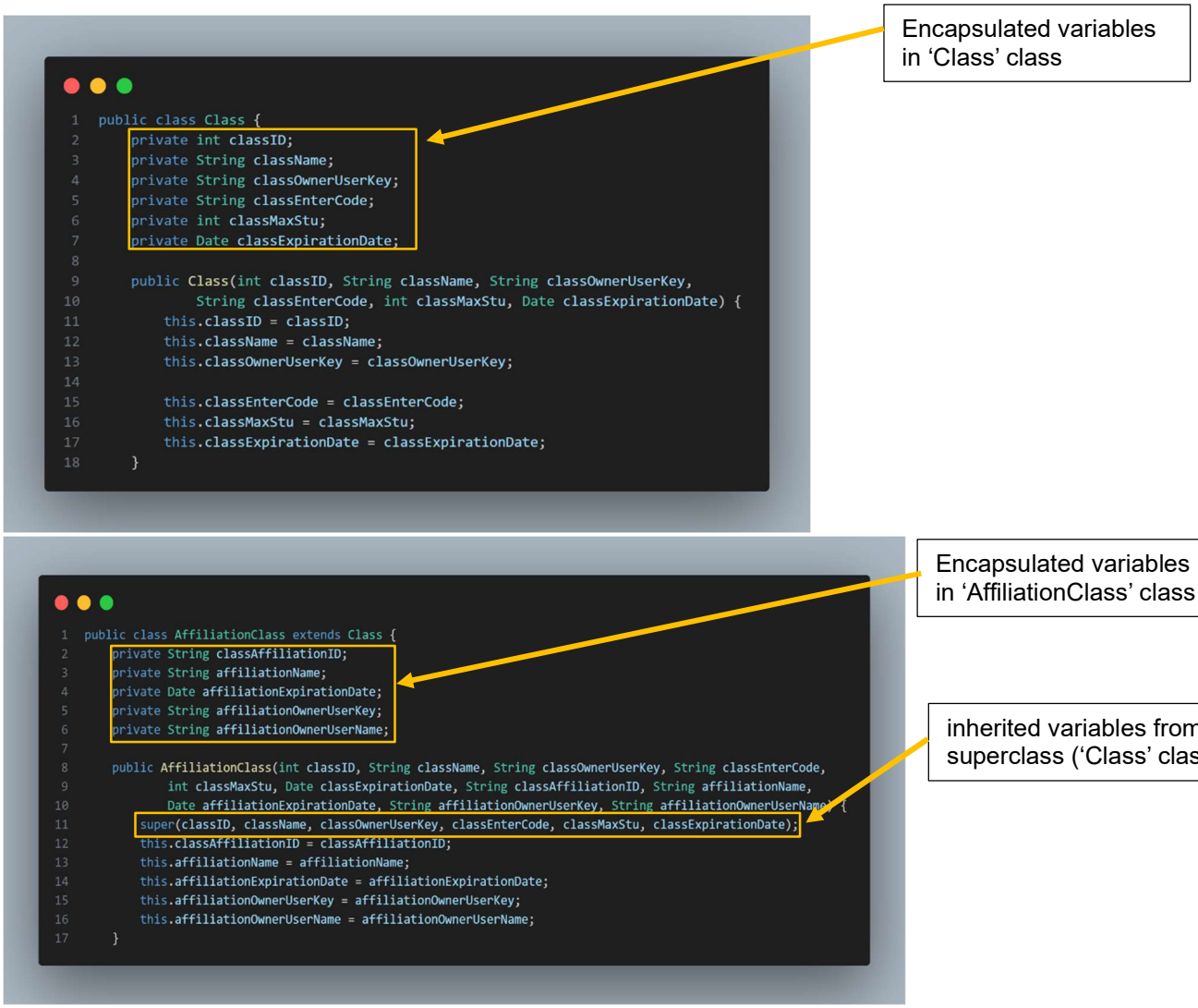

```java
public class Class {
    private int classID;
    private String className;
    private String classOwnerUserKey;
    private String classEnterCode;
    private int classMaxStu;
    private Date classExpirationDate;

    public Class(int classID, String className, String classOwnerUserKey,
            String classEnterCode, int classMaxStu, Date classExpirationDate) {
        this.classID = classID;
        this.className = className;
        this.classOwnerUserKey = classOwnerUserKey;

        this.classEnterCode = classEnterCode;
        this.classMaxStu = classMaxStu;
        this.classExpirationDate = classExpirationDate;
    }

    public int getClassID() {
        return classID;
    }

    public void setClassID(int classID) {
        this.classID = classID;
    }
```

Inheritance

I utilized inheritance to reduce time since, in the end, the makeAffiliationClassGUI basically adds different components to the makeClassGUI and adds pertinent features to the Affiliation.

