Razvoj web aplikacije korištenjem okvira Django i biblioteke HTMX

Kaurinović, Tvrtko

Sveučilište u Rijeci – Fakultet Informatike i Digitalnih Tehnologija

Informatika

Tvrtko Kaurinović

Razvoj web aplikacije korištenjem okvira Django i biblioteke HTMX

Završni rad

Mentor: doc. dr. sc. Vedran Miletić

Rijeka, 28. rujna 2022.

Sažetak

U ovom završnom radu opisano je kako instalirati i koristiti programski okvir Django i njegovu biblioteku htmx, te kako uz pomoć njihove tehnologije izraditi dinamičku web aplikaciju. Django je besplatan, web programski okvir otvorenog izvora baziran na programskom jeziku Python i koristi strukturu Model-Pogled-Predložak za obradu podataka i komunikaciju između klijenta i servera. Model nam predstavlja našu bazu podataka, svaki model je kao jedna tablica sa svojim atributima. U pogledu se nalazi sva logika koja povezuje model i predložak i kojom upravljamo kako će podatci biti obrađeni i prikazani. Na posljetku predložak služi za generiranje korisničkog sučelja i prikaz podataka. Iz njega šaljemo zahtjeve i na njega primamo odgovore. Biblioteka htmx je veoma jednostavna za korištenje. Potrebno je samo uključiti ga u projekt pomoću jedne linije unutar oznake. Htmx nam omogućuje korištenje AJAX-a, CSS tranzicija i WebSocketsa bez korištenja JavaScripta. Korištenjem htmx-a na jednostavan način možemo obrađivati podatke i slati zahtjeve i primati odgovore navedenih tehnologija, sve unutar predloška bez ponovnog učitavanja web stranice. Korištenjem Djanga i htmx-a sam bez previše nepotrebnog koda kreirao dinamičku web aplikaciju koja sadrži sve moderne značajke poput registracije i prijave, kreiranje, uređivanje i brisanje objava i njihovo filtriraje i pretraživanje te detaljni pogledi svake od tih objava i profila svakog korisnika.

Ključne riječi

završni rad, informatika, programski okvir Django, Python, biblioteka htmx

Sadržaj

1. Uvod

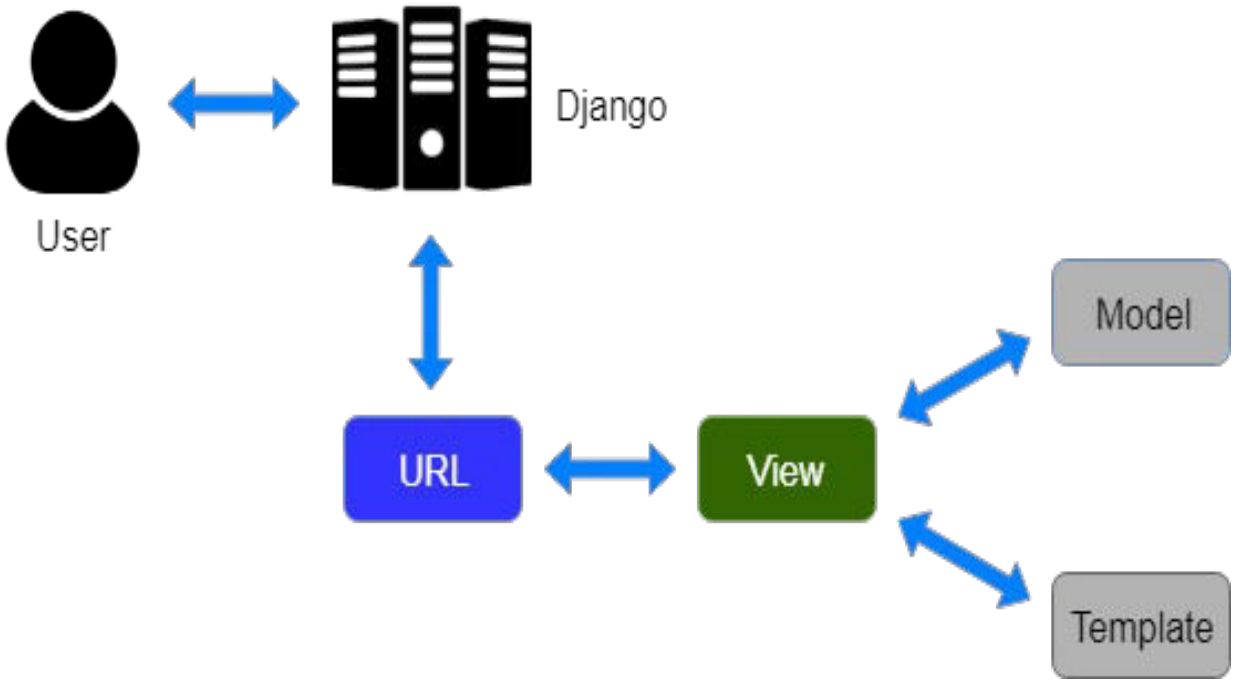
Dinamičke web aplikacije nisu oduvijek nudile korisničko iskustvo kao danas. Prethodnih godina ljudi su ih koristili isključivo kao izvor informacija; nije bilo važno kako izgledaju već da obavljaju određenu funkciju za koju su programirane. Razvojem tehnologije i modernizacijom doba sve više stranica bilo je potrebno prilagoditi modernom načinu života. Ove aplikacije postale su više od samog obavljanja zadataka, ljudi su ih počeli koristiti kao izvor zabave i razonode. Estetski su postale bolje te sada korisnicima pružaju više opcija kao što su komunikacija s prijateljima, objavljivanje slika, tekstualno opisivanje i mnoge druge. Važno je spomenuti dva pojma bez kojih ove aplikacije ne bi mogle raditi, a to su server i klijent. Server označava računalo zaduženo za primanje i slanje podataka dobivenih od klijenata, odnosno korisnika stranice.

Web aplikacija opisana u ovom završnom radu služi za prikazivanje simulacija izrađenih u alatu AnyLogic. Glavni korisnik aplikacije može umetnuti hipervezu sa stranice AnyLogic i tako kreirati vlastitu bazu podataka u kojoj će imati zapamćene sve željene simulacije. Također, korisnik može kreirati vlastiti korisnički račun u koji unosi osobne podatke (ime, prezime) i dodjeljuje si jedinstveno korisničko ime te lozinku. Još jedna značajka koju korisnici mogu provoditi jest označavanje modela sa "sviđa mi se" pritiskom na srce ispod spomenutog modela. Modele koji čine bazu podataka moguće je urediti ili posve ukloniti.

Sve spomenute značajke i navigiranje kroz aplikaciju obavlja se korištenjem biblioteke htmx. Ta biblioteka nam omogućuje slanje zahtjeva direktno iz HTML predloška koji potom šalje odgovor i bez ponovnog učitavanja stranice mijenja željeni dio predloška s odgovorom.

2. Programski okvir Django

Django je besplatan, web programski okvir otvorenog izvora (eng. Open Source) baziran na programskom jeziku Python. Django prati Model-Pogled-Predložak strukturu izrade web aplikacija. Model nam daje mogućnost jednostavne izrade baze podataka gdje sa samo par linija koda kreiramo tip entiteta i njegove atribute te veze između ostalih tipova entiteta. Pogled nam omogućuje da upravljamo podatcima iz Modela te ih prikazujemo na Predlošku. Isto tako preko Pogleda možemo dohvaćati podatke koje su upisani na Predlošcima npr. u Formama, te upravljamo tim podatcima kako bi izmijenili ili napravili nove entitete.[1]



Slika 1: Graf koji prikazuje tok Model-Pogled-predložak strukturu

(Slika 1) Korisnik šalje Djangu zahtjev za nekim resursom, te Django kao kontroler provjerava resurs u URL-u. Ako postoji takav URL, poziva se Pogled koji interaktira s Modelom i Predloškom te stvara predložak korisniku i šalje ga nazad korisniku kao odgovor.

