Assignment A2

**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 8

5. Class Design 8

6. Data Model 10

7. System Testing 10

8. Bibliography 10

# Requirements Analysis

## Assignment Specification

The application is used in the management of the educational process in the CS Department at the Technical University of Cluj-Napoca. The application has two types of users: students and teachers.

All users have the possibility to view their profiles and to modify their personal information after they have created an account. Additionally, students are able to manage their enrollment into coursers and to see the schedule of the exams and their grades. Teachers can view course information, the list of students enrolled in a course or the profile of a student and the courses in which he/she is enrolled. Teachers can also 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. The registration process is carried at the time when the teacher is employed at the university or the student starts his studies, so it is not performed directly by and is transparent to the user. However, the registration process requires information like first and last name, identity card number, address, phone number, and email. The generated credentials (username and password) are handed out to the user after the registration and can be changed from inside the application at any time.

For easy access and high availability, the application is deployed as a simple and lightweight web service.

## 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

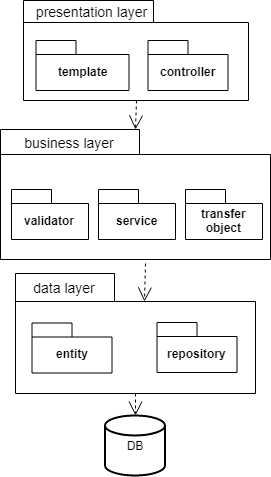
## Architectural Pattern Description

In terms of high level architecture, the application is structured in layers, based on the principles of the layered architecture. The principle of the layered architecture is to separate 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. Such a system behaves like a linear pipeline of modules where each layer uses the functions of the layer immediately beneath itself and data passes through the layers being processed at each step. The difference is that data flows in both directions, either from the data source towards the user, or from the user, who has access to input mechanisms, towards the data source.

The advantage is that the layers are decoupled, while inside the layers, cohesion is high, making the system more stable and easier to extend, maintain and test. 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. Also, such a system is more complex and more difficult to design.

Furthermore, this architecture is well suited for web-based applications, since it allows a clean separation of responsibilities between the server and the client. In this case, processing and rendering is accomplished server-side, while the browser clients only display that view and pass the user inputs.

## Diagrams

**

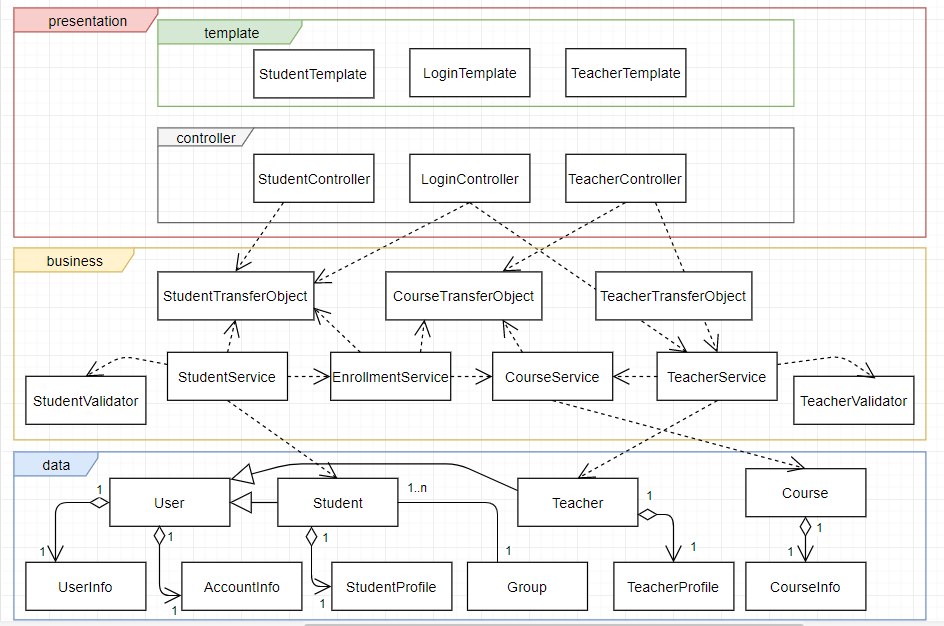
*Architecture diagram*

The architecture diagram presents the layers of the application that abstract out the main processes and operations.

The data layer is concerned with retrieving data from the data source, in this case a database and storing in objects that can be processed by the next layer, so it contains entities and repositories.

The business layer holds the business logic of the application, which includes the services that implements the workflows and the helper classes, like the validators and the transfer objects that are used as intermediary containers when passing data to and from the user interface.

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



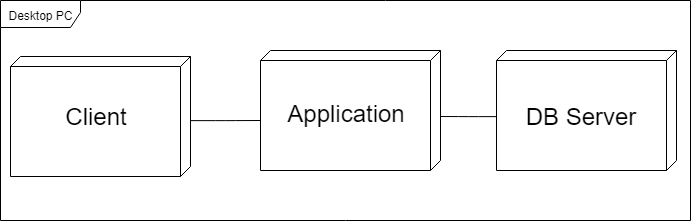
*Package diagram*

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.

