



Facultatea de Automatică și Calculatoare
Departamentul Calculatoare

PROIECT

~Graduos platform~



la disciplina

INGINERIE SOFTWARE

Proiect realizat de: Băldean Adela-Ștefania

Blaj Sergiu-Emanuel

Borbei Raul-Aurelian

Cusco Ana-Maria

Profesor coordonator: Marghescu Luminița-Mădălina

Grupa: 30235



Cuprins

1. OBIECTIVUL ȘI DESCRIEREA TEMEI	3
2. DIAGrame	4
2.1. DIAGRAMA CAZURILOR DE UTILIZARE	4
2.2. DIAGRAMA BAZEI DE DATE SI DE CLASE	5
2.3. DIAGrame DE INTERACȚIUNE	6
2.3.1. <i>Diagrama de activități -- Raul</i>	6
2.3.2. <i>Diagrama de comunicare -- Sergiu</i>	7
2.3.3. <i>Diagrama de secvențe -- Ana</i>	8
2.3.4. <i>Diagrama de stări -- Adela</i>	9
3. DESIGN PATTERNS	10
3.1. OBSERVER -- RAUL	10
3.2. FACTORY METHOD -- ANA	11
3.3. CHAIN OF RESPONSIBILITY -- ADELA	13
3.4. BUILDER -- SERGIU	15
4. MEDII DE PROIECTARE	17
5. MANUAL DE UTILIZARE	22
6. BIBLIOGRAFIE	29



1. Obiectivul și descrierea temei

Obiectivul proiectului este de a realiza o platformă de E-learning, o platformă de e-learning este un soft complex care permite gestionarea unui domeniu și al utilizatorilor acestuia. În cadrul acestei platforme se pot crea cursuri, se pot efectua evaluări sau autoevaluări, iar utilizatorii pot comunica în mod sincron sau asincron. Administrarea se face în mod facil, platforma fiind intuitivă și accesibilă.

Soluția pe care o oferim se bazează pe platforma Grados. Grados este o platformă puternică, sigură, open source, care ajută la crearea de cursuri eficiente în mediul on-line și la crearea experiențelor de învățare într-un mediu de colaborare, privat. Grados poate fi folosit în domenii de activitate precum educație.



Acest tip de platformă de e-learning oferă un mediu de învățare integrat și interactiv, care se bazează pe învățarea prin colaborare, lucru în echipă și schimbul de opinii. Aceasta înseamnă că profesorul construiește materialul de învățare cu care elevii interacționează. Nu este suficient ca elevii doar să parcurgă materialele, ei trebuie să le înțeleagă, și să explice și celorlalți elevi pentru a putea lucra împreună.

Reușita unui curs este dată de modalitatea elaborării lui. Platforma permite profesorilor să elaboreze cursurile în funcție de specificul grupului de elevi supus instruirii. Materialul poate fi structurat pe teme, unități de învățare sau lecții, astfel încât să ofere o abordare prietenoasă, facilă, intuitivă și accesibilă. Accentul nu se pune pe conținut, ci pe activitățile care presupun interacțiunea dintre participanți.



Platforma dispune de unelte și activități colaborative cu care profesorii pot lucra și învăța împreună cu elevii, folosind diverse module, precum: forum sau chat.



Unul dintre cele mai importante aspecte este evaluarea elevilor. Este pasul care asigură corelarea dintre obiective și performanțe.

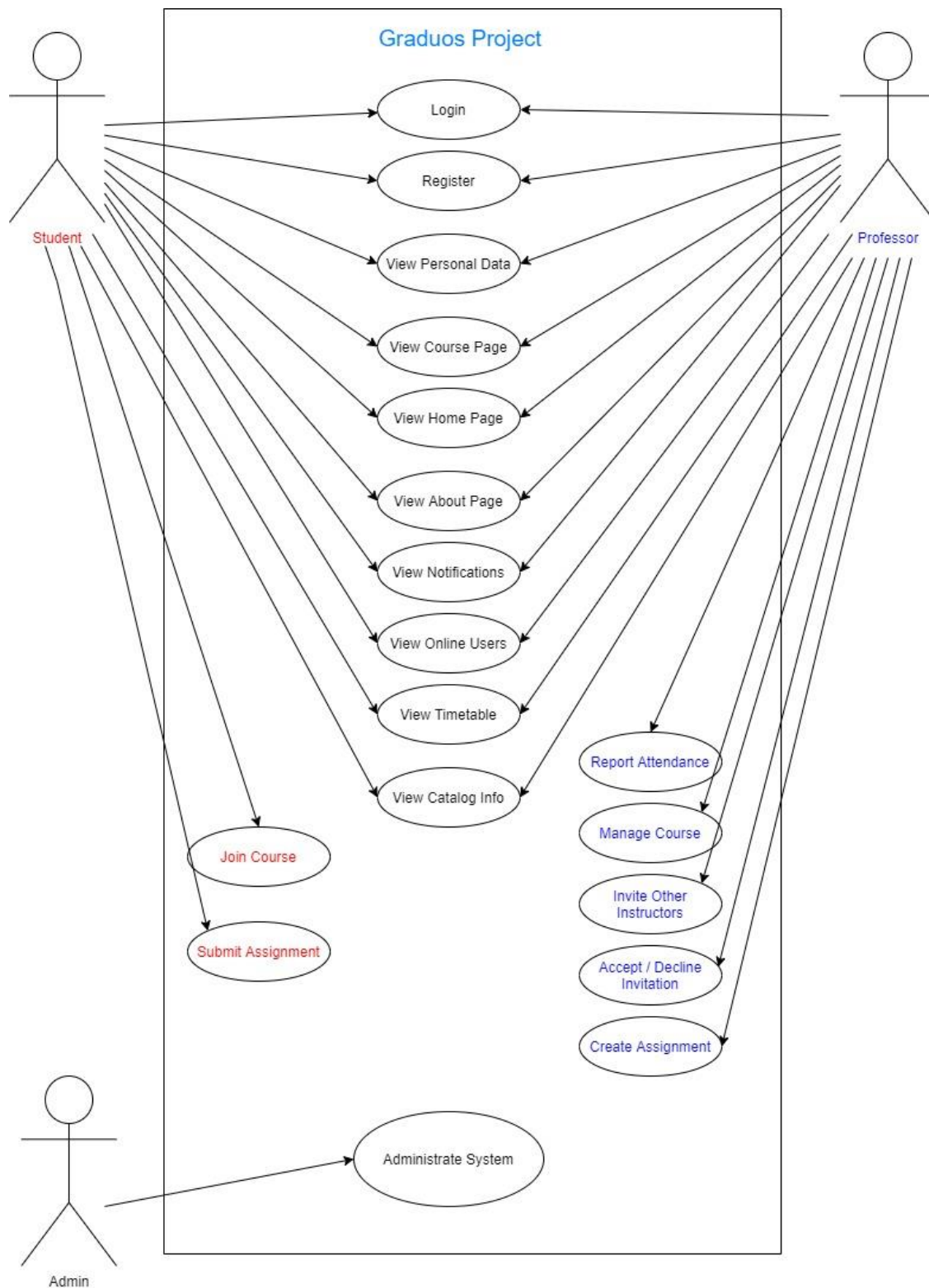
Profesorii pot crea assignment-uri care le pot fi asociate fiecărui student în parte. Studenții au posibilitatea de a participa activ la procesul de evaluare, încărcând fișiere cu rezolvările conforme cerințelor profesorului.

O evaluare corectă a studentului include atât rezultatele la teme și teste, cât și participarea activă la activități.



