UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICA SI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

PROIECT

PLATFORMA DE STUDIU

Zbughin Cezar, grupa 30222

Zagan Bogdan, grupa 30222

Vijoli Ioana, grupa 30222

An academic: 2021 – 2022

Cuprins

1. Introducere

În cadrul acestui proiect, am realizat o aplicaţie cu interfaţă grafică care va oferi utilizatorilor funcţionalităţi, corespunzător cu drepturile pe care aceştia le deţin în sistem, descriseprin intermediul unor roluri.

Proiectul nostru implementează o aplicaţie care se ocupă cu gestiunea unei platforme de studiu. Aplicația folosește un sistem de gestiune pentru baze de date MySQL, iar interacțiunea cu aceasta se realizează doar prin interfața grafica.

Funcționalitățile pe care le oferă programul vizează operații ce țin de gestiunea studenților, profesorilor si administrarea operațiilor curente din cadrul unor programe de studiu. Aplicația este accesată, pe baza unui proces de autentificare, de către mai multe tipuri de utilizatori: studenți, profesori, administratori.

În cadrul proiectului am avut ca scopuri principale:

- crearea tabelelor mentionate in cerinţă
- scrierea de proceduri in MYSQL, aplicabile fiecarui tip de utilizator
- definitivarea mai multor tipuri de utilizatori
- crearea interfeței grafice pentru a facilita interacțiunea utilizatorului cu aplicația
- modificarea bazei de date in funcție de evenimentele generate in interfața grafică

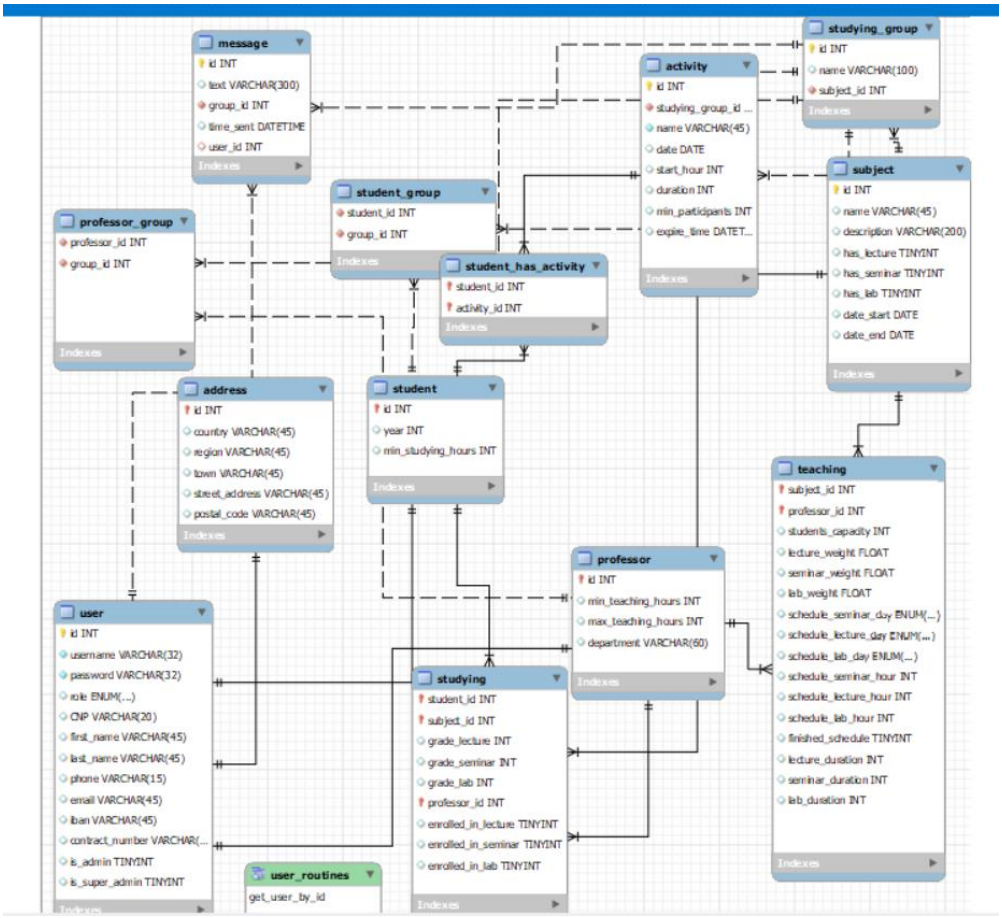
In realizarea acestui proiect s-au folosit:

- programarea în limbajul Java
- manipularea bazelor de date folosind MySQL;
- folosirea JDBC - API pentru SQL Connection

2. Diagrama de tabele

In imaginea de mai jos se pot observa îndeaproape tabelele folosite si conexiunile realizate intre acestea (prin relații 1….1, 1….n, n….n). Acestea alcătuiesc baza de date pentru sistemul nostru bancar. Procedurile si operațiile efectuate au efect direct asupra modificării informațiilor existente in tabelele de mai jos. Conexiunile dintre tabelele de mai jos este

realizata prin relațiile de Primary key – Foreign Key, pentru a păstra unicitatea tuplelor si, mai mult decât atât, utilizareaeficienta a procedurilor pe datele care populează tabelele noastre.



