MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE SEMESTRU

la disciplina
Introducere în baze de date

Sistem informatic destinat gestiunii unei platforme de studiu

Kovacs Alexandru

Priscuri Larisa

Litu Saviana

An academic 2020 - 2021

CUPRINS

1. Introducere 3

2. Analiza cerințelor utilizatorului (specificațiile de proiect) 3

2.1. Ipoteze specifice domeniului ales pentru proiect 3

2.2. Organizarea structurată a cerințelor utilizatorilor 4

2.3. Determinarea și caracterizarea de profiluri de utilizatori 4

3. Modelul de date și descrierea acestuia 6

3.1. Entități și atributele lor 6

3.2. Diagrama EER/UML pentru modelul de date complet 9

3.3. Proceduri/view-uri/triggere/event-uri 9

3.4. Normalizarea datelor 14

3.5. Interogări MySQL 15

3.6. Cod MySQL 16

3.6.1. Cod proceduri 16

3.6.2. Cod triggere 20

3.6.3. Cod pentru view/event 21

4. Detalii de implementare 23

4.1. Structura de clase în Java. Diagrama UML 23

4.2. Manual de utilizare/instalare 28

4.3. Elemente de securitate a aplicației 28

5. Concluzii. Limitări și dezvoltări ulterioare 29

1. Introducere

Proiectul a presupus crearea unui sistem informatic destinat gestiunii unei platforme de studiu. Scopul aplicației este de a facilita gestiunea datelor personale ale utilizatorilor, gestiunea activităților didactice și notarea studenților, precum și crearea grupurilor de studiu și a activităților extracurriculare, accesându-se baza de date a universității. Toate acestea se realizează de către utilizator, care este clasificat după rolul pe care îl deține în cadrul platformei (super administrator, administrator, profesor sau student), exclusiv prin intermediul interfeței grafice.

Instrumente software folosite în cadrul realizării proiectului:

- **MySQL Workbench și MySQL Server 8.0** – realizarea bazei de date utilizând limbajul MySQL
- **Draw.io** – realizarea diagramei UML a structurii bazei de date
- **IntelliJ** – mediu de programare Java
- **JDBC** – realizarea conexiunii dintre aplicație și baza de date
- **Spring MVC** – framework folosit pentru realizarea aplicației web
- **Bootstrap 4** – realizarea interfeței grafice

2. Analiza cerințelor utilizatorului (specificațiile de proiect)

2.1. Ipoteze specifice domeniului ales pentru proiect

Se dorește implementarea unui sistem informatic destinat gestiunii unei platforme de studiu. Aplicația folosește un sistem de gestiune pentru baze de date MySQL, iar interacțiunea cu aceasta se realizează exclusiv prin intermediul interfeței grafice. Funcționalitățile pe care le oferă programul vizează operații ce țin de gestiunea studenților, profesorilor și administrarea operațiilor curente din cadrul unor programe de studiu.

Aplicaţia poate fi accesată, pe baza unui proces de autentificare, de către mai multe tipuri de utilizatori: studenți, profesori, administratori. Fiecare utilizator își va putea vizualiza datele personale imediat după ce va accesa sistemul informatic, fără a avea însă posibilitatea de a le modifica. Totodată, platforma oferă și o funcționalitate pentru deautentificare, prin care se revine la fereastra care solicită datele de acces, astfel încât și un alt utilizator să îl poată folosi ulterior, fără a fi necesară repornirea sa. Aplicația oferă utilizatorului diferite funcționalități în funcție de rolul acestuia.

Aplicația va permite gestiunea cu ușurință a activităților didactice și astfel a interacțiunilor dintre studenți și profesori, facilitând accesul la informațiile din baza de date a universității.

Fiecare activitate didactică se desfasoara recursiv intre doua date, pe o anumita perioada de timp, și are un număr maxim de participanți. Totodată, va permite notarea studenților, oferă posibilitatea înscrierii la grupuri de studiu, la activitășI extracurriculare și nu în ultimul rând descărcarea listelor de studenți sau de activități de către utilizatorii cu un anumit rol.

2.2. Organizarea structurată a cerințelor utilizatorilor

Baza de date trebuie să satisfacă următoarele cerințe:

- Stocarea datelor personale a utilizatorilor, diferite în funcție de rolul acestuia în cadrul platformei
- Stocarea materiilor și activităților didactice susținute de un profesor identificat printr-un ID unic, precum și a detaliilor ce definesc activitățile respective
- Stocarea informațiilor legate de grupurile de studiu la care se pot înscrie studenții
- Stocarea informațiilor legate de activitățile extracurriculare desfășurate ăn cadrul activităților de studiu
- Să conțină proceduri și view-uri care să selecteze și să filtreze datele solicitate, odată cu verificarea acestora

Aplicația Java trebuie să satisfacă următoarele funcționalități:

- Expunerea către utilizator a informațiilor solicitate și a modificărilor făcute asupra bazei de date prin intermediul interfeței grafice în timp real
- Administrarea cerințelor utilizatorului și extragerea informațiilor din baza de date
- Prelucrarea bazei de date prin modificarea(update, create, delete) datelor reținute de aceasta

2.3. Determinarea și caracterizarea de profiluri de utilizatori

Aplicația poate fi folosită de către utilizatori clasificați în 4 tipuri, fiecare putând realiza acțiuni diferite în cadrul platformei.

1. **Administrator (**poate opera numai asupra utilizatorilor cu rolul de student sau profesor**)**:
    - filtrarea utilizatorilor după tip căutarea acestora după nume și/sau prenume
    - ștergerea sau adăugarea unui utilizator de tip profesor sau student
    - editarea datelor unui utilizator selectat din listă

- căutarea unui curs după nume, vizualizarea profesorilor care il predau și a studenților înscriși la el
- asignarea profesorilor la materii, precum și selectarea activităților didactice pe care aceștia le vor susține

2. **Super-Administrator** (poate opera asupra tuturor celorlalte roluri):
   - aceleași drepturi ca și utilizatorul de tip admin
   - poate în plus să efectueze operații de tipul filtrare, adăugare, editare și ștergere și asupra utilizatorilor de tip admin

3. **Profesor**:
   - vizualizarea cursurilor la care este asignat și setarea ponderilor pentru fiecare activitate de la cursurile respective
   - programarea activităților într-un calendar pe date și ore și a examinărilor
   - posibilitatea de a asista la activitățile extracurriculare din cadrul grupurilor de studiu
   - accesarea unui catalog unde vede studenții asignați la cursul predat de acesta și adăugarea de note la activitățile la care este acesta înscris, precum și descărcarea catalogului
   - vizualizarea activităților din ziua curentă, precum și cele la care sunt asignati sau înscriși și descărcarea listelor sub formă de document

4. **Student**:
   - încrierea la cursuri și renunțarea la acestea
   - înscrierea la activități la care doresc să participe (în cazul în care nu există suprapuneri de program și mai sunt locuri disponibile)
   - vizualizarea mediei finale pe care o au la o anumită materie
   - înscrierea și părăsirea grupurilor de studii
   - crearea de grupuri de studii
   - trimiterea și vizualizarea mesajelor în grupuri de studii
   - crearea de activități extracurriculare din cadrul grupurilor de studii
   - înscrierea la activitățile extracurriculare
   - vizualizarea activităților din ziua curentă, precum și cele la care sunt asignati sau înscriși și descărcarea listelor sub formă de document

3. Modelul de date și descrierea acestuia

3.1. Entități și atributele lor

Tabele:

- **Utilizator**: tabel care stochează informațiile personale prezente la toți utilizatorii (atributele *id tip utilizator, cnp, nume, prenume, adresă, număr de telefon, email, IBAN și numărul de contract*), care se identifică printr-un id unic()

- **Date Logare**: stochează datele de logare (atributele *nume de utilizator și parolă* sunt folosite la logarea în aplicație) pentru fiecare utilizator, al cărui id de identificare corespunde celui din tabelul Utilizator.

