Students management system
Analysis and Design Document

Student: Fazekas Vlad
Group: 30234

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. There are two kinds of users (students and professors) and each of them with the respective permissions and functionalities.

1.2 Functional Requirements

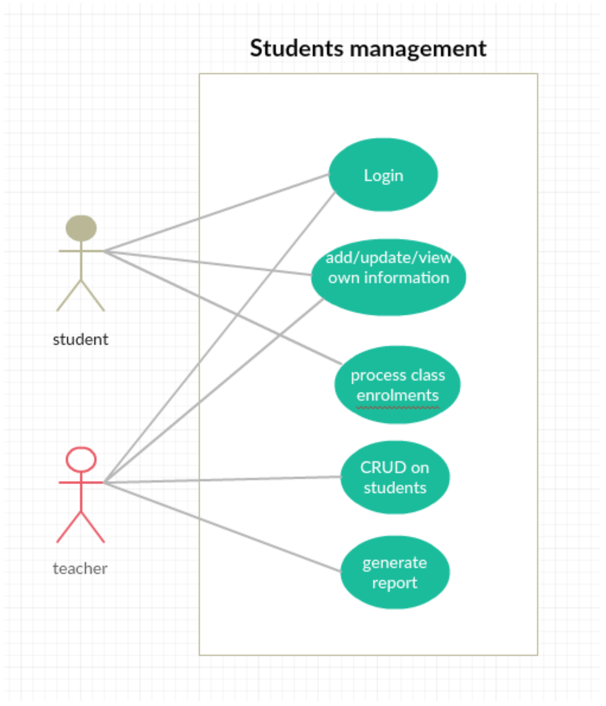
Between the functional requirements reside the following pints:

- Two types of users: students and teachers
- Login section(only hardcoded data)
- Students should be able to:
  - add/update/view their own personal information
  - Create/update/delete/view school related information(group, enrolments, grades)
- Teachers should be able to:
  - Process their own personal information
  - Apply CRUD(create, read, update, delete) operations on students
  - Generate reports for a particular period containing the activities performed by a student
- Relational database model for storing the data
- Layers architectural pattern
- Modev-view-controller
- Dependency injection
- Hibernate
- JUnit Testing with Mockito

1.3 Non-functional Requirements

The system should validate the data in order to maintain the data integrity, should be reliable and should be able to be extendable.

2. Use-Case Model



Use case: Login
Level: login level
Primary actor: students or professors
Main success scenario: tries to login, check data with database, login successful
Extensions: login failed, application won't provide further functionality

---

Use case: Own information processes
Level: information level
Primary actor: students
Main success scenario: add, update or view their own information
Extensions: no information is displayed

Use case: Process class enrolments
Level: information level
Primary actor: students
Main success scenario: having the ability to enroll to different classes
Extensions: not enrolled in any class

Use case: Crud operations on students
Level: information level
Primary actor: teachers
Main success scenario: having the ability to apply CRUD operations on student data
Extensions: no new modification saved

Use case: Generate reports
Level: information level
Primary actor: teachers
Main success scenario: reports should be generated for a specified period of time
Extensions: no report is generated

3. System Architectural Design

3.1 Architectural Pattern Description

Architectural layer pattern:
Dividing an application into separate layers that have distinct roles and functionalities helps you to maximize maintainability of the code, optimize the way that the application works when deployed in different ways, and provides a clear delineation between locations where certain technology or design decisions must be made.
That's why we will divide our application in three big layers:

- Presentation layer (where all the interaction of the user with the system will reside)
- Business layer (where the core functionality and relevant logic of the system will be implemented)
- Data layer (where access to data is provided(generally through generic interfaces that the components in the business layer can consume)

Model–View–Controller pattern:

Model–View–Controller (usually known as 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. That's why we will divide our application in the following structure:

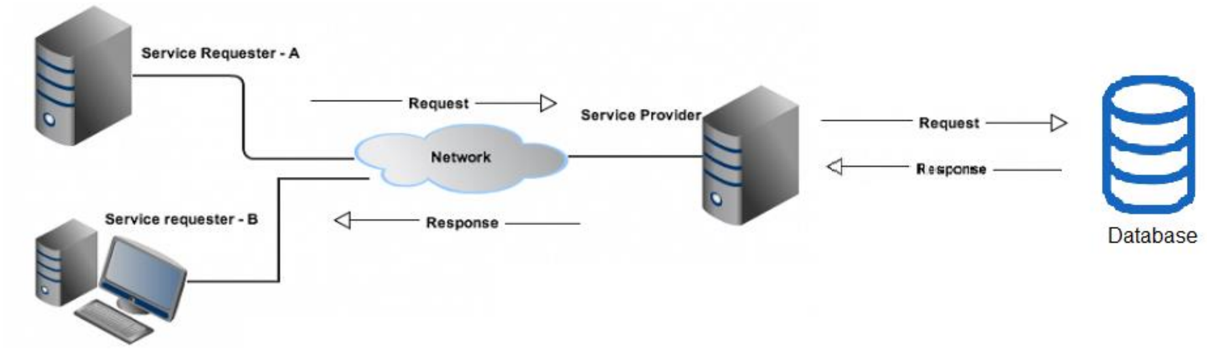
- Controller
- View
- Model
  - Entities
  - Repositories
  - Services
- Database

Dependency injection is a technique whereby one object (or static method) supplies the dependencies of another object. For dependency injection guice was used.

3.2 Diagrams

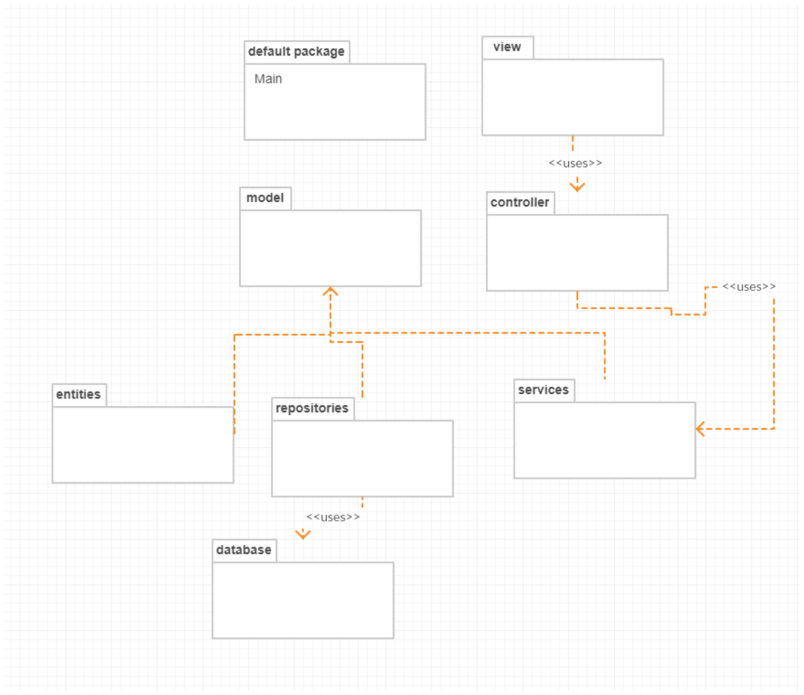
System general diagram:

The system diagram represents a classic web application system, that has clients, which request a service from a server, the server gets the data from a database, applies some logic and returns a response to the client.

