Student: Grigor Sonia Eufrosina Maria
Group: 30233

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 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.

1.2 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.

1.3 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).

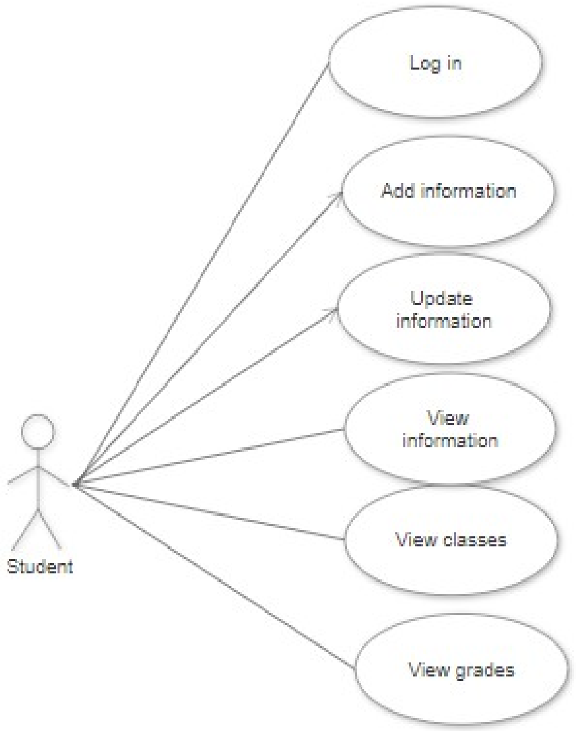
2. Use-Case Model

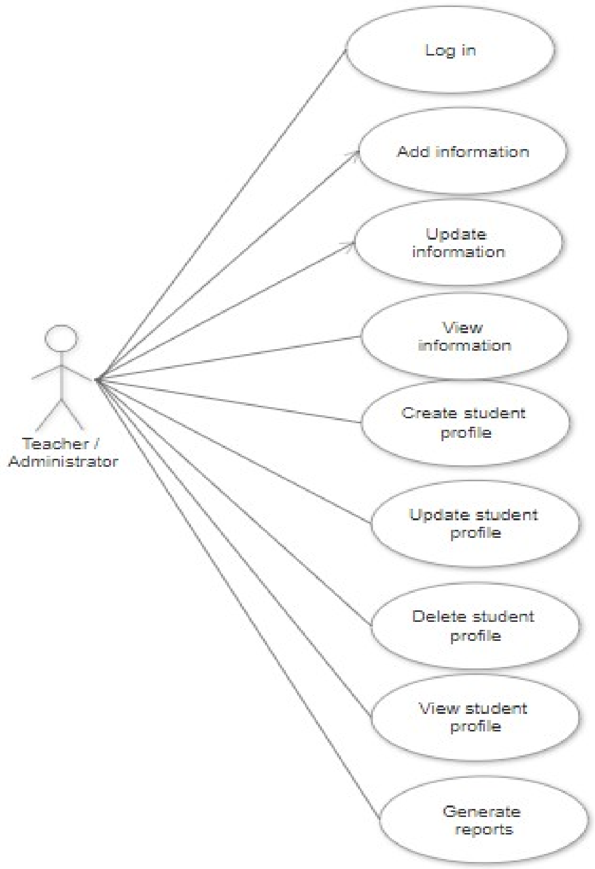
Use case title: Student interaction with this system in order to log in.
Level: User-goal level which means something the actor is trying to get done. The properties of a user-goal level use-case model is a high-level interaction between the actor and the system, identifies user and system roles.
Primary 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.




Log in
Add information
Update information
View information
Create student profile
Update student profile
Delete student profile
View student profile
Generate reports
Teacher / Administrator

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.

3.1 Architectural Pattern Description

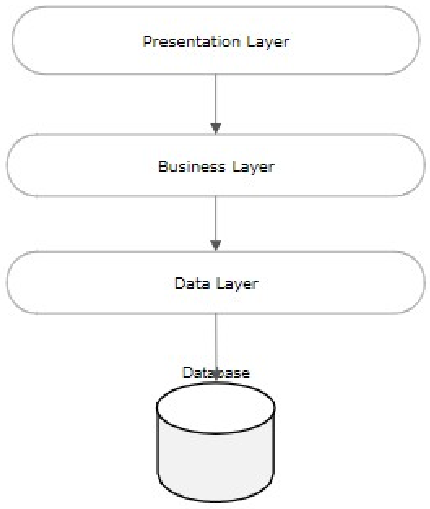
Layered pattern can be used to structure programs that can be decomposed into group of sub-tasks, each of which is a particular level of abstraction. Each layer provides services to the next high layer. The most commonly found four layers of a general application systems are as follows:

- Presentation Layer – User Interface Layer
- Application Layer – Service Layer
- Business Logic Layer – Domain Layer
- Data Access Layer – Persistence Layer

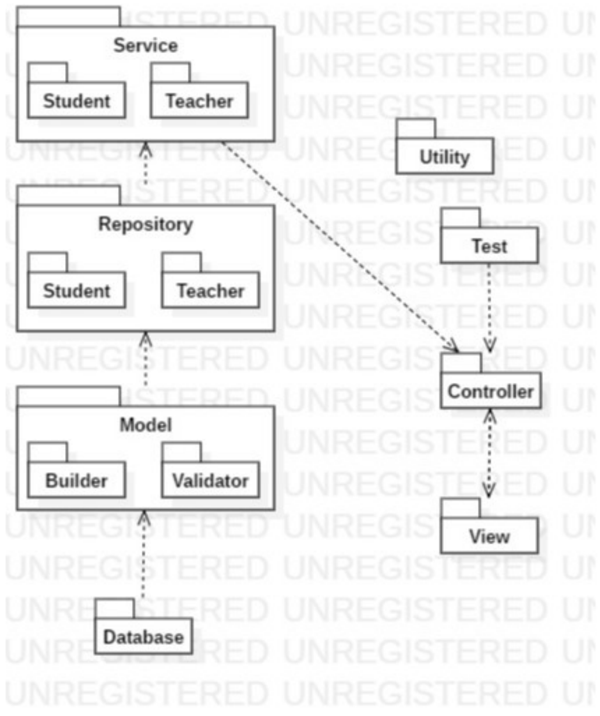
3.2 Diagrams

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). For example, a presentation layer would be responsible for handling all user interface like students and teachers or administrators, a business layer has the responsibility to call method in order to respond to requests. In the same way, a data layer would be responsible with access the database in order to get all the information from database(student’s names, enrollments) to add students, to enroll students to different courses. Basically everything related to database is implemented here.

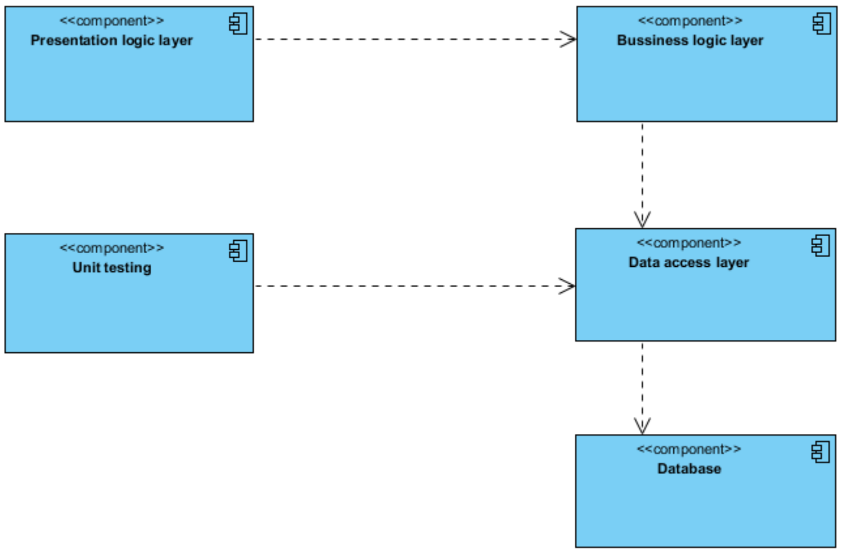
Each layer of the architecture has a specific role and responsibility, but also each layer of the architecture forms an abstraction around the work. The presentation layer doesn’t need to know how to get students from database. This is the responsibility of data layer. The presentation layer only need to display that information on a screen in a particular way.



