Assignment 2 – Student Management System
Analysis and Design Document

Student: Ana-Maria Nanes
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. The application should have two types of users (student and teacher/administrator user) which have to provide a username and a password in order to use the application.

1.2 Functional Requirements

The functional requirements of the system are the following:

- The student 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 teacher can perform the following operations:

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

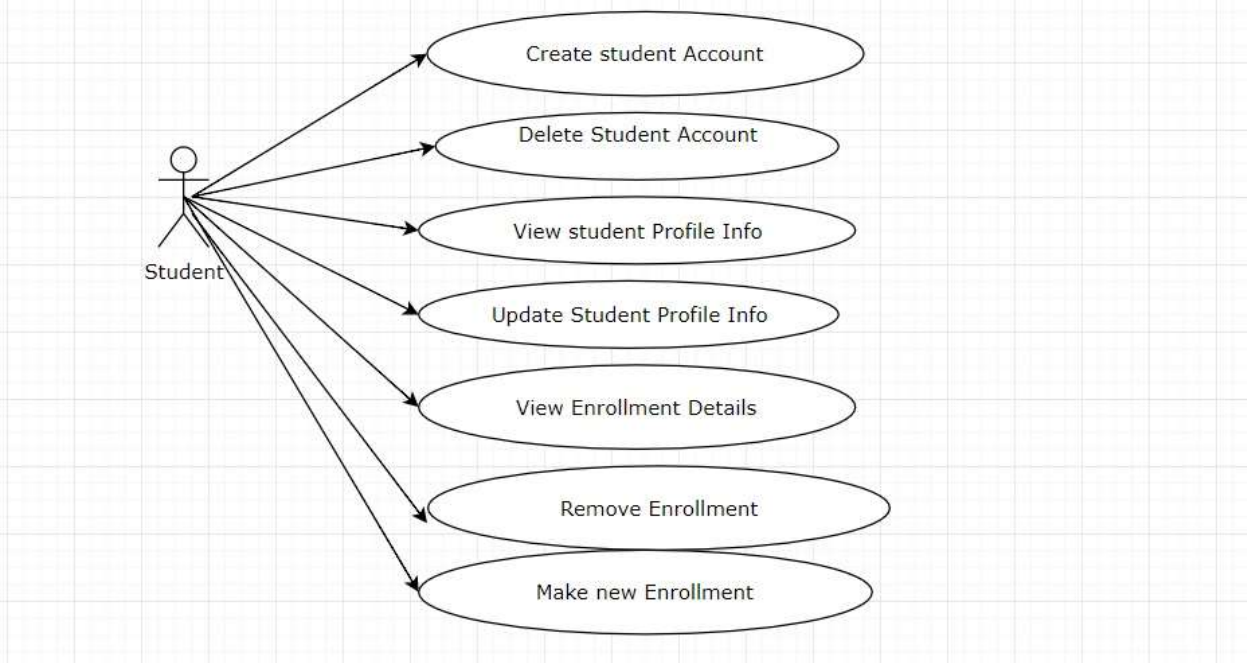
1.3 Non-functional Requirements

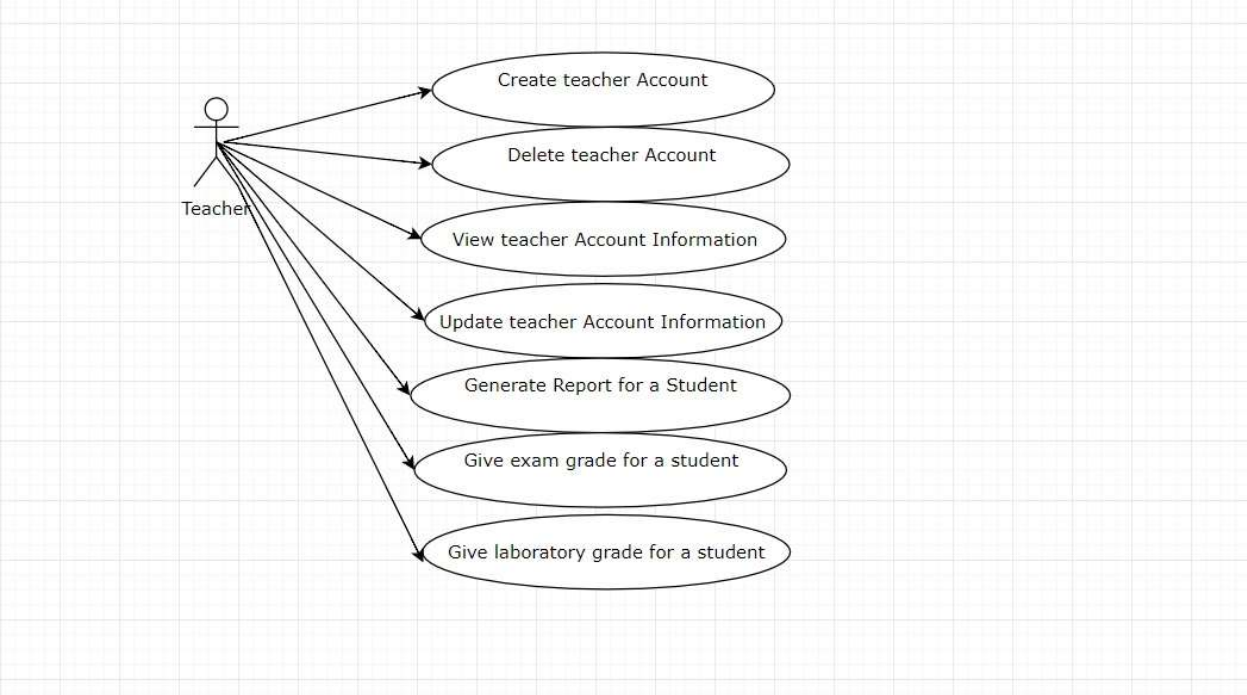
The non-functional requirements :

- Security - making distinction between the two types of users – students and administrators – the data is protected.
- Data integrity – is implemented using data validation each time new information is added to the system.
- Extensibility - due to the Layers Architectural Pattern it is easy to extend the system in more directions.
- BackUp - the system`s information could be also stored in files – make a backup after a certain period of time.

2. Use-Case Model

The following diagram is the use-case diagram:





The two type of users are:

- The student.
- The teacher.

Use case: Create student profile.
Level: Subfunction.
Primary actor: A student in the CS Department at TUCN.
Main success scenario:

- Provide a username.
- Provide the required information.
- Validate the input data.
- Perform the registration operation.

Extensions:

Scenarios of failure :

- the username is already taken.
- the name is already taken.

3. System Architectural Design

3.1 Architectural Pattern Description

The architectural pattern used are the following:

- Layered Architectural Pattern
- MVC Architectural Pattern

Layered Architectural Pattern

This pattern can be used in order to structure the application such that it can be decomposed into groups of subtasks. Each layer implements subtasks at a particular level of abstraction and each layer provides services to the next higher level.

The following 3 layers will be used in order to offer the application the accurate structure:

- Presentation Layer
- Business Logic Layer
- Data Access Layer

These packages are modeled using different packages inside the application.

MVC Architectural Pattern

Model–view–controller (MVC) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.

