Assignment A1

**Student Management System**

Student: Ieremiaș Viorel

**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 4

3. System Architectural Design 4

4. UML Sequence Diagrams 5

5. Class Design 5

6. Data Model 5

7. System Testing 5

8. Bibliography 5

# Requirements Analysis

## Assignment Specification

The application will be used in the management of the educational process in the CS Department at the Technical University of Cluj-Napoca. The application will have two major types of users - students and teachers.

All users will have the possibility to view their profiles and to modify their personal information. Additionally, it will allow students to manage their enrollment into coursers and to see the schedule of the exams and their grades. Teachers will be able to view student profiles and its enrollment into courses, course information and the list of students enrolled in a course. They will also be able to add new courses and the edit the information of the courses they teach.

Both students and teachers have to provide a username and a password in order to use the application. In order to obtain an account in the system, users will go through a registration process, that will require filling in information like name, identity card number, personal numerical code, address, phone number, and email.

## Functional Requirements

All users can perform the following operations:

* add/update/view account information
* see course information
* see exam planification
* see another user’s profile

Students will also be able to:

* enroll in a course
* see their grades

Teachers will have the ability to:

* add new course
* modify existing course info
* set exam date
* see list of students enrolled in a course

## Non-functional Requirements

* Accessibility – the system must provide a visual interface that will provide easy access to all the functions
* Availability – the system must be operational 24/7, except in cases when planned and announced maintenance procedures are underway
* Capacity – the system must accommodate around 1000 users and must serve 10000 requests per month in the agreed performance parameters
* Extensibility – the system must easily accept extensions, like including all the faculties in TUCN in the same system, or new features, like the possibility to be enrolled in 2 faculties at the same time or the implementation of a basic internal messaging system
* Performance – the system must have a response time of at most 1s
* Platform compatibility – the system must be compatible will all platforms that have installed a Java runtime environment, version 6 or higher
* Safety – the system must provide only a simple, one-step authentication, but constraints must be enforced on the strength of the passwords
* Scalability – the system will be able to accommodate another 1000 users with at most 10% loss of performance; this criterion applies up to a number of 10000 users

# Use-Case Model

Use case: Enroll in a course

Level: User-goal level

Primary actor: Student

Main success scenario:

* Login
* Access View Courses
* Select desired course
* Press Enroll to course

Extensions:

In case the student is not eligible to enroll in that course, a message will be displayed with the reason of the error.

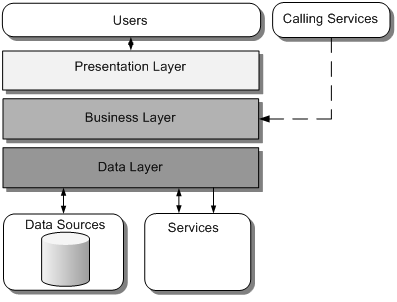
# System Architectural Design

## Architectural Pattern Description

In terms of high level architecture, the application is structured in layers, based on the principles of the layered architecture. The layered architecture consists in separating the components of the system that perform similar functions into isolated groups which share information inside the layer they form, but only expose through an interface the communication with other layers.

The architecture is in principle linear in the sense that one layer uses the functions of the layer immediately beneath itself and data passes through the layers and in processes in a pipeline. The advantage stands in higher cohesion and lower coupling, which in terms means that the system is easier to maintain, test and extend. The disadvantage is that there may be layers in which some data is not processed too much or is not processed at all, which affects performance for no gain.

## Diagrams



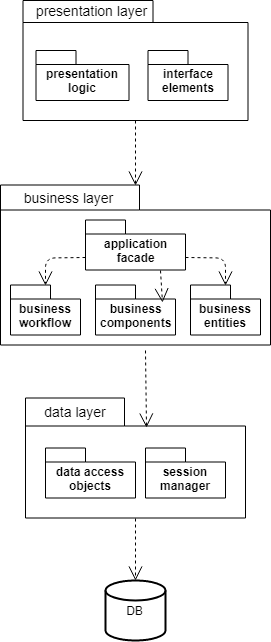
*Layered architecture*

The layered architecture diagram presents the general layers in the application, layers that abstract the main processing steps applied to the state.

