**ASSIGNMENT 1 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** |  | | |
| **Submission date** |  | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** |  | **Student ID** |  |
| **Class** |  | **Assessor name** |  |
| **Student declaration**  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
|  |  | **Student’s signature** |  |

**Grading grid**

|  |
| --- |
| Grade (0-10) |
|  |

|  |  |  |
| --- | --- | --- |
| **❒ Summative Feedback: ❒ Resubmission Feedback:** | | |
| **Grade:** | **Assessor Signature:** | **Date:** |
| **IV Signature:** | | |

Table of Contents

[I. Introduction 4](#_Toc106928223)

[II. Requirements 4](#_Toc106928224)

[III. UI design 5](#_Toc106928225)

[1. Login 5](#_Toc106928226)

[2. Registering 6](#_Toc106928227)

[3. Student list 7](#_Toc106928228)

[IV. Implementation 9](#_Toc106928229)

[V. Test 9](#_Toc106928230)

[VI. Result 9](#_Toc106928231)

[VII. Conclusion 9](#_Toc106928232)

# Introduction

# Requirements

# UI design

## Login

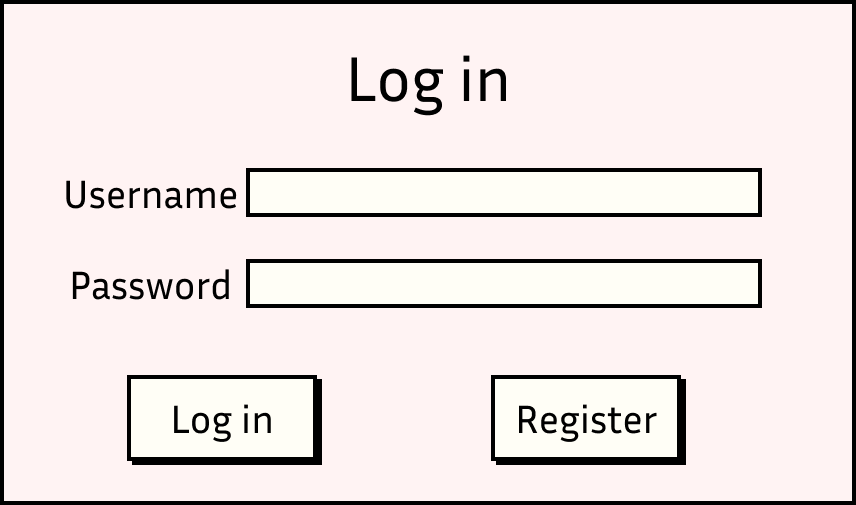