- **Tip utilizator**: tabel care reține rolul pe care il are un utilizator (atributul *nume* reprezintă denumirea rolului) identificat printr-un id unic(*id_tip_utilizator*), rolurile fiind predefinite.

- **Profesori**: reține informațiile personale specifice tipului de utilizator profesor(atributele fiind *minimul și maximul de ore* ore pe care acesta le poate preda și *departamentul* din care acesta face parte). Acesta se identifică printr-un id unic care îi corespunde unui utilizator, făcând legătura cu tabelul utilizator(*id_profesor*).

- **Studenți**: reține informațiile personale specifice tipului de utilizator student(atributele fiind *anul* în care se află studentul și *numărul de ore* pe care trebuie să le susțină). Acesta se identifică printr-un id unic care îi corespunde unui utilizator, făcând legătura cu tabelul utilizator(*id_student*).

- **Materie**: tabel care reține informațiile despre materiile care pot fi predate în cadrul universității(atribuele fiind *numele* materiei, o descriere scurtă a acesteia și *anul* în care este predată).

- **Predare**: se rețin profesorii care predau un anumit curs (atributele fiind *id_materie* prin care se identifica materia și face legătura cu tabelul Materie și *id_profesor* prin care se identifica profesorul și se face legătura cu tabelul Profesor). Relația este identificată printr-un id unic(*id_predare*).

- **Activitate**: se reține tipuri de activități predefinite (atributele sunt *denumire activitate didactica,* care poate fi laborator, seminar sau curs și *denumire evaluare,* care reprezintă tipul de evaluare pe care îl are fiecare activitate didactică, care poate fi colocviu, test seminar sau examen). Activitățile se identifică prin id-uri unice(*id_activitate*).

- **Condiție**: se reține procentajul pentru fiecare activitate pe care o susține un profesor la cursul pe care în predă(atributele sunt *id_predare*, care face legătura în mod indirect cu tabelul Materie și cu tabelul Profesor, *procenta*j, care reține procentajul ales pentru activitatea respectivă și *id_activitate*, care face legătura cu tabelul Activități pentru a ști activitatea la care se setează ponderea). O astfel de condiție de forma curs-activitate-procentaj se identifică după id-ul unic *id_conditie*.

- **Calendar**: tabel care reține programările activităților didactice, legătură care se face prin atributul *id_conditie*, precum și ale evaluărilor la acestea. Activitățile didactice au loc recursiv între *o dată de începere* și *o dată de finalizare*, odată la un *număr de zile* setat de profesor (de obicei 7 sau 14), iar în cazul evaluării, data de începere corespunde datei de finalizare și atributul *evaluare* va avea valoarea true. Pentru fiecare înregistrare de acest tip se va reține și *numărul maxim de studen*ți care pot participa O astfel de programare se identifică printr-un id unic(*id_programare*).

- **Catalog**: tabel care reține notele pe care le primește un student la o anumită activitate(atributele fiind *id_student* care face legătura cu tabelul Student, *id_condiție* care face legătura cu tabelul Condiție și valoarea notei.). Fiecare înregistrare din catalog se identifică printr-un id unic(*id_catalog*).

- **Înscriere Materie**: reține materiile la care se înscrie un anumit student(atributele fiind *id_student* care face legătura cu tabelul Student și *id_predare* care face indirect legătura cu tabelul materie și Profesor, deoarece studentul este asignat automat profesorului cu cei mai puțini studenți la momentul respectiv). O astfel de înscriere se identifică printr-un id unic(*id_înscriere_materie*).

- **Înscriere Activitate:** se reține activitățile didactice sau evaluarea la care se înscrie studentul(atributele sunt *id_student*, care face legătura cu tabelul Student și astfel se identifică studentul care se înscrie, și *id_programare*, care face legătura cu tabelul Calendar pentru a cunoaște detaliile activității). Fiecare astfel de înscriere se identifică printr-un id unic(*id_înscriere_materie*).

- **Grupuri de studiu:** reține grupurile de studiu create de studenți(atributele sunt *numele* grupului și *id_materie*, care reprezintă materia pentru care se face grupul de studiu și face legătura cu tabelul Materie). Grupurile de studiu sunt identificate printr-un id unic(id_grup_studiu).

- **Membrii Grup:** reține membrii unui grup de studiu(atributele sunt *id_grup_studiu* prin care se face legătura cu tabelul Grup de studiu și *id_student* care face legătura cu tabelul Student). Fiecare înscriere în grup a unui student se identifică printr-un id unic(id_membrii_grup).

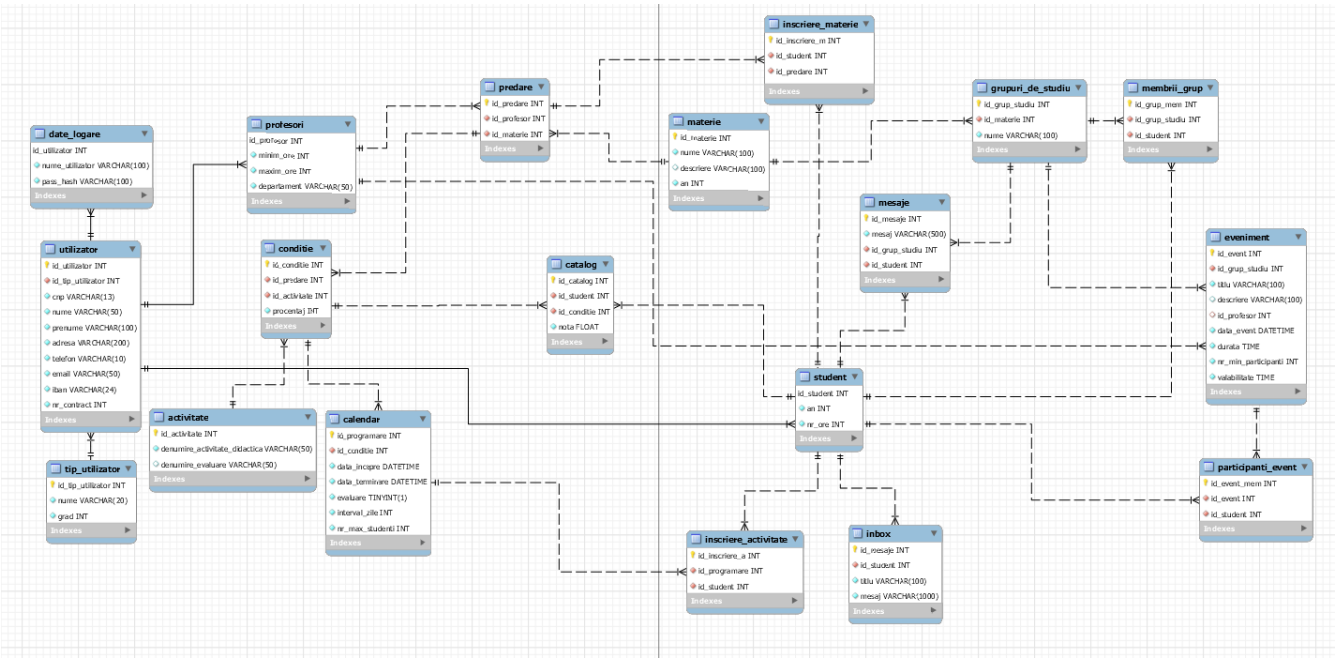
- **Mesaje:** reține mesajele postate de studenți pe un anumit grup(atributele fiind *id_student* care face legătura cu tabelul Student, *id_grup* care face legătura cu tabelul Grup de studiu și *mesajul* propriu-zis). Fiecare mesaj postat într-un grup de studiu este identificat printr-un id unic(*id_mesaj*).

