Assignment 1

Student: Raț Gabriel Cătălin

**Group:30432**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 4

4. UML Sequence Diagrams 6

5. Class Design 6

6. Data Model 7

7. System Testing 8

8. Bibliography 8

1. Requirements Analysis

# Assignment Specification

In this assignment we are required to design and implement an application that has as main usage case the management of students in faculty. The main actors are the professors and the students who can log in using an account & password authentication method.

# Functional Requirements

*The application is required to provide the following functionalities for the:*

*-STUDENT:*

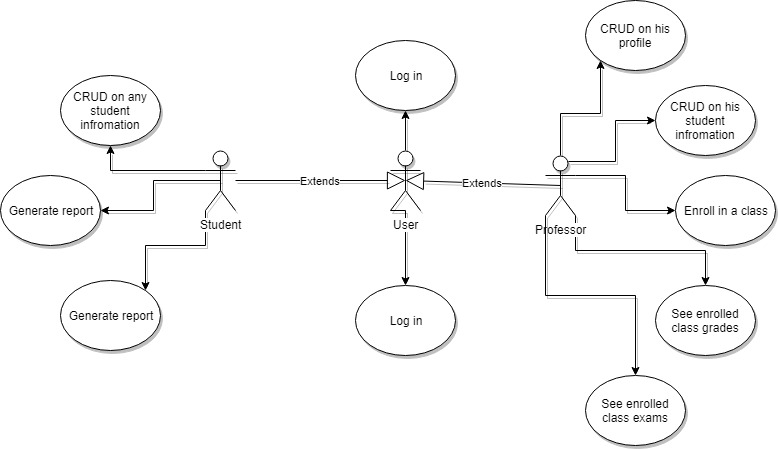
* *CRUD operations on its information (name, phone number, identity card number, personal numerical code, address)*
* *CRUD operations on its student information ( identification number, group, enrolments, grades)*
* *See class enrolment information like grades and exams*

*-PROFESSOR:*

* *CRUD operations on any student information ( identification number, group, enrolments, grades)*
* *Generate reports for a period containing the activities performed by a student*

# Non-functional Requirements

* *Security – each type of user must have access only to its meant functionalities*
* *Usability – easy to understand and use user interface*
* *Response time – under 0.5 seconds*
* *Portability – perk of the java programming language*
* *Robustness – the system can cope with erroneous input because all the input data is validated*

2. Use-Case Model

*Use case: Student enrolls in a class*

*Level: user-goal level*

*Primary actor: student*

*Main success scenario: After a student logged in he/she presses the enroll button and a dropdown with the possible classes shows up. The student chooses one and click the ok button. Now the user is enrolled in and can see the exam dates and the possible grades after an exam passes.*

*Extensions: The chosen class is full, and the student will see a warning letting him know this class is unavailable for the moment.*

3. System Architectural Design

**3.1 Architectural Pattern Description**

* *Layered Architectural Pattern*



*Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific rules associated with the request.*

*Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data. It only needs to display that information on a screen format.*

*Similarly, the business layer doesn’t need to be concerned about how to format the customer data for display on a screen or even where the customer data is coming from. it only needs to get the data from the persistence layer, perform business logic against the data (e.g. calculate values or aggregate data or validate), and pass that information up to the presentation layer.*

* *Model-View-Presenter pattern*

*Like MVC, MVP is based on three components: the model, the view, and the presenter.*

*Model*

*The model represents the logic of the view. This can also be the business logic. However, all functionality must be accessible via the model in order to operate the view. The model is controlled solely by the presenter. The model itself knows neither the view nor the presenter.*

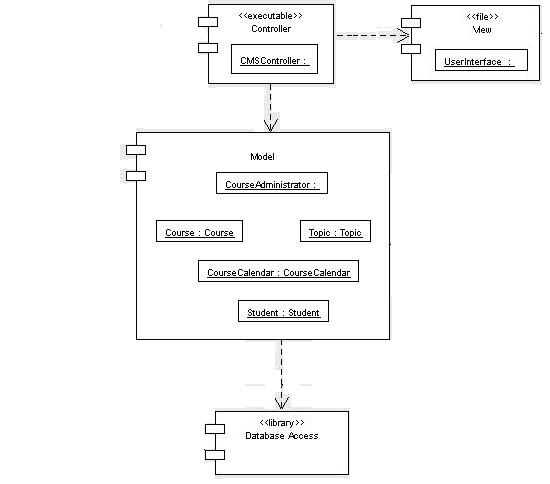
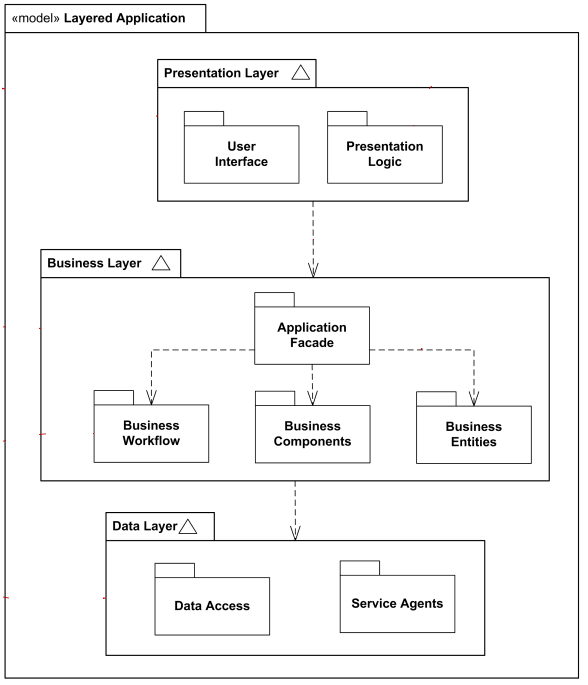
*View*

*The view contains no controlling logic and is solely responsible for the representation and the inputs and outputs. It gives neither access to the functionality of the presenter nor to the model. All control of the view is done by the presenter.*

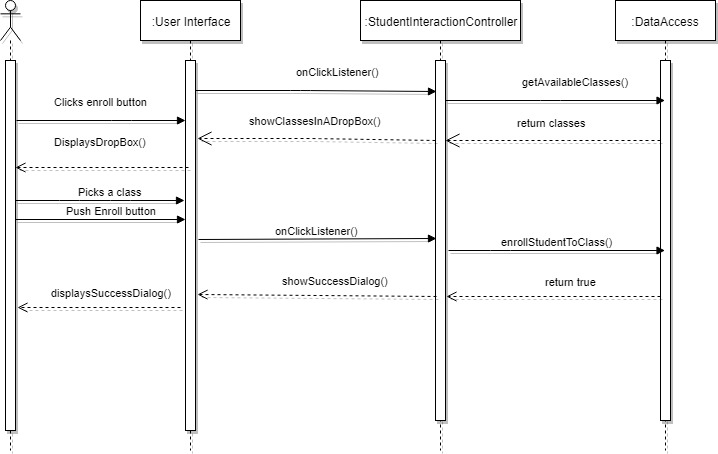
*Presenter*

*The presenter is the link between model and view. It controls the logical processes between the other two layers and ensures that the view can fulfill its functionality.*

**3.2 Diagrams**

****

4. UML Sequence Diagrams



5. Class Design

**5.1 Design Patterns Description**

* **Observer design pattern**

*I am going to use Observer pattern to keep in sync the UI with the information inside the database.*

*The observer pattern allows objects (observer, observing object) to register with another object (subject, observed object) and from then on, be informed of this as soon as it changes.*

*For the Observer a uniform interface with at least one update method is defined. This is called by the subject in the case of updates and is in most cases parameterized with more detailed data for the change. Concrete observers implement the interface and thus the update method and thus determine how the observer should react to the notification.*

