"Vrtić Connect" - Aplikacija za povezivanje roditelja i vrtića

Roža, Marko

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:954669

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-08**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Diplomski studij informatike – modul Informacijski i komunikacijski sustavi

Marko Roža

"Vrtić Connect" - Aplikacija za povezivanje roditelja i vrtića

Diplomski rad

Mentor: Doc. dr. sc. Lucia Načinović Prskalo

Rijeka, rujan 2023

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 25.4.2023.

Zadatak za diplomski rad

Pristupnik: Marko Roža

Naziv diplomskog rada: "Vrtić Connect" - Aplikacija za povezivanje roditelja i vrtića

Naziv diplomskog rada na eng. jeziku: "Vrtić Connect" - An application for connecting parents and kindergarten

Sadržaj zadatka: Zadatak diplomskog rada je izraditi aplikaciju za povezivanje roditelja s vrtićem koja će pružati funkcionalnosti slanja obavijesti i razmjenu trenutnih poruka. U radu će detaljno biti opisane sve tehnologije koje će se koristiti za izradu aplikacije te sve funkcionalnosti koje aplikacija pruža.

Mentorica:

Doc. dr. sc. Lucia Načinović Prskalo

Voditeljica za diplomske radove:

Prof. dr. sc. Ana Meštrović

Zadatak preuzet: 25.4.2023.

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: **+385(0)51 584 700**
E-mail: **ured@inf.uniri.hr**
OIB: **64218323816**
IBAN: **HR1524020061400006966**

Sažetak

U ovom diplomskom radu opisan je tijek razvoja web aplikacije za povezivanje roditelja i vrtića „Vrtić connect“. Aplikacija je razvijena u Django razvojnom okviru uz upotrebu HTMX biblioteke i AWS usluga te je postavljena na Heroku platformu. U sklopu rada objašnjene su i sve korištene tehnologije.

Ključne riječi: Django**,** HTMX, Heroku, Python, web aplikacija.

Sadržaj

1. Uvod

Razvojem interneta i World Wide Weba (WWW) te sve većim utjecajem digitalizacije i tehnologije na svaki aspekt života promijenjeni su i načini komunikacije ljudi. Omogućeni su različiti načini direktne ili grupne komunikacije bez obzira na udaljenost (video pozivi, chat aplikacije, forumi, društvene mreže i sl.). Primjenom ovih mogućnosti i tehnologija promijenjen je i način komunikacije roditelja, odgojitelja i predškolskih ustanova. Uz komunikaciju uživo roditelja i odgojitelja danas većina odgojnih skupina koristi i aplikacije poput Vibera ili WhatsAppa za dogovaranje i dijeljenje bitnih informacija te fotografija. Uz prednosti koje su navedeni načini komunikacije donijeli došlo je i do neželjenih posljedica. Određene bitne obavijesti zanemarene su od strane roditelja zbog velikog broja poruka te su odgojitelji izloženi kontaktiranju roditelja i izvan radnog vremena.

U ovom radu bit će prikazan tijek razvoja web aplikacije „Vrtić connect“ kojoj je cilj olakšati komunikaciju roditelja i odgojitelja u predškolskim odgojnim skupinama. Svrha aplikacije je zamijeniti ostale načine komunikacije i pružiti roditeljima i odgojiteljima jedinstvenu platformu koja će sadržavati sve informacije vezane uz odgojnu skupinu. Sastojat će se od odjeljka za obavijesti, chat grupe, prikaza nadolazećih aktivnosti i odjeljka s fotografijama skupine. S obzirom na odgojnu skupinu, roditeljima će biti prikazan različit sadržaj koji će odgojitelji moći uređivati. Odgojiteljima će biti omogućeno dodavanje i uređivanje članova skupine, obavijesti, aktivnosti i fotografija.

Web aplikacija bit će razvijena u razvojnom okviru Django te će koristiti bazu podataka PostgreSQL i usluge u oblaku Amazon RDS, Amazon S3 i Heroku. U drugom poglavlju opisane su korištene tehnologije. Treće poglavlje opisuje početno postavljanje projekta i aplikacije dok su u četvrtom poglavlju definirani modeli podataka aplikacije. Peto poglavlje opisuje izradu pogleda i predložaka a u šestom poglavlju je objašnjeno povezivanje aplikacije s AWS uslugama Amazon RDS i Amazon S3. U sedmom poglavlju opisan je tijek postavljanja aplikacije na Heroku platformu i na kraju je dan zaključak rada.

2. Korištene tehnologije

2.1. Django

Django je razvojni okvir otvorenog koda temeljen na programskom jeziku Python. Početno je razvijen za kreiranje i održavanje web stranica novina Lawrence Journal-World dok je njegova prva inačica kao razvojni okvir otvorenog koda izdana 2005. godine pod BSD licencom (Django introduction 2023) (Django (web framework) n.d.). Neke od glavnih značajki ovog razvojnog okvira su:

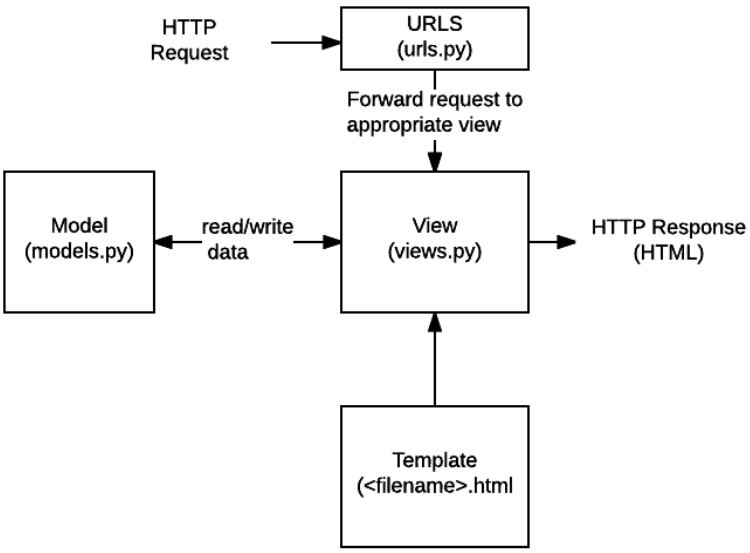
- Omogućavanje brzog razvoja web aplikacije
- Veliki broj paketa za rješavanje uobičajenih zadataka u web programiranju
- Osiguravanje visoke razine sigurnosti i gotov sustav za autentifikaciju
- Brzo i fleksibilno skaliranje
- Mogućnost korištenja za različite vrste web aplikacija poput sustava za upravljanje sadržajem i društvenih mreža (Why Django? 2023)

Kako Django razvojni okvir omogućava brz razvoj web aplikacije može se objasniti kroz glavna načela kojih se on pridržava: „labav spoj i jaka kohezija“ komponenti, što manje programskog koda i izbjegavanje ponavljanja (eng. Don't repeat yourself, DRY) (Design philosophies 2023). Brz razvoj aplikacija omogućava i veliki broj paketa i funkcija koje razvojni okvir, zajedno s paketom `contrib`, pruža korisnicima bez potrebe za instaliranjem dodataka. Neki od značajnijih paketa su: web poslužitelj za testiranje i razvoj, sustav za provjeru valjanosti i serijalizaciju obrazaca, sustav za korištenje predmemoriranja (eng. caching), sustav za autentifikaciju i sučelje za administraciju aplikacije (Django (web framework) n.d.). Upotrebom ovih funkcija i sustava značajno se smanjuje vrijeme potrebno za razvoj i testiranje web aplikacije. Osim sustava za autentifikaciju, Django, sadrži i različite mjere za zaštitu od najčešćih vrsta napada poput CSRF napada, napada skriptama (eng. Cross site scripting) i SQL injekcija (Django introduction 2023). Brzo i fleksibilno skaliranje aplikacija te njihovo održavanje omogućuje „shared-nothing“ Model-Pogled-Predložak (eng. Model-View-Template, MTV) arhitektura ovog razvojnog okvira (Django introduction 2023). Prema podacima web stranice webtechsurvey iz kolovoza 2023. godine 85,777 web stranica i web aplikacija koristi Django a neke od najpoznatijih su Mozilla, Instagram, Pinterest, Spotify, National Geographic i Disqus (webtechsurvey - Django 2023) (Brewster 2021).

2.1.1. Model-Predložak-Pogled arhitektura u Djangu

Arhitektura Djanga temelji se na Model-Pogled-Kontroler (eng. Model-View-Controller, MVC) arhitekturi koja se u dokumentaciji naziva Model-Predložak-Pogled gdje Predložak zamjenjuje Pogled a Pogled zamjenjuje Kontroler (FAQ: General 2023). Ovakva vrsta arhitekture osigurava podjelu logike i prikaza softvera primjenom tri povezane komponente: korisničkog sučelja, podataka i upravljačke logike (MVC 2023). U slučaju Djangove interpretacije MVC arhitekture Model označava podatke, Pogled definira koji podaci će se prikazati a Predložak određuje kako će podaci biti prikazani. Ulogu Kontrolera iz originalne arhitekture (usmjeravanje naredba Modelu i Pogledu) preuzima sam razvojni okvir (FAQ: General 2023) (MVC 2023).

Kada aplikacija rađena u Djangu i utemeljena na MTV arhitekturi primi HTTP zahtjev prosljeđuje ga URL maperu (eng. mapper) u datoteci `urls.py`. Datoteka `urls.py` sadrži listu `path()` i `re_path()` funkcija gdje prvi argument označava URL putanju, drugi argument označava odgovarajuću funkciju pogleda (eng. view) a treći argument može označavati ime korišteno umjesto URL uzorka. Korištenjem URL mapera omogućena je izrada upotrebljivih i „čistih“ URL putanja koje mogu sadržavati i parametre potrebne aplikaciji. Uloga mapera je usporediti URL primljenog zahtjeva s uzorcima u listi, ako maper pronađe URL uzorak koji se podudara s primljenim zahtjevom uvozi i poziva odgovarajuću funkciju pogleda te prosljeđuje zahtjev i dodatne argumente (URL Dispatcher 2023).



Slika 1. Obrada HTTP zahtjeva u Djangu preuzeto s: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

Pogledi se u pravilu nalaze u `views.py` datoteci i mogu biti Python funkcije ili klase. Zadatak pogleda je obraditi zaprimljen zahtjev i vratiti odgovor s odgovarajućim podacima u određenom obliku. Pritom pogled pristupa podacima iz modela te prosljeđuje ih predlošku koji ih prikazuje na definirani način (Django introduction 2023). Modeli su Python objekti u kojima se pomoću objektno relacijskog preslikavanja (eng. object relational mapping, ORM) svaki model preslikava u jednu tablicu baze podataka i svaki atribut modela predstavlja jedno polje iz tablice. Modeli su definirani u datoteci `models.py` i pojednostavljuju komunikaciju između aplikacije i baze podataka kojoj se pristupa pomoću automatski generiranog API-ja, zamjenjujući korištenje SQL naredbi (Django introduction 2023) (Jandhyala 2022) (Django Documentation - Models 2023). Nakon što je pogled dohvatio željene podatke prosljeđuje ih predlošku u obliku konteksta kako bi se generirao HTTP odgovor na zahtjev. U razvojnom okviru Django predlošci sadrže strukturu HTML dokumenta (ali mogu biti i drugog formata) koji će s dohvaćenim podacima dinamički proizvesti odgovor na zahtjev. Osim statičke strukture dokumenta predlošci sadrže i određene naredbe koje će sustavu za predloške (eng. template engine) opisati kako da se prikažu i koriste podaci proslijeđeni u kontekstu. Naredbe mogu biti pisane za Djangov sustav za predloške koji koristi jezik Django template language (DTL) ili za vanjski sustav za predloške poput sustava Jinja2. Primjenom ovih sustava za predloške mogu se koristiti razne uvjetne naredbe (poput if uvjeta i for petlja), filtriranja i druge pogodnosti kako bi se olakšalo i ubrzalo generiranje sadržaja (Django Documentation - Templates 2023) (Django introduction 2023).

2.2. PostgreSQL

PostgreSQL je sustav za upravljanje bazama podataka otvorenog koda koji je nastao 1986. godine u sklopu projekta POSTGRES Sveučilišta Kalifornije u Berkeleyju. Pridržava se ACID principa kod izvođenja transakcija i smatra se jednim od najpopularnijih sustava za upravljanje bazama podataka zbog svoje dokazane arhitekture, osiguravanja integriteta podataka, pouzdanosti, velikog broja značajki te detaljne i ažurne dokumentacije. Može se izvoditi na svim popularnijim operacijskim sustavima te osim kao baze podataka opće namjene može se koristiti i u LAPP (Linux, Apache, PostgreSQL i PHP, Python ili Perl) stogu, kao geoprostorna baza podataka (koristeći PostGIS ekstenziju) te kao središte sustava različitih baza podataka (zbog podržavanja SQL upita i JSON upita) (PostgreSQL - About 2023) (AWS - What is

PostgreSQL? 2023) (What is PostgreSQL? 2023). Za korištenje PostgreSQL sustava s Python programskim jezikom potrebno je koristiti adapter poput modula psycopg2 koji je trenutačno najpopularniji (psycopg2 - Python-PostgreSQL Database Adapter 2021).

2.3. HTMX

HTMX je JavaScript biblioteka koja omogućava izvođenje AJAX zahtjeva, pokretanje CSS tranzicija, implementiranje Web utičnica (eng. Websockets) i događaja poslanih od strane poslužitelja pomoću HTML elemenata (</> htmx n.d.). Svrha ove biblioteke je kombinirati vrline „Single page“ i „multi page“ aplikacija.

„Single page“ aplikacije funkcioniraju tako što samo jednom učitavaju web stranicu s poslužitelja a dijelovi sadržaja tijela stranice ažuriraju se kada je potrebno čime se korisnicima nudi bolje i modernije iskustvo bez kompletnog učitavanja web stranice svaki put kada korisnik klikne na određeni dio stranice. Za ažuriranje dijelova koriste se JavaScript API-evi preko kojih se šalju zahtjevi i zaprimaju podaci (često u JSON formatu). Navedene aplikacije često zahtijevaju značajnu obradu podataka na klijentskoj strani (za obradu podataka primljenih preko API-a) i učitavanje velikih JavaScript paketa pri prvom otvaranju web aplikacije. Suprotno „single page“ aplikacijama, „multi page“ aplikacije ponovno učitavaju cijelu web stranicu pri svakoj interakciji korisnika s njom. Ovakvim pristupom podaci se obrađuju samo na poslužiteljskoj strani (čime je osigurana veća razina sigurnosti) i ne koriste se veliki i kompleksni JavaScript paketi ali je korisničko iskustvo nespretno i lošije. Primjenom HTMX biblioteke nastoji se osigurati korisničko iskustvo „single page“ aplikacija s jednostavnošću „multi page“ aplikacije, smanjujući kompleksnost na klijentskoj strani i pretjerano oslanjanje na JavaScript u web pregledniku te istovremeno pružajući moderno, responzivno i ugodno korisničko iskustvo. HTMX biblioteka manja je od 10 KB i može se preuzeti ili učitati preko mreže za isporuku sadržaja (eng. content delivery network, CDN) (Hibbard 2023) (What Is HTMX n.d.) (SPA (Single-page application) 2023) (Asaolu 2021).

