Professor Portal

Algarve University

Faculty of Sciences and Technology

In Partial Fulfillment

of the Requirements for the subject of

Web Application Development during my Erasmus grant

By:

Yousef Hisham Shaban

Presented To:

Noélia Susana Costa Correia

Faro, Portugal

Fall 202

Table of Contents

CHAPTER 1: Introduction

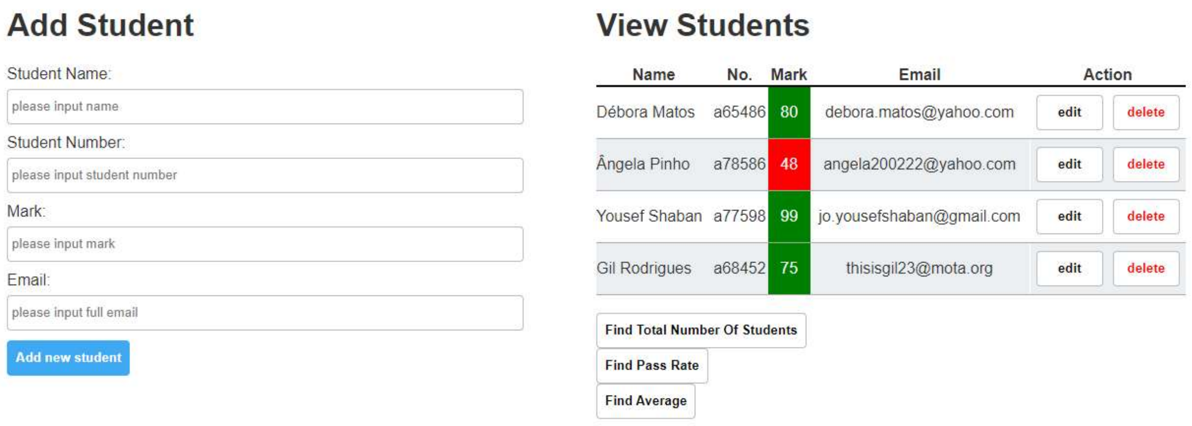
1.1 Overview

University cores are without a doubt a significant component of modern societies and education; consequently, there must be systems that oversee such units, such as the subjects registration system, in which teachers input the students' information in general into the main system. Because the method may be frustrating and unfair to certain students, an alternative to Excel must be found with more communication with the student.

1.2 Project Motivation

Most professors in multiple universities have the issue in which they cannot access certain functionalities such as “calculating the average, viewing passed/failed students, calculating the total number of students, rating passed students, sending emails regarding that, etc." unless they submit their grades on the portal, which causes another problem in that they cannot edit their grades without approval from certain sides, and this is where the idea of PPO(Professor Portal) comes from.

CHAPTER 2: Project Layout



Add Student

Student Name:

please input name

Student Number:

please input student number

Mark:

please input mark

Email:

please input full email

Add new student

View Students

Name	No.	Mark	Email	Action	
Débora Matos	a65486	80	debora.matos@yahoo.com	edit	delete
Ângela Pinho	a78586	48	angela200222@yahoo.com	edit	delete
Yousef Shaban	a77598	99	jo.yousefshaban@gmail.com	edit	delete
Gil Rodrigues	a68452	75	thisisgil23@mota.org	edit	delete

Find Total Number Of Students

Find Pass Rate

Find Average

Chapter 2 figure shows the main project look, which contains the following components:

2.1 Students Table

This table will contain all students stored in the database, also it will contain an action for each record of any student.
Under the table, there will be multiple buttons to get certain statistics based on the above records.
Be aware that each student mark background will be changing based on failing or passing the subject.

