MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT

la disciplina

INTRODUCERE IN BAZE DE DATE

Profesor coordonator:
Cosmina Ivan

Echipa de proiect:
Damaris-Claudia Mocodean
Ana-Maria Cusco
Adriana-Teodora Fariseu
Grupa: 30225, semestrul I, 2021

An academic: 2020 – 2021

Cuprins

1.Intoducere

Dezvoltarea exponenţiala cunoscuta de tehnologia informatiei si comunicatiilor in ultima vreme a condus la inregistrarea unei adevarate revolutii in domeniul instruirii asistate de calculator. Pe fondul schimbarilor rapide si progresului tehnologic inregistrat, precum şi pe fondul tendinţei de globalizare a educaţiei universitare, s-au deschis noi perspective pentru sistemul educational universitar, aceasta fiind completat cu metode moderne de abordare a educatiei precum si legaturii cu studentii.

Una dintre aceste metode este digitalizarea sistemului universitar, iar proiectul nostru incearca o abordare in acest sens. Proiectul presupune dezvoltarea unei aplicatii ce lucreaza cu baze de date pentru managementul unei platforme de studiu: “Numele platformei”. Scopul acesteia este simplificarea operatiilor cu baza de date prin oferirea unei interfete grafice prietenoase pe care utilizatorii de diferite tipuri o pot utiliza cu usurinta. Aplicatia ofera sprijin atat pentru interogarea bazei de date cat si pentru manipularea acesteia.

Pentru dezvoltarea proiectului au fost folosite:

MySQL Workbench 8.0CE - pentru crearea bazei de date, popularea initiala, dezvoltarea de vederi,proceduri si triggere si pentru crearea diagramei UML a tabelelor
IntelliJ, Eclipse – medii de dezvoltare Java
JDBC– pentru stabilirea conexiuni aplicatiei cu baza de date
Apache PDFBox + libraria Boxable Tilman- pentru generare de PDF-uri

2.Suportul teoretic

Limbajul MySQL
MySQL este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeza MySQL AB și distribuit sub Licența Publică Generală GNU. Este cel mai popular SGBD open-source la ora actuală[6], fiind o componentă cheie a stivei LAMP (Linux, Apache, MySQL, PHP).

Deși este folosit foarte des împreună cu limbajul de programare PHP, cu MySQL se pot construi aplicații în orice limbaj major. Există multe scheme API disponibile pentru MySQL ce permit scrierea aplicațiilor în numeroase limbaje de programare pentru accesarea bazelor de date MySQL, cum ar fi: C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc., fiecare dintre acestea folosind un tip specific API. O interfață de tip ODBC denumită MyODBC permite altor limbaje de programare ce folosesc această interfață, să interacționeze cu bazele de date MySQL cum ar fi ASP sau Visual Basic. În sprijinul acestor limbaje de programare, unele companii produc componente de tip COM/COM+ sau .NET (pentru Windows) prin intermediul cărora respectivele limbaje să poată folosi acest SGBD mult mai ușor decât prin intermediul sistemului ODBC. Aceste componente pot fi gratuite (ca de exemplu MyVBQL) sau comerciale.

Licența GNU GPL nu permite încorporarea MySQL în softuri comerciale; cei care doresc să facă acest lucru pot achiziționa, contra cost, o licență comercială de la compania producătoare, MySQL AB.

MySQL este componentă integrată a platformelor LAMP sau WAMP (Linux/Windows-Apache-MySQL-PHP/Perl/Python). Popularitatea sa ca aplicație web este strâns legată de cea a PHP-ului care este adesea combinat cu MySQL și denumit Duo-ul Dinamic. În multe cărți de specialitate este precizat faptul ca MySQL este mult mai ușor de învățat și folosit decât multe din aplicațiile de gestiune a bazelor de date, ca exemplu comanda de ieșire fiind una simplă și evidentă: „exit" sau „quit".

Pentru a administra bazele de date MySQL se poate folosi modul linie de comandă sau, prin descărcare de pe internet, o interfață grafică: MySQL Administrator și MySQL Query Browser. Un alt instrument de management al acestor baze de date este aplicația gratuită, scrisă în PHP, phpMyAdmin.

MySQL poate fi rulat pe multe dintre platformele software existente: AIX, FreeBSD, GNU/Linux, Mac OS X, NetBSD, Solaris, SunOS, Windows 9x/NT/2000/XP/Vista.[2]

Limbajul Java

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems (acum filială Oracle) la începutul anilor '90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor.

Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java (engleză Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

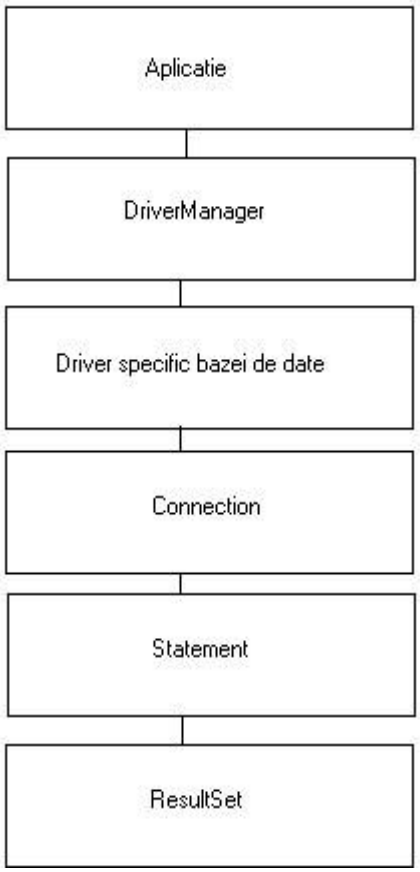
Mașina virtuală Java este mediul în care se execută programele Java. În prezent, există mai mulți furnizori de JVM, printre care Oracle, IBM, Bea, FSF. În 2006, Sun a anunțat că face disponibilă varianta sa de JVM ca open-source.[1]

JDBC (Java Database Connectivity)

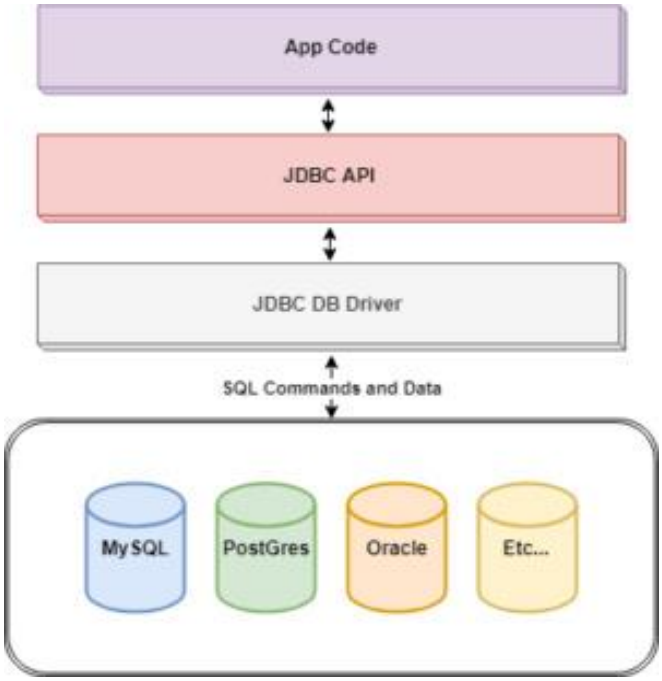
JDBC este bazat pe doua standarde: SQL (Structured Query Language) si CLI (X/Open Call Level Interface). CLI este implementat in interfata Microsoft ODBC. ODBC a fost implementat in limbajul C, deci nu se putea utiliza direct din Java deoarece s-ar fi stricat portabilitatea acestor aplicatii. O alta problema cu ODBC era ca fiind scris intr-un limbaj

procedural in care caramida de constructie este functia era mai greu atasarea la Java, care este un limbaj obiectual. Multe functii C din ODBC returneaza si pointeri void care violeaza caracteristica fundamentala a limbajului Java, siguranta tipului.
aplicatie JDBC utilizeaza unul sau mai multe drivere din pachetul java.sql care sunt utilizate de catre clasa DriverManager. Driverele sunt specifice bazelor de date, deci pentru fiecare tip de baza de date se utilizeaza un driver special. In aceeasi aplicatie putem lucra cu baze de date diferite, deci implicit si cu mai multe drivere. Putem avea nevoie de un driver care comunica cu o baza de date Oracle aflata la distanta si de un altul care reprezinta legatura spre driverul ODBC local si comunica cu un server SQL. In aplicatii comunicatia cu o baza de date necesita urmatorii pasi:

-Se apeleaza la DriverManager cerand un driver specific pentru baza de date
-Driverul specific creaza legatura cu baza de date si returneaza un obiect de tip Connection
-Cu ajutorul obiectului de tip Connection se creaza un obiect Statement care contine si o cerere SQL catre baza de date
-Obiectul Statement returneaza rezultatele intr-un obiect ResultSet [3]



[3]



[4]

3.Analiza cerintelor utilizatorilor (Specificatiile de proiect)

3.1. Ipotezele specifice domeniului ales pentru proiect (cerinte, constrangei)

Cerinta:
Se doreste implementarea unui sistem informatic destinat gestiunii unei platforme de studiu.
Aplicatia va folosi un sistem de gestiune pentru baze de date MySQL, iar interactiunea cu aceasta va fi realizata doar prin interfata grafica. Functionalitatile pe care le va oferi programul vizeaza operatii ce tin de gestiunea studentilor, profesorilor si administrarea operatiilor curente din cadrul unor programe de studiu.
Aplicația va putea fi accesată, pe baza unui proces de autentificare, de către mai multe tipuri de utilizatori: studenti, profesori, administratori. Pentru fiecare tip de utilizator se vor reține informații precum CNP, nume,prenume, adresa, număr de telefon, email, cont IBAN, numărul de contract. 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, programul trebuie să ofere 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.
Utilizatorul de tip administrator poate adăuga, modifica și șterge informații în baza de date , informații legate de utilizatori. De asemenea, va exista și un rol de super-administrator care poate opera inclusiv asupra utilizatorilor de tip administrator.
Administratorii pot sa caute utilizatorii dupa nume si ii pot filtra dupa tip, pot asigna profesorii la cursuri si pot face cautare dupa numele cursului. La cautarea unui curs se afiseaza si numele profesorilor de la acel curs si un buton care permite vizualizarea tuturor studentilor inscrisi la cursul respectiv.
Pentru un utilizator de tip profesor se vor retine si cursurile predate, numarul minim si numarul maxim de ore pe care le poate preda si departamentul din care face parte.
Pentru un utilizator de tip student se va retine si anul de studiu si numarul de ore pe care trebuie sa le sustina.
Aplicatia va permite gestiunea cu usurinta a activitatilor didactice si astfel a interacțiunilor dintre studenti si profesori. Cursurile sunt predate de mai multi profesori și au una sau mai multe tipuri de activitati (curs, seminar, laborator), o descriere, si un numar maxim de studenti participanti. Studentii se pot inscrie la cursuri si sunt asignati profesorului cu cei mai putini

