Assignment A1

Student: Prața Mirela

**Group:30233**

Table of Contents

Contents

[1. Requirements Analysis 3](#_Toc3932953)

[1.1 Assignment Specification 3](#_Toc3932954)

[1.2 Functional Requirements 3](#_Toc3932955)

[1.3 Non-functional Requirements 3](#_Toc3932956)

[2. 2. Use-Case Model 4](#_Toc3932957)

[3. 3. System Architectural Design 4](#_Toc3932958)

[3.1 Architectural Pattern Description 5](#_Toc3932959)

[3.2 Diagrams 5](#_Toc3932960)

[4. UML Sequence Diagrams 7](#_Toc3932961)

[5. Class Design 7](#_Toc3932962)

[5.1 Design Patterns Description 7](#_Toc3932963)

[5.2 UML Class Diagram 7](#_Toc3932964)

[6. Data Model 8](#_Toc3932965)

[7. System Testing 8](#_Toc3932966)

[8. Bibliography 9](#_Toc3932967)

# 1. Requirements Analysis

## Assignment Specification

The Java application implements the management of students in the CS Department at TUCN. The application has two types of users (student and teacher user) which have to provide a username and a password in order to use the application.

## Functional Requirements

There are two types of users: the regular user (student) and the administrator user (teacher).

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 student’s information.

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

The data is stored in a relational database (MySql). The Layers architectural pattern is used to organize the application. The Data Access Layer is created by using SQL statements.

All the inputs of the application are validated against invalid data before submitting the data and saving it in the database.

## Non-functional Requirements

Availability – system must be available all the time, for both types of users

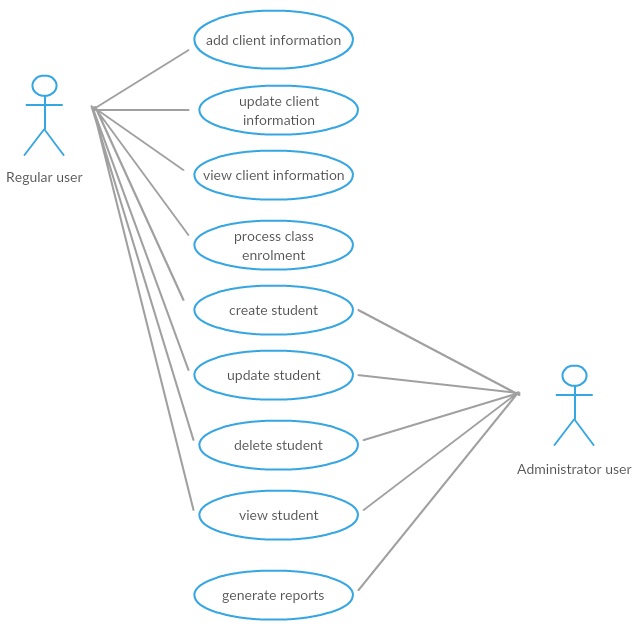
Reusability - layers help to differentiate between the different kinds of tasks performed by the components, making it easier to create a design that supports reusability of components.

Maintainability, Extensibility, Flexibility - adopting a layered approach can add some complexity, and may increase initial development time, but if implemented correctly will significantly improve the maintainability, extensibility, and flexibility of your application.

Security - application must be secured, all field must be validated for protection against SQL injection and all data should be protected by a password

Testability - because components belong to specific layers in the architecture, other layers can be mocked or stubbed, making this pattern is relatively easy to test.

# 2. Use-Case Model



*Use case: Create student*

*Level: user-goal level*

*Primary actor: Teacher (administrator user)*

*Main success scenario:*

1. The teacher selects “create student”
2. The system displays a bank student form
3. The teacher enters the following information for the student: identification number, group, enrolments, grades
4. The system validates the data to insure the proper format and searches for an existing student with the specified identification number. If the data is valid, the system creates a new student
5. Steps 2-3-4 are repeated for each student added to the system. When the teacher is finished adding students to the system the use case ends

*Extensions: Student already exists*

If the system finds an existing student with the same identification number an error message is displayed “Student Already Exists”. The teacher can either change the id number or cancel the operation at which point the use case ends.

# 3. System Architectural Design

## Architectural Pattern Description

**Layered Architecture**

The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database.

One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer.

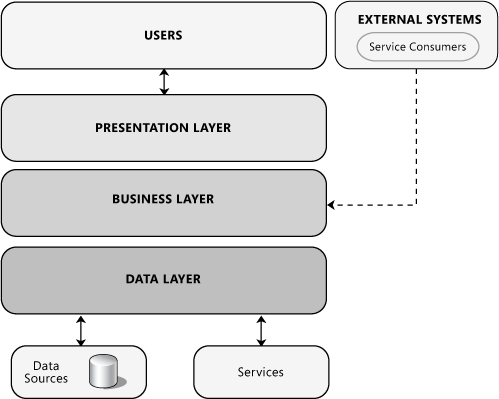
**Presentation layer.** This layer contains the user oriented functionality responsible for managing user interaction with the system, and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer.

