


ASSIGNMENT 1 FRONT SHEET

Qualification	BTEC Level 5 HND Diploma in Computing		
Unit number and title			
Submission date	27 – June – 2022	Date Received 1st submission	
Re-submission Date		Date Received 2nd submission	
Student Name	Trinh Duc Anh	Student ID	GCH210829
Class	GCH1002	Assessor name	Lecturer Manh
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	5
II.	Requirements.....	5
III.	UI design	6
1.	Login	6
2.	Registering	7
3.	Student list.....	8
4.	Add student	9
5.	Edit student	10
6.	View student information	11
IV.	Implementation	11
1.	Technologies used	11
1.1.	MVC model	11
1.2.	NetBeans IDE	12
1.3.	Visual Studio Code	12
2.	Classes	12
2.1.	PersonModel.....	12
2.2.	StudentModel	13
2.3.	UserModel	14
2.4.	LoginController	14
2.5.	RegisterController.....	14
2.6.	StudentController	14
3.	Important algorithms	15
3.1.	calculateGPA	15
3.2.	ReadUserFromFile	15
3.3.	UserLogin	16
3.4.	WriteToFile	17
3.5.	CheckIfExist.....	18
3.6.	FindStudentByID	19
3.7.	FindStudentByName	19
3.8.	DeleteStudent.....	20
3.9.	DeleteAllStudent.....	20
3.10.	SortStudentByGPA	21
3.11.	populateTable	22
4.	Error handling.....	23

4.1.	Checking email address	23
4.2.	Checking phone number	24
4.3.	Check date	24
4.4.	Check grade	25
4.5.	Try – Catch	25
V.	Test	26
1.	Test plan	26
2.	JUnit test.....	31
2.1.	testRegisterStudent	33
2.2.	testGetStudentList	34
2.3.	testCheckEmail	34
2.4.	testCheckDate.....	35
2.5.	testCheckPhone	35
2.6.	testCheckStudent.....	36
2.7.	testReadStudentFromFile	37
2.8.	testDeleteStudent.....	37
2.9.	testDeleteAllStudent.....	38
2.10.	testSortStudentByGpaAscending.....	39
2.11.	testSortStudentByGpaDescending.....	40
	JUnit tests results	41
VI.	Result	42
VII.	Conclusion	59
1.	Evaluation.....	59
	Pro:.....	59
	Cons	60
	Potential improvement	60
	References	61

I. Introduction

The staff from Doan Ket secondary school is in need of a program that allows them to conveniently manage students' information. For that, they have reached out to FPT Software, where the author is currently employed as a software engineer, to request for a small software that has functions where students' information can be modified and updated easily and intuitively.

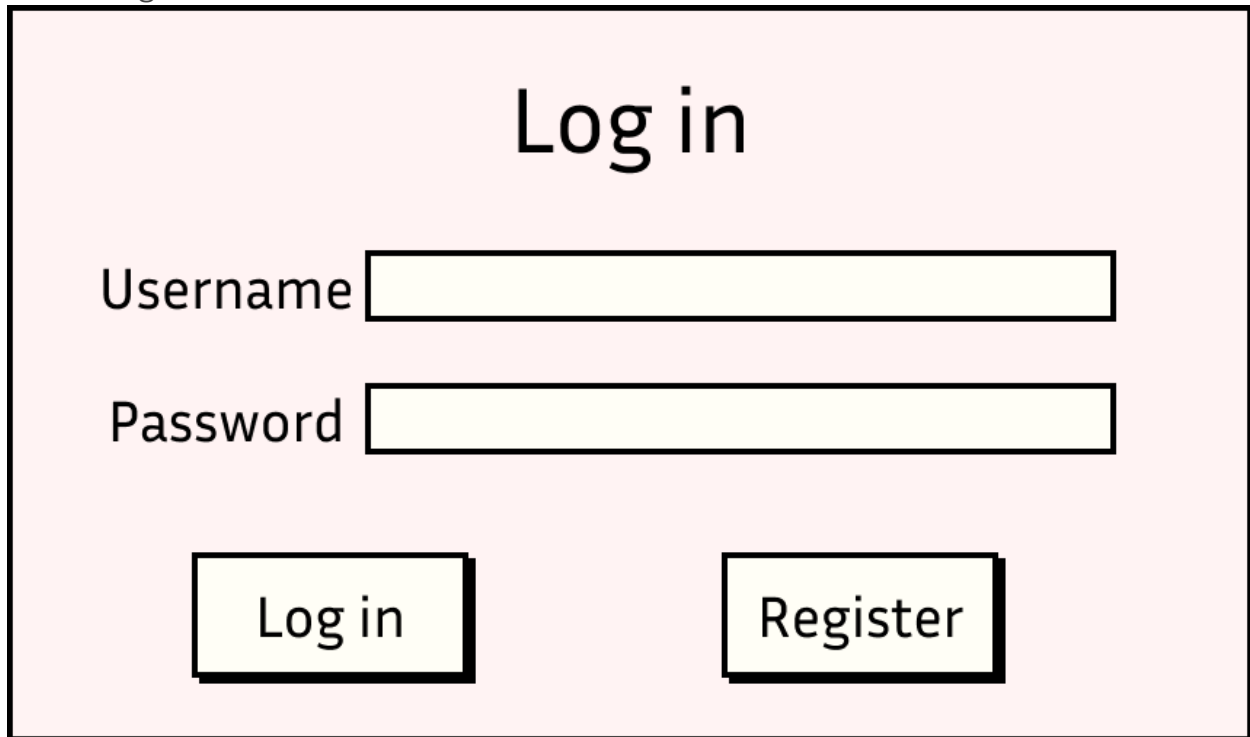
II. Requirements

From the perspective of the school's staff, the users of this program, there is a number of requirements expected from this application

- The program must have a log in process so that only authorized personnel can have access to the system and modify the information on it.
- The program must have an index screen where basic information of students is displayed and buttons to perform different functionalities.
- The program must have features that let the user add, view, and edit a student's information.
- The program needs to have a search function that lets the user search for student(s) by their ID or name.
- There must be a sort button that sorts the student list by their GPA to find the top-performing students.
- The program must check for ID duplication before inserting a new student into the system.
- The program must store the student data in a CSV file with “,” delimiter so the data can be read by multiple software.

III. UI design

1. Login



The image shows a login screen with a light pink background. At the top center is the title "Log in" in a large, black, sans-serif font. Below the title are two input fields. The first is labeled "Username" and the second is labeled "Password", both in a black, sans-serif font. Each label is followed by a yellow rectangular input box with a black border. At the bottom of the screen are two buttons. The left button is labeled "Log in" and the right button is labeled "Register", both in a black, sans-serif font. Each button is a yellow rectangle with a black border.

Log in

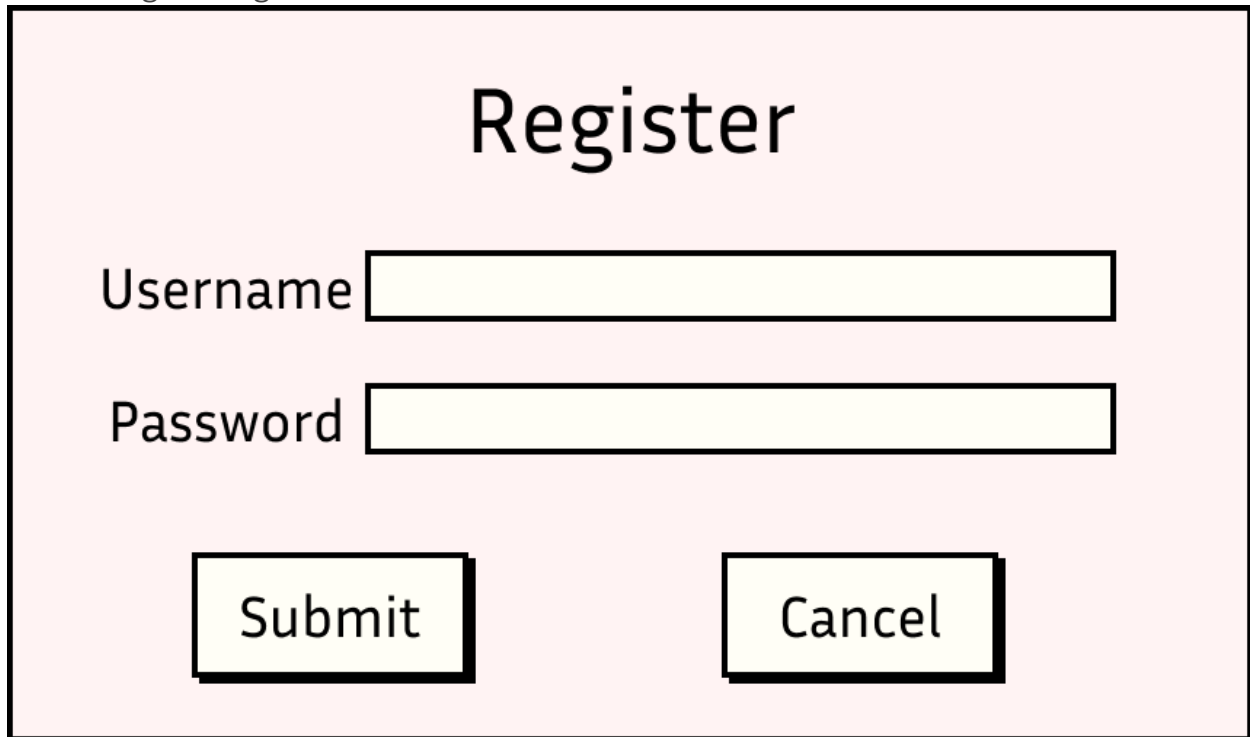
Username

Password

Log in Register

FIGURE 1 LOGIN SCREEN

2. Registering



The diagram shows a rectangular window with a light pink background and a black border. At the top center is the title "Register" in a large, black, sans-serif font. Below the title are two input fields. The first is labeled "Username" in black text to its left, followed by a yellow rectangular box with a black border. The second is labeled "Password" in black text to its left, followed by another yellow rectangular box with a black border. At the bottom of the window are two buttons. The left button is labeled "Submit" in black text and the right button is labeled "Cancel" in black text. Both buttons are yellow rectangles with black borders.

