Assignment A1

Analysis and Design Document

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 (PostgreSQL). The Model View Controller architectural pattern is used to organize the application.

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

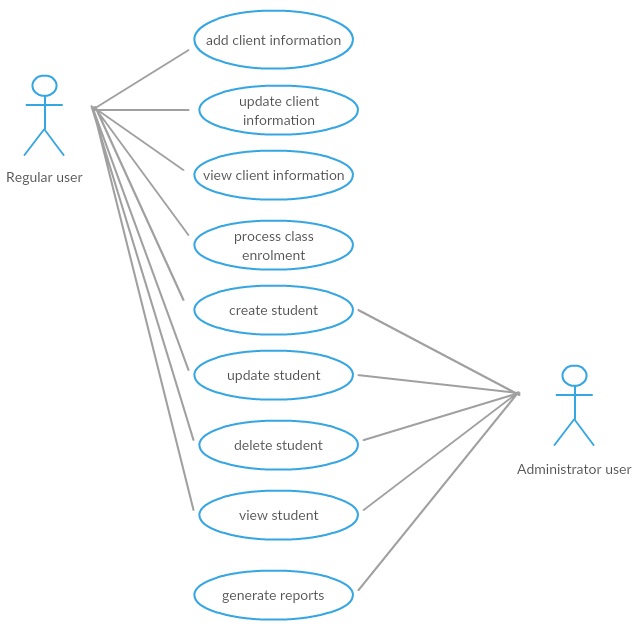
Reusability - layers help to differentiate between the different kinds of tasks performed by the components, making it easier to create a design that supports reusability of components.

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

**Model View Controller Architecture**

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

**Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

**View** - View represents the visualization of the data that model contains.

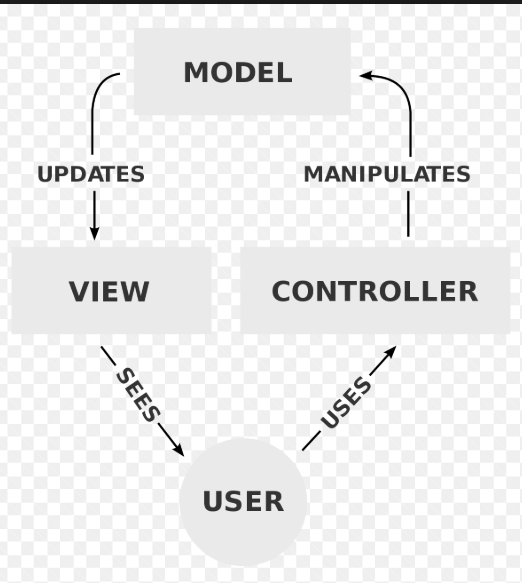
**Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

The application also has a mini layered architectural pattern. 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. 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.

For dependency injection I used Guice - is an open source, Java-based dependency injection framework. It is quiet lightweight and is actively developed/managed by Google. Guice embraces Java's type safe nature, especially when it comes to features introduced in Java 5 such as generics and annotations. You might think of Guice as filling in missing features for core Java. Ideally, the language itself would provide most of the same features, but until such a language comes along, we have Guice. Instead of creating the dependencies in the controller for the business logic classes, we inject them.

## Diagrams

**System’s conceptual architecture**

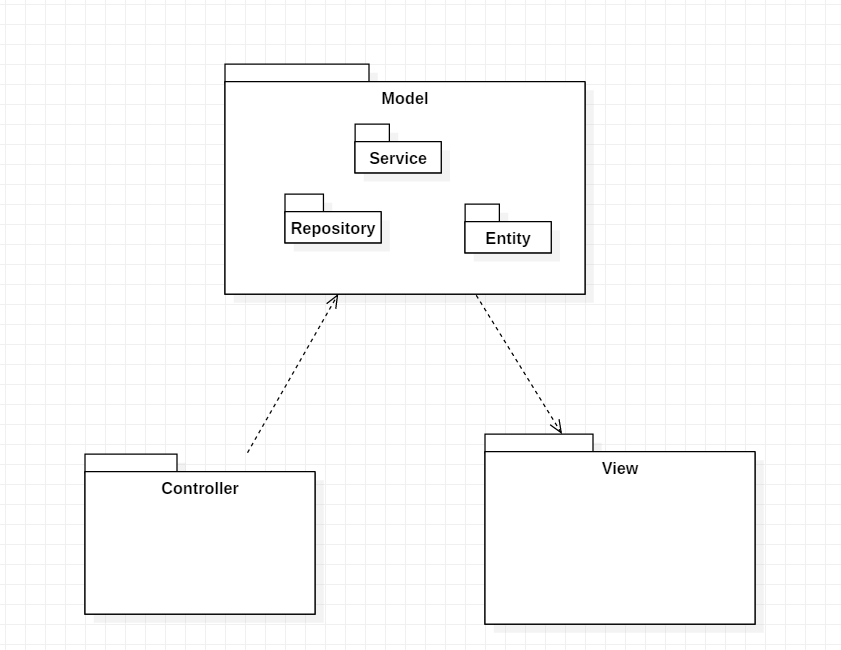


View includes the UI classes.