HTMX biblioteka sadrži atribute (hx-get, hx-put, hx-patch, hx-delete i hx-post) kojima je omogućeno direktno slanje AJAX zahtjeva iz HTML elementa. Slanje zahtjeva može se pokrenuti i primjenom vremenskog intervala za slanje ili kod određenog događaja (primjenom atributa hx-trigger). Bitno je napomenuti i atribut hx-swap kojim se definira element koji će biti

zamijenjen s pristiglim odgovorom na zahtjev. Navedenim atributom može se zamijeniti sadržaj unutar elementa, sami element te razne druge opcije. Za korištenje web utičnica i događaja poslanih od strane poslužitelja dostupna je probna podrška te će od verzije HTMX 2.0 ove mogućnosti biti dostupne samo kao ekstenzije na HTMX biblioteku (</> htmx n.d.).

```
<div hx-get="/update_time/">
  <p>The current time is: {{ current_time }}</p>
  <button type="button" hx-swap=".time">Update time</button>
</div>
```

Programski kod 1. Primjer slanja GET zahtjeva pomoću HTMX-a preuzeto s: https://vegibit.com/what-is-htmx/#some-htmx-examples

2.4. Amazon Web Services (AWS)

Amazon Web Services (AWS) je platforma za računalstvo u oblaku. Pokrenuta je 2002. godine kao interno rješenje za potrebe online trgovine Amazon.com dok je 2006. godine počela nuditi proizvode tipa infrastruktura-kao-usluga (eng. infrastructure-as-a-service, IaaS). Danas je AWS platforma koja nudi više od 200 skalabilnih rješenja i usluga koje su tipa infrastruktura-kao-usluga, platforma-kao-usluga (eng. platform-as-a-service, PaaS) i softver-kao-usluga (eng. software-as-a-service, SaaS). Usluge AWS platforme mogu se koristiti u različite svrhe: za pohranu podataka, upravljanje relacijskim i nerelacijskim bazama podataka, strojno učenje, primjenu umjetne inteligencije, upravljanje i analizu podataka velikog obujma, usluge poslužitelja, umrežavanje i u razne druge namjene. Platforma implementira plaćanje po potrošnji (eng. „pay as you go“) uz razne besplatne pogodnosti za nove korisnike (oko 60 usluga) i svoje usluge nudi u više od 190 zemalja (Amazon web services 2023) (Barney 2022) (Cloud computing with AWS 2023) (Jha 2023). Od mnogobrojnih usluga koje AWS nudi najpopularnije su Amazon EC2 (Elastic Cloud Compute), Amazon RDS (Relational Database Services), Amazon S3 (Simple Storage Service) Amazon IAM (Identity and Access Management) i Amazon EBS (Elastic Block Store) (Top 25 AWS Services List 2023 2023).

Amazon EC2 (Elastic Cloud Compute) je usluga za računalstvo u oblaku koja se najčešće koristi za: pokretanje poslovnih aplikacija (eng. enterprise applications) i aplikacija dizajniranih za računalstvo u oblaku (eng. cloud-native applications), skaliranje aplikacija za računalstvo visokih performansi (eng. high performance computing applications, HPC), razvoj softvera za Apple platforme i treniranje i implementaciju aplikacija strojnog učenja. Amazon EC2 nudi

korisnicima više od 600 različitih instanci virtualnih poslužitelja uz odabir procesora, pohrane, umrežavanja i operacijskog sustava. Navedena usluga nudi i mogućnost skaliranja resursa s obzirom na trenutačne potrebe te povezivanje usluge s drugim AWS uslugama poput Amazon RDS usluge (EC2 2023) (What is Amazon EC2 2023).

Amazon RDS je zbirka usluga kojima se pojednostavljuje postavljanje, rad i skaliranje relacijskih baza podataka u oblaku. Podržava PostgreSQL, MySQL, MariaDB, Oracle i SQL server tipove baza podataka te nudi i Aurora bazu podataka koju je razvio sam Amazon. Ova usluga osigurava korisnicima visoku dostupnost baze podataka te sama upravlja sigurnosnim kopijama, oporavkom baze podataka, nadogradnjom softvera i otkrivanjem kvarova (Amazon Relational Database Service 2023) (What is Amazon Relational Database Service (Amazon RDS)? 2023).

Uz navedene usluge bitno je istaknuti i Amazon S3 uslugu. Amazon Simple Storage Service (Amazon S3) usluga je za pohranu objekata u oblaku koja osigurava visoku razinu sigurnosti, dostupnost podataka, skalabilnost te visoke performanse kod pohrane objekata. Objekti se spremaju u spremnike (eng. buckets) slične direktorijima u datotečnom sustavu i svaki objekt u spremniku sastoji se od sadržaja, jedinstvenog identifikatora i meta podataka. Amazon S3 usluga nudi različite klase spremnika s obzirom na učestalost dohvaćanja spremljenih objekata i razne opcije i dopuštenja za dohvaćanje spremljenih objekata (Amazon S3 2023) (Yasar 2023) (What is Amazon S3? 2023).

2.5. Heroku

Heroku je platforma u oblaku (platforma-kao-servis) koja nudi usluge implementacije, skaliranja i održavanja web aplikacija u oblaku. U potpunosti upravlja održavanjem infrastrukture, hardvera i poslužitelja te podržava pokretanje web aplikacija pisanih u programskim jezicima Python, Ruby, PHP, Java, Scala, Node.js, Go i Clojure (How Heroku Works 2023). Za pokretanje i upravljanje aplikacijama Heroku primjenjuje pametne kontejnere nazvane Dyno. Dyno kontejneri su izolirani i virtualni Linux kontejneri koji pohranjuju i upravljaju web aplikacijom (Heroku Dynos 2023). Postoje tri vrste Dyno kontejnera: „web“, „worker“ i „one-off“. „Web“ kontejneri primaju HTTP promet i koriste se za obradu web procesa aplikacije. „Worker“ kontejneri se u pravilu koriste za pozadinske procese, procese koji

se izvode u vremenskim intervalima te za sustave čekanja dok se „one-off“ kontejneri koriste za izvođenje jednokratnih procesa poput migracija baze podataka i obavljanja administrativnih zadataka (Dynos and the Dyno Manager 2023). Za implementaciju i pokretanje web aplikacija na Heroku platformi koriste se Git sustavi za verzioniranje i Procfile datoteke (u kojima su definirani tipovi Dyno kontejnera i naredbe koje oni pokreću). Od pokretanja platforme 2007.godine, Heroku je korišten za implementaciju više od 13 000 000 aplikacija i danas obrađuje više od 60 000 000 000 zahtjeva dnevno (Heroku 2023) (How Heroku Works 2023).

3. Postavljanje projekta i aplikacije

Za postavljanje projekta i aplikacije korišteni su slijedeći izvori: (Django Documentation - Databases 2023), (Django Documentation - Settings 2023), (Django Documentation - Writing your first Django app 2023), (PostgreSQL 15.4 Documentation 2023) i (python-dotenv n.d.).

Za razvoj aplikacije u Django razvojnom okviru potrebno je kreirati projekt te aplikaciju unutar projekta. Projekt označava Python paket koji sadrži sve postavke potrebne za instancu Djanga, uključujući i postavke vezane uz aplikaciju i konfiguraciju vanjskih servisa poput baze podataka (Glossary 2023). Aplikacija u Django razvojnom okviru označuje Python paket koji sadrži određene funkcionalnosti i može se koristiti u više različitih projekta (Applications 2023). Za pokretanje projekta u naredbenom retku je korištena naredba `django-admin startproject diplomski` koja kreira korijenski direktorij `diplomski` i u njemu direktorij `diplomski` i Python datoteku `manage.py`. U unutarnjem direktoriju `diplomski` kreirano je 5 Python datoteka u kojima su definirane razne postavke.

```
diplomski
    ├── diplomski
    │   ├── __init__.py
    │   ├── asgi.py
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    └── manage.py
```

Dijagram 1. Direktoriji i datoteke kreirane pomoću naredbe django-admin startproject diplomski

Za kreiranje aplikacije u projektu potrebno je promijeniti radni direktorij u korijenski direktorij projekta (u ovom slučaju direktorij `diplomski`) i u naredbenom retku koristiti naredbu `python3 manage.py startapp vrticconnect`. Pokretanjem navedene naredbe kreiran je direktorij `vrticconnect` i u njemu više datoteka koje će biti potrebne za definiciju modela, pogleda i raznih drugih značajki aplikacije.

Za početno postavljanje projekta i aplikacije bilo je potrebno postaviti i lokalnu bazu podataka koja će se koristiti u razvoju aplikacije. Instalirana je verzija 15.1 sustava za upravljanje bazama

podataka PostgreSQL te je u njemu kreirana baza podataka „vrticconnect“ i korisnik baze podataka s lozinkom. Za potrebe aplikacije promijenjen je standard kodiranja u UTF-8, zadana izolacija transakcija postavljena je na „read_commited“ te je vremenska zona postavljena na UTC.

Nakon konfiguriranja lokalne baze podataka postavljanje projekta i aplikacije nastavljeno je uređivanjem datoteke `settings.py` koja se nalazi na putanji `diplomski/diplomski/settings.py`. U spomenutoj datoteci, u listu instaliranih aplikacija `INSTALLED_APPS` dodana je aplikacija vrticconnect te definirana je veza na bazu podataka u Python rječniku `DATABASES`. Za korištenje PostgreSQL baze podataka s Python programskim jezikom instaliran je modul `psycopg2-binary` koji služi kao adapter između aplikacije i baze podataka. U rječniku `DATABASES` definiran je `'ENGINE'` kao 'django.db.backends.postgresql_psycopg2' te su definirane potrebne varijable baze podataka (`NAME`, `USER`, `PASSWORD`, `HOST` i `PORT`) s vrijednostima prijašnje postavljene lokalne baze podataka. Kako vrijednosti varijabla potrebnih za konfiguraciju aplikacije (tajni ključ Django projekta, veza na bazu podataka ili drugi servis, podaci za povezivanje s AWS uslugama i sl.) ne bi bile vidljive u samom kodu aplikacije korištene su varijable okoline. Naziv varijable i njena vrijednost zapisane su u .env datoteci kreiranoj u istom direktoriju gdje se nalazi i `settings.py` datoteka (direktorij `diplomski`). Kako bi aplikacija mogla dohvatiti potrebne varijable okoline korišten je modul `dotenv` koji je uvezen u datoteku `settings.py` i njegova funkcija `load_dotenv()` koja učitava sve varijable definirane u .env datoteci. Za korištenje učitanih varijabli primijenjena je funkcija `getenv()` modula `os`.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': os.getenv('DB_NAME'),
        'USER': os.getenv('DB_USER'),
        'PASSWORD': os.getenv('DB_PASSWORD'),
        'HOST': os.getenv('DB_HOST'),
        'PORT': '5432',
    }
}
```

Programski kod 2. Definiranje baze podataka u datoteci settings.py koristeći varijable okoline

Prije početka razvoja same aplikacije konfiguriran je i URL maper aplikacije. U direktoriju `vrticconnect` unutar korijenskog direktorija kreirana je datoteka `urls.py` i u njoj lista

`urlpatterns` u kojoj će biti definirane `path` funkcije s URL putanjama i povezanim pogledima. Nakon toga je korijenski URL maper (koji se nalazi u mapi `diplomski`) usmjeren prema maperu aplikacije koristeći funkcije `path` i `include` modula `django.urls`.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('', include('vrticconnect.urls')),
    path("admin/", admin.site.urls),
]
```

Programski kod 3. Usmjeravanje korijenskog URL maper prema URL maperu aplikacije

4. Definiranje modela

Prilikom definiranja modela aplikacije korišteni su sljedeći izvori: (Django Documentation - Models 2023), (Django Documentation - Model instance reference 2023), (Django Documentation - Model field reference 2023), (Django Documentation - Settings 2023), (Django Documentation - Writing your first Django app, part 2 2023) i (Django Documentation - Customizing authentication in Django 2023).

Nakon postavljanja projekta i uređivanja potrebnih postavki definirani su modeli podataka u datoteci `models.py` koji će se koristiti u aplikaciji. Svaki model podataka definiran je kao podklasa klase `django.db.models.Model` (osim modela User).

