جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⴰⵢⵏ

AL AKHAWAYN UNIVERSITY

Design Report: Tutoring Center Management Software (TCMS)

Project Team:

Touti Nour Elhouda

Imane Chtaini

Mehdi Boujrada

Mahmoud Erradi

Client Information:

Business name: Pr.Achraf

Business address:

rizqi.school@gmail.com

Business contact number: 06 64 33 84 58 / 08 08 55 06 15 /

Personal contact number: 06 63 26 85 53

Supervised by:

Dr. Kettani Driss

Table of Contents

I. Introduction:

After completing the requirement engineering phase, we initiated the design phase of the software engineering life cycle. In the paragraphs below we will illustrate the technologies that will be chosen and justify their choice. We will go over our IDE of choice, the programming language, and the database management system. Further, we will take a glimpse at the system architecture that we will employ, its description, and why it was used. Lastly, we will discuss the functions' interfaces and interactions in order to create the final version of the entity-relationship model, which will be adopted as a logical model.

II. Technology Enablers

When deliberating on the technology stack, we came to the agreement that our choice must be limited to free and open-source technologies that would minimize costs from our client's side. We sought to minimize the expenses that would arise from the maintenance of the software. In addition, our technology stack had to provide reasonable performance and an overall good experience for the user.

a. Programming language:

When it came to the back end of our solution, we decided to use Java as our main programming language. This choice was mainly due to the fact that the language offers flexibility in programming. Furthermore, the language components use separate model architecture, the Model-View-Controller paradigm (MVC) and thus can provide a much more flexible User Interface. The Model-View-Controller pattern provides clean separation between the application data/logic and the GUI controls. Along with this, Java provides the ability to create graphical user interface (GUI) components, such as buttons

and scroll bars, check boxes, labels, text areas that are independent of the windowing system for specific operating systems. Java offers the advantage of being platform-independent, which allows for the software to be deployed on any platform. This feature exists thanks to the java virtual machine.

b. Database Management System:

As for the DBMS, we settled on using PostgreSQL, a reliable and professional open-source DBMS that supports relational database management systems. It will be our choice for an enterprise information system that will allow us to retrieve and store information. Indeed, it is associated with all the fundamental features needed to manage our database system as object-relational features, extensibility, and security.

This DBMS, furthermore, allows java programs to connect to a PostgreSQL database using standard database independent java code.

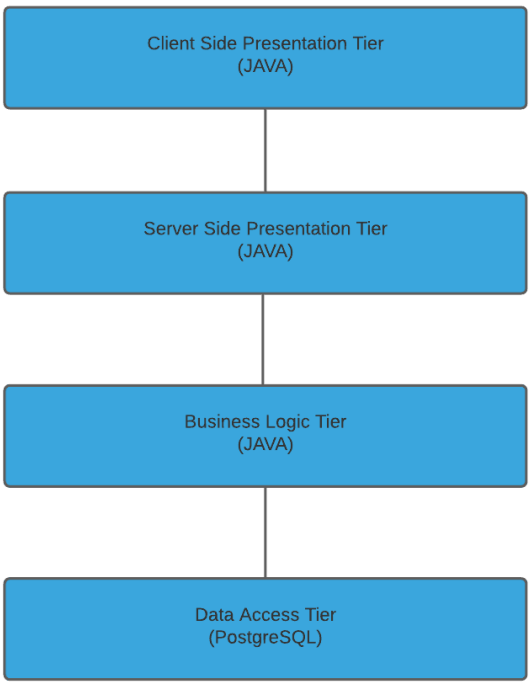
c. Integrated Development Environments:

There are many Integrated Development Environments (IDE) used by software developers to quickly develop applications utilizing IDE's features to design, debug, compile, test and deploy features of developed functionalities as expected.

For the integrated Development Environment, we will rely on Oracle's NetBeans IDE. It is a cross-platform IDE that works on all operating systems that support Java, i.e, Windows, Linux, Mac OSX, and BSD. The main purpose of using this IDE is the fact that it offers a productive development environment for Python and specifically for

Django development. NetBeans offers a development environment and framework to build software applications in Java and integrate with external tools and services.

III. System Architecture:

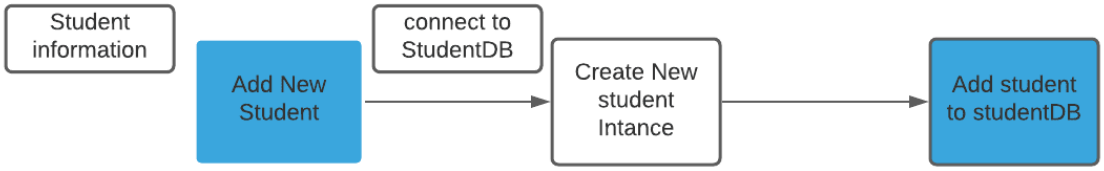


As shown in the illustration above, our system architecture relies on a 4-tiers architecture. The first tier is the Data Access Tier that will contain the PostgreSQL database. It includes all the lists and tables that will be utilized by our application, as well as provide data storage. This tier exists in an environment separate from the other tiers. Allowing for the preservation and protection of data. The 2nd tier contains the business logic of our system, which encompasses all Java models and functions used by our software. This tier is responsible for all the business logic along with all the protocols. The 3rd tier is the server-side presentation tier. This tier encompasses the server-side logic for formatting

