Students Management

Student: Costea Ana-Maria

**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

Aplicatia pe care urmeaza sa o implementez este una pentru management ul studentilor din departamentul de Computer Science al UTCN. Aceasta aplicatie trebuie sa aiba doua tipuri de utilizatori, si anume student sau profesor/administrator retea. Utilizatorul trebuie sa se inregistreze cu numele si parola pentru a putea folosi aplicatia.

# Functional Requirements

Utilizatorul normal, adica studentul, are acces la urmatoarele operatii :

* Adaugarea, editarea, vizualizarea informatiilor de utilizator ( nume, numar card de identitate, cod numeric personal, adresa, etc)
* Crearea, editarea, stergerea, vizualizarea profilului de student ( informatiile contului : numar de identificare, grupa, inrolari, note)
* Inrolarea la clase ( inrolare, examene, note)

Utilizatorul administrator are acces la urmatoarele operatii :

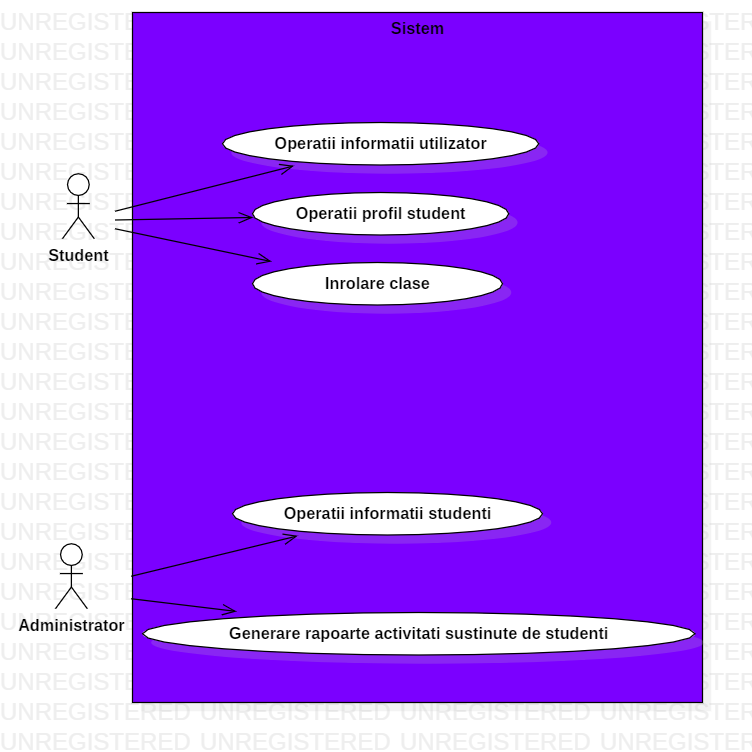
* CRUD pe informatiile studentilor
* Generarea de rapoarte pentru o perioada particulara ce contine activitatile sustinute de un student

Ca si cerinte functionale, trebuie sa implementez operatiile de mai sus si apoi sa testez aplicatia.

# Non-functional Requirements

Ca cerinte non-functionale ale aplicatiei pe care urmeaza sa o implementez se pot mentiona performanta programului ( de exemplu timpul de raspuns, utilizarea), valabilitatea, securitatea, integritatea datelor. Toate datele cu care urmeaza sa lucrez vor fi introduce prin intermediul unei baze de date. Diferenta dintre cerintele functionale si cele non-functionale este ca prima categorie descrie ce anume ar trebui sistemul sa faca, pe cand a doua categorie descrie cum un sistem lucreaza.

2. Use-Case Model



Use-Case description :

* **Use case** : selectarea operatiei dorite de catre utilizator
* **Level** : use-goal level
* **Primary actor** : student sau administrator retea/professor
* **Main success scenario** : Utilizatorul se inregistreaza, alegand tipul de utilizator, introduce numele si parola contului, apoi alege operatia pe care doreste sa o efectueze. Sistemul analizeaza cererea utilizatorului si afiseaza rezultatul dorit.
* **Extensions** : Utilizatorul alege tipul de utilizator, dupa care introduce datele cu care se conecteaza. Un scenariu de succes poate fi reprezentat de : utilizatorul introduce corect datele, avand astfel acces la operatiile destinate tipului sau de utilizator. Un scenario de esec poate fi : utilizatorul nu introduce corect datele corespunzatoare si nu are acces la operatiile destinate lui, afisandu-se un mesaj de eroare ( de exemplul, “Nume de utilizator sau parola introduce gresit!”).

