TUCN student management system
Analysis and Design Document

Student: Ács Dávid
Group: 30432

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

A student management system, where information is stored and managed, related to students, courses, exams, enrollments. Teachers and administrators can view reports of student's activities. Access to the system is only available if the a correct username and password is provided.

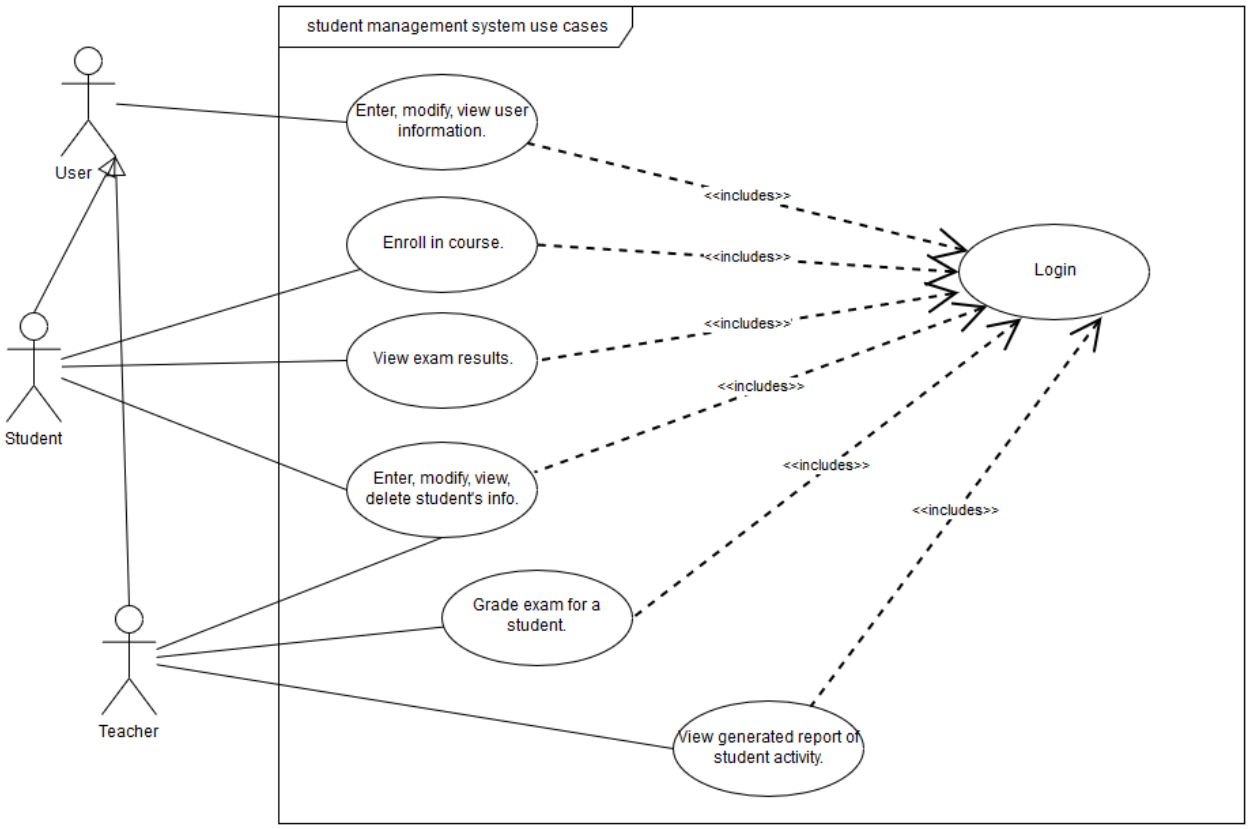
1.2 Functional Requirements

1.2.1 **Login:** Users can log in if they provide the correct username, password combination.
1.2.2 **User information:** Users can enter, modify, view information about themselves: such as name, ICN, PNC, address.
1.2.3 **Student information:** Students can enter, modify, view, delete, view student profile: ID, group, enrolments, grades.
1.2.4 **Process class enrollment:** Enroll students, grade students, view exams.
1.2.5 **Administrators can manage student's info:** administrators can create, read, update and delete all the information of students.
1.2.6 **Reports for teachers:** Create a report, assessing a performance of a student for a specific period.

