

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA: Tehnologia informației

Aplicație interactivă pentru realizarea prezentei

Nume si prenume : Galan Ionut Andrei 1410A

An universitar 2019-2020

Cuprins

Capitolul 1. Scopul documentului.....	1
Capitolul 2. Continutul Documentului.....	2
Capitolul 3. Proiectarea Bazei de date.....	3
3.1. Entitati.....	3
3.2. Relații între Entitati.....	4
3.3. Realizarea și implementarea relațiilor.....	5
3.4. DML (data manipulation language).....	6
3.5. Trigger.....	7
Capitolul 4. Backend.....	10
Capitolul 5. Frontend.....	12
Concluzii.....	15
Bibliografie.....	16
Anexe.....	17

Capitolul 1. Scopul documentului

Acest document are intenția de a descrie caracteristicile și implementarea Aplicației de realizarea a prezentei online în cadrul cursurilor, laboratoarelor și a seminariilor în mediul academic. Descrierea va prezenta partea de interfață cât și partea de implementare a funcționalităților oferite de acesta. Pe parcursul acestui document ne vom raporta mai mult la modul de lucru cu baza de date celelalte capitole putând fi tratate ulterior.

Scopul aplicației este de a digitaliza și centraliza procesul de realizare a prezentei în cadrul cursurilor, laboratoarelor și a seminarilor unor facultati. O imbunatatire ulterioara a aplicației poate oferi și date statistice despre numărul persoanelor prezente și activitatea acestora în cadrul prelegerilor.

Capitolul 2. Conținutul Documentului

Documentul conține informații utile despre modul în care a fost creată aplicația, tehnologiile folosite și cum trebuie utilizată acesta de către persoanele cărora se adresează. Se va analiza modul de conexiune cu baza de date, interogarea bazei de date, schema logică și relațională a tabelor. De asemenea va fi prezentată și interfața aplicației cât și partea de business logic.

Capitolul 3. Proiectarea Bazei de date

3.1. Entitati

Entitatile pe care le-am folosit în realizarea aplicației sunt următoarele: Classrooms, Teachers, Groups, Students, Attendance, Attendance Lists.

Classrooms

Entitatea conține campurile nume clasei și id. Id-ul este cheie primara. Pentru o mai buna sau stricta implementare se poate adauga constrangerea de *UNIQUE* pe câmpul nume al acestei entitati.

Teachers

Tabela Teachers conține informații despre numele profesorului și Id-ul acestuia care de asemea este cheie primara. Informațiile suplimentare se pot adauga pe parcursul dezvoltarii proiectului (disciplina/ discipline și informații despre profesor).

Groups

Entitatea Groups reprezentata de campurile id-ul și nume grupa ofera infomatii despre grupa în care se afla studentul. De asemenea am setat constrangerea de *UNIQUE* pe câmpul nume grupa. Acesta poate fi parsat astfel încât să se determine anul în care se afla studentul. Acesta operație poate fi evitata și prin crearea unui alte tabele care sa conțină și anul, tabele cu specializare , și programul de studiu.

Students

Tabelul Stutdents conține id-ul unul student și numele acestuia care este de tip *VARCHAR* cu o dimensiunea 250 de caractere, codul de indentitate al fiecarui student format de 7 cifre este de aseama un câmp care are setata constrangerea de *UNIQUE*.

Attendance Lists

Tabele conține numele numele listei de prezenta, tipul listei de prezenta(curs, laborator, seminar sau alt tip de prelegere), săptămâna în care s-a realizat prezenta. Numele listei de prezenta conține disciplina și grupele care participa la prelegere. Tebela acesta este utilizata pentru a identifica ușor fiecare lista de prezenta, după modul de indexare a acestora în dosare în funcție de disciplina, grupa/grupe, săptămâna. Poate fi asociata cu titlul din antetul documentului.