studenti la data inscrierii. Acestia sunt evaluati cu note pentru fiecare tip de activitate si primesc o nota finala ca medie ponderata intre tipurile de activitati. Profesorul stabileste din interfata grafica impartirea procentuala pe tipurile de activitati (ex. 20% seminar, 35% laborator, 45% curs/examenul de la curs).
Fiecare activitate se desfasoara recursiv intre doua date, pe o anumita perioada de timp. La asignarea unui profesor la un curs se vor alege tipurile de activitati. De exemplu, profesorul X preda cursul Y cu activitatile: curs – saptamanal, laborator – saptamanal. Ulterior, profesorul poate programa activitatile (curs, seminar, laborator, colocviu, examen) intr-un calendar, pe zile si ore, specificand si numarul maxim de participanti.
Activitatile pot fi programate doar in viitor. Profesorii pot accesa un catalog, unde pot filtra studentii dupa cursuri si le pot adauga note. Cataloagele pot fi descarcate sub forma de fisier.
La logare, studentii si profesorii pot sa isi vada activitatile din ziua curenta sau pot accesa o pagina cu toate activitatile la care sunt asignati / inscrisi. Aceste liste pot fi descarcate din sistem sub forma unor fisiere.
Studentii se pot inscrie la cursuri, pot renunta la cursuri si isi pot vedea notele. Acestia trebuie sa aleaga activitatile la care vor sa participe si pot participa la ele doar daca mai sunt locuri sau nu exista o suprapunere cu o alta activitate (de exemplu, studentul doreste sa participe la laboratorul de BD marti la ora 10. Se inscrie la acea activitate, iar inscrierea este valida doar daca nu are deja o alta activitate marti la ora 10 sau daca mai sunt locuri disponibile. In caz contrar, se afiseaza un mesaj de eroare).
Totodata, studentii se pot inscrie in grupuri de studiu pentru o anumita materie, daca sunt inscrisi la materia respectiva. Acestia pot sa vada toti membrii grupului si sa lase mesaje. Pe grup, studentii pot adauga activitati si sa defineasca un numar minim de participanti si o perioada in care ceilalti studenti pot sa anunte
participarea (de exemplu, un student adauga o activitate de aprofundare a cursului pentru data de 12.12.2020, ora 16:00, 2 ore, cu numar minim de participanti 5 si timp de expirare 2 ore). Daca numarul minim nu este atins, activitatea se anuleaza, iar studentii inscrisi la ea primesc un mesaj de informare.

3.2. Organizare structurata(tabelar) a cerintelor utilizatorilor

Baza de date trebuie sa stocheze urmatoarele informatii:

- Informatiile utlizatorilor (super-admin, admini, profesori, studenti)
- Cursurile
- Profesorii asignati la un curs
- Activitatile de la fiecare materie
- Organizatorii unor anumite activitati la diversele materii
- Participantii la un curs
- Participantii la o activitate
- Grupurile de studiu
- Activitatile de pe grupurile de studiu
- Mesajele de pe grupurile de studiu
- Membrii unui grup
- Notificarile fiecarui utilizator
- Profesorii invitati la activitatile de grup

Mai mult trebuie sa permita si urmatoarele operatii:

- Adaugarea de utilizatori noi daca nu exista
- Adaugarea de cursuri daca nu exista
- Adaugarea de activitati
- Adaugarea de grupuri
- Stergerea automata a notificarilor
- Stergerea automata a activitatilor de grup
- Calcularea notei finale a unui student la un curs
- Verifica suprapunerea a doua activitati in momentul asignarii unui student la o materie
- Calcularea numarului total de partiicpanti la o activitate de grup (studenti inscrisi + profesori invitati)
- Gasirea profesorului cu numarul minim de participanti
- Atentionarea prin notificari a participantilor unei activitati atunci cand aceasta poate avea loc
- Contorizarea in timp real a timpului de anuntare al unei activitati de grup

Aplicatia Java trebuie sa permita prelucrarea informatiilor din baza de date, cautarea, stergerea, modificarea si afisarea informatiilor si conectarea utilizatorilor inregistrati.

3.3. Determinarea si caracterizarea de profiluri de utilizatori

Utilizator -autentificare/deautentificare -vizualizare date personale -vizualizare activitati curs/grup dintr-o anumita zi -vizualizare cursuri -vizualizare participanti curs -vizualizare notificari -vizualizare orar		
Student -inscriere/parasire curs -cautare curs -vizualizare note -inscriere in grupuri de studiu -creare grup de studiu -creare activitate grup de studiu -creare/editare mesaje de pe grupuri de studiu -vizualizare membrii grupuri de studiu -insciere la activitati din grupurile de studiu -vizualizare participanti grupuri de studiu	**Profesor** -creare/programare activitati -gestionare ponderi dispcipline -notare studenti -participare la activitate grup de studiu -vizualizare liste studenti -descarcare cataloage	**Admin/Super-Admin** -gestionare conturi (creare, stergere , modificare, vizualizare) -asignare profesori la curs -creare curs -creare/ programare activitati la cursuri -vizualizare/ editare/ stergere grupuri de studiu -vizualizare/stergere activitati din grupurile de studiu -stergere membrii grup de studiu -vizualizare/ postare / editare/ stergere mesaje

4. Modelul de date si descrierea acestuia

4.1 . Entitati si atributele lor

Tabelele:

Users-informatii despre utlizatori. Atribute: id (PK), CNP,username, parola, nume, prenume,adressa, telefon, email, nr_contract, IBAN, tip_user (SUPER-ADMIN, ADMIN, PROFESOR, STUDENT).

Profesor - informatii suplimentare despre un profesor pe langa cele din user. Atribute: user_id (PK, FK), min_ore, max_ore, departament. Se leaga de tabela users.

Student - informatii suplimentare despre un student pe langa cele din user. Atribute: user_id (PK, FK), an, nr_ore. Se leaga de tabela users.

Fisa postului curs - informatii despre profesorii asignati unor discipline. Atribute: id (PK), user_id (FK), ID_curs (FK). Se leaga direct de tabelele profesor, curs, organizator.

Fisa participare - informatii despre studentii inscrisi la cursuri. Atribute: id (PK), user_id (FK), ID_curs (FK). Se leaga direct de tabelele student, curs, participant_curs.

Curs- informatii despre cursuri. Atribute: ID_curs(PK), materie, an, tip (OBLIGATORIU, OPTIONAL), descriere. Se leaga de tabelele fisa_postului_curs, fisa_participare, activitate

Organizator - informatii despre organizatorii activitatilor de la diversele discipline. Atribute:profesor_principal(FK), user_id(FK), ID_activitate(FK). Se leaga de tabelele fisa_postului_curs, profesor, activitate.

Activitate- informatii despre activitati. Atribute: ID_activitate(PK), ID_curs(FK), tip (LABORATOR, SEMINAR, CURS), data_inceperii, data_incheierii, frecventa (SAPTAMANL, SAPTAMANAL-ALTERNANT), durata, nr_maxim_participanti, pondere. Se leaga de tabelele organizator, curs, profesori_invitati, participant_curs.

Participant curs- informatii despre participantii la o activitate. Atribute: id(PK), ID_activitate(FK), id_fisa_participare(FK), user_id, nota. Se leaga de tabelele fisa_participare, activitate.

Notificare - informatii despre notificari. Atribute: receiver_ID, sender_ID, sender_name, descriere, expirare. Se leaga de tabela users.

Grup- informatii despre grupurile de studiu. Atribute: ID_grup (PK), ID_curs(FK), nume, descriere. Se leaga de tabelele fisa_participare, membru_grup, activitati_grup.

Membru grup- informatii despre membrii unui grup. Atribute:id(PK), ID_grup(FK), user_id(FK). Se leaga de tabelele grup, student, participant_grup.

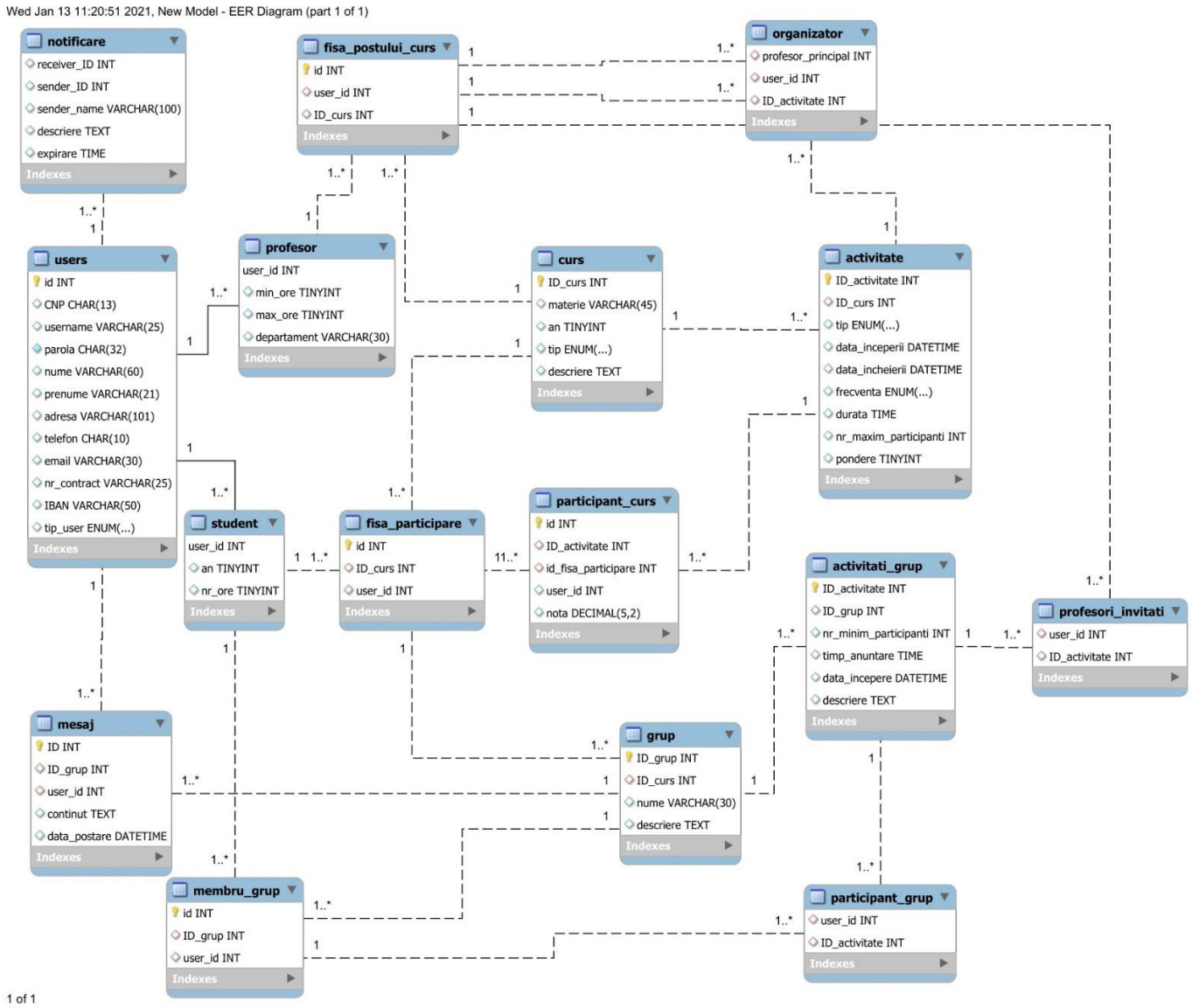
Activitati grup- informatii despre activitatile desfasurate in cadrul unui grup de studiu. Atribute:ID_activitate(PK), ID_grup(FK), nr_minim_participanti, timp_anuntare, data_incepere, descriere. Se leaga de tabelele participant_grup, profesori_invitati, grup.

Participant grup- informatii desprea participantii la o activitate programata in cadrul unui grup. Atribute: user_id(FK), ID_activitate(FK). Se leaga de tabelele membru_grup, activitati_grup.

