Student Information System

Server API with Clients

This document is written by Roemer Bakker and Joshua Turpijn for the course 'Object Oriented Analysis and Design' as thought by Riemer van Rozen at the Amsterdam University of Applied Sciences. This is assignment 1. The document is composed as followows:

- **1.0:** Vision
- **1.1:** Use Case Model
- **1.2:** Supplementary Specification
- **1.3:** Glossary
- **1.4:** Report

1.0 – Vision

This case describes the new Student Information System. The old system is becoming obsolete and it's not easy enough for students to do things like check their grades, apply for (re-)exams and check their progress. The new system should make this much easier. The system is composed of a central server and a set of clients. The server acts as a central information system from which clients can request information. This composition makes it way easier to write new clients like mobile applications or websites. An Application Programming Interface also allows third parties to interact with the system.

The most important tasks the new system should be able to are as follows:

- Student can check grades
- Student can check progress
- Student can enroll for a (re-)exam
- Teacher can register grades
- Teacher can review grades by class
- Management can generate analytics

1.1 – Use-Case Model

1. Identify the actors

This system will have 3 different actors, one if which has two sub-actors. They are as follows:

- Student
- Teacher
    - Subject teacher
    - Advisory teacher
- Management

In this system, a teacher can be both a subject and an advisory teacher, but can also be one of the two.

2. Identify the goals for each primary actor

Student

- Can review it's own grades on a per subject basis
- Can check the date of a (re-)exam
- Can review ECTS progress
- Can request emails for (re-)exam reminders and new or edited grades
- Can enroll of a re-exam

Subject teacher

- Can register grades
- Can generate grade reports for own subject per class

Advisory teacher

- Can generate grade reports from all subjects for own class

Management

- Can create studies
- Can create groups by graduation year
- Can register student accounts
- Can bind students to group
- Can register teacher accounts
- Can generate statistics about student

3. Define use cases that satisfy user goals

Use-Case Section	Description
Use-Case Name	Student registers study planning
Scope	Student Information System
Level	User-goal
Primary Actor	Student
Stakeholders and interests	1. The HvA requires students to think about the planning of their study in advance. 2. Advisory teachers want a up-to-date overview of their students study planning
Preconditions	1. Student has logged in 2. The student is in their second year of enrollment
Success Guarantee	The student has registered a definite version of their study planning and it has been send to it's advisory teacher for approval.
Main Success Scenario	1. Student selects study planning 2. Student selects its intended course program on a per-semester-basis 3. The system requests confirmation 4. Student confirms the registration 5. The system sends the planning to the student's advisory teacher for approval
Extensions	1a. The student is not in their second year of enrollment so the option of going to the study planning page is not available. 2a. The student has not filled out all the semesters and receives a warning from the system. 4a. Student chooses not to confirm and is redirected back to step 2
Special Requirements	None
Technology and Data Variations List	User is in possession of a device that can run a SIS client.
Frequency of occurrence	Once in the second year of enrollment.
Miscellaneous	None

Use-Case Section	Description
Use-Case Name	Student can un-register for re-exam
Scope	Student Information System
Level	User-goal

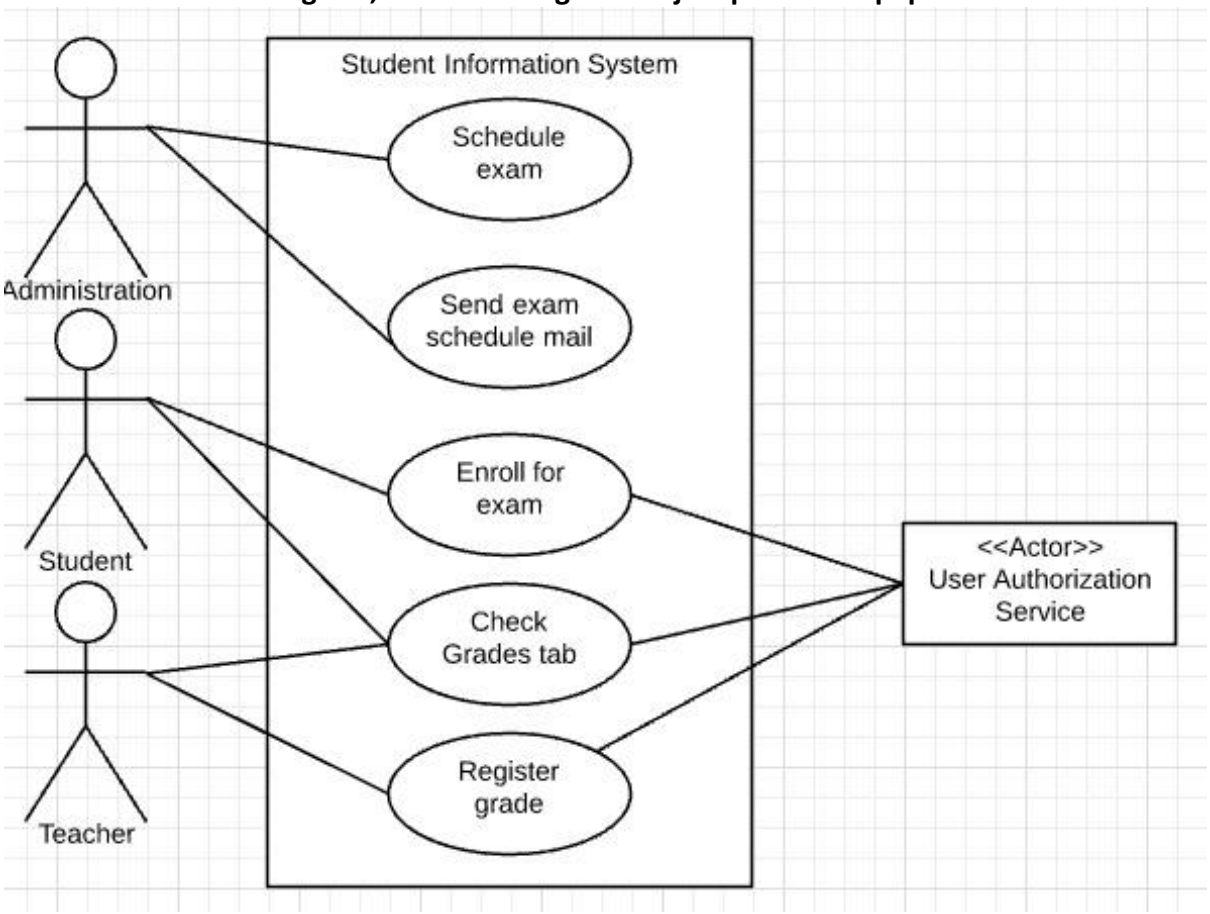
Primary Actor	Student
Stakeholders and Interests	As a student, I want to automatically be registered for re-exams. If I'm unavailable for that time, I want to be able to un-register. The HvA wants to provide such a service.
Preconditions	- User has logged in - Teacher has registered a grade that keeps the student from passing - The system has registered the student for the re-exam of the subject for which he is not passed
Success Guarantee	The student is not registered for the re-exam anymore
Main Success Scenario	1. Student navigates to subjects 2. Student selects the subject corresponding to the re-exam 3. Student selects the exam section 4. Student selects un-register for the re-exam
Extensions	1.a The user has not yet enrolled in a study programme 1.b Teachers have not yet registered any grades 3a. There are no exams available for this course 4a. The date for the re-exam is not yet available so the re-exam does not show up in this list.
Special Requirements	None
Technology and Data Variations List	User is in possession of a device that can run a SIS client.
Frequency of occurrence	Every time the student failed an exam and is not available for the re examination date.
Miscellaneous	None

Use-Case Section	**Description**
Use-Case Name	Subject-Teacher can register grades
Scope	Student Information System
Level	Teacher-goal
Primary Actor	Subject-teacher
Stakeholders and Interests	- Teachers want to spend as little time as possible entering grades. The system should be as easy and intuitive as possible for this process.
Preconditions	- The teacher has logged in - The teacher is done grading a test or exam
Success Guarantee	The teacher is done registering the grades earlier than he would be in the current system and has more time to do other things.
Main Success Scenario	1. Teacher navigates to Register Grades 2. Teacher selects subject and class 3. Teacher enters grades on a per-student basis 4. The system requests confirmation

