


## ASSIGNMENT 1 FRONT SHEET

<b>Qualification</b>	<b>BTEC Level 5 HND Diploma in Computing</b>		
<b>Unit number and title</b>			
<b>Submission date</b>	24 – June – 2022	<b>Date Received 1st submission</b>	
<b>Re-submission Date</b>		<b>Date Received 2nd submission</b>	
<b>Student Name</b>	Trinh Duc Anh	<b>Student ID</b>	GCH210829
<b>Class</b>	GCH1002	<b>Assessor name</b>	Lecturer Manh
<b>Student declaration</b> <p>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.</p>			
		<b>Student's signature</b>	

### 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.	Program structure.....	11
2.	Classes .....	12
2.1.	PersonModel .....	12
2.2.	StudentModel .....	12
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 .....	21
3.10.	SortStudentByGPA .....	21
3.11.	populateTable .....	23
4.	error handling .....	24
4.1.	Checking email address.....	24

4.2.	Checking phone number .....	24
4.3.	Check date.....	24
4.4.	Check grade.....	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 .....	36
2.6.	testCheckStudent.....	36
2.7.	testReadStudentFromFile .....	37
2.8.	testDeleteStudent.....	38
2.9.	testDeleteAllStudent.....	38
2.10.	testSortStudentByGpaAscending.....	39
2.11.	testSortStudentByGpaDescending.....	40
VI.	Result .....	41
VII.	Conclusion.....	59
	References .....	60

## 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.

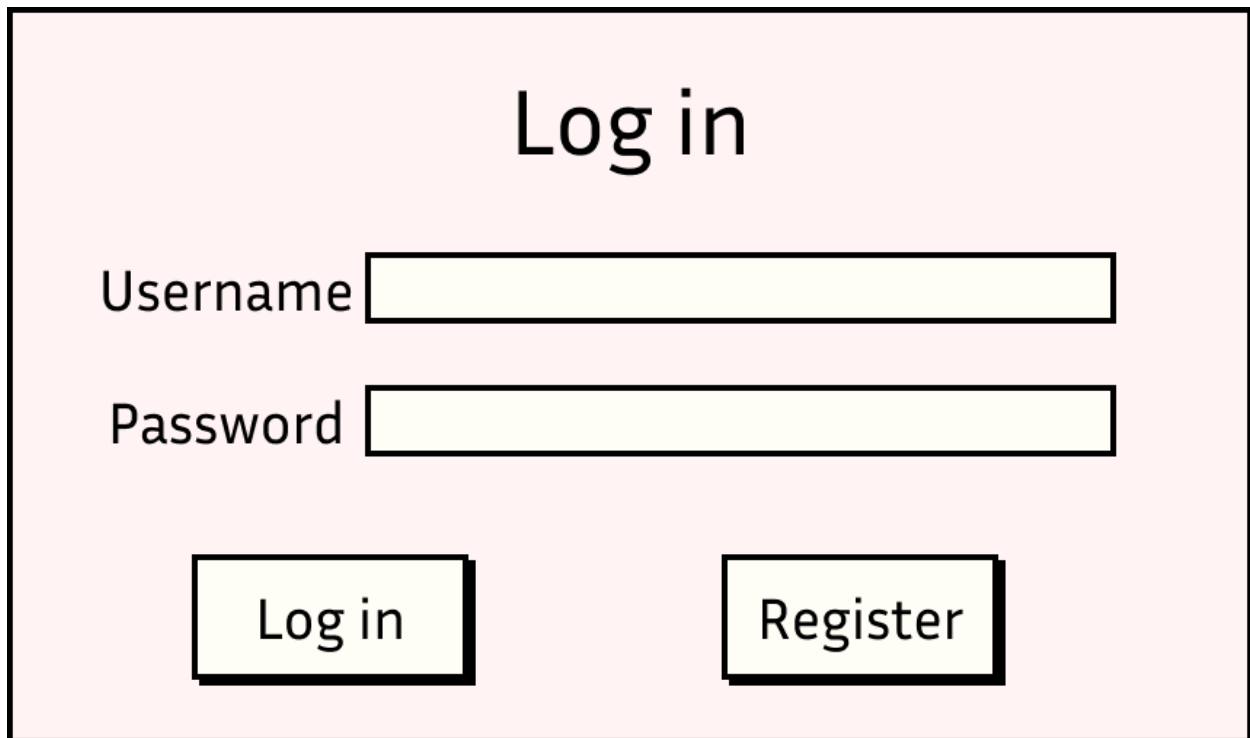
## II. REQUIREMENTS

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 and a black border. At the top center is the text "Log in" in a large, black, sans-serif font. Below this, there are two input fields. The first is labeled "Username" in a black, sans-serif font, followed by a yellow rectangular input box with a black border. The second is labeled "Password" in a black, sans-serif font, followed by a yellow rectangular input box with a black border. At the bottom, there are two buttons. The left button is yellow with a black border and contains the text "Log in" in a black, sans-serif font. The right button is also yellow with a black border and contains the text "Register" in a black, sans-serif font.

