**People's Democratic Republic of Algeria**
**Ministry of Higher Education and Scientific Research**

**University M'Hamed BOUGARA – Boumerdès**


**Institute of Electrical and Electronic Engineering**
**Department of Electronics**

Project Report Presented in Partial Fulfilment of

the Requirements of the Degree of

# 'MASTER'

In **Electronics**

Option: **Computer Engineering**

Title:

# Design and Implementation of web-based Management System. Case of the "IGEE" Registrar's Office.

Presented By:

- **MESLEM Toufik Abdelrezak.**
- **ZERMOUT Abdelaziz.**

Supervisor:

**Dr NAMANE R.**

Registration Number:...../2021

# Abstract

This project proposes a technical solution to automate the management of the registrar's office "*service de scolarité*" of our institute. The objective of this project is, therefore, to build a web based application for the automation of the registrar's office work mainly the final deliberation. The application should facilitate the work of both the office stuff and the faculty members. Faculty members "Teachers" submit their grades of their concerned students for every course assigned to them through the web application. Thereafter, the application calculates students' averages, and generates the final deliberation transcripts for all promotions.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

We would like to thank our supervisor Dr. NAMANE Rachid who gave us the opportunity to work on this wonderful project. We would like to thank our colleagues who helped us in this project SAIDI Omar and BENBOUZID Sohaib.

# Dedication

I dedicate my work to my dear parents, my sister, all my family and friends, and to every Muslim in the world.

Toufik Abdelrezak

I dedicate my work to my dear parents, cousins, all my family and friends.

Abdelaziz

# General Introduction

Nowadays, the internet has become a part of our life. The internet is supposed to be a tool to facilitate our daily tasks in regard to the duties we have to do. Because of that, we try in this project to provide a problem solution using the internet by creating a web application that can be accessed from different users from anywhere at any time they need.

The registrar's office is a vital component in every educational institution as it is the responsible of delivering most of the official documents. However, with the great number of students increasing every year, the work of the office becomes overloaded and the probability for making errors increases too. To facilitate the work of the office and to automate the correct generation of the different documents and transcripts, we propose to design and implement a web based application for the management of the registrar's office. In our work we focus more on generating the final deliberation transcripts for all promotions of the institute which considered as mandatory step that should be done at the end of every academic year.

To make things more clearly for the reader, the present report is divided into four chapters: **The first chapter** is an introduction to web application and gives an overview of our project objectives. **The second chapter** briefly introduces the tools and technologies we used through the project. **The third chapter** is about the design of the application both in UML modelling language and the database organization. **The fourth and last chapter** concerns the implementation of the application by representing the user interfaces. The report is finally ended with a general conclusion and some suggestions for future works.

# CHAPTER ONE Project Overview and Web Applications

## 1.1 Introduction

This chapter is about the project problem identification. Then, we discuss the desired objectives and the way to follow for a better solution of the given problem. Finally we give an introduction to web apps and some web related concepts as the topic of the project is about the design and the implementation of a web application.

## 1.2 Subject presentation

Our objective is to design and implement a web application to manage our institute registrar's office. This theme is proposed and chosen to solve the problems faced by the stuff of that office in generating some official documents and transcripts.

### 1.2.1 Problem statement

In order to build our project, we have discussed with the registrar’s office stuff and we came up with some of the problems they are facing among which:

- Lateness of deliberation of students’ grades.
- Lot of tedious work because of copy paste work.
- The difficulty of correcting errors.

### 1.2.2 Objectives

Our objective is to design and implement a web application to solve the problems mentioned above by:

- Automating the grades submission through our web application.
- Exporting the grades in pdf file format, then posting them.

## 1.3 Web application

### 1.3.1 Definition

A web application or "web app" is a software program that runs on a web server. Unlike traditional desktop applications, which are launched by your operating system, web apps must be accessed through a web browser. Web apps have several advantages over desktop applications. Since they run inside web browsers, developers do not need to develop web apps for multiple platforms. For example, a single application that runs in Chrome will work on both Windows and OS X. Developers do not need to distribute software updates to users when the web app is updated. By updating the application on the server, all users have access to the updated version [1].

### 1.3.2 Single-Page Application:

A single-page application is an app that works inside a browser and does not require page reloading during use. You are using this type of applications every day. These are, for instance: Gmail, Google Maps, Facebook or GitHub.

SPAs are all about serving an outstanding UX by trying to imitate a “natural” environment in the browser — no page reloads, no extra wait time. It is just one web page that you visit which then loads all other content using JavaScript — which they heavily depend on.

SPA requests the markup and data independently and renders pages straight in the browser. We can do this thanks to the advanced JavaScript frameworks like AngularJS, Ember.js, Meteor.js, Knockout.js .

Single-page sites help keep the user in one, comfortable web space where content is presented to the user in a simple, easy and workable fashion. [2]

### 1.3.3 Multi-Page Application:

Multiple-page applications work in a “traditional” way. Every change, for instance, display the data or submit data back to server requests rendering a new page from the server in the browser. These applications are large, bigger than SPAs because they need to be. Due to the amount of content, these applications have many levels of UI (User Interface). Luckily, it is not a problem anymore. Thanks to AJAX, we don’t have to worry that big and complex applications have to transfer a lot of data between server and browser. That solution improves and it allows to refresh only particular parts of the application. On the other hand, it adds more complexity and it is more difficult to develop than a single-page application. [2]

### 1.3.4 Web client and Web server:

#### 1.3.4.1 Web client

A Web client typically refers to the Web browser in the user's machine or mobile device. It may also refer to extensions and helper applications that enhance the browser to support special services from the site. [3]

#### 1.3.4.1 Web server

The term web server can refer to hardware or software, or both of them working together.

On the hardware side, a web server is a computer that stores web server software and a website's component files. (For example, HTML documents, images, CSS stylesheets, and JavaScript files) A web server connects to the Internet and supports physical data interchange with other devices connected to the web.

On the software side, a web server includes several parts that control how web users access hosted files. At a minimum, this is an HTTP server. An HTTP server is software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view

webpages). An HTTP server can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device. [4]

## 1.4 HTTP

HTTP is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more. [5]

Communication between clients and servers is done by requests and responses:

1. A client (a browser) sends an HTTP request to the web.
2. A web server receives the request.
3. The server runs an application to process the request.
4. The server returns an HTTP response (output) to the browser.
5. The client (the browser) receives the response. [6]

## 1.5 URL

URL stands for Uniform Resource Locator. A URL is nothing more than the address of a given unique resource on the Web. In theory, each valid URL points to a unique resource. Such resources can be an HTML page, a CSS document, an image, etc. In practice, there are some exceptions, the most common being a URL pointing to a resource that no longer exists or that has moved. As the resource represented by the URL and the URL itself are handled by the Web server, it is up to the owner of the web server to carefully manage that resource and its associated URL. [7]

A URL is composed of different parts, some mandatory and others optional. The most important parts are highlighted on the URL parts on Figure 1.1

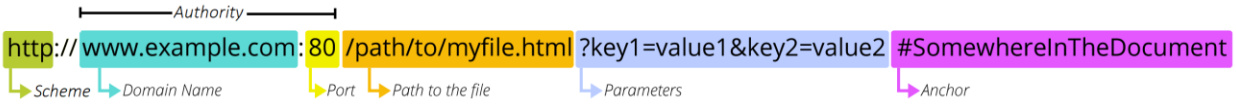


**Figure 1.1 URL parts [7]**

### 1.5.1 Scheme

The first part of the URL is the scheme, which indicates the protocol that the browser must use to request the resource (a protocol is a set method for exchanging or transferring data around a computer network). [7]

### 1.5.2 Authority

Next follows the authority, which is separated from the scheme by the character pattern ://. If present the authority includes both the domain (e.g. www.example.com) and the port (80), separated by a colon:

- The domain indicates which Web server is being requested. Usually this is a domain name, but an IP address may also be used (but this is rare as it is much less convenient).

- The port indicates the technical "gate" used to access the resources on the web server. It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources. Otherwise it is mandatory. [7]

### 1.5.3 Path to resource

/path/to/myfile.html is the path to the resource on the Web server. In the early days of the Web, a path like this represented a physical file location on the Web server. Nowadays, it is mostly an abstraction handled by Web servers without any physical reality. [7]

### 1.5.4 Parameters

?key1=value1&key2=value2 are extra parameters provided to the Web server. Those parameters are a list of key/value pairs separated with the & symbol. The Web server can use

those parameters to do extra stuff before returning the resource. Each Web server has its own rules regarding parameters, and the only reliable way to know if a specific Web server is handling parameters is by asking the Web server owner. [7]

### 1.5.5 Anchor

#SomewhereInTheDocument is an anchor to another part of the resource itself. An anchor represents a sort of "bookmark" inside the resource, giving the browser the directions to show the content located at that "bookmarked" spot. On an HTML document, for example, the browser will scroll to the point where the anchor is defined; on a video or audio document, the browser will try to go to the time the anchor represents. It is worth noting that the part after the **#**, also known as the **fragment identifier**, is never sent to the server with the request. [7]

## 1.6 Conclusion:

Through this chapter, we specified the problems that the registrar's office stuff are facing and we proposed to help solving them using a web application. Then we introduced a summary of a web application and some concepts of the web.