Pentru realizarea acestei baze de date au fost construite si utilizate 13 tabele si anume:

1. User
2. Student
3. Professor
4. Teaching
5. Studying

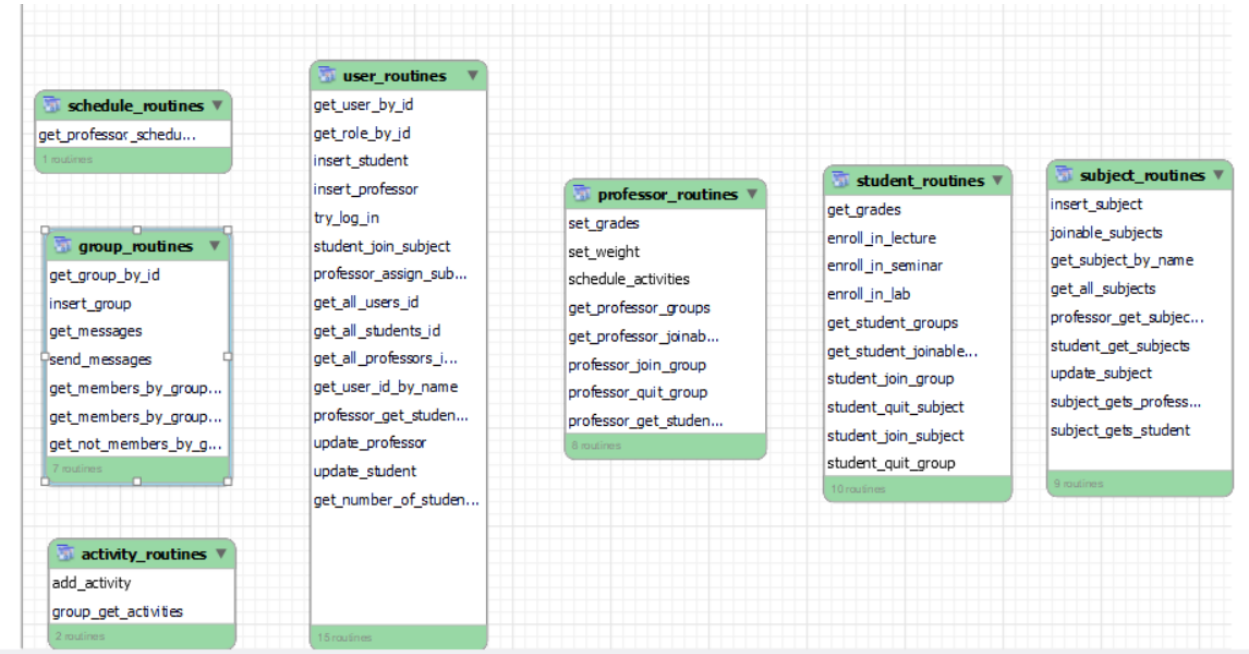
6. Subject
7. Address
8. Sudying_group
9. Professor_group
10. Student_group
11. Activity
12. Student_has_activity
13. Message

1. USER: această tabelă reţine principalele informaţii despre un utilizator, precum username, parola, CNP, nume, prenume, numar de telefon, email, IBAN etc. De asemenea, prin intermediul atributelor is_admin şi is_super_admin se memorează daca utilizatorul are permisiune de administrator, respectiv superadministrator al bazei de date.
2. STUDENT: tabela memorează anul de studiu şi numarul minim de ore de studiu pentru un utilizator de tip student.
3. PROFESSOR: în această tabelă se reţin departamentul, numarul minim respectiv maxim de ore predate de către un utilizator de tip profesor.
4. TEACHING: această tabelă este utilizată pentru a modela realţia de tipul many-to-many între Professor şi Subject. Tot prin intermediul acestei entităţi se memorează ponderile pentru fiecare tip de activitate şi se reţine ziua, ora şi durata fiecărei activităţi programate de către profesor.
5. STUDYING: similar tabelei Teaching, tabela Studying face legătura între tabelele Student, Professor şi Subject. Astfel, pentru fiecare tuplu format din student, materie şi profesor se vor memora nota obţinută de student la seminar, laborator, respectiv curs, dar se va ţine minte şi dacă acesta este înscris sau nu la fiecare tip de activitate.
6. SUBJECT: este folosită pentru a memora principalele date despre o materie (numele, descrierea, data de început, data de final, dacă are sau nu seminar, curs, sau laborator).
7. ADDRESS: folosită pentru a stoca adresa fiecărui user
8. STUDYING_GROUP: este legată printr-o Cheie Străină de tabela Subject şi memorează fiecare grup de studiu creat pe platformă

9. PROFESSOR_GROUP: este tabela de legătură dintre Professor şi Studying_Group (deoarece avem relaţie de tipul many-to-many).
10. STUDENT_GROUP: este tabela de legătură dintre Student şi Studying_Group (deoarece avem relaţie de tipul many-to-many).
11. ACTIVITY: este utilizată pentru a putea adauga o nouă activitate în grupul de studiu. Din acest motiv, putem observa cu uşurinţă că este legată printr-o cheie străină de tabela Studying_Group.
12. STUDENT_HAS_ACTIVITY: un student poate fi înscris la mai multe activităţi din grupul de studiu, iar o activitate poate avea mai mulţi studenţi. Prin urmare avem o relaţie de tipul many-to-many între student şi activitate, iar tabela Student_has_activity ne ajută să modelăm corespunzător această relaţie.
13. MESSAGE: această tabelă este folosită pentru a putea reţine mesajele pe care utilizatorii, înscrişi într-un grup de studiu, le trimit ca să comunice între ei.