2.1. Modeli

Django model je ugrađena značajka kojom izgrađujemo tablice, polja i njihove veze i ograničenja. Umjesto direktnog rada sa SQL-om kreiramo objekte sa njihovim atributima (eng. object properties) i metodama. Zadani adapter za bazu podataka u Djangu je SQLite. Model u Djangu je specijalni tip objekta koji se sprema u bazu podataka. Modeli se definiraju u datoteci *models.py* koja se nalazi u direktoriju aplikacije. Definiraju se kao klase koje najčešće nasljeđuju klasu django.db.models.Model no mogu i nasljeđivati klase poput django.contrib.auth.models.AbstractUser. Klasa AbstractUser se ponaša jednako kao i zadana User klasa ali kreiranjem modela koristeći klasu AbstractUser omogućujemo prilagođavanje modela po našoj potrebi. Ovdje je primjer modela *Profil. Profil* sadrži atribute *username* s vezom one to one na tablicu User gdje svaki korisnik(User) ima točno 1 Profil. Ostali atributi su *ImageField, CharField* i *DateField.[2]*

```
class Profil(models.Model):
    username = models.OneToOneField(User, default=None, null=True,
    unique=True, on_delete=models.CASCADE)
    image = models.ImageField(blank=False, null=False,
    default='images/profile/default.jpg', upload_to="images/profile/")
    opis = models.CharField(max_length=200, blank=True)
    datum = models.DateField(default=timezone.now)

    @receiver(post_save, sender=User)
    def create_username_profil(sender, instance, created, **kwargs):
        if created:
            Profil.objects.create(username=instance)

    @receiver(post_save, sender=User)
    def save_username_profil(sender, instance, **kwargs):
        instance.profil.save()

    def __str__(self):
        return "%s" % self.username
```

Slika 2: Model Profil s njegovim atributima i metodama

Metode *create_username_profil* i *save_username_profil* se poziva nakon što se izvrši *save()* metoda unutar pogleda *registration*, koji ćemo vidjeti kasnije, te provjerava je li User kreiran. Ako je, kreira se entitet kojem se polje *username* popunjava s prilogom koji je upisan u formi. Nakon toga se taj

profil sprema u bazu podataka. Posljednja metoda vraća string varijablu username svakog korisnika s kojom obilježavamo kako će se svaki profil prikazivati.

2.2. Pogledi

U model-pogled-kontroler arhitekturi, komponenta pogled je zadužena za predstavljanje podataka korisnicima. U programskom okviru Django pogledi su funkcije ili klase koje primaju network zahtjev te nakon toga vraćaju odgovor. Taj odgovor može biti HTTP odgovor, HTTP usmjeravanje ili neki HTML predložak, te kontekst iz pogleda. Pogledi sadrže logiku koja nam je potrebna da bismo procesirali podatke i dali korisniku odgovor. Logiku pogleda pišemo u datoteci *views.py.[3], [4]*

2.2.1. Pogledi bazirani na klasama

Pogledi bazirani na klasama nam omogućuju kreiranje pogleda nasljeđivanjem klasa, što čini proces puno lakšim jer trebamo pisati manje koda, ali mana toga je što u njima puno manje kod možemo prilagođavati specijalnim potrebama, te ga je puno teže čitati i razumjeti. Također korištenjem pogleda baziranih na klasama puno lakše nam je koristiti ga u pozivima za ostale poglede, te možemo koristiti razne predloške klase kao odgovore na HTTP zahtjeve.[4] U pogledima baziranima na klasama uvijek je potrebno navesti model na koji se pogled odnosi. Pet glavnih vrsta pogleda baziranih na klasama su:

- **ListView** – koriste se za generiranje liste svih entiteta navedenog modela iz baze podataka
- **DetailView** – koristi se za prikazivanje podataka pojedinačnog entiteta modela
- **CreateView** – kreira novi entitet u bazi podataka za specificirani model
- **UpdateView** – obavlja ažuriranje podataka za postojeći entitet
- **DeleteView** – obavlja brisanje entiteta iz baze podataka

```
class UpdatePostView(UpdateView):
    model = Post
    template_name = 'update_post.html'
    form_class = PostUpdateForm
```

Slika 3: Ažuriranje modela Post koristeći pogled baziran na klasi

2.2.2. Pogledi bazirani na funkcijama

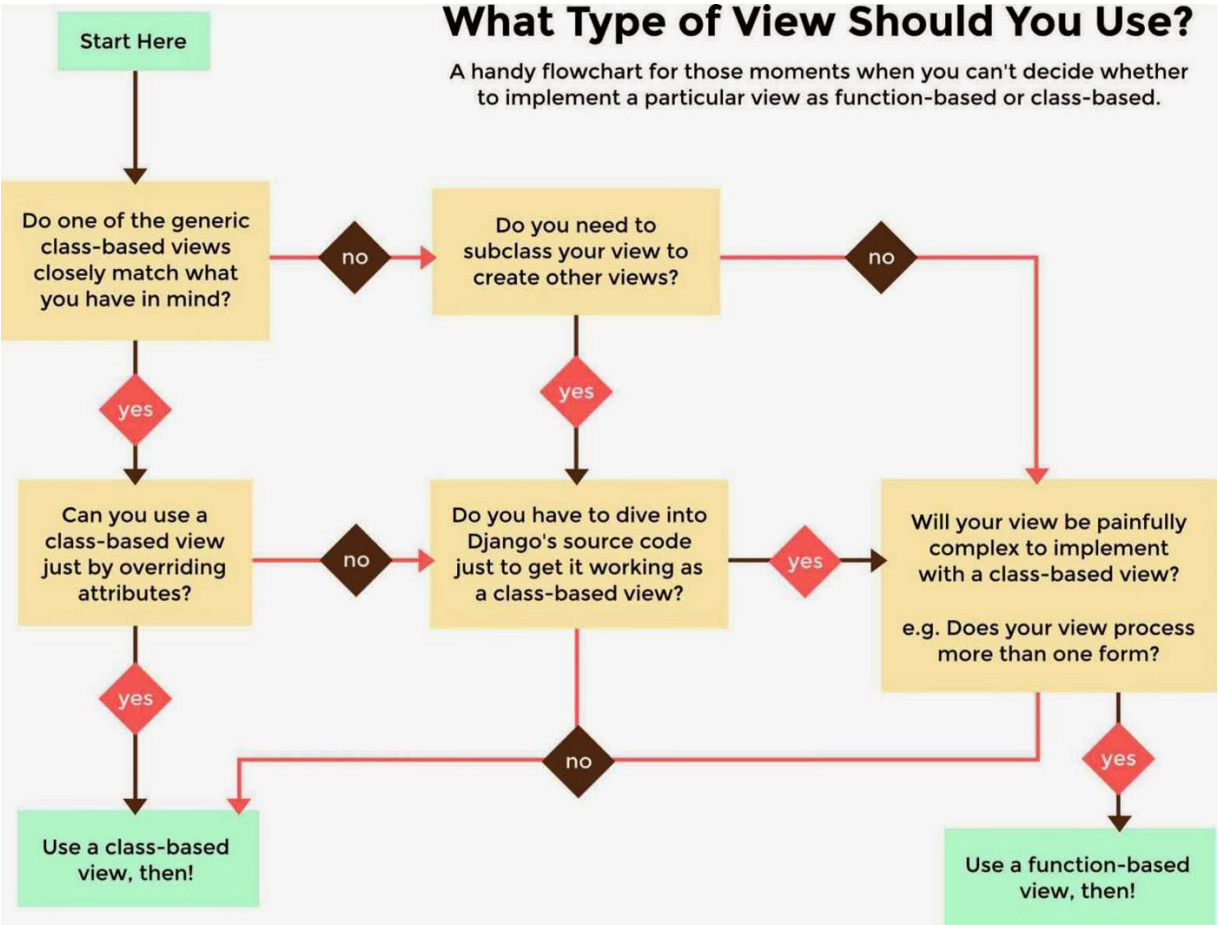
Pogledi bazirani na funkcijama su funkcije koje vraćaju HTTP odgovor nakon završetka logike funkcije. Pogledi bazirani na funkcijama su puno lakši za implementirati u razne specijalne slučajeve i puno ih je lakše razumjeti jer svaki dio funkcije pišemo sami, za razliku od pogleda baziranih na klasama gdje je veliki dio funkcije napisan u klasi koja se nasljeđuje. Mana ovih pogleda je redundantnost koda u većim projektima.[4]

```
def edit_post(request, pk):
    post = get_object_or_404(Post, pk=pk)
    form = PostForm(request.POST, request.FILES, instance=post)
    if request.method == "POST":
        videofile_path = post.videofile.path
        if form.is_valid():
            if os.path.exists(videofile_path):
                os.remove(videofile_path)
            form.save()
            return HttpResponse(headers={'HX-Trigger':
            'editPostSuccessful'})
    else:
        form = PostForm(instance=post)
    return render(request, 'add_post.html', {
        'form': form,
        'post': post,
    })
```