1.3 Non-functional Requirements

1.3.1 **Portability:** The system must be able to run on all major operating systems: Windows, Linux.
1.3.2 **Response time:** The system must respond to user input in 1s in 99% of cases.
1.3.3 **Maintainability:** Mean time to repair should be as low as 40 hours.
1.3.4 **Readability:** The formatting of the code should reflect the logical structure of code.
1.3.5 **Performance:** Performance (throughput) is not a priority.

2. Use-Case Model



Use-case diagram detailing the needed functionality of the system.

Use case: Enroll in course.
Level: user-goal level.
Primary actor: Student.
Main success scenario:

1. The student completes the login process (prerequisite).
2. The student selects the course, he/she wishes to attend and presses the enroll button.
3. If enrollment is possible, the student will be enrolled in the course.

Extensions: the student may not have permission to enroll in the course, so he/she will be presented with an error message.

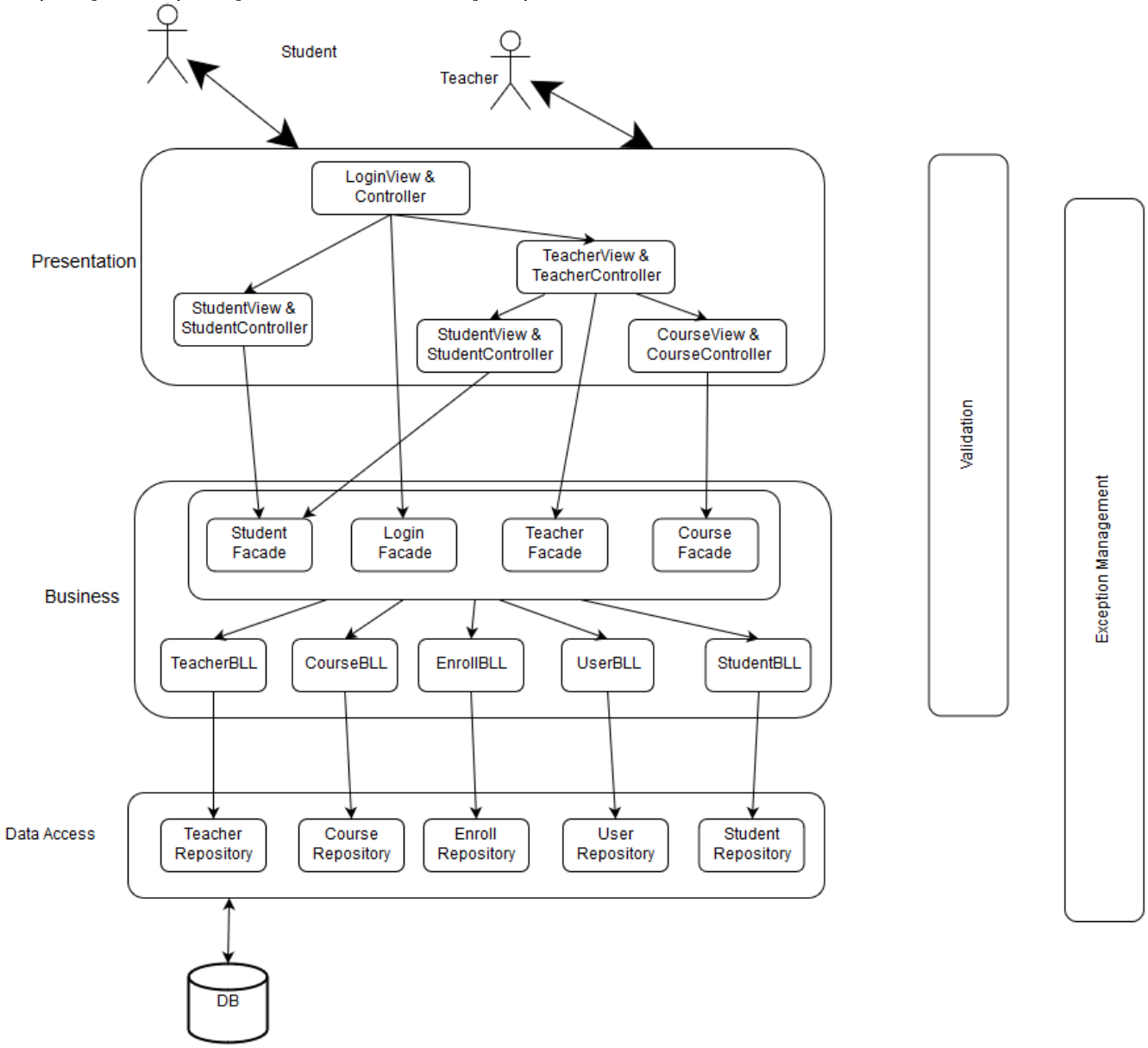
3. System Architectural Design