**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.

## Diagrams

**System’s conceptual architecture**

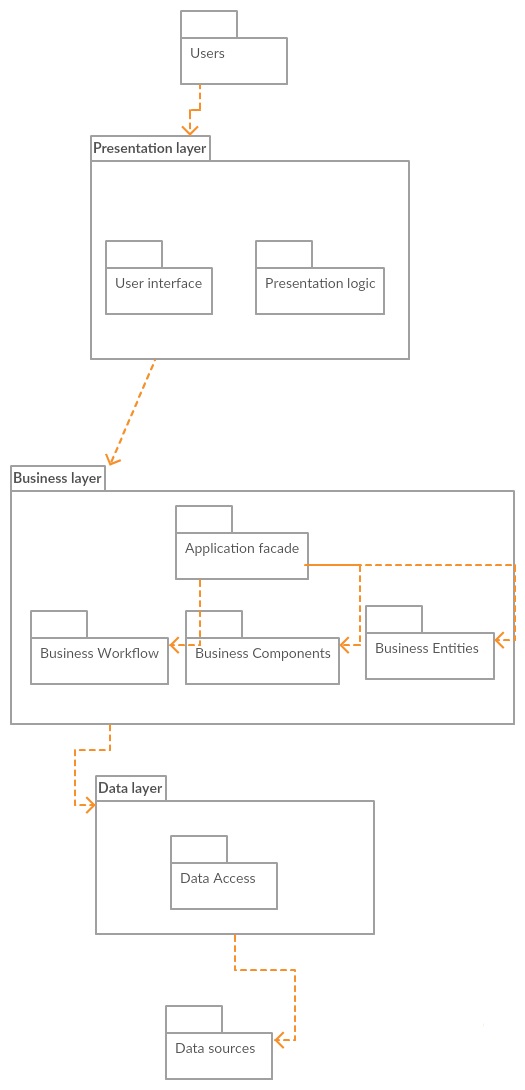


Presentation layer includes the application UI classes along with the JavaFX.

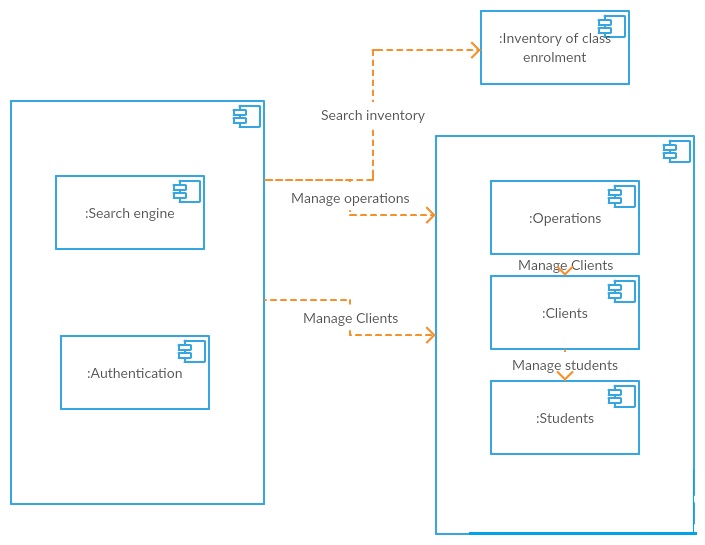
Business layer includes the application control classes – the logic of the application.

Data layer includes jdbc, javaSQL, relational classes which communicate with the database.

