Student: Grigor Sonia Eufrosina Maria

**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 4

3. System Architectural Design 5

4. UML Sequence Diagrams 9

5. Class Design 10

6. Data Model 12

7. System Testing 13

8. Bibliography 14

1. Requirements Analysis

# Assignment Specification

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

# Functional Requirements

The regular user can perform the following operations:

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

The administrator user can perform the following operations:

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

# Non-functional Requirements

Non-functional requirements specify the system’s quality characteristics’ or quality attributes. Some of the non-functional requirements are accessibility witch refers to help text to be in an international language, accuracy (date of birth to be in the past), performance (system responses should be no more than one second), efficiency (the system restart cycle must execute in less than one minute), safety(not causing harm, injury or damage).

2. Use-Case Model

Use case title: Student interaction with this system in order to log in.

Level: User-goal level which means something the actor is trying to get done. The properties of a user-goal level use-case model is a high-level interaction between the actor and the system, identifies user and system roles.

Primary actor: Student is the primary actor in this use case model diagram.

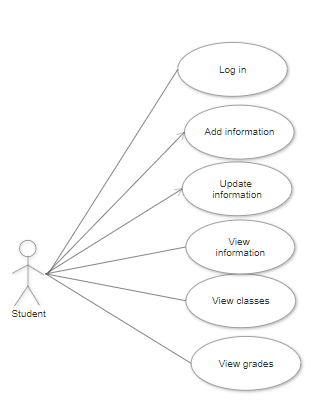
Main success scenario:

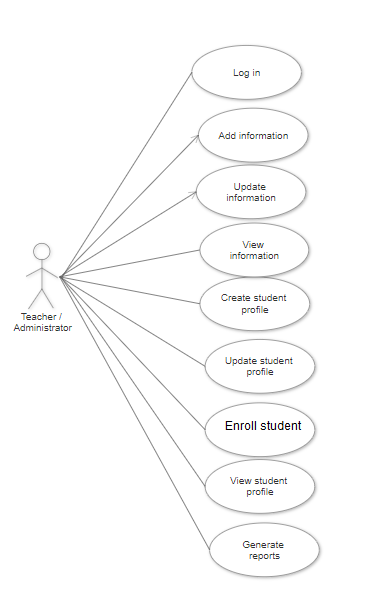
1. The student connects to the system.
2. The user enters his/her username and password.
3. The system validates the username and password.
4. The system determines the user’s role.
5. The system displays a list of actions the student can perform based on the user’s role.

Extensions:

3a. The system determines that the username does not match a user-name for any account.

1. The system displays an error message.





3. System Architectural Design

An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design pattern but have a broader scope.

In this project is used a model-view-controller(MVC) architectural pattern combine with layered

architecture.

**3.1 Architectural Pattern Description**

Layered pattern can be used to structure programs that can be decomposed into group of sub-tasks, each of which is a particular level of abstraction. Each layer provides services to the next high layer. The most commonly found four layers of a general application systems are as follows:

* Presentation Layer – User Interface Layer
* Business Logic Layer – Domain Layer
* Data Access Layer – Persistence Layer

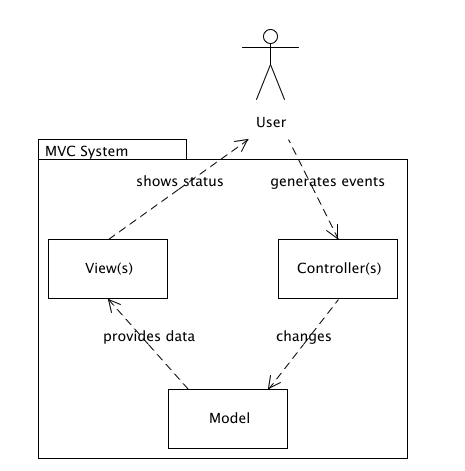
Model-View-Controller architectural pattern follows an elementary idea – we must separate the responsibilities in any application on the following basis:

* **Model:** Handles data and business logic. Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* **View:** Presents the data to the user whenever asked for. View represents the visualization of the data that model contains.
* **Controller:** Entertains user requests and fetch necessary resources. Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

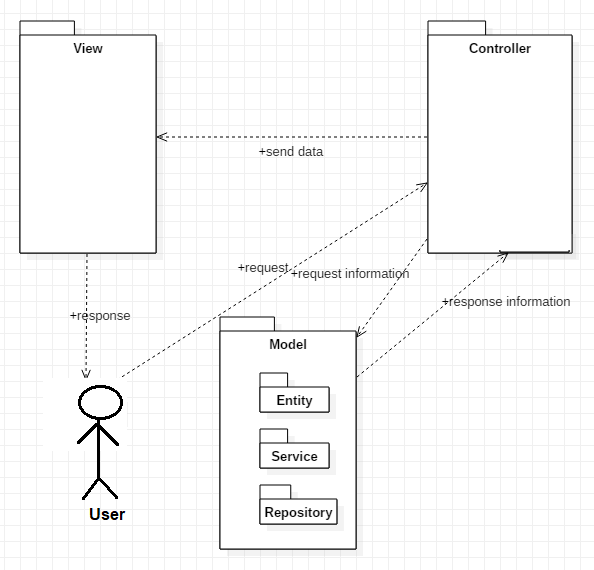
Each of the components has a demarcated set of tasks which ensures smooth functioning of the entire application along with complete modularity.

**3.2 Diagrams**

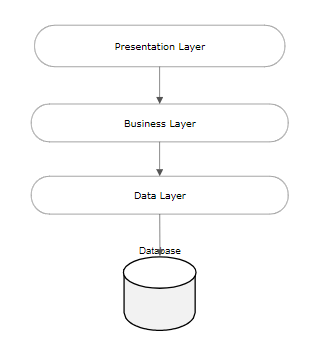
The main architectural pattern user in this project is MVC. MVC proposes three types of objects in an application, the Model, Views and Controllers. These objects are separated by abstract boundaries which makes MVC more of a paradigm rather than an actual pattern since the communication with each other across those boundaries is not further specified. How the objects inside MVC communicate differs not only by the type of application you are describing (GUI, web) but also by which part of the application you are currently looking at (frontend, backend).



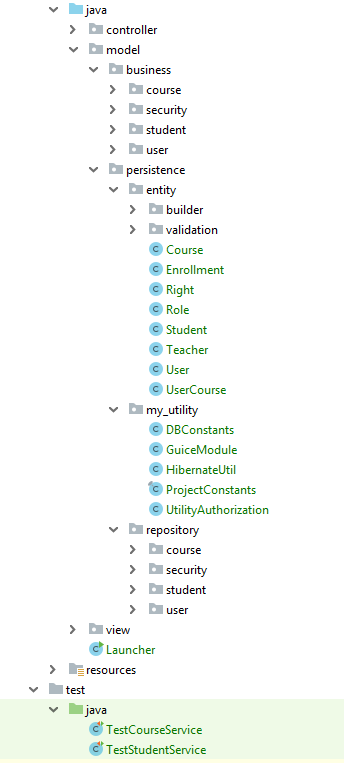
Beside Model-View-Controller, this project use also Layered Architecture. The layered pattern is included in Model part of MVC architecture.



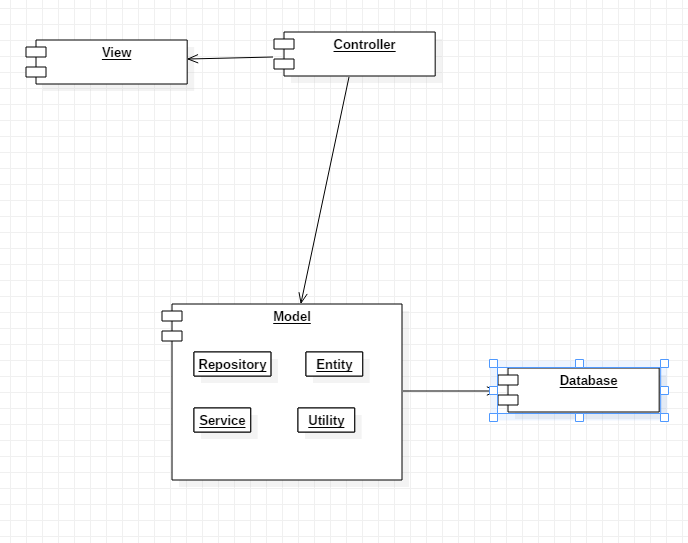
The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic).



