The management of students in the CS Department

Student: Pop Raluca

**Group: 30233**

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

3.1 Architectural Pattern Description 4

3.2 Diagrams 5-6

4. UML Sequence Diagrams 7

5. Class Design 7

5.1 Design Patterns Description 7-8

5.2UML Class Diagram 8

6. Data Model 8-9

7. System Testing 9

8. Bibliography 9

1. Requirements Analysis

# Assignment Specification

Design and implement a Java application for the management of students in the CS Department at TUCN. The application should have two types of users (student and teacher/administrator user) which have to provide a username and a password in order to use the application.

The regular user can perform the following operations:

- Add/update/view client information (name, identity card number, personal numerical code, address, etc.).

- Create/update/delete/view student profile (account information: identification number, group, enrolments, grades).

- Process class enrolment (enroll, exams, grades).

**Requirements**

- Create the analysis and design document

- Implement and test the application.

# Functional Requirements

To login on the application, every user has a username and a password to provide its identity. Depending on the user who is logged in, it will appear a different interface which has the possibility to do CRUD operations on it. The admin/ teacher user can update a students grade or delete him and the student can see his grades or change different personal information.

The requirements for the student:

- Create/update/delete/view on own profile (account information: identification number, group, enrolments, grades).

- Process class enrolment (enroll, exams, grades).

The requirements for the admin/teacher:

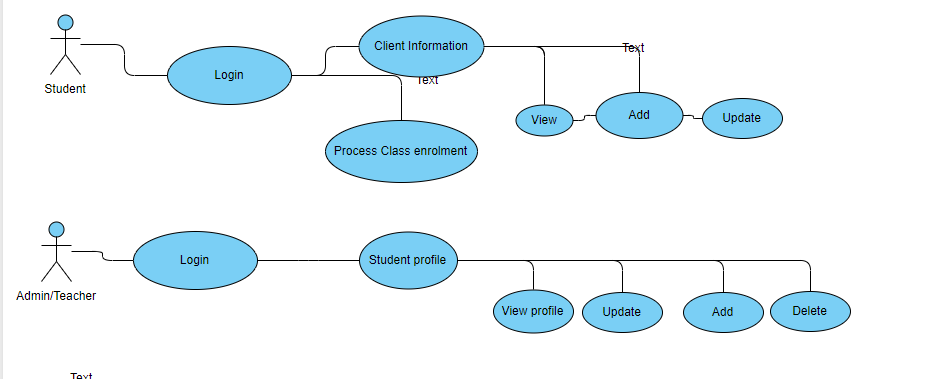
- CRUD on students information.

- Generate reports for a particular period containing the activities performed by a student

# Non-functional Requirements

This application will use MySql as a relational database, which should update when a change appears. Also, the passwords will be stored in the database.

2. Use-Case Model



Use-Case description format:

Use case: Login

Level: Summary level

Primary actor: Admin/Teacher

Main success scenario: The actor can modify Student’s profile

Extensions: scholarships, school situation of the last years

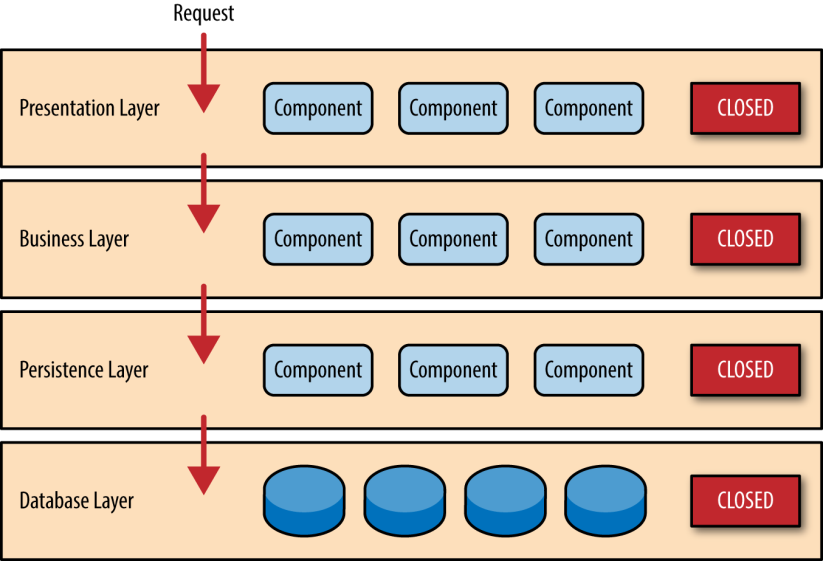
3. System Architectural Design

**3.1 Architectural Pattern Description**

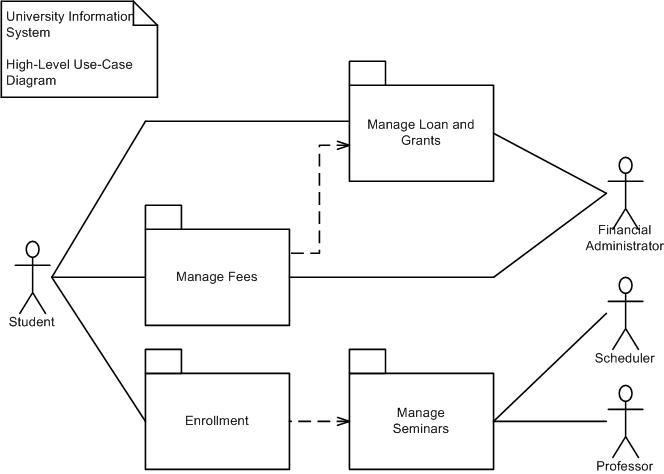
The following applications design is based on layered architecture pattern, otherwise known as the n-tier architecture pattern.  The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.

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 business rules associated with the request.

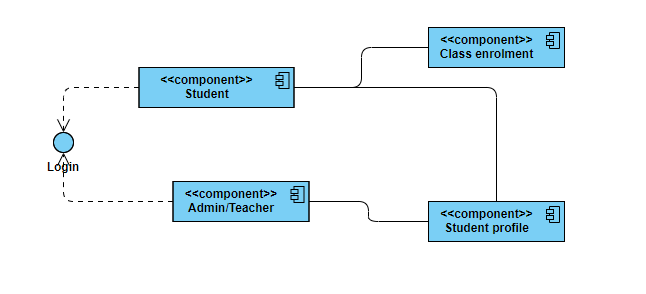
**3.2 Diagrams**



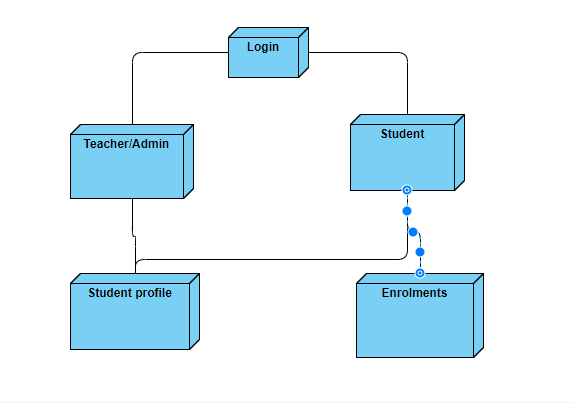
* **Presentation layer**. This layer contains the user oriented functionality responsible for managing user interaction with the system, and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer.
* **Business layer**. This layer implements the core functionality of the system, and encapsulates the relevant business logic. It generally consists of components, some of which may expose service interfaces that other callers can use**.**
* **Data layer**. This layer provides access to data hosted within the boundaries of the system, and data exposed by other networked systems; perhaps accessed through services. The data layer exposes generic interfaces that the components in the business layer can consume.



