<Assignment 1 - Student Management Application>

Analysis and Design Document

Student:Jakab Gyöngyi Anikó

**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 5

4. UML Sequence Diagrams 6

5. Class Design 7

6. Data Model 7

7. System Testing 7

8. Bibliography 7

1. Requirements Analysis

# Assignment Specification

Proiectarea și implementarea în Java a unei aplicații de management a studenților la departamentrul de CTI din UTCN. Aplicațiia ar trebuii să aibă două tipuri de utilizatori (student și profesor/administrator) care trebuie să furnizeze un username și password pentru a putea utiliza aplicația.

Utilizatorul de tip student este capabil de execuția următoarelor operații:

* Add/update/view pe informațiile personale (name,identity card number,personal numerical code,address etc.)
* CRUD pe informațiile legate de profilul de student (identification number, group, enrolments, grades)
* Operație de înrolare și procesare pentru cursuri înrolate

Utilizatorul de tip administrator este capabil de execuția următoarelor operații:

* CRUD pe profiluri de student
* Generare de rapoarte cu informații legate de activitatea unui student pe o perioadă anume

# Functional Requirements

Cerințele funcționale ale acestui proiect presupun următoarele operații: adăugarea, actualizarea și vizualizarea informațiilor personale ale unui student, crearea, vizualizarea, modificarea și ștergerea informațiilor din profilul de student, respectiv realizarea înrolării unui student la diferite cursuri și execuția operațiilor precum vizualizare data examenelor și a notelor finale a unui student pentru materiile înrolate. Aplicați furnizează suport și pe partea de admin, care presupune abiliatetea de a crea, vizualiza, modifica și șterge profile de student, respectiv de a genera raporturi despre activitatea unui student pe o perioadă anume.