The data layer holds the utility classes and components that are responsible of retrieving and updating information in the database. If the application also offers services to externals clients, the service agents are also included in the data layer.

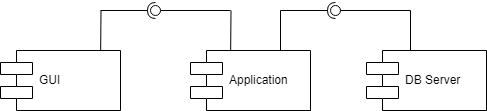
The business layer holds the business logic of the application, which includes the business entities (model of the application), the workflows that represent the processing of the model objects and the components responsible of carrying out the processing from the business point of view. The application façade is a class that encompasses the capabilities that the business layer exposes to the presentation layer or external services to use.

The presentation layer is represented by the set of UI components and the classes responsible of the handling input events coming from the user and the processing done in the presentation layer, which includes validation and transmission of input data to the business layer.

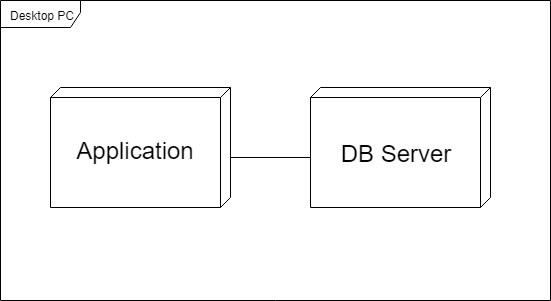
*Package diagram*

Since the application is a simple one, and doesn’t rely on external dependencies, there is not a large number of components. There is a DB Server, abstracted as a database connector, which is used by the application to handle the communication with the database, by means of a set of methods defined in interface.

Another component is the GUI. The user interface is implemented as desktop windows and is populated with data coming from the business layer of the application that also defines in an interface a set of capabilities.