Log in

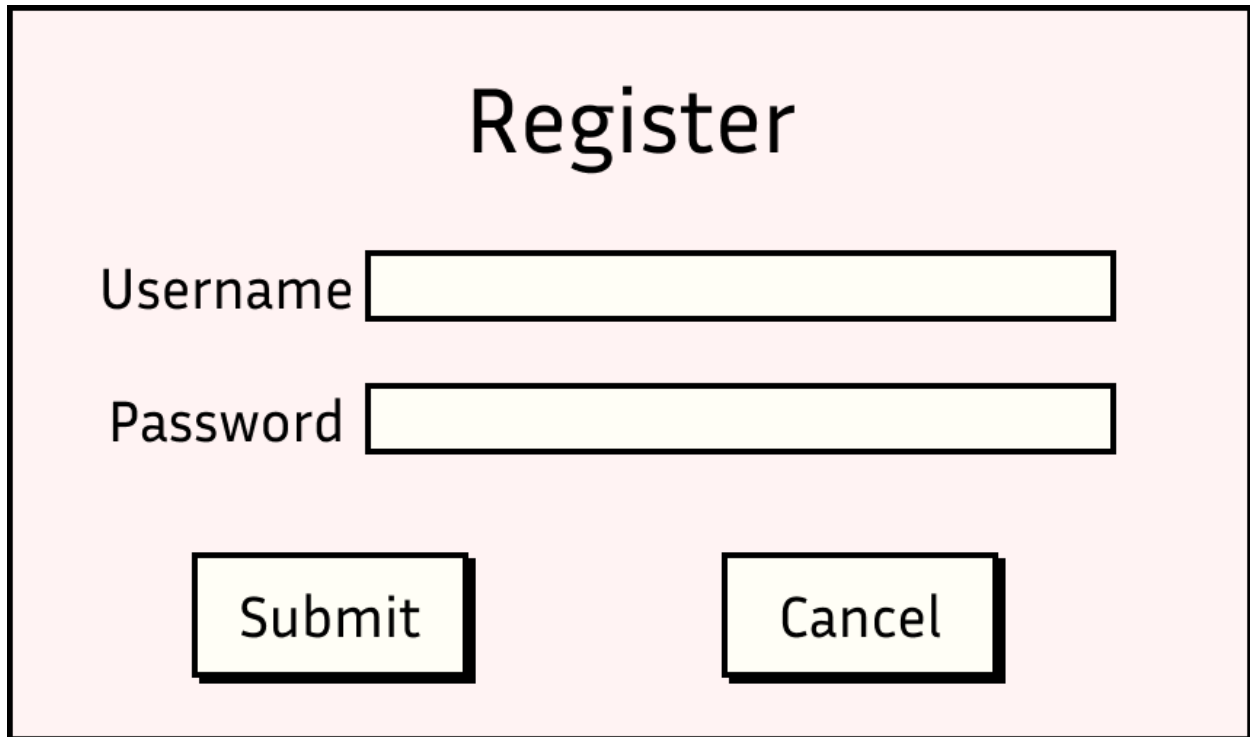
Username

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 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. 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.

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.

## 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
<b>ID</b>	string
<b>Name</b>	string
<b>Email</b>	string
<b>Phone number</b>	string
<b>Address</b>	string
<b>Date of birth</b>	string

TABLE 1 VARIABLES OF PERSONMODEL

Method	Description
<b>Constructor</b>	To create a new person with all the variables included within the model
<b>Empty constructor</b>	To create a new person with no information and then use setters to set each information
<b>Getters</b>	Gets the information of the instance and returns it
<b>Setters</b>	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
<b>mathGrade</b>	int
<b>englishGrade</b>	int
<b>gpa</b>	double
<b>classID</b>	String

TABLE 3 VARIABLES OF STUDENTMODEL

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

**TABLE 4 METHODS OF STUDENT MODEL**

### 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
<b>Constructor</b>	To create a new person with all the variables included within the model
<b>Empty constructor</b>	To create a new person with no information and then use setters to set each information
<b>Getters</b>	Gets the information of the instance and returns it
<b>Setters</b>	Sets the information of the instance
<b>getUserInfo</b>	Print the user information out in JSON format
<b>getUserInfo2</b>	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. WRITE TO FILE

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. POPULATE TABLE

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.





```

                                JOptionPane.showMessageDialog(th
is, "Please enter valid grades", "Error",
                                JOptionPane.ERROR
R_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;
}

```

## 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

<p>Edit a student with wrong phone number format</p>	<p>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</p>	<p>A failure message display, user is directed to student index window, no data is added to table or student.csv</p>	<p>As expected</p>	<p>Pass</p>
<p>Edit a student with wrong date format</p>	<p>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</p>	<p>A failure message display, user is directed to student index window, no data is added to table or student.csv</p>	<p>As expected</p>	<p>Pass</p>
<p>Edit a student with wrong grade format</p>	<p>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</p>	<p>A failure message display, no data is added to table or student.csv</p>	<p>As expected</p>	<p>Pass</p>