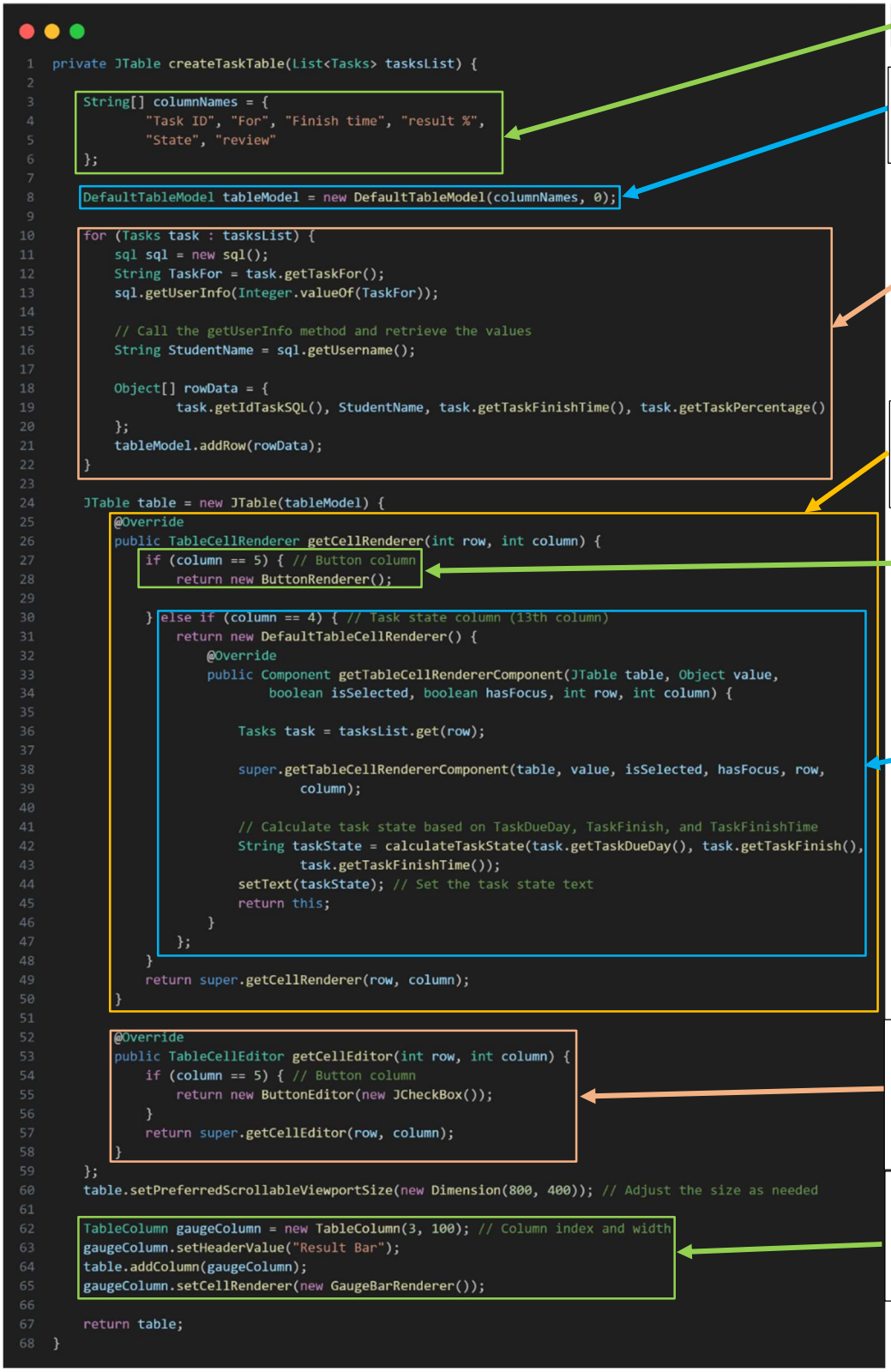
```
1  public class makeAffiliationClassGUI extends makeClassGUI
```

Inheritance was also used in the class for encapsulation. This is the case with Affiliation Class and Class. AffiliationClass's object is suitable for using inheritance because it only needs to add a few variables such as 'affiliationName' along with the variables in Class.