Extensions	2a. Teacher has not been registered to any courses 4a. Teacher has not entered a grade for all students and receives a notice of this from the system 4b. Teacher chooses not to confirm and is redirected to step 2
Special Requirements	Graded tests
Technology and Data Variations List	Teacher is in possession of a device that can run a SIS client
Frequency of occurrence	Whenever a teacher has graded a test
Miscellaneous	None

Use-Case Section	**Description**
Use-Case Name	Management can create student accounts
Scope	Student Information System
Level	Management
Primary Actor	Management
Stakeholders and Interests	At the start of every academic year, new students should receive an account for the Student Information System. Since the HvA accepts hundreds of new students each year, there should be a way to do this as fast as possible. It's in the interest of a management employee and the HvA that this process is accomplished as quickly as possible.
Preconditions	- The study program that the student is going to follow has been created - The class the student will be in has been created - Management has a list with the student's names - Management employee has logged in
Success Guarantee	All new students have login credentials for the Student Information System
Main Success Scenario	1. Employee navigates to Create Student Accounts 2. Select study programme and class 3. Employee enters meta information about students 4. The system requests confirmation
Extensions	2a. There are no study programs and/or classes in the system 4a. Employee chooses not to confirm and is redirected to step
Special Requirements	None
Technology and Data Variations List	The employee is in possession of a device that can run a SIS client
Frequency of occurrence	Annually
Miscellaneous	None

4. Make a use case diagram, use a drawing tool or just pencil and paper



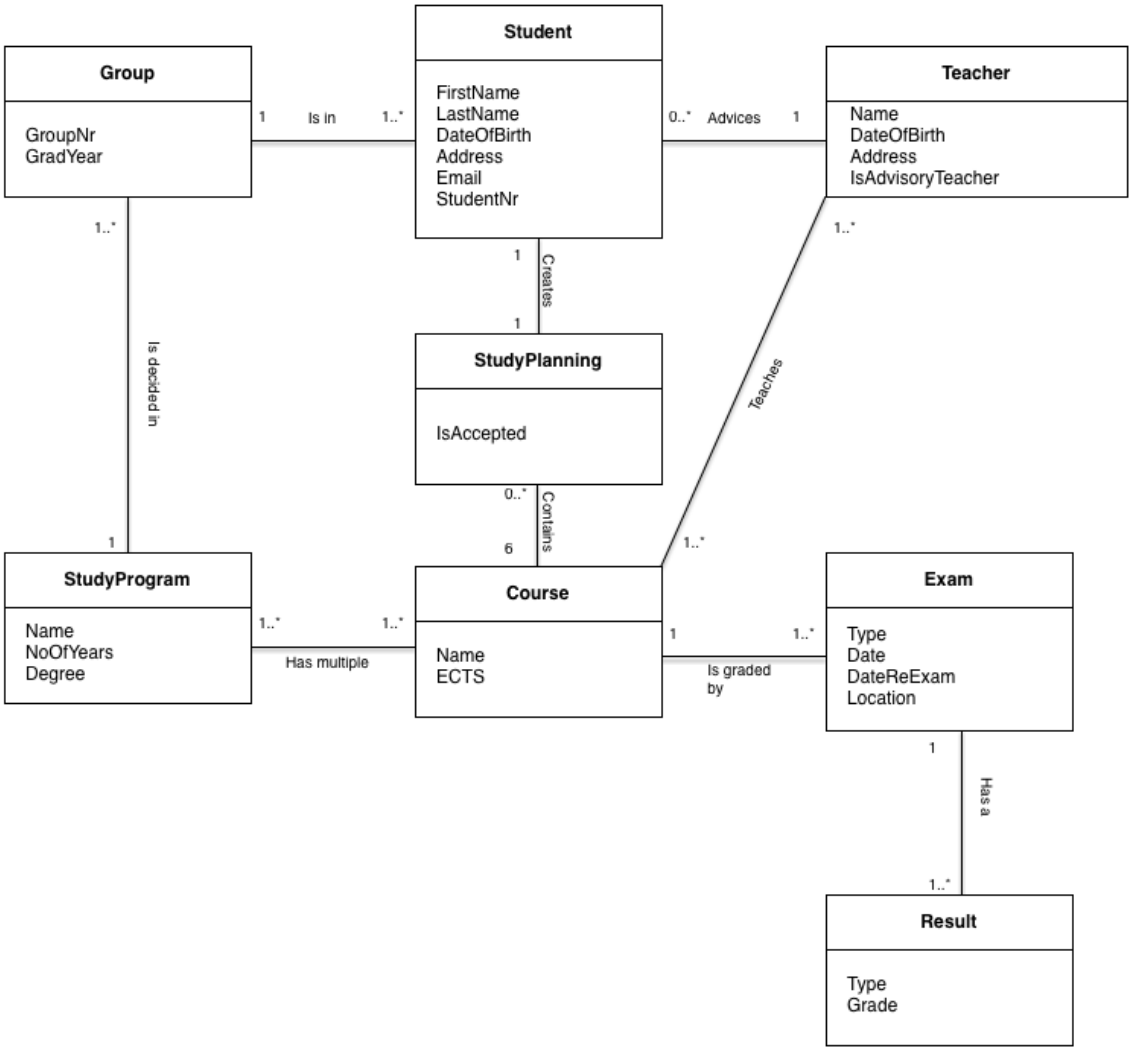
Object Oriented Analysis and Design Assignment 2

This document is written by Roemer Bakker and Joshua Turpijn for the course ‘Object Oriented Analysis and Design’ as thought by Riemer van Rozen at the Amsterdam University of Applied Sciences. This is assignment 2. The document is composed as follows:

- **2.1** : Domain model
- **2.2** : System sequance diagram
- **2.3** : Operation Contracts
- **2.4** : Physical or Logical Architecture
- **2.5** : Refine the artifacts of Assignment 1
- **2.6** : Glossary