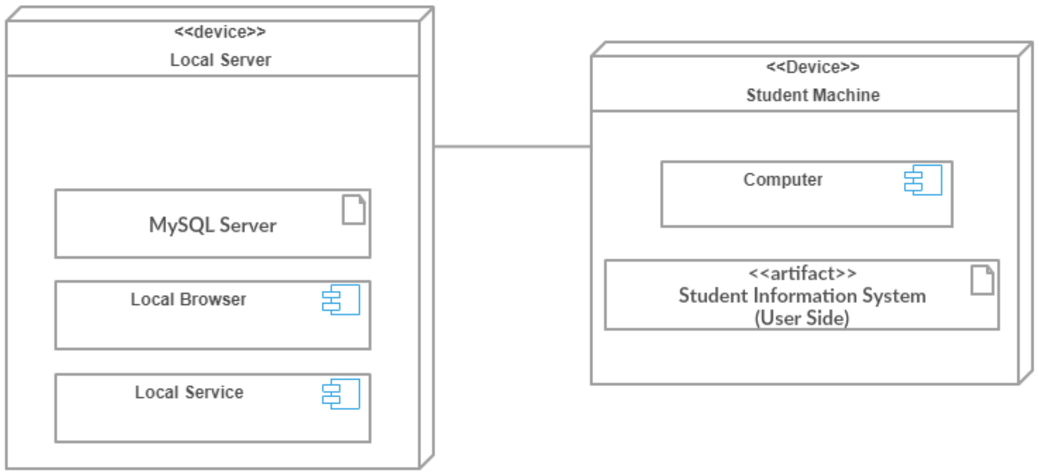
The solution is separated in seven packages: Model, Repository, Service, Presentation and Utility. Each of them has a specific role and it has a specific layer associated. For example: presentation package is associated with presentation layer and the main purpose of presentation package is to keep in a single place and organized as much as possible all the classes related to graphical user interface.



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

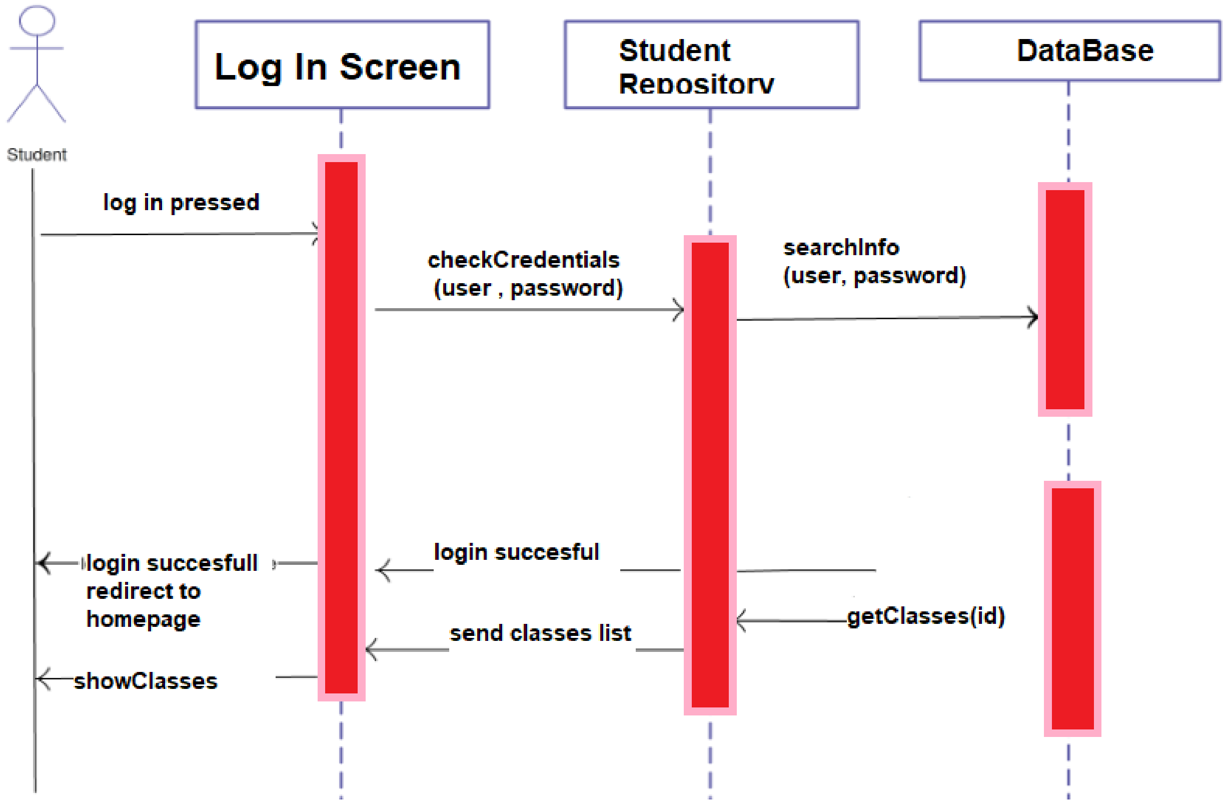


Deployment diagrams show their physical configurations of software and hardware.



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

Builder is 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)?

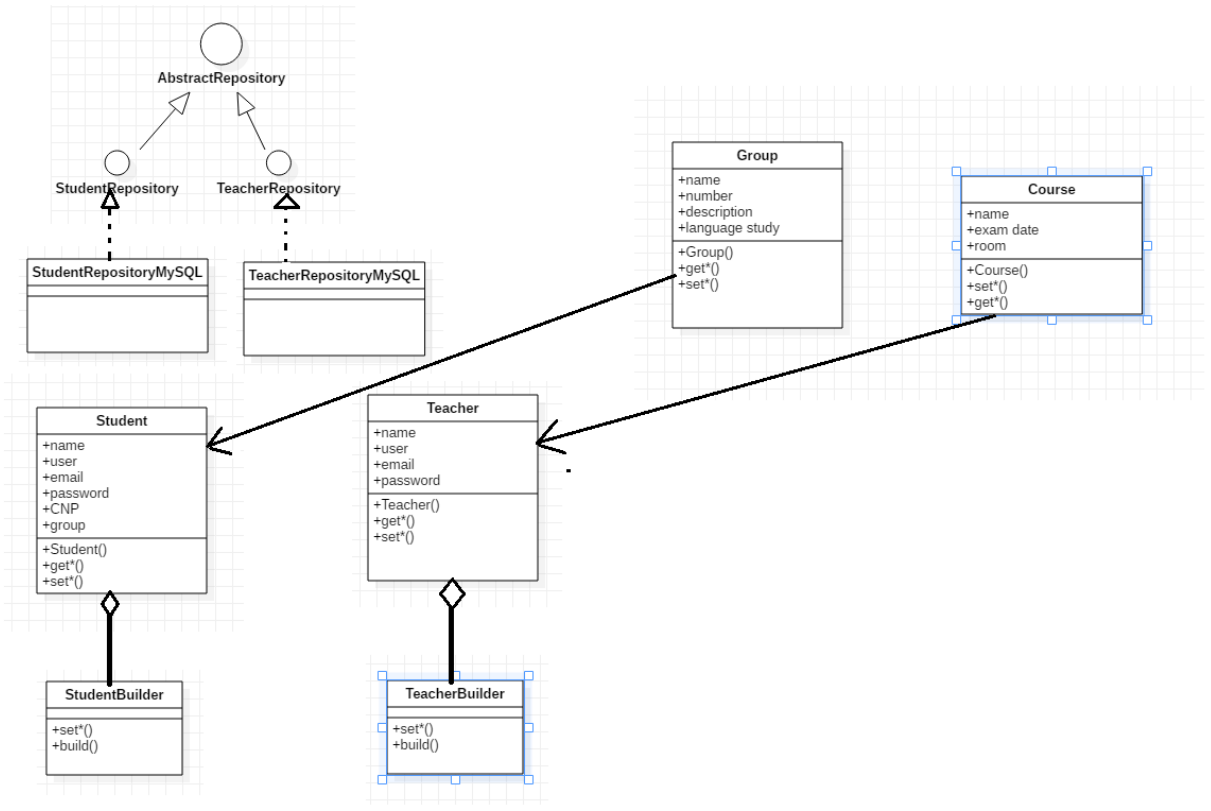
Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance. Singleton pattern solve violating of the Single Responsibility Principle to be sure that a class has just a single instance and provide a global access point to the instance.

Factory Method is a creational design pattern that provides an interface for creating objects in a super-class, but allows sub-classes to alter the type of objects that will be created. The Factory Method pattern suggests that you replace direct object construction calls (using the new operator) with calls to a special factory method. Don't worry: the objects are still created via the new operator, but it's being called from within the factory method. Objects returned by a factory method are often referred to as "products."

