MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF TUNIS EL MANAR

HIGHER INSTITUTE OF COMPUTING

Report of an academic project

Presented with as part of an academic project
Mini-project Operating Systems

Specialty: Engineering in Computer Systems and Software

By

Amine BEN MANSOUR

Conceiving and developing A students grades management solution

Academic supervisor **Mr Hatem Ben Sta**

College year 2020/2021

Contents

Chapter 1

Context of the project

Introduction

This chapter mainly consists of presenting a preliminary study of the project. We will start with a general presentation of the project's context. Then we will make a study showing the existing situation in order to criticize it which will lead us to the optimal solution.

1.1 Subject presentation

This project consists of studying, designing and enforcing a student grade management application web solution. This modern solution built basically on the popular general purposes language, python. This solution will contributing to the digitizing circuit of universities and colleges and being up to date with the latest technologies

1.1.1 Study of the existing

The idea is to study the current system and recognize its weaknesses. We must got a critical thinking in order to get around any imperfection.

1.1.2 Analysis of the existing

Most of the universities still taking the hard load manually, using the usual tools of sheets and pencils, maybe google sheets at best and going through the long process from gathering students' grades to publishing results in public which include most of the time fragile informations like ID cards' numbers'.

1.2 Solution

Suggested solution named "Notety" is an end-to-end solution which will help both student and administration from the very beginning of the process when the student subscribes to the platform till it graduates.

Conclusion

During this introductory chapter, we specified our current lacks in universities' system and we also sat our solution which still abstract until now. Therefore we will make our goals clearer in the next chapter

Chapter 2

Planning And Specification Of Needs

Introduction

Throughout this chapter we will highlight the needs of this project and pave the way for its realization. We will also identify actors interacting with our system. Furthermore, we will list our functional and non-functional needs.

2.1 Identification of actors

Actors that are going to deal with our application are the following:

- **Admin:** Pre-loaded upon delivery of the application. Charged of setting up the environment of the university.
- **Student:** Got the right to inspect their results

Our solution contains exactly one administrator and zero or more students.

2.2 Needs Capture

This part will be devoted to identify the functional and non-functional needs of our project.

2.2.1 Identification of functional needs

Functional needs responses to features required by the product owner. This is a primary list it could be altered later on.

Here our functional requirements.

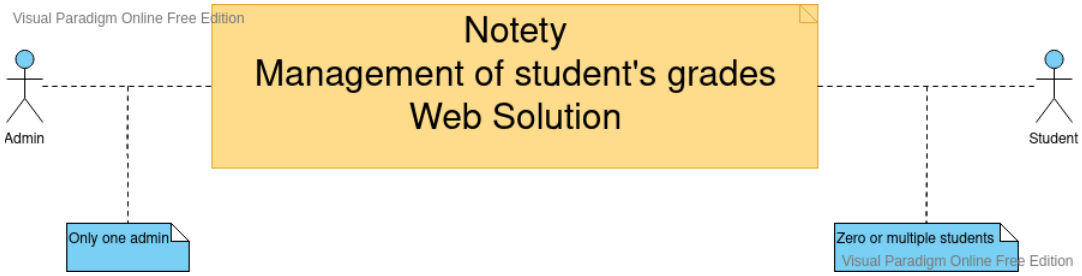

FIGURE 2.1: Static context diagram

- **Authentication:** Actors should be able to sign-up, login and logout to and from our application.
- **System Management:** Freely add classes, their associating modules and their associating subject fluently so students could subscribe to classes they belong to.
- **Grades Remission:** Assign students within different classes with their own grades. In the other hand each students will see their grades individually respecting other's privacy
- **Classes Consultation:** View different classes with the ability to delete them individually. In case of deletion all the students associated and their grades will be deleted.
- **Classes Consultation:** Choose any class in system and delete it. This option is very useful in case of system rectification.
- **Students Consultation:** View students within different classes with the ability to delete them individually. In case of deletion their associated grades will also be deleted.
- **Students Deletion:** Choose any subscribed student and delete it. This option is very useful in case of ban or graduation.
- **Grades Consultation:** Actor should be able to see their own grades. This is the primary purpose the solution was built for.

2.2.2 Identification of non-functional needs

Non-functional requirements are either optional requirements or requirements / constraints related to implementation and interoperability.

Here our non-functional requirements.

- **Availability:** A big part of the university's system depends on it which means that availability matters a lot in this application.
- **Security:** Every single data is considered a priority in this solution. Protecting users from cyber attacks is taken seriously.
- **Performance:** For brilliance reasons, performance is a sensitive point.
- **Portability and Compatibility:** Our solution would be deployed on different systems depending on the university, therefore "Notety" must be deployed easily no matter the software environment is.
- **Scalability:** Traffic would increase as the university's capacity increases. Scalability is delicate
- **Evolutivity:** This version of the product is for sure only the MVP (Minimum Viable Product). Other features will be added by time and the way we structure our application would either help or prevent us from this goal.
- **Conviviality:** Our solution should look like modern applications in terms of look and feel. Different actors should got no issues using this application.

2.3 Global use case diagram

After having defined the actors and the functionalities of the system, we are then able to write a diagram of the global use cases in which we distribute the functionalities on our actors.

2.4 Prototyping of interfaces

We say interface prototype or screen mockup a graphic representation of medium to high fidelity. Its objective is to demonstrate rough diagrams of the final appearance of the user interface. Prototypes have been built thanks to Balsamiq Mockups. Thanks to this stage we did not face any problems elaborating our final interfaces

Conclusion

Now that we have initiated the basics of carrying out the project by specifying the requests of the owner of the application and the actors within the product, we still missing the specification of the architecture of the application and our working environment, which we will discover in the next chapter.

Chapter 3

Working Environment

Introduction

In this chapter, we specify the hardware and software environments as well as the logical and physical architecture of the application and the programming languages.

3.1 Hardware and software environment

Throughout this project we used a LENOVO laptop powered by a linux environment which offers numerous advantages. We are running an Arch based distribution called Manjaro KDE.

The picture below describes the whole environment.

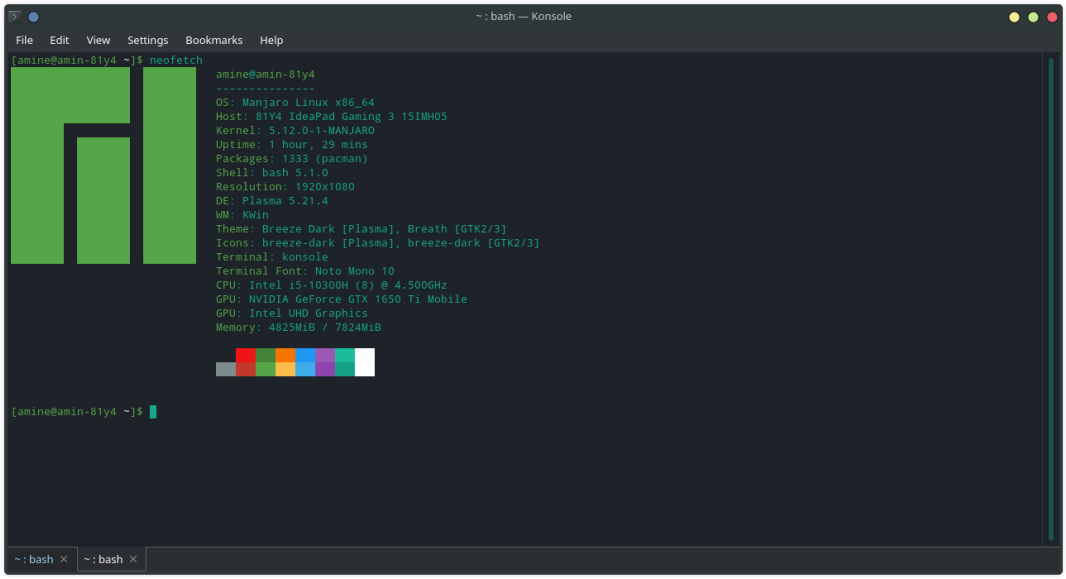

FIGURE 3.1: Linux neofetch command

Software and technologies used in order to accomplish this project will be shown in the following subsections

3.1.1 Development environment

Depending on the development environment our task could be either smooth or severe. That is why we should rely on the right tools from the beginning.

Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications.

3.1.2 Development technologies

The choice of the development technologies would have a great impact on the final product.

Python is a free and open source interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability. Python also shines in web development area.

Flask is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries were pre-existing. Only third-party libraries provide common functions.

