Laboratory Assignment AND Assessment Requirements Specification

Version 1.0

March, 2024

Developed by:

Anca Nistor, Samuel Zirbo

935

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description of Change | Author | Date |
| V01 | Initial | - | 16.03.2020 |
| V02 | Completion of document | Nistor Anca, Zirbo Samuel | 16.03.2024 |

**Contents**

[Laboratory Assignment AND Assessment Requirements Specification 1](#_Toc161593100)

[Version 1.0 1](#_Toc161593101)

[March, 2024 1](#_Toc161593102)

[1 Functional Requirements 3](#_Toc161593103)

[2 Actors 3](#_Toc161593104)

[3 Use cases – diagram 3](#_Toc161593105)

[3.1 Use case number 1 (Description of the use case) 4](#_Toc161593106)

[3.2 Use case number 2 (Description of the use case) 4](#_Toc161593107)

[3.3 Use case number 3 (Description of the use case) 4](#_Toc161593108)

[3.4 Use case number 4 (Description of the use case) 5](#_Toc161593109)

[3.5 Use case number 5 (Description of the use case) 5](#_Toc161593110)

[3.6 Use case number 6 (Description of the use case) 5](#_Toc161593111)

[3.7 Use case number 7 (Description of the use case) 6](#_Toc161593112)

[3.8 Use case number 8 (Description of the use case) 6](#_Toc161593113)

[3.9 Use case number 9 (Description of the use case) 7](#_Toc161593114)

[4 Analysis 7](#_Toc161593115)

[4.1 Entities 7](#_Toc161593116)

[4.2 Relations between entities 7](#_Toc161593117)

[4.3 Attributes 7](#_Toc161593118)

[4.4 System behavior 7](#_Toc161593119)

[4.4.1 Use case 1-2-3 7](#_Toc161593120)

[4.5 System events 7](#_Toc161593121)

[5 Design 8](#_Toc161593122)

**Analysis and design Document**

# Functional Requirements

List the functional requirements (FR) of the system.

|  |  |
| --- | --- |
| Section/ Requirement ID | Requirement Definition |
| FR1.0 | Add a new student |
| FR1.1 | Remove a student |
| FR1.1.1 | Update student |
| FR1.1.1.1 | Print all students |
| FR2.0 | Add a new assignment |
| FR2.1 | Remove an assignment |
| FR2.1.1 | Extend an assignment’s deadline |
| FR2.1.1.1 | Print all assignments |
| FR3.0 | Add a grade |
| FR3.1 | Print all grades |

# Actors

Teacher

# Use cases – diagram



## Use case number 1 (Description of the use case)

Actors: teacher

Description: create a new student

Precondition: - all fields are specified

Postcondition: - a new student was added in the list

|  |  |
| --- | --- |
| Action | System Response |
| 1. Completes the necessary fields for adding |  |
|  | 1. Checks if everything is alright, adds a new element in the list if so |
|  | 1. If the input is invalid, throws an exception |

Exceptions: When the fields aren’t filled.

## Use case number 2 (Description of the use case)

Actors: teacher

Description: delete student

Precondition: - valid id belonging to an existing student is specified

Postcondition: - the student with the specified id is removed from the list

|  |  |
| --- | --- |
| Action | System response |
| 1. Give an id as input |  |
|  | 1. Checks if it is a valid id and there is a student with that id and deletes the student |
| 3 - | 1. If the input is invalid, throws an exception |

## Use case number 3 (Description of the use case)

Actors: teacher

Description: update student

Precondition: - valid id belonging to an existing student and all other fields for student are specified

Postcondition: - the student with the specified id has the data updated

|  |  |
| --- | --- |
| Action | System response |
| 1. Give an id and all other fields for the Student entity as input |  |
|  | 1. Checks if it is a valid id and there is a student with that id, than checks if the rest of the input is valid, and updates the data for that student |
| 3 - | 1. If the input is invalid, throws an exception |

## Use case number 4 (Description of the use case)

Actors: teacher

Description: create a new assignment

Precondition: - valid id which does not belong to any other assignment and all the other fields are specified