The solution has the following structure:

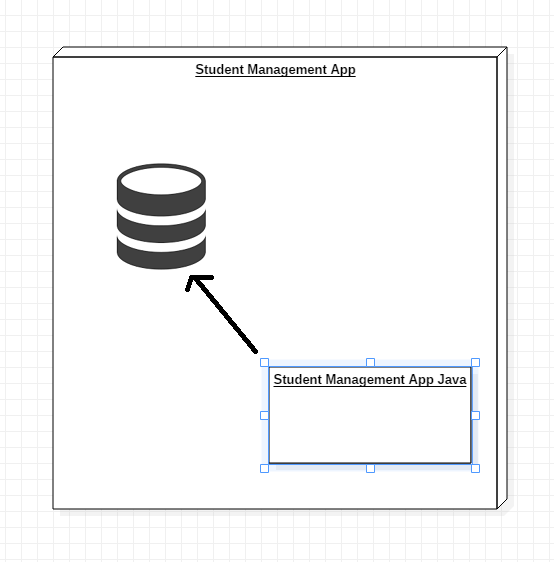


A component is a code module. Component diagram are physically analogs of class diagram.



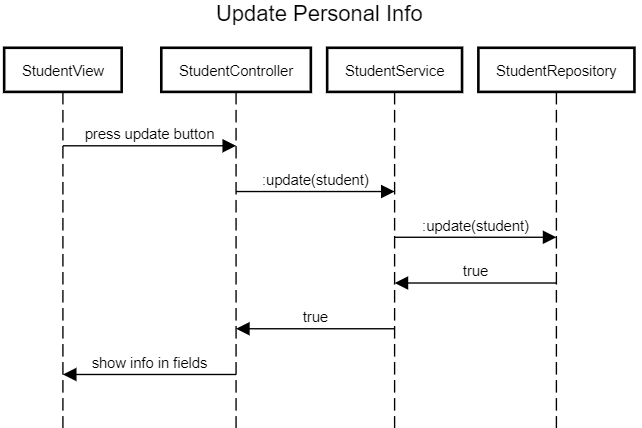
Deployment diagrams show their physical configurations of software and hardware.

In this project is used PostgreSQL as a database server.



4. UML Sequence Diagrams

UML Sequence Diagram for login into system. When user enter his credentials into mandatory and proper field and press the login button, he will be redirect to homepage where he can see all his enrollment courses. In order to do this, he need to enter his details (username and password). After that, the information is transferred to student repository. Student Repository is responsible to send information through data access layer to database. If there is a match, database response positively and the user will be redirect to homepage. For homepage, there is necessary to populate the screen with all the courses attended by a student.



