Assignment 2 - Student Management Application

Analysis and Design Document

Student:Jakab Gyöngyi Anikó

**Group:30233**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 6

4. UML Sequence Diagrams 7

5. Class Design 8

6. Data Model 10

7. System Testing 10

8. Bibliography 10

1. Requirements Analysis

# Assignment Specification

Design and implement a Java application for the management of students in the CS Department at UTCN. The application should have two types of users (student and teacher/administrator user) which have to provide a username and password in order to use the application.

The regular user can perform the following operations:

* Add/update/view personal information (name, identity card number, personal numerical code, address etc.)
* Create/update/delete/view student profile (identification number, group, enrolments, grades)
* Process class enrollment (enroll, exams, grades).

The administrator user can perform the following operations:

* CRUD pe student information
* Generate reports for a particular time period containing the activities performed by a student

# Functional Requirements

The functional requirements of this project include the following: adding, updating and viewing a student's personal information, creating, viewing, modifying and deleting student profile information, also a management part for student enrollment processing for different courses, viewing grades. The application provides support on the admin side too, which involves the ability to create, view, modify and delete student profiles and generate reports about a student's activity for a specific period.

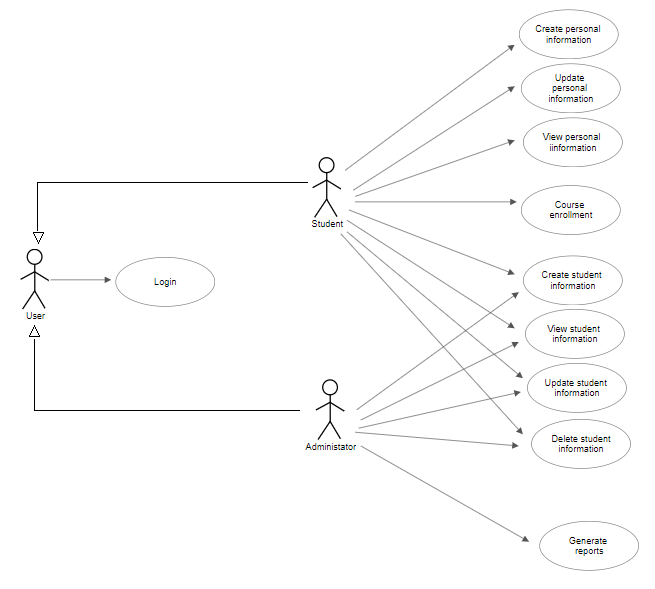
# Non-functional Requirements

Non-functional requirements of the project are:

* Security: the system provides valid data/operation for different requests based on user authentication data and the authorization that is provided by the system for a specific user type
* User – friendliness: the application should be easy to use independently of the user type
* SOLID Principles: the system should be designed in a way that modifications, maintenance and extensions are easy to make in the future

2. Use-Case Model

The actor called user - the general user before connecting to the application - by using a username and password is logged into the application and from that moment we are talking about two types of specific users: the student can perform the operations: add / update / view personal information and CRUD on student information, also enrollment processing, while the administrator can create/view/update or delete a student's information and generate reports based on the activity of a student for a specific time period.



*Use case: Login*

*Level: < user-goal level >*

*Primary actor: <User >*

*Main success scenario: The application opens, the user provides a username and a password, the system responds to it with opening up the main student or administrator user interface based on user type, so that each user can operate based on its authorization level*

*Extensions: if a user does not used the system before, the system administrator will provide for him/her a default account with the student name as username and student identification number as password*

*Use case: Create personal information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: The student user inserts the desired information into the textboxes on the user interface and clicking on the button adds a new information to the part assigned for him/her in the database*

*Extensions:*

*Use case: Update personal information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: The student user provides the data in textboxes, which will be processed by the system and further communicated to the database for update of the current informational state*

*Extensions:*

*Use case: View Personal information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: The student user can view his/her personal information without any special request, it is displayed by the system by default is the database holds any kind of information of him/her in its current state*

*Extensions:*

*Use case: Create/View/Update/Delete student information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: For the student user type the system will provide a specific interface where he/she can view student profile if the system holds any information about it in its current state. The student can provide data in textboxes in order to create student profile or update different aspects and also is a able to delete the account at a request.*