2.2 Add Student Form

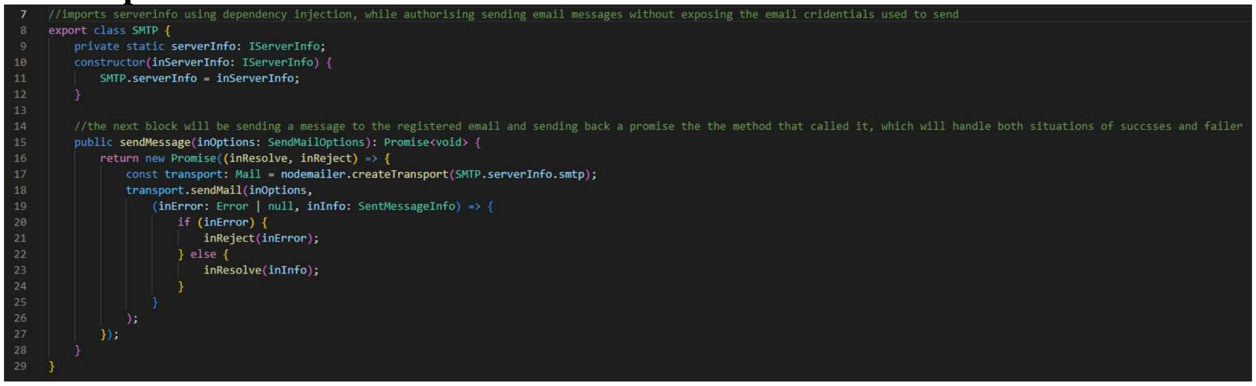
This form will be the client-side method to add any students into the students table (on the right) and to the database.
Please notice that each student that is being added will receive an email regarding that.

2.3 Edit Student Form

Same as adding a student, this form will be the client-side method to edit any students in the students table (on the right) and to the database.
Please notice that each student that who got edited will receive an email regarding that.

CHAPTER 3: Functionalities and Requirements

3.1 Simple Mail Transfer Protocol

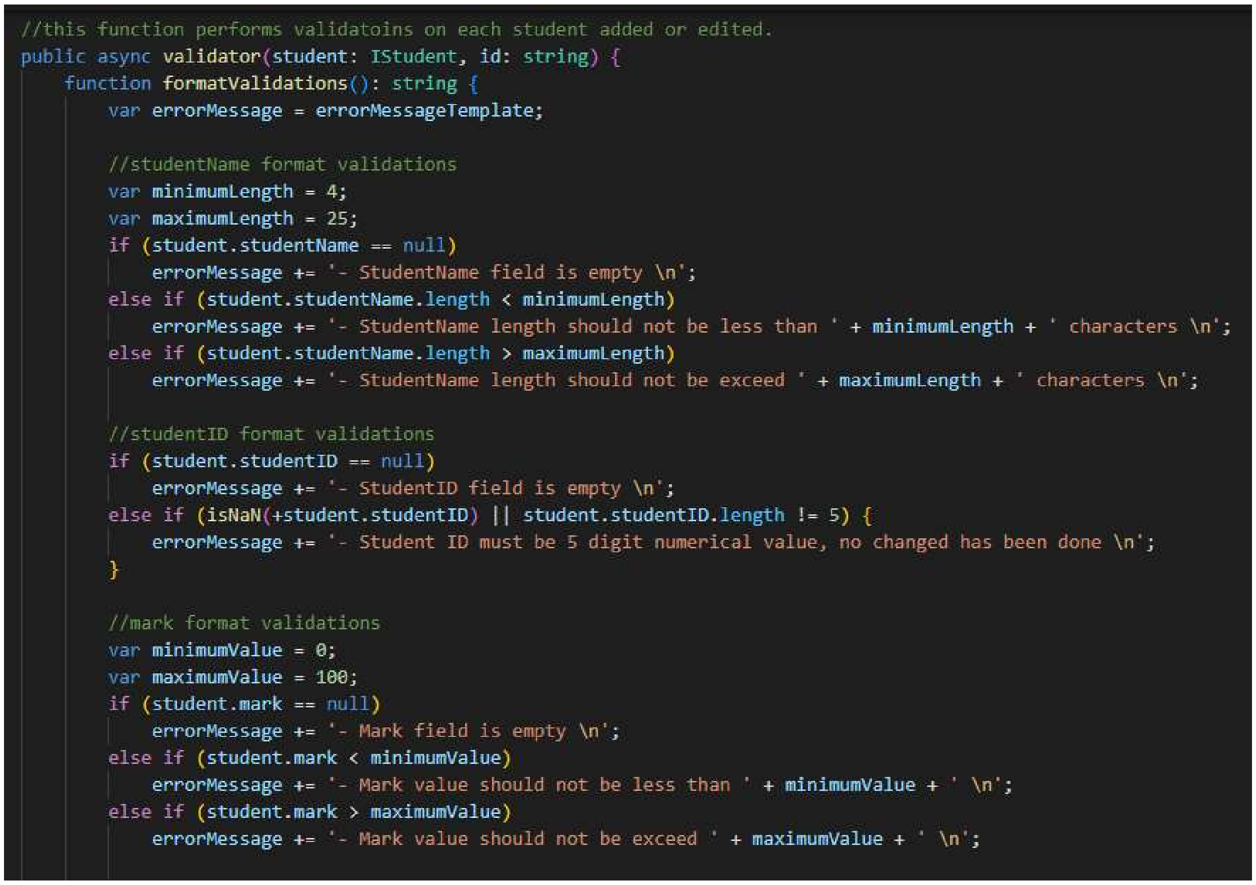


```
//imports serverinfo using dependency injection, while authorising sending email messages without exposing the email cridentials used to send
export class SMTP {
    private static serverInfo: IServerInfo;
    constructor(inServerInfo: IServerInfo) {
        SMTP.serverInfo = inServerInfo;
    }

    //the next block will be sending a message to the registered email and sending back a promise the the method that called it, which will handle both situations of succsses and failer
    public sendMessage(inOptions: SendMailOptions): Promise<void> {
        return new Promise((inResolve, inReject) => {
            const transport: Mail = nodemailer.createTransport(SMTP.serverInfo.smtp);
            transport.sendMail(inOptions,
                (inError: Error | null, inInfo: SentMessageInfo) => {
                    if (inError) {
                        inReject(inError);
                    } else {
                        inResolve(inInfo);
                    }
                }
            );
        });
    }
}
```