Postcondition: - the assignment with the specified fields has been created

|  |  |
| --- | --- |
| Action | System response |
| 1. Give an id and all other fields for the Assignment entity as input |  |
|  | 1. Checks if it is a valid id and there is no other assignment with that id, than checks if the rest of the input is valid, and creates the assignment |
| 3 - | 1. If the input is invalid, throws an exception |

## 3.5 Use case number 5 (Description of the use case)

Actors: teacher

Description: extend the deadline of an assignment

Precondition: - valid id which belongs to an assignment and number of extended weeks specified

Postcondition: - the assignment’s deadline has been changed

|  |  |
| --- | --- |
| Action | System response |
| 1. Give the assignment’s id and the number of weeks added to the deadline as input |  |
|  | 1. Checks if there is an assignment with such id and adds the number of weeks to the deadline |
| 3 - | 1. If the input is invalid, throws an exception |

## 3.6 Use case number 6 (Description of the use case)

Actors: teacher

Description: add a grade

Precondition: - valid id of the student and id of the assignment

Postcondition: - the assignment’s deadline has been changed

|  |  |
| --- | --- |
| Action | System response |
| 1. Give the student’s and assignment’s id, the grade, the feedback and the week the assignment has been handed in |  |
|  | 1. Checks if there is an assignment and a student with such ids. If an assignment is turned in late, 2.5 points are deducted from the grade for each week of delay. At the end, it adds the new grade. |
| 3 - | 1. If the input is invalid, throws an exception. If there is no student or assignment with the given ids, a message is printed. |

## Use case number 7 (Description of the use case)

Actors: teacher

Description: print students

Precondition: -

Postcondition: - all the available students are printed

|  |  |
| --- | --- |
| Action | System response |
|  | 1. Prints the students with all the information: studentId, name, group, email, professor name. |

## Use case number 8 (Description of the use case)

Actors: teacher

Description: print assignments

Precondition: -

Postcondition: - all the available assignments are printed

|  |  |
| --- | --- |
| Action | System response |
|  | 1. Prints the assignments with all the information: id, description, deadline, assignation date. |

## Use case number 9 (Description of the use case)

Actors: teacher

Description: print grades

Precondition: -

Postcondition: - all the available grades are printed

|  |  |
| --- | --- |
| Action | System response |
|  | 1. Prints the grades with all the information: id(studentId, assignmentId), value, deliver date, feedback. |

# Analysis

## Entities

Student, Assignment, Grade

## Relations between entities

One student can have multiple assignments and one assignment can be assigned to many students. It is a many-to-many relationship between the two classes. Class Grade has as id, a pair consisting of studentId and assignmentId and it is the association class between the Student and Assignment classes.

## Attributes

Student: id, name, group, email, professor name

Assignment: id, description, deadline, assignation date

Grade: id(studentId, assignmentId), value, deliver date, feedback

## System behavior

## Use case 1-8

The system will act as a subsystem to a larger environment, in order to speed up a certain process in the company’s workflow.

## System events

After each operation a message is shown to the user either if the command terminated succesfully or with an error message.

# Design

* 1. **Class diagram**

****

* 1. **Sequence diagrams (for each use case, without error handling)**
* **Add Student Sequence Diagram**

****

* **Delete Student Sequence Diagram**

****

* **Update Student Sequence Diagram**

****

* 1. **GRASP**

GRASP is set of exactly 9 **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns:

1. Information Expert: Assign responsibilities to the class with the most information needed to fulfill them.
2. Creator: Delegate responsibility for creating objects to a class that logically belongs to it.
3. Controller: Assign responsibility for managing and coordinating activities within a system.
4. Low Coupling: Minimize dependencies between classes to increase flexibility and maintainability.
5. High Cohesion: Ensure that classes have a narrow, focused responsibility to improve readability and maintainability.
6. Indirection: Introduce an intermediary to decouple classes and improve flexibility.
7. Polymorphism: Allow objects of different types to be treated interchangeably, enabling flexibility and extensibility.
8. Pure Fabrication: Introduce artificial classes to promote low coupling and high cohesion.
9. Protected Variations: Encapsulate the variations in the system to protect the integrity of the design against future changes.