		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.

```

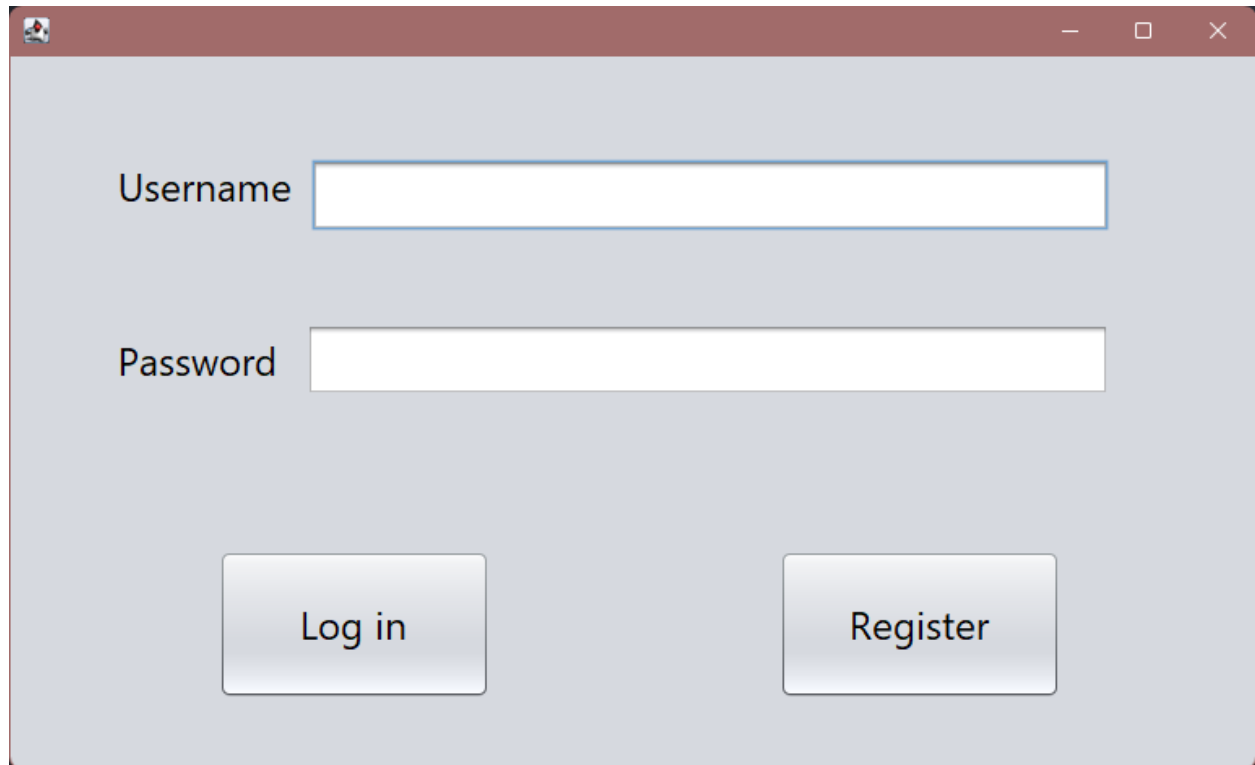
@Test
    public void testSortStudentByGpaDescending() {
        // sort the student list by the order of gpa
descending
        // check the gpa of student from first to last,
if it's in
        // descending order, then this test is passed
        System.out.println("sortStudentByGpaDescending")
;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

## VI. RESULT



The image shows a graphical user interface for a login system. It features a light gray background with a dark red title bar at the top. The title bar contains a small icon on the left and standard window control buttons (minimize, maximize, close) on the right. The main area of the window contains two text input fields. The first field is labeled "Username" and the second is labeled "Password". Below these fields are two buttons: "Log in" on the left and "Register" on the right. Both buttons have a light gray gradient and a thin black border.

FIGURE 8 LOGIN WINDOW

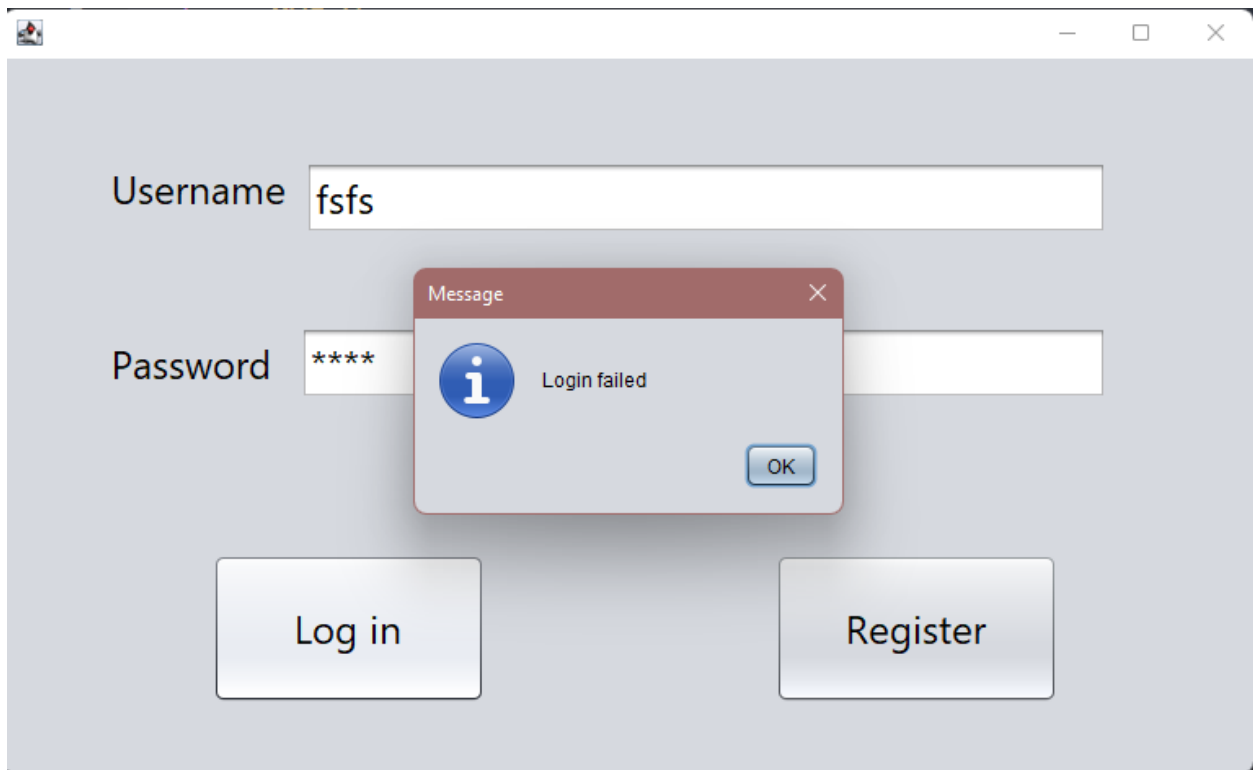


FIGURE 9 LOGIN FAILED