# CHAPTER TWO Tools and Technologies

## 2.1 Introduction

Nowadays, the tools and technologies take different forms. In this chapter, we will introduce the different tools and web technologies we used to implement our application.

## 2.2 Development tools

### 2.2.1 Visual studio code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity). [8]

### 2.2.2 ReactJS

React.js is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications. It’s used for handling the view layer for web and mobile apps. React also allows us to create reusable UI components.

React allows developers to create large web applications that can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple. [9]

### 2.2.3 NodeJS

Node.js is a server-side platform wrapped around the JavaScript language for building scalable, event-driven applications. This is confusing for even experienced programmers because the traditional JavaScript environment has always been client-side - in a user's browser or in an application that is talking to a server. JavaScript has not been considered when it comes to the server responding to client requests, but that is exactly what Node.js provides.

Node.js is not written in JavaScript (it is written in C++) but it uses the JavaScript language as an interpretive language for server-side request/response processing. In other words, Node.js runs stand-alone JavaScript programs. The advantage is that programmers can use their current, albeit client-side, programming knowledge and begin coding with Node.js much more easily. [10]

### 2.2.4 ExpressJS

Express.js is a Node js web application server framework, which is specifically designed for building single-page, multi-page, and hybrid web applications. [11]

### 2.2.5 MongoDB

MongoDB is a powerful, flexible, and scalable general-purpose database management system (DBMS) that uses a document-oriented database model which supports various forms of data. A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Instead of using tables and rows as in relational databases, the MongoDB architecture is made up of collections and documents. [12]

### 2.2.6 Mongoose

Mongoose is a third-party package that builds upon the official MongoDB driver, but it makes accessing MongoDB much easier and more convenient. Mongoose use schemas that MongoDB does not really use, so we can define how our data should look like and that allows us to conveniently store and fetch data. [13]

### 2.2.7 MongoDB compass

It is a GUI (Graphical User Interface) for querying, aggregating, and analyzing your MongoDB data in a visual environment. Compass is free to use and source available, and can be run on macOS, Windows, and Linux. [14]

### 2.2.8 Web browser

A web browser is a software program that allows a user to locate, access, and display web pages. Web browsers are used primarily for displaying and accessing websites on the internet, as well as other content created using languages such as Hypertext Markup Language (HTML). Browsers translate web pages and websites delivered using Hypertext Transfer Protocol (HTTP) into human-readable content. [15]

### 2.2.9 Lucid chart

Lucidchart is a web-based proprietary platform that allows users to collaborate on drawing, revising and sharing charts and diagrams. It is produced by Lucid Software Inc., based in Utah, United States. Lucidchart runs on browsers that support HTML5. This means it does not require updates of a third-party software like Adobe Flash. [16]

## 2.3 Programming languages

### 2.3.1 HTML

Hypertext markup language (HTML) is the major markup language used to display Web pages on the Internet. In other words, Web pages are composed of HTML, which is used to display text, images or other resources through a Web browser.

All HTML is plain text, meaning it is not compiled and may be read by humans. The file extension for an HTML file is .htm or .html. [17]

### 2.3.2 CSS

Cascading Style Sheets (CSS) is a standard (or language) that describes the formatting of markup language pages. CSS enables developers to separate content and visual elements for greater page control and flexibility. A CSS file is normally attached to an HTML file by means of a link in the HTML file. [18]

### 2.3.3 JavaScript

JavaScript (JS) is a scripting languages, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code. JavaScript is an interpreted language. Thus, it doesn't need to be compiled. JavaScript renders web pages in an interactive and dynamic fashion. This allowing the pages to react to events, exhibit special effects, accept variable text, validate data, create cookies, detect a user’s browser, etc. [19]

## 2.4 Security

### 2.4.1 Bcrypt

Bcrypt is a password-hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher and presented at USENIX in 1999. Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power. [20]

### 2.4.2 JSON Web Token

JSON web token (JWT), pronounced "jot", is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. Again, JWT is a standard, meaning that all JWTs are tokens, but not all tokens are JWTs.

Because of its relatively small size, a JWT can be sent through a URL, through a POST parameter, or inside an HTTP header, and it is transmitted quickly. A JWT contains all the required information about an entity to avoid querying a database more than once. The recipient of a JWT also does not need to call a server to validate the token.

JWTs can be used in various ways:

Authentication: When a user successfully logs in using their credentials, an ID token is returned. According to the OpenID Connect (OIDC) specs, an ID token is always a JWT.

Authorization: Once a user is successfully logged in, an application may request to access routes, services, or resources (e.g., APIs) on behalf of that user. To do so, in every request, it must pass an Access Token, which may be in the form of a JWT. Single Sign-on (SSO) widely uses JWT because of the small overhead of the format, and its ability to easily be used across different domains.

Information Exchange: JWTs are a good way of securely transmitting information between parties because they can be signed, which means you can be sure that the senders are who they say they are. Additionally, the structure of a JWT allows you to verify that the content hasn't been tampered with. [21]

## 2.5 Conclusion

Through this chapter, brief definitions have been presented to clarify the different tools and technologies we used to implement our web application.

# CHAPTER THREE Design

## 3.1 Introduction

In this chapter, we are going to model our web application using UML, and we will describe our database.

## 3.2 Design requirements

### 3.2.1 Modeling language

Modeling language is mainly used in the field of computer science and engineering for designing models of new software, systems, devices and equipment. The context of modeling language is primarily textual and graphical, but based on the requirements and specific domain in use, modeling languages fall into the following four categories:

- System modeling language
- Object modeling languages
- Virtual reality modeling language
- Data modeling language

Unified modeling language (UML) is a popular modeling language that is used to build system and object models graphically. [22]

### 3.2.2 Unified modeling language (UML)

Unified Modeling Language (UML) is a general purpose modelling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis. [23]

## 3.3 Modelling

### 3.3.1 System actors

An actor is a person, organization, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures. [24]

In our application we have two types of actors:

#### 3.3.1.1 Administrator

The responsible of managing the user accounts, the courses, the students, and the deliberation.

#### 3.3.1.2 Teacher

The teacher can submit grades of the students he teaches them.

### 3.3.2 Use case description

In our application, we define the roles for each actor as shown in Table 3.1

**Table 3.1 Roles of actors in the use case diagram**

| Actor | Use case |
|---|---|
| **Admin** | 1. **Login:** access his account with an email and password.<br>2. **User management:** the admin can add, remove users (teachers), and assign courses for them.<br>3. **Students' management:** add/remove students, specify the level and groups of them.<br>4. **Deliberation:** can see the final deliberation |
| **Teacher** | 1. **Login:** access his account with an email and password.<br>2. **Submit grades:** submits/edit the grades of his students (control, exam, makeup), see the grades. |

### 3.3.3 Use case identification

a. **User management:** the admin can add, remove users (teachers), and edit details about them such as courses assigned to them.
b. **Students management:** the admin can add/remove list of students, specify/modify the level and groups of them.
c. **Courses management:** the admin can assign/remove courses for teachers.
d. **Deliberation:** the admin is the only one who can see the final deliberation.
e. **Login:** allow the users to access their accounts.
f. **Submit grades:** teacher can submit/edit grades of his students (control, exam, and makeup). Also he can see the grades after the submission.

### 3.3.4 Use case diagram

**Use case diagrams**are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. [25]

**An actor** portrays any entity (or entities) that perform certain roles in a given system. The different roles the actor represents are the actual business roles of users in a given system. An actor in a use case diagram interacts with a use case. [26]

**A use case** in a use case diagram is a visual representation of a distinct business functionality in a system. [26]

In a use case diagram, some types of lines are used to specify the relationship between the elements of the use case diagram. We can summarize them in Table 3.2

**Table 3.2 Summary of the used arrows [12]**

| Symbols | Description |
|---|---|
| Association<br>——————— | **Association:** Use cases are associated with the actors that perform them. A line is used to link actors to use cases |
| Include<br>---------→ | **Include**: an include relationship shows dependency between a base use case and an included use case Every time the base use case is executed the included use case is executed as well, another way to think of it is that the base use case requires an included use case in order to be complete. When we have an included use case we draw a dashed line with an arrow that points towards the included use case. |
| Extend<br>←--------- | Extend: it has also a base use case and an extend use case when the base use case is executed the extend use case will happen sometimes but not every time, the extended use case will only happen if certain criteria are met, another way to think of it is that you have the option to extend the behavior of the base use case, when we have an extended use case we draw a dashed line with an arrow that points towards the base use case. |

