Students Management II
Analysis and Design Document

Student: Danila Vlad-Mihai
Group: 30432

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

This application aims to develop a management system for TUCN CS Department students using MVC design pattern and Layers architectural pattern.

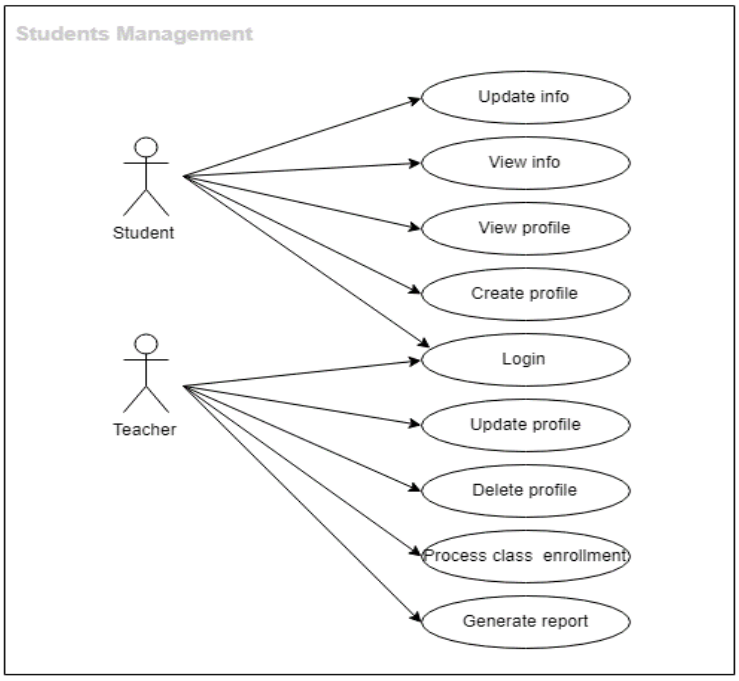
1.2 Functional Requirements

- System shall block users from access to use the application if they are not logged in
- Add/Update/View client information
- Create/Read/Update/Delete student profile
- Process class enrolment
- Generate reports
- Validate input before saving it in the database

1.3 Non-functional Requirements

- Availability: the system should be available all the time, except the planned maintenance periods
- Performance: the system should have a time response no longer than 5 seconds
- Reusability: the system should be design as having components that could be reused by other systems if necessary
- Usability: the system will have a user-friendly interface and will be easy to use

2. Use-Case Model



Use case: Student login using username and password
Level: User-goal level
Primary actor: Student
Main success scenario:

1. Student opens the application
2. Student enters the username and password
3. Student receives a message confirming the authentication
4. Student login successfully completed

Extensions:

1. Student opens the application
2. Student enters the username and password
3. Student receives an error message
4. Student enters the username and password again
5. Student receives a message confirming the authentication
6. Student login successfully completed

