

# Orar - Calitatea sistemelor software

## I. Implementare

### I.1. Prezentare generala a entitatilor

Aplicația are ca scop generarea unui orar pentru studenții de la facultate. Aceasta a fost dezvoltată în Python cu ajutorul unei baze de date de tip PostgreSQL. Fiind o aplicație de tip Single Page, navigarea facilă din pagina în pagina.

O primă componentă a aplicației noastre este reprezentată de către entitățile pe care le-am definit. Structura acestora este identică în cadrul bazei de date. Pentru fiecare entitate, operațiile pe care le efectuăm în cadrul repository urilor noastre sunt următoarele:

1. Profesor
  - creare a entitatii în baza de date
  - diferite selecturi din baza de date necesare pentru procesarea și agregare a datelor despre această entitate
2. Student
  - creare a entitatii în baza de date
  - select pentru toți studenții din baza de date
  - ștergere a entitatii din baza de date
3. Disciplina
  - creare a entitatii în baza de date
  - diferite selecturi din baza de date necesare pentru procesarea și agregare a datelor despre această entitate
  - ștergere a entitatii din baza de date
4. Grupa, an, semian, sala, tip de sala, interval orar, zile din săptămână
  - diferite selecturi din baza de date necesare pentru procesarea și agregare a datelor despre această entitate
5. Orar, înregistrare în orar
  - creare a entitatii în baza de date
  - diferite selecturi din baza de date necesare pentru procesarea și agregare a datelor despre această entitate

Exemplu de entitate:

```
4 usages  AdrianSmau +1
class Student:
    AdrianSmau +1
    def __init__(self, id: int, first_name: str, last_name: str, study_year: int, semi_year: int,
        student_group: str):
        self.id = id
        self.first_name = first_name
        self.last_name = last_name
        self.study_year = study_year
        self.semi_year = semi_year
        self.student_group = student_group
```

Conexiunea la baza de date necesara pentru toate aceste operații este făcută in felul următor:

```
rares01 +1 *
def connection():
    conn = psycopg2.connect(
        host='localhost',
        port='5433',
        user='power-user',
        password='root',
        database='Oran'
    )

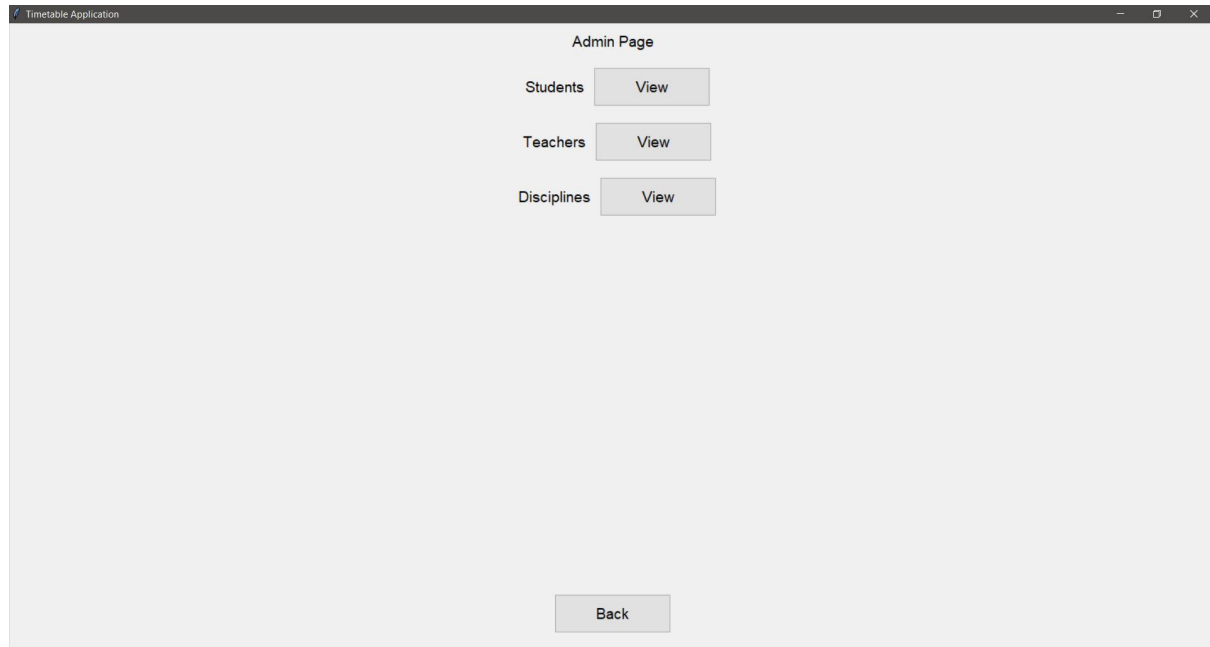
    conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
    return conn
```

## I.2. Flow-uri de utilizare și componente de UI

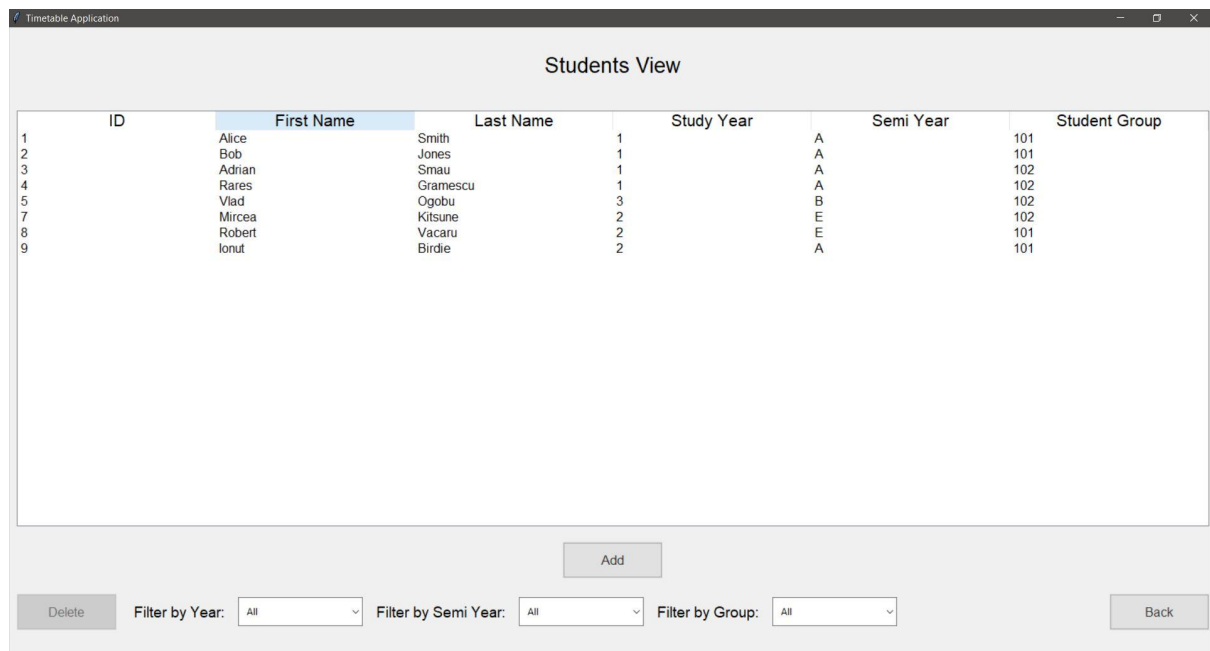
La pornirea aplicației, utilizatorul poate selecta una dintre cele doua componente ale programului: Admin Page sau Timetable Page.

## I.2.1. Admin Page

Componenta de Admin Page permite vizualizarea, filtrarea, adăugarea și ștergerea studentilor, profesorilor și disciplinelor, oferind posibilitatea navigarii între pagini.



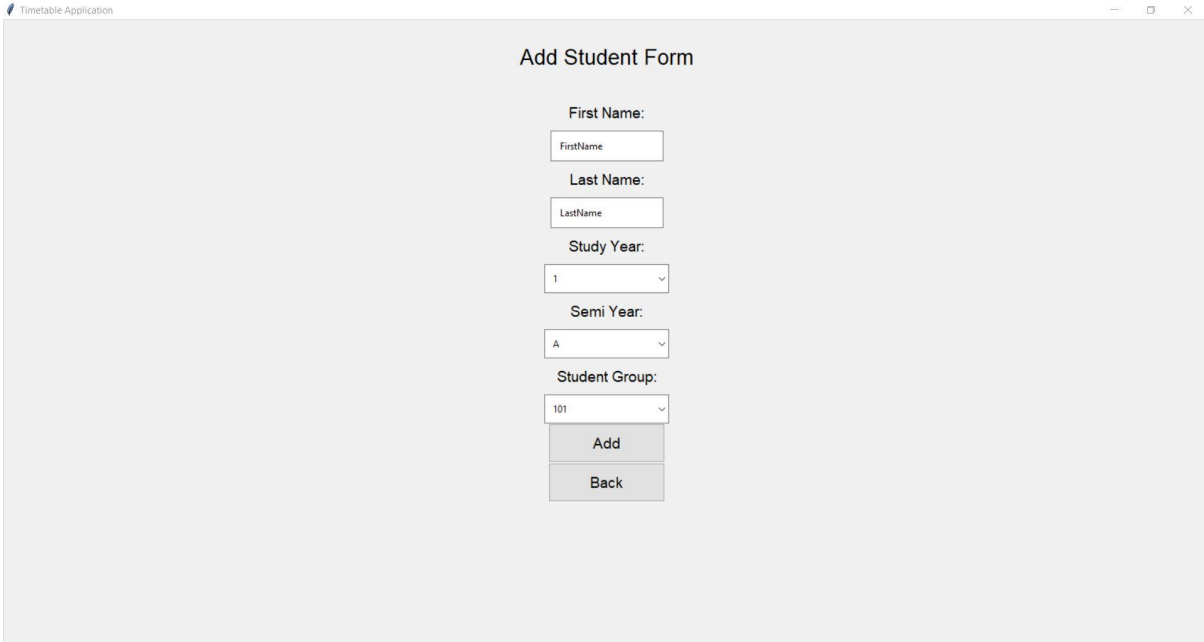
La actionarea oricarei dintre cele trei sectiuni, utilizatorul este intampinat cu un ecran ce incorporeaza atat datele curente din baza de date, cat și alte butoane ce, atunci cand sunt actionate, modifica baza de date. De asemenea, exista mecanisme de filtrare a datelor pentru a facilita navigarea printre date. In cele ce urmează, vom oferi capturi de ecran pentru a exemplifica aceste functionalitati pentru pagina Students, celelalte doua fiind identice, însă pentru alte tipuri de entități.



Aceste date care sunt afișate în interfața sunt deja agregate. De exemplu, în cazul de față, valorile pentru Study Year, Semi Year și Student Group sunt preluate din baza de date, agregate într-un obiect de tip Student și mai apoi afișate corespunzător.

La selectarea unuia dintre aceste entități și acționarea butonului de Delete, aceasta va fi ștearsă din baza de date iar view-ul cu entitățile va fi refacut, astfel încât schimbările din baza de date să fie reflectate și în interfața cu utilizatorul. Butonul de Delete este dezactivat până la selectarea unui anumit record din view.

La acționarea butonului de Add, un formular de adăugare este deschis. Atunci când este cazul ca o entitate să aibă o cheie străină către un alt tip de entitate (cum ar fi, în cazul de față, Study Year, Semi Year și Student Group), este creat un dropdown cu toate obiectele de acel tip care există în baza de date. Astfel, ne asigurăm că nu pot apărea erori neprevăzute care să producă un foreign key violation.



Timetable Application

### Add Student Form

First Name:

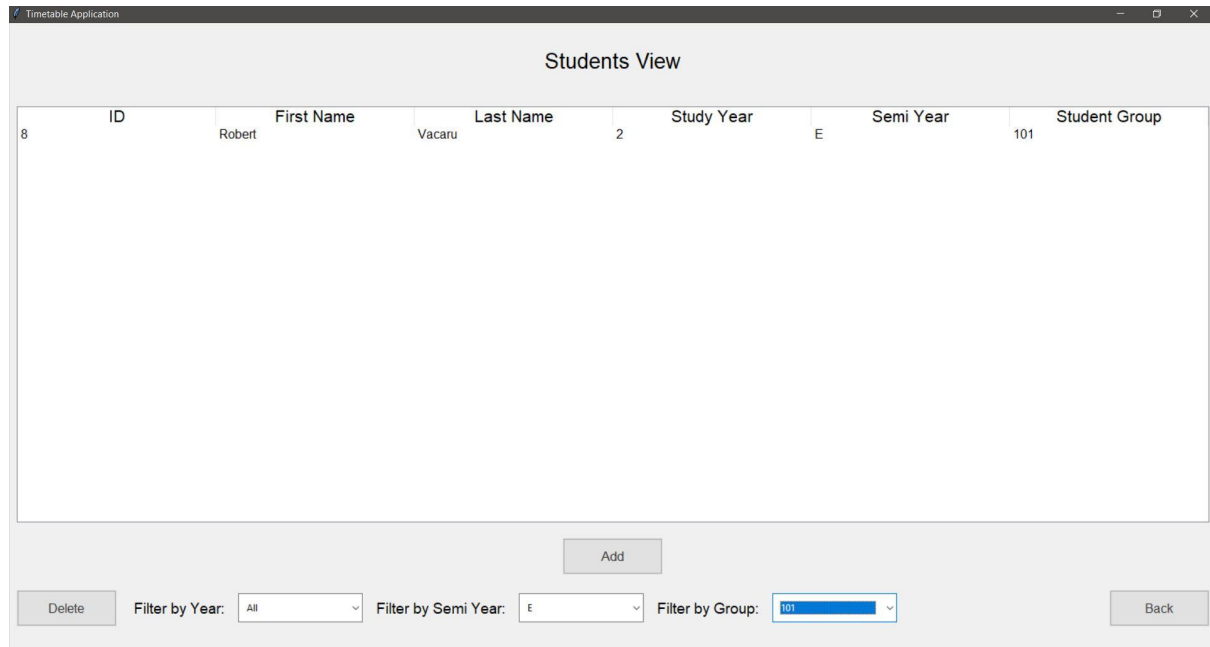
Last Name:

Study Year:

Semi Year:

Student Group:

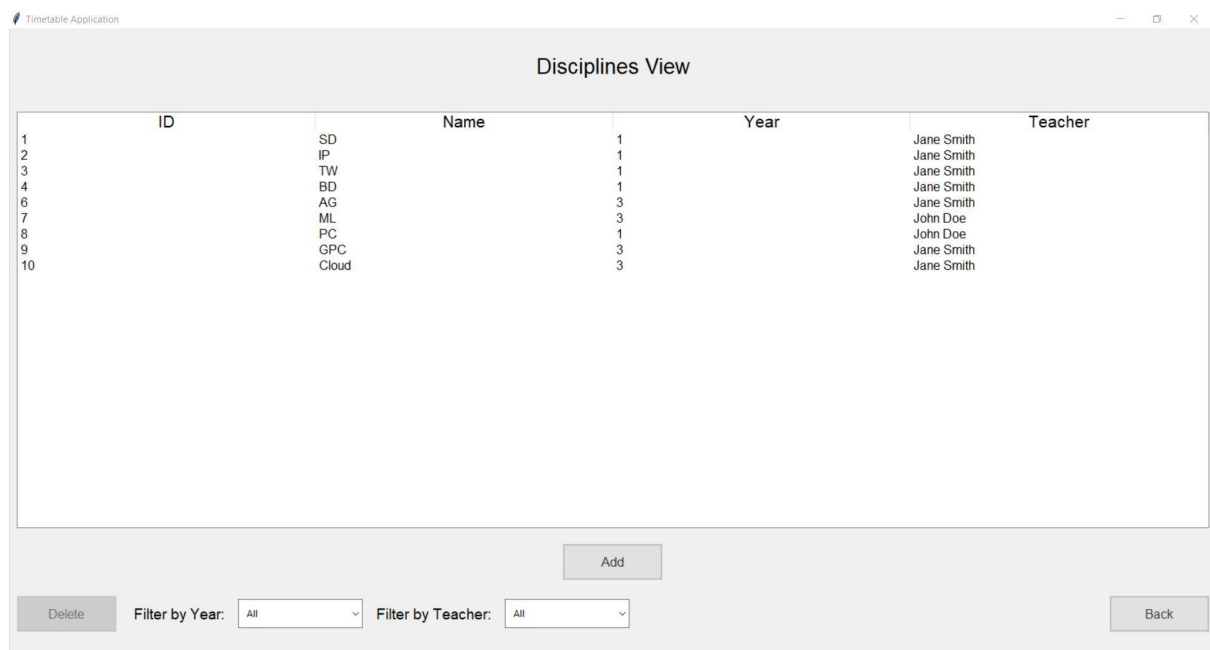
In cele din urma, filtrele din josul paginii de view sunt complet funcționale și ajuta utilizatorul sa gaseasca anumite date pe care le urmărește. Filtrele sunt create folosind dropdown-uri cu toate posibilele valori ale acelui camp, similar cu formularul de Add. De exemplu, filtrand studenții existenți după semian E și grupa 101, ne este returnat următorul rezultat:



ID	First Name	Last Name	Study Year	Semi Year	Student Group
8	Robert	Vacaru	2	E	101

Control bar: Delete, Filter by Year: All, Filter by Semi Year: E, Filter by Group: 101, Add, Back

Așa cum este menționat mai sus, celelalte pagini sunt identice si au aceleași functionalitati, cum ar fi pagina cu discipline:



ID	Name	Year	Teacher
1	SD	1	Jane Smith
2	IP	1	Jane Smith
3	TW	1	Jane Smith
4	BD	1	Jane Smith
6	AG	3	Jane Smith
7	ML	3	John Doe
8	PC	1	John Doe
9	GPC	3	Jane Smith
10	Cloud	3	Jane Smith

Control bar: Delete, Filter by Year: All, Filter by Teacher: All, Add, Back

## I.2.2. Timetable Page

Componenta de Timetable Page permite vizualizarea, filtrarea, adăugarea și generarea în format HTML a înregistrărilor în orarul facultății, oferind posibilitatea navigării între pagini.

The screenshot shows the 'Timetable Application' window. At the top, there's a 'Timetable View' title. Below it is a table with the following data:

ID	Weekday	Time Slot	Teacher	Discipline	Study Year	Semi Year	Student Group
4	Monday	12:00-14:00	Jane Smith	SD	1	A	101
5	Thursday	8:00-10:00	John Doe	Cloud	2	B	101
6	Tuesday	18:00-20:00	Jane Smith	ML	3	E	102
7	Friday	16:00-18:00	John Doe	BD	1	B	102
8	Friday	14:00-16:00	Jane Smith	AG	3	A	102
9	Friday	12:00-14:00	Jane Smith	AG	3	B	101
10	Wednesday	18:00-20:00	John Doe	TW	2	A	101

Below the table is an 'Add' button. At the bottom, there's a 'Generate HTML' button and a filter section with the following controls:

- Filter by Weekday: All (dropdown)
- Filter by Time Slot: All (dropdown)
- Filter by Teacher: All (dropdown)
- Filter by Discipline: All (dropdown)
- Filter by Year: All (dropdown)
- Filter by Semi Year: All (dropdown)
- Filter by Group: All (dropdown)
- Back button

După cum se poate vedea din poza de mai sus, fiecare înregistrare în orar are un ID unic, o zi din săptămână și un interval orar de desfășurare, alături de materia și profesorul de predă acea materie, precum și anul, grupa și semianul pentru care este destinată acea înregistrare.

Cu ajutorul butonului de Add, putem adăuga o nouă înregistrare, într-un mod similar cu cel din Admin Page.

The screenshot shows the 'Add Timetable Entry Form' in the 'Timetable Application' window. The form contains the following fields:

- Weekday: Monday (dropdown)
- Time Slot: 8-10 (dropdown)
- Professor: Jane Smith (dropdown)
- Discipline: SD (dropdown)
- Study Year: 1 (dropdown)
- Semi Year: A (dropdown)
- Student Group: 101 (dropdown)
- Add Entry button
- Back button

Filtrele ajuta la selecția după diverse criterii a inregistrarilor in orar. De exemplu, filtrand după orele susținute miercurea, ne este returnat următorul view:

Timetable Application

Timetable View

ID	Weekday	Time Slot	Teacher	Discipline	Study Year	Semi Year	Student Group
10	Wednesday	18.00-20.00	John Doe	TW	2	A	101

Add

Generate HTML

Filter by Weekday: Wednesday

Filter by Time Slot: All

Filter by Teacher: All

Filter by Discipline: All

Filter by Year: All

Filter by Semi Year: All

Filter by Group: All

Back

Butonul de Generate HTML creaza o copie a orarului ce poate fi văzută din browser:

Weekday	Time Slot	Teacher	Discipline	Study Year	Semi Year	Student Group	Scheduler ID
Monday	12:00-14:00	Jane Smith	SD	1	A	101	1
Thursday	8:00-10:00	John Doe	Cloud	2	B	101	1
Tuesday	18:00-20:00	Jane Smith	ML	3	E	102	1
Friday	16:00-18:00	John Doe	BD	1	B	102	1
Friday	14:00-16:00	Jane Smith	AG	3	A	102	1
Friday	12:00-14:00	Jane Smith	AG	3	B	101	1
Wednesday	18:00-20:00	John Doe	TW	2	A	101	1

### I.2.3. Timetable Page

Tehnologiile folosite pentru dezvoltarea acestei aplicații au fost:

- Pycharm Community edition + Python 3.9
- Tkinter
- Psycpg2
- Unittests framework

## II. Testare

Pentru a testa funcționarea corectă a fiecărui modul al aplicației, s-au creat Unit Tests. Testele create pot fi împărțite în 3 categorii, anume teste pentru modulul de formulare, teste pentru repository și teste pentru diferitele view-uri din aplicație.

### II.1. Testele pe repository

Pentru testele de repository, s-au efectuat mock-uri pentru conexiunea la baza de date, dar și pentru diferite metode din repository ce se ocupă cu funcționalitate CRUD.

```
@patch('repositories.student_repo.connection')
@patch('repositories.student_repo.get_value_study_year')
@patch('repositories.student_repo.get_value_semi_year')
@patch('repositories.student_repo.get_value_student_group')
def test_given_student_repo_when_get_students_with_bad_foreign_keys_then_throw_index_error(self,
                                                    mock_student_group_method,
                                                    mock_semi_year_method,
                                                    mock_study_year_method,
                                                    mock_conn):

    mock_cursor = MagicMock()
    mock_cursor.fetchall.return_value = [(1, "John", "Smith", 1, 1, 1,)]
    mock_conn.return_value.closed = 1
    mock_conn.return_value.cursor.return_value = mock_cursor
    mock_study_year_method.side_effect = study_year_method_side_effect
    mock_student_group_method.side_effect = side_effect_error
    mock_semi_year_method.side_effect = get_semiyear_id_side_effect

    self.assertRaises(IndexError, get_students)

    mock_conn.assert_called_once()
    mock_cursor.close.assert_called_once()
    mock_semi_year_method.assert_any_call(1)
    mock_study_year_method.assert_any_call(1)
```

În exemplul de mai sus, s-a folosit adnotarea ”@patch” pentru mock-urile necesare acestui test, ce va trece doar dacă, în cazul în care metoda get\_students, folosită pentru a citi date din baza de date,



primește foreign key-uri greșite, caz în care ar trebui aruncată eroarea "IndexError". Atributele "side\_effect" sunt folosite pentru a modifica comportamentul modificărilor patch-uite astfel încât să reproducă rezultatul așteptat. De asemenea, se fac assert-uri pentru existența conexiunii cu baza de date, precum și apelul unic al metodelor din repository.