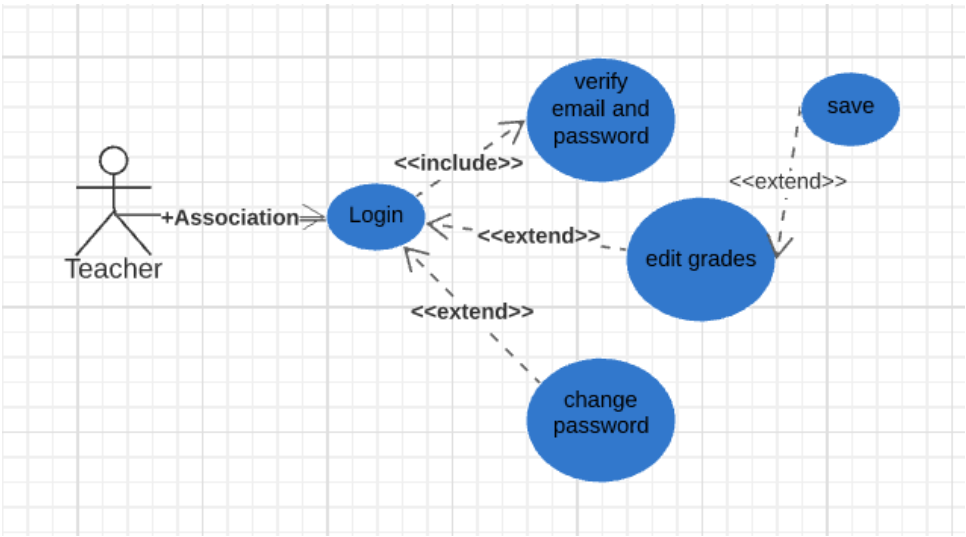


**Figure 3.2 teacher use case diagram**

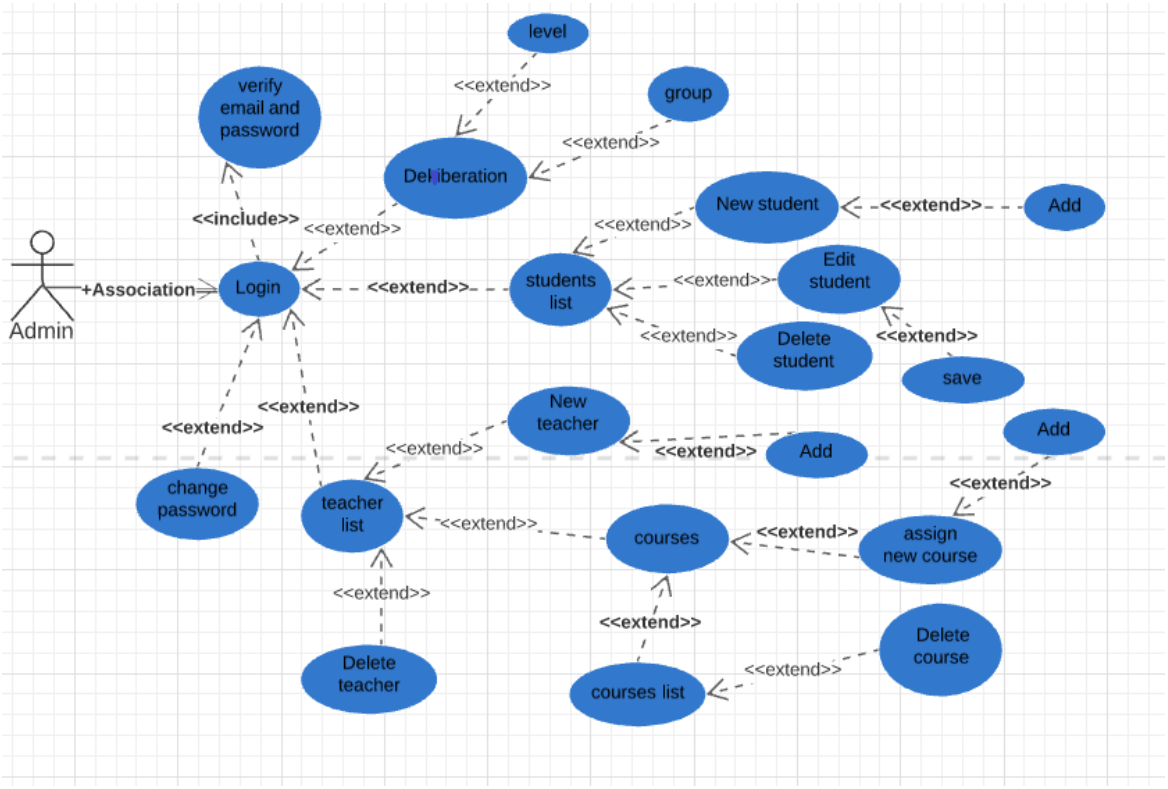


**Figure 3.3 Admin use case diagram**

## 3.4 Introduction to database

### 3.4.1 Definition

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. [27]

### 3.4.2 Relational model

A relational database is a collection of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows. Tables are used to hold information about the objects to be represented in the database. Each column in a table holds a certain kind of data and a field stores the actual value of an attribute. The rows in the table represent a collection of related values of one object or entity. Each row in a table could be marked with a unique identifier called a primary key, and rows among multiple tables can be made related using foreign keys. [28]

### 3.4.3 Non-relational model

A non-relational database is a database that does not use the tabular schema of rows and columns found in most traditional database systems. Instead, non-relational databases use a storage model that is optimized for the specific requirements of the type of data being stored.A document data store manages a set of named string fields and object data values in an entity referred to as a document. These data stores typically store data in the form of JSON documents. Each field value could be a scalar item, such as a number, or a compound element, such as a list or a parent-child collection. The data in the fields of a document can be encoded in a variety of ways, including XML, YAML, JSON, BSON, or even stored as plain text. The fields within documents are exposed to the storage management system, enabling an application to query and filter data by using the values in these fields. [29]

### 3.4.4 MySQL

MySQL is a popular, free-to-use, and open-source relational database management system (RDBMS) developed by Oracle. As with other relational systems, MySQL stores data using tables and rows, enforces referential integrity and uses structured query language (SQL) for data access. When users need to retrieve data from a MySQL database, they must construct an SQL query that joins multiple tables together to create the view on the data they require.

Database schemas and data models need to be defined ahead of time, and data must match this schema to be stored in the database. This rigid approach to storing data offers some degree of safety, but trades this for flexibility. If a new type or format of data needs to be stored in the database, schema migration must occur, which can become complex and expensive as the size of the database grows. [30]

### 3.4.5 MongoDB

MongoDB is a non-relational database developed by MongoDB, Inc. MongoDB stores data as documents in a binary representation called BSON (Binary JSON). Related information is stored together for fast query access through the MongoDB query language. Fields can vary from document to document; there is no need to declare the structure of documents to the system – documents are self-describing. If a new field needs to be added to a document, then the field can be created without affecting all other documents in the collection, without updating a central system catalog, and without taking the system offline. Optionally, schema validation can be used to enforce data governance controls over each collection. A NoSQL database like MongoDB is schema-less. Instead of storing rows in a table you create “documents” that are stored in collections. Each document can be thought of as a kind of very efficient JSON file and a collection is, well, a collection of those documents. [12]

### 3.4.6 Benefits of using documents in database

"**• Documents are natural**. Documents represent data in the same way that applications do. Unlike the tabular rows and columns of a relational database, data can be structured with arrays and subdocuments – in the same way applications represent data, as lists and members / instance variables respectively. This makes it much simpler and faster for developers to model how data in the application will map to data stored in the database.

• **Documents are flexible.** Each document can store data with different attributes from other documents. With JSON documents, we can add new attributes when we need to, without having to alter a centralized database schema.

• **Documents make applications fast.** With data for an entity stored in a single document, rather than spread across multiple relational tables, the database only needs to read and write to a single place. Having all the data for an object in one place also makes it easier for developers to understand and optimize query performance." [12]

### 3.4.7 The description of the database of our application

We used three collections in our database which are: admins, students, and teachers.

| Collection Name | Documents | Avg. Document Size | Total Document Size | Num. Indexes | Total Index Size | Properties |
|---|---|---|---|---|---|---|
| admins | 1 | 142.0 B | 142.0 B | 1 | 20.0 KB | |
| students | 36 | 2.0 KB | 73.2 KB | 1 | 36.0 KB | |
| teachers | 2 | 372.5 B | 745.0 B | 1 | 36.0 KB | |

**Figure 3.4 database collections interface**

#### 3.4.7.1 Admin collection

Documents in the admin collection contain five attributes:

- **_id:** an identification number given by default from the database.

- **fname:** first name of the admin.
- **lname:** last name of the admin.
- **email:** email of the admin account.
- **password:** hashed password of the admin account.

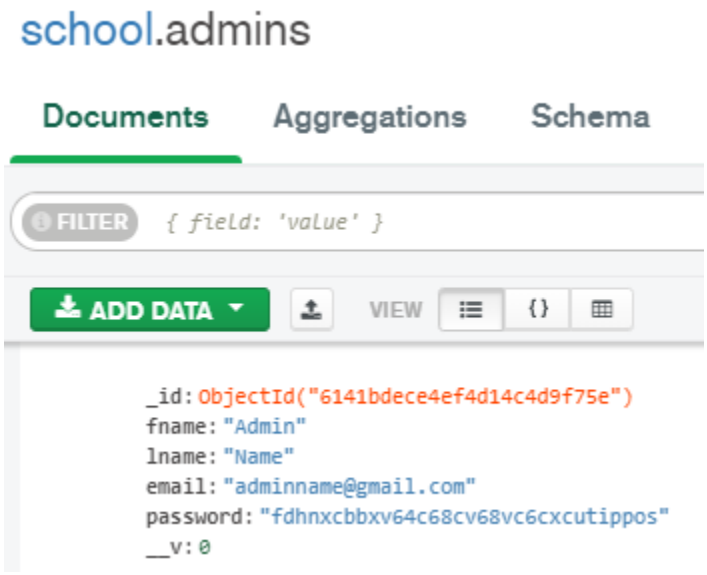


**Figure 3.5 admin collection**

### 3.4.7.2 Teacher collection

Documents in the teacher collection contain six attributes:

- **_id:** an identification number given by default from the database.
- **fname:** first name of the admin.
- **lname:** last name of the admin.
- **email:** email of the admin account.
- **password:** hashed password of the admin account.
- **courses:** array of courses that are assigned to the teacher described by
    - **corsename**: the name of the course in English (Math, Physics..)
    - **coursecode**: a unique code of the course (EE121, EE125L..)