Companies and development teams of all sizes use MongoDB because the document data model is a powerful way to store and retrieve data. MongoDB's horizontal, scale-out architecture can support huge volumes of both data and traffic.

Nginx is an application server that plays several roles including the routing of requests to the server (s). it is mainly used in huge projects and added for scalability purposes.

Aside from HTML5 and CSS3, Bootstrap is a surprisingly powerful and effective tool for developers building responsive websites and web applications.

Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere. Different universities use different environments. Thanks to docker this will not be a hindrance.

.

3.2 Solution architecture

The logical architecture ends with the identification of the functional elements of the system, while the physical architecture takes an additional step specifying the actual devices on which these functional elements run.

3.2.1 Logical architecture

This will be a monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform

3.2.2 Physical architecture

The use of docker besides of Nginx will radically change the physical architecture, mentioning that Nginx also work as a load-balancer and our application could easily scale up and down horizontally depending on the load.

A container is a process the more we launch containers (instance of an image) the more the performance increases. Hence, our application could either be deployed on a single server or multiple ones.

The following deployment diagram will highlight the physical architecture of our solution.

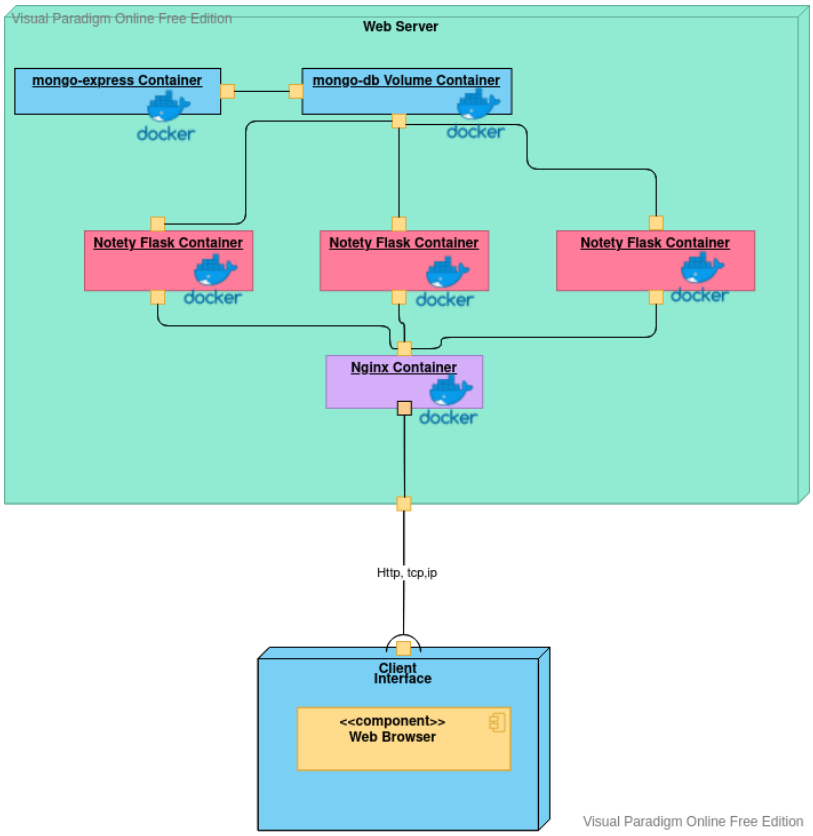

FIGURE 3.2: Deployment Diagram

As we previously said there are multiple approaches to deploy our application. We cannot describe them all but i chose the one with only web server including multiple containers of different images communication among them through ports

thanks to the extension of docker, docker-composer.

The number of instances is completely random.
These are our different containers. We mentioned the number of instances for clarity purposes.

- **Nginx container (1 instance):** This container will be listening to any incoming request from clients and then route the request to a "Notety Flask Container" which include our application.
- **Notety Flask Container (3 instances):** This container is running python and its micro-framework Flask it serves the application logic and front-end.
- **Mongo-DB Volume container (1 instance):** Since normal containers lose their data after being stopped or restarted. This one has to be a little bit unique as it holds our database and data inside of it should persist. These kind of containers is called "Volume Container".
- **Mongo-Express Container (1 instance):** This container offers a graphical interface to our database so we can manipulate our database easily from a user interface apart the mongo shell running natively within the previous container.

Conclusion

Now that we defined our application architecture and the technologies to build the anticipated application, we are now eligible to start the design and development of the application

Chapter 4

Application Design

Introduction

This chapter is the foundation of the construction of our application. Since we got no time to apply other development methodologies than the classic one, like agile methodologies. This chapter will contain most of our UML diagrams. We are going to gain much valuable time after organizing our ideas thanks to this step.

4.1 Common diagrams

We will be presenting our global use case diagram and class diagram before

4.1.1 Global use-case diagram

This global use-case diagram shows us the big picture we will get into details later on

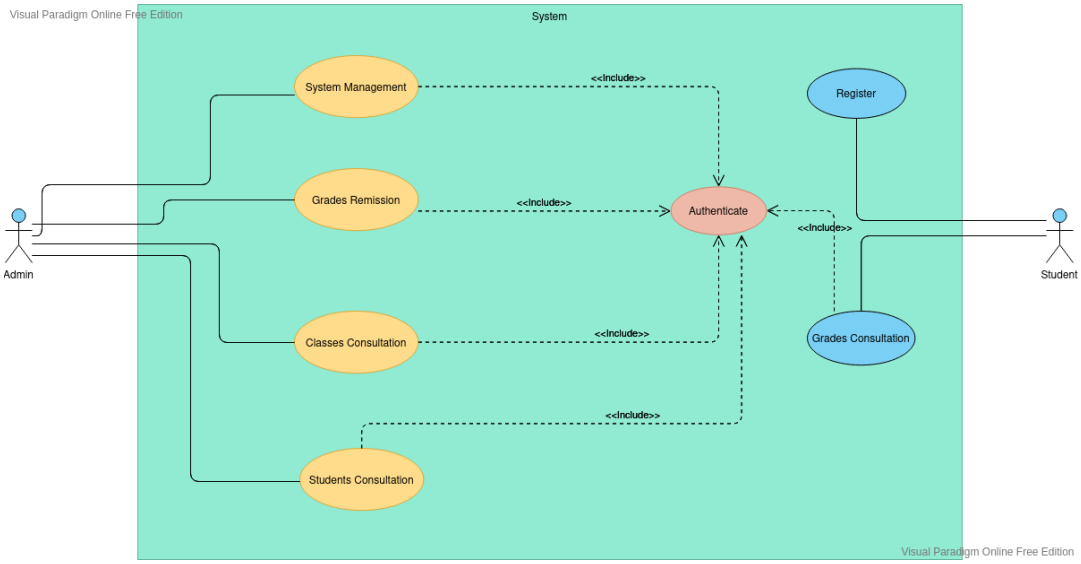

FIGURE 4.1: Global use-case diagram

The administrator will get more privileges setting up system structure, remitting grades and consulting classes and students within classes whereas student should register to the platform to get included into our system then he could authenticate to view their grades

4.1.2 Class diagram

This is our class diagram. Do not be surprised !. without relationships between entities this is the simplest class diagram I have ever designed

FIGURE 4.2: Class diagram

We encountered this simplicity thanks to the NoSQL database. We only need entities that holds our data. the file including all of our classes will be called models.py

4.2 Conceiving administrator diagrams

This section will gather all UML diagrams dedicated to the administrator including use-case, sequence and class diagrams. We will make sure that all diagrams should be clear and does not require a lot of explanation.
We separated each actor by himself to see how the application looks like from different points of view.

4.2.1 Use-case diagrams

Now that we have seen the big picture, we are going to see each use-case one by one.

Grades remission use-case diagram

This is one of the use cases that we will explain it apart.

