CoursesManagementApp

Sprint Report

Βουρδουγιάννης Δημήτριος 4326
Βάια Νικολαΐδου 4448
Θεόδωρος Φλορέντζης 4517

1 Introduction

In this project we are making an application that stores courses and student registrations and any instructor can login to the system and modify their own courses or student registrations.

2 Use Cases

2.1 LoginToApplication

Use case ID	UC1
Actors	Instructor
Pre conditions	The instructor has to be registered to the database.
Main flow of events	1. The use case starts when the instructor starts the application. 2. The system asks for user name and password. 3. The instructor types his user name and password. 3.1. If the user name and password are correct, the use case ends. 3.2. If the user name and password are incorrect, the use case goes back to step 2.
Post conditions	The instructor is logged in to the system.

2.2 BrowseListofCourses

Use case ID	UC2
Actors	Instructor
Pre conditions	The instructor has to be logged in to the system.
Main flow of events	1. The use case starts right after the login. 2. The system shows a list of the instructor's courses.
Post conditions	The system shows a list of the courses.

2.3 AddCourse

Use case ID	UC3
Actors	Instructor
Pre conditions	The instructor has to be logged in to the system.
Main flow of events	1. The use case starts when the instructor presses the button "add course". 2. The system asks information about the course such as name, description, instructor, year and semester. 3. The instructor gives the required information. 4. The instructor presses the button "save".
Basic flow Post conditions	1. The instructor is redirected to the list of courses. 2. A new course is added to the system with its required fields.

2.4 RemoveCourse

Use case ID	UC4
Actors	Instructor
Pre conditions	1. The instructor has to be logged in to the system. 2. At least one course is submitted to the application.
Main flow of events	1. The use case starts when the instructor presses the button "delete course". 2. The system removes the course.
Post conditions	The system has removed the certain course from the database and is no longer available.

2.5 UpdateDescriptionofCourse

Use case ID	UC5
Actors	Instructor
Pre conditions	1. The instructor has to be logged in to the system. 2. At least one course is submitted to the application.
Main flow of events	1. The use case starts when the instructor presses the button "update course". 2. The system asks for the new information about the course (name, description, instructor, year and semester). 3. The instructor makes the desirable changes. 4. The use case ends when instructor presses the "save" button.
Alternative Flow	The instructor can exit the use case at any point by pressing the button "back to courses list".
Post conditions	1. There is a new description for the course. 2. The instructor is redirected to the list of courses.

2.6 BrowseListofStudents

Use case ID	UC6
Actors	Instructor
Pre conditions	The instructor has to be logged in to the system.
Main flow of events	1. The use case starts when the instructor presses the button “show registrations”. 2. The system shows a list of the instructor’s students in the specific course.
Post conditions	The system shows a list of the students.

2.7 AddStudent

Use case ID	UC7
Actors	Instructor
Pre conditions	The instructor has to be logged in to the system.
Main flow of events	1. The use case starts when the instructor presses the button “add student”. 2. The system asks information about the student such as id, name, year of registration, semester, projectgrade, examsgrade. 3. The instructor gives the required information.
Alternative flow 1	The instructor can exit the use case at any point by pressing the button “back to student registrations list”.
Post conditions	1. A new student is added to the system with its required fields. 2. The instructor is redirected to the list of student registrations.

2.8 RemoveStudent

Use case ID	UC8
Actors	Instructor
Pre conditions	1. The instructor has to be logged in to the system. 2. At least one course is submitted to the application.
Main flow of events	1. The use case starts when the instructor presses the button “remove student”. 2. The system removes the student.
Post conditions	The system has removed the certain student from the database and is no longer available.

2.9 UpdateInformationofStudent

Use case ID	UC9
Actors	Instructor
Pre conditions	1. The instructor has to be logged in to the system. 2. At least one course is submitted to the application.
Main flow of events	1. The use case starts when the instructor presses the button “update student registration”. 2. The system shows a customizable text editor for the student’s information. 3. The instructor makes the desirable changes for the student’s id, name, year of studies etc. 4. The use case ends when instructor presses the “save” button.
Alternative Flow	The instructor can exit the use case at any point by pressing the button “back to student registrations list”
Post conditions	1. There is an updated version of the student’s information. 2. The instructor is redirected to the list of student registrations.

