CoursesManagementApp

Sprint Report

AFRODITI VAMVA, A.M:4604

XRISTINA MAI, A.M:4612

SOFIA MOISIADOU, A.M:4446

VERSIONS HISTORY

Date	Version	Description	Author
21/03/2022	Version 1	We created the server and the database for the project	Sofia Moisiadou
27/03/2022	Version 2	We used Spring Initializr to create a maven project	Afroditi Vamva
29/03/2022	Version 3	We added all the necessary dependencies for our project	Xristina Mai
31/03/2022	Version 4	We connected the database into the project using the jpa-connector	Sofia Moisiadou
3/04/2022	Version 5	We created the application using the localhost and the hello.html	Afroditi Vamva
5/04/2022	Version 6	We created the Login Application for the user to sign in	Sofia Moisiadou
10/04/2022	Version 7	We created the all the necessary classes for the courses	Xristina Mai
16/04/2022	Version 8	We created the all the necessary classes for the student	Sofia Moisiadou
23/0342022	Version 9	We created the all the necessary classes for the grade calculation	Xristina Mai
3/05/2022	Version 10	We created the html files and added all the redirect buttons	Afroditi Vamva
15/05/2022	Version 11	We created the Junit and Mockito tests	Afroditi Vamva
17/05/2022	Version 12	We composed the pdf report and filmed the video	The whole team
18/05/2022	Version 13	We send the project	The whole team

1 Introduction

The objective of this project is to develop a Web application that allows an instructor to manage the courses that he teaches.

Purpose

The purpose of the application is the instructor being able to manage all the details including the names of the students that enrolled in the courses that he teaches their grades, semester and id.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	A.Zarras
Scrum Master	Sofia Moisiadou
Development Team	Afroditi Vamva, Xristina Mai, Sofia Moisiadou

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**5/04/2022**	**9/04/2022**	**4 days**	To login to the application with my user name and password.
2	**10/04/2022**	**13/04/2022**	**3 days**	To browse the list of my courses.
3	**14/04/2022**	**15/04/2022**	**1 day**	To add a course in the list, by giving information like the course id, name, syllabus, year, semester, etc.
4	**14/04/2022**	**14/04/2022**	**1 day**	To remove a course from the list.
5	**15/04/2022**	**15/04/2022**	**1 day**	To update the description of a course.
6	**16/04/2022**	**19/04/2022**	**3 days**	To browse the list of students that enrolled to a particular course.
7	**20/04/2022**	**21/04/2022**	**1 days**	To add a student to the list of a particular course, by giving information like the student id, name, year of registration, semester, etc.
8	**22/04/2022**	**22/04/2022**	**1 days**	To remove a student from the list of a particular course.
9	**22/04/2022**	**22/04/2022**	**1 days**	To update students' information (id, name, year of studies, etc.).
10	**23/04/2022**	**25/04/2022**	**2 days**	To register the grades of the student in the final exam and the project of the course.
11	**26/04/2022**	**28/04/2022**	**2 days**	To calculate the overall grades of the students that enrolled in a particular course with respected to a weighted average.
12	**29/04/2022**	**1/05/2022**	**2 days**	To calculate descriptive statistics about the students grades in a particular course ,including min, max, mean, standard deviation, variance, percentiles, skewness, kurtosis, median.

3 Use Cases

3.1 <Use Case 1>

Use case ID	1
Actors	Instructor
Pre conditions	The LoginApplication running and the instructor connected to localhost:8080 in his browser.
Main flow of events	1. The use case starts when the user inserts his password and username. 2. He is directed to the index page 3. He presses the button that directs him to the main menu
Alternative flow 1	1.The instructor provides wrong username and/or password, a message “Bad Credentials” shows up and he doesn’t have access to the app.
Post conditions	To safely manage the courses that the instructor teaches

3.2 <Use Case 2>

Use case ID	2
Actors	Instructor
Pre conditions	1.The instructor to be connected to the application 2.To browse the list of my courses.
Main flow of events	1. The use case starts when the instructor is connected to the application 2. He is in the main menu 3.He clicks the link that shows the all the courses that he teaches
Post conditions	To manage the courses descriptions and the students' grades.