3.1 Architectural Pattern Description

The layers pattern is used in this project. The layers pattern consists of grouping the application into logical group of components, called layers, which resemble each other from technical point of view.
This achieves better modularization, less coupling, and thus increasing the maintainability, testability and ease of portability of the software. The downside is that each layer introduces another point of indirection, meaning performance will suffer.

3.2 Diagrams

The layers will be deployed to the same physical machine. The purpose of the layers architecture is to create a logical separation of the components. A three-tier architecture was chosen, as to do not introduce additional complexity but keep the system modular and loosely coupled.



Each of the layers has a distinct “responsibility”:

1.1.1 *Presentation layer:* Interacts with the user, showing the GUI and handles user events.

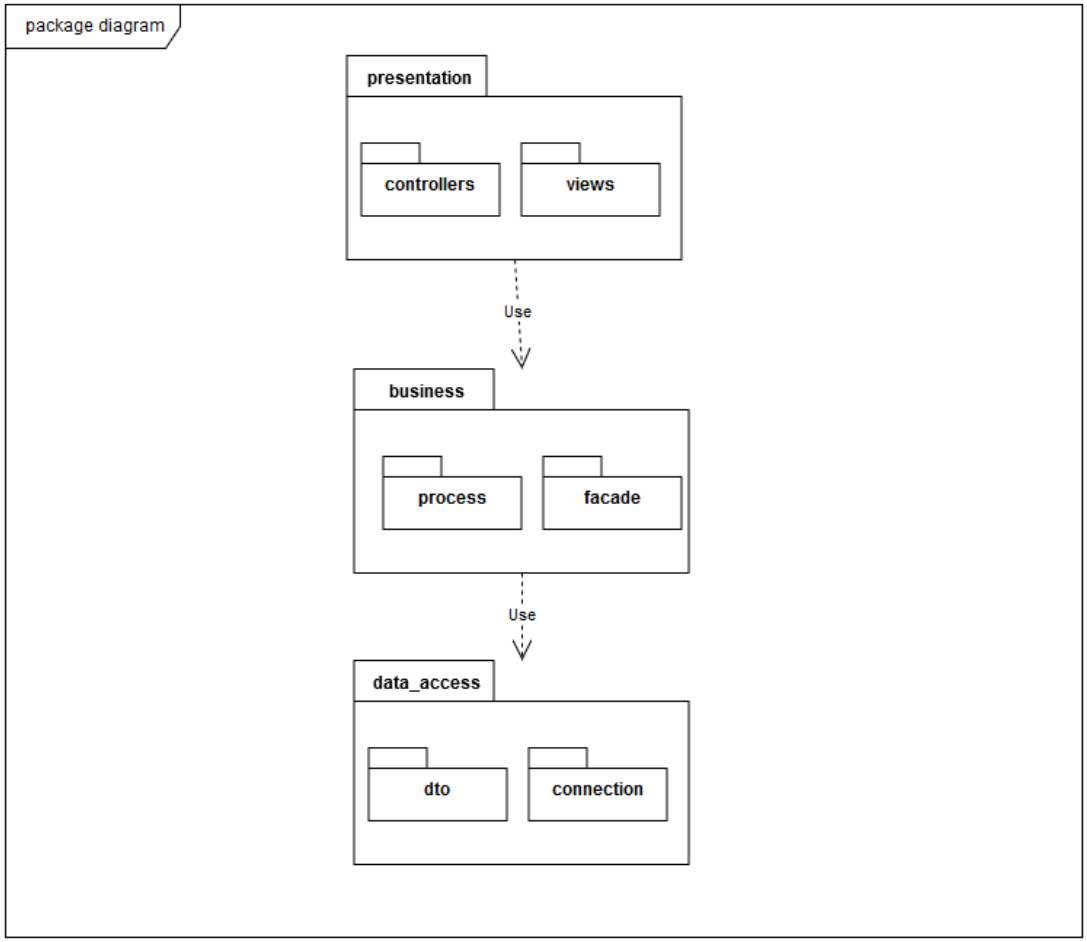
1.1.2 *Business layer:* Handles the business rules of the application, exposes logic which can be used by other components.