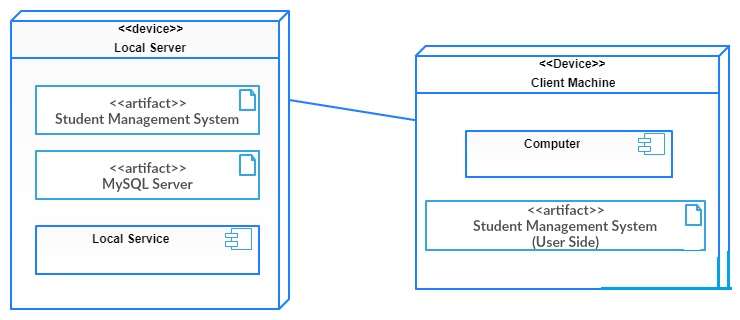
**Package diagram**



**Component diagram**

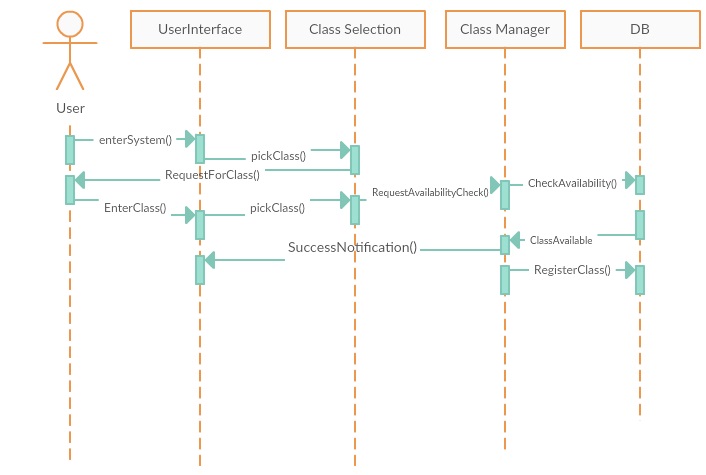
****

**Deployment diagram**



# UML Sequence Diagrams

--for process class enrolment—



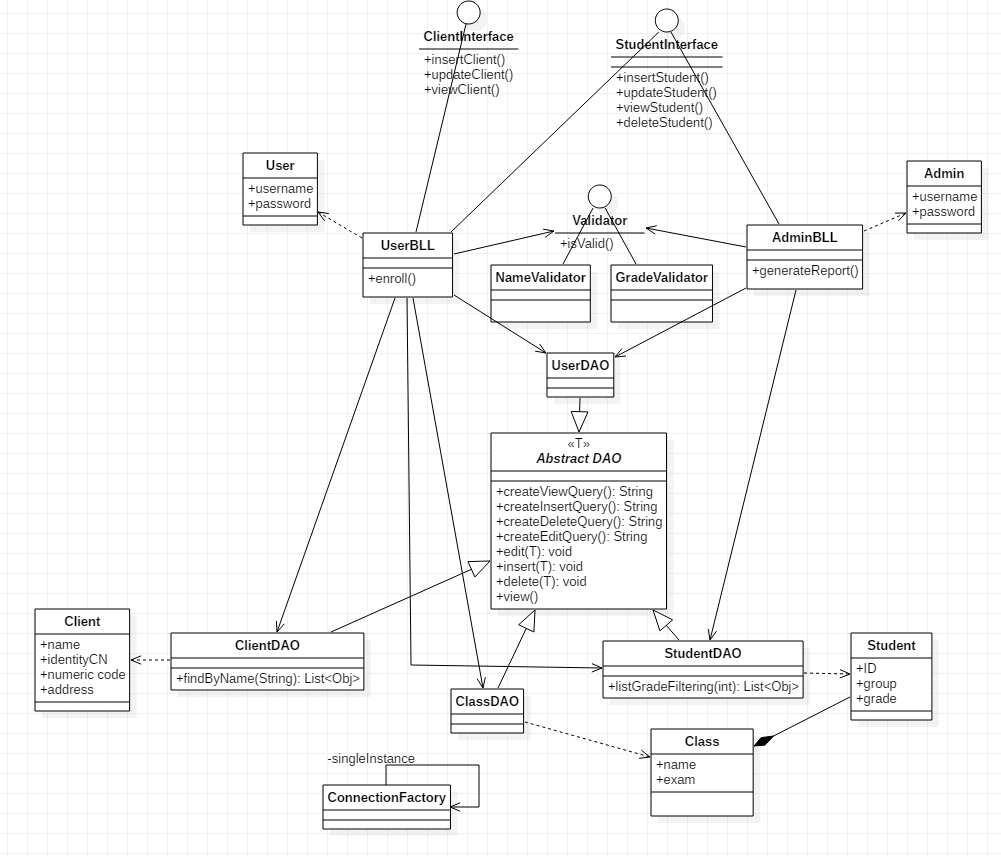
# Class Design

## Design Patterns Description

**Singleton pattern** is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class. *The singleton pattern is used for the connection with MySQL database.*

**Builder pattern** builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. A Builder class builds the final object step by step. This builder is independent of other objects. *This pattern is used to describe the data models in the project.*

## UML Class Diagram

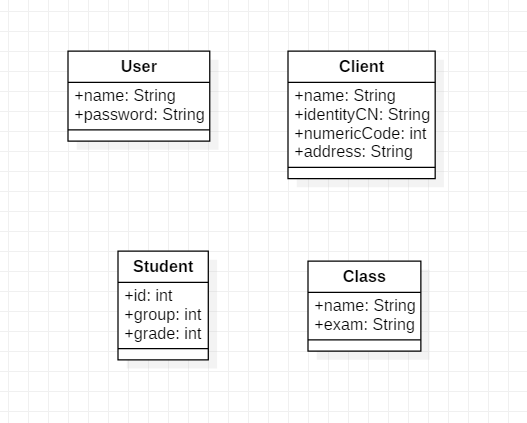


# Data Model

Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its members and display them. Therefore, for the access to the database I used the reflection technique, meaning a generic class which can query all the databases with the generic type <T>.

I created model classes for Student (identification number :int, group :String) Client (name: String, identity card number: String, numeric code : int, address: String) , Class (name: String, grade: int), User and Admin, both with username and password.

To respect the Layered Architectural Pattern the system is split in 3 layers: Presentation Layer, Business Layer, Data Access Layer(DAO).



# System Testing

For validating the data, an interface Validator is created and the method isValid is implemented regarding the name, password, email. Validating the data is used to introduce the correct data in the database.

UNIT TESTING is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. White box testing is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Unit testing are made for different methods of the application.

# Bibliography

|  |  |
| --- | --- |
| [1] | [Online]. Available: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html. |
| [2] | [Online]. Available: http://softwaretestingfundamentals.com/software-testing-methods/. |
| [3] | https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658109(v%3dpandp.10). [Online]. |