Complex Methods to customise JTable

The code extends the JTable class to create unique cell renderers and editors. Where the JTable is inherited, and the method is overridden to vary the behavior for each column. It allows the modification of the JTable as purpose. This is required to increase user usability by changing JTable for build Task manager (see Graphical User Interface (GUI) - Task manager in page 2).




```
private JTable createTaskTable(List<Tasks> tasksList) {

    String[] columnNames = {
            "Task ID", "For", "Finish time", "result %",
            "State", "review"
    };

    DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0);

    for (Tasks task : tasksList) {
        sql sql = new sql();
        String TaskFor = task.getTaskFor();
        sql.getUserInfo(Integer.valueOf(TaskFor));

        // Call the getUserInfo method and retrieve the values
        String StudentName = sql.getUsername();

        Object[] rowData = {
                task.getIdTaskSQL(), StudentName, task.getTaskFinishTime(), task.getTaskPercentage()
        };
        tableModel.addRow(rowData);
    }

    JTable table = new JTable(tableModel) {
        @Override
        public TableCellRenderer getCellRenderer(int row, int column) {
            if (column == 5) { // Button column
                return new ButtonRenderer();

            } else if (column == 4) { // Task state column (13th column)
                return new DefaultTableCellRenderer() {
                    @Override
                    public Component getTableCellRendererComponent(JTable table, Object value,
                            boolean isSelected, boolean hasFocus, int row, int column) {

                        Tasks task = tasksList.get(row);

                        super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row,
                                column);

                        // Calculate task state based on TaskDueDay, TaskFinish, and TaskFinishTime
                        String taskState = calculateTaskState(task.getTaskDueDay(), task.getTaskFinish(),
                                task.getTaskFinishTime());
                        setText(taskState); // Set the task state text
                        return this;
                    }
                };
            }
            return super.getCellRenderer(row, column);
        }

        @Override
        public TableCellEditor getCellEditor(int row, int column) {
            if (column == 5) { // Button column
                return new ButtonEditor(new JCheckBox());
            }
            return super.getCellEditor(row, column);
        }
    };
    table.setPreferredScrollableViewportSize(new Dimension(800, 400)); // Adjust the size as needed

    TableColumn gaugeColumn = new TableColumn(3, 100); // Column index and width
    gaugeColumn.setHeaderValue("Result Bar");
    table.addColumn(gaugeColumn);
    gaugeColumn.setCellRenderer(new GaugeBarRenderer());

    return table;
}
```

```java
class GaugeBarRenderer extends JProgressBar implements TableCellRenderer {
    public GaugeBarRenderer() {
        setBorderPainted(false);
        setStringPainted(true);
        setString(""); // Empty string to show only the gauge bar
    }

    @Override
    public Component getTableCellRendererComponent(JTable table, Object value,
            boolean isSelected, boolean hasFocus, int row, int column) {
        int percentage = Integer.parseInt(value.toString());
        setValue(percentage);
        return this;
    }
}

class ButtonRenderer extends JButton implements TableCellRenderer {
    public ButtonRenderer() {
        setOpaque(true);
    }

    @Override
    public Component getTableCellRendererComponent(JTable table, Object value,
            boolean isSelected, boolean hasFocus, int row, int column) {
        setText("Review"); // Set button text
        return this;
    }
}

class ButtonEditor extends DefaultCellEditor {
    protected JButton button;
    private int currentRow;

    public ButtonEditor(JCheckBox checkBox) {
        super(checkBox);
        button = new JButton();
        button.setOpaque(true);
        setClickCountToStart(1);
        button.addActionListener(e -> {
            // Open a new window based on the button click
            Tasks task = tasksList.get(currentRow);

            if (task.getTaskFinish().equals("1")) {
                // Task is finished, open revision page
                revisionPage revisionPage = new revisionPage(userKey, task);
                revisionPage.setVisible(true);
            } else if (task.getTaskFinish().equals("0")) {
                // Task is not finished, show a message
                JOptionPane.showMessageDialog(null, "Task is not finished.");
            }
        });
    }

    @Override
    public Component getTableCellEditorComponent(JTable table, Object value,
            boolean isSelected, int row, int column) {
        currentRow = row; // Store the current row for reference
        button.setText("Review"); // Set button text
        return button;
    }
}
```

Initialize gaugeBarRenderer with not showing border and showing empty string.

Overriding the method to customize the rendering component for the result bar column and set percentage from the parameter object.

Initialize ButtonRenderer with making the button opaque.

Overriding the method to customize the rendering component for the review button column and set the text of the button to "Review".

Initializes the button editor with specific settings and action listener to handle button clicks. When the button is clicks, open the revisionPage(review page) [see GUI – revision page (student) review page (class owner]

Overriding the method to customize the editor component for the review button column

Security (Hashing password)

By using SHA-256 algorithm, the password encrypted when the password upload to database. The login system hashes user input with SHA-265 algorithm and compares with data from database. It improves the level of security. Even when the database gets hacked the password will not be able to decrypt.

```
Connection con = DriverManager.getConnection("jdbc:mysql://13.50.31.134/IB_CS_IA",
                    "kado",
                    "---------");
            String query = "SELECT * FROM user_info WHERE Email = ? AND pass = ?";
            PreparedStatement statement = con.prepareStatement(query);
            statement.setString(1, userValue);
            passenc = SHA256.encrSHA(passValue);
            statement.setString(2, passenc);
            System.err.println(passenc);
            ResultSet resultSet = statement.executeQuery();
```