*Extensions:*

*Use case: Course enrollment*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: The system provides an enrollment processor for student. The student can search for a specific course and by providing a specific enrollment key, they are able to enroll themselves to the course previously mentioned.*

*Extensions:*

*Use case: Create/View/Update/Delete student information*

*Level: < user-goal level >*

*Primary actor: <Administrator>*

*Main success scenario: The system provide user interface specific for administrator task, by viewing the currently registered student. By introducing data in specific fields, admin user are able to create new student, create student profile, alter or even delete the currently stored data.*

*Extensions:*

*Use case: Generate reports*

*Level: < user-goal level >*

*Primary actor: <Administrator>*

*Main success scenario: The admin user can search for a specific student and view their activity logs, or even filter the displayed data specifying a time interval.*

*Extensions:*

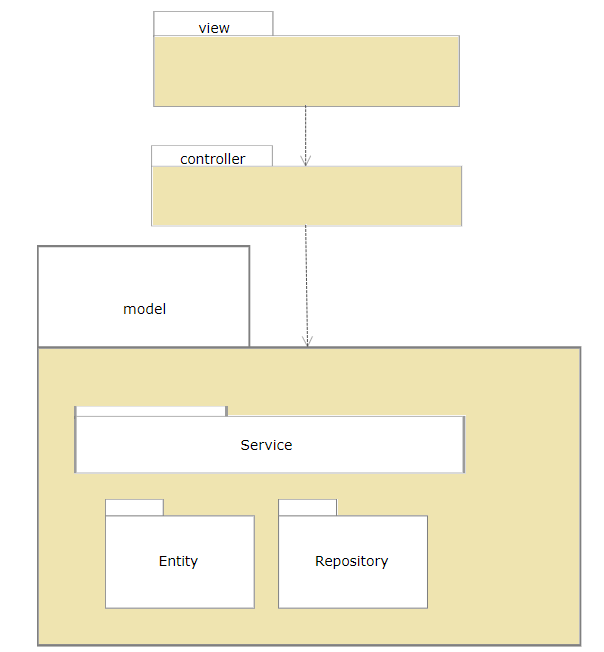
3. System Architectural Design

**3.1 Architectural Pattern Description**

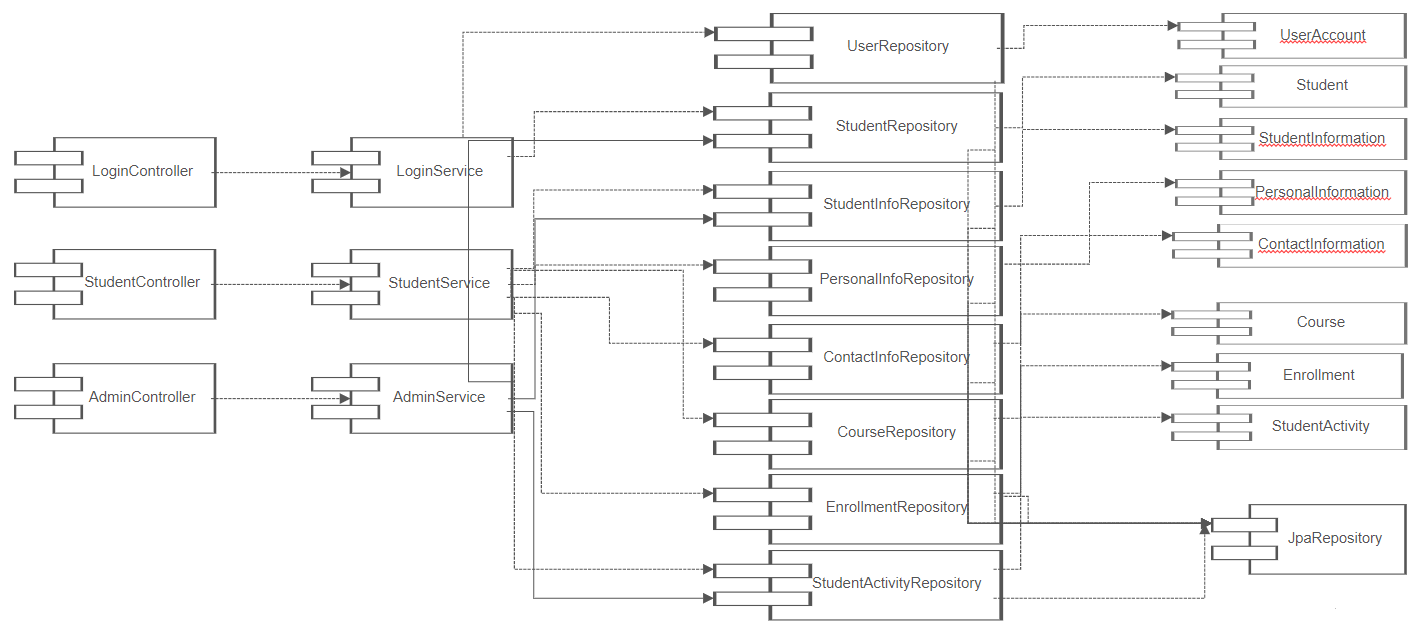
The projects design is based on the Layer architectural pattern, which is a logical separation of the following concepts: Presentation Layer, Business Logic or Domain Layer and Data Access Layer. This architectural pattern divides the project in logical layers, such that the higher-level layers only depends on lower-level layers, lower levels directly beneath the current layer or even lower depending on the strictness on dependencies. This project however is implemented with the support of Spring Framework, this way the engine template of the system is more similar of an MVC implementation. This way the system is divided in 3 different types: controller – which layer or package will contain Controller classes that manage requests from the views, view – which layer or package will contain view types objects, in this case specifically html files and css for view styling; finally the model – which layer or package will contain the business logic and the persistence layer of the project, the business package will contain services and the persistence layer will contain entities and data access objects or repositories.