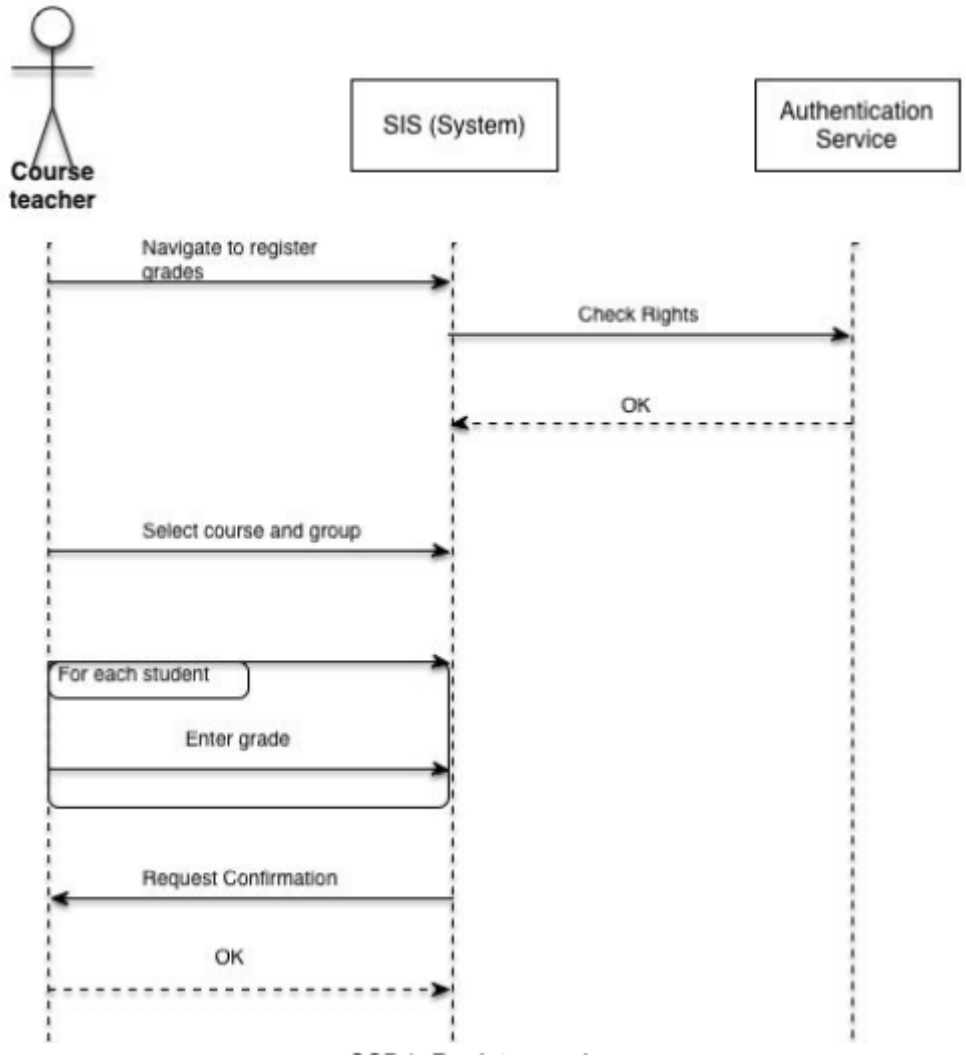
2.1 Domain model

The domain model contain the conceptual classes that we decided to be relevant and necessary for our project. By going through the 6 steps of creating a domain model, we’ve came to the model that’s shown below. The way we save a study planning is by making associations with the courses the student will follow.



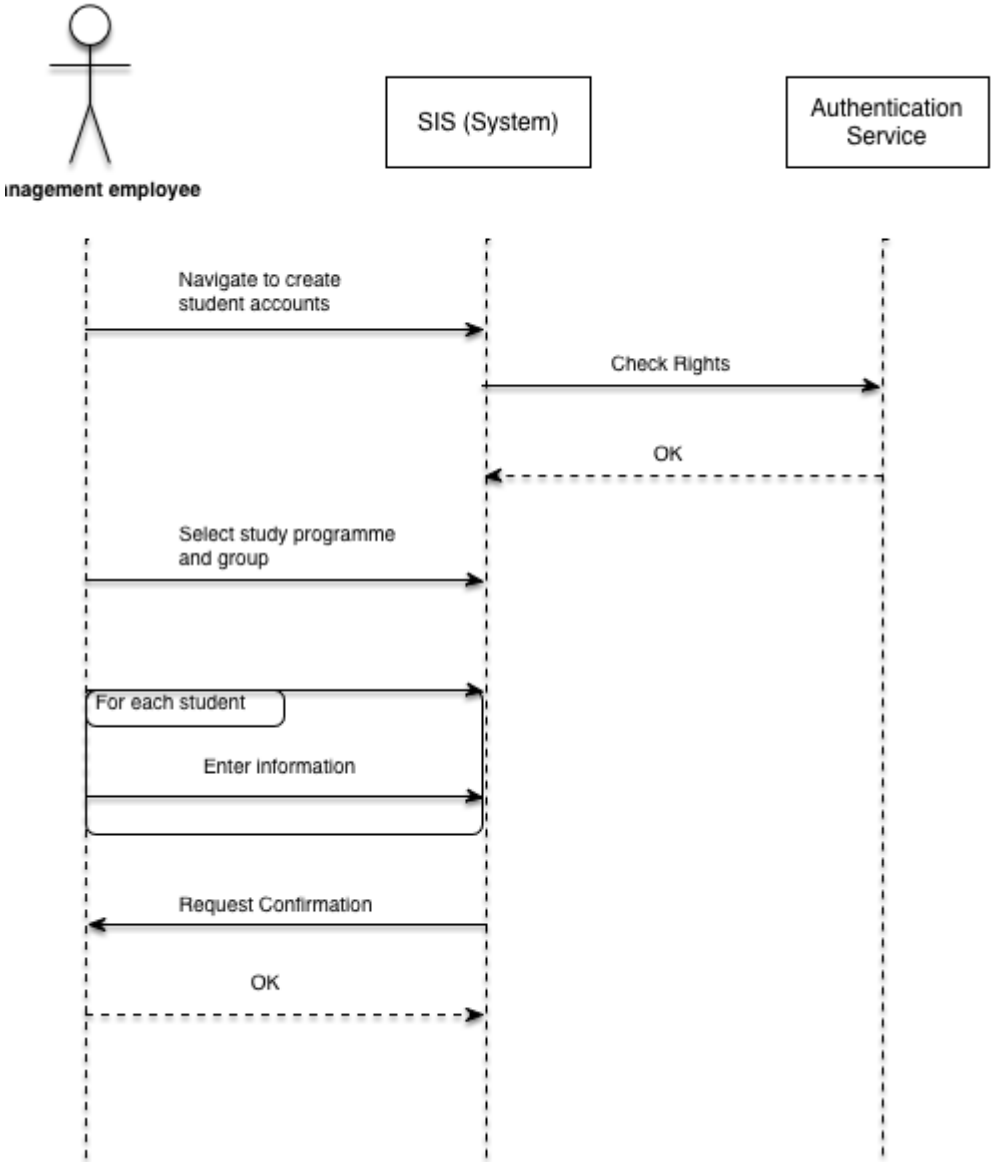
2.2 System Sequence Diagram

Based on the 4 use cases that we've elaborated on in our first assignment, we've made system sequence diagrams. With the use cases we've shown what processes are necessary for the main success scenarios. Shown below are the sequence diagrams who map out where the functions go and how they interact with the actors and supporting actors.



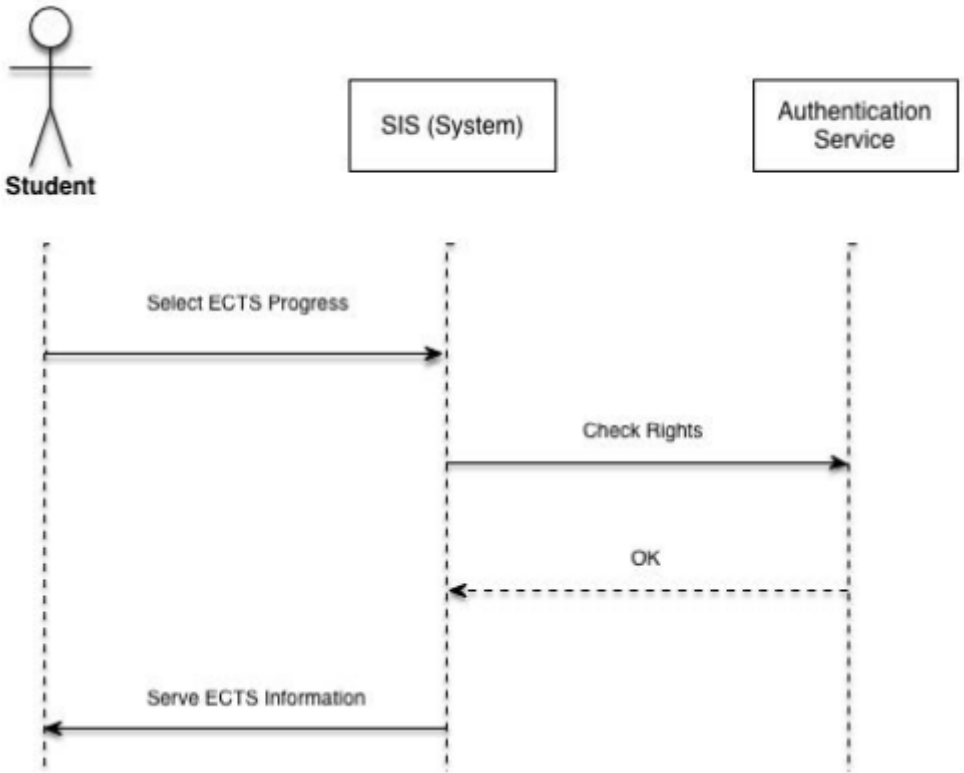
SSD1. Register grades

First up we have a course teacher to register new grades. The pre-condition is that a student finished and submitted an exam. The teacher graded it and resulted a grade. This grade needs to be registered into the system. The course teacher begins by navigating to the page where the grades can be registered. After the authentication service checked the rights of the user, the course teacher gets redirected to the page. The course teacher continues by selecting a course and group where the grades will be registered. The course teacher registers a grade for each student, and when that process is finished, the system(SIS) will create a review of the grades and ask for a confirmation.