**­­**

3. System Architectural Design

**3.1 Architectural Pattern Description**

In implementarea aplicatiei, ma voi folosi de urmatoarele pattern-uri arhitecturale:

* **Layered pattern**

Acest pattern il folosesc pentru a structura programele ce pot fi compuse in grupuri de subtask-uri, fiecare din acestea fiind la un nivel particular de abstractie. Fiecare layer furnizeaza servicii urmatorului layer mai mare. Cele mai comune layere folosite sunt :

- Presentation layer ( UI layer)

- Application layer ( service layer)

- Business logic layer ( domain layer)

- Data access layer ( persistence layer)

* **Client-server pattern**

Acest pattern este format din 2 parti, dupa cum spune si denumirea sa : un server si multipli client. Server-ul furnizeaza servicii multiplilor client. Clientii cer servicii de la server, iar acesta furnizeaza servicii relevante acestor client. Mai mult, server-ul continua sa asculte cererile clientului.

* **Broker pattern**

Acest pattern e folosit pentru a structura sistemele distribuite cu componente decuplate. Aceste componente pot interactiona unele cu altele prin invocari de servicii la distanta. Componenta broker este responsabila pentru coordonarea comunicatiei dintre componente.

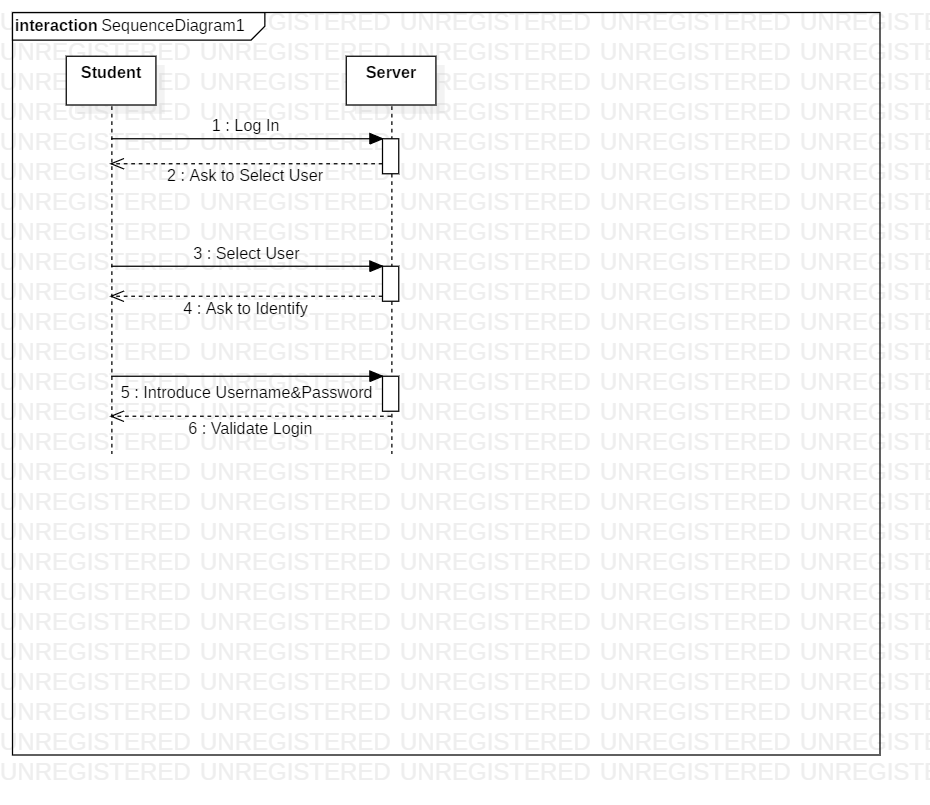
Server-ele isi publica capabilitatile broker-ului. Clientii cer un serviciu de la broker, iar acesta redirectioneaza apoi clientul spre un serviciu potrivit din registrul sau.

**3.2 Diagrams**

****

4. UML Sequence Diagrams

Voi proiecta diagrama de secventa pentru scenariul de logare al unui utilizator.