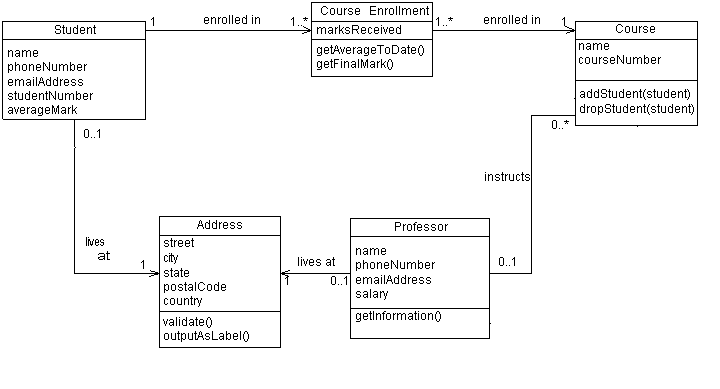
*The subject needs administration methods for Observer to log in and out. When an observer logs on, the subject includes it in its list of objects to be notified. If changes to the subject state occur, all registered observers are informed (notifyObservers ()). This is done by iterating over the subject's observer list and invoking the update method () on each observer.*

* *Dependency injection*

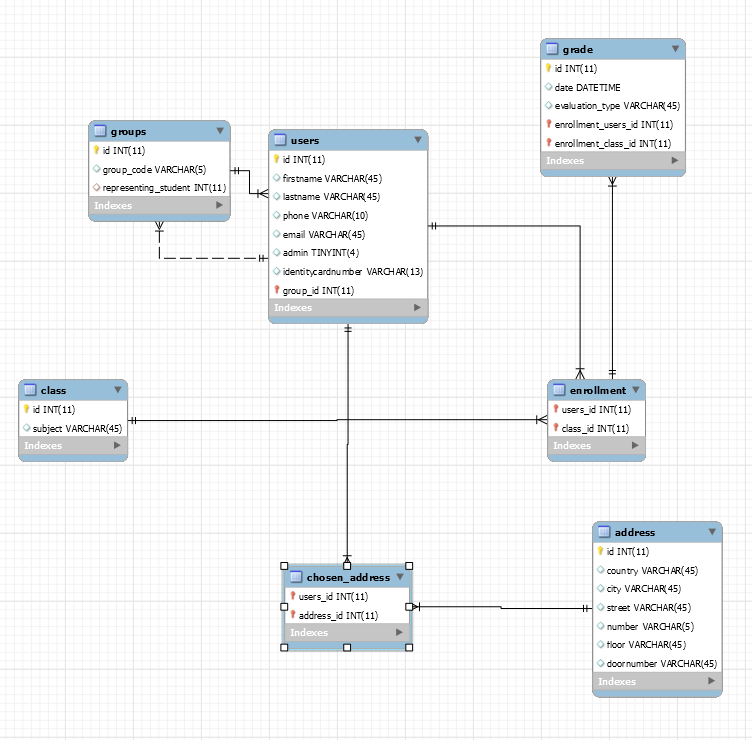
*Java Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement dependency injection in java to move the dependency resolution from compile-time to runtime.*

*This can also help when unit testing. In this exact case I will be injection the view and the data access in the controller. This way I will be able to unit test the controller.*

**5.2 UML Class Diagram**

**

6. Data Model



7. System Testing

* Unit testing

The unit test is particularly important because it considers the test object in isolation and thus precludes interactions with other components. Therefore, occurring error effects can be clearly attributed to the tested software module, which may severely limit the search for the defect as the cause of the error.

The aim of the unit test is to check whether the software component fulfills the functional and non-functional requirements specified in the specification.

Functional requirements are defined as those requirements that specify the input or output behavior about the correctness of the delivered results.

By contrast, non-functional requirements include, for example, performance aspects or memory consumption during runtime.

* Integration testing

Integration tests check the cooperation of multiple system parts of increasing complexity from individual modules to subsystems up to the overall system.

The goal of the integration test is to check the interaction of different parts of a system. In doing so, both the correct interaction, such as the exchange of data through messages or shared memory, access to databases or the use of functionality through calls for interface functions beyond individual parts, must be checked, as well as the non-occurrence of undesired effects. Errors typically detected include incorrect use of interfaces, unauthorized parameter values, but also shared resource or race-conditional blocks due to unordered changing of shared data to inconsistent data states. Integration tests are performed because of the predominantly internal system focus of the creator of the software.

8. Bibliography

<https://en.wikipedia.org/wiki/Non-functional_requirement>

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>

<https://sourcemaking.com/design_patterns/observer>