Profesori invitati- informatii despre profesorii invitati in cadrul unei activitati de grup. Atribute: user_id(FK), ID_activitate(FK). Se leaga de tabele fisa_postului_curs, activitati_grup.

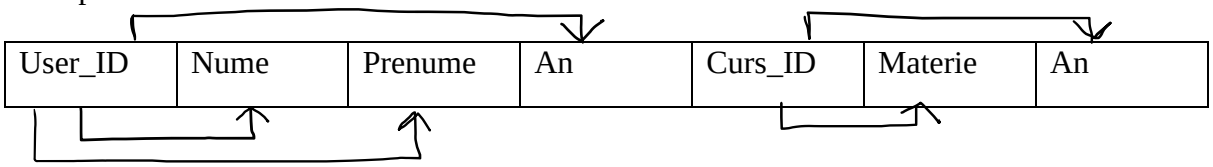
Mesaj- informatii despre mesajele postate in cadrul unui grup. Atribute: id(PK), ID_grup(FK), user_id(FK), continut, data_postare. Se leaga de tabelele users, grup.

4.2. Diagrama EER/UML pentru modelul de date complet



4.3. Normalizarea datelor

Normalizare bazelor de date este un proces de optimizare a bazei de date prin care se incearca minimizarea redundantei datelor, si a anomaliilor de introducere, actualizare si stergere.
Exemplu:



Daca tabelul nostru ar avea o structura conform figurii de mai sus, atunci acesta ar prezenta redundante in date:

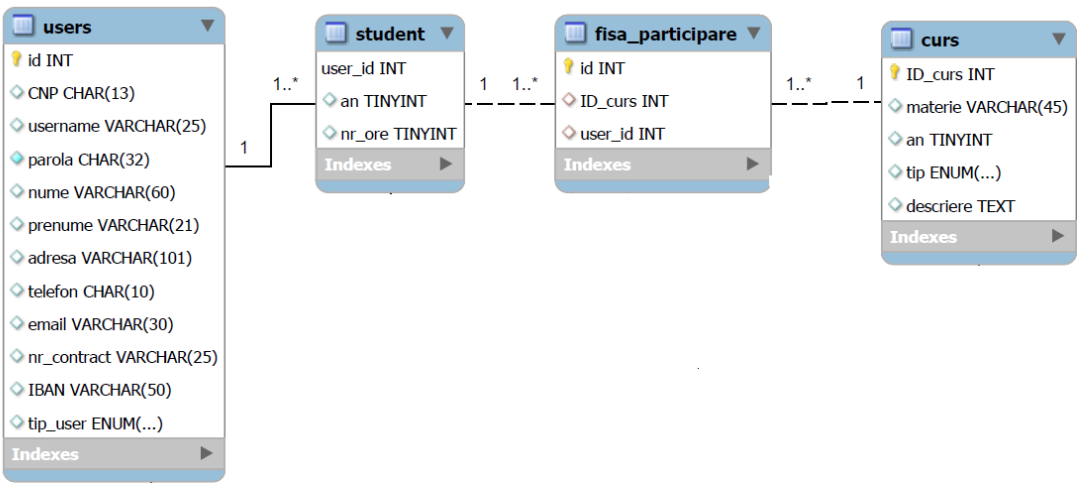
User_ID	Nume	Prenume	An	Curs_ID	Materie	An
1	Popescu	Ion	1	1	Matematici speciale	
1	Popescu	Ion	1	2	Proiectare Logica	
1	Popescu	Ion	1	3	Franceza	

! Informatia de mai sus contine date care se repeta despre student in 3 tuple. In acest caz nu am putea sa alegem User_ID ca si cheie candidat, si totodata apar anomalii de actualizare si stergere intrucat, daca am modifica datele despre un student ar trebui sa modificam toate tuplele

in care apar informatii referitoare la acel student.

De asemenea, ar putea sa apara si o anomalie de insertie in cazul introducerii in baza de date a unui student cu acelasi nume, presupunand ca am alege ca si cheie primara grupul nume-prenume.

In acest fel nu este asigurata consistenta datelor existand riscuri foarte mari de pierdere a datelor.



Solutia abordata de noi a fost foarte simpla, si presupune separarea tabelelor student si curs, pentru a tine evidenta separata a studentilor si a cursurilor, precum si rezolvarea relatiei many-to-many (un student poate avea mai multe cursuri, la cursuri pot participa mai multi studenti) prin introducerea unei tabele auxiliare fisa_participare care retine atat id-ul studentului participant la curs cat si id-ul cursului la care acesta participa, ne mai retinandu-se informatia duplicata despre studenti.

Ca urmare, intreaga noasta baza de date a fost construita pe baza acestor principii, incercand sa asiguram corectitudinea datelor prin eliminarea oricaror posibile anomalii rezultate din proiectarea defectuoasa a schemei unor relatii.

Dependente functionale - reprezinta un set de restrictii intre atributele unei relatii si anume valoare unui atribut (sau grup de atribute) poate fi determinata prin valoarea altui atribut (sau grup)

X -> Y adica X determina pe Y sau Y depinde functional de X

Baza noastra de date respectă **BCNF (forma normala Boyce-Codd)**. Atributele fiecărui tabel nu depind de alte atribute, si totodatarespecta principuiul atomicitatii. 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.

5.Detalii de implementare SQL

5.1 DDL (Data Definition Language)

- *Crearea bazei de date*

```
create database studiu;
use studiu;
```

- *Crearea unui tabel*

```
create table users(
        id int auto_increment primary key,
        CNP char(13) ,
        username VARCHAR(25) unique,
        parola char(32) not null,
        nume varchar(60),
        prenume varchar(21),
        adresa varchar(101),
        telefon char(10),
        email varchar(30),
        nr_contract VARCHAR(25),
        IBAN varchar(50),
        tip_user enum('SUPER-   ADMIN', 'ADMIN',
                'PROFESOR', 'STUDENT'));
```

- *Adaugarea constrangerilor*

```
alter table curs
add unique (materie);
```

```
alter table users
modify CNP VARCHAR(13) not null;
```

- *Adaugarea cheilor straine*

```
alter table mesaj
add constraint fk_mesaj_grup
foreign key(ID_grup)
references grup(ID_grup)
on update cascade
on delete cascade;
```

Mentiune: Atat pentru stergere cat si pentru update, am folosit optiunea de ’cascade’ pentru a ne asigura ca in momentul in care vom sterge sau modifica o anumita informatie, vom regasi aceste modificari (sau stergeri) si la nivelul atributelor din tabelelele care sunt in legatura directa, prin intermediul cheilor straine, cu informatia pe care noi dorim sa o stergem.

5.2 DML (Data Manipulation Language)

- *Inserarea datelor in tabele*

```
INSERT INTO curs(materie,an,tip)
VALUES('Programarea Calculatoarelor',1,2),
('Structuri de date si algoritmi',1,2),
('Proiectare logica',1,2),
('Proiectarea sistemelor numerice',1,2),
('Matematici speciale I',1,2);
```

- Modificarea datelor din tabele

```
UPDATE activitate,organizator SET activitate.pondere=70
WHERE organizator.profesor_principal=3 AND activitate.ID_curs=1
AND activitate.tip='CURS';
```

5.3. SQL programatic

- **Proceduri stocate**
  - Procedura pentru calcularea notei finale a unui student

```
CREATE PROCEDURE calculaeza_media(IN nota_1 DECIMAL(5,2),IN nota_2 DECIMAL(5,2),IN nota_3 DECIMAL(5,2),IN pondere_1 INT,IN pondere_2 INT,IN pondere_3 INT)
BEGIN
    IF(nota_3 IS NULL)THEN
    BEGIN
        SET @var1=(nota_1*pondere_1)/100;
        SET @var2=(nota_2*pondere_2)/100;
        SET @result=@var1+@var2;
            END;
    ELSE IF(nota_2 IS NULL)THEN
    BEGIN
        SET @var1=(nota_1*pondere_1)/100;
        SET @var3=(nota_3*pondere_3)/100;
        SET @result=@var1+@var3;
            END;
    ELSE
    BEGIN
       SET @var1=(nota_1*pondere_1)/100;
                    SET @var2=(nota_2*pondere_2)/100;
       SET @var3=(nota_3*pondere_3)/100;
                    SET @result=@var1+@var2+@var3;
            END;
           END IF;
    INSERT INTO fisa_participare(nota_finala) VALUES(@result);
    END IF;
END //
DELIMITER ;
```

  - Procedura creare activitate

```
CREATE PROCEDURE creaza_activitate(IN id_activitate INT,IN tip INT, IN data_inceperii DATETIME, IN data_incheierii DATETIME,IN frecventa VARCHAR(20), IN durata TIME, IN nr_max INT,IN pondere INT)
BEGIN
    INSERT INTO activitate(ID_activitate,tip,data_inceperii,data_incheierii,frecventa,durata,nr_max,pondere) VALUES
    (id_activitate,tip,data_inceperii,data_incheierii,frecventa,durata,nr_max,pondere);
END //
```

➢ Triggere

Atunci când o activitate de grup este anulată. Participanții ei primesc automat o notificare legată de asta. O activitate, indiferent de circumstanțe, este considerată anulată atunci când este ștearsă din baza de date. Datorită cheilor străine, trebuie să se șteargă și participanții ei, lucru care permite crearea unui trigger inainte de ștergere pe tabela de participanți, care obține în același timp și persoanele care trebuie să primească această informație. Similar este realizat și trigger-ul care anunță profesorii participanți la actțiune (adaptat la tabela profesori_invitați)

- Trigger notificare profesor anulare activitate de grup

```
DELIMITER ;;
CREATE TRIGGER notify_teacher_cancel BEFORE DELETE ON profesori_invitati FOR EACH ROW
BEGIN
DECLARE nume_grup VARCHAR(30);
DECLARE data_activitate DATETIME;

SELECT nume
INTO nume_grup
FROM grup
INNER JOIN activitati_grup
ON grup.ID_grup=activitati_grup.ID_grup AND activitati_grup.ID_activitate=OLD.ID_activitate ;

SELECT data_incepere INTO data_activitate
FROM activitati_grup
WHERE activitati_grup.ID_activitate=OLD.ID_activitate ;
INSERT INTO notificare(receiver_ID,sender_ID,sender_name,descriere,expirare)
VALUES(old.user_ID,0,nume_grup,CONCAT("Activitatea de pe data de ",data_activitate," a fost anulata"),TIMEDIFF(data_activitate,now()));
END ;;
```

- Trigger notificare studenti anulare activitate de grup

```
DELIMITER ;;
CREATE TRIGGER notify_cancel BEFORE DELETE ON participant_grup FOR EACH ROW
BEGIN
DECLARE nume_grup VARCHAR(30);
DECLARE data_activitate DATETIME;

SELECT nume
INTO nume_grup
FROM grup
INNER JOIN activitati_grup
ON grup.ID_grup=activitati_grup.ID_grup AND activitati_grup.ID_activitate=OLD.ID_activitate ;

SELECT data_incepere INTO data_activitate
FROM activitati_grup
WHERE activitati_grup.ID_activitate=OLD.ID_activitate ;
INSERT INTO notificare(receiver_ID,sender_ID,sender_name,descriere,expirare)
VALUES(old.user_ID,0,nume_grup,CONCAT("Activitatea de pe data de ",data_activitate," a fost
anulata"),TIMEDIFF(data_activitate,now()));
END ;;
```