- **Event:** reține evenimentele extracurriculare care se crează în cadrul unui grup de studiu(atributele fiind *id_grup_studiu* care face legptura cu tabelul Grup de studiu, tilul și descrierea evenimentului, *id_profesor*, care reprezintă profesorul ce poate asista la activitatea extracurriculara și face legătura cu tabelul Profesor, *data* la care va avea loc și *durata* activității, precum și *numărul minim* de participanți și *valabilitatea* evenimentului, interval în care trebuie ca studenții să se înscrie). O astfel de activitate extracurriculară se identifică printr-un id unic(*id_event*).

- **Participanți event:** tabel care reține participanții la o anumită activitate extracurriculară(atributele fiind *id_event* care face legătura cu tabelul Event și *id_student* care face legătura cu tabelul Student). O înscriere a unui student la un eveniment de acest tip se identifică printr-un id unic(*id_event_mem*).

- **Inbox:** reține mesajele pe care le primește un utilizator, în special mesajele care alertează anularea unui eveniment dintr-un grup de studiu, în momentul în care nu s-au strâns destui membrii care să participe(atributele sunt *id_student*, care reprezintă studentul care primește un astfel de mesaj și care face legătura cu tabelul Student, *titlul* mesajului și *mesajul* propriu-zis). Un astfel de mesaj transmis se identifică printr-un id unic(*id_mesaj*).

3.2. Diagrama EER/UML pentru modelul de date complet



3.3. Proceduri/view-uri/triggere/event-uri

Proceduri:

1) admin_search_user (nume varchar(50), prenume varchar(100), tip varchar(20))
Caută un user.
Nume – numele utilizatorului
Prenume – prenumele utilizatorului
Tip – tipul de utilizator

2) admin_search_courses(course_name varchar(100))
Caută un curs.
Course_name – numele cursului

3) admin_professors_by_course(course_id varchar(100))
Caută un profesor după curs.
Course_id – id-ul cursului căutat

4) admin_professors_by_not_course(course_id varchar(100))
Caută un profesor care nu predă cursul respectiv.
Course_id – id-ul cursului

5) admin_students_by_course(course_id varchar(100))
Caută un student după curs.
Course_id – id-ul cursului

6) student_joinable_courses(student_id varchar(255))
Selectează cursurile la care se poate înscrie un student.
Student_id – id-ul studentului

7) student_selected_courses(student_id varchar(255))
Selectează cursurile alese de student.
Student_id – id-ul studentului

8) student_join_cours(student_id varchar(255), curs_id varchar(255))
Înscrierea la un curs.
Student_id – id-ul studentului
Curs_id – id-ul cursului

9) student_drop_cours(student_id varchar(255), curs_id varchar(255))
Renunțarea la un curs.
Student_id – id-ul studentului
Curs_id – id-ul cursului

10) student_get_note(student_id varchar(255))
Studentul își poate vedea notele.
Student_id – id-ul studentului

11) profesor_assigned_courses(profesor_id varchar(255))
Selectează cursurile la care un profesor a fost asignat.
Profesor_id – id-ul profesorului

12) profesor_activities(profesor_id varchar(255), course_id varchar(255))
Selectează activitățile unui profesor pentru un anumit curs pe care il predă.
Profesor_id – id-ul profesorului
Course_id – id-ul cursului

13) profesor_students_by_course(profesor_id varchar(255), course_id varchar(255))
Profesorul vede studenții înscriși la un anumit curs.

Profesor_id – id-ul profesorului
Course_id – id-ul cursului

14) profesor_programare_activitati (id_conditie int, data_incepere datetime, data_terminare datetime, evaluare bool, interval_zile int, nr_studenti int)

Profesorul poate programa anumite activități pentru o materie pe care o predă.
Id_condiție – id-ul unei activități predate de un anumit profesor după condițiile alese de el
Data_incepere – data la care începe activitatea
Data_terminare – data la care se încheie activitatea
Evaluare – astfel profesorul decide dacă o activitate presupune și o modalitate de evaluare
Interval_zile – cât timp durează
Nr_studenti – numărul de studenți participanți

15) profesor_calendar(profesor_id varchar(255))

Profesorul își poate vedea calendarul.
Profesor_id – id-ul profesorului

16) profesor_calendar_curent(profesor_id varchar(255))

Profesorul își poate vedea doar activitățile curente din calendar.
Profesor_id – id-ul profesorului

17) student_calendar(student_id varchar(255))

Studentul își poate vedea calendarul.
Student_id – id-ul studentului

18) student_calendar_curent(student_id varchar(255))

Studentul își poate vedea doar activitățile curente din calendar.
Student_id – id-ul studentului

19) student_calendar_check_collision(student_id varchar(255), programare_id varchar(255))

Se verifică dacă se suprapun anumite activități programate.
Student_id – id-ul studentului
Programare_id – id-ul unei activități programate în calendar

20) student_enroll_activity(student_id varchar(255), programare_id varchar(255))

Studentul se înscrie la o activitate.
Student_id – id-ul studentului
Programare_id – id-ul unei activități programate în calendar

21) student_my_study_groups(student_id varchar(255))

Studentul își vede grupurile de studiu.
Student_id – id-ul studentului

22) student_not_my_study_groups(student_id varchar(255))

Studentul vede grupurile de studiu în care nu este înscris.

