Assignment A2

Student: Adrian Timis

**Group: 30432**

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 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

This application aims to develop a management system for students enrolled in the Computer Science department at the Technical University of Cluj-Napoca. The application will help students to better manage their courses and make it easier for teachers to grade and generate reports for the students.

# Functional Requirements

The application has two types of users: student and teacher/administrator. The functional requirements are described for each type of user.

The regular user can perform the following operations:

* Add/update/view client information
* Create/update/delete/view student profile
* Process class enrolment

The teacher/administrator can perform the following operations:

* CRUD on student’s information
* Generate reports for a particular period containing the activities performed by a student

# Non-functional Requirements

Non-functional requirements for this assignment include:

* Manageability
* Data integrity
* Reliability

Each of these represent a requirement for the software product in order for it to be viable in real life conditions. Manageability means that the system will not be too difficult to manage, as it is expected to have a large number of users, and a difficult managing process could take its toll on the user experience. Data integrity is vital as we are dealing with personal data, and data concerning grades which matter for such things like scholarships, Erasmus selection, etc. Reliability means that even in conditions which are not favorable, ex: many users using the service at once, the system can still behave as normal, despite unfriendly conditions.

2. Use-Case Model

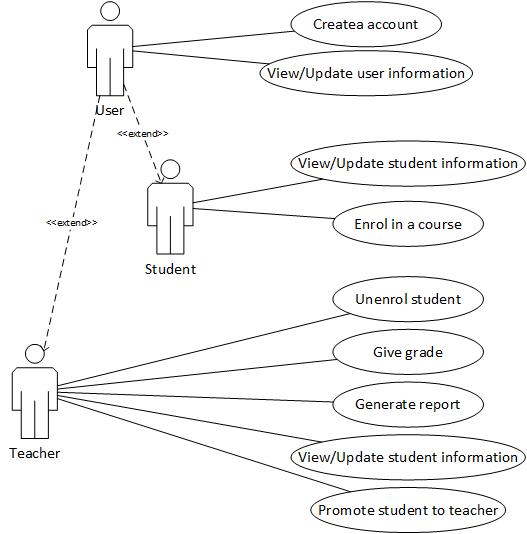
Use case: Application Uses

Level: Summary Level

Primary actor: Regular User

Main success scenario: Create Account

Extensions: view/update student’s information, generate reports, enroll in a course, update/view user information, update/delete/view student profile, promote student to teacher