3. System Architectural Design

3.1 Architectural Patterns Description

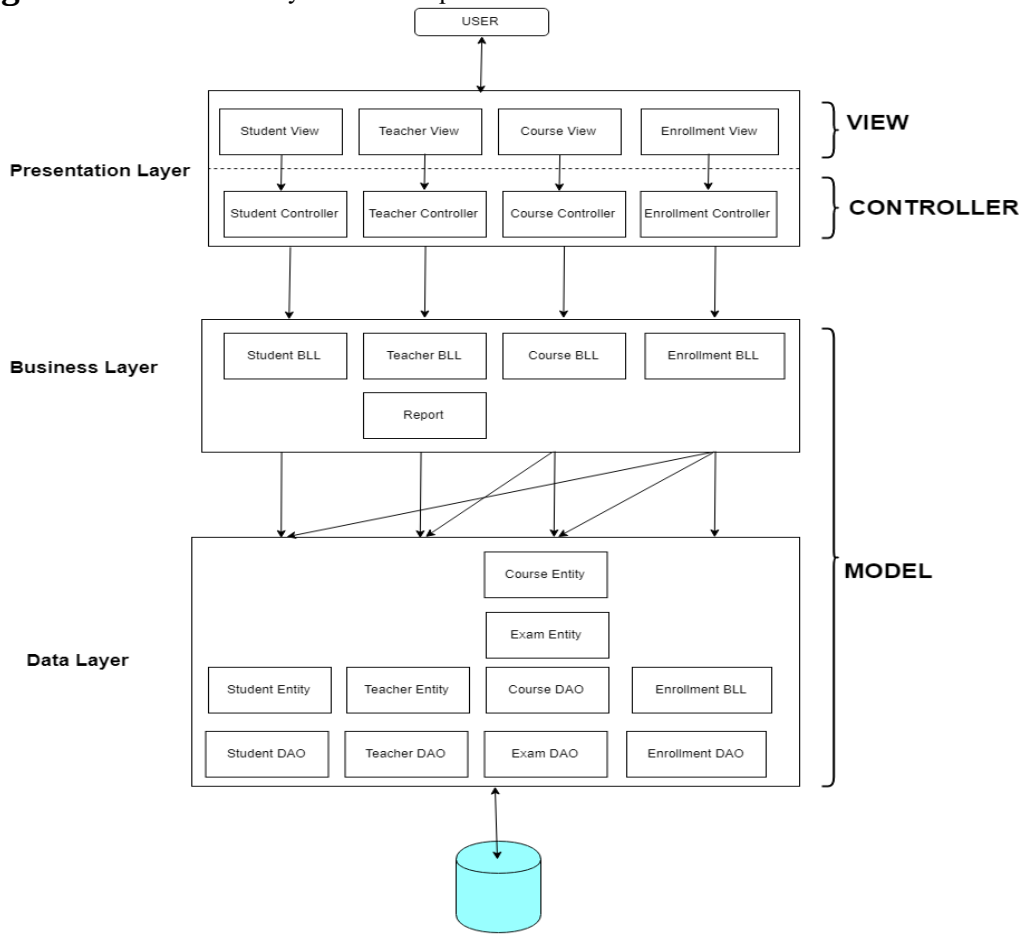
The system will be built using Layers architectural pattern. Logically similar functionalities will be grouped from the technical point of view by dividing the entire program logic into three layers: Presentation Layer, Business Layer and Database Layer. Our goal is to minimize the number of overlapping functionalities across the entire application.

MVC (Model-View-Controller) architectural pattern is used to divide the system intro three interconnected parts:

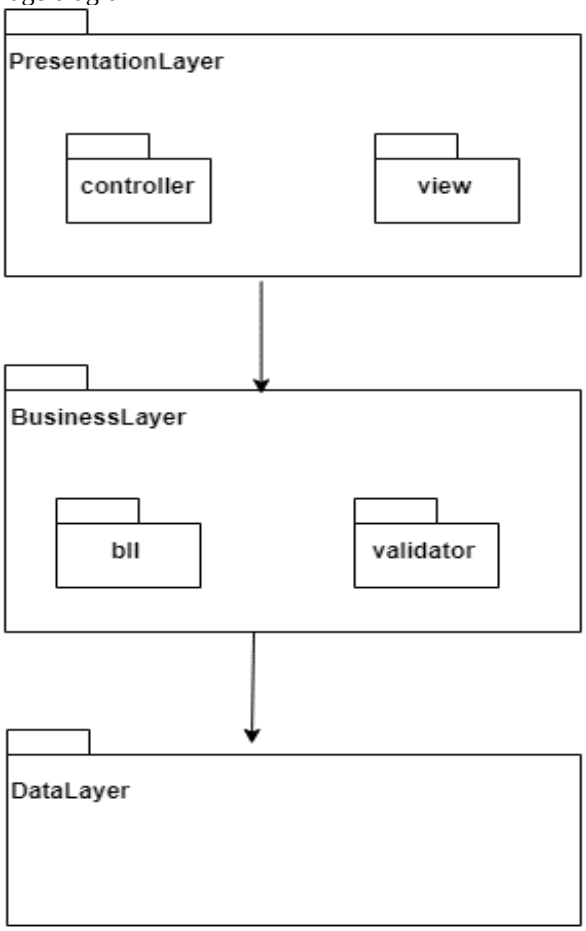
- Model represents the data and business logic of the application.
- View module is responsible to display data i.e. it represents the presentation.
- Controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