1.1.3 Data layer: Responsible for fetching data from various data sources, primarily from the database.

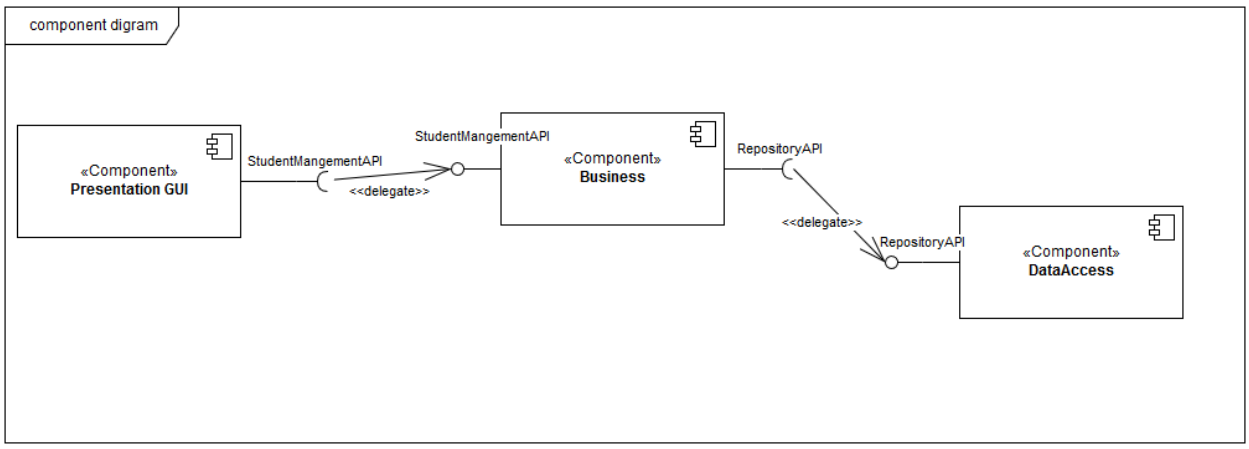
Strict interaction must be maintained between layers, meaning layers can only interact with the layers directly below them.

The cross-cutting concerns of the application are: validation and exception management, authentication is a possible candidate, but most probably it will be implemented in a single location.

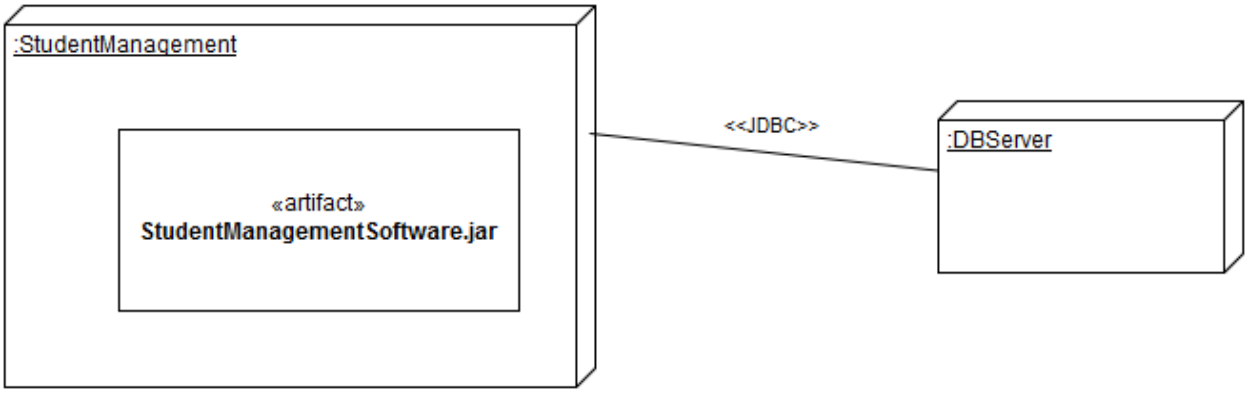
For each of the layers an abstract interface must be defined, to minimize dependencies.