- Trigger stergere grup daca nu mai sunt participanti

```
DELIMITER ..
CREATE TRIGGER leave_group BEFORE DELETE ON membru_grup FOR EACH ROW
BEGIN
IF((SELECT COUNT(*) FROM membru_grup WHERE membru_grup=OLD.grup_ID)=0)THEN
DELETE FROM grup WHERE grup.grup_ID=OLD.grup_ID;
END IF;
END..
```

- Trigger notificare studenti anulare activitate de grup

```
DELIMITER ;;
CREATE TRIGGER notify_cancel BEFORE DELETE ON participant_grup FOR EACH ROW
BEGIN
DECLARE nume_grup VARCHAR(30);
DECLARE data_activitate DATETIME;

SELECT nume
INTO nume_grup
FROM grup
INNER JOIN activitati_grup
ON grup.ID_grup=activitati_grup.ID_grup AND activitati_grup.ID_activitate=OLD.ID_activitate ;

SELECT data_incepere INTO data_activitate
FROM activitati_grup
WHERE activitati_grup.ID_activitate=OLD.ID_activitate ;
INSERT INTO notificare(receiver_ID,sender_ID,sender_name,descriere,expirare)
VALUES(old.user_ID,0,nume_grup,CONCAT("Activitatea de pe data de ",data_activitate," a fost
anulata"),TIMEDIFF(data_activitate,now()));
END ;;
```

- Trigger notificare desfasurare activitate

```
DELIMITER ;;
CREATE TRIGGER notify_confirmation AFTER INSERT ON participant_grup FOR EACH ROW
BEGIN
DECLARE participanti_min INT;
 SELECT activitati_grup.nr_minim_participanti INTO participanti_min
 FROM activitati_grup WHERE ID_activitate=NEW.ID_activitate;

IF(nr_participanti_grup(NEW.ID_activitate)=participanti_min) THEN

 SELECT nume
 INTO @nume_grup
 FROM grup
 INNER JOIN activitati_grup
 ON grup.ID_grup=activitati_grup.ID_grup AND activitati_grup.ID_activitate=NEW.ID_activitate ;

SELECT data_incepere INTO @data_activitate
 FROM activitati_grup
 WHERE activitati_grup.ID_activitate=NEW.ID_activitate ;
INSERT INTO notificare(receiver_ID,sender_ID,sender_name,descriere,expirare)
 VALUES(NEW.user_ID,NEW.ID_activitate,@nume_grup,CONCAT("Activitatea de pe data de ",@data_activitate,"
a adunat suficienti participanti!"),TIMEDIFF(@data_activitate,now()));
END IF;
END ;;
```

Atunci când o activitate de grup atinge numărul minim de participanți. Din nou, acest lucru se întamplă în mod automat atunci când numărul persoanelor participante (COUNT din participanții studenti + COUNT din participantii profesori) este egal cu numărul minim de participanți. Aici trebuie să se țină evidența inserărilor în tabelele pentru participanți.Din nou, trigger-ul pentru profesorii invitati este la fel, doar că adaptat la tabela profesori_invitați.

➢ Event

- Evidența timpului

Atât în cazul expirării notificărilor, cât și în cazul timpului de anunțare de la activitățile de grup, trebuie ținută o evidență a cât timp a trecut din acestea. Acest lucru este ușor realizabil de un event global care scade automat câte un minut din timpul respectiv, perioada în care acesta se repetă.

```
DELIMITER ;;
DELIMITER ..
CREATE EVENT countdown
ON SCHEDULE EVERY 1 MINUTE
 STARTS NOW()
ON COMPLETION PRESERVE
DO
BEGIN
UPDATE activitati_grup SET timp_anuntare=timediff(timp_anuntare, '00:01:00')  WHERE timp_anuntare>TIME('00:00:00');
 UPDATE notificare SET expirare=timediff(expirare, '00:01:00');
DELETE FROM activitati_grup WHERE timp_anuntare=TIME('00:00:00') AND nr_participanti_grup(ID_activitate)<nr_minim_participanti;
DELETE FROM notificare WHERE expirare=TIME('00:00:00');
END ..
```

Deoarece unele notificări pot să aibă un timp de expirare care să aibă o precizie temporară mai mare decât un minut (să aibă secundele diferite de 0), trebuie să se verifice și dacă timpul de expirare a scăzut sub 0. Din cauza asta, trebuie să se dezactiveze modul de SAFE UPDATES, scriindu-se SET SQL_SAFE_UPDATES = 0.

➢ Functii

Vastitatea și trecerea de neoprit a timpului uman pot intimida oamenii în găsirea unor criterii după care să descifreze dacă două date/intervale temporare coincid.
Observând modul de organizare calendaristică al timpului uman, se poate descifra cu ușurință un mod de aflare dacă două date calendaristice coincid în materie de zi.
Scâzând, de exemplu, datele a două zile cu aceeași frecvență, se poate observa periodicitatea în domeniul organizatoric (de exemplu: Ceva este săptămânal dacă are loc odată la 7 zile).

- Functie verificare suprapunere periodicitate zile

```
DELIMITER ;;
CREATE FUNCTION days_collide(original_date DATE,chosen_date DATE,freq VARCHAR(20)) RETURNS
BOOLEAN
DETERMINISTIC
BEGIN
DECLARE zile INT;
SELECT DATEDIFF(original_date,chosen_date) INTO @no_days;
CASE freq
-- 'zilnic','saptamanal','saptamanal_alternant','lunar'
WHEN 'zilnic' THEN SET zile:=@no_days%1;
WHEN 'saptamanal' THEN SET zile:=@no_days%7;
WHEN 'saptamanal_alternant' THEN SET zile:=@no_days%14;
WHEN 'lunar' THEN SET zile:=@no_days%30;
END CASE;
IF(zile=0) THEN
RETURN TRUE;
END IF;
RETURN FALSE;
END;;
```

- Functie verificare suprapunere intervale orare

Odată ce am realizat acest lucru, verificarea temporară a coinciderii a două intervale orare se reduce la o problemă generică de determinare a intersecției a două intervale.

```
DELIMITER ..
CREATE FUNCTION times_collide(d1 DATETIME,t1 TIME,d2 DATETIME,t2 TIME) RETURNS BOOLEAN
DETERMINISTIC
BEGIN
DECLARE final_rez BOOLEAN;
 SET @begin1=DATE_FORMAT(d1,'%H:%i') ;
 SET @end1= DATE_FORMAT(ADDTIME(d1,t1),'%H:%i');
 SET @begin2= DATE_FORMAT(d2,'%H:%i');
 SET @end2= DATE_FORMAT(ADDTIME(d2,t2),'%H:%i') ;
 IF(@begin1>=@end2 OR @begin2>=@end1) THEN
SET final_rez:=FALSE;
ELSE
SET final_rez:=TRUE;
END IF;
 RETURN final_rez;
END..
```

- Functie de calculare a numarului de participanti la o activitate

```
DELIMITER //
CREATE FUNCTION nr_participanti(ID int) RETURNS INT
DETERMINISTIC
BEGIN
DECLARE nr INT;
SELECT IFNULL(COUNT(*),0) FROM participant_curs p
WHERE ID_activitate=ID INTO nr;
RETURN NR;
END //
```

5.4. Elemente de securizare a aplicatiei

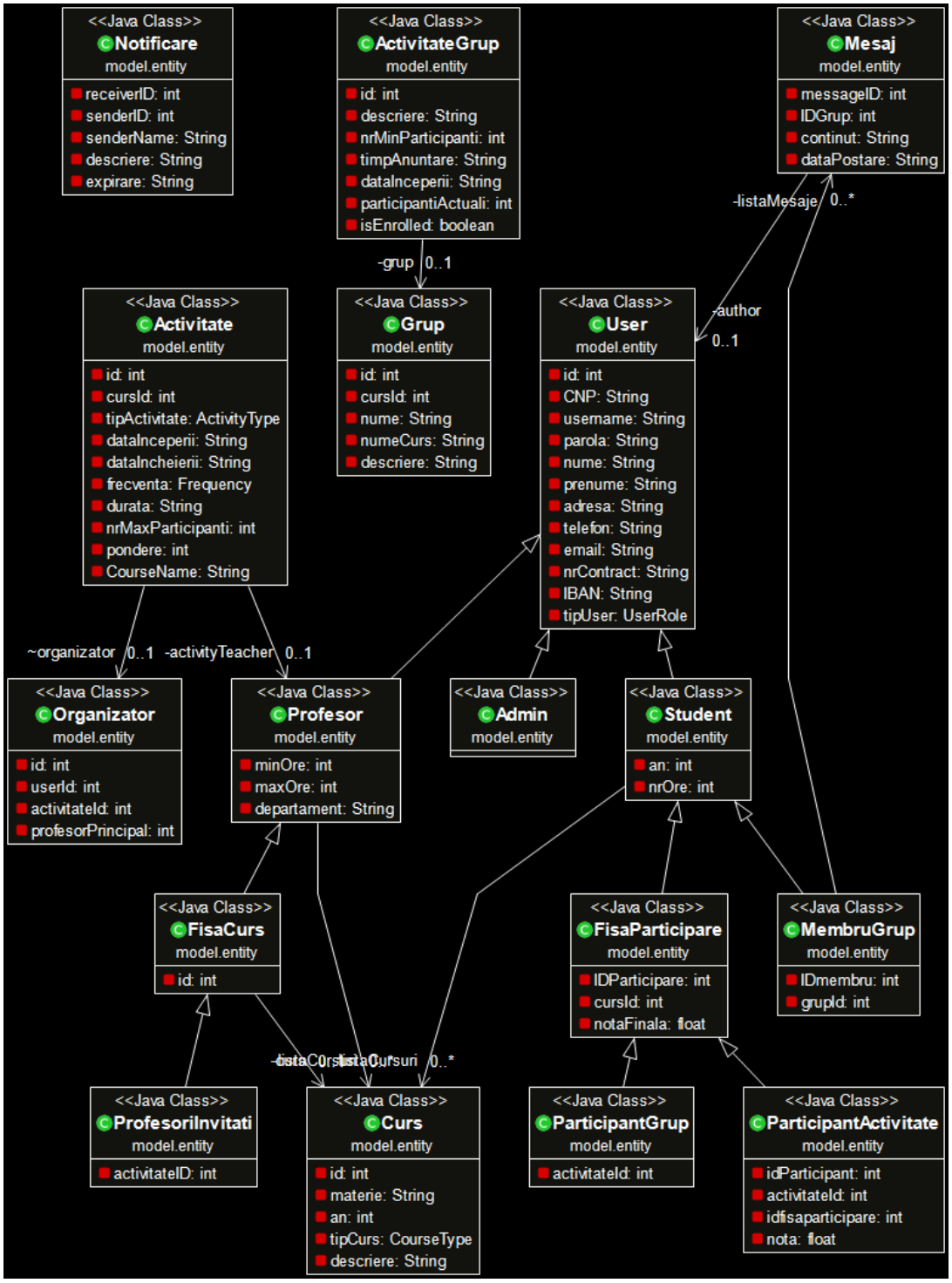
Funcția pricipală a sistemului de securitate al Sistemului de gestiune a bazelor de date MySQL este aceea de a autentifica și autoriza utilizatorii conectați pentru a accesa datele stocate.
Autorizarea se referă la permisiunea de a rula interogări/modificări ale datelor precum SELECT, INSERT, UPDATE sau DELETE. O clasă aparte de privilegii se referă la drepturile de administrare a bazei de date și de interacțiune cu sistemul de operare.
Termenul privilegiu denotă în general un drept al unui utilizator de a acționa într-un anumit fel asupra unui obiect al bazei de date (tabelă, câmp, index etc.).

```
DELIMITER ;;
CREATE ROLE 'ADMIN';
CREATE ROLE 'PROFESOR';
CREATE ROLE 'STUDENT';

GRANT ALL ON studiu.activitate TO 'ADMIN';
GRANT SELECT,DELETE ON studiu.activitati_grup TO 'ADMIN';
GRANT ALL ON studiu.curs TO 'ADMIN';
GRANT SELECT ON studiu.fisa_participare TO 'ADMIN';
GRANT INSERT,SELECT ON studiu.fisa_postului_curs TO 'ADMIN';
GRANT DELETE,SELECT,UPDATE ON studiu.grup TO 'ADMIN';
GRANT SELECT,DELETE ON studiu.membru_grup TO 'ADMIN';
GRANT ALL ON studiu.mesaj TO 'ADMIN';
GRANT ALL ON studiu.notificare TO 'ADMIN';
GRANT INSERT,SELECT ON studiu.organizator TO 'ADMIN';
GRANT SELECT,DELETE ON studiu.participant_curs TO 'ADMIN';
GRANT SELECT,DELETE ON studiu.participant_grup TO 'ADMIN';
GRANT SELECT ON studiu.profesori_invitati TO 'ADMIN';
GRANT ALL ON studiu.users TO 'ADMIN';
GRANT ALL ON studiu.student TO 'ADMIN';
GRANT ALL ON studiu.profesor TO 'ADMIN';


GRANT SELECT,UPDATE,INSERT ON studiu.activitate TO 'PROFESOR';
GRANT SELECT ON studiu.activitati_grup TO 'PROFESOR';
GRANT SELECT ON studiu.curs TO 'PROFESOR';
GRANT SELECT ON studiu.fisa_participare TO 'PROFESOR';
GRANT SELECT ON studiu.fisa_postului_curs TO 'PROFESOR';
GRANT ALL ON studiu.notificare TO 'PROFESOR';
GRANT ALL ON studiu.organizator TO 'PROFESOR';
GRANT SELECT,INSERT ON studiu.profesori_invitati TO 'PROFESOR';
GRANT SELECT ON studiu.users TO 'PROFESOR';
GRANT SELECT ON studiu.profesor TO 'PROFESOR';
GRANT SELECT ON studiu.student TO 'PROFESOR';
GRANT SELECT,UPDATE ON studiu.participant_curs TO 'PROFESOR';


GRANT SELECT ON studiu.activitate TO 'STUDENT';
GRANT SELECT,INSERT ON studiu.activitati_grup TO 'STUDENT';
GRANT SELECT ON studiu.curs TO 'STUDENT';
GRANT ALL ON studiu.fisa_participare TO 'STUDENT';
GRANT SELECT ON studiu.fisa_postului_curs TO 'STUDENT';
GRANT ALL ON studiu.grup TO 'STUDENT';
GRANT SELECT,INSERT ON studiu.membru_grup TO 'STUDENT';
GRANT ALL ON studiu.mesaj TO 'STUDENT';
GRANT ALL ON studiu.notificare TO 'STUDENT';
GRANT SELECT ON studiu.organizator TO 'STUDENT';
GRANT INSERT ON studiu.participant_curs TO 'STUDENT';
GRANT INSERT,SELECT ON studiu.participant_grup TO 'STUDENT';
GRANT SELECT ON studiu.profesori_invitati TO 'STUDENT';
GRANT SELECT ON studiu.student TO 'STUDENT';
GRANT SELECT ON studiu.users TO 'STUDENT';END ;;


CREATE USER 'demostudent'@'localhost' IDENTIFIED BY 'demostudent';
GRANT 'STUDENT' TO 'demostudent'@'localhost';
CREATE USER 'demoprofesor'@'localhost' IDENTIFIED BY 'demoprofesor';
GRANT 'PROFESOR' TO 'demoprofesor'@'localhost';
CREATE USER 'demoadmin'@'localhost' IDENTIFIED BY 'demoadmin';
GRANT 'ADMIN' TO 'demoadmin'@'localhost'
```