Student_id – id-ul studentului

23) student_group_members(student_id varchar(255), group_id varchar(255))
Studentul vede membrii grupului de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

24) student_group_suggestions(student_id varchar(255), group_id varchar(255))
Studentul vede sugestii de grupuri de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

25) student_group_messages(student_id varchar(255), group_id varchar(255))
Studentul vede mesajele trimise pe grupul de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

26) student_group_add_msg(student_id varchar(255), group_id varchar(255), msg varchar(2056))
Studentul poate trimite un mesaj pe un grup de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu
Msg – mesajul pe care vrea să îl trimită
27) student_inbox(student_id varchar(255))
Studentul își vede mesajele din inbox.
Student_id – id-ul studentului

28) student_join_group(student_id varchar(255), group_id varchar(255))
Studentul se alătură unui grup de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

29) student_exit_group(student_id varchar(255), group_id varchar(255))
Studentul iese dintr-un grup de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

30) student_create_group(student_id varchar(255), materie_id varchar(255), nume varchar(255))
Studentul poate crea un grup de studiu.
Student_id – id-ul studentului
Materie_id – id-ul materiei pentru care se face grupul de studiu
Nume – numele grupului

31) student_delete_group(student_id varchar(255), group_id varchar(255))
Studentul poate șterge un grup de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

32) student_get_group_events(student_id varchar(255), group_id varchar(255))
Studentul poate vedea evenimentele din grupul de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

33) student_get_group_professors(student_id varchar(255), group_id varchar(255))
Studentul poate vedea profesorii din grupul de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu

34) student_create_event(student_id varchar(255), group_id varchar(255), titlu varchar(255), descriere varchar(255), id_profesor int, data_event varchar(255), durata varchar(255), nr_min_participanti varchar(255), valabilitate varchar(255))
Studentul poate crea un eveniment în cadrul grupului de studiu.
Student_id – id-ul studentului
Group_id – id-ul grupului de studiu
Titlu – titlul evenimentului
Descrierea – descrierea evenimentului
Id_ profesor – id-ul profesorului
Data_event – data evenimentului
Durata – durata evenimentului
Nr_min_participanți – numărul de participanți de la eveniment
Valabilitate – cât timp este valabil evenimentul

35) student_join_event(student_id varchar(255), group_id varchar(255), event_id varchar(255))
Studentul se poate alătura unui eveniment
Student_id – id-ul studentului
Group_id – id-ul grupului
Event_id – id-ul evenimentului

36) getActivities(idStudent varchar(20), idMaterie varchar(20), tip_activitate varchar(100))
Afișează activitățile ce țin de o anumită materie.
idStudent – id-ul studentului
idMaterie – id-ul materiei

Tip_activitate – tipul de activitate

37) getStudentActivities(idStudent varchar(20))
Afișează activitățile unui student.
idStudent – id-ul studentului

38) getStudentGroupActivities(studentId varchar(20))
Afișează activitățile unui student în cadrul unui grup de studiu.
studentId – id-ul studentului

Triggere:
1) anulare_eveniment – anularea unui eveniment
2) nr_contract_lene – generează numărul de contract

View-uri:
1) vw_calendar_viitor – view pentru calendar
2) vw_catalog – view pentru catalog

Event-uri:
1) verificare_evenimente – event care updatează timpul de valabilitate rămas al unui eveniment dintr-un grup de studiu

3.4. Normalizarea datelor

Normalizare bazelor de date este un proces de optimizare a bazei de date prin care se încearcă minimizarea redundanței datelor și a anomaliilor de adăugare, actualizare și stergere. O relație aflată pe un anumit nivel de normalizare satisface toate restricțiile cerute de nivelurile inferioare.

Așadar, o schemă de baze de date relațională se află în prima formă normală FN1 daca și numai dacă toate atributele sale iau numai valori atomice. Apoi, o relație R este în a doua formă normală FN2 dacă este în FN1 și orice atribut neprim (nu este cheie/parte de cheie) este total dependent față de oricare cheie a relației. O relație este în a treia formă normală FN3 dacă este în FN2 și nici un atribut neprim nu este dependent funcțional de un alt atribut neprim. În final, se spune că R este în formă normală Boyce-Codd dacă și numai dacă oricare ar fi o dependență netrivială $X \rightarrow Y$ din F, atunci X este supercheie pentru R.

Observăm că tabelele bazei de date utilizate în cadrul proiectului respectă FNBC (forma normală Boyce-Codd). Atributele fiecărui tabel nu depind de alte atribute. Fiecare tabel are o singură cheie primară după care sunt identificate înregistrările și este suficientă pentru a identifica în mod unic orice înregistrare din baza de date. În fiecare tabel avem doar o cheie și toate dependențele au în partea stângă o supercheie (cheia primară a tabelei).

De exemplu, pentru tabela Materie avem cheia primară *id_materie* și toate dependențele au în stânga această supercheie, în tabelul Utilizator această supercheie este *id_utilizator*.

Demonstrație:

utilizator = id_utilizator, id_tip_utilizator, cnp, nume, prenume, adresa, telefon, email, iban, nr_contract

DF = { id_utilizator -> id_tip_utilizator, id_utilizator -> cnp, id_utilizator -> nume, id_utilizator -> prenume, id_utilizator -> prenume, id_utilizator -> adresa, id_utilizator -> telefon, id_utilizator -> email, id_utilizator -> iban, id_utilizator -> nr_contract }

În stânga dependențelor funcționale netriviale apare id_utilizator.

id_utilizator este o supercheie => utilizator se află în BCNF.

utilizator se află în 3NF pentru că se află în BCNF.

materie = id_materie, nume, descriere, an

DF = { id_materie -> nume, id_materie -> nume, id_materie -> descriere, id_materie -> an }

În stânga dependențelor funcționale netriviale apare id_materie.

id_materie este o supercheie => materie se află în BCNF.

materie se află în 3NF pentru că se află în BCNF.

Identic cu modelele prezentate mai sus, se poate demonstra că totalitatea tabelelor se încadreaza în BCNF, deci și în 3NF.

3.5. Interogări MySQL

```
select * from date_logare where nume_utilizator = username and pass_hash = pass;
```

Selectează id-ul utilizatorului, usernameul și parola tuplei din tabelul date_logare care să aibă usernameul și parola egale cu parametrii username, respectiv pass. Este o interogare folosita în procedura apelată într-o metodă Java.

Translatată în algebra relațională:

π*(date_logare)σnume_utiilizator = username, pass_hash = password(date_logare)

select * from tip_utilizator;

Selectează toate tuplele tabelei tip_utilizator.

Translatată în algebra relațională:

σ*(tip_utilizator)

select id_predare from predare where id_profesor = id_prof and id_materie = id_mat;

Selectează id-ul de predare din tabela predare care să aibă id-ul profesorului și id-ul materiei egale cu parametrii id_prof și id_mat. Este o interogare folosită în procedura apelată într-o metodă Java.

Translatată în algebra relațională:

πid_predareσid_profesor = id_prof, id_materie = id_mat(predare)

3.6. Cod MySQL

Codul pentru crearea bazei de date și a tabelelor, dar și cel pentru inserarea de date în tabele și crearea de chei străine se află în Anexă.

