Management of students from UTCN

Analysis and Design Document

Student: Olar Paul Dan

**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 3

3. System Architectural Design 4

4. UML Sequence Diagrams 6

5. Class Design 7

6. Data Model 8

7. System Testing 8

8. Bibliography 9

1. Requirements Analysis

# Assignment Specification

Design and implement a Java application for the management of students in the CS Department at UTCN. 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).

The administrator user can perform the following operations:

- CRUD on students information.

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

# Functional Requirements

The application should 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).
  + Generate reports for a particular period containing the activities performed by a student

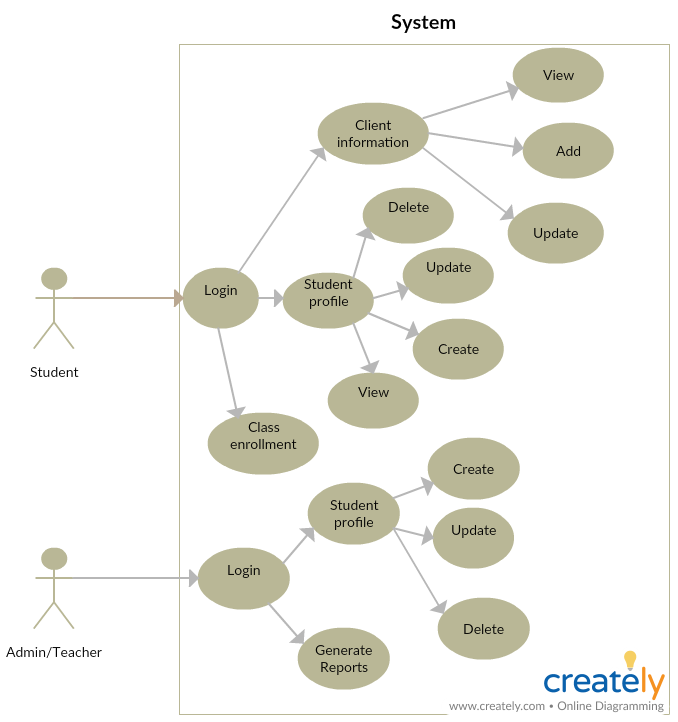
# Non-functional Requirements

The system will be secured using https encrypted connections. Also we will demand user authentication and will not keep passwords in plain text. Other user data will not be encrypted as we do not find it as being sensible information.

The data will be stored in a relational database. The non-functional requirements of the Java application are:

* Availability
* Performance
* Usability
* Testability

2. Use-Case Model



Use case: Selection of the operations an user can do

Level: User-goal

Primary actor: Student or Admin/Teacher

Main success scenario: The student register successfully after he choose the type of user and he login into his account. He choose one of the operations he can do. The system receive the information and analyse it and display or do the user’s request.

Extensions: Another scenario could be user can’t login into his account or the login fails and he can’t access the operations he want to do.

3. System Architectural Design

**3.1 Architectural Pattern Description**

The application design is based on layered architecture pattern. This pattern is a client-server architecture in which presentation, application processing, and data management functions are physically separated.

Components within the layered architecture pattern are formed into horizontal layers, each layer performing a different act within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not define the number and types of layers that must exist in the pattern, most layered architectures consist of four basic layers: presentation, business, persistence, and database.

**3.2 Diagrams**

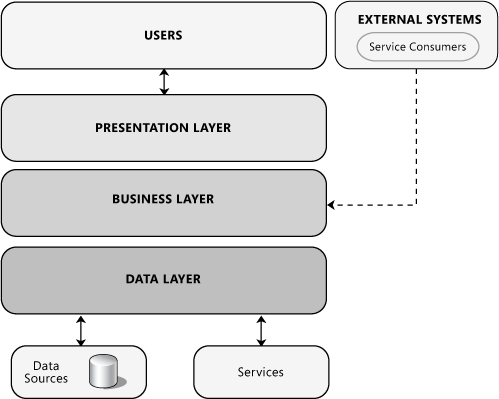


Figure 1 The logical architecture view of a layered system

* **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.