Package diagram

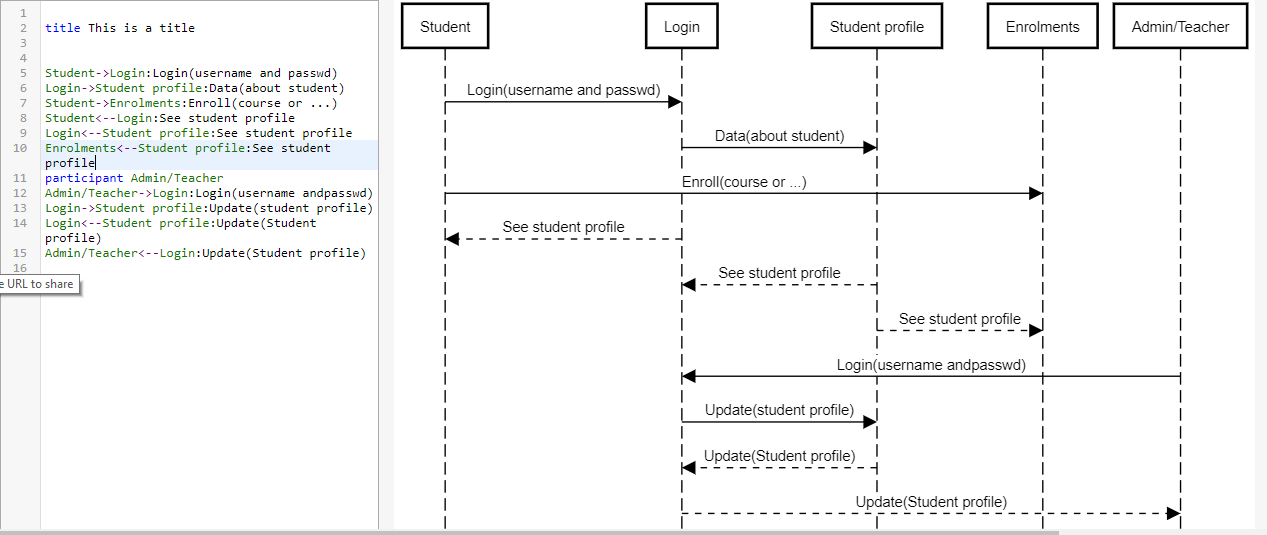


Component Diagram



Deployment diagram

4. UML Sequence Diagrams



5. Class Design

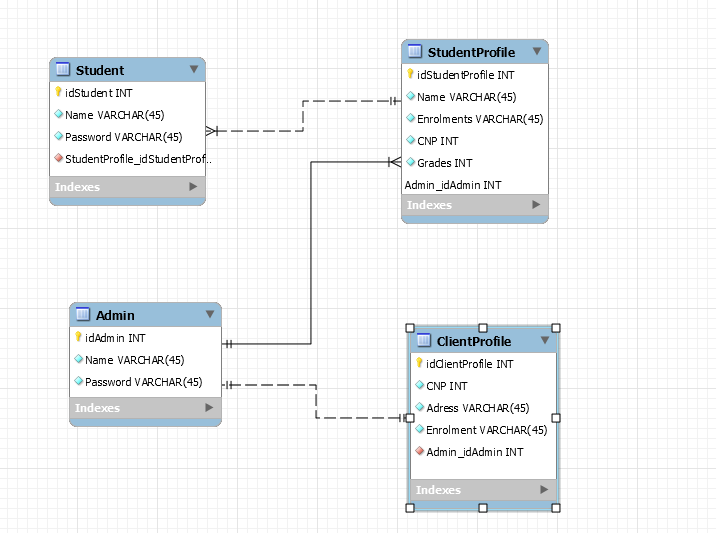
**5.1 Design Patterns Description**

|  |  |
| --- | --- |
| [Abstract factory](https://en.wikipedia.org/wiki/Abstract_factory_pattern) | Provide an interface for creating *families* of related or dependent objects without specifying their concrete classes. |
| [Builder](https://en.wikipedia.org/wiki/Builder_pattern) | Separate the construction of a complex object from its representation, allowing the same construction process to create various representations. |
| [Dependency Injection](https://en.wikipedia.org/wiki/Dependency_injection) | A class accepts the objects it requires from an injector instead of creating the objects directly. |
| [Factory method](https://en.wikipedia.org/wiki/Factory_method_pattern) | Define an interface for creating a *single* object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses. |
| [Lazy initialization](https://en.wikipedia.org/wiki/Lazy_initialization) | Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed. This pattern appears in the GoF catalog as "virtual proxy", an implementation strategy for the [Proxy](https://en.wikipedia.org/wiki/Proxy_pattern" \o "Proxy pattern)pattern. |
| [Multiton](https://en.wikipedia.org/wiki/Multiton_pattern) | Ensure a class has only named instances, and provide a global point of access to them. |
| [Object pool](https://en.wikipedia.org/wiki/Object_pool_pattern) | Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of [connection pool](https://en.wikipedia.org/wiki/Connection_pool) and [thread pool](https://en.wikipedia.org/wiki/Thread_pool) patterns. |
| [Prototype](https://en.wikipedia.org/wiki/Prototype_pattern) | Specify the kinds of objects to create using a prototypical instance, and create new objects from the 'skeleton' of an existing object, thus boosting performance and keeping memory footprints to a minimum. |
| [Resource acquisition is initialization](https://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization) (RAII) | Ensure that resources are properly released by tying them to the lifespan of suitable objects. |
| [Singleton](https://en.wikipedia.org/wiki/Singleton_pattern) | Ensure a class has only one instance, and provide a global point of access to it. |

**5.2 UML Class Diagram**

*[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]*

6. Data Model



The data model contains four tables: Student, Admin, StudentProfile and ClientProfile.

7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

8. Bibliography

<https://online.visual-paradigm.com/tutorials/use-case-diagram-tutorial/>

<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

<https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff650706(v=pandp.10)>

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

<https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiGlfPxj5DhAhVD-6QKHZSBC-oQjRx6BAgBEAU&url=http%3A%2F%2Fagilemodeling.com%2Fartifacts%2FpackageDiagram.htm&psig=AOvVaw1dHSmwc8klkrwvYKrk8LQ9&ust=1553150786444276>