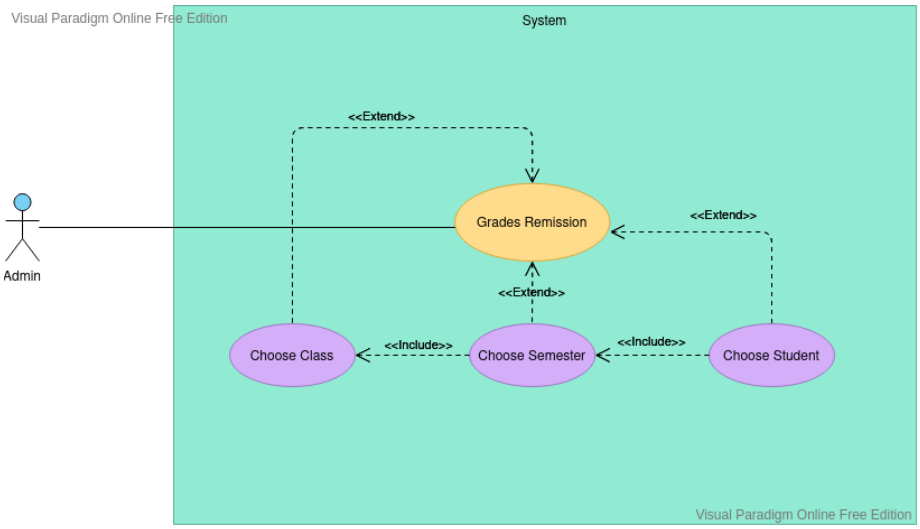

FIGURE 4.3: Grades remission use-case diagram

The process does look simple. Grades remission consists of mainly three consecutive steps in order to filter option from step to another.At any point the administrator will have the ability to rollback to any previous step.

System management use-case diagram

This diagram is too similar to the one in the figure 4.3.

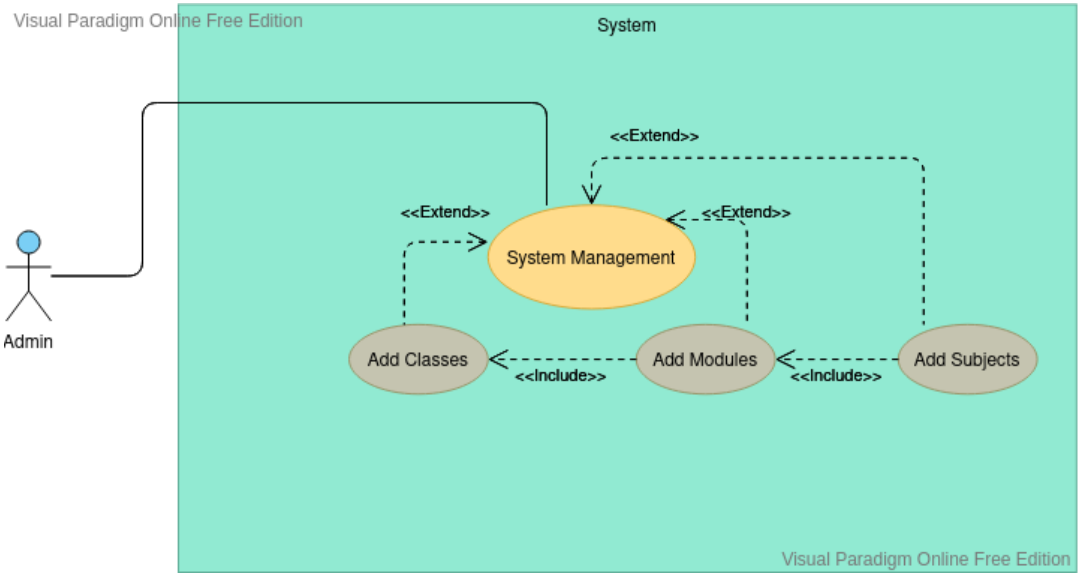

FIGURE 4.4: System management use-case diagram

This diagram relies on the same logical sequence as the previous diagram.

Students and classes use-cases diagrams

Both use-cases are too similar that we presented the in the same figure.

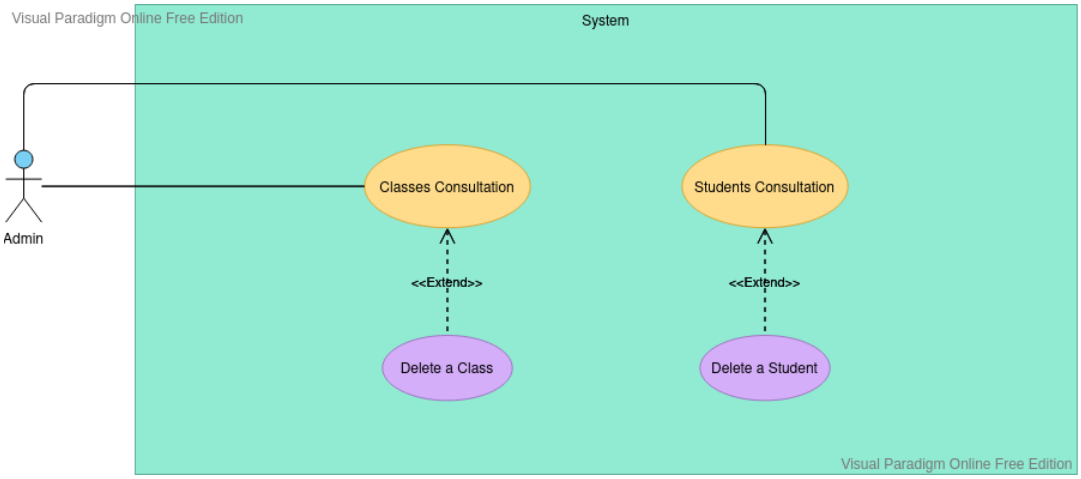

FIGURE 4.5: Students and classes use-case diagrams

Administrator could ever view students or classes list with the option to delete whatever he wants.

4.2.2 Sequence diagrams

Sequence diagrams will help us clarify our ideas. So in this section we will be presenting the most important ones.

System management sequence diagram

This diagram below describes in details what happens what happens during a system management process.

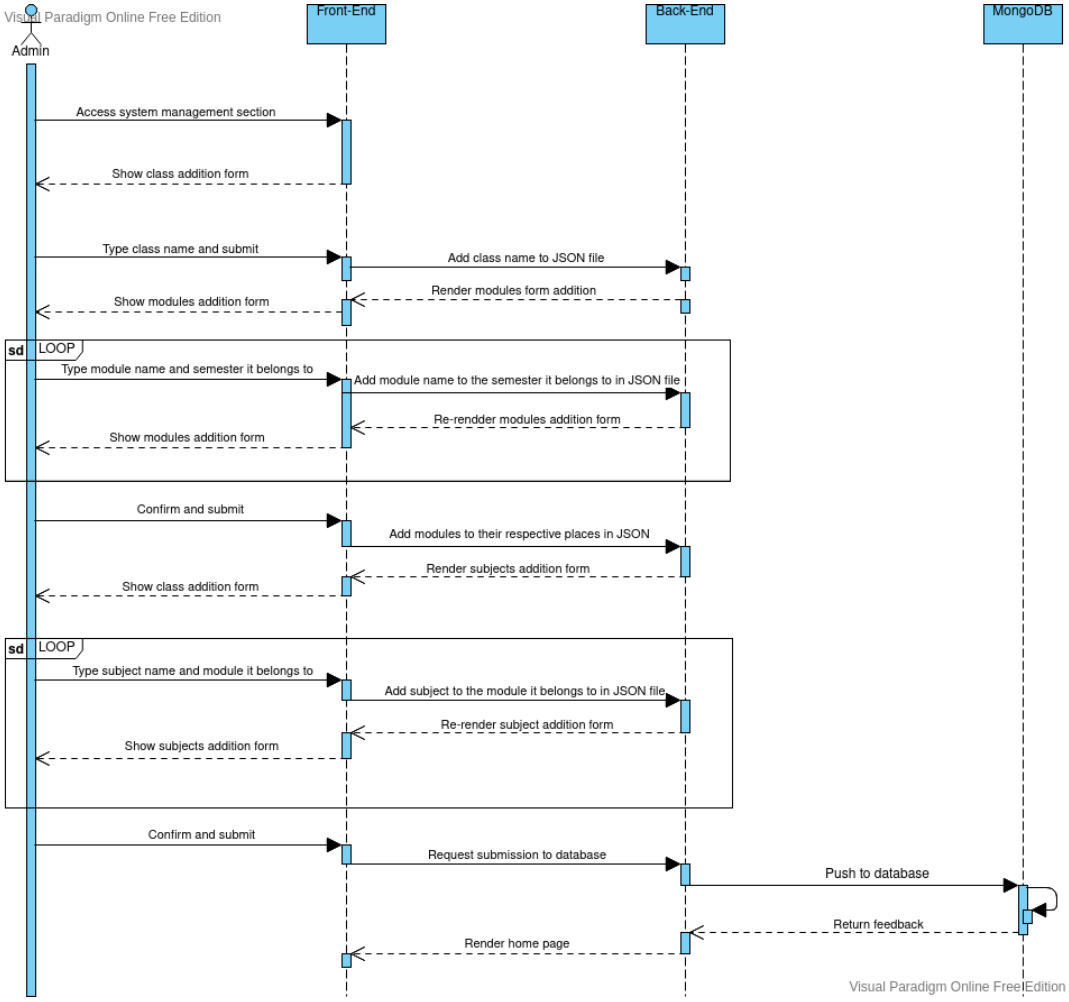

FIGURE 4.6: System management sequence diagrams

We tried to make the less possible interaction with the database to prevent unexpected problems and for efficiency purposes.