Potom, obzirom da će se sadržaj aplikacije mijenjati s obzirom na skupinu, potrebno je definirati model Vrtić koji će sadržavati atribute naziv (naziv ustanove) i adresu te model Grupa koji se sastoji od naziva (nazive grupe) i vanjskog ključa naziva vrtića. Za vanjski ključ definiran je argument `on_delete` s vrijednošću `models.CASCADE` kako bi se objekt modela izbrisao kod brisanja objekta povezanog vanjskim ključem. Kod određivanja modela u Django razvojnom okviru nije potrebno definirati primarni ključ ako će primarni ključ biti identifikacijski broj (id). Kod postavljanja projekta, u datoteci `settings.py` je varijabla `DEFAULT_AUTO_FIELD` postavljena na vrijednost `'django.db.models.BigAutoField'` čime se primarni ključ modela automatski postavlja na slijedeći slobodan broj. Nakon definiranja vrtića i grupe definiran je model korisnika User koji će se koristiti i kod prijave korisnika. S obzirom da Django već nudi gotov model korisnika koji se sastoji od imena, prezimena, korisničkog imena, email adrese i lozinke bilo je potrebno samo ga proširiti s potrebnim atributima koristeći podklasu klase `django.contrib.auth.models.AbstractUser`. Modelu su dodani atributi grupa (kao vanjski ključ) i Boolean atribut Zaposlen kojim će se razlikovati roditelji od odgojitelja.

```
from django.db import models
from django.contrib.auth.models import AbstractUser
# Create your models here.

class Vrtic(models.Model):
    Naziv = models.CharField(max_length=100)
    Adresa = models.TextField(max_length=500)
```

```
    def __str__(self):
        return self.Naziv

class Grupa(models.Model):
    Naziv = models.CharField(max_length=100)
    Vrtic = models.ForeignKey(Vrtic, on_delete=models.CASCADE)

    def __str__(self):
        return self.Naziv
class User(AbstractUser):
    Grupa = models.ForeignKey(Grupa, on_delete=models.CASCADE, null=True)
    Zaposlen = models.BooleanField(default=False)

    def __str__(self):
        return self.username
```

Programski kod 4.Modeli Vrtić, Grupa i User definirani u datoteci models.py

Uz spomenute modele bilo je potrebno definirati i modele Objava (s atributima naziv, sadržaj, datum i vanjskim ključevima autor i grupa), Aktivnost (koji se sastoji od naziva, datuma aktivnosti i vanjskog ključa grupa), Poruka (s atributima sadržaj, datum i vanjskim ključevima autor i grupa) i model Fotografija koji je u nastavku detaljnije objašnjen.

```
class Objava(models.Model):
    Naslov = models.CharField(max_length=200)
    Tekst = models.TextField(max_length=1000)
    Datum_objave = models.DateTimeField("date published")
    Autor = models.ForeignKey(User, on_delete=models.CASCADE)
    Grupa = models.ForeignKey(Grupa, on_delete=models.CASCADE,  null=True)

    def __str__(self):
        return self.Naslov

class Poruka(models.Model):
    Tekst_poruka = models.TextField(max_length=1000)
    Datum_objave = models.DateTimeField("date published")
    Autor = models.ForeignKey(User, on_delete=models.CASCADE)
    Grupa = models.ForeignKey(Grupa, on_delete=models.CASCADE,  null=True)

    def __str__(self):
        return (str(self.Datum_objave) + " - " +
                str(self.Autor.username) + " - " +
                str(self.Tekst_poruka))

class Aktivnost(models.Model):
    Naziv = models.CharField(max_length=200)
    Datum_aktivnosti = models.DateField()
```

```
    Grupa = models.ForeignKey(Grupa, on_delete=models.CASCADE,  null=True)

    def __str__(self):
        return self.Naziv
```

Programski kod 5. Modeli Objava, Poruka i Aktivnost u datoteci models.py

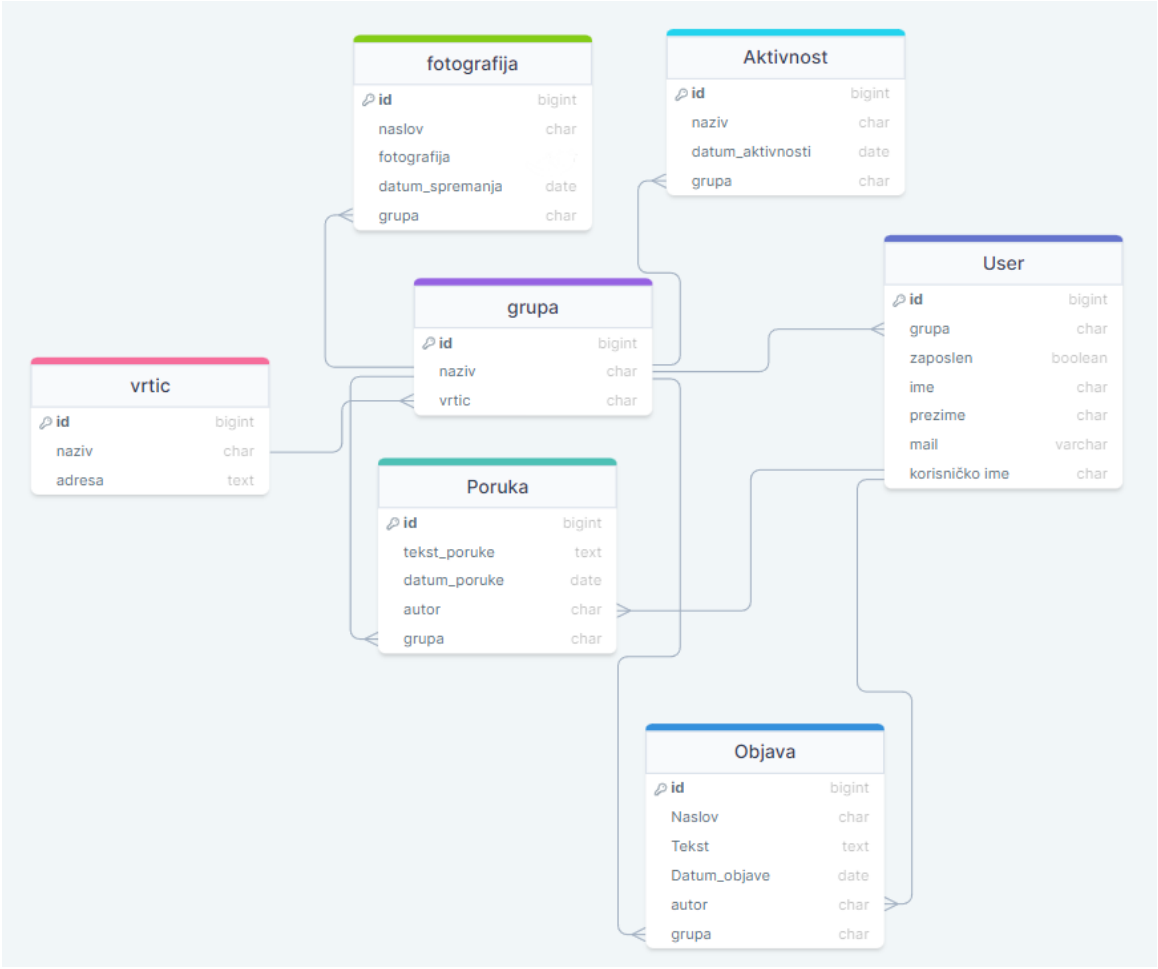
U modelu Fotografija definirani su atributi naslov, datum, fotografija i grupa (vanjski ključ). Kod atributa datum određen je i argument `auto_now_add` s vrijednošću `True` kako bi se automatski upisivao datum spremanja dok je atribut fotografija tipa `ImageField` koji se koristi za pohranjivanje i dohvaćanje objekata. Tip atributa `ImageField` nasljeđuje sve atribute i metode tipa `FileField` te provjerava da je učitani objekt fotografija. Ovom tipu atributa potrebno je definirati argument `upload_to` s putanjom direktorija za spremanje objekata.

```
class Fotografija(models.Model):
    Naslov = models.CharField(max_length=100)
    Fotografija = models.ImageField(upload_to='fotografije/')
    Datum_spremanja = models.DateTimeField(auto_now_add=True)
    Grupa = models.ForeignKey(Grupa, on_delete=models.CASCADE,  null=True)

    def __str__(self):
        return self.Naslov
```

Programski kod 6. Model Fotografija u datoteci models.py

Kod definiranja modela u Django razvojnom okviru dobra je praksa odrediti i metodu `__str__` kojim se određuje vrijednost koja će biti prikazana kod pozivanja objekta modela.

Nakon određivanja modela potrebno je napravljene promjene primijeniti i na shemu baze podataka. U Djangu se promjene primjenjuju pomoću migracija. Iako su migracije većinom automatizirane, potrebne su određene naredbe za pokretanje i izvršavanje istih. Naredbom `python3 manage.py makemigrations` skenirana je datoteka `models.py` i kreirane su migracije (.py datoteke sa zabilježenim promjenama u modelima) u direktoriju `migrations`. Nakon kreiranja migracija naredbom `python3 manage.py migrate` se zabilježene promjene primjenjuju na shemu baze podataka.



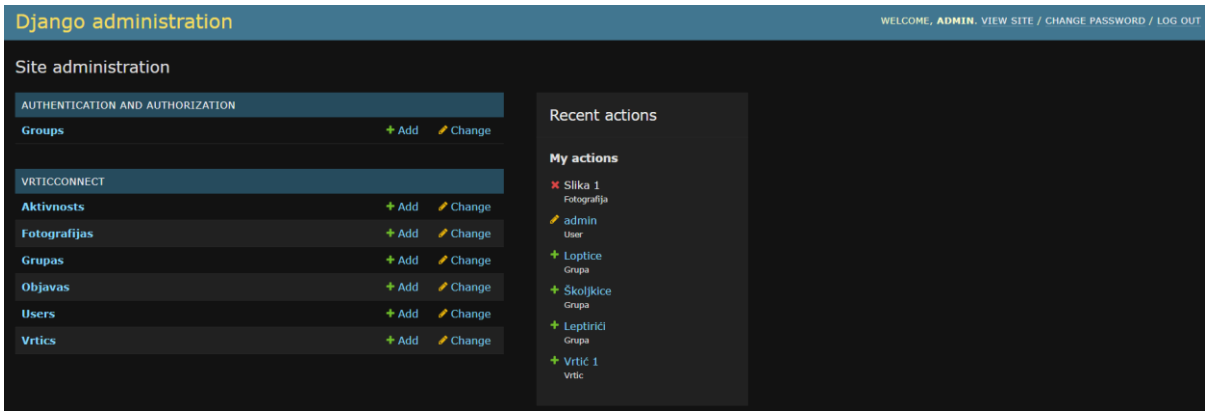
Dijagram 2. Modeli korišteni u web aplikaciji

Definirani modeli mogu se popuniti objektima u administrativnom sučelju za upravljanje modelima web aplikacije. Kod kreiranja projekta, Django automatski generira administrativno sučelje te nije potrebno podešavati nikakve postavke. Za pristupanje administrativnom sučelju kreiran je administrator naredbom `python3 manage.py createsuperuser` i registrirani su modeli u datoteci `admin.py`.

```
from django.contrib import admin
from .models import User, Grupa, Vrtic, Objava, Aktivnost, Fotografija

# Register your models here.
model_list = [User, Grupa, Vrtic, Objava, Aktivnost, Fotografija]
admin.site.register(model_list)
```

Programski kod 7. Registriranje modela u datoteci admin.py

Nakon što je razvojni poslužitelj pokrenut naredbom `python3 manage.py runserver` u administrativnom sučelju dodani su objekti modela.



Slika 2. Administrativno sučelje za upravljanje modelima

5. Uređivanje pogleda i predložaka

Kod uređivanja pogleda i predložaka korišteni su sljedeći izvori: (Django channels Tutorial Part 2: Implement a Chat Server 2022), (Django Documentation - Templates 2023), (Django Documentation - Writing your first Django app, part 3 2023), (Django Documentation - Writing your first Django app, part 4 2023), (Yenigün 2022), (Django Documentation - Customizing authentication in Django 2023), (htmx - Documentation n.d.), (AlpineJS n.d.), (Django Channels WebSockets Quickstart Tutorial n.d.), (Django Documentation - Pagination 2023), (How to customize Django forms using Django Widget Tweaks ? 2023), (Django Documentation - Working with forms 2023) i (Django Documentation - Using the Django authentication system 2023).

Prije samog definiranja pogleda i povezanih predložaka, u direktoriju aplikacije `vrticconnect` kreirani su direktorij `templates` (u kojem će biti pohranjeni svi korišteni predlošci za vrijeme razvoja aplikacije) i direktorij `static` (u kojem će za vrijeme razvoja aplikacije biti spremljeni stilski predlošci i korištene grafike). Kako bi Django mogao pronaći generirane direktorije njihove putanje upisane su u varijable `TEMPLATES`, `STATIC_URL` i `STATICFILES_DIRS` u datoteci `settings.py`.

Budući da Djangov sustav za predloške omogućuje nasljeđivanje predložaka, u direktoriju `templates` kreiran je predložak `base.html` koji će sadržavati osnovne dijelove HTML strukture koji se ponavljaju u svakom predlošku. U zaglavlju predloška definiran je opis aplikacije, skup znakova te poveznice na vanjske resurse (JavaScript biblioteke, HTMX i Bootstrap 5). Za učitavanje css datoteke stvorene u direktoriju `static` korištena je oznaka `{% load static %}`. Osim zaglavlja HTML dokumenta `base.html` sadrži i tijelo u kojem je oznakama `{% block content %}` i `{% endblock %}` označeno područje koje će predlošci koji ga nasljeđuju ispunjavati svojim sadržajem.

```
<body>
  <main>
        {% block content %}
        {% endblock %}
  </main>
  <footer class="text-center text-muted" style="border-top: solid 2px
#858585;">
    <div class="text-center p-4" style="background-color: whitesmoke;">
```

```
      © 2023, Marko Roža, diplomski rad<br>
      Slike preuzete s <a style="color: black;"
href="https:www.freepik.com">Freepik.com </a>  i
        <a style="color: black;" href="https://unsplash.com">unsplash.com</a>
    </div>
  </footer>
</body>
```

Programski kod 8. Tijelo predloška base.html

Unutar direktorija `templates` stvoreni su direktoriji `vrticconnect` i `partials` koji će sadržavati predloške stranica i parcijalne predloške uključene u stranice. U direktoriju `partials` kreiran je predložak za navigacijsku traku `navbar.html` koji će se uključivati u ostale predloške. Navigacijska traka bit će uređena nakon ostalih pogleda i predložaka.

5.1. Prijava korisnika

S obzirom da Django već nudi gotov sustav autentifikacije korisnika nije bilo potrebno definirati pogled za prijavu već samo urediti predložak, postavke u datoteci `settings.py` i putanju u korijenskom URL maperu. U datoteci `settings.py` definirane su varijable `LOGIN_REDIRECT_URL` i `LOGOUT_REDIRECT_URL` kako bi se odredile URL putanje za preusmjeravanje korisnika kod prijave i odjave. U istoj datoteci dodana je i varijabla `AUTH_USER_MODEL` kako bi se za autentifikaciju koristio prošireni model User iz datoteke `models.py`. U korijenskom URL maperu (datoteci `urls.py` u direktoriju `diplomski`) dodana je `path` funkcija s putanjom `accounts/` za Djangovu aplikaciju za autentifikaciju.

Kako bi Django prepoznao predložak za prijavu, potrebno je imenovati ga `login.html` i smjestiti ga u direktorij `registration` unutar direktorija `templates`. U predlošku je korištena oznaka `{% extends "base.html" %}` kako bi predložak naslijedio `base.html` i primijenjene su Bootstrap klase za oblikovanje izgleda elemenata. Uz Bootstrap klase korišteni su i stilski predlošci (eng. Cascading Style Sheets, CSS) spremljeni u datoteci `style.css` u direktoriju `static`.