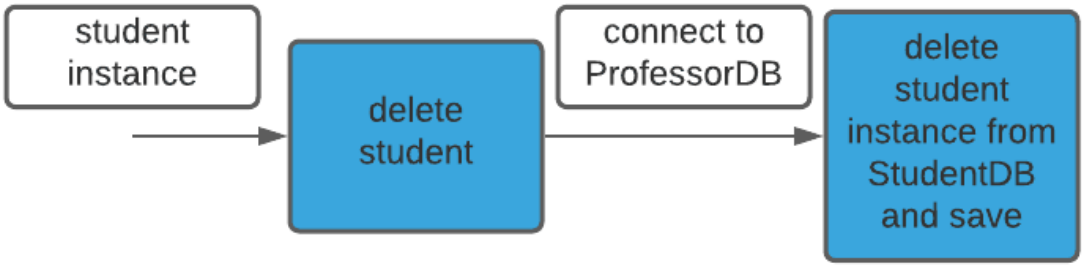
output, input correctness checking, forms, displaying some interactions if the user was logged in only, and so on. The 4th tier will include the forms, the buttons, the user interface, as well as user interaction components.

IV. Functions Interactions:

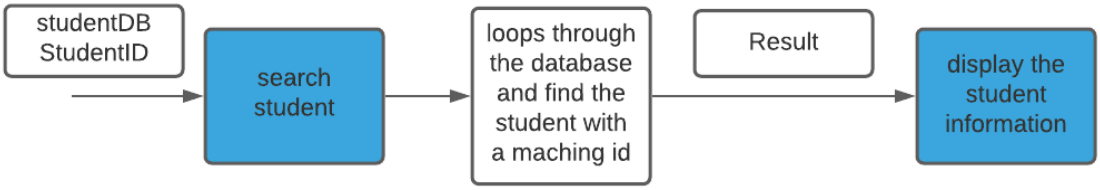
Add student:



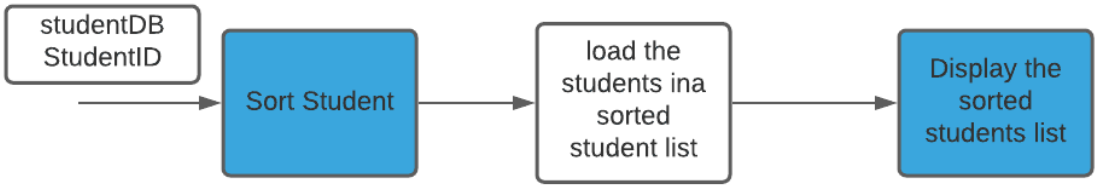
Delete student:



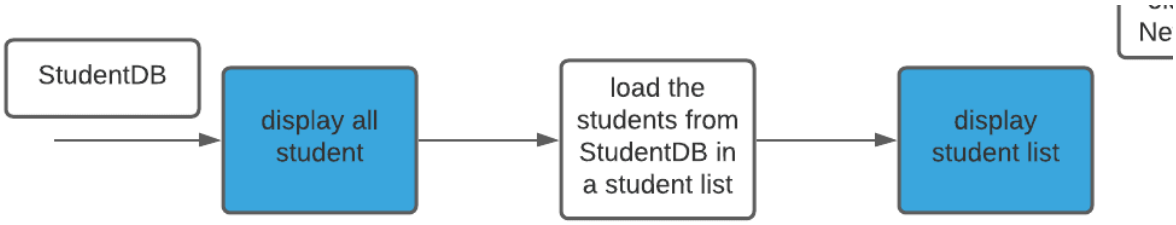
Search student:



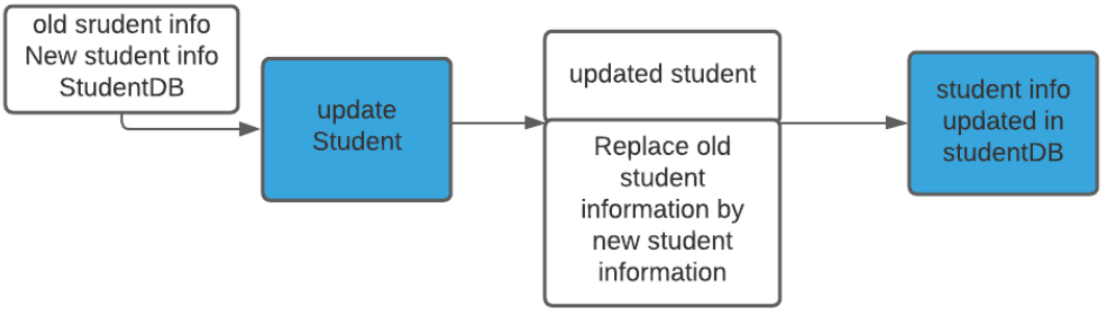
Sort student:



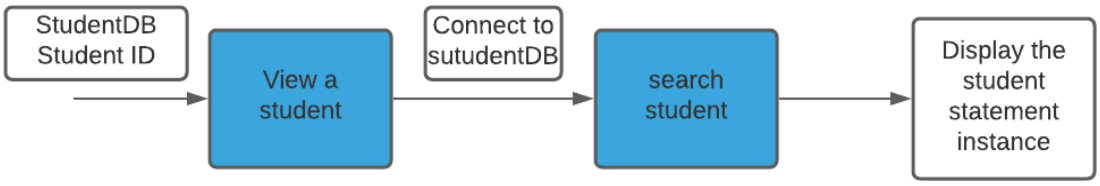
Display all student:



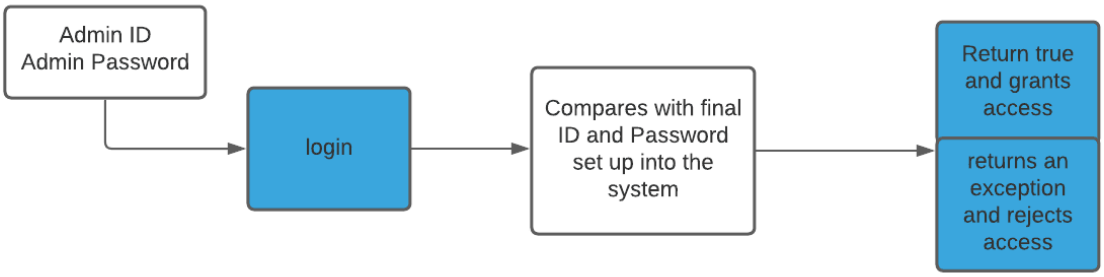
Update student:



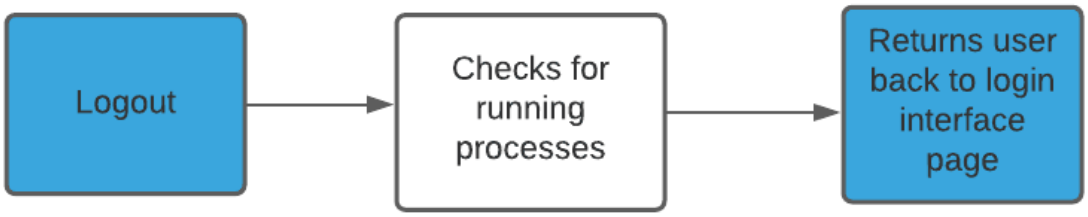
View Student:



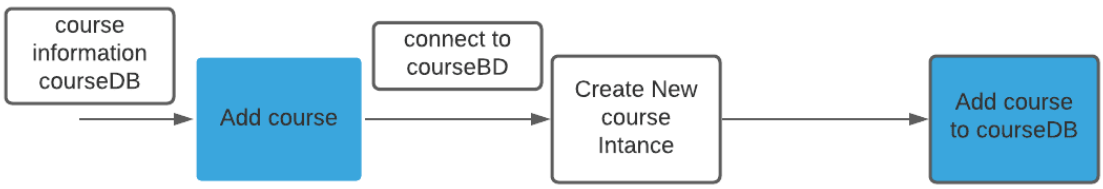
Login



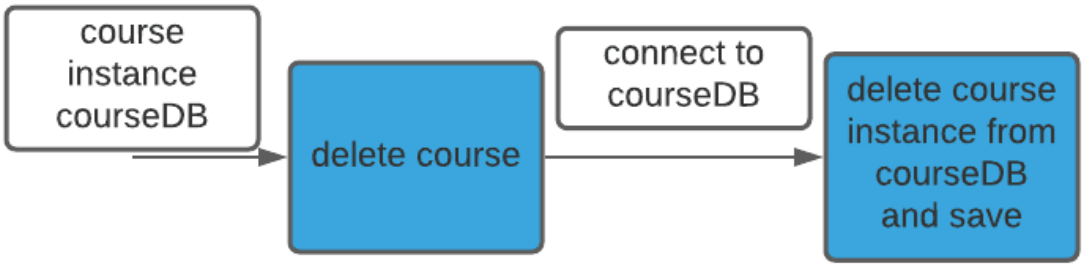
Logout



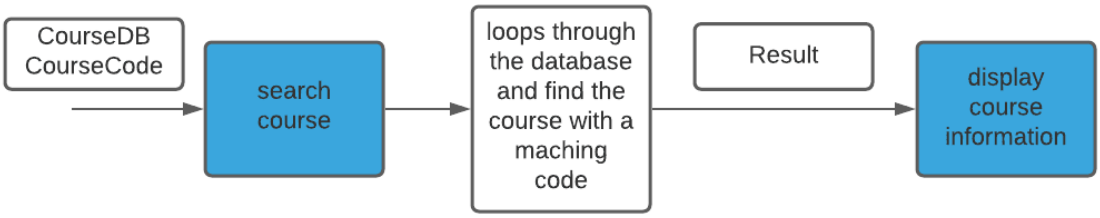
Add course:



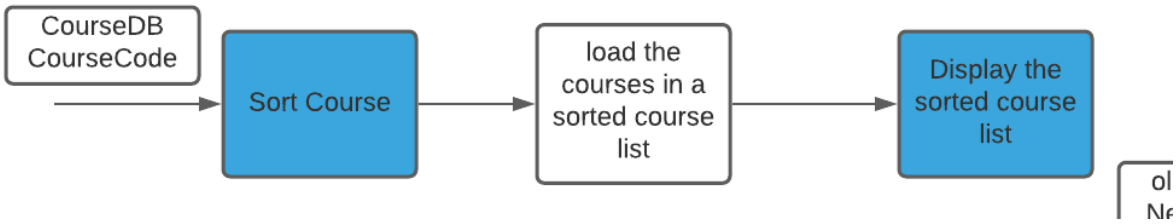
Delete course:



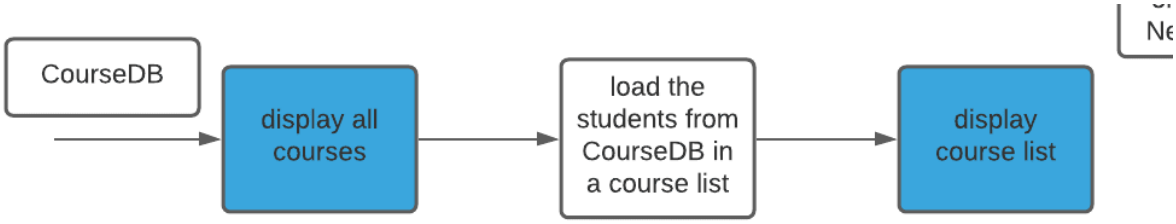
Search course:



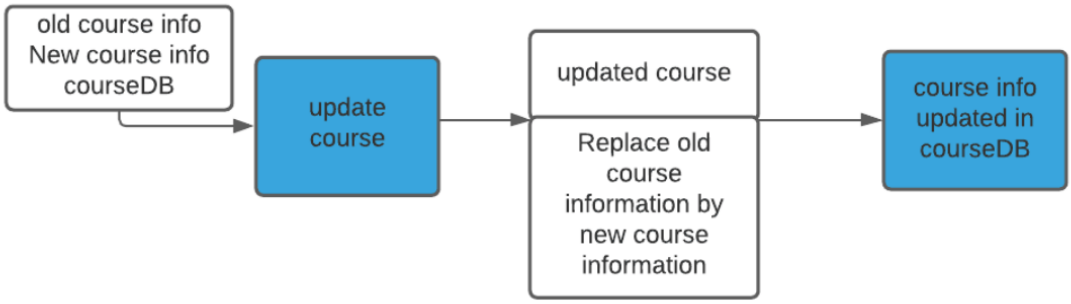
Sort course:



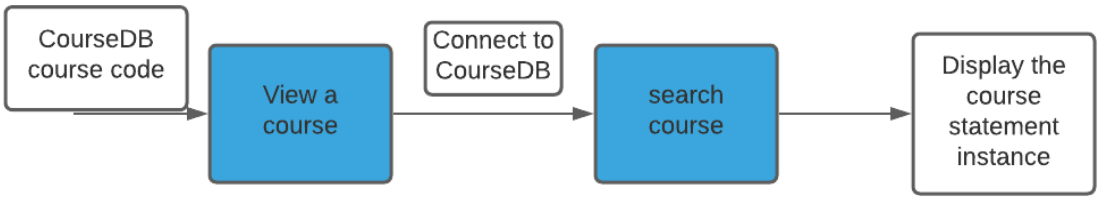
Display course:



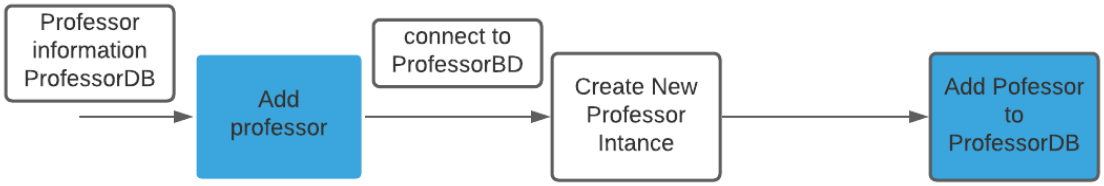
Update course:



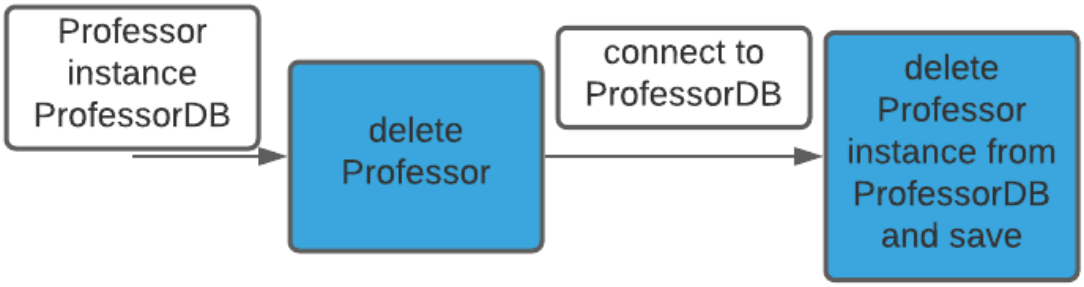
View course:



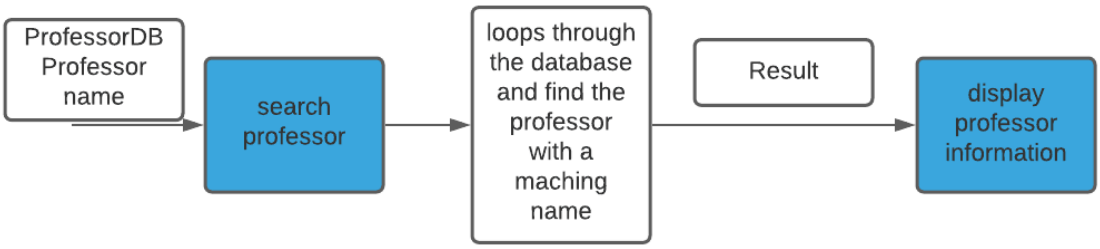
Add professor:



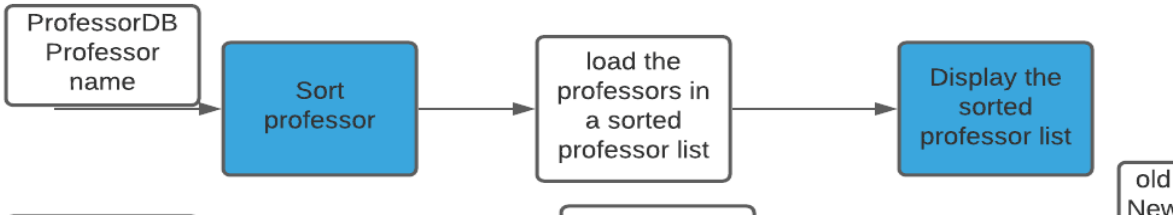
Delete professor:



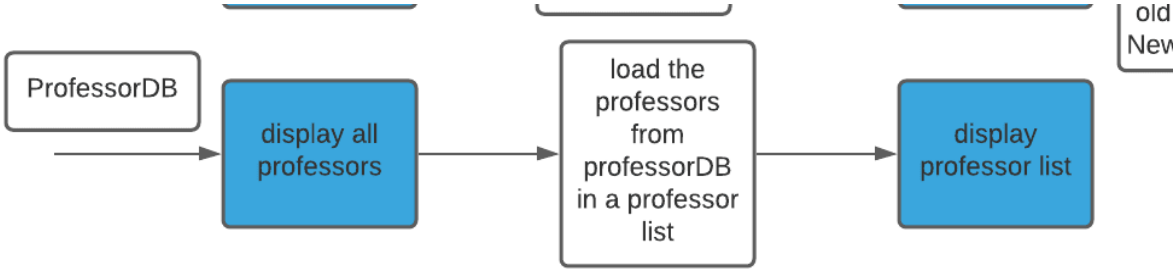
Search professor by name:



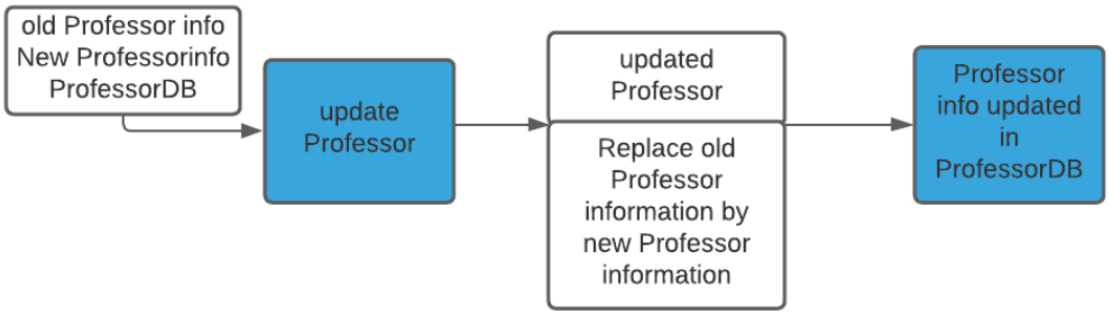
Sort professor by name:



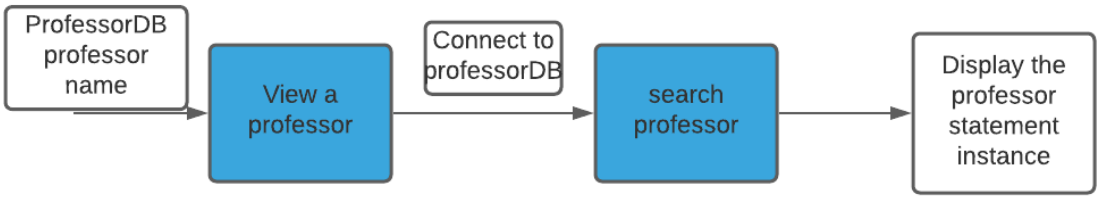
Display professor:



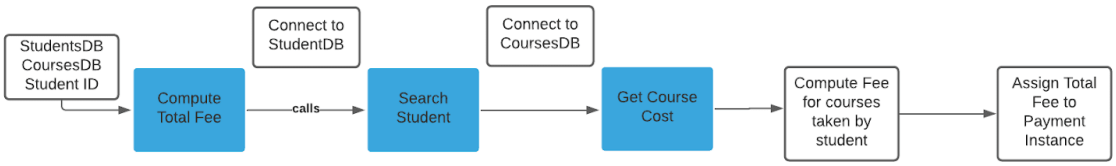
Update professor:



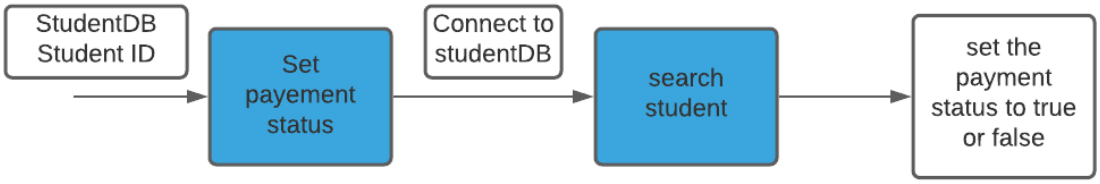
View professor:



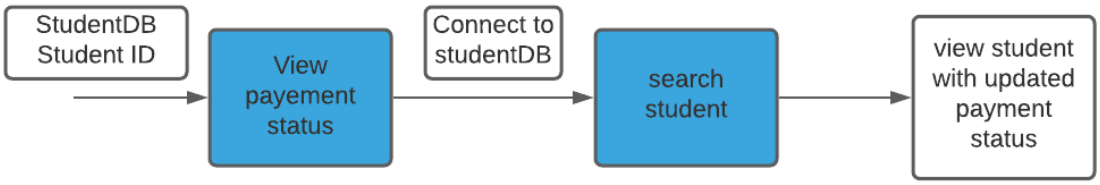
Compute Total Fee:



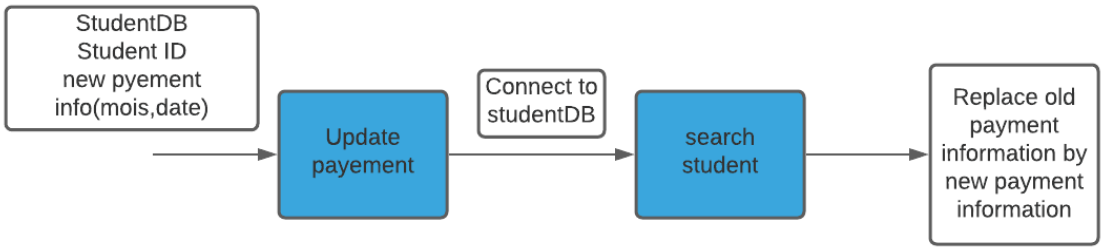
Set payment status:



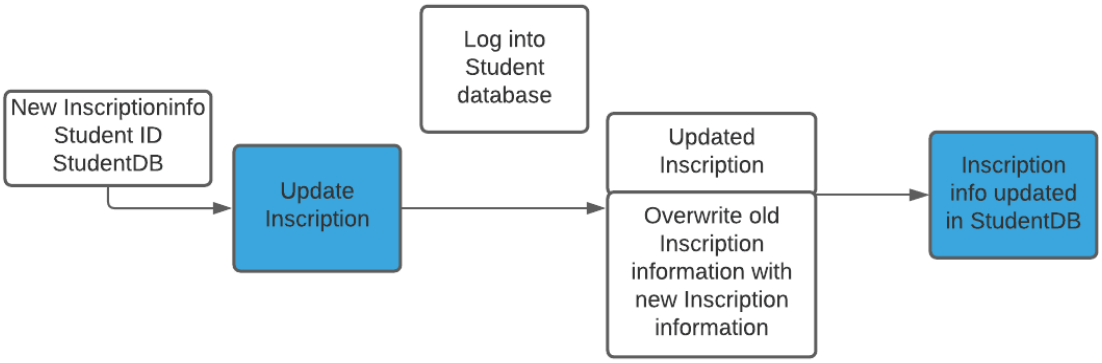
View payment status:



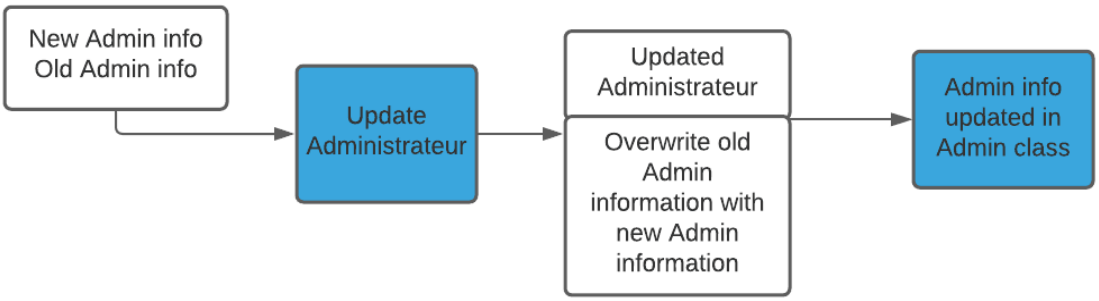
Update payment :



Update Inscription:



Update Administrateur:



V. Functions Stereotypes/Interfaces:

a. Manage Étudiant building block

Class Student{//These functions can only be accessed through the Student interface

public AddStudent(Student aStudent, DB studentsDB) returns boolean{

Takes an instance of Student then adds aStudent to the studentsDB database and returns a True if added successfully. Else it returns a False.

}

public DeleteStudent(Student aStudent, DB StudentsDB) returns boolean{

Takes an object of Student and removes it from the StudentsDB database and returns True when the operation is successful.

}

public SearchStudent(String studentName, DB StudentsDB) returns List{

Performs a search operation for the corresponding name of the Student. If there are students with a matching name or that start with the student's name they are appended to the list. Once the search is completed, it returns the list.

}

public SortStudents (DB StudentsDB){

Connects to the StudentsDB, adds the students to a list, then sorts them in an ascending order from oldest to newest. Lastly, it displays the list.

}

public DisplayAllStudents(DB StudentsDB){

Loads the students' database into a list of students that is then displayed into a table.

}

public UpdateStudent(Student Student, New information, DB StudentsDB) returns boolean{

Takes an instance of Student and applies the modifications to its attributes. Returns true when successful.

}

public ViewStudent(String StudentID, DB StudentsDB){

This function will display all the information about the student that has the given ID as a reference.

}

}

/***/

Manage Matière building block:

Class Course{ //These functions can only be accessed through the Course interface

public AddCourse(Course aCourse, DB coursesDB) returns boolean{

Takes an instance of Course then adds aCourse to the courseDB database and returns a True if added successfully. Else it returns a False

}

public DeleteCourse(Course aCourse, DB CoursesDB) returns boolean{

Takes an object of Course and removes it from the CoursesDB database and returns True when the operation is successful.

}

public SearchCourse(String courseName, DB CoursesDB) returns List{

Performs a search operation for the corresponding name of the Course. If there are courses with a matching name or that start with the course name, they are appended to the list. Once the search is completed, it returns the list.

}

public SortCourses (DB CoursesDB,CourseCode){

Connects to the CoursessDB, adds the courses to a list, then sorts them in an ascending order from oldest to newest. Lastly, it displays the list.

}

public DisplayAllCourses(DB CoursesDB){

Loads the courses' database into a list of courses that is then displayed into a table.

}

public UpdateCourse(Student Course, New information, DB CoursesDB) returns boolean{

Takes an instance of Course and applies the modifications to its attributes. Returns true when successful

}

public ViewCourse(String CourseID, DB CourseDB){

This function will display all the information about the course that has the given ID as a reference.

}

}

/***/

Manage Professeur building block:

Class Professor{ //These functions can only be accessed through the Professor interface

public AddProfessor(Professor aProfessor, DB ProfessorsDB) returns boolean{

Takes an instance of Professor then adds aProfessor to the ProfessorsDB database and returns a True if added successfully. Else it returns a False

}

public DeleteProfessor(Professor aProfessor, DB ProfessorsDB) returns boolean{

Takes an object of Professor and removes it from the ProfessorsDB database and returns True when the operation is successful.

}

public SearchProfessor(String professorName, DB ProfessorsDB) returns List{

Performs a search operation for the corresponding name of the Professor. If there are professors with a matching name or that start with the professor's name, they are appended to the list. Once the search is completed, it returns the list.

}

public SortProfessors(DB ProfessorsDB){

Connects to the ProfessorsDB, adds the professors to a list, then sorts them in an ascending order from oldest to newest. Lastly, it displays the list.

}

public DisplayProfessor(DB ProfessorsDB){

Loads the professors' database into a list of professors that is then displayed into a table.

}

public UpdateProfessor(Student Professor, New information, DB ProfessorsDB) returns boolean{

Takes an instance of Professor and applies the modifications to its attributes. Returns true when successful

}

public ViewProfessors(String ProfessorID, DB ProfessorsDB){

This function will display all the information about the professor that has the given ID as a reference.