Package diagram, showing the static structure of the product. This diagram reflects the layered, architecture, since each package represents a separate layer. The cross-cutting concerns are separated from the layer packages.

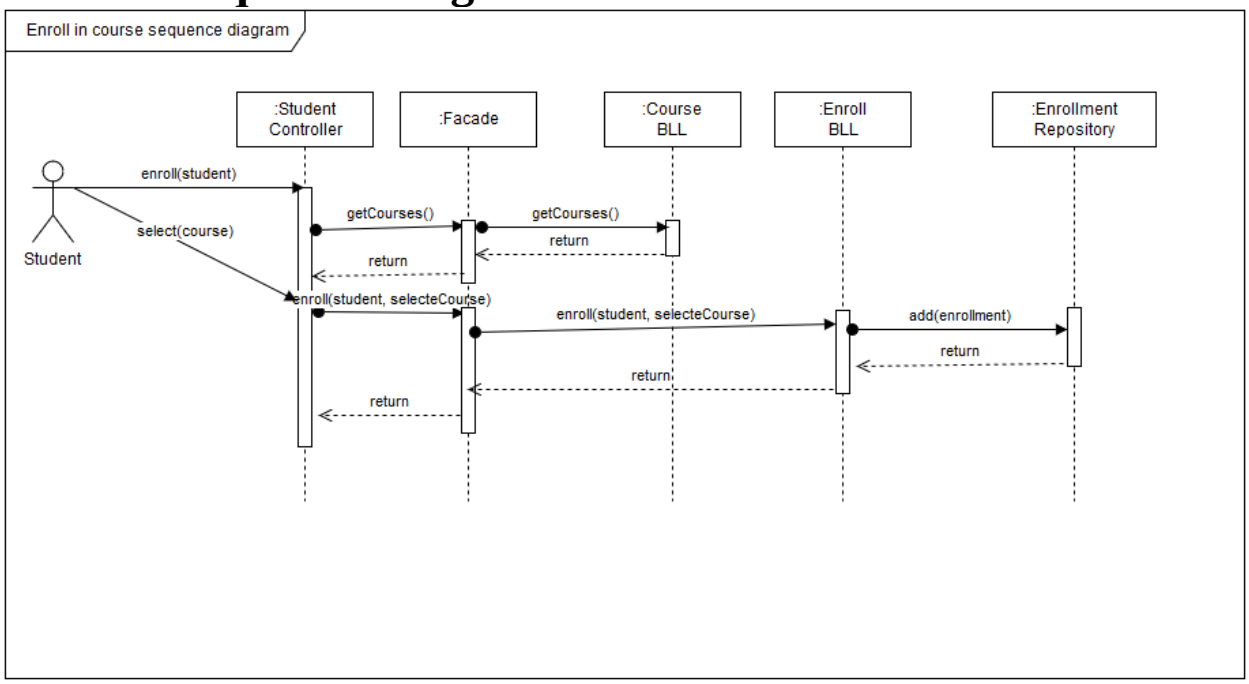


The component diagram reflects the package diagram since there is no significant difference between static construction and run-time interaction.



The deployment diagram shows us that there will be only one process, having a single jar file. This process will connect to a database and fetch the data through JDBC.