- **level**: the level of the course.
- **section**: the students' section.
- **group**: the students' group.

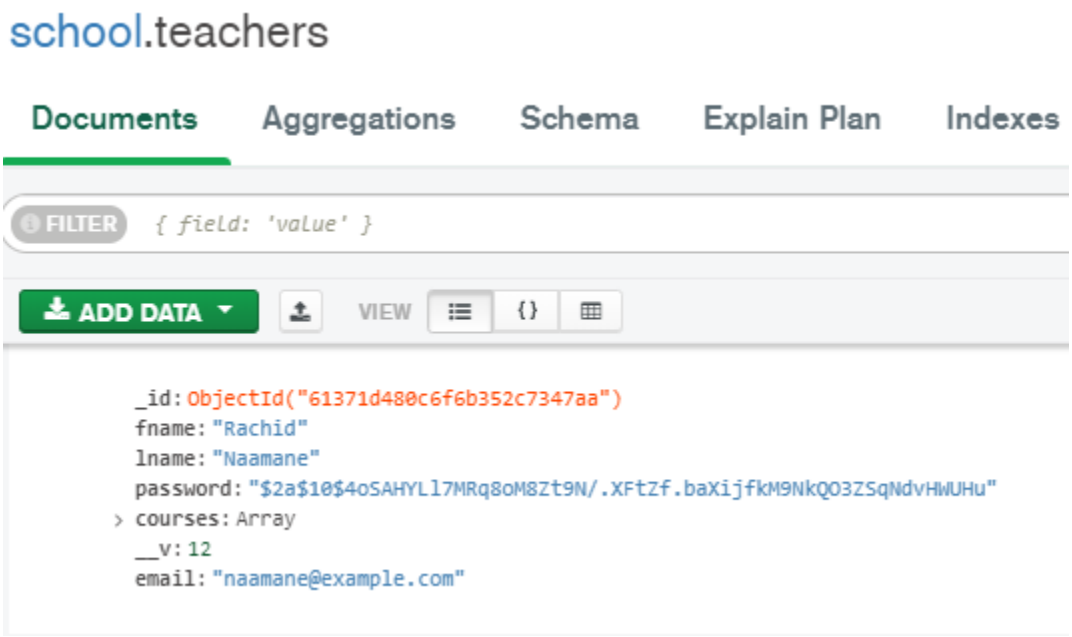


**Figure 3.6 teacher collection**

### 3.4.7.3 Student collection

Documents in the student collection contain ten attributes (Figure 3.7 (a)):

- **_id:** an identification number given by default from the database.
- **grades1, grades2, grades3:** three arrays of all the modules of first year, second year, and third year, respectively, in the license cycle. And each module is stored with its grade (key-value pair).

  Figure 3.7 (b) shows more details about these three fields. Each one of the grades array contain modules. There are two kinds of modules:

  - Ordinary module which contains: control, exam, and makeup grade.
  - Lab module which contains only one grade.
- **fname:** first name of the student.

- **lname:** last name of the student.
- **rnumber:** a register number composed of 12 digits given by the administration.
- **level:** the current level of the student.
- **section:** the section of the student.
- **group:** the group of the student.

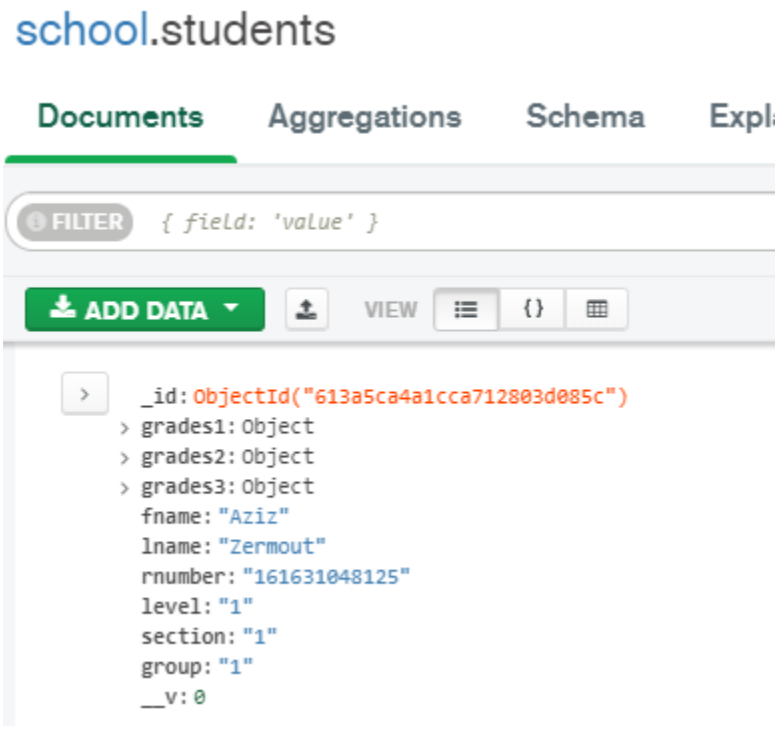



**Figure 3.7** **(a) student collection** **(b) Grades**

## 3.5 Conclusion:

In this chapter, we have used UML to model our web app. And we have presented the database of the application.

# CHAPTER FOUR Implementation

## 4.1 Introduction

In the previous chapters we introduced the different tools we used and the design of our application. In this chapter we present the resulting implemented application user interfaces with some explanations.

## 4.2 Application Interfaces

### 4.2.1 Login Interface

The login interface is the first page to be loaded when the user (admin or teacher) wants to access the application. To login, the user must enter a valid email within the appropriate password. A checkbox is used to distinguish between the admin and a normal teacher. The browser sends a query to the server, the server then checks the information in the database. If any information is wrong the logging is denied, otherwise the user logs in. Here we have two cases, the first one is when the user is the admin. In this case, the first page that is displayed for the admin is student list (Figure 4.10). The other case is when the user is a teacher. Here, the first page that is displayed for the teacher is the submit grades page (Figure 4.16 for ordinary module, or Figure 4.17 for lab module). Figure 4.8 shows the login interface.