## II.2. Testele pe formulare

Pentru partea de formulare, s-a testat, în principal, funcționalitatea din UI de adăugare a unor entități în baza de date, precum și persistarea acestora în view după ce au fost adăugate.

```
def test_fetch_added_student(self, mock_conn_students):
    form = view.AddStudentForm(self.root)

    mock_master = Mock()
    form.master = mock_master

    expected_students = [
        (5, "Alice", "Smith", 1, 1, 1),
        (6, "Bob", "Jones", 1, 1, 1),
        (7, "Adrian", "Smau", 1, 1, 2),
        (8, "Rares", "Gramescu", 1, 1, 2)
    ]

    mock_cursor_students = MagicMock()
    mock_cursor_students.fetchall.return_value = expected_students
    mock_conn_students.return_value.cursor.return_value = mock_cursor_students
    mock_conn_students.return_value.closed = 1
    form.students_view = view.StudentsView(self.root)
    form.students_view.set_students(expected_students)

    form.go_back()

    current_frame = self.root.wininfo_children()[-1]

    self.assertIsInstance(current_frame, view.StudentsView)

    updated_students = form.students_view.students
    self.assertEqual(len(updated_students), len(expected_students))
    for index in range(len(updated_students)):
        self.assertEqual(updated_students[index].first_name, expected_students[index][1])
```

În exemplul de mai sus, se testează cazul în care după adăugarea din formular a unui nou student, fereastra curentă se întoarce la view-ul pentru studenți, unde se află și studentul nou adăugat. Se folosește iarăși "@patch" pentru mock-uirea conexiunii la baza de date, și se stabilește lista de studenți la care ne așteptăm în view. După apelul funcției "go\_back", ce trimite userul înapoi în view, se verifică dacă frame-ul este cel corect, precum și dacă lista din view este cea la care ne așteptăm.

## II.3. Testele pe view

Pentru view, s-a testat inițializarea corectă a elementelor de UI, precum tree-view-uri, butoane și label-uri, precum și afișarea corectă a datelor în view.

```
@patch('repositories.teacher_repo.connection')
def test_display(self, mock_conn):

    teachers = [
        (1, "Jane", "Smith"),
        (2, "John", "Doe"),
        (3, "Jane", "Smith"),
        (4, "John", "Doe")
    ]

    mock_cursor = MagicMock()
    mock_cursor.fetchall.return_value = teachers
    mock_conn.return_value.cursor.return_value = mock_cursor
    mock_conn.return_value.closed = 1

    self.view.display()

    tree_items = self.view.tree.get_children()
    for i, teacher in enumerate(teachers):
        item_values = self.view.tree.item(tree_items[i])['values']
        self.assertEqual(list(item_values), list(teacher))

    self.assertIsInstance(self.view.tree, ttk.Treeview)
    self.assertIsInstance(self.view.add_button, ttk.Button)
    self.assertIsInstance(self.view.delete_button, ttk.Button)
    self.assertIsInstance(self.view.back_button, ttk.Button)

    self.assertEqual(str(self.view.delete_button["state"]), "disabled")
```

În poza de mai sus, se testează metoda ce afișează toate elementele din TeacherView și aduce toate datele ce trebuie afișate inițial. Ca la testele prezentate anterior, se mock-uieste conexiunea la baza de date, apoi se verifică dacă datele populate în tree-view corespund cu cele la care ne așteptăm. În final, se verifică inițializarea corectă a restului elementelor de UI de pe pagină.

## III. Utilizarea assert-urilor

Assert-urile au fost introduse în codul sursă, cu scopul de a verifica condițiile, postcondițiile și invarianții operațiilor menționate în primul capitol. Astfel, conform implementării propuse de noi, am

convenit la 3 “tipuri” de assert-uri, fiecare tip, aruncând un mesaj de eroare când condiția din assert nu este respectată.

### III.1. Database connection asserts

Aceste assert-uri au fost introduse înainte și după interacțiunea cu baza de date, prin verificarea dacă conexiunea cu baza de date a fost inițiată, respectiv, terminată.

```
def get_disciplines():
    conn = connection()
    assert conn is not None, "Connection unstable"
    cur = conn.cursor()

    cur.execute("SELECT * FROM discipline")
    rows = cur.fetchall()

    cur.close()
    conn.close()
    assert conn.closed == 1, "Connection is not closed"
```

Astfel ne asigurăm că tranzațiile din baza de date se efectuează într-o manieră eficientă, semnalând o eventuală problemă despre conexiunea cu aceasta.