Figure 3.1 shows the code block which is responsible to be sending a promise regarding if the student successfully received an email or not, knowing that this feature will me essential for adding students or editing them.

3.2 Server Side



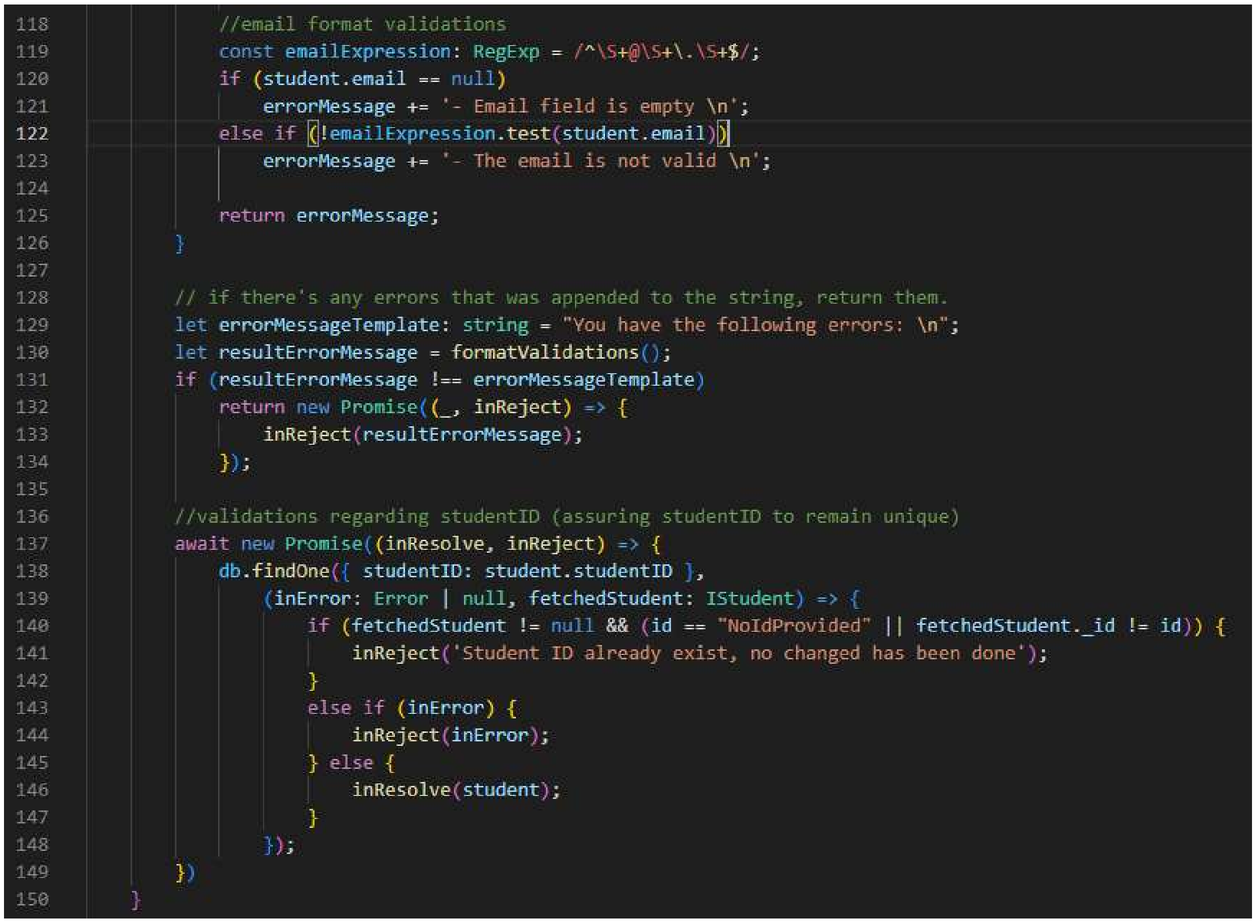
```
//this function performs validatoins on each student added or edited.
public async validator(student: IStudent, id: string) {
    function formatValidations(): string {
        var errorMessage = errorMessageTemplate;

        //studentName format validations
        var minimumLength = 4;
        var maximumLength = 25;
        if (student.studentName == null)
            errorMessage += '- StudentName field is empty \n';
        else if (student.studentName.length < minimumLength)
            errorMessage += '- StudentName length should not be less than ' + minimumLength + ' characters \n';
        else if (student.studentName.length > maximumLength)
            errorMessage += '- StudentName length should not be exceed ' + maximumLength + ' characters \n';

        //studentID format validations
        if (student.studentID == null)
            errorMessage += '- StudentID field is empty \n';
        else if (isNaN(+student.studentID) || student.studentID.length != 5) {
            errorMessage += '- Student ID must be 5 digit numerical value, no changed has been done \n';
        }

        //mark format validations
        var minimumValue = 0;
        var maximumValue = 100;
        if (student.mark == null)
            errorMessage += '- Mark field is empty \n';
        else if (student.mark < minimumValue)
            errorMessage += '- Mark value should not be less than ' + minimumValue + ' \n';