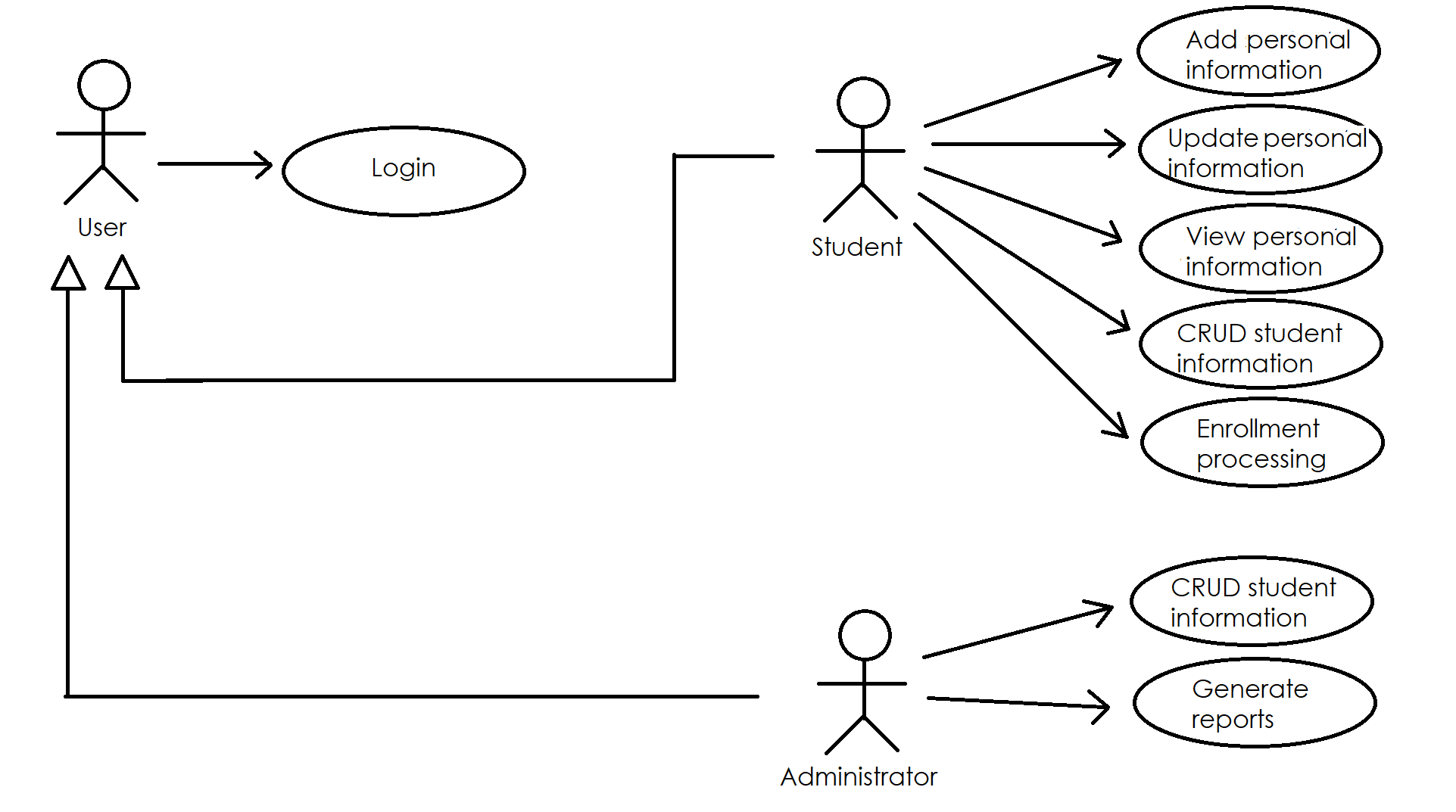
# Non-functional Requirements

Cerințele non-fincționale ale proiectului sunt:

* Siguranță: sistemul furnizează răspunsuri corecte pentru diferite date de intrare, pe baza autentificării el autorizează anumite operații bazate pe tipul de utilizator
* User – friendliness: aplicație ușor de utilizat pentru toate tipurile de utilizator
* Principii SOLID: sistemul să fie în așa fel proiectat încât modificarea, extinderea, mentenanță să fie ușor realizabile cu timpul

2. Use-Case Model

Actorul numit user – utilizatorul general înainte de conectare la aplicație – prin funrnizarea unui username și password se logează la aplicație și din acel moment vorbim despre două tipuri de utilizatori concreți: studentul poate executa operațiile: add / update / view personal information și crud pe student information, repsectiv enrollment processing, pe când administatorul poate face crud pe informațiile unui student și poate genera rapoarte bazate pe activitatea unui stundent pentru o perioadă anume.



*Use case: Login*

*Level: < user-goal level >*

*Primary actor: <User >*

*Main success scenario: Se deschide aplicația, utilizatorul furnizează un username și un password, sistemul ca răspuns îi afișează un view principal de student sau administrator doar cu operațiile pe care acesta este capabil să le săvârșească.*

*Extensions: dacă utilizatorul nu are cont, acesta va avea posibilitatea de signup la aplicație*

*Use case: Add personal information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: Utilizatorul de tip student introduce în textbox-urile de pe interfața de utilizator informațiile dorite și apăsând pe un buton din acesta adaugă informațiile noi în partea alocată lui în baza de date*

*Extensions:*

*Use case: Update personal information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: Utilizatorul de tip student introduce în textbox-urile de pe interfața de utilizator informațiile dorite și apăsând pe un buton din acesta modifică informațiile vechi cu cele noi în partea alocată lui în baza de date*

*Extensions:*

*Use case: View Personal information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: Utilizatorul de tip student vizualizează informațiile personale înregistrate despre el în momentul curent*

*Extensions:*

*Use case: Crud student information*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: Utilizatorul de tip student vizualizează datele curent înregistrate despre situația lui școlară, intorducând date noi poate să le suprascrie pe cele vechi sau poate să ștreagă datele din profilul lui de student*

*Extensions:*

*Use case: Enrollment processing*

*Level: < user-goal level >*

*Primary actor: <Student>*

*Main success scenario: Pentru un utilizator de tip student îi sunt afișate toate cursurile la care poate să se înroleze bazat pe situația lui școlară, introducând un cod specific poate să se înroleze la o materie anume, mai târziu având posibilitatea de a vizualiza data examenului și nota finală la materia anume.*