3.3<Use Case 3>

Use case ID	3
Actors	Instructor
Pre conditions	The instructor needs to be logged in , in order to be able to add a course
Main flow of events	The use case starts when the instructor presses the button “add course ” then the page redirects him to another html where he can add a course with all the credentials
Alternative flow 1	-
Post conditions	To populate the list of the courses with new ones

3.4<Use Case 4>

Use case ID	4
Actors	Instructor
Pre conditions	There must be a course on the list
Main flow of events	The use case starts when the instructor presses the button “delete” , then a window appears to reinsure that the instructor wants to delete the course . if he presses the “ok” button then the course gets deleted
Alternative flow 1	In case the instructor doesn’t want to delete the course they can press the “cancel” button to cancel the action
Post conditions	Clean up the list of courses.

3.5<Use Case 5>

Use case ID	5
Actors	Instructor
Pre conditions	There must be a course on the list in order to update it
Main flow of events	The use case starts when the instructor presses the button “Update”, then the page redirects him to an html where he can update the course’s information
Alternative flow 1	-
Post conditions	Correct possible mistakes and keep the information up to date with the Current situation Correct possible mistakes and keep the information up to date with the current situation

3.6<Use Case 6>

Use case ID	6
Actors	Instructor
Pre conditions	The course needs to have students that enrolled into this particular course
Main flow of events	The use case starts when the instructor presses the button “students” then he gets redirected to another page where he can the browse through the list of students
Alternative flow 1	If no students have enrolled in the particular course there will not be a list for the instructor to browse through
Post conditions	To manage the students that enrolled in the course

3.7<Use Case 7>

Use case ID	7
Actors	Instructor
Pre conditions	There needs to be a course to be able to add a student
Main flow of events	The use case starts when the instructor presses on the button "student " , after the redirection the instructor should press the button "Add Student " to get redirected again into the student form so he can add a student to the course
Alternative flow 1	-
Post conditions	To populate the list with the students that enrolled in the course

3.8<Use Case 8>

Use case ID	8
Actors	Instructor
Pre conditions	There needs to be a student enrolled into the course
Main flow of events	The use case starts when the instructor presses the button "Delete" that exists into the student list " ,then a window appears to reinsure that the instructor wants to delete the course . if he presses the "ok" button then the course gets deleted
Alternative flow 1	In case the instructor doesn't want to delete the student they can press the "cancel" button to cancel the action
Post conditions	To deal with students that resigned from the course

3.9<Use Case 9>

Use case ID	9
Actors	Instructor
Pre conditions	There needs to be a student into the list in order to update the information
Main flow of events	The use case starts when the instructor presses the button "Update" that exists into the student list , then the page redirects him to an html where he can update the course's information
Alternative flow 1	-
Post conditions	Correct possible mistakes and keep the information up to date with the current situation

3.10<Use Case 10>

Use case ID	10
Actors	Instructor
Pre conditions	There need to be a student into the course
Main flow of events	The use case starts when the instructor chooses to add grades for the students who is enrolled into the course
Alternative flow 1	-
Post conditions	To manage the grading of the students that enrolled in the course