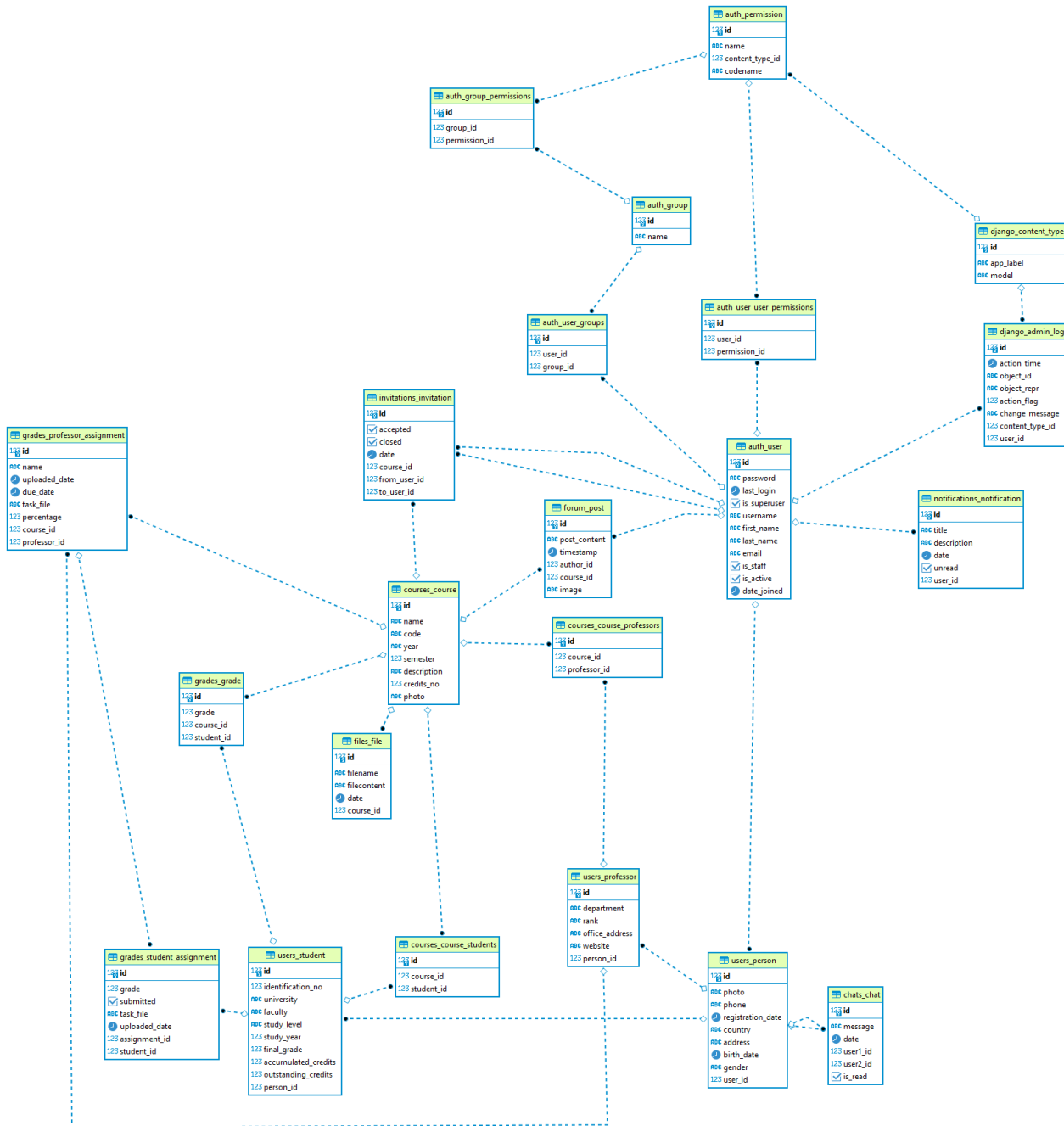
2. Diagrame

2.1. Diagrama cazurilor de utilizare





2.2. Diagrama bazei de date si de clase

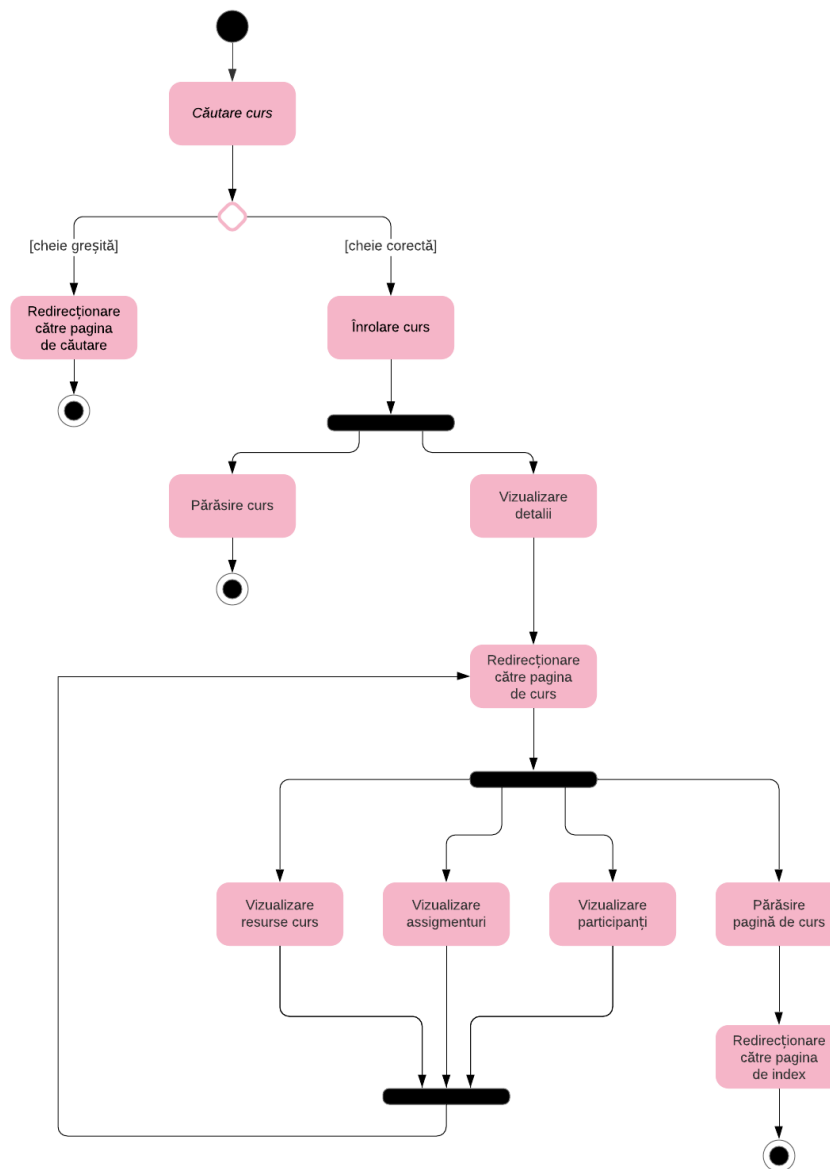




2.3. Diagrame de interacțiune

2.3.1. Diagrama de activități -- Raul

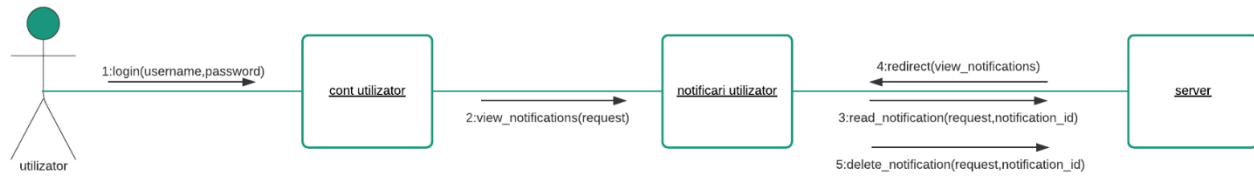
Diagramă de activități





2.3.2. Diagrama de comunicare -- Sergiu

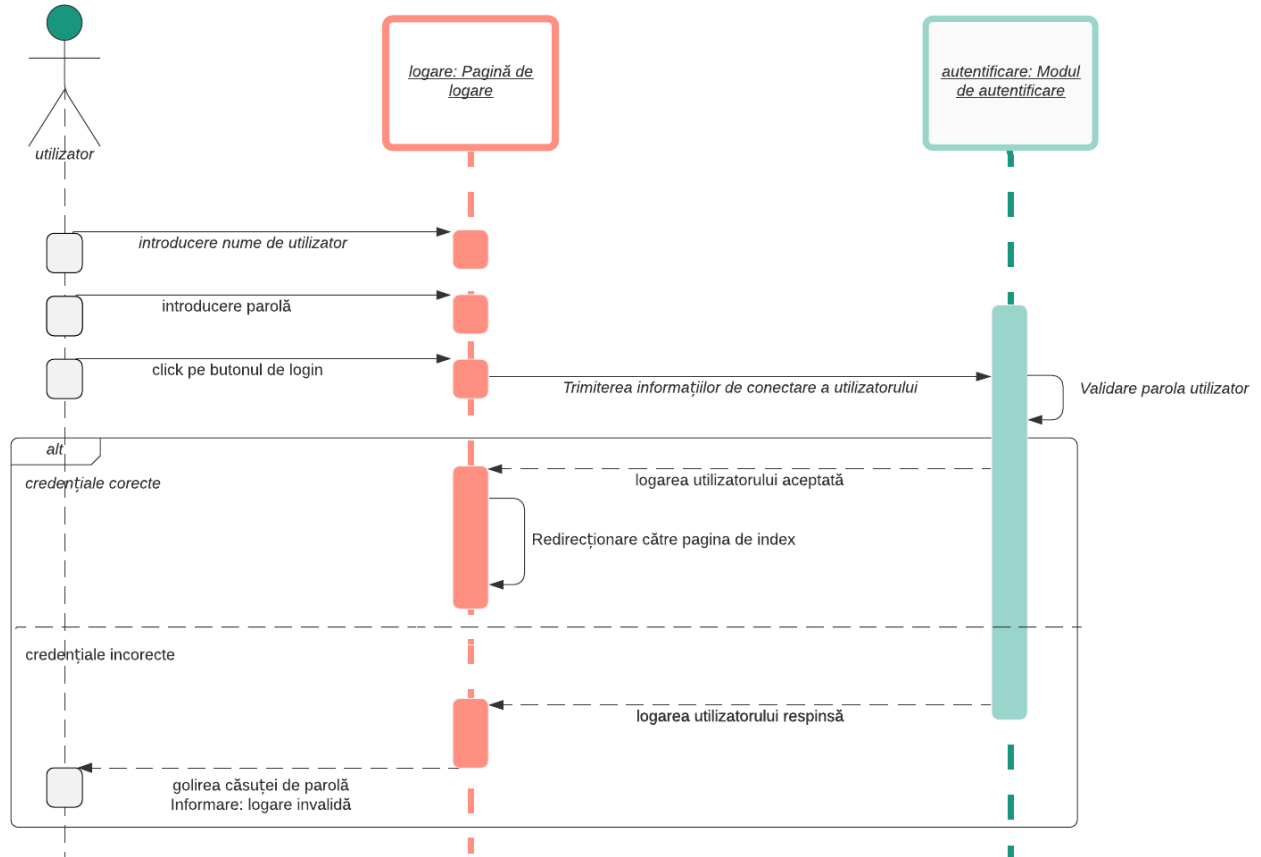
Diagramă de comunicare





2.3.3. Diagrama de secvențe -- Ana

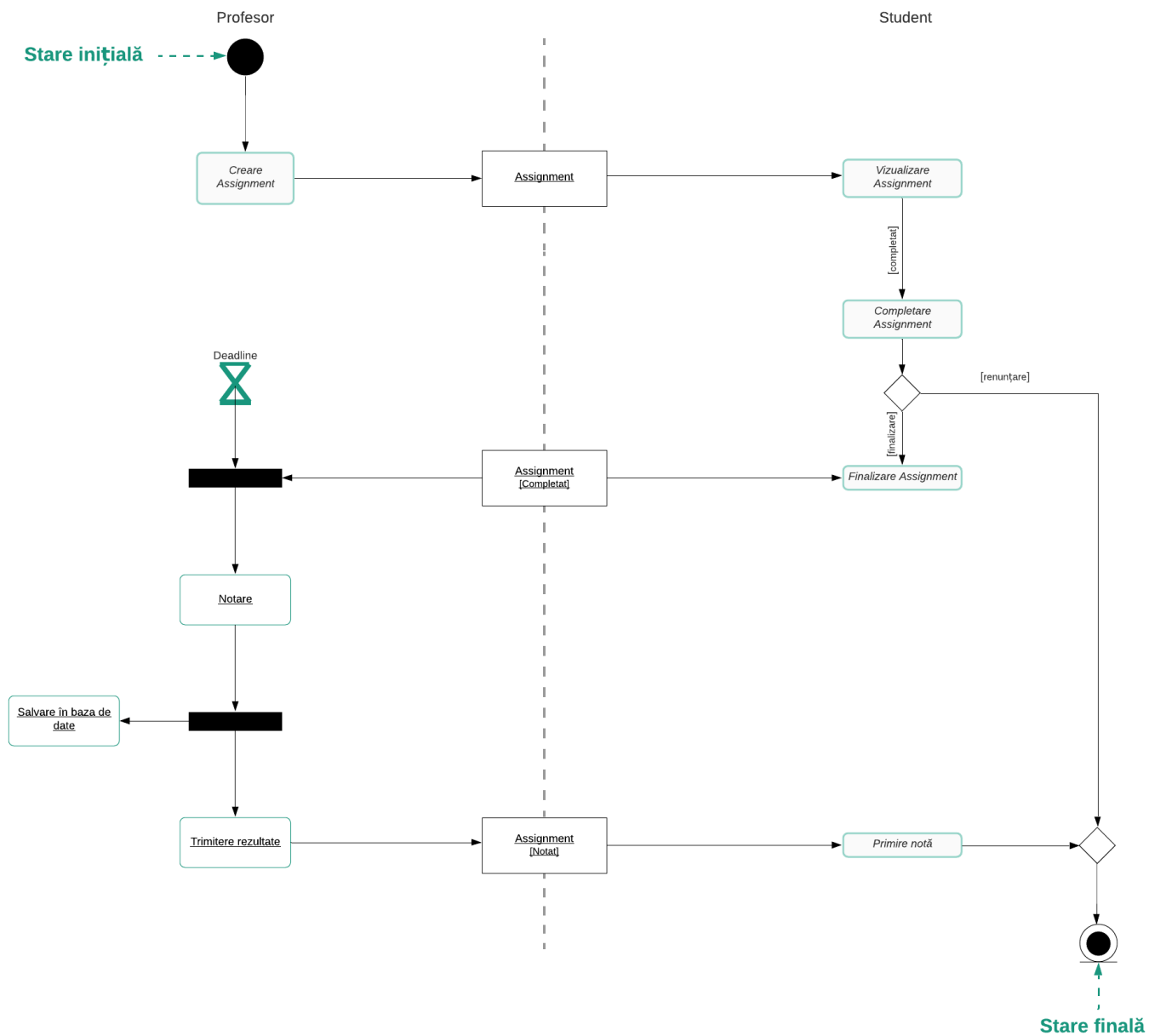
Diagramă de secvențe





2.3.4. Diagrama de stări -- Adela

Diagramă de stări





3. Design patterns

3.1. Observer -- Raul

Definiție

Modelul Observer este un model de proiectare software în care un obiect, numit subiect, menține o listă a dependenților săi, numiți observatori, și îi notifică automat despre orice schimbare de stare, de obicei apelând una dintre metodele lor. Folosirea acestui pattern implică existența unui obiect cu rolul de *subiect*, care are asociată o listă de obiecte dependente, cu rolul de *observatori*, pe care le apelează automat de fiecare dată când se întâmplă o acțiune.

Scop

Definește o dependență unu-la-mulți între obiecte, astfel încât atunci când un obiect își schimbă starea, toți dependenții săi să fie notificați și actualizați automat.

Încapsulează componentele de bază (sau comune sau motor) într-o abstractizare Subiect și componentele variabile (sau optionale sau interfața utilizator) într-o ierarhie Observer.

Problemă

Modelul Observer este folosit în procesul de creare al unui cont nou pentru a notifica utilizatorul în momentul în care datele unui câmp din înregistrare nu sunt corecte/ nu sunt permise sau acesta este gol. Verificarea datelor se face folosind modelul **Chain of Responsibility** descris în cele ce urmează, iar în momentul în care apare o problemă se setează starea modelului observat și se notifica observatorii pentru a afișa mesajul de eroare corespunzător .

```
user_observable.set_state('Successfully registered! You can now log in.')  
user_observable.notify_observers()
```

```
class UserObservable:  
    def __init__(self):  
        self.observers = []  
        self.state = None  
  
    def add_observer(self, observer):  
        self.observers.append(observer)  
  
    def set_state(self, new_state):  
        self.state = new_state
```



```
def notify_observers(self):
    for observer in self.observers:
        observer.update(self.state)

class UserObserver:
    def __init__(self, observer):
        self.observer = observer

    def update(self, state):
        messages.success(self.observer, state)
```

3.2. Factory Method -- Ana

Definiție

Pattern-urile de tip Factory sunt folosite pentru obiecte care generează instanțe de clase înrudite (implementează aceeași interfață, moștenesc aceeași clasă abstractă). Acestea sunt utilizate atunci când dorim să izolăm obiectul care are nevoie de o instanță de un anumit tip, de crearea efectivă acesteia. În plus clasa care va folosi instanța nici nu are nevoie să specifice exact subclasa obiectului ce urmează a fi creat, deci nu trebuie să cunoască toate implementările acelui tip, ci doar ce caracteristici trebuie să aibă obiectul creat. Din acest motiv, Factory face parte din categoria *Creational Patterns*, deoarece oferă o soluție legată de crearea obiectelor.

Folosind pattern-ul Factory Method se poate defini o interfață pentru crearea unui obiect. Clientul care apelează metoda factory nu știe/nu îl interesează de ce subtip va fi la runtime instanța primită.

Spre deosebire de Abstract Factory, Factory Method ascunde construcția unui obiect, nu a unei familii de obiecte “înrudite”, care extind un anumit tip. Clasele care implementează Abstract Factory conțin de obicei mai multe metode factory.

Scop

Definește o interfață pentru crearea unui obiect, dar lasă subclasele să decidă ce clasă să instanțieze. Metoda Factory permite unei clase să amâne instanțierea la subclase.

Problemă

Modelul Factory Method este folosit la înregistrarea unui utilizator, fie student, fie profesor. Astfel acesta oferă o metoda uniformă de înregistrare indiferent de tipul de persoana prin intermediul metodei *register_user*, distincția făcându-se la nivelul altor metode specifice: *register_student* si *register_professor*. Diferențierea între cele 2 tipuri de utilizatori se face folosind câmpul *is_student*.



```
class UserFactory:
    def __init__(self, request, is_student):
        self.request = request
        self.is_student = is_student

    def register_user(self):
        if self.is_student == 'True':
            self.register_student()
        else:
            self.register_professor()

    def register_student(self):
        person = Person.objects.get(user=User.objects.get(
            username=self.request.POST['username']))
        identification_no = self.request.POST['identification_no']
        university = self.request.POST['university']
        faculty = self.request.POST['faculty']
        study_level = self.request.POST['study_level']
        study_year = self.request.POST['study_year']

        student = Student.objects.create(person=person, identification_no=identification_no,
                                         university=university, faculty=faculty,
study_level=study_level,
                                         study_year=study_year)

        student.save()

    def register_professor(self):
        person = Person.objects.get(user=User.objects.get(
            username=self.request.POST['username']))
        department = self.request.POST['department']
        rank = self.request.POST['rank']
        office_address = self.request.POST['office_address']
        website = self.request.POST['website']

        professor = Professor.objects.create(
            person=person, department=department, rank=rank, office_address=office_address,
website=website)

        professor.save()
```



3.3. Chain of Responsibility -- Adela

Definiție

Chain of Responsibility este un model de design comportamental care vă permite să transmiteți cereri de-a lungul unui lanț de gestionari. La primirea unei cereri, fiecare handler decide fie să proceseze cererea, fie să o transmită următorului handler din lanț.

Scop

Evită cuplarea între expeditorul unei cereri și destinatarul acesteia, acordând mai multor obiecte șansa de a gestiona cererea. Se înlănțuie obiectele receptoare și se transmite cererea de-a lungul lanțului până când un obiect o gestionează.

Launch-and-leave cererile cu o singură conductă de procesare care conține mai mulți gestionari posibili.

Problemă

Modelul Chain of Responsibility este folosit pentru verificarea datelor de înregistrare a unui utilizator. Fiecare problema care poate apărea la nivelul unui câmp are o clasa specială (ex. EmailHandler, PasswordHandler – în cazul problemelor legate de e-mail, parola), clasa care definește o metoda *handle* care descrie modul de tratare al erorii și în care se verifică condițiile de validitate (ex. parolele trebuie să corespundă, altfel se va afișa un mesaj de eroare).

```
class Handler:
    def __init__(self):
        pass

    def handle(self):
        pass

class UserHandler(Handler):
    def __init__(self):
        super(UserHandler, self).__init__()
        self.handlers = []

    def handle(self):
        for handler in self.handlers:
            handler.handle()

    def add_handler(self, handler):
        self.handlers.append(handler)
```



```
class UsernameHandler(Handler):
    def __init__(self, username, all_users, request):
        super(UsernameHandler, self).__init__()
        self.username = username
        self.all_users = all_users
        self.request = request

    def handle(self):
        if self.all_users.filter(username=self.username).exists():
            messages.error(self.request, 'Username already taken!')

            return render(self.request, 'users/register.html')

class EmailHandler(Handler):
    def __init__(self, email, all_users, request):
        super(EmailHandler, self).__init__()
        self.email = email
        self.all_users = all_users
        self.request = request

    def handle(self):
        if self.all_users.filter(email=self.email).exists():
            messages.error(self.request, 'Email already taken!')

            return render(self.request, 'users/register.html')

class PasswordHandler(Handler):
    def __init__(self, password, password2, all_users, request):
        super>PasswordHandler, self).__init__()
        self.password = password
        self.password2 = password2
        self.all_users = all_users
        self.request = request

    def handle(self):
        if self.password != self.password2:
            messages.error(self.request, 'Passwords do not match!')

            return render(self.request, 'users/register.html')
```



3.4. Builder -- Sergiu

Definiție

Modelul Builder este un model de proiectare conceput pentru a oferi o soluție flexibilă la diverse probleme de creare de obiecte în programarea orientată pe obiecte. Intenția modelului de proiectare Builder este de a separa construcția unui obiect complex de reprezentarea sa. Acesta este unul dintre modelele de proiectare Gang of Four.

Scop

Separă construcția unui obiect complex de reprezentarea acestuia, astfel încât același proces de construcție să poată crea reprezentări diferite.

Analizează o reprezentare complexă, creează una din mai multe ținte.

Problemă

Design pattern-ul Builder este folosit pentru a crea un request în procesul de înregistrare al unui student/profesor nou. Astfel înregistrarea în 3 pași este construită în metoda `register_account` astfel: se salvează în baza de date informațiile despre utilizator (*django.user*) urmate de salvarea informațiilor despre o persoană, iar în final folosind câmpul *is_student* se va salva în baza de date obiectul student/profesor.