2.10 RegisterGrades

Use case ID	UC10
Actors	Instructor
Pre conditions	1. The instructor has to be logged in to the system. 2. At least one student is submitted to the corresponding course.
Main flow of events	1. The use case starts when the instructor presses the button “update student registration” for a particular student. 2. The system redirects the instructor to update form of student registrations. 3. The instructor types the projectgrade and finalexamgrade. 4. The instructor presses the “save” button.
Alternative flow	The instructor can exit the use case at any point by pressing the button “back to student registrations list”
Post conditions	1. The grades of the student have been registered. 2. The instructor is redirected to the list of student registrations.

2.11 CalculateOverallGrades

Use case ID	UC11
Actors	Instructor
Pre conditions	1. The instructor has to be logged in to the system. 2. At least one student is submitted to the corresponding course.
Main flow of events	1. The use case starts when the instructor updates the information of projectgrade and the finalexamgrade. 2. The system calculates the overall grades respecting the weighted average when the instructor presses the “save” button.
Post conditions	The overall grades are shown in the application.

2.12 CalculateStatistics

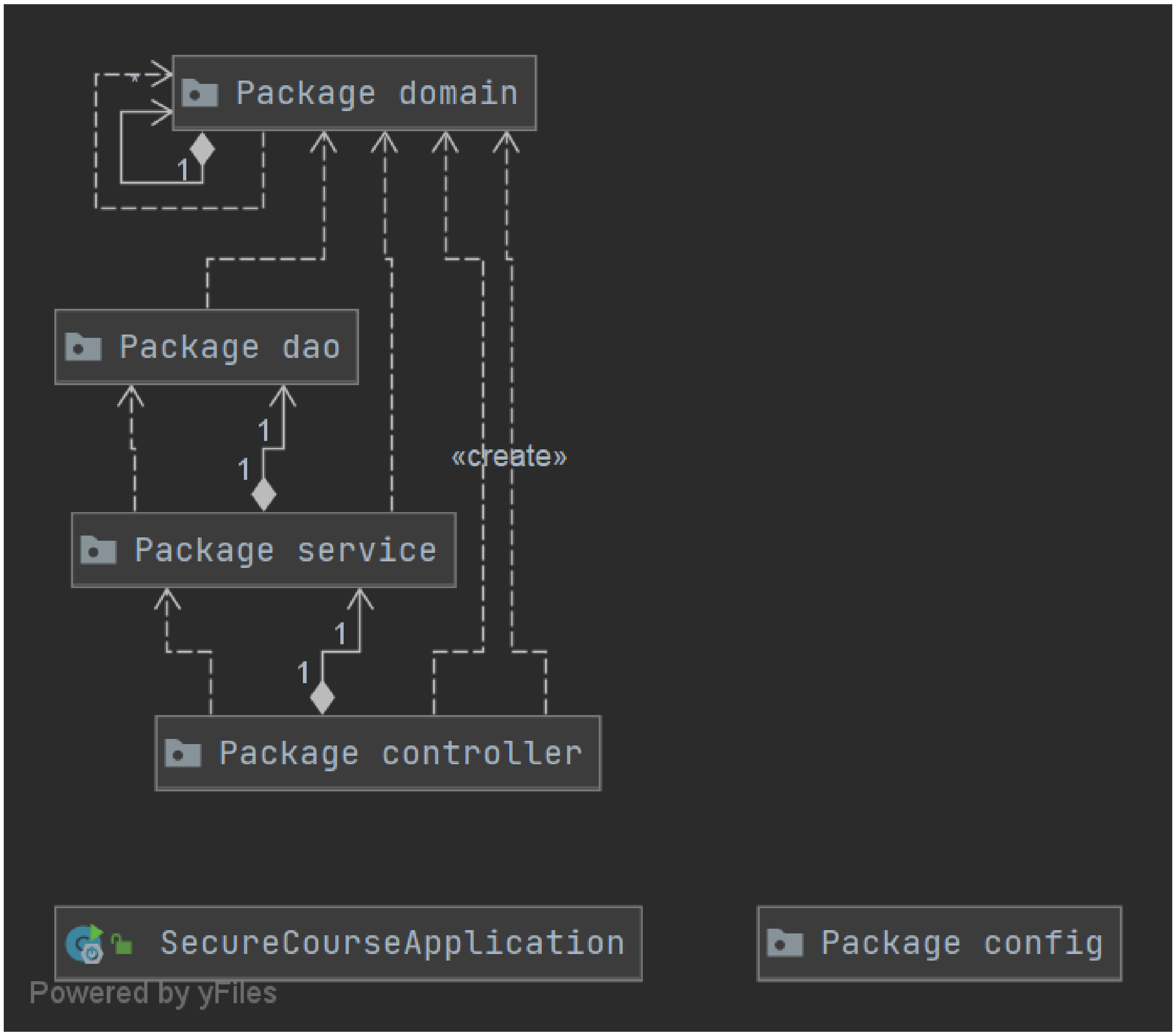
Use case ID	UC12
Actors	Instructor
Pre conditions	1. The instructor has to be logged in to the system. 2. At least one student is submitted to the corresponding course.
Main flow of events	1. The use case starts when the instructor presses the button "calculate statistics" for a certain course. 2. The system shows a list of the available statistics strategies, which are min, max, mean, standard deviation, variance, percentiles, skewness, kurtosis, median. 3. The instructor chooses the desirable strategies. 4. The system calculates the asked result.
Post conditions	The system prints the final results to the screen.