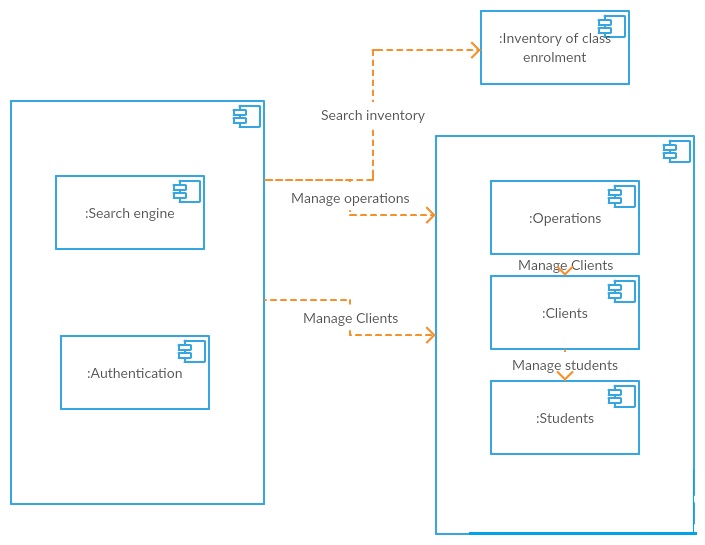
Controller includes the manipulation of UI classes (the logic behind the view).

Model includes the logic of the application, the pojo classes and the data access classes with the Hibernate connection.

