Università di Pisa

Department of Information Engineering

Information Systems
Task 0 Documentation

Students:
Adriano Botti
Antonio Le Caldare
Francesco Merola
Giacomo Ponziani

Contents

List of Figures

Application Specifications

The goal of the application that we implemented is to provide a way for both students and professors to manage the registration process for exams. More specifically, we want the application to exert the following functionalities:

- For Students:
  1. Check past exams results
  2. Register to an exam date
  3. Delete an exam registration
- For Professors:
  1. Add grades to an exam
  2. Create a new exam date

The application is realized using the Java language, with the JavaFX extension to manage a graphic interface. The back-end uses a MySQL database to store the information.

Requirements and Use Cases

Functional Requirements

The Professor:

1. shall be able to insert an exam, associated with a course he holds, in a date of choice
2. shall not be able to inster an exam for a date precedent to the current date
3. shall be able to insert the corresponding grade for a student in his registration for that exam.
4. shall insert all grades in the exact date of the exam

The Student:

1. shall be able to check the results of past exams
2. shall be able to register to an exam not yet took
3. shall not be able to register to an exam after the exam date.
4. shall be able to register to more successive exams for the same course
5. If the student registered to successive exams for the same course he just got a mark for, then those future registrations shall be deleted
6. shall be able to deregister from an exam he was previously registered to
7. shall not be able to deregister from an exam alredy took
8. shall not be able to deregister from an exam after the exam date.

Non-Functional Requirements

For the application we identified Consistency and Availability as the two most important non-functional requirements. Both the requirements can be satisfied by the use of a RDBMS, since for our application we don't manage high volumes of data. For this reason we employed a MySQL DBMS to implement our back-end