Register

Username

Password

FIGURE 2 REGISTER WINDOW

3. Student list

ID	Name	Class	Math	Eng	GPA

Number of students:

FIGURE 3 INDEX WINDOW

4. Add student

ID	<input type="text"/>
Name	<input type="text"/>
Date of birth	<input type="text"/>
Phone number	<input type="text"/>
Email address	<input type="text"/>
Address	<input type="text"/>
Class	<input type="text"/>
English	<input type="text"/>
Math	<input type="text"/>
<input type="button" value="Submit"/>	
<input type="button" value="Cancel"/>	

FIGURE 4 WINDOW TO ADD NEW STUDENT

5. Edit student

ID	<input type="text"/>
Name	<input type="text"/>
Date of birth	<input type="text"/>
Phone number	<input type="text"/>
Email address	<input type="text"/>
Address	<input type="text"/>
Class	<input type="text"/>
English	<input type="text"/>
Math	<input type="text"/>
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

FIGURE 5 EDIT A STUDENT'S INFORMATION

6. View student information

ID	<input type="text"/>
Name	<input type="text"/>
Date of birth	<input type="text"/>
Phone number	<input type="text"/>
Email address	<input type="text"/>
Address	<input type="text"/>
Class	<input type="text"/>
English	<input type="text"/>
Math	<input type="text"/>
<div><input type="button" value="Submit"/><input type="button" value="Cancel"/></div>	

FIGURE 6 VIEW A STUDENT'S INFORMATION

IV. Implementation

1. Technologies used

1.1. MVC model

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)

MVC is an architectural paradigm that divides an application into three basic logical components: the model, the view, and the controller. Each of these components is designed to handle particular parts of an application's development. MVC is a popular industry-standard web development framework for creating scalable and flexible projects. (Tutorialspoint, 2022)

The Model component represents all of the data-related logic with which the user interacts. This might represent either the data being transmitted between the View and Controller components, or any additional data related to business logic. A Customer object, for example, will take customer information from a database, change it, and then return the data to the database or utilize it to render data. (Tutorialspoint, 2022)

The View component is responsible for the application's UI logic. For example, the Customer view will have all UI components that the final user interacts with, such as text fields, dropdowns, and so on. (Tutorialspoint, 2022)

Controllers serve as an interface between the Model and View components, processing all business logic and incoming requests, manipulating data using the Model component, and interacting with Views to generate the

final output. For example, the Customer controller will handle all interactions and inputs from the Customer View and will use the Customer Model to update the database. To display the Customer data, the same controller will be utilized. (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.

In this program, any change to the data and memory is immediately recorded onto a file on the drive and the data on the memory would be wiped after any operation. This is done to prevent any conflict that might happened between stored data and the data memory. Furthermore, in the event that the computer suddenly shut down during the usage of this program, the data is already automatically saved onto the drive, preventing data loss for unexpected risk.

1.2. NetBeans IDE

NetBeans is an open-source integrated development environment (IDE) for programming languages such as Java, PHP, C++, and others. NetBeans is also known as a modular component platform for creating Java desktop applications. (Techopedia, 2011)

Netbeans offer powerful tools to build a Java application with Graphical User Interface. For this reason, it is used in this project to create an application that allow a user to manage student information with an intuitive user experience.

1.3. Visual Studio Code

Visual Studio Code is a lightweight yet sophisticated source code editor for Windows, macOS, and Linux that runs on the desktop. It has built-in support for JavaScript, TypeScript, and Node.js, as well as a robust ecosystem of extensions for additional languages and runtimes (including C++, C#, Java, Python, PHP, Go, and.NET). (Visualstudio, 2022)

Visual Studio Code offers powerful extensions that aid developers in the process of writing code such as IntelliSense and Copilot. These two extensions are those that take a look at the whole project's code and make suggestions based on the context of the code. One strong upside of both is that they recognize and remember variables and functions' name along with how they work. As these technologies are developed with AI and machine learning, the developers can write the description of what they want to program, and the extensions would base on the code's context and suggest relevant code for them to implement. Due to this great helpfulness, developers can avoid making trivial human errors that can lead to humongous consequences.

2. Classes

2.1. 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

2.2. 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

2.3. 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

2.4. 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.

2.5. 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.

2.6. 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.

3. Important algorithms

3.1. 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

```
// StudentModel method but dont take in gpa
public StudentModel(String name, String id, String email, String phone,
    int englishGrade, String classId) {
    super(name, id, email, phone, address, dob);
    this.mathGrade = mathGrade;
    this.englishGrade = englishGrade;
    this.gpa = calculateGpa();
    this.classId = classId;
}
```

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;
}
```

3.2. 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(csvSplitBy);
            // 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();
    }
}

```

3.3. 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");
    }
}

```

3.4. 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();
    }
}

```

3.5. 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;
}

```

3.6. 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;
}

```

3.7. 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;
    }
}

```

3.8. 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");
    }
}
}

```

3.9. 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();
        }
    }
}
```

3.10. SortStudentByGPA

This algorithm 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();
}

```

3.11. 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;
    }
}
}

```

4. Error handling

4.1. Checking email address

The conventional email format is [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;
}

```

```
}
```

4.2. 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;
}
```

4.3. 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})$|"
        + "(?:29(\\//|-|\\.))0?2\\d3(?:((?:1[6-9]|"
        + "[2-9]\\d)?(?:0[48]|[2468][048]|[13579][26]))|"
        + "(?:((?:16|[2468][048]|[3579][26])00)))$|"
        + "(?: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;
    }

```

4.4. 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;
}

```

4.5. Try – Catch

In multiple parts of the program's code, try – catch is used to handle error. The program first executes the code inside the Try part, should an error happen, the code inside the Catch part would be executed. This way of programming is implemented throughout the program. For example,

```

try {
    Integer.parseInt(engGradeField.getText());
    Integer.parseInt(mathGradeField.getText());
} catch (NumberFormatException e) {

```

```
JOptionPane.showMessageDialog(this, "Please enter valid
grades", "Error",
JOptionPane.ERROR_MESSAGE);
return;
}
```

In this piece of code, the program tries to parse the text inside two fields as integers, if one of them is not an integer, the error happens and the code inside the Catch part is execute, which is a message window popping up saying that the user needs to enter a valid grade. Another example:

```
try {
    File file = new File(fileName);
    if (!file.exists()) {
        file.createNewFile();
    }
    // write to file
    FileWriter fw = new FileWriter(file, true);
    fw.write(studentInfo + "\n");
    fw.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

In this code, the program attempts to open a file, if a file does not exist, it creates the file before writing anything onto it. If anything happens during the execution of the code, the code inside Catch would be executed, printing out what error happens into the console log for the developer to debug.