5. Class Design

**­**

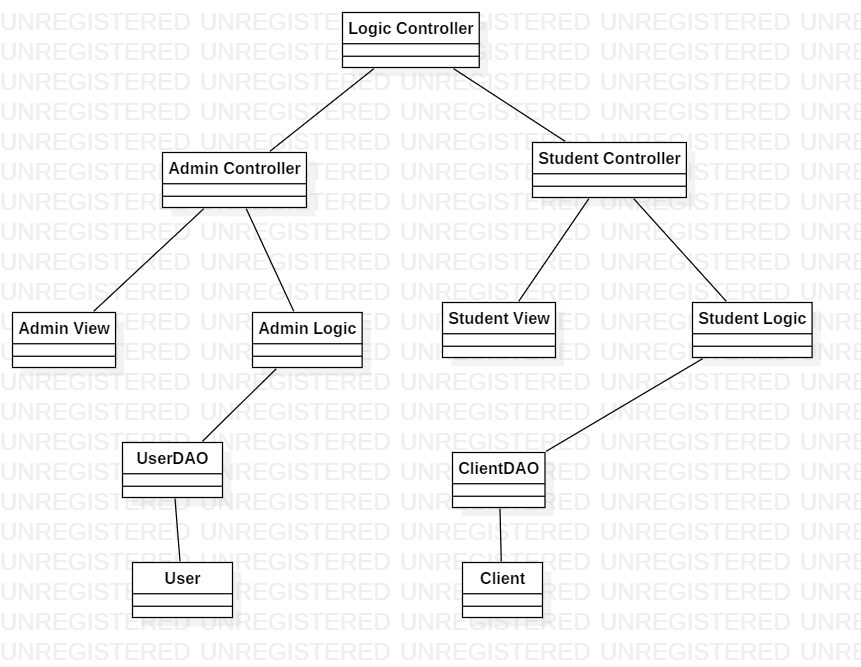
**5.1 Design Patterns Description**

Pentru aceasta aplicatie, am folosit urmatoarele design pattern-uri:

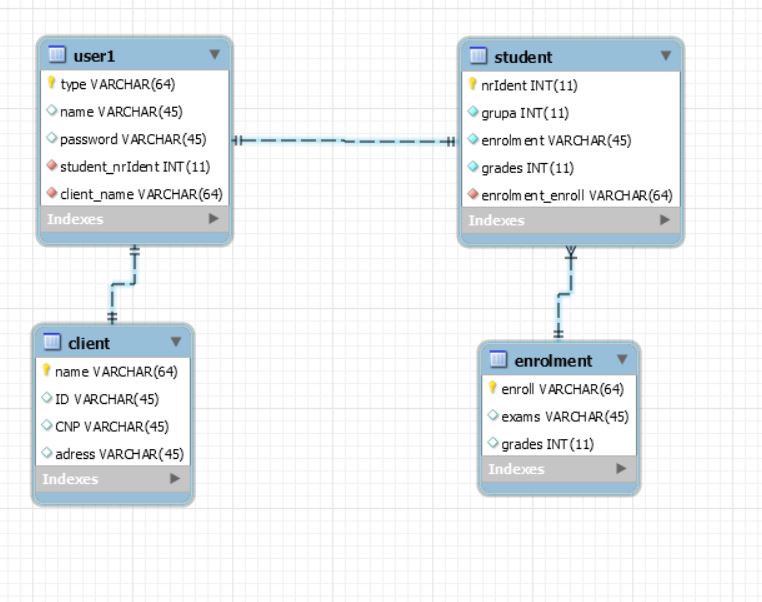
* **MVC** ( Model-View-Controller), pentru a conecta diferitele parti ale aplicatiei in model, view, controller. Modelul reprezinta obiectele Java, view reprezinta modul in care datele pot fi vizualizate si controller-ul controleaza fluxul de date si modifica vizualizarea atunci cand partile din model se schimba
* De asemenea, pot folosi **Observer** **Pattern**. Acesta este foarte util atunci cand suntem interesati de starea unui obiect si vrem sa fim notificati de fiecare data cand are loc o schimbare. In acest design pattern, obiectul care observa starea altui obiect se numeste **Observer**, iar obiectul care este observant se numeste **Subiect**. In aplicatia mea, Observer Pattern este util atunci cand un profesor posteaza o nota la o materie unui student; studentul respective va fi notificat atunci cand o noua nota va aparea in situatia sa scolara. De asemenea, atunci cand un student doreste sa se inroleze la o noua clasa, profesorul respective va fi notificat si va accepta sau nu cererea facuta de student.

**5.2 UML Class Diagram**

Diagrama de clasa UML completa o voi face la finalul implementarii intregii aplicatii, pentru a putea ilustra cat mai complex toate clasele si metodele folosite. Acum, voi crea o diagram de clasa provizorie.



6. Data Model



7. System Testing

Cazurile de testare vor fi create pentru a verifica daca un student isi poate modifica o informative din profilul sau ori daca un profesor poate sterge una din notele adaugate unui student.

8. Bibliography

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

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

[3] <https://www.journaldev.com/1827/java-design-patterns-example-tutorial>