*This use case (2.12 Calculate Statistics) has not been implemented.

3 Design

3.1 Architecture

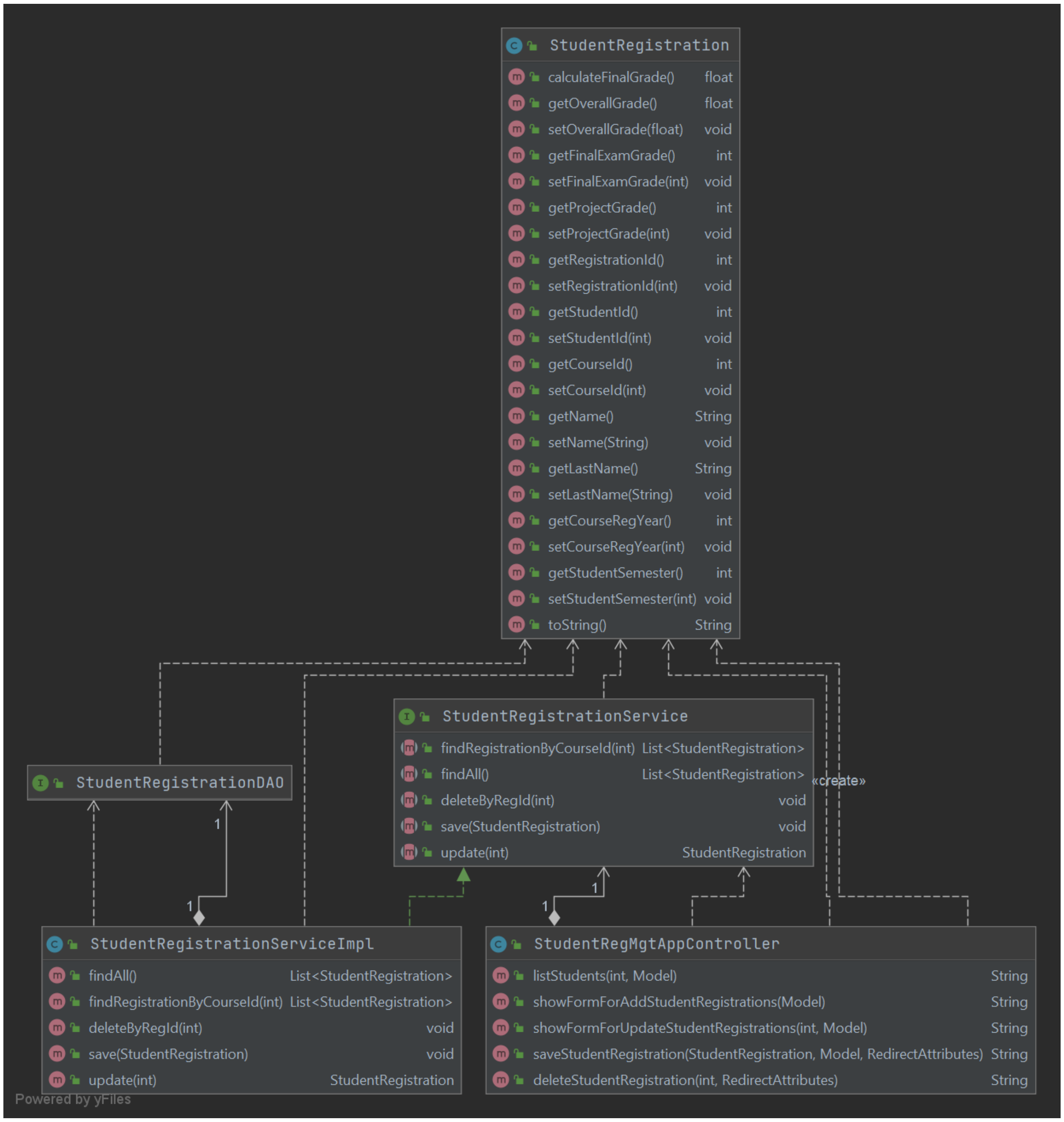
UML package diagram



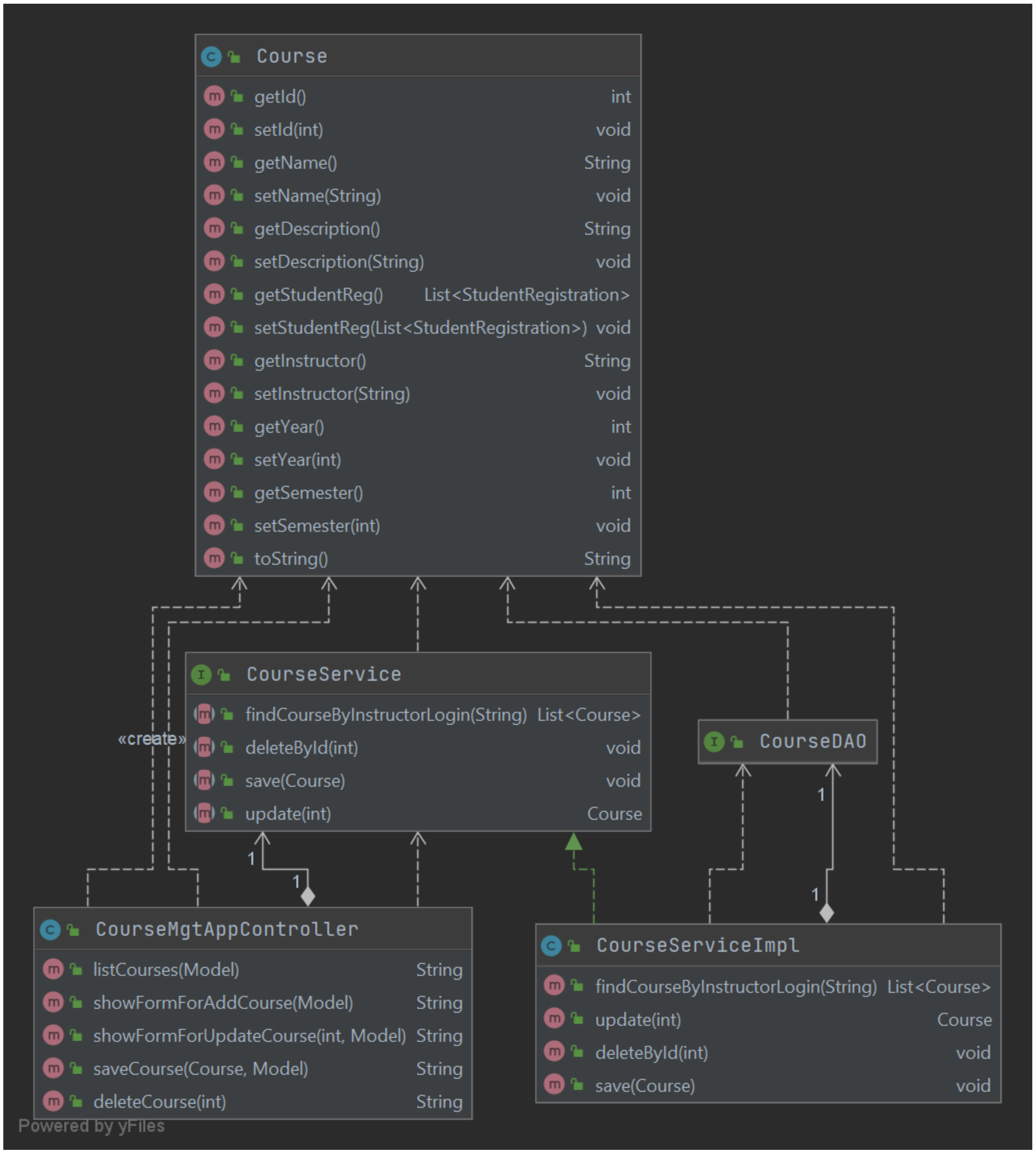