V. Test

1. Test plan

No	Description	Data	Expected result	Actual result	Status
1	Register with valid data	username: admin password: admin	A success message display, user is returned to login screen, and "admin,admin" is added into user.csv	As expected	Pass
	Register with invalid data by input the same information as last step	username: admin password: admin	A fail message displays, no new data is written on the file user.csv	As expected	Pass
2	Login with valid data	username: admin password: admin	A success message display, user is directed to student index window	As expected	Pass

	Login with invalid data	username: asasasasadmin password: admin	A fail message display, user stays on the login window	As expected	Pass
3	Add a student with valid data	ID:1 name: A date of birth: 12/12/2000 phone number: 1234567890 email address: A@gmail.com address: A street class: 12D english: 40 math: 50	A success message display, user is directed to student index window, the new student is added into the table and student.csv	As expected	Pass
	Add a student with duplicated ID	ID:1 name: B date of birth: 12/12/2000 phone number: 1234567890 email address: B@gmail.com address: A street class: 10A english: 90 math: 32	A failure message display, user is directed to student index window, no data is added to table or student.csv	As expected	Pass
	Add a student with wrong email format	ID:12 name: B date of birth: 12/12/2000 phone number: 1234567890 email address: Bgmailcom address: A street class: 10A english: 90 math: 32	A failure message display, user is directed to student index window, no data is added to table or student.csv	As expected	Pass

Add a student with wrong phone number format	ID:12 name: B date of birth: 12/12/2000 phone number: 1234asffe email address: B@gmail.com address: A street class: 10A english: 90 math: 32	A failure message display, user is directed to student index window, no data is added to table or student.csv	As expected	Pass
Add a student with wrong date format	ID:12 name: B date of birth: 12122000 phone number: 1234567890 email address: B@gmail.com address: A street class: 10A english: 90 math: 32	A failure message display, user is directed to student index window, no data is added to table or student.csv	As expected	Pass
Add a student with wrong grade format	ID:12 name: B date of birth: 12/12/2000 phone number: 1234567890 email address: B@gmail.com address: A street class: 10A english: 90e math: 32	A failure message display, no data is added to table or student.csv	As expected	Pass

		ID:12 name: B date of birth: 12/12/2000 phone number: 1234567890 email address: B@gmail.com address: A street class: 10A english: 90 math: 32e	A failure message display, no data is added to table or student.csv	As expected	Pass
4	Edit a student with valid data	ID:1 name: A edited date of birth: 12/12/2000 phone number: 1234567890 email address: A@gmail.com address: A street class: 12D english: 40 math: 50	Success message display, the user is directed back to the index menu and the data of student whose ID is 1 is updated	As expected	Pass
	Edit a student with wrong email format	ID:1 name: A edited date of birth: 12/12/2000 phone number: 1234567890 email address: Agmailcom address: A street class: 12D english: 40 math: 50	A failure message display, user is directed to student index window, no data is added to table or student.csv	As expected	Pass

Edit a student with wrong phone number format	ID:1 name: A edited date of birth: 12/12/2000 phone number: 12345asd email address: A@gmail.com address: A street class: 12D english: 40 math: 50	A failure message display, user is directed to student index window, no data is added to table or student.csv	As expected	Pass
Edit a student with wrong date format	ID:1 name: A edited date of birth: 12122000 phone number: 1234567890 email address: A@gmail.com address: A street class: 12D english: 40 math: 50	A failure message display, user is directed to student index window, no data is added to table or student.csv	As expected	Pass
Edit a student with wrong grade format	ID:1 name: A edited date of birth: 12/12/2000 phone number: 1234567890 email address: A@gmail.com address: A street class: 12D english: 40e math: 50	A failure message display, no data is added to table or student.csv	As expected	Pass

		ID:1 name: A edited date of birth: 12/12/2000 phone number: 1234567890 email address: A@gmail.com address: A street class: 12D english: 40 math: 50e	A failure message display, no data is added to table or student.csv	As expected	Pass
5	Search for student by name	name: A	All students whose name contains "A" is displayed on the table	As expected	Pass
6	Search for students by ID	ID:1	The student whose ID is 1 is displayed	As expected	Pass
7	Delete all student		All students are deleted from student.csv	As expected	Pass
8	Delete student	ID: 1	The student whose ID is 1 is deleted	As expected	Pass

TABLE 7 TEST PLAN

2. JUnit test

To make the testing done more easily and conveniently, Junit has been implemented to perform an array of multiple tests simultaneously.

To aid the test, some generator has been programmed to generate random data for each column.

```
// random string generator
public String randomString(int length) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < length; i++) {
        sb.append((char) ((int) (Math.random() * 26)
+ 97));
    }
    return sb.toString();
}

// random email generator
public String randomEmail(int length) {
    StringBuilder sb = new StringBuilder();
```

```
        for (int i = 0; i < length; i++) {
            sb.append((char) ((int) (Math.random() * 26)
+ 97));
        }
        return sb.toString() + "@gmail.com";
    }
}
```

// random phone number generator that consists of
all integer

```
public String randomPhone(int length) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < length; i++) {
        sb.append((int) (Math.random() * 10));
    }
    return sb.toString();
}
```

// random date of birth generator, date, month, year
are separated by "/"

```
public String randomDob() {
    StringBuilder sb = new StringBuilder();
    int date = (int) (Math.random() * 30) + 1;
    int month = (int) (Math.random() * 12) + 1;
    int year = (int) (Math.random() * 100) + 1900;
    sb.append(date);
    sb.append("/");
    sb.append(month);
    sb.append("/");
    sb.append(year);
    return sb.toString();
}
```



```
// random grade generator
public int randomGrade() {
    return (int) (Math.random() * 100) + 1;
}
```

2.1. testRegisterStudent

The first implemented test is testRegisterStudent which constructs a new Student object. When a student is successfully registered, the Controller assigns the value “true” to the variable “isRegister”. The test read this variable and determine if the test passes.

```
@Test
public void testRegisterStudent() {
    System.out.println("registerStudent");
    // if a message dialog saying "Register
successfully" is shown, then this test
    // is passed
    // student information include name, id, email,
phone number, address, dob,
    // mathGrade, englishGrade, classId
    String name = randomString(20);
    String id = randomString(20);
    String email = randomEmail(20);
    String phone = randomPhone(10);
    String address = randomString(20);
    String dob = randomDob();
    int mathGrade = randomGrade();
    int englishGrade = randomGrade();
    String classId = randomString(20);
    studentController instance = new
studentController();
    boolean expResult = true;
    instance.registerStudent(name, id, email, phone,
address, dob, mathGrade, englishGrade, classId);
```

```

        boolean result = instance.isRegister;
        assertEquals(expResult, result);
        // TODO review the generated test code and
        remove the default call to fail.
    }

```

2.2. testGetStudentList

The second test is the test of the feature that reads a file and pass the data into a list of Students. The test then read the list, if the list is not empty, then it passes.

```

@Test
    public void testGetStudentList() {
        System.out.println("getStudentList");
        studentController instance = new
studentController();
        // student list is returned
        // ArrayList<Student> expResult = null;
        studentController.studentList.clear();
        ArrayList<StudentModel> result = null;
        instance.getStudentList();
        // if result is not null, then this test is
passed
        assertNotNull(studentController.studentList);

        // TODO review the generated test code and
        remove the default call to fail.
        // fail("The test case is a prototype.");
    }

```

2.3. testCheckEmail

The next test is to see if the checkEmail function can properly validate an email formatted String. The test generate a random String followed by "@gmail.com". If the String follows the conventional email format of email@domain.domain2, then it returns a boolean value "true" and the test passes.

```

@Test
    public void testCheckEmail() {
        // check email with a random email
    }

```

```

        System.out.println("checkEmail");
        String email = randomEmail(20);
        studentController instance = new
studentController();
        boolean expResult = true;
        boolean result = instance.checkEmail(email);
        assertEquals(expResult, result);
        // TODO review the generated test code and
remove the default call to fail.
    }

```

2.4. testCheckDate

The next test is to see if the checkDate function can properly validate a date formatted String. The test generate a random String in the format of dd/mm/yyyy. If the String follows the conventional format, then it returns a boolean value "true" and the test passes.

```

@Test
    public void testCheckDate() {
        // check date in the dd/mm/yyyy
        System.out.println("checkDate");
        String date = randomDob();
        studentController instance = new
studentController();
        boolean expResult = true;
        boolean result = instance.checkDate(date);
        assertEquals(expResult, result);
        // TODO review the generated test code and
remove the default call to fail.
    }

```

2.5. testCheckPhone

The next test is to see if the checkPhone function can properly validate a phone formatted String. The test generate a random String in the format of 10 integer digits. If the String follows the conventional format, then it returns a boolean value "true" and the test passes.

```

@Test
    public void testCheckPhone() {
        // check phone number with a random phone number
        System.out.println("checkPhone");
        String phone = randomPhone(10);
        studentController instance = new
studentController();
        boolean expResult = true;
        boolean result = instance.checkPhone(phone);
        assertEquals(expResult, result);
        // TODO review the generated test code and
remove the default call to fail.
    }

```

2.6. testCheckStudent

The next test is to see if the program can properly return a student when a user search for an ID. The test read the file student.csv, make an ArrayList of Student out of it and check the id column of the first row and calls for the function checkStudent(). If a student is found, then the function returns a "true" boolean value.

```

@Test
    public void testCheckStudent() {
        // read the file student.csv
        // read the second column of the file as id
        // check if the id is in the student list
        // if the id is in the student list, then this
test is passed
        System.out.println("checkStudent");
        studentController instance = new
studentController();
        instance.getStudentList();
        String id =
studentController.studentList.get(1).getId();
        boolean expResult = true;
        boolean result = instance.checkStudent(id);
    }

```

```
        assertEquals(expResult, result);
        // TODO review the generated test code and
        remove the default call to fail.

    }
```

2.7. testReadStudentFromFile

The test `testReadStudentFromFile()` is to test the ability to read from a file and make a List of Student out of it. If the list is no longer null after reading the file, then the test passes

```
@Test
    public void testReadStudentFromFile() {
        // read the file student.csv
        // if student list is not null, then this test
        is passed
        System.out.println("readStudentFromFile");
        studentController instance = new
        studentController();
        instance.readStudentFromFile("student.csv",
        instance.studentList);
        assertNotNull(studentController.studentList);
        // TODO review the generated test code and
        remove the default call to fail.
    }
```