3.11<Use Case 11>

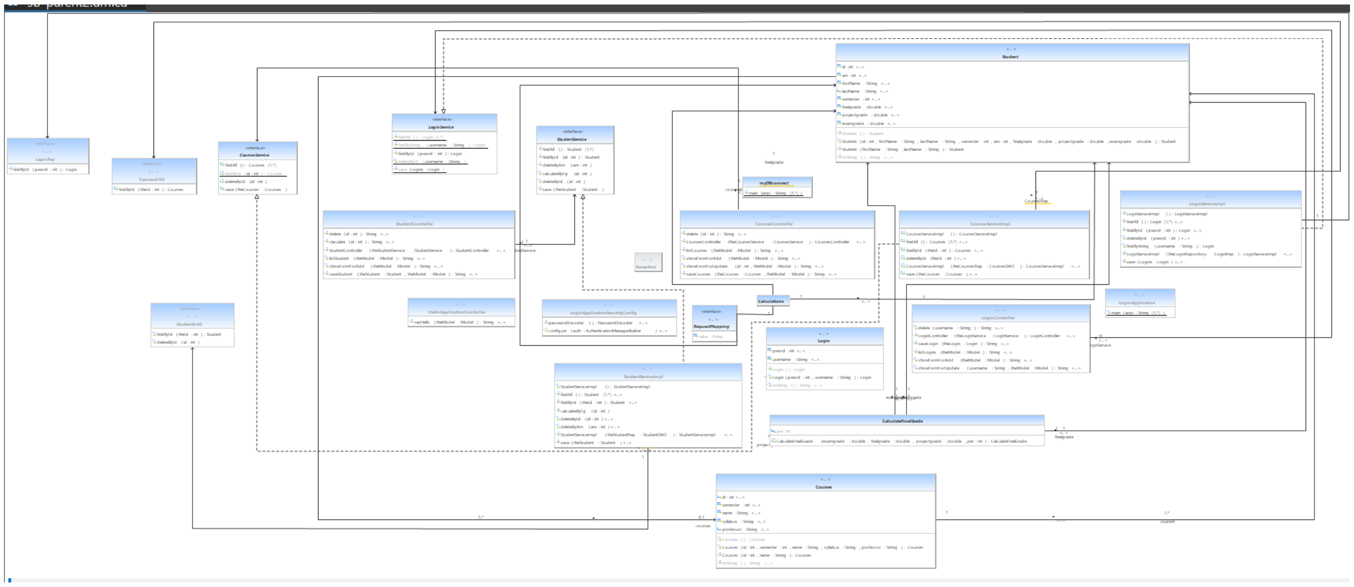
Use case ID	11
Actors	Instructor
Pre conditions	There has to be a student enrolled into the course
Main flow of events	The use case starts when the instructor adds grades for a student to calculate the final grade
Post conditions	To manage the grading of the students that enrolled in the course

