Managmentul studentilor din cadrul UTCN

Assignment 2

Document de Analiza si Design

Student:Dura Dragos Catalin

**Grupa:30233**

Cuprins

[**1. Analiza cerintelor** 2](#_Toc4616628)

[**1.1 Specificarea cerintei** 2](#_Toc4616629)

[**1.2** **Cerinte functionale** 2](#_Toc4616630)

[**1.3 Cerinte non-functionale** 2](#_Toc4616631)

[**2. Model use case** 2](#_Toc4616632)

[**3. Designul architectural al sistemului** 3](#_Toc4616633)

[**3.1 Descrierea pattern-ului architectural** 3](#_Toc4616634)

[**4. Diagrama de secventa** 6](#_Toc4616635)

[**5. Class Design** 7](#_Toc4616636)

[**5.1 Design Pattern** 7](#_Toc4616637)

[**5.2 Diagrama de clase** 7](#_Toc4616638)

[**6. Data Model** 8](#_Toc4616639)

[**7. Testarea Sistemului** 9](#_Toc4616640)

[**8.Bibliografie** 9](#_Toc4616641)

# **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.

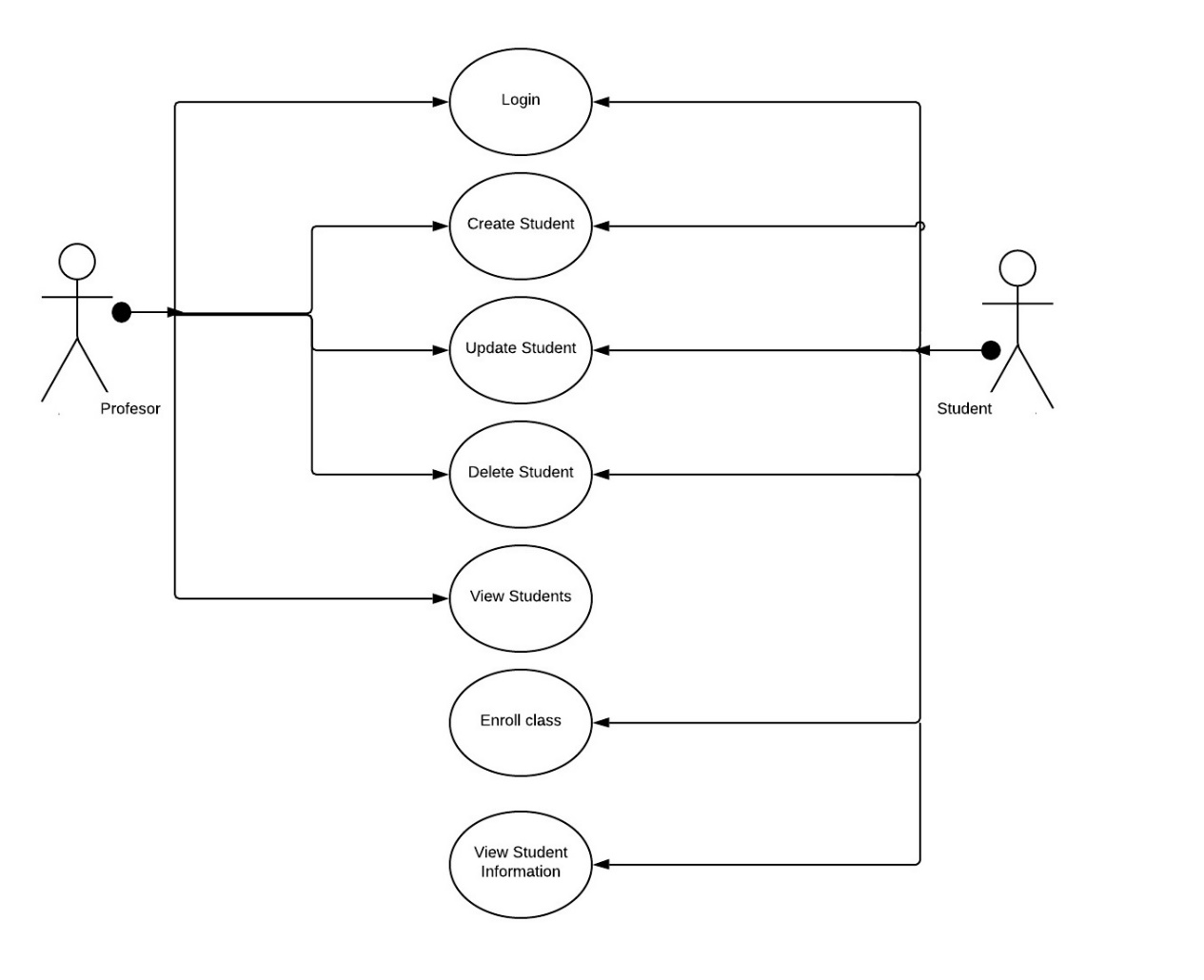
Profesorul poate face urmatoarele operatii: CRUD pe informatia despre student, precum si sa genereze rapoarte continand activitatile pe care un student le-a avut intr-o anumita perioada.