2.8. testDeleteStudent

This test uses `deleteStudent()` to delete a student then use `findStudent` to check for it in the file, if the deleted student is not found anymore, the function returns false and the test passes.

```
@Test
    public void testDeleteStudent() {
        // read the file student.csv, get an id from the
        student list
        // delete the student with the id
```

```

        // if the student is deleted, then the student
when finding with id is null
        // and this test is passed
        System.out.println("deleteStudent");
        studentController instance = new
studentController();
        instance.getStudentList();
        String id =
studentController.studentList.get(1).getId();
        instance.deleteStudent(id);
        StudentModel result = instance.findStudent(id);
        assertNull(result);
        // TODO review the generated test code and
remove the default call to fail.
    }

```

2.9. testDeleteAllStudent

This test uses deleteAllStudent() to delete all data on the file and then compare the file to an empty list, if they are equal, then the test passes.

```

@Test
    public void testDeleteAllStudent() {
        // delete all student in the student list
        // if the student list is empty, then this test
is passed
        System.out.println("deleteAllStudent");
        studentController instance = new
studentController();
        instance.getStudentList();
        instance.deleteAllStudent();
        // expected result = studentList
        // studentList.clear
        ArrayList<StudentModel> expResult =
instance.studentList;
    }

```

```

        expResult.clear();
        assertEquals(expResult,
studentController.studentList);
        // TODO review the generated test code and
remove the default call to fail.
    }

```

2.10. testSortStudentByGpaAscending

This test uses `sortStudentByGpaAscending()` to sort the list of students in the order of GPA ascending and then goes through the list comparing each pair of student's gpa. If the GPA of the next student is greater than the current one, the loop continues. If it keeps happening until the loop ends, it means that the list is in the correct order and the test passes.

```

@Test
    public void testSortStudentByGpaAscending() {
        // sort the student list by the order of gpa
ascending
        // check the gpa of student from first to last,
if it's in
        // ascending order, then this test is passed
        System.out.println("sortStudentByGpaAscending");
        studentController instance = new
studentController();
        instance.getStudentList();
        instance.sortStudentByGpaAscending();
        ArrayList<StudentModel> result =
instance.studentList;
        for (int i = 0; i < result.size() - 1; i++) {
            assertTrue(result.get(i).getGpa() <=
result.get(i + 1).getGpa());
        }
        // TODO review the generated test code and
remove the default call to fail.
    }

```

2.11. testSortStudentByGpaDescending

This test uses `sortStudentByGpaDescending()` to sort the list of students in the order of GPA ascending and then goes through the list comparing each pair of student's gpa. If the GPA of the next student is smaller than the current one, the loop continues. If it keeps happening until the loop ends, it means that the list is in the correct order and the test passes.

```
@Test
    public void testSortStudentByGpaDescending() {
        // sort the student list by the order of gpa
        descending
        // check the gpa of student from first to last,
        if it's in
        // descending order, then this test is passed
        System.out.println("sortStudentByGpaDescending")
    ;

        studentController instance = new
studentController();
        instance.getStudentList();
        instance.sortStudentByGpaDescending();
        ArrayList<StudentModel> result =
instance.studentList;
        for (int i = 0; i < result.size() - 1; i++) {
            assertTrue(result.get(i).getGpa() >=
result.get(i + 1).getGpa());
        }
        // TODO review the generated test code and
        remove the default call to fail.
    }
```


Junit tests results

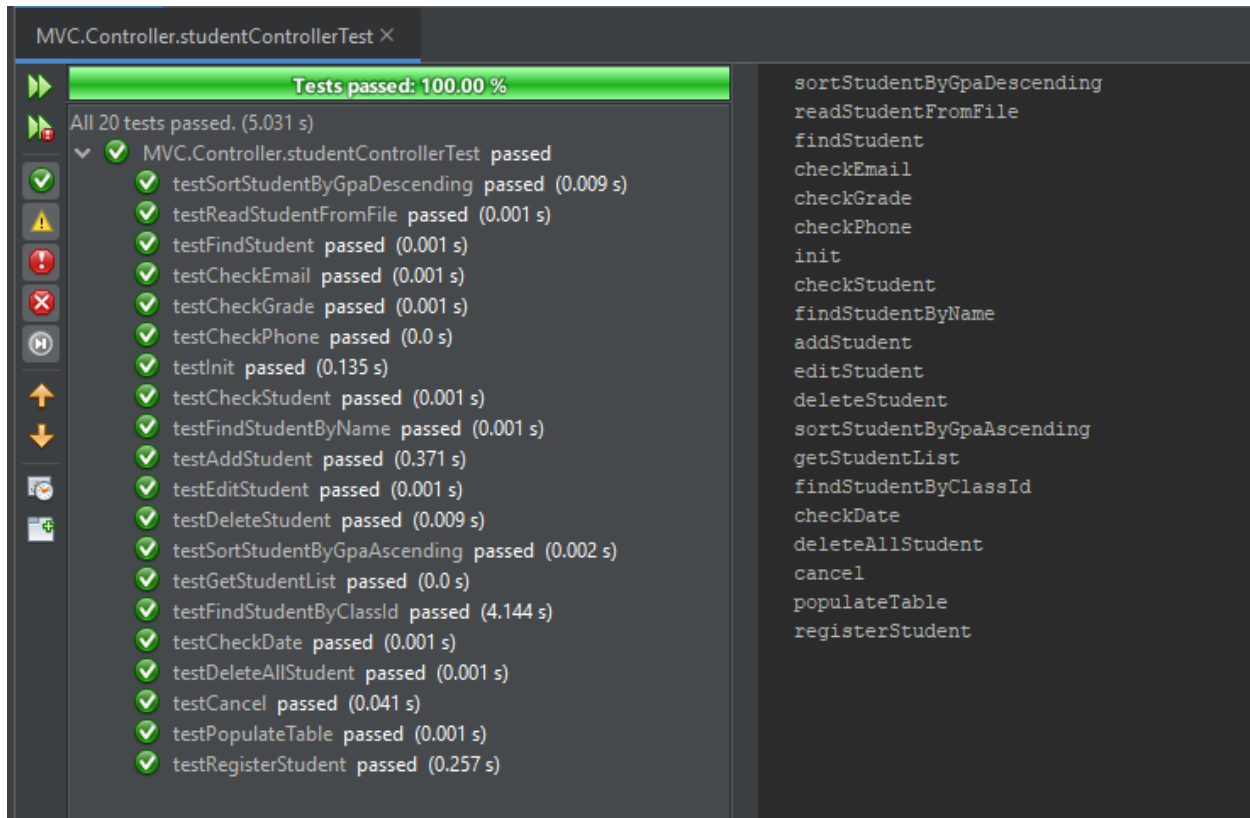
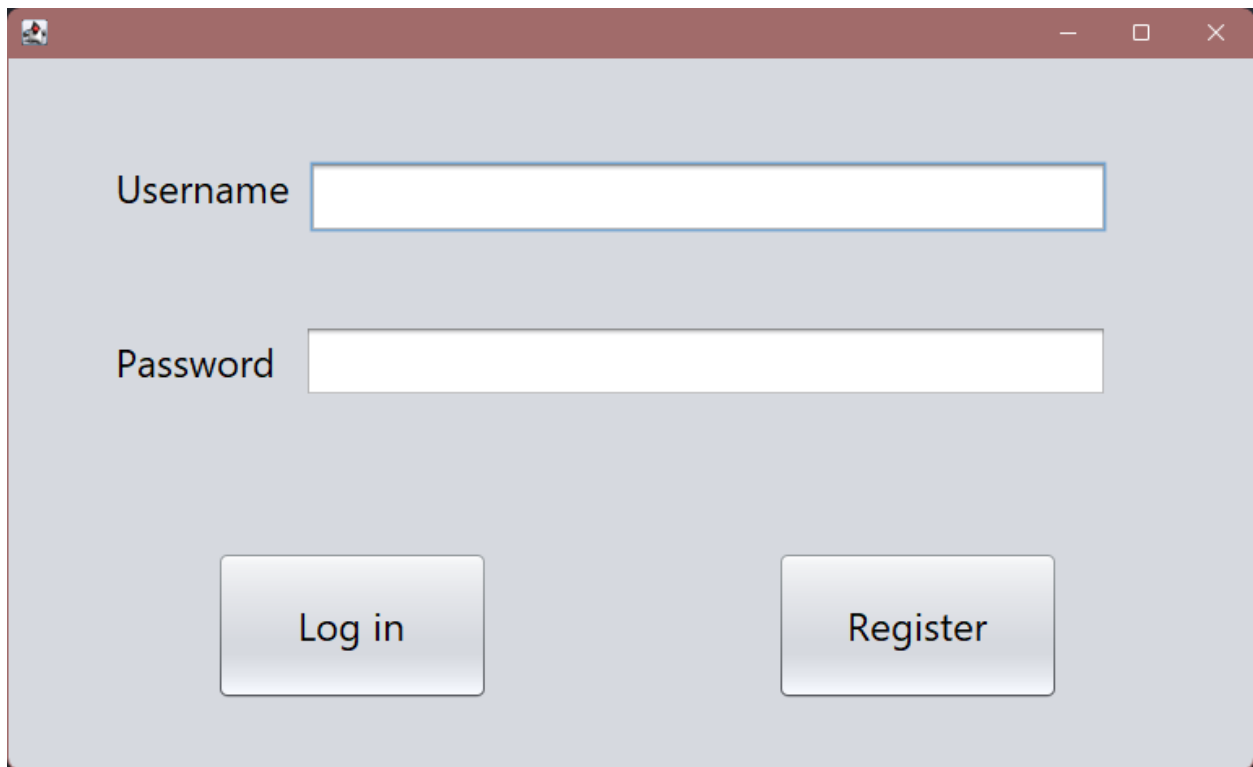