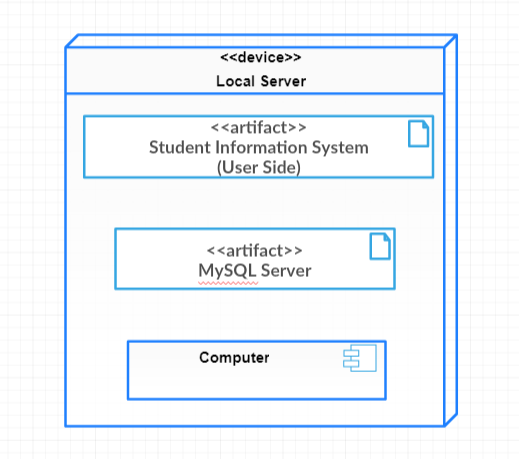
**Package diagram**



**Component diagram**

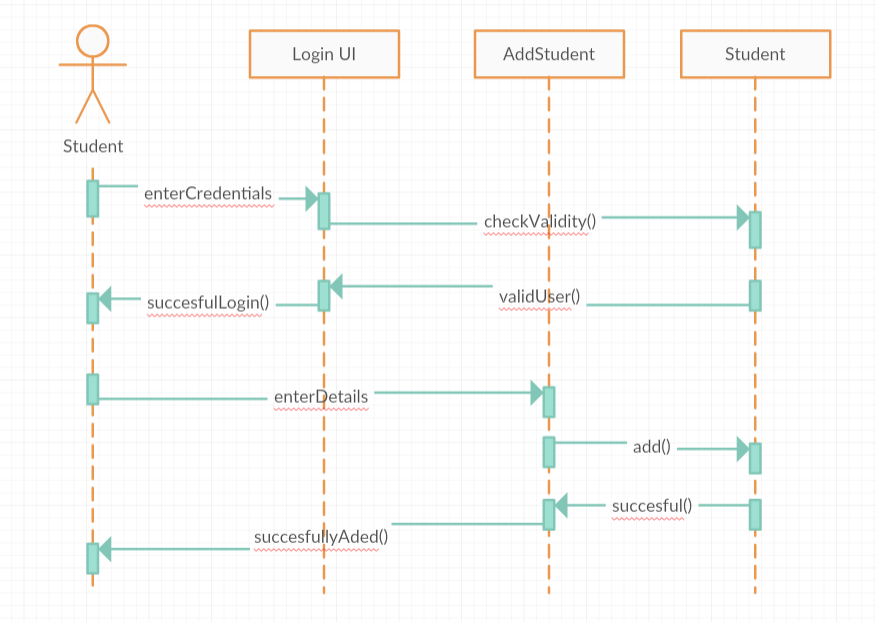
****

**Deployment diagram**



# UML Sequence Diagrams

--for login student--

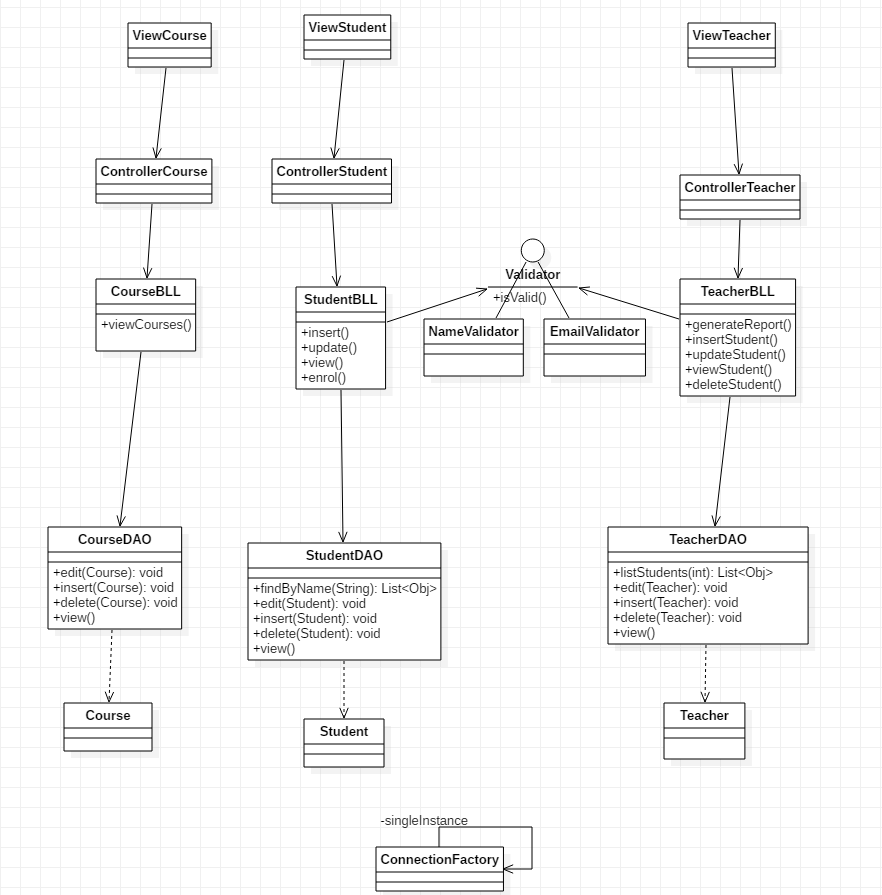


# Class Design

## Design Patterns Description

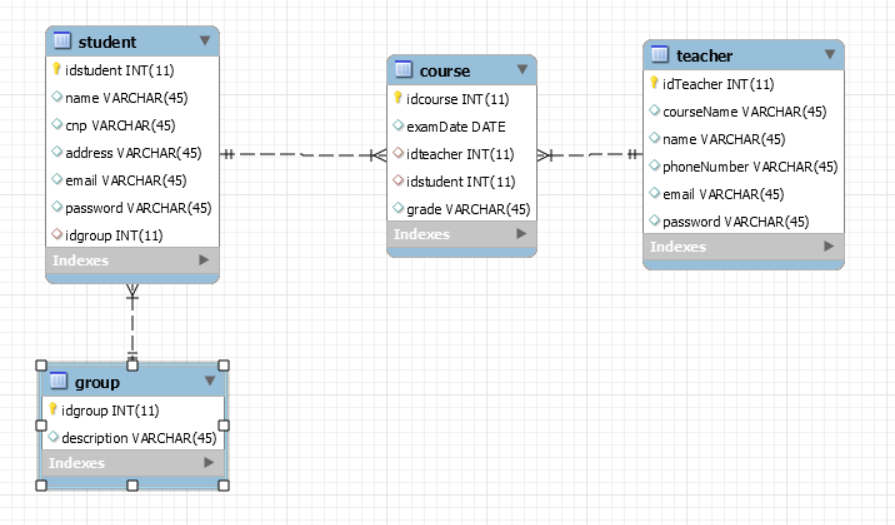
**Singleton pattern** is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. This pattern involves a single class which is responsible to create an object while making sure that only 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 PostgreSQL database.*

## UML Class Diagram



# Data Model

I chose to use 4 tables: student(id, name, cnp, address) and teacher(id, courseName, phone) and both of them have email and password. Each student can be enrolled to multiple courses, so there is a one to many relationship between the tables student-course Each teacher can teach multiple courses, so there is a one to many relationship between teacher-course. The table group contains the id of a group. Many students can be enrolled in a group, so the is a one to many relationship,too.



# System Testing

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

The values introduced in database will be test by cnp-13 numbers, the name has to be between 3 and 20 characters long, and the email should respect a regex pattern.

# 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]. |