Attendance

Entitatea attendance sau prezenta conține informații despre fiecare student care este adaugat pe o lista de prezenta. Mai exect acesta tabela stocheaza toți studenți care sunt trecuti pe vreo lista de prezenta. Acesta tabela este strâns legată de Attendance Lists care ne ofera informații generale despre prezenta.

3.2. Relații între Entități

În subcapitolul anterior am descris fiecare entitate în parte iar în acest capitol vom aduce la cunoștință relațiile dintre entități și eventualele tabele suplimentare folosite pentru o bună înțelegere și o simplificare a întregii arhitecturi.

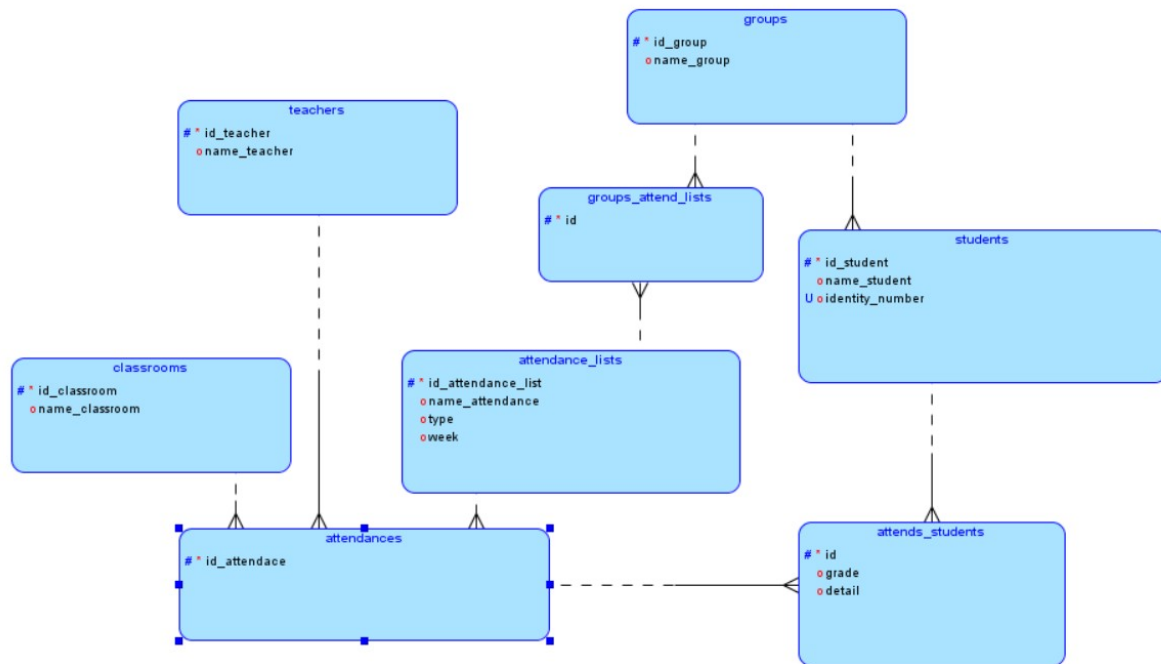


Figura 3.1 Schema Logică de Relații între entități

În jurul tabelelor *attendance_lists*, *attends_students* și *attendances* se realizează toate funcționalitățile principale ale aplicației.

Tabela *groups* are o relație de M:M cu tabela *attendance_lists* ceea ce pune în evidență faptul că mai multe grupe se pot afla pe aceeași listă de prezență și mai multe liste de prezență pot conține o grupă. Această relație a fost realizată prin intermediul unei tabele *groups_attend_lists*.

Attendance_lists are o relație de 1:M cu tabela *groups_attend_lists* ceea ce va permite ca mai multe grupe să fie pe o listă de prezență astfel încât în momentul în care se selectează o grupă aceasta să aibă posibilitatea de a identifica toți studenții din acea grupă. Această mapare oferă posibilitatea navigării și în mod invers în momentul în care pentru o grupă se dorește să se obțină o listă cu toate prezențele la acea grupă.