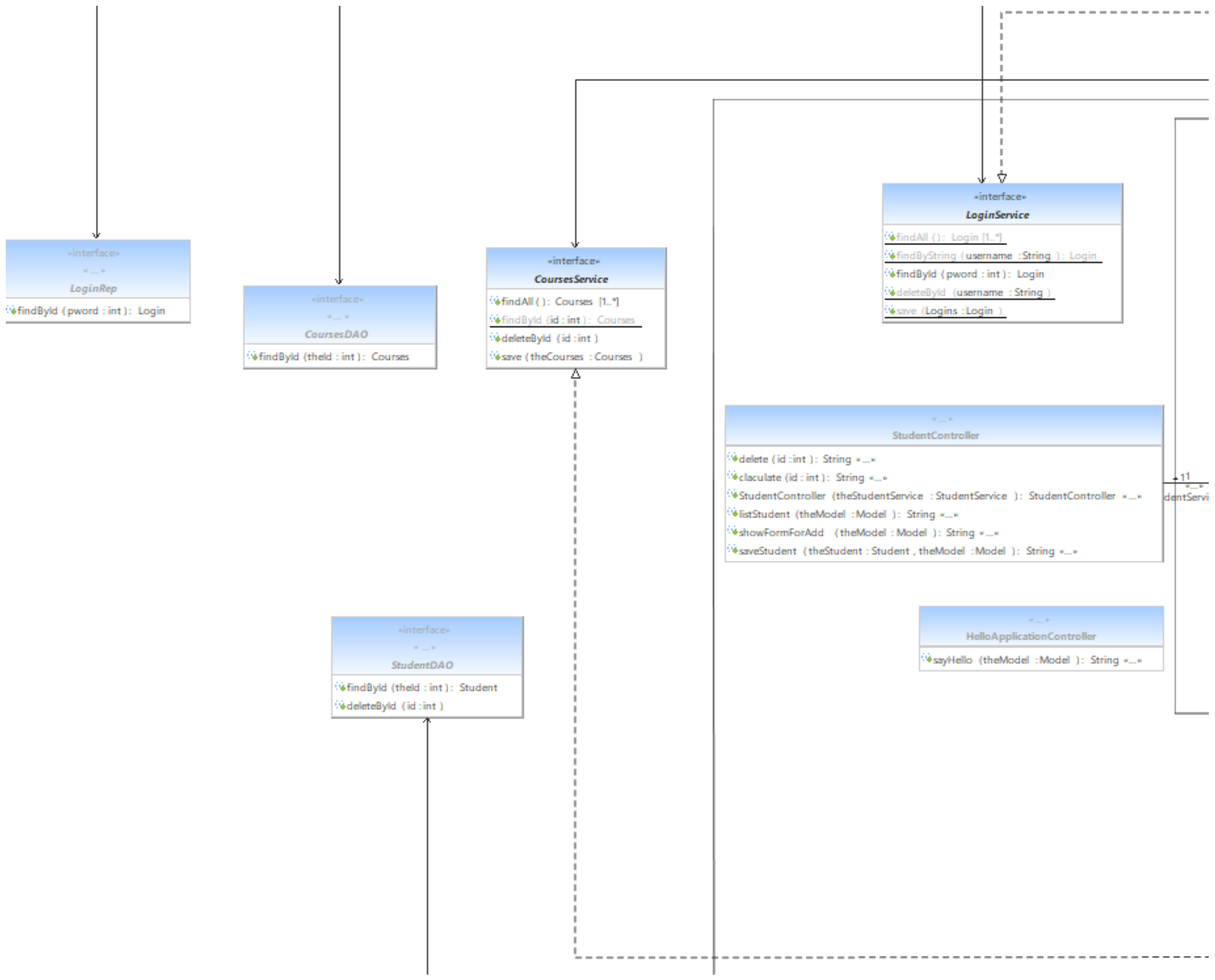
3.12<Use Case 12>

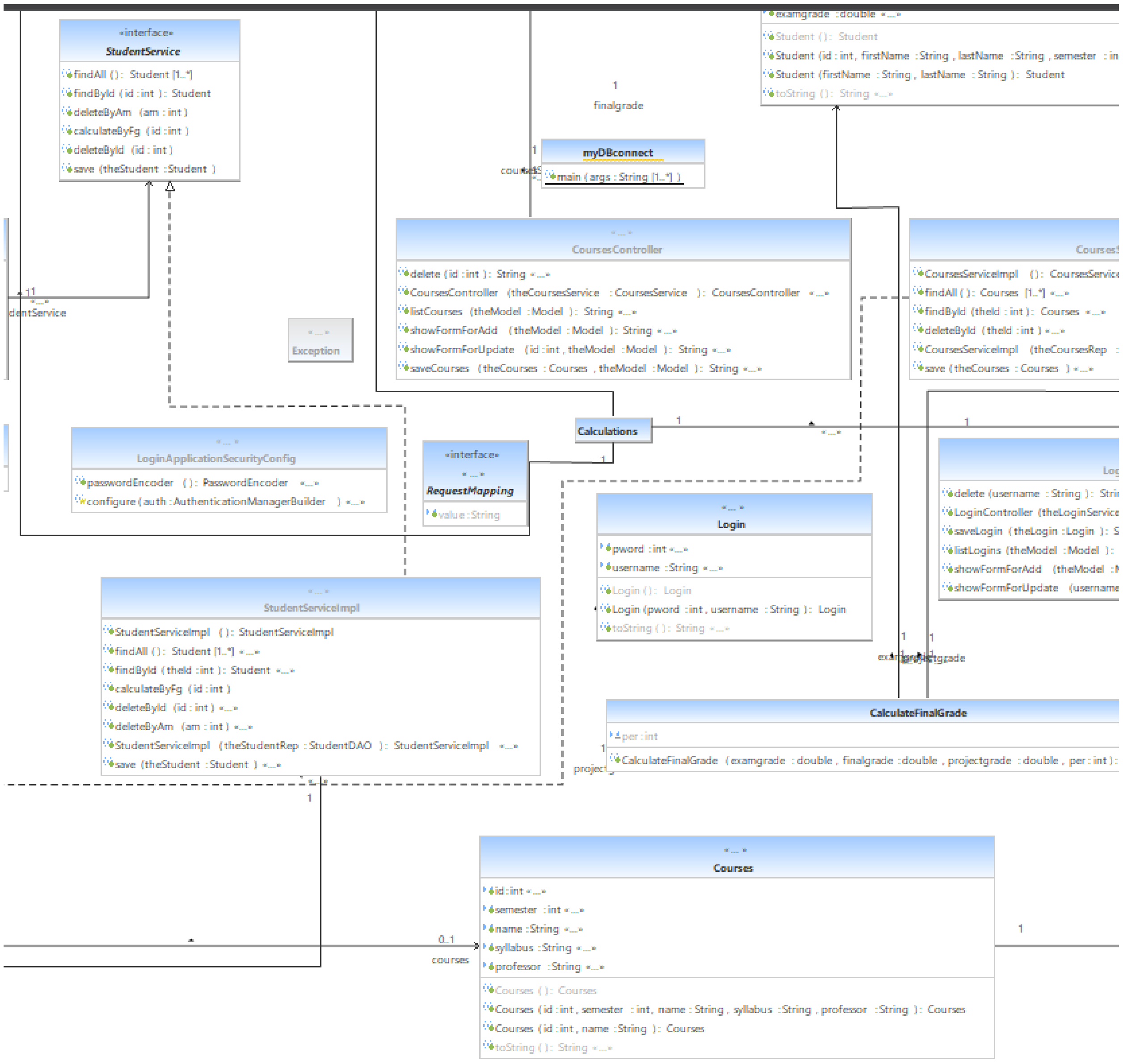
Use case ID	12
Actors	Instructor
Pre conditions	Students need to be enrolled into the class
Main flow of events	The use case starts when the instructor presses the button “Statistics ” then he is redirected into a page where the min, max , mean, standard deviation ,variance etc. are calculated
Alternative flow 1	If one or less student is enrolled into the course then the statistics will not be able to calculate because we need two or more students to exist
Post conditions	Study the distribution of the students grades in the course

