Enunțul problemei:

Scrieți o aplicație pentru gestiunea notelor și a problemelor de laborator pentru o disciplină.

Lista de funcționalități ale aplicației:

1. Adaugă, șterge, modifică – pentru lista de studenți, respectiv listă de probleme
    a. Adaugă student
    b. Șterge student
    c. Modifică nume student
    d. Modifică grupă student
    e. Adaugă problemă de laborator
    f. Șterge problemă de laborator
    g. Modifică descriere problemă de laborator
    h. Modifică deadline problemă de laborator
2. Căutare
    a. Student
    b. Problemă de laborator
3. Asignare laborator/Notare laborator
4. Creare statistici:
    a. lista de studenți și notele lor la o problema de laborator dat, ordonat: alfabetic după nume, după notă.
    b. Toți studenții cu media notelor de laborator mai mic decât 5. (nume student și notă)

Planul de iterații:

Iterația	Funcționalități planificate
I1 (laboratorul 7)	1.a., 1.b., 1.c., 1.d., 1.e., 1.f., 1.g., 1.h. 2.a., 2.b.
I2 (laboratorul 8)	3
I3 (laboratorul 9)	4.a., 4.b.

Scenarii de rulare:

- **Pentru funcționalitatea 1. a. Adaugă student**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	adauga_student	Introdu id student:	Trebuie introdus id (int > 0)
C	42	Introdu numele studentului:	Trebuie introdus numele
D	Ana Baciu	Grupa studentului:	Trebuie introdusa grupa (int > 0)
E	211	Student adaugat cu succes! Introdu o comanda:	Mesaj care indica faptul ca adaugarea s-a facut cu succes!
F	adauga_student	Introdu id student:	Trebuie introdus id (int > 0)
G	14	Introdu numele studentului:	Trebuie introdus numele
H	Matei Vasilescu	Grupa studentului:	Trebuie introdusa grupa (int > 0)
I	211	Student adaugat cu succes! Introdu o comanda:	Mesaj care indica faptul ca adaugarea s-a facut cu succes!
J	adauga_student	Introdu id student:	Trebuie introdus id (int > 0)
K	22	Introdu numele studentului:	Trebuie introdus numele
L	Maria Radu	Grupa studentului:	Trebuie introdusa grupa (int > 0)
M	212	Student adaugat cu succes! Introdu o comanda:	Mesaj care indica faptul ca adaugarea s-a facut cu succes!

- **Pentru funcționalitatea 1. b. (imposibil de realizat) Șterge student**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	sterge_student	Introdu id-ul studentului pe care doresti sa il stergi:	Trebuie introdus id (int > 0)
C	43	Se intampina o eroare la nivelul repozitoriului de studenti: Id student inexistent! Introdu o comanda:	Nu se poate efectua ștergerea!

- **Pentru funcționalitatea 1. b. Șterge student**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	sterge_student	Introdu id-ul studentului pe care doresti sa il stergi:	Trebuie introdus id (int > 0)
C	42	Studentul cu id-ul: 42 a fost sters cu succes! Introdu o comanda:	S-a efectuat ștergerea

- **Pentru funcționalitatea 1. c. Modifică nume student**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	modifica_nume	Introdu id-ul studentului al carui nume doresti sa il modifici:	Trebuie introdus id (int > 0)
C	22	Introdu noul nume:	Trebuie introdus numele
D	Maria Popescu	Numele studentului cu id-ul: 22 a fost modificat cu succes! Introdu o comanda:	S-a modificat numele

- **Pentru funcționalitatea 1. d. Modifică grupă student**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	modifica_grupa	Introdu id-ul studentului a carui grupa doresti sa o modifici:	Trebuie introdus id (int > 0)
C	14	Introdu noua grupa:	Trebuie introdusa grupa (int > 0)
D	216	Grupa studentul cu id-ul: 22 a fost modificata cu succes! Introdu o comanda:	S-a modificat grupa

- **Pentru funcționalitatea 1. e. Adaugă problemă de laborator**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	adauga_lab	Introdu id lab, sub forma numar laborator + <<bara jos>> + numar problema. Unde e numar se asteapta un int mai mare decat 0, iar intre cele 3 elemente nu se pune spatiu (ex: 12_4)):	Trebuie introdus id
C	1_2	Introdu descrierea:	Trebuie introdusa descrierea
D	Mare grija la subpunctul c!	Introdu ziua deadline-ului:	Int > 0 , int < 32
E	12	Introdu luna deadline-ului:	Int > 0 , int < 13
F	11	Introdu anul deadline-ului:	Int > 2020
G	2021	Laborator adaugat cu succes! Introdu o comanda:	S-a făcut adăugarea
B	adauga_lab	Introdu id lab, sub forma numar laborator + <<bara jos>> + numar problema. Unde e numar se asteapta un int mai mare decat 0, iar intre cele 3 elemente nu se pune spatiu (ex: 12_4)):	Trebuie introdus id
C	2_-2	Introdu descrierea:	Trebuie introdusa descrierea
D		Introdu ziua deadline-ului:	Int > 0 , int < 32
E	121	Introdu luna deadline-ului:	Int > 0 , int < 13
F	111	Introdu anul deadline-ului:	Int > 2020
G	202	Se intampina o eroare la validare: Id invalid! Descriere invalida! Deadline invalid! Introdu o comanda:	Nu s-a efectuat adăugarea, căci id-ul are in componenta un întreg negativ, descrierea este vidă, iar formatul deadline-ului este necorespunzător.
B	adauga_lab	Introdu id lab, sub forma numar laborator + <<bara jos>> + numar problema. Unde e numar se asteapta un int mai mare decat 0, iar intre cele 3 elemente nu se pune spatiu (ex: 12_4)):	Trebuie introdus id
C	1_3	Introdu descrierea:	Trebuie introdusa descrierea
D	Mare grija la subpunctul d!	Introdu ziua deadline-ului:	Int > 0 , int < 32
E	15	Introdu luna deadline-ului:	Int > 0 , int < 13
F	11	Introdu anul deadline-ului:	Int > 2020
G	2021	Laborator adaugat cu succes! Introdu o comanda:	S-a făcut adăugarea

- **Pentru funcționalitatea 1. f. Șterge problemă de laborator**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	sterge_lab	Introdu id-ul laboratorului pe care vrei sa il stergi:	Trebuie introdus id
C	1_2	Laboratorul cu id-ul: 1_2 a fost sters cu succes! Introdu o comanda:	S-a efectuat ștergerea

- **Pentru funcționalitatea 1. g. Modifică descriere problemă de laborator**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	modifica_descriere	Introdu id-ul laboratorului a carui descriere doresti sa o modifici:	Trebuie introdus id
C	1_3	Introdu noua descriere:	Trebuie introdusa noua descriere
D	Trebuie justificat pas cu pas felul în care se rezolvă problema.	Descrierea laboratorului cu id-ul: 1_3 a fost modificata cu succes! Introdu o comanda:	S-a modificat descrierea

- **Pentru funcționalitatea 1. h. Modifică deadline problemă de laborator**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	modifica_deadline	Introdu id-ul laboratorului al carui deadline doresti sa il modifici:	Trebuie introdus id
C	1_3	Introdu ziua noului deadline:	Int > 0 , int < 32
D	21	Introdu luna noului deadline:	Int > 0 , int < 13
E	12	Introdu anul noului deadline:	Int > 2020
F	2021	Deadline-ul laboratoului cu id-ul: 1_3 a fost modificat cu succes! Introdu o comanda:	S-a modificat deadline-ul

- **Pentru funcționalitatea 2. a. Căutare student**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	cauta_student	Introdu id-ul studentului cautat:	Trebuie introdus id
C	22	Studentul cautat: Id: 22 -> Nume: Maria Popescu, din grupa 212. Introdu o comanda:	S-a afișat studentul găsit.

- **Pentru funcționalitatea 2. b. Căutare problemă de laborator**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	cauta_lab	Introdu id-ul laboratorului cautat:	Trebuie introdus id
C	141_2	Se intampina o eroare la nivelul repozitoriului de studenti: Id lab inexistent! Introdu o comanda:	Nu a găsit
D	cauta_lab	Introdu id-ul laboratorului cautat:	Trebuie introdus id
E	1	Se intampina o eroare la validare: Id invalid! Introdu o comanda:	Nu e valid id-ul
F	cauta_lab	Introdu id-ul laboratorului cautat:	Trebuie introdus id
G	1_3	Laboratorul cautat: Numar laborator_numar problema: 1_3 Descrierea: Trebuie justificat pas cu pas felul în care se rezolvă problema. Deadline-ul: 21.12.2021. Introdu o comanda:	S-a afișat laboratorul găsit.