3.6.1. Cod proceduri

1) Procedură prin care administratorul caută un utilizator

```
drop procedure if exists admin_search_user;
delimiter //
create procedure admin_search_user (nume varchar(50), prenume varchar(100), tip varchar(20))
begin
        set @nume           = nullif(nume, '');
        set @prenume        = nullif(prenume, '');
        set @tip_id         = nullif(tip, '');

    if @tip_id is not null then
            if @tip_id = 1 then

                    select u.*, t.nume as tip, t.grad as grad  from utilizator u
                    inner join tip_utilizator t on t.id_tip_utilizator = u.id_tip_utilizator
```

```
                where (u.nume = @nume or @nume is null)
                and (u.prenume = @prenume or @prenume is null)
                and (u.id_tip_utilizator = @tip_id or @tip_id is null);

    elseif @tip_id = 2 then

                select u.*, t.nume as tip, t.grad as grad  from utilizator u
                inner join tip_utilizator t on t.id_tip_utilizator = u.id_tip_utilizator
                where (u.nume = @nume or @nume is null)
                and (u.prenume = @prenume or @prenume is null)
                and (u.id_tip_utilizator = @tip_id or @tip_id is null);

            elseif @tip_id = 3 then

                select tbl.*, p.* from (
                        select u.*, t.nume as tip, t.grad as grad  from utilizator u
                        inner join tip_utilizator t on t.id_tip_utilizator = u.id_tip_utilizator
                        where (u.nume = @nume or @nume is null)
                        and (u.prenume = @prenume or @prenume is null)
                        and (u.id_tip_utilizator = @tip_id or @tip_id is null)
                ) tbl
                inner join profesori p on p.id_profesor = tbl.id_utilizator;

            elseif @tip_id = 4 then

                select tbl.*, s.* from (
                        select u.*, t.nume as tip, t.grad as grad  from utilizator u
                        inner join tip_utilizator t on t.id_tip_utilizator = u.id_tip_utilizator
                        where (u.nume = @nume or @nume is null)
                        and (u.prenume = @prenume or @prenume is null)
                        and (u.id_tip_utilizator = @tip_id or @tip_id is null)
                ) tbl
                inner join student s on s.id_student = tbl.id_utilizator;

      end if;
   end if;
end//
delimiter ;
```

2) Procedură prin care administratorul caută un curs

```
drop procedure if exists admin_search_courses;
delimiter //
create procedure admin_search_courses(course_name varchar(100))
```

```
begin
        set @cname = nullif(course_name, '');

   select * from materie where nume = @cname or @cname is null;
end//
```

3) Procedură prin care profesorul poate vedea cursurile la care a fost asignat

```
drop procedure if exists profesor_assigned_courses;
delimiter //
create procedure profesor_assigned_courses(profesor_id varchar(255))
begin
        set @p_id = nullif(profesor_id, '');

        select m.* from predare p
        inner join materie m on m.id_materie = p.id_materie
        where p.id_profesor = @p_id;
end; //
```

4) Procedură prin care profesorul poate programa activități

```
drop procedure if exists profesor_programare_activitati;
delimiter //
create procedure profesor_programare_activitati (id_conditie int, data_incepere datetime,
data_terminare datetime, evaluare bool,
interval_zile int, nr_studenti int)
begin
   set @id_conditie = id_conditie;
   set @data_incepere = data_incepere;
   set @data_terminare = data_terminare;
   set @evaluare = evaluare;
   set @nr_studenti = nr_studenti;

   set @id_conditie_valid = if(exists (select id_conditie from conditie where conditie.id_conditie
= @id_conditie), 1, 0);
   if @id_conditie_valid = 1 then
                insert into calendar
      values(0, @id_conditie, @data_incepere, @data_terminare, @evaluare, @nr_studenti);
        else
                set @mesaj = 'Eroare la programare activitate';
                signal sqlstate '45000' set message_text = @mesaj;
        end if;
```

5) **Procedură prin care studentul își poate vedea calendarul**

```
drop procedure if exists student_calendar;
delimiter //
create procedure student_calendar(student_id varchar(255))
begin
        set @s_id = nullif(student_id, '');

   select tbl.*, case when ia.id_student is not null then 'Inscris' else 'Neinscris' end as inscris
        from inscriere_materie im
        inner join (
                select vw.*
      from vw_calendar_viitor vw
        ) tbl on tbl.id_predare = im.id_predare
        left join inscriere_activitate ia on ia.id_student = im.id_student and ia.id_programare = tbl.id_programare
        where im.id_student = @s_id;
end; //
```

6) **Procedură prin care studentul își poate vedea grupurile de studiu**

```
drop procedure if exists student_my_study_groups;
delimiter //
create procedure student_my_study_groups(student_id varchar(255))
begin
        set @s_id = nullif(student_id, '');

   select gs.*, m.nume as nume_curs
        from inscriere_materie im
        inner join predare p on p.id_predare = im.id_predare
   inner join materie m on m.id_materie = p.id_materie
        inner join grupuri_de_studiu gs on gs.id_materie = p.id_materie
        where im.id_student = @s_id
   and gs.id_grup_studiu in (
                select mg.id_grup_studiu
      from membrii_grup mg
      where mg.id_student = @s_id
   );
end; //
```

3.6.2. Cod triggere

1) Trigger pentru anularea unui eveniment

```
drop trigger if exists anulare_eveniment;
delimiter //
create trigger anulare_eveniment
before delete on participanti_event
for each row
begin
        set @titlu = "filler";
   set @nume = "filler";
   select e.titlu, gs.nume into @titlu, @nume
   from eveniment e
   inner join grupuri_de_studiu gs on gs.id_grup_studiu = e.id_grup_studiu
        where e.id_event = old.id_event;

   insert into inbox values(0, old.id_student, "Eveniment anulat", concat("Evenimentul '", @titlu,
"' din grupul '", @nume, "' a fost anulat!"));
end//
delimiter ;
```

2) Trigger pentru generarea numărului de contract

```
drop trigger if exists nr_contract_lene;
delimiter //
create trigger nr_contract_lene
before insert on utilizator
for each row
begin
   select max(nr_contract) into @v from utilizator;

   if(@v is null) then
                set @v = 0;
   end if;

   set new.nr_contract = @v + 1;
end//
delimiter ;
```

3.6.3. Cod pentru view/event

1) View calendar

```
drop view if exists vw_calendar_viitor;
delimiter //
create view vw_calendar_viitor as
	select cal.*, a.interval_zile, cnd.id_predare, cnd.id_activitate, p.id_profesor, p.id_materie, m.nume, m.descriere, m.an, a.denumire_activitate_didactica, a.denumire_evaluare, ifnull(count(ia.id_student), 0) as nr_studenti
   from calendar cal
	inner join conditie cnd on cnd.id_conditie = cal.id_conditie
	inner join predare p on p.id_predare = cnd.id_predare
	inner join materie m on m.id_materie = p.id_materie
	inner join activitate a on a.id_activitate = cnd.id_activitate
   left join inscriere_activitate ia on ia.id_programare = cal.id_programare
	where now() <= cal.data_terminare
   group by cal.id_programare;
//
```