Slika 4: Ažuriranje modela Post koristeći pogled baziran na funkciji

2.2.3. Koje poglede koristiti?

Oba gore primjera služe za ažuriranje podataka modela *Post*. Kao što se može vidjeti, pogled baziran na klasi treba puno manje linija koda od pogleda baziranog na funkciji da bi izveli istu stvar. No ako imamo neki poseban slučaj, potrebno je koristiti pogled baziran na funkciji kako bi odredili logiku koju želimo da pogled izvodi. Pošto model *Post* sadrži polje *videofile* koji prima datoteku i sprema ju u direktorij specificiran u *models.py,* svaki put kada korisnik ažurira svoj post novi video se učitava i sprema u bazu podataka, a stari video ostane i dalje spremljen u bazi podataka. Tako nam se samo zauzima nepotreban prostor s datotekama koje se više ni ne koriste u aplikaciji. Prema tome u pogledu baziranom na funkciji možemo napraviti provjeru postoji li već datoteka koja pripada polju tog entiteta te ako postoji izbrišemo ju, nakon čega spremamo formu s novom datotekom. Oba tipa pogleda imaju svoje prednosti i mane, prema tome potrebno je odrediti prema potrebi koji tip pogleda nam najviše odgovara.[5]



Slika 5: Kako odrediti koji tip pogleda koristiti? [5]

2.3. Predlošci

Predlošci u Djangu su tekstualni dokumenti s ekstenzijom *.html* u *kojem* pišemo naš HTML, CSS i JavaScript. Predlošci sadrže svoj vlastiti jezik s kojim označavamo razne varijable, filtere i oznake kojima kontroliramo logiku predloška i činimo naše html dokumente dinamičnima.[6]

2.3.1. Varijable

Varijable ispisuju iznos upisan u kontekstu pogleda. Varijable se pišu unutar dvostrukih špicastih zagrada: **{{ varijabla }}**. Primjer varijable u predlošku iz pogleda *edit_post,* koji je prikazan prije, bio bi **{{ post.naslov }}**. Razlog tomu je što nam *post* u pogledu označava objekt koji smo dohvatili iz modela Post, te smo unutar kontekst deklarirali unutar jednostrukih navodnika string *'post'* kojim ćemo pozivati i ispisivati taj objekt i njegove atribute, a naslov označava polje tog objekta. Također s varijablama možemo koristiti filtere da bismo prilagodili prikaz podataka. Na primjer **{{ post.naslov|lower }}** će prikazati vrijednost polja *naslov* sve s malim slovima.[3], [6]

2.3.2. Oznake

Oznake se koriste za logiku u prikazivanju podataka na predlošku. S oznakama možemo ispisivati liste i polja listi ili raditi razne provjere te namještati kako da se predložak prikazuje ovisno o tim provjerama. Oznake se pišu unutar dvaju znakova za postotak koji se nalaze unutar špicastih zagrada: {% oznaka %}. Primjer oznake bio bi {% if post.naslov %} ... {% endif %}. U ovoj oznaci provjeravamo sadrži li objekt *post* uneseni *naslov* to jest je li polje *naslov* prazno ili ne. Ako nije prazno ulazimo u *if* naredbu i izvršava se kod unutar petlje.[3], [6]

3. Htmx

Htmx je biblioteka koja nam omogućuje dodavanje i pristup modernih značajki web preglednika, poput CSS tranzicija, AJAX-a, WebSockets-a i događaja sa strane servera direktno unutar HTML-a bez korištenja JavaScripta.[7], [8] Postoji mnogo htmx atributa s kojima možemo kontrolirati kako slati zahtjeve i kako primati i mijenjati DOM model. Dolje su navedeni najbitniji i najčešće korišteni atributi.

3.1. Kako započeti s korištenjem htmx-a

Postoji više načina na koji možete uključiti htmx u vaš projekt. Najbrži način je samo ga uključiti preko CDN (Content Delivery Network). Samo dodamo liniju u <head> oznaku i možemo započeti:

```
<script src="https://unpkg.com/htmx.org@1.7.0"></script>
```

Drugi način je da preuzmemo htmx.min.js datoteku sa unpkg.com i dodamo ju u direktorij projekta te tamo gdje je potrebno uključimo s oznakom:

```
<script src="/path/to/htmx.min.js"></script>
```

Također ga možemo instalirati putem npm:

```
npm install htmx.org
```

3.2. AJAX

Glavnina htmx-a je set atributa koji nam omogućuju da šaljemo AJAX zahtjeve direktno iz html-a.

Table 1: Tablica s atributima za AJAX

ATRIBUT	**OPIS**
hx-get	Šalje GET zahtjev za dani URL
hx-post	Šalje POST zahtjev za dani URL
hx-put	Šalje PUT zahtjev za dani URL
hx-patch	Šalje PATCH zahtjev za dani URL
hx-delete	Šalje DELETE zahtjev za dani URL

Svaki atribut šalje opisani AJAX zahtjev na specificirani url. Zahtjev se šalje svaki put kad je element pokrenut.

3.3. Načini pokretanja zahtjeva

Bez specificiranja načina pokretanja zahtjeva, svi elementi pokreću zahtjev na svoj zadani način, input na promjenu unosa, forma na podnošenje i ostali elementi na klik. No sa **hx-trigger** atributom možemo specificirati različite događaje koji će nam pokrenuti zahtjeve kao što su focus, keyup, mouseenter i drugi.[7], [9]

3.3.1. Modificiranje pokretanja

Sve naše događaje možemo modificirati tako da nam se pokrene zahtjev na malo drugačiji način.

once – izdaje zahtjev samo jednom

changed - izdaje zahtjev na promjenu vrijednosti elementa

delay:<vremenski interval> - zahtjev se izdaje nakon što prođe vremenski interval (npr. **500ms**), ako se zahtjev pokrene ponovo brojač se resetira

throttle:<vremenski interval> - zahtjev se izdaje nakon što prođe vremenski interval (npr. **500ms**), ako se zahtjev pokrene ponovo dok brojač još nije istekao, novi zahtjev se odbacuje

from:<CSS selector> - prima zahtjev od nekog drugog elementa

3.4. Ciljni element

Ako želimo da se odgovor na zahtjev učita u neki drugi element, a ne onaj koji je izdao zahtjev, onda to možemo učiniti s atributom **hx-target** u kojem specificiramo id tog elementa. Hx-target=”#leftDiv” označuje da je element s id leftDiv ciljni element u kojem će se zamjena obaviti. [7], [9]

3.5. Zamjena

Postoji više načina za obaviti zamjenu HTML-a iz odgovora u DOM model. Te načine specificiramo s atributom **hx-swap.**

innerHTML – objavljuje odgovor unutar ciljanog elementa, zadana

outerHTML – zamjenjuje cijeli ciljani element sa sadržajem iz odgovora

beforebegin – objavljuje sadržaj ispred ciljanog elementa u roditelju ciljanog elementa

afterbegin - objavljuje sadržaj ispred prvog djeteta unutar ciljanog elementa

beforeend - prima zahtjev od nekog drugog elementa

afterend - prima zahtjev od nekog drugog elementa

none - ne objavljuje sadržaj iz odgovora

3.5.1. Zamjena izvan okvira

Postoji također način zamjene elemenata izvan specificiranog ciljanog elementa. Atribut **hx-swap-oob** takozvani „Out of band swaps” kojeg dodijelimo elementu koji se nalazi izvan ciljanog elementa a kojeg također želimo da se zamjeni svaki put kad se dogodi hx zahtjev koji mijenja neki dio DOM modela.[7], [9]

4. Moja web aplikacija

Ova web aplikacija služi za prikazivanje simulacija izrađenih u alatu AnyLogic. Učitavanje simulacija u bazu podataka se radi tako da se u formu za učitavanje postova umetne hipervezu sa stranice AnyLogic te ih nakon toga prikazujemo na aplikaciji. Također, korisnicima je omogućeno kreiranje vlastitog korisničkog računa i unos osobnih podatka te korisničkog imena i lozinke. Svi registrirani korisnici mogu označiti pojedine modele simulacija s oznakom „sviđa mi se”. Pri vrhu stranice se nalazi navigacijska traka u kojoj unutar trake za pretraživanje možemo pretraživati različite modele simulacija prema njihovom imenu. S lijeve strane trake za pretraživanje nalazi se gumb koji vodi na glavnu stranicu aplikacije gdje se prikazuju sve simulacije učitane u bazi podataka. Također dok se korisnik nalazi na glavnoj stranici aplikacije uz gumb koji vodi na početnu stranicu nalazi se također gumb kojim se otvara skrivena traka s lijeve strane u kojoj su izlistane sve kategorije te broj modela koji se nalazi unutar svake kategorije. Klikom na neku od tih kategorija nam se unutar glavne stranice učitavaju modeli samo te kategorije i zamjenjuju se sa svim prikazanim modelima. S desne strane trake za pretraživanje nalazi se ikona profilne slike, koja dok korisnik nije registriran prikazuje klasičnu ikonu ne registriranog korisnika. Nakon što korisnik klikne na ikonu slike profila otvara se padajući izbornik u kojem se nalaze dodatne navigacijske funkcije te opcije da se korisnik prijavi ili registrira u aplikaciju, ili ako je već prijavljen da se odjavi iz aplikacije i opcija da prijavljeni korisnik posjeti stranicu svog profila.