4. UML Sequence Diagrams



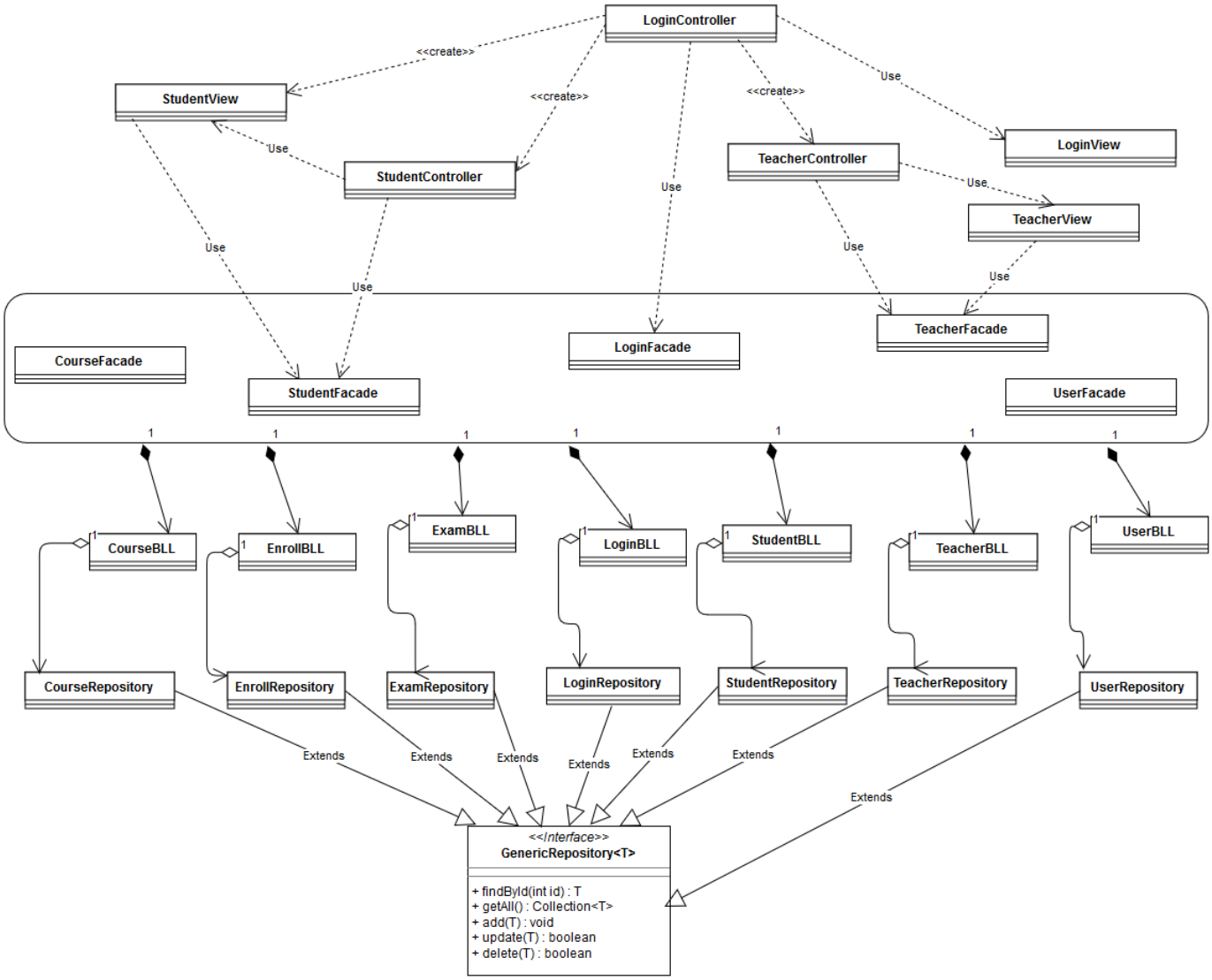
The use case diagram for the enroll in course can be seen when a student initiates the action. The user interacts with the GUI (presentation layer), which passes data to the business logic, the business logic alters the database by inserting the student id and the course id into the enrollments table.