Grades remission sequence diagram

This diagram below describes in details what happens what happens during a grades remission process.

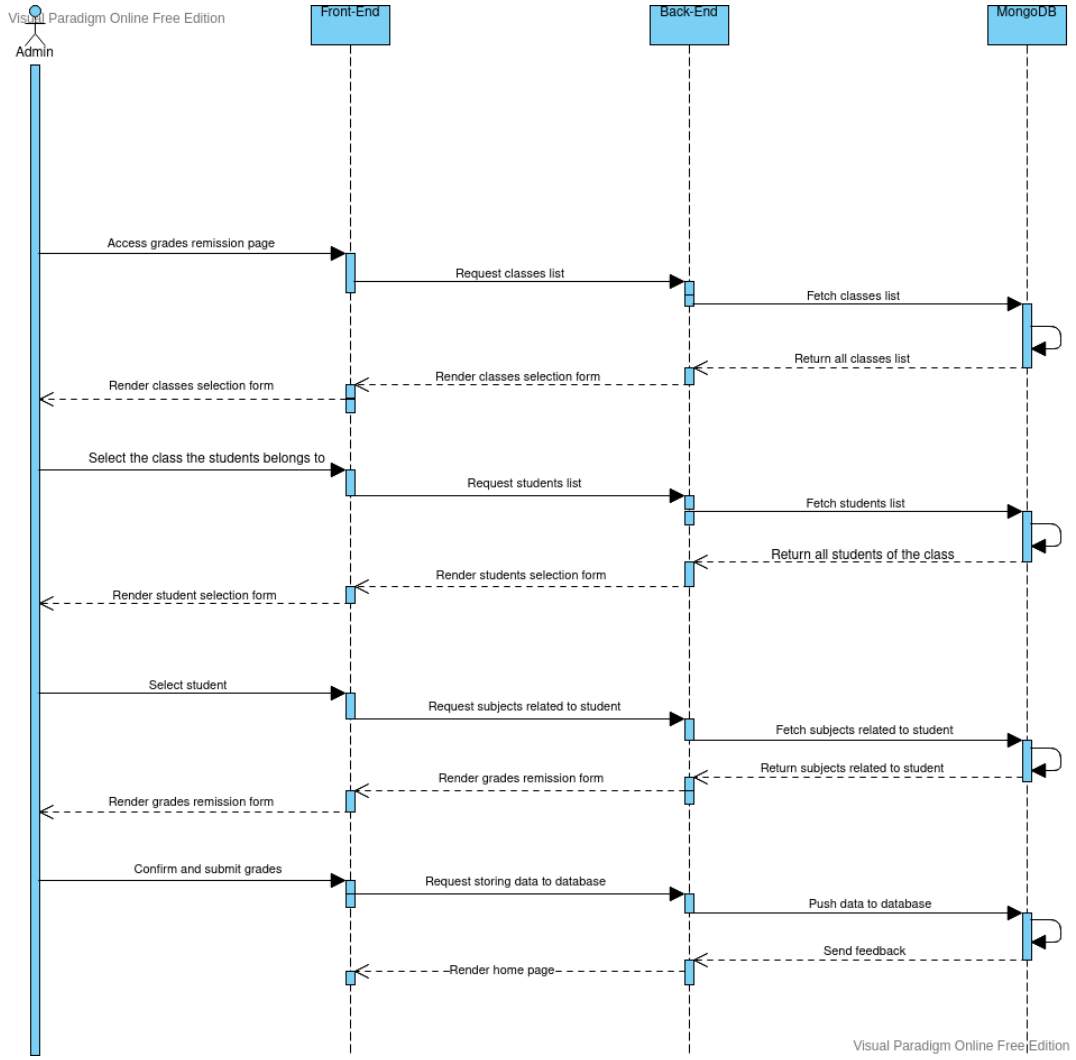

FIGURE 4.7: Grades remission sequence diagrams

Class consultation sequence diagram

As we said multiple times, class consultation and student consultation processes are way too similar. We will be satisfied by only presenting one of them.

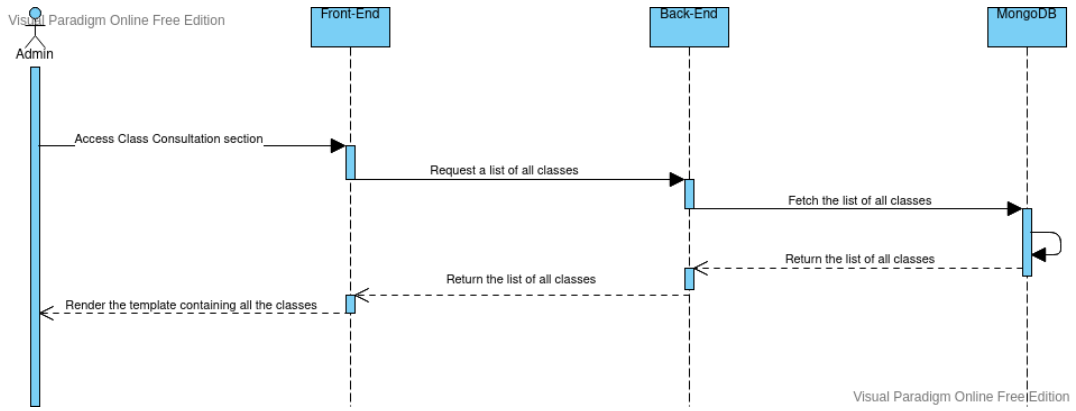

FIGURE 4.8: Class consultation sequence diagram

4.3 Conceiving student diagrams

This section will gather all of our UML diagrams dedicated for students including use-case and sequence diagrams.
We will make sure that all diagrams should be clear and does not require a lot of explanation.

4.3.1 Use-case diagrams

We will present the most important use case diagrams here.

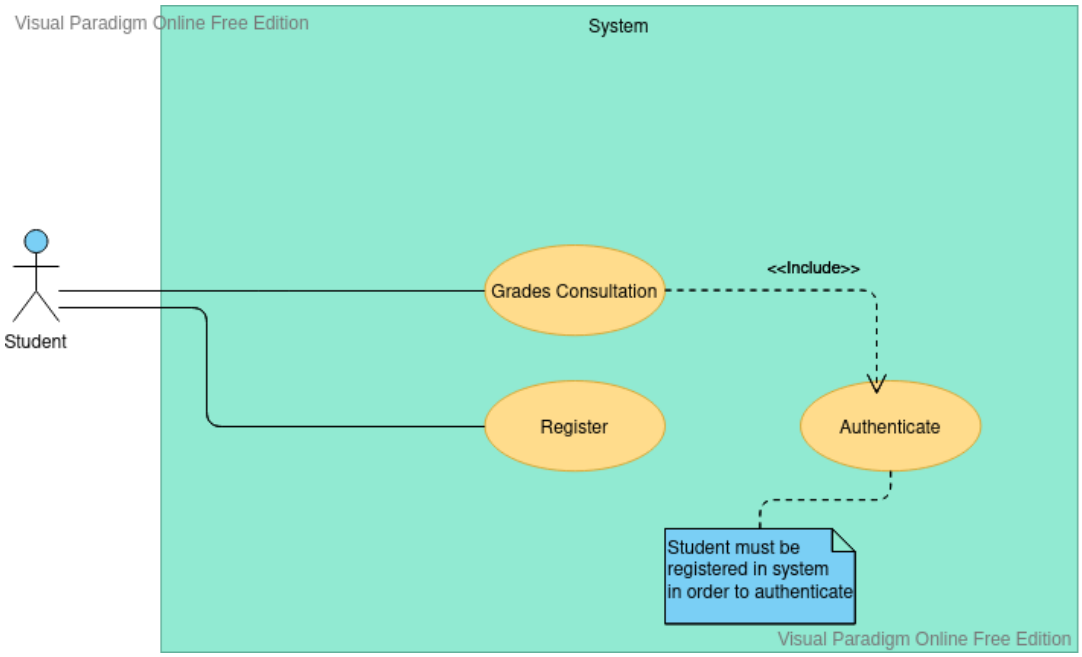

FIGURE 4.9: Student use case diagram

Student will be able to view their grades only if he is logged in into "Notety". Student can not login if he is not registered to the platform.