Modelul Builder construiește astfel procesul de înregistrare în setul de 3 pași descriși mai sus, pentru a se crea un cont nou este nevoie să se parcurgă toți pașii descriși de metoda **register_account**: *register_user()*, *register_person()* și *register_student()/register_professor(Factory Method)*.

```
class UserBuilder:
    def __init__(self, request):
        self.request = request

    def register_account(self):
        self.register_user()

        self.register_person()

        user_factory = UserFactory(self.request, self.request.POST['is_student'])
        user_factory.register_user()

    def register_user(self):
        first_name = self.request.POST['first_name'].capitalize()
        last_name = self.request.POST['last_name'].capitalize()
```



```
username = self.request.POST['username']
email = self.request.POST['email']
password = self.request.POST['password']
password2 = self.request.POST['password2']

all_users = User.objects.all()

user_handler = UserHandler()

user_handler.add_handler(UsernameHandler(username, all_users, self.request))
user_handler.add_handler(EmailHandler(email, all_users, self.request))
user_handler.add_handler>PasswordHandler(password, password2, all_users,
self.request))

user = User.objects.create_user(first_name=first_name, last_name=last_name,
                                username=username, email=email)
user.set_password(password)
user.save()

def register_person(self):
    user = User.objects.get(username=self.request.POST['username'])
    photo = self.request.FILES['photo']
    phone = self.request.POST['phone']
    country = self.request.POST['country']
    address = self.request.POST['address']
    birth_date = self.request.POST['birth_date']
    gender = self.request.POST['gender']

    person = Person.objects.create(user=user, photo=photo, phone=phone,
                                   country=country, address=address,
birth_date=birth_date, gender=gender)
    person.save()
```




4. Medii de proiectare

Pentru realizarea proiectului au fost folosite următoarele limbaje și framework-uri: python, html, css, javaScript, bootstrap, django. Aplicațiile folosite au fost: VSCode, PyCharm și pgAdmin.



Python este un limbaj de programare dinamic, de nivel înalt, ce pune accent pe expresivitatea și înțelegerea ușoară a codului. Sintaxa sa permite implementări echivalente cu alte limbaje în mai puține linii de cod. Datorită acestui fapt, Python este foarte răspândit atât în programarea de aplicații, cât și în zona de scripting.

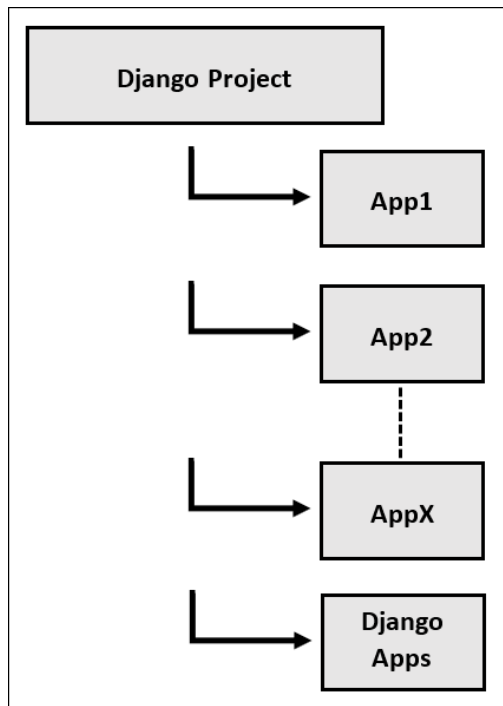
Limbajul facilitează mai multe paradigme de programare, în special paradigma imperativa (C) și pe cea orientată pe obiecte (Java). Spre deosebire de C, Python nu este un limbaj compilat, ci interpretat. Acest fapt are atât avantaje, cât și dezavantaje. Pe de-o parte, Python este mai lent decât C. Pe de altă parte, aplicațiile Python sunt foarte ușor de depanat, codul putând fi ușor inspectat în timpul rulării. De asemenea, este foarte ușor de experimentat cu mici fragmente de cod folosind interpretorul Python.



Django este un soft cadru pentru dezvoltarea aplicațiilor web (en. web application framework) gratuit și cu sursă deschisă, scris în Python, care urmează modelul arhitectural Model-View-Controller.

Scopul principal al acestui soft cadru pentru dezvoltarea aplicațiilor web este de a facilita crearea de website-uri complexe, fondate pe baze de date. Django pune accent pe reutilizarea codului, pe modularitate, dezvoltare rapidă a site-urilor web, ghidându-se după principiul "nu te repeta" (en. Don't repeat yourself - DRY). Django este codat de la un capăt la altul în Python, chiar și fișierele de configurare și modelele de date sunt implementate în acest limbaj de programare. Django oferă și un panou administrativ, care, deși vine preinstalat, este opțional, prin intermediul acestuia se pot crea, citi, actualiza și șterge cu ușurință informații din baza de date. Acest panou de administrare este generat dinamic prin introspecție (prin analizarea tabelor din baza de date) și poate fi ușor configurat prin modelele administrative de date.

Câteva website-uri bine cunoscute care utilizează Django sunt Pinterest, Instagram, Mozilla, The Washington Times, Disqus, Public Broadcasting Service și Bitbucket.



Unitatea fundamentală a unei aplicații web Django este un proiect Django. Un proiect Django cuprinde una sau mai multe aplicații Django.

O aplicație Django este un pachet de sine stătător care trebuie să facă un singur lucru. De exemplu, un blog, o aplicație pentru membri sau un calendar de evenimente. Observați că în partea de jos a figurii, există un pachet suplimentar numit Django Apps.

Acesta este un alt caz în care logica Django se transpune direct în cadru - Django însuși este o colecție de aplicații, fiecare dintre ele fiind concepută pentru a face un singur lucru. Cu aplicațiile încorporate în Django, toate sunt concepute pentru a face viața mai ușoară, ceea ce este un lucru bun.

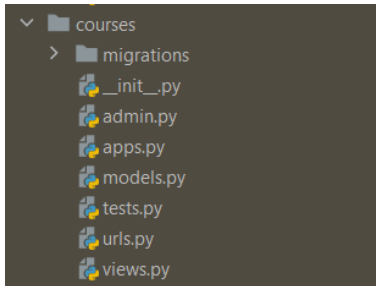
Aplicațiile create sunt vizibile în fișierul *settings.py*:

```
# Application definition

INSTALLED_APPS = [
    'users.apps.UsersConfig',
    'courses.apps.CoursesConfig',
    'grades.apps.GradesConfig',
    'chats.apps.ChatsConfig',
    'notifications.apps.NotificationsConfig',
    'files.apps.FilesConfig',
    'invitations.apps.InvitationsConfig',
    'forum.apps.ForumConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.humanize',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```



Structura unei aplicații Django:



Dosarul **migrations** este folderul în care Django stochează migrațiile sau modificările aduse bazei de date.

- **__init__.py** îi spune lui Python că aplicația dvs. este un pachet.
- **admin.py** este locul în care se înregistrează modelele aplicației cu aplicația de administrare Django.
- **apps.py** este un fișier de configurare comun tuturor aplicațiilor Django.
- **models.py** este modulul care conține modelele pentru aplicație
- **tests.py** conține procedurile de testare care se execută la testarea aplicației.
- **views.py** este modulul care conține paginile pentru aplicația dumneavoastră.

URLconfs - Navigatorul Django

Mai există o ultimă piesă în puzzle-ul cadrului Django - calea de comunicare critică care face să corespundă o cerere din partea clientului cu o resursă a proiectului (săgețile dintre vizualizare și șablon din figura de mai jos). La fel ca toate aplicațiile web, Django utilizează localizatori uniformi de resurse (Uniform Resource Locators - URL) pentru a face să corespundă conținutul cu o cerere.

Pachetul `urls` din Django oferă zeci de funcții și clase pentru a lucra cu diferite formate URL, rezolvarea numelor, gestionarea excepțiilor și alte utilități de navigare. Cu toate acestea, la modul cel mai elementar, acesta permite asocierea unui URL cu o funcție sau o clasă din cadrul proiectului Django.

O configurație URL Django (sau, pe scurt, URLconf) potrivește un URL unic cu o resursă a proiectului. Vă puteți gândi că este ca și cum ar fi ca și cum ați potrivi numele unei persoane cu adresa acesteia. Cu excepția faptului că în Django, nu potrivim o adresă de stradă, ci o cale Python folosind notația Python cu puncte.

Notația cu puncte este un idiom comun în programarea orientată pe obiecte. În Python, operatorul punct indică următorul obiect din lanțul de obiecte.

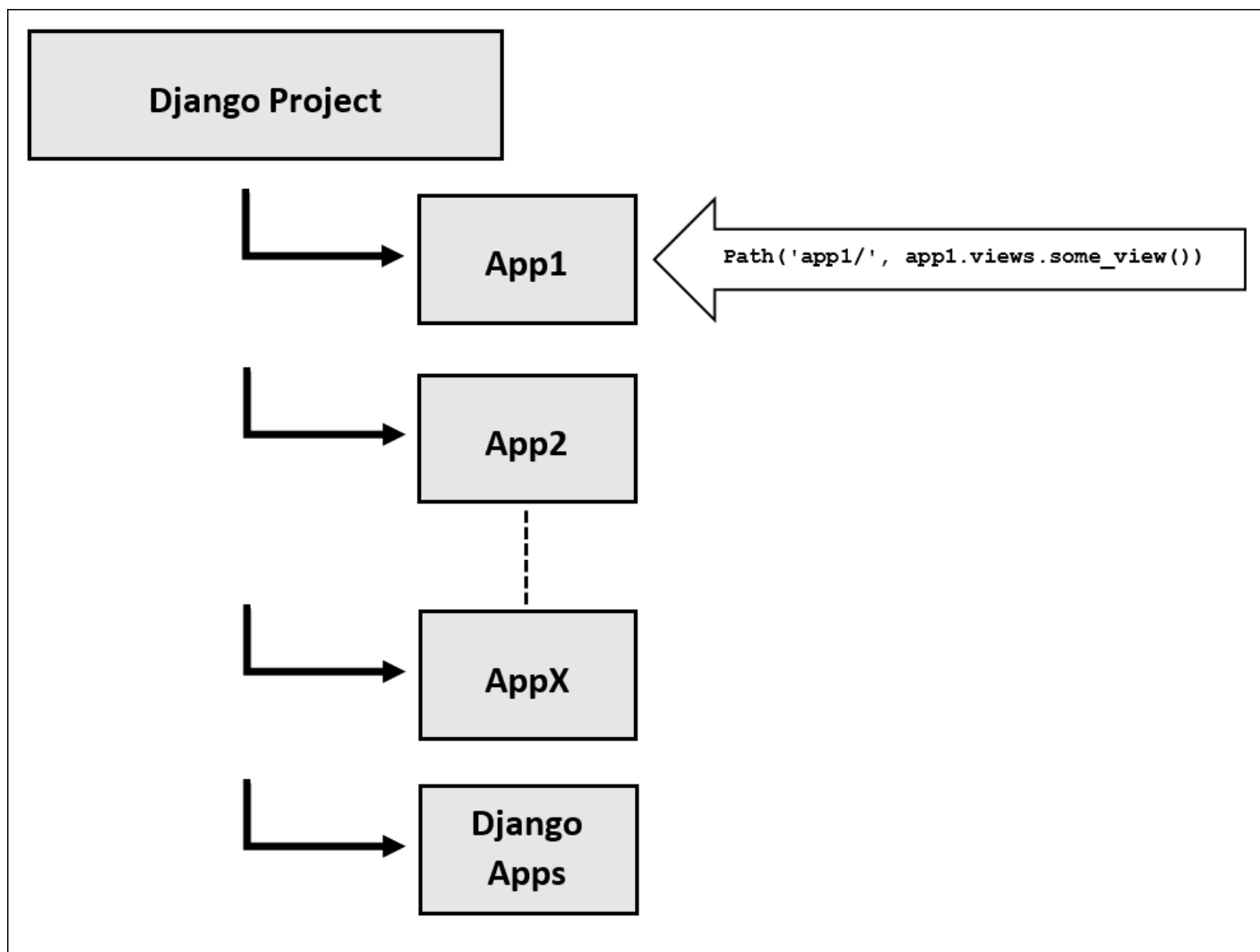
În clasele Django, lanțul de obiecte este astfel:

- *package.module.class.method*

Sau cu funcții:

- *package.module.function.attribute*

În cazul unui URLconf, calea de acces indică o funcție sau o clasă din interiorul unui modul (fișier .py).



Pentru a crea un URLconf, se utilizează funcția `path()`. Prima parte a funcției este URL-ul, astfel încât în figura de mai jos URL-ul este `app1/`. Funcția `path()` mapează apoi acest URL către `app1.views.some_view()`.

Presupunând că adresa site-ului dvs. este `http://www.graduos.com` vom avea:

"Atunci când cineva navighează la `http://www.graduos.com/app1/`, executați funcția `some_view()` în interiorul fișierului `views.py` al lui `app1`".



HTML este o formă de marcare orientată către prezentarea documentelor text pe o singură pagină, utilizând un software de redare specializat, numit agent utilizator HTML, cel mai bun exemplu de astfel de software fiind browser-ul web. HTML furnizează mijloacele prin care conținutul unui document poate fi adnotat cu diverse tipuri de metadata și indicații de redare. Indicațiile de redare pot varia de la decorațiuni minore ale textului, cum ar fi specificarea faptului că un anumit cuvânt trebuie subliniat sau că o imagine trebuie introdusă, până la scripturi sofisticate, hărți de imagini și formulare. Metadatale pot include informații despre titlul și autorul documentului, informații structurale despre cum este împărțit documentul în diferite segmente, paragrafe, liste, titluri etc. și informații cruciale care permit ca documentul să poată fi legat de alte documente pentru a forma astfel hyperlink-uri (sau web-ul).

HTML



CSS