Za prikaz i uređivanje polja obrasca za prijavu instaliran je Python modul django-widget-tweaks. Za korištenje modula bilo je potrebno upisati ga u listu instaliranih aplikacija `INSTALLED_APPS` u datoteci `settings.py` te učitati ga u predlošku oznakom `{% load`

`widget_tweaks %}`. Kako bi se spriječili zlonamjerni napadi i neovlaštene prijave u obrascu je korištena i `{% csrf_token %}` oznaka kojom se prilikom podnošenja zahtjeva za prijavu uključuje i CSRF token.

```
<form method="post" class="text-center">
        <h3 class="mb-2">Prijava</h3>
        <div>
          <p class="text-center">Dobro došli na portal
VrtićConnect!<br>Prijavite se za pregled najnovijih<br>aktivnosti vrtićke
grupe Vaše djece.</p>
        </div>
        {% csrf_token %}
        <div class="mt-4 text-center">
          {% render_field form.username placeholder="Korisničko ime"
class="form-control text-center"%}
        </div>
        <div class="mt-4 text-center">
          {% render_field form.password placeholder="Lozinka" class="form-
control text-center"%}
        </div>
        <div class="mt-4 text-center">
          <button class="btn logout-btn text-white btn-custom" type="submit"
style=" border-radius: 0.4rem; border-width: 2px; background-color:
#ee8926;  font-size: 1.2rem;">Prijavi se</button>
        </div>
      </form>
```

Programski kod 9. Obrazac za prijavu korisnika

Slika 3. Prijava korisnika

5.2. Početna stranica

Nakon uspješne prijave korisnici se preusmjeravaju na početnu stranicu web aplikacije koja se sastoji od navigacijske trake, odjeljka s obavijestima, prikaza nadolazećih aktivnosti, chata skupine i liste članova skupine (koja se prikazuje samo odgojiteljima).

Predložak početne stranice nazvan je `portal.html` i spremljen je u direktorij `vrticconnect` u direktoriju `templates`. Uz već spomenutu oznaku `extend` kojom se nasljeđuje `base.html` u predlošku su korištene i oznake za naslov (`{% block title %}`) i određivanje početka sadržaja (`{% block content %}`). Radi preglednijeg koda i jednostavnijeg uređivanja odjeljci s obavijestima, chatom skupine, prikazom aktivnosti i listom članova skupine nisu direktno definirani u predlošku `portal.html` već je za svakog kreiran parcijalni predložak

koji je uključen pomoću oznake {% include %}. Primjenom oznake {% include %} parcijalni se predlošci učitavaju unutar predloška s trenutnim kontekstom (proslijeđenim od strane pogleda).

U datoteci views.py definiran je pogled portal s dekoratorom login_required. Primjenom ovog dekoratora sprječava se pristup neprijavljenim korisnicima. Obzirom da prikazani sadržaj ovisi o skupini kojoj korisnik pripada i njegovoj ulozi (roditelj ili odgojitelj) u varijablu grupa spremljena je vrijednost atributa Grupa iz objekta korisnika a u varijablu Zaposlen spremljena je vrijednost atributa Zaposlen. Ostatak pogleda portal detaljnije je opisan u slijedećim potpoglavljima Obavijesti, Chat, Aktivnosti i Prikaz članova skupine.

5.2.1. Obavijesti

Za prikaz obavijesti na početnoj stranici kreiran je predložak objave.html koji će biti uključen u portal.html. U pogledu portal (u datoteci views.py) filtrirani su objekti modela Objava na temelju atributa grupa i spremljeni su u varijablu obavijesti_lista. Kako bi se najprije prikazivale najnovije obavijesti objekti su poredani obrnutim redoslijedom po datumu objave (koristeći funkciju order_by). Kod prikaza obavijesti korištena je i paginacija pomoću Django klase Paginator. Za primjenu paginacije, osim varijable tipa Paginator u kojoj su definirani lista objekta (obavijesti_lista) i broj objekta po stranici (4) definirane su i varijable page_number i page_obj koje su potrebne za spremanje broja stranice i objekata za željenu stranicu. U kontekstu pogleda portal proslijeđena je varijabla page_obj koja je potrebna predlošku objave.html.

U predlošku objave.html provjerava se postojanje objekta s obavijestima (page_obj) te se za svaku obavijest u objektu prikazuje naslov, sadržaj, datum i autor. Ako je korisnik odgojitelj (što se provjerava oznakom {% if Zaposlen %}) kod svake obavijesti prikazuju se i gumbovi za brisanje i uređivanje obavijesti. Za brisanje i uređivanje obavijesti korištena je HTMX biblioteka. U elementu tipa button koji označava brisanje objave dodani su HTMX atributi hx-confirm i hx-delete. Primjenom atributa hx-confirm zahtijeva se potvrda za brisanje obavijesti dok hx-delete služi za slanje AJAX zahtjeva tipa DELETE na definiranu putanju prilikom pritiska gumba. AJAX zahtjev šalje se na putanju vrticconnect/objave_list/delete/<int:id> gdje <int:id> označava id objekta za

brisanje. Navedena putanja registrirana je u URL maperu u datoteci `urls.py` te je povezana s pogledom `delete_objava` u datoteci `views.py`. Kod brisanja obavijesti pogled `delete_objava` prima zahtjev tipa DELETE i id objave za brisanje, briše odabrani objekt obavijesti (pomoću funkcija `filter` i `delete`), ponovno dohvaća vrijednosti svih potrebnih varijabli konteksta te vraća učitani parcijalni predložak `objave.html` i kontekst.

```
@require_http_methods(['DELETE'])
def delete_objava(request, id):
    Objava.objects.filter(id=id).delete()
    grupa = getattr(User.objects.get(username=request.user.username), "Grupa")
    objave_lista =
Objava.objects.filter(Grupa__Naziv__contains=grupa).order_by("-Datum_objave")
    Zaposlen =  getattr(User.objects.get(username=request.user.username),
"Zaposlen")
    paginator = Paginator(objave_lista, 5)
    page_number = request.GET.get("page")
    page_obj = paginator.get_page(page_number)
    users = User.objects.filter( Grupa__Naziv__contains=grupa )
    context = {'Zaposlen':Zaposlen, "page_obj": page_obj, 'grupa':grupa,
'users':users}
    return render(request, 'partials/objave.html', context)
```

Programski kod 10. Pogled za brisanje objava delete_objava u datoteci views.py

Isto tako, za uređivanje obavijesti definiran je element tipa button koji sadrži atribut `hx-get` s putanjom za slanje AJAX zahtjeva. Za uređivanje objava definirana je putanja `vrticconnect/objave_list/objava_edit/<int:id>` koja je registrirana u URL maperu i povezana s pogledom `objava_edit`. Pogled `objava_edit` dohvaća objekt obavijesti pomoću proslijeđenog identifikatora te obrazac za dodavanje obavijesti koji se nalazi u datoteci `forms.py`. Datoteka `forms.py` sadrži klase obrazaca u kojima je definirano koji će se atributi povezanog modela preslikavati u polja obrasca u predlošku. U obrascu za dodavanje obavijesti `ObjavaForm` određen je model Objava te atributi naslov i tekst koji će se moći ispuniti.

```
class ObjavaForm(forms.ModelForm):
    class Meta:
        model = Objava
        fields = ("Naslov", "Tekst")
```

Programski kod 11. Obrazac ObjavaForm u datoteci forms.py

Opisani pogled vraća kontekst (uključujući obrazac) i parcijalni predložak `dodaj_objavu.html` u kojem su prikazana polja obrasca `ObjavaForm` (uređeni pomoću modula django-widget-tweaks). Nakon podnošenja obrasca pogled `objava_edit` provjerava valjanost obrasca, mijenja objekt Objava s podnesenim vrijednostima te popunjava ostale atribute modela sa zadanim vrijednostima (autor, grupa i datum). Nakon toga objekt se sprema i izvodi se preusmjeravanje na početnu stranicu.

```
def objava_edit(request, id):
    objava = Objava.objects.get(id=id)
    grupa = getattr(User.objects.get(username=request.user.username), "Grupa")
    Zaposlen =  getattr(User.objects.get(username=request.user.username),
"Zaposlen")
    if request.method == 'POST':
            form = ObjavaForm(request.POST, instance=objava)
            if form.is_valid():
                obj = form.save(commit=False)
                obj.Autor = User.objects.get(username=request.user.username)
                obj.Grupa = grupa
                obj.Datum_objave = timezone.now()
                obj.save()
                return redirect("/")
    else:
        form = ObjavaForm(instance=objava)
    context={'grupa':grupa, 'form': form, 'Zaposlen':Zaposlen, 'id':id}
    return render(request, 'partials/dodaj_objavu.html', context)
```

Programski kod 12. Pogled za uređivanje objava objava_edit u datoteci views.py

Kako bi se nakon brisanja ili uređivanja obavijesti učitao vraćeni parcijalni predložak, u predlošku `objave.html` definiran je vanjski div element koji će biti zamijenjen. U div elementu definirani su atributi `hx-target`, `hx-swap` i `hx-headers`. Atributom `hx-target` određen je HTML element koji će biti zamijenjen odgovorom na AJAX zahtjev (u ovom slučaju isti element s navedenim atributom), atributom `hx-swap` definiran je tip zamijene sadržaja (dodavanje sadržaja na početak ili kraj elementa, zamjena cijelog HTML elementa ili njegove unutrašnjosti i sl.) dok je u atributu `hx-headers` određen sadržaj zaglavlja AJAX zahtjeva (u ovom primjeru definiran je CSRF token).

```
<div hx-target="this" hx-swap="outerHTML" hx-headers='{"X-CSRFToken":
"{{csrf_token }}"}'>
  {% if page_obj %}
      {% for objava in page_obj %}
```

```
      <div class="card" style="background-color: whitesmoke;">
        <div class="card-body d-flex flex-column">
          <h3 class="card-title text-center fw-bold"  style="text-decoration:
                  underline #ee8926 3px;">{{ objava.Naslov }}</h3>
          <p class="card-text text-justify">{{ objava.Tekst }}</p>
          <div class="d-flex justify-content-between align-items-end">
            <span class="card-text" style="font-size: smaller; color:gray;">
                  {{ objava.Datum_objave|date:'d-m-Y H:i' }}</span>
            <div><span class="card-text" style="font-size: smaller;
                  color:gray;"> {{ objava.Autor }}</span>
              {% if Zaposlen%}
                <button class= "delete-btn ms-2" style="border-radius: 0.4rem;
                  border:solid 2px #dbdbdb; cursor: pointer;"
                hx-confirm="Potvrdite brisanje objave"
                hx-delete="vrticconnect/objave_list/delete/{{ objava.id}}">
                  <i class="fa fa-trash"></i> </button>
                <button class= "delete-btn " style="border-radius: 0.4rem;
                  border:solid 2px #dbdbdb; cursor: pointer;"
                  hx-get="vrticconnect/objave_list/objava_edit/{{objava.id}}">
                    <i class="fa fa-edit"></i> </button>
              {% endif %}
            </div>
          </div>
        </div>
      </div>
      {% endfor %}
  {% else %}
      <div class="card">
        <div class="card-body d-flex flex-column">
          <h4 class="card-title text-center">Nema obavijesti</h4>
        </div>
      </div>
  {% endif %}
<div>
```

Programski kod 13. Prikaz objava u predlošku objave.html

Na dnu odjeljka s obavijestima implementiran je gumb za učitavanje starijih obavijesti ako objekt paginacije `page_obj` sadrži slijedeću stranicu. Gumbu su definirani već opisani HTMX atributi `hx-target` i `hx-swap` te atribut `hx-get` za slanje GET zahtjeva na putanju `/vrticconnect/objave_list/?page={{ page_obj.next_page_number }}` gdje oznaka `{{ page_obj.next_page_number }}` označava broj slijedeće stranice. Priložena putanja vezana je uz pogled `objave_list` koji dohvaća objekt paginacije `page_obj` sa slijedećim obavijestima te sve ostale već opisane varijable potrebne predlošku `objave.html`.

Pogled vraća kontekst s varijablama i učitava predložak `objave.html` koji zamjenjuje gumb za učitavanje s novim obavijestima.

```
<div>
  <div id="replaceMe" class="d-flex justify-content-center">
    {% if page_obj.has_next %}
    <div class="text-center mb-2">
      <div class="d-flex justify-content-center">
        <button class="btn card-btn mb-2 " style=" border-radius: 0.4rem;
background-color: whitesmoke;"
                hx-get="/vrticconnect/objave_list/?page={{
page_obj.next_page_number }}"
                hx-target="#replaceMe"
                hx-swap="outerHTML">Učitaj još</button>
      </div>
    </div>
    {% endif %}
  </div>
</div>
```