- **Pentru funcționalitatea 3. Asignare laborator – adauga nota**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Presupunem ca avem deja studenti si laboaratoare
B	asignare_lab	Introdu id nota:	Trebuie introdus id (int > 0)
C	42	Introdu id-ul studentului caruia vrei sa ii dai nota (nr intreg pozitiv):	Trebuie introdus numele
D	14	Introdu id lab, sub forma numar laborator + <<bara jos>> + numar problema. Unde e numar se asteapta un	Trebuie introdusa grupa (int > 0)

		int mai mare decat 0, iar intre cele 3 elemente nu se pune spatiu (ex: 12_4)):	
E	1_3	Laborator asignat cu succes!	Mesaj care indica faptul ca adaugarea s-a facut cu succes!
F	adauga_nota	Introdu id student:	Trebuie introdus id (int > 0)
G	42	Introdu nota:	Trebuie introdus numele
H	6	Nota cu succes	Trebuie introdusa grupa (int > 0)

- **Pentru funcționalitatea 4. a. lista de studenți și notele lor la o problema de laborator dat, ordonat: alfabetic după nume**
- Presupunem ca avem: 1tudor 5, 2ana 6, 3gabi 8 – pt lab 1_1

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	raport1a	Dati id-ul problemei pentru care doriti sa vedeti lista de studenti si notele lor, ordonati alfabetic dupa nume:	Trebuie introdus id (int > 0)
C	1_1	Nota cu id: 2 -> Student: ana -> problema: 1_1 -> nota: 6.0. Nota cu id: 3 -> Student: gabi -> problema: 1_1 -> nota: 8.0. Nota cu id: 1 -> Student: tudor -> problema: 1_1 -> nota: 5.0.	

- **Pentru funcționalitatea 4. a. lista de studenți și notele lor la o problema de laborator dat, ordonat: după nota**
- Presupunem ca avem: 1tudor 5, 2ana 6, 3gabi 8 – pt lab 1_1

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	raport1b	Dati id-ul problemei pentru care doriti sa vedeti lista de studenti si notele lor, ordonati descrescator dupa nota:	Trebuie introdus id (int > 0)
C	1_1	Nota cu id: 3 -> Student: gabi -> problema: 1_1 -> nota: 8.0. Nota cu id: 2 -> Student: ana -> problema: 1_1 -> nota: 6.0. Nota cu id: 1 -> Student: tudor -> problema: 1_1 -> nota: 5.0.	

- **Pentru funcționalitatea 4. b. Toți studenții cu media notelor de laborator mai mic decât 5**

	Utilizator	Program	Descriere
A		Introdu o comanda:	Este tipărit mesajul de introducere a primei comenzi
B	raport2	**Nu sunt studenti cu media notelor mai mica decat 5!**	Pentru presupunerea anterioara
C	Presupunem ca exista un nou lab, lab 1_2 si tudor primeste 3		
D	raport2	Studentul: tudor are media notelor de laborator: 4.0	

Cazuri de testare:

Date de intrare	**Date de ieșire (sau ce este în spate)**
1 a 37 1 2 3 4 5 6 7 8 9 1	lista_toti = [{“id_participant”: 37, “scor”: 46.0}]
1 a 122 1 2 3 6 5 6 7 8 9 4	lista_toti = [{“id_participant”: 37, “scor”: 46.0}, {“id_participant”: 122, “scor”: 51.0}]
1 a 37 10 2 3 6.4 5 6 7 8 9 4	Participant existent! Lista e nemodificată
1 a 22 F 1 2 3 4 5 23 6 7 8 -1 9 10	Nu ati introdus o valoare reala, cuprinsa intre 1 si 10! Nu ati introdus o valoare reala, cuprinsa intre 1 si 10! Nu ati introdus o valoare reala, cuprinsa intre 1 si 10! lista_toti = [{“id_participant”: 37, “scor”: 46.0}, {“id_participant”: 122, “scor”: 51.0}, {“id_participant”: 22, “scor”: 55.0}]

```
def test_creeaza_student_succes(self):
    id_student = 1023
    nume = "Mircea Farcas"
    grupa = 213
    student = Student(id_student,nume,grupa)
    self.assertEqual(student.get_id_student(),id_student)
    self.assertEqual(student.get_nume(),nume)
    self.assertGreater(0.0001, abs(student.get_grupa()-grupa))


def test_egalitate_studenti(self):
    id_student = 1023
    nume = "Mircea Farcas"
```

```
    grupa = 213
    student = Student(id_student,nume,grupa)
    alt_nume = "Andrei Georgescu"
    alta_grupa = 212
    alt_student_acelasi_id = Student(id_student,alt_nume,alta_grupa)
    self.assertEqual(alt_student_acelasi_id,student)

def test_printeaza_student(self):
    id_student = 1023
    nume = "Mircea Farcas"
    grupa = 213
    student = Student(id_student,nume,grupa)
    self.assertEqual(str(student),"Id: 1023 -> Nume: Mircea Farcas, din grupa 213.")

def test_valideaza_student_succes(self):
    student = Student(1023,"Mircea Farcas",213)
    valid_student = ValidatorStudent()
    self.assertIsNone(valid_student.valideaza(student))


def test_valideaza_student_id_invalid(self):
    nume = "Mircea Farcas"
    grupa = 213
    inv_id_student = -23
    valid_student = ValidatorStudent()
    student = Student(inv_id_student,nume,grupa)
    with self.assertRaises(ValidationError) as exceptia_mea:
        valid_student.valideaza(student)
    self.assertEqual("Id invalid!\n", str(exceptia_mea.exception))


def test_valideaza_student_nume_invalid(self):
    id_student = 1023
    inv_nume = ""
```

```
    grupa = 213
    valid_student = ValidatorStudent()
    student = Student(id_student,inv_nume,grupa)
    with self.assertRaises(ValidationError) as exceptia_mea:
        valid_student.valideaza(student)
    self.assertEqual("Nume invalid!\n", str(exceptia_mea.exception))


def test_valideaza_student_grupa_invalida(self):
    id_student = 1023
    nume = "Toni"
    inv_grupa = -213
    valid_student = ValidatorStudent()
    student = Student(id_student,nume,inv_grupa)
    with self.assertRaises(ValidationError) as exceptia_mea:
        valid_student.valideaza(student)
    self.assertEqual("Grupa invalida!\n", str(exceptia_mea.exception))

def test_valideaza_student_doua_invalide1(self):
    inv_id_student = -1023
    inv_nume = ""
    grupa = 213
    valid_student = ValidatorStudent()
    student = Student(inv_id_student,inv_nume,grupa)
    with self.assertRaises(ValidationError) as exceptia_mea:
        valid_student.valideaza(student)
    self.assertEqual("Id invalid!\nNume invalid!\n", str(exceptia_mea.exception))

def test_valideaza_student_doua_invalide2(self):
    inv_id_student = -1023
    nume = "Raluca"
    inv_grupa = -213
    valid_student = ValidatorStudent()
    student = Student(inv_id_student,nume,inv_grupa)
```

```
        with self.assertRaises(ValidationError) as exceptia_mea:
            valid_student.valideaza(student)
        self.assertEqual("Id invalid!\nGrupa invalida!\n", str(exceptia_mea.exception))

    def test_valideaza_student_doua_invalide3(self):
        id_student = 1023
        inv_nume = ""
        inv_grupa = -213
        valid_student = ValidatorStudent()
        student = Student(id_student,inv_nume,inv_grupa)
        with self.assertRaises(ValidationError) as exceptia_mea:
            valid_student.valideaza(student)
        self.assertEqual("Nume invalid!\nGrupa invalida!\n", str(exceptia_mea.exception))

    def test_valideaza_student_toate_invalide(self):
        inv_id_student = -1023
        inv_nume = ""
        inv_grupa = -213
        valid_student = ValidatorStudent()
        student = Student(inv_id_student,inv_nume,inv_grupa)
        with self.assertRaises(ValidationError) as exceptia_mea:
            valid_student.valideaza(student)
        self.assertEqual("Id invalid!\nNume invalid!\nGrupa invalida!\n", str(exceptia_mea.exception))



    def test_creeaza_repo_vid(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        self.assertEqual(len(repo),0)



    def test_adauga_student_repo_succes(self):
```

```
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(1,"Lucas",211)
        repo.adauga_student(student)
        self.assertEqual(len(repo),1)

    def test_cauta_student_id_inexistent(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(1,"Lucas",211)
        repo.adauga_student(student)
        with self.assertRaises(RepositoryError) as re:
            repo.cauta_student_dupa_id(2)
        self.assertEqual("Id student inexistent!\n", str(re.exception))

    def test_adauga_student_acelasi_id(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(1,"Lucas",211)
        repo.adauga_student(student)
        alt_student_acelasi_id = Student(1,"Vali",212)
        with self.assertRaises(RepositoryError) as re:
            repo.adauga_student(alt_student_acelasi_id)
        self.assertEqual("Id student existent!\n", str(re.exception))

    def test_adauga_student_srv_succes(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
```

```
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        id_student = 1023
        nume = "Mircea Farcas"
        grupa = 213
        all_el = srv.get_all_studenti()
        self.assertEqual(len(all_el),0)
        srv.adauga_student(id_student,nume,grupa)
        all_el = srv.get_all_studenti()
        self.assertEqual(len(all_el),1)
        self.assertEqual(all_el[0].get_id_student(),id_student)
        self.assertEqual(all_el[0].get_nume(),nume)
        self.assertGreater(0.0001, abs(all_el[0].get_grupa()-grupa))

    def test_adauga_student_srv_id_invalid(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        inv_id_student = -1023
        nume = "Mircea Farcas"
        grupa = 213
        with self.assertRaises(ValidationError) as ve:
            srv.adauga_student(inv_id_student,nume,grupa)
        self.assertEqual("Id invalid!\n", str(ve.exception))


    def test_adauga_student_srv_nume_invalid(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
```

```
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        id_student = 123
        inv_nume = ""
        grupa = 213
        with self.assertRaises(ValidationError) as ve:
            srv.adauga_student(id_student,inv_nume,grupa)
        self.assertEqual("Nume invalid!\n", str(ve.exception))


    def test_adauga_student_srv_grupa_invalida(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        id_student = 123
        nume = "Andra"
        inv_grupa = -213
        with self.assertRaises(ValidationError) as ve:
            srv.adauga_student(id_student,nume,inv_grupa)
        self.assertEqual("Grupa invalida!\n", str(ve.exception))
    #ar mai merge si 1-2,1-3,2-3
    def test_adauga_student_srv_toate_invalide(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        inv_id_student = -123
```

```
        inv_nume = ""
        inv_grupa = -213
        with self.assertRaises(ValidationError) as ve:
            srv.adauga_student(inv_id_student,inv_nume,inv_grupa)
        self.assertEqual("Id invalid!\nNume invalid!\nGrupa invalida!\n", str(ve.exception))

    def test_sterge_student_repo_succes(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(12,"Gabi",211)
        repo.adauga_student(student)
        self.assertEqual(len(repo),1)
        id_student = student.get_id_student()
        repo.sterge_student_dupa_id(id_student)
        self.assertEqual(len(repo),0)


    def test_sterge_student_repo_id_inexistent(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(12,"Gabi",211)
        repo.adauga_student(student)
        with self.assertRaises(RepositoryError) as re:
            repo.sterge_student_dupa_id(143)
        self.assertEqual("Id student inexistent!\n", str(re.exception))

    def test_sterge_student_srv_succes(self):
        repo = RepoStudent()
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
```

```
        id_student = 1023
        nume = "Mircea Farcas"
        grupa = 213
        all_el = srv.get_all_studenti()
        self.assertEqual(len(all_el),0)
        srv.adauga_student(id_student,nume,grupa)
        all_el = srv.get_all_studenti()
        self.assertEqual(len(all_el),1)
        self.assertEqual(all_el[0].get_id_student(),id_student)
        self.assertEqual(all_el[0].get_nume(),nume)