        else if (student.mark > maximumValue)
            errorMessage += '- Mark value should not be exceed ' + maximumValue + ' \n';
```

Figure 3.2.1 and Figure 3.2.2 will be showing the server-side validations for any input accruing, which will be very similar to client-side validations, which each will include all case scenarios.



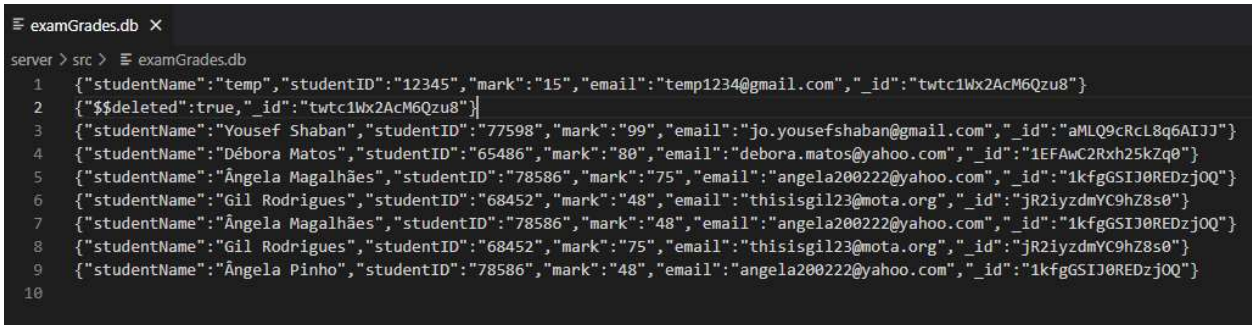
```
        //email format validations
        const emailExpression: RegExp = /^\S+@\S+\.\S+$/;
        if (student.email == null)
            errorMessage += '- Email field is empty \n';
        else if (!emailExpression.test(student.email))
            errorMessage += '- The email is not valid \n';

        return errorMessage;
    }

    // if there's any errors that was appended to the string, return them.
    let errorMessageTemplate: string = "You have the following errors: \n";
    let resultErrorMessage = formatValidations();
    if (resultErrorMessage !== errorMessageTemplate)
        return new Promise((_, inReject) => {
            inReject(resultErrorMessage);
        });

    //validations regarding studentID (assuring studentID to remain unique)
    await new Promise((inResolve, inReject) => {
        db.findOne({ studentID: student.studentID },
            (inError: Error | null, fetchedStudent: IStudent) => {
                if (fetchedStudent != null && (id == "NoIdProvided" || fetchedStudent._id != id)) {
                    inReject('Student ID already exist, no changed has been done');
                }
                else if (inError) {
                    inReject(inError);
                } else {
                    inResolve(student);
                }
            });
    })
}
```