3. Lista de proceduri

Pentru a implementa funcționalitățile cerute, am utilizat numeroase proceduri, scrise în MYSQL, care vor fi ulterior apelate de către aplicația Java şi folosite de interfața grafică.



Mai jos se regăsesc câteva dintre principalele proceduri utilizate:

1. STUDENT_JOIN_SUBJECT

Această procedură este utilizată pentru a înscrie un student la o anumită materie. Ea primeşte ca parametri id-ul studentului şi id-ul materiei la care doreşte să se înscrie şi va fi apelată în StudentService.

Codul procedurii:

```
CREATE PROCEDURE student_join_subject(p_student_id INT, p_subject_id INT)
BEGIN
DECLARE v_professor_id INT;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;  -- rollback any changes made in the transaction
      RESIGNAL;  -- raise again the sql exception to the caller
   END;
START TRANSACTION;
set v_professor_id = (select professor_id from studying where subject_id = p_subject_id group
by professor_id order by count(professor_id) limit 1);
INSERT INTO studying(student_id,subject_id,professor_id)
VALUES (p_student_id, p_subject_id, v_professor_id);
COMMIT;
END
```

2. JOINABLE_SUBJECTS

Procedura primeşte ca parametru id-ul unui student şi returnează toate materiile la care acesta se poate înscrie. Folosind ,,not in" se impune condiţia ca studentul să nu fie deja înscris la materia respective şi astfel va putea da join ulterior.

Codul procedurii:

```
CREATE PROCEDURE joinable_subjects(sstudent_id INT)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;  -- rollback any changes made in the transaction
      RESIGNAL;  -- raise again the sql exception to the caller
```

```
   END;
START TRANSACTION;
    select * from subject where (id, sstudent_id) not in (select subject_id, student_id from
studying);
COMMIT;
END
```

3. SET_WEIGHT

Această procedură este utilizată pentru ca un profesor să poată seta ponderile fiecărui tip de activitate, seminar, laborator, respectiv curs.

Codul procedurii:

```
CREATE PROCEDURE set_weight(id_professor INT, id_subject INT, lecture_w
FLOAT, seminar_w FLOAT, lab_w FLOAT)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;  -- rollback any changes made in the transaction
      RESIGNAL;  -- raise again the sql exception to the caller
   END;
START TRANSACTION;
UPDATE teaching t
SET t.lecture_weight = lecture_w WHERE t.professor_id = id_professor AND
t.subject_id = id_subject;
UPDATE teaching t
SET t.seminar_weight = seminar_w WHERE t.professor_id = id_professor AND
t.subject_id = id_subject;
UPDATE teaching t
SET t.lab_weight = lab_w WHERE t.professor_id = id_professor AND t.subject_id =
id_subject;
COMMIT;
END
```

4. GET_PROFESSOR_GROUPS

Procedura conţine o interogare ce returnează toate grupurile în care un profesor este înscris, în funcţie de id-ul acestuia, trimis ca parametru.

Codul procedurii:

```
CREATE PROCEDURE get_professor_groups (given_id INT)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;  -- rollback any changes made in the transaction
      RESIGNAL;  -- raise again the sql exception to the caller
   END;
START TRANSACTION;
select g.id, g.name, g.subject_id from studying_group g where g.id in (select group_id
from professor_group pg where pg.professor_id = given_id);
COMMIT;
END
```

5. TRY_LOG_IN

Procedura este utilizată pentru a efectua logarea în aplicaţia Studying Platform. În cazul în care username-ul şi parola introduse există în baza de date, înseamnă că utilizatorul se poate conecta şi se returnează mesajul ,,successful". În caz contrar, log_in_status va memora un mesaj corespunzător, cu scopul de a-l avertiza pe user că datele introduce nu sunt corecte, făcând astfel imposibil procesul de autentificare.

Codul procedurii:

```
CREATE PROCEDURE try_log_in(pusername VARCHAR(32),ppassword VARCHAR(32))
BEGIN
DECLARE log_in_status varchar(45);
DECLARE user_id int default null;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;  -- rollback any changes made in the transaction
      RESIGNAL;  -- raise again the sql exception to the caller
   END;
START TRANSACTION;
        set user_id = (select id from user where username=pusername and password =
ppassword);
        IF (exists(select id from user where username=pusername and password = ppassword))
then
                set log_in_status = "successful";
        ELSEIF(exists(select id from user where username = pusername)) then
                set log_in_status = "wrong password";
        ELSE
                set log_in_status = "username not found";
```

```
        END IF;
   SELECT log_in_status,user_id;
COMMIT;
END
```

6. GET_NUMBER_OF_STUDENTS

Procedura returnează numărul de studenţi înscris la un anumit profesor, de la o anumită materie cu id-ul psubject_id. Această procedură este foarte importantă, deoarece o folosim ulterior când asignăm fiecare utilizator de tip student la profesorul cu cei mai puţini studenţi înscrişi de la momentul respectiv.