        self.assertGreater(0.0001, abs(all_el[0].get_grupa()-grupa))
        srv.sterge_student(id_student)
        all_el = srv.get_all_studenti()
        self.assertEqual(len(all_el),0)


    def test_sterge_student_srv_id_inexistent(self):
        repo = RepoStudent()
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        srv.adauga_student(1,"Bogdan",12)
        with self.assertRaises(RepositoryError) as re:
            srv.sterge_student(143)
        self.assertEqual("Id student inexistent!\n", str(re.exception))


    def test_modifica_nume_student_repo_succes(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(12,"Gabi",211)
        repo.adauga_student(student)
        nume_nou = "Denis"
        repo.modifica_nume_student(12,nume_nou)
```

```
        student = repo.cauta_student_dupa_id(12)
        self.assertTrue(student.get_nume()==nume_nou)


    def test_modifica_nume_student_repo_id_inexistent(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(12,"Gabi",211)
        repo.adauga_student(student)
        nume_nou = "Denis"
        with self.assertRaises(RepositoryError) as re:
            repo.modifica_nume_student(13,nume_nou)
        self.assertEqual("Id student inexistent!\n", str(re.exception))

    def test_modifica_nume_student_srv_succes(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        srv.adauga_student(4,"Mircea",212)
        all_el = srv.get_all_studenti()
        self.assertEqual(all_el[0].get_nume(),"Mircea")
        srv.modifica_nume_student(4,"Florin")
        all_el = srv.get_all_studenti()
        self.assertTrue(all_el[0].get_nume()=="Florin")


    def test_modifica_nume_student_srv_nume_invalid(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
```

```
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(4,"Mircea",212)
    with self.assertRaises(ValidationError) as ve:
        srv.modifica_nume_student(4,"")
    self.assertEqual("Nume invalid!\n", str(ve.exception))


def test_modifica_nume_student_srv_id_inexistent(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(4,"Mircea",212)
    with self.assertRaises(RepositoryError) as re:
        srv.modifica_nume_student(1001,"Lup")
    self.assertEqual("Id student inexistent!\n", str(re.exception))

def test_modifica_grupa_student_repo_succes(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    student = Student(12,"Gabi",211)
    repo.adauga_student(student)
    repo.modifica_grupa_student(12,215)
    student = repo.cauta_student_dupa_id(12)
    self.assertEqual(student.get_grupa(),215)

def test_modifica_grupa_student_srv_nume_invalid(self):
```

```
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        srv.adauga_student(4,"Mircea",212)
        with self.assertRaises(ValidationError) as ve:
            srv.modifica_grupa_student(4, -2)
        self.assertEqual("Grupa invalida!\n", str(ve.exception))

    def test_modifica_grupa_student_repo_id_inexistent(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(12,"Gabi",211)
        repo.adauga_student(student)
        with self.assertRaises(RepositoryError) as re:
            repo.modifica_grupa_student(300,215)
        self.assertEqual("Id student inexistent!\n", str(re.exception))

    def test_modifica_grupa_student_srv_succes(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        srv.adauga_student(1023,"Mircea Farcas",213)
        srv.modifica_grupa_student(1023,200)
        all_el = srv.get_all_studenti()
        self.assertEqual(all_el[0].get_grupa(),200)
```

```
def test_modifica_grupa_student_srv_id_inexistent(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(1,"Dan",214)
    with self.assertRaises(RepositoryError) as re:
        srv.modifica_grupa_student(2,212)
    self.assertEqual("Id student inexistent!\n", str(re.exception))

def test_cauta_student_repo_succes(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    student = Student(12,"Gabi",211)
    repo.adauga_student(student)
    st = repo.cauta_student_dupa_id(12)
    self.assertTrue(st.get_id_student()==student.get_id_student())
    self.assertTrue(st.get_nume()==student.get_nume())
    self.assertTrue(abs(st.get_grupa()-student.get_grupa())<0.0001)

def test_cauta_student_repo_id_inexistent(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    student = Student(12,"Gabi",211)
    repo.adauga_student(student)
    with self.assertRaises(RepositoryError) as re:
        repo.cauta_student_dupa_id(10000)
    self.assertEqual("Id student inexistent!\n", str(re.exception))
```

```
def test_cauta_student_srv_succes(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(1,"Tat",210)
    gasit = srv.cauta_student(1)
    self.assertTrue(gasit.get_id_student()==1)
    self.assertEqual(gasit.get_nume(),"Tat")
    self.assertGreater(0.0001,abs(gasit.get_grupa()-210))

def test_cauta_student_srv_id_cautat_invalid(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(1,"Tat",210)
    with self.assertRaises(ValidationError) as ve:
        srv.cauta_student(-32)
    self.assertEqual("Id invalid!\n", str(ve.exception))

def test_cauta_student_srv_id_inexistent(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(1,"Tat",210)
```

```
        with self.assertRaises(RepositoryError) as re:
            srv.cauta_student(32)
        self.assertEqual("Id student inexistent!\n", str(re.exception))


    def test_creeaza_lab_succes(self):
        lab = Laborator("7_2","Lab 7 pb 2 - cu studenti si laboratoare.",[12,11,2021])
        self.assertEqual (lab.get_nrlab_nrpb(),"7_2")
        self.assertEqual(lab.get_descriere(),"Lab 7 pb 2 - cu studenti si laboratoare.")
        self.assertEqual(lab.get_deadline(),[12,11,2021])


    def test_egalitate_lab(self):
        lab = Laborator("7_2","Lab 7 pb 2 - cu studenti si laboratoare.",[12,11,2021])
        alt_lab_acelasi_id = Laborator("7_2","Lab 7 pb 2 - cu filme si actori.",[12,12,2021])
        self.assertTrue(alt_lab_acelasi_id == lab)


    def test_printeaza_lab(self):
        lab = Laborator("7_2","Lab 7 pb 2 - cu studenti si laboratoare.",[12,11,2021])
        self.assertTrue(str(lab)=="Numar laborator_numar problema: 7_2 Descrierea: Lab 7 pb 2 - cu studenti
si laboratoare. Deadline-ul: 12.11.2021.")


    def test_valideaza_lab_succes(self):
        lab = Laborator("7_2","Lab 7 pb 2 - cu studenti si laboratoare.",[12,11,2021])
        valid_lab = ValidatorLaborator()
        valid_lab.valideaza(lab)
        self.assertTrue(True)


    def test_valideaza_lab_id_invalid(self):
        valid_lab = ValidatorLaborator()
        lab = Laborator("","Lab 7 pb 2 - cu studenti si laboratoare.",[12,11,2021])
        with self.assertRaises(ValidationError) as ve:
```

```
        valid_lab.valideaza(lab)
    self.assertEqual("Id invalid!\n", str(ve.exception))


def test_valideaza_lab_descriere_invalid(self):
    valid_lab = ValidatorLaborator()
    lab = Laborator("1_1","",[12,11,2021])
    with self.assertRaises(ValidationError) as ve:
        valid_lab.valideaza(lab)
    self.assertEqual("Descriere invalida!\n", str(ve.exception))


def test_valideaza_lab_deadline_invalida(self):
    valid_lab = ValidatorLaborator()
    lab = Laborator("1_1","da",[121,11,2021])
    with self.assertRaises(ValidationError) as ve:
        valid_lab.valideaza(lab)
    self.assertEqual("Deadline invalid!\n", str(ve.exception))


def test_valideaza_lab_toate_invalide(self):
    valid_lab = ValidatorLaborator()
    lab = Laborator("","",[11,13,2021])
    with self.assertRaises(ValidationError) as ve:
        valid_lab.valideaza(lab)
    self.assertEqual("Id invalid!\nDescriere invalida!\nDeadline invalid!\n", str(ve.exception))


def test_creeaza_repo_lab_vid(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    self.assertTrue(len(repo)==0)
```

```
def test_adauga_lab_repo_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("1_1","da",[12,11,2021])
    repo.adauga_lab(lab)
    self.assertEqual(len(repo),1)


def test_cauta_lab_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("1_1","da",[12,11,2021])
    repo.adauga_lab(lab)
    with self.assertRaises(RepositoryError) as re:
        repo.cauta_lab_dupa_id("1_2")
    self.assertEqual("Id lab inexistent!\n", str(re.exception))


def test_adauga_lab_acelasi_id(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("1_1","da",[12,11,2021])
    alt_lab_acelasi_id = Laborator("1_1","nu",[12,11,2021])
    repo.adauga_lab(lab)
    with self.assertRaises(RepositoryError) as re:
        repo.adauga_lab(alt_lab_acelasi_id)
    self.assertEqual("Id lab existent!\n", str(re.exception))
```

```
def test_adauga_lab_srv_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("1_1","da",[12,11,2021])
    all_el = srv.get_all_lab()
    self.assertEqual(len(all_el),1)
    self.assertEqual(all_el[0].get_nrlab_nrpb(),"1_1")
    self.assertEqual(all_el[0].get_descriere(),"da")
    self.assertEqual(all_el[0].get_deadline(),[12,11,2021])