3.3 Local Database



```
{"studentName":"temp","studentID":"12345","mark":"15","email":"temp1234@gmail.com","_id":"twtc1Wx2AcM6Qzu8"}
{"$$deleted":true,"_id":"twtc1Wx2AcM6Qzu8"}
{"studentName":"Yousef Shaban","studentID":"77598","mark":"99","email":"jo.yousefshaban@gmail.com","_id":"aMLQ9cRcL8q6AIJJ"}
{"studentName":"Débora Matos","studentID":"65486","mark":"80","email":"debora.matos@yahoo.com","_id":"1EFAwC2Rxh25kZq0"}
{"studentName":"Ângela Magalhães","studentID":"78586","mark":"75","email":"angela200222@yahoo.com","_id":"1kfgGSIJ0REDzjOQ"}
{"studentName":"Gil Rodrigues","studentID":"68452","mark":"48","email":"thisisgil23@mota.org","_id":"jR2iyzdmYC9hZ8s0"}
{"studentName":"Ângela Magalhães","studentID":"78586","mark":"48","email":"angela200222@yahoo.com","_id":"1kfgGSIJ0REDzjOQ"}
{"studentName":"Gil Rodrigues","studentID":"68452","mark":"75","email":"thisisgil23@mota.org","_id":"jR2iyzdmYC9hZ8s0"}
{"studentName":"Ângela Pinho","studentID":"78586","mark":"48","email":"angela200222@yahoo.com","_id":"1kfgGSIJ0REDzjOQ"}
```

Figure 3.3 shows the local database which contains the record of the students that was displayed in chapter2.

CHAPTER 4: Project Setup

4.1 installing the development environment

- First, please make sure of which Operating system you are using currently

- After so, go ahead and press here to go ahead and download visual studio code.

- Finally, make sure to download Node.js.

4.2 Getting familiar with directories

- Open Visual Studio Code, and Open Folder from File menu.

- Open the folder of the project from the destination you extracted it.

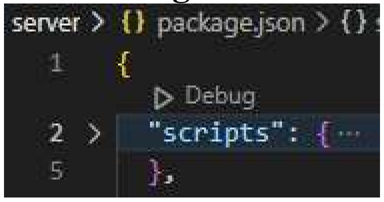
- scroll through and start memorizing the files inside.

4.3 Downloading libraries

- Make sure to check each file required libraries by simply checking errors they give in the Problems sections down the screen, and after so go ahead and google they proper syntax for the installation in the terminal, some of the most important imports will be:

path, nedb, nodemailer, fs, express, react, axios, etc.

4.4 Starting the environment



```
{
  "scripts": {⋯
  },
```

- Go ahead and check for the “package.json” located in both client and server folders, you will see a grey “Debug” button above the scripts, please go ahead and debug the server then the client side.

4.5 Enjoy the experience!

Congratulations, you just launched the portal and now it’s ready to be used!

4.6 Common Errors

-Make sure to always start the debug of the serve-side before the client-side.

-Make sure that the ports used for server and client sides is not reserved by other processes.