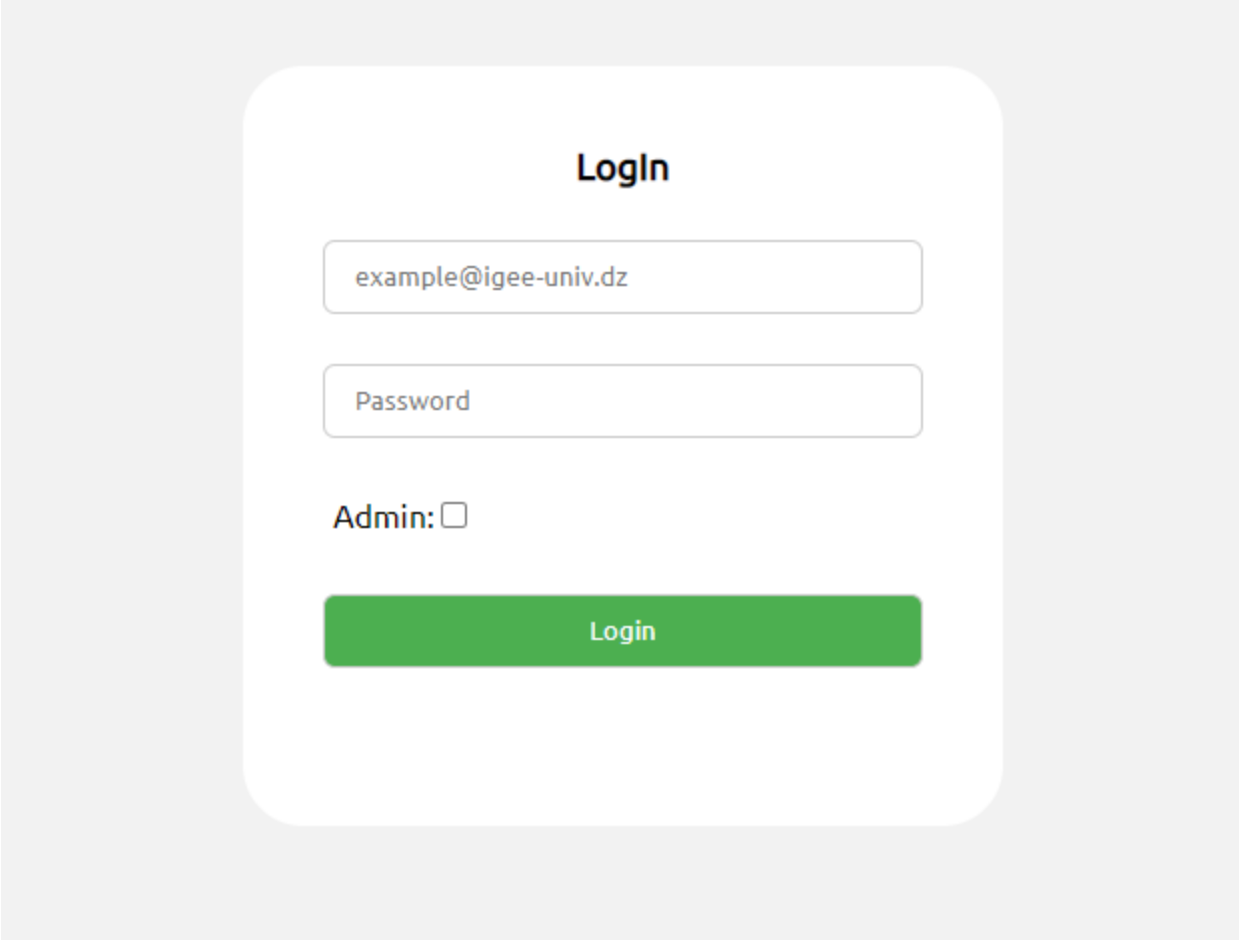


**Figure 4.8 Login interface**

### 4.2.2 Interface of "New Student" page

The admin is the responsible of adding new students to the students list by filling the form shown in Figure 4.9. The form contains 6 fields to be filled. The first two are personal information about the student (first name and last name), where the other fields are academic information. Registration Number is a unique unchangeable id given for the student by the administration in the first academic year. Level is a number from one to five. Section and group are used for organizing the students of each level.

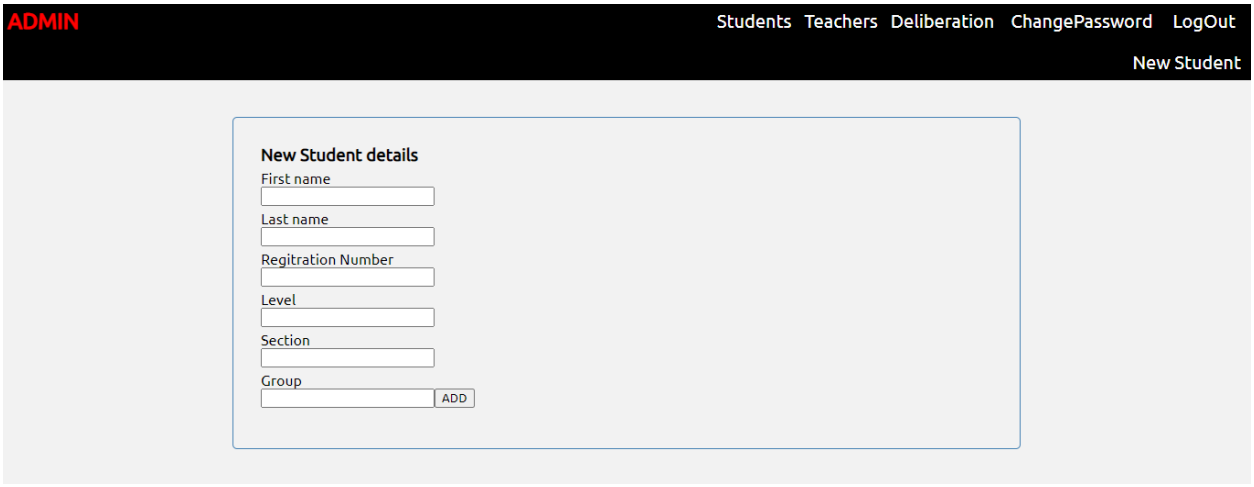


**Figure 4.9 "New student" interface**

### 4.2.3 Students list interface

Figure 4.10 shows the list of all students saved in the database. This page can be seen from the admin only. When a new student is added, his information is displayed in the list. To delete any student the admin must go for that one by one by clicking the "delete" button appeared below each student. Also to edit information of any student the admin must do the same scenario in deletion but by clicking "edit" button this time.

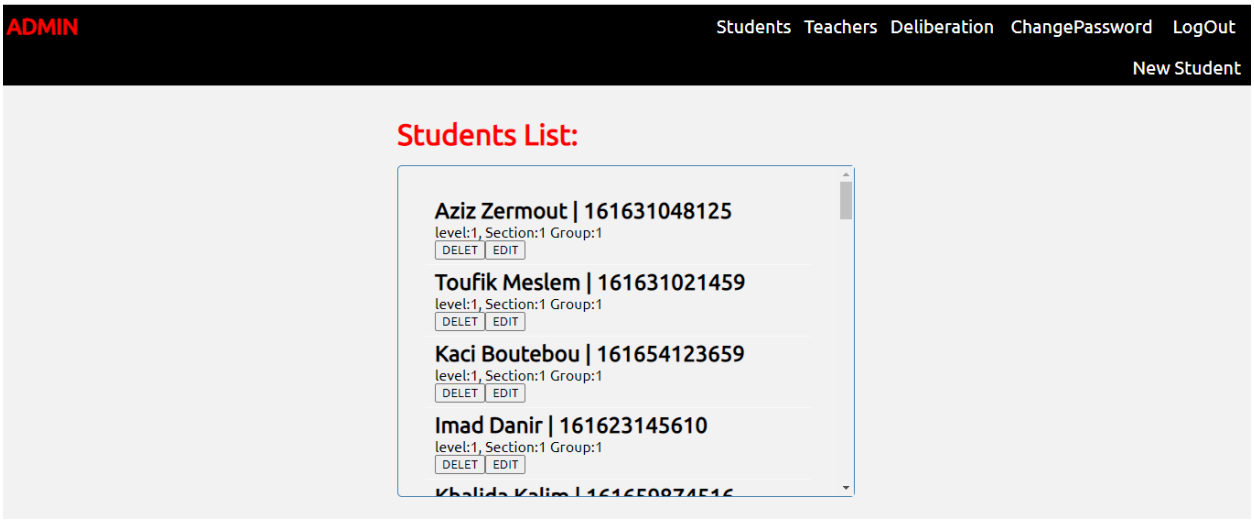


**Figure 4.10 students list interface**

### 4.2.4 Interface of “New Teacher”

The admin is the responsible of registering teachers by filling the form shown in Figure 4.11. After a successful registration, the teacher information appears in the teacher list as shown in Figure 4.10 .The admin gives a password to the teacher so he can access the application. Then the teacher can change the password as shown in Figure 4.15.

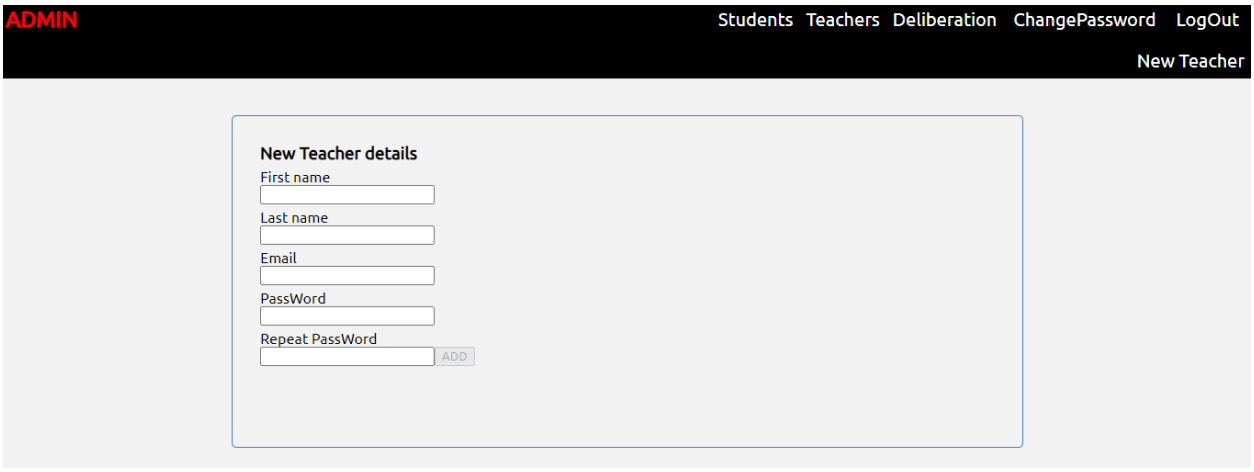


**Figure 4.11 Interface of “New Teacher”**

### 4.2.5 Teacher list interface

Figure 4.12 shows the list of all teachers. The admin can delete any teacher by clicking the button bellow the teacher name. Another button, called “courses” is used for assigning or deleting courses for teachers as shown in Figure 4.13.