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( PostGreSQL) si trebuie folosit patternul MVC. De asemenea trebuie folosit in cadrul aplicatiei Dependency Injection iar proiectul trebuie sa fie Maven.

# **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**

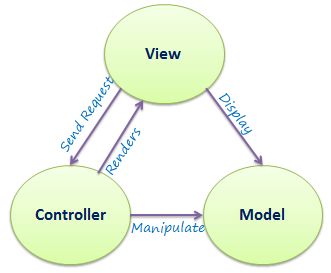
Ca si pattern architectural in cadrul acestei aplicatii am folosit MVC pattern. Componentele in cadrul acestui pattern sunt urmatoarele:

* Model
* View
* Controller

In componenta model avem clasele care reprezinta baza de date precum si logica aplicatiei.

In componenta view avem clasele care reprezinta interfata aplicatiei, interfata cu care utilizatorul va interactiona.

Componentata controller face legatura dintre View si Model, preluand din View operatia dorita de catre User si apeland din Model metodele necesare pentru a realiza operatia respective.

****

**3.2 Diagrame**

Diagrame de pachete:

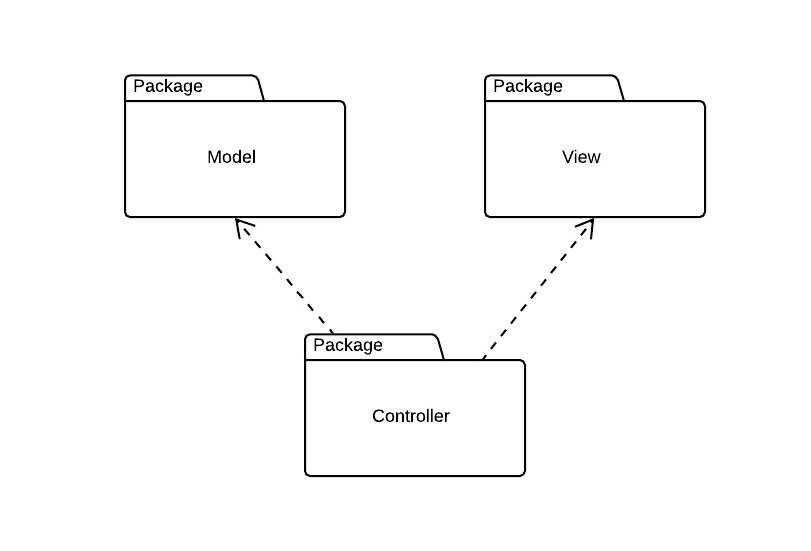


Diagrama de pachete pentru pachetul model:

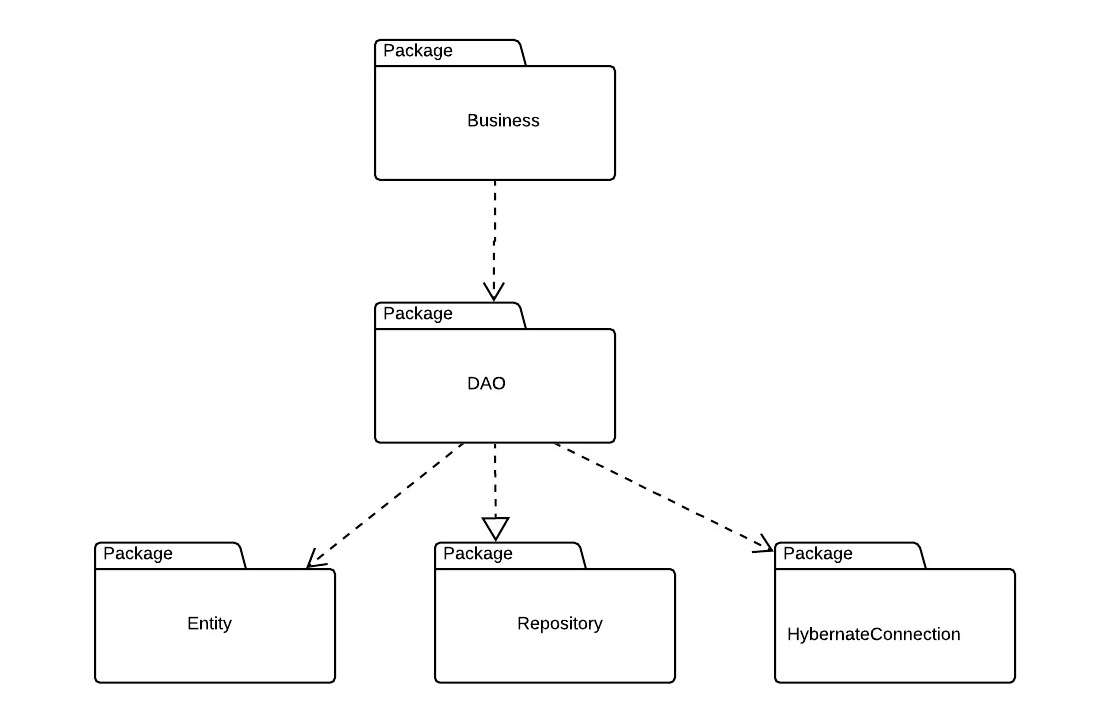


Diagrama de componente

