Managmentul studentilor din cadrul UTCN

Document de Analiza si Design

Student:Dura Dragos Catalin

**Grupa:30233**

Cuprins

[**1. Analiza cerintelor** 2](#_Toc3985217)

[**1.1 Specificarea cerintei** 2](#_Toc3985218)

[**1.2** **Cerinte functionale** 2](#_Toc3985219)

[**1.3 Cerinte non-functionale** 2](#_Toc3985220)

[**2. Model use case** 3](#_Toc3985221)

[**3. Designul architectural al sistemului** 3](#_Toc3985222)

[**3.1 Descrierea pattern-ului architectural** 4](#_Toc3985223)

[**3.2 Diagrame** 4](#_Toc3985224)

[**4. Diagrama de secventa** 6](#_Toc3985225)

[**5. Class Design** 6](#_Toc3985226)

[**5.1 Design Pattern** 6](#_Toc3985227)

[**5.2 Diagrama de clase** 7](#_Toc3985228)

[**6. Data Model** 8](#_Toc3985229)

[**7. Testarea Sistemului** 8](#_Toc3985230)

[**8.Bibliografie** 8](#_Toc3985231)

# **1. Analiza cerintelor**

## **1.1 Specificarea cerintei**

Sa se proiecteze o aplicatie Java care se ocupa de managmentul studentilor din departamentul CS al UTCN.

## **1.2** **Cerinte functionale**

Aplicatia ar trebui sa aiba 2 tip de useri: student si professor fiecare trebuind sa aiba introduca username-ul si parola pentru a putea folosi aplicatia.

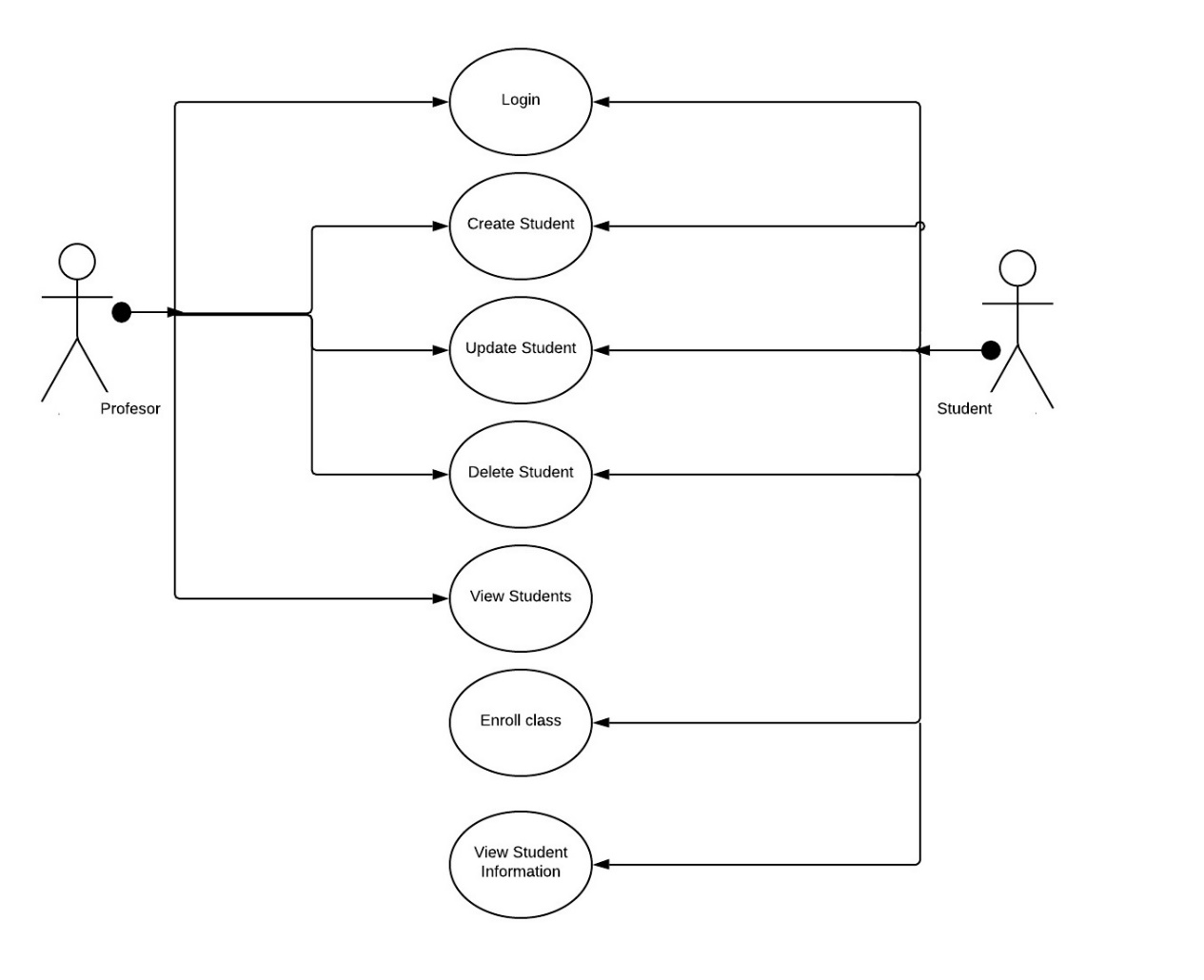
Profesorul poate face urmatoarele operatii: CRUD pe informatia despre student, precum si sa genereze rapoarte continand activitatile pe care un student le-a avut intr-o anumita perioada.

