Assignment 2

Analysis and Design Document

Student: Boros Hanniel

**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 3

3. System Architectural Design 4

4. UML Sequence Diagrams 7

5. Class Design 8

6. Data Model 10

7. System Testing 10

8. Bibliography 11

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 (students and teacher/administrator user) which have to provide a username and a password in order to use the application.

# Functional Requirements

* Authentication
* Edit client information
* Edit profile
* Process class enrolment
* Generate reports

# Non-functional Requirements

* Usability
* Performance (response time < 1s)
* Security
* Data integrity
* Documentation

2. Use-Case Model

|  |  |
| --- | --- |
|  |  |

One use case description

Use case: Update client information

Level: sub-function

Primary actor: student <a role name for the actor who initiates the use case>

Main success scenario: <the steps of the main success scenario from trigger to goal delivery>

Student selects Update Personal Information option from menu

Student updates the necessary information (name, identity card number, personal numerical code, address)

Student presses Update button, finishing update.

Student waits a short time while information is updated in the database.

Information is updated and student is notified.

Extensions:

Alternate scenario for failure:

if connection to database fails for some particular reasons, or an Exception is encountered, the information may not be updated.

3. System Architectural Design

**3.1 Architectural Pattern Description**

**Layers Architectural Pattern**

The architectural pattern used in this project is the Layers architecture, otherwise known as the n-tier architecture pattern. This architectural pattern helps us to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction.

**Controller 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.

**Persistence 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.

**Client-Server Architectural Pattern**

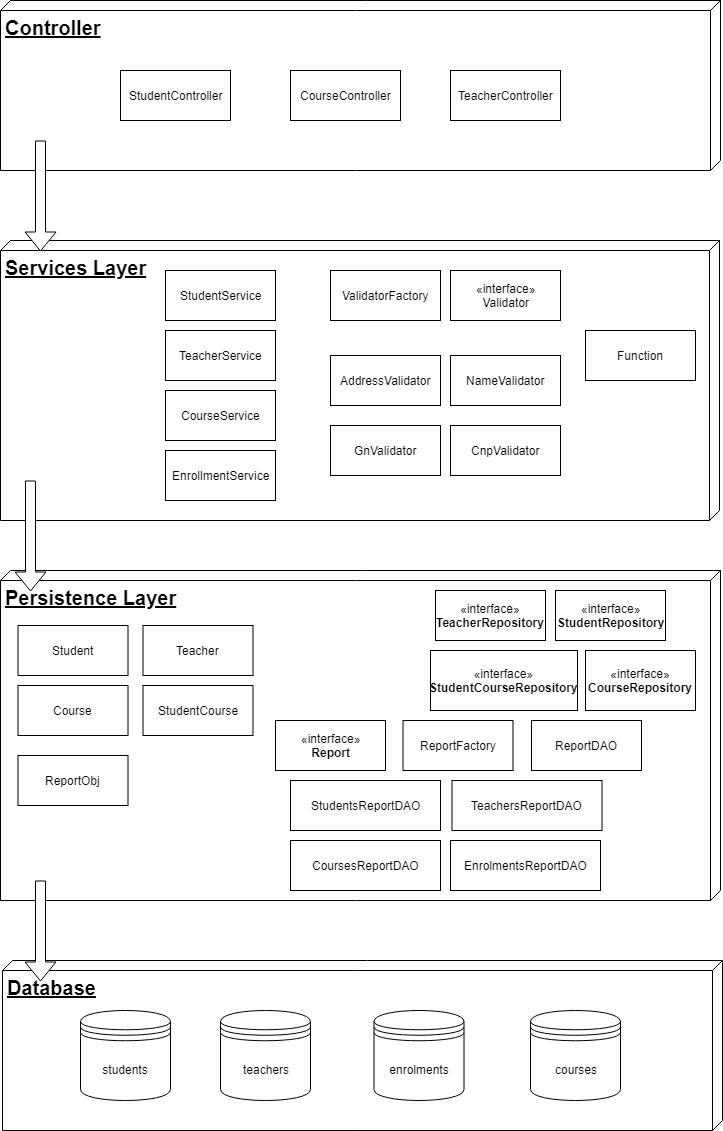
The client/server architectural style describes distributed systems that involve a separate client and server system, and a connecting network. The simplest form of client/server system involves a server application that is accessed directly by multiple clients, referred to as a 2-Tier architectural style. The current assignment was developed in Spring Framework which has a Client-Server design structure handling requests from clients (represented by the html files) in the controller, which then calls the services to perform the required actions.