5. Class Design

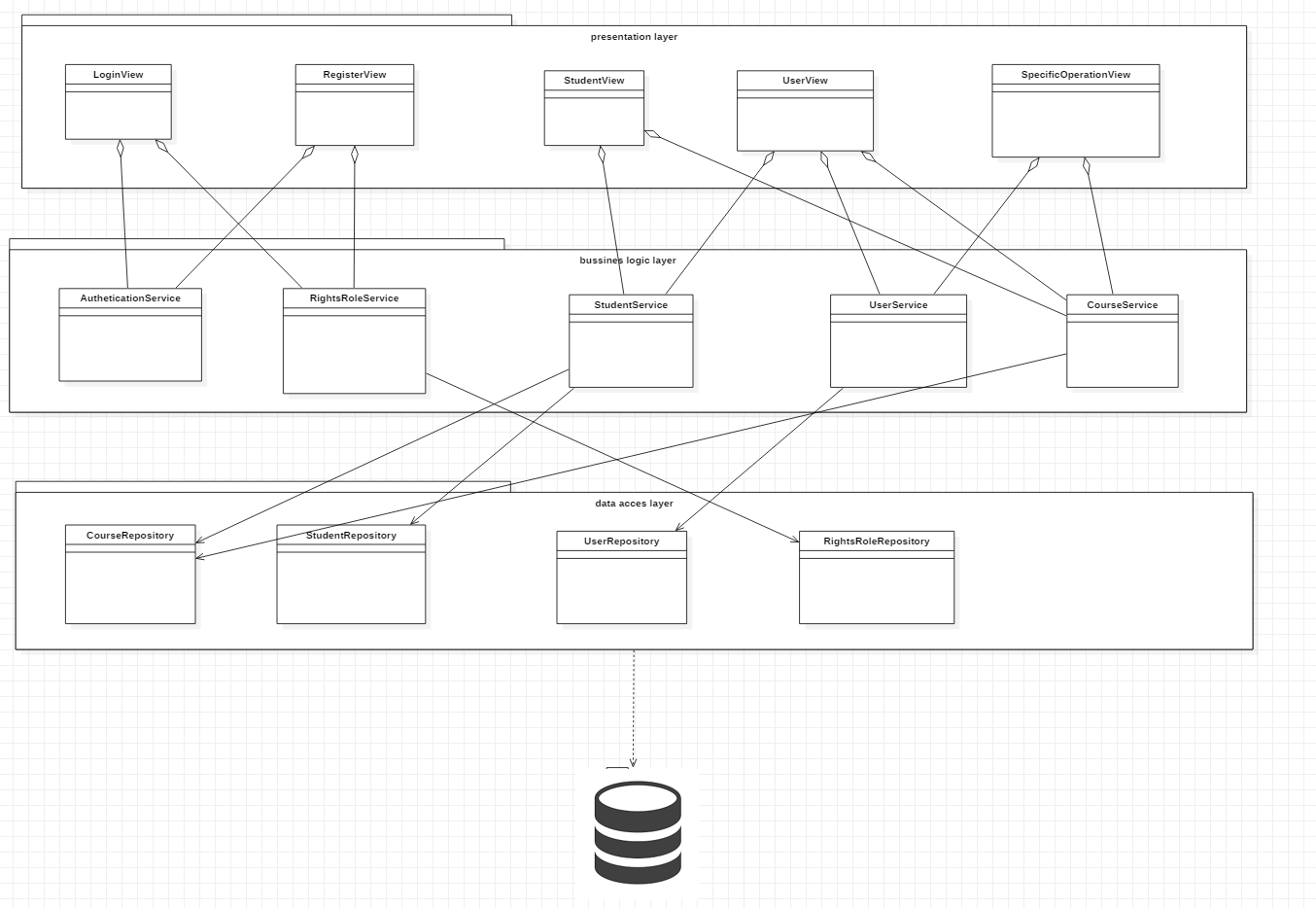
**5.1 Design Patterns Description**

Builderis a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.For example, let’s think about how to create a House object. To build a simple house, you need to construct four walls and a floor, install a door, fit a pair of windows, and build a roof. But what if you want a bigger, brighter house, with a backyard and other goodies (like a heating system, plumbing, and electrical wiring)?

**5.2 UML Class Diagram**

In my project builder is used in model package in order to build all the objects like Students, Teachers, etc. The implementation of this pattern is easy. You need to create a new object using a constructor default, and implements all the setters for every single attribute in the class (setName, setCardNo, setCourses), and the last step is to implement a method build() witch return your student.

In this project Singleton is used in order to be sure that there is a single instance of the database context.