Use Case Diagram

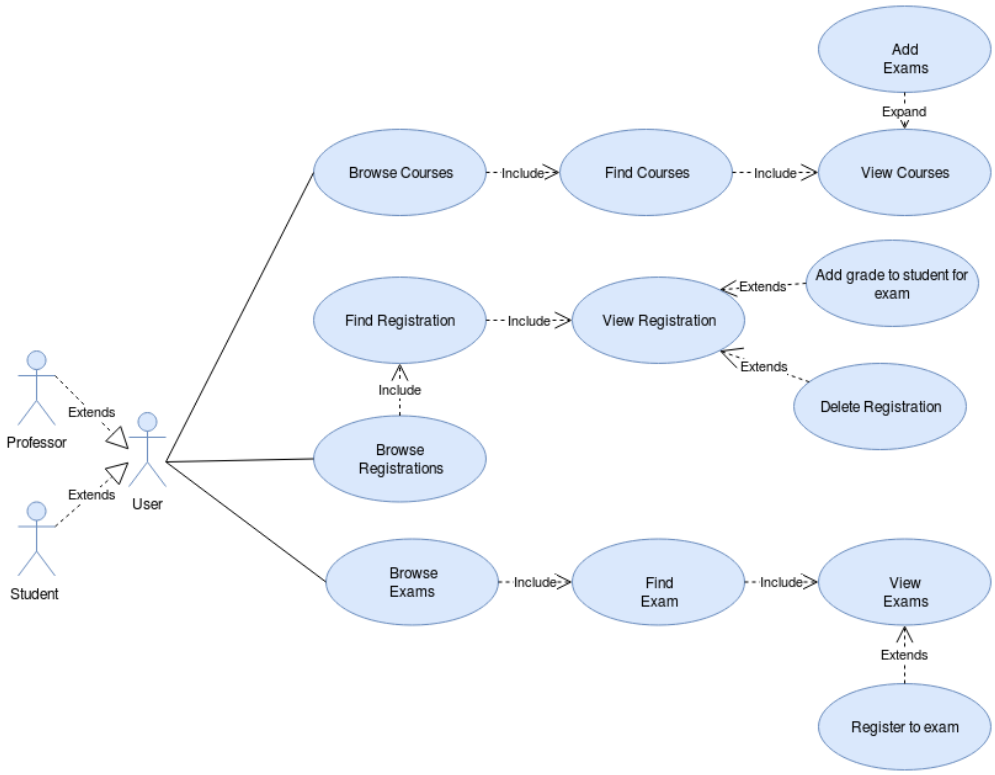

Figure 1: Use Case Diagram

- **Add Exam** : This operation can only be performed by a Professor.
  He/She will be able to browse his/her tenured courses, select the one to add a new exam to and then add it.
- **Add Grade** : This operation can only be performed by a Professor.
  He/She will be able to browse the registrations for his/her tenured courses, select the registration to add a grade to and then add it.
- **Register to Exam** : This operation can only be performed by a Student.
  He/She will be able to browse all the exams to which is possible to register, select one and perform the registration.
- **Deregister to Exam** : This operation can only be performed by a Student.
  He/She will be able to browse his/her active registrations, select one and deregister from it.
- **See Grades** : This operation can only be performed by a Student.
  He/She will be able to browse his/her registrations and show only the one with a sufficient.

For more detail and a step-by-step description of the different scenarios, see chapter *User's Manual*.

UML Class and Entity-Relationship Diagrams

UML Class Diagram

The UML Class Diagram shows the main objects involved in our application, with the respective logical relationships. From this model we derived an Entity-Relationship Diagram, where the

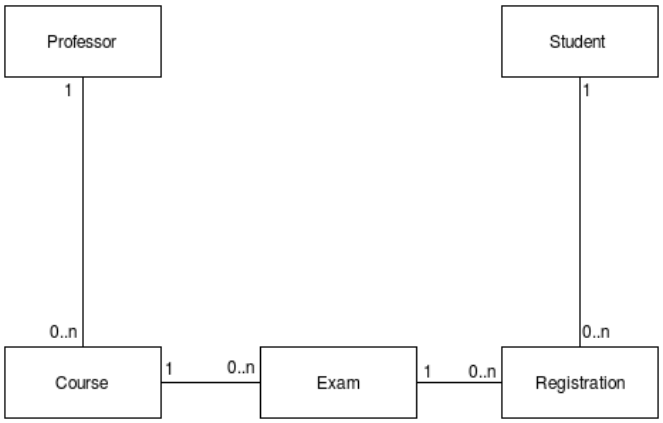

Figure 2: UML Class Diagram

dependecies are more deeply identified. Finally we will show the final database schema implemented on the MySQL RDBMS.

ER Diagram

Names definition

Here we define in detail the terms for the main entites and relationships we will use in the following:

- *Student*
  A student is an entity which is able to perform the operations already defined in the requirements. He's an actor for our application.
- *Professor*
  A professor is an entity which is able to perform the operations already defined in the requirements. He's an actor for our application.
- *Course*
  A course is held by one and only one professor. The course object only includes information about its name, cfu and the helding professor. It holds no information about when exams for that course will take place.
- *Exam*
  An exam represents the actual date of the examination for a course.
- *Exam Result* An exam result relates an exam to all the students who registered to that exam, adding the information of the grade, if meaningful.

Entities

Entity	Description	Attributes
Student	Holds all the information related to the students	• $\underline{id}$ • $name$ • $surname$
Professor	Holds all the information related to the professors	• $\underline{id}$ • $name$ • $surname$
Course	Holds the information related to the courses	• $\underline{id}$ • $name$ • cfu • $professor$
Exam	Holds all the new and past exams	• $\underline{course(ext)}$ • $\underline{date}$

Relationships

Relationship	Description	Participants	Attributes
Teaching	Links Professors to their held courses	• Professor(1,N) • Course(1,1)	
Exam Result	Links students to exams, with the respective grade	• Student(0,N) • Exam(0,N)	• $grade$
Exam Date Creation	Links the courses to the exams through a date	• Course(0,N) • Exam(1,1)	