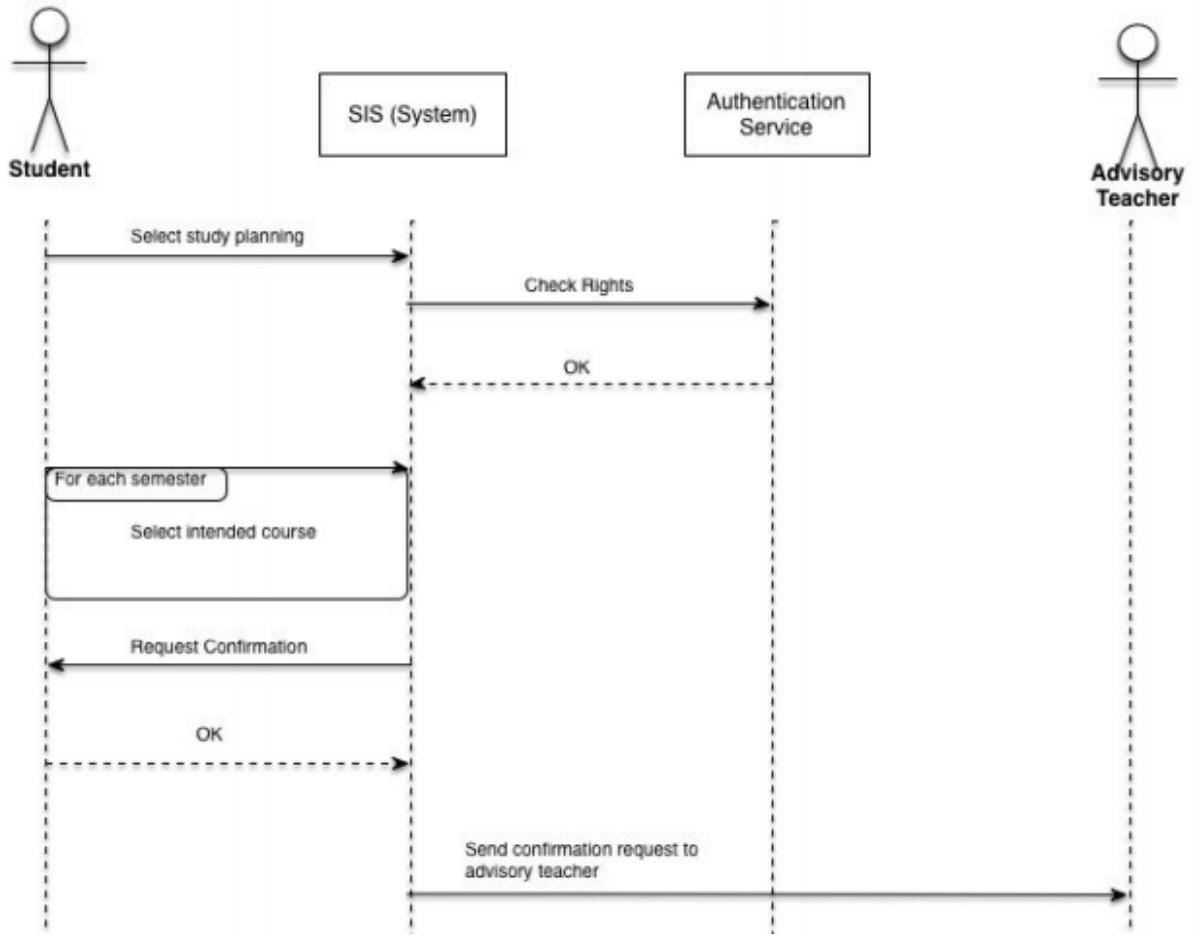
SSD2. Create new Student

What is the student board good for, without any students. In the ssd above, it's shown how management creates a student. The employee starts by navigating to the necessary page. After the Authentication service checks if the user has the correct rights, the user gets redirected. The employee selects a study programme and a group, and follows up by entering all the information for each student. After all the information is entered into the system, the system creates a review page and asks for a confirmation.



SSD3. Review ECTS progress

The third ssd that we'll be discussing concerns the student reviewing his ECTS progress. The student needs a set amount of ECTS points to continue the study and receive a degree for the study. To review this progress is a key element in the motivation of the student. The student begins with navigating to the ECTS progress page. After the authentication service checks the rights of the user, the user is redirected to the page. On this page the student can review the ECTS information.



SSD4. Register Study planning

As last but not least, we have the student registering a study planning. Another big element in the motivation of a student, is planning out his/her study. By quantifying their foreseeable future at the school, they'll realise what they'll work towards. This is known to boost the student's moral and motivation significantly. The student begins by navigating to the study planning page, after the Authentication service checks the rights, the student is redirected. The student continues by choosing a course for each upcoming semester. After filling in all the semesters with the designated courses, the student puts it up for confirmation. The study planning gets reviewed by the advisory teacher, and will be registered into the system.

2.3 Operation Contracts

Sequence diagrams show several actions that map out the functions and the involved actors and supporting actors. Operation Contracts define the functions further. They tell us what parameters are necessary and what kind of state changes are made.

Student registers study planning

Name	Description
Operation	Student commits study planning to system for evaluation by advisory teacher.
Cross References	Student registers study planning
Preconditions	- The delta of the current year and the graduation year on the group associated with the student is -2.
Postconditions	- A new instance of *Study Planning* is created - The new instance is associated with a course - The student is associated with the new *Study Planning* - A *Course* is associated with the new *Study Planning* - The student's advisory teacher is notified about the new *Study Planning* registration.

Course teacher registers grade

Name	Description
Operation	A subject teacher submits the scores of graded tests to the system.
Cross References	Course-Teacher can register grades.
Preconditions	- The teacher is associated with one or more courses.
Postconditions	- A new *result* instance is created - The new instance is associated with an *exam* instance. - The new instance is associated with a *student* instance - The student is notified about the result

Course teacher registers grade

Name	Description
Operation	Management commits new student accounts to the system.
Cross References	Management can create student accounts
Preconditions	- There is at least one study programme in the system - There is at least one group associated with a study programme
Postconditions	- For each student, a new instance of *Student* is created. - All new students are associated with a group