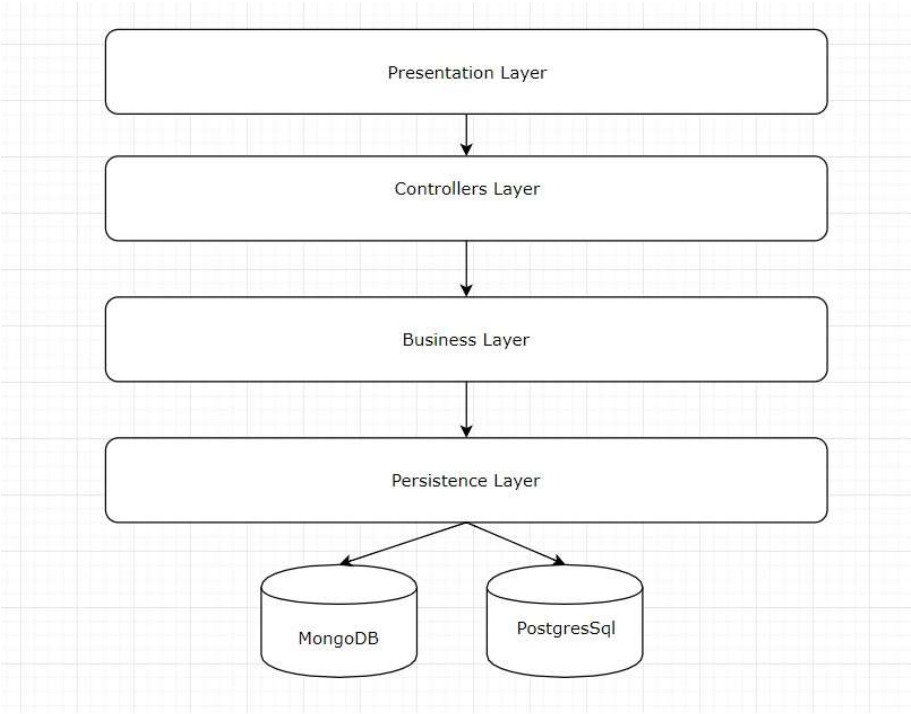
The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

- 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.

3.2 Diagrams

❖ The system conceptual diagram:

The layers and the MVC structure are depicted in the next system conceptual diagram:



How the Layered Architectural Pattern is applied:

The Persistence Layer provides access to the data hosted within the boundaries of the system, and data exposed by other networked systems; perhaps accessed through services.

The Controllers Layer is used to separate completely the model and the view. The controller classes are responsible with the manipulation of data flow and with the updating of the view.

The Business Layer implements the core functionality of the system, and encapsulates the relevant business logic. It consists of components which expose service interfaces other callers can use.

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

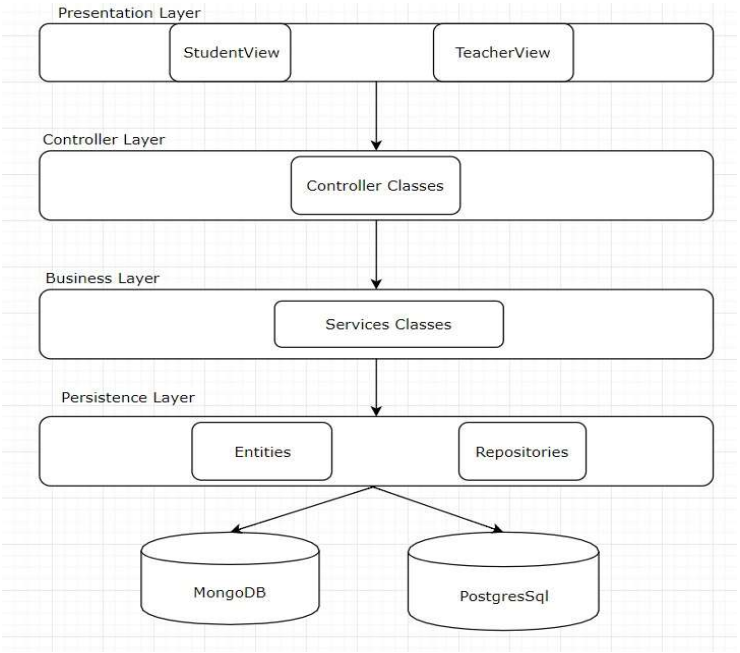
How the MVC Architectural Pattern is applied:

The MODEL: consists of the classes that represent the entities from the databasses and that hold the required data to be processed. These classes can be found in the persistence Layer.

The VIEW: is modeled in the presentation layer and consists of the classes that are used to create the GUI.

The CONTROLLER: consists of the classes that are used to separate the view from the model and that control the flow of data.