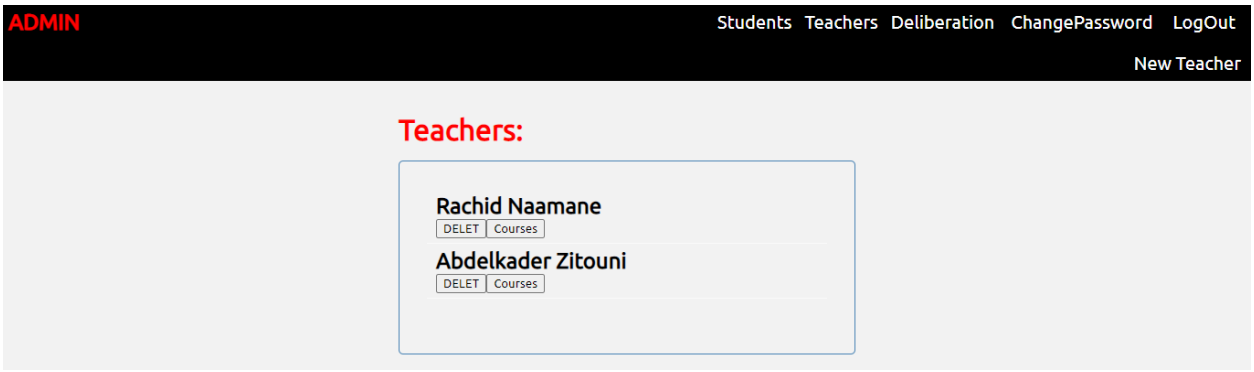


**Figure 4.12 Teacher list interface**

### 4.2.6 Courses interface

Figure 4.13 shows two boxes, the top box is a form to assign a new course to a teacher. Assign courses is done by choosing the level and the course from two drop-down lists. If the teacher instructs one section, the admin enter the number of the section. But if the teacher instructs the whole level, this field is left empty. The same thing can be said for the group field. The bottom box is the list of all courses assigned to the teacher. Under each course name a delete button can be used to delete the course from this list.

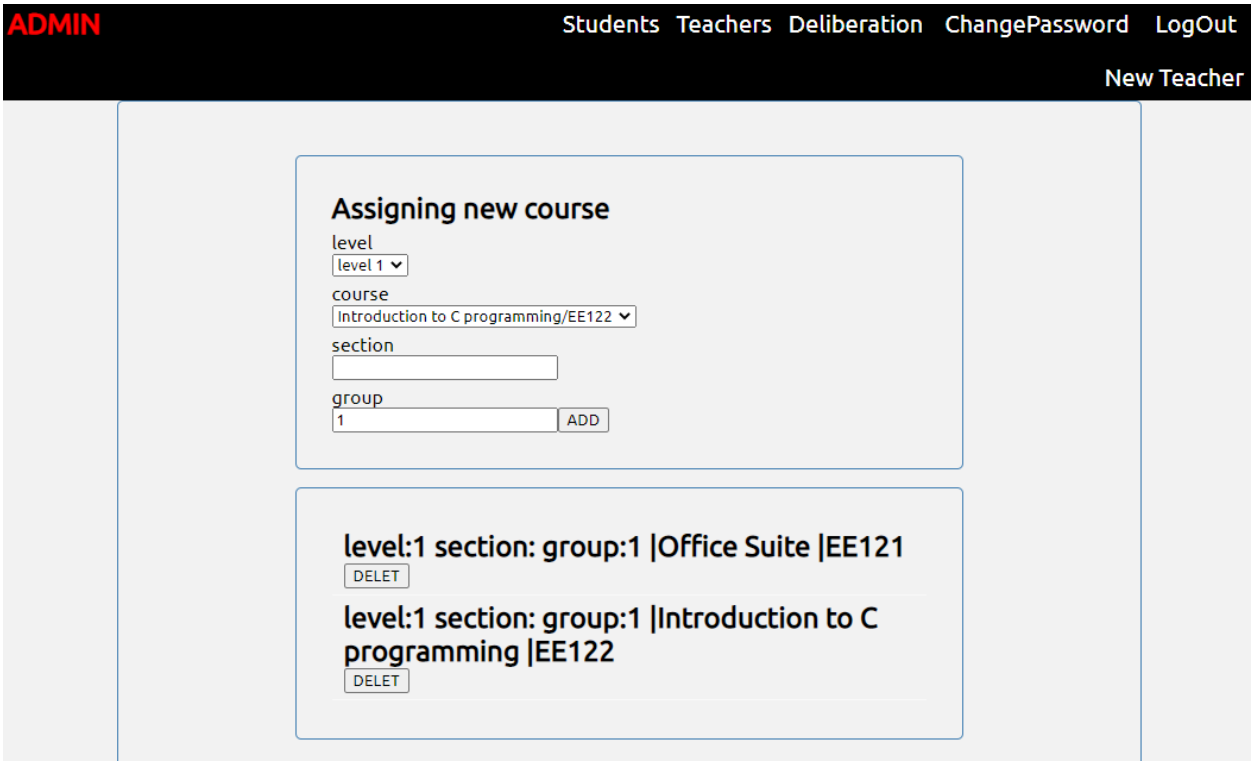


**Figure 4.13 courses interface**

### 4.2.7 Interface of change password for

Figure 4.14 and Figure 4.15 have the same fields to change the password. The user (admin or teacher) must enter the correct old password, and enter an identical password in the two other fields to be able to change the password. As a sort of security the “change” button is

disabled (the button is in light gray) until the fields are filled correctly. The difference between the two figures is the navigation bar,

**a. Admin**

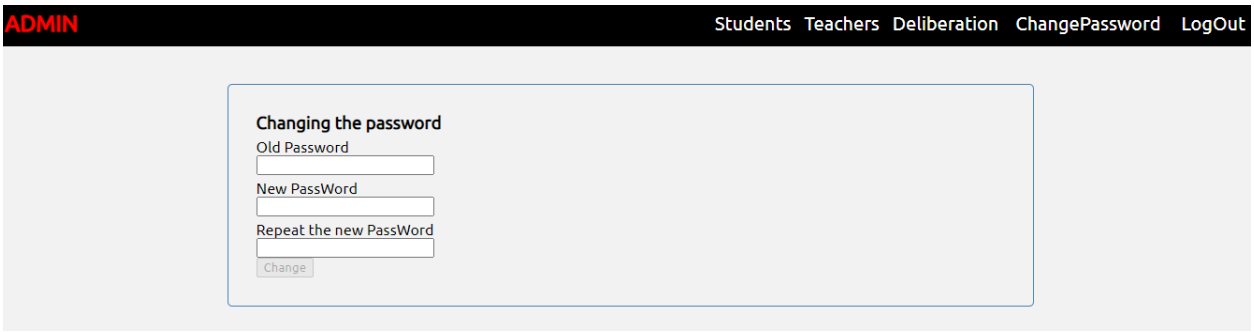


**Figure 4.14 change password for admin**

**b. Teacher**

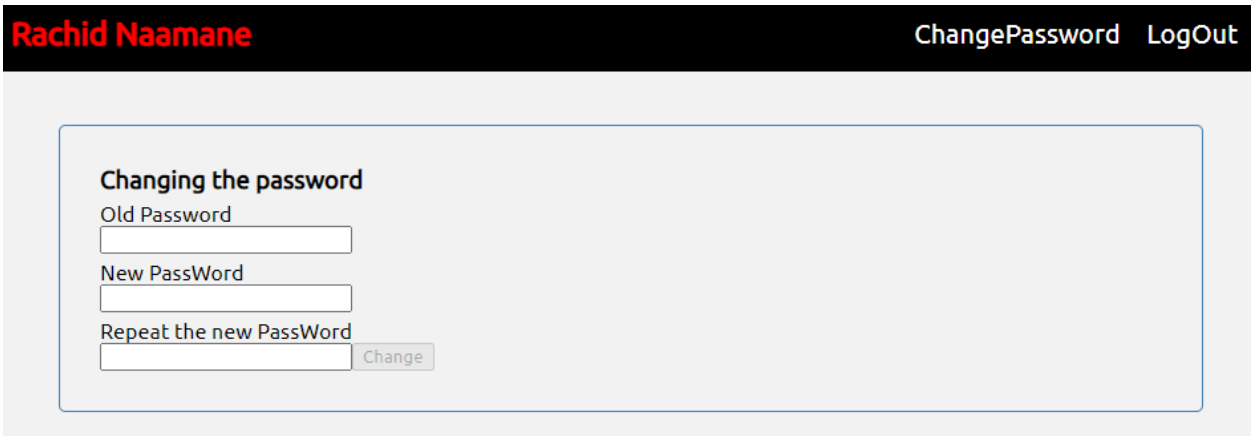


**Figure 4.15 change password for teacher**

### 4.2.8 Interface of submit grades

After a successful login, the teacher can choose one of the courses and enter the grades then save them. After the submission, the teacher can view the grades and correct any error. For the courses module three type of grades can be submitted (Figure 4.16) where the lab modules have one grade (Figure 4.17).