Programski kod 14. Primjena gumba i HTMX atributa pri paginaciji

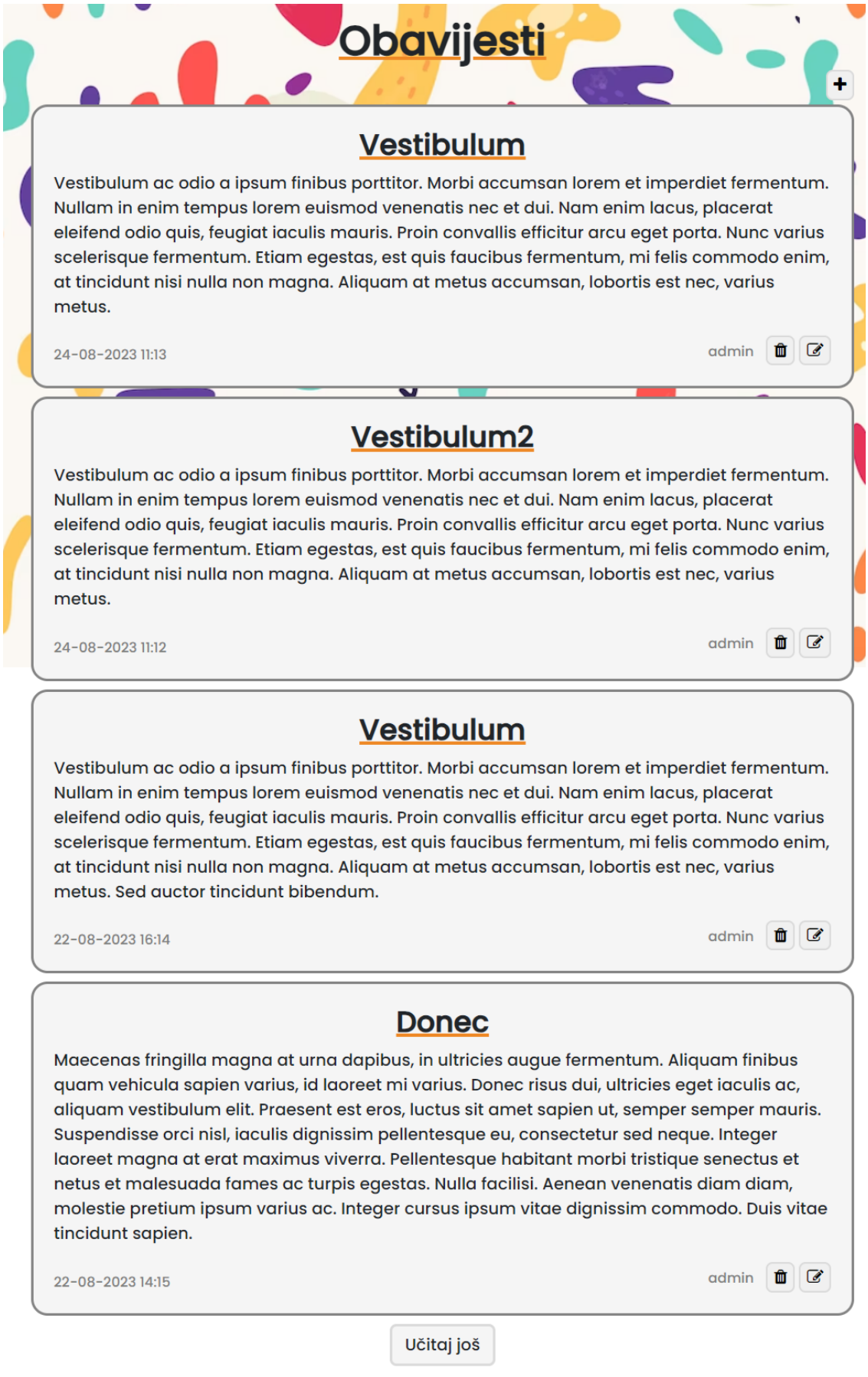
```
def objave_list(request):
    grupa = getattr(User.objects.get(username=request.user.username), "Grupa")
    objave_lista =
Objava.objects.filter(Grupa__Naziv__contains=grupa).order_by("-Datum_objave")
    Zaposlen =  getattr(User.objects.get(username=request.user.username),
"Zaposlen")
    paginator = Paginator(objave_lista, 4)
    page_number = request.GET.get("page")
    page_obj = paginator.get_page(page_number)
    users = User.objects.filter( Grupa__Naziv__contains=grupa )
    context = {'Zaposlen':Zaposlen, "page_obj": page_obj, 'grupa':grupa,
'users':users}
    return render(request, 'partials/objave.html', context)
```

Programski kod 15. Pogled za učitavanje dodatnih obavijesti objave_list u datoteci views.py

Dodavanje novih obavijesti implementirano je uključivanjem predloška `dodaj_obavijest.html` (koji se koristi i za uređivanje postojećih obavijesti) u predložak `portal.html`. U predlošku `portal.html` provjerava se uloga korisnika (provjerom varijable Zaposlen) te se za odgajatelje prikazuje gumb za dodavanje obavijesti. Vidljivost obrasca za dodavanje obavijesti uvjetovano je pomoću direktiva biblioteke Alpine.js. U vanjskom div elementu korištena je direktiva `x-data` kako bi se definirala Bool varijabla `open` i postavila

njena vrijednost na `False`. U div elementu koji sadrži parcijalni predložak za dodavanje obavijesti određena je direktiva `x-show` koja određuje vidljivost elementa i poprima vrijednost varijable `open`. Vrijednošću varijable `open` upravlja gumb za dodavanje obavijesti pomoću argumenta `@click`. Klikom na gumb varijabla `open` poprima suprotnu vrijednost te otkriva ili skriva obrazac za dodavanje obavijesti. Obrazac za dodavanje obavijesti (već opisani obrazac `ObjavaForm`) potrebno je dohvatiti u pogledu `portal` i proslijediti ga kroz kontekst . Nakon podnošenja i validacije obrasca kreira se objekt modela obavijest, popunjavaju se potrebni atributi (autor, datum i grupa) i sprema se objekt.

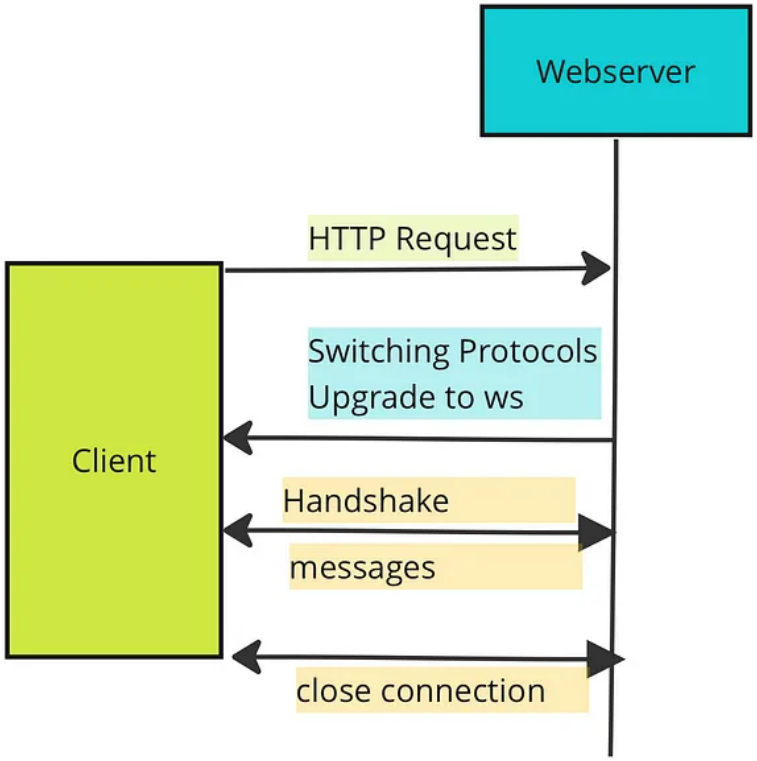
Slika 4. Obrazac za dodavanje obavijesti



Slika 5. Prikaz obavijesti na početnoj stranici web aplikacije (prikaz odgojiteljima)

5.2.2. Chat

Za implementaciju chata u aplikaciju bilo je potrebno promijeniti način na koji Django rukuje zahtjevima. Prema zadanim postavkama Django primjenjuje WSGI sučelje (Web Server Gateway Interface) za rukovanje zahtjevima između poslužitelja i aplikacije. Navedeno sučelje namijenjeno je za obradu HTTP zahtjeva i nema mogućnost obrađivanja više zahtjeva od jednom. Chat funkcionalnost zahtijeva upotrebu web utičnica i dvosmjernu komunikaciju između aplikacije i poslužitelja za što je potrebno promijeniti Djangovo sučelje u ASGI sučelje (Asynchronous Gateway Interface) koje podržava rad s web utičnicama uz standardnu obradu HTTP zahtjeva (WSGI vs ASGI n.d.) (Kaul 2023) (Yenigün 2022).



Slika 6. Rad poslužitelja s web utičnicama, slika preuzeta s: https://levelup.gitconnected.com/step-by-step-django-channels-b17a58141de1

Za implementaciju ASGI sučelja instalirani su Python moduli channels i Daphne. Modul channels proširuje Djangovu podršku za ASGI sučelje i omogućava korištenje web utičnica dok modul Daphne služi kao poslužitelj za web utičnice i HTTP zahtjeve. Instalirani moduli upisani su u listu instaliranih aplikacija `INSTALLED_APPS` u datoteci `settings.py` te je nakon toga izmijenjena datoteka `asgi.py`. U datoteci `asgi.py` dodana je funkcija `setup()` modula

`django` te je u varijablu `application` spremljen rezultat funkcije `get_default_application()` iz modula `channels.routing`.

```
import os
import django
from channels.routing import get_default_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "diplomski.settings")
django.setup()
application = get_default_application()
```

Programski kod 16. asgi.py datoteka projekta

Za primjenu web utičnica kreirana je datoteka `consumers.py` u direktoriju `vrticconnect`. Datoteka sadrži potrošače (eng.consumers) koji imaju sličnu ulogu za web utičnice kao što pogledi imaju za HTTP zahtjeve. U datoteci je definirana klasa potrošača `grupaConsumer` koja proširuje klasu `AsyncWebsocketConsumer` i u njoj su određene asinkrone funkcije koje će se pozvati kod određenih događaja. Najprije je uređena funkcija za povezivanje nazvana `connect`. Iz dobivene putanje web utičnice funkcija izvlači argument s nazivom grupe, generira naziv chat kanala (`chat_<naziv_grupe>`) i priključuje se chat grupi preko imenovanog kanala.

```
    async def connect(self):
        self.grupa = self.scope['url_route']['kwargs']['grupa']
        self.grupa_naziv = 'chat_%s' % str(unidecode(self.grupa))

        await self.channel_layer.group_add(
            self.grupa_naziv,
            self.channel_name
        )

        await self.accept()

    async def disconnect(self, close_code):
        await self.channel_layer.group_discard(
            self.grupa_naziv,
            self.channel_name
        )
```

Programski kod 17. Funkcije za povezivanje i prekidanje veze potrošača i web utičnice za potrošača grupaConsumer

Za prekid veze s chat grupom uređena je funkcija `disconnect` u kojoj se pomoću funkcije `group_discard` prekida veza. Slanje i primanje poruka iz chat grupe uređeno je funkcijama

`receive` i `chatroom_message`. Funkcija `receive` poziva se kod primanja poruke od strane web utičnice (kada sam korisnik pošalje poruku). Ona prima poruku u JSON formatu te sprema sadržaj i autora u zasebne varijable. Pomoću funkcije `database_sync_to_async` otvara se veza na bazu podataka i dohvaća se objekt modela Grupa za trenutnog korisnika. S obzirom da je definirani potrošač asinkron a baza podataka radi u sinkronom načinu rada za povezivanje je potrebno koristiti funkciju `database_sync_to_async`. Nakon povlačenja objekta modela Grupa kreira se objekt modela Poruka s autorom (sam korisnik), sadržajem poruke (primljenim kroz web utičnicu), grupom (dohvaćenom iz baze podataka) i datumom (upisanim pomoću funkcije `timezone.now()` modula `django.utils`). Poruka se sprema u bazu podataka pomoću funkcije `database_sync_to_async` i nakon toga šalje se u chat grupu pomoću funkcije `group_send`.

```
async def receive(self, text_data):
    text_data_json = json.loads(text_data)
    message = text_data_json['message']
    username = text_data_json['username']

    grupa_poruke =
        await database_sync_to_async(Grupa.objects.get)(Naziv=self.grupa)

    poruka = Poruka(
        Tekst_poruka=message,
        Autor=self.scope['user'],
        Grupa=grupa_poruke,
        Datum_objave=timezone.now()
        )

    await database_sync_to_async(poruka.save)()

    await self.channel_layer.group_send(
        self.grupa_naziv,
        {
            'type':'chatroom_message',
            'message': message,
            'username': username,
        }
    )
```

Programski kod 18. Funkcija za slanje poruka u potrošaču grupaConsumer

Za primanje poruka drugih korisnika korištena je funkcija `chatroom_message`. Spomenuta funkcija iz primljenog događaja izvlači sadržaj i korisničko ime te prikazuje ih u JSON formatu

(Django channels - databases 2022) (Django channels Tutorial Part 2: Implement a Chat Server 2022) (Guide to Django Channels: What it is, pros and cons and use cases 2022).

```
    async def chatroom_message(self, event):
        message = event['message']
        username = event['username']

        await self.send(text_data=json.dumps({
            'message':message,
            'username': username,
        }))
```

Programski kod 19. Funkcija za primanje poruka iz chat grupe u potrošaču grupaConsumer

Kako bi potrošači mogli koristiti web utičnice potrebno je definiranje `routing.py` datoteka. `Routing.py` datoteke obavljaju zadaće koje URL maper obavlja za HTTP zahtjeve. Kreirane su dvije datoteke `routing.py`, jedna za samu aplikaciju u direktoriju `vrticconnect` te jedna za cijeli projekt u direktoriju `diplomski`. U datoteci na putanji `vrticconnect/routing.py` definirana je lista uzoraka `websocket_urlpatterns` koja sadrži `re_path` funkciju s URL uzorkom putanje web utičnice i povezanim potrošačem `grupaConsumer`. Kako bi se web utičnice razlikovale od HTTP zahtjeva njihove putanje započinju s `wss/` što označava sigurne web utičnice (eng. WebSocket Secure).

```
from django.urls import re_path
from . import consumers

websocket_urlpatterns = [
    re_path(r'wss/vrticconnect/(?P<grupa>\w+)/$',
            consumers.grupaConsumer.as_asgi()),]
```

Programski kod 20. routing.py datoteka aplikacije

U datoteci `routing.py` projekta (u direktoriju `diplomski`) postavljeno je usmjeravanje zahtjeva s obzirom na korišteni protokol. Pomoću klase `ProtocolTypeRouter` definirano je usmjeravanje HTTP zahtjeva i web utičnica. Zaprimljene HTTP zahtjeve usmjerava se u Django ASGI aplikaciju (u korijenski URL maper) dok zaprimljene URL putanje web utičnica uspoređuju se s uzorcima u listi `websocket_urlpatterns` (iz datoteke `routing.py` aplikacije) pomoću klase URLRouter. Ako je pronađen isti URL uzorak u listi poziva se povezani potrošač.

```
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
import vrticconnect.routing
from django.core.asgi import get_asgi_application

django_asgi_app = get_asgi_application()

application = ProtocolTypeRouter({
    "http": django_asgi_app,
    'websocket': AuthMiddlewareStack(
        URLRouter(
            vrticconnect.routing.websocket_urlpatterns
        )
    )
})
```

Programski kod 21. routing.py datoteka projekta

Kako bi se HTTP zahtjevi usmjerili prema Django ASGI aplikaciji ponovno je uređena `settings.py` datoteka. Varijabla `WSGI_APPLICATION` zamijenjena je s varijablom `ASGI_APPLICATION` u koju je uvezena aplikacija za usmjeravanje zahtjeva iz datoteke `routing.py`. U istoj datoteci dodan je i sloj za kanale kako bi se omogućila komunikacija između različitih instanci aplikacije.

```
CHANNEL_LAYERS = {
    'default': {
        "BACKEND": "channels.layers.InMemoryChannelLayer",}
}
ASGI_APPLICATION = "diplomski.routing.application"
```

Programski kod 22. Promjene u datoteci settings.py za rad s ASGI sučeljem i slojevima kanala

Nakon implementacije ASGI sučelja i konfiguriranja web utičnica uređeni su pogled i predložak za korištenje chata u aplikaciji. S obzirom da će chat skupine biti prikazan na početnoj stranici aplikacije uređen je pogled `portal`. U pogledu je dodana varijabla `poruke` u koju su spremljeni svi objekti modela poruke za korisnikovu grupu i varijabla je uključena u kontekst. Za prikaz chata kreiran je parcijalni predložak `chat.html` koji je uključen u predložak `portal.html`. U predlošku `chat.html` dodani su elementi za prikazivanje poruka, pisanje poruke i slanje. Za učitavanje prijašnjih poruka te slanje i primanje novih poruka definirane su JavaScript skripte u kombinaciji s Django sustavom za predloške i varijablama proslijeđenim u kontekstu.

```
<div class="card d-flex align-items-center justify-content-center"
            style="border: none; height:550px; ">
  <div style="overflow:auto; display:flex; flex-direction:column-reverse;
            width: 100%;">
     <!-- prikaz poruka -->
    <div class="card-body d-flex flex-column align-items-start" id="chat-log"
            style="background-color: #ececec; border-left: solid 2px #dfdddd;
            border-right: solid 2px #dfdddd; "></div>
  </div>
       <!-- unos poruke -->
  <input id="chat-message-input" type="text" size="60"
            class="form-control mb-2">
       <!-- slanje poruke -->
  <button id="chat-message-submit" class="btn logout-btn btn-custom mt-1"
            style="background-color: #ee8926; border-radius: 0.3rem;
            background-color: #ee8926;  font-size: 1.1rem;">
      Pošalji</button>
</div>
```

Programski kod 23. Predložak chat.html

Prva JavaScript skripta učitava poruke proslijeđene u kontekstu te ih prikazuje u HTML elementu. Potrebni atributi Python objekta Poruka (sadržaj, autor i datum) spremljeni su u JavaScript varijable (`poruka_autor`, `poruka_tekst` i `poruka_datum`) pomoću oznake `escapejs` Django sustava za predloške. Za prikaz poruka kreirani su HTML elementi tipa div (za sadržaj poruke) i span (za prikaz autora i datuma). Kreiranim elementima dodan je sadržaj varijabli pomoću svojstva `textContent` te je uređen izgled. Kako bi se poruke prijavljenog korisnika razlikovale od poruka ostalih članova grupe uspoređena je vrijednost varijable `poruka_autor` s korisničkim imenom prijavljenog korisnika (proslijeđenog skripti u JSON formatu pomoću `json_script` oznake sustava za predloške).

```
{{ request.user.username|json_script:"user_username" }}
     <!-- učitavanje i prikaz poruka iz varijable poruke -->
{% if poruke %}
  {% for poruka in poruke %}
    <script>
      var user_user =
JSON.parse(document.getElementById('user_username').textContent);
      var poruka_autor = '{{ poruka.Autor.username|escapejs }}';
      var poruka_tekst = '{{ poruka.Tekst_poruka|escapejs }}';
      var poruka_datum = '{{ poruka.Datum_objave|date:"d/m/Y, H:i" |escapejs
}}';
      var chatLog = document.querySelector('#chat-log');
```

```
      var div = document.createElement('div');
      var autorSpan = document.createElement('span');

      autorSpan.textContent = poruka_autor;
      autorSpan.style.color = 'grey';
      autorSpan.style.fontSize = '0.8rem';

      var infoSpan = document.createElement('span');
      infoSpan.textContent = poruka_datum
      infoSpan.style.color = 'grey';
      infoSpan.style.fontSize = '0.8rem';
      infoSpan.style.marginBottom = '8px';

      div.textContent = poruka_tekst;
      div.classList.add('chat-message');
      if (poruka_autor === user_user) {
        div.classList.add('chat-message-right');
        div.classList.add('ms-auto');
        autorSpan.classList.add('ms-auto');
        infoSpan.classList.add('ms-auto');
        div.classList.add('me-2');
        autorSpan.classList.add('me-2');
        infoSpan.classList.add('me-2');
      }

      document.querySelector('#chat-log').appendChild(div);
      document.querySelector('#chat-log').appendChild(autorSpan);
      document.querySelector('#chat-log').appendChild(infoSpan);
    </script>
  {% endfor %}
{% endif %}
```

Programski kod 24. JavaScript skripta za učitavanje poruka proslijeđenih u kontekstu u predlošku chat.html

Druga JavaScript skripta povezuje se na web utičnicu i omogućava slanje i primanje poruka. Povezivanje na web utičnicu izvedeno je pomoću konstante `chatSocket` tipa `WebSocket` kojoj je proslijeđena URL putanja web utičnice. URL putanja web utičnice sastoji se od protokola web utičnice (`'wss://'`), naziva domaćina (dohvaćenog pomoću JavaScript funkcije `window.location.host`) i putanje na poslužitelju (`'/wss/vrticconnect/'`) kojoj je pridodana dinamički generirana vrijednost skupine (`grupa`).

```
// povezivanje s web utičnicom
  const chatSocket = new WebSocket(
    'wss://' + window.location.host + '/wss/vrticconnect/' + grupa + '/'  );
```

Programski kod 25. Povezivanje klijenta s web utičnicom

Konstanti `chatSocket` definirano je i ponašanje kod prekida veze s web utičnicom. Prilikom prekida veze s web utičnicom ispisuje se poruka o grešci u konzolu.

```
// prekidanje veze s web utičnicom
  chatSocket.onclose = function(e) {
    console.error('Chat socket closed unexpectedly');
  };
```

Programski kod 26. Prekidanje veze s web utičnicom

Za primanje novih poruka definirana je funkcija za događaj `onmessage` varijable `chatSocket`. Funkcija raščlanjuje poruku primljenu u JSON formatu i dodaje ju u element za prikaz poruka na isti način kao što je prikazano u prijašnjoj skripti za učitavanje poruka iz baze podataka.

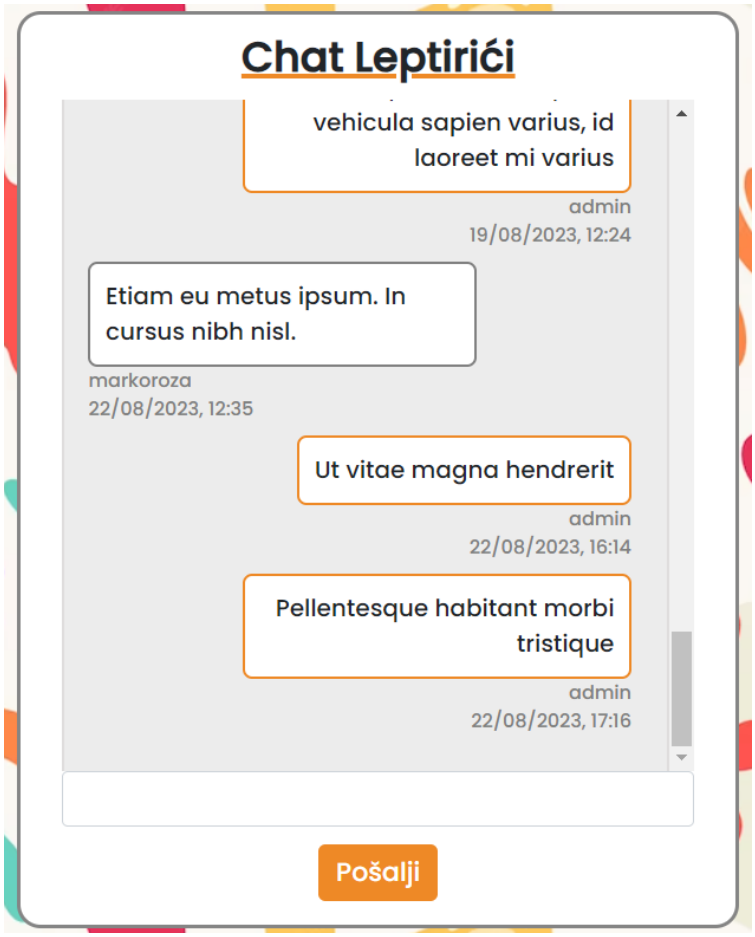
```
// primanje novih poruka
  chatSocket.onmessage = function (e) {
    const data = JSON.parse(e.data);
    const date = new Date();
    const infoSpan = document.createElement('span');
    infoSpan.textContent = date.toLocaleString([], { dateStyle: 'short',
      timeStyle: 'short' });
    infoSpan.style.color = 'grey';
    infoSpan.style.fontSize = '0.8rem';
    infoSpan.style.marginBottom = '8px';
    const autorSpan = document.createElement('span');
    autorSpan.textContent = data.username;
    autorSpan.style.color = 'grey';
    autorSpan.style.fontSize = '0.8rem';
    const messageDiv = document.createElement('div');
    messageDiv.textContent = data.message;
    messageDiv.classList.add('chat-message');
    if (data.username === user_username) {
        messageDiv.classList.add('chat-message-right');
        messageDiv.classList.add('ms-auto');
        autorSpan.classList.add('ms-auto');
        infoSpan.classList.add('ms-auto');
      }
    document.querySelector('#chat-log').appendChild(messageDiv);
    document.querySelector('#chat-log').appendChild(autorSpan);
    document.querySelector('#chat-log').appendChild(infoSpan);
  };
```

Programski kod 27. Primanje i prikazivanje novih poruka primljenih kroz web utičnicu

Isto tako definirana je funkcija za slanje novih poruka klikom na gumb za slanje. Prilikom slanja poruke pokreće se događaj `sent` konstante `chatSocket` u kojem se JavaScript vrijednost sadržaja poruke i autora konvertira u JSON format i šalje web utičnici.

```
// slanje poruke
  document.querySelector('#chat-message-submit').onclick = function(e) {
    const messageInputDom = document.querySelector('#chat-message-input');
    const message = messageInputDom.value;
    chatSocket.send(JSON.stringify({
      'message': message,
      'username': user_username,
    }));
    messageInputDom.value = '';
  };
```

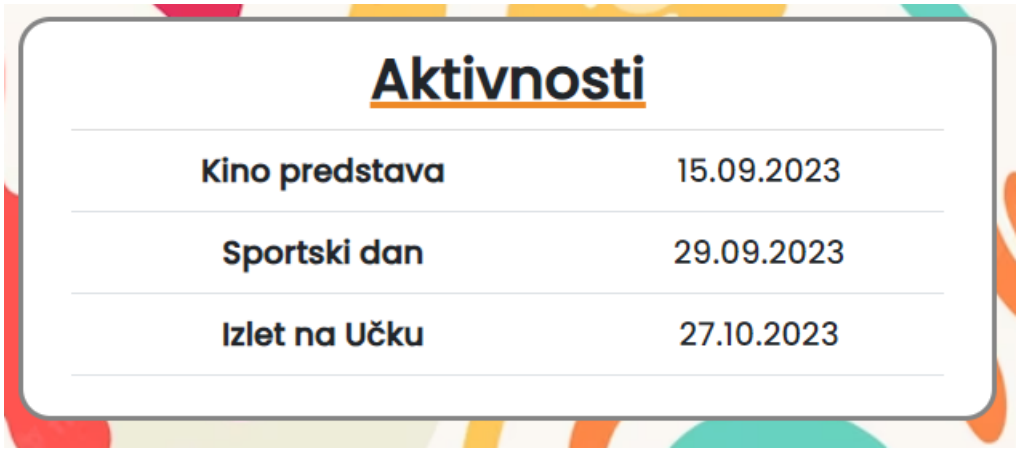
Programski kod 28. Slanje poruka web utičnici



Slika 7. Prikaz chata na početnoj stranici aplikacije

5.2.3. Aktivnosti

Uz obavijesti i chat korisnicima je na početnoj stranici web aplikacije prikazana i lista nadolazećih aktivnosti vrtićke skupine. U pogledu `portal` dohvaćeni su objekti modela Aktivnost koji se odnose na zadanu vrtićku grupu. Kod dohvaćanja objekata primijenjeno je dvostruko filtriranje na temelju atributa grupe i atributa datuma aktivnosti. Za filtriranje na temelju datuma aktivnosti korištene su funkcije `date` i `timedelta` modula `datetime`. Pomoću navedenih funkcija definiran je raspon datuma za koji će se prikazivati aktivnosti kako se korisnicima ne bi prikazivale aktivnosti koje su već odvile ili su planirane za daljnju budućnost. Za prikaz aktivnosti kreiran je parcijalni predložak `aktivnost_list.html` u kojem su pomoću HTML tablice prikazani nazivi i datumi aktivnosti.