5. Class Design

5.1 Design Patterns Description

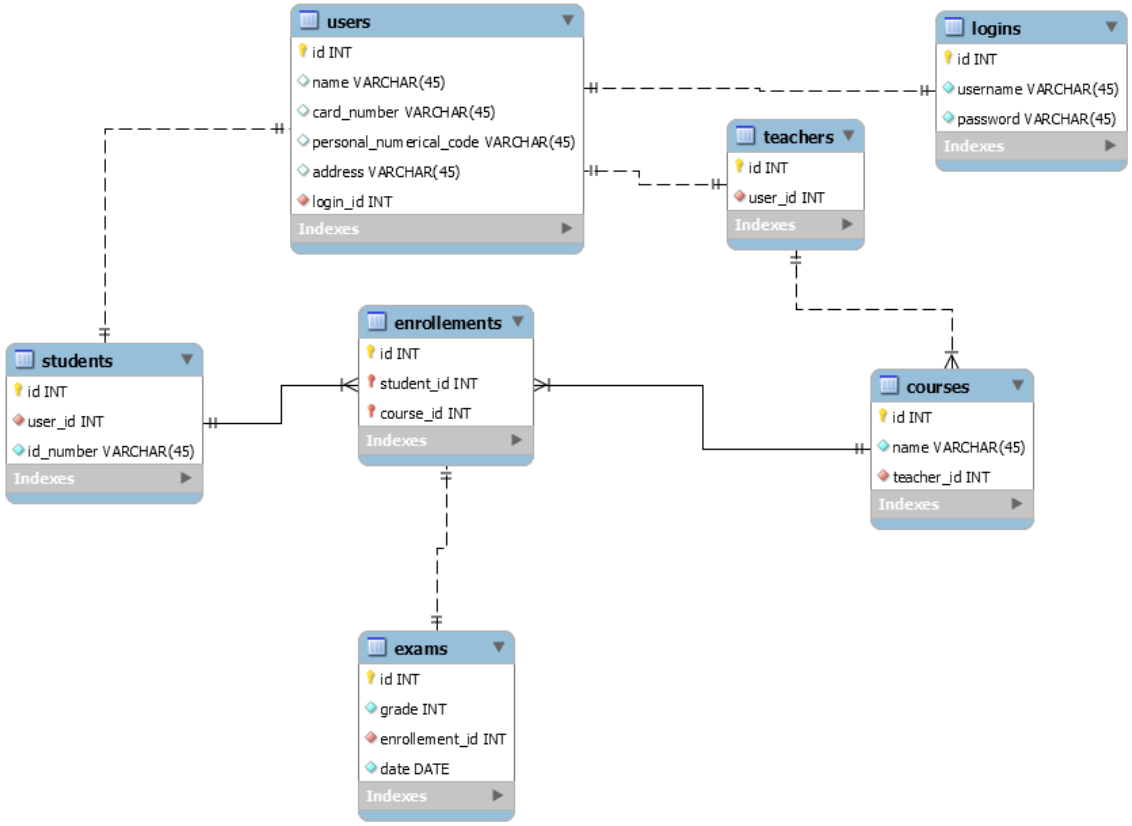
Repository design pattern: each one of the data sources is presented as an abstract repository.
The database is just an implementation of such a said repository.
Façade design pattern: the façade provides a simple interface to its' clients, by abstracting away a more complex subsystem.

5.2 UML Class Diagram



Each of the entities in the model are access through the repository (see the repository pattern).
Facades are put in a different package and used by the presentation layer.

6. Data Model



The data model (entity relationship model) is presented above.

7. System Testing

Unit testing and system testing will be performed as the system is implemented and once the system has been completely built. The unit testing method chosen is mocking the objects.

8. Bibliography

Conceptual architecture: http://www.bredemeyer.com/ArchitectingProcess/ConceptualArchitecture.htm
Component diagram: https://www.ibm.com/developerworks/rational/library/dec04/bell/
Package diagram: https://www.uml-diagrams.org/package-diagrams/model.html
Layered architecture: https://msdn.microsoft.com/en-us/library/ee658109.aspx
Deployment diagram: https://www.tutorialspoint.com/uml/uml_deployment_diagram.htm
http://www.agilemodeling.com/artifacts/deploymentDiagram.htm