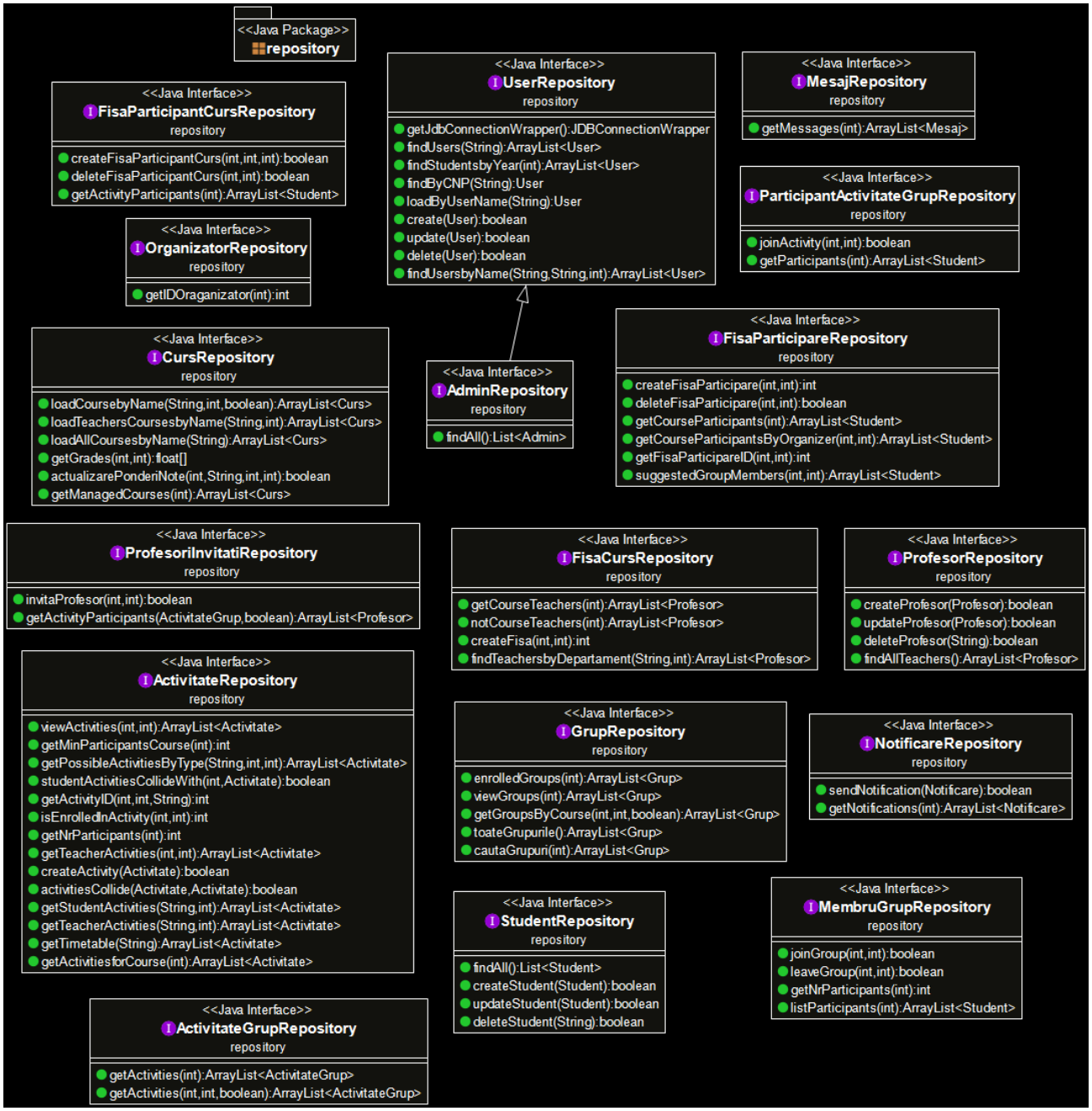
6. Detalii de implementare Java

6.1. Diagrama UML

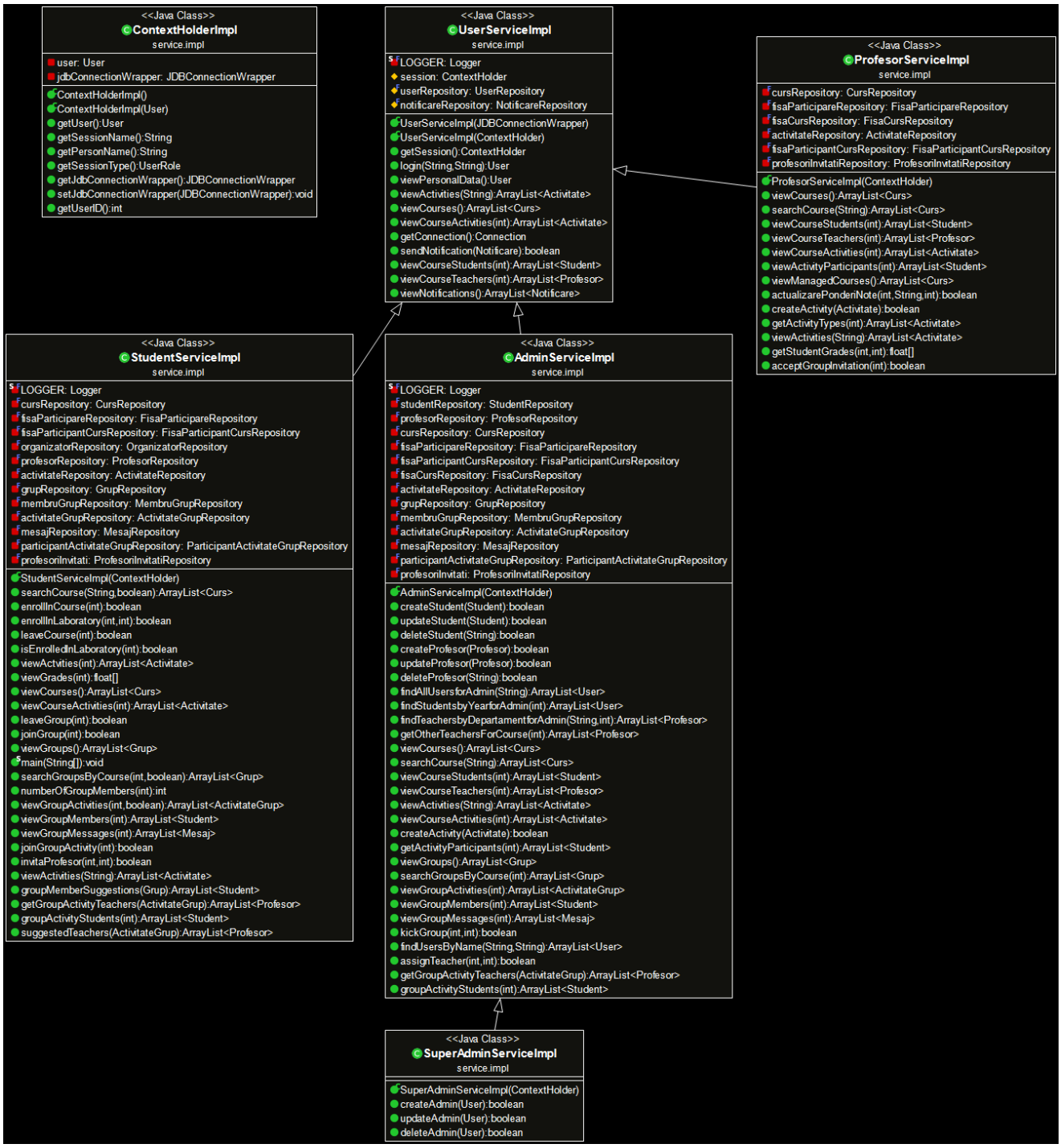
- Pachetul Entity



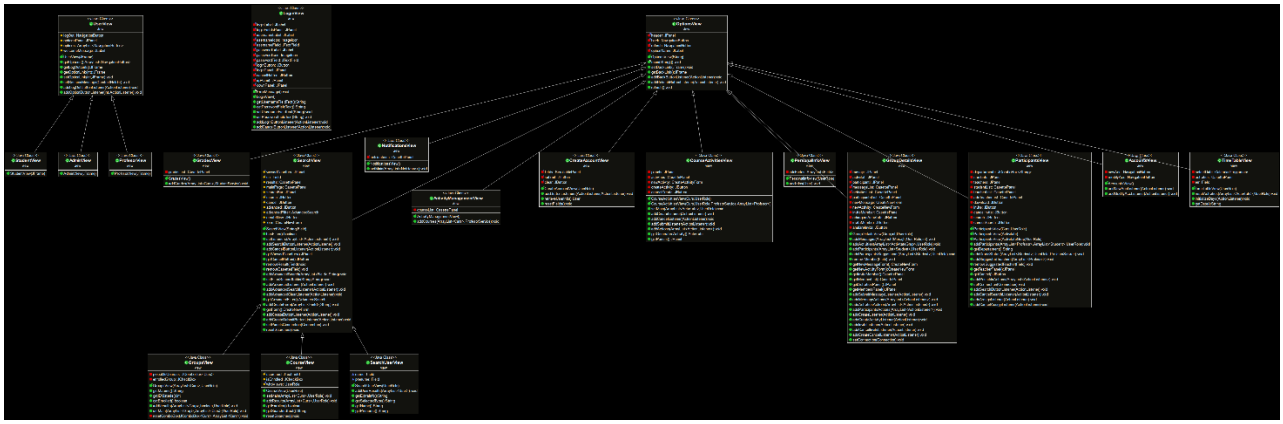
- Pachetul Repository



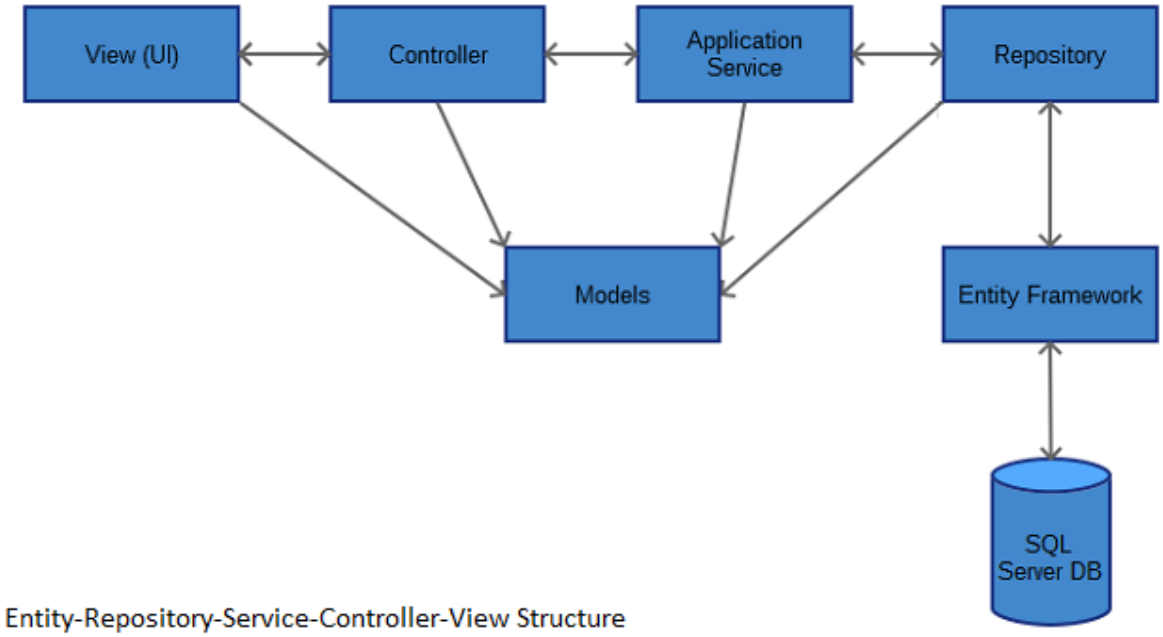
- Pachetul Service



- Pachetul View



6.2. Structura de clase in Java



Entity-Repository-Service-Controller-View Structure

- Scurta prezentare

Structura ierarhica Entitaty-Repository-Service-View-Controller structureaza aplicatia noastra pe 5 nivele distincte:

- Enitity – realizeaza maparea bazei de date;
- Repository – asigura comunicarea cu baza de date;
- Service – se foloseste de mai multe clase de Repository pentru a asigura functionalitatea specifica cerintelor;
- View – il ajuta pe utilizator sa interactioneze cu aplicatia intr-o maniera interactiva;
- Controller- face legatura dintre Service si View si controleaza functionalitatea.

- Construirea modelului

Am realizat o mapare 1:1 a bazei de date in java. Mai jos avem reprezentat in cod de java un tabel din daza de date.

```
package model.entity;
import model.enumeration.UserRole;
public class User {
    private int id;
    private String CNP;
    private String username;
    private String parola;
    private String nume;
    private String prenume;
    private String adresa;
    private String telefon;
    private String email;
    private String nrContract;
    private String IBAN;
    private UserRole tipUser;

    public int getId() { return id; }
    public void setId(int id) {...}
    public String getCNP() {...}
    public void setCNP(String CNP) {...}
    public String getUsername() {...}
    public void setUsername(String username) {...}
    public String getParola() {...}
    public void setParola(String parola) {...}
    public String getNume() {...}
    public void setNume(String nume) {...}
    public String getPrenume() {...}
    public void setPrenume(String prenume) {...}
    public String getAdresa() {...}
    public void setAdresa(String adresa) {...}
    public String getTelefon() {...}
    public void setTelefon(String telefon) {...}
    public String getEmail() {...}
```

- Cu ajutorul mostenirii si al functionalitatilor din java am reusit sa simplificam legaturile dintre tabele in procesul de mapare a bazei de date.

- Exemplu1: Clasa Studenti si clasa Profesori extind clasa User.

- Exemplu 2: Un obiect de tip FisaCurs contine un obiect de tip Curs si un obiect de tip Profesor, reunind intr-o singura clasa toate informatiile necesare.

- Construirea nivelului Repository

Clasele de Repository se ocupa cu operatii simple de interogare a bazei de date care sunt in principal realizate pe un singur tabel. Pe langa aceasta functionalitate CRUD aceste clase pot sa includa si metode mai complexe care pot sa includa pana la 3-4 tabele si care reunesc informatiile din acestea pentru a returnate rezultate de tip colectie. Exemplu: Calcularea notei finale a unui student la o disciplina prin insumarea notelor obtinute la fiecare activitate din cadrul unui curs la care este participant.

Un avantaj al claselor de Repository il constitue separea clara a preluarii si a procesarii datelor:
-datele sunt preluate la nivelul acestor clase din baza de date;
- datele sunt procesate la nivelul claselor de Service.

- Interfata UserRepository

```java
package repository;
import model.entity.User;
import repository.impl.JDBConnectionWrapper;

import java.util.ArrayList;
import model.enumeration.UserRole;
public interface UserRepository {
    public JDBConnectionWrapper getJdbConnectionWrapper();
    ArrayList<User> findUsers(String type_user);
    ArrayList<User> findStudentsbyYear(int an);
    ArrayList<User> findTeachersbyDepartament(String nume_departament);
    User loadByUserName(String userName);
    boolean create(User user);
    boolean update(User user);
    boolean delete(User user);
}
```

- Interogarea datelor de catre clasa UserRepositoryImpl

