Assignment 2
Analysis and Design Document

Student: Ciucescu Vlad Andrei
Group: 30432

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

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

1.2 Functional Requirements

The application should have two types of users: student and teacher/administrator User. The authentication mechanism for users is not necessary.
The regular user can perform the following operations:

- View personal information (name, identity card number, address etc.).
- Update/view student profile (telephone number, email, Enrollments etc.).
- Process class enrolment (enroll, exams, grades).

The administrator user can perform the following operations:

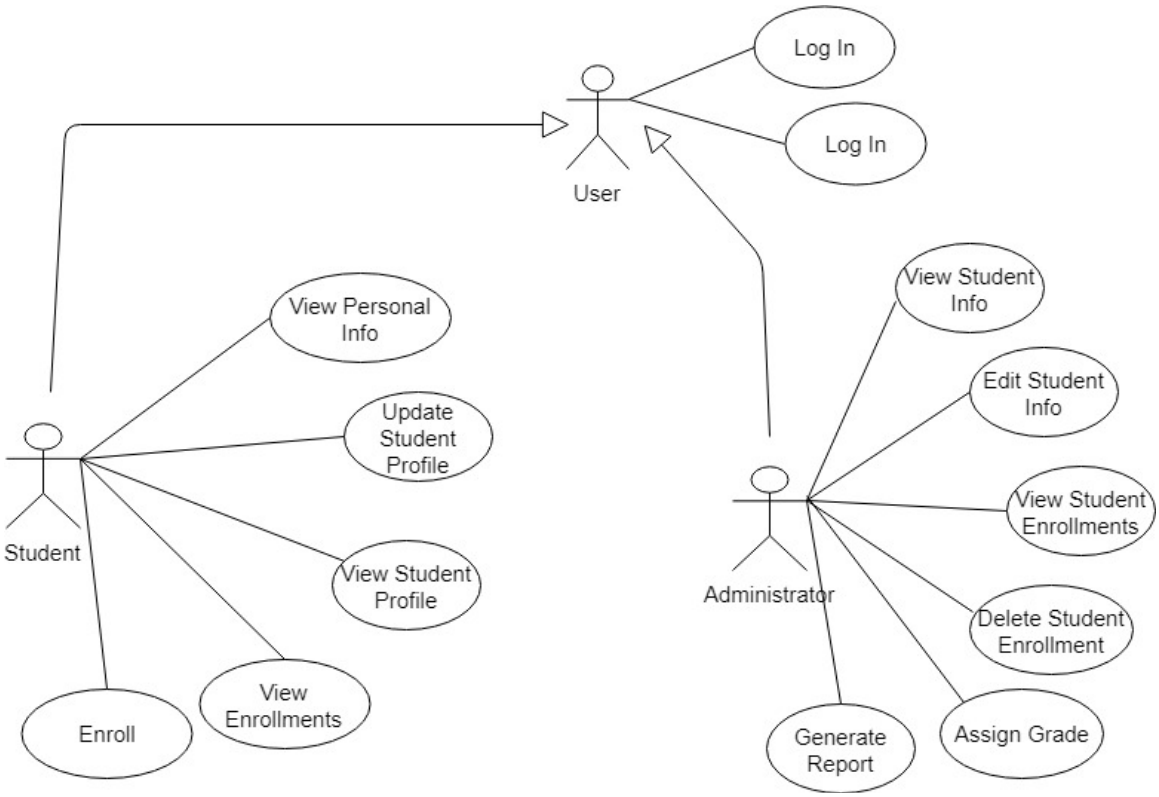
- CRUD on student's information.
- Generate reports for a period containing the activities performed by a student

1.3 Non-Functional Requirements

- The application should be developed in a Java MV* development platform, such as Spring.
- The application should be structured using the Layers and MVC architectural patterns.
- Build automation and dependencies should be handled by a specialized tool such as Maven or Gradle.
- Reports should be stored using a NoSQL DBMS such as MongoDB
- Integration tests and dependency injection should be used

2. Use-Case Model

Use case: Update Email
Level: user-goal level
Primary actor: Student
Main success scenario: The Student presses the “Update Email” button in the Profile window, edits the email, presses the “Update Email” button and the changes to the email are saved.
Extensions: alternatively, the student could press the “Cancel” button, which cancels any modifications. If the student introduced invalid data, such as the same email as the current one, pressing the “Update Email” button will not update the profile, but instead will display an error message.



3. System Architectural Design

3.1 Architectural Patterns Description

The main architectural pattern used for the application is the layered architecture pattern. Irrespective of the type of application being created, one can decompose the design into logical groupings of software components. These logical groupings are called layers. 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. Each logical layer contains several discrete component types grouped into sub layers, with each sub layer performing a specific type of task.