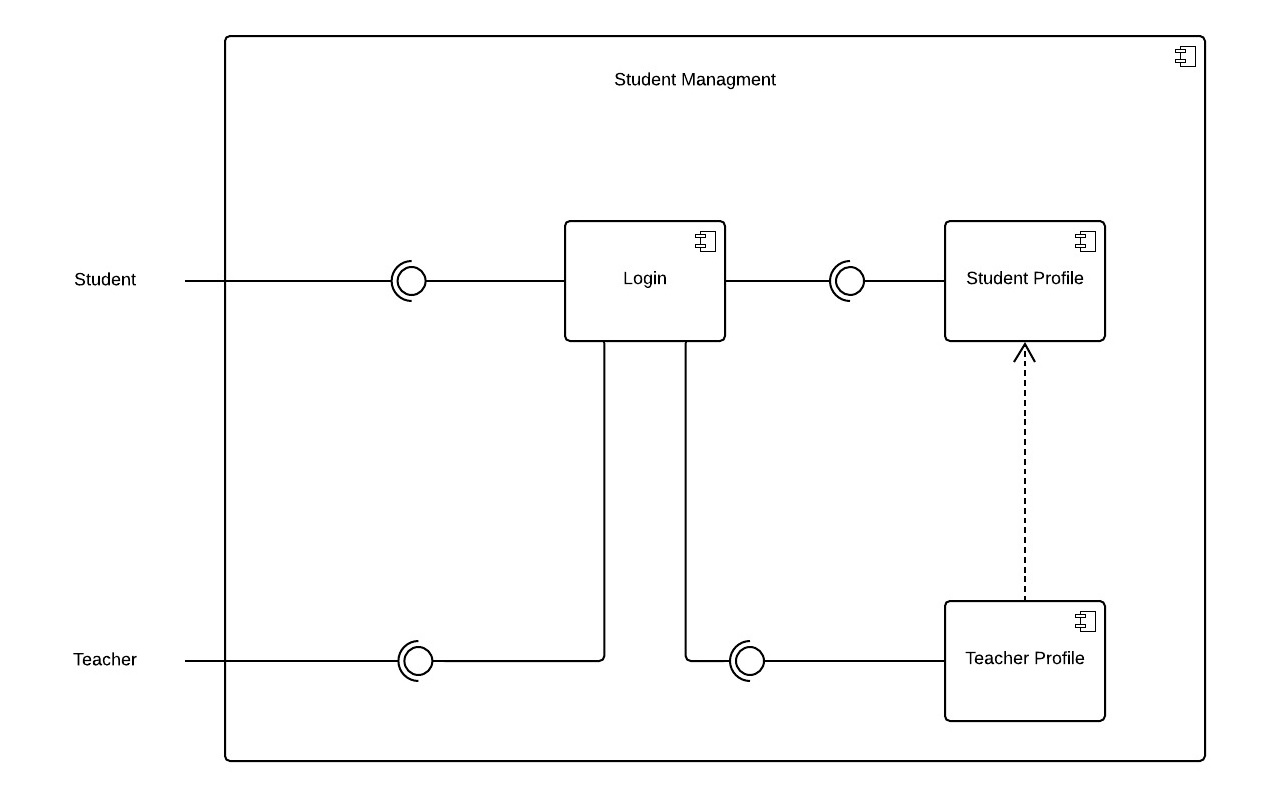
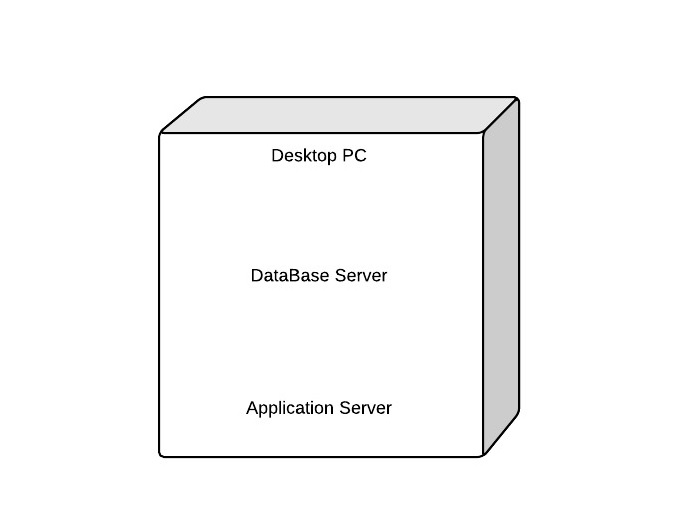


