2

Student: Rednic Ana

**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 project consists of designing and implementing a Java application for the management of students in the CS Department at TUCN. The application has two types of users which can select their profiles.

# Functional Requirements

The system should allow a normal user (student) to perform basic operations on his profile and to process class enrolment. The administrator user (teacher) should be able to perform CRUD operations on students’ information and to generate reports with activities of students.

# Non-functional Requirements

Performance: The system should have a minimal response time. Any operation should take less than 2 seconds.

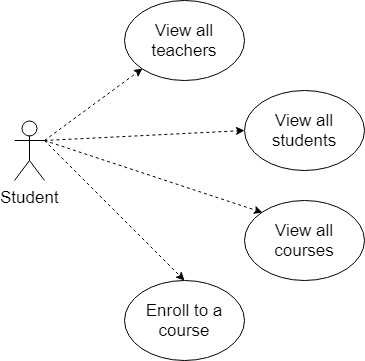
Testability: The application will be tested using JUnit tests.

Usability: The application will be user-friendly. It will have a GUI and all possible operations for a selected user will be displayed.

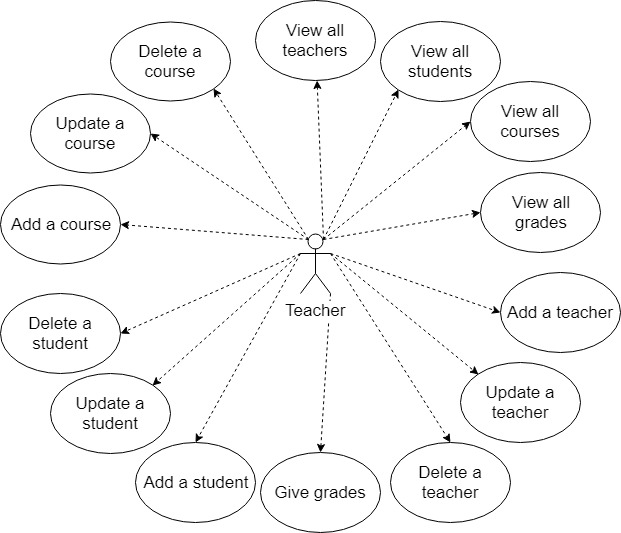
Security: The system will not be secured with log-in.

2. Use-Case Model

**Use case Diagram for Student**

****

**Use case Diagram for Teacher**

****

**Use case: Process Class Enrollment**

Primary actor: Student

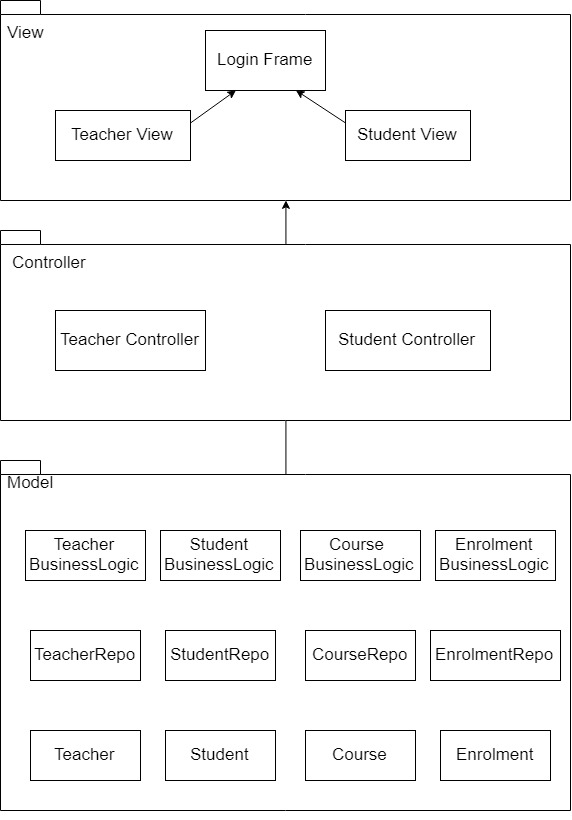
Main success scenario: The student enters the student profile in the application, a student interface shows up, he selects to see all the available courses to which he is not already enrolled, inputs the ID of a certain course, and enrolls in it. If successful, the application returns to the rest of the available courses.

Extensions: The student might not be able to enroll to some of the courses because of some constraints.