4.3.2 Sequence diagrams

Students internet skill may vary from one to another so their only task should remain as simple as possible so everyone could see their grades without relying to anyone else.

Login sequence diagram

Authentication is a necessary part of almost any development project. It helps specify the beneficiary of our services and protect his data.

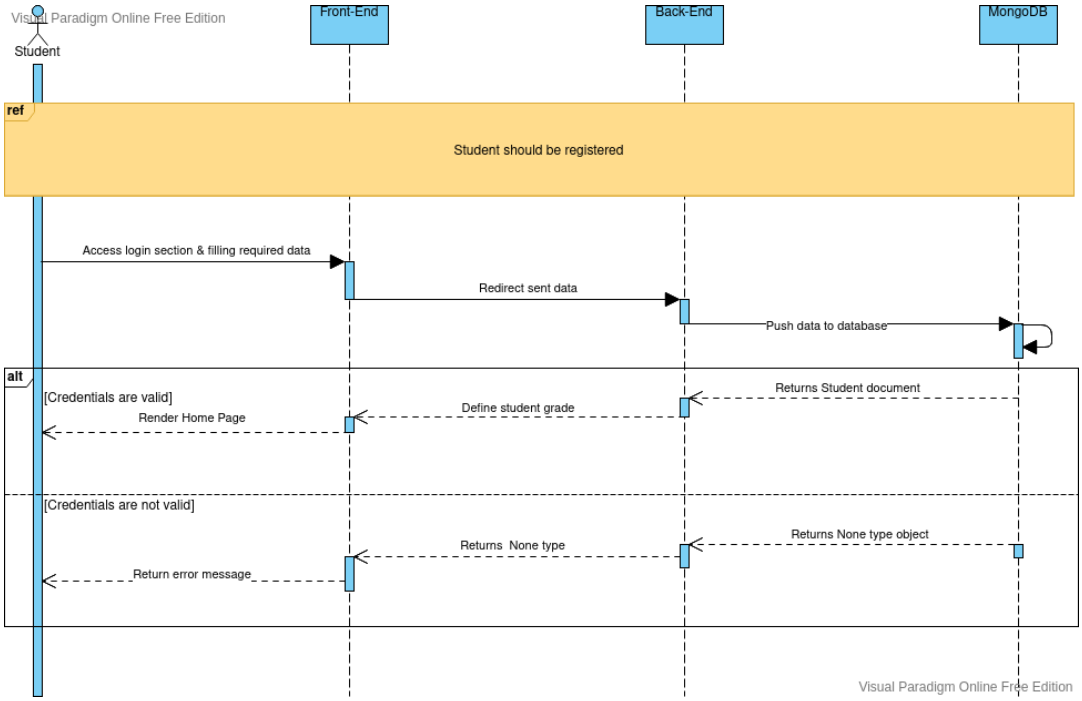

FIGURE 4.10: Login sequence diagram

Grades consultation sequence diagram

This is the main feature of "Notety". All other processes are done for the student to get advantage of this solution and can view his grades in private. The diagram below show us how it is done.

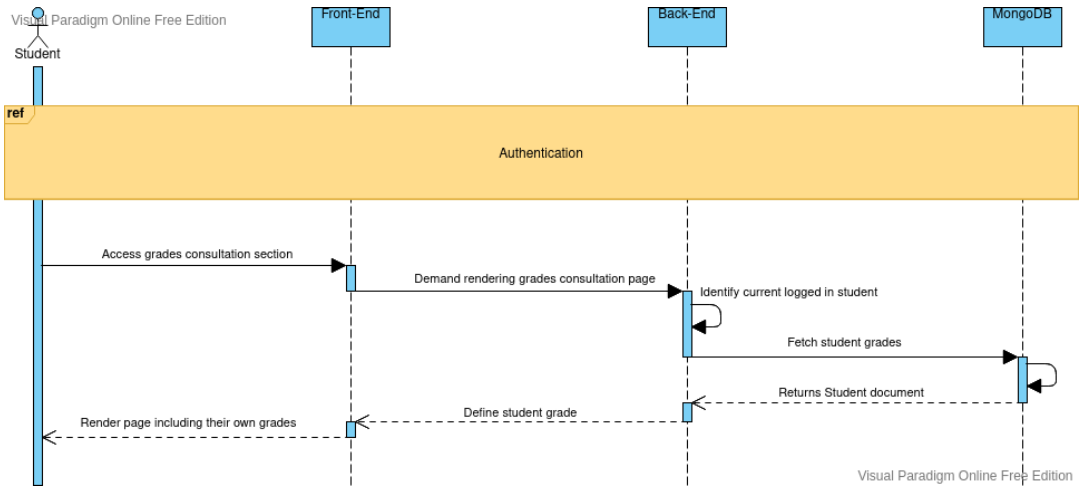

FIGURE 4.11: Grades consultation sequence diagram

4.4 Database modeling

Unlike relational databases, even if mongo-db offers a very flexible and intuitive data modeling system, NoSQL databases requires a special modeling in order to specify our fields along with their types.

Here are the collections within our database.

- **Admin Collection:** Contains required data to identify the administrator, these comes pre-populated when we ship the application and they could be only altered through mongo-shell or mongo-express for security purposes. the default username is "admin" and the default password is "admin".
- **Data collection:** Each document of this collections contains informations about a specific class including its name, its modules and its subjects. So that enrolled students should necessarily pass them all.
- **Student collection:** Each document includes all informations about the students like email and password
- **Grades Collection:** Each document encloses a document from data collection and student collection in order to assign students to their right classes and later on it is going to be altered to include each student grades.

We were able to get rid of student and data collection and gather all the informations in grades collection and it still works. But for evolutivity purposes we decided to decouple data apart.

Conclusion

Our diagrams are all set up and we are from now on able to start the development process. The next chapter will build our application.

Chapter 5

Realization

Introduction

The application has grew up and this chapter will not be sufficient to discover all of it. This chapter will only be a teaser for you to discover the application but i still highly recommend reading it. It will include some features through some screenshots featuring our application, database and the structure of the project.

5.1 Project structure

Writing a clean code is very import in order to guarantee that the extensibility. if the project is well structured and the code is well written we will got no worries adding new features.

We tried to follow programming best practices as much as possible. The figure below underlines our all of our project directories and files

FIGURE 5.1: Project structure

5.2 About application

The following subsections will include some screenshots of the application.

5.3 Common interfaces

Our admin and students will only share the same login page, each one got his interface but they are way too similar.

5.3.1 Administrator specific sections

This section is dedicated for administrator's part of the application.