*Extensions:*

*Use case: Crud student information*

*Level: < user-goal level >*

*Primary actor: <Administator>*

*Main success scenario: Utilizatorului de ti adminitator îi vor fi specificate studenții curenți ai aplicației având posibilitatea de a schimba informațiile acestora sau de a șterge conturi, respectiv de a introduce noi studenți în sistem.*

*Extensions:*

*Use case: Generate reports*

*Level: < user-goal level >*

*Primary actor: <Administator>*

*Main success scenario: Utilizatorul are posibiliteate de a genera raporte despre un student anume, vizualizând materiile la care acesta este înrolat și notele obținute din sesiunea curentă sau cele anterioare.*

*Extensions:*

3. System Architectural Design

**3.1 Architectural Pattern Description**

Pentru organizarea logică a proiectului am utilizat șablonul arhitectural Layer, acesta presupune structurarea logică a proiectului în 3 nivele principale anume: Presentation Layer, Business Logic Layer și Data Access Layer.

În nivelul Presentation se construiește pe șablonul MVC (Model-View-Controller) în implementare javafx adică un șablon arhitectural mai aproape MP (Model-Presentation).

La nivelul de Data Access Layer s-a folosit Reflection API pentru generarea unei DAO abstract care este moștenit de toate celelalte pentru a realiza o implementare mai compactă a acestui nivel.

**3.2 Diagrams**

Diagrama de pachete a sistemului:

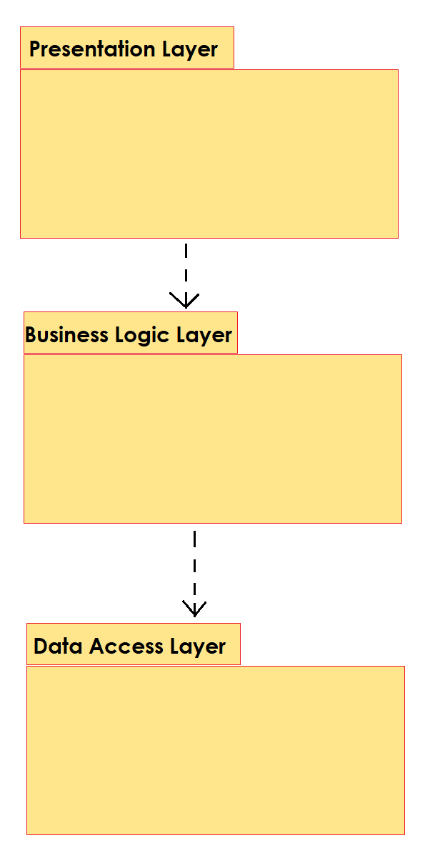
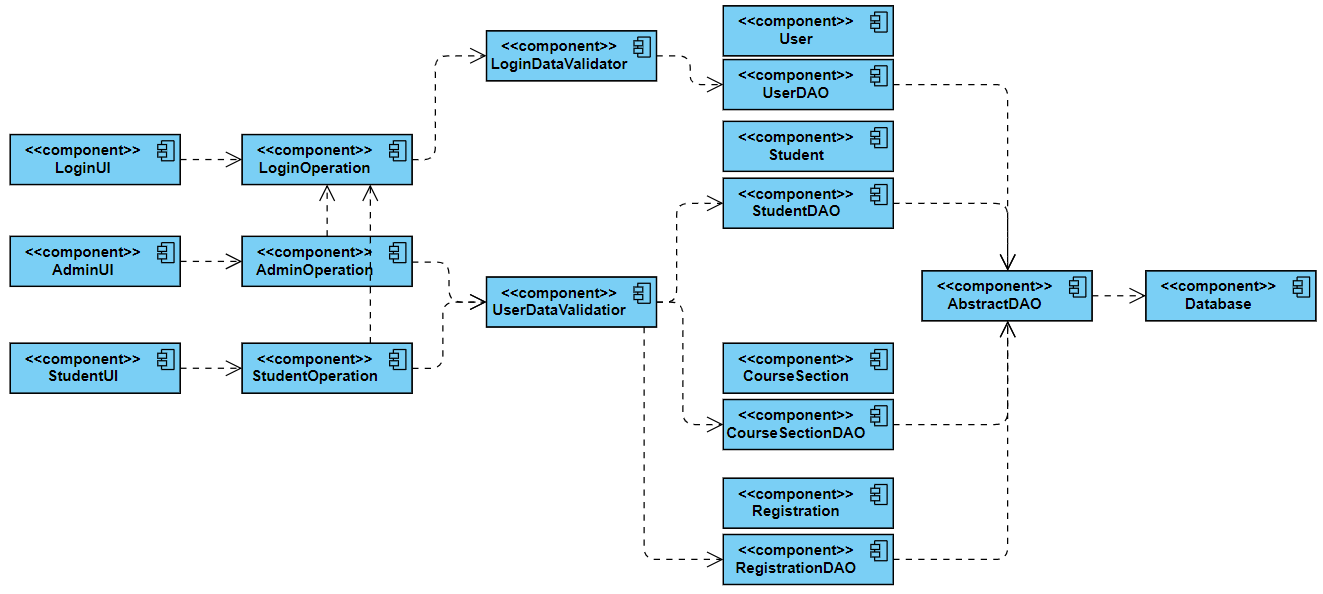
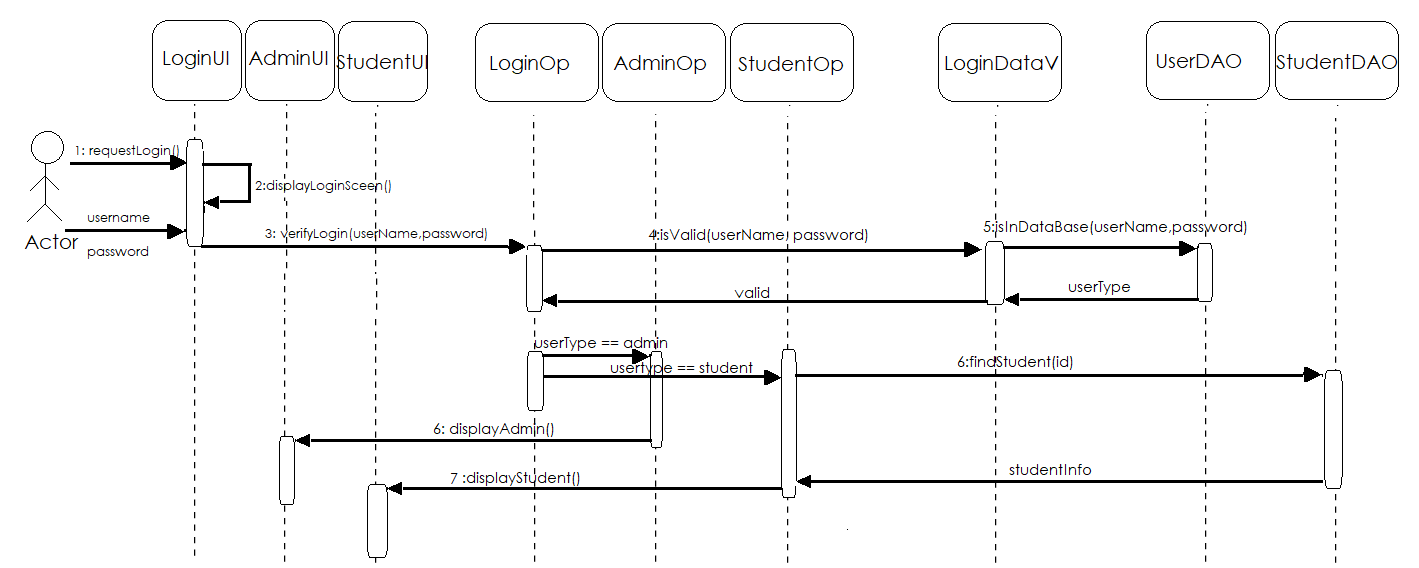
****