```java
@Override
public User loadByUserName(String userName) {
    Connection connection = jdbConnectionWrapper.getConnection();
    try {
        PreparedStatement preparedStatement = connection.prepareStatement( sql: "SELECT * FROM users WHERE username=?");
        preparedStatement.setString( parameterIndex: 1, userName);

        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
            return User.parseUser(resultSet, connection);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return null;
}
```

- Construirea nivelului Service

Clasele din layer-ul Service sunt concepute sa faca doua lucruri:

1. Sa se foloseasca se mai multe clase de Repository ;
2. In cadrul lor se implementeaza o functionalitate proprie pe baza cerintelor specifice ale aplicatiei. Aceasta functionalitate poate fi de cele mai multe ori complexa, intrucat reuneste informatiile din diferite clase de Repository;

Nivelul de Servicii defineste limitele aplicatiei si setul de actiuni posibile din perspectiva interactiunii cu utilizatorului cu interfata grafica. Incapsuleaza logica aplicatiei, controleaza tranzactiile si coordoneaza raspunsurile la diversele operatii ale utilizatorului.

- Serviciile unui utilizitator

```java
package service;

import java.sql.Connection;
import java.util.ArrayList;

import model.entity.*;

public interface UserService {
          User login(String username, String password);
          User viewPersonalData();
          ContextHolder getSession();
          ArrayList<Activitate> viewActivities(String selectedDate);
          ArrayList<Activitate> viewCourseActivities(int courseID);
          ArrayList<Profesor> viewCourseTeachers(int courseID);
          ArrayList<Student> viewCourseStudents(int courseID);
          ArrayList<Curs> viewCourses();
          public Connection getConnection();
          boolean sendNotification(Notificare notificare);
}
```

- Serviciul de Login

```java
@Override
public User login(String username, String password) {
    User user = userRepository.loadByUserName(username);
    if(user != null) {
        if(Objects.equals(user.getParola(), password)) {
            return user;
        } else {
            LOGGER.warning( msg: "Wrong password for user " + username);
        }
    } else {
        LOGGER.warning( msg: "User with username: " + username + " was not found");
    }
    return null;
}
```

- Construirea nivelului View + Controller

Interfața grafică este realizată în Framework-ul javax.swing pe un model de Controller-View
Partea de View cuprinde partea de vizualizare a interfeței grafice, iar Controller-ul stabilește comunicarea dintre acțiunile utilizatorului și aceasta. În principiu, partea de vizualizare este alcătuită din diferite ferestre(JFrame) legate între ele printr-o clasă specială de butoane (NavigationButton) ce reține link-ul la pagina anterioară (sau următoare în cazul butoanelor de pe pagina care apare după logare). Moștenirea permite realizarea unui șablon ce constituie tematica aplicației
Clase ajutătoare relevante :

- Clasa Field

Este o clasă ce reține legătura dintre un câmp de informație(TextField,TextArea,ComboBox etc.) și denumirea acestuia (Label). Această legătură, alături de sintaxa MySQL, permit realizarea cu ușurință a unor operații CRUD generale pentru eliminarea de nevoii de implementare a multor funcții particularizate, atâta timp cât Label-urile sunt denumite EXACT după numele coloanelor din tabele(eventual cu spații în loc de underscore). Cazurile speciale de operare pe mai multe tabele sunt tratate fien prin constrângeri MySQL, fie prin metode Java ajutătoare. Informația din câmpul de Info este extrasă în format String pentru a putea fi prelucrată ulterior, constrângerile de câmpuri realizându-se în mod extern în funcție de situație. De asemenea, există metode de verificare a constrângerilor de introducere a datelor, care trebuie apelate în funcție de situație.