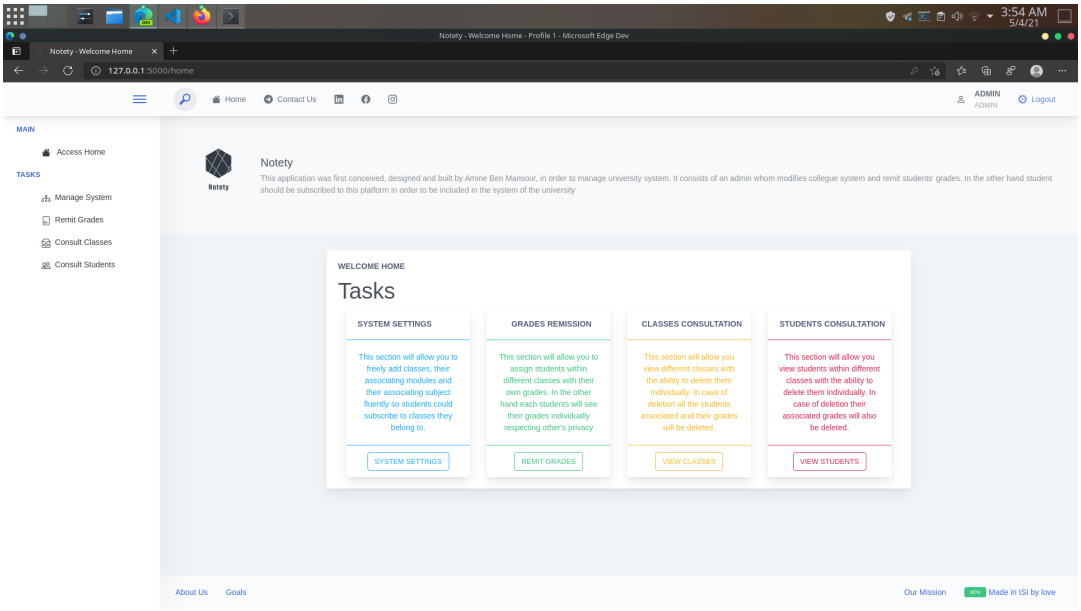

FIGURE 5.2: Admin Home Page

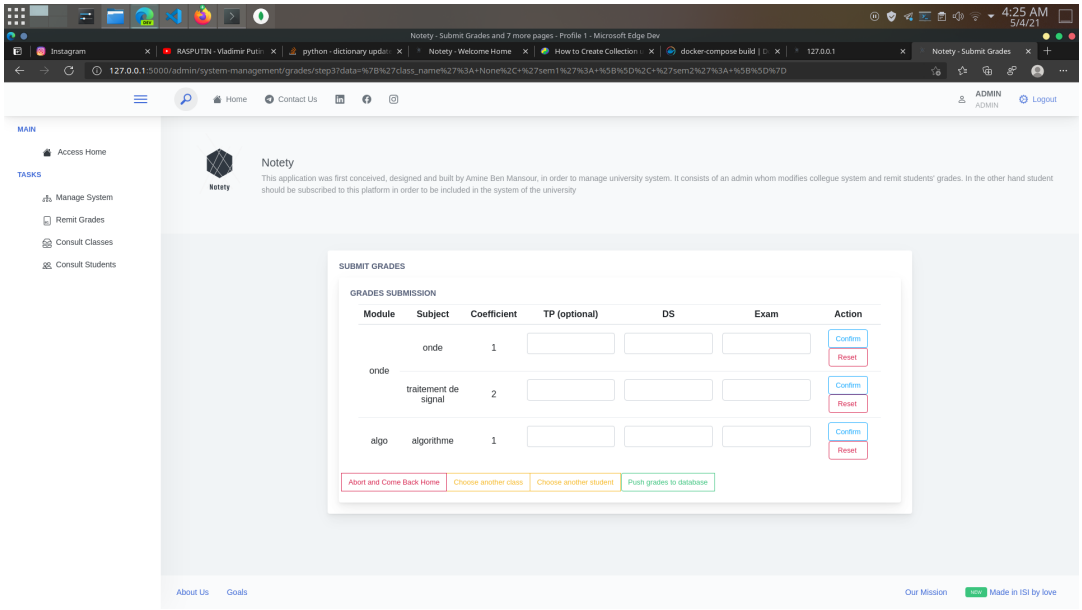

FIGURE 5.3: Admin remission interface

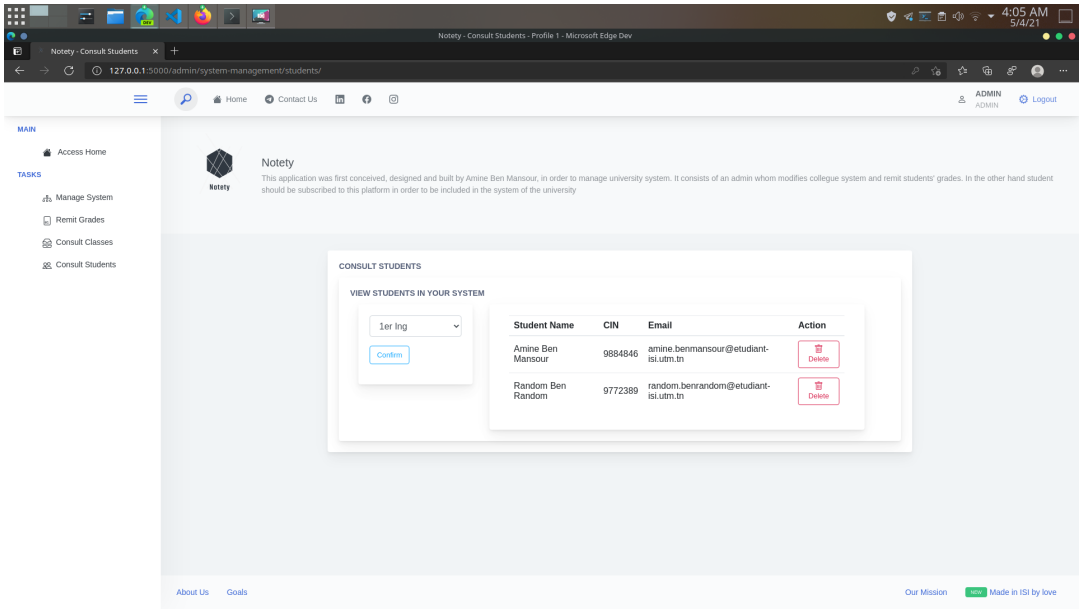

FIGURE 5.4: Admin students consultation interface

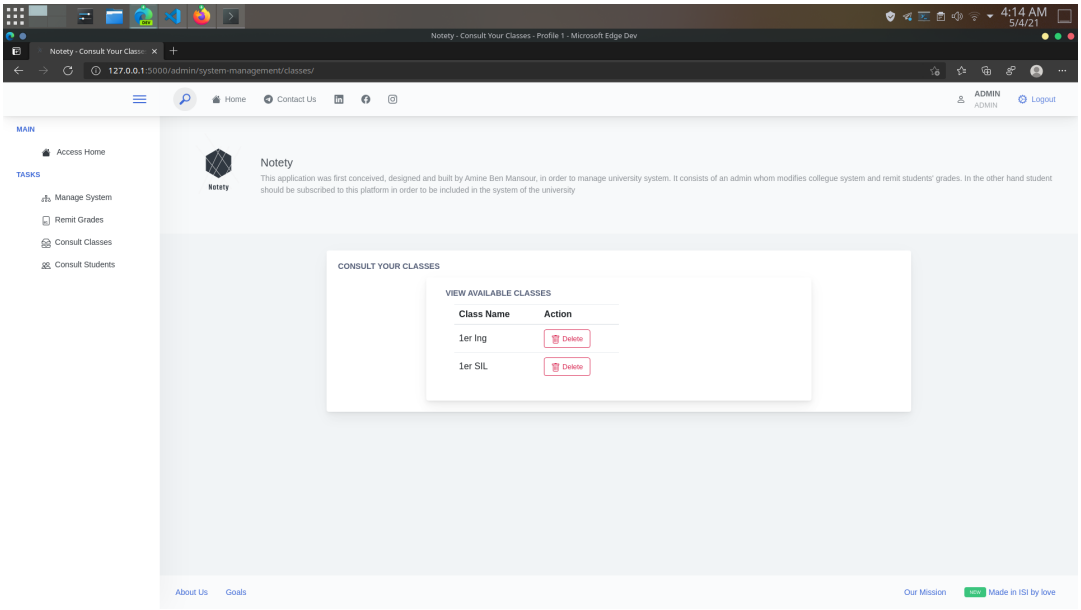

FIGURE 5.5: Admin class consultation interface

5.3.2 Students specific section

This section is dedicated for student's part of the application.