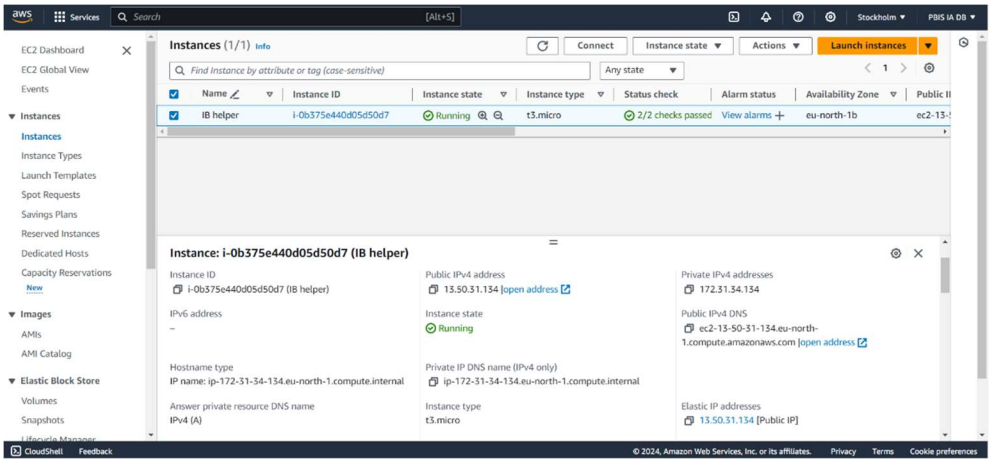
Data encrypting.

```
public static String encrSHA(String input) {
    try {
        // Create MessageDigest instance for SHA-256
        MessageDigest sha256 = MessageDigest.getInstance("SHA-256");

        // Apply SHA-256 encryption to the input
        byte[] hashBytes = sha256.digest(input.getBytes());

        // Convert byte array to a string of hexadecimal values
        StringBuilder hexString = new StringBuilder();
        for (byte hashByte : hashBytes) {
            String hex = Integer.toHexString(0xff & hashByte);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();

    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;
    }
}
```

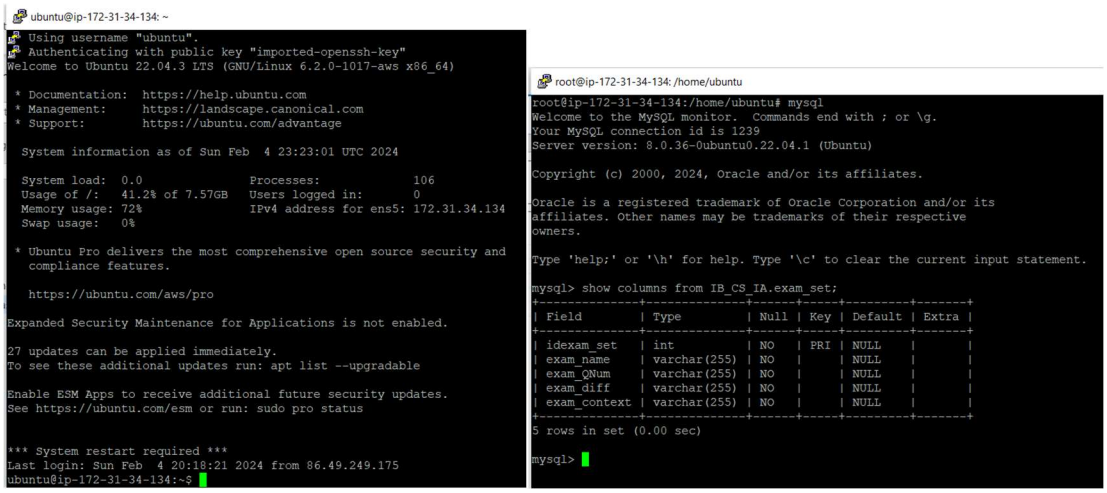
Web database server

This program requires students to check the task set by the teacher on the teacher's computer in real-time and take the test. All the data in the program must interact with the online database organically and in real-time. Therefore, building a web online database server is desirable. This means that students and teachers don't have to work on the same computer, while also giving students the ability to go home and do their homework.

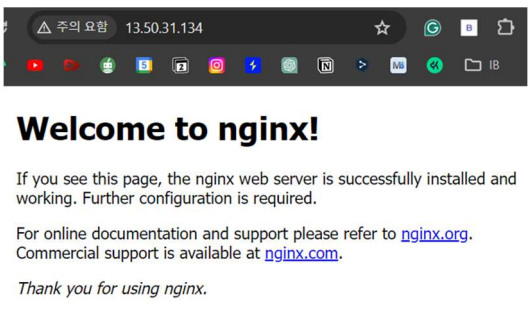
AWS EC2 service that lends server computer has ubuntu OS:



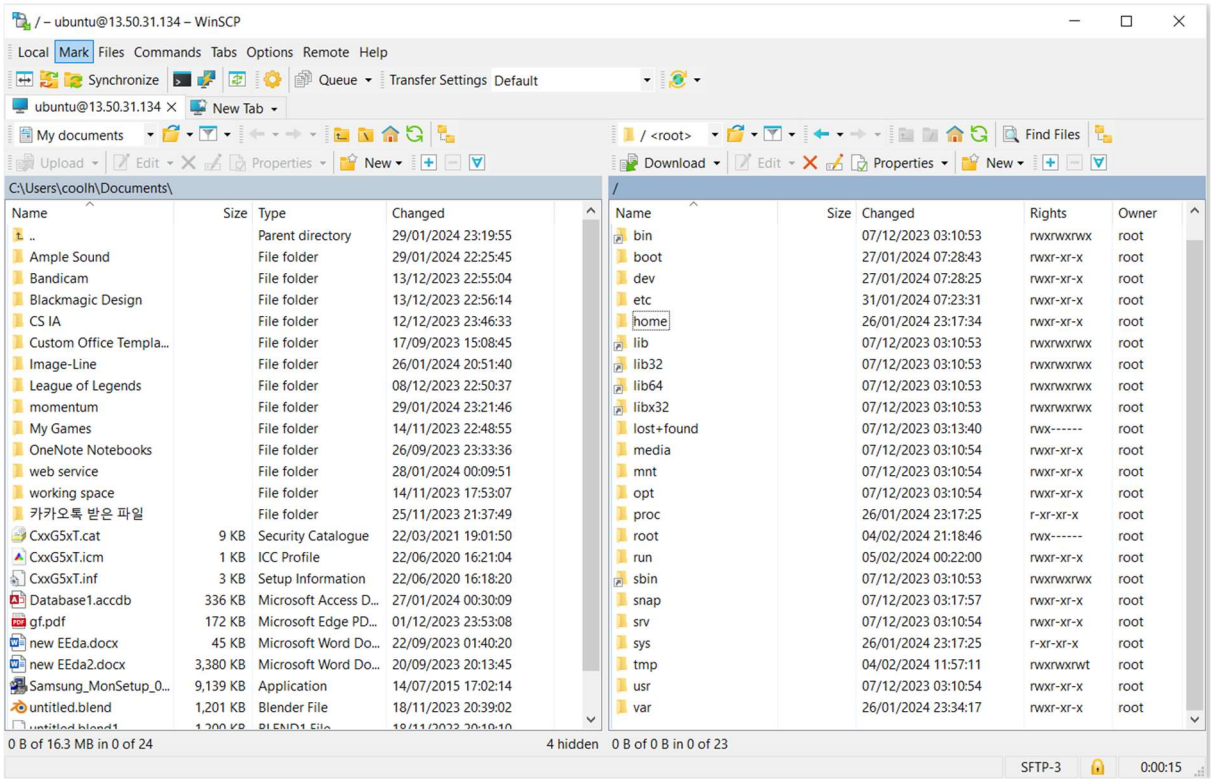
SSL connection to my web database server that has Nginx, Node.js, MySQL:



Connect to webserver with chrome:



File view of the webserver:



Implication:

```java
public void getUserInfo(int userKey) {
        try {
            Connection con = DriverManager.getConnection("jdbc:mysql://13.50.31.134/IB_CS_IA",
            "kado","-------");
            String query = "SELECT * FROM user_info WHERE userKey = ?";
            PreparedStatement statement = con.prepareStatement(query);
            statement.setInt(1, userKey);
            ResultSet resultSet = statement.executeQuery();

            if (resultSet.next()) {
                // Retrieve values from the result set
                username = resultSet.getString("name");
                email = resultSet.getString("Email");
                Class = resultSet.getString("class");
                userRole = resultSet.getString("userRole");
                classId = resultSet.getInt("classID");

                userRoleL = getUserRoleString(userRole);

            }

            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

2D Array

The use of a 2D Array made the program clearer. Using a 2d array was a logical choice of data structure as it allowed the program to not only store the exam and questions. This enabled the program to have an appropriate structure to store the questions as they read from the database. This was the efficient way to handle and transfer data from the database to the KNN algorithm.

```
examSetData = new Object[examSetList.length][10];
try {
    Connection con = DriverManager.getConnection("jdbc:mysql://13.50.31.134/IB_CS_IA",
    "kado","-------");
    String query = "SELECT * FROM q_set WHERE Q_examSet_id = ?";
    int flag_examSet = 0;
    for (ExamSet examSet : examSetList) {

        PreparedStatement statement = con.prepareStatement(query);
        int examId = examSet.getIdexam_set();
        statement.setInt(1, examId);
        ResultSet resultSet = statement.executeQuery();
        int exam_QNum = Integer.valueOf(examSet.getExam_QNum());
        Object[] setInfo = new Object[exam_QNum + 1];
        setInfo[0] = examSet;
        int flag_setInfo = 1;

        while (resultSet.next()) {

            // Your existing code for fetching question details
            int idQ_sets = resultSet.getInt("idQ_set");
            String Q_questions = resultSet.getString("Q_question");
            int Q_examSet_ids = resultSet.getInt("Q_examSet_id");
            String Q_diffs = resultSet.getString("Q_diff");
            byte[] Q_images = resultSet.getBytes("Q_image");
            String Q_answers = resultSet.getString("Q_answer");

            String Q_type = resultSet.getString("Q_type");
            String Q_points = resultSet.getString("Q_points");
            String Q_topic = resultSet.getString("Q_topic");

            Question question = new Question(idQ_sets, Q_questions, Q_examSet_ids,
                                        Q_diffs, Q_images, Q_type,
                                        Q_answers,
                                        Q_points, Q_topic);

            // Now set the value of x in the setInfo array

            setInfo[flag_setInfo] = question; // -1 because arrays are 0-based
            flag_setInfo++;