Figure 1 Login screen

## Registering

Graphical user interface, application

Description automatically generated with medium confidence

Figure 2 Register window

## Student list

Graphical user interface

Description automatically generated

Figure 3 Index window

## Add student

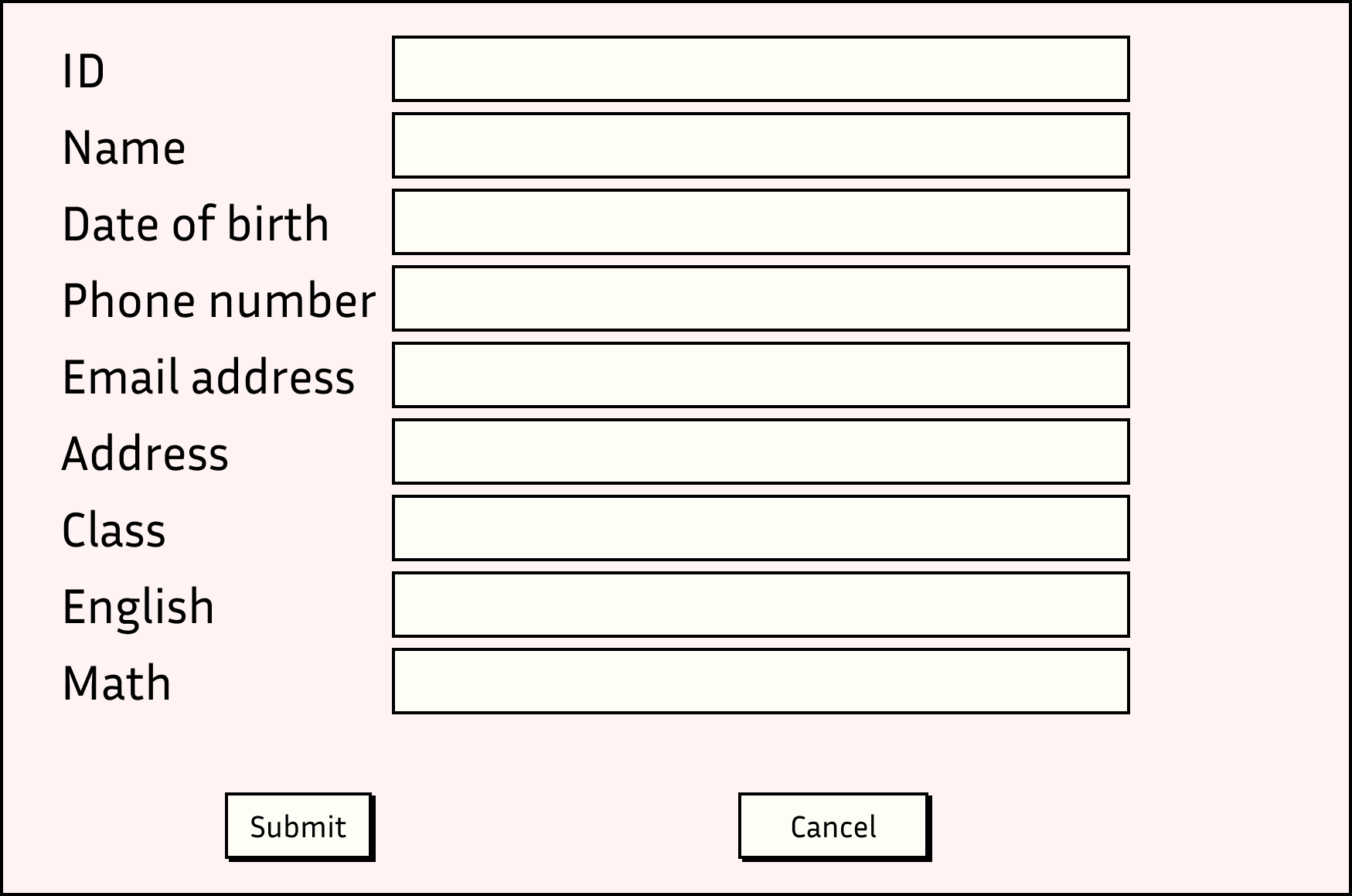


Figure 4 Window to add new student

## Edit student

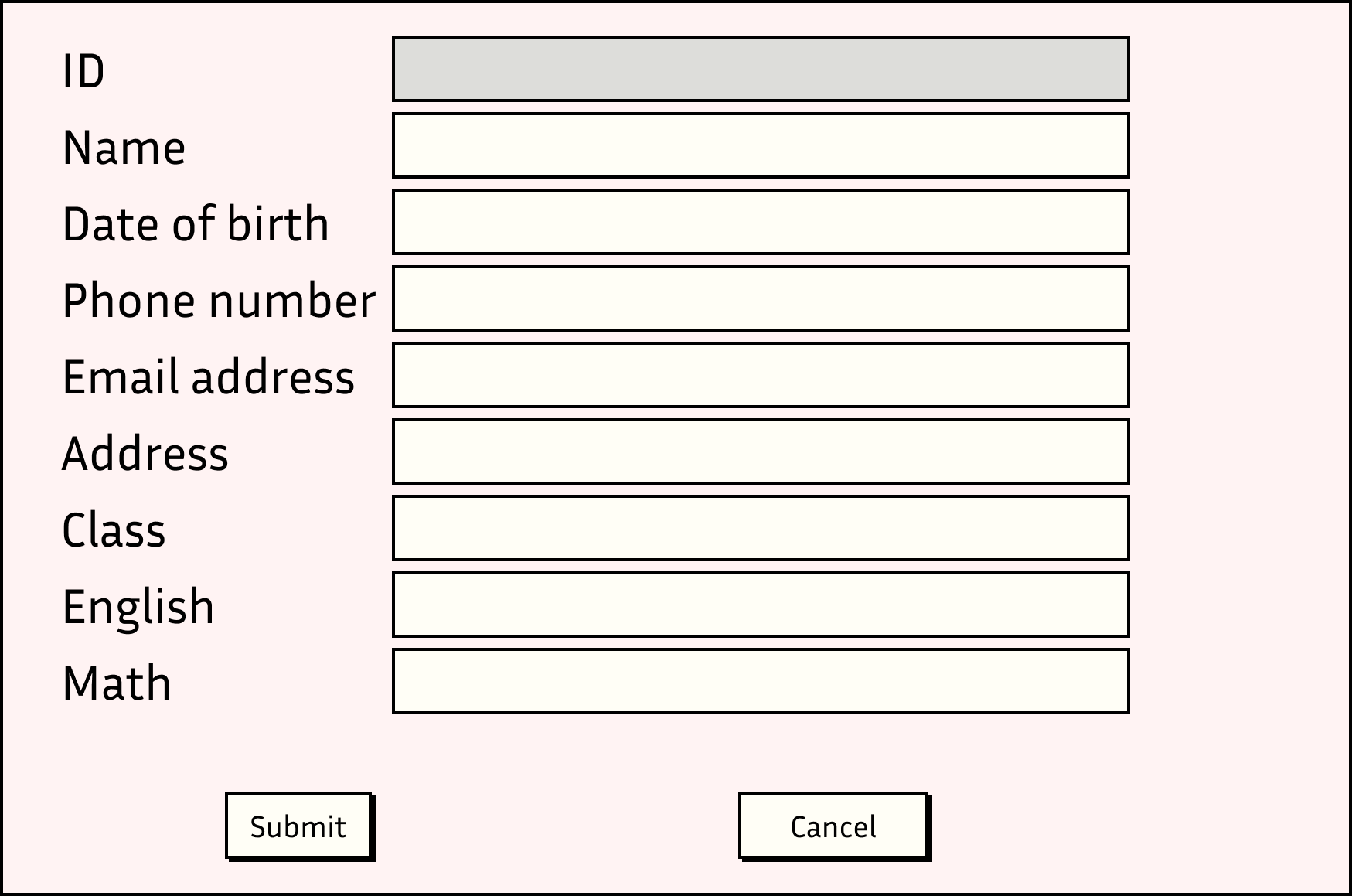


Figure 5 Edit a student's information

## View student information

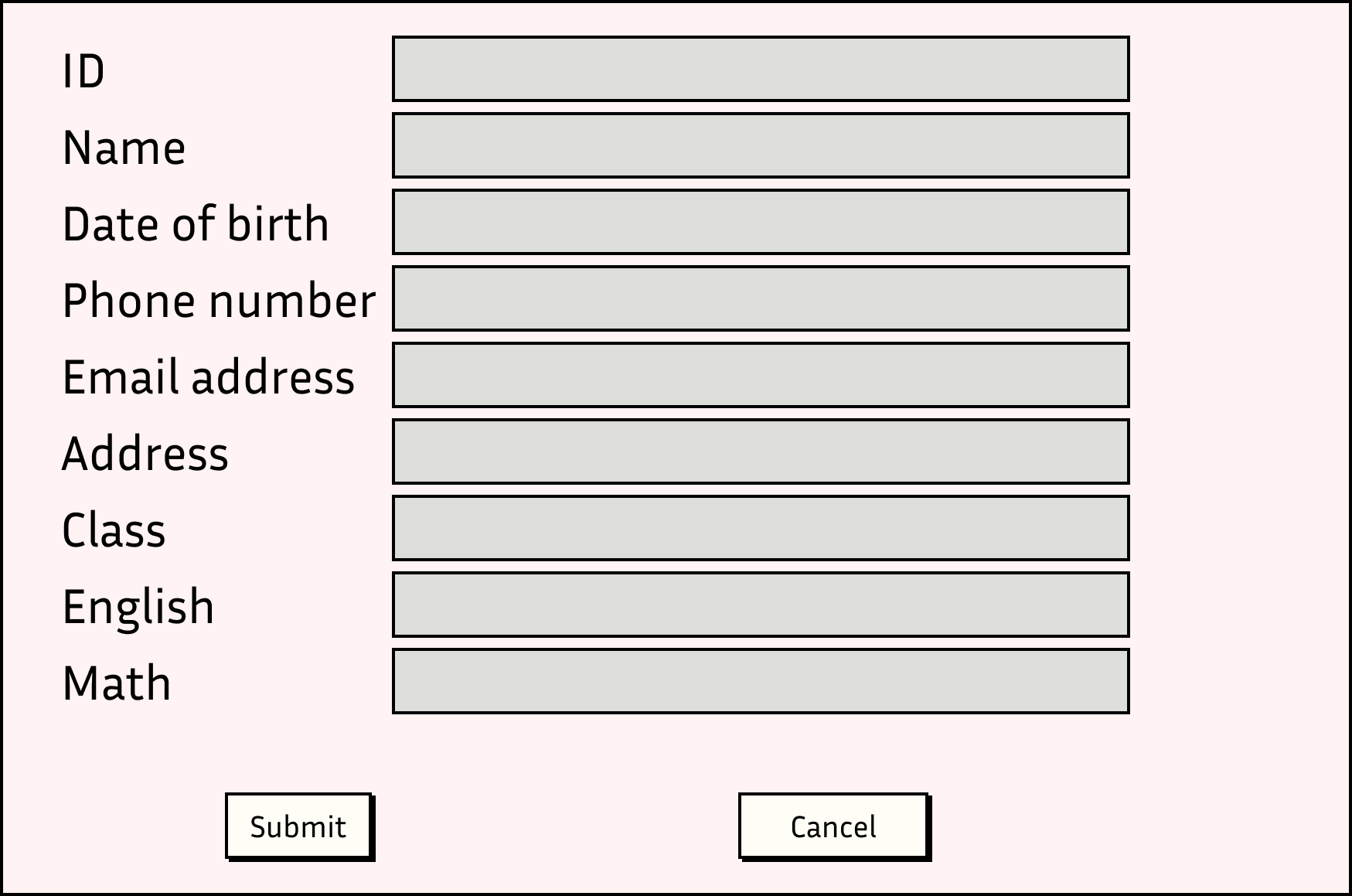


Figure 6 View a student's information

# Implementation

## Program structure

This program uses the MVC model which divides the code into three different components called Model, View and Controller. Each components handles a specific side of the program. (Tutorialspoint, 2022)

In this program, the Model component is used to store the structure of every object data type. The Controller the uses it to create instances of the objects and then store them in a file.