A more detailed overview is the following, in which we can see the classes that can be found in each layer:



Classes in the Presentation Layer:

- Student View Classes
- Teacher View Classes

Classes in the Controller Layer:

- Controller Classes

Classes in the Business Logic Layer:

- Services Classes

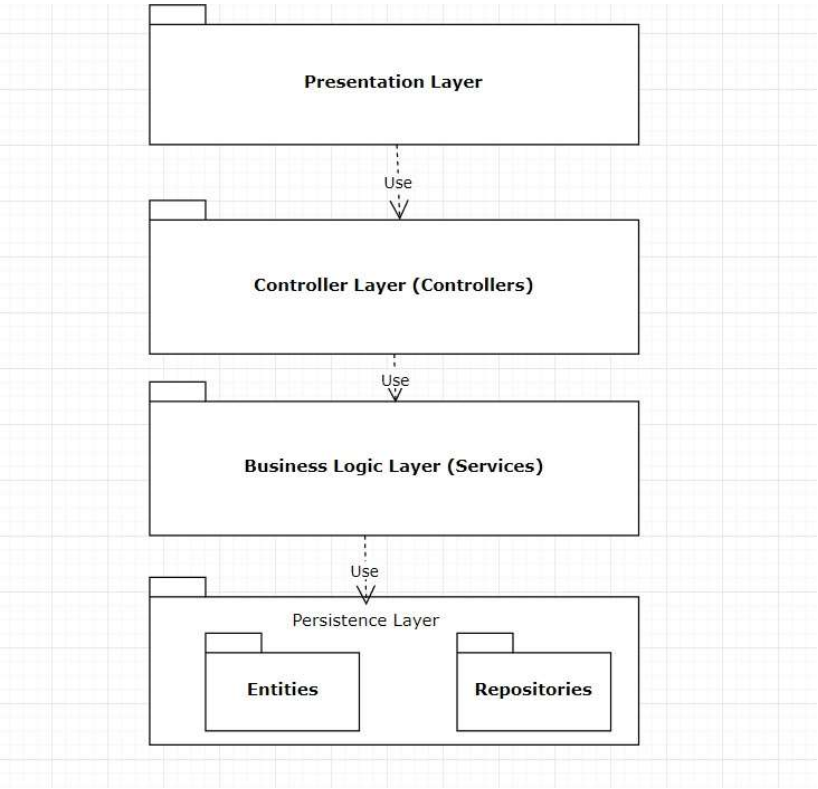
Classes in the Persistance Layer:

- Entities
- Repositories

❖ The package diagram:

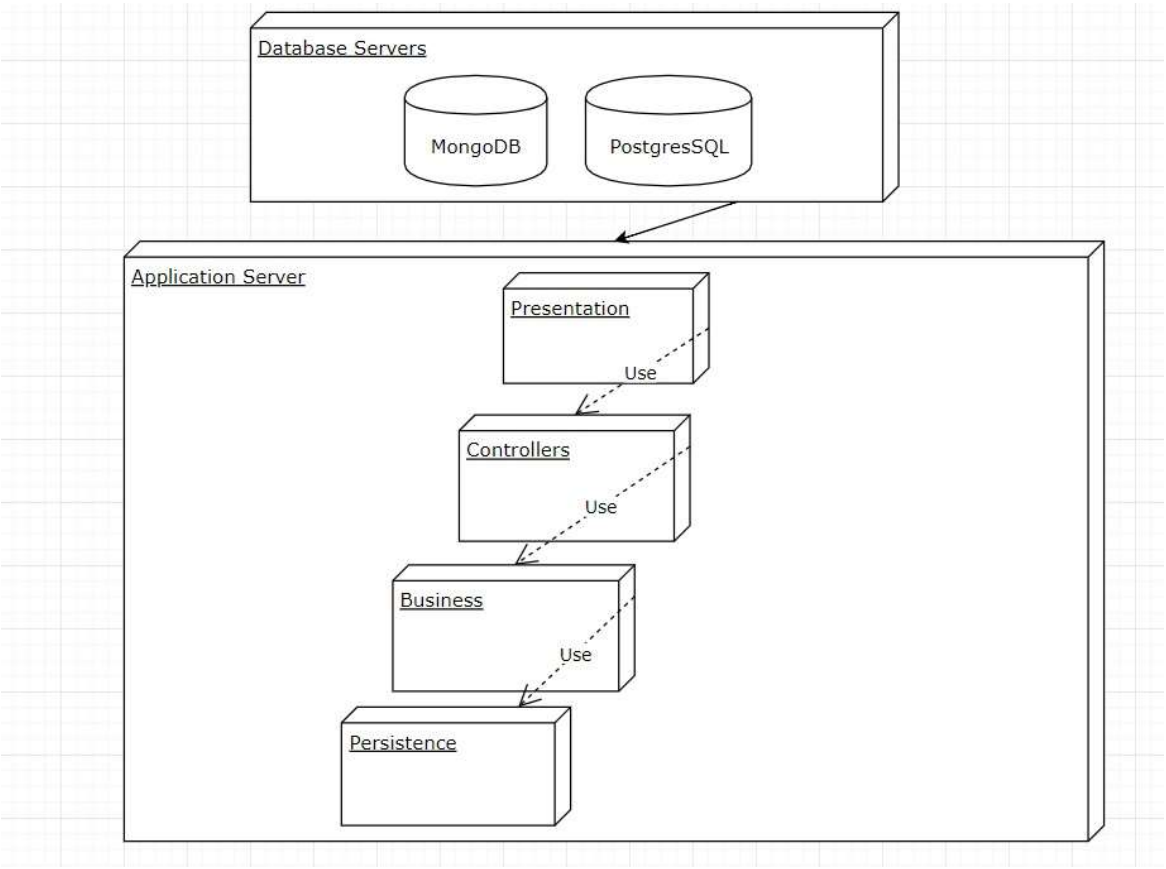
The Layered achitectural pattern is implemented using a separation of the application classes into packages according to their functionalities.
In the application we have the following packages:

- presentation
- controller
- business
- persistance with the subpackages:
  - persistance.entities
  - persistence.repositories



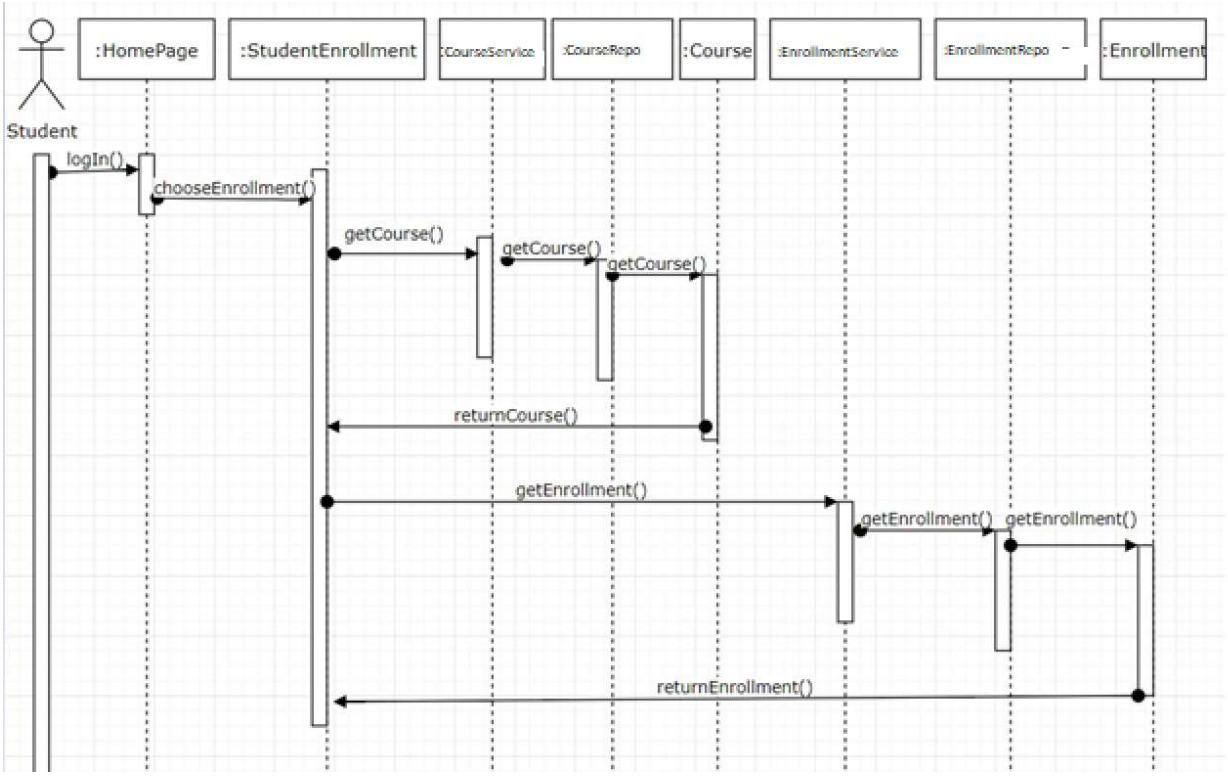
❖ The deployment diagram:

The deployment diagram is a structure diagram through which we see the architecture of our application distributed through artifacts. The artifacts represent physical elements from the real world that are a result of a development process. In our application, we only have 2 components, i.e the Database Server and the app itself, which is a desktop application.
The principal elements of the deployment diagram are the nodes.



4. UML Sequence Diagrams

A relevant scenario is the one in which a student views the course and the enrollment details for a chosen course:



5. Class Design

5.1 Design Patterns Description

I have chosen to use the Factory Method design pattern.

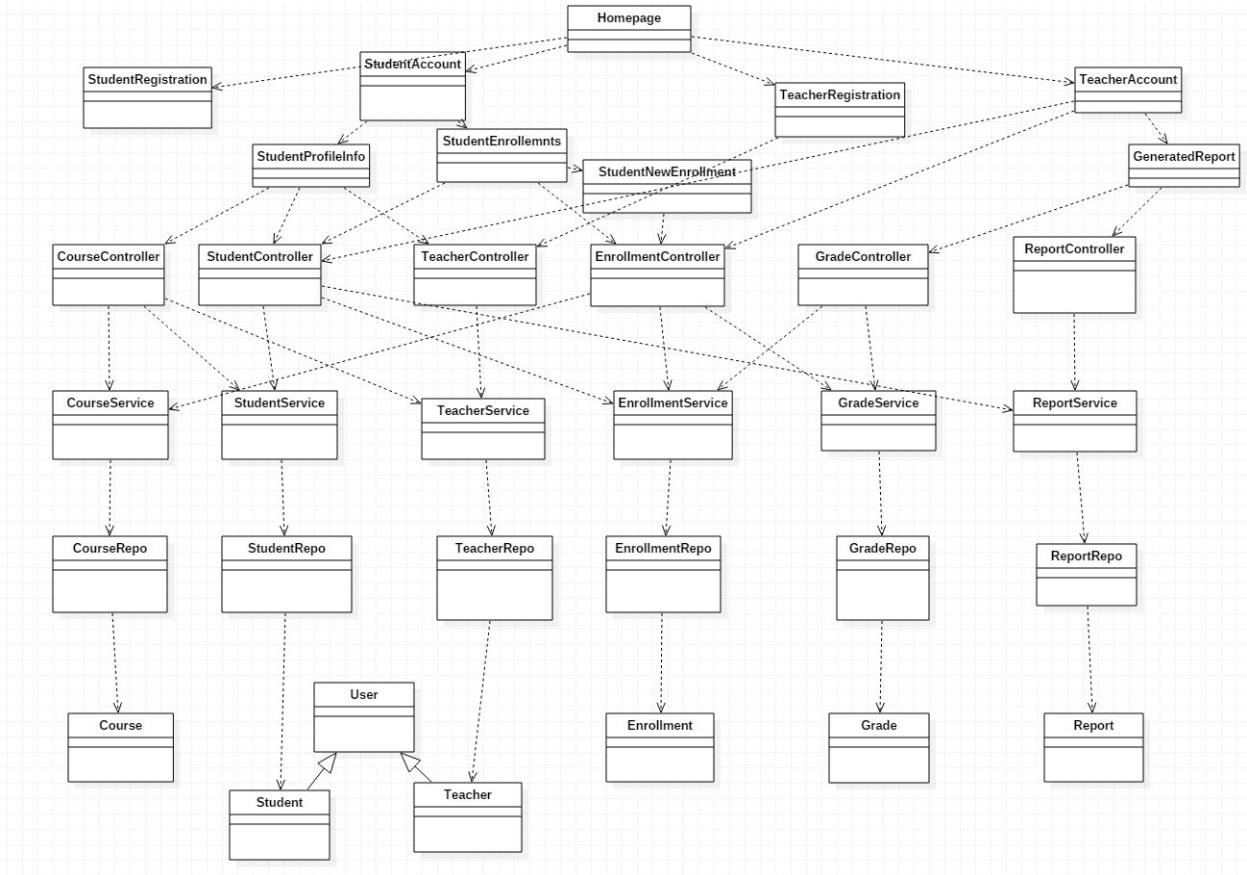
The Factory Method design pattern:

Factory design pattern is used when we have a super class with multiple sub-classes and based on input, we need to return one of the sub-class. This pattern take out the responsibility of instantiation of a class from client program to the factory class.

The context in which it will be used:

In the application will have a superclass USER and two sub-classes TEACHER and STUDENT and we will create a Factory class that will have a method that based on the parameters value (input data coming from the GUI) will decide which kind of account to create – for student or for teacher.

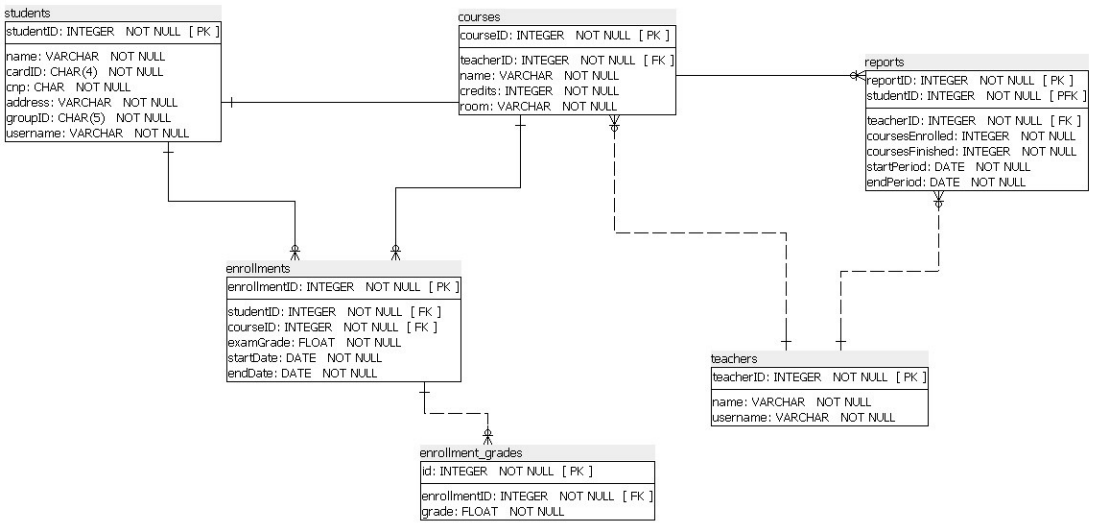
5.2 UML Class Diagram



6. Data Model

I have chosen to represent the data model used in the system`s implementation using the Relational Database Model. In this way both the data, and the exact relationships between it are clearly specified.

The database will be created in PostgresSql and the student reports will be stored in MongoDB.



7. System Testing

- The used testing strategies are the following:

Unit Testing

By unit testing the developer writes a piece of code that executes a specific functionality in the code to be tested ans asserts a certain behavior or state.

I choose to use Unit Testing to test methods that implement the CRUD operations on student information.

Individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

When we combine different units that were separately tested (unit testing) we test if the units can be correctly combined in order to obtain the entire architecture of the system.

In perform Junit testing by testing some of the methods of classes from the Business Layer – Services classes. We create the test classes StudentServiceTest and TeacherServiceTest in order to test some of the methods these classes provide. We use assertions to check if the provided result after is the expected one.

8. Bibliography

https://dzone.com/articles/java-the-factory-pattern
https://spring.io/guides/gs/accessing-data-mongodb/
https://www.tutorialspoint.com/hibernate/index.htm
https://www.codeproject.com/Articles/879896/Programming-in-Java-using-the-MVC-architecture
https://springframework.guru/configuring-spring-boot-for-postgresql/