ER Diagram

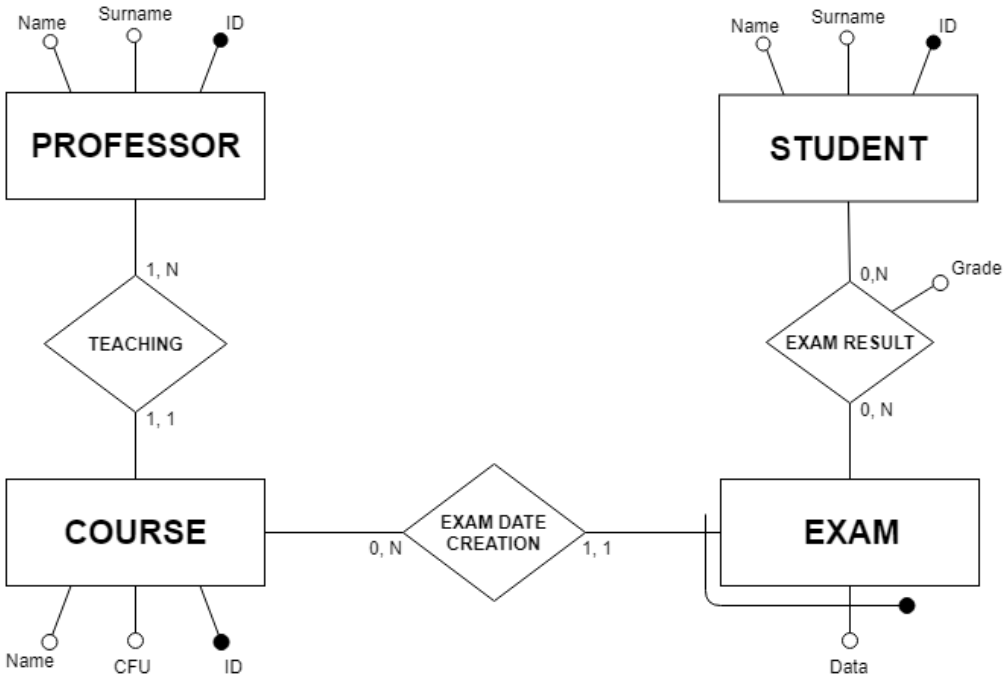

Figure 3: Entity - Relationship Diagram

Database Implementation

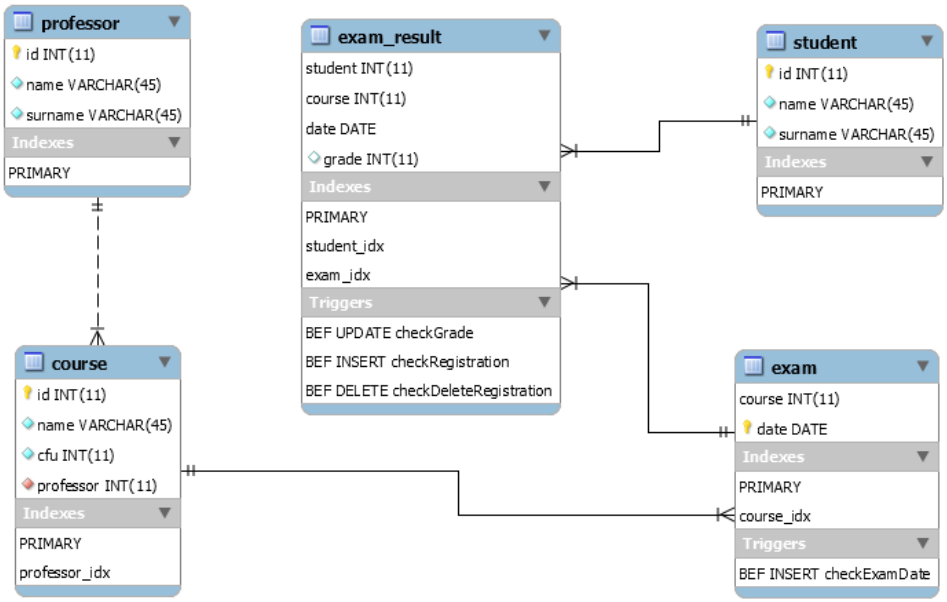

Figure 4: MySQL Schema

User's Manual

The application starts and shows a minimal user interface, fig. 5, composed of:

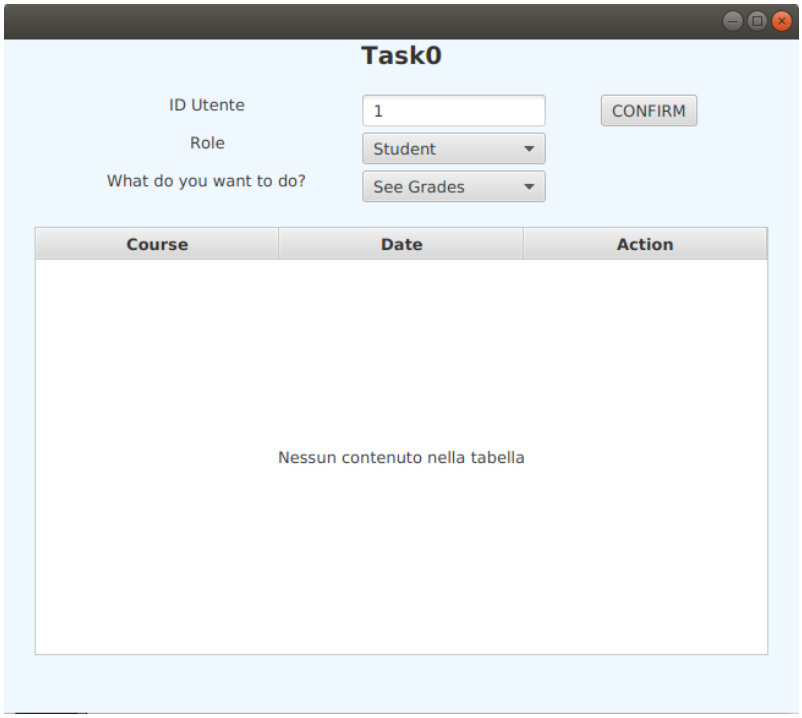

Figure 5: User Interface when the application starts

- *UserID* field, where the user specifies his id number;
- *Role* choise box, whith the option:
  1. Professor, in case the user is a Professor;
  2. Student, in case the user is a Student.
- *What do you want to do?* choise box, where the user specifies the action he wants to perform. The options change according to the role chosen. Professor can select among:
  1. *Add Exam* if he wants to add an exam;
  2. *Add Grade* if he wants to add a grade.

  A Student can select among:
  1. *Register to Exam* if he wants to register to an exam;
  2. *Deregister to Exam* if he wants to deregister to an exam;
  3. *See Grades* if he wants to see the grades he got.
- *Confirm* button;
- A table showing the results of the selected operation. The layout of the table can change according depending on the selected operation.

Professor

A Professor:

1. Inserts his ID number into the *UserID* field;
2. Selects *Professor* in the *Role* choise box;
3. Selects the action he wants to perform;
4. Pushes the *Confirm* button.

Add Exam

If the Professor select the *Add Exam* option, the table, fig. 6, is populated with a list of all the courses he is currently holding. Each element of the list is composed of:

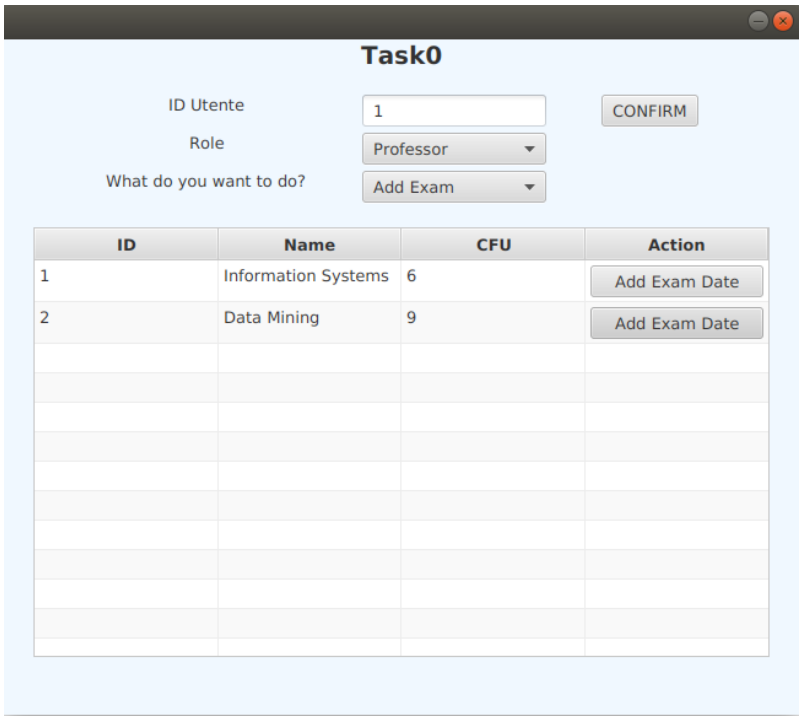

Figure 6: Example of Add Exam

- *ID*, the id of the course;
- *Name*, the name of the course;
- *CFU*, the number of credits assigned to the course;
- *Add Exam Date*, button the professor has to push in order to add an exam corresponding to the course.

If the Professor pushes one of the *Add Exam Date* buttons a confirm dialog is presented and it asks for the date of the exam to insert, fig. **??**. If the Professor wants to confirm he:

1. Selects the date using the datepicker;
2. Pushes the *Confirm* button in the dialog.

To go back and undo the operation the Professor just pushes the *Delete* button.

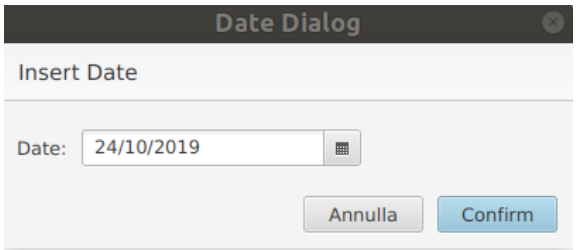

Figure 7: Example of Add Exam dialog

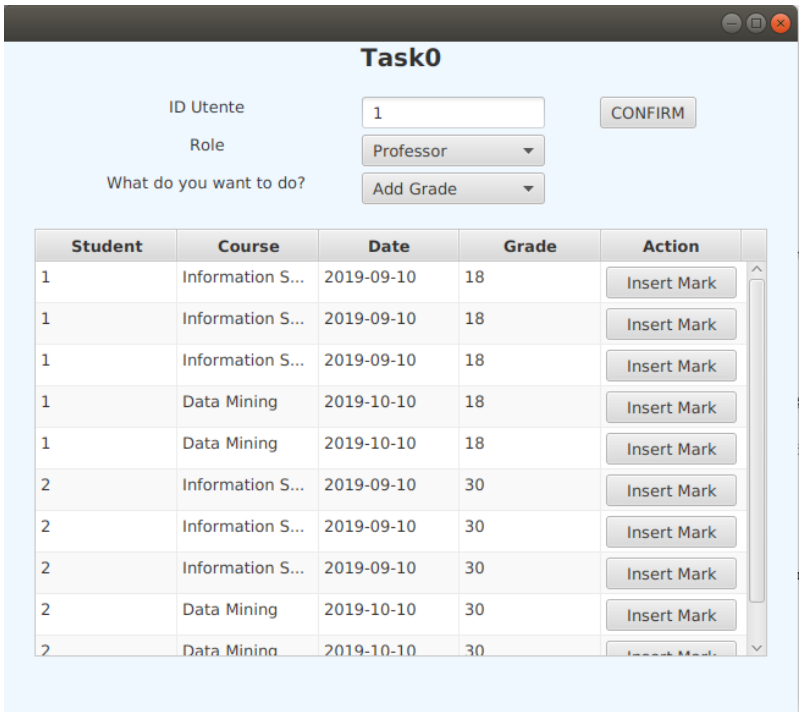

Figure 8: Example of Add Grade

Add Grade

If the Professor select the *Add Grade* option, the table, fig. 8, is populated with a list of all the registrations to the courses he is currently holding. Each element of the list is composed of:

- *Student*, the id of the student enrolled to the exam;
- *Course*, the name of the course;
- *Date*, the date of the exam;
- *Insert Mark* button that the professor has to push in order to insert a grade to the corresponding registration.