**a. Ordinary course (Control, exam, and makeup)**

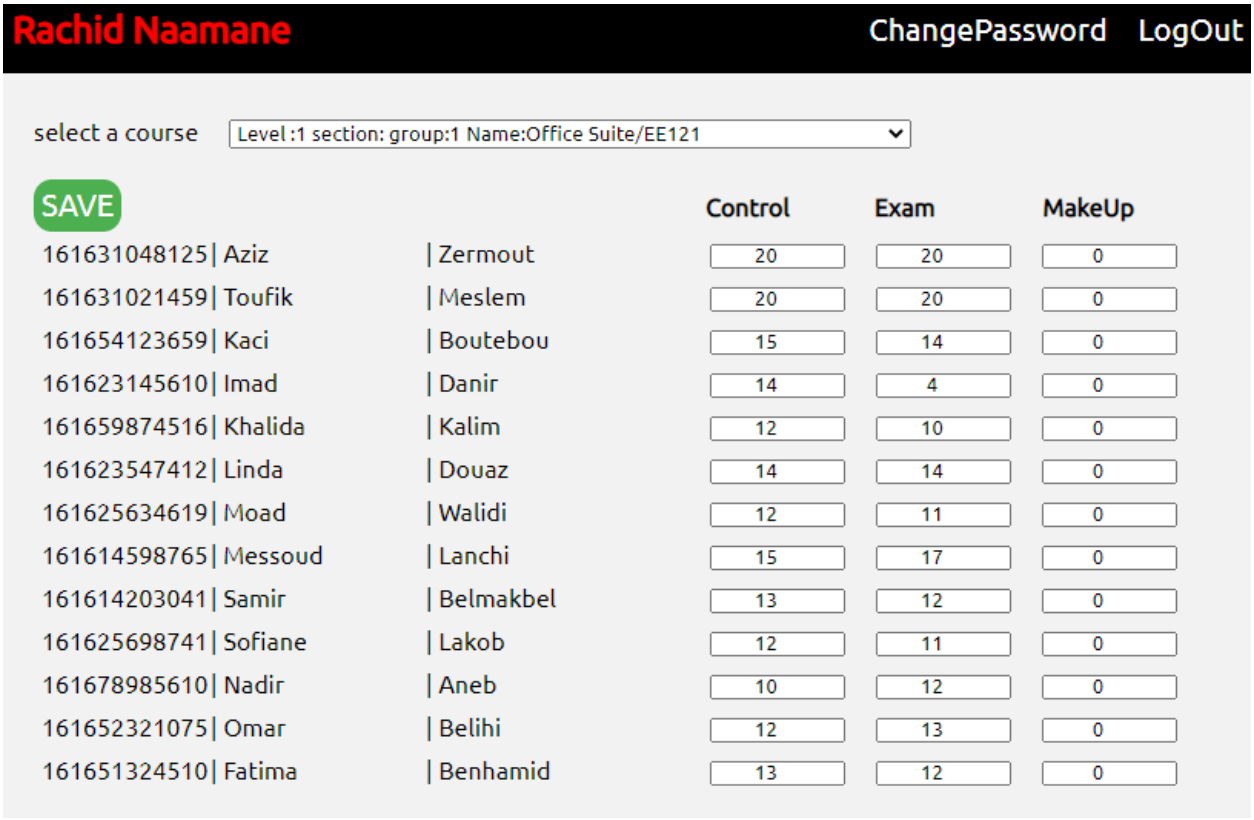


**Figure 4.16 interface of submit grades of an ordinary module**

**b. Lab**

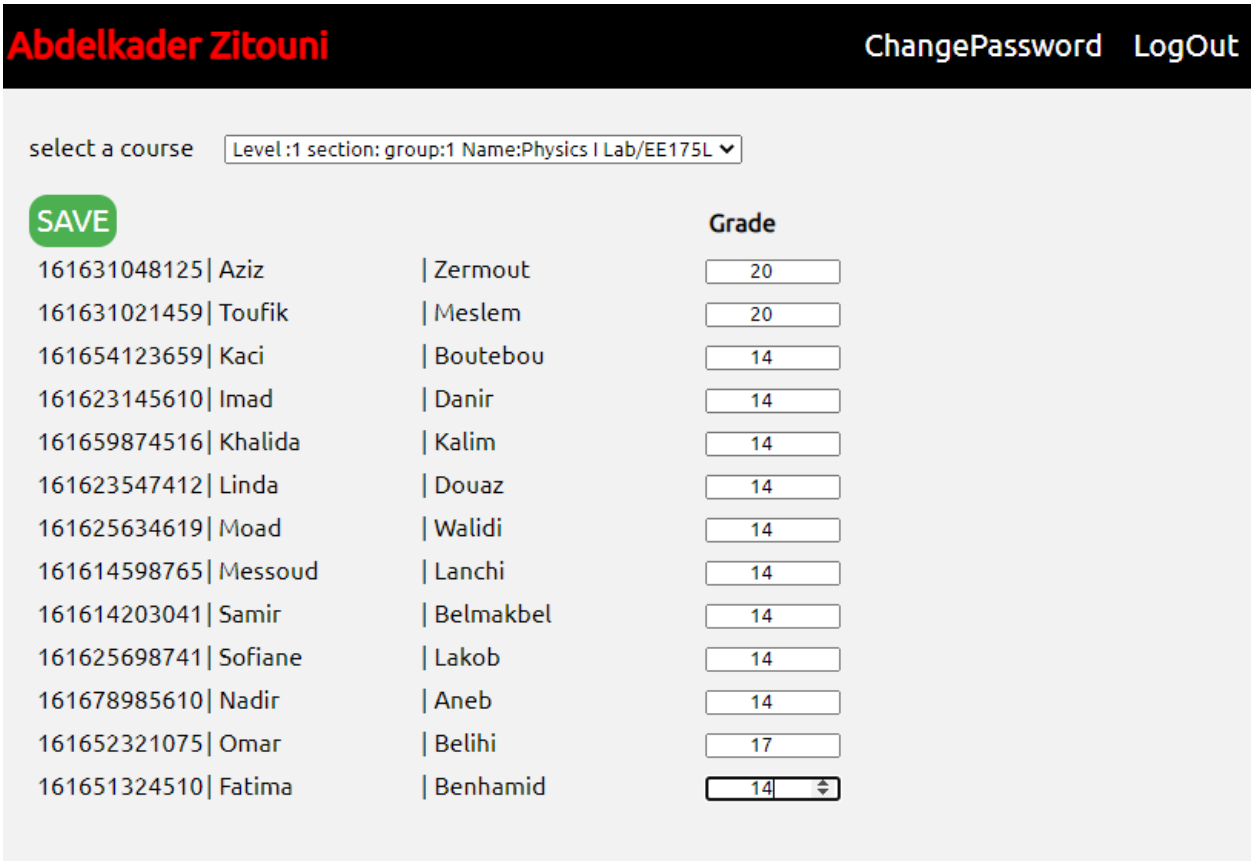


**Figure 4.17 interface of submit Lab grades**

### 4.2.9 Final deliberation interface

Figure 4.18 shows the final deliberation of level 1 group 1 with 13 students and the grades are set to zero by default. We tested the functionality of our application by entering some values for the first two students and the twelfth student in the table.

**N:** the order of the students.

**Nom, Prenom:** Family name and First name respectively.

**Group:** group of the students.

**Moy EE171:** Moy is the average of the module with the code EE171. The word moy always mean the average.

**Moy UEF11:** Stands for "Unite d'Enseignement Fondamentale", which means fundamental teaching unit. The average of the fundamental modules.

**Moy UET11:** Stands for "Unite d'Enseignement Transversale", which means transversal teaching unit. The average of English module and Office Suite module.

**Moy sem1, Moy sem2:** The average of the first and second semesters.

**Cr.sem1, Cr.sem2: Cr**edit **sem**ester one and two means the total of all module credits. The range is from zero to thirty.

**Moy L1:** the average of the first year in license cycle.

**Décision:** the final decision. It can be :

- Admis: when the student has more than ten in the total average. The student is admitted to pass to the next year.
- Ajourné: when the student has less than ten in the total average. And the student has to repeat the year.

- Admis avec dette: when the student has less than ten in the total average. But in this case the student is allowed to pass to the next level. Some modules are considered as debts.

ADMIN

Students Teachers Deliberation ChangePassword LogO

select Level Level 1 ▾ group 1

| N° | Matricule | Nom | Prénom | Objectif | Moy EE171 | Moy EE173 | Moy EE175 | Moy EE175 | Moy EL103 | Moy EE121 | Moy UET11 | Moy sem1 | Objectif S3 | Moy EE172 | Moy EE174 | Moy EE176 | Moy EE178 | Moy UEF21 | Moy EE122 | Moy EE178 | Moy UEM21 | Moy EL104 | Moy EE102 | Moy sem2 | Objectif S3 N | Moy L1 | Objectif L | Décision |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 161631048125 | Zermout | Aziz | 1 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 30 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 30 | 20.00 | 60 | Admis |
| 2 | 161631021459 | Meslem | Toufik | 1 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 30 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 | 30 | 20.00 | 60 | Admis |
| 3 | 161654123659 | Boutebou | Kaci | 1 | 10.00 | 0.00 | 0.00 | 3.33 | 0.00 | 0.00 | 0.00 | 1.58 | 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.83 | 5 | Ajourné |
| 4 | 161623145610 | Danir | Imad | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 5 | 161659874516 | Kalim | Khalida | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 6 | 161623547412 | Douaz | Linda | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 7 | 161625634619 | Walidi | Moad | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 8 | 161614598765 | Lanchi | Messoud | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 9 | 161614203041 | Belmakbel | Samir | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 10 | 161625698741 | Lakob | Sofiane | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 11 | 161678985610 | Aneb | Nadir | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |
| 12 | 161652321075 | Belihi | Omar | 1 | 15.00 | 12.40 | 11.60 | 13.00 | 17.00 | 13.00 | 4.80 | 12.09 | 12.78 | 30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 6.74 | 30 | Admis avec Dette |
| 13 | 161651324510 | Benhamid | Fatima | 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0 | 0.00 | 0 | Ajourné |

**Figure 4.18 final deliberation interface**

## 4.3 Conclusion

In this chapter, we presented the interfaces of the application with some explanations.