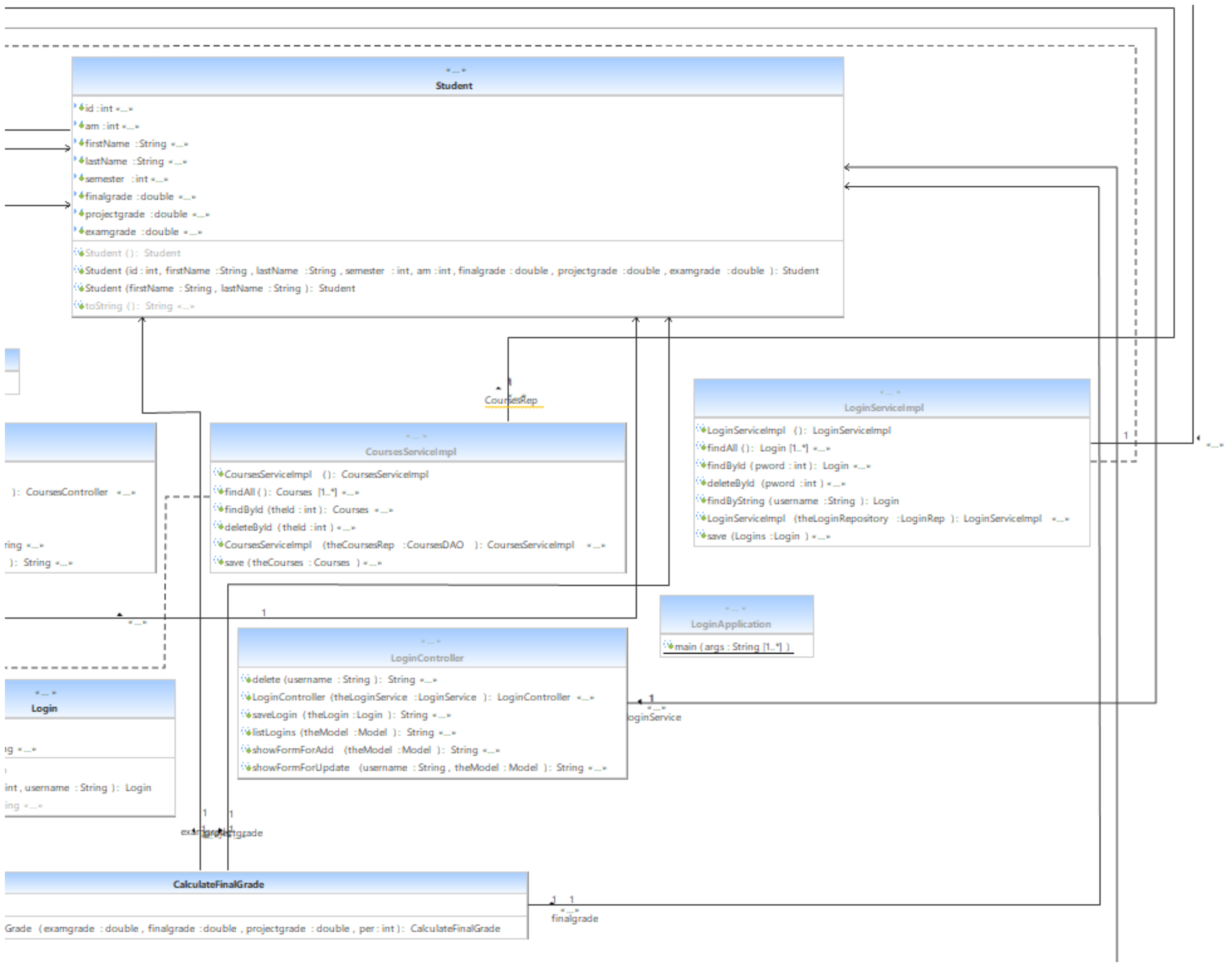
4 Design

4.1 Architecture




«interface»
LoginRep
findById (pword : int): Login
«interface»
CoursesDAO
findById (theId : int): Courses
«interface»
CoursesService
findAll ( ): Courses [1..*]
findById (id : int): Courses
deleteById (id : int)
save (theCourses : Courses)
«interface»
LoginService
findAll ( ): Login [1..*]
findByString (username : String): Login
findById (pword : int): Login
deleteById (username : String)
save (Logins : Login)
StudentController
delete (id : int): String «...»
claculate (id : int): String «...»
StudentController (theStudentService : StudentService): StudentController «...»
listStudent (theModel : Model): String «...»
showFormForAdd (theModel : Model): String «...»
saveStudent (theStudent : Student , theModel : Model): String «...»
HelloApplicationController
sayHello (theModel : Model): String «...»
«interface»
StudentDAO
findById (theId : int): Student
deleteById (id : int)
+1
dentServi