3. System Architectural Design

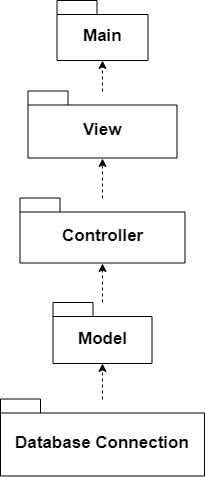
**3.1 Architectural Pattern Description**

The Model View Controller (MVC) architectural pattern is used to design this application. The Model is the collection of classes describing the business model and the data model. The View is the user interface part. It displays the data received from the Controller. Usually, Model and View interact through the Observer pattern. In our case, Model and View do not communicate. The Controller processes user's data through the Model and gives the results back to View. The Controller is a mediator between View and Model.

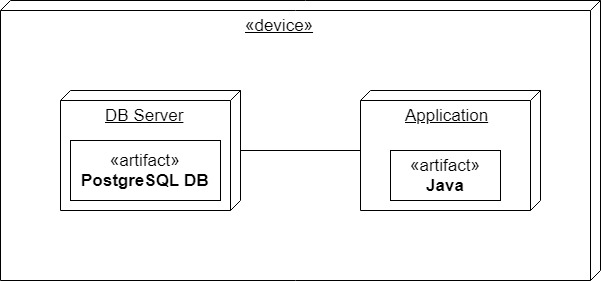


**3.2 Diagrams**

**Package Diagram**

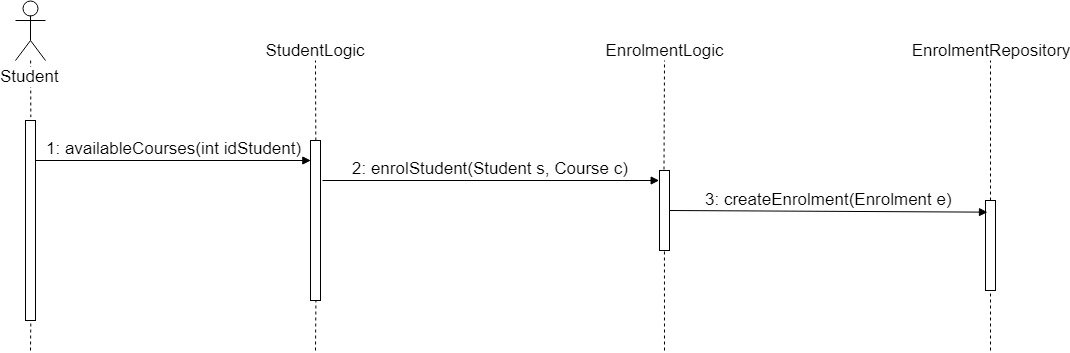
****

**Deployment Diagram**

****

4. UML Sequence Diagrams

**Sequence Diagram for Process Class Enrolment**

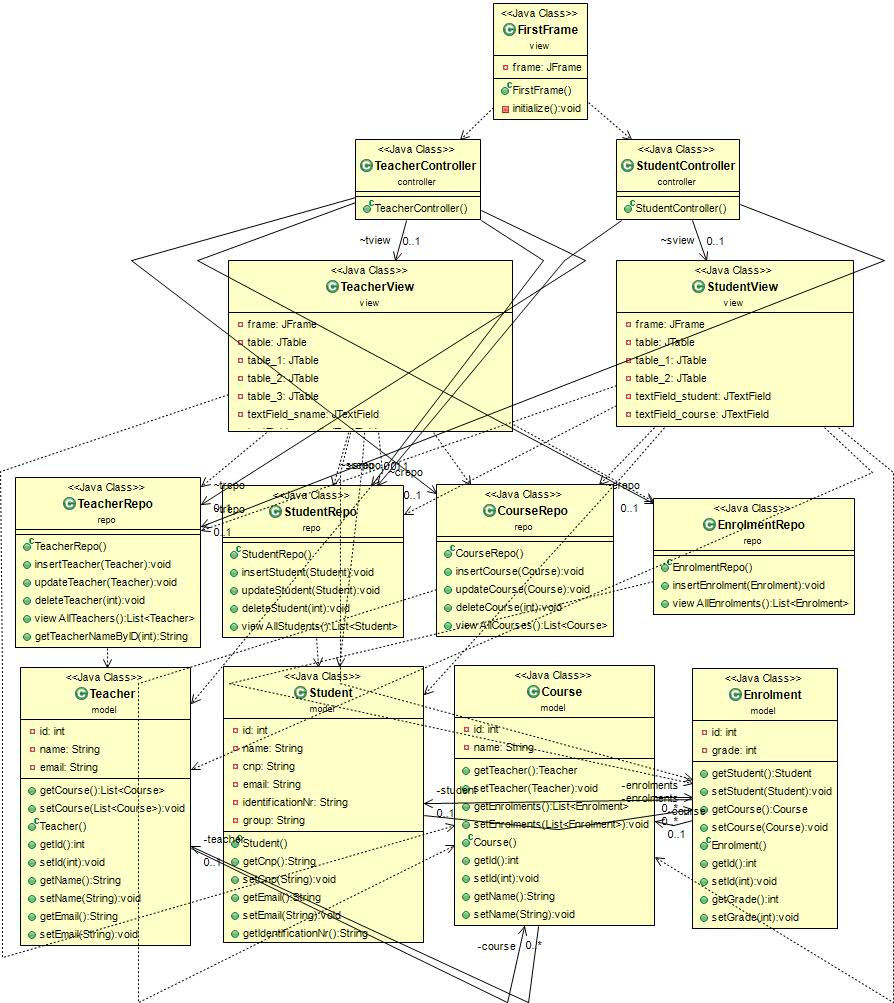
****

**5. Class Design**

**5.1 Design Patterns Description**

Factory design pattern will be used to export students’ grades in two formats (PDF and CSV). In Factory design pattern, an object can be created without exposing the creation logic to the client and refer to the newly created object using a common interface. A super class specifies all standard and generic behavior and then delegates the creation details to subclasses that are supplied by the client. Factory makes a design more customizable and only a little more complicated.

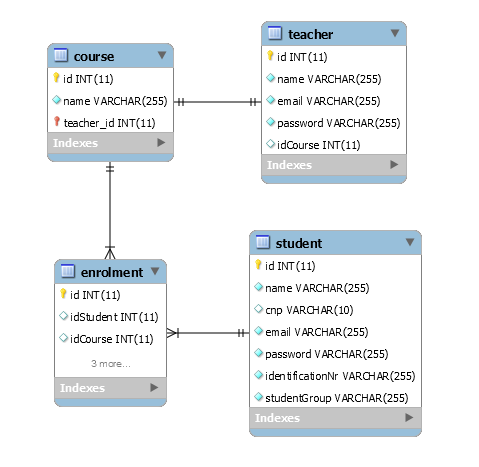
**5.2 UML Class Diagram**

****

**6. Data Model**

Hibernate is a high-performance Object/Relational persistence and query service, that takes care of the mapping from Java classes to database tables using XML files without writing any line of code. Hibernate also provides data query and retrieval facilities. It provides simple APIs for storing and retrieving Java objects directly to and from the database. It abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.