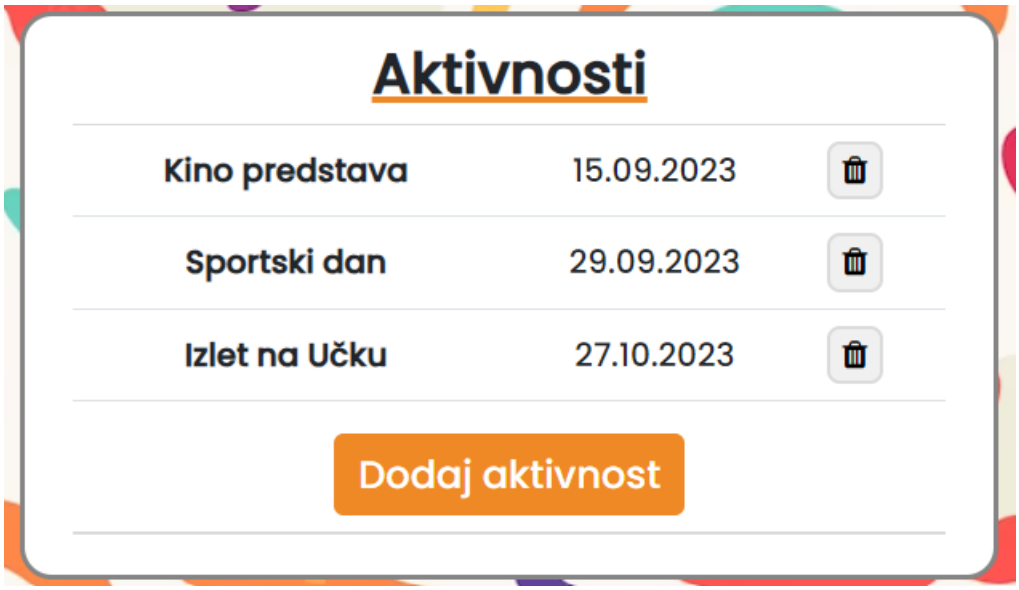


Programski kod 29. Prikaz aktivnosti roditeljima

Odgojiteljima je omogućeno i brisanje te dodavanje novih aktivnosti. Brisanje aktivnosti implementirano je gumbom s HTMX atributima na isti način kao i brisanje obavijesti. Klikom na gumb šalje se AJAX zahtjev tipa DELETE na URL putanju `vrticconnect/aktivnost_delete/{{ aktivnost.id }}/` gdje `aktivnost.id` označava identifikacijski broj objekta modela Aktivnost. Navedena putanja je u URL maperu povezana s pogledom `aktivnost_delete` koji prihvaća samo HTTP zahtjeve tipa DELETE (pomoću dekoratora `@require_http_methods (['DELETE'])`). Pogled `aktivnost_delete` briše aktivnost na temelju proslijeđenog identifikatora, dohvaća sve varijable potrebne za kontekst (atribute grupa i zaposlen korisnika te skup aktivnosti za određeni raspon datuma) i vraća učitani parcijalni predložak `aktivnost_list.html` s kontekstom.

```
@require_http_methods(['DELETE'])
def aktivnost_delete(request, id):
    Aktivnost.objects.filter(id=id).delete()
    grupa = getattr(User.objects.get(username=request.user.username), "Grupa")
    start_date =  date.today() - timedelta(days=14)
    end_date = date.today() + timedelta(weeks=25)
    aktivnosti =
Aktivnost.objects.filter(Grupa__Naziv__contains=grupa).filter(Datum_aktivnosti
__range=(start_date, end_date)).order_by('Datum_aktivnosti')
    Zaposlen =  getattr(User.objects.get(username=request.user.username),
"Zaposlen")
    context={'grupa':grupa, 'aktivnosti': aktivnosti, 'Zaposlen':Zaposlen}
    return render(request, 'partials/aktivnost_list.html',context)
```

Programski kod 30. Pogled aktivnost_delete za brisanje aktivnosti

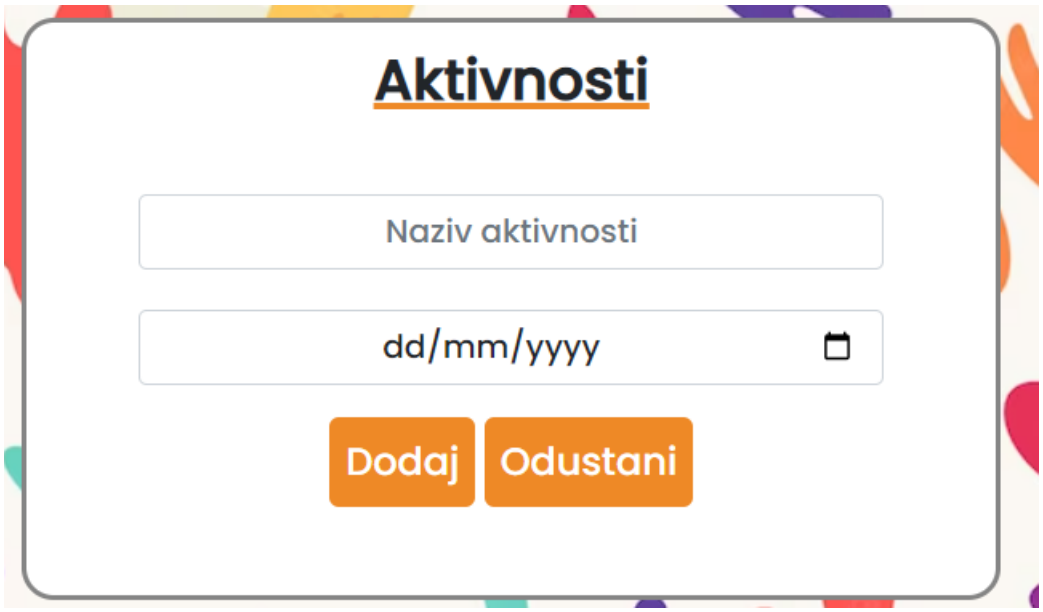


Programski kod 31. Prikaz aktivnosti odgojiteljima

Dodavanje novih aktivnosti implementirano je pomoću gumba ispod HTML tablice u predlošku `aktivnost_list.html`. Klikom na gumb izvodi se AJAX zahtjev pogledu `dodaj_aktivnost` na putanji `vrticconnect/dodaj_aktivnost/`. Pogled `dodaj_aktivnost` dohvaća obrazac za dodavanje aktivnosti `AktivnostForm` (koji se nalazi u datoteci `forms.py`) i prikazuje ga u predlošku `dodaj_aktivnost.html` kojim se zamjenjuje parcijalni predložak `aktivnost_list.html`. Nakon ispunjavanja i podnošenja obrasca, pogled provjerava ispravnost podataka. Ako je obrazac ispravno ispunjen kreira se objekt modela Aktivnost, dodaje mu se atribut grupe i sprema se. Nakon spremanja objekta ponovno se dohvaćaju potrebni objekti modela Aktivnost i učitava se predložak `aktivnost_list.html`.

```
def dodaj_aktivnost(request):
    Zaposlen =  getattr(User.objects.get(username=request.user.username),
            "Zaposlen")
    grupa =  getattr(User.objects.get(username=request.user.username),
            "Grupa")

    if request.method == 'POST':
        form = AktivnostForm(request.POST)

        if form.is_valid():
            obj = form.save(commit=False)
            obj.Grupa = grupa
            obj.save()
            start_date =  date.today() - timedelta(days=14)
            end_date = date.today() + timedelta(weeks=25)
            aktivnosti=Aktivnost.objects.filter(Grupa__Naziv__contains=grupa).
                  filter(Datum_aktivnosti__range=(start_date,end_date))
                  .order_by('Datum_aktivnosti')
            context={'grupa':grupa, 'aktivnosti': aktivnosti,
                  'Zaposlen':Zaposlen}
            return render(request, 'partials/aktivnost_list.html',context)

    else:
        form = AktivnostForm()

    context={'grupa':grupa, 'form': form, 'Zaposlen':Zaposlen}
    return render(request, 'partials/dodaj_aktivnost.html', context)
```

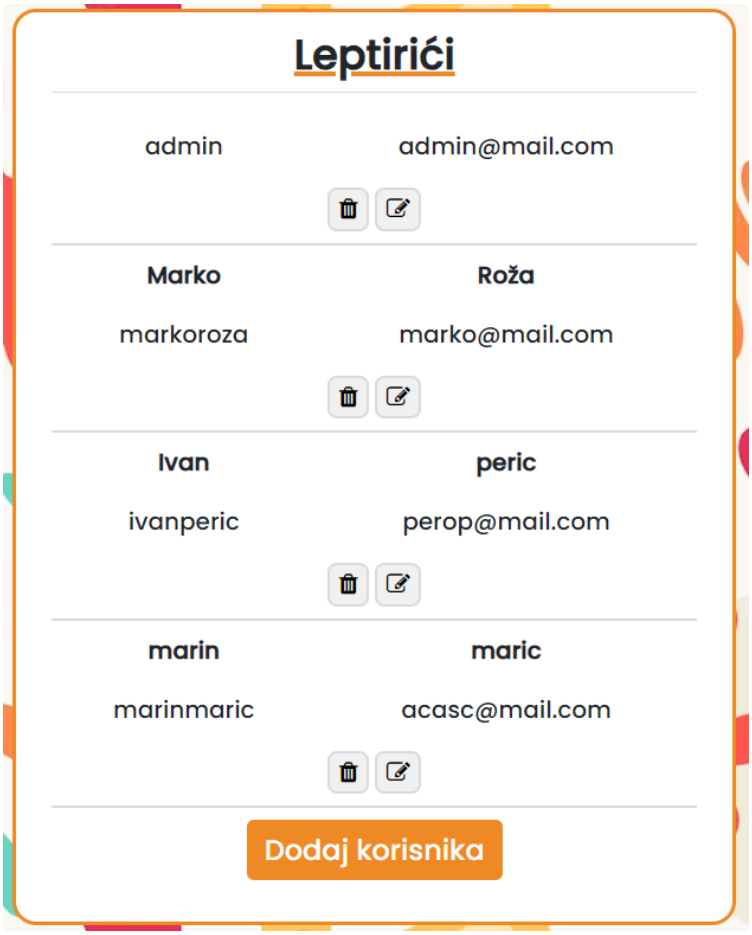
Programski kod 32. Pogled dodaj_aktivnost za dodavanje aktivnosti



Slika 8. Funkcionalnost dodavanja novih aktivnosti

5.2.4. Prikaz članova skupine

Na početnoj stranici web aplikacije odgojiteljima je prikazana i lista prijavljenih članova skupine. Za prikaz članova skupine kreiran je parcijalni predložak `user_list.html` kojem pogled `portal` prosljeđuje listu korisnika za zadanu skupinu. U navedenom predlošku prikazani su podaci svakog člana skupine te su uz svakog člana postavljeni i gumbovi za brisanje i uređivanje korisnika. Kao i kod obavijesti i aktivnosti, za uređivanje i brisanje korisnika korišteni su HTMX atributi.



Slika 9. Prikaz članova skupine na početnoj stranici aplikacije

Kod brisanja korisnika šalje se AJAX zahtjev tipa DELETE na URL putanju `vrticconnect/<int:id>/delete/` koja je povezana s pogledom `delete_user`. U pogledu `delete_user` (koji prihvaća samo zahtjeve tipa DELETE) briše se objekt modela User definiranog proslijeđenim identifikatorom (`<int:id>` definiranog u URL putanji). Nakon brisanja ponovno se dohvaćaju svi korisnici s istim atributom grupa i učitava se ažurirana lista članova skupine u predlošku `user_list.html`.

```
@require_http_methods(['DELETE'])
def delete_user(request, id):
    grupa = getattr(User.objects.get(username=request.user.username), "Grupa")
    User.objects.filter(id=id).delete()
    users = User.objects.filter( Grupa__Naziv__contains=grupa )
    context = {'grupa':grupa, 'users':users}
    return render(request, 'partials/user_list.html', context)
```

Programski kod 33. Pogled delete_user za brisanje korisnika

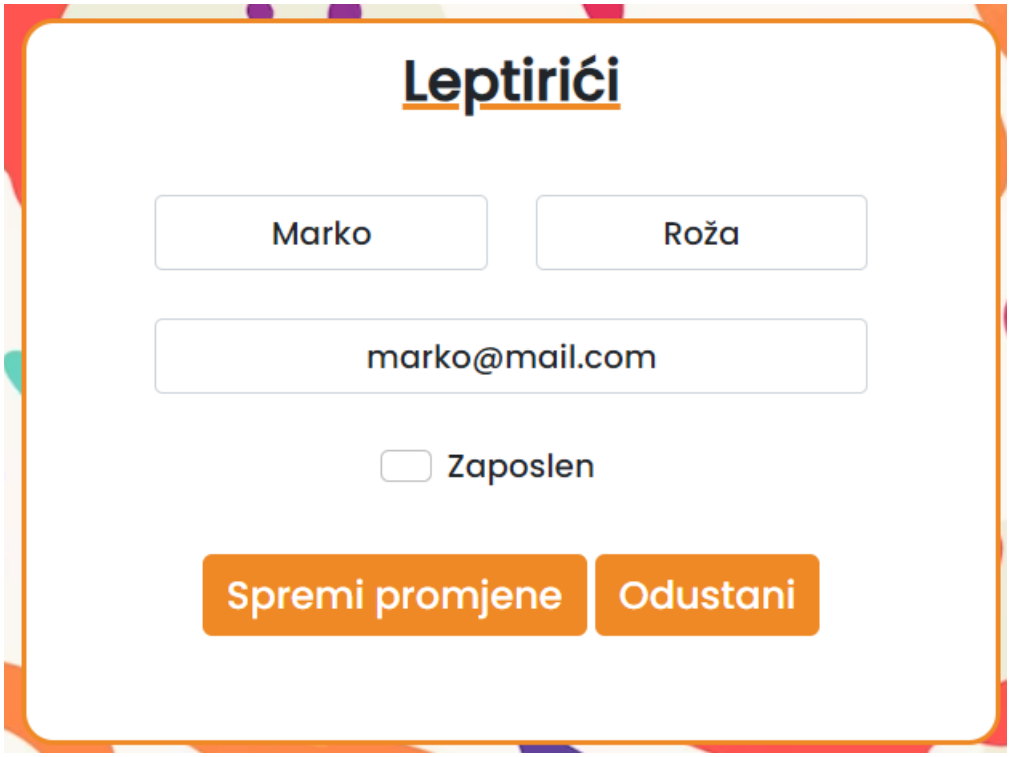
Pritiskom gumba za uređivanje šalje se zahtjev na URL putanju `vrticconnect/<int:id>/profil_edit/` (u kojoj `<int:id>` označava id korisnika) koji poziva pogled `profil_edit`.. Pogled `profil_edit` dohvaća objekt modela User s proslijeđenim identifikatorom i obrazac za uređivanje korisnika `CustomUserChangeForm` (pohranjen u datoteci `forms.py`). Obrazac `CustomUserChangeForm` proširuje klasu `UserChangeForm` i omogućuje promjenu imena, prezimena, email adrese i uloge (atributa zaposlen) korisnika. Obrazac popunjen atributima korisnika prosljeđuje se parcijalnom predlošku `profil_edit.html` koji zamjenjuje predložak `user_list.html`. Nakon uređivanja i podnošenja obrasca pogled `profil_edit` provjerava njihovu valjanost. Ako su podaci ispravni, spremaju se promjene u objekt modela User, ponovno se dohvaća lista korisnika i učitava se predložak `user_list.html`.

```
@login_required
def profil_edit(request, id):
    Zaposlen =  getattr(User.objects.get(username=request.user.username),
"Zaposlen")
    grupa = getattr(User.objects.get(username=request.user.username), "Grupa")
    user = User.objects.get(id=id)
    if request.method == 'POST':
        user_form = CustomUserChangeForm(request.POST, instance=user)
        if user_form.is_valid():
            user_form.save()
            users = User.objects.filter( Grupa__Naziv__contains=grupa )
            context = {'Zaposlen':Zaposlen, 'users': users, 'grupa': grupa}
            return render(request, 'partials/user_list.html',context)
    else:
        user_form = CustomUserChangeForm(instance=user)
    context = {'Zaposlen':Zaposlen,'grupa':grupa, 'user_form': user_form}
    return render(request, 'partials/profil_edit.html', context)
```