The View component is responsible for the application's UI logic. It presents the user with forms, fields, and table for them to interact with the program intuitively.

## Classes

### PersonModel

StudentModel holds the components of necessary to create a person object and modify it. Having a Person model is optimal because should the program expand and implement the feature to manage teachers’ information, the development would take much less time and efforts

|  |  |
| --- | --- |
| Variable | Type |
| ID | string |
| Name | string |
| Email | string |
| Phone number | string |
| Address | string |
| Date of birth | string |

Table 1 Variables of PersonModel

|  |  |
| --- | --- |
| Method | Description |
| Constructor | To create a new person with all the variables included within the model |
| Empty constructor | To create a new person with no information and then use setters to set each information |
| Getters | Gets the information of the instance and returns it |
| Setters | Sets the information of the instance |

Table 2 Methods of PersonModel

### StudentModel

StudentModel holds the components of necessary to create a student object and modify it.

|  |  |
| --- | --- |
| Variable | Type |
| mathGrade | int |
| englishGrade | int |
| gpa | double |
| classID | String |

Table 3 Variables of StudentModel

|  |  |
| --- | --- |
| Method | Description |
| Constructor | To create a new person with all the variables included within the model |
| Empty constructor | To create a new person with no information and then use setters to set each information |
| Getters | Gets the information of the instance and returns it |
| Setters | Sets the information of the instance |
| calculateGPA | This function is to calculate the GPA of the student by dividing the sum of Math and English grade of the student |
| getStudentInfo | Print the student's information out in JSON format |
| getStudentInfo2 | Print the student's information out in CSV format |

Table 4 Methods of StudentModel

### UserModel

UserModel is a model used to hold the information of every user’s login information including username and password.