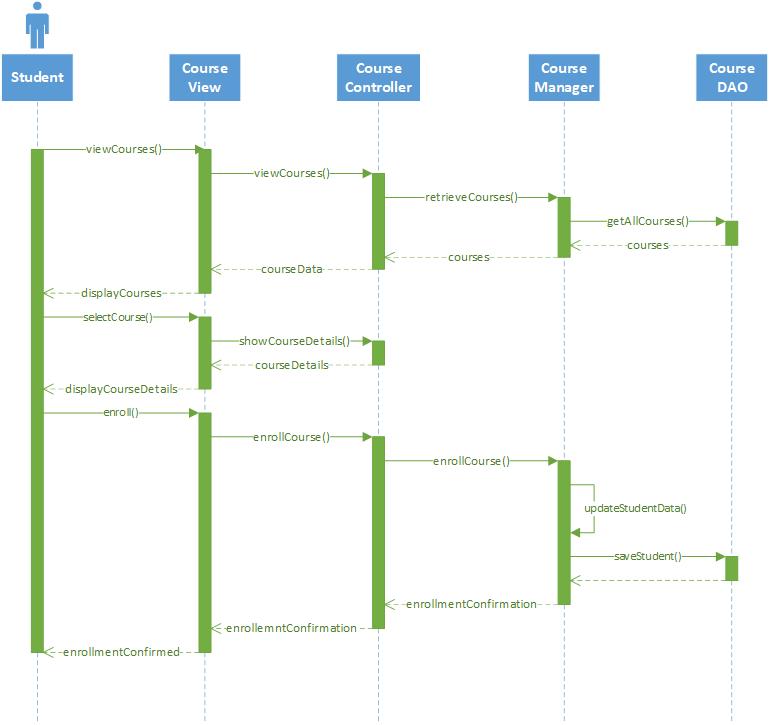
**

*Component diagram*



*Deployment diagram*

# UML Sequence Diagrams



# Class Design

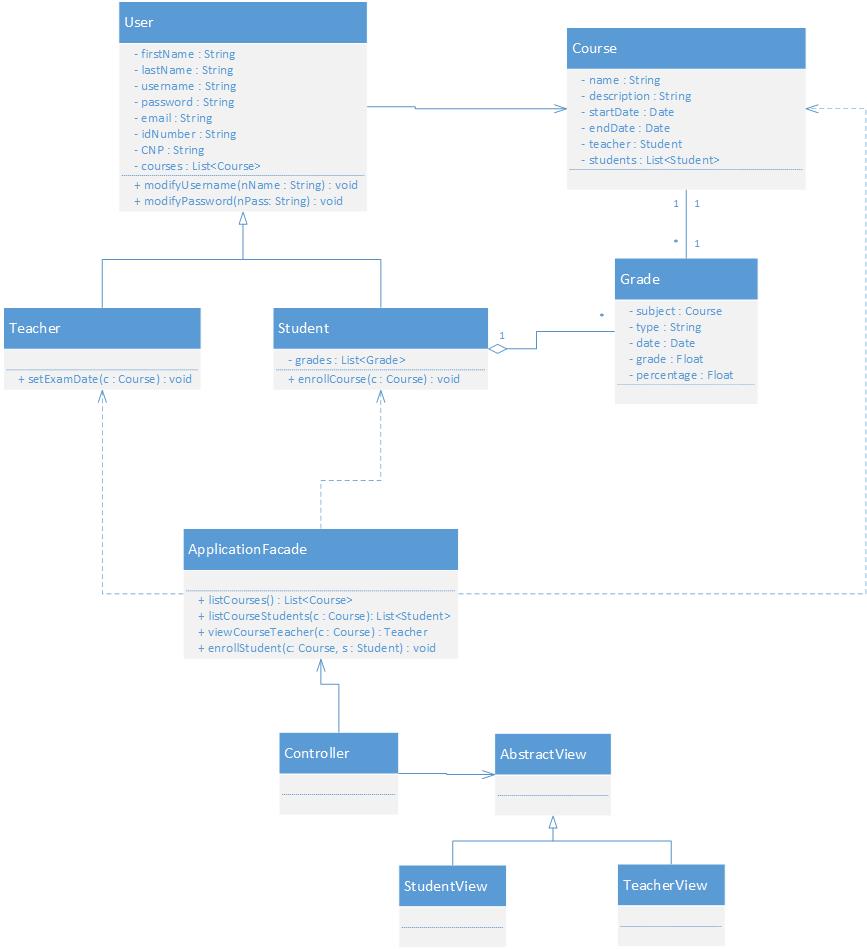
## Design Patterns Description

From the use-case diagram it can be noticed that the two main actors, Student and Teacher have a set of functions that is common to both of them. As a result, we can abstract out the common functionality in super classes that encapsulated parts of the model, meaning attributes like name, address, email, phone number, unique numerical code etc. or parts of the view, like viewing courses, viewing a user’s profile or editing own account.

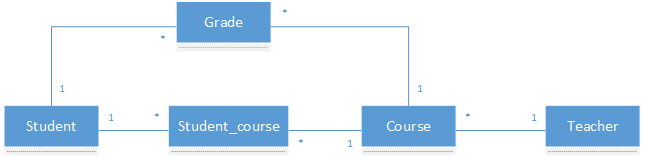
Also, façade pattern is used at the level of the business layer in order to abstract out the implementation details and externalize a simpler view of the system’s capabilities.

Finally, an implementation of the observer pattern, where the Course is the Observable and the Students are the Observers is used to notify and update the students when a change occurs in the state of the course.

## UML Class Diagram



# Data Model



# System Testing

JUnit tests will be performed on the model classes of Student and Teacher and on the application façade, to make sure the basic operations are performed correctly.

# Bibliography

<https://www.oreilly.com/ideas/software-architecture-patterns/page/2/layered-architecture>

<https://en.wikipedia.org/wiki/Observer_pattern>

<http://www.agiledata.org/essays/dataModeling101.html>

<https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose>