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

Aplicație interactiva pentru realizarea prezentei

Nume si prenume : Galan Ionut Andrei 1410A

An universitar 2019-2020

Cuprins

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 interfata cât și partea de implementare a functionalitatilor oferite de acesta. Pe parcursul acestui document ne vom raporta mai mult la modul de lucru cu baza de date celelalte capitole putand 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 ulteriora a aplicației poate oferii și date statistice despre numărul persoanelor prezente și activitatea acestora în cadrul prelegerilor.

Capitolul 2. Continutul Documentului

Documentul conține informații utile despre modul în care a fost creata aplicația, tehnologiile folosite și cum trebuie utilizata acesta de către persoanelor careia se adreseaza. Se va analiza modul de conexiune cu baza de date, interogarea bazei de date, schema logica și relationala a tabelelor. De asemenea va fi prezentata și interfata 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 urmatoarele: 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 dezolvatii 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 acestuie 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 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 partea de sus a documentului fizic.

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 intre Entitati

În subcapitolul anterior am descris fiecare entitate în parte iar în acest capitol vom aduce la cunoștința relatiile dintre entitati și eventualele tabele suplimentare folosite pentru o buna înțelegere și o simplificare a întregii arhitecturi.

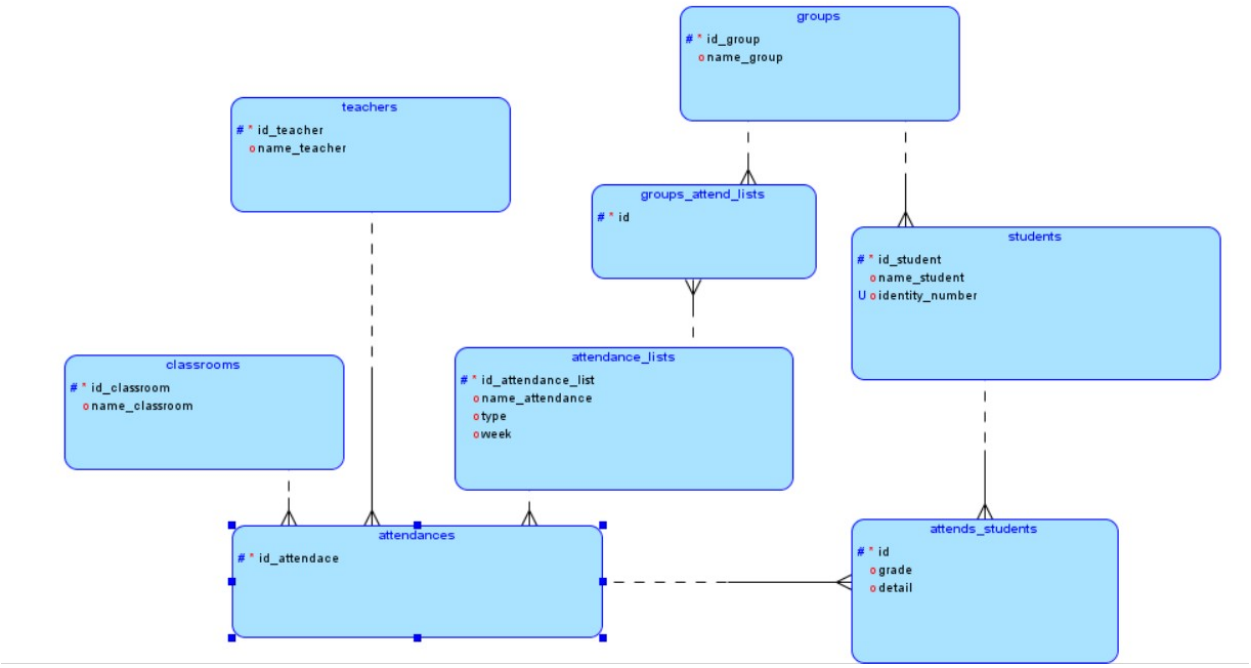


Figura 3.1 Schema Logica de Relații intre entitati

În jurul tabelelor *attendance_lists* , *attends_students* și *attendances* se realizeaza toate functionalitatile principale ale aplicației.

Tabela *groups* are o relație de M:M cu tabela *attendance_lists* ceea ce pune în evidenta faptul ca mai multe grupe se pot afla pe aceea lista de prezenta și mai multe liste de prezenta pot conține o grupa. Acesta relație a fost realizata 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 lista de prezenta astfel încât în momentul în care se selecteaza o grupa acesta sa aibă posibilitatea de a identifica toți studenți din acea grupa. Acesta mapare ofera posibilitatea navigarii și în mod invers în momentul în care pentru o grupa se dorește să se obțină o lista cu toate prezentele la acea grupa.

Attendances este o tabela intermediara ce permite unei liste de prezenta sa aibă profesori și sali de clasa diferite. Acesta are relații de 1:M cu *teachers* și de 1:M cu c*lassrooms* .

Attends_students tabela care va conține toți studenții care se afla pe toate listele de prezenta. Așadar ea va avea relații 1:M cu tabela *attendances* care este o tabela extinsa a tabelei *attendance_lists* la care se adauga informații suplimentare cu privire la cine ține ora și unde se desfășoară acesta ora. De asemenea este în relație de 1:M cu tabela *students*.

3.3. Realizarea și implementarea relațiilor

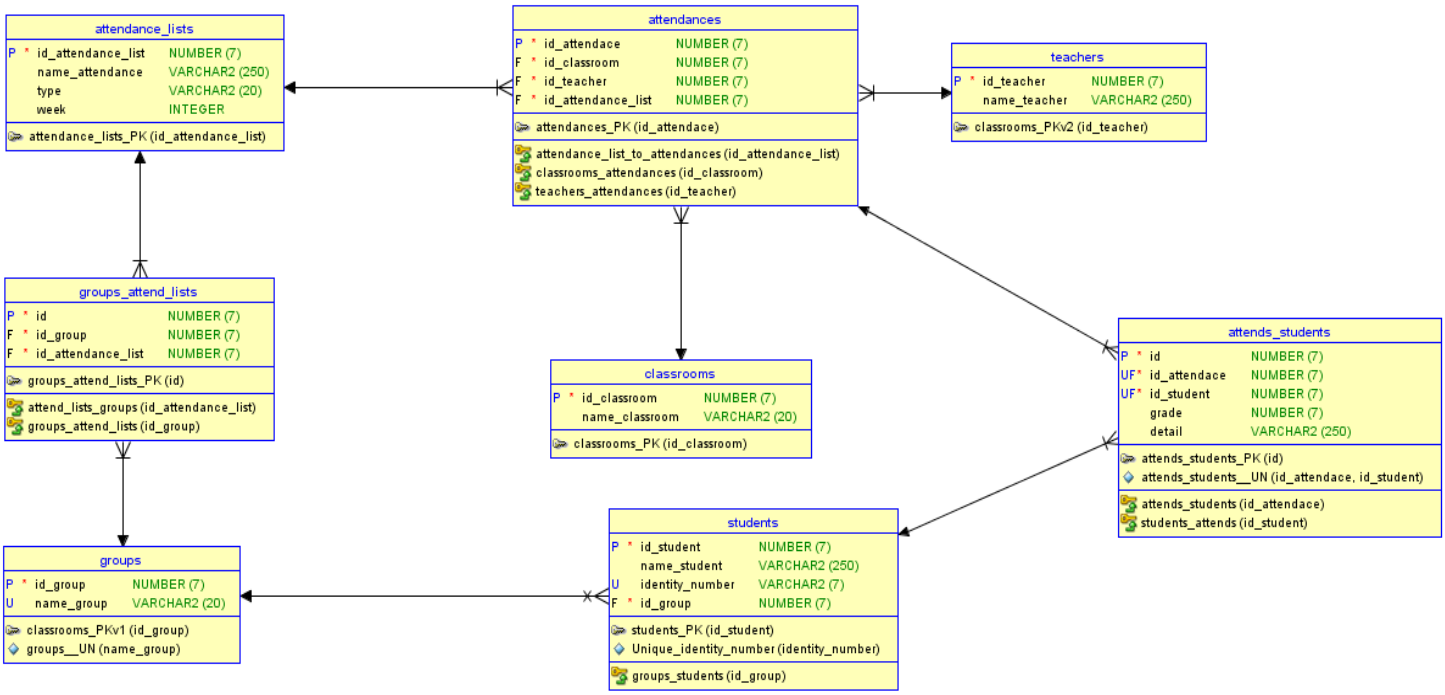


Figura 3.2 Relatiile intre tabele la nivel Relational

În Figura 3.2 se observa relatiile intre tabele și constrangerele aferente acestor relații. Aici este mai vizibila utilizararea tabelelor suplimentare sau intermediere.

Totodata mai sunt vizibile și celelalte constrangeri cât și denumirile acestora, tipurile de date utilizare pentru proprietatile entitatilor și dimensiunea acestora dacă este cazul.

Trebuie de precizat faptul ca unele constrangere poate nu sunt atât de vizibele dar acestea impun utilizatorului anumite condiții pe care acesta trebuie sa le ia în considerare la implementare.

Aceste constrangeri sunt:

- *attends_students_UN* (id_attendance și id_student): nu permite introducere pe lista de prezenta a unui student de 2 ori.
- *Unique_identity_number* (identity_number): constrangerea 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)

Operatiile aferente de inregistrare a unor noi entitati se realizeaza prin intermediul procedurilor împărțite pe pachete în funcție de tabele asupra carola se efectueaza operația.

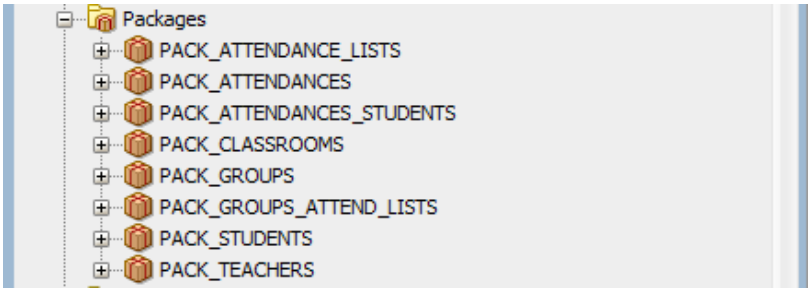


Figura 3.4.1 Organizarea pachetelor

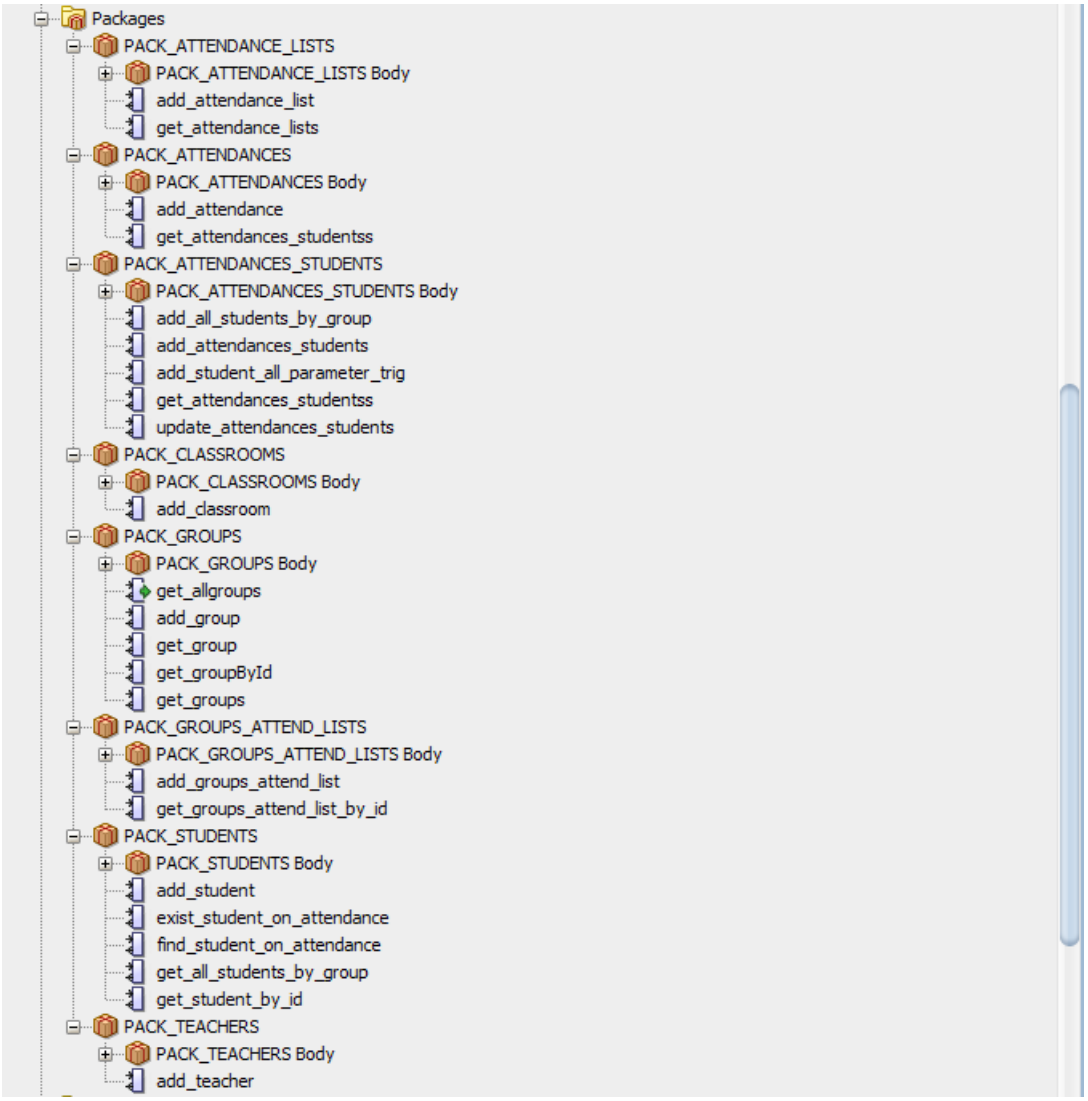


Figura 3.4.2 Procedurile create

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 toți studenți dintr-o grupa. Lista de prezenta și grupa sunt transmise ca și argument transmise ca și 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_attedance_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_attendace
        INTO v_id_attendance
        FROM
            attendances
        WHERE
            id_attendance_list = v_id_attedance_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 Functionalitatea trigger-ului

Figura 3.5.3 permite inserarea intr-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.

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

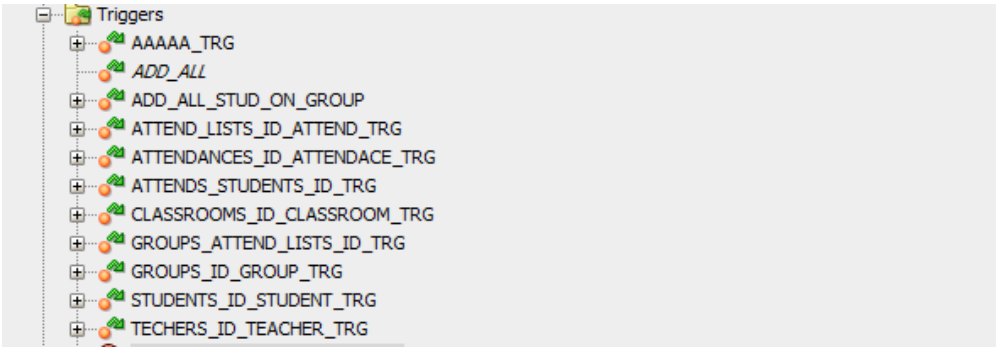


Figura 5.3.4 Triggere utilizate în aplicație

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 principile de SOLID.

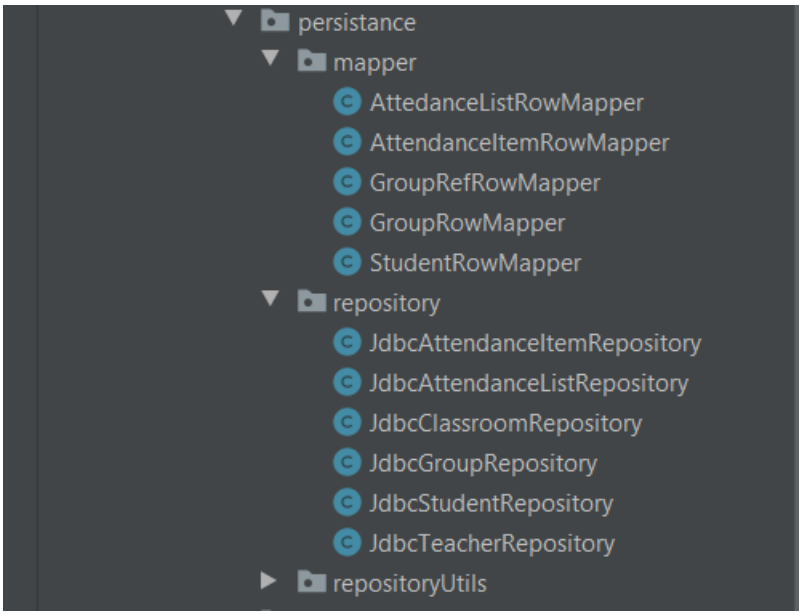


Figura 4.1 Maparea tabelelor în obiecte

```
  @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 nameAttedance=attendanceItem.getAttendanceList().getName();
            SqlParameterSource in = new MapSqlParameterSource()
                    .addValue("v_identity_number",
attendanceItem.getStudents().get(0).getIdentityNumber())
                    .addValue("v_attendance_list_name",nameAttedance )
                    .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 cumunicarea 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 functiilor java.

Capitolul 5. Frontend

Partea de interfata a aplicație a fost realizata integral prin definirea componentelor și crearea designului prin intermediu css și html. Aplicația a fost dezvoltata în React Redux care este un puternic framework dezvoltat și utilizat de marile companii precum Facebook, Instagram, Netflix și multe altele. Acesta creaza un DOM virtual pe care îl populeaza cu anumite componente și apoi este randat către DOM paginii web. Acesta are la baza ES6 (ECMA2015).

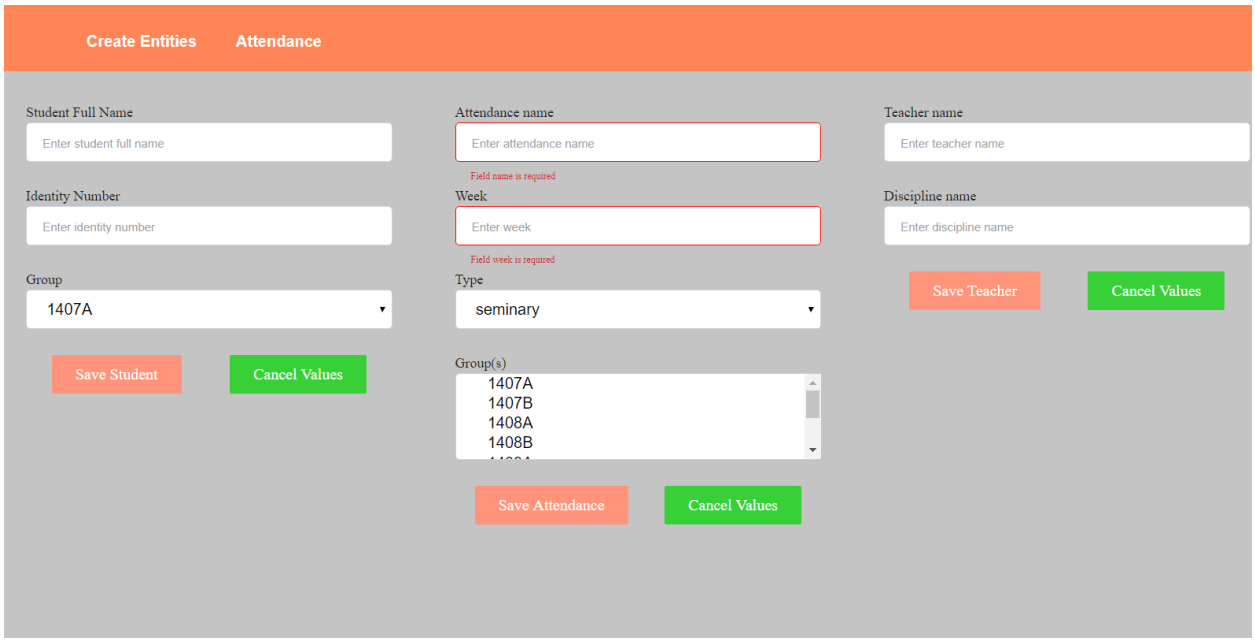


Figura 5.1 Crearea entitatilor

Grupele și studenții vor fi adăugați de către profesor și lista de prezenta (Attendance name) va fi creata de către profesor. Sunt adaugate 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 intre valoarea 1 și 14 având în vedere lungimea unui semestru.

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

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

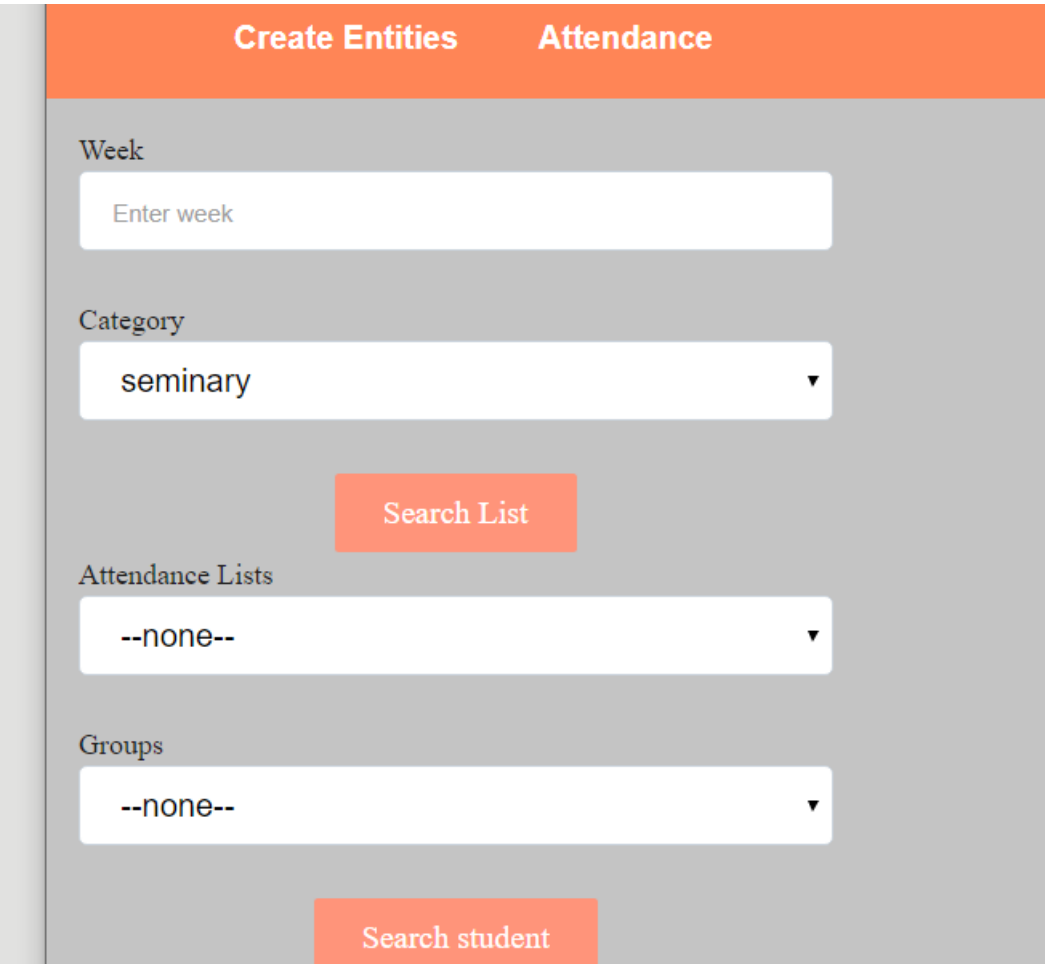


Figura 5.2 Căutarea unei liste de prezenta

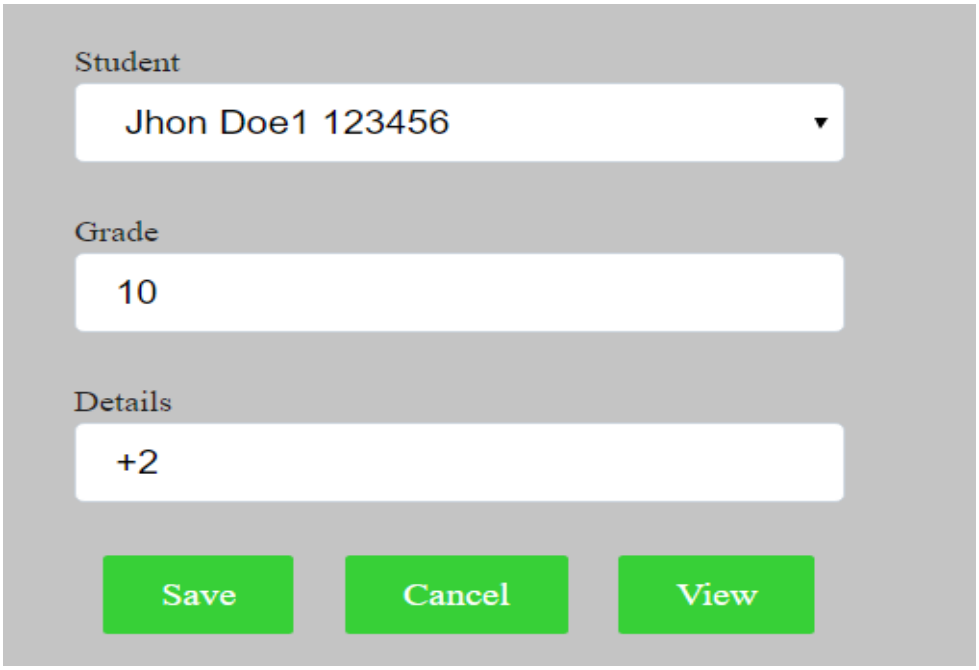
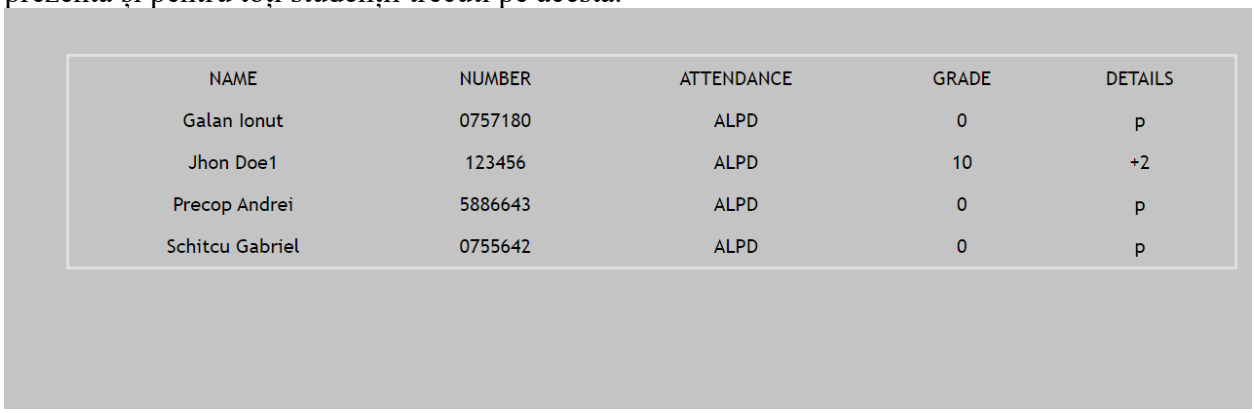


Figura 5.2 Adaugarea unui student

Figura 5.2 prezinta partea în care un student este adaugat pe o lista de prezenta. Acesta poate fi selectat dintr-un drop down conform grupei în care se afla. Acest drop down poate conține și optiunea de ALL care va adauga toți studenții din grupa respectiva sau pe care nu exista pe prezenta. Optiunea ALL va completă lista de prezenta cu toți studenți din grupa respectiva.

A doua modalitate de adaugare a unui student este selectarea sa din lista și completarea optionala a campurilor *grade* și *details*. În momentul în care acesta exista se face un update cu valorile completate sau dacă nu este prezenta este adaugat.