# General conclusion

In this project, we have presented the different steps to design and develop a web application intended for managing the registrar's office of the institute. In order to implement this project, we have gathered and analyzed problems faced by the office stuff and faculty members. For that purpose several discussions have been taken with them and then a web application has been implemented for solving some of their problems. The application is used mainly for generating the final deliberation transcripts for all promotions of the institute.

# Future works

For more completion of the work of this application and making it fulfill the requirement of the registrar's office, many things can be added:

The expansion of the application work to generate other documents and transcripts like the transcript of grades for every student. The application can be improved more by making a better and rich user interface to ease its use. The extension includes also the enhancement of its reliability and security, also improve database design to include more data about the users, and add backup database, and adding an interface for students.

# Appendix of some Back-end APIs

## These APIs are in a route that has a path:"/teachers".

```
router.get('/',(req, res,)=>{   //get the list of teachers----------------------
    Teacher.find({},function(err,data){
        res.json(data)
    })
})

router.post('/',async(req, res,)=>{  //add new teacher---------------------------
-
  const teacher = new Teacher( req.body)
  teacher.password = await bcrypt.hash(req.body.password,12)
  teacher.save()
  res.status(201).end()

    })

    router.delete('/:id',(req, res,)=>{  //delete teacher-----------------------
----
        Teacher.findByIdAndRemove(req.params.id,{useFindAndModify: false}, ()=>{r
es.status(200).end()} )
      })

        router.get('/:id',(req, res,)=>{   //get only one teacher---------------
-------
          Teacher.find({_id:req.params.id},function(err,data){
          //  find() will return an array named data of length=1 because _id is
unique
              res.json(data[0])
          })
      })


        router.post('/addcourse/:id',(req, res,)=>{  //addcourse----------------
---------
          Teacher.findOne({_id:req.params.id}, function (err,data){

          data.courses.push(req.body)
          data.save()

          })
      res.status(200).end()
```

```
        })

        router.put('/deletecourse/:id',  (req, res,)=>{  //deletecourse----------
         console.log(req.params.id)

          console.log(req.body)

          Teacher.findOne(req.body,  async function (err,data ){
          data.courses.id(req.params.id).remove()
           data.save().then(res.status(200).end())

          })
        })
```

**These APIs are in a route that has a path:"/students".**

```
router.get('/',(req, res,)=>{   //get the list of students-----------------------
    Student.find({},function(err,data){
        res.json(data)
    })
    console.log("the reqest is /students");})

router.get('/:id',(req, res,)=>{   //get only one student-----------------------
    const one =Student.find({_id:req.params.id},function(err,data){
        console.log(data[0])
        res.json(data[0])
    })
    console.log("the reqest is /students/id");
})

router.post('/',(req, res,)=>{  //add new student---------------------------
    console.log(req.body)
  const student = new Student( req.body)
  student.save()
  res.status(200).end()
    })

    router.delete('/:id',(req, res,)=>{  //delete student-----------------------
----
        Student.findByIdAndRemove(req.params.id,{useFindAndModify: false}, ()=>{}
 )
        console.log(`the of the object is ${req.params.id}`)
```

```
      res.status(200).end()
        })

        router.put('/:id',(req, res,)=>{  //update student----------------------
-----
          console.log(req.params.id)
           Student.findByIdAndUpdate(req.params.id, req.body,{useFindAndModify: f
                                    alse},   ()=>{})
      res.status(200).end()
        })

        router.put('/getgrades/',(req, res,)=>{  //get grades of student---------
---------
          Student.find(req.body,'level',function(err,data){
           // console.log(`the requestt is `)
            res.json(data)
            res.status(200).end()
           // console.log(data)
        })
        })
```

**These APIs are in a route that has a path:"/grades".**

```
        router.put('/',(req, res,)=>{  //get grades of student------------------
---------
          //the filter function is used to eliminate the empty string from the co
             urse inside the req.body
          const filter=function(course){
            if(course.section==='' && course.group==='') return {level:course.lev
el}
            else if(course.section==='' && course.group!=='') return {level:cours
                                                                    e.level, gr
                                                                    oup:course.
                                                                    group}
```

```
            else if(course.section!=='' && course.group==='') return {level:cours
                                                                    e.level, sec
                                                                    tion:course.
                                                                    section}
            else return {level:course.level, section:course.section,group:course.
group}}

          Student.find(filter(req.body.course),`fname lname rnumber grades${req.b
             ody.course.level}.${req.body.course.coursecode}`,function(err,data){
           console.log(req.body)
           console.log("------------")
           console.log(data)
            res.status(200).json(data).end()
        })
        })

        router.put('/save/',(req, res,)=>{ // put the grades of the students to i
                                           ts corres-ponding students
          const grades = req.body.grades
          const course= req.body.course
          grades.map((grade)=>{
            Student.findById(grade._id, function (err, doc) {
              doc[`grades${course.level}`][`${course.coursecode}`] = grade[`grade
                s${course.level}`][`${course.coursecode}`] ;
              doc.save();
            })
          })
          res.status(200).end()
        })

        router.put('/deliberation/',(req, res,)=>{ // fetching the grades of the
students to its corresponding selection
          console.log("this is deliberation")
          console.log(req.body)

          const filter=function(req){
            if(req.body.group==='' ) return {level:req.body.level}
            else return {level:req.body.level, group:req.body.group}}

          Student.find( filter(req) ,`fname lname rnumber group grades${req.body.
level}`, function(err,data){
              res.status(200).json(data).end()

         })

        })
```

# Bibliography


[1] "techterms," [Online]. Available: https://techterms.com/definition/web_application. [Accessed 18 August 2021].

[2] "medium," [Online]. Available: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. [Accessed 21 August 2021].

[3] "web clinet," [Online]. Available: https://www.pcmag.com/encyclopedia/term/web-client. [Accessed 7 September 2021].

[4] "web server," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server. [Accessed 7 September 2021].

[5] "An overview of HTTP," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. [Accessed 8 September 2021].

[6] "what is HTTP ?," [Online]. Available: https://www.w3schools.com/whatis/whatis_http.asp. [Accessed 8 September 2021].

[7] "What is a URL ?," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL. [Accessed 8 September 2021].

[8] 2021. [Online]. Available: https://code.visualstudio.com/docs. [Accessed 14 August 2021].

[9] "c-sharpcorner," [Online]. Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/. [Accessed 14 August 2021].

[10] [Online]. Available: https://www.techopedia.com/definition/27927/nodejs. [Accessed 14 August 2021].

[11] "guru99," [Online]. Available: https://www.guru99.com/node-js-express.html. [Accessed 14 August 2021].

[12] A. M. N. D. N. BAATCHIA Riyadh, "Design and Implementation of a Web-based Management System. Case of TheScientific Council of the “IGEE”.," Boumerdes, 2019.

[13] "Adding Mongoose in MEAN Stack," [Online]. Available: https://www.javatpoint.com/adding-mongoose-in-mean-stack. [Accessed 8

September 2021].

[14] "what is MongoDB compass ?," [Online]. Available: https://docs.mongodb.com/compass/current/. [Accessed 10 September 2021].

[15] "web browser," [Online]. Available: https://www.techopedia.com/definition/288/web-browser. [Accessed 10 September 2021].

[16] "Lucidchart," [Online]. Available: https://en.wikipedia.org/wiki/Lucidchart. [Accessed Septemver 2021].

[17] "techopedia," [Online]. Available: https://www.techopedia.com/definition/1892/hypertext-markup-language-html. [Accessed 14 August 2021].

[18] "techopedia," [Online]. Available: https://www.techopedia.com/definition/26268/cascading-style-sheet-css. [Accessed 14 August 2021].

[19] "techopedia," [Online]. Available: https://www.techopedia.com/definition/3929/javascript-js. [Accessed 14 August 2021].

[20] "Bycrypt," [Online]. Available: https://en.wikipedia.org/wiki/Bcrypt. [Accessed 29 September 2021].

[21] "JSON Web Tokens," [Online]. Available: https://auth0.com/docs/security/tokens/json-web-tokens. [Accessed 29 September 2021].

[22] "Modelling Language," [Online]. Available: https://www.techopedia.com/definition/20810/modeling-language. [Accessed 11 September 2021].

[23] "Unified Modeling Language (UML) | An Introduction," [Online]. Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/. [Accessed 14 September 2021].

[24] "Use Case Diagrams: An Agile Introduction," [Online]. Available: http://agilemodeling.com/artifacts/useCaseDiagram.htm. [Accessed 14 September 2021].

[25] "UML use case diagrams," [Online]. Available: https://www.uml-diagrams.org/use-case-diagrams.html.

[26] "Elements of a Use Case Diagram," [Online]. Available: https://www.e-education.psu.edu/geog468/l8_p4.html.

[27] "database," [Online]. Available: https://searchdatamanagement.techtarget.com/definition/database.

[28] "what is relational database ?," [Online]. Available: https://aws.amazon.com/relational-database/.

[29] "Non-relational data and NoSQL," [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data. [Accessed September 2021].

[30] "MongoDB vs MySQL Differences," [Online]. Available: https://www.mongodb.com/compare/mongodb-mysql. [Accessed 16 Semptember 2021].

[31] "oreilly," [Online]. Available: https://www.oreilly.com/openbook/webclient/ch01.html. [Accessed 21 August 2021].