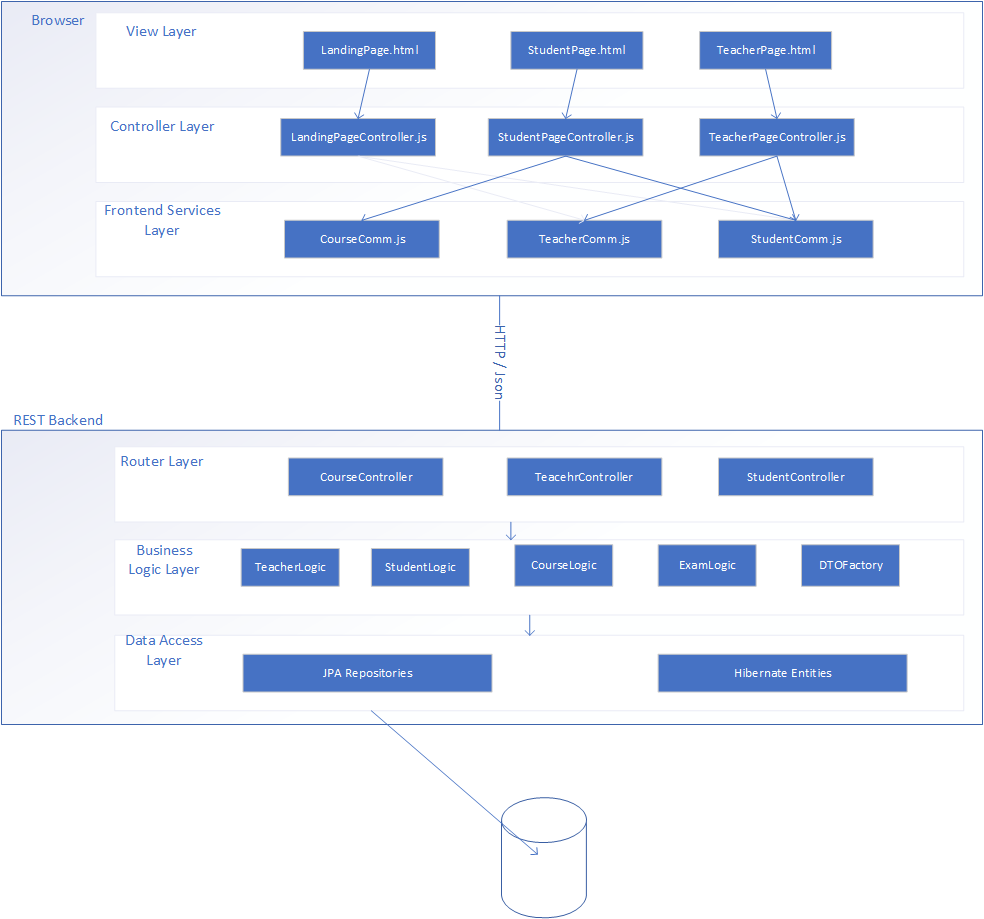


3. System Architectural Design

**3.1 Architectural Pattern Description**

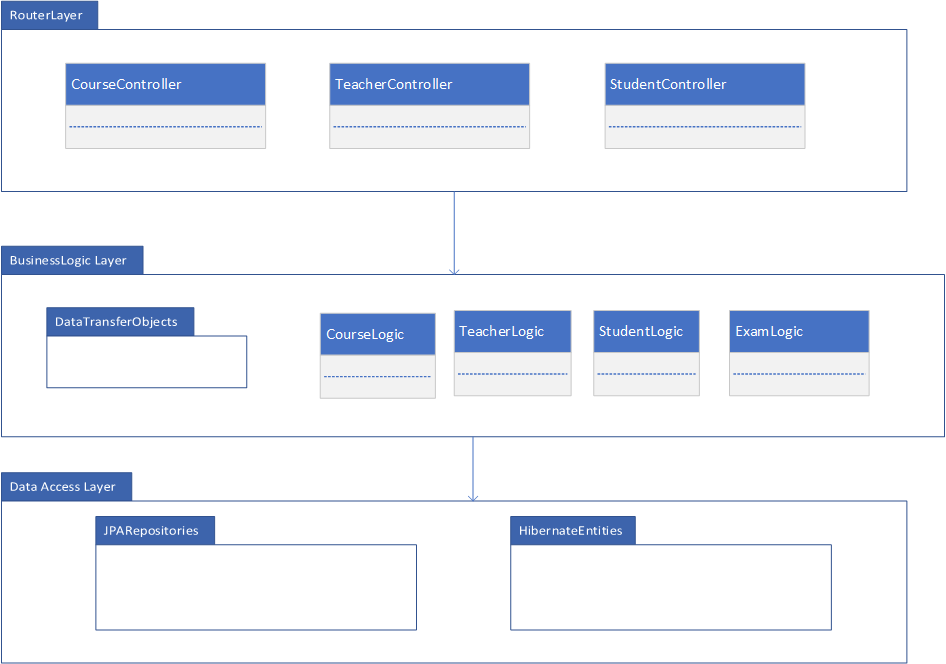
Spring MVC and AngularJs together make for a really productive and appealing frontend development workflow, therefore, this application will be built using these two frameworks. The frontend is MVC-capable and contains all the presentation logic which is separated in a view layer, a controller layer, and a frontend services layer. After the initial application startup, only JSON data goes over the wire between client and server. The main idea compared to other more traditional server-side architectures is to build the server as a set of stateless reusable REST services, and from an MVC perspective to take the controller out of the backend and move it into the browser.

**3.2 Diagrams**

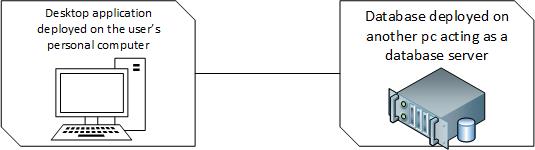
**

Above is the architectural diagram. The separation between the components is made clear by the diagram.