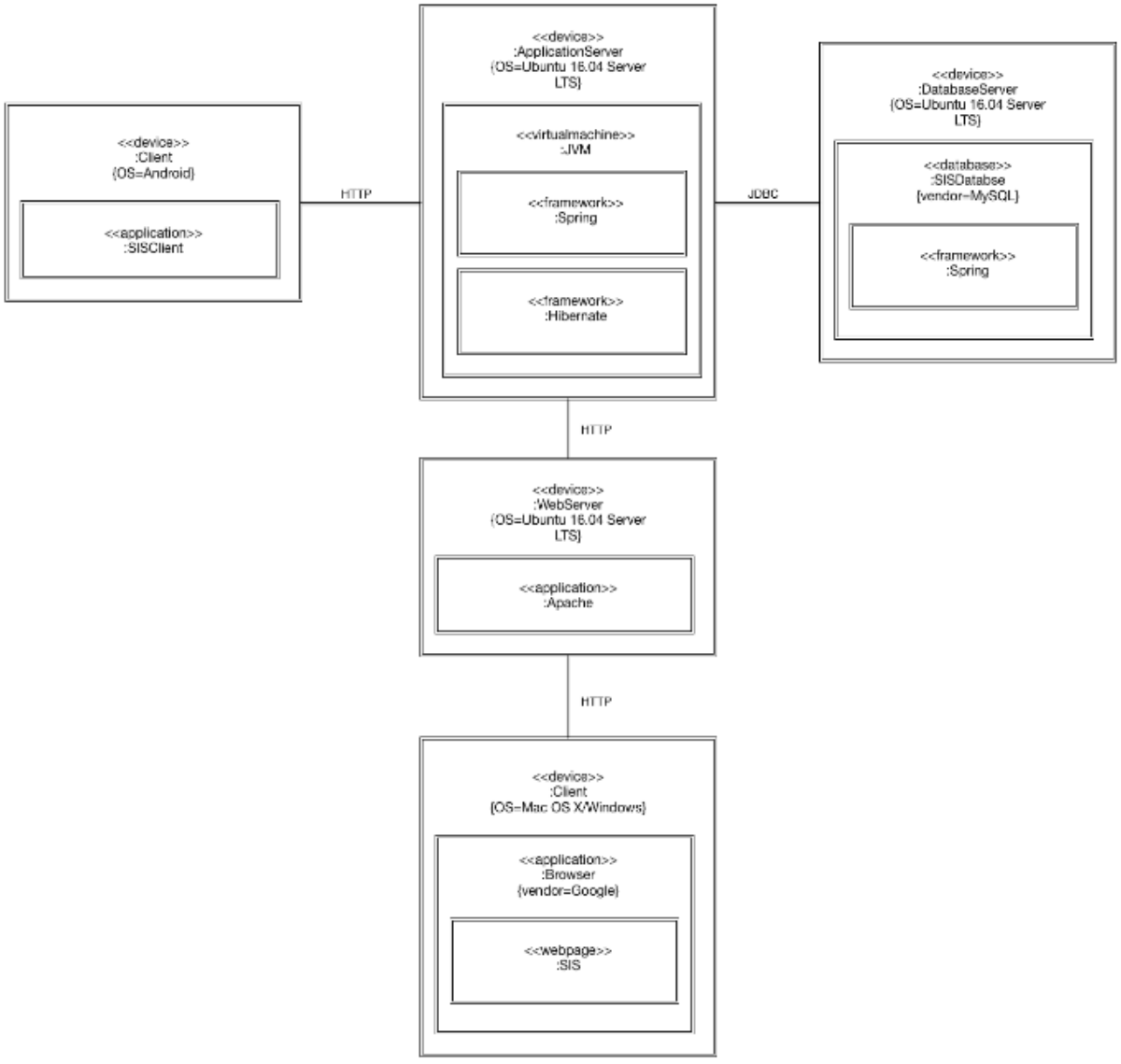
Student can un-register for re-exam

Name	Description
Operation	A student un-registers for a re-exam
Cross References	Student can un-register for re-exam
Preconditions	- The date for the re-exam is not yet available so the student can't select it in the list of exams
Postconditions	- The association between a student and this re-exam is removed

2.4 Logical Architecture or Physical Architecture

For this chapter we've made a deployment diagram. The deployment diagram shows the physical connections that'll be needed for our application. In the diagram below, it is shown that our main application is in the center, it will run on a virtual machine, with two frameworks. Spring is a web framework which allows us to easily form the functions in our application. Besides Spring we also implemented a Hibernate framework which allows structured communication between our application and the database.

Our application is connected to an Android client with an HTTP association, representing a user checking the application on its phone. The connection with the Ubuntu database is a JDBC association. In the database we use MySQL as a language, and implemented the spring framework as well(for the dependencies). Lastly we connected our application with an HTTP association to a web server. The web server acts as a passageway for our application to navigate to the end user's browser. Which is shown in the diagram by the HTTP association to the web server.



2.5 Refine the artifacts of Assignment 1 Vision Nothing changed.

Use-Case Model We changed the subject to course.

Supplementary Specification Nothing changed.

2.6 Glossary Domain Model a diagram showing the conceptual classes of our application. System Sequence Diagrams a diagram showing the interactions between functions and (supporting) actors Operation Contracts a scheme showing the creation or updates of an instance. (based on use cases) Physical or Logical Architecture a diagram showing the physical architecture of our application.

Object Oriented Analysis and Design

Assignment 3

This document is written by Roemer Bakker and Joshua Turpijn for the course 'Object Oriented Analysis and Design' as thought by Riemer van Rozen at the Amsterdam University of Applied Sciences. This is assignment 2. The document is composed as follows:

- **3.1 :** GRASP
- **3.2 :** Improved domain model
- **3.3 :** Design sequance diagram
- **3.4 :** Design class diagrams
- **3.5 :** Prototype Java Implementation

3.1 GRASP

GRASP, stands for**: General responsibility assignment software principles.** These are like patterns throughout the application. Some patterns are very simple and should sound like common knowledge, but some are a little trickier. As software engineers, it's important to know and work with these patterns, so our code will stay readable and interchangeable between programmers. While designing our application we can already start by using the controller and creator patterns.

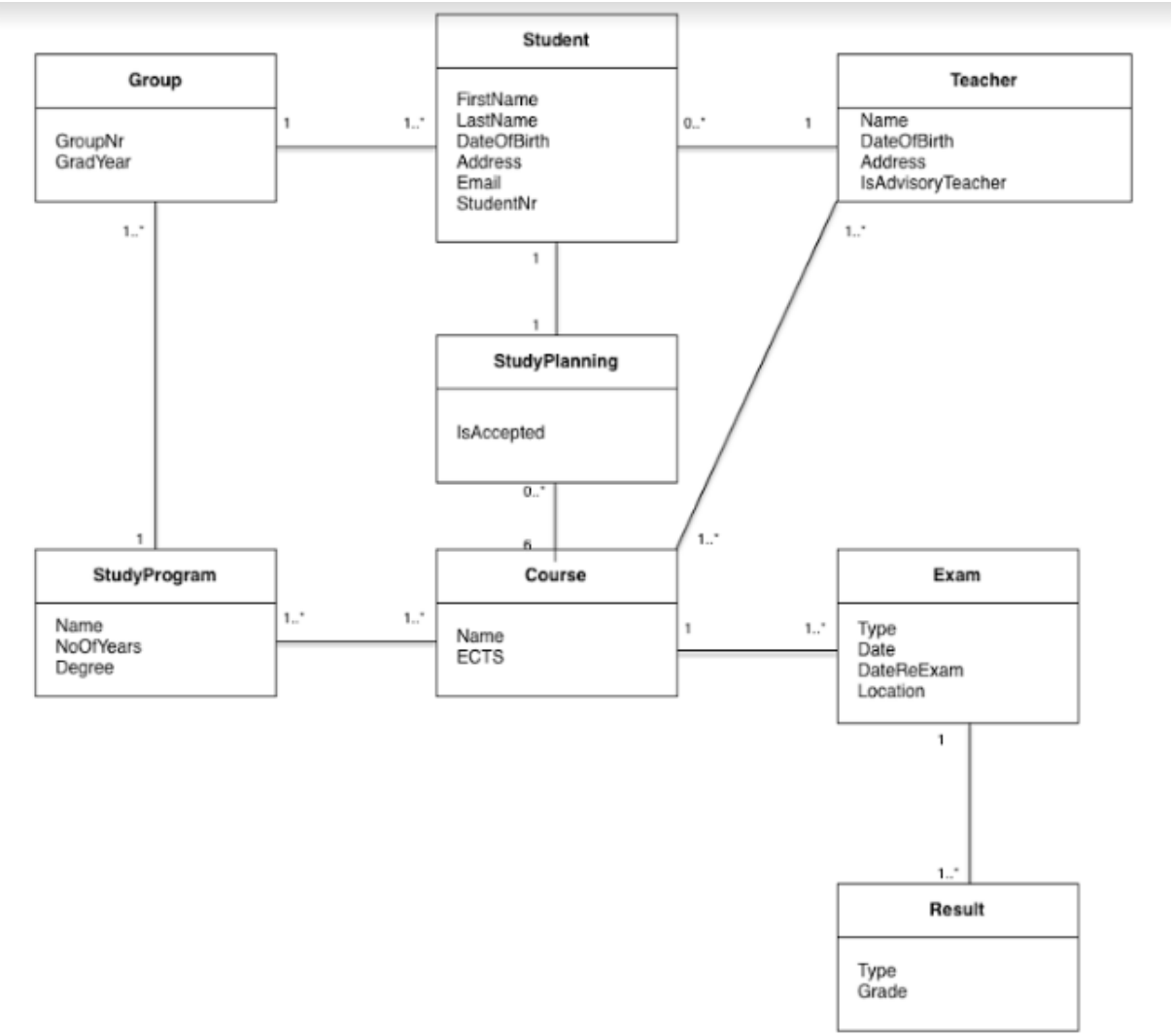
The controller uses a pattern to design our code with(model-view-controller). After that, the creator determines where objects will be created. These two, lay the fundamentals of how our application will operate. We use the controller pattern to create an easy way to interact with our classes, database and front end. The creator pattern is visible in our class diagram. We thought about where we wanted certain objects to be made and build our class diagram in that aspect.