«interface»
StudentService
findAll ( ): Student [1..*]
findById (id : int): Student
deleteByAm (am : int)
calculateByFg (id : int)
deleteById (id : int)
save (theStudent : Student)
examgrade : double «...»
Student ( ): Student
Student (id : int, firstName : String , lastName : String , semester : in
Student (firstName : String , lastName : String): Student
toString ( ): String «...»
1
finalgrade
myDBconnect
main (args : String [1..*])
courses
CoursesController
delete (id : int): String «...»
CoursesController (theCoursesService : CoursesService): CoursesController «...»
listCourses (theModel : Model): String «...»
showFormForAdd (theModel : Model): String «...»
showFormForUpdate (id : int , theModel : Model): String «...»
saveCourses (theCourses : Courses , theModel : Model): String «...»
Courses
CoursesServiceImpl ( ): CoursesService
findAll ( ): Courses [1..*] «...»
findById (theId : int): Courses «...»
deleteById (theId : int) «...»
CoursesServiceImpl (theCoursesRep :
save (theCourses : Courses) «...»
dentService
Exception
Calculations
LoginApplicationSecurityConfig
passwordEncoder ( ): PasswordEncoder «...»
configure (auth : AuthenticationManagerBuilder) «...»
«interface»
RequestMapping
value : String
Login
pword : int «...»
username : String «...»
Login ( ): Login
Login (pword : int , username : String): Login
toString ( ): String «...»
Log
delete (username : String): Strin
LoginController (theLoginService
saveLogin (theLogin : Login): S
listLogins (theModel : Model):
showFormForAdd (theModel : M
showFormForUpdate (username
StudentServiceImpl
StudentServiceImpl ( ): StudentServiceImpl
findAll ( ): Student [1..*] «...»
findById (theId : int): Student «...»
calculateByFg (id : int)
deleteById (id : int) «...»
deleteByAm (am : int) «...»
StudentServiceImpl (theStudentRep : StudentDAO): StudentServiceImpl «...»
save (theStudent : Student) «...»
examgrade
projectgrade
CalculateFinalGrade
per : int
CalculateFinalGrade (examgrade : double , finalgrade : double , projectgrade : double , per : int):
project
Courses
id : int «...»
semester : int «...»
name : String «...»
syllabus : String «...»
professor : String «...»
Courses ( ): Courses
Courses (id : int , semester : int, name : String , syllabus : String , professor : String): Courses
Courses (id : int , name : String): Courses
toString ( ): String «...»
0..1
courses
1



Student
id :int
am :int
firstName :String
lastName :String
semester :int
finalgrade :double
projectgrade :double
examgrade :double
Student (): Student
Student (id : int, firstName :String , lastName :String , semester :int, am :int , finalgrade :double , projectgrade :double , examgrade :double): Student
Student (firstName :String , lastName :String): Student
toString (): String
CoursesRep
CoursesServiceImpl
CoursesServiceImpl (): CoursesServiceImpl
findAll ( ): Courses [1..*]
findById (theId :int): Courses
deleteById (theId :int)
CoursesServiceImpl (theCoursesRep :CoursesDAO): CoursesServiceImpl
save (theCourses :Courses)
): CoursesController
LoginServiceImpl
LoginServiceImpl (): LoginServiceImpl
findAll (): Login [1..*]
findById (pword : int): Login
deleteById (pword :int)
findByString (username :String): Login
LoginServiceImpl (theLoginRepository :LoginRep): LoginServiceImpl
save (Logins :Login)
LoginApplication
main (args : String [1..*])
LoginController
delete (username :String): String
LoginController (theLoginService :LoginService): LoginController
saveLogin (theLogin :Login): String
listLogins (theModel :Model): String
showFormForAdd (theModel :Model): String
showFormForUpdate (username :String , theModel :Model): String
loginService
Login
int , username :String): Login
CalculateFinalGrade
Grade (examgrade :double , finalgrade :double , projectgrade :double , per : int): CalculateFinalGrade
finalgrade

4.2 Design

Class Name: Login	
Responsibilities:	**Collaborations:**
▪ To process the data from the login table from database	▪ Our Database

Class Name: LoginApplication	
Responsibilities:	**Collaborations:**
▪ Run the application	-

Class Name: LoginApplicationSecurityConfig	
Responsibilities: ▪ Holds the user’s details such as username, encrypted password, role	**Collaborations:** -

Class Name: LoginController	
Responsibilities: ▪ Maps the corresponding features from our project to the html page	**Collaborations:** ▪ HTML