        }
        examSetData[flag_examSet] = setInfo;
        flag_examSet++;
    }

}
```

initializes a 2D array named examSetData with the first dimension's length is the length of the examSetList array and the second dimension's length is 10.

Iterates over the examSetList and the examSet Object is used to next lines.

Get the exam ID from exam set object in exam set list and set query with this exam ID.

iterates over the result from the database. The detailed information of each question is used to construct a Question object.

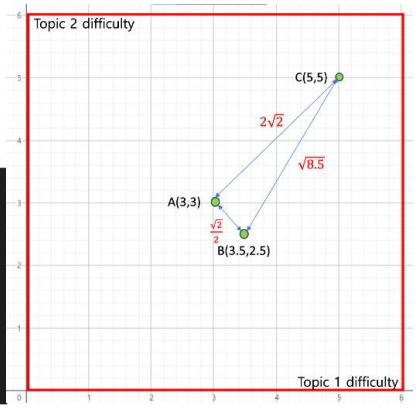
assigns the Question object to the setInfo array at the position indicated by flag_setInfo, then increments flag_setInfo to prepare for the next iteration.

assigns the setInfo array to the examSetData array at the position indicated by flag_eamSet, then increments flag_eamSet to prepare for the next iteration.

KNN algorithm for recommend system

K-Nearest Neighbors Algorithm *is a type of supervised learning algorithm used for both regression and classification [6]*. It calculates similarity between user's topical predicted grades and topical difficulties exam set. It recommends the exam set with the highest similarity.

```
ExamSetInfo: [20.0, 20.0, 0.0, 0.0, 0.0, 0.0]
topicListInfo: [50.0, 10.0, 60.0, 20.0, 30.0, 70.0]
103.92304845413264
ExamSetInfo: [46.666668, 46.666668, 46.666668, 46.666668, 46.666668, 46.666668]
topicListInfo: [50.0, 10.0, 60.0, 20.0, 30.0, 70.0]
55.37749333792508
```



The similarity corresponds to the distance between two data sets which can be calculated follow.

For points with data coordinates (p_1, p_2, p_3, ..., p_n) and (q_1, q_2, q_3, ..., q_n), the distance is calculated as:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

It is used because a mathematically simple model and relatively easy to code compared to other machine learning algorithms, does not require training steps, and is suitable for both classification and regression tasks. For input, a 2D array is used. This is used to recommend exam sets that are appropriate for each student's level, helping them to improve their performance.