2) View catalog

```
drop view if exists vw_catalog;
delimiter //
create view vw_catalog as
	select p.id_profesor, concat(prof.nume, ' ', prof.prenume) as profesor, im.id_student, concat(u.nume, ' ', u.prenume) as student, m.id_materie, m.nume as materie, im.id_predare, cnd.id_activitate, a.denumire_activitate_didactica as activitate, c.nota
	from inscriere_materie im
	inner join utilizator u on u.id_utilizator = im.id_student
	inner join predare p on p.id_predare = im.id_predare
	inner join utilizator prof on prof.id_utilizator = p.id_profesor
	inner join materie m on m.id_materie = p.id_materie
	inner join (
		select id_predare, count(id_activitate) as nr_note_necesar
		from conditie
		group by id_predare
	) cnr on cnr.id_predare = p.id_predare
	inner join conditie cnd on cnd.id_predare = im.id_predare
	inner join activitate a on a.id_activitate = cnd.id_activitate
	inner join catalog c on c.id_conditie = cnd.id_conditie;
//
```

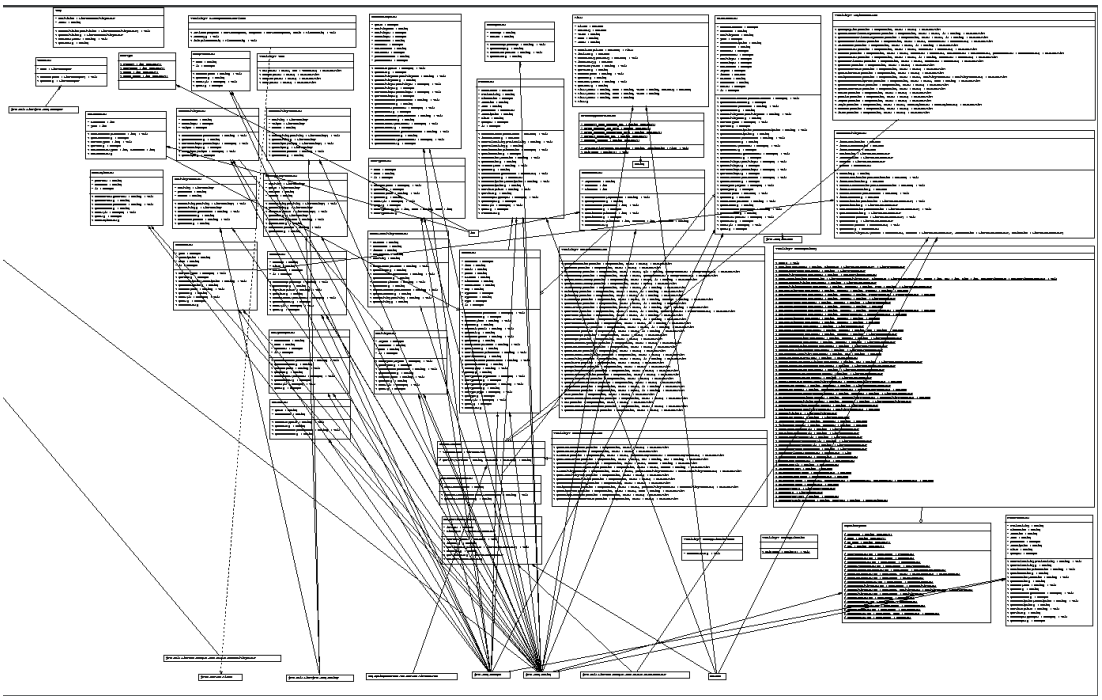
3) Verificarea evenimentelor

```
drop event if exists verificare_evenimente;
delimiter //
CREATE EVENT verificare_evenimente
ON SCHEDULE EVERY 1 MINUTE
STARTS CURRENT_TIMESTAMP
ENDS CURRENT_TIMESTAMP + INTERVAL 1 year
DO
begin
        update eveniment
   set valabilitate = valabilitate - interval 1 minute
   where valabilitate > time("00:00:00");

   delete from participanti_event
   where id_event in (
                select tbl.id_event from (
                        select e.* from
                        eveniment e
                        left join participanti_event pe on pe.id_event = e.id_event
                        where e.valabilitate <= time("00:00:00")
                        group by e.id_event
                        having count(pe.id_student) < e.nr_min_participanti
                ) tbl
   );

   delete from eveniment
   where id_event in (
                select tbl.id_event from (
                        select e.* from
                        eveniment e
                        left join participanti_event pe on pe.id_event = e.id_event
                        where e.valabilitate <= time("00:00:00")
                        group by e.id_event
                        having count(pe.id_student) < e.nr_min_participanti
                ) tbl
   );
end//
delimiter ;
```

4. Detalii de implementare

4.1. Structura de clase în Java. Diagrama UML

Diagrama UML:



Clasele din Java sunt grupate în:

- **Repository**(gestiunea bazei de date)**:**
  - **UserRepository**: gestiunea bazei de date și a informațiilor legate de utilizatori
  - **StudentRepository**: gestiunea datelor din baza de date ce privesc utilizatorii de tip student
  - **ProfessorRepository**: gestiunea informațiilor din baza de date ce țin de utilizatorii de tip profesor

- **Models**(clase care facilitează transmiterea informațiilor dinspre Repository către interfața vizuală):

- **UserModel**: poate fi populat cu informațiile comune tuturor utilizatorilor din tabela *utilizatori*

```
private Integer id;
private Integer type;
private String typeName;
private Integer typeRank;
private String cnp;
private String secondName;
private String firstName;
private String address;
private String phone;
private String email;
private String iban;
private Integer contract;
```

- **ProfessorModel/StudentModel**: pot fi populate informațiile specifice tipului de utilizator profesor, respectiv student, care diferă de orice alt tip de utilizator

  - ProfessorModel:

```
private int minHours;
private int maxHours;
private String department;
```

  - StudentModel:

```
private int year;
private int classesNb;
```

- **CalendarModel**: poate fi populat cu o înregistrare din tabela *calendar*

```
private Integer id;
private Integer condId;
private String startDate;
private String endDate;
private Boolean isExam;
private Integer daysNb;
private Integer studentsNb;
private Integer teachingId;
private Integer activityId;
private Integer professorId;
private Integer courseId;
private String courseName;
private String courseDescription;
private Integer year;
private String activityName;
private String examName;
private Integer currStudentsNb;
```