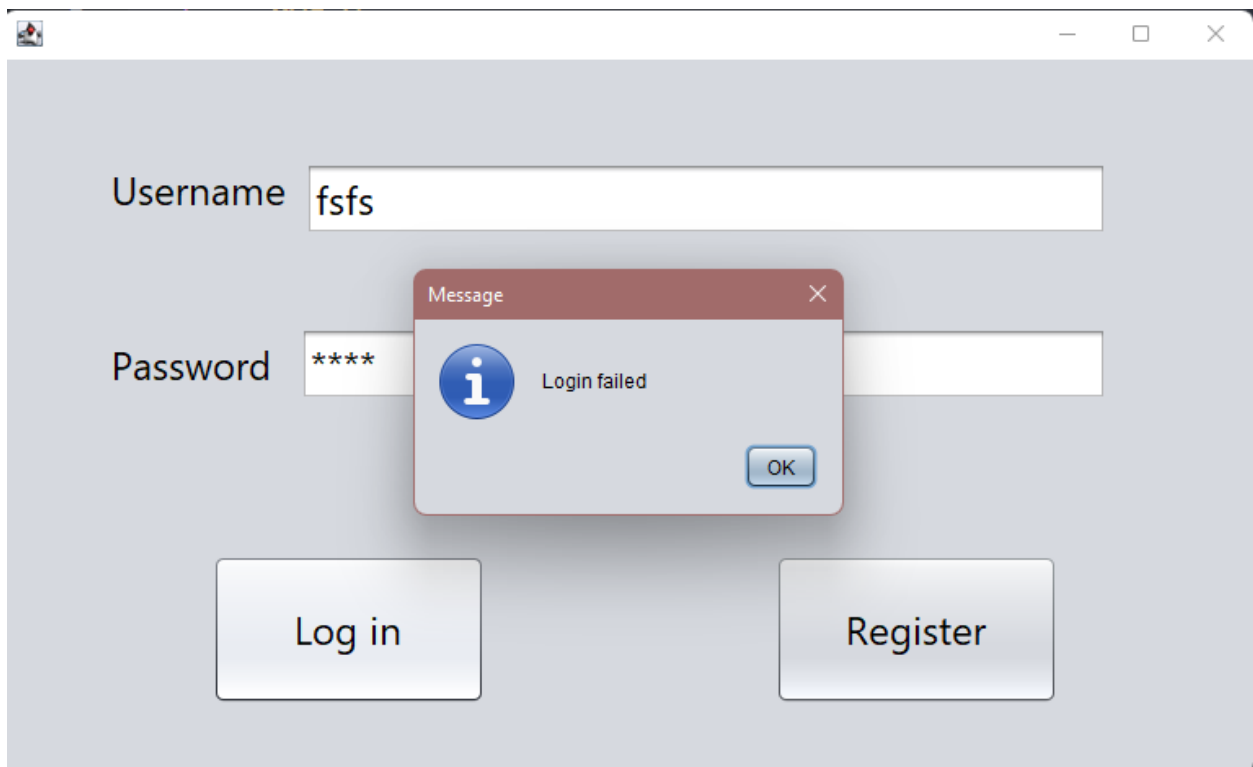
FIGURE 8 THE TESTS RAN SUCCESSFULLY

VI. Result



A screenshot of a login window. It features a light gray background with a dark red title bar at the top containing standard window controls. The window contains two text input fields: one for 'Username' and one for 'Password'. Below these fields are two buttons: 'Log in' and 'Register', both with a light blue gradient and rounded corners.

FIGURE 9 LOGIN WINDOW



A screenshot of the login window showing a failed login attempt. The 'Username' field contains the text 'fsfs' and the 'Password' field contains four asterisks '****'. A modal message box is centered over the form. The message box has a dark red title bar with the text 'Message' and a close button. It contains a blue information icon, the text 'Login failed', and an 'OK' button.