**MVC Architectural Pattern**

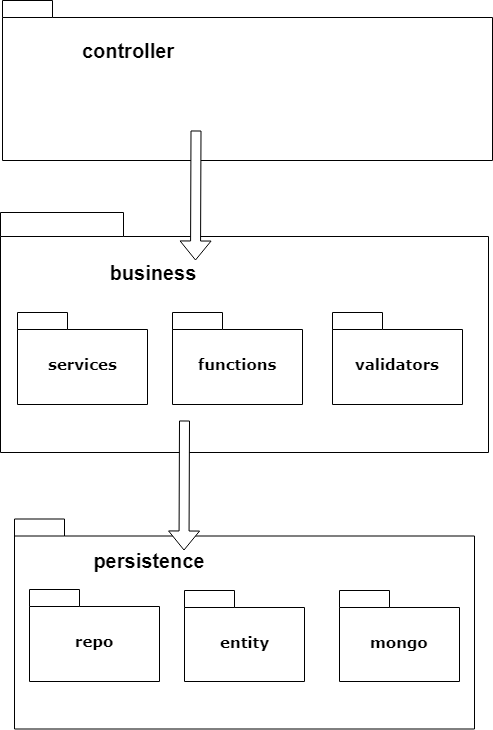
The major (model, controller, view) are decoupled, allowing for efficient code reuse and parallel development. Model-View-Controller patterns have a wide usage and forms. In this project I used Spring MVC which provides a good front-to-back-end data model mapping to UI and back to services, invoking actions.

**3.2 Diagrams**

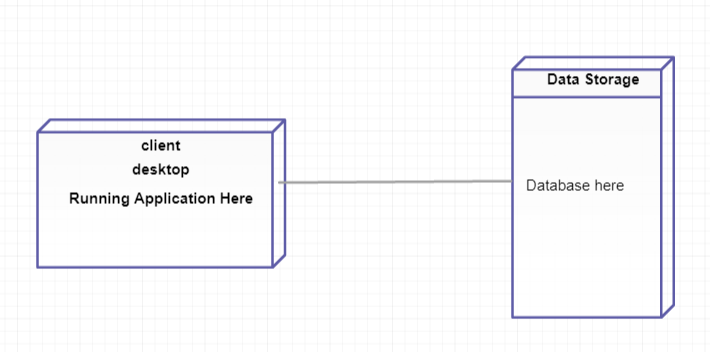
Layered Architecture Pattern of the solution



Package diagram

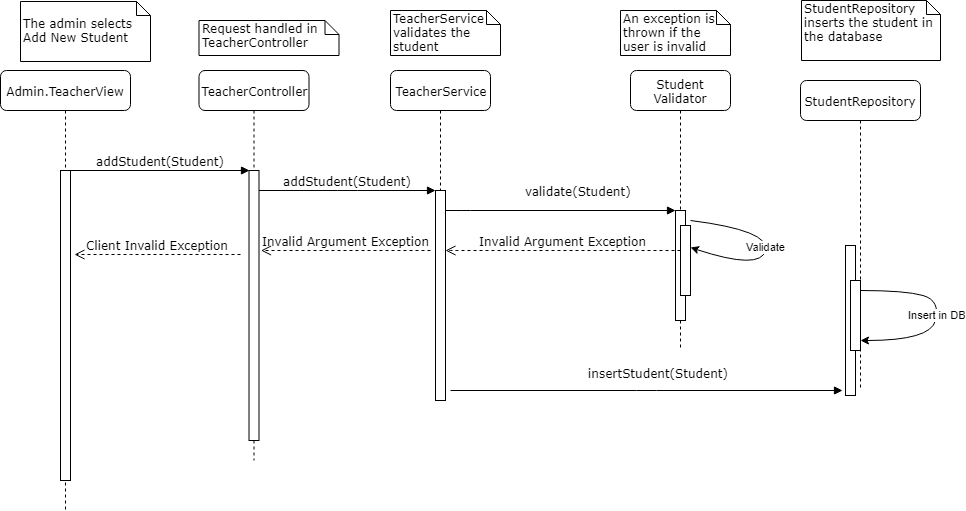


Deployment diagrams



4. UML Sequence Diagrams

Sequence diagram for inserting a new student.