Class Name: LoginRep	
Responsibilities: ▪ Extends to the JPARepository	**Collaborations:** ▪ LoginServiceImpl

Class Name: LoginService	
Responsibilities: ▪ Interface for LoginServiceImpl	**Collaborations:** ▪ LoginServiceImpl ▪ LoginController

Class Name: LoginServiceImpl	
Responsibilities: ▪ FindAll() to show all the logins ▪ FindByString() to show logins based on username ▪ FindById() to show logins based on password ▪ DeleteById() deletes login based on password	**Collaborations:** ▪ LoginService ▪ LoginRep

▪ Save() to save a login	

Class Name: Courses	
Responsibilities: ▪ To process the data from the courses table from database	**Collaborations:** ▪ Our database

Class Name: CoursesController	
Responsibilities: ▪ Maps the corresponding features from our project to the html page	**Collaborations:** ▪ HTML

Class Name: CoursesDAO	
Responsibilities: ▪ Extends to the JPARepository ▪ Declares the FindById() method	**Collaborations:** ▪ CoursesServiceImpl

Class Name: CoursesService	
Responsibilities: ▪ Interface for CoursesServiceImpl	**Collaborations:** ▪ CoursesController ▪ CoursesServiceImpl

Class Name: CoursesServiceImpl	
Responsibilities: ▪ FindAll() to show all the courses ▪ FindById() to show courses based on password ▪ DeleteById() deletes courses based on password ▪ Save() to save a course ▪ A constructor to connect with coursesDAO which is the repository	**Collaborations:** ▪ CoursesService ▪ CoursesDAO

Class Name: Student	
Responsibilities: ▪ To process the data from the student table from database	**Collaborations:** -

Class Name: StudentController	
Responsibilities: ▪ Maps the corresponding features from our project to the html page	**Collaborations:** ▪ StudentService

Class Name: StudentDAO	
Responsibilities: ▪ Extends to the JPARepository ▪ Declares the FindById() and	**Collaborations:** ▪ StudentServiceImpl

DeleteById() methods	

Class Name: StudentService	
Responsibilities: ▪ Interface for StudentServiceImpl	**Collaborations:** ▪ StudentController ▪ StudentServiceImpl

Class Name: StudentServiceImpl	
Responsibilities: ▪ FindAll() to show all the students ▪ FindById() to show students based on password ▪ DeleteById() deletes student based on password ▪ Save() to save a student ▪ A constructor to connect with studentDAO which is the repository	**Collaborations:** ▪ StudentService ▪ StudentDAO

Class Name: HelloApplicationController	
Responsibilities: ▪ Maps the corresponding features from our project to the html page	**Collaborations:** ▪ -

Class Name: CalculateFinalGrade	
Responsibilities: ▪ CalculateFinalGrade()	**Collaborations:** ▪ Student

5 Difficulties We Encountered

1)The User Story 5 which demanded to update the course's information was not executed propertly

- We created the method showFormForUpdate() which creates new entries based on id and then adds them to the model. Then we redirect to the method that adds courses. Unfortunately it does not solve our problem.

2)The User Story 9 like the previous user story does not update the student's information. Probably for the same reasons.

3)Our next problem was about the student entries into the course . In our database we defined the course id as foreign key into our student table because we need to register the students into the proper course . The mysql database lets us to insert multiple values with the same foreign key . In order to achieve that in our project we tried to implement the annotation one to many for our parent class which let us use the same course id into many students , also we used the annotation many to one for our child class student which let us use one foreign key for multiple student entries . Although these annotations do not seem to work for our project and that is the reason that we have entered only one student per course .

4)For the user story 11 we did not create any classes because we faced many problems regarding the overall entries and redirections so we focus on solving those problems for our previous user stories.

5)Finally for the User Story 12 we did not calculate the statistics that the user story requires , as we previously stated we could not add more than one student per course so we did not have enough grades to be able to do the calculations .