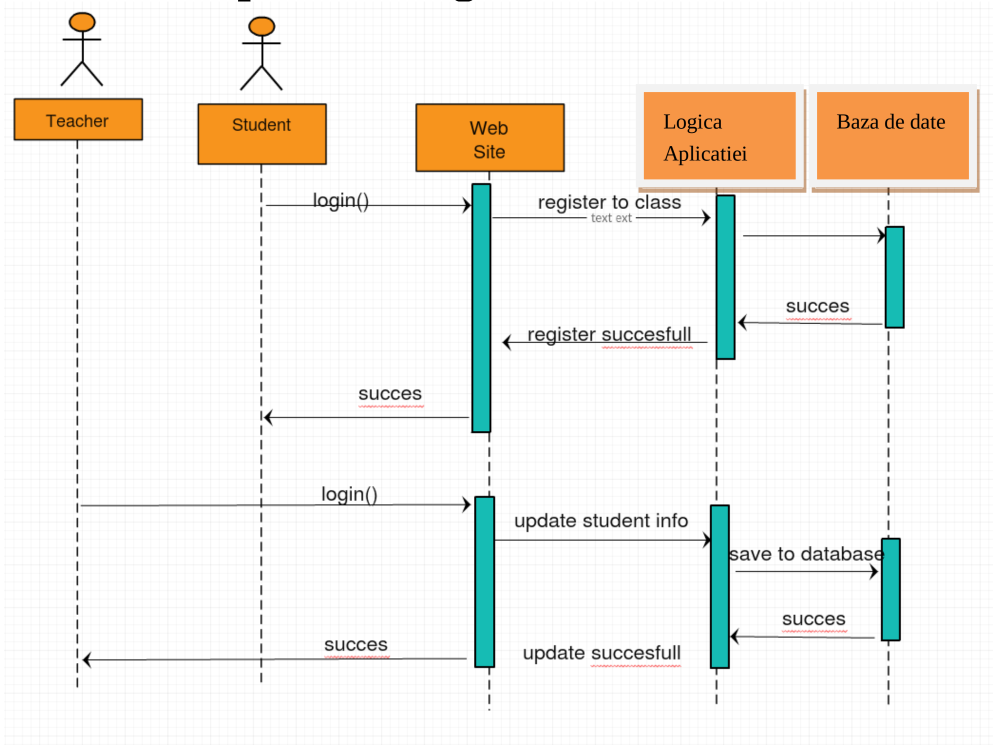


Package diagram:

We try to maintain the packages simple and structured, so we don't split the application in too many folders, but we keep an organized structure and clearly delimitate the layers based on functionality.