Patterns like low coupling and high cohesive should always lay the groundwork of our code. They maintain logical communication between classes and maintain an order in those classes by grouping code (that affect the same classes). Low coupling, is most visible in our sequence diagrams. Because of low coupling, we need a minimal amount of interactions between the classes. The high cohesive pattern is visible in the final chapter of this document.. This way, even the code needs a minimal amount of referencing.

When it comes to polymorphism, we tend to avoid the ad hoc polymorphism(functing overloading), but focus more on the subtyping kind of polymorphism. We create abstract classes and define specific classes that will inherit the overall functions of the abstract class. This is visible in our class diagram. We created an overall teacher class that has the ability to be an advisory teacher, a subject teacher or both. We intentionally chose not to create 3 different classes because it would disrupt the structured way we build the foundation of our application.

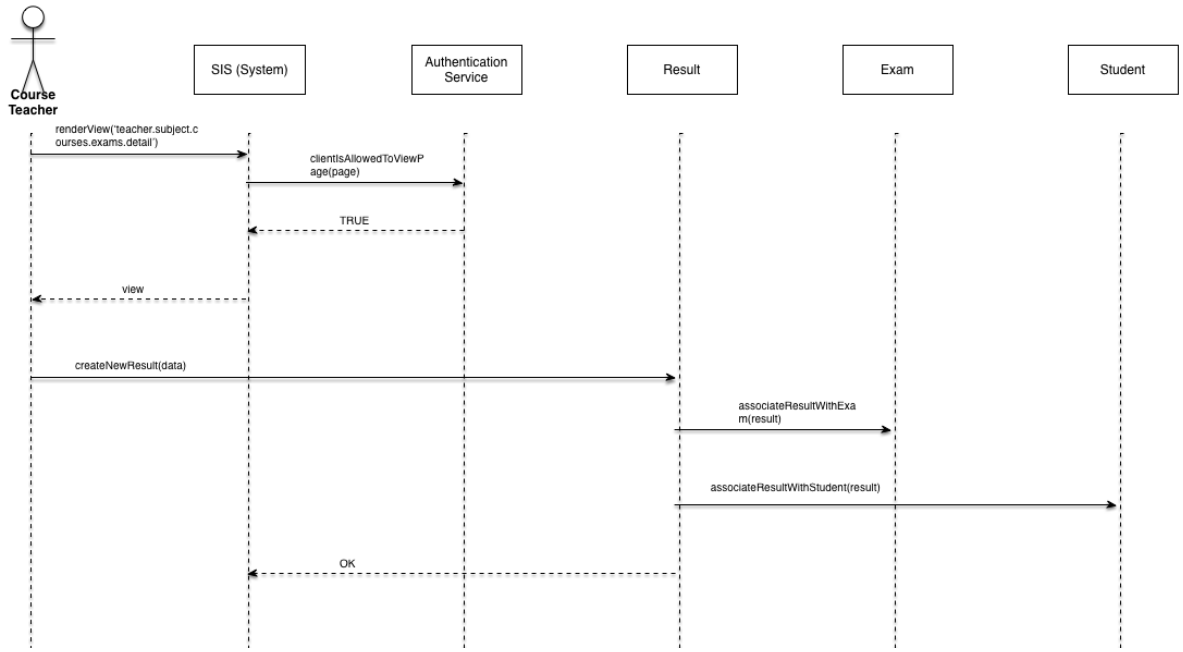
3.2 Improved domain model

The image below, shows our improved domain model. In the previous assignment we had to create a domain model, in this chapter we improved the domain model so our conceptual classes have more structure and it's easier to see what's going on in our application.



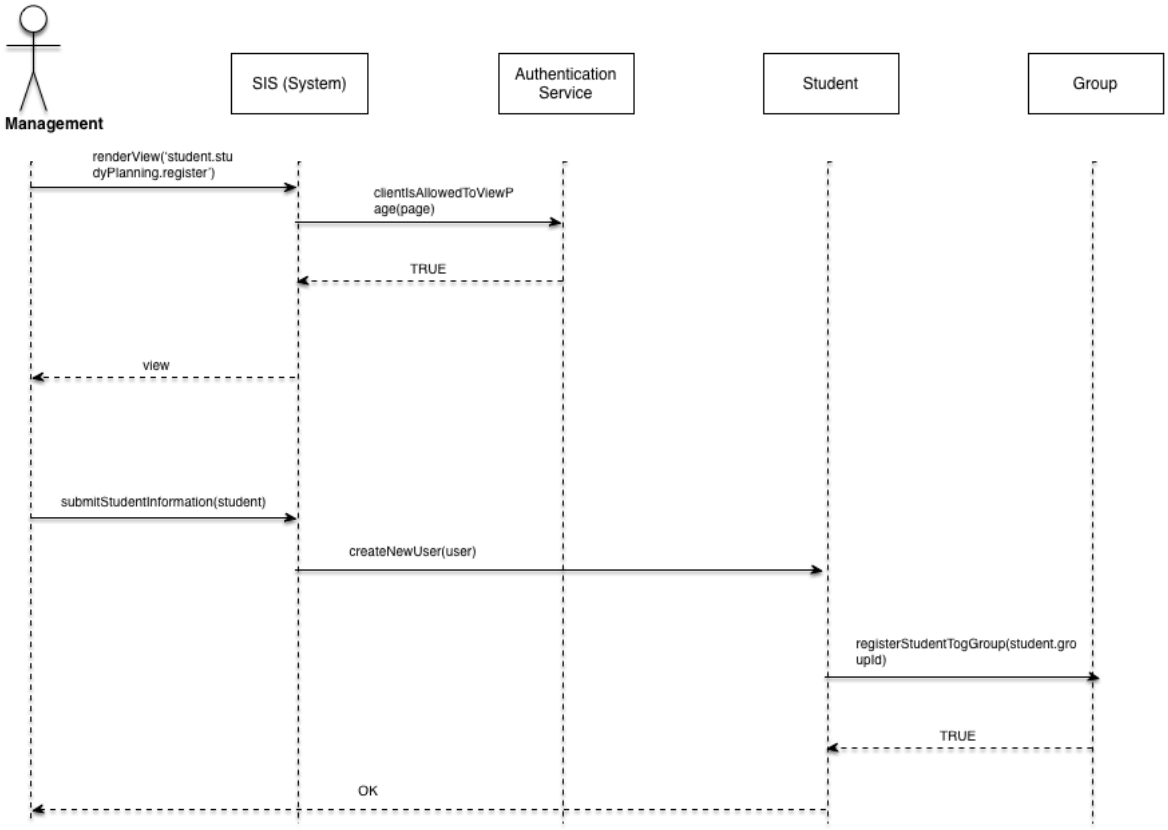
3.3 Design Sequence Diagram

Now that we've shown the system sequence diagrams, we're going to show how the application will work on a design level. The difference lies in the interaction. The system sequence diagram shows the interaction between classes. The design sequence diagram shows the interaction between the user and our system's user interface.