Css-ul este un limbaj de stilizare al elementelor html, al tagurilor html. Denumirea CSS provine din expresia Cascading Style Sheets. În Web Design-ul modern, pentru stilizarea paginilor web se folosește numai CSS. Acest lucru înseamnă că de la culoarea literelor și a backgroundului până și la poziționarea elementelor de pe o pagină web, totul este stilizat prin CSS. Stilurile folosite pe o pagină pot fi incorporate în pagina respectivă sau pot fi chemate din fișiere externe, fișiere css.

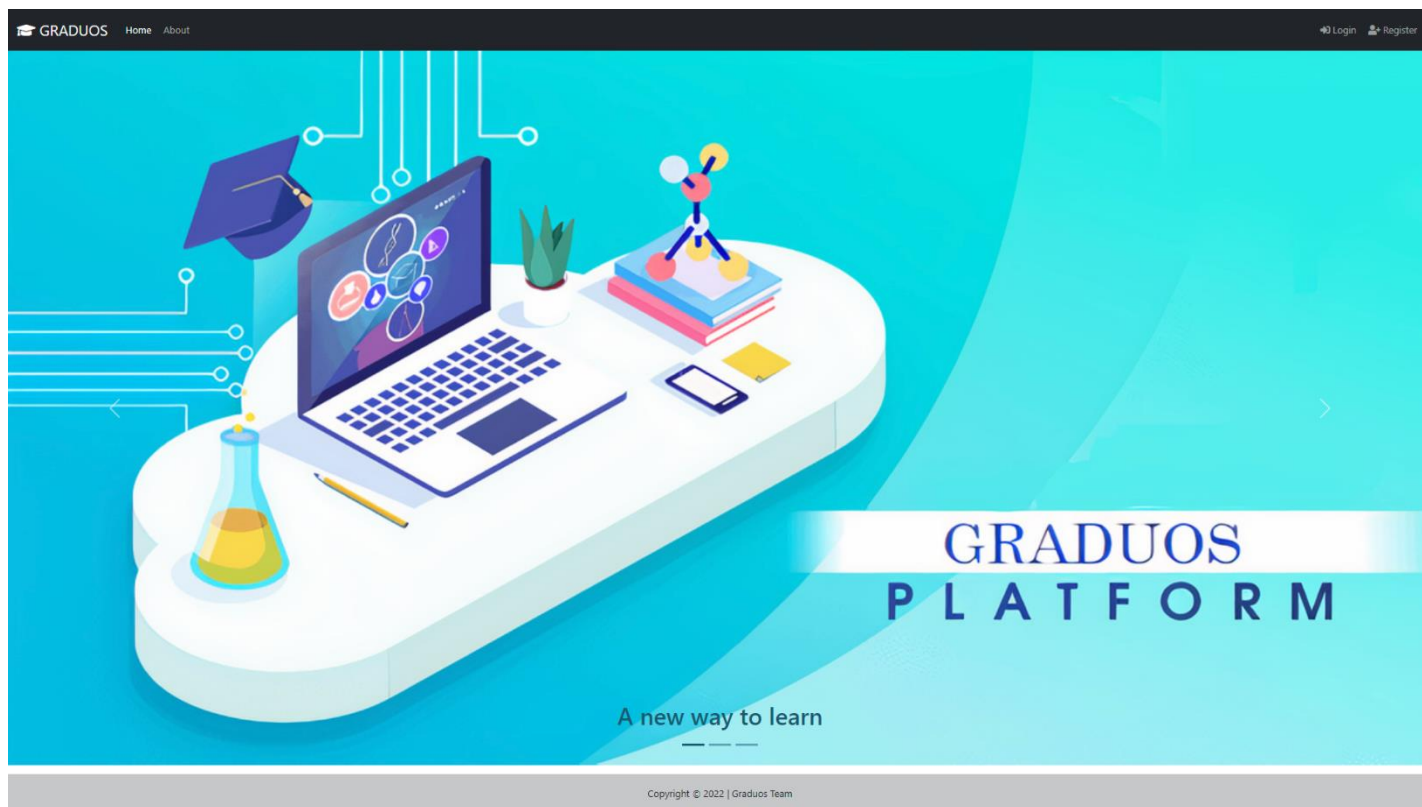
Bootstrap este o bibliotecă gratuită, open-source, pentru proiectarea de site-uri web și aplicații web. Acesta conține șabloane de design HTML și CSS pentru orice, de la meniuri de navigație, butoane, imagini, fonturi, formulare și alte componente de interfață, precum și extensii JavaScript.





5. Manual de utilizare

La deschiderea site-ului suntem întâmpinați de o fereastră de Home care conține un slideshow de imagini relevante.



Se poate observa că site-ul este îmbrățișează un stil minimalist și simplu pentru a facilita o navigare și o utilizare ușoară. Partea de sus a paginii conține un navbar fixat care se afla pe toate paginile din cadrul site-ului și cu ajutorul căruia se poate naviga între pagini.

Pagina de about a site-ului vorbește puțin despre proiectul realizat de noi și despre echipa GraduOS.



Graduos History

It all started when a group of students noticed that their university platform crashed when needed the most.
 So they came with the idea of creating a new learning platform.
 After the name has changed from 'Juky' to 'Graduos', the team started to elaborate the platform.

Graduos Team

Everyone says it, but in our case it's true: our team is the secret to our success. Each of our member is amazing in their own right, but together they are what makes Graduos such a fun and rewarding place to work.
The Graduos team is a talented group with a shared vision of delivering consistently great results for the users of our platform. We're very proud of the team we've built – there's just four of us now but it feels like we are a family.

All team members are unique individuals who are united by a set of five core values that apply to everything we do.

BE BOLD: be proactive, make decisions, take responsibility, try new things.

BE CURIOUS: ask questions, do some research, learn new techniques, study our clients and their industries.

BE TOGETHER: play an active role in the team, support your colleagues, collaborate, have fun.

BE CONNECTED: meet people, make contacts, build relationships, see the bigger picture.

BE BETTER: look for ways to improve, challenge yourself, never stop learning, strive to be the best.

Building, developing, training, retaining and engaging the Graduos team is a daily commitment.
 We work hard every day to make sure that the users of the platform have an exceptional experience.



Băldean Adela Ștefania

Front-end dev



Blaj Sergiu Emanuel

Full-stack dev



Borbei Raul Aurelian

Front-end dev



Cusco Ana Maria

Back-end dev

Pentru a crea un cont pe platforma noastră se va accesa pagina de register și se vor introduce datele cerute. Utilizatorul va alege daca este un student sau un profesor și va apăsa pe butonul de register.




	Username	<input type="text" value="Randrei"/>
	Password	<input type="password" value="....."/>
	Confirm password	<input type="password" value="....."/>
	Email	<input type="text" value="Randrei@graduos.com"/>
	First name	<input type="text" value="Andrei"/>
	Last name	<input type="text" value="Rusu"/>


	Photo	<input type="text" value="Choose File andrei.jpg"/>
	Phone	<input type="text" value="0754821365"/>
	Country	<input type="text" value="Romania"/>
	Address	<input type="text" value="Cluj Napoca nr.77"/>
	Photo	<input type="text" value="Choose File andrei.jpg"/>
	Phone	<input type="text" value="0754821365"/>
	Country	<input type="text" value="Romania"/>
	Address	<input type="text" value="Cluj Napoca nr.77"/>
	Birthdate	<input type="text" value="13-Jul-2000"/>
	Gender	<input type="radio"/> F <input checked="" type="radio"/> M
	Student	<input checked="" type="radio"/> Yes <input type="radio"/> No


	University	<input type="text" value="Technical University of Cluj-Napoca"/>