```
1  private List<Integer> recommendExamSet(int k) {
2          Map<Integer, Double> distances = new HashMap<>();
3
4          // Iterate through each setInfo in examSetData and calculate distances
5          for (Object[] setInfo : examSetData) {
6              ExamSet examSet = (ExamSet) setInfo[0];
7              int examSetID = examSet.getIdexam_set();
8
9              double distance = calculateDistance(topicList, setInfo);
10             distances.put(examSetID, distance);
11         }
12
13         // Sort distances and find k-nearest neighbors
14         List<Integer> recommendedExamSetIDs = new ArrayList<>();
15         distances.entrySet().stream()
16                 .sorted(Map.Entry.comparingByValue())
17                 .limit(k)
18                 .forEach(entry -> recommendedExamSetIDs.add(entry.getKey()));
19         System.out.println(recommendedExamSetIDs);
20
21         return recommendedExamSetIDs;
22     }
```

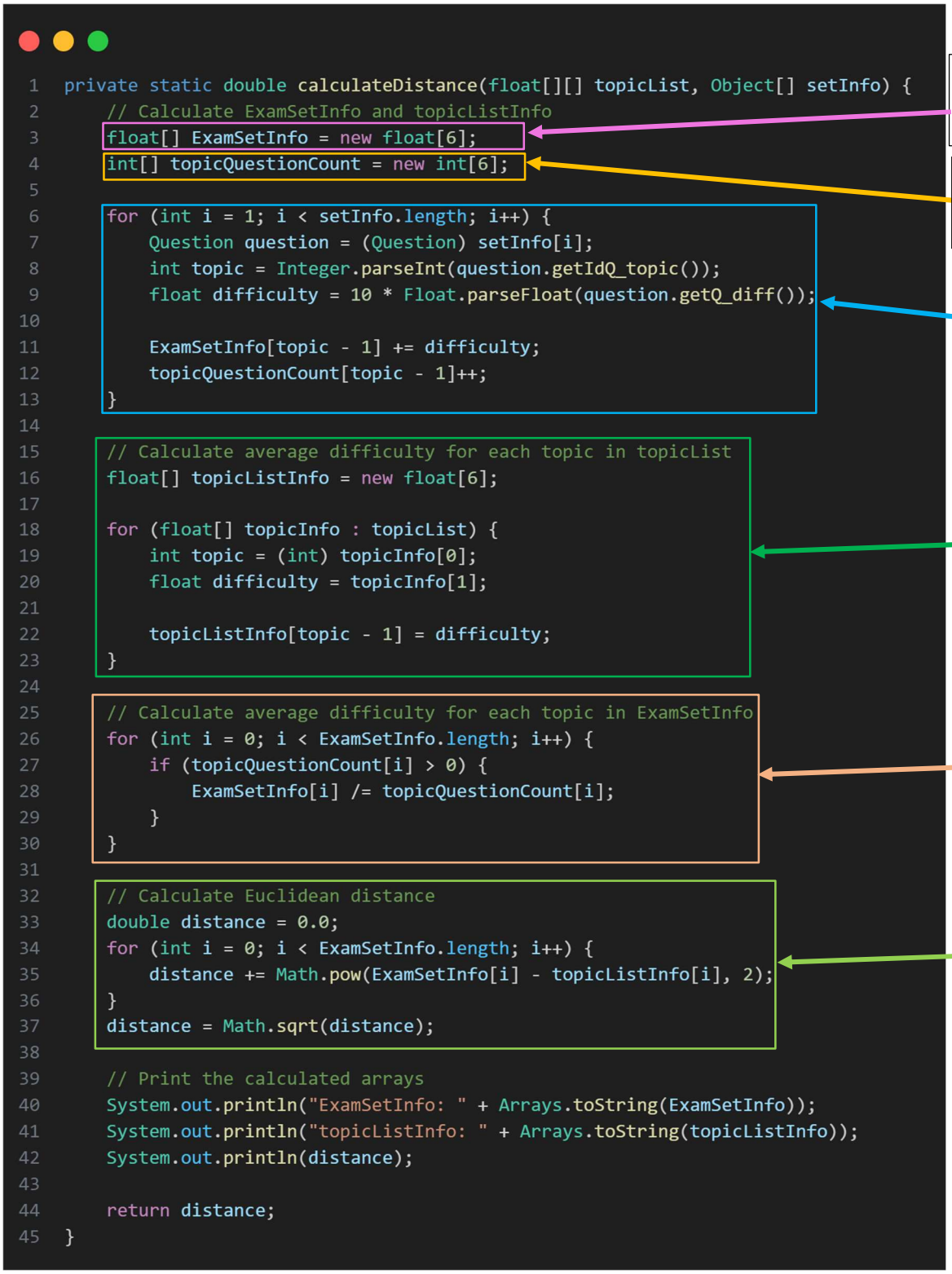
Return list containing k number of the exam set IDs.

Declare a hash table.

Use setInfo array in examSetData 2D array to compare itself and topicList and calculate distance between two data sets and add distance to hash table using examsetID as key.

User's predicted grade list

Sort hash table by distance, limit k number of nearest neighbors, add their exam ID to recommendedExamSetIDs list.



Declare list to store aggregated difficulty value.

Declare list to track of the number of questions for each topic.

iterates through each question in the exam set and aggregate difficulty and total question number.

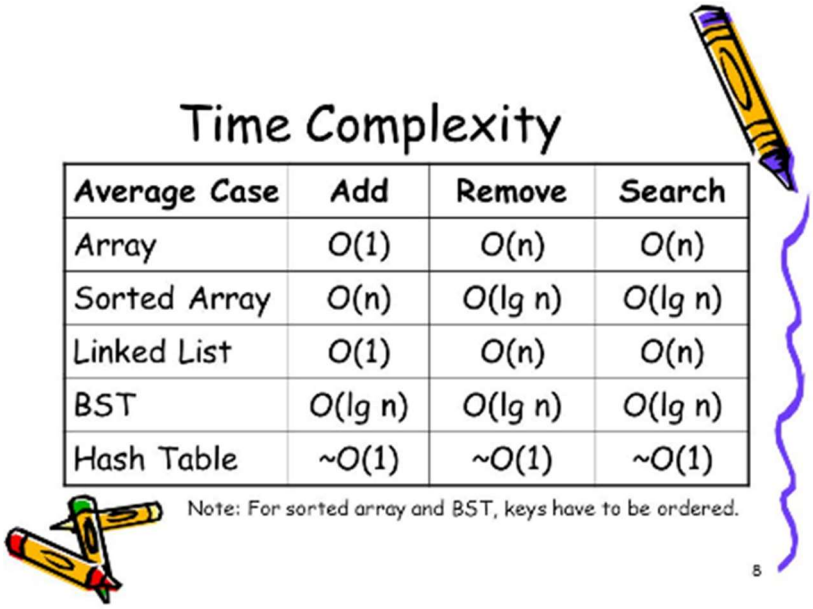
iterates through topic list and get info from this array such as each topic's number and difficulty and store those in topicListInfo list.

Get average difficulty of each topic by dividing aggregated difficulty by number of questions.

By using the distance formula, calculate distance between two data sets.

Hash table

The hash table stores ExamSet ID and the distances between the exam set topical difficulties and user's predicted grades for the KNN algorithm. The hash function has an average time complexity of O(1) in search and insertion-deletion for a large number of data. It takes advantage of when the program has massive amount of examset data.



Time Complexity

Average Case	Add	Remove	Search
Array	O(1)	O(n)	O(n)
Sorted Array	O(n)	O(lg n)	O(lg n)
Linked List	O(1)	O(n)	O(n)
BST	O(lg n)	O(lg n)	O(lg n)
Hash Table	~O(1)	~O(1)	~O(1)

Note: For sorted array and BST, keys have to be ordered.

8

[5]