|  |  |
| --- | --- |
| Variable | Type |
| username | string |
| password | string |

Table 5 Variables of UserModel

|  |  |
| --- | --- |
| Method | Description |
| Constructor | To create a new person with all the variables included within the model |
| Empty constructor | To create a new person with no information and then use setters to set each information |
| Getters | Gets the information of the instance and returns it |
| Setters | Sets the information of the instance |
| getUserInfo | Print the user information out in JSON format |
| getUserInfo2 | Print the user information out in CSV format |

Table 6 Methods of UserModel

### LoginController

This Controller is used to control events regarding logins. When a login is performed, the program reads the file containing a list of users, parses them into an ArrayList of User objects before trying to find a match with the login input by the user. After the login is performed, a message displaying its status will pop up and lead the user to the appropriate window. When the user click on the “Register” button, they are taken to the register window.

### RegisterController

This Controller is used to control events regarding registration. When a registration is performed, the program reads the file containing a list of users, parses them into an ArrayList of User objects before trying to find a match with the registration input by the user. If there is a match with the username, an error message is displayed and asks the user to enter a different username. If there is no matching username, a success message is displayed, and the user is taken to the login window.

### StudentController

This Controller is used to control events regarding student information. It is responsible for writing student information onto a file. It is also responsible for reading student information from a file and putting them into an ArrayList of Student objects. Another important part of this controller is that it handles the data validation and error handling when the user tries to submit a student’s information.

## important algorithms

### calculateGPA

When a new Student is created or edited using a constructor, the program automatically calculates the GPA of that student based on the English and Math grade of that Student input by the user Text

Description automatically generated

Figure 7 GPA is automatically calculated

GPA is calculated by averaging the sum of Math and English grade and parse it as a double data type

public double calculateGpa() {

        double gpa = (double) (getMathGrade() + getEnglishGrade()) / 2;

        return gpa;

    }

### ReadUserFromFile

This algorithm takes in a String as the name of the file and an ArrayList of User so it can write into it. Then it declares the variable line for each User and delimiter to know how each column is separated. Then it uses BufferedReader to read each line of the file, put the information into a new Student object and put that object into the ArrayList.

public static void readUserFromFile(String fileName, ArrayList<UserModel> userList) {

        String line = "";

        String cvsSplitBy = ",";

        try {

            // Read from file user.csv

            java.io.FileReader fileReader = new java.io.FileReader(fileName);

            java.io.BufferedReader bufferedReader = new java.io.BufferedReader(fileReader);

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

                // Split the line by comma

                String[] userInfo = line.split(cvsSplitBy);

                // Create a new user object

                UserModel user = new UserModel(userInfo[0], userInfo[1]);

                // Add the user object to the userList

                userList.add(user);

            }

            bufferedReader.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

### UserLogin

After the user has input the login credentials and click on the “Login” button, the program read the file for a list of Users and compare each of them with the input made by the user. If there is a match, the program displays a success message and takes the user to the index window. Otherwise, an error message notifying the user of wrong login credentials and ask them to re-enter the information.

// Read from file user.csv and create a new user object for each line

    // user.csv

    public static void loginUser(String userName, String userPassword) {

        String fileName = "user.csv";

        readUserFromFile(fileName, userList);

        isLogin = false;

        for (UserModel user : userList) {

            if (user.getUserName().equals(userName) && user.getUserPassword().equals(userPassword)) {

                isLogin = true;

                break;

            }

        }

        if (isLogin) {

            JOptionPane.showMessageDialog(null, "Login successfully");

            // go to StudenListGUI

            StudentListGUI studentListGUI = new StudentListGUI();

            studentListGUI.setVisible(true);

        } else {

            JOptionPane.showMessageDialog(null, "Login failed");

        }

    }

### WriteToFile

When the user performs an account or student registration, the program writes their data onto a file, if the file does not yet exist, the program creates a file for it to be written on. It uses FileWriter to write each User information on to a line and end each User information with a “\n” meaning a line break.