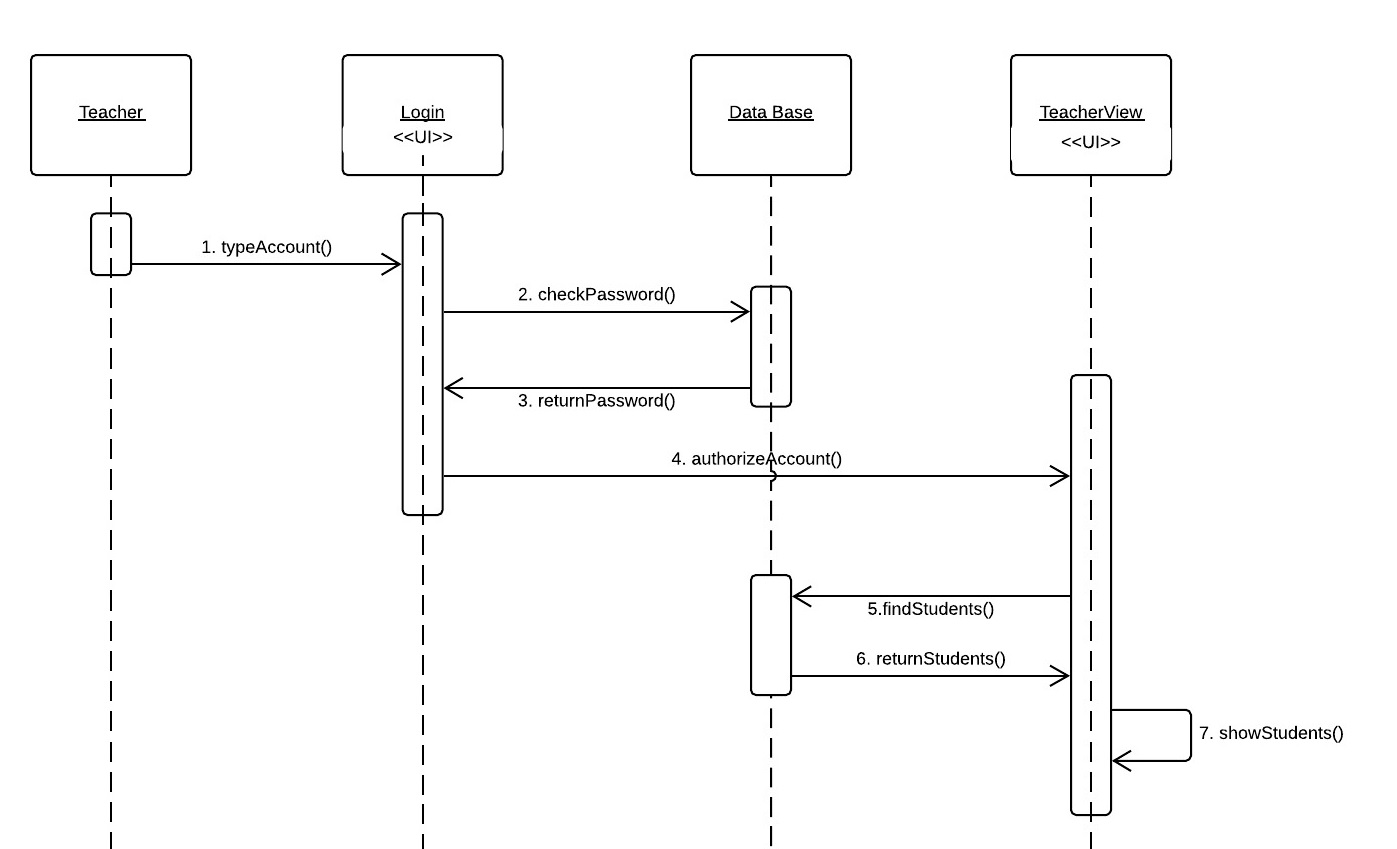
Diagrama de deployment:



# 

# **4. Diagrama de secventa**

Caz: Vizualizare studenti de catre professor.