3.2 Design

UML class diagrams

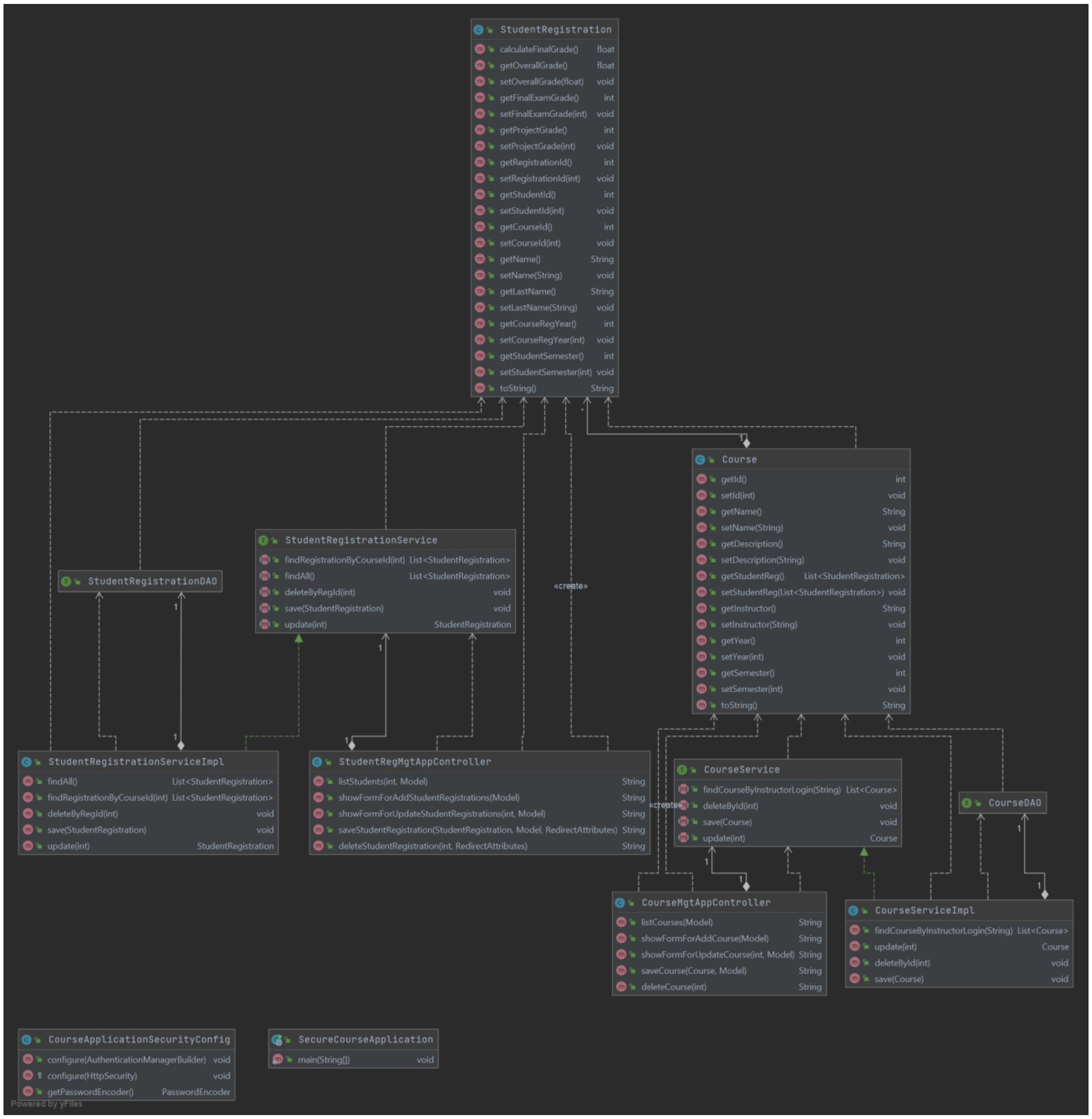
- StudentRegistration:


StudentRegistration
calculateFinalGrade() float
getOverallGrade() float
setOverallGrade(float) void
getFinalExamGrade() int
setFinalExamGrade(int) void
getProjectGrade() int
setProjectGrade(int) void
getRegistrationId() int
setRegistrationId(int) void
getStudentId() int
setStudentId(int) void
getCourseId() int
setCourseId(int) void
getName() String
setName(String) void
getLastName() String
setLastName(String) void
getCourseRegYear() int
setCourseRegYear(int) void
getStudentSemester() int
setStudentSemester(int) void
toString() String
StudentRegistrationService
findRegistrationByCourseId(int) List<StudentRegistration>
findAll() List<StudentRegistration>
deleteByRegId(int) void
save(StudentRegistration) void
update(int) StudentRegistration
«create»
StudentRegistrationDAO
1
1
1
1
StudentRegistrationServiceImpl
findAll() List<StudentRegistration>
findRegistrationByCourseId(int) List<StudentRegistration>
deleteByRegId(int) void
save(StudentRegistration) void
update(int) StudentRegistration
StudentRegMgtAppController
listStudents(int, Model) String
showFormForAddStudentRegistrations(Model) String
showFormForUpdateStudentRegistrations(int, Model) String
saveStudentRegistration(StudentRegistration, Model, RedirectAttributes) String
deleteStudentRegistration(int, RedirectAttributes) String
Powered by yFiles

- Course:


Course
getId() int
setId(int) void
getName() String
setName(String) void
getDescription() String
setDescription(String) void
getStudentReg() List<StudentRegistration>
setStudentReg(List<StudentRegistration>) void
getInstructor() String
setInstructor(String) void
getYear() int
setYear(int) void
getSemester() int
setSemester(int) void
toString() String
CourseService
findCourseByInstructorLogin(String) List<Course>
deleteById(int) void
save(Course) void
update(int) Course
«create»
CourseDAO
1
1
1
1
CourseMgtAppController
listCourses(Model) String
showFormForAddCourse(Model) String
showFormForUpdateCourse(int, Model) String
saveCourse(Course, Model) String
deleteCourse(int) String
CourseServiceImpl
findCourseByInstructorLogin(String) List<Course>
update(int) Course
deleteById(int) void
save(Course) void
Powered by yFiles

- Skeleton:


StudentRegistration
calculateFinalGrade() float
getOverallGrade() float
setOverallGrade(float) void
getFinalExamGrade() int
setFinalExamGrade(int) void
getProjectGrade() int
setProjectGrade(int) void
getRegistrationId() int
setRegistrationId(int) void
getStudentId() int
setStudentId(int) void
getCourseId() int
setCourseId(int) void
getName() String
setName(String) void
getLastName() String
setLastName(String) void
getCourseRegYear() int
setCourseRegYear(int) void
getStudentSemester() int
setStudentSemester(int) void
toString() String
Course
getId() int
setId(int) void
getName() String
setName(String) void
getDescription() String
setDescription(String) void
getStudentReg() List<StudentRegistration>
setStudentReg(List<StudentRegistration>) void
getInstructor() String
setInstructor(String) void
getYear() int
setYear(int) void
getSemester() int
setSemester(int) void
toString() String
StudentRegistrationService
findRegistrationByCourseId(int) List<StudentRegistration>
findAll() List<StudentRegistration>
deleteByRegId(int) void
save(StudentRegistration) void
update(int) StudentRegistration
StudentRegistrationDAO
«create»
StudentRegistrationServiceImpl
findAll() List<StudentRegistration>
findRegistrationByCourseId(int) List<StudentRegistration>
deleteByRegId(int) void
save(StudentRegistration) void
update(int) StudentRegistration
StudentRegMgtAppController
listStudents(int, Model) String
showFormForAddStudentRegistrations(Model) String
showFormForUpdateStudentRegistrations(int, Model) String
saveStudentRegistration(StudentRegistration, Model, RedirectAttributes) String
deleteStudentRegistration(int, RedirectAttributes) String
CourseService
findCourseByInstructorLogin(String) List<Course>
deleteById(int) void
save(Course) void
update(int) Course
«create»
CourseDAO
CourseMgtAppController
listCourses(Model) String
showFormForAddCourse(Model) String
showFormForUpdateCourse(int, Model) String
saveCourse(Course, Model) String
deleteCourse(int) String
CourseServiceImpl
findCourseByInstructorLogin(String) List<Course>
update(int) Course
deleteById(int) void
save(Course) void
CourseApplicationSecurityConfig
configure(AuthenticationManagerBuilder) void
configure(HttpSecurity) void
getPasswordEncoder() PasswordEncoder
SecureCourseApplication
main(String[]) void
1
1
1
1
1
1
1
1
1
Powered by yFiles

Details of the classes:

Class Name: SecurityCourseApplication	
Responsibilities: ▪ This class includes the main method and runs the application	**Collaborations:** -

Package config:

Class Name: CourseApplicationSecurityConfig	
Responsibilities: ▪ Security of the system ▪ Safely login to the application	**Collaborations:** **-**

Package controller:

Class Name: CourseMgtAppController	
Responsibilities: ▪ List the courses of an instructor ▪ Adds courses ▪ Update courses ▪ Save courses ▪ Delete courses	**Collaborations:** ▪ Course ▪ CourseService

Class Name: StudentRegMgtAppController	
Responsibilities: ▪ List the students of a course ▪ Update students ▪ Adds students ▪ Delete students	**Collaborations:** ▪ StudentRegistration ▪ StudentRegistrationService

Package dao:

Class Name: CourseDAO	
Responsibilities:	**Collaborations:**
▪ This class gives us access to JpaRepository	▪ Course ▪ CourseServiceImpl

Class Name: StudentRegistrationDAO	
Responsibilities:	**Collaborations:**
▪ This class gives us access to JpaRepository	▪ StudentRegistration ▪ StudentRegistrationServiceImpl

Package domain:

Class Name: Course	
Responsibilities:	**Collaborations:**
▪ Creating the Course	▪ StudentRegistration ▪ CourseDAO ▪ CourseMgtAppController ▪ CourseService ▪ CourseServiceImpl

Class Name: StudentRegistration	
Responsibilities:	**Collaborations:**
▪ Registering the student ▪ Get statistics of a student (Overall grade, final exam grade etc.)	▪ Course ▪ StudentRegistrationDAO ▪ StudentRegistrationMgtAppController ▪ StudentRegistrationService ▪ StudentRegistrationServiceImpl

Package service:

Class Name: CourseService	
Responsibilities:	**Collaborations:**
▪ The front-end class of CourseService	▪ Course ▪ CourseMgtAppController ▪ CourseService

Class Name: CourseServiceImpl	
Responsibilities:	**Collaborations:**
▪ The back-end class of CourseService ▪ Finds the courses of an instructor ▪ Implements update method ▪ Implements delete method ▪ Implements save method	▪ CourseService ▪ CourseDAO ▪ Course

Class Name: StudentRegistrationService	
Responsibilities:	**Collaborations:**
▪ The front-end class of StudentRegistrationService	▪ StudentRegistration ▪ StudentRegistrationMgtAppController ▪ StudentRegistrationService

Class Name: StudentRegistrationServiceImpl	
Responsibilities:	**Collaborations:**
▪ The back-end class of StudentRegistrationService ▪ Finds all the students registrations ▪ Finds students registrations of a course ▪ Implements delete method ▪ Implements save method ▪ Implements update method	▪ StudentRegistrationService ▪ StudentRegistrationDAO ▪ StudentRegistration