- Clasa ScrollablePane

Este o clasă pentru un panou care se rotește automat, care reține câmpuri(Field) pe o aliniere în linie verticală a paginii. Ea este clasa principală a două clase relevante:

- Clasa CreateNewForm

Este o clasă pentru un formular de inserare a unei entități în baza de date. Acesta are două metode mai importante:

```
/**
 * formateaza o lista de valori/coloane din tabel din lista de String-uri in forma de tupla de inserat
 * adica (coloana1,coloana2,...,coloanaN) sau ('valoare1','valoare2',...,'valoareN') in functie de isValue
 *
 * @param tabel tabelul pe care se va face insertul, retinut doar pentru un caz special
 * @param source lista de informatii in format string din care extrag informatiile
 * @param isValue indica daca trebuie sa se puna ghilimele
 * @return returneaza valorile formatate obtinute, unite intr-un singur string
 */
```

```java
public static String makeInsertedVals(String tabel,ArrayList<String> source,boolean isValue){
    String rez="(";
    for(String i: source)
    {
        if(i.equals("Materie:")&&tabel.equals("grup"))
            i="ID_curs";
        if(isValue==true)
            rez+="'"+i+"'";
        else
            rez+=i.replaceAll( regex: " ",  replacement: "_").replaceAll( regex: ":", replacement: "");
        rez+=",";
    }
    if(rez.endsWith(","))
        rez=rez.substring(0, rez.length()-1);
    rez+=")";
    return rez;
}
```

Alaturi de

```java
/**
 * Insereaza un set de valori intr-un tabel, stiind ca sintaxa pentru inserare este
 * INSERT INTO tabel(coloana1,...,coloanan)
 * VALUES('valoare1',...,'valoare2')
 * @return ID-ul tuplei inserate pentru operatii ulterioare, sau 0 daca nu s-a putut insera
 */
public int insertTouple() {
    int ID=-1;
    try {
        ArrayList<String> col=this.getLabels();
        ArrayList<String> row=this.getDatas();

        if(this.rootedValues!=null) {
            ArrayList<String> rootedCol=this.getRootedInfos( isValue: false);
            col.addAll(rootedCol);
            ArrayList<String> rootedRow=this.getRootedInfos( isValue: true);
            row.addAll(rootedRow);
        }
        if(tabel.equals("mesaj")) {
            col.add("data_postare");
            DateTimeFormatter myFormatObj=DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
            row.add(myFormatObj.format(LocalDateTime.now()));
        }
        String cols=CreateNewForm.makeInsertedVals(tabel,col,  isValue: false);
        String vals=CreateNewForm.makeInsertedVals(tabel,row,  isValue: true);
        PreparedStatement s=connection.prepareStatement(
                 sql: "INSERT INTO "+tabel+cols+" VALUES"+vals, Statement.RETURN_GENERATED_KEYS);
        System.out.println(
                "INSERT INTO "+tabel+cols+" VALUES"+vals);
        s.execute();
        ResultSet rst=s.getGeneratedKeys();
        if(rst.next()) {
            System.out.println(rst.getString( columnIndex: 1));
            ID =rst.getInt( columnIndex: 1);
        }
        else System.out.println("NULL");

    }catch(Exception e) {
        ID=-1;
        e.printStackTrace();
    }
    return ID;
}
```

Care realizează o operație generală de INSERT. Este folosită în crearea cursurilor, grupurilor, mesajelor și a activităților dintr-un grup

- Clasa AdvancedSearch

O clasă pentru un filtru de căutare avansat care caută după mai multe coloane simultan, având și o condiție de join. Metodele relevante de aici sunt:

```java
/**
 * Formateaza mai multe Field-uri la o conditie de forma
 *  [WHERE] coloana1 LIKE 'valoare1' AND coloana2 LIKE 'valoare2' etc.
 *  De asemenea, ignora spatiile goale
 */
public void toCondition() {

    if(!mainCondition.equals(""))
        mainCondition+=" AND";
    if(mainFilters.datas!=null)
    {
        for(Field i:mainFilters.datas)
        {
            if(i.getInfoField() instanceof JPanel || i.getInfoField() instanceof JCheckBox)
                break;
            if(!i.getInfo().trim().equals(""))
                mainCondition+=" "+i.getName().replace( target: ":",  replacement: "").replace( target: " ",  replacement: "_")+" LIKE '"+i.getInfo()+"' AND";
        }
        if(mainCondition.endsWith("AND"))
            mainCondition=mainCondition.substring(0, mainCondition.length()-3);
    }
}
```

Alături de

```java
/**
 * Select cu o conditie mai complexa. Se bazeaza pe structura generala
 * SELECT *
 * FROM tabel
 * [INNER JOIN tabel2 ON conditieJoin]
 * WHERE conditiePrincipala;
 * @return ResultSet-ul cautarii pentru convertirea in entitati si folosirea lor ulterioara
 */
public ResultSet findRowInTableFiltered() {
    toCondition();
    String statement="SELECT * FROM "+tabel;
    try {
        Statement st = connection.createStatement();
        ResultSet rez = null;
        if(tabel2!=null&&doJoin)
        {
            if(tabel.equals("user"));
                statement+=" INNER JOIN "+tabel2+" ON users.ID="+tabel2+"."+commonCol;
            if(!joinCondition.trim().equals(""))
                statement+=" AND "+joinCondition;
            }
        if(!mainCondition.trim().equals(""))
            statement+=" WHERE "+mainCondition;
        System.out.println(statement);
        st.execute(statement);
        rez = st.getResultSet();
        mainCondition="";
        joinCondition="";
        return rez;

    } catch (Exception ae) {
        ae.printStackTrace();
        return null;
    }
}
```

Aceste filtre pot fi adăugate la orice pagină de căutare unde dorim să avem un filtru avansat (ex: căutare avansată de useri). De asemenea, se pot adăuga condiții suplimentare, atât la condiția de join, cât și în cea principală.

- Clasa Casette

Este o clasă generală pentru realizarea casetelor de informații, care reține în spate o entitate pentru eventuale acțiuni asupra acesteia(obținute dintr-o listă de acțiuni posibile ale unui panou de casete și determinate în funcție de situație).
De asemenea, clasa vine și cu o variantă editabliă a acesteia, care oferă un buton de editare și unul de ștergere. Aceste butoane au integrate acțiuni de editare și ștergere pe un format de detele și update general pentru MySql (care pot fi eventual scoase după situație). Câmpurile editabile sunt stabilite după o listă de Field-uri, iar prin convenție primul field este cheia de identificare după care se ștegre actualizează informația

Pentru partea de editare, se realizează un UPDATE general

```
/**
 * Funcie generala de UPDATE dupa sintaxa
 * UPDATE tabel
 * SET coloana1='valoare1', coloana2='valoare2' ...
 * WHERE coloana_de_identificare=valoare_de_identificare
 * @return o valoare booleana care reprezinta daca editarea s-a realizat cu succes
 */
public boolean updateDatas()
{
    boolean succes=true;
    try {
        Statement st=connection.createStatement();
        System.out.println("UPDATE "+tabel+" SET "+Casette.toUpdateValues(editableDatas)+" WHERE "+editableDatas.get(0).getName()+"="+edit.getValue());
        st.execute( sql: "UPDATE "+tabel+" SET "+Casette.toUpdateValues(editableDatas)+" WHERE "+editableDatas.get(0).getName()+"="+edit.getValue());

    }catch(SQLException e) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Nu s-au putut actualiza datele", title: "Eroare",JOptionPane.ERROR_MESSAGE);
        succes=false;
        e.printStackTrace();
    }
    return succes;
}
```

Iar pentru ștergere un DELETE general. Cazurile când se șterg din mai multe tabele sunt acoperite de cascadarea pe ștergere a cheilor străine:

```
/**
 * Sterge dintr-un tabel dupa o cheie de identificare
 * Foloseste sintaxa generala
 * DELETE FROM tabel WHERE coloana_identificare=valoare_identificare
 * @return o valoare booleana care reprezinta daca stergerea s-a realizat cu succes
 */
public boolean deleteEntity() {
    boolean succes=true;
    try {
        Statement st=connection.createStatement();
        System.out.println("DELETE FROM "+tabel+" WHERE "+editableDatas.get(0).getName()+"="+delete.getValue());
        st.execute( sql: "DELETE FROM "+tabel+" WHERE "+editableDatas.get(0).getName()+"="+delete.getValue());

    }catch(SQLException e) {

        succes=false;
        e.printStackTrace();
    }
    return succes;
}
public void setSource(Field f) {
    if(edit!=null)
        edit.setSource(f);
    if(delete!=null)
        delete.setSource(f);
}
```

Pentru formatarea informațiilor din Field-uri, există următoarea metodă

```
/**
 * Formateaza lista de valori editabile dupa sintaxa de valori de actualizat in MySQL
 * [UPDATE tabel SET] coloana1='valoare1', coloana2='valoare2' etc.
 * Metoda e statica pentru a se apela in mai multe situatii
 * @param editableDatas datele editabile dupa care se obtin rezultatele
 * @return rezultatul formatat intr-un singur string
 */
public static String toUpdateValues(ArrayList<Field> editableDatas) {
    String rez="";
    for(Field i:editableDatas) {
        if(!(i.getInfoField() instanceof JLabel))
        {
            if(!i.getInfo().replace( target: ":", replacement: "").trim().equals(""))
                rez+=i.getName()+"='"+i.getInfo()+"',";
        }
    }
    if(rez.endsWith(","))
        rez=rez.substring(0, rez.length()-1);
    return rez;
}
/**
```

Sistemul de notificări

Notificările primite de un utilizator sunt reținute temporar într-o tabelă de notificări în baza de date deoarece trebuie să se țină cont de asocierea cu utilizatorul căreia i-a fost trimisă. Ele reprezintă mesaje primite în mod automat de cineva (fără ca o persoană să o insereze manual în baza de date). Tabela de notificări e făcută pe un model mai general, fără a avea nici măcar o cheie străină pentru sender_ID (ID-ul entității despre care este notificarea), ele fiind diferențiate prin conținut și numele entității despre care este descrierea.

Trimiterea notificărilor

Deoarece notificările trebuie trimise fără a fi neapărat logat, se pune problema implementării unui sistem care să declanșeze singur generarea acestora, lucru care este ușor de realizat prin triggere care trimit notificări. Corpul și condițiile acestor triggere sunt stabilite în funcție de situația și condiția cu care se trimite mesajul generat automat.

De asemenea, există și unele situații în care se dorește generarea unui mesaj în funcție de o persoană logată (invitații), lucru realizabil din codul în Java.

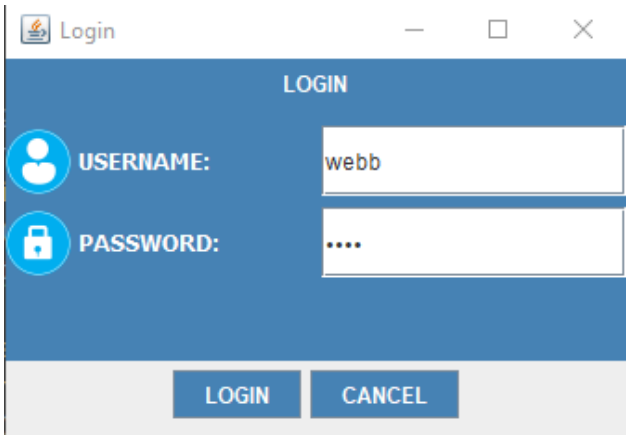
Tipuri de notificări și modul/condițiile de trimitere

1. *Atunci când o activitate de grup este anulată.* Participanții ei primesc automat o notificare legată de asta.