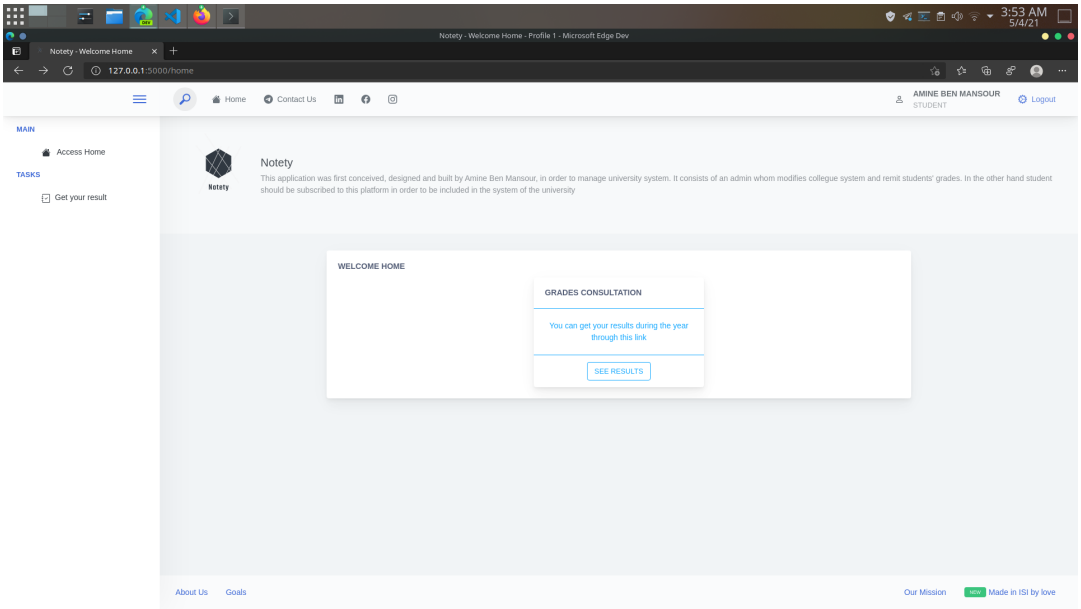

FIGURE 5.6: Student Home Page

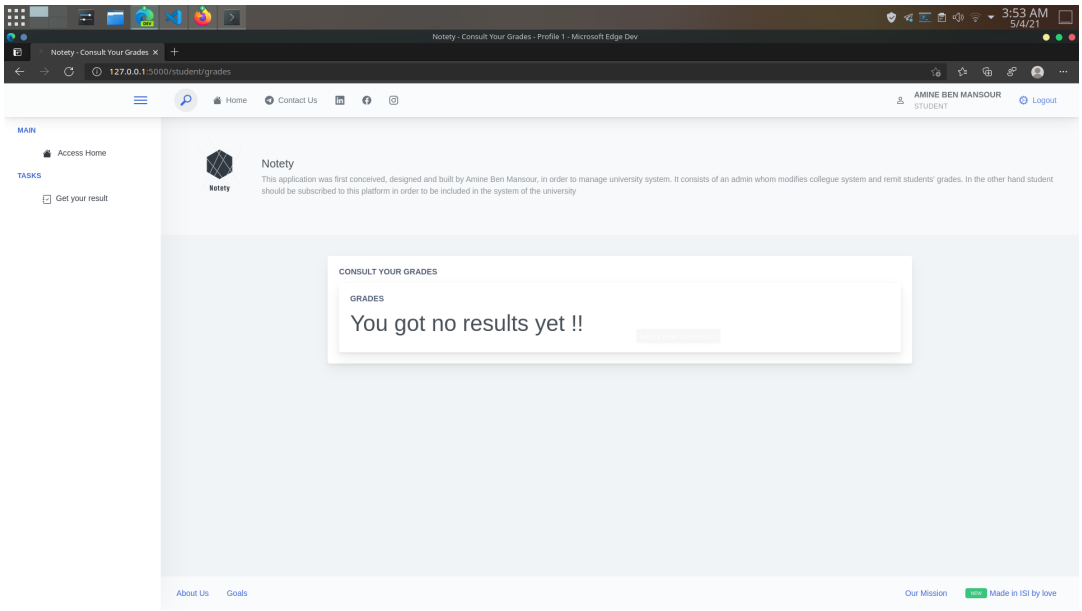

FIGURE 5.7: Student consulting his grades - No results interface

In case student has consulted his grades before admin remitted them he will see a specific page

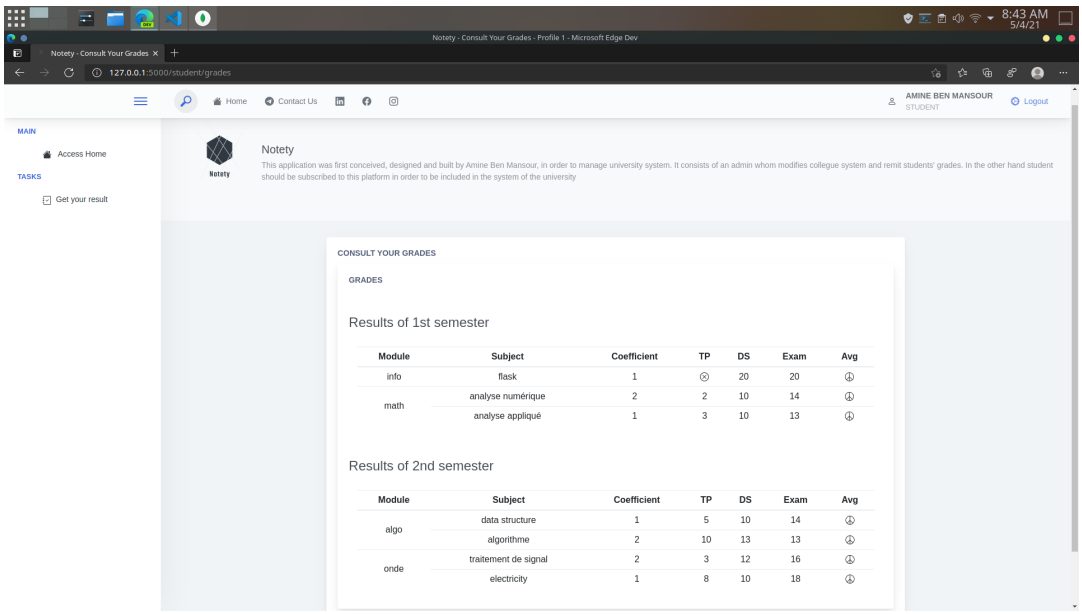

FIGURE 5.8: Student consulting his grades - With results interface

5.3.3 Handling unexpected errors

The application uses feedback messages a lot and can handle different errors like 404 Not Fount, 403 Unauthorized and 500 Server Error. The picture below shows 404 Not Found Error Page.

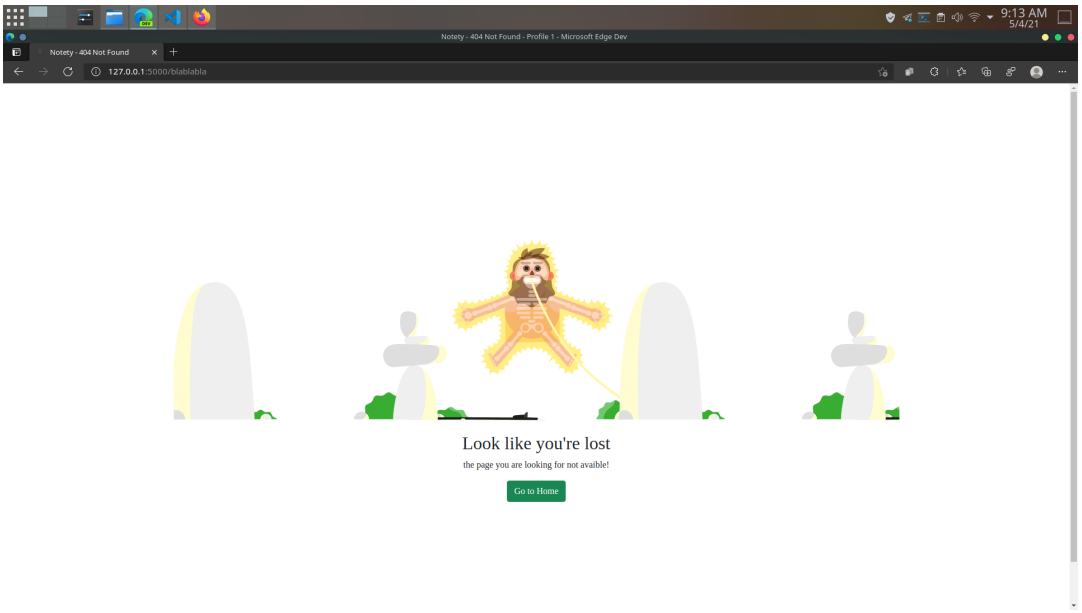

FIGURE 5.9: 404 Not Found page error interface

5.3.4 Database

Compass is the graphical interface of our database it is a way too much similar to the image of mongo-express but it is installed on local machines. It offers a nice interface to explore our database, its collections and data.
Through these three figures below we will get an idea about how does our data-collection, students-collection and grades-collection look like.