5. Class Design

**5.1 Design Patterns Description**

**Factory Creational Design Pattern**

This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

The object is created without exposing its creation logic to the client and reference to the newly created object is done using a common interface.

Two Factory classes were implemented:

* Validator Factory, for the creation of different types of validators in the business logic layer
* ReportFactory, for the creation of different types of reportDAO objects in the persistence layer

**Data Mapper Design Pattern**

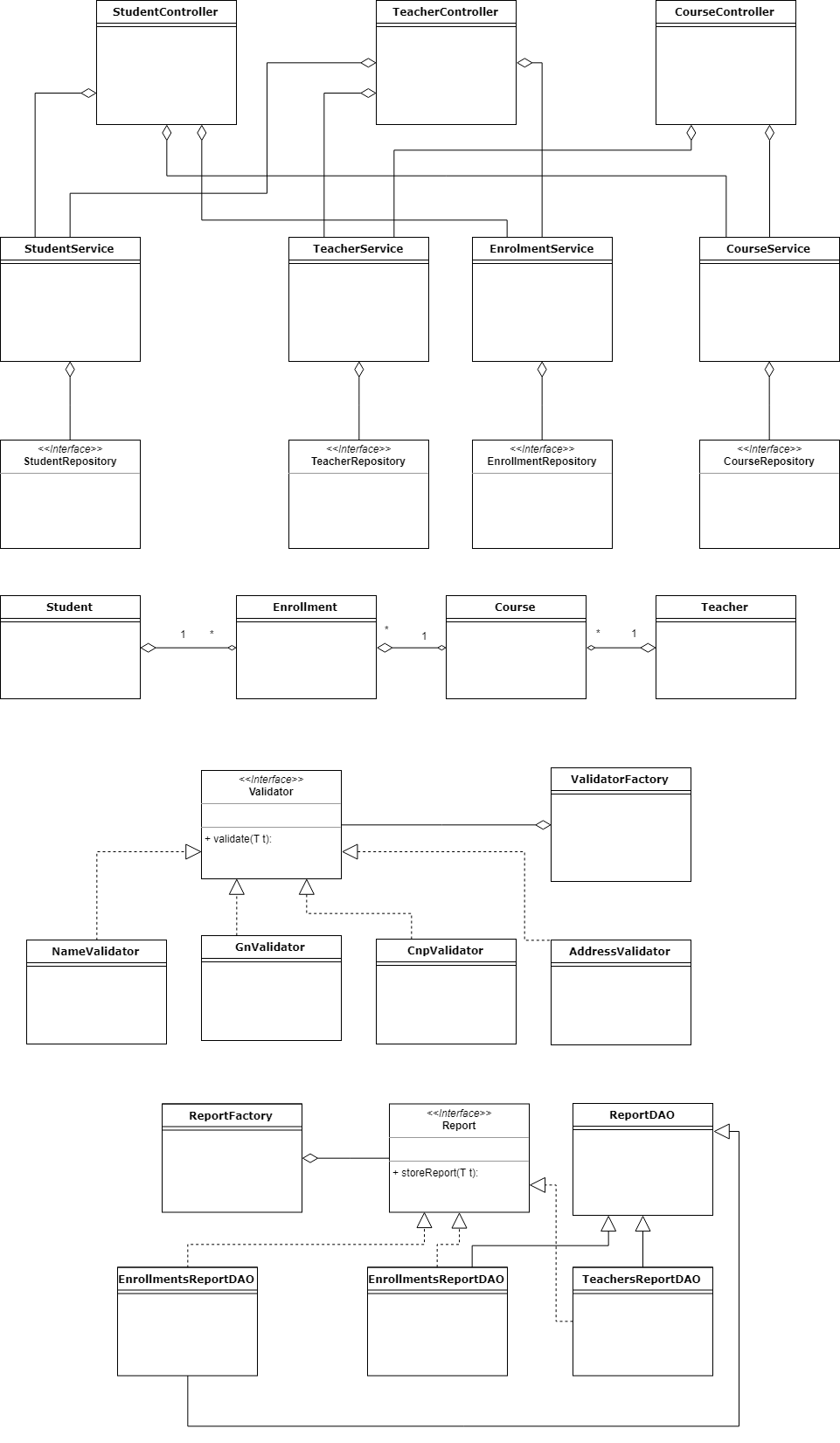
A Data Mapper is a Data Access Layer that performs bidirectional transfer of data between a persistent data store (often a relational database) and an in-memory data representation (the domain layer). The goal of the pattern is to keep the in-memory representation and the persistent data store independent of each other and the data mapper itself. The layer is composed of one or more mappers (or Data Access Objects), performing the data transfer.