// write to file

    public static void writeUserToFile(String fileName, String userInfo2) {

        try {

            File file = new File(fileName);

            if (!file.exists()) {

                file.createNewFile();

            }

            // write to file

            FileWriter fw = new FileWriter(file, true);

            fw.write(userInfo2 + "\n");

            fw.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

### CheckIfExist

When a user registers for a new User, the program checks if the username already exists on the system. It does this by reading the File for Users and try to find a match with the input by the user. If there is a match, it returns a boolean depicting the status of the finding.

// Check if userName is already exist

    public static boolean checkUserName(String userName) {

        boolean isExist = false;

        String fileName = "user.csv";

        readUserFromFile(fileName, userList);

        for (UserModel user : userList) {

            if (user.getUserName().equals(userName)) {

                isExist = true;

                break;

            }

        }

        return isExist;

    }

Similar algorithms is applied for checking existing Student when registering for a new one

// check if the student is already exist

    public static boolean checkStudent(String id) {

        boolean isExist = false;

        String fileName = "student.csv";

        readStudentFromFile(fileName, studentList);

        for (StudentModel student : studentList) {

            if (student.getId().equals(id)) {

                isExist = true;

            }

        }

        return isExist;

    }

### FindStudentByID

This feature takes in a String as the ID needed to be searched for. It reads the file for a list of Students and go through the list to find the Student with matching id before returning that Student.

// Find student by id

    public static StudentModel findStudent(String id) {

        StudentModel student = null;

        String fileName = "student.csv";

        // clear studentList

        studentList.clear();

        readStudentFromFile(fileName, studentList);

        for (StudentModel s : studentList) {

            if (s.getId().equals(id)) {

                student = s;

            }

        }

        return student;

    }

### FindStudentByName

This operates similarly to FindStudentByID, but instead of returning just one student, it return a list of students that have name containing the input characters.

// Find student by name and return a list of students

    public static ArrayList<StudentModel> findStudentByName(String name) {

        ArrayList<StudentModel> studentList = new ArrayList<StudentModel>();

        String fileName = "student.csv";

        readStudentFromFile(fileName, studentList);

        ArrayList<StudentModel> studentList2 = new ArrayList<StudentModel>();

        for (StudentModel student : studentList) {

            if (student.getName().contains(name)) {

                studentList2.add(student);

            }

        }

        return studentList2;

    }

### DeleteStudent

This feature takes in the ID of the student needed to delete, find the Student with that ID and deletes it from the ArrayList. It then clears all the data in the saved file and replace it with the updated list

// Find student id and delete it

    public static void deleteStudent(String id) {

        StudentModel student = findStudent(id);

        if (student != null) {

            studentList.remove(student);

            String studentInfo2 = student.getStudentInfo2();

            String fileName = "student.csv";

            deleteAllStudent();

            for (StudentModel s : studentList) {

                writeStudentToFile(fileName, s.getStudentInfo2());

            }

        } else {

            JOptionPane.showMessageDialog(null, "Student not found");

        }

    }

### DeleteAllStudent

This feature deletes all students in the file. It works by opening up the student file and replace everything with a single space “ “.

// Delete data on the file and replace with the new studentList

    public static void deleteAllStudent() {

        String fileName = "student.csv";

        try {

            File file = new File(fileName);

            if (!file.exists()) {

                file.createNewFile();

            }

            FileWriter fw = new FileWriter(file, false);

            fw.write("");

            fw.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

### SortStudentByGPA

This algorithms first clears all the elements in the studentList and get a new studentList by reading the file to avoid data duplication. It then uses comparator to sort the list by GPA in ascending or descending order

// Sort the student list by gpa ascending

    public static void sortStudentByGpaAscending() {

        studentList.clear();

        studentList = getStudentList();

        Collections.sort(studentList, new Comparator<StudentModel>() {

            @Override

            public int compare(StudentModel o1, StudentModel o2) {

                // round up the return

                return (int) Math.round(o1.getGpa() - o2.getGpa());

            }

        });

        // studentList.clear();

    }

    // Sort the student list by gpa descending

    public static void sortStudentByGpaDescending() {

        studentList.clear();

        studentList = getStudentList();

        Collections.sort(studentList, new Comparator<StudentModel>() {

            @Override

            public int compare(StudentModel o1, StudentModel o2) {

                // round up the return

                return (int) Math.round(o2.getGpa() - o1.getGpa());

            }

        });

        // studentList.clear();

    }

### populateTable

After the program has read and put the information into an ArrayList, the table needs to be populated with the list data to display it to the user. It takes in the table needed to populate, goes through each of the Student and fill each row with the information of each Student.