}

/**/

Manage Inscription building block:

public updateInscription(Student ID, DB StudentsDB){

Searches for a student using his ID in StudentsDB, accesses his associated payment instance, then overwrites old attributes with updated ones

}

/**/

Manage Administration building block:

public Login(ID, Password) returns boolean{

Checks if entered login information matches with the ones set up into the system. Returns true then grants access if matching, else returns an exception

}

public Logout() returns{

Checks for running processes then returns the user back to the login page interface.

}

public UpdateAdmin(Administrateur Admin, newInfo) returns{

Overwrites old admin info with new ones.

}

/**/

Manage Payment building block:

public ComputeTotalFee(DB studentsDB, DB coursesDB, StudentID) returns totalFee{

Computes the total fee for a student using the course list the student is taking and the price for each course

}

public setPaymentStatus(StudentsDB, Student ID){

Search for the student using the searchStudent method, which takes his ID, then sets his paymentStatus to either true or false.

}

public viewPaymentStatus(DB StudentsDB, Student ID) returns paymentStatus{

Search for a student using the searchStudent, which has access to StudentsDB method then displays his payment status

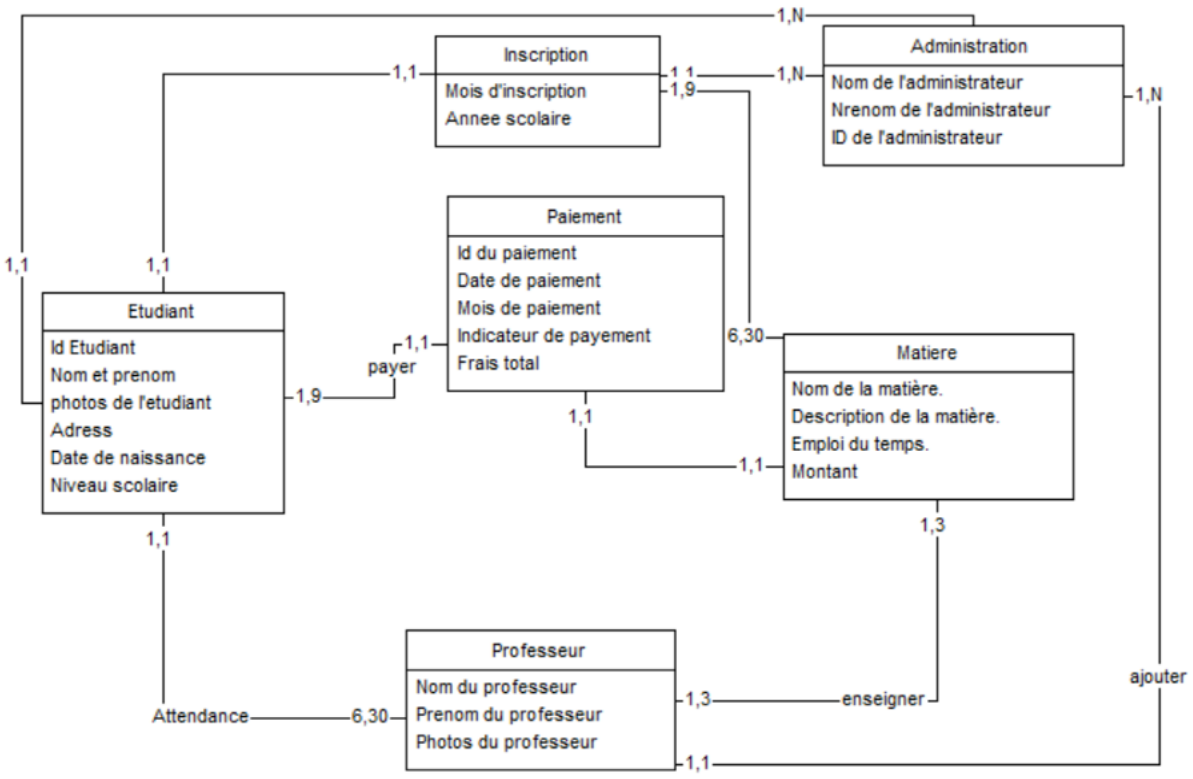
}

public updatePayment(DB StudentsDB, Student ID, newMonth, newDate){

Searches for a student using his ID in StudentsDB, then overwrites old attributes with updated ones

}

VI. Entity Relationship Diagram



VII. Conclusion:

By building on the foundation laid by the requirements engineering phase of our process, we were able to transition smoothly and effectively into the design part of our project. Indeed,

utilizing the entity-relationship model established in the previous step and concretizing it was our next objective. In the design step, we moved to digitalize our paper-based solutions. We defined our technology stack, as well as the physical architecture where our functions will be implemented. Then we described the function interactions through the Data flow diagrams to comprehend the steps that each method goes through. Lastly, we defined the function prototypes. They shall serve as a high-level template to clarify the functioning of each method that will be implemented in the following phase. All in all, the design phase has allowed us to further elaborate on the functioning of our system. We are now ready to initiate the development and implementation of our system.