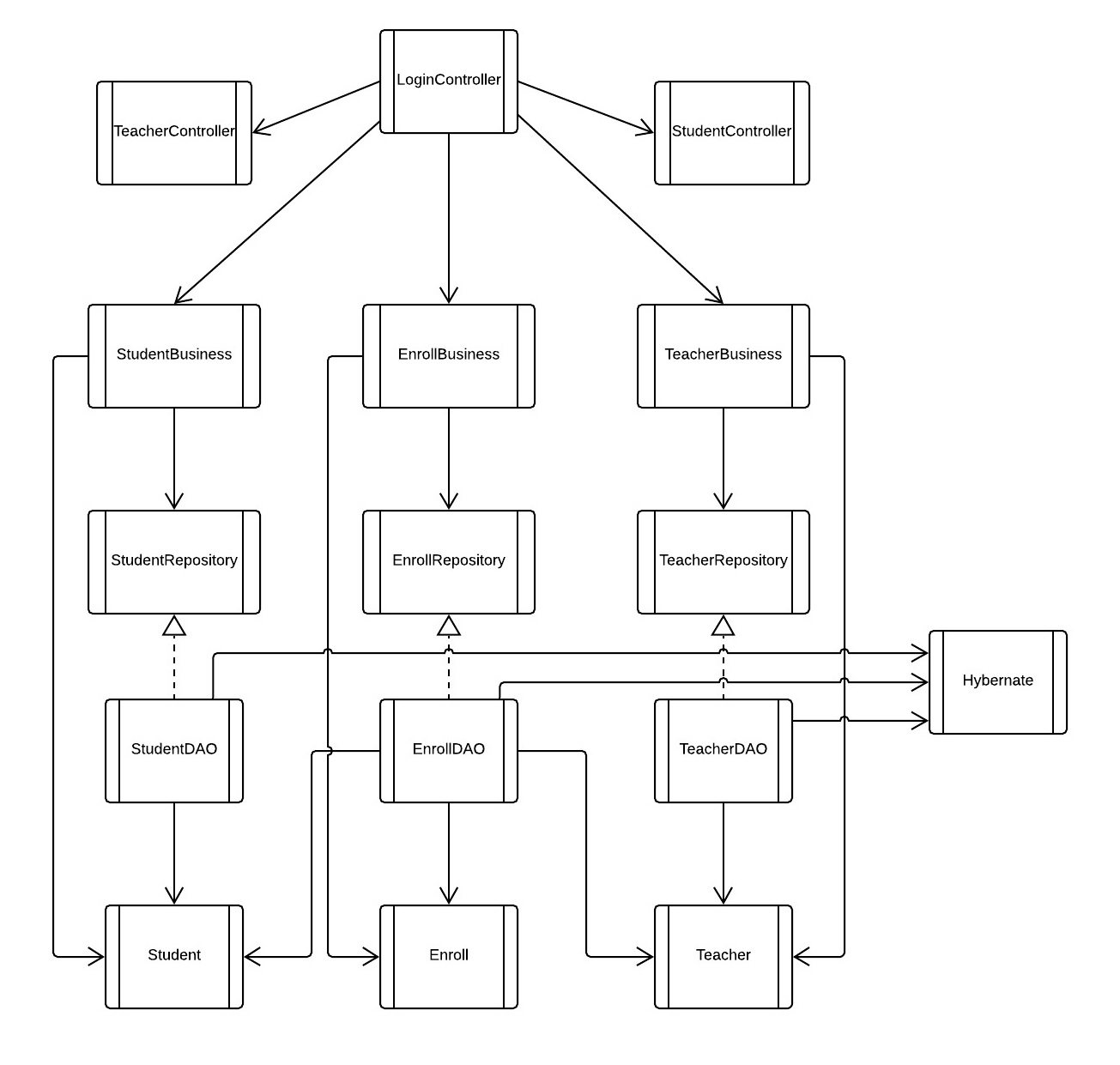
# **5. Class Design**

## **5.1 Design Pattern**

Design pattern-ul DAO(Data Acces Object). Vom folosit acest pattern pentru a executa queri-uri asupra bazei de date, queri-uri executate prin intermediul Hybernate. Pentru fiecare tabela din baza de date vom avea in Pachetul Model o clasa pojo, precum si in Pachetul DAO cate o clasa care sa se ocupe cu operatiile asupra unei anume tabele din baza de date. De asemenea pentru a putea implementa Dependency Injection fiecare clasa din DAO va extinde interfata care ii corespunde din Pachetul Repository, pachet care contine cate o interfata pentru fiecare tabela din baza de date, cu operatiile posibile asupra ei.

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**



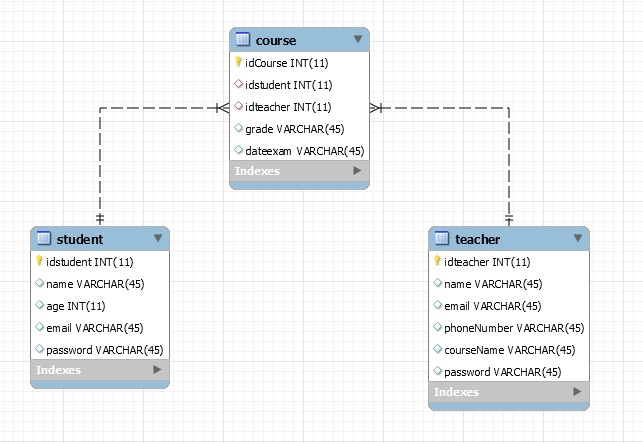
# 

# **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 Enroll este folosita pentru a reprezenta tabela din baza de date Enroll, tabela care contine cheia primara a tabelei Student , nota pe care acesta o are, data examenului, precum si cheia prima a profesorului pentru a sti materia si numele profesorului.



# **7. Testarea Sistemului**

Testarea sistemului a fost implementata folosindu-se Junit. In cadrul testarii am testat daca valoarea returnata pentru parola in cazul logarii unui student este cea asteptata, la fel si pentru cazul in care un profesor incerca sa se logheze. De asemenea am testat si operatiile de creare, stergere si actualizare student.

# **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>