**3.2 Diagrams**

Package diagram of the application:

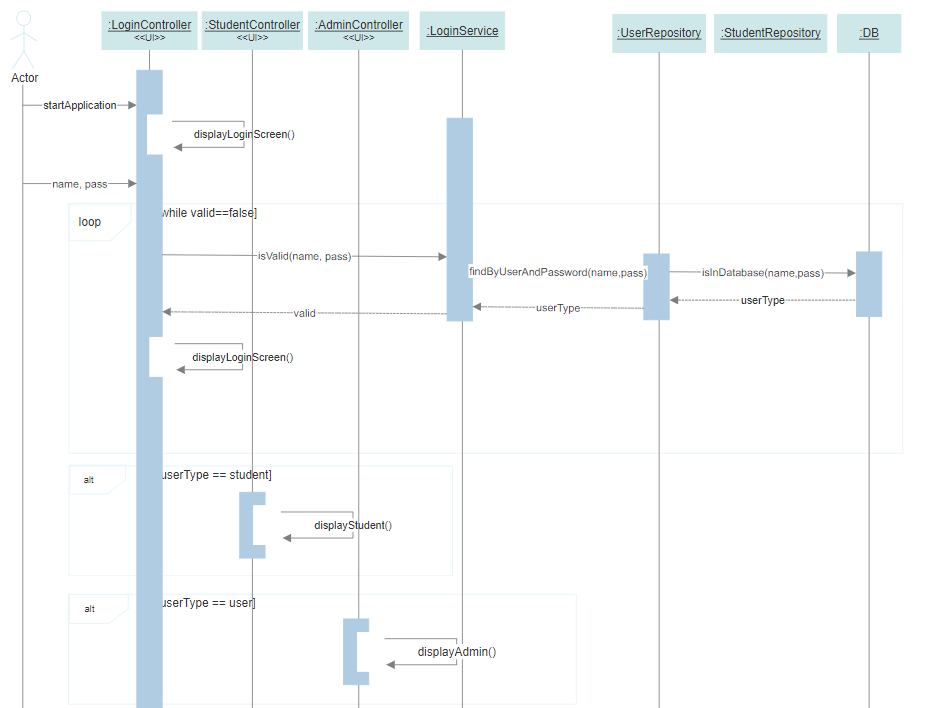
****

Component diagram of the application:



4. UML Sequence Diagrams

Sequence diagram that represents a general user login operation, where user provides a username and a password, and the system responds with a specific interface based on users type if the data provided is valid:



5. Class Design

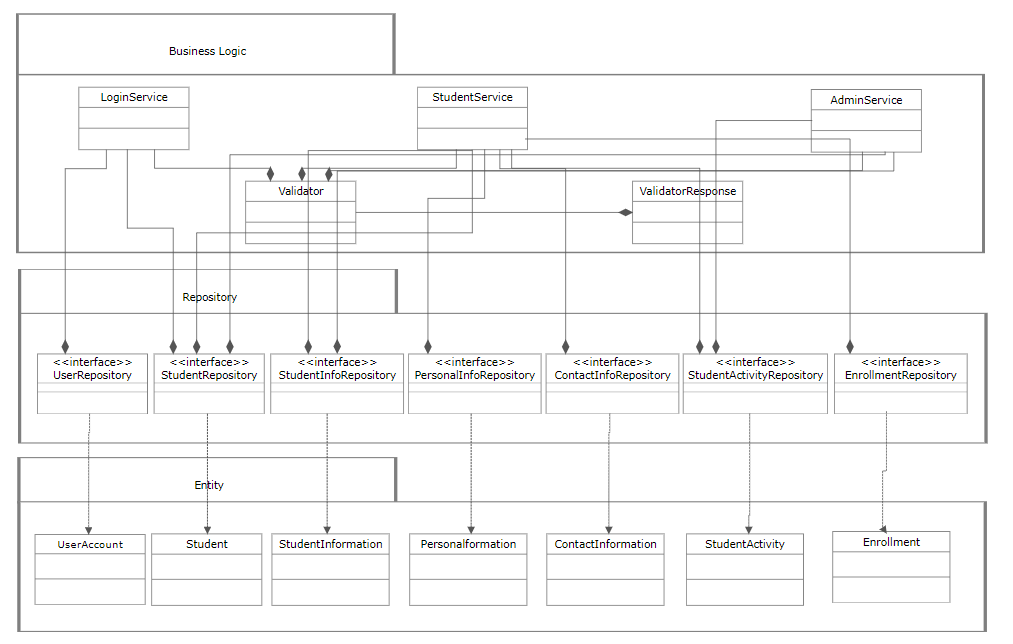
**5.1 Design Patterns Description**

Architectural patterns used in the project implementation:

* Layer: Layer architectural pattern is user for logically structure and division of the project implementation. It divides the application in 3 main layers, or packages in the specific implementation language. The project is structured in a way the high-level concepts only depend on lower-level concepts, and implementation depends only on more abstract concepts.
* MVC: Model-View-Controller architectural pattern is a presentation layer pattern, used in user interface implementation. In a SpringBoot application this architectural pattern applies for the complete structure of the implementation: the model layer – business logic/services and data access layer, the view layer – html templates and finally the controller layer – spring controllers for html request management.

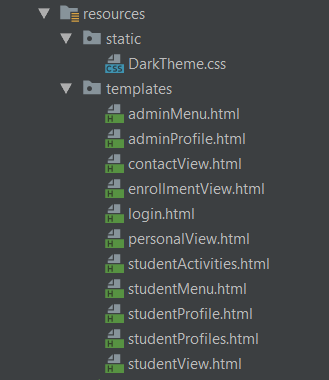
**5.2 UML Class Diagram - MVC**

Class diagram of Model package:



Class diagram of View package:

In the case of Thymeleaf templates the view package of a project consists only of HTML pages and/or CSS files for styling. These should be in the section of resources in a maven SpringBoot project, HTML file under templates directory and CSS files under static.



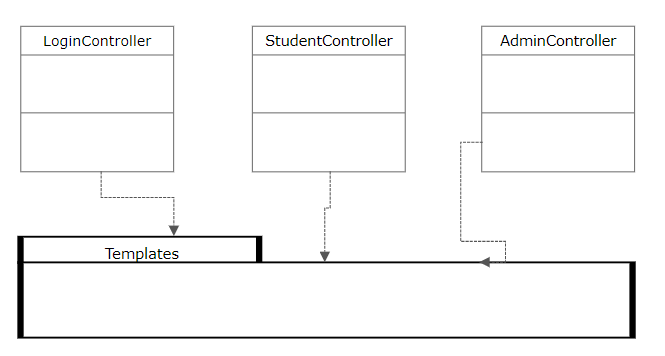
Each controller processes request for a specific view or HTML page.