5.2 UML Class Diagram

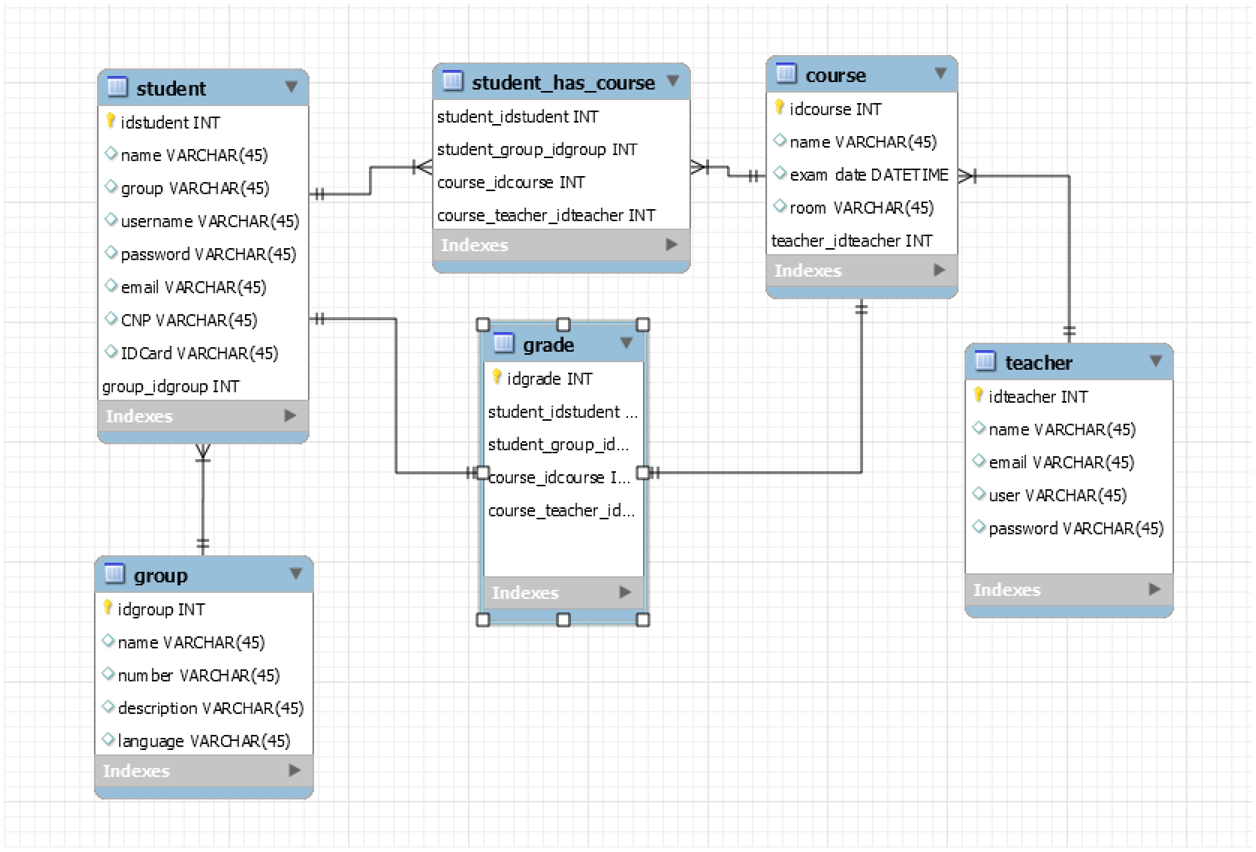
In my project builder is used in model package in order to build all the objects like Students, Teachers, etc. The implementation of this pattern is easy. You need to create a new object using a constructor default, and implements all the setters for every single attribute in the class (setName, setCardNo,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


AbstractRepository
StudentRepository
TeacherRepository
StudentRepositoryMySQL
TeacherRepositoryMySQL
Group
+name
+number
+description
+language study
+Group()
+get*()
+set*()
Course
+name
+exam date
+room
+Course()
+set*()
+get*()
Student
+name
+user
+email
+password
+CNP
+group
+Student()
+get*()
+set*()
Teacher
+name
+user
+email
+password
+Teacher()
+get*()
+set*()
StudentBuilder
+set*()
+build()
TeacherBuilder
+set*()
+build()

6. Data Model

Data Model contains 6 main tables:
Student- with all the information: name, group, username, password, email, CNP,
Grade- course, grade, idStudent(as foreign key),
Teacher- name, email, user, password,
Course- name, exam date, room
Group- name, number, description, language study
StudentCourse- is the table to solve many-to-many relation between student and course because a student can have multiple courses and a course is attend by multiple students.
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. Two Category of Software Testing

- Black Box Testing
- White Box Testing

System test falls under the black box testing category of software testing.
White box testing is the testing of the internal workings or code of a software application. In contrast, black box or System Testing is the opposite. System test involves the external workings of the software from the user's perspective.

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.

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
6. https://refactoring.guru/design-patterns
7. https://www.uml-diagrams.org/package-diagrams-overview.html
8. https://www.guru99.com/system-testing.html