	Faculty	<input type="text" value="AC"/>
	Study level	<input checked="" type="radio"/> Bachelor <input type="radio"/> Master <input type="radio"/> Doctor
	Year of study	<input type="text" value="3"/>
	Identification number	<input type="text" value="1256598"/>

[Register](#)

În urma apăsării butonului register, dacă datele introduse sunt în conformitate cu cerințele utilizatorului va fi redirecționat către pagina de login unde se va putea loga folosind credențialele sale.



 Username

 Password

[Log in](#)

După introducerea datelor de către studentul nostru acesta este redirecționat către pagina de dashboard unde observăm că a fost invitat să se înscrie la un curs.

Dashboard

Logged in as Andrei Rusu. [View profile](#)

New invitation

You have been invited to join course Grafica pe calculator by Prodan Mihaela

[Accept](#)[Decline](#)

Jan 2, 2022, 12:25 a.m.

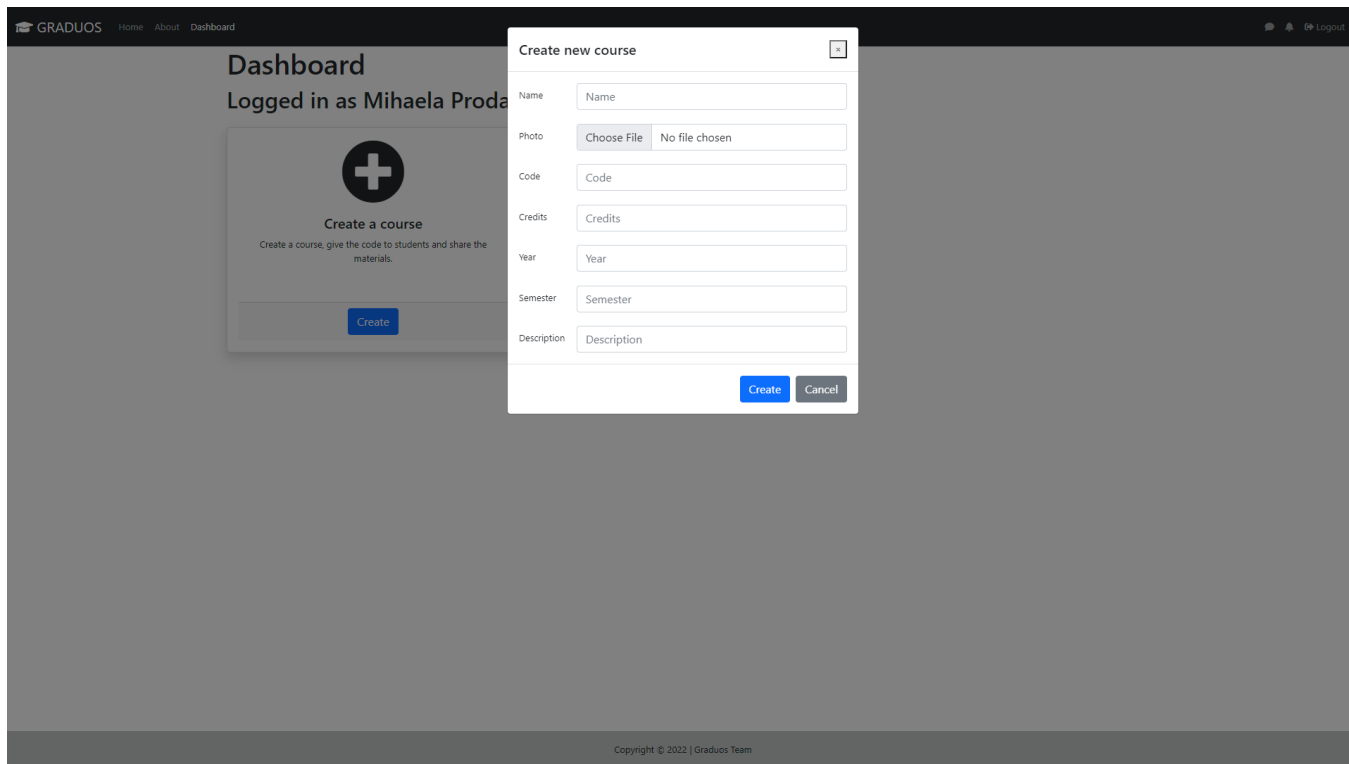


Join a course

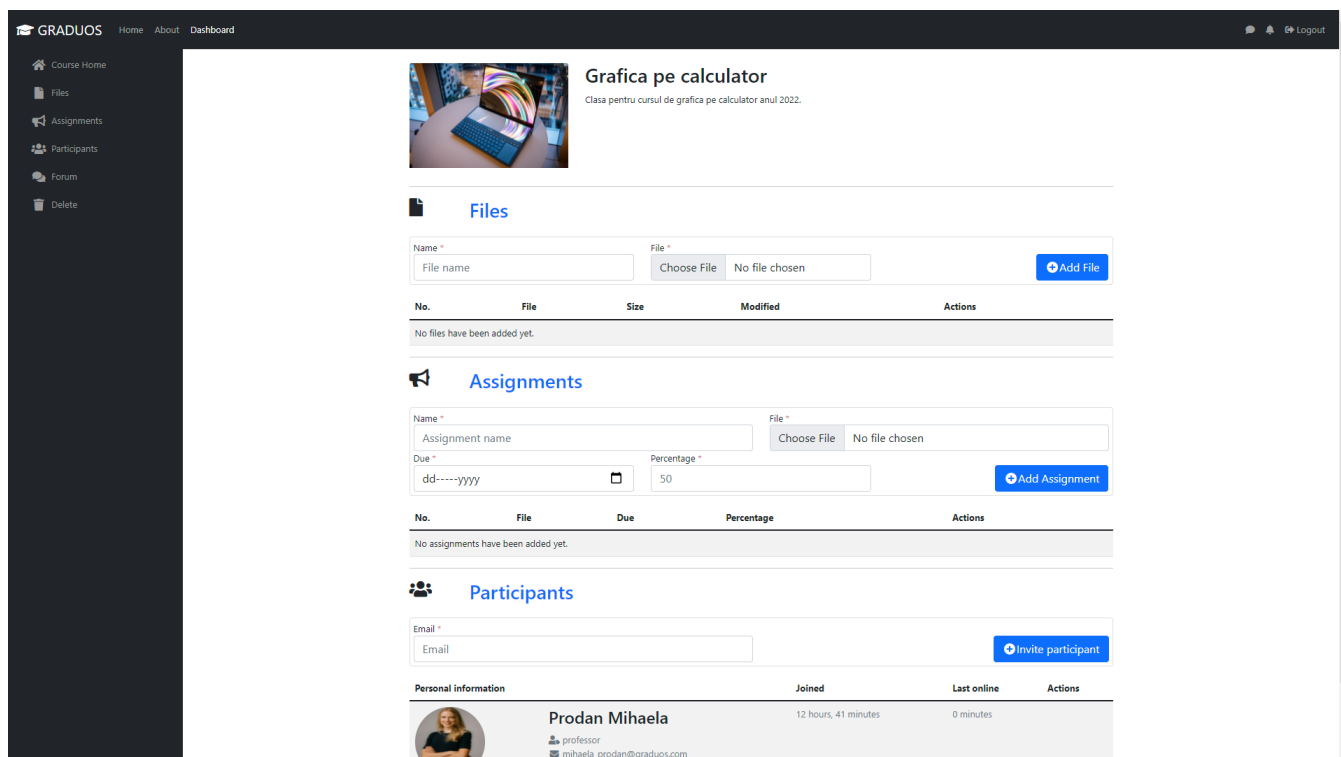
Join a course by entering the code given by teacher

[Join](#)

Cursul era deja creat de către un profesor folosind meniul de creare a cursului



Pagina de curs este următoarea:





În cadrul paginii de curs un profesor poate posta fișiere de curs pentru studenți, poate crea un assignment pe care studenții să îl rezolve și poate să își invite sau să dea afara participanții. Pe lângă meniul de sus, pagina de curs dispune de un meniu lateral fixat care să te redirecționeze cât mai repede la secțiunea dorită, deoarece spre finele semestrului, pe măsura ce se adaugă fișiere și assignmenturi pagina poate să devină foarte mare. Pagina de curs dispune de asemenea de un forum accesibil participanților la curs.

GRADUOS
Home
About
Dashboard
Logout

Grafica pe calculator Discussion Forum

This is the Q&A section. Here you can chat with your colleagues and teachers.

Jan. 2, 2022, 12:48 p.m.
Mihaelapro
hello

Jan. 2, 2022, 12:48 p.m.
Mihaelapro
e ok

Type your message
Send

Copyright © 2022 | Graduos Team

Profilul de Graduos poate fi văzut și modificat din cadrul paginii de profil accesibilă din dashboard.

GRADUOS
Home
About
Dashboard
Logout

Choose File
No file chosen

Randrei
Randrei@graduos.com

Profile Settings

Username
Randrei

Email
Randrei@graduos.com

First Name
Andrei

Last Name
Rusu

Phone
0754821365

Country
Romania

Address
Cluj Napoca nr.77

Birth date
July 13, 2000

Gender
M

Student info

University
Technical University of Cluj-Napoca

Faculty
AC

Study level
Bachelor

Year of study
3

Identification number
1256598

Grades

No.	Course	Grade
No assignments have been added yet.		

Save
Cancel

Copyright © 2022 | Graduos Team



De asemenea site-ul dispune de o pagina cu notificări importante pentru utilizatori.

The screenshot shows the 'GRADUOS' dashboard with a dark header bar containing navigation links (Home, About, Dashboard) and a 'Logout' button. The main content area is titled 'Notifications' and displays five notification cards, each with a blue dot icon, a title, a message, and a timestamp.

Notification Title	Message	Timestamp
File added	You have successfully added file Curs2 to course Grafica pe calculator. File size: 281156 kB.	Jan. 2, 2022, 12:27 a.m.
File added	You have successfully added file Curs1 to course Grafica pe calculator. File size: 8183411 kB.	Jan. 2, 2022, 12:27 a.m.
Invitation accepted	Rusu Andrei accepted your invitation to join course Grafica pe calculator. All the best!	Jan. 2, 2022, 12:26 a.m.
Invitation sent	You have successfully invited Rusu Andrei to join course Grafica pe calculator. Hope he accepts the invitation.	Jan. 2, 2022, 12:25 a.m.
New course created	Congratulations! You have successfully created course Grafica pe calculator. Share the code GC2022 to your students.	Jan. 2, 2022, 12:24 a.m.

La finalizarea muncii se apasă butonul de logout care deloghează utilizatorul și îl redirecționează spre pagina de login.



6. Bibliografie

<https://lucid.app/documents>

https://sourcemaking.com/design_patterns/creational_patterns

<https://refactoring.guru/design-patterns/python>

<https://djangobook.com/mdj2-django-structure/>

<https://docs.python.org/3/>

<https://docs.djangoproject.com/en/4.0/>