Codul procedurii:

```
CREATE PROCEDURE get_number_of_students(prof_id INT, psubject_id INT)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;  -- rollback any changes made in the transaction
      RESIGNAL;  -- raise again the sql exception to the caller
   END;
START TRANSACTION;
         select count(student_id) as number_of_students from studying where
professor_id = prof_id and psubject_id = subject_id;
COMMIT;
END
```

7. GET_MESSAGES:

Procedura este utilizata in cadrul unui grup de studiu. Ea returnează mesajele trimise înainte de start date primită ca parametru, ordonate în funcţie de timpul la care au fost trimise.

Codul procedurii:

```
CREATE PROCEDURE get_messages(given_group INT, start_date DATETIME, lim INT)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;  -- rollback any changes made in the transaction
```

```
        RESIGNAL;  -- raise again the sql exception to the caller
      END;
    START TRANSACTION;
    select m.id, m.text, m.group_id, m.time_sent, m.user_id from message m where m.group_id
= given_group AND m.time_sent < start_date order by m.time_sent desc limit lim;
    COMMIT;
    END
```

4. Implementarea aplicaţiei Java

Aplicația java se conectează la un server MySQL local pe care funcționează baza de date descrisa mai sus. In clasa „DatabaseService” din pachetul „sevice” exista o serie de parametri de conectare la baza de date. Pentru a rula aplicația pe alte calculatoare acești parametri trebuie sa fie modificați corespunzător.

```
public final static String DB_USERNAME = "root";
public final static String DB_PASSWORD = "root";
public final static String DB_NAME = "StudyingPlatform";
public final static String DB_CONNECTION_LINK = "jdbc:mysql://localhost:3306/";
```

➢ Utilizarea JavaFX

JavaFX este un set de pachete media si grafice care au o filosofie diferita de Swing-up din java. Swing pune la dispoziție o serie de clase grafice care pot fi create si adăugate unei ferestre sau altor obiecte grafice.

JavaFX, pe lângă clase asemănătoare celor din Swing si a unei serii de clase noi, pune la dispoziție si Clasa „FXMLLoader”. Obiecte de acest tip sunt capabile sa refere fișiere de tip .fxml, iar la nevoie sa încarce(=interpreteze) conținutul acestor fișiere.

Fișierele .fxml conțin instrucțiuni intr-un limbaj propriu care descriu layout-ul interfeței grafice; acestea sunt asemănătoare fișierelor .html. Loaderul primește aceste fișiere și creează obiectele necesare și le organizează intr-o structura arborescenta pe care o returnează.

Exemplu de fișier .fxml responzabil de descrierea GUI-ului pentru pagina de logare.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Hyperlink?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>

<VBox fx:id="root" alignment="TOP_CENTER" maxHeight="-Infinity" maxWidth="-
Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="500.0"
prefWidth="400.0" xmlns="http://javafx.com/javafx/17"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.StudyingPlatform.controllers.LogInController">
    <children>
        <HBox alignment="BOTTOM_CENTER" prefHeight="115.0" prefWidth="400.0">
            <children>
                <Label alignment="TOP_CENTER" contentDisplay="CENTER"
prefHeight="55.0" prefWidth="350.0" text="Studying Platform">
                    <font>
                        <Font name="System Bold" size="36.0" />
                    </font>
                </Label>
            </children>
        </HBox>
        <HBox alignment="BOTTOM_CENTER" prefHeight="109.0" prefWidth="400.0">
            <children>
                <TextField fx:id="usernameField" prefHeight="50.0"
prefWidth="235.0" promptText="username">
                    <font>
                        <Font size="18.0" />
                    </font>
                </TextField>
            </children>
        </HBox>
        <HBox alignment="BOTTOM_CENTER" prefHeight="79.0" prefWidth="400.0">
            <children>
                <PasswordField fx:id="passwordField" prefHeight="55.0"
prefWidth="231.0" promptText="password">
                    <font>
                        <Font size="18.0" />
                    </font>
                </PasswordField>
            </children>
        </HBox>
      <HBox alignment="BOTTOM_CENTER" layoutX="10.0" layoutY="10.0"
prefHeight="47.0" prefWidth="400.0">
         <children>
            <Label fx:id="errorLabel" alignment="TOP_CENTER"
contentDisplay="CENTER" prefHeight="42.0" prefWidth="229.0" text="Something
went wrong!" textFill="RED" visible="false" wrapText="true">
               <font>
                  <Font size="14.0" />
               </font>
            </Label>
         </children>
```