Studenti pot sa creeze, stearga, updateze si sa vizualizeze profilul lor. Studentii pot de asemenea sa se inscrie in unele clase.

## **1.3 Cerinte non-functionale**

Datele aplicatiei trebuie stocate intr-o baza de date relationala( MySql/ MsSql) si trebuie folosit patternul Layered Architecture pentru organizarea aplicatiei. Data Acces Layer ar trebui creat folosind SQL statements. De asemenea toate intrarile aplicatiei trebuie validate inainte de a fi permisa trimiterea lor spre baza de date.

# **2. Model use case**



**Exemplu Use-Case:**

Use-Case: Login

Main Actor: Student

Main success scenario: 1. Studentul introduce username-ul

2. Studentul introduce parola

3. Studentul apasa butonul de login

4. Aplicatia permite conectarea studentului, acesta avand un cont creat

Extension: 4. Aplicatia nu permite conectarea studentului

* Studentul este informat de catre aplicatie ca nu are un cont creat

# **3. Designul architectural al sistemului**

## **3.1 Descrierea pattern-ului architectural**

Ca si pattern architectural in cadrul acestei aplicatii am folosit Layered Architecture pattern. Componentele in cadrul acestui pattern sunt organizate in layere orizontale, fiecare dintre acestea avand un rol specific in aplicatie. Layerele ne ajuta sa facem diferenta dintre diferite sarcini pe care le executa componentele, facilitand crearea unui design care sa suporte reutilizarea componentelor.

In cazul acestei aplicatii avem urmatoarele Layere:

* Data Acces Layer
* Business Layer
* Presentation Layer

Comunicare dintre layere trebuie sa fie de sus in jos, adica layerele care sunt situate mai jos nu pot comunica cu layerele de desupra lor. De exemplu Data Acces nu poate apela o metoda din cadrul Business Layer.

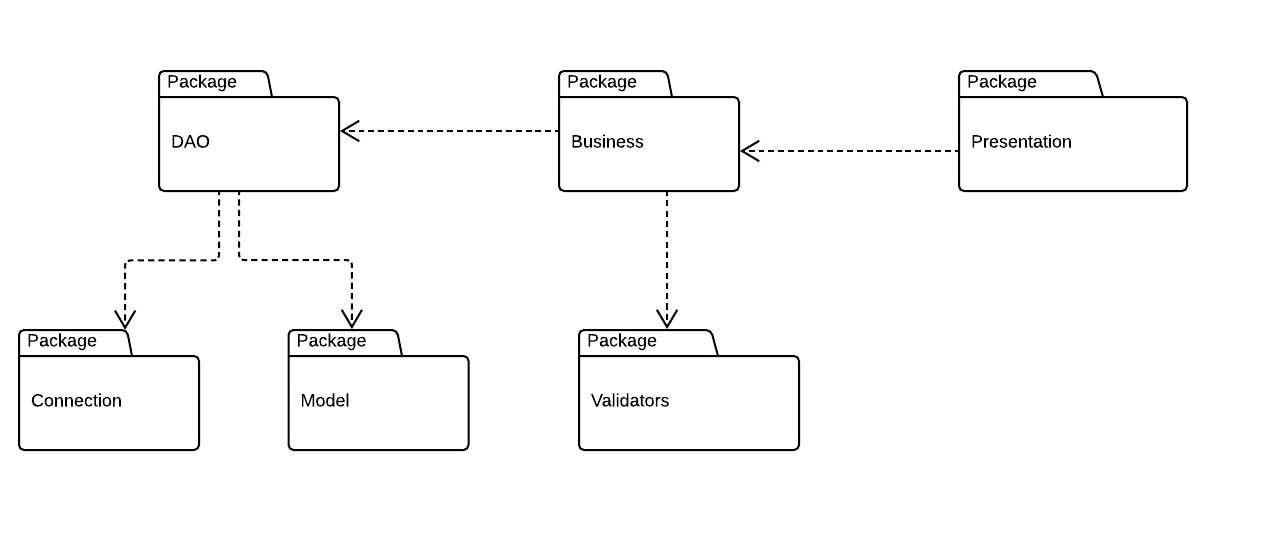
In Data Acces layer avem operatiile asupra baze de date: conectarea, executare de SQL statements.

In Business layer avem operatiile care implementeaza functionalitatea aplicatiei.

In Presentation layer avem interfata cu utilizatorul.

## **3.2 Diagrame**

Diagrama de pachete:



Din aceasta diagram de pachete se poate observa structura pe layere a aplicatiei. Pachetele DAO, Connecton si Model reprezinta DataAcces Layer, pachetele Business si Validators reprezina Business Layer, iar Presentation Layer este reprezentat de Presentation.

Diagrama de componente

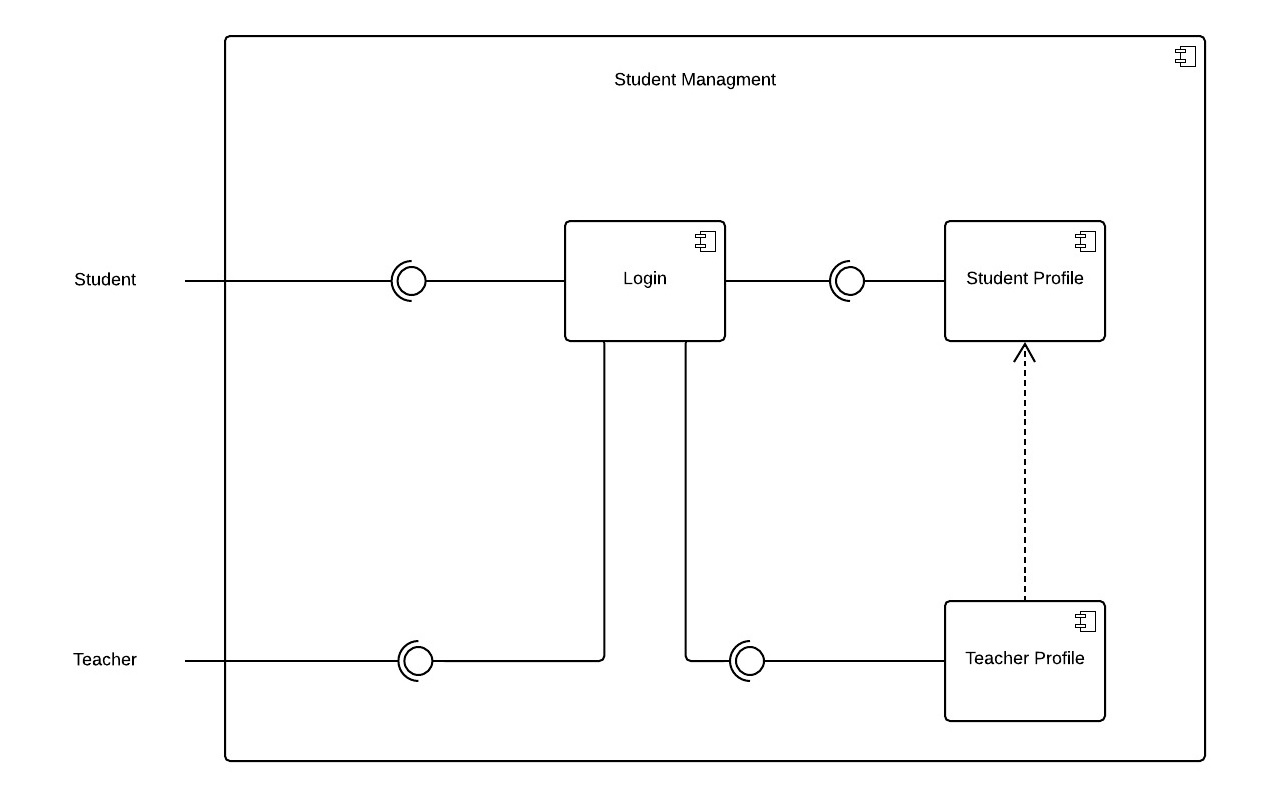