If the Professor pushes one of the *Insert mark* buttons a confirm dialog is presented and it asks for the grade of the exam to insert, fig. 9. If the Professor wants to confirm he:

1. Inserts the grade in the corresponding field;
2. Pushes the *Confirm* button in the dialog.

To go back and undo the operation the Professor just pushes the *Delete* button.

Student

A Student:

1. Inserts his ID number in the *UserID* field;

Figure 9: Example of Add Grade dialog

2. Selects *Student* in the *Role* choise box;
3. Selects the action he wants to perform;
4. Pushes the *Confirm* button.

Register to Exam

If the Student select the *Register to Exam* option, the table, fig. 10, is populated with a list of all the available exams. Each element of the list is composed of:

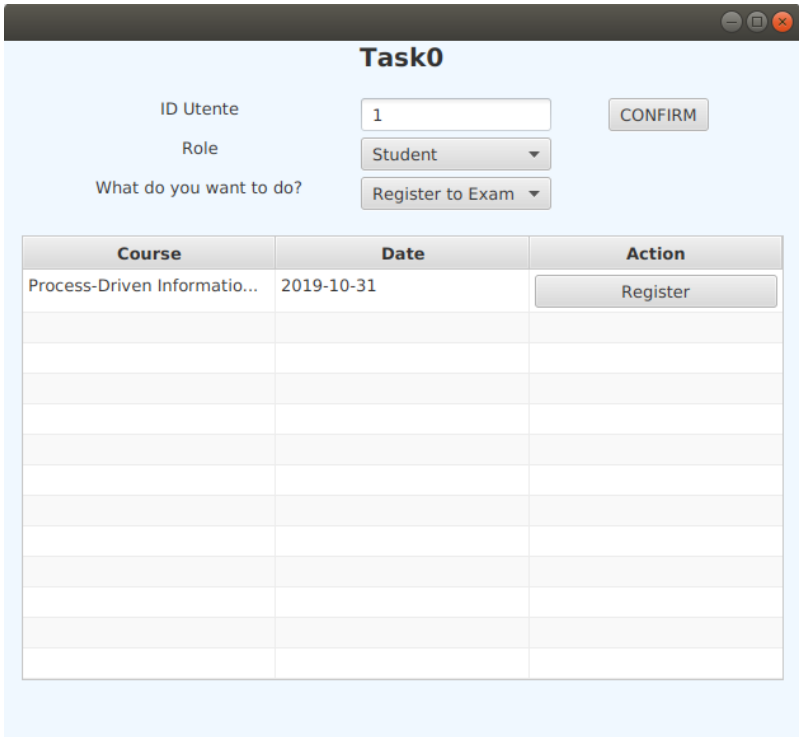

Figure 10: Example of Register to Exam

- *Course*, the name of the course;
- *Date*, the date of the exam;
- *Register*, button the student has to push in order to register to the selected exam.

If the Student pushes one of the *Register to Exam* the table is updated so that it shows all the exams to which the Student is not enrolled.