Diagrama de componente a sistemului:



4. UML Sequence Diagrams

Diagrama de secvență pentru logarea a unui utilizator oarecare, care prin returul de informație va fi tratat în viitor ca student sau administator:



5. Class Design

**5.1 Design Patterns Description**

Șablonulrile arhitecturale folosite pentru implementarea proiectului:

* Layer: Șablonul arhitectural Layer este unul utilizat pentru stucrurarea / împărțirea logică a unei aplicații în așa fel ca conceptele legate ca funționalitatea sau un domeniu anume să se afle în aceleași pachet. Există dependințe între aceste pachete, anume pachetele de nivel mai înalt depind de cele de nivel jos, dar nu și invers.
* MVC like MP: Un șablon arhitectural utilizat pentru împărțirea logică a nivelului sau unei componente de tip interfață utilizator, în Model,Vizualizare și Controller. În cazul nostru folosind javafx modul de implementare ar semana mai mult cu un șablon Presentation-Model, unde Model-ul este entitatea vizualizată în partea de prezentare, care în sine este View-ul și Contoller-ul(se pot folosii fișiere de tip fxml drept view prin crearea acestora folosind sceneBuilder-uri, dar Controller-ul și în acest caz trebuie să aibă toată entitățile declarate în fișierul fxml pentru a avea acces la ele).
* Reflection API: Șablon folosit pentru genera dinamic (la runtime) diferite requesturi către baza de date cu scupul de a avea o implementare compactă și a nu avea secvențe de cod duplicate la nivelul de aces la bază de date.
* Pentru creare conexiuni la baza de date s-a folosit conectorul jdbc prin dependecy injection suport oferit de maven. Totuși conexiune concreată este realizată de clasa numită ConnectionFactory, realizat prin șablonul Singleton.

**5.2 UML Class Diagram**

Diagrame de clase al pachetului Data Acces Layer:

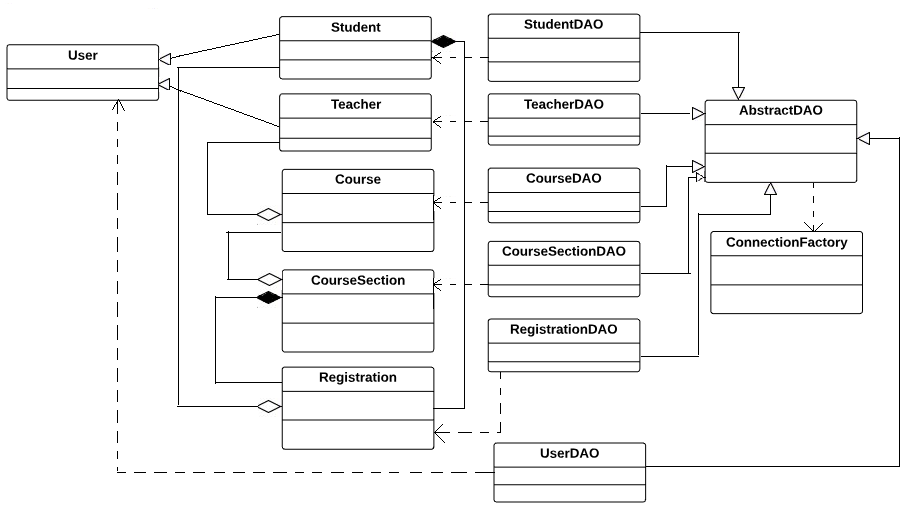
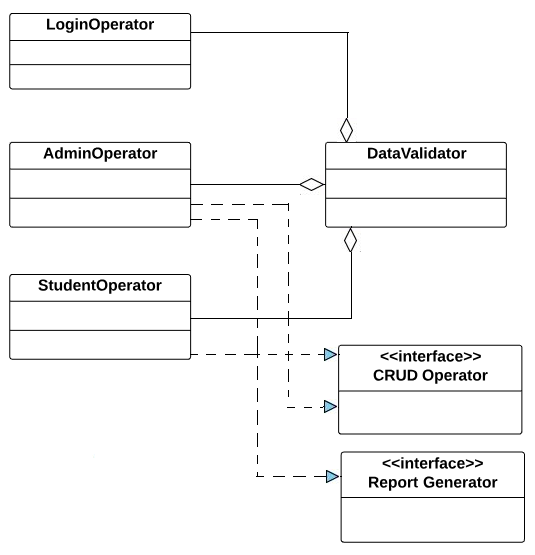
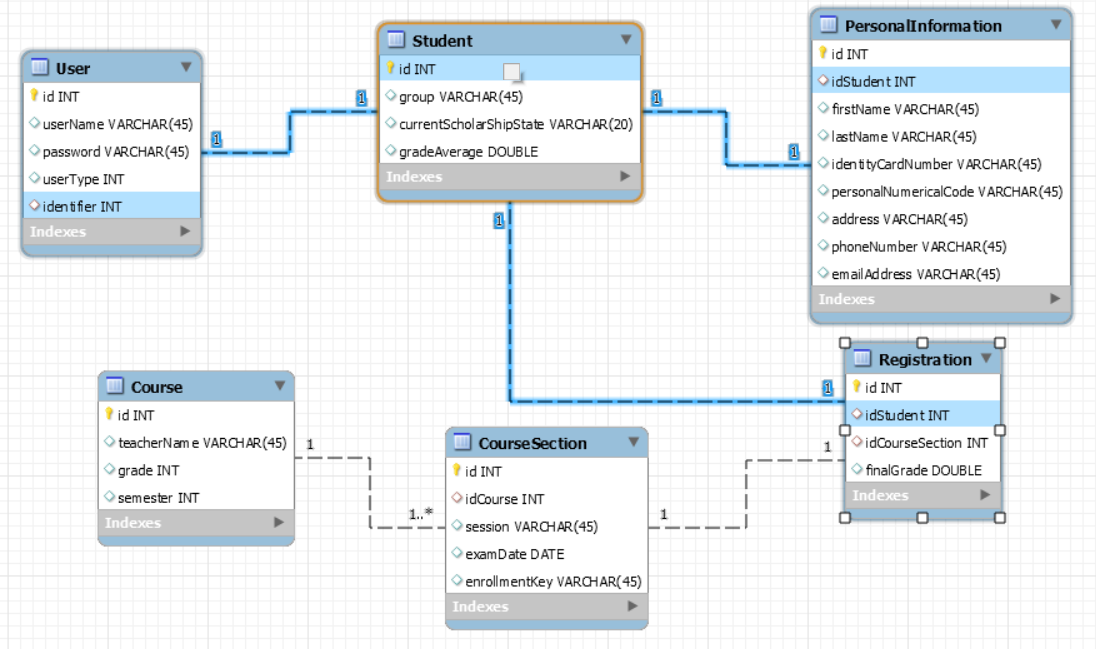


Diagrama de clase al pachetului Business Logic Layer:



6. Data Model

Organizarea bazei de date pe tabele:



7. System Testing

Testarea sistemului se face separat pe pachete pornind de nivelul cel mai de jos. Pentru testarea modulelor se va folosii suportul de testare oferit de mediul Intelij IDEA, anume Junit. Ca și prim pas se va testa conexiunea la baza de date, apoi funcționarea corectă a modulelor DAO, și incremental se va ajunge la nivel interfeței utilizator.

8. Bibliography

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

[2] <https://www.lucidchart.com/pages/uml-class-diagram>

[3] <https://www.visual-paradigm.com/solution/sequencedgm/uml-sequence-diagram-tool/?gclid=CjwKCAjwycfkBRAFEiwAnLX5IePLvOG3NWH7Jocx-kpfjzzQ0OBdSez5VQ8CRpxyAdElK6ML1CoDRxoCdwEQAvD_BwE>