The JPA module contains a custom namespace that allows defining repository beans. It also contains certain features and element attributes that are special to JPA. Generally the JPA repositories can be set up using the repositories element.



7. System Testing

Unit testing will be performed on each component of the system, validating it. The focus falls on the tests that impact the behavior of the system. This will be done using Dataflow testing.

In the Integration testing part, individual units will be combined and tested as a group. This will look forward to expose defects in the interfaces and in the interaction between integrated components.

The System testing part will verify if the whole application meets the specified requirements.

Validation testing will be performed to decide whether or not the application is ready to be made available to the end-users. The last three stages will be done using Black Box testing.

8. Bibliography

<https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>

<https://msdn.microsoft.com/en-us/library/ee658109.aspx>

<https://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET>

<https://martinfowler.com/eaaCatalog/>

<http://softwaretestingfundamentals.com/>

<https://www.tutorialspoint.com/software_testing_dictionary/data_flow_testing.htm>

<https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad>

<https://www.tutorialspoint.com/design_pattern/factory_pattern.htm>

<https://sourcemaking.com/design_patterns/factory_method>

<https://www.tutorialspoint.com/hibernate/index.htm>

<https://www.tutorialspoint.com/hibernate/images/hibernate_position.jpg>

<https://docs.spring.io/spring-data/jpa/docs/1.5.0.RC1/reference/html/jpa.repositories.html>