Butonul *View* permite vizualizarea sub forma tabelara a prezentei. Acesta va oferii informații nu doar pentru studenții din grupa selectata ci pentru toate grupele adaugate pe lista de prezenta și pentru toți studenții trecuti 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 prezenta

Aplicația permite conectarea și unui student care are posibilitatea de a cauta, vizualiza și de a se adauga pe o lista de prezenta. Datorită faptului ca aplicație nu deține un sistem de logare sigur un student poate aduga și alți studenți. Acesta functionalitate va fi completă în momentul în care fiecare student are un cont și se conecteaza independent. Figura 5.4

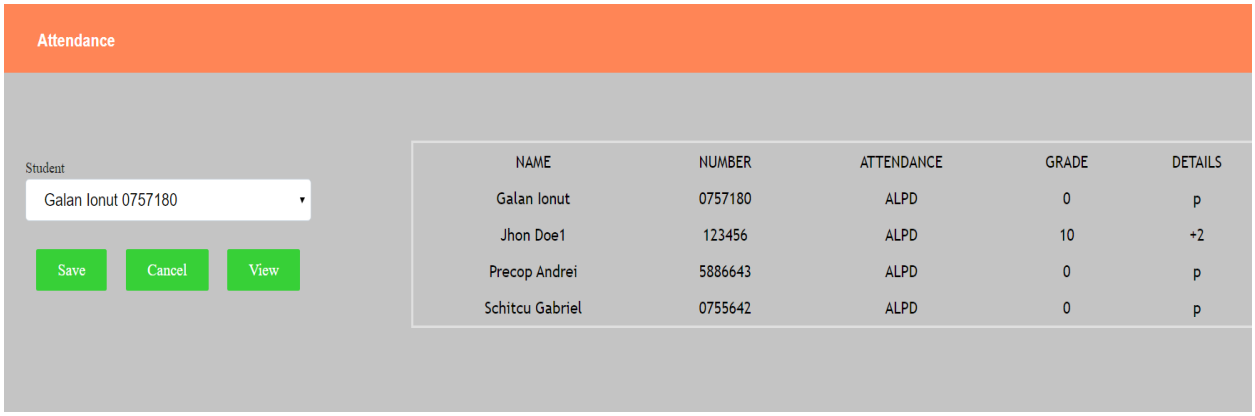


Figura 5.4 Aplicația privita din perspectiva unui student

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 toturor 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.