```
1  Map<Integer, Double> distances = new HashMap<>();
```

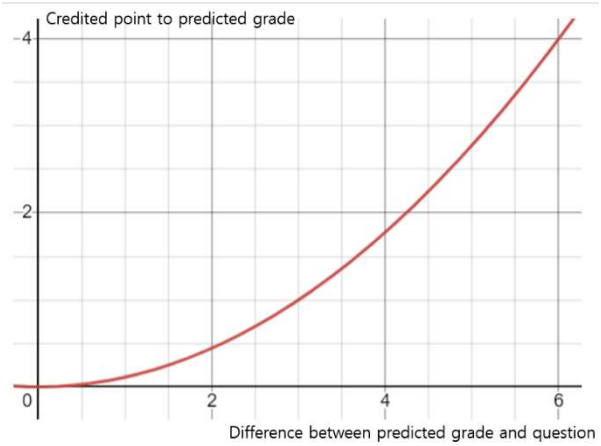
```
1  distances.entrySet().stream()
2                  .sorted(Map.Entry.comparingByValue())
3                  .limit(k)
4                  .forEach(entry -> recommendedExamSetIDs.add(entry.getKey()))
```

Recursion

The square function was needed to create a system that would give students more credit if they did more difficult questions than their predicted score, and less credit if they did less difficult questions, and factor that into their predicted score. This code allows the users to see their predicted grade rise should they do well with the more difficult questions. This also serves to motivate the user to keep trying harder questions.

```java
public float PowerFunction(float base, int power) {

    if (power == 1) {
        return 1;
    } else if (power > 0) {
        return base * PowerFunction(base, power - 1);
    } else if (power < 0) {
        return 1 / PowerFunction(base, -power);
    } else {
        return 0.0f;
    }
}
```

```java
float base = (diffgrade / 3);
adjustPlusGrade = PowerFunction(base, 2);
```



Error handling

To handle failures (such as SQL errors and input errors) that occur in the program, I employed try-catch methods throughout the program. This helps avoid a disastrous situation in which the application must terminate owing to a mistake or the data in the database is destroyed.

```java
    public String requestToAI(String content) {
        String responseString = "";

        try {
            HttpClient httpClient = HttpClient.newHttpClient();
            HttpRequest request = HttpRequest.newBuilder()
                    .uri(new URI(url))
                    .header("Content-Type", "application/json")
                    .header("X-RapidAPI-Key", "API-KEY FROM RapidAPI")
                    .header("X-RapidAPI-Host", "chatgpt-best-price.p.rapidapi.com")
                    .POST(HttpRequest.BodyPublishers.ofString("{ \"model\": \"" + model
                            + "\", \"messages\": [{\"role\": \"user\", \"content\": \"" + content + "\"}] }"))
                    .build();

            HttpResponse<String> response = httpClient.send(request, BodyHandlers.ofString());
            responseString = response.body();
        } catch (URISyntaxException | IOException | InterruptedException e) {
            e.printStackTrace();
        }

        return responseString;
    }

}
```

To make sure that the incoming e-mail follows the normal format, the program validates it with the following code:

```java
private boolean isValidEmail(String email) {
        String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$";
        return email.matches(emailRegex);
    }
```

Lambda expression

It has streamlined the program's structure by assigning tasks to distinct functions. This approach provides a better organized and effective code structure in addition to making development easier.

```
logoutButton.addActionListener(e -> {
    try {
        CreateLoginForm form = new CreateLoginForm();
        form.setSize(900, 600);
        centreWindow(form);
        form.setVisible(true);
    } catch (Exception E) {
        JOptionPane.showMessageDialog(null, E.getMessage());
    }
    this.dispose();
});
```

```
Timer timer = new Timer(2000, e -> {
    try {
        getTaskInfo(userKey);
        showTask(tasksList);
        getGradeInfo(userKey);
        setGauge(userKey);
        getInfo();
    } catch (SQLException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
});
```

```
String[] options = {"Class", "Logout" };
comboBox = new JComboBox<>(options);
comboBox.addActionListener(e -> {
    String selectedOption = (String) comboBox.getSelectedItem();
    if (selectedOption.equals("Class")) {
        openClassWiondow(userKey);
    } else if (selectedOption.equals("Logout")) {
        try {
            CreateLoginForm form = new CreateLoginForm();
            form.setSize(900, 600);
            centreWindow(form);
            form.setVisible(true);
        } catch (Exception E) {
            JOptionPane.showMessageDialog(null, E.getMessage());
        }
        this.dispose();
    }
});
```

Bibliography

[1] Bhargava, a. y. (2017). hello coding 그림으로 개념을 이해하는 알고리즘(An Algorithm for Understanding Concepts with Hello Coding Figures). Seoul: 한빛미디어(Hanbit Media).

[2] PlanetScale Documentation. (n.d.). PlanetScale Documentation | PlanetScale Documentation. https://planetscale.com/docs

[3] Java Tutorial. (n.d.). Java Tutorial. https://www.w3schools.com/java/default.asp

[4] ChatGPT Best Price API Documentation (truongvuhung102) | RapidAPI. (n.d.). ChatGPT Best Price API Documentation (Truongvuhung102) | RapidAPI. https://rapidapi.com/truongvuhung102/api/chatgpt-best-price

[5] hash table. (n.d.). Hash Table. https://velog.io/@gringrape/hash-table

[6] Christopher, A. (2021, December 29). K-Nearest Neighbor - The Startup - Medium. Medium. https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4

[7] OpenAI. ChatGPT, Version 3.5, 9 Mar. 2024, https://chat.openai.com

Word count: 985 (exclude heading, table of context, equation, labels, screenshot, bibliography)