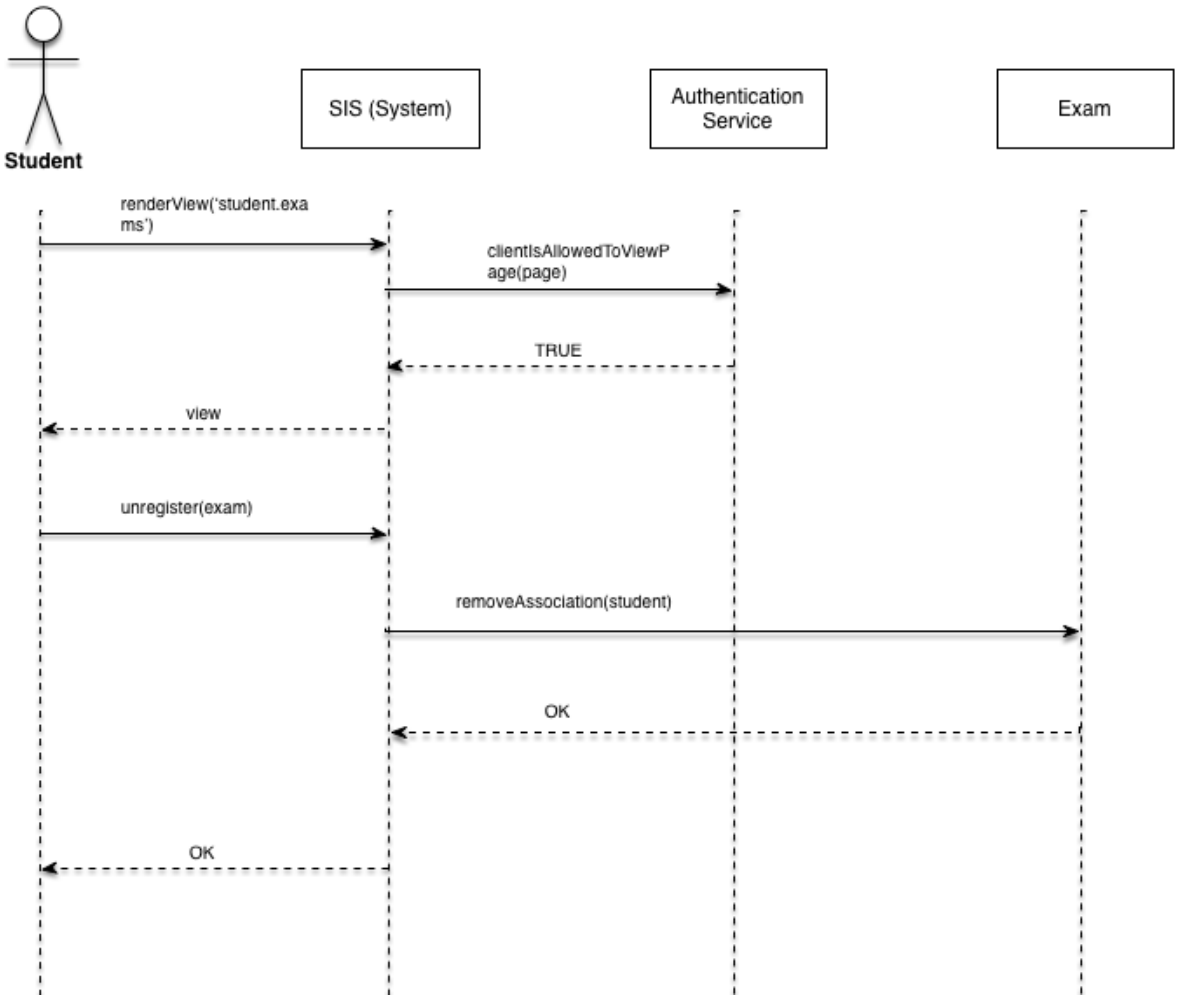
SDD. Course teacher submits the scores of graded tests to the system

First up, we show a SDD containing the logic for a course teacher registering some grades for a test he has just reviewed. To make sure only course teachers can do this, his request is validated by the authentication service before the view for this can be rendered. If the authentication process is successful, a new instance of a result is created. Upon this action, the result is associated with an exam and a student. The teacher is then notified of the success of his action.



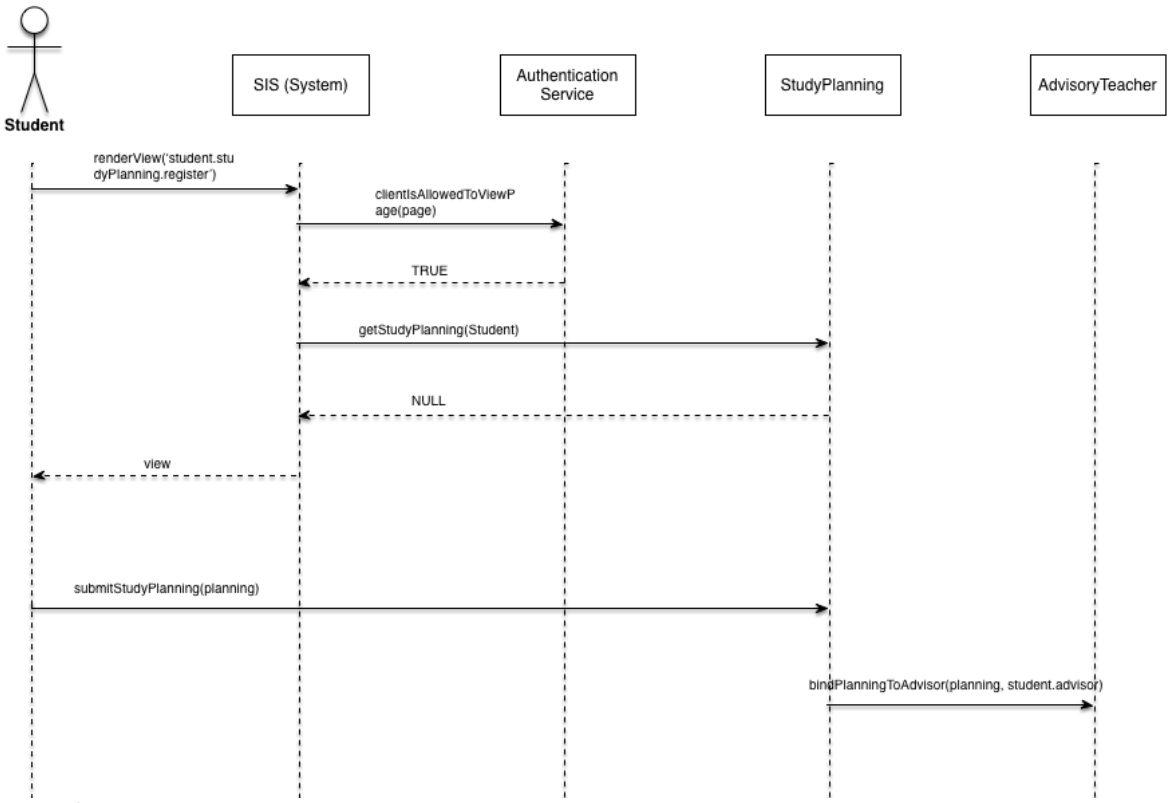
SSD2. Management commits new student accounts to the system

What is a student dashboard, without any students. The SDD above is a description of how management can register new students. First off all, the authentication service is used to check if the request for the view can be authorized. Then, when the management employee submits a request to the system containing data about the new student, a data instance of this is created and associated with the group the employee selected previously. The employee is then redirected to a page notifying him of the success of his action.



SSD3. Student can un-register for a re-exam

The third SDD that will be discussed is the one for the 'Student can un-register for a re-exam' contract. As always, upon requesting to render the view for this page, the authentication service is used to check if the current user is allowed to do so. Then, the user submits a request to un-register for a re-exam. Within the request, the exam is provided. The association between the student and the exam will be removed and the student will be notified of the success of his action.