This pattern is implemented in the persistence layer: classes are created representing database tables, and using Hibernate and Spring framework the Data Access is executed.

**5.2 UML Class Diagram**

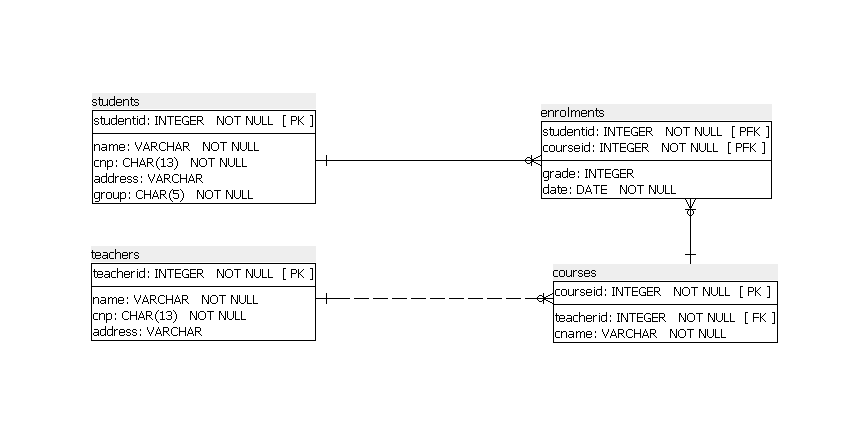
Sinces the UML Diagram for the whole system would be of great proportions, I decided to present the class diagrams of the relevant classes of each module and layer, simplifying it where I had the possibility.

As well, the general use-relationship is not presented below.



**6. Data Model IR diagram**

Below I present the Data IR diagram. This was used to represent and model the data of the application. There are four tables: three of them for direct data storage (client and other information as well as courses) and the other one for relationships (it models a many-to-many relationship by a one-to-many ⬄ many-to-one relationship).



7. System Testing

I will use unit testing in order to test my application.

I tried to make an implementation which will be favorable and sustain testing.

Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behavior of the object one wants to test, it replaces the other objects by mocks that simulate the behavior of the real objects. This is useful if the real objects are impractical to incorporate into the unit test.

In short, mocking is creating objects that simulate the behavior of real objects.

8. Bibliography

Here are mentioned some resources which were helpful for writing this documentation and as well gave me a better understanding of the concepts and guided me in the implementation process:

**Information and knowledge:**

Microsoft Application Architecture Guide, 2nd Edition

Software Architecture Patterns, by Mark Richards

<https://www.tutorialspoint.com>

<https://en.wikipedia.org>

<https://stackoverflow.com>

<http://www.java2s.com>

<http://www.mkyong.com>

**Tools and technologies:**

<https://spring.io>

<http://hibernate.org>

<https://www.mongodb.com>

<https://jdbc.postgresql.org>

**Tools helpful for the documentation:**

<https://www.draw.io/>

<https://www.smartdraw.com>

<https://creately.com>