- **MessageModel**: poate fi populat cu înregistrări din tabela *mesaje*

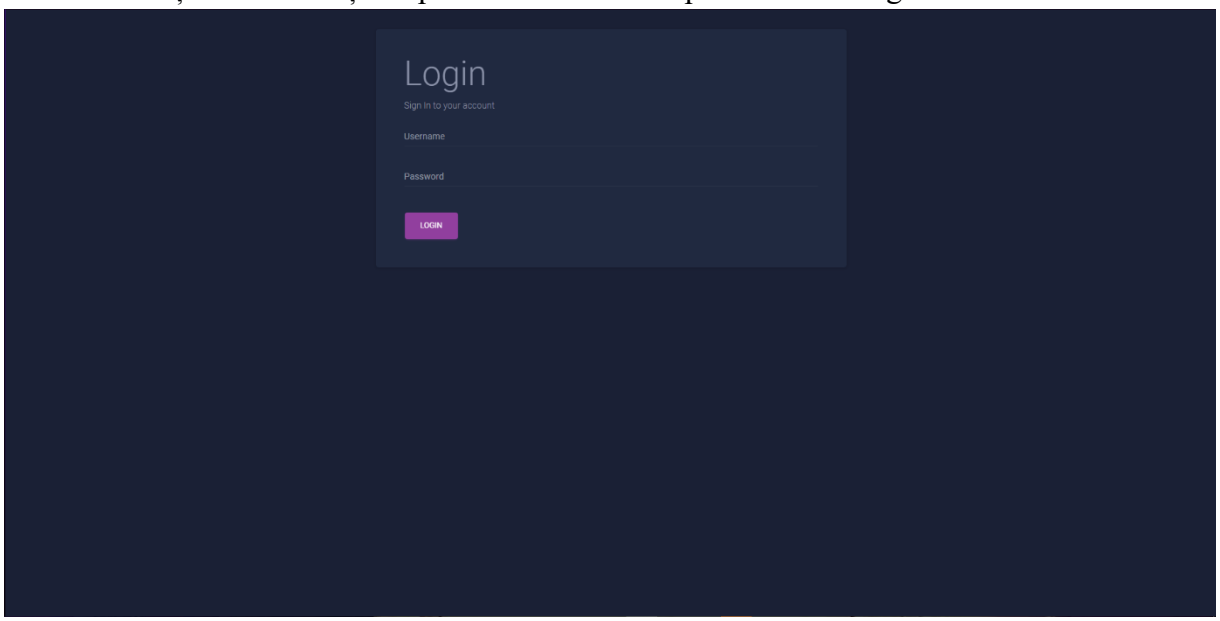
```
private String sender;
private String message;
```

- **StudyGroupModel**: poate fi populat cu înregistrări din tabela *grup_de_studiu*

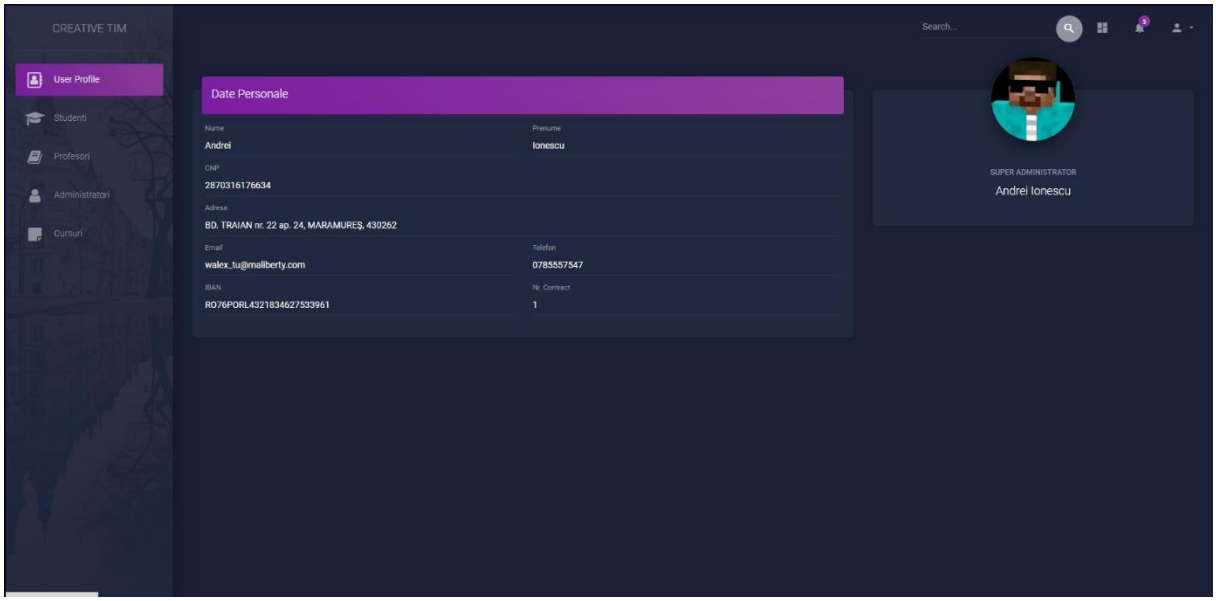
```
private Integer id;
private Integer courseId;
private String name;
```

```
private String courseName;
```

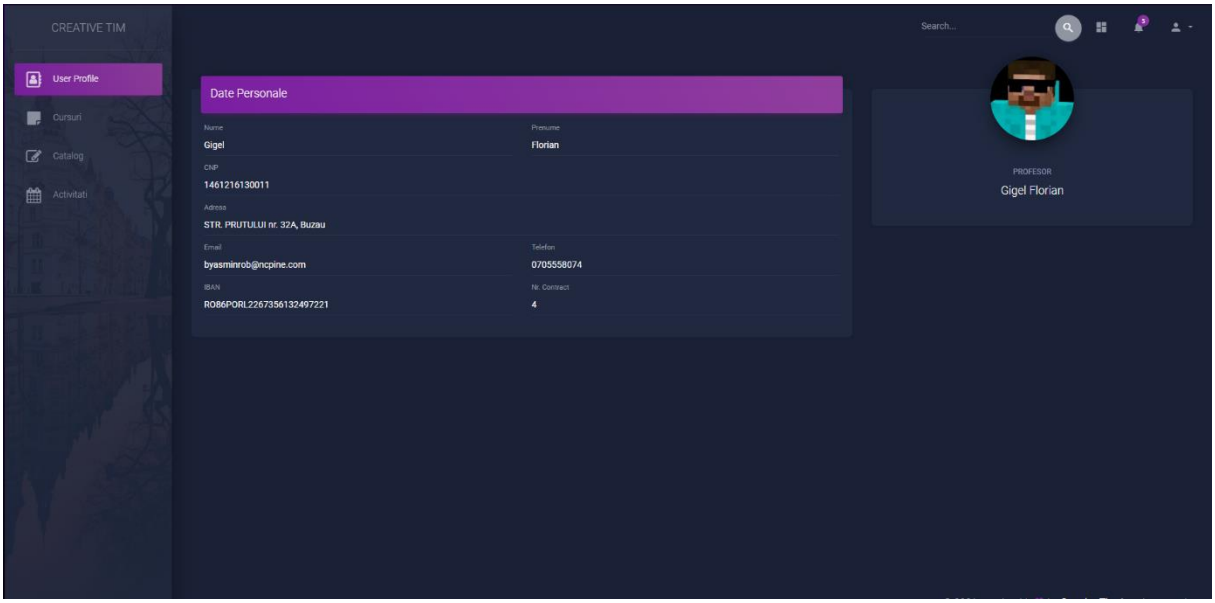
- **Controller**(gestiunea interfeței grafice și a afișării datelor extrase și filtrate din baza de date):
    - **LoginController**: gestiunea logării în aplicație a utilizatorilor de orice tip și afișarea informațiilor personale imediat după momentul logării