Programski kod 34. Pogled profil_edit za uređivanje atributa korisnika

```python
class CustomUserChangeForm(UserChangeForm):
    class Meta:
        model = User
        fields = ("email", "first_name", "last_name", "Zaposlen")
        exclude = ["password"]
```

Programski kod 35. Obrazac za uređivanje korisnika CustomUserChangeForm



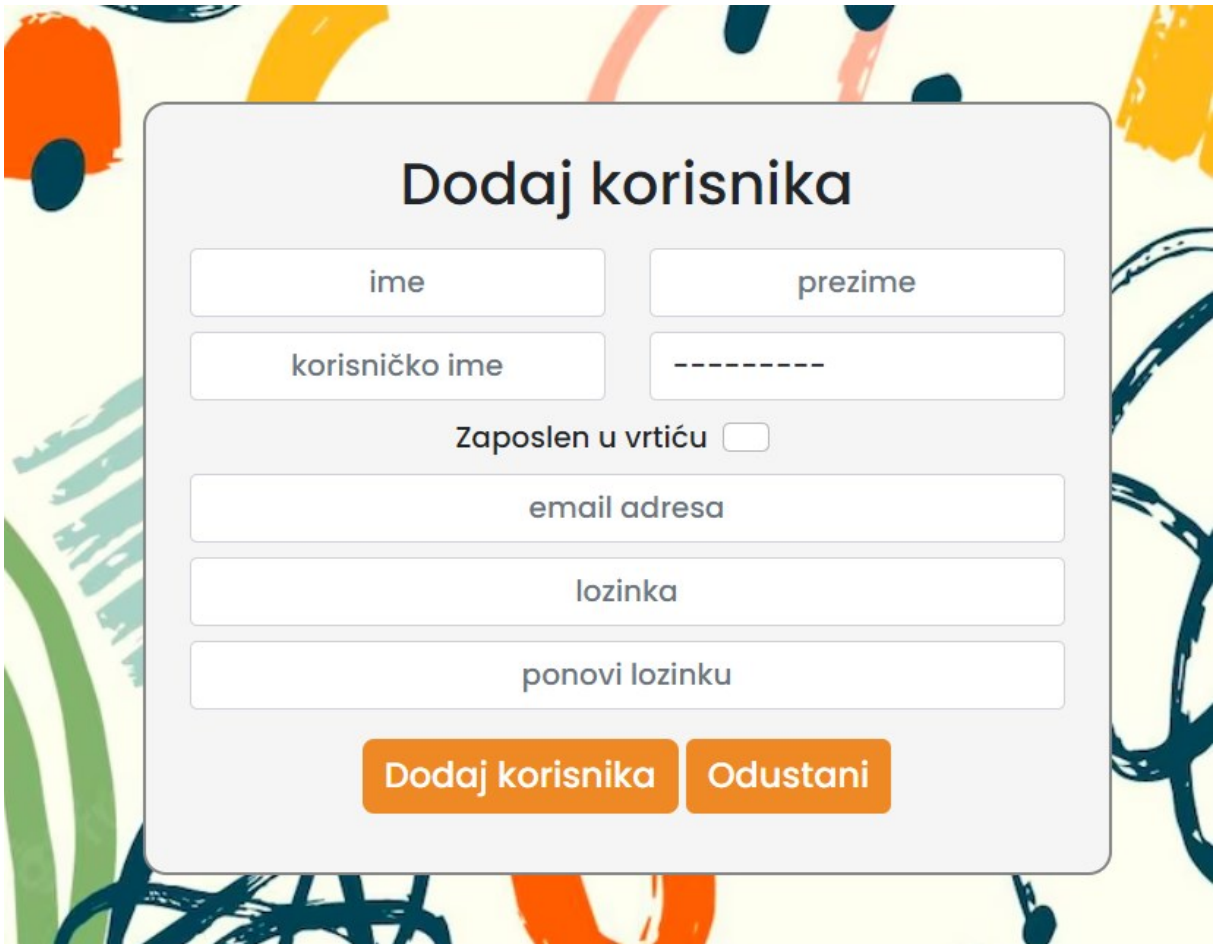
Slika 10. Uređivanje korisnika na početnoj stranici aplikacije

Osim liste članova skupine predložak `user_list.html` sadrži i gumb za dodavanje novih korisnika. Klikom na gumb odgojitelje se preusmjerava na zasebnu stranicu za dodavanje korisnika koja je opisana u sljedećem poglavlju.

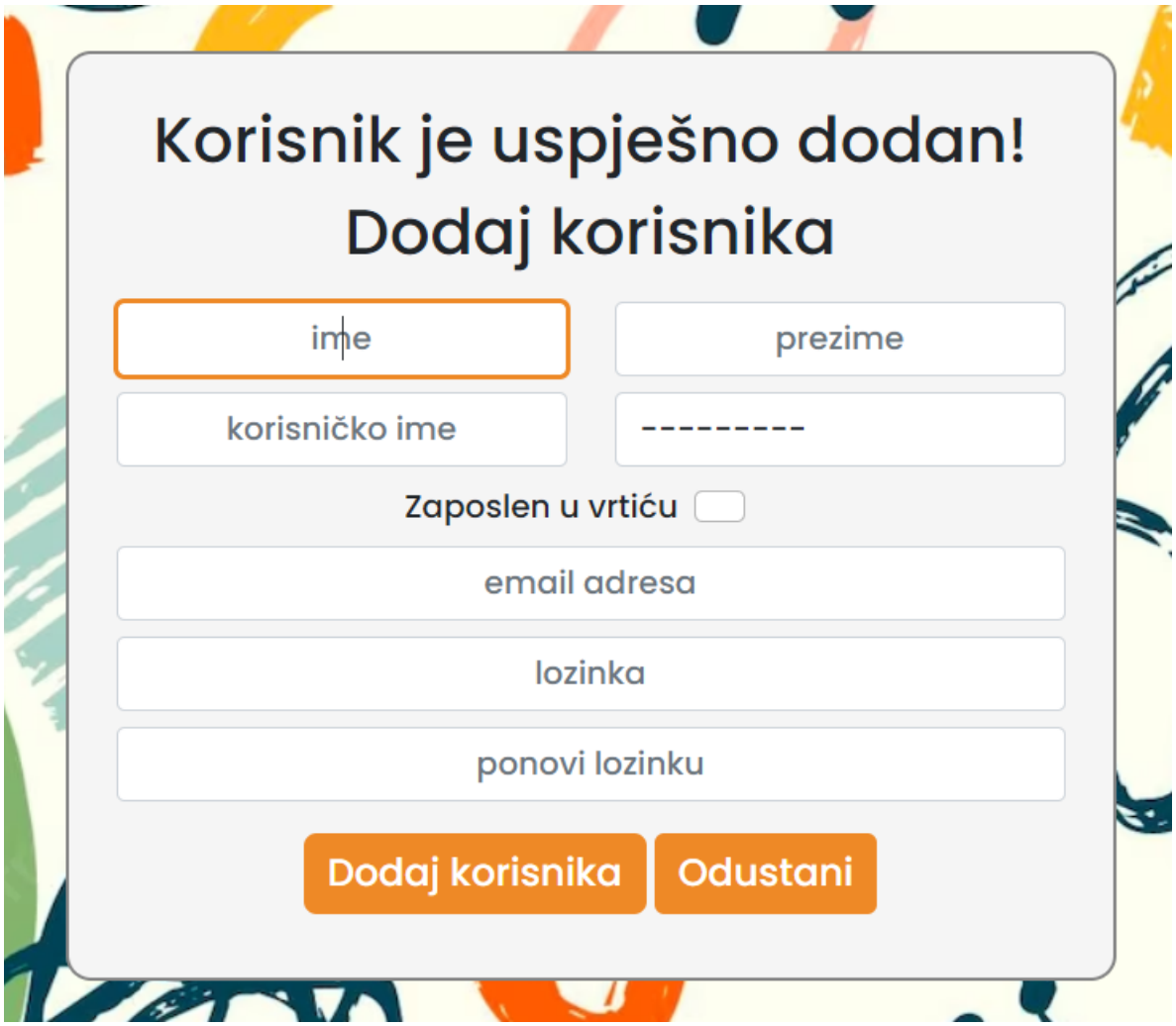
5.3. Dodavanje novih korisnika

Uz mogućnost dodavanja novih korisnika putem administrativnog sučelja dodana je i opcija dodavanja putem same web aplikacije. Kreiran je predložak `signup.html` u direktoriju `registration` (unutar direktorija `templates`) kojeg će učitavati pogled `SignUpView`. U URL maperu aplikacije u listu URL uzoraka dodana je `path` funkcija koja pogled `SignUpView` povezuje s putanjom `signup/`. Sam pogled `SignUpView` uređen je na sličan način kao i

pogledi za dodavanje novih aktivnosti i obavijesti. Pogled dohvaća potrebne varijable konteksta i obrazac za dodavanje novih korisnika `CustomUserCreationForm` te ih učitava u predlošku `signup.html`. Kako bi samo odgojitelji mogli dodavati nove korisnike, prikaz obrasca u predlošku uvjetovan je atributom zaposlen kojim se provjerava uloga korisnika. Nakon podnošenja ispunjenog obrasca pogled provjerava podatke i kreira novog korisnika. Nakon spremanja novog korisnika definira se nova varijabla `dodan_korisnik` s vrijednošću `True` te se ponovno učitava predložak `signup.html` s novim kontekstom (uključujući i varijablu `dodan_korisnik`). Opisana varijabla `dodan_korisnik` korištena je u predlošku kako bi se korisniku ispisala poruka koja potvrđuje kreiranje novog korisnika.



Slika 11. Obrazac za dodavanje novog korisnika

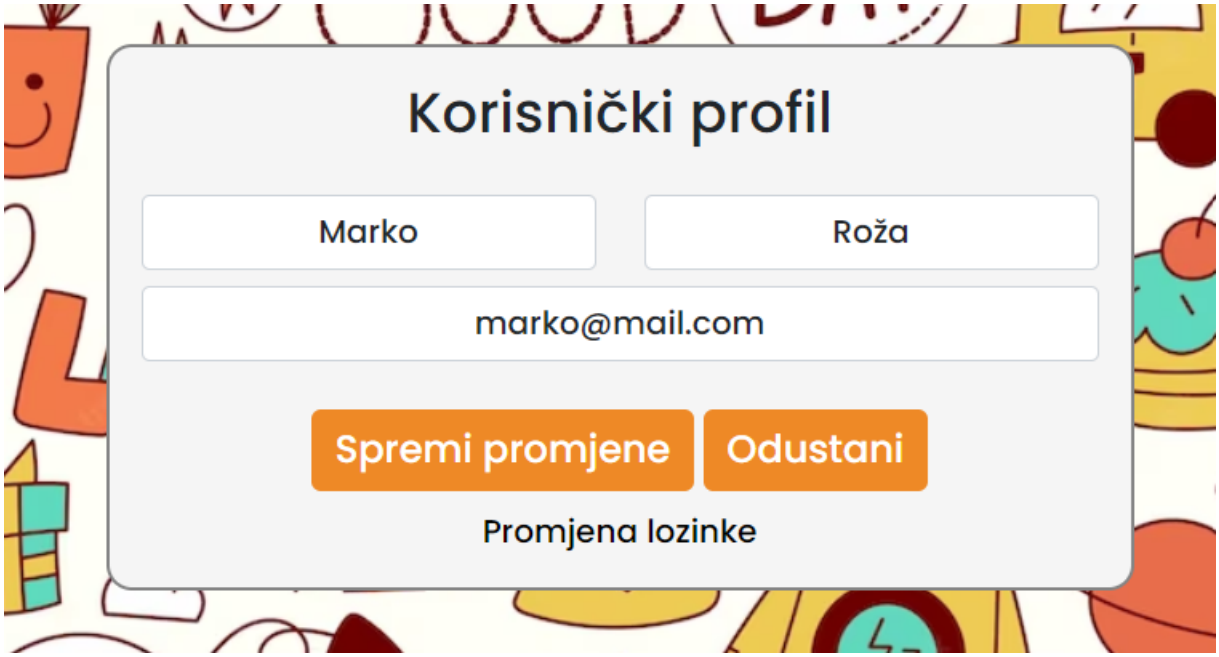


Slika 12. Potvrda o dodavanju novog korisnika

5.4. Korisnički profil

Svaki korisnik aplikacije ima mogućnost uređivanja svog korisničkog profila što je implementirano pomoću pogleda `korisnik_profil` na URL putanji `vrticconnect/korisnik_profil/`. Pogled `korisnik_profil` funkcionira na jednak način kao pogled `profil_edit` (opisan u poglavlju 5.2.4. Prikaz članova skupine) ali omogućava uređivanje samo vlastitog korisničkog profila. Kao i prethodno opisani pogled, pogled `korisnik_profil` dohvaća obrazac `CustomUserChangeForm` i varijable konteksta te ih učitava u predlošku `korisnik_profil.html`. Predložak `korisnik_profil.html` nasljeđuje predložak `base.html` i uključuje parcijalni predložak `profil_edit.html` koji je korišten i kod pogleda `profil_edit`. Kako bi se predložak `profil_edit.html` mogao

istovremeno koristiti na početnoj stranici (gdje odgajatelji mogu uređivati podatke članova skupine) i na stranici korisničkog profila bilo je potrebno urediti pogled i sam predložak `profil_edit.html`. U pogledu `profil_edit` dodana je dodatna varijabla u kontekst pomoću koje će predložak moći prikazivati različit sadržaj s obzirom na pogled koji ga učitava (pogled `profil_edit` ili pogled `korisnik_profil`). Kod prikazivanja predloška na stranici korisničkog profila prikazan je naslov „Korisnički profil“, gumbovi s HTMX atributima zamijenjeni su običnim gumbovima i prikazana je poveznica za promjenu lozinke.



Slika 13. Stranica korisničkog profila web aplikacije

5.5. Fotografije skupine