Attendances este o tabelă intermediară ce permite unei liste de prezență să aibă profesori și săli de clasă diferite. Aceasta are relații de 1:M cu *teachers* și de 1:M cu *classrooms*.

Attends_students este o tabelă care va conține toți studenții care se află pe toate listele de prezență. Așadar ea va avea relații 1:M cu tabela *attendances* care este o tabelă extinsă a tabelului *attendance_lists* la care se adaugă informații suplimentare cu privire la cine ține ora și unde se desfășoară aceasta. De asemenea este în relație de 1:M cu tabela *students*.

3.3. Realizarea și implementarea relațiilor

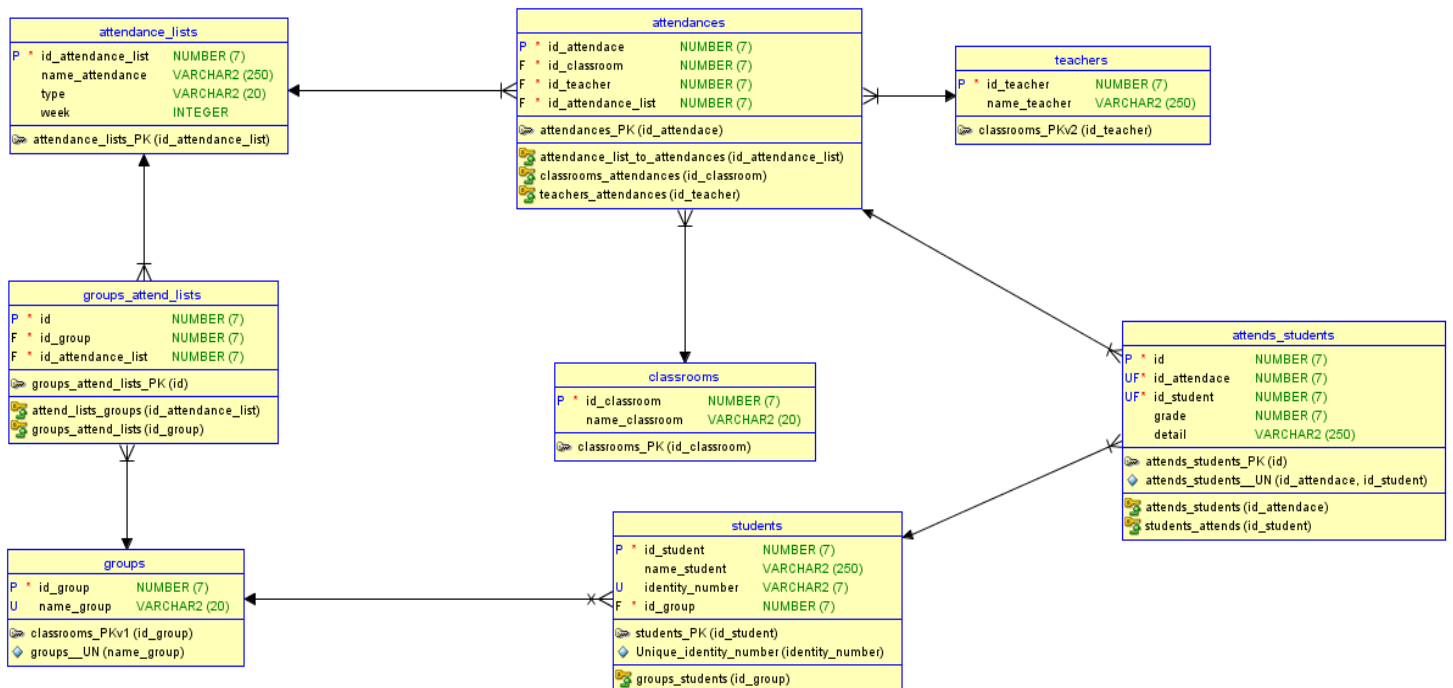


Figura 3.2 Relațiile între tabele la nivel Relational

În Figura 3.2 se observă relațiile între tabele și constrangerile aferente acestor relații. Aici este mai vizibilă utilizarea tabelor suplimentare sau intermediere.