4.1. Registracija i prijava

Za registraciju i prijavu u Djangu koristi se ugrađeni paket django.contrib.auth iz kojeg možemo uključiti authenticate i login koji su nam potrebni za registraciju i prijavu. Djano.contrib je modul s paketima koji su najbitniji za razvijanje web aplikacija, u kojima se nalaze gotove funkcije da bi olakšale i ubrzale izradu web aplikacija. U funkciji registracije prvo provjeravamo je li metoda zahtjeva POST. Ako nije, tada je metoda GET te nam se prikazuje SignUpForm. Ako je, onda prikazujemo istu tu formu u kojoj zahtijevamo metodu POST te nakon toga provjeravamo je li forma koja je poslana metodom POST valjana. Ako je onda dobavljamo upisane vrijednosti iz input polja s predloška te ih spremamo u varijable iz modela pomoću cleaned_data. Nakon toga spremamo korisnika u bazu podataka te ga provjeravamo autentičnost korisničkog imena i lozinke, i logiramo ga u našu aplikaciju, te kao odgovor na zahtjev vraćamo http odgovor pomoću kojeg javljamo htmx-u da pokrene svoj zahtjev nakon što dobije odgovor loginSuccessful s atributom hx-

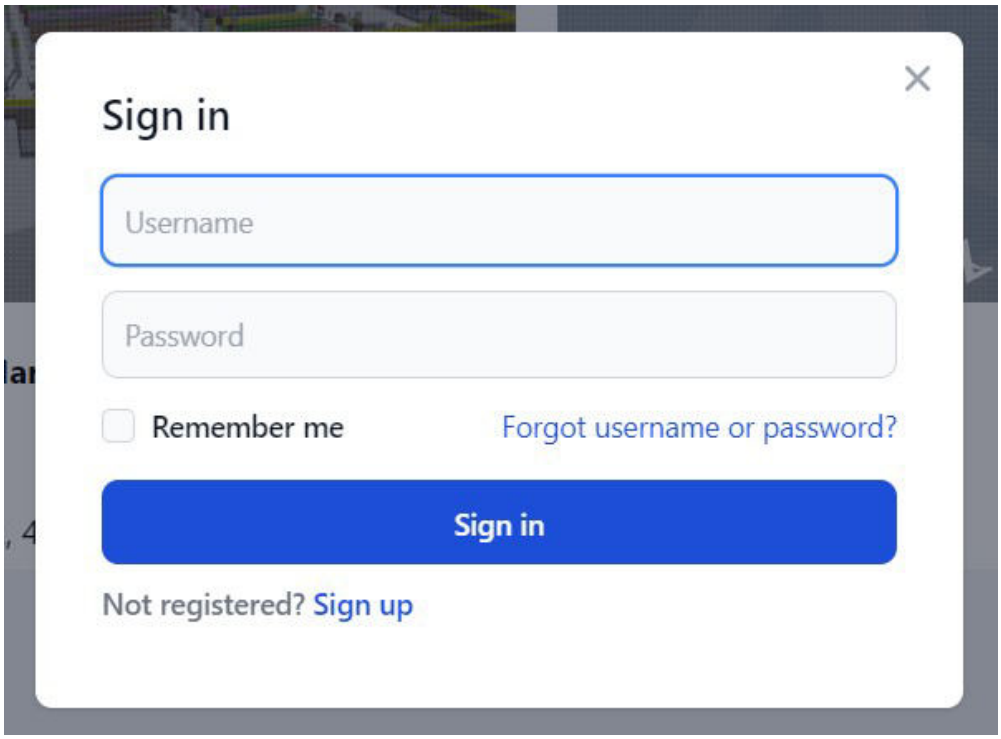
trigger=”loginSuccessful”. Funkcija login_user radi na sličan način samo što nema spremanja u bazu podataka nego dohvaćamo podatke iz forme i autenticiramo korisnika. Ako postoji korisnik s danim korisničkim imenom i lozinkom onda ga logiramo u aplikaciju inače javljamo grešku da su korisničko ime ili lozinka pogrešni.

```python
def registration(request):
    if request.method == 'POST':
        form = SignUpForm(request.POST)

        if form.is_valid():
            user = form.save()
            user.refresh_from_db()
            user.first_name = form.cleaned_data.get('first_name')
            user.last_name = form.cleaned_data.get('last_name')
            user.email = form.cleaned_data.get('email')
            user.save()
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password1')
            user = authenticate(username=username, password=password)
            login(request, user)
            return HttpResponse(headers={'HX-Trigger': 'loginSuccessful'})

    else:
        form = SignUpForm()

    context = {'form': form}
    context.update(csrf(request))

    return render(request, 'registration/register.html', context)
```

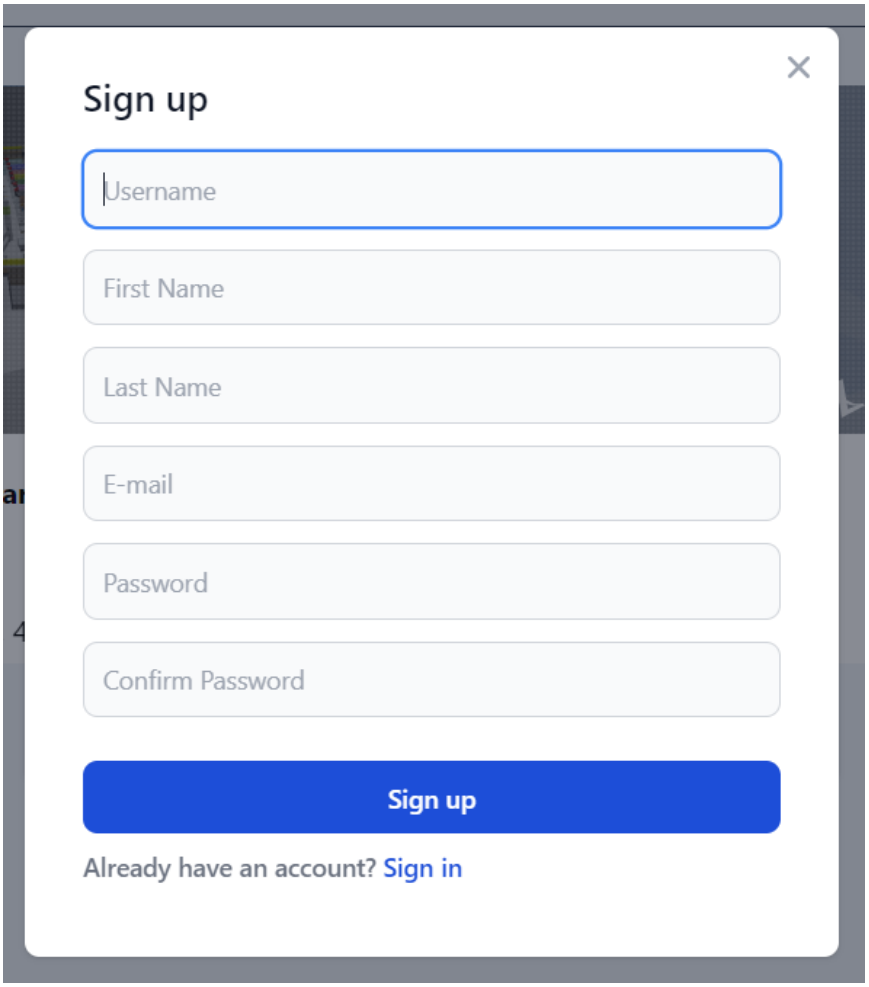
Slika 6: Funkcija registracije korisnika

```python
def login_user(request):
    if request.method == 'POST':
        form = LoginForm(request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            remember_me = form.cleaned_data['remember_me']
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                if not remember_me:
                    request.session.set_expiry(0)
                return HttpResponse(headers={'HX-Trigger': 'loginSuccessful'})

            else:
                messages.error(request, "Invalid username or password.")
        else:
            messages.error(request, "Invalid username or password.")
    else:
        form = LoginForm()
    context = {'form': form}
    context.update(csrf(request))
    form.fields['username'].widget.attrs['class'] = "my-3 bg-gray-50 border bor
    form.fields['username'].widget.attrs['placeholder'] = "Username"
    return render(request, "registration/login.html", context)
```

Slika 7: Funkcija prijave korisnika



Slika 8: Forma prijave u aplikaciju



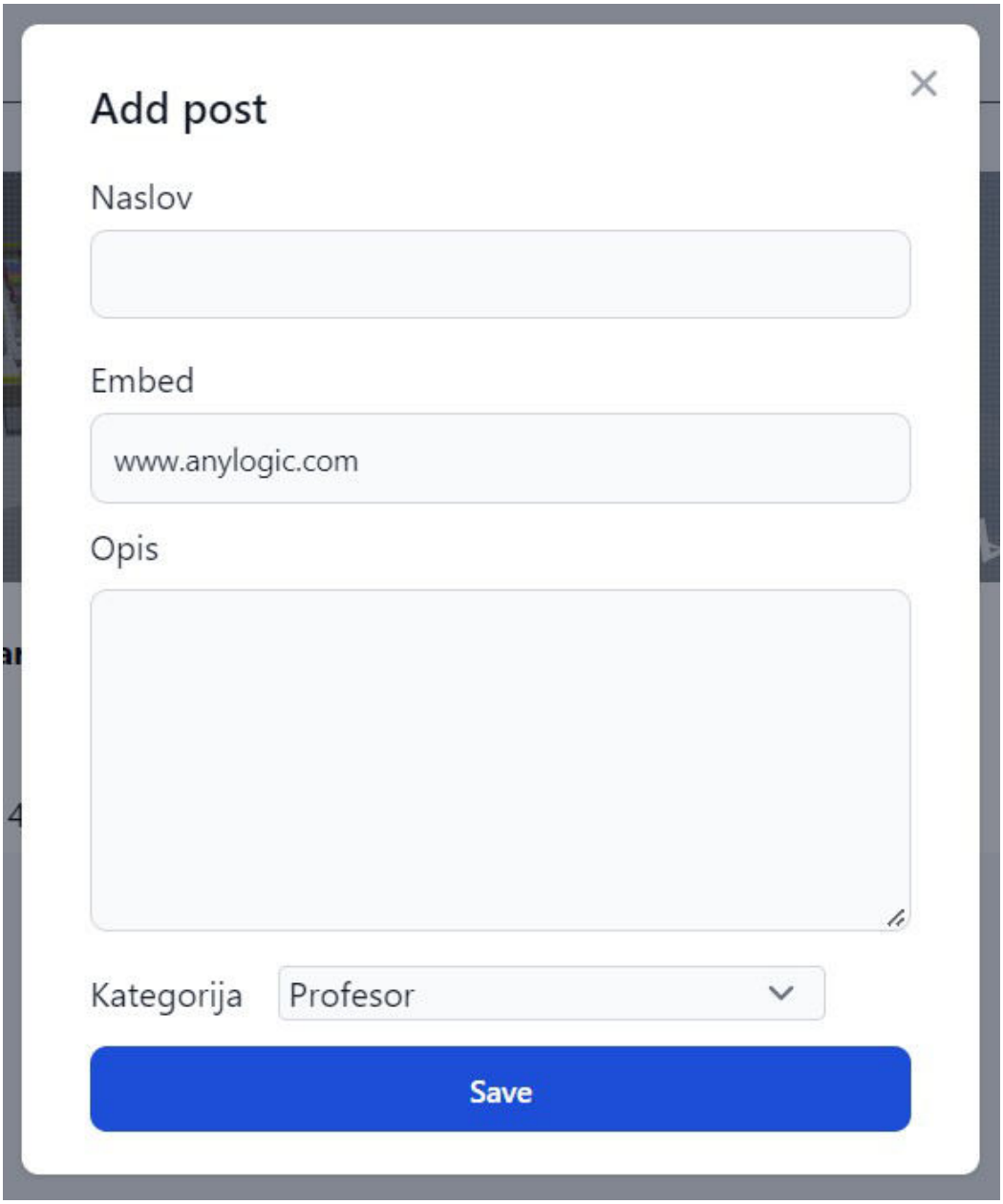
Slika 9: Forma za registraciju

4.2. Dodavanje, uređivanje i brisanje objava

U ovoj aplikaciji je omogućeno super korisnicima (eng. superuser) da nakon što kreiraju profil mogu dodavati objave u aplikaciji da bi ih mogli prikazati ostalim korisnicima. Također nakon što se kreira post moguće je urediti njegova polja ili ga izbrisati iz baze podataka. Za to nam služe tri funkcije add_post, edit_post i remove_post. Funkcija add_post prikazuje formu koju ispunimo podatcima za svako polje tablice osim za polje profil koje je tipa ForeignKey. To polje automatski popunjavamo s username korisnika koji je poslao zahtjev za funkcijom. Ako je forma pravilna kreira se novi post i sprema u bazu podataka. Funkcija edit_post pomoću naredbe get_ovject_or_404 dohvaća post za koji smo izdali zahtjev i onda nam otvara formu ispunjenu s instancom tog posta, te onda možemo urediti ta polja za taj post. U funkciji delete_post isto kao u funkciji edit_post dohvaćamo post preko njegovog primarnog ključa te nakon toga ga izbrišemo iz baze podataka pomoću post.delete(). Nakon toga usmjerimo korisnika na posts stranicu.

```
def add_post(request):
    if request.method == "POST":
        form = PostForm(request.POST, request.FILES,)
        username = request.user.id
        if form.is_valid():
            profil_form = form.save(commit=False)
            profil_form.profil = Profil.objects.get(username=username)
            profil_form.save()
            return HttpResponse(headers={'HX-Trigger': 'addPostSuccessful'})

    else:
        form = PostForm()
    context = {'form': form,}
    context.update(csrf(request))
    return render(request, 'add_post.html', context)

def edit_post(request, pk):
    post = get_object_or_404(Post, pk=pk)
    form = PostForm(request.POST, request.FILES, instance=post)
    if request.method == "POST":
        if form.is_valid():
            form.save()
            return HttpResponse(headers={'HX-Trigger': 'editPostSuccessful'})
    else:
        form = PostForm(instance=post)
    context = {'form': form, 'post': post,}
    context.update(csrf(request))
    return render(request, 'add_post.html', context)

@ require_POST
@method_decorator(csrf_exempt, name='dispatch')
def remove_post(request, pk):
    post = get_object_or_404(Post, pk=pk)
    post.delete()
    url = reverse_lazy('posts')
    return HttpResponseRedirect(url)
```

Slika 10: Funkcije dodavanja, uređivanja i brisanja objava objava



Slika 11: Forma za dodavanje/uređivanje objava

4.3. Prikaz i pretraživanje postova

Nakon što se objekt post doda u bazu podataka prikazuje se na stranici posts zajedno sa svim ostalim postovima u obrnuto kronološkom poretku. S lijeve strane nam se nalazi sekcija sa svim kategorijama pomoću koje možemo odabrati željenu kategoriju i prikazati samo postove koji su u toj kategoriji. Pomoću Django paketa Paginator uključenog iz django.core.paginator je postavljeno da se prikazuje samo 24 rezultata te se daljnji rezultati učitavaju pritiskom na gumb load more na dnu stranice. Uz postove po kategorijama imamo i filtriranje postova po profilu gdje se prikazuju svi postovi samo jednog odabranog profila.

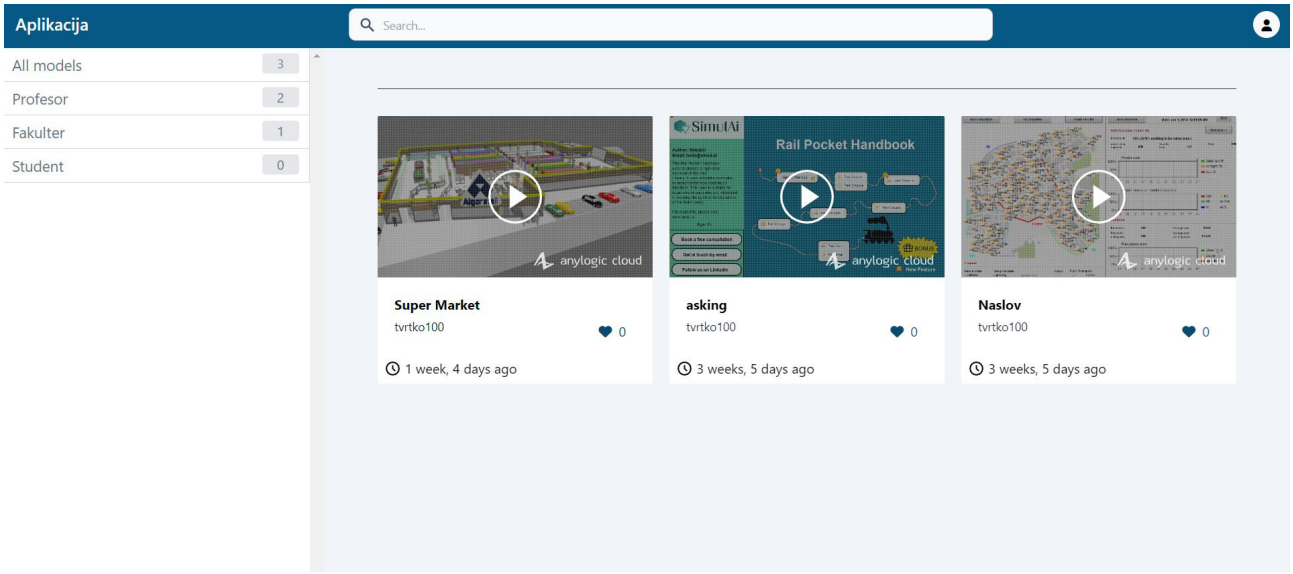
```
def post_list(request):
    users = User.objects.all()
    posts = Post.objects.all().order_by('-datum')
    paginator = Paginator(posts, 24)
    page_number = request.GET.get('page', 1)
    page_obj = paginator.get_page(page_number)
    posts_all = Post.objects.all()
    kategorijas = Kategorija.objects.all()
    count = Kategorija.objects.annotate(num_posts=models.Count('post'))

    context = {'posts':posts,  'page_obj':page_obj, 'count':count, 'users':users,
    context.update(csrf(request))
    return render(request, 'post_list.html', context)


def post_kategorija_list(request, kategorija):
    kategorijas = Kategorija.objects.all()
    posts_all = Post.objects.all()
    kategorijas_filter = Kategorija.objects.get(naziv=kategorija)
    posts = Post.objects.filter(kategorija=kategorijas_filter).order_by('-datum')
    paginator = Paginator(posts, 24)
    page_number = request.GET.get('page', 1)
    page_obj = paginator.get_page(page_number)
    count = Kategorija.objects.annotate(num_posts=models.Count('post'))

    context = {'page_obj':page_obj, 'count':count, 'kategorijas_filter':kategorija
    context.update(csrf(request))
    return render(request, 'post_list.html', context)

def post_list_by_profile(request, profil):
    profil_filter = Profil.objects.get(username=profil)
    posts = Post.objects.filter(profil=profil_filter).order_by('-datum')
    paginator = Paginator(posts, 24)
    page_number = request.GET.get('page', 1)
    page_obj = paginator.get_page(page_number)
    posts_all = Post.objects.all()
    kategorijas = Kategorija.objects.all()
    count = Kategorija.objects.annotate(num_posts=models.Count('post'))

    context = {'posts':posts,  'page_obj':page_obj, 'count':count, 'posts_all':pos
    context.update(csrf(request))
    return render(request, 'post_list.html', context)
```

Slika 12: Funkcije za prikaz svih objava te za prikaz objava filtriranih po kategorijama i po profilu

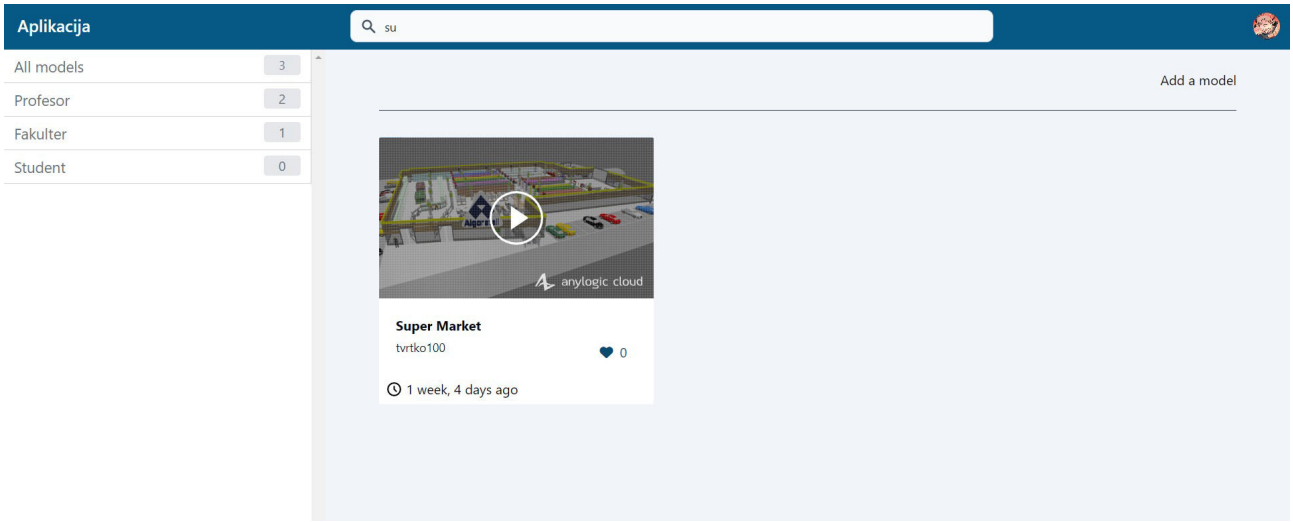
Pri vrhu stranice u navigacijskoj traci nam se nalazi traka za pretraživanje u kojoj možemo pretraživati na više načina. Dok se nalazimo na stranici *posts* pretražuju se svi postovi u bazi podataka. Dok se nalazimo na stranici pojedine kategorije pretražuju se i prikazuju samo postovi te kategorije. Dok se nalazimo na stranici postova pojedinih profila pretražuju se postovi samo tog profila.

```
def search_model(request):
    search_text = request.POST.get('search')
    posts = Post.objects.filter(naslov__icontains = search_text).order_by('-datum')
    paginator = Paginator(posts, 24)
    page_number = request.GET.get('page', 1)
    page_obj = paginator.get_page(page_number)
    context = {'posts':posts, 'page_obj':page_obj}
    return render(request, 'partials/search_results.html', context)

def search_model_category(request, kategorija):
    search_text = request.POST.get('search')
    kategorijas_filter = Kategorija.objects.get(naziv=kategorija)
    posts = Post.objects.filter(kategorija=kategorijas_filter).order_by('-datum')
    posts = posts.filter(naslov__icontains = search_text).order_by('-datum')
    paginator = Paginator(posts, 24)
    page_number = request.GET.get('page', 1)
    page_obj = paginator.get_page(page_number)
    context = {'posts':posts, 'page_obj':page_obj}
    return render(request, 'partials/search_results.html', context)

def search_model_profile(request, profil):
    search_text = request.POST.get('search')
    profil_filter = Profil.objects.get(username=profil)
    posts = Post.objects.filter(profil=profil_filter).order_by('-datum')
    posts = posts.filter(naslov__icontains = search_text).order_by('-datum')
    paginator = Paginator(posts, 24)
    page_number = request.GET.get('page', 1)
    page_obj = paginator.get_page(page_number)
    context = {'posts':posts, 'page_obj':page_obj}
    return render(request, 'partials/search_results.html', context)
```

Slika 13: Funkcije za pretraživanje svih objava te filtriranih objava



Slika 14: Prikaz objava na početnoj stranici uz filtriranje po kategorijama u prozorčiću s lijeve strane



Slika 15: Pretraživanje modela

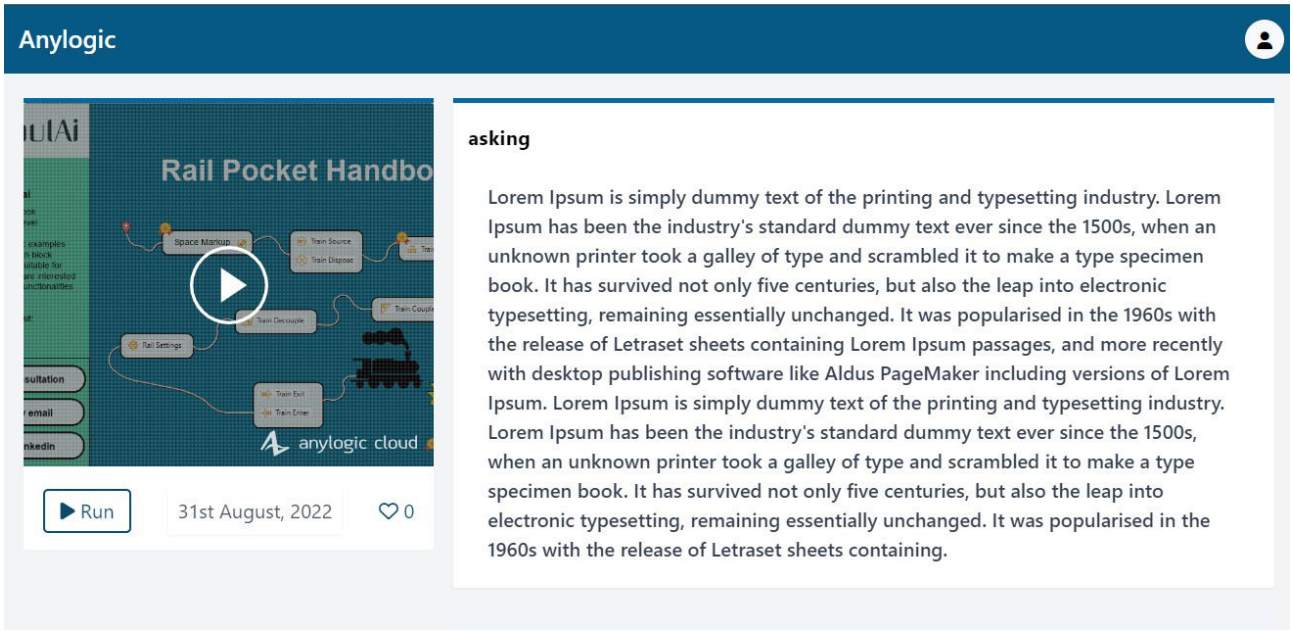
4.4. Detaljni pogled modela

Za svaki prikazani model postoji njegov vlastiti pogled s detaljima vezanima uz simulaciju kao što su ime autora, kategorija, datum objavljivanja modela i opis. Također unutar detaljnog pogleda nam

se nalazi funkcija s kojom za svaki model možemo pritiskom na gumb srca označiti da nam se simulacija sviđa. Uz njega se nalazi broj koji označava koliko je ukupno puta simulacija označena sa „sviđa mi se”. S lijeve strane gumba za označavanje modela sa „sviđa mi se” nalazi se gumb za pokretanje simulacije koji kada ga kliknemo učitava novi predložak u kojem se simulacija proteže preko cijelog ekrana.

```
class PostSettingsView(DetailView):
    model = Post
    template_name = 'post_settings.html'

    def get_context_data(self, *args, **kwargs):
        context = super(PostSettingsView, self).get_context_data()

        stuff = get_object_or_404(Post, id=self.kwargs['pk'])
        total_likes = stuff.total_likes()

        liked = False
        if stuff.likes.filter(id=self.request.user.id).exists():
            liked = True

        context["total_likes"] = total_likes
        context["liked"] = liked
        context.update(csrf(self.request))
        return context
```

Slika 16: Detaljni pogled modela



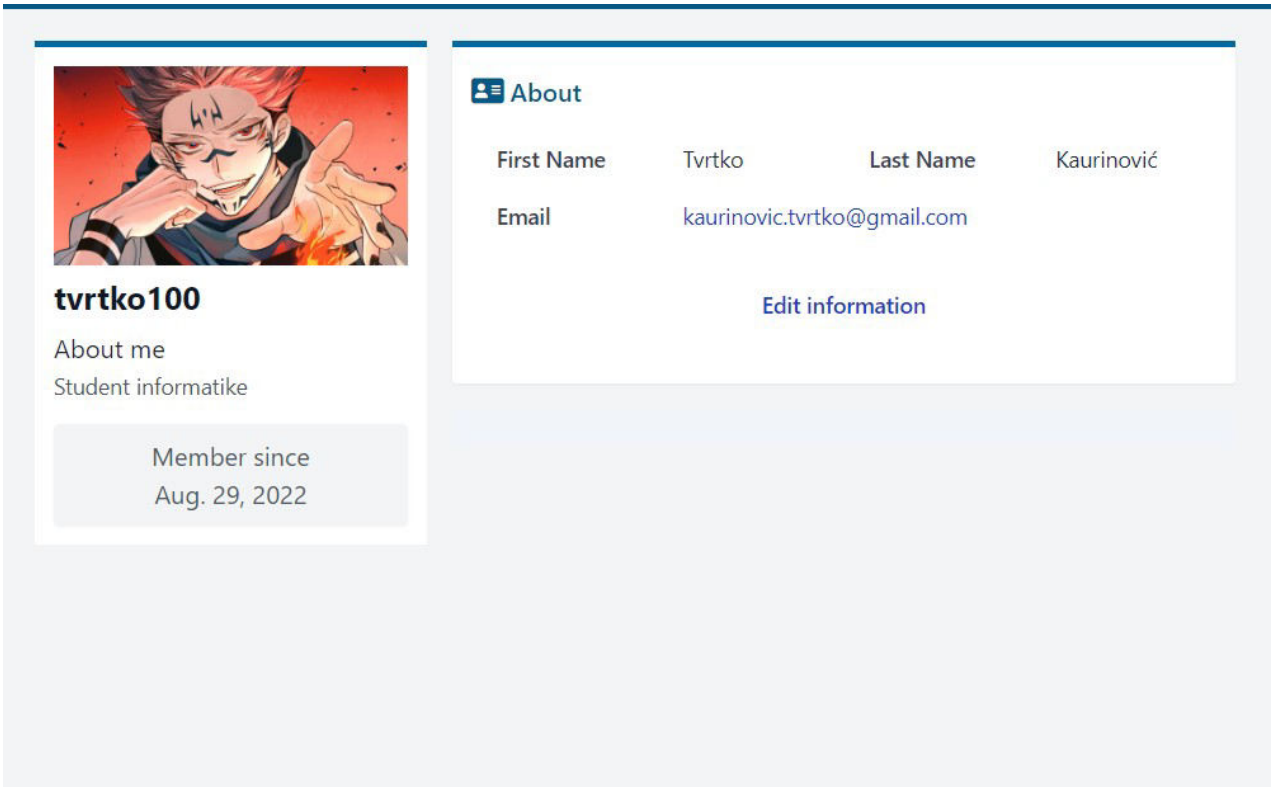
Slika 17: Predložak detaljnog pogleda modela