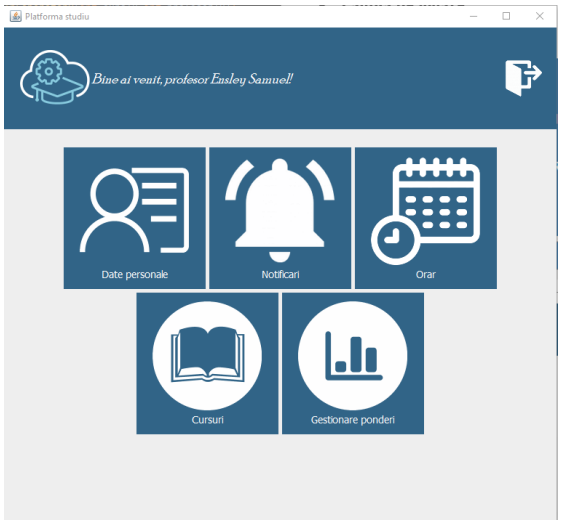
2. *Atunci când o activitate de grup atinge numărul minim de participanți.* Participanții ei primesc automat o notificare legată de asta.
3. *Atunci când un profesor este invitat la o activitate pe un grup de studiu*. În această situație trebuie să se țină deja cont de cine este logat pentru evidența grupului sursă exact și eventual a numelui persoanei care a trimis invitația. Tocmai de aceea, asta se realizează din Java, unde se reține cine este logat.
4. *Atunci când un student este invitat într-un grup de studiu.* Se realizează de asemenea în Java pentru că este din nou nevoie de un singur grup exact din care se trimite invitația.

7. Utilizarea aplicatiei

➢ **Pagina de logare**

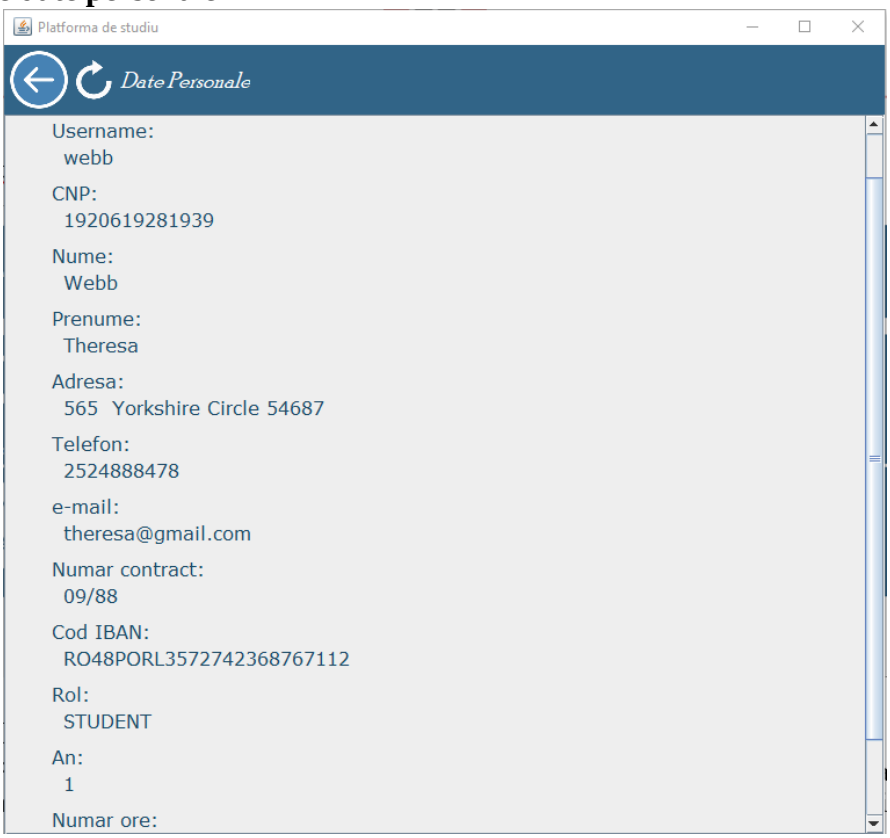


➢ **Interfata student/profesor**

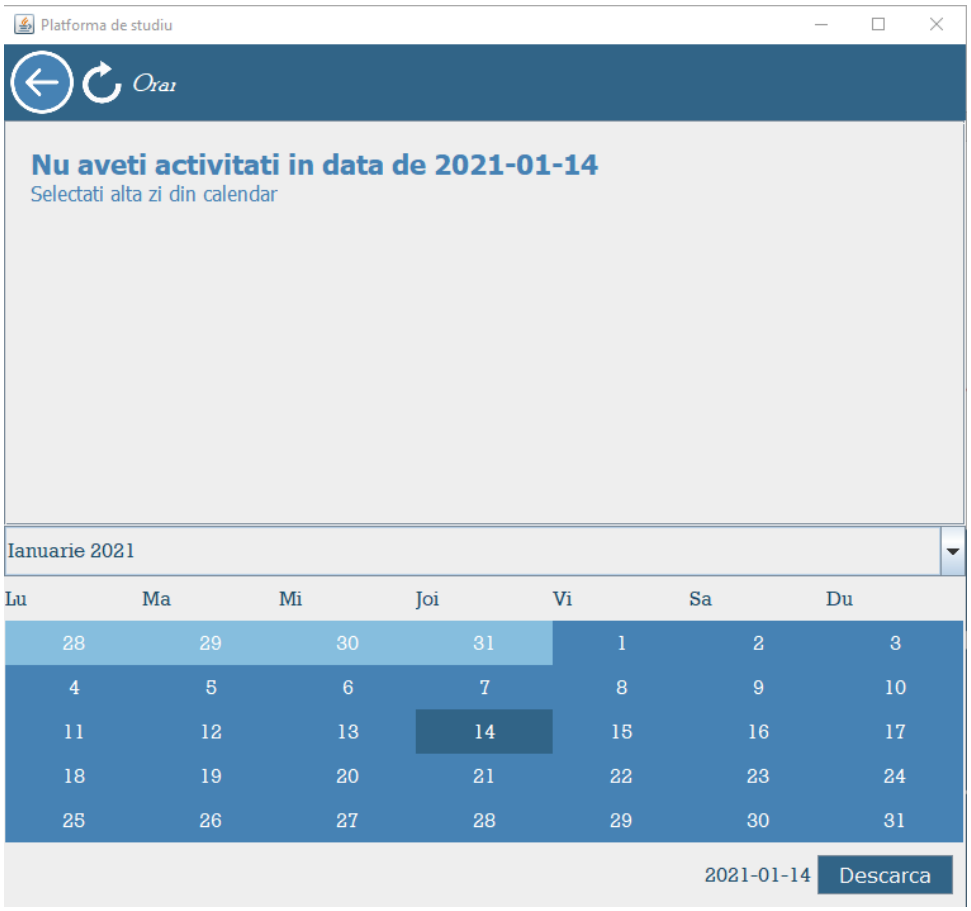


- **Interfata admin/super-admin**

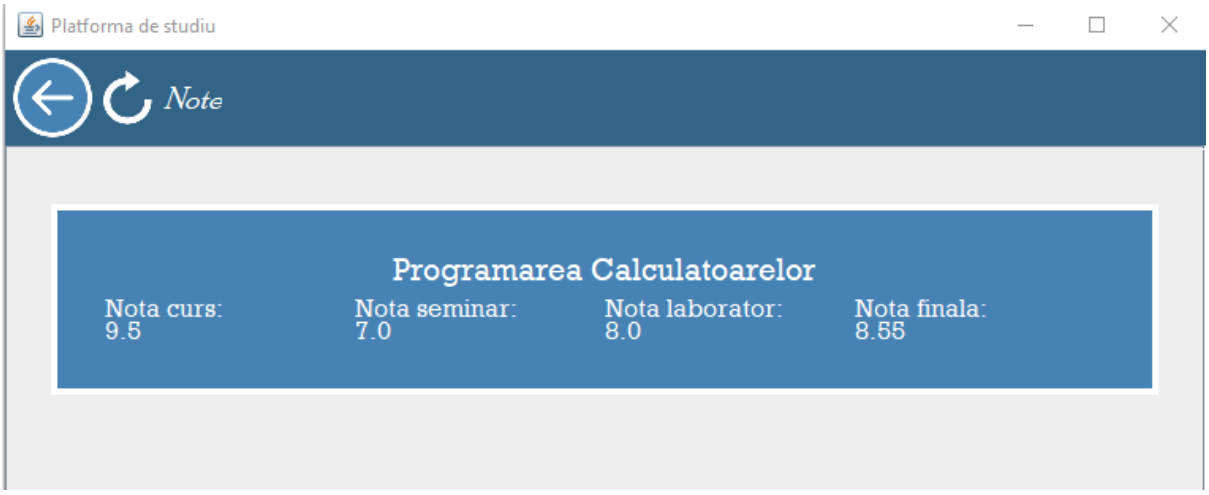
- **Pagina de date personale**



- **Pagina de Orar**



- **Pagina de notificari**



- **Pagina de cursuri (student/profesor/admin(super-admin)**

8. Concluzii. Limitari si dezvoltari ulterioare

Proiectul nostru oferă gestionarea în mod eficient a sistemului informatic al unei platforme de studiu in cadrul careia sunt oferite diferite functionalitati pentru utilizatorii acesteia, care pot fi studenti, profesori, admini si super-adminul.
Aplicatia noastra ofera servicii de diferite tipuri in functie de tipul de utilizator care se logheaza: vizualizarea datelor personale, vizualizarea orarului, vizualizarea si gestionarea cursurilor si activitatilor, grupurilor, toate acestea fiind realizate intr-o maniera cat mai interactiva pentru utilizatorii ei.
Aceste servicii sunt facilitate de interfata grafica care face navigarea printre optiuni foarte usoara si asigura corectitudinea datelor.

Uilizatorii au acces la toată informația care-l interesează cu un singur click. De asemenea, aplicația oferă suport doar personalului specific: studenti, profesori si admini (respectiv super-admin), acest lucru oferind o mai ușoară gestiune a bazei de date.
Niciun utilizator nu poate accesa baza de date daca nu are creat in prealabil un cont de catre unul din administratorii bazei de date, asigurandu-se in acest fel securitatea informatiilor.
Spre exemplu, studentul poate avea acces la informatii despre cursuri, se poate inscrie la acestea, profesorul poate sa gestioneze ponderile activitatilor de la materiile pe care acesta le preda, administratorul are ușor acces la absolut toate datele și poate efectua direct modificările dorite, super-administratorul avand aceleasi drepturi ca si administratorul la care se adauga crearea de conturi de admini.

In ceea ce priveste dezvoltarea ulterioara, s-ar mai putea aduce imbunatiri pe parte de interfata grafica, s-ar putea adauga un sistem de examinare pe aceasta platforma unde profesorii sa poata crea activitati de tip: colocviu/examen utilizand formulare de raspuns.
De asemenea, aceasta platfoma ar mai putea fi imbunatatita prin adaugarea unei spatiu de colaborare unde studentii isi pot incarca temele.

O limitare o constituie faptul ca conexiunea la baza de date este una locala, accesul pe platforma se poate face doar daca dispozitivul de pe care se face conectarea are creata in prealabil baza de date.

9. Bibliografie

[1]- https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)
[2]- https://ro.wikipedia.org/wiki/MySQL
[3]- https://ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs10/curs10.html
[4]- https://tytie.com/what-is-jdbc-introduction-to-the-java-database-connectivity-api/