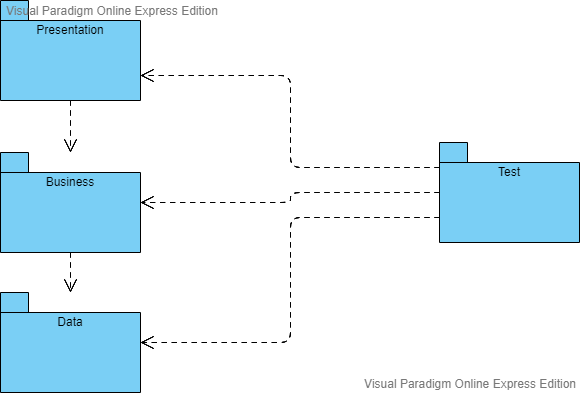


Figure 2 Package diagram

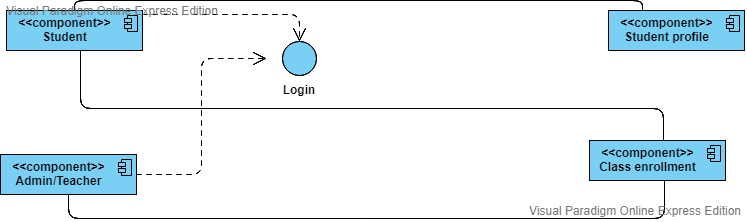
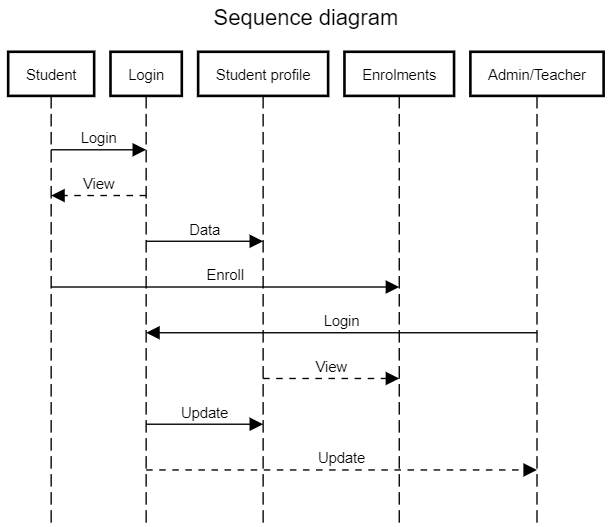


Figure 3 Component Diagram

4. UML Sequence Diagrams

A sequence diagram simply depicts interaction between objects in a sequential order, the order in which these interactions take place.



5. Class Design

**5.1 Design Patterns Description**

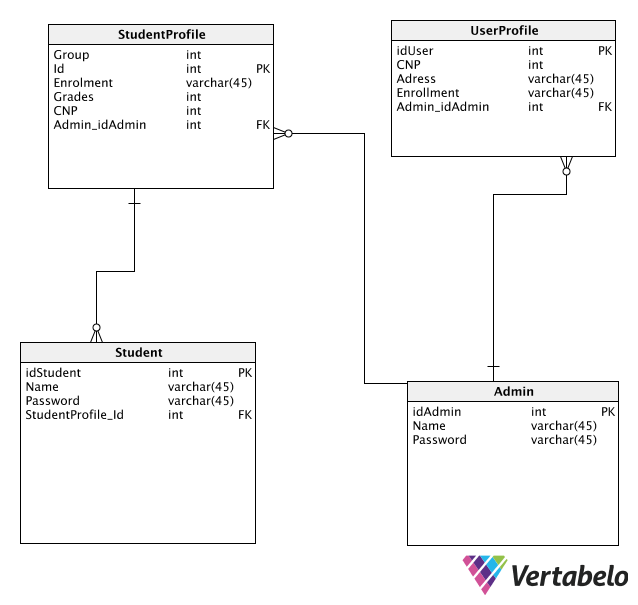
In the implementation of this application will use some design patterns. For example we will use the design patterns such as: SINGLETON, FACTORY METHOD or OBSERVER.

* **SINGLETON:**  the singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system. The term comes from the mathematical concept of a singleton.
* **FACTORY METHOD:** the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.
* **OBSERVER:** The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

**5.2 UML Class Diagram**

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

6. Data Model



7. System Testing

A good test case design technique is crucial to improving the quality of the software testing process. This helps to improve the overall quality and effectiveness of the released software. Following are the test case design techniques to ensure high-quality of the released software.

In the application I am about to develop I will use:

* Boundary Value Analysis (BVA): this technique is applied to explore errors at the boundary of the input domain
* Equivalence Partitioning (EP): - the test input data is partitioned into a number of classes having an equivalent number of data. This helps to reduce the number of test cases.

8. Bibliography

* <https://github.com/utcn-cs-software-design-tudorv-2019/LabInfo/blob/master/assignments/Assignment1_2019_TudorV.pdf>
* <https://en.wikipedia.org/wiki/Non-functional_requirement>
* <https://en.wikipedia.org/wiki/Observer_pattern>
* <https://en.wikipedia.org/wiki/Software_design_pattern>
* <https://en.wikipedia.org/wiki/Singleton_pattern>
* <https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/>
* <https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658109%28v%3dpandp.10%29>
* <https://online.visual-paradigm.com/tutorials/package-diagram-tutorial/>
* <https://reqtest.com/testing-blog/test-case-design-techniques/>