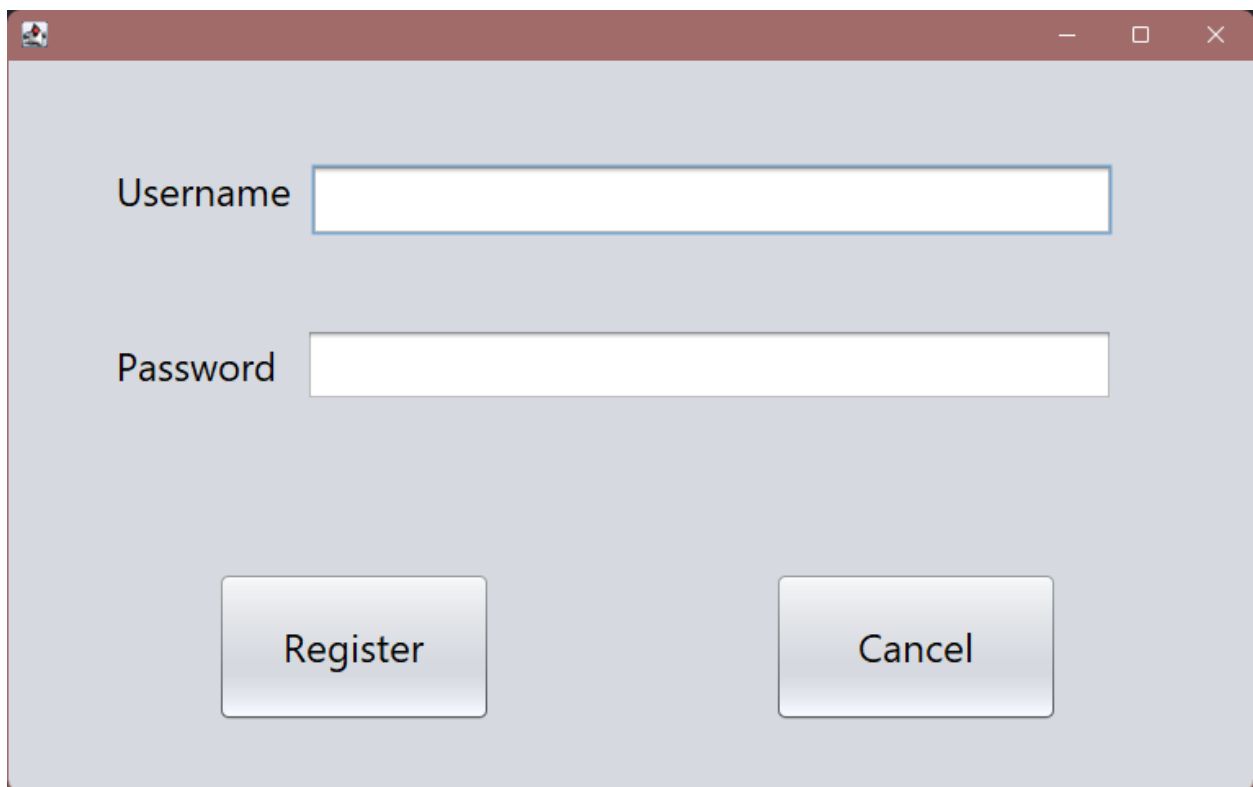


FIGURE 10 REGISTER WINDOW

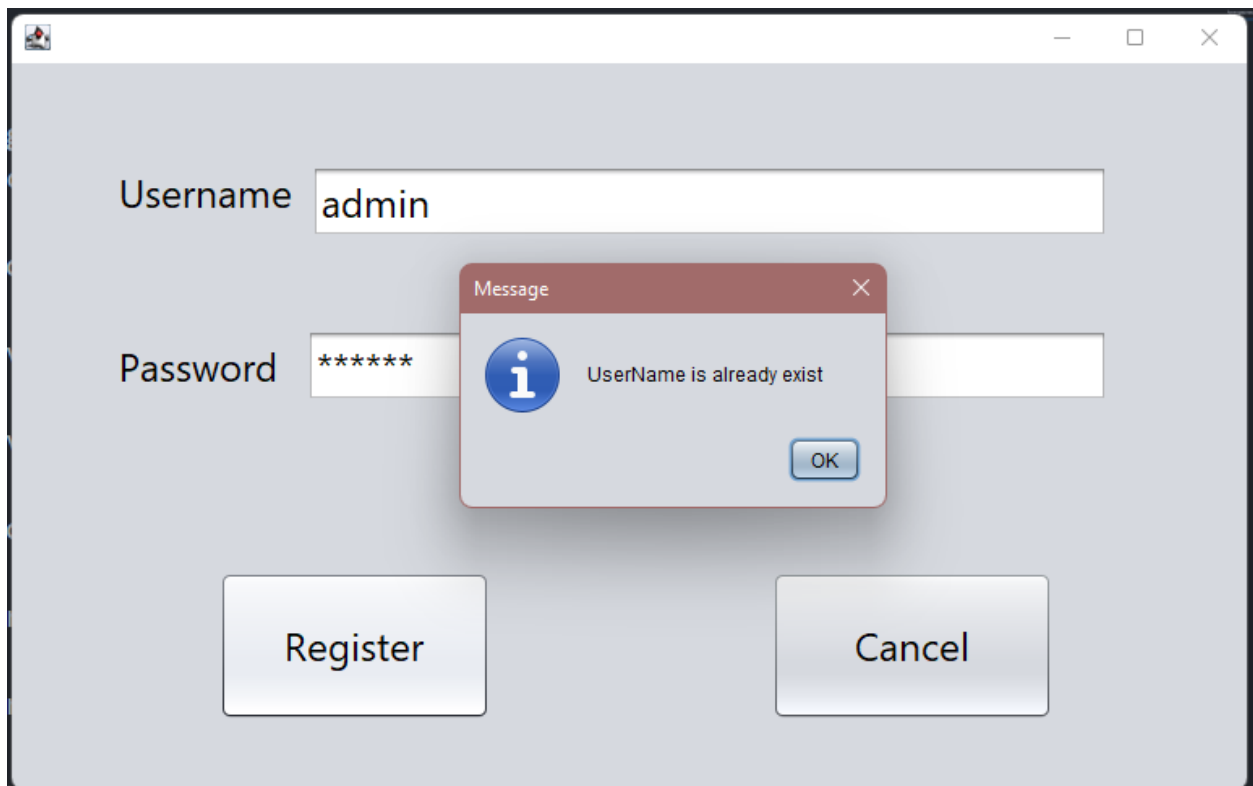


FIGURE 11 REGISTER FAILED

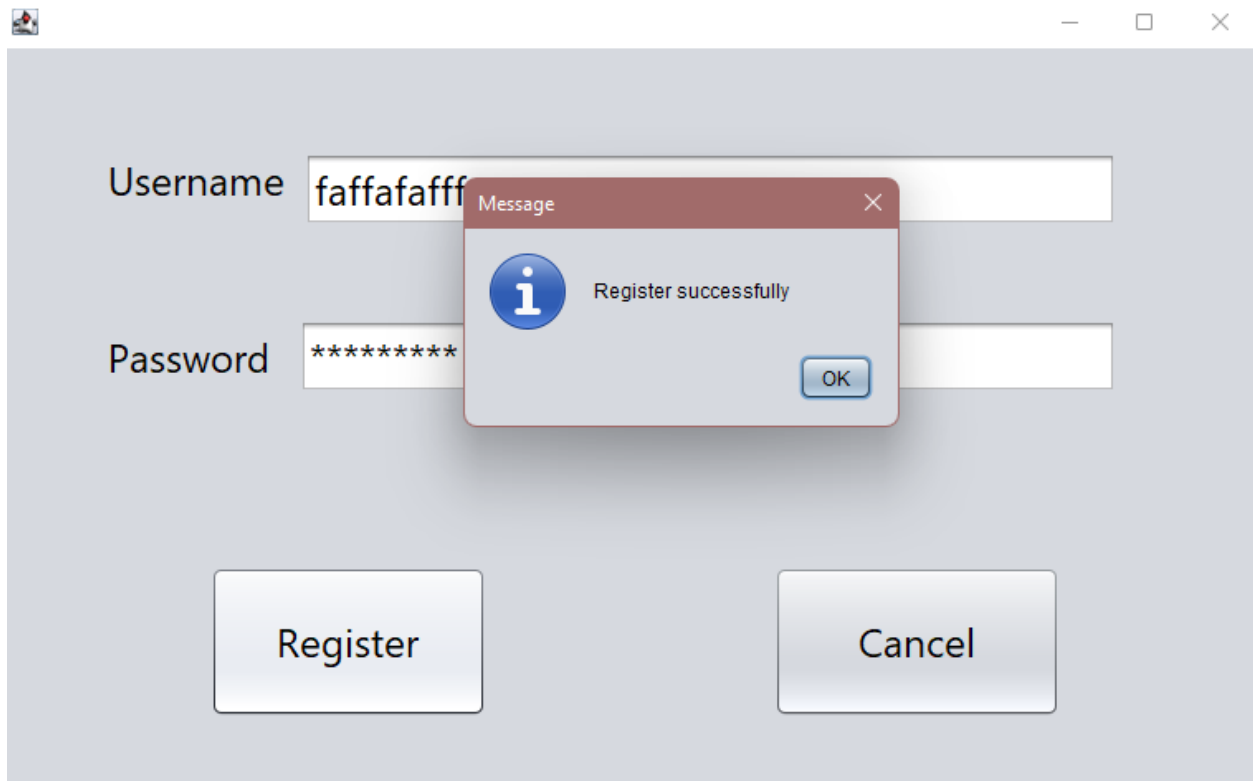


FIGURE 12 SUCCESSFUL REGISTER

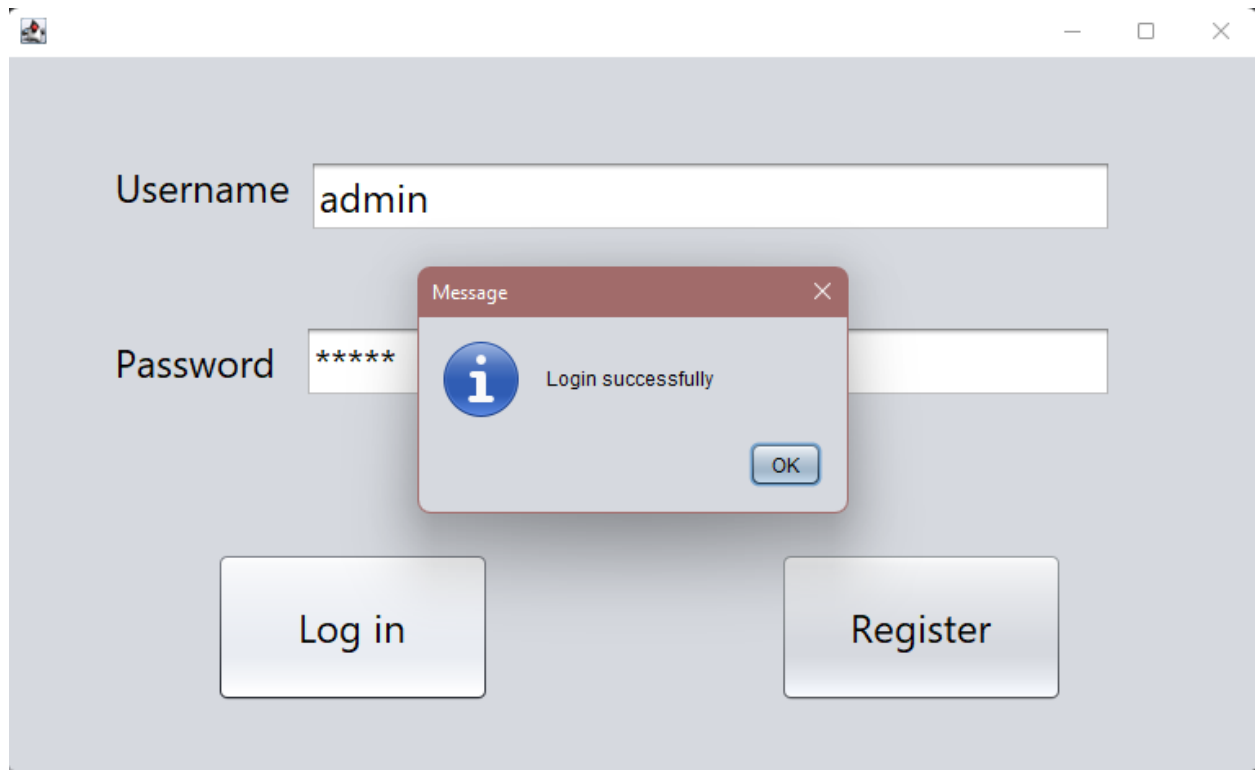
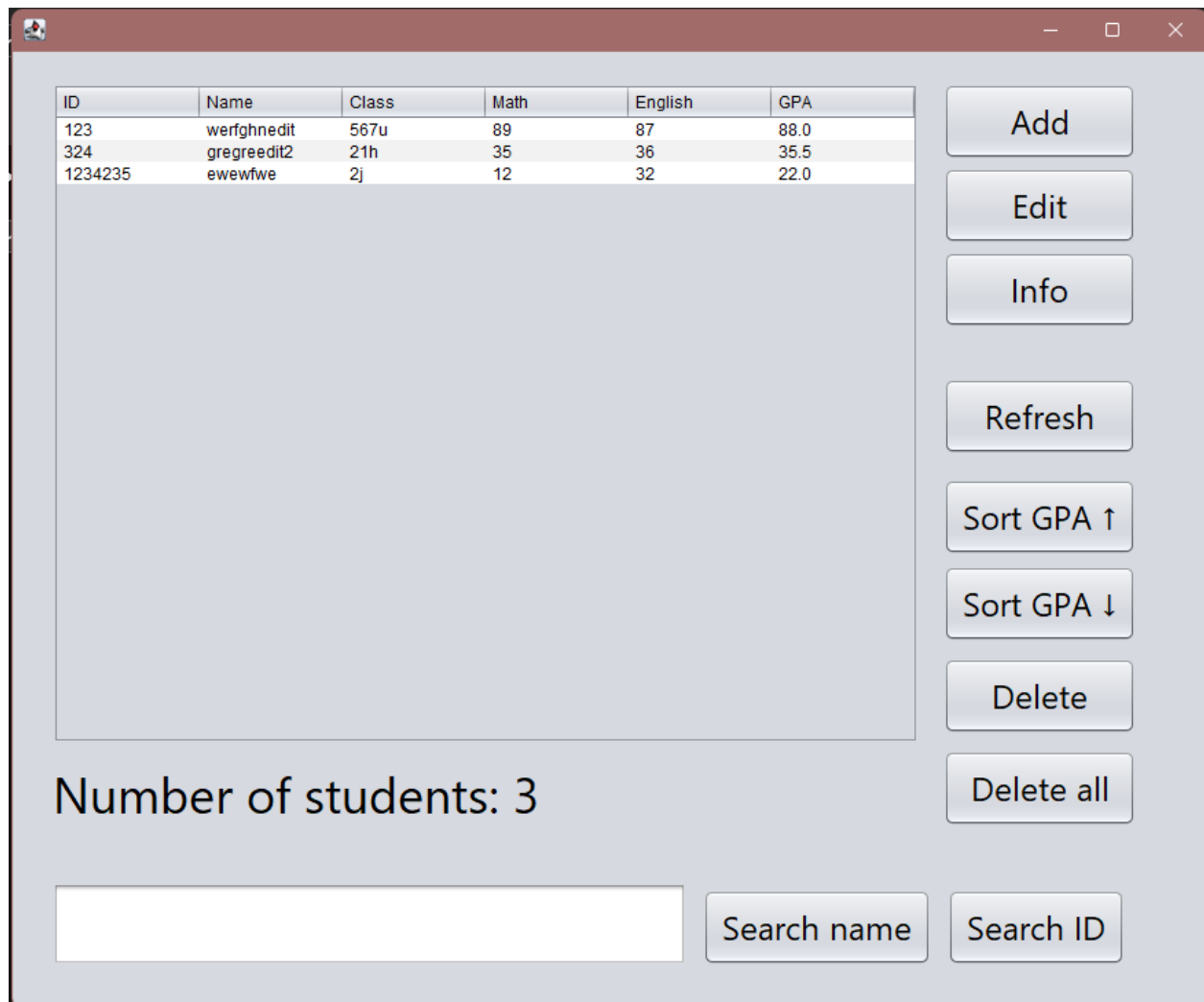


FIGURE 13 SUCCESSFUL LOGIN



A screenshot of a web application window titled "Student Index Window". The window has a light gray background and a dark red title bar with standard window controls (minimize, maximize, close). The main content area is divided into two sections. The top section contains a table with student data. The table has six columns: ID, Name, Class, Math, English, and GPA. It lists three students: 123 (werfghnedit, 567u, 89, 87, 88.0), 324 (gregreedit2, 21h, 35, 36, 35.5), and 1234235 (ewewfwe, 2j, 12, 32, 22.0). Below the table is a large, empty rectangular area. The bottom section of the window displays the text "Number of students: 3" and a search input field. To the right of the table and search area is a vertical column of buttons: "Add", "Edit", "Info", "Refresh", "Sort GPA ↑", "Sort GPA ↓", "Delete", and "Delete all". The search input field is located at the bottom left, and the "Search name" and "Search ID" buttons are at the bottom right.