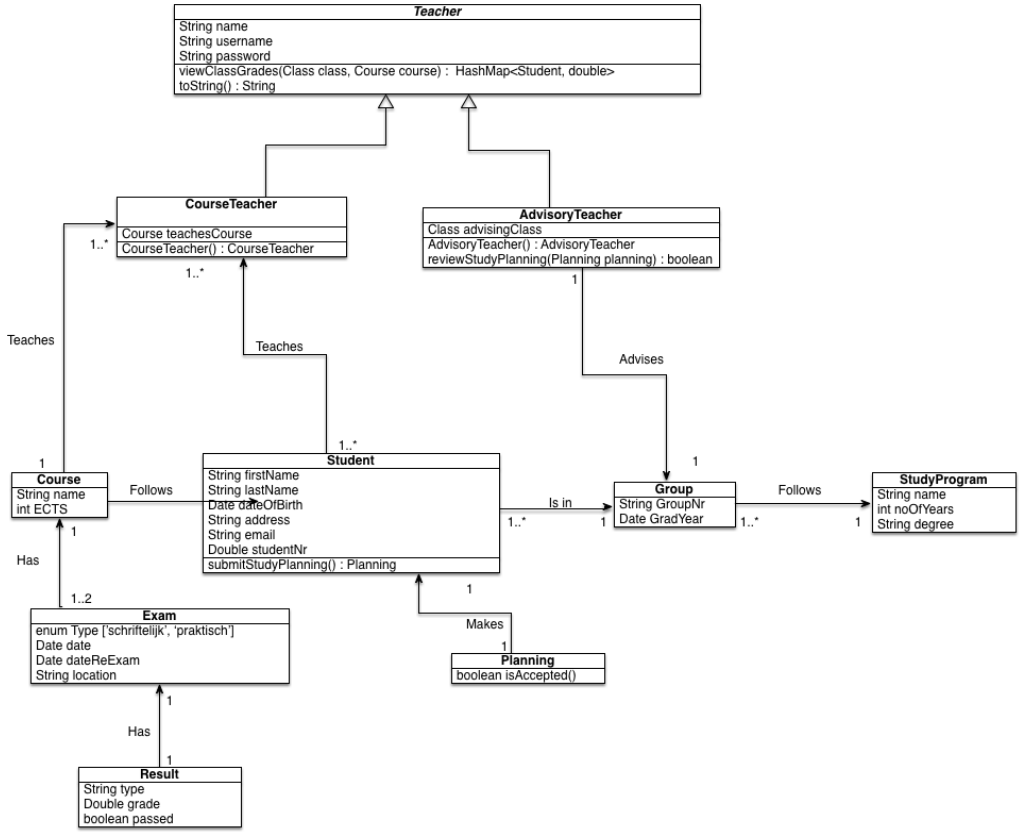


SDD4. Student commits study planning to the system

Lastly, there is the student registering a study planning. Of course, access to this part of the system is first checked by the Authentication Service. If this test passes, the rendered view is send back to the student. The student can then make a request with the appropriate information to register a study planning. When this happens, a new instance of a study planning is initiated and the student's advisory teacher is notified of its creation.

3.4 Design Class diagram

Now that we have the foundation of our application in diagrams. We can shape the structure of our application. We'll do this by creating a class diagram. This diagram will show us what classes contain which objects and contain the functions. This gives us an insight in whether or not we've built a proper foundation. If we run into problems here, it means that we missed something while creating the domain model or system sequence diagrams.


Teacher
String name
String username
String password
viewClassGrades(Class class, Course course) : HashMap<Student, double>
toString() : String
CourseTeacher
Course teachesCourse
CourseTeacher() : CourseTeacher
AdvisoryTeacher
Class advisingClass
AdvisoryTeacher() : AdvisoryTeacher
reviewStudyPlanning(Planning planning) : boolean
Teaches
Teaches
Advises
Course
String name
int ECTS
Follows
Student
String firstName
String lastName
Date dateOfBirth
String address
String email
Double studentNr
submitStudyPlanning() : Planning
Is in
Group
String GroupNr
Date GradYear
Follows
StudyProgram
String name
int noOfYears
String degree
Has
Exam
enum Type ['schriftelijk', 'praktisch']
Date date
Date dateReExam
String location
Makes
Planning
boolean isAccepted()
Has
Result
String type
Double grade
boolean passed

3.5 Prototype Java Implementation

Tis chapter contains 4 classes in which we show how we built our application. We'll put as much clarification in the comments of the code, the rest will be explained underneath the piece of code.
For this document, we'll focus on the structure of the classes, associations and defining of attributes(also Attribute type information). The core methods that'll matter for the sake of explaining (thereby leaving out code like getters and setters). And as last, navigability, how easy is it to navigate through the code without losing focus on what it does or losing track on where the methods lead.

```
/*
* Management can create student accounts
*/

// Study program
StudyProgram program1 = new StudyProgram("HBO-ICT Software Engineering", 4, "BsC");

// Group
Group group1 = new Group(program1, "IS-2015", new Date(2019));

//Students
Student student1 = new Student("Roemer", "Bakker", new Date(1997, 2, 28), "Raasdorperweg 44,
1175KW, Lijnden",
      "roemer.bakker@hva.nl", (double) 500728669, group1);
```

The code above is an example of the usecase 'Management can create student accounts'. Two students are initiated and registered to a group. The outout of the code looks like this:

```
********************************* STUDENT *********************************
Name:                                                    Group
Roemer Bakker                                            HBO-ICT Software Engineering - IS-2015

Date of birth:                                           Address:
Sun Mar 28 00:00:00 CET 3897                             Raasdorperweg 44, 1175KW, Lijnden

Email address                                            Student nr
roemer.bakker@hva.nl                                     5.00728669E8
*****************************************************************************
********************************* STUDENT *********************************
Name:                                                    Group
Joshua Turpijn                                           HBO-ICT Software Engineering - IS-2015

Date of birth:                                           Address:
Thu Sep 13 00:00:00 CEST 3894                            Raasdorperweg 44, 1175KW, Lijnden
```

```
public Student(String firstName, String lastName, Date dateOfBirth, String address, String email, Double studentNr,
               Group group)
{
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
    this.address = address;
    this.email = email;
    this.studentNr = studentNr;
    this.group = group;
}

public Planning submitStudyPlanning(String q3, String q5, String q6, boolean isAccepted)
{
    this.setStudyPlanning(new Planning(q3, q5, q6, isAccepted));

    return this.getStudyPlanning();
}
```

The above piece of code shows the Student class. The code sets the parameters from the main, into the parameters of the class. Also shown in the code is a student submitting a study planning. This is not a system sequence diagram, but it shows more depth into the application.

A student is part of a Group, the code below shows the Group class. The Group has 3 parameters, A studyprogrem, a group number and a graduation year. These are made private so this class is the only place that can access and use the parameters. The last line of code shows how a group number is set.

```
public class Group
{
    private StudyProgram studyProgram;
    private String groupNr;
    private Date gradYear;

    public Group(StudyProgram studyProgram, String groupNr, Date gradYear)
    {
        this.studyProgram = studyProgram;
        this.groupNr = groupNr;
        this.gradYear = gradYear;
    }

    public String getGroupNr() { return groupNr; }
```

The group class uses a studyProgram object. The code below is the StudyProgram class, this code shows how the study progam is made. If only the name of a studyProgram is needed, the last line of code shows how that's used.

```java
public class StudyProgram
{
    public String name;
    int noOfYears;
    public String degree;

    public StudyProgram(String name, int noOfYears, String degree)
    {
        this.name = name;
        this.noOfYears = noOfYears;
        this.degree = degree;
    }

    public String getName() { return name; }
```

A student has exams, and exams have grades. Below are shown a simple java implementation of the exam and grade classes.

```java
public class Exam
{
    private enum Type {SCHRIFTELIJK, PRAKTISCH};
    private Date date;
    private Date dateReExam;
    private String location;

    public Date getDate() { return date; }

    public void setDate(Date date) { this.date = date; }

    public Date getDateReExam() { return dateReExam; }

    public void setDateReExam(Date dateReExam) { this.dateReExam = dateReExam; }

    public String getLocation() { return location; }

    public void setLocation(String location) { this.location = location; }
}
```

```java
public class Result
{
    String type;
    Double grade;
    boolean Passed;

    public String getType() { return type; }

    public void setType(String type) { this.type = type; }

    public Double getGrade() { return grade; }

    public void setGrade(Double grade) { this.grade = grade; }

    public boolean isPassed() { return Passed; }

    public void setPassed(boolean passed) { Passed = passed; }
}
```

3.5 Glossary

Domain Model
a diagram showing the conceptual classes of our application.
System Sequence Diagrams
a diagram showing the interactions between functions and (supporting) actors
Operation Contracts
a scheme showing the creation or updates of an instance. (based on use cases)
Physical or Logical Architecture
a diagram showing the physical architecture of our application.