def test_adauga_lab_srv_id_invalid(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    with self.assertRaises(ValidationError) as ve:
        srv.adauga_lab("","da",[12,11,2021])
    self.assertEqual("Id invalid!\n", str(ve.exception))


def test_adauga_lab_srv_descriere_invalid(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
```

```
    srv = ServiceLaborator(valid_lab,repo)
    with self.assertRaises(ValidationError) as ve:
        srv.adauga_lab("1_1","",[12,11,2021])
    self.assertEqual("Descriere invalida!\n", str(ve.exception))


def test_adauga_lab_srv_deadline_invalida(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    with self.assertRaises(ValidationError) as ve:
        srv.adauga_lab("1_1","ceva",[12,14,2021])
    self.assertEqual("Deadline invalid!\n", str(ve.exception))

def test_adauga_lab_srv_toate_invalide(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    with self.assertRaises(ValidationError) as ve:
        srv.adauga_lab("","",[12,14,2021])
    self.assertEqual("Id invalid!\nDescriere invalida!\nDeadline invalid!\n", str(ve.exception))


def test_sterge_lab_repo_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
```

```
    lab = Laborator("7_2","Gabi",[12,12,2021])
    repo.adauga_lab(lab)
    self.assertTrue(len(repo)==1)
    repo.sterge_lab_dupa_id("7_2")
    self.assertEqual(len(repo),0)


def test_sterge_lab_repo_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("7_2","Gabi",[12,12,2021])
    repo.adauga_lab(lab)
    with self.assertRaises(RepositoryError) as re:
        repo.sterge_lab_dupa_id("7_7")
    self.assertEqual("Id lab inexistent!\n", str(re.exception))

def test_sterge_lab_srv_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    all_el = srv.get_all_lab()
    self.assertTrue(all_el[0].get_nrlab_nrpb()=="7_2")
    self.assertTrue(all_el[0].get_descriere()=="descriere")
    self.assertTrue(all_el[0].get_deadline()==[12,12,2021])
    srv.sterge_lab("7_2")
    all_el = srv.get_all_lab()
    self.assertTrue(len(all_el)==0)
```

```
def test_sterge_lab_srv_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    with self.assertRaises(RepositoryError) as re:
        srv.sterge_lab("1_3")
    self.assertEqual("Id lab inexistent!\n", str(re.exception))

def test_modifica_descriere_lab_repo_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("7_2","descriere",[13,12,2022])
    repo.adauga_lab(lab)
    repo.modifica_descriere_lab("7_2","descriere_noua")
    lab = repo.cauta_lab_dupa_id("7_2")
    assert(lab.get_descriere()=="descriere_noua")


def test_modifica_descriere_lab_repo_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("7_2","descriere",[13,12,2022])
    repo.adauga_lab(lab)
    with self.assertRaises(RepositoryError) as re:
        repo.modifica_descriere_lab("1_2","descriere_noua")
    self.assertEqual("Id lab inexistent!\n", str(re.exception))
```

```
def test_modifica_descriere_lab_srv_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    all_el = srv.get_all_lab()
    self.assertTrue(all_el[0].get_descriere()=="descriere")
    srv.modifica_descriere_lab("7_2","descriere_noua")
    all_el = srv.get_all_lab()
    self.assertTrue(all_el[0].get_descriere()=="descriere_noua")


def test_modifica_descriere_lab_srv_descriere_invalid(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    with self.assertRaises(ValidationError) as ve:
        srv.modifica_descriere_lab("7_2","")
    self.assertEqual("Descriere invalida!\n", str(ve.exception))


def test_modifica_descriere_lab_srv_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
```

```
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    with self.assertRaises(RepositoryError) as re:
        srv.modifica_descriere_lab("7_3","descriere_noua")
    self.assertEqual("Id lab inexistent!\n", str(re.exception))

def test_modifica_deadline_lab_repo_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("7_2","descriere",[13,12,2022])
    repo.adauga_lab(lab)
    repo.modifica_deadline_lab("7_2",[19,12,2021])
    lab = repo.cauta_lab_dupa_id("7_2")
    self.assertEqual(lab.get_deadline(),[19,12,2021])


def test_modifica_deadline_lab_repo_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("7_2","descriere",[13,12,2022])
    repo.adauga_lab(lab)
    with self.assertRaises(RepositoryError) as re:
        repo.modifica_deadline_lab("8_2",[19,12,2021])
    self.assertEqual("Id lab inexistent!\n", str(re.exception))

def test_modifica_deadline_lab_srv_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
```

```
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    srv.modifica_deadline_lab("7_2",[19,12,2021])
    all_el = srv.get_all_lab()
    self.assertTrue(all_el[0].get_deadline()==[19,12,2021])


def test_modifica_deadline_lab_srv_descriere_invalid(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    with self.assertRaises(ValidationError) as ve:
        srv.modifica_deadline_lab("7_2",[12,10])
    self.assertEqual("Deadline invalid!\n", str(ve.exception))


def test_modifica_deadline_lab_srv_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    with self.assertRaises(RepositoryError) as re:
        srv.modifica_deadline_lab("1_4",[19,12,2021])
    self.assertEqual("Id lab inexistent!\n", str(re.exception))
```

```
def test_cauta_lab_repo_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("1_2","Atentie mare la detalii.",[15,11,2021])
    repo.adauga_lab(lab)
    lb = repo.cauta_lab_dupa_id("1_2")
    self.assertTrue(lb.get_nrlab_nrpb()==lab.get_nrlab_nrpb())
    self.assertTrue(lb.get_descriere()==lab.get_descriere())
    self.assertTrue(lb.get_deadline()==lab.get_deadline())

def test_cauta_lab_repo_id_inexistent(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    lab = Laborator("1_2","Atentie mare la detalii.",[15,11,2021])
    repo.adauga_lab(lab)
    with self.assertRaises(RepositoryError) as re:
        repo.cauta_lab_dupa_id("1_4")
    self.assertEqual("Id lab inexistent!\n", str(re.exception))

def test_cauta_lab_srv_succes(self):
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoLaborator(file_path)
    valid_lab = ValidatorLaborator()
    srv = ServiceLaborator(valid_lab,repo)
    srv.adauga_lab("7_2","descriere",[12,12,2021])
    gasit = srv.cauta_lab("7_2")
    self.assertTrue(gasit.get_nrlab_nrpb()=="7_2")
    self.assertTrue(gasit.get_descriere()=="descriere")
```

```
        self.assertTrue(gasit.get_deadline()==[12,12,2021])


    def test_cauta_lab_srv_id_cautat_invalid(self):
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoLaborator(file_path)
        valid_lab = ValidatorLaborator()
        srv = ServiceLaborator(valid_lab,repo)
        srv.adauga_lab("7_2","descriere",[12,12,2021])
        with self.assertRaises(ValidationError) as ve:
            srv.cauta_lab("17")
        self.assertEqual("Id invalid!\n", str(ve.exception))


    def test_cauta_lab_srv_id_inexistent(self):
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoLaborator(file_path)
        valid_lab = ValidatorLaborator()
        srv = ServiceLaborator(valid_lab,repo)
        srv.adauga_lab("7_2","descriere",[12,12,2021])
        with self.assertRaises(RepositoryError) as re:
            srv.cauta_lab("1_88")
        self.assertEqual("Id lab inexistent!\n", str(re.exception))

    def test_filtreaza_dupa_prefix_repo_succes(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(1,"George Pop",213)
```

```
        repo.adauga_student(student)
        prefix = "Ge"
        studenti = repo.filtreaza_dupa_prefix_nume(prefix)
        self.assertTrue(studenti[0].get_id_student()==student.get_id_student())
        self.assertTrue(studenti[0].get_nume()==student.get_nume())
        self.assertGreater(0.0001,abs(studenti[0].get_grupa()-student.get_grupa()))


    def test_filtreaza_dupa_prefix_repo_inexistent(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        student = Student(1,"George Pop",213)
        repo.adauga_student(student)
        prefix_inex = "Ra"
        with self.assertRaises(RepositoryError) as re:
            repo.filtreaza_dupa_prefix_nume(prefix_inex)
        self.assertEqual("Nu exista studenti care sa aiba prefix sirul citit!\n", str(re.exception))

    def test_filtreaza_dupa_prefix_srv_succes(self):
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoStudent(file_path)
        valid_student = ValidatorStudent()
        srv = ServiceStudent(valid_student,repo)
        srv.adauga_student(1,"Tomas",12)
        prefix = "Tom"
        studenti = srv.filtreaza_nume(prefix)
        self.assertTrue(studenti[0].get_id_student()==1)
        self.assertTrue(studenti[0].get_nume()=="Tomas")
        self.assertTrue(abs(studenti[0].get_grupa()-12)<0.0001)
```

```
def test_filtreaza_dupa_prefix_srv_prefix_invalid(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(1,"Tomas",12)
    prefix_inv = ""
    with self.assertRaises(ValidationError) as ve:
        srv.filtreaza_nume(prefix_inv)
    self.assertEqual("Prefix invalid!\n", str(ve.exception))


def test_filtreaza_dupa_prefix_srv_prefix_negasit(self):
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoStudent(file_path)
    valid_student = ValidatorStudent()
    srv = ServiceStudent(valid_student,repo)
    srv.adauga_student(1,"Tomas",12)
    prefix_inex = "Tor"
    with self.assertRaises(RepositoryError) as re:
        srv.filtreaza_nume(prefix_inex)
    self.assertEqual("Nu exista studenti care sa aiba prefix sirul citit!\n", str(re.exception))



def test_creeaza_nota_succes(self):
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = Note(1,student,lab)
    self.assertEqual(nota.get_id_nota(),1)
```

```
        self.assertEqual(nota.get_student(),student)
        self.assertEqual(nota.get_lab(),lab)

    def test_egalitate_nota(self):
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = Note(1,student,lab)
        alt_student = Student(2,"Gabriel",213)
        alta_nota_acelasi_id = Note(1,alt_student,lab)
        self.assertEqual(alta_nota_acelasi_id, nota)

    def test_valideaza_nota_asignata_succes(self):
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = Note(2,student,lab)
        valid_note = ValidatorNote()
        valid_note.valideaza_id(nota)
        self.assertTrue(True)


    def test_valideaza_nota_asignata_id_invalid(self):
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = Note(-2,student,lab)
        valid_nota = ValidatorNote()
        with self.assertRaises(ValidationError) as ve:
            valid_nota.valideaza_id(nota)
        self.assertEqual("Id invalid!\n", str(ve.exception))


    def test_valideaza_nota_adaugata_succes(self):
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = Note(2,student,lab)
```

```
    valid_note = ValidatorNote()
    valid_note.valideaza(nota,7.6)
    self.assertTrue(True)

def test_valideaza_nota_adaugata_invalida(self):
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = Note(2,student,lab)
    valid_nota = ValidatorNote()
    with self.assertRaises(ValidationError) as ve:
        valid_nota.valideaza(nota,0.3)
    self.assertEqual("Nota invalida!\n", str(ve.exception))


def test_creeaza_repo_nota_vid(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    self.assertTrue(len(repo)==0)


def test_asignare_nota_repo_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(1,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    self.assertTrue(len(repo)==1)
```

```
def test_cauta_nota_id_inexistent(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(1,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    with self.assertRaises(RepositoryError) as re:
        repo.cauta_nota_dupa_id(14)
    self.assertEqual("Id nota laborator inexistent!\n", str(re.exception))


def test_asignare_nota_acelasi_id(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(1,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    alt_student = Student(2,"Tudor",214)
    alta_nota_acelasi_id = NoteDTO(1,alt_student.get_id_student(),lab.get_nrlab_nrpb(),0)
    with self.assertRaises(RepositoryError) as re:
        repo.asignare_nota(alta_nota_acelasi_id)
    self.assertEqual("Problema de laborator deja asignata!\n", str(re.exception))


def test_asignare_nota_problema_deja_asignata(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
```

```
    repo = FileRepoNote(file_path)
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(1,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    problema_deja_asignata = NoteDTO(14,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    with self.assertRaises(RepositoryError) as re:
        repo.asignare_nota(problema_deja_asignata)
    self.assertEqual("Problema de laborator deja asignata!\n", str(re.exception))


def test_asignare_nota_srv_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==0)
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
    srv.asignare_nota(1, 1, "1_1")
```

```
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==1)
    self.assertTrue(all_el[0].get_id_nota()==1)
    self.assertTrue(all_el[0].get_id_student()==1)
    self.assertTrue(all_el[0].get_nrlab_nrpb()=="1_1")
    self.assertTrue(all_el[0].get_nota()==0)


def test_asignare_nota_srv_id_invalid(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==0)
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
    with self.assertRaises(ValidationError) as ve:
        srv.asignare_nota(-1,1,"1_1")
    self.assertEqual("Id invalid!\n", str(ve.exception))
```

```
def test_adaugare_nota_repo_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(1,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo_note.asignare_nota(nota)
    repo_note.adauga_nota(nota,9.5)
    self.assertTrue(len(repo_note)==1)
    self.assertTrue(repo_note.cauta_nota_dupa_id(nota.get_id_nota()).get_nota()==9.5)
    '''Facem si o cautare cu id inexistent, sa verificam'''
    with self.assertRaises(RepositoryError) as re:
        repo_note.cauta_nota_dupa_id(19)
    self.assertEqual("Id nota laborator inexistent!\n", str(re.exception))

def test_adaugare_nota_srv_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
```

```
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==0)
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
    srv.asignare_nota(3,1,"1_1")
    srv_lab.adauga_lab("1_5","ceva",[12,12,2021])
    srv.adauga_nota(3,10)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==1)
    self.assertTrue(all_el[0].get_id_nota()==3)
    self.assertTrue(all_el[0].get_id_student()==1)
    self.assertTrue(all_el[0].get_nrlab_nrpb()=="1_1")
    self.assertTrue(all_el[0].get_nota()==10)


def test_adaugare_nota_srv_id_inexistent(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
```

```
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
    srv.asignare_nota(3,1,"1_1")
    srv_lab.adauga_lab("1_5","ceva",[12,12,2021])
    srv.asignare_nota(13,1,"1_5")
    with self.assertRaises(RepositoryError) as re:
        srv.adauga_nota(1777,3.8)
    self.assertEqual("Id nota laborator inexistent!\n", str(re.exception))


def test_sterge_nota_repo_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    id_nota = 12
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    self.assertTrue(len(repo)==1)
    id_nota = nota.get_id_nota()
    repo.sterge_nota_dupa_id_nota(id_nota)
    self.assertTrue(len(repo)==0)


def test_sterge_nota_repo_id_inexistent(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
```

```
    id_nota = 12
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    with self.assertRaises(RepositoryError) as re:
        repo.sterge_nota_dupa_id_nota(72)
    self.assertEqual("Id nota laborator inexistent!\n", str(re.exception))


def test_sterge_nota_repo_dupa_id_student_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    id_nota = 12
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    repo.sterge_nota_dupa_id_student(student.get_id_student())
    self.assertTrue(len(repo)==0)


def test_sterge_nota_repo_dupa_id_student_inexistent(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    id_nota = 12
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
```

```
        with self.assertRaises(RepositoryError) as re:
            repo.sterge_nota_dupa_id_student(72)
        self.assertEqual("Id student inexistent!\n", str(re.exception))


    def test_sterge_nota_repo_dupa_id_lab_succes(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoNote(file_path)
        id_nota = 12
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
        repo.asignare_nota(nota)
        repo.sterge_nota_dupa_id_lab(lab.get_nrlab_nrpb())
        self.assertTrue(len(repo)==0)


    def test_sterge_nota_repo_dupa_id_lab_inexistent(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoNote(file_path)
        id_nota = 12
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
        repo.asignare_nota(nota)
        with self.assertRaises(RepositoryError) as re:
            repo.sterge_nota_dupa_id_lab("1_333")
        self.assertEqual("Id lab inexistent!\n", str(re.exception))

    def test_sterge_nota_srv_succes(self):
```

```
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
    srv.asignare_nota(1414,1,"1_1")
    srv.adauga_nota(1414,8)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==1)
    srv.sterge_nota(1414)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==0)


def test_sterge_nota_srv_id_inexistent(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
```

```
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
    srv.asignare_nota(1414,1,"1_1")
    srv.adauga_nota(1414,8)
    with self.assertRaises(RepositoryError) as re:
        srv.sterge_nota(16)
    self.assertEqual("Id nota laborator inexistent!\n", str(re.exception))


def test_sterge_nota_srv_id_invalid(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
```

```
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