Package diagram:

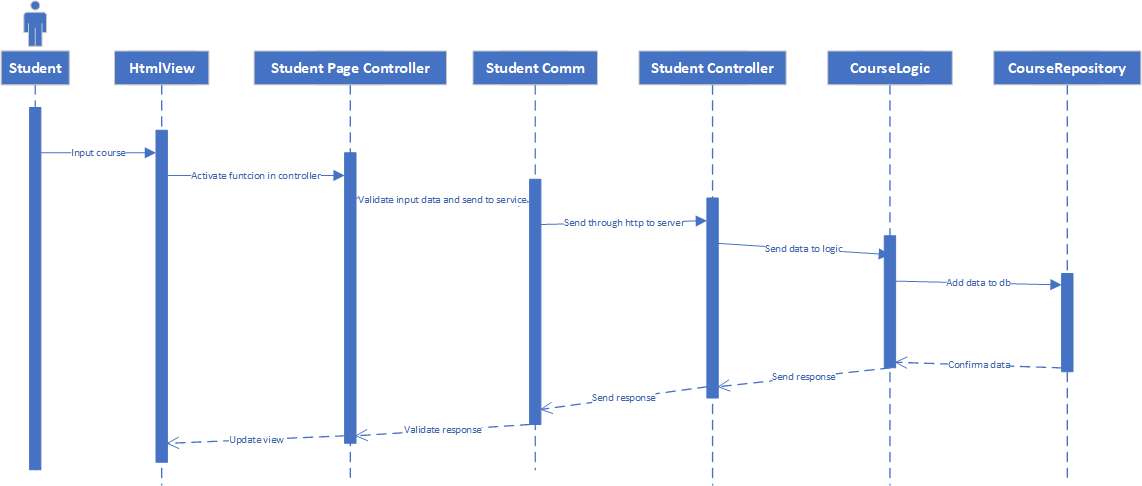


Deployment diagram:



4. UML Sequence Diagrams

Sequence diagram for enrolling in a course:

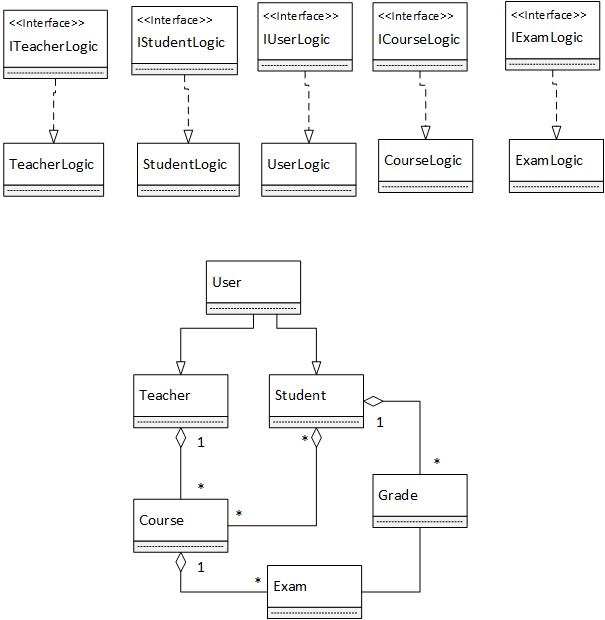


5. Class Design

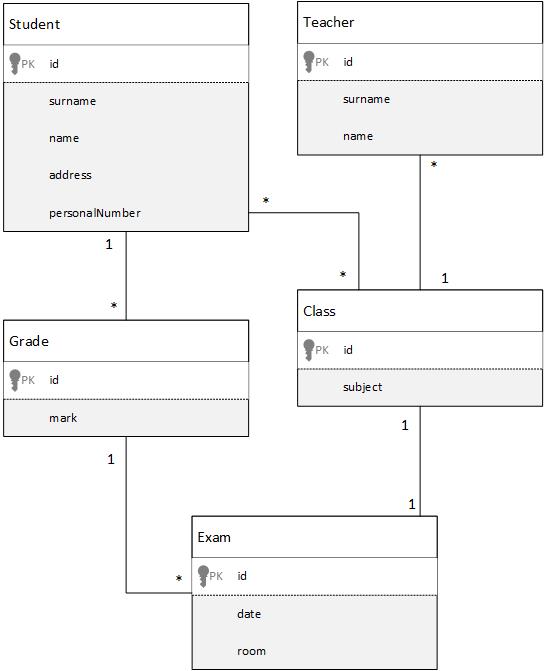
**5.1 Design Patterns Description**

The application uses the factory design pattern, which removes the need of calling constructors of the data transfer objects in logic classes. The dto factory simply return a new dto object, corresponding to which entity the method is called.

**5.2 UML Class Diagram**

**

6. Data Model

**

7. System Testing

The system testing will be done mainly using unit testing and validation testing. As for methods, data-flow will be the preferred method.

8. Bibliography

* <https://msdn.microsoft.com/en-us/library/ee658109.aspx>
* <https://www.tutorialspoint.com/uml>
* https://herbertograca.com/2017/08/03/layered-architecture/