Totodată mai sunt vizibile și celelalte constrângeri cât și denumirile acestora, tipurile de date utilizare pentru proprietățile entitatilor și dimensiunea acestora dacă este cazul.

Trebuie de precizat faptul că unele constrângeri poate nu sunt atât de vizibile dar acestea impun utilizatorului anumite condiții pe care acesta trebuie să le ia în considerare la implementare.

Aceste constrângeri sunt:

- *attends_students_UN* (id_attendance și id_student): nu permite introducerea pe lista de prezență a unui student de 2 ori.
- *Unique_identity_number* (identity_number): constrângerea de unique are rolul de a face acest câmp unic, acesta fiind numărul matricol al studentului, element unic de identificare a unui student.
- *groups_UN* (name_group): numele unei grupe este unic.

3.4. DML (data manipulation language)

Operațiile aferente de înregistrare a unor noi entități se realizează prin intermediul procedurilor împărțite pe pachete în funcție de tabele asupra cărora se efectuează operația.

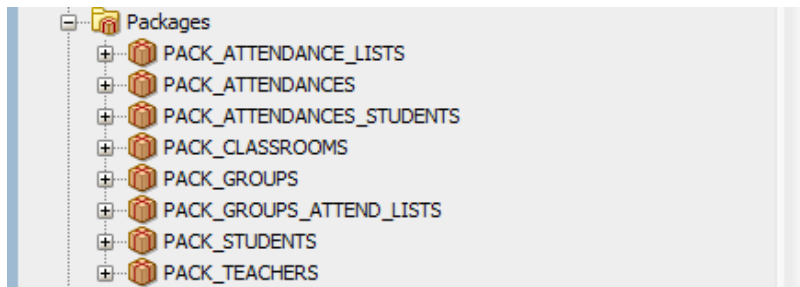


Figura 3.4.1 Organizarea pachetelor

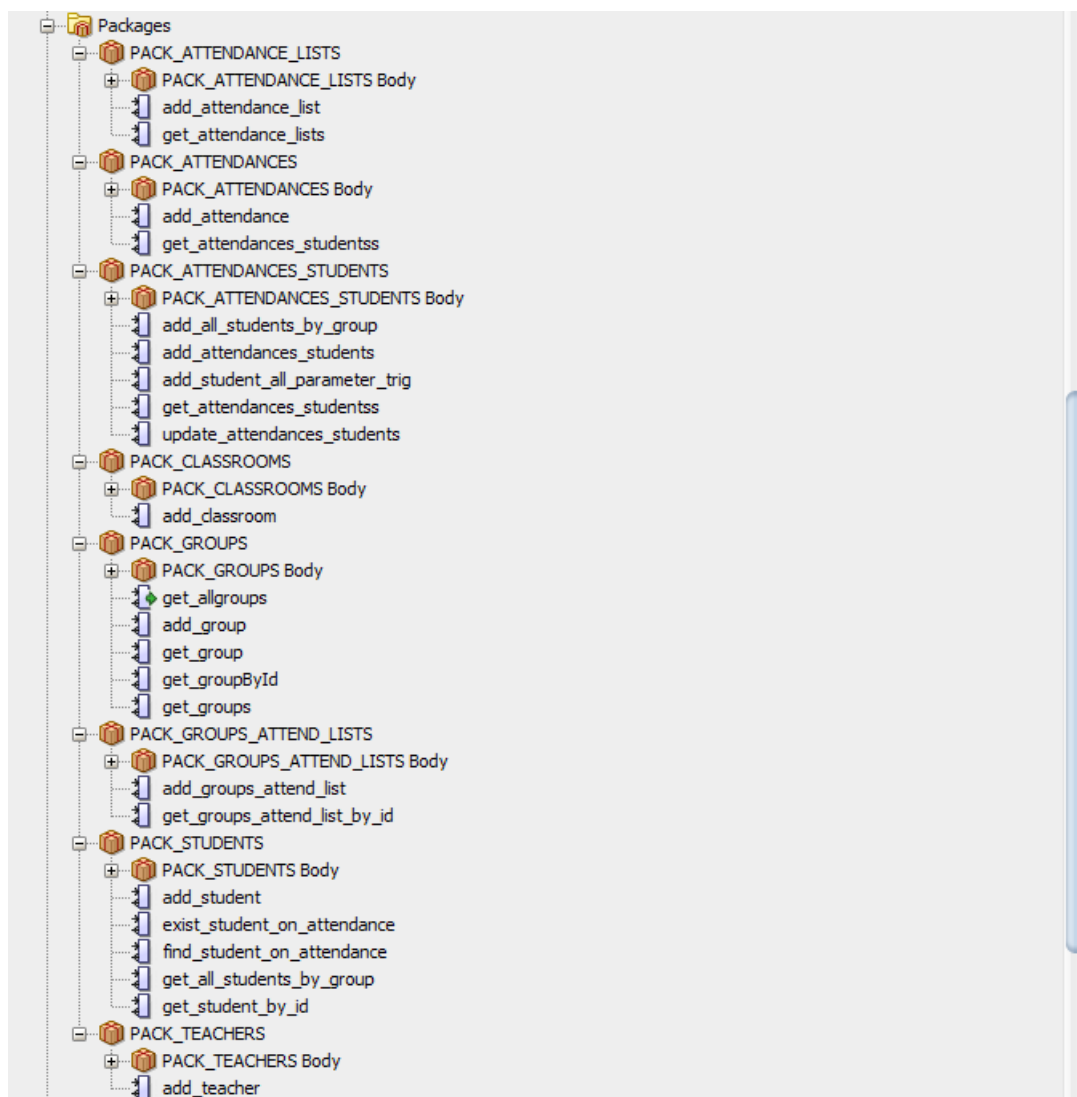


Figura 3.4.2 Procedurile create

3.5. Trigger