Deregister to Exam

If the Student select the *Deregister* option, the table, fig. 11, is populated with a list of all the registrations corresponding to the Student. Each element of the list is composed of:

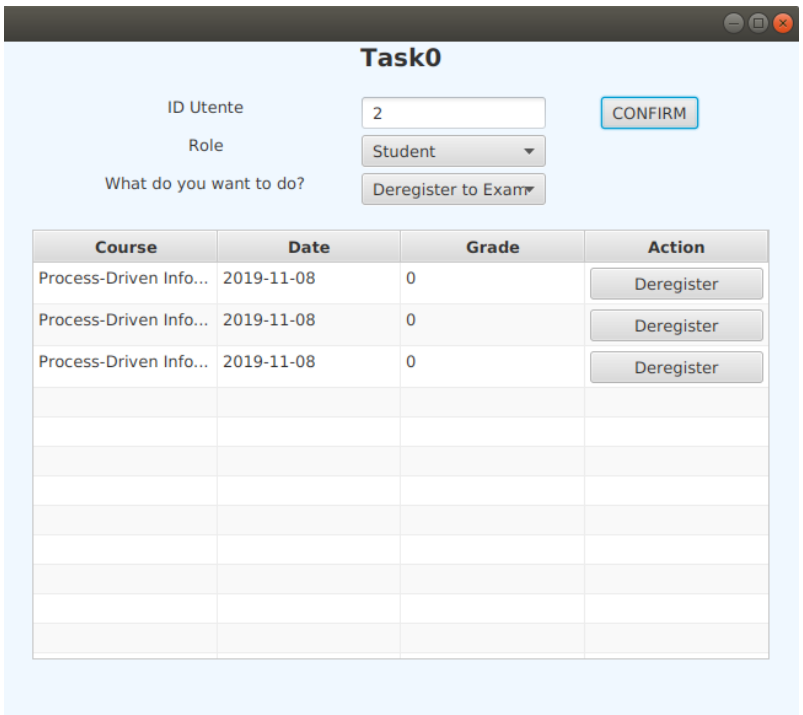

Figure 11: Example of Deregister to Exam

- *Course*, the name of the course of the exam the Student is enrolled to;
- *Date*, the date of the exam;
- *Deregister* button that the Stuedent has to push in order to do the deregistration.

If the Professor pushes one of the *Deregister* the table is updated so that it shows all the exams to which the Student is not enrolled.

See Grades

If the Student select the *See Grades* option, the table, fig. 12, is populated with a list of all the exams the student has done. Each element of the list is composed of:

- *Course*, the name of the course;
- *Date*, the date when the student passed the exam;
- *Grade*, the grade the Student got.

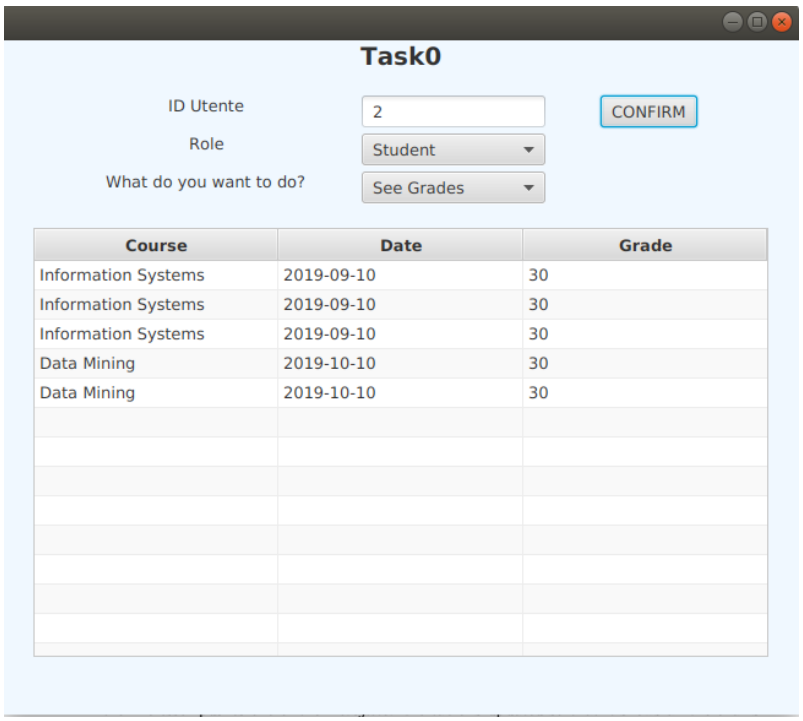

Figure 12: Example of See Grades