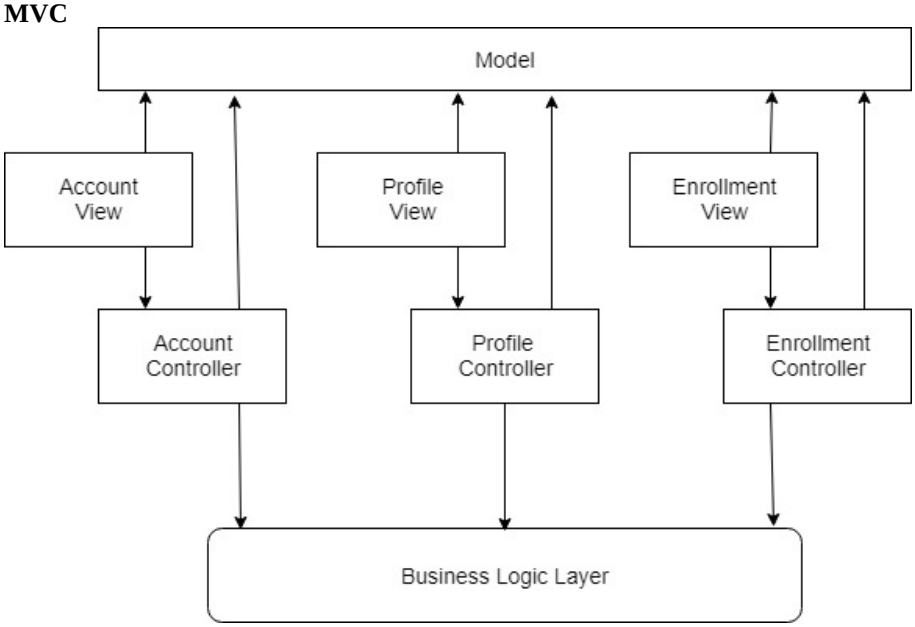
Beside the layers pattern, Model-View-Controller (MVC) will also be used. MVC is generally a pattern used for separating UI concerns:

- The model is responsible for managing the data of the application. It receives used input from the controller.
- The view is a representation of the model in a particular format.
- The controller is responsible for responding to the user input and performing interactions on the data model objects.

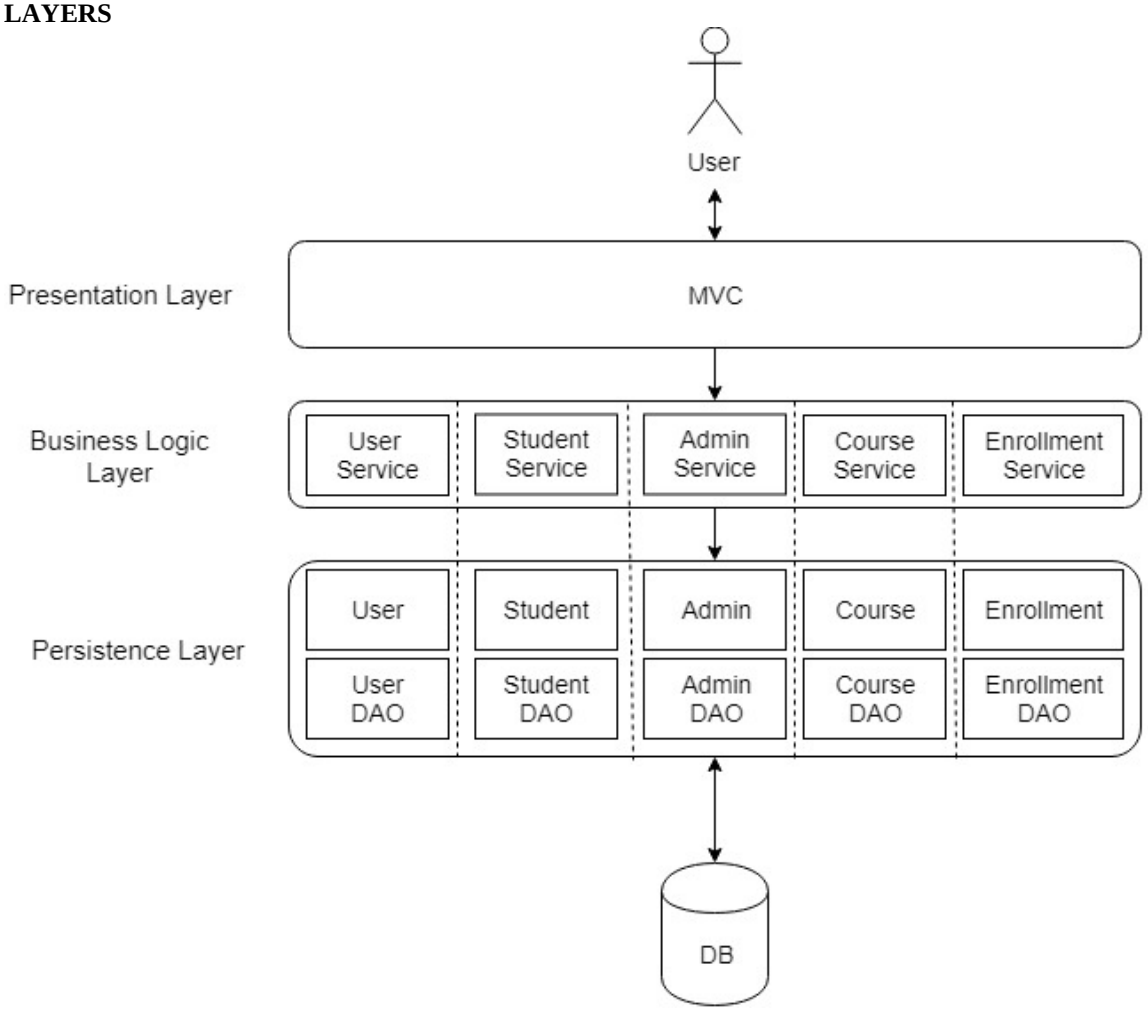
In relation to Layers, the Business Logic Layer(BLL) will be called from controllers, which will aggregate, create or map data from the BLL into the UI model. The UI model is decoupled from the BLL.