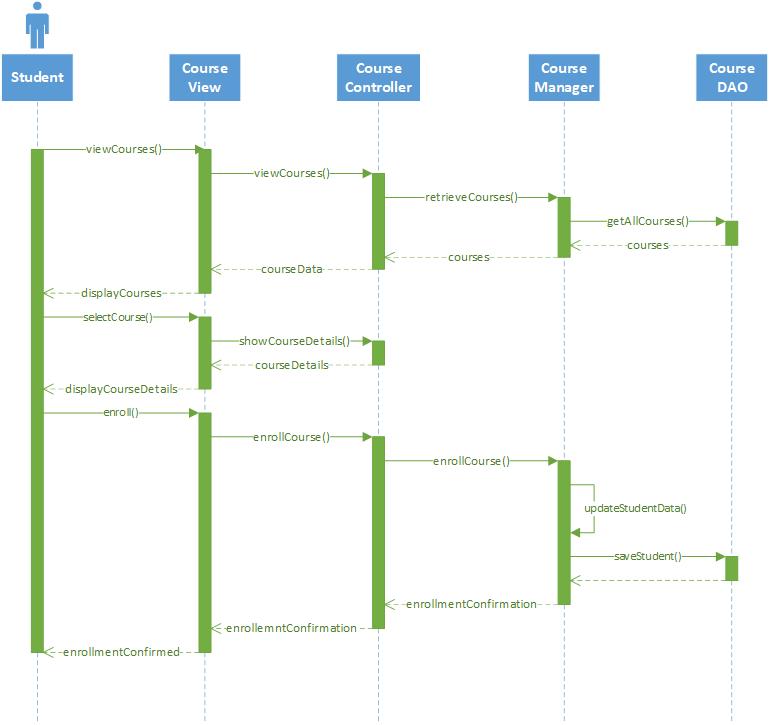
**

*Component diagram*

**

*Deployment diagram*

# UML Sequence Diagrams



# Class Design

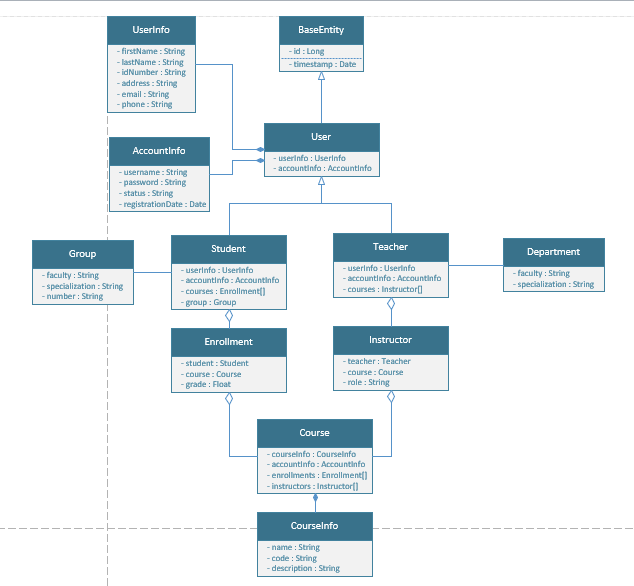
## Design Patterns Description

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.

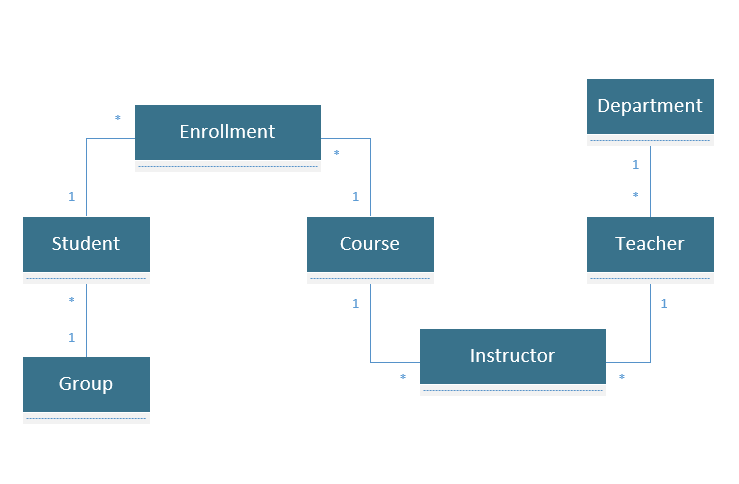
Finally, an implementation of the observer pattern, where the Course is the Observable and the Students are the Observers is used to notify and update the students when a change occurs in the state of the course.

## UML Class Diagram



*Class diagram*

# Data Model



*Data model diagram*

# 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>