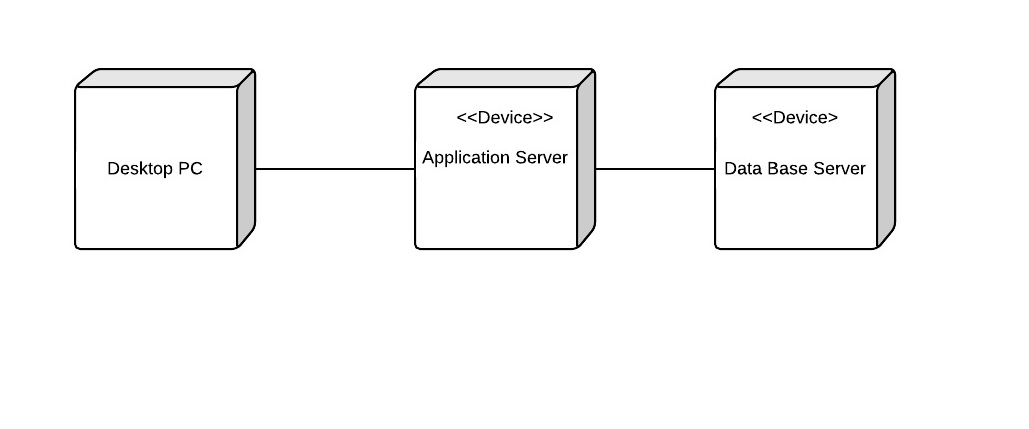
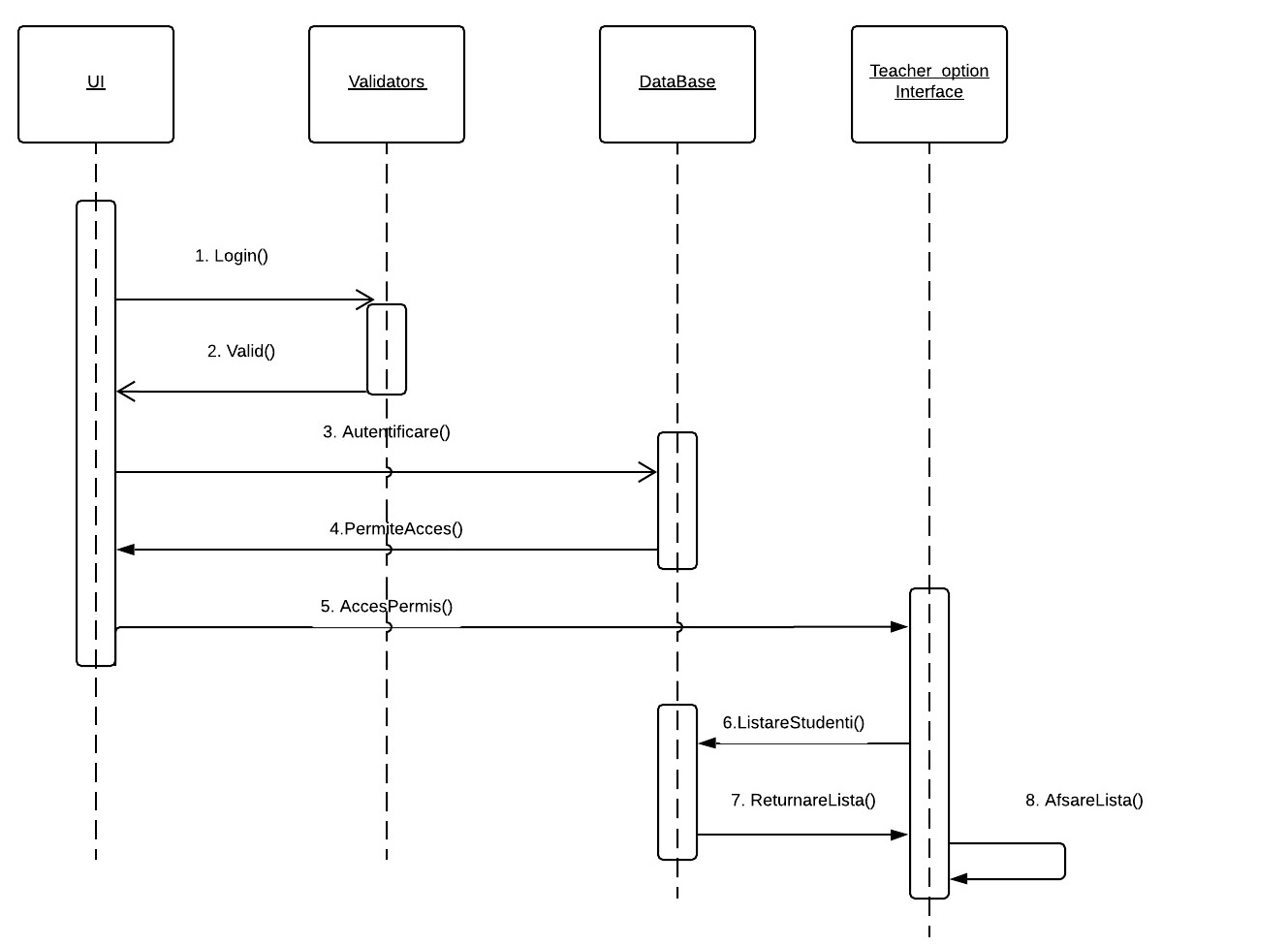