3.2 Diagrams

System's conceptual architecture

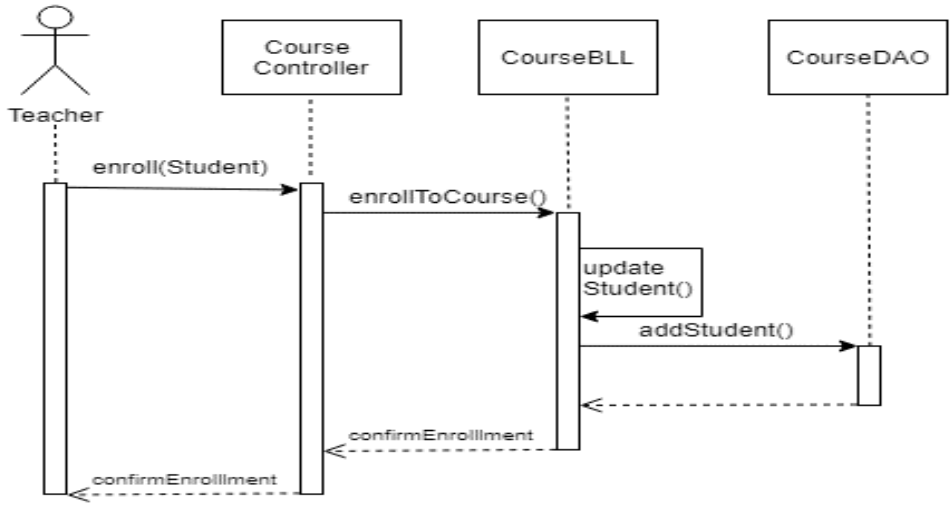


Package diagram

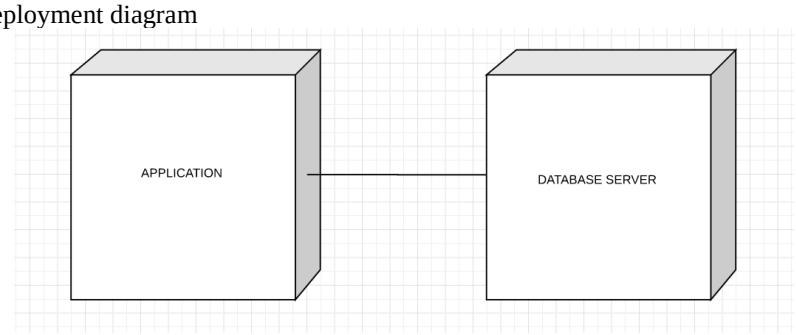


5. UML Sequence Diagrams

Scenario: Student enrollment



Deployment diagram

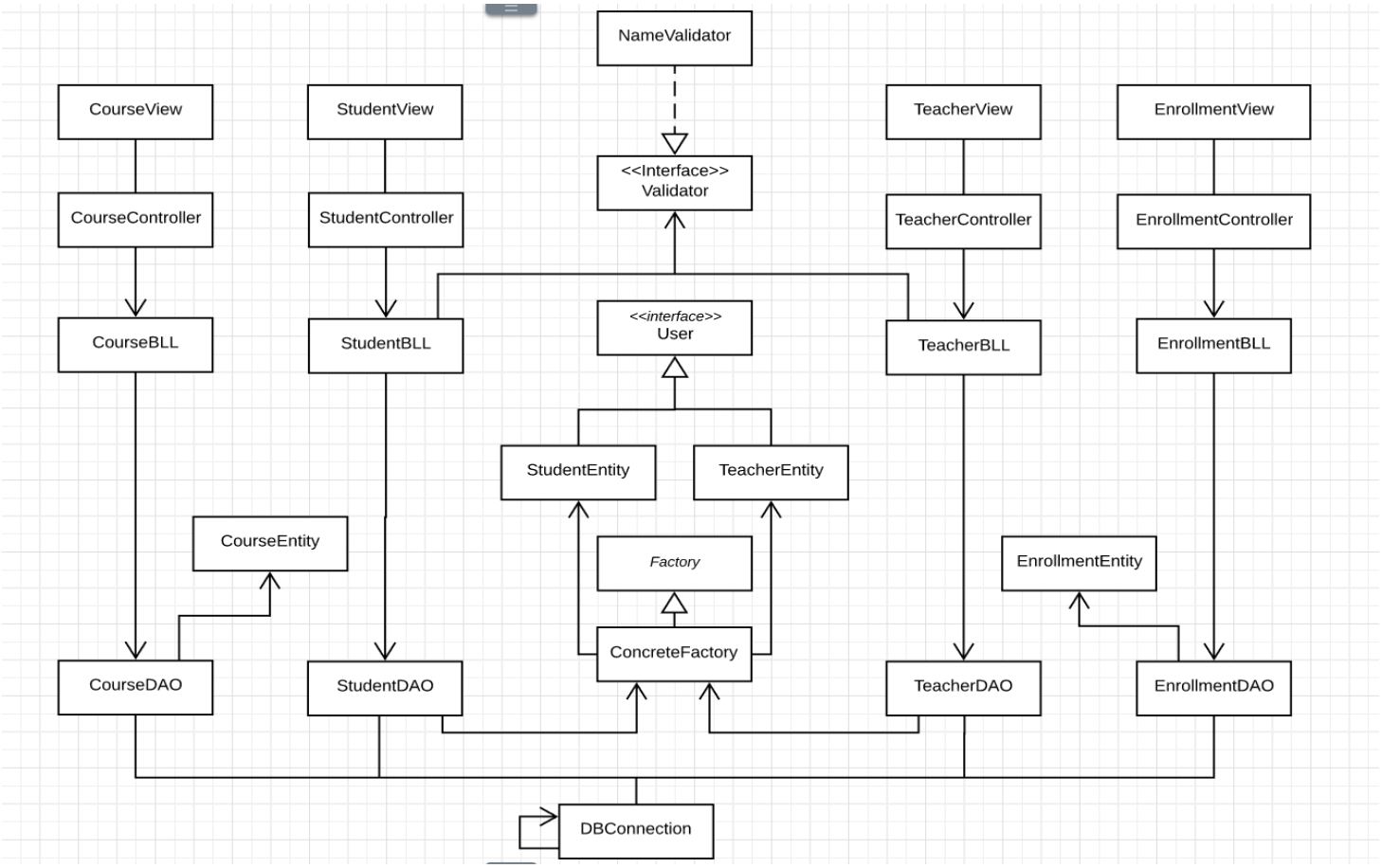


5. Class Design

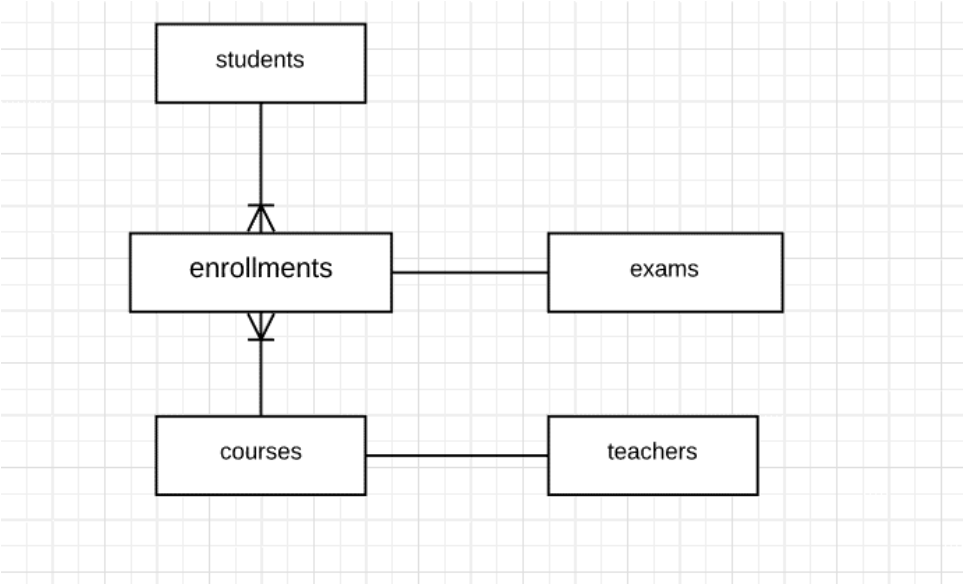
5.1 Design Patterns Description

Singleton design pattern – one single database connection
Factory method design pattern – also known as Virtual Constructor, defines an interface for creating objects (i.e. User) but leaves the choice of its type to the subclasses (i.e. StudentEntity, TeacherEntity). We will refer to the newly created object through a common interface.

5.2 UML Class Diagram



6. Data Model



7. System Testing

- Unit Testing – write test to verify that a relatively small piece of code is doing what it is intended to do. Junit will be used for testing at method or class level, by checking that the actual output matches the expected output.
- Integration Testing - testing that the actual service or database behavior is correct. This ties in with unit tests so that is known what data should be retrieved and what should be done with that data.

8. Bibliography

stackoverflow.com
lucidchart.com
http://users.utcluj.ro/~dinso/PS2018/