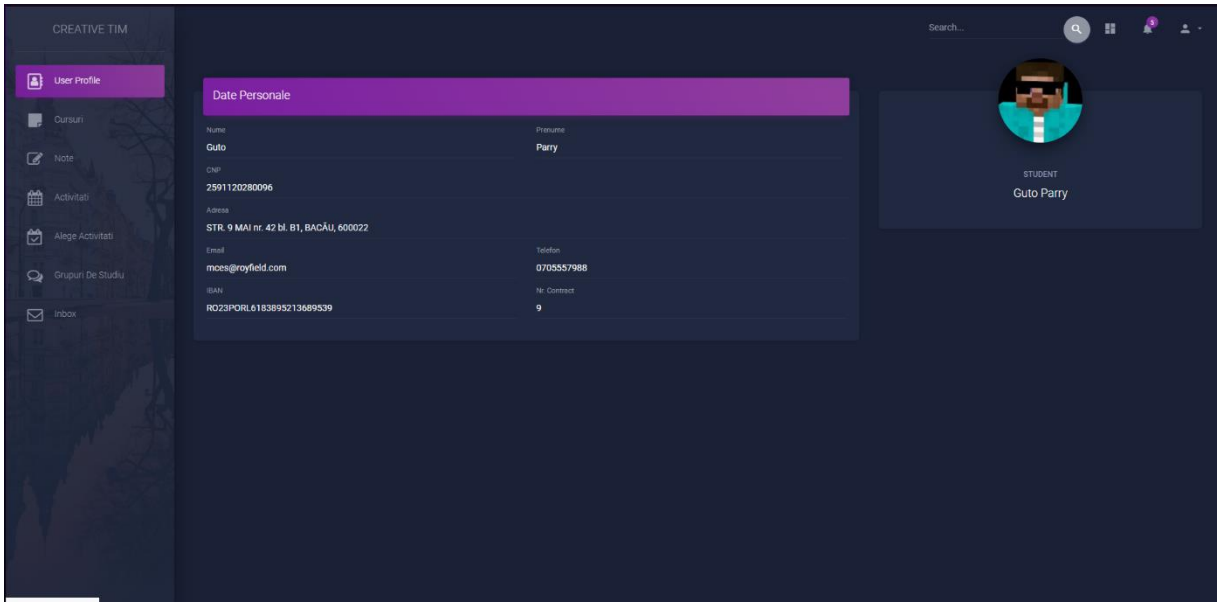
    - **AdminController**: gestionarea interfeței grafice pentru utilizatorii de tip admin și super_admin precum și a acțiunilor pe care le pot realiza aceștia



- **ProfessorController**: gestionarea interfeței grafice pentru utilizatorii de tip profesor precum și a acțiunilor pe care le pot realiza aceștia



- **StudentController**: gestionarea interfeței grafice pentru utilizatorii de tip student precum și a acțiunilor pe care le pot realiza aceștia

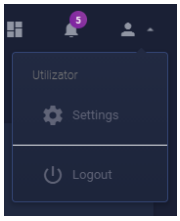


4.2. Manual de utilizare/instalare

Interfața realizată pentru interacțiunea utilizatorului cu aplicația este una simplă și intuitivă, prin intermediul căreia acesta poate să efectueze diverse operații asupra bazei de date. Interfața conține 12 ferestre(User Profile, Studenți, Profesori, Administratori, Cursuri, Catalog, Activitati, Înscriere Activități, Grupuri de studiu, Inbox, Note) care sunt filtrate pentru fiecare utilizator în funcție de rolul pe care acesta îl deține în cadrul aplicației: 4 ferestre pentru super_admin, 3 ferestre pentru admin, 4 ferestre pentru profesor și 7 ferestre pentru student.

La deschiderea aplicației trebuie introduse username-ul și parola contului care trebuie să se afle în baza de date a aplicației. În cazul unei combinații greșite de date de logare, va apărea un mesaj de eroare. În urma autentificării cu succes în aplicație, utilizatorul va fi redirecționat pe pagina cu profilul utilizatorului, unde își va putea vizualiza datele personale, fără a le putea modifica. În partea stângă a ferestrei avem meniul de acțiuni pe care acesta le poate efectua, în funcție de rolul acestuia, care face legătura cu restul ferestrelor.

Delogarea se poate efectua apăsând pe pictograma din stânga sus a ecranului care va deschide un dropdown ce conține opțiunile Settings și Logout. Pentru delogare se va efectua click pe butonul Logout, aplicația putând fi ulterior utilizată de un alt utilizator.



4.3. Elemente de securitate a aplicației

Aplicația prezintă un nivel de securitate în momentul conectării la baza de date, dar și în momentul logării propriu-zise.

În funcție de modul de conectare ales (super-administrator, administrator, profesor sau student), sunt disponibile următoarele acțiuni:

- Super-administrator –poate adăuga, modifica şi şterge informaţii în baza de date, informații legate de utilizatori, dar și informații legate de administratori.
- Administrator – poate poate adăuga, modifica şi şterge informaţii în baza de date, informații legate de utilizatori. Poate să caute utilizatorii după nume și îi poate filtra după tip, poate asigna profesorii la cursuri și poate face căutare după numele cursului.
- Profesor – poate adăuga și programa activități, poate gestiona ponderile notelor, poate nota studenți, poate vizualiza listele de studenți și poate descărca cataloage, poate vizualiza / descarca activitățile curente / din viitor.
- Student – poate căuta un curs, se poate înscriere la un curs, își poate vizualiza notele și grupurile de studiu, dar și membrii acestora, poate vedea mesajele trimise pe grupul de studiu și își poate vizualiza / descărca activitățile curente / din viitor.

5. Concluzii. Limitări și dezvoltări ulterioare

Având în vedere cele prezentate anterior, putem afirma că proiectul realizat este unu complex, ce înglobează cunoștințe din multiple domenii și întrunește cerințele specificate.

Rezolvarea propusă pentru acest proiect prezintă o interfață plăcută din punct de vedere vizual și ușor de folosit atât pentru profesori, cât și pentru studenți. Așadar, aceasta are un potențial crescut de dezvoltare ulterioară, constituind scheletul și o importantă parte dintr-un viitor sistem de gestiune a activităților din cadrul unei universități.

Propunerile noastre pentru viitoarele dezvoltări sunt: adăugarea altor activități didactice, în afara laboratorului, seminarului și cursului, în tabelul activități, astfel extinzându-se funcționalitățile deja implementate, realizarea unui ranking al studenților pe grupurile de studiu în functie de nivelul lor de activitate, implementarea forumurilor student-profesor, unde studenților li se răspunde la întrebări.