Diagrama de deployment:



# **4. Diagrama de secventa**

Caz: Vizualizare studenti de catre professor.



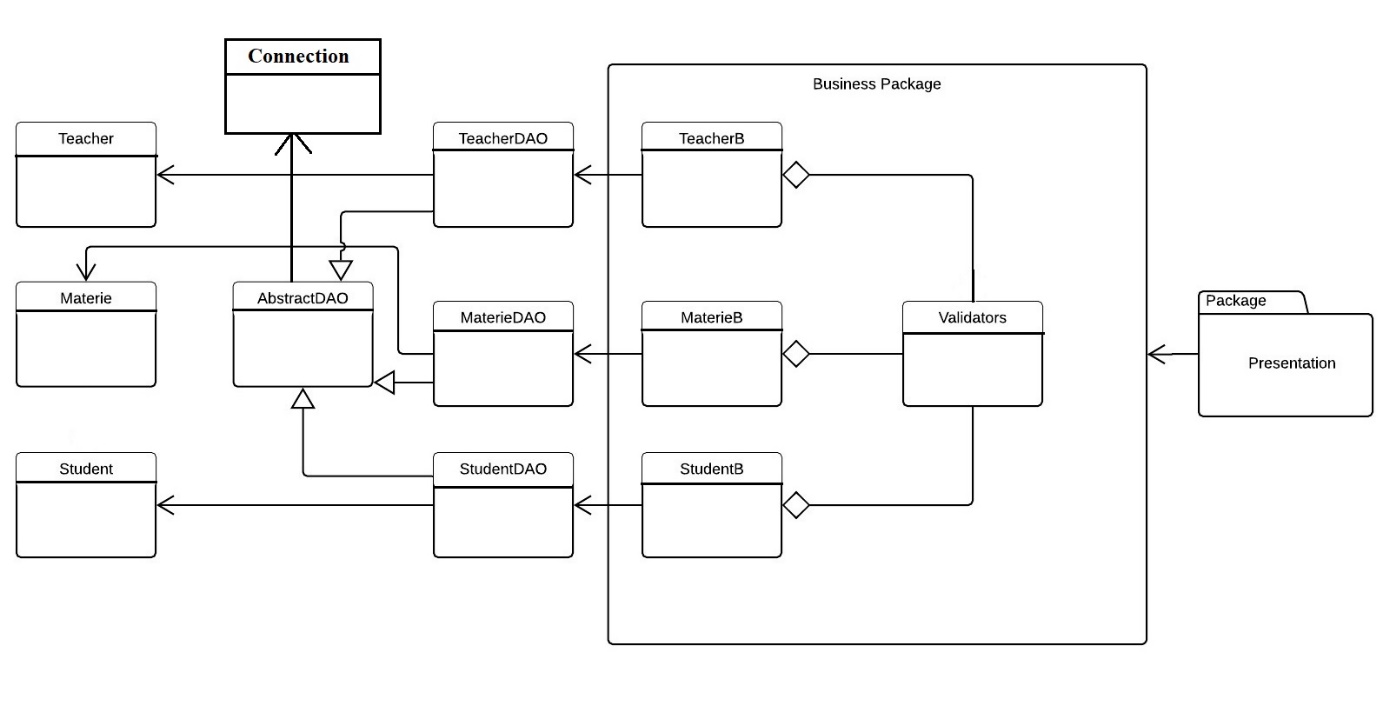
# **5. Class Design**

## **5.1 Design Pattern**

Design pattern-ul DAO(Data Acces Object). Vom folosit o clasa AbstractDAO prin care vom crea query generale pe care le putem aplica pe tabela din baza de date. Dupa ce am implementat aceasta clasa, ea va fi extinsa de o noua clasa astfel incat sa fie specifica pentru o anumita tabela din baza de date. De exemplu vom extinda clasa AbstractDAO de clasa StudentDAO lucru care va determina ca query-urile din AbstractDAO apelate din aceasta clasa sa opereze asupra tebelei Student din baza de date.

Un alt Design Pattern folosit este Singleton, design pattern care se asigura ca doar o instant a unei anume clase va fi create. Folosim acest design pattern in aplicatie in clasa care este responsabila cu conectarea la baza de date a aplicatiei.

## **5.2 Diagrama de clase**



Clasa AbstractDAO se foloseste de clasa Connection pentru a crea o conexiune cu baza de date, aici fiind pus in evidenta design patternul Singleton. Dup ce se termina operatia asupra bazei de date, conexiunea este inchisa, ea fiind deschisa de fieacare data cand are loc o noua operatie pe baza de date. Astfel avem o singura conexiune la baza de date la un moment dat in aplicatie.

Design patternul DAO este folosit dupa cum se poate vedea cu ajutorul claselor Student, Materie si Teacher care sunt entitati care reprezeinta tabelele din baza de date. Fiecare atribut al acestor clase reprezinta de fapt o coloana din tabelul din baza de date al carui nume clasa il poarta. De exemplu daca in baza de date exista o coloana in tabelul Student cu numele varsta, in clasa din Student din aplicatie va exista un atribut cu numele varsta.

# **6. Data Model**

Folosim clasa Student pentru a reprezenta tabela din baza de date Student. Avem ca atribute idStudent, nume, varsta, email precum si parola care este necesara pentru a te loga in aplicatie si o lista cu materiile pe care le are Studentul. Aceasta lista este reprezentata ca si un ArrrayLisT<Materie> unde materie este o alta tabela din baza de date.

Clasa Teacher este folosita pentru e reprezenta tabela din baza de date Teacher. Avem ca atribute idTeacher,nume, varsta , email , parola si materie. In cazul acestei clase, atributul materie reprezinta materia pe care o preda respectivul professor.

Clasa Materie este folosita pentru a reprezenta tabela din baza de date Materie, tabela care contine cheia primara a tabelei Student pentru a face legatura intre student si notele pe care acesta le are,precum si cheia prima a profesorului pentru a sti materia si numele profesorului.

# **7. Testarea Sistemului**

Pentru testarea sistemului folosim Pachetul Validators care contine metode care se asigura ca datele de intrare ale sistemului sunt corecte.

In acest pachet avem clase care verifica daca email-ul este introdus correct precum si daca numele sau id sunt valide.

# **8.Bibliografie**

<https://www.visual-paradigm.com/>

<https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658109%28v%3dpandp.10%29>

<https://www.tutorialspoint.com/jdbc/jdbc-db-connections.htm>

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

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