        srv_lab = ServiceLaborator(valid_lab,repo_lab)
        srv_student.adauga_student(1,"Tudor",213)
        srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
        srv.asignare_nota(1414,1,"1_1")
        srv.adauga_nota(1414,8)
        with self.assertRaises(ValidationError) as ve:
            srv.sterge_nota(-3)
        self.assertEqual("Id nota laborator invalid!\n", str(ve.exception))

    def test_sterge_nota_dupa_id_student_srv_succes(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_note = FileRepoNote(file_path)
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_studenti = FileRepoStudent(file_path)
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
```

```
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==0)
    srv_student.adauga_student(101,"Tudor",213)
    srv_lab.adauga_lab("1_4","ceva",[12,12,2021])
    srv.asignare_nota(78,101,"1_4")
    srv.adauga_nota(78,5.9)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==1)
    srv.sterge_student_nota(101)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==0)


def test_sterge_nota_dupa_id_student_inexistent_srv_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    srv_student.adauga_student(101,"Tudor",213)
```

```
    srv_lab.adauga_lab("1_4","ceva",[12,12,2021])
    srv.asignare_nota(1,101,"1_4")
    srv.adauga_nota(1,7)
    with self.assertRaises(RepositoryError) as re:
        srv.sterge_student_nota(1923)
    self.assertEqual("Id student inexistent!\n", str(re.exception))


def test_sterge_nota_dupa_id_lab_srv_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    all_el = srv.get_all_note_sir()
    self.assertTrue(len(all_el)==0)
    srv_student.adauga_student(101,"Tudor",213)
    srv_lab.adauga_lab("1_4","ceva",[12,12,2021])
    srv.asignare_nota(78,101,"1_4")
    srv.adauga_nota(78,5.9)
    all_el = srv.get_all_note_sir()
```

```
        self.assertTrue(len(all_el)==1)
        srv.sterge_lab_nota("1_4")
        all_el = srv.get_all_note_sir()
        self.assertTrue(len(all_el)==0)

    def test_sterge_nota_dupa_id_lab_inexistent_srv_succes(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_note = FileRepoNote(file_path)
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_studenti = FileRepoStudent(file_path)
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
        srv_lab = ServiceLaborator(valid_lab,repo_lab)
        srv_student.adauga_student(101,"Tudor",213)
        srv_lab.adauga_lab("1_4","ceva",[12,12,2021])
        srv.asignare_nota(1,101,"1_4")
        srv.adauga_nota(1,7)
        with self.assertRaises(RepositoryError) as re:
            srv.sterge_lab_nota("1_111")
        self.assertEqual("Id lab inexistent!\n", str(re.exception))


    def test_get_all_note_dupa_id_problema_repo_succes(self):
```

```
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoNote(file_path)
        id_nota = 12
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
        repo.asignare_nota(nota)
        self.assertTrue(len(repo)==1)
        repo.adauga_nota(nota,9.25)
        nrlab_nrpb = lab.get_nrlab_nrpb()
        note_anumita_problema = repo.get_all_note_dupa_id_problema(nrlab_nrpb)
        self.assertTrue(len(note_anumita_problema)==1)
        id_nota2 = 888
        student2 = Student(2,"Titi",2143)
        nota2 = NoteDTO(id_nota2,student2.get_id_student(),lab.get_nrlab_nrpb(),0)
        repo.asignare_nota(nota2)
        self.assertTrue(len(repo)==2)
        with self.assertRaises(RepositoryError) as re:
            repo.get_all_note_dupa_id_problema(nrlab_nrpb)
        self.assertEqual("Exista laborator/oare asignat/e si nenotat/e!\n", str(re.exception))
        repo.adauga_nota(nota2,5.8)
        note_anumita_problema2 = repo.get_all_note_dupa_id_problema(nrlab_nrpb)
        self.assertTrue(len(note_anumita_problema2)==2)
        id_lab_inexistent = nrlab_nrpb + "sarepelimba"
        with self.assertRaises(RepositoryError) as re:
            repo.get_all_note_dupa_id_problema(id_lab_inexistent)
        self.assertEqual("Aceasta problema nu a fost asignata niciunui student SAU a fost asignata si nu
notata!\n", str(re.exception))



    def test_get_all_note_dupa_id_problema_repo_lab_nenotat(self):
        file_path = "test_note.txt"
```

```
        with open(file_path,"w") as f:
            f.write("")
        repo = FileRepoNote(file_path)
        id_nota = 12
        student = Student(1,"Gabi",213)
        lab = Laborator("1_1","usor",[13,12,2021])
        nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
        repo.asignare_nota(nota)
        repo.adauga_nota(nota,9.25)
        id_nota2 = 888
        student2 = Student(2,"Titi",2143)
        nota2 = NoteDTO(id_nota2,student2.get_id_student(),lab.get_nrlab_nrpb(),0)
        repo.asignare_nota(nota2)
        repo.adauga_nota(nota2,7.65)
        id_nota3 = 342
        student3 = Student(3,"Firuta",23)
        lab3 = Laborator("1_33","minune",[8,8,2022])
        nota3 = NoteDTO(id_nota3,student3.get_id_student(),lab3.get_nrlab_nrpb(),0)
        nrlab_nrpb3 = lab3.get_nrlab_nrpb()
        repo.asignare_nota(nota3)
        self.assertTrue(len(repo)==3)
        with self.assertRaises(RepositoryError) as re:
            repo.get_all_note_dupa_id_problema(nrlab_nrpb3)
        self.assertEqual("Aceasta problema nu a fost asignata niciunui student SAU a fost asignata si nu notata!\n", str(re.exception))


    def test_raport_note_studenti_lab_dat_ordonati_alfabetic_srv_succes(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_note = FileRepoNote(file_path)
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
```

```
        repo_studenti = FileRepoStudent(file_path)
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
        srv_lab = ServiceLaborator(valid_lab,repo_lab)
        id_nota = 1
        nota_pb = 9.5
        srv_student.adauga_student(1,"Tudor",213)
        srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
        srv.asignare_nota(id_nota,1,"1_1")
        srv.adauga_nota(id_nota,nota_pb)
        srv_student.adauga_student(2,"Ana",211)
        srv.asignare_nota(id_nota+1,2,"1_1")
        srv.adauga_nota(id_nota+1,nota_pb+0.3)
        all_el = srv.get_all_note_sir()
        self.assertTrue(len(all_el)==2)
        alfabetic = srv.raport_studenti_cu_note_alfabetic("1_1")
        #self.assertTrue(str(alfabetic[0]) == "Nota cu id: 2 -> Student:  Ana -> problema: 1_1 -> nota: 9.8.")
        #self.assertTrue(str(alfabetic[1]) == "Nota cu id: 1 -> Student:  Tudor -> problema: 1_1 -> nota:
9.5.")
        self.assertTrue(str(alfabetic[0]) == "Ana are nota: 9.8.")
        self.assertTrue(str(alfabetic[1]) == "Tudor are nota: 9.5.")



    def test_raport_note_studenti_lab_dat_ordonati_alfabetic_srv_id_lab_invalid(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_note = FileRepoNote(file_path)
```

```
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    id_nota = 1
    nota_pb = 9.5
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
    srv.asignare_nota(id_nota,1,"1_1")
    srv.adauga_nota(id_nota,nota_pb)
    srv_student.adauga_student(2,"Ana",211)
    srv.asignare_nota(id_nota+1,2,"1_1")
    srv.adauga_nota(id_nota+1,nota_pb+0.3)
    with self.assertRaises(ValidationError) as ve:
        srv.raport_studenti_cu_note_alfabetic("")
    self.assertEqual("Id lab invalid!\n", str(ve.exception))


def test_raport_note_studenti_lab_dat_ordonati_alfabetic_srv_id_lab_inexistent(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
```

```
        with open(file_path,"w") as f:
            f.write("")
        repo_studenti = FileRepoStudent(file_path)
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
        srv_lab = ServiceLaborator(valid_lab,repo_lab)
        id_nota = 1
        nota_pb = 9.5
        srv_student.adauga_student(1,"Tudor",213)
        srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
        srv.asignare_nota(id_nota,1,"1_1")
        srv.adauga_nota(id_nota,nota_pb)
        srv_student.adauga_student(2,"Ana",211)
        srv.asignare_nota(id_nota+1,2,"1_1")
        srv.adauga_nota(id_nota+1,nota_pb+0.3)
        with self.assertRaises(RepositoryError) as re:
            srv.raport_studenti_cu_note_alfabetic("219_2")
        self.assertEqual("Aceasta problema nu a fost asignata niciunui student SAU a fost asignata si nu notata!\n", str(re.exception))


    def test_raport_note_studenti_lab_dat_ordonati_dupa_nota_srv_succes(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_note = FileRepoNote(file_path)
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
```

```
            f.write("")
        repo_studenti = FileRepoStudent(file_path)
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
        srv_lab = ServiceLaborator(valid_lab,repo_lab)
        id_nota = 1
        nota_pb = 9.5
        srv_student.adauga_student(1,"Tudor",213)
        srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