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

Number of students: 3

Search name Search ID

Add Edit Info Refresh Sort GPA ↑ Sort GPA ↓ Delete Delete all

FIGURE 14 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 15 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 16 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 17 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 18 SORT BY GPA DESCENDING

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

Close

FIGURE 19 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 20 EDIT STUDENT INFORMATION

ADD A NEW STUDENT

ID

Name

Date of birth

Phone number

Email address

Address

Class

English

Math

FIGURE 21 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 22 ADD STUDENT WITH INVALID PHONE NUMBER

The image shows a web application window for adding a student. The form contains several input fields with the following data: ID (54353), Name (ffqwfw), Date of birth (12/12/2012), Phone number (1234567890), Email address (aefgmailcom), Address (empty), Class (empty), English (12), and Math (32). At the bottom are 'Submit' and 'Cancel' buttons. A modal error message box is displayed over the form, titled 'Message', with an information icon and the text 'Invalid email address'. The message box has an 'OK' button.

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

Buttons: Submit, Cancel

Message: Invalid email address (OK)

FIGURE 23 ADD STUDENT WITH INVALID EMAIL FORMAT

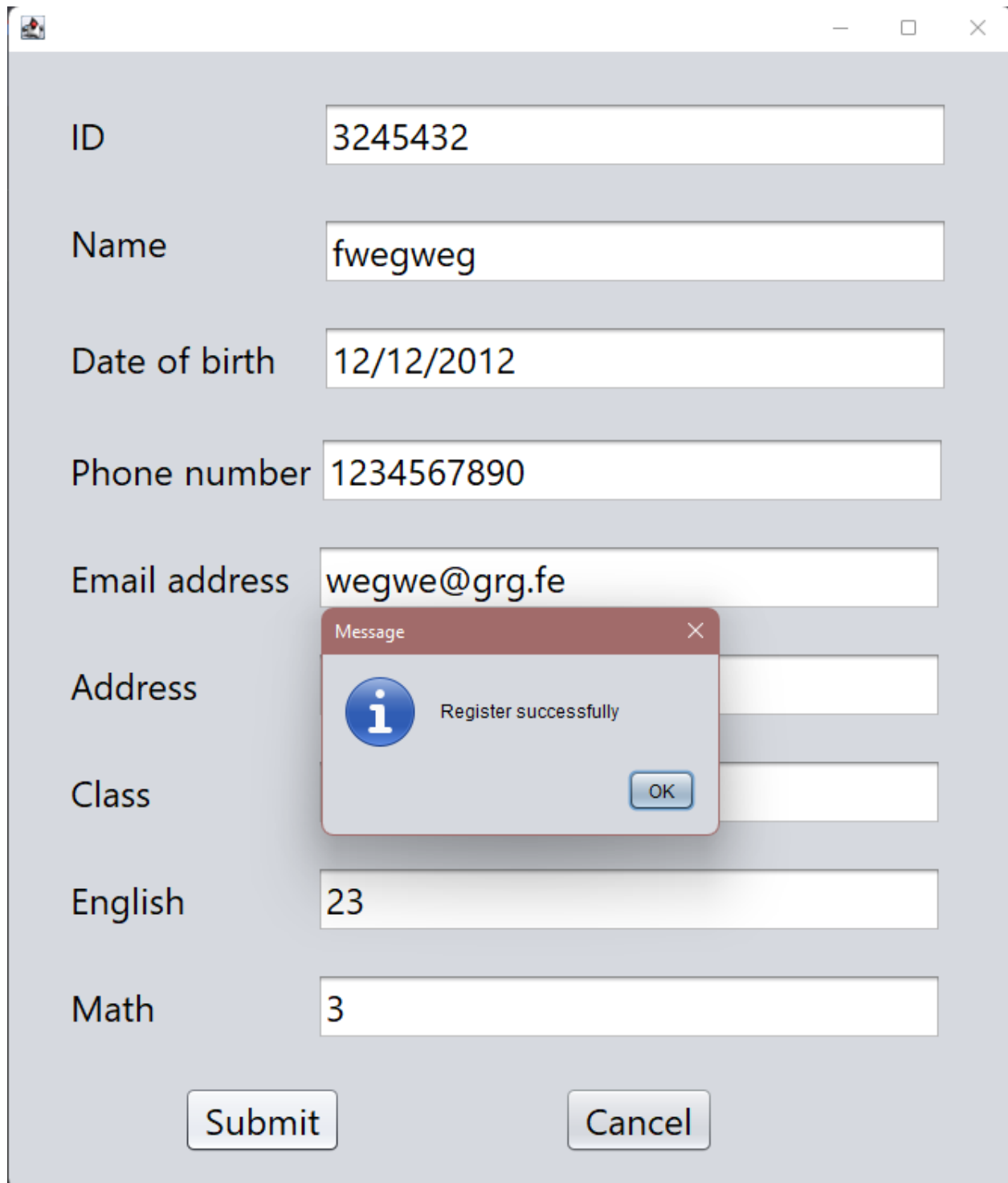
Form titled "Add Student" with fields for:

- 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

Error dialog box: Error. Please enter valid grades. OK

FIGURE 24 ADD STUDENT WITH INVALID GRADE



A screenshot of a web application window titled "Add Student" (indicated by a small icon in the top-left corner). The window has a light gray background and standard window controls (minimize, maximize, close) in the top-right corner. The form contains several input fields with labels to their left:

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

At the bottom of the form are two buttons: "Submit" and "Cancel". A modal message box is overlaid on the form, titled "Message" with a close button (X) in the top-right corner. The message box contains a blue information icon (i) and the text "Register successfully". An "OK" button is located at the bottom right of the message box.

FIGURE 25 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

Number of students: 4

Search name

Search ID

Add

Edit

Info

Refresh

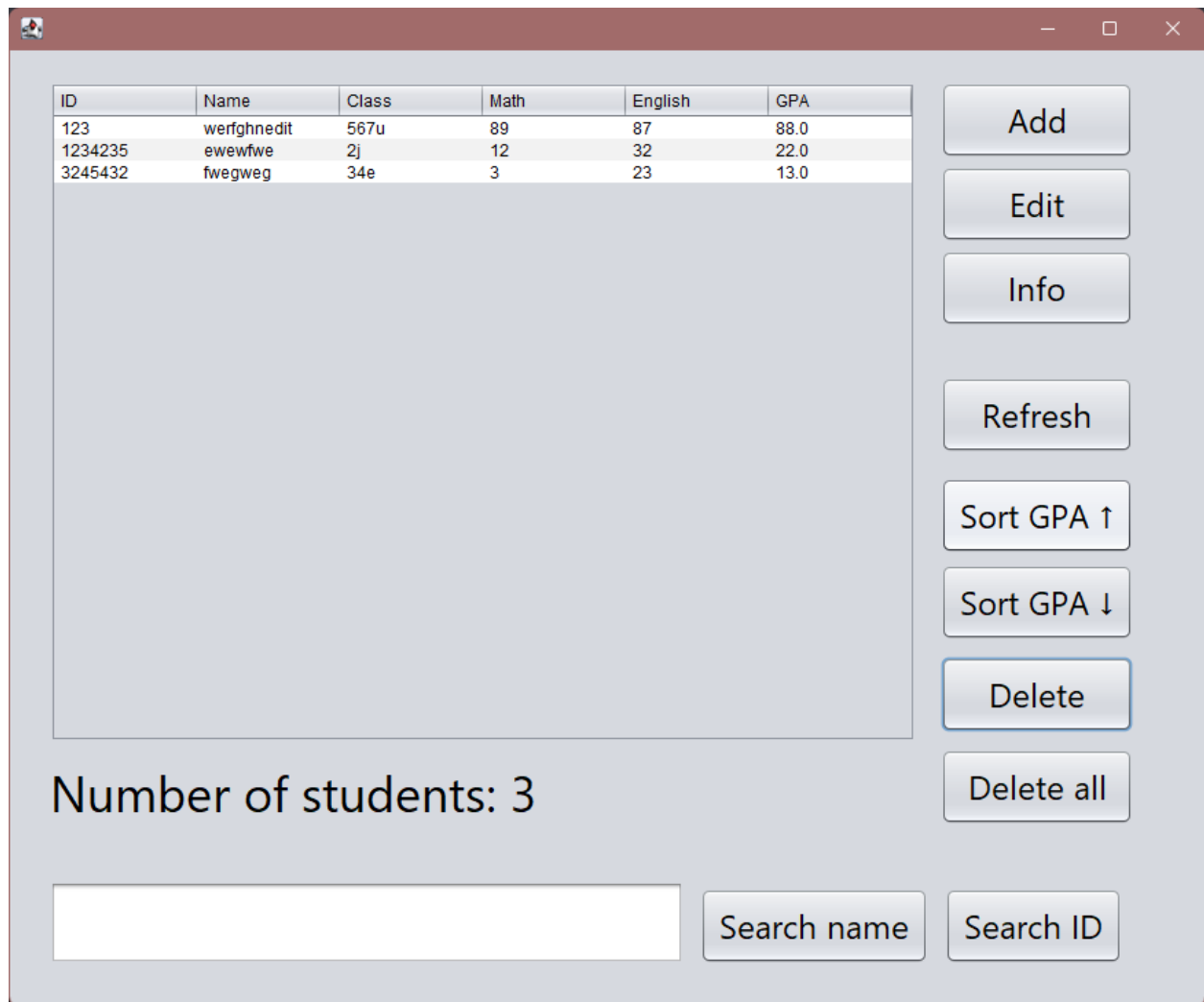
Sort GPA ↑

Sort GPA ↓

Delete

Delete all

FIGURE 26 STUDENT LIST AFTER ADDING



The screenshot shows a software application window with a title bar containing standard window controls (minimize, maximize, close). The main area features a table with student data. To the right of the table is a vertical stack of buttons: 'Add', 'Edit', 'Info', 'Refresh', 'Sort GPA ↑', 'Sort GPA ↓', 'Delete' (highlighted with a blue border), and 'Delete all'. Below the table, the text 'Number of students: 3' is displayed. At the bottom, there is a search input field and two buttons labeled 'Search name' and 'Search ID'.

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

Number of students: 3

Search input field:

Buttons: Add, Edit, Info, Refresh, Sort GPA ↑, Sort GPA ↓, Delete, Delete all, Search name, Search ID

FIGURE 27 DELETE SELECTED STUDENT

Figure 28 is a screenshot of a web application interface for managing students. The interface is contained within a window with a standard title bar (minimize, maximize, close buttons). The main area features a table with the following columns: ID, Name, Class, Math, English, and GPA. Below the table, the text "Number of students: 0" is displayed. To the right of the table, there is a vertical stack of buttons: "Add", "Edit", "Info", "Refresh", "Sort GPA ↑", "Sort GPA ↓", "Delete", and "Delete all". The "Delete all" button is highlighted with a blue border. At the bottom of the interface, there is a search bar (input field) and two buttons: "Search name" and "Search ID".

FIGURE 28 DELETE ALL STUDENTS

## VII. CONCLUSION

## REFERENCES

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

Available at: [https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)

[Accessed 24 June 2022].