4. UML Sequence Diagrams

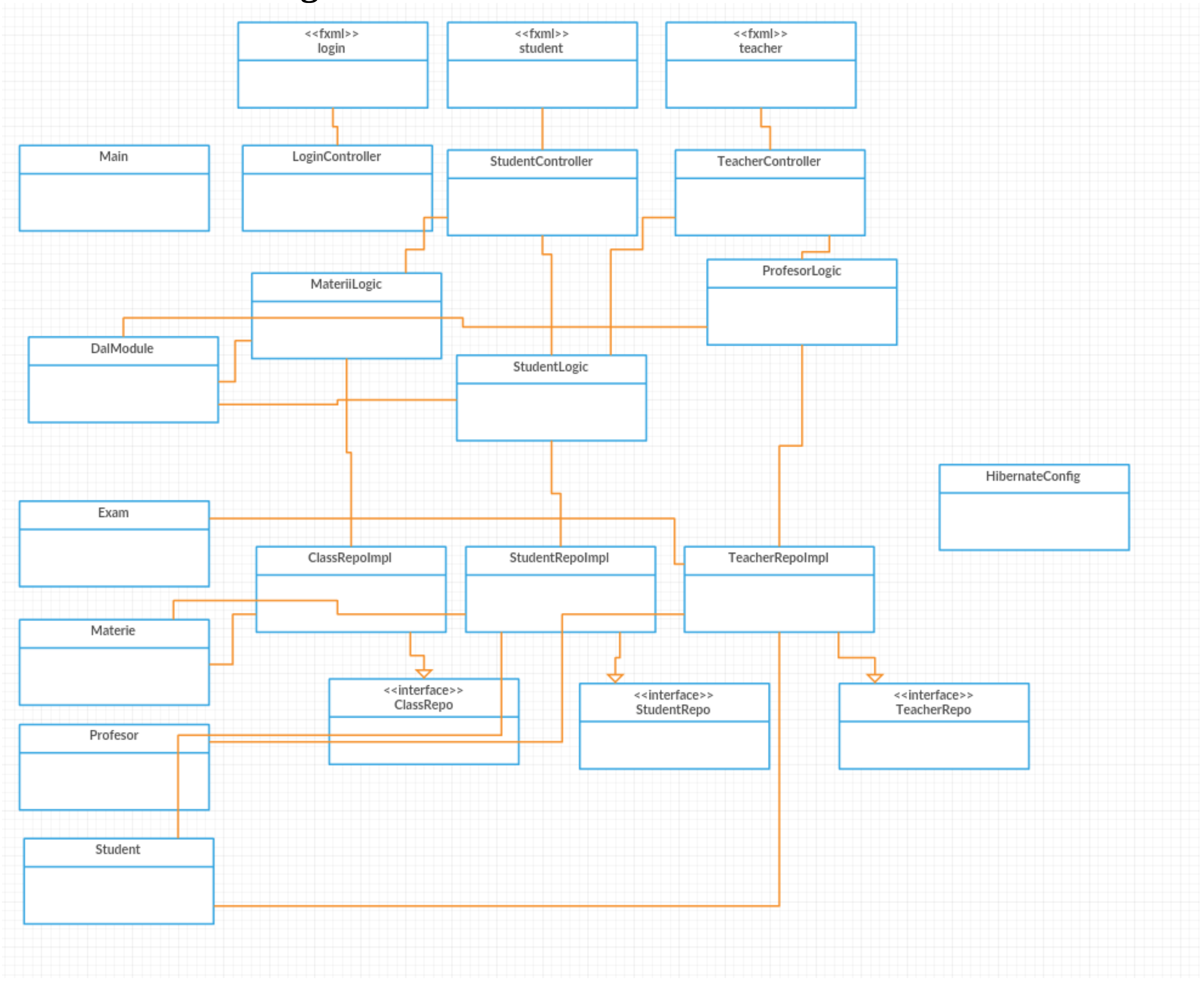


5. Class Design

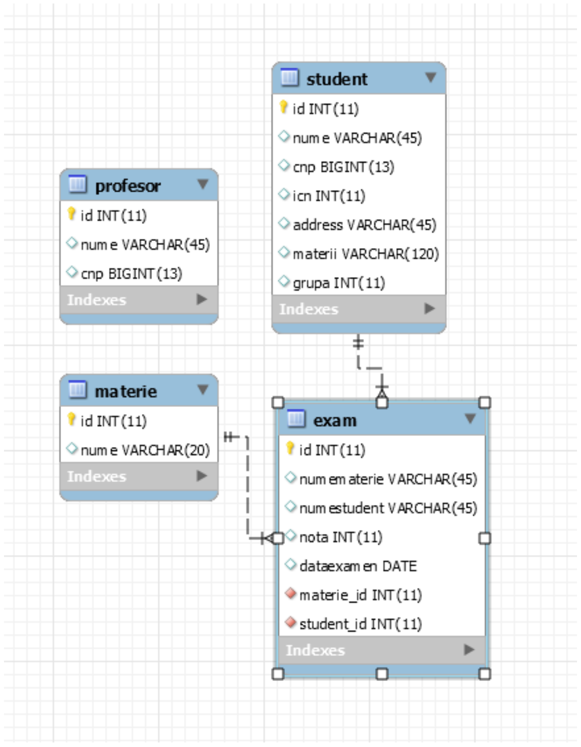
5.1 Design Patterns Description

Singleton design pattern:
This pattern resides under the creational patterns signature and involves the concept that a single class is responsible to create an object while making sure that that one is the only single object that is created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

5.2 UML Class Diagram



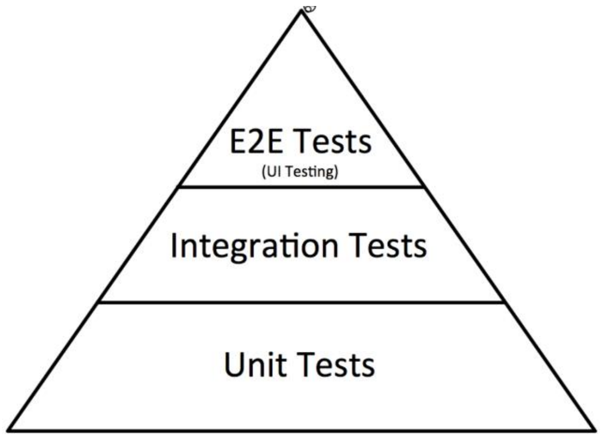
6. Data Model



7. System Testing

Following the testing pyramid model, we know that the most tests should be written at the unit layer, because that is also the easiest and most important. That’s why we will focus primary on unit tests written using Junit and mockito. When we come to Integration tests we consider that it is not necessary to implement those kind of tests at such a simple application.

The logic of the application is tested using Mockito and mocks. The Classes that are tested are ProfesorLogic, StudentLogic, MateriiLogic.



8. Bibliography

Information about Layered Application:

https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658109%28v%3dpandp.10%29

Information about practical testing pyramid:

https://martinfowler.com/articles/practical-test-pyramid.html

Testing with Mockito:

https://www.baeldung.com/mockito-series

Guice dependency injection

https://www.baeldung.com/guice