6. Data Model

Data Model contains nine main tables:

user- with all the information: name, username, password, email, CNP,

student- with information related to students: cardNo, group

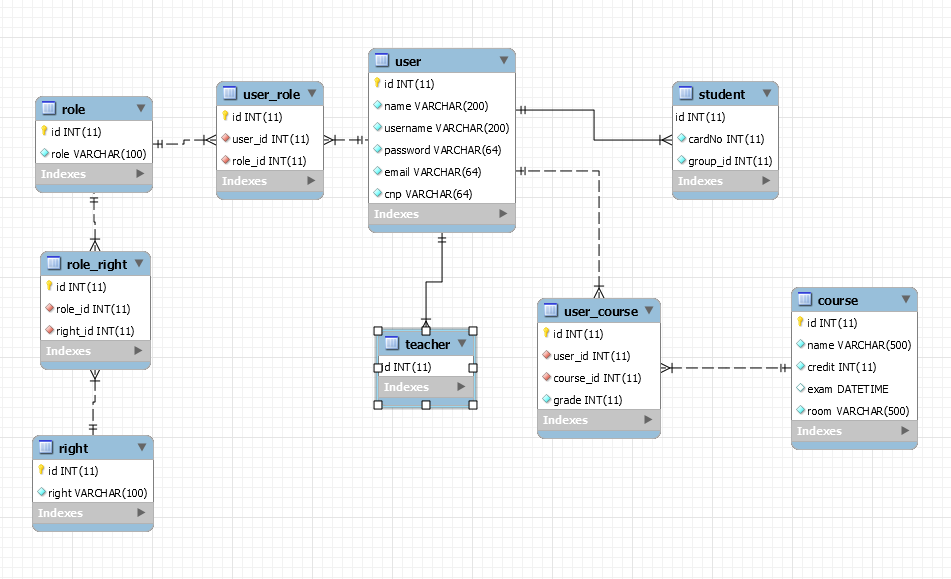
teacher- it is exactly like a user

course- name, credit, exam date, room

user\_course- is the table to solve many-to-many relation between user and course because a user can have multiple courses and a course is attend by multiple users. Even teachers can attend courses for study or in order to teach the students. Also this table contains grade field.

There are four more tables in order to determine the roles and rights for each user.

Each table contains an id witch is primary key auto increment.



7. System Testing

System Testing is the testing of a complete and fully integrated software product.

Unit testing -testing performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.

Unit tests on the service classes, which will test the correctness of the implementation of these classes by testing separately the methods that these classes exposed. This can be considered a type of white box testing.

Integration test on a whole flow of the application, which can be considered a type of black box testing because we are not interested about the results of any particular component, but on the whole application. It does not test any specific class, but the way the classes are work as a whole. This means that we provide the application with a specific input and we test the results provided by the application.

Unit tests with Mokito are based on an understanding of unit testing with the JUnit framework. A *mock object* is a dummy implementation for an interface or a class in which you define the output of certain method calls. Mock objects are configured to perform a certain behavior during a test. They typically record the interaction with the system and tests can validate that.

If you use Mockito in tests you typically:

* Mock away external dependencies and insert the mocks into the code under test
* Execute the code under test
* Validate that the code executed correctly

Mocks can return different values depending on arguments passed into a method. The *when(…).thenReturn(…)* method chain is used to specify a return value for a method call with pre-defined parameters. You also can use methods like *anyString()* or *anyInt()* to define that dependent on the input type a certain value should be returned.

8. Bibliography

1. <https://searchsoftwarequality.techtarget.com/tip/Using-a-nonfunctional-requirements-template-plus-examples>
2. <https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>
3. <http://kti.tugraz.at/staff/rkern/courses/sa/slides_ca.pdf>
4. <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>
5. [https://github.com/utcn-cs-software-design-tudorv-2019/LabInfo/blob/master/resources/UML%20-%20embarcadero.pdf](https://github.com/utcn-cs-software-design-tudorv-2019/LabInfo/blob/master/resources/UML - embarcadero.pdf)
6. <https://refactoring.guru/design-patterns>
7. <https://www.uml-diagrams.org/package-diagrams-overview.html>
8. <https://www.guru99.com/system-testing.html>
9. <https://www.vogella.com/tutorials/Mockito/article.html>
10. <https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66>