LoginController: login.html

StudentController: studentProfile, studentMenu, studentView, personalView, contactView, enrollmentView

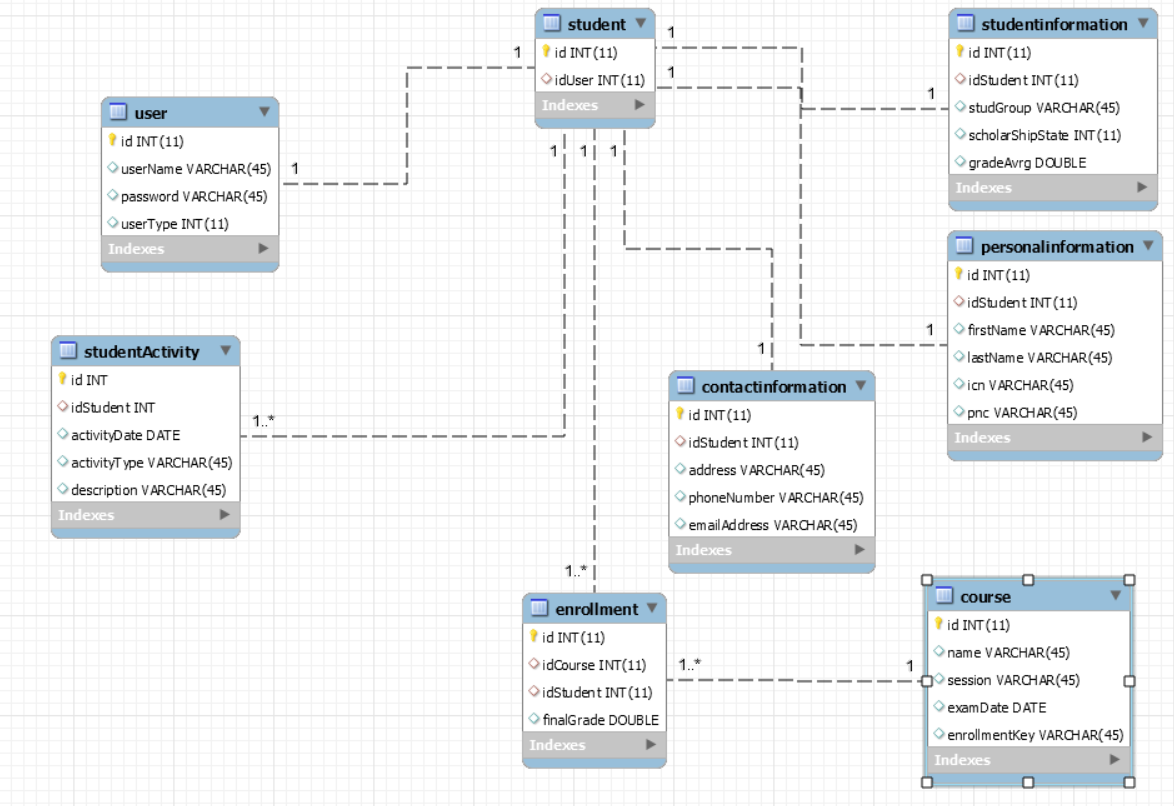
AdminController: adminProfile, adminMenu, studentProfiles, studentActivities

Class diagram of Contoller Layer:



6. Data Model

Data Model of the application – data divided in different tables and the relationships between them:



7. System Testing

System testing is made incrementally, first steps are testing the data access layer modules, then business logic layer elements, and finally the presentation layer, by testing the different use cases of the system from the perspective of the user. The module test are made in Junit.

8. Bibliography

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

[2] <https://www.lucidchart.com/pages/uml-class-diagram>

[3] <https://www.visual-paradigm.com/solution/sequencedgm/uml-sequence-diagram-tool/?gclid=CjwKCAjwycfkBRAFEiwAnLX5IePLvOG3NWH7Jocx-kpfjzzQ0OBdSez5VQ8CRpxyAdElK6ML1CoDRxoCdwEQAvD_BwE>