// populate table, read from studentList and write to table with their id, name,

    // class, math, eng, gpa

    public static void populateTable(JTable table) {

        // clear table

        table.setModel(new DefaultTableModel());

        // populate table

        for (StudentModel student : studentList) {

            String[] row = { student.getId(), student.getName(), student.getClassId(), student.getMathGrade() + "",

                    student.getEnglishGrade() + "", student.getGpa() + "" };

            ((DefaultTableModel) table.getModel()).addRow(row);

        }

        // if number of rows in the table

        if (table.getRowCount() > 0) {

            isNull = false;

        } else {

            isNull = true;

        }

    }

## error handling

### Checking email address

The conventional email format is [name@domain.domain](mailto:name@domain.domain). To ensure that the user enter the correct email format in order for the student to be registered into the system. To do this, regex is utilized so that the program can check for the formatting of the input.

// check the email format

    public static boolean checkEmail(String email) {

        boolean isValid = false;

        String emailRegex = "^[a-zA-Z0-9\_+&\*-]+(?:\\." + "[a-zA-Z0-9\_+&\*-]+)\*@"

                + "(?:[a-zA-Z0-9-]+\\.)+[a-z" + "A-Z]{2,7}$";

        if (email.matches(emailRegex)) {

            isValid = true;

        }

        return isValid;

    }

### Checking phone number

The phone number conventional format is 10 digits of integer from 0 to 9, the user has to input the correct format to add a new student into the system.

// Check phone number format

    public static boolean checkPhone(String phone) {

        boolean isValid = false;

        String phoneRegex = "^[0-9]{10}$";

        if (phone.matches(phoneRegex)) {

            isValid = true;

        }

        return isValid;

    }

### Check date

The conventional date format is dd/mm/yyyy, to make sure that the user input the correct format before a student is added into the system, a date checker using regex is implemented.

// check the date format

    public static boolean checkDate(String date) {

        boolean isValid = false;

        String dateRegex = "^(?:(?:31(\\/|-|\\.)(?:0?[13578]|1[02]))\\1|(?:(?:29|30)(\\/|-|\\.)"

                + "(?:0?[1,3-9]|1[0-2])\\2))(?:(?:1[6-9]|[2-9]\\d)?\\d{2})$|"

                + "(?:29(\\/|-|\\.)0?2\\3(?:(?:(?:1[6-9]|[2-9]\\d)?(?:0[48]|[2468][048]|[13579][26])|"

                + "(?:(?:16|[2468][048]|[3579][26])00))))$|"

                + "(?:0?[1-9]|1\\d|2[0-8])(\\/|-|\\.)(?:(?:0?[1-9])|(?:1[0-2]))\\4(?:(?:1[6-9]|[2-9]\\d)?\\d{2})$";

        if (date.matches(dateRegex)) {

            isValid = true;

        }

        return isValid;

    }

### Check grade

The grading format for this program is an integer from 0-100 inclusive, methods have been implemented to make sure that the user enters the right format if they want to add a new student into the system. First the program reads the input to see if it is an integer, if not, the user needs to re-enter, if yes, the program checks if the integer is in the range of 0-100 before letting the Student be added into the program.

 try {

                        Integer.parseInt(engGradeField.getText());

                        Integer.parseInt(mathGradeField.getText());

                } catch (NumberFormatException e) {

                        JOptionPane.showMessageDialog(this, "Please enter valid grades", "Error",

                                        JOptionPane.ERROR\_MESSAGE);

                        return;

                }

// check grade if it is valid

    public static boolean checkGrade(int grade) {

        boolean isValid = false;

        if (grade >= 0 && grade <= 100) {

            isValid = true;

        }

        return isValid;

    }

# Test

# Result

# Conclusion

# References

Tutorialspoint, 2022. *MVC Framework - Introduction.* [Online]   
Available at: https://www.tutorialspoint.com/mvc\_framework/mvc\_framework\_introduction.htm  
[Accessed 24 June 2022].