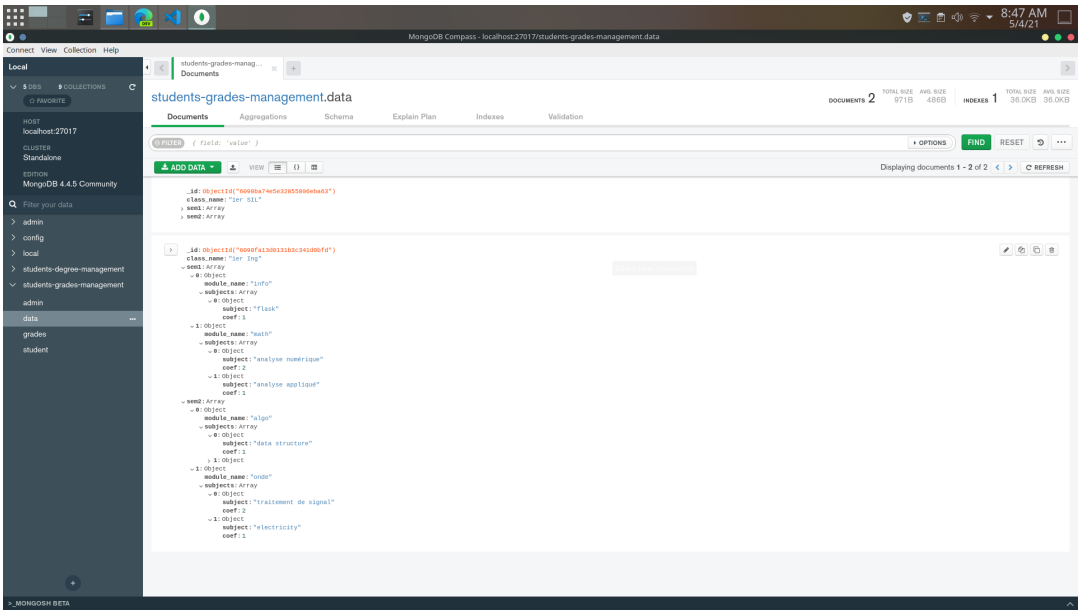

FIGURE 5.10: Data collection

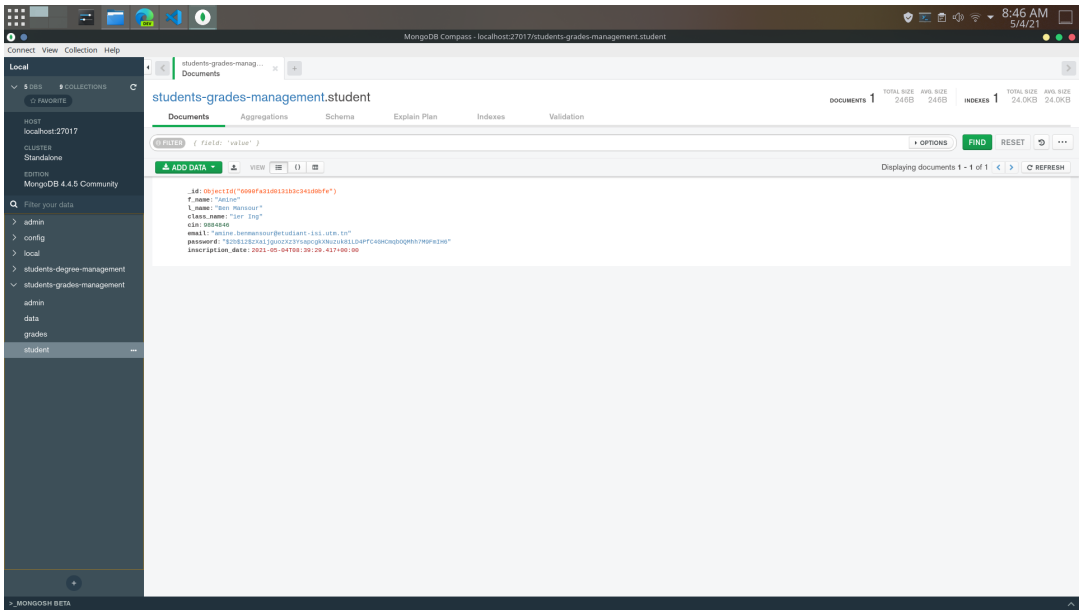

FIGURE 5.11: Students collection

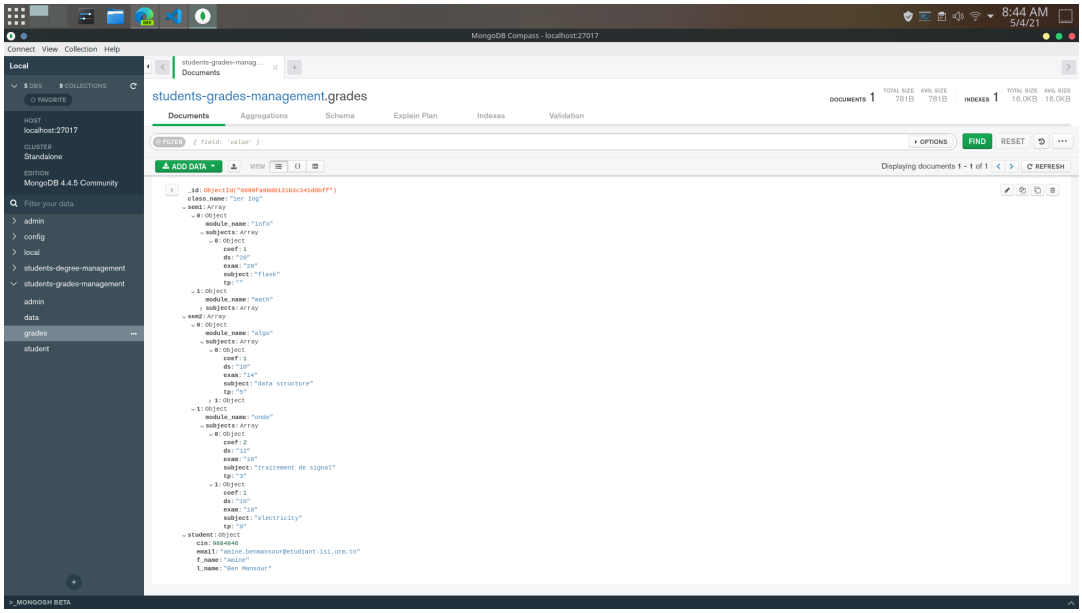

FIGURE 5.12: Grades collection

5.4 Deployment

You can launch our application through a development server which requires a lot of configurations, installations and dependencies but guess what!!. Even cooler, our application is already deployed in production and we are ready to go. Our application is runnable on any platform and any environment with no issues or headaches. All you have to install is docker and docker-compose.

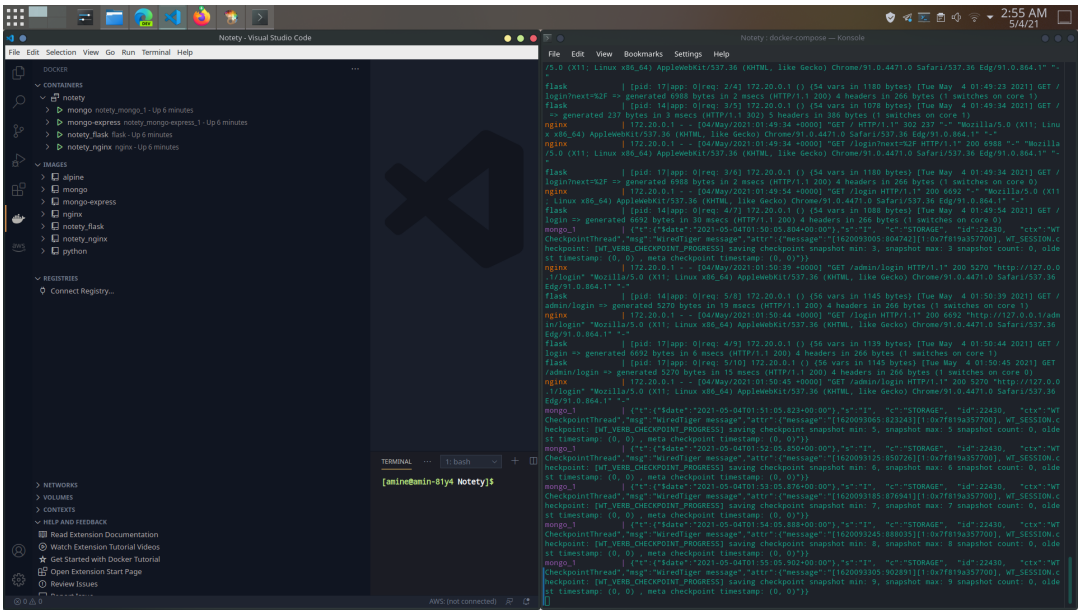

FIGURE 5.13: Terminal and Graphical proof for our application running on docker

As the picture shows we are able to view our containers also from vs-code thanks to the extension called docker.

Conclusion

We reached the final step of our project and the results as we see are fascinating we matched our functional and non-functional requirements defined in chapter 2. Now both student and administrator are taking advantage of digitization and modern well conceived solutions like "Notety".