### III.2. Repositories functions asserts

De fiecare dată când interacționăm cu funcții implementate în clasele repository, verificăm ca datele de intrare pentru aceste funcții sunt valide, precum și că modificarea implementată are loc.

```
def handle(first_name_entry=None, last_name_entry=None, study_year_var=None, semi_year_var=None,
           student_group_var=None):
    first_name = first_name_entry.get()
    last_name = last_name_entry.get()
    study_year = study_year_var.get()
    semi_year = semi_year_var.get()
    student_group = student_group_var.get()

    assert first_name[0].isupper() and first_name.isalpha(), "First name should start with a capital letter " \
        "and contain only letters "
    assert last_name[0].isupper() and last_name.isalpha(), "Last name should start with a capital letter " \
        "and contain only letters "
    assert study_year.isnumeric(), "Study year should be a number"
    assert semi_year.isalpha(), "Semi year should be a letter"
    assert student_group.isnumeric(), "Student group should be a number"
    add_student(first_name=first_name, last_name=last_name, study_year=study_year, semi_year=semi_year,
                student_group=student_group)
    assert any(student.first_name == first_name for student in get_students())
```

În acest exemplu, la adăugarea unui student, se apelează funcția `add_student`, ce are ca precondiții legate de nume, prenume, an, etc. După adăugarea unui student, verificăm în baza noastră de date dacă există acest student.

### III.3. UI asserts

Folosind Tkinter, la fel ca în faza de unit testing, verificăm că instanțele noastre de entități ce interacționează cu user-ul (butoane, label-uri etc.), sunt entități din librăria respectivă.

```
assert isinstance(self.tree, ttk.Treeview), "self.tree must be a valid ttk.Treeview object."
assert isinstance(self.add_button, ttk.Button), "self.add_button must be a valid ttk.Button object."
assert isinstance(self.delete_button, ttk.Button), "self.delete_button must be a valid ttk.Button object."
assert isinstance(self.back_button, ttk.Button), "self.back_button must be a valid ttk.Button object."
assert callable(self.add_student), "self.add_student must be a callable function."
assert callable(self.delete_student), "self.delete_student must be a callable function."
assert callable(self.go_back), "self.go_back must be a callable function."
assert isinstance(self.study_year_filter,
                  ttk.Combobox), "self.study_year_filter must be a valid ttk.Combobox object."
assert isinstance(self.semi_year_filter,
                  ttk.Combobox), "self.semi_year_filter must be a valid ttk.Combobox object."
assert isinstance(self.group_filter, ttk.Combobox), "self.group_filter must be a valid ttk.Combobox object."
```

De asemenea, verificăm dacă funcțiile din backend pe care le folosim sunt importate corect și că pot fi apelate.

În ultimul rând, invarianții verificați de noi sunt tot la nivelul de UI. Astfel, la afișarea entităților din baza de date, verificăm dacă elementele de UI conțin aceste obiecte ce reprezintă aceste entități.

```
for i, student in enumerate(filtered_students):
    item_values = self.tree.item(children[i])["values"]
    assert item_values == (student.id, student.first_name, student.study_year,
                           student.semi_year, student.student_group), "Mismatch in treeview item values"
```

Aici, la aplicarea unui filtru pe studenți, prin acest assert, invariantul nostru constă în faptul că elementele din interfață conțin tupla de valori corespondente studenților filtrați.

## IV. Contribuții

Mosor Andrei

- implementare repositories
- implementare unit testing(predominant partea de ui + o parte de repositories)
- implementare assertions (partea de repositories)

Gramescu Rares

- implementare repositories si views
- implementare unit tests pe repository
- implementare assertions pe partea de views

Smau Adrian

- implementare repositories si views
- implementare unit testing pe repositories
- implementare assertions pe partea de repositories + bug fixing

Horceag Codrin

- implementare views
- implementare unit tests pe views
- implementare assertions pe partea de views