3.2 Diagrams

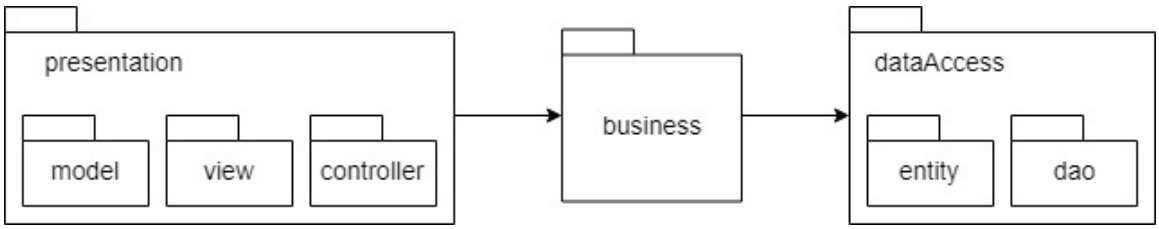
MVC



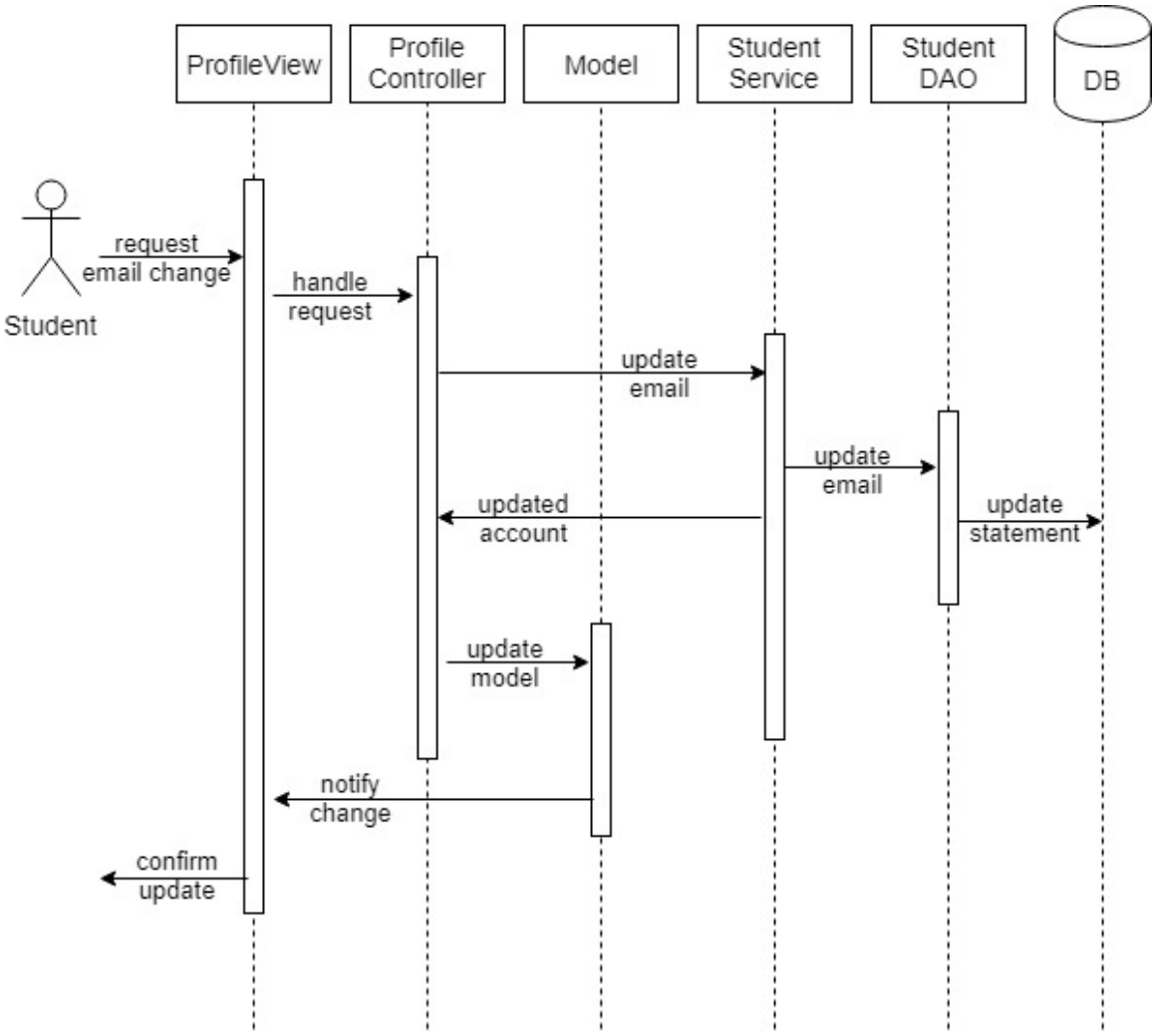
LAYERS



Package Diagram



4. UML Sequence Diagrams



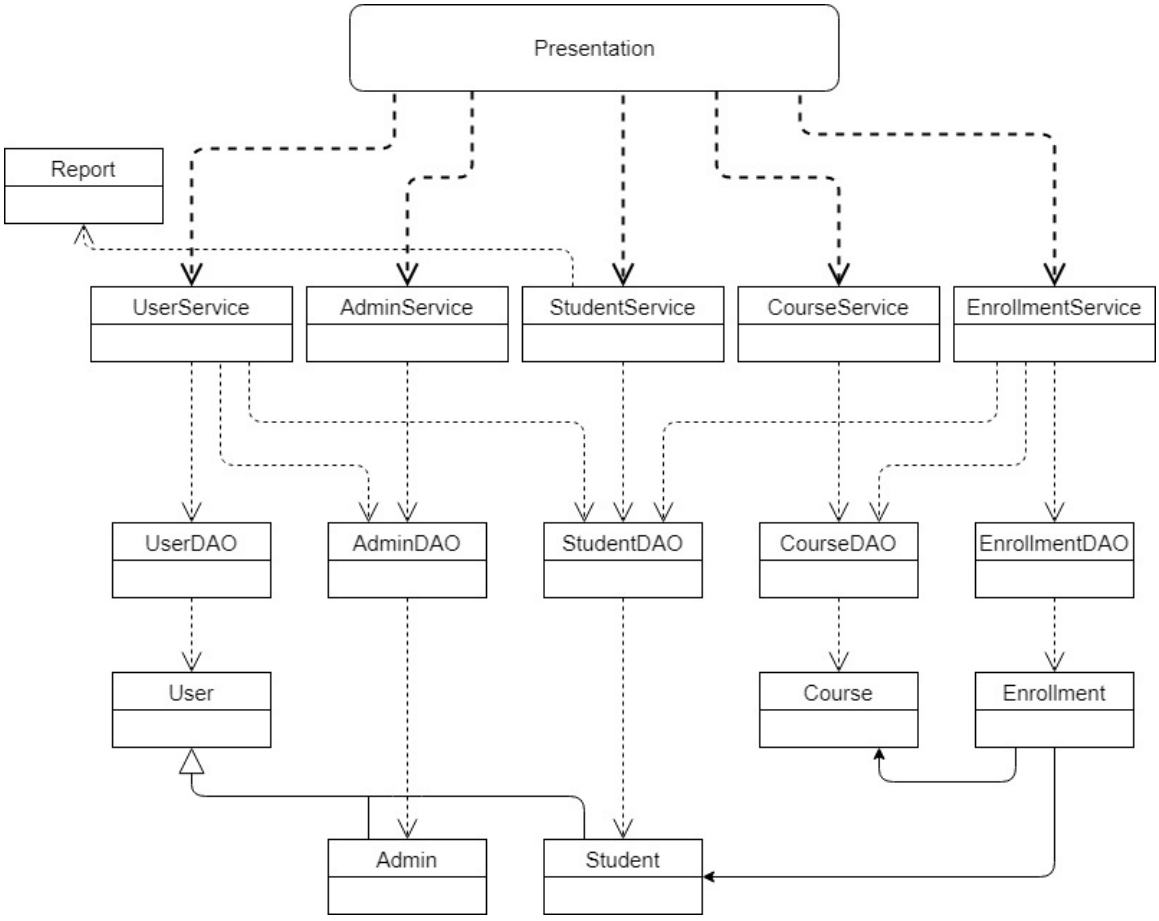
5. Class Design

5.1 Design Patterns Description

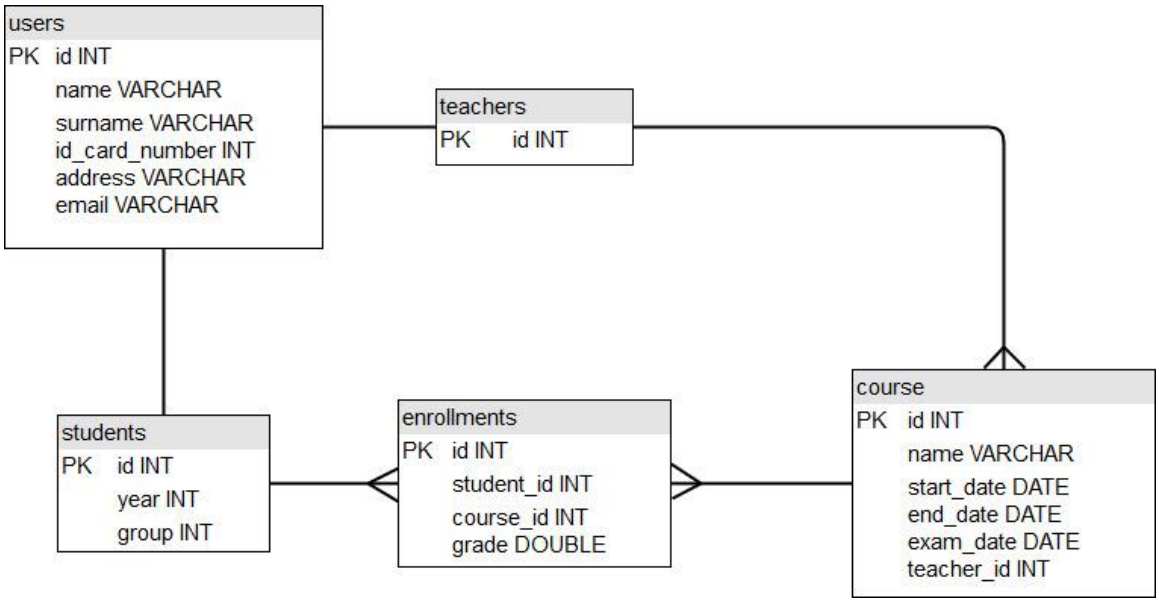
The Builder pattern is a creational pattern whose purpose is to aid in the construction of complex objects. It is an alternative to using constructors with a large number of parameters, which can be difficult and can lead to an incorrect object state. The pattern is often implemented with a fluent interface, meaning that builder methods return the builder itself, and a terminal operation is used to build and return the final object.

In the current application, a Builder class will be provided for each of the entity objects, which will facilitate testing.

5.2 UML Class Diagrams



6. Data Model



7. System Testing

The testing strategy used is unit testing. A unit is the smallest testable part of software. As such, unit testing involves testing individual units or components (such as methods) of a software. Its purpose is to check whether each unit of the software performs as designed.

For this application, unit testing will be performed with Mockito, a mock framework which allows programmers to create mock objects at runtime and define their behavior. It allows us to mock away external dependencies and insert the mocks into the code under test, execute the code under test and validate that the code executed correctly. Mockito can be used in conjunction with JUnit.

8. Bibliography

http://softwaretestingfundamentals.com/unit-testing/
http://www.vogella.com/tutorials/Mockito/article.html
http://www.agilemodeling.com/artifacts/
https://msdn.microsoft.com/en-us/library/ee658109.aspx
https://www.javaworld.com/article/2074938/core-java/too-many-parameters-in-java-methods-part-3-builder-pattern.html
http://www.baeldung.com/integration-testing-in-spring