Assignment 1

Student:Takacs Diana-Ingrid

**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 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

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

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, enrolments, grades).

- Process class enrolment (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.

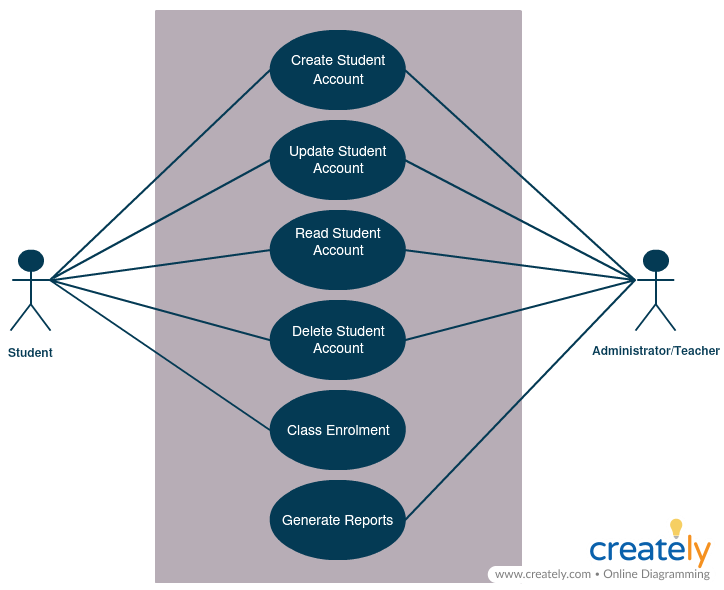
# Functional Requirements

* Define 2 types of users: Student and Admin/Teacher
* The administrator can perform CRUD operations on students information
* Process class enrolment
* Authentication using username and password

# Non-functional Requirements

* Validate all the input data against invalid data before submitting the data and saving it to the data base
* Data will be stored in a relational database
* Use layered architectural pattern to organize the application
* Documentation

2. Use-Case Model

**

Use case: Generate reports for a particular period containing activities performed by a student

Level: user goal level

Primary actor: Admin/Teacher

Main success scenario:

* Profesorul sau adminul trebuie in prima instanta sa se autentifice pe platforma cu numele de cont si parola.
* Va efectua o afisare a tuturor studentilor dintr-o grupa/serie/an/specializare si va selecta un student dorit.
* Adminul va genera un raport cu perioada dorita pentru a vizualiza activitatea academica a studentului selectat.

Extensions:

Scenariu efectuat cu success: adminul/profesorul se logeaza si gaseste studentul dorit pentru ai genera rapoartele. Scenariu efectuat cu esec: adminul/proseforul introduce date incorecte de autentificare, studentul dorit nu este inrolat, studentul dorit este inrolat cu nume gresit.

3. System Architectural Design

**3.1 Architectural Pattern Description**

Arhitectura pe trei niveluri este un model de arhitectură software în care interfața utilizator (prezentare), logica proceselor funcționale (application logic), stocarea datelor și accesul la date sunt dezvoltate și menținute ca module independente.

* Presentation layer - functionalitate orientată spre utilizator, responsabilă de gestionarea interacțiunii utilizatorilor cu sistemul
* Application layer - functionalitatea centrală a sistemului și încorporează logica de afaceri relevantă. În general, acesta este alcătuit din componente, dintre care unele pot expune interfețele de serviciu pe care ceilalți apelanți le pot utiliza.
* Data layer - oferă acces la datele găzduite în limitele sistemului și la datele expuse de alte sisteme din rețea; poate accesat prin servicii. Stratul de date expune interfețele generice pe care le pot consuma componentele din stratul de afacere.

Model-View-Controller este un model arhitectural utilizat în mod obișnuit pentru dezvoltarea interfețelor utilizator care împarte o aplicație în trei părți interconectate. Aceasta se face pentru a separa reprezentările interne ale informațiilor de modalitățile în care informațiile sunt prezentate și acceptate de la utilizator. Modelul de design MVC decuplează aceste componente majore, permițând reutilizarea eficientă a codurilor și dezvoltarea paralelă.

* Model - componenta centrală a arhitecturii. Este structura de date dinamică a aplicației, independentă de interfața cu utilizatorul. Administrează direct datele, logica și regulile aplicației.
* View - orice reprezentare a informațiilor. Sunt posibile vizionări multiple ale acelorași informații, cum ar fi o diagramă de stare pentru gestionare și o vizualizare tabelă pentru contabili.
* Controller - Acceptă intrarea și o convertește în comenzi pentru model sau vizualizare.

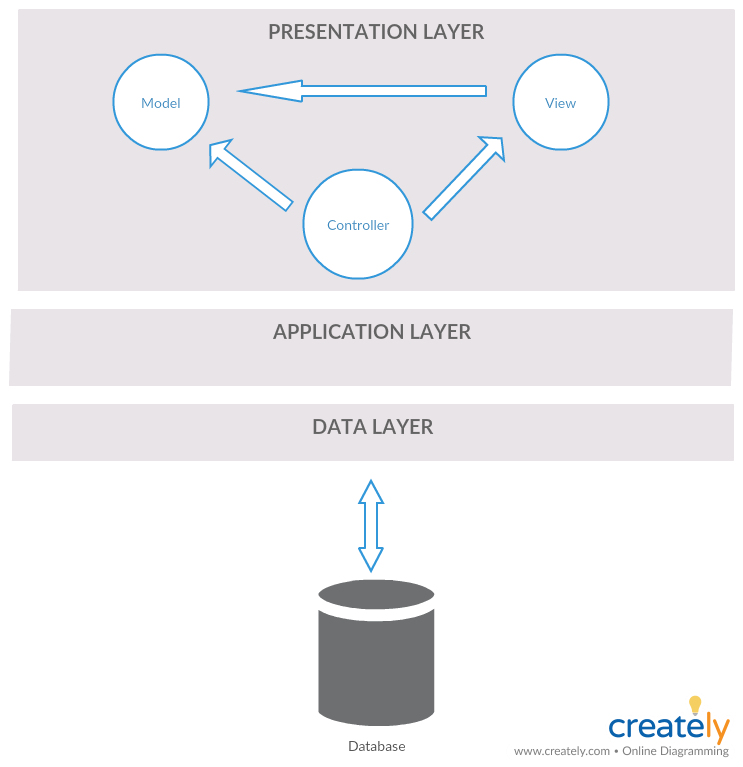
Modelul este responsabil pentru gestionarea datelor aplicației. Acesta primește intrarea utilizatorului de la controler.

View înseamnă prezentarea modelului într-un anumit format.

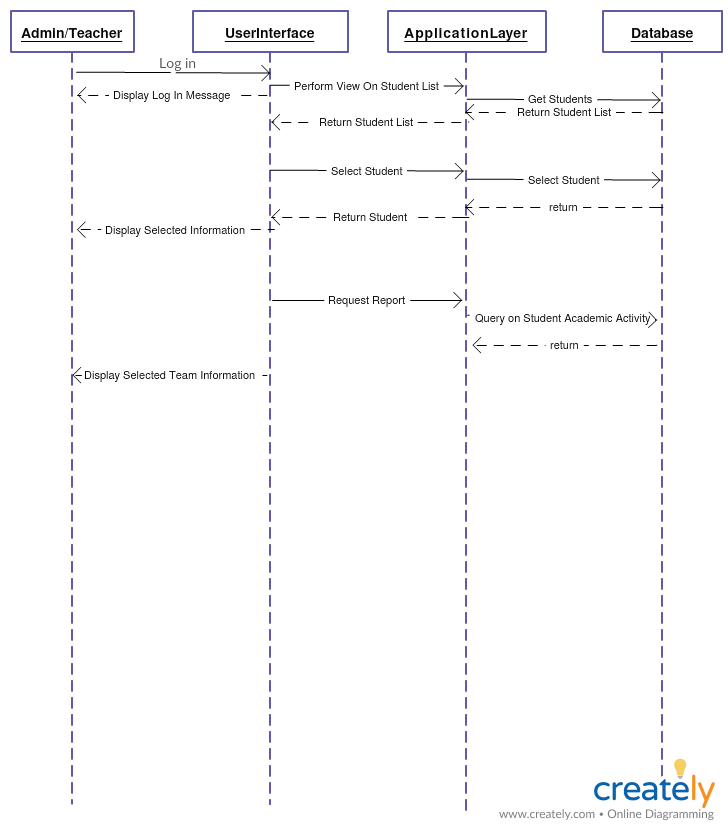
Controllerul răspunde la intrarea utilizatorului și efectuează interacțiuni cu obiectele modelului de date. Controlorul primește intrarea, validează opțional și apoi trece intrarea în model.

Ca și în cazul altor modele software, MVC exprimă "miezul soluției" unei probleme, permițând în același timp adaptarea acesteia pentru fiecare sistem.

**3.2 Diagrams**

**

4. UML Sequence Diagrams



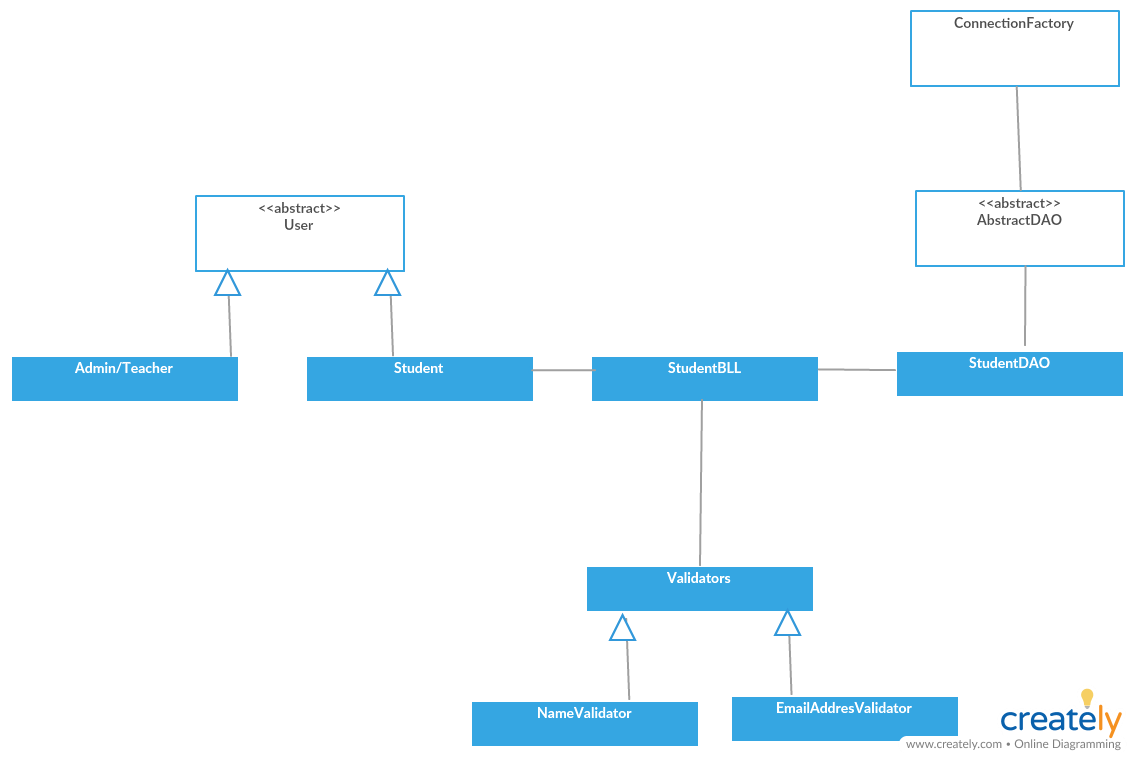
5. Class Design

**5.1 Design Patterns Description**

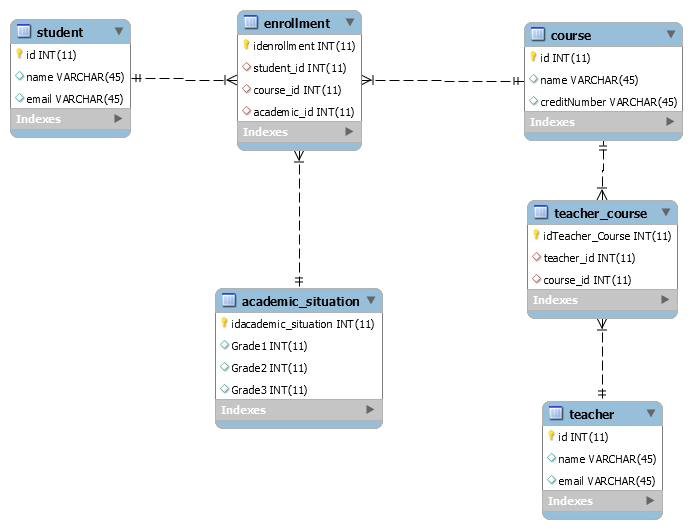
Un obiect de acces la date (DAO) este un obiect care oferă o interfață abstractă pentru un tip de bază de date sau un alt mecanism de persistență. Prin maparea apelurilor aplicației către stratul de persistență, DAO oferă anumite operații de date specifice fără a expune detalii ale bazei de date. Această izolare susține principiul responsabilității unice. Se separă ce date de acces necesită aplicația, în ceea ce privește obiectele specifice domeniului și tipurile de date (interfața publică a DAO), de la modul în care aceste nevoi pot fi satisfăcute cu un anumit DBMS, schema bazei de date etc. (implementarea DAO).

Deși acest model de proiectare este aplicabil la majoritatea limbajelor de programare, majoritatea tipurilor de software cu nevoi de persistență și cele mai multe tipuri de baze de date este asociat în mod tradițional cu aplicațiile Java EE și cu baze de date relaționale (accesate prin JDBC API datorită originii sale în Sun Orientările privind cele mai bune practici ale microsistemelor [1] "Modele J2EE de bază" pentru platforma respectivă).

**5.2 UML Class Diagram**



6. Data Model



7. Bibliography

<https://creately.com/diagram/example/hszw2rmn2/College+Enrollment+System>

<https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

<https://en.wikipedia.org/wiki/Data_access_object>

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

<https://reqtest.com/requirements-blog/understanding-the-difference-between-functional-and-non-functional-requirements/>

<https://stackoverflow.com/questions/16475979/what-is-the-difference-between-functional-and-non-functional-requirement>