FIGURE 10 LOGIN FAILED

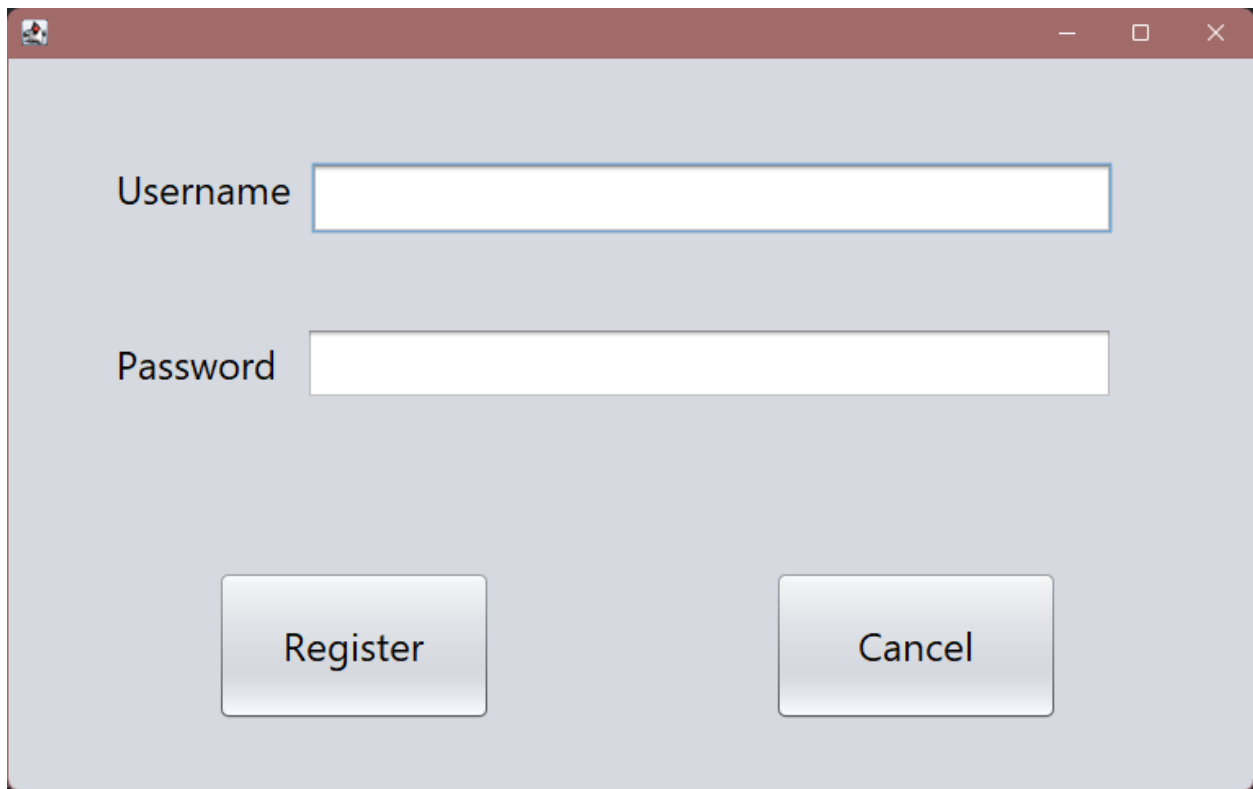


FIGURE 11 REGISTER WINDOW

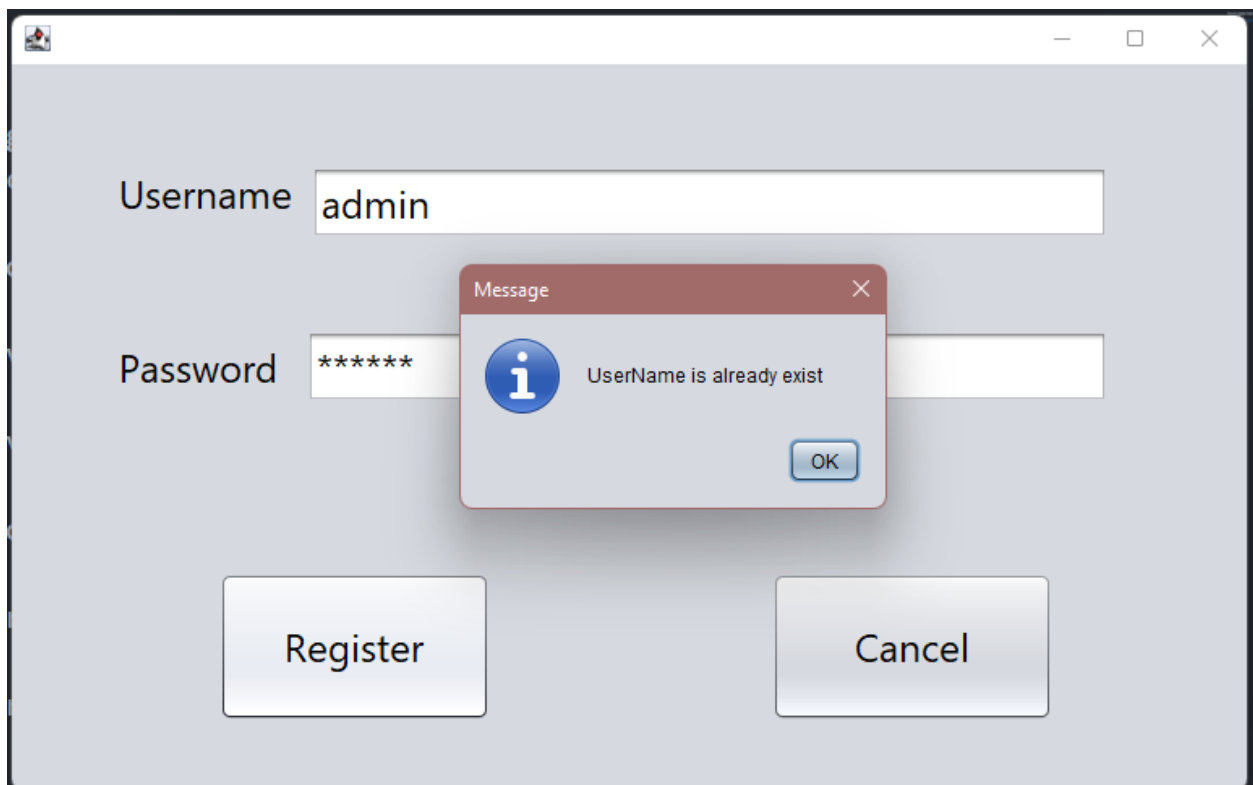


FIGURE 12 REGISTER FAILED

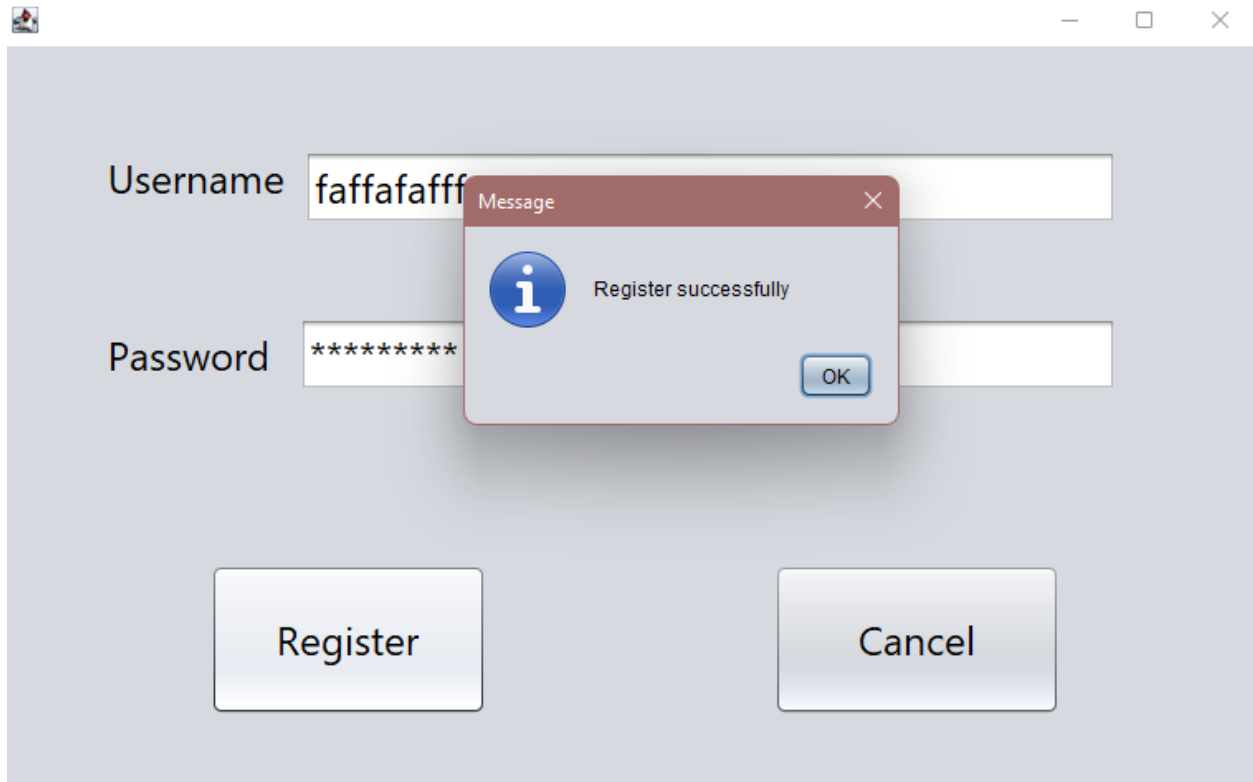


FIGURE 13 SUCCESSFUL REGISTER

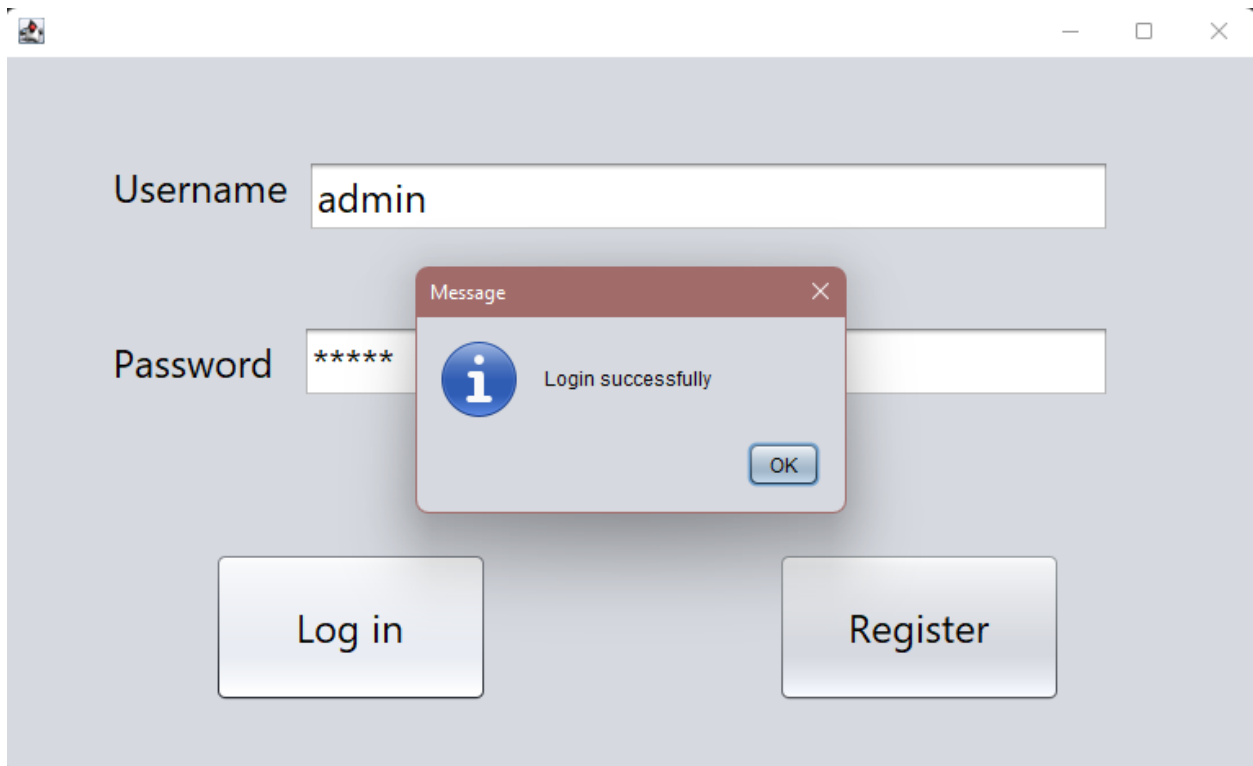


FIGURE 14 SUCCESSFUL LOGIN

—□×

ID	Name	Class	Math	English	GPA
123	werfghnedit	567u	89	87	88.0
324	gregreedit2	21h	35	36	35.5
1234235	ewewfwe	2j	12	32	22.0

Add

Edit

Info

Refresh

Sort GPA ↑

Sort GPA ↓

Delete

Delete all

Number of students: 3

Search name

Search ID

FIGURE 15 STUDENT INDEX WINDOW

— □ ×

ID	Name	Class	Math	English	GPA
123	werfghnedit	567u	89	87	88.0
1234235	ewewfwe	2j	12	32	22.0