Utilizarea unui trigger pentru inserarea automata a tuturor studenților dintr-o grupa selectata de utilizator.

```
create or replace TRIGGER add_all BEFORE
  INSERT ON trigger_table_store
  FOR EACH ROW
BEGIN
  IF :new.names > 0 THEN

pack_attendances_students.add_all_students_by_group(:new.names,:new.groupn);
  END IF;
END;
```

Figura 3.5.1 Crearea trigger

Utilizarea unei tabele *trigger_table_store* suplimentare pentru lansarea triggerului care va apela o procedura care insereaza în lista de prezenta data ca argument toți studenți dintr-o grupa care este și ea transmisa ca argument. Tabela *trigger_table_store* conține numele grupei și id-ul listei de prezenta. Am utilizat acesta metoda de a crea un tabel suplimentar deoarece execuție insertului care ar trebui sa lanseze trigger-ul încălca constrangerile stabilite din data modeler. Acesta situație neputând fi evitata în modul în care am făcut acesta proiectare, sau soluția ar fi destul de complicata.

```
PROCEDURE add_student_all_parameter_trig (
  v_group_name      IN   VARCHAR,
  v_id_attendance   IN   INTEGER
) IS
BEGIN
  INSERT INTO trigger_store VALUES (
    NULL,
    v_group_name,
    v_id_attendance
  );

END add_student_all_parameter_trig;
```