        srv.asignare_nota(id_nota,1,"1_1")
        srv.adauga_nota(id_nota,nota_pb)
        srv_student.adauga_student(2,"Ana",211)
        srv.asignare_nota(id_nota+1,2,"1_1")
        srv.adauga_nota(id_nota+1,nota_pb+0.3)
        srv_student.adauga_student(3,"Gabi",211)
        srv.asignare_nota(id_nota+3,3,"1_1")
        srv.adauga_nota(id_nota+3,nota_pb+0.4)
        all_el = srv.get_all_note_sir()
        self.assertTrue(len(all_el)==3)
        descrescator_dupa_nota = srv.raport_studenti_cu_note_dupa_nota("1_1")
        self.assertTrue(len(descrescator_dupa_nota)==3)
        #self.assertTrue(str(descrescator_dupa_nota[0]) == "Nota cu id: 4 -> Student:  Gabi -> problema:
1_1 -> nota: 9.9.")
        #self.assertTrue(str(descrescator_dupa_nota[1]) == "Nota cu id: 2 -> Student:  Ana -> problema: 1_1
-> nota: 9.8.")
        #self.assertTrue(str(descrescator_dupa_nota[2]) == "Nota cu id: 1 -> Student:  Tudor -> problema:
1_1 -> nota: 9.5.")
        self.assertTrue(str(descrescator_dupa_nota[0]) == "Gabi are nota: 9.9.")
```

```
        self.assertTrue(str(descrescator_dupa_nota[1]) == "Ana are nota: 9.8.")
        self.assertTrue(str(descrescator_dupa_nota[2]) == "Tudor are nota: 9.5.")


    def test_raport_note_studenti_lab_dat_ordonati_dupa_nota_srv_id_lab_invalid(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_note = FileRepoNote(file_path)
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_studenti = FileRepoStudent(file_path)
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
        srv_lab = ServiceLaborator(valid_lab,repo_lab)
        id_nota = 1
        nota_pb = 9.5
        srv_student.adauga_student(1,"Tudor",213)
        srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
        srv.asignare_nota(id_nota,1,"1_1")
        srv.adauga_nota(id_nota,nota_pb)
        srv_student.adauga_student(2,"Ana",211)
        srv.asignare_nota(id_nota+1,2,"1_1")
        srv.adauga_nota(id_nota+1,nota_pb+0.3)
        with self.assertRaises(ValidationError) as ve:
            srv.raport_studenti_cu_note_dupa_nota("42")
```

```
        self.assertEqual("Id lab invalid!\n", str(ve.exception))



    def test_raport_note_studenti_lab_dat_ordonati_dupa_nota_srv_id_lab_inexistent(self):
        file_path = "test_note.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_note = FileRepoNote(file_path)
        file_path = "test_studenti.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_studenti = FileRepoStudent(file_path)
        file_path = "test_laboratoare.txt"
        with open(file_path,"w") as f:
            f.write("")
        repo_lab = FileRepoLaborator(file_path)
        valid_note = ValidatorNote()
        srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
        valid_student = ValidatorStudent()
        valid_lab = ValidatorLaborator()
        srv_student = ServiceStudent(valid_student,repo_studenti)
        srv_lab = ServiceLaborator(valid_lab,repo_lab)
        id_nota = 1
        nota_pb = 9.5
        srv_student.adauga_student(1,"Tudor",213)
        srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
        srv.asignare_nota(id_nota,1,"1_1")
        srv.adauga_nota(id_nota,nota_pb)
        srv_student.adauga_student(2,"Ana",211)
        srv.asignare_nota(id_nota+1,2,"1_1")
        srv.adauga_nota(id_nota+1,nota_pb+0.3)
        with self.assertRaises(RepositoryError) as re:
            srv.raport_studenti_cu_note_dupa_nota("219_2")
        self.assertEqual("Aceasta problema nu a fost asignata niciunui student SAU a fost asignata si nu
notata!\n", str(re.exception))
```

```
def test_get_all_pt_raport2_repo_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo = FileRepoNote(file_path)
    id_nota = 12
    student = Student(1,"Gabi",213)
    lab = Laborator("1_1","usor",[13,12,2021])
    nota = NoteDTO(id_nota,student.get_id_student(),lab.get_nrlab_nrpb(),0)
    nota_pb = 8
    id_nota2 = 888
    student2 = Student(2,"Titi",2143)
    lab2 = Laborator("1_2","mediu",[11,12,2021])
    nota2 = NoteDTO(id_nota2,student2.get_id_student(),lab2.get_nrlab_nrpb(),0)
    repo.asignare_nota(nota)
    self.assertTrue(len(repo)==1)
    repo.adauga_nota(nota,nota_pb)
    get_all_fara_none1 = repo.get_all_pt_raport()
    self.assertTrue(len(get_all_fara_none1)==1)
    repo.asignare_nota(nota2)
    self.assertTrue(len(repo)==2)
    with self.assertRaises(RepositoryError) as re:
        repo.get_all_pt_raport()
    self.assertEqual("Exista laborator asignat si nenotat!\n", str(re.exception))
    repo.adauga_nota(nota2,nota_pb+1)
    get_all_fara_none2 = repo.get_all_pt_raport()
    self.assertTrue(len(get_all_fara_none2)==2)


def test_raport_medie_mai_mica_decat_cinci_srv_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
```

```
repo_note = FileRepoNote(file_path)
file_path = "test_studenti.txt"
with open(file_path,"w") as f:
    f.write("")
repo_studenti = FileRepoStudent(file_path)
file_path = "test_laboratoare.txt"
with open(file_path,"w") as f:
    f.write("")
repo_lab = FileRepoLaborator(file_path)
valid_note = ValidatorNote()
srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
valid_student = ValidatorStudent()
valid_lab = ValidatorLaborator()
srv_student = ServiceStudent(valid_student,repo_studenti)
srv_lab = ServiceLaborator(valid_lab,repo_lab)
id_nota = 1
nota_pb = 9.5
srv_student.adauga_student(1,"Tudor",213)
srv_lab.adauga_lab("1_1","ceva",[12,12,2021])
srv_lab.adauga_lab("1_2","greu",[29,12,2021])
srv.asignare_nota(id_nota,1,"1_1")
srv.asignare_nota(id_nota+1,1,"1_2")
srv.adauga_nota(id_nota,nota_pb)
srv.adauga_nota(id_nota+1,nota_pb-0.5)
medii = srv.raport_studenti_medie_sub_cinci()
self.assertTrue(len(medii)==0)
srv_student.adauga_student(2,"Vlad",214)
srv.asignare_nota(id_nota+2,2,"1_1")
srv.asignare_nota(id_nota+3,2,"1_2")
srv.adauga_nota(id_nota+2,4)
srv.adauga_nota(id_nota+3,3.5)
medii2 = srv.raport_studenti_medie_sub_cinci()
self.assertTrue(len(medii2)==1)
self.assertTrue(str(medii2[0][0])=="Vlad")
```

```
    self.assertTrue(medii2[0][1]==3.75)



def test_raport_primele_cincizeci_la_suta_medie_peste_cinci_srv_succes(self):
    file_path = "test_note.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_note = FileRepoNote(file_path)
    file_path = "test_studenti.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_studenti = FileRepoStudent(file_path)
    file_path = "test_laboratoare.txt"
    with open(file_path,"w") as f:
        f.write("")
    repo_lab = FileRepoLaborator(file_path)
    valid_note = ValidatorNote()
    srv = ServiceNote(valid_note,repo_note,repo_studenti,repo_lab)
    valid_student = ValidatorStudent()
    valid_lab = ValidatorLaborator()
    srv_student = ServiceStudent(valid_student,repo_studenti)
    srv_lab = ServiceLaborator(valid_lab,repo_lab)
    id_nota = 1
    nota_pb = 9.5
    srv_student.adauga_student(1,"Tudor",213)
    srv_lab.adauga_lab("1_3","ceva",[12,12,2021])
    srv_lab.adauga_lab("1_2","greu",[29,12,2021])
    srv.asignare_nota(id_nota,1,"1_3")
    srv.asignare_nota(id_nota+1,1,"1_2")
    srv.adauga_nota(id_nota,nota_pb-0.5) #9
    srv.adauga_nota(id_nota+1,nota_pb-1.5) #8
    srv_student.adauga_student(2,"Vlad",214)
    srv.asignare_nota(id_nota+2,2,"1_3")
```

```
        srv.asignare_nota(id_nota+3,2,"1_2")
        srv.adauga_nota(id_nota+2,6)
        srv.adauga_nota(id_nota+3,5)
        rezultat = srv.raport_primele_cincizeci_la_suta_note_lab_medie_peste_cinci()
        self.assertTrue(len(rezultat)==1)
        self.assertTrue(str(rezultat[0])=="Laboratorul 1_2 are 2 note, iar media notelor de la acest laborator este: 6.5!")
        srv_student.adauga_student(3,"Mihai",214)
        srv_lab.adauga_lab("1_5","ceva",[12,12,2021])
        srv.asignare_nota(id_nota+4,1,"1_5")
        srv.asignare_nota(id_nota+5,2,"1_5")
        srv.adauga_nota(id_nota+4,6)
        srv.adauga_nota(id_nota+5,5)
        rezultat2 = srv.raport_primele_cincizeci_la_suta_note_lab_medie_peste_cinci()
        self.assertTrue(len(rezultat2)==1)
        srv_lab.adauga_lab("1_10","ceva",[12,12,2021])
        srv.asignare_nota(id_nota+6,1,"1_10")
        srv.asignare_nota(id_nota+7,3,"1_10")
        srv.adauga_nota(id_nota+6,3)
        srv.adauga_nota(id_nota+7,5)
        rezultat3 = srv.raport_primele_cincizeci_la_suta_note_lab_medie_peste_cinci()
        self.assertTrue(len(rezultat3)==1)
        srv_lab.adauga_lab("1_101","ceva",[12,12,2021])
        srv.asignare_nota(id_nota+8,1,"1_101")
        srv.asignare_nota(id_nota+9,3,"1_101")
        srv.adauga_nota(id_nota+8,7)
        srv.adauga_nota(id_nota+9,5)
        rezultat4 = srv.raport_primele_cincizeci_la_suta_note_lab_medie_peste_cinci()
        self.assertTrue(len(rezultat4)==2)
        self.assertTrue(str(rezultat4[0])=="Laboratorul 1_101 are 2 note, iar media notelor de la acest laborator este: 6.0!")
        self.assertTrue(str(rezultat4[1])=="Laboratorul 1_2 are 2 note, iar media notelor de la acest laborator este: 6.5!")
```