Add

Edit

Info

Refresh

Sort GPA ↑

Sort GPA ↓

Delete

Delete all

Number of students: 3

we

Search name

Search ID

FIGURE 16 SEARCH BY NAME

— □ ×

ID	Name	Class	Math	English	GPA
123	werfghnedit	567u	89	87	88.0

Add

Edit

Info

Refresh

Sort GPA ↑

Sort GPA ↓

Delete

Delete all

Number of students: 3

123

Search name

Search ID

FIGURE 17 SEARCH BY ID

ID	Name	Class	Math	English	GPA
1234235	ewewfwe	2j	12	32	22.0
324	gregreedit2	21h	35	36	35.5
123	werfghnedit	567u	89	87	88.0

Add

Edit

Info

Refresh

Sort GPA ↑

Sort GPA ↓

Delete

Delete all

Number of students: 3

Search name

Search ID

FIGURE 18 SORT BY GPA ASCENDING

ID	Name	Class	Math	English	GPA
123	werfghnedit	567u	89	87	88.0
324	gregreedit2	21h	35	36	35.5
1234235	ewewfwe	2j	12	32	22.0

Add

Edit

Info

Refresh

Sort GPA ↑

Sort GPA ↓

Delete

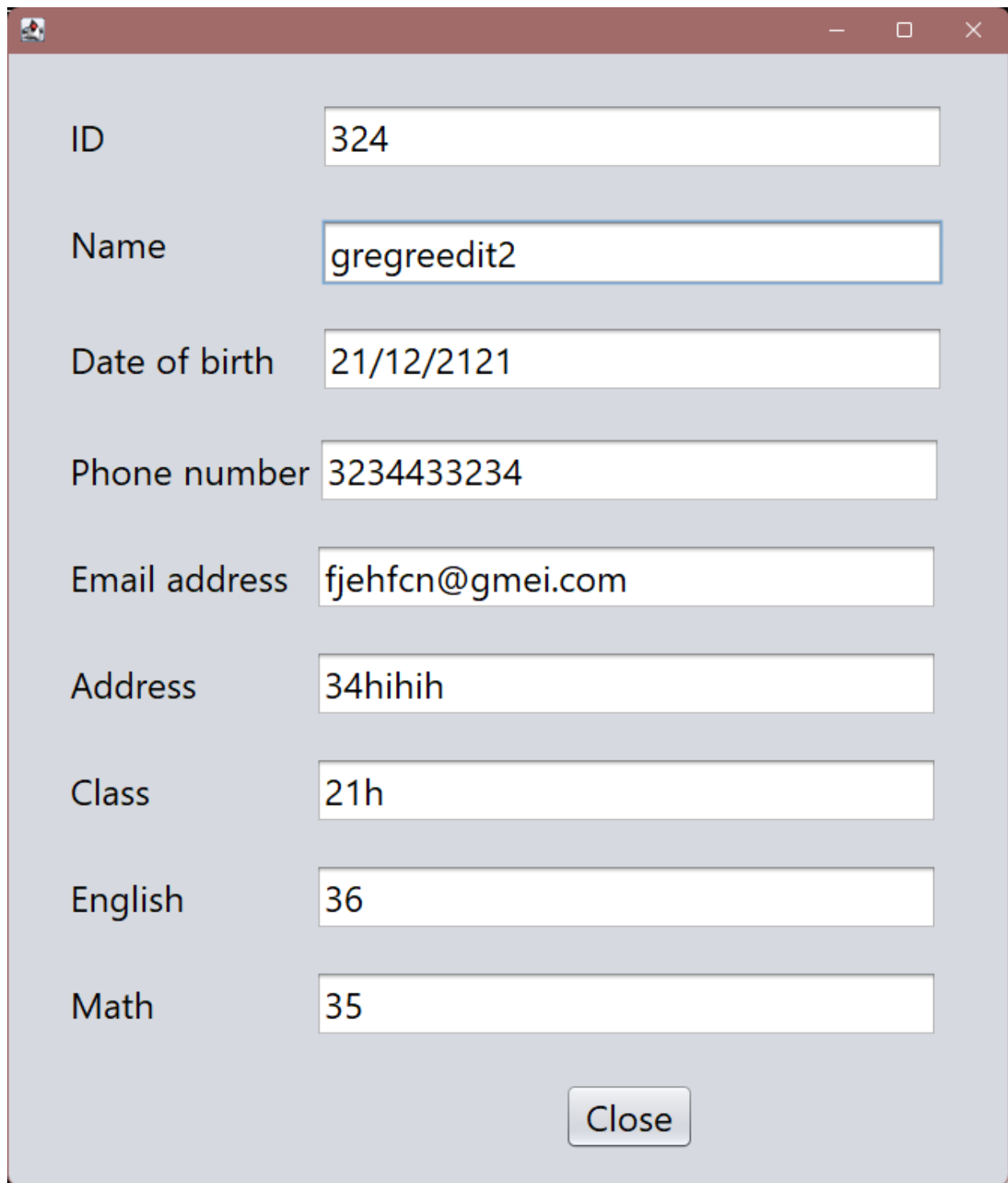
Delete all

Number of students: 3

Search name

Search ID

FIGURE 19 SORT BY GPA DESCENDING



A screenshot of a web application window titled "STUDENT INFORMATION". The window has a light gray background and a dark red title bar with standard window controls (minimize, maximize, close). The form contains the following fields:

Field	Value
ID	324
Name	gregreedit2
Date of birth	21/12/2121
Phone number	3234433234
Email address	fjehfcn@gmei.com
Address	34hihih
Class	21h
English	36
Math	35

A "Close" button is located at the bottom right of the form.

FIGURE 20 STUDENT INFORMATION

ID	1234235
Name	ewewfwe
Date of birth	12/12/1221
Phone number	3251235432
Email address	eggg@ga.ca
Address	32h2hh
Class	2j
English	32
Math	12

FIGURE 21 EDIT STUDENT INFORMATION

ADD A NEW STUDENT

ID

Name

Date of birth

Phone number

Email address

Address

Class

English

Math

FIGURE 22 ADD A NEW STUDENT

— □ ×

ID 12545454

Name fefefef

Date of birth 12/12/2012

Phone number 2313r223rr

Email address

Address


Class 324f

English 43

Math 23

Submit Cancel

Message

 Invalid phone number

OK

FIGURE 23 ADD STUDENT WITH INVALID PHONE NUMBER


Form for adding a student with the following fields and values:

Field	Value
ID	54353
Name	ffqwfwqw
Date of birth	12/12/2012
Phone number	1234567890
Email address	aefgmailcom
Address	
Class	
English	12
Math	32

Buttons: Submit, Cancel

A message dialog box is displayed over the form, indicating an error:

Message

 Invalid email address

OK

FIGURE 24 ADD STUDENT WITH INVALID EMAIL FORMAT

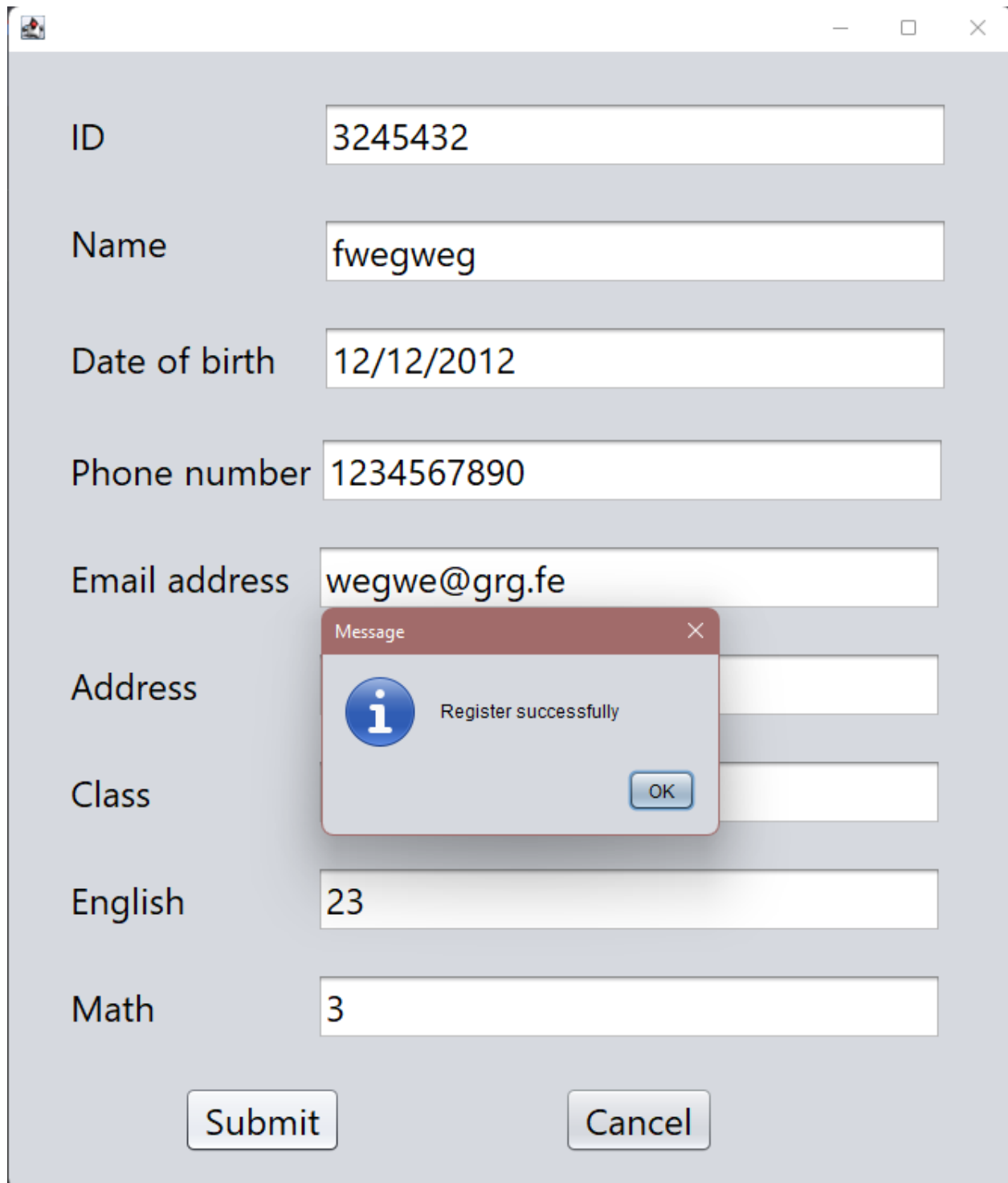
Form for adding a student with the following fields:

- ID: 3245432
- Name: fwegweg
- Date of birth: 12/12/2012
- Phone number: 1234567890
- Email address: wegwe@grg.fe
- Address: (empty)
- Class: (empty)
- English: 23t
- Math: 3t

Buttons: Submit, Cancel

An error dialog box is displayed over the Class field with the message: "Please enter valid grades".

FIGURE 25 ADD STUDENT WITH INVALID GRADE



A screenshot of a student registration form. The form has a light gray background and contains several input fields. A modal message box is overlaid on the form, displaying a blue information icon and the text "Register successfully" with an "OK" button. The form fields are as follows:

Field Label	Value
ID	3245432
Name	fwegweg
Date of birth	12/12/2012
Phone number	1234567890
Email address	wegwe@grg.fe
Address	
Class	
English	23
Math	3

At the bottom of the form, there are two buttons: "Submit" and "Cancel".

FIGURE 26 ADD STUDENT WITH VALID INFORMATION

ID	Name	Class	Math	English	GPA
123	werfghnedit	567u	89	87	88.0
324	gregreedit2	21h	35	36	35.5
1234235	ewewfwe	2j	12	32	22.0
3245432	fwegweg	34e	3	23	13.0

Add

Edit

Info

Refresh

Sort GPA ↑

Sort GPA ↓

Delete

Delete all

Number of students: 4

Search name

Search ID

FIGURE 27 STUDENT LIST AFTER ADDING

—□×

ID	Name	Class	Math	English	GPA
123	werfghnedit	567u	89	87	88.0
1234235	ewewfwe	2j	12	32	22.0
3245432	fwegweg	34e	3	23	13.0

Add

Edit

Info

Refresh

Sort GPA ↑

Sort GPA ↓

Delete

Delete all

Number of students: 3

Search name

Search ID

FIGURE 28 DELETE SELECTED STUDENT

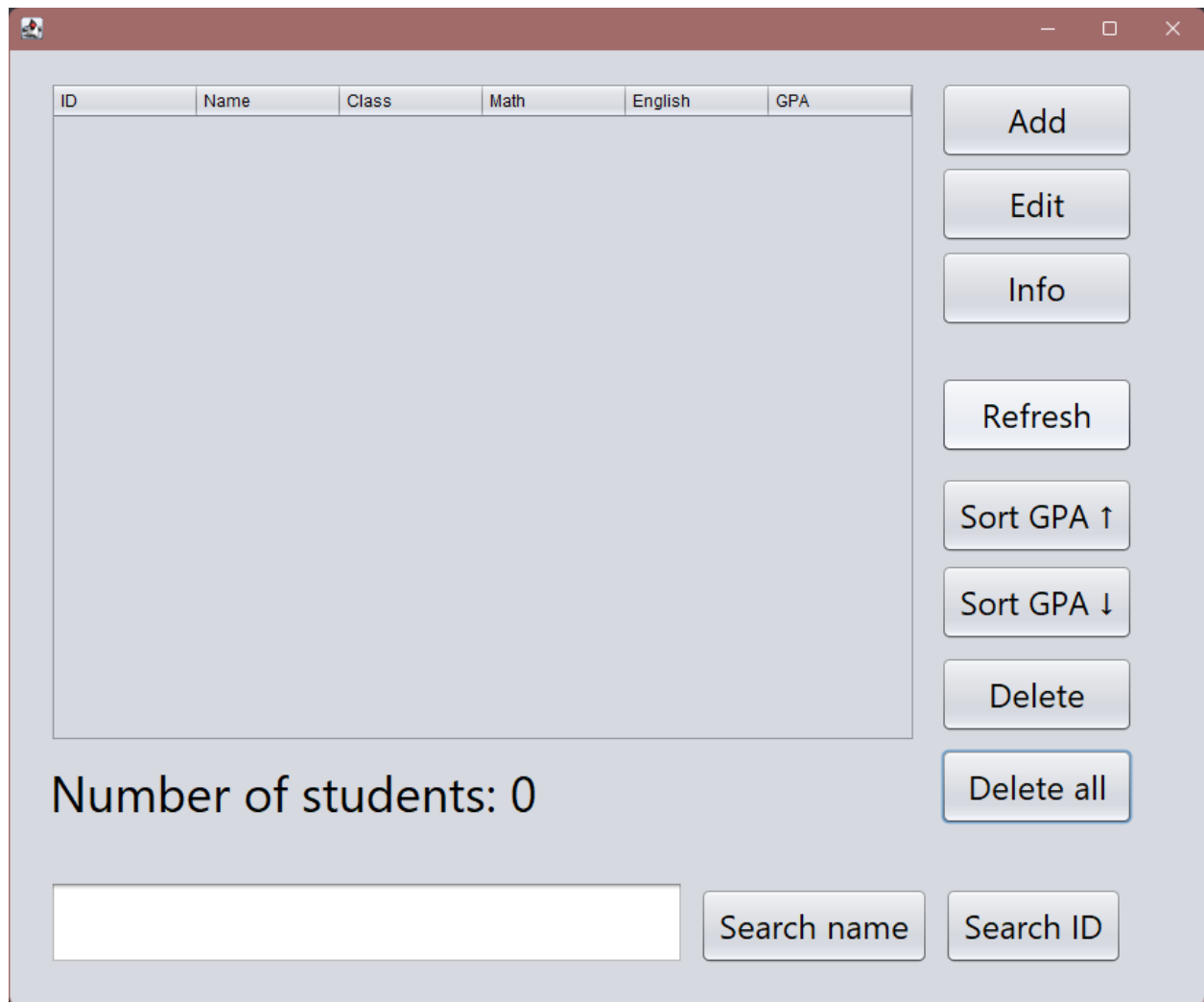


FIGURE 29 DELETE ALL STUDENTS

VII. Conclusion

In this report, the author has addressed the implementation of a software to be developed, which is a student management program used by the school's staff to manage the information of the students.

1. Evaluation

Pro:

- The program has met all expectations of migrating the essential features and functionality of the previous application into one with GUI for an intuitive user experience. The user can enter the information in text boxes, add them into the employees list or edit an existing employee. The employees' data are displayed in a table on the bottom half of the application window. The user can then use other functions to manipulate the employees' information like Delete, Sort, Search, Import, Export.
- The program's GUI is designed using Netbean IDE to give the user an intuitive user experience using provided tools in the software's library.
- The program's Junit tests was done using Netbean IDE to test multiple functions simultaneously, saving time for the developer during the testing phase.

- The program's code is edited using Visual Studio Code. Thanks to powerful extension like IntelliSense and Github Copilot that can give the developers code suggestions on-the-go. The developers could write more accurate and concise code because of these tools.
- Object-oriented, event-driven, procedural programming have all be incorporated to store data, perform function up to the user wants.
- Data validation is implemented to make sure that the input data follows the right convention.
- The program saves the data onto a file after every operation and clears the one on the memory (RAM). This way, in case of power outage, the data is less likely to be lost.

Cons

- The program only validates the date format (dd/mm/yyyy) but not the date data. If the user enters a valid date format but invalid date data such as 30/02/2022, the program automatically interprets it as 02/03/2022
- The program still lacks a logout button, so the user has to exit the program in order to log out of the account.

Potential improvement

- The program can have an encryption/decryption process where login information is encrypted.
- The program can implement the features to manage the information of classes and lecturers in the future development.
- Features where student reports are generated automatically monthly can be developed should the program require any additional development.

References

Tutorialspoint, 2022. *MVC Framework - Introduction*. [Online]

Available at: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

[Accessed 24 June 2022].