4.5. Detaljni pogled stranice profila

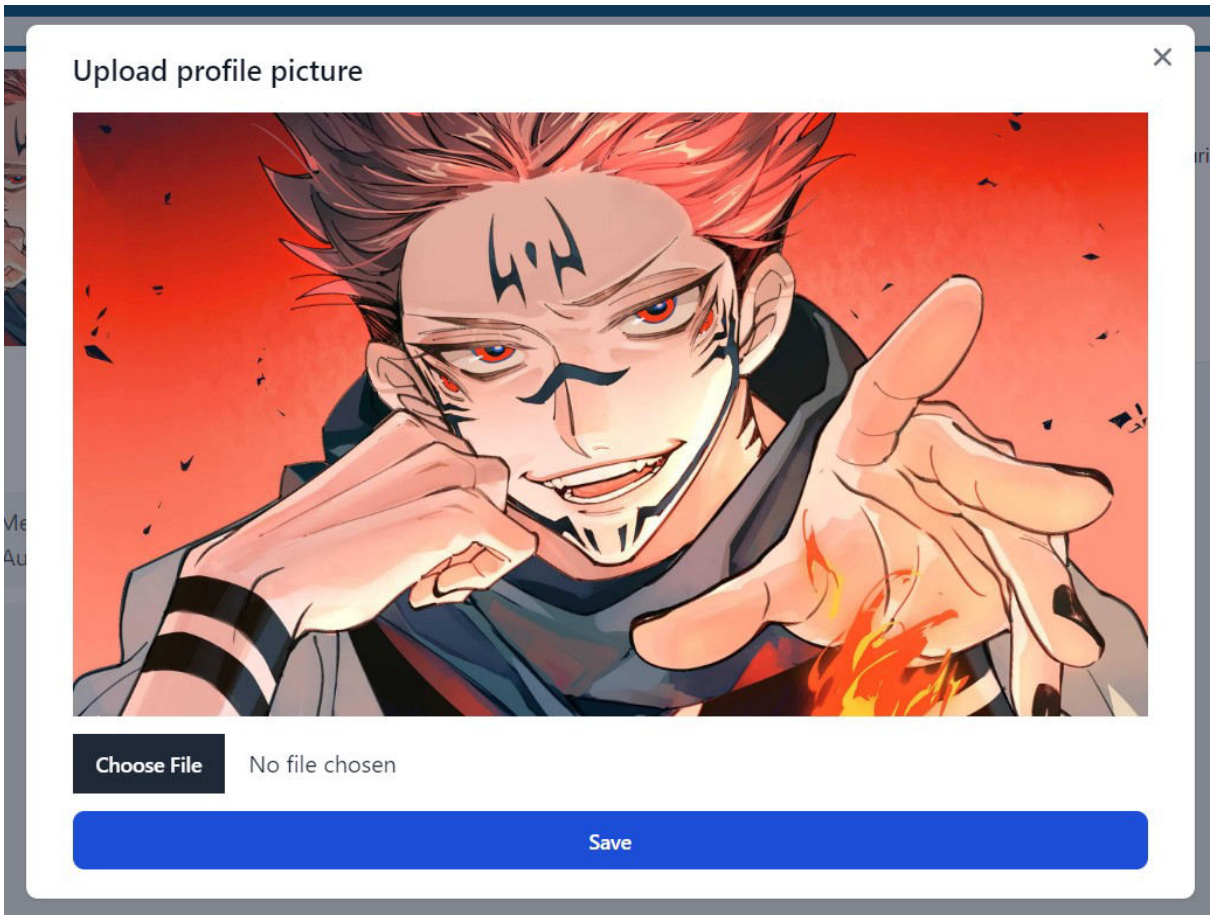
Svakom korisniku nakon registracije se kreira njegov vlastiti profil kao što je prije prikazano na slici modela Profil u odlomku 2.1. Korisnik nakon registracije i prijave može posjetiti stranicu svog profila klikom na ikonu profilne slike u navigacijskoj traci te u padajućem izborniku klikne na svoje korisničko ime. Također svaki korisnik može pogledati profil nekog drugog korisnika klikom na njegovo korisničko ime u sekciji *autor* na stranicama njegovih simulacija. Stranica profila sadrži informacije o korisniku kao što su njegova elektronička pošta, ime, prezime, datum kreiranja profila i kratki opis korisnika. Uz to svaki korisnik si može postaviti i mijenjati korisničku sliku koja se također promjeni i u navigacijskoj traci. Uz mijenjanje slike moguće je i mijenjanje svih navedenih osobnih informacija i korisničkog imena.

```python
def get_user_profile(request, id):
    user = User.objects.get(id=id)
    profil = Profil.objects.get(id=id)
    posts = Post.objects.filter(profil=profil).order_by('-datum')
    context = {'user':user, 'profil': profil, 'posts': posts}
    if request.method == 'GET':
        return render(request, 'profil_list.html', context)
    elif request.method == 'PUT':
        data = QueryDict(request.body).dict()
        user_form = UserInfoForm(data, instance=user)
        profile_form = ProfileInfoForm(data, instance=profil)
        if user_form.is_valid() and profile_form.is_valid():
            user_form.save()
            profile_form.save()
            return HttpResponse(headers={'HX-Trigger': 'editInfoSuccessful'})
        else:
            user_form = UserInfoForm(instance=user)
            profile_form = ProfileInfoForm(instance=profil)

    context = {'user_form':user_form, 'profile_form': profile_form, 'user':use
    context.update(csrf(request))
    return render(request, 'partials/edit_profile_detail.html', context)
```

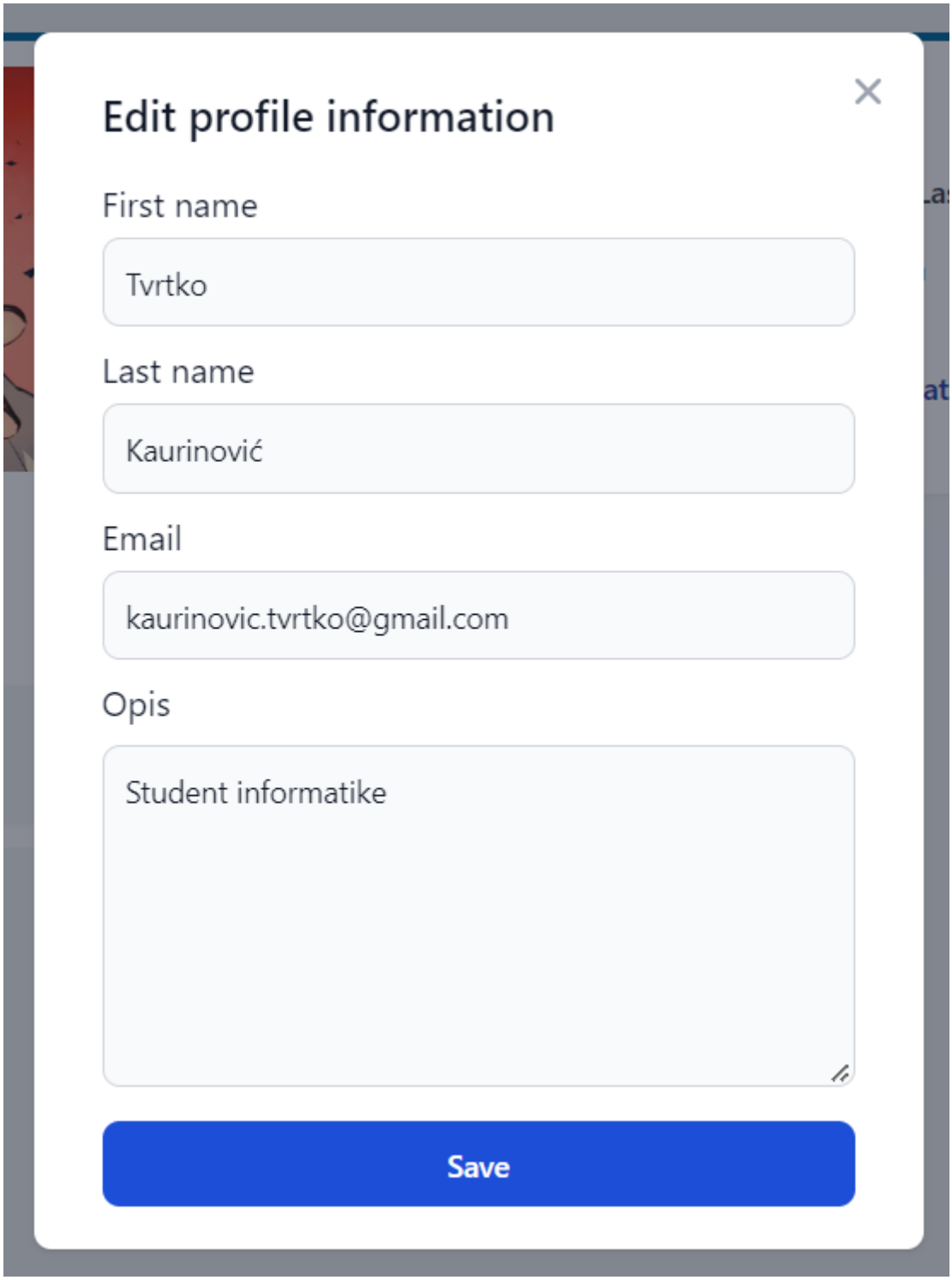
Slika 18: Detaljni pogled stranice profila



Slika 19: Predložak detaljnog pogleda stranice profila



Slika 20: Forma za mijenjanje slike profila



Slika 21: Forma za mijenjanje informacija profila

5. Zaključak

Modernizacijom tehnologije aplikacije su počele pružati korisnicima sve više funkcija i značajki. Različite značajke; od komentiranja objave korisnika do dodavanja osoba kojih poznajemo, omogućuju ljudima zabavniji način korištenja određenih aplikacija. Također, dostupno je više informacija i provjerenog sadržaja pogodnog za edukaciju i proširenje znanja. Korištenjem Djanga i htmx biblioteke na puno jednostavniji način mogu se kreirati web aplikacije bogatog korisničkog sučelja te ih razvijamo bez korištenja single page application okvira poput React, Angular i Vue. Naime, Django i htmx pružaju razne atribute uz pomoć kojih se mogu bez korištenja puno koda izvoditi kompleksne funkcije. Ovakve aplikacije ljudima omogućuju pohranu različitih podataka te njihovo uređivanje ili brisanje. Korištenjem kombinacije programa Django i biblioteke htmx moguće je napraviti kvalitetnu i modernu aplikaciju uz manju količinu potrošenog vremena.

Literatura

[1]V. Miletić, “Dinamičke web aplikacije 2 - GASERI – Group for apps and services on exascale research infrastructure.” https://gaseri.org/hr/nastava/kolegiji/DWA2/ (accessed Sep. 15, 2022).
[2] “Django models · HonKit.” https://tutorial.djangogirls.org/en/django_models/ (accessed Sep. 27, 2022).
[3]“Django documentation | Django documentation | Django.” https://docs.djangoproject.com/en/4.1/ (accessed Sep. 18, 2022).
[4]“Introduction to Django Views.” https://www.pluralsight.com/utilities/promo-only?noLaunch=true (accessed Sep. 27, 2022).
[5]S. Kumar, “Dja

Popis slika

Popis tablica