```
      </HBox>
        <VBox alignment="BOTTOM_CENTER" prefHeight="73.0" prefWidth="400.0">
            <children>
                <Button mnemonicParsing="false"
onMouseClicked="#onLogInButtonClick" prefHeight="43.0" prefWidth="172.0"
text="Log in">
                    <font>
                        <Font name="System Bold" size="18.0" />
                    </font>
                </Button>
                <Hyperlink alignment="TOP_CENTER" contentDisplay="CENTER"
onMouseClicked="#onCreateAccountClick" prefHeight="24.0" prefWidth="169.0"
text="create an account">
                    <font>
                        <Font size="14.0" />
                    </font>
                </Hyperlink>
            </children>
        </VBox>
    </children>
</VBox>
```

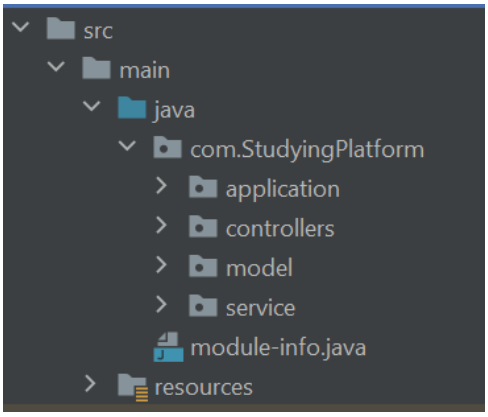
Important legat de fișierele .fxml este noțiunea de controller. Aceste fișiere de descriere trimit la clase controler definite de utilizator. Ex:

```
…fx:controller="com.StudyingPlatform.controllers.LogInController">
```

Clasele controller primesc responsabilitatea controlării interfeței grafice pe care o reprezintă. Astfel, ea devine responsabila de încălcarea datelor in Labal-uri, ori in TextField-uri. In fișiere .fxml. Butoanele sau orice alte obiecte pot sa trimită numele metodelor care sa fie apelate in cazul unui event. De exemplu, conform cu fișierul „log-in-view.fxml”, logInButton va a apela metoda onLogInButtonClick pe care o va caută in clasa controller oferita.

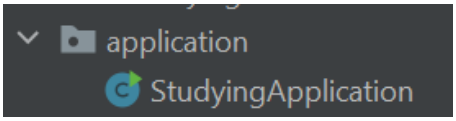
Obiectele descrise in fișier pot primi un fx:id prin care acestea pot fi folosite in Clasa controller. Adică, la încărcare, se creează un obiect controller care are un set de variabile instanța ce refera obiectele grafice create conform cu fx:id primit.

- Organizarea pe pachete a proiectului.

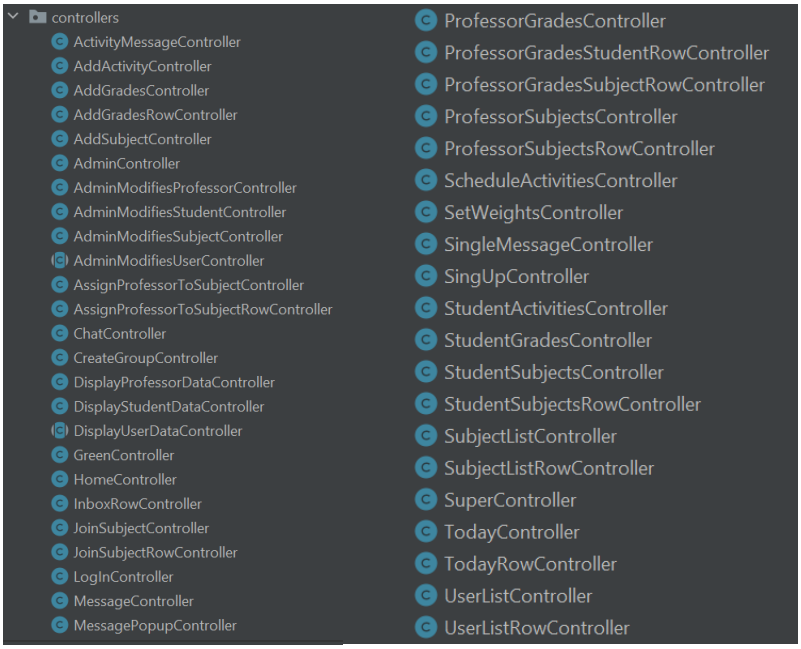


Aplicația este organizata pe in doua foldere principale: java si resources. In java se găsește codul sursa pentru toate clasele folosite, iar in resources se gasesc fisierele .fxml.

In „application”:

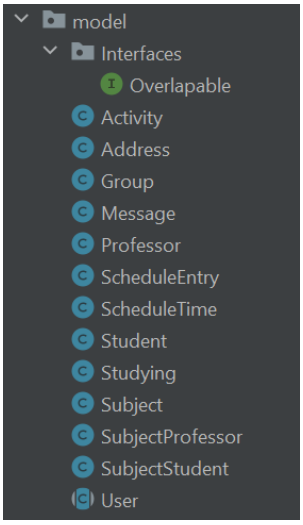


O singura clasa StudyingApplication care extinde Application si care contine metodele main si start specifice java si respectiv javaFX.

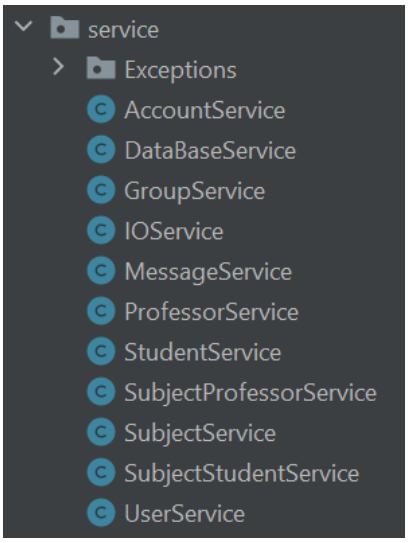


In „controllers”: Conține toate controllerele necesare pentru rularea aplicației.

In „model”: Modelele sunt clase care echivalează tabele din baza de date sau modelează orice noțiune concreta care este folosita in definirea tabelelor. (vezi ScheduleEntry).



In „service”: Serviciile sunt clase experți in anumite domenii, exemplu DataBaseService este o clasa experta in comunicarea cu baza de date, StudentService este o clasa experta in Studenti, astfel orice comportament care pare a fi prea specific pentru a fi direct nativ unui model poate fi adăugat intr-un serviciu expert in modelul respectiv. Exemplu: studentGetJoinableSubjects(). Returneaza toate cursurile la care un student se poate alătura.



➢ Flow-ul programului.

Totul pornește de la controllere. Acestea au doua responsabilități:

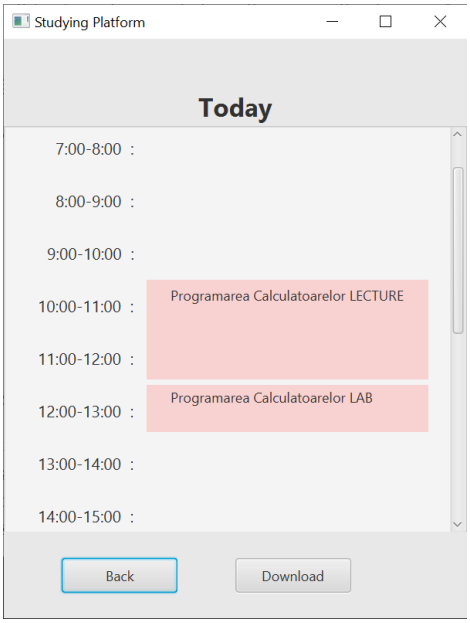
- mentenanța structurii grafice, de exemplu poluarea TextField-urilor, inserarea de linii noi in liste.
- Controlul flow-ului programului, de exemplu apariția unei noi ferestre, schimbarea vederii la apăsarea butonului „back”.

Astfel, in controllere nu se găsește cod care sa facă operații pe studenți, profesori, cursuri, ori sa facă interogări in baza de date, ori sa facă mentenanța conturilor. Toate aceste responsabilități cad pe servicii. Astfel, zicem: controllerele primesc input de la utilizator, datele sunt îmbrăcate in modele care sunt trimise la servicii. Serviciile fie returnează modele ca răspuns, fie executa operații de management.

5. Funcționalități specifice tipurilor de useri

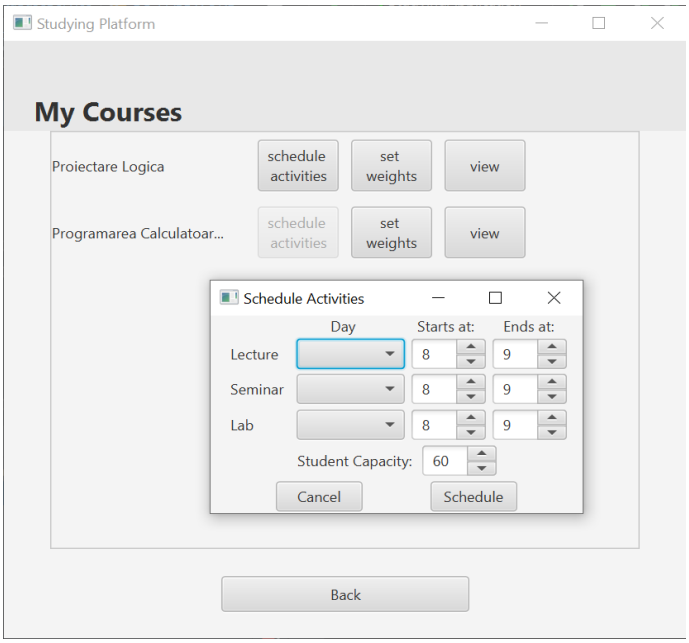
➢ Today

Today este o funcționalitate disponibila atât studenților cat si profesorilor care oferă user-ului posibilitatea de a-si vedea activitățile pe ziua curenta.



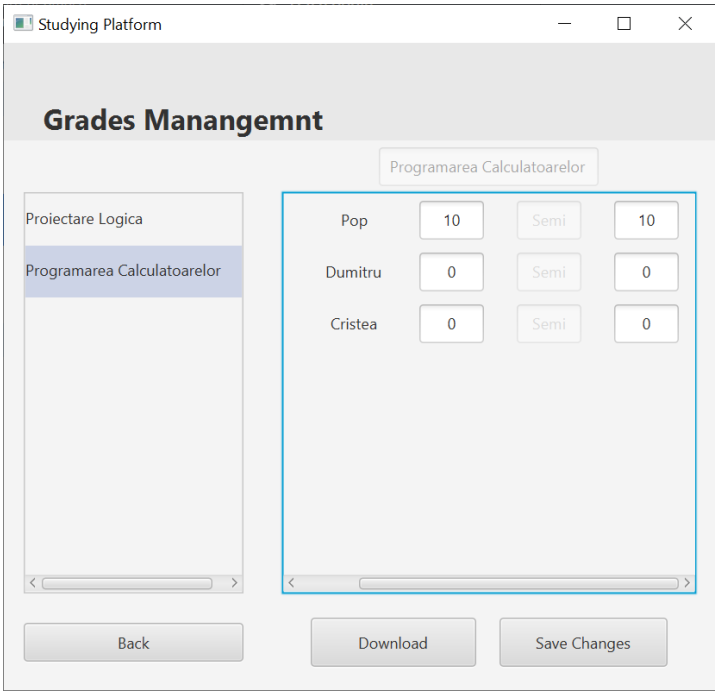
➢ Courses pentru profesori

Pentru profesori avem pagina „Courses”. Pe acesta pagina profesorii pot programeze activități, sa seteze ponderile notelor si sa vadă date despre curs.



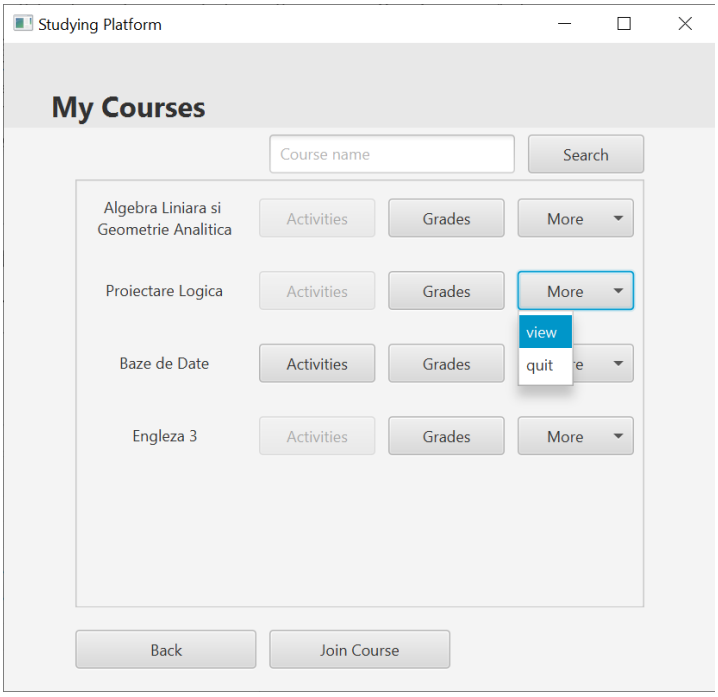
➢ Grades.

Aceasta pagina oferă profesorului posibilitatea de a nota studenții înscriși la cursurile lui.

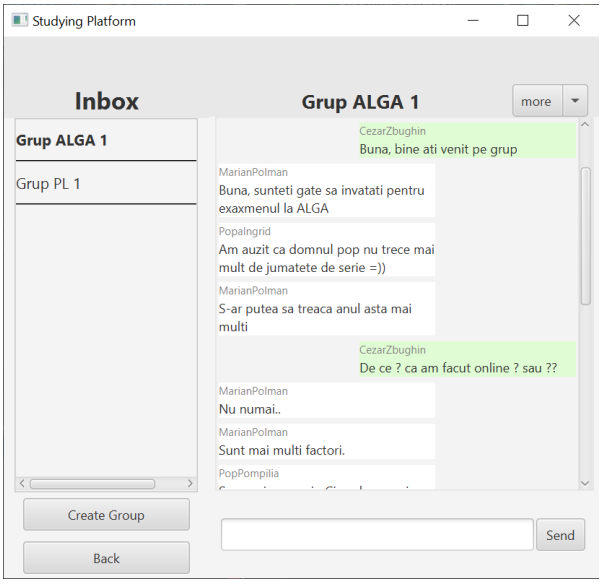


- Courses pentru studenți.

Studenții pot prin pagina „Courses” sa își vadă notele, sa se înscrie la activitățile specifice cursurilor sa vada date despre cursuri si sa părăsească cursurile la care sunt înscriși

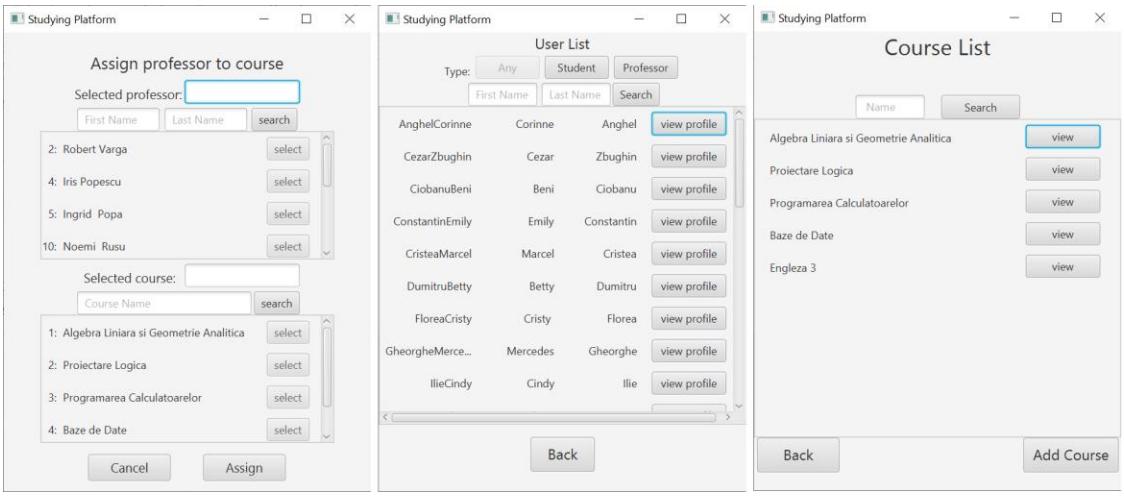


- Groups



- Admin view

Admin-ul poate sa asigneze profesori la cursuri, sa editeze useri, sa creeze sa șteargă si cursuri



6. Interogări în algebra relaţională

1. Procedura professor_get_schedule conţine interogarea:

SELECT * FROM teaching AS t INNER JOIN subject AS s ON t . professor_id = given_id AND t . subject_id = s . id

- În algebra relaţională avem:

ρ_t teaching ⋈ t . professor_id = given_id AND t . subject_id = s . id ρ_s subject

2. Procedura get_user_by_id conţine interogarea:

SELECT al1 . id, username, password, role, cnp, first_name, last_name, phone, email, iban, contract_number, is_admin, is_super_admin, year, min_studying_hours, country, region, town, street_address, postal_code

FROM ((SELECT *
FROM user AS u
WHERE u . id = parameter)AS al1 INNER JOIN student AS s ON s . id = parameter)AS al2
INNER JOIN address AS a ON a . id = parameter

- În algebra relaţională avem:

π al1 . id, username, password, role, cnp, first_name, last_name, phone, email, iban, contract_number, is_admin, is_super_admin, year, min_studying_hours, country, region, town, street_address, postal_code
(ρ al2
(ρ al1
σ u. id = parameter
ρ_u user ⋈ s . id = parameter
ρ_s student) ⋈ a . id = parameter
ρ_a address)

3. Procedura professor_get_subjects conţine interogarea:

SELECT s.id, s.name, s.description, s.has_lecture, s.has_seminar, s.has_lab, s.date_start, s.date_end, al1.students_capacity, al1.lecture_weight, al1.seminar_weight, al1.lab_weight, al1.schedule_seminar_day, al1.schedule_lecture_day, al1.schedule_lab_day, al1.schedule_seminar_hour, al1.schedule_lecture_hour, al1.schedule_lab_hour, al1.finished_schedule, al1.lecture_duration, al1.seminar_duration, al1.lab_duration
FROM (SELECT * FROM teaching t
WHERE t.professor_id = given_id) AS al1 JOIN subject s ON al1.subject_id = s.id;

- În algebra relaţională avem:

π s . id, s . name, s . description, s . has_lecture, s . has_seminar, s . has_lab, s . date_start, s . date_end, al1 . students_capacity, al1 . lecture_weight, al1 . seminar_weight, al1 . lab_weight, al1 . schedule_seminar_day, al1 . schedule_lecture_day, al1 . schedule_lab_day, al1 . schedule_seminar_hour, al1 . schedule_lecture_hour, al1 . schedule_lab_hour, al1 . finished_schedule, al1 . lecture_duration, al1 . seminar_duration, al1 . lab_duration

(ρ al1
σ t . professor_id = given_id
ρ_t teaching ⋈ al1 . subject_id = s . id
ρ_s subject)

4. Procedura get_grades conţine interogarea:

```
SELECT grade_lecture, grade_seminar, grade_lab
FROM studying
WHERE student_id = pstudent_id AND subject_id = psubject_id
```

- În algebra relaţională avem:

π grade_lecture, grade_seminar, grade_lab
σ student_id = pstudent_id AND subject_id = psubject_id (studying)

5. Procedura get_student_joinable_groups conţine interogarea:

```
SELECT g.id, g.subject_id, g.name
FROM studying_group g WHERE  g.subject_id in (select subject_id from studying st where
st.student_id = given_id) AND g.id NOT IN (select group_id from student_group sg where
sg.student_id = given_id);
```

- În algebra relaţională avem:

π g.id, g.subject, g.name
σ g.subject_id ∈ (π subject_id σ st.student_id = given_id (studying)) AND g.id ∉ (π group_id σ sg.student_id = given_id) (studying_group)

7. Concluzii şi dezvoltări ulterioare

Aşadar, aplicaţia noastră ne permite să gestionăm cu uşurinţă date despre profesori şi studenţi, interacţiunea cu aceasta realizându-se prin intermediul interfeţei grafice.

Îmbunătățiri:

• Trecerea Claselor service in POO. In faza actuală aceste servicii sunt mai mult niște colecții de metode statice. În practica exista un SuperService care conține instanțe ale claselor serviciu care la rândul lor pot conține alte servicii.

• Paginarea fiecărei liste. La un volum mare de date este posibil ca listele sa depășească limitele de ram. De exemplu, Users List încarcă toți userii din baza de date, la 10 milioane de useri este o problema.

• Adăugarea de view pentru ultimul mesaj de pe grup. Si interogarea bazei de date doar daca mesajul ultim diferă.

• Reciclarea unor fișiere fxml asemănătoare pentru a evita codul duplicat

Dezvoltări ulterioare:

• Posibilitatea de a urca fișiere pe chat

. • Sistem de notificări pentru studenti/profesori.