Figura 3.5.2 Lansarea trigger

Procedura *add_student_all_parameter_trig* va fi apelata în backend și se vor specifica parametrii ceruti. Functia va fi apelata doar în momentul în care utilizatorul alege din fronted optiunea ALL.

```

PROCEDURE add_all_students_by_group (
    v_id_attendance_list IN INTEGER,
    v_group_name         IN  VARCHAR
) IS

    v_id_attendance  INTEGER;
    v_id_group       INTEGER;
    v_id_student     INTEGER;
    v_exists         INTEGER;
    CURSOR c1 IS
    SELECT
        id_student
    FROM
        students
    WHERE
        id_group = v_id_group;

BEGIN
    pack_groups.get_group(v_group_name, v_id_group);
    SELECT
        id_attendance
    INTO v_id_attendance
    FROM
        attendances
    WHERE
        id_attendance_list = v_id_attendance_list;

    OPEN c1;
    LOOP
        v_exists := 0;
        FETCH c1 INTO v_id_student;
        EXIT WHEN c1%notfound OR c1%notfound IS NULL;

        pack_students.exist_student_on_attendance(v_id_student, v_id_attendance,
v_exists);

        IF v_exists = 0 THEN
            INSERT INTO attends_students VALUES (
                NULL,
                v_id_attendance,
                v_id_student,
                0,
                'p'
            );
        END IF;
    END LOOP;
    CLOSE c1;
END add_all_students_by_group;

```

Figura 3.5.3 Funcționalitatea trigger-ului

Figura 3.5.3 permite inserarea într-o lista de prezenta a tuturor studenților din grupa. Inserarea se face în interiorul unui cursor care va returna o lista de studenți din acea grupa.

Funcția *pack_students.exist_student_on_attendance(v_id_student, v_id_attendance, v_exists)*; va verifica dacă nu exista deja acel student inserat în lista de prezenta. În caz contrar poate fi generată o constrângere de UNIQUE pe campurile *id_attendance* și *id_student*.

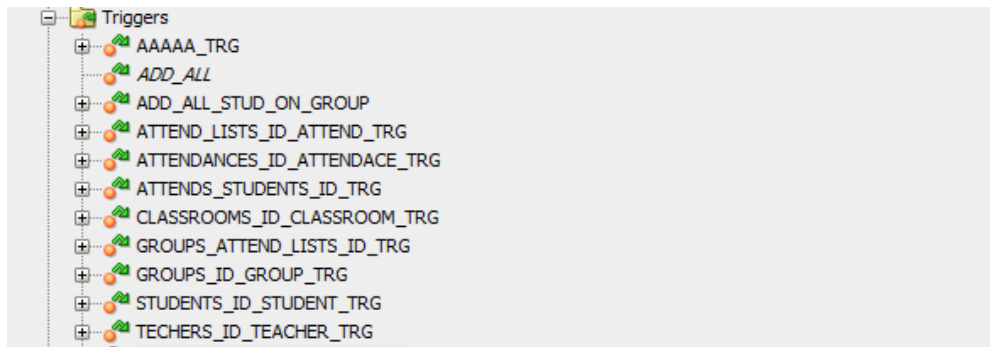


Figura 5.3.4 Triggere utilizate în aplicație

Capitolul 4. Backend

Partea de business logic a fost făcută integral în java, java spring, spring boot iar pentru apelarea procedurilor create în SqlDeveloper am folosit jdbc. Arhitectura este una conform modelului MVC în care se respecta principiile de SOLID.

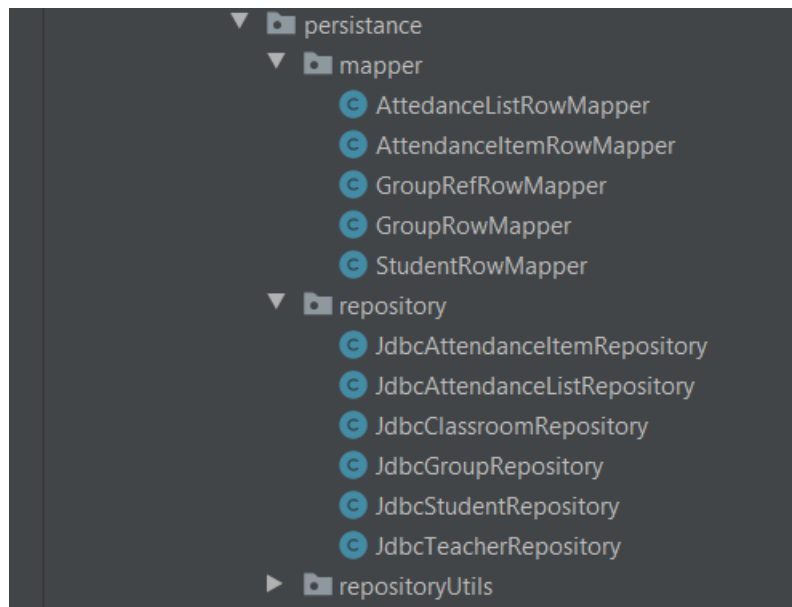


Figura 4.1 Maparea tabelelor în obiecte

```
@PostConstruct
void init() {
    jdbcTemplate.setResultsMapCaseInsensitive(true);

    simpleJdbcCall = new SimpleJdbcCall(jdbcTemplate)
        .withCatalogName("pack_attendances_students")
        .withProcedureName("add_attendances_students");

    simpleJdbcCallUpdate=new SimpleJdbcCall(jdbcTemplate)
        .withCatalogName("pack_attendances_students")
        .withProcedureName("update_attendances_students");

    simpleJdbcCallGetAllById=new SimpleJdbcCall(jdbcTemplate)
        .withCatalogName("pack_attendances_students")
        .withProcedureName("get_attendances_studentss").returningResultSet
("out_lists",attendanceItemRowMapper);

    simpleJdbcCallAddAllStudents=new SimpleJdbcCall(jdbcTemplate)
        .withCatalogName("pack_attendances_students")
        .withProcedureName("add_student_all_parameter_trig");
}
```

Figura 4.2 Instantierea obiectelor care vor apela procedurile

```

public AttendanceItem saveOneStudent(AttendanceItem attendanceItem){
    log.info("Repository Save new attendance student");

    String nameAttendance=attendanceItem.getAttendanceList().getName();
    SqlParameterSource in = new MapSqlParameterSource()
        .addValue("v_identity_number",
attendanceItem.getStudents().get(0).getIdentityNumber())
        .addValue("v_attendance_list_name",nameAttendance )
        .addValue("v_grade",attendanceItem.getGrade())
        .addValue("v_detail",attendanceItem.getDetails());

    Map out = simpleJdbcCall.execute(in);

    return attendanceItem;
}

```

Figura 4.3 Adaugarea parametrilor și executarea procedurilor

```

public Optional<List<AttendanceItem>> findAllById(Long attendanceId)
{
    SqlParameterSource in = new MapSqlParameterSource()
        .addValue("v_id_attendance_list",attendanceId);

    Map out= simpleJdbcCallGetAllById.execute(in);

    ArrayList<AttendanceItem> attendanceItems = (ArrayList<AttendanceItem>)
out.get("out_lists");

    return Optional.of(attendanceItems);
}

```

Figura 4.4 Returnarea variabilei în cadrul procedurii

Figurile adaugate mai sus (4.1, 4.2, 4.3, 4.4) prezinta modul în care s-a realizat conexiunea și comunicarea cu procedurile prezentate în capitolul anterior. Toate aceste se bazează pe clasa *JdbcTemplate* și metodele acesteia care sunt un puternic mecanism de comunicare cu o baza de date și pot executa sintaxa SQL din interiorul funcțiilor java.

Capitolul 5. Frontend

Partea de interfață a aplicației a fost realizată integral prin definirea componentelor și crearea designului prin intermediu CSS și HTML. Aplicația a fost dezvoltată în React Redux care este un puternic framework dezvoltat și utilizat de marile companii precum Facebook, Instagram, Netflix și multe altele. Acesta creează un DOM virtual pe care îl populează cu anumite componente și apoi este randat către DOM paginii web. Acesta are la bază ES6 (ECMA2015).

Figura 5.1 Crearea entitatilor

Grupele și studenții vor fi adăugați de către profesor și lista de prezenta (Attendance name) va fi creată de către profesor. Sunt adăugate și constrangeri pe fiecare câmp acestea fiind echivalente și cu cele din baza de date. Constrangerile pe partea de backend lipsind. Se pot selecta mai multe grupe iar săptămâna trebuie să fie între valoarea 1 și 14 având în vedere lungimea unui semestru.

Type reprezintă tipul de prelegere care se ține seminar, laborator sau curs.

Grupele sunt disponibile în momentul în care acesta au fost create.

Create Entities Attendance

Week
Enter week

Category
seminary

Search List

Attendance Lists
--none--

Groups
--none--

Search student

Figura 5.2 Căutarea unei liste de prezenta

Student
Jhon Doe1 123456

Grade
10

Details
+2

Save Cancel View

Figura 5.2 Adaugarea unui student

Figura 5.2 prezintă partea în care un student este adăugat pe o listă de prezență. Acesta poate fi selectat dintr-un drop down conform grupei în care se afla. Acest drop down poate conține și opțiunea de ALL care va adăuga toți studenții din grupa respectivă sau pe care nu există pe prezență. Opțiunea ALL va completa lista de prezență cu toți studenții din grupa respectivă.

A doua modalitate de adăugare a unui student este selectarea sa din listă și completarea opțională a câmpurilor *grade* și *details*. În momentul în care acesta există se face un update cu valorile completate sau dacă nu este prezență este adăugat.

Butonul *View* permite vizualizarea sub formă tabelară a prezenței. Acesta va oferi informații nu doar pentru studenții din grupa selectată ci pentru toate grupele adăugate pe lista de prezență și pentru toți studenții trecuți pe acesta.

NAME	NUMBER	ATTENDANCE	GRADE	DETAILS
Galan Ionut	0757180	ALPD	0	p
Jhon Doe1	123456	ALPD	10	+2
Precop Andrei	5886643	ALPD	0	p
Schitcu Gabriel	0755642	ALPD	0	p

Figura 5.3 Vizualizare lista de prezență

Aplicația permite conectarea și unui student care are posibilitatea de a căuta, vizualiza și de a se adăuga pe o listă de prezență. Datorită faptului că aplicația nu deține un sistem de logare sigur un student poate adăuga și alți studenți. Această funcționalitate va fi completată în momentul în care fiecare student are un cont și se conectează independent. Figura 5.4

Attendance

Student

Galan Ionut 0757180

Save

Cancel

View

NAME	NUMBER	ATTENDANCE	GRADE	DETAILS
Galan Ionut	0757180	ALPD	0	p
Jhon Doe1	123456	ALPD	10	+2
Precop Andrei	5886643	ALPD	0	p
Schitcu Gabriel	0755642	ALPD	0	p

Figura 5.4 Aplicația privită din perspectiva unui student

Concluzii

Aplicația are scop didactic și poate fi o soluție pentru realizarea unei prezente centralizate și digitalizate cu eventuale date statistice cât și o evaluare a cadrelor didactice cât și a studenților.

Aplicația nu are ca scop implementarea completă a tuturor cazurilor pe care le poate prevedea o lista de prezenta și nici a tuturor functionalitatilor posibile.

Aplicație poate fi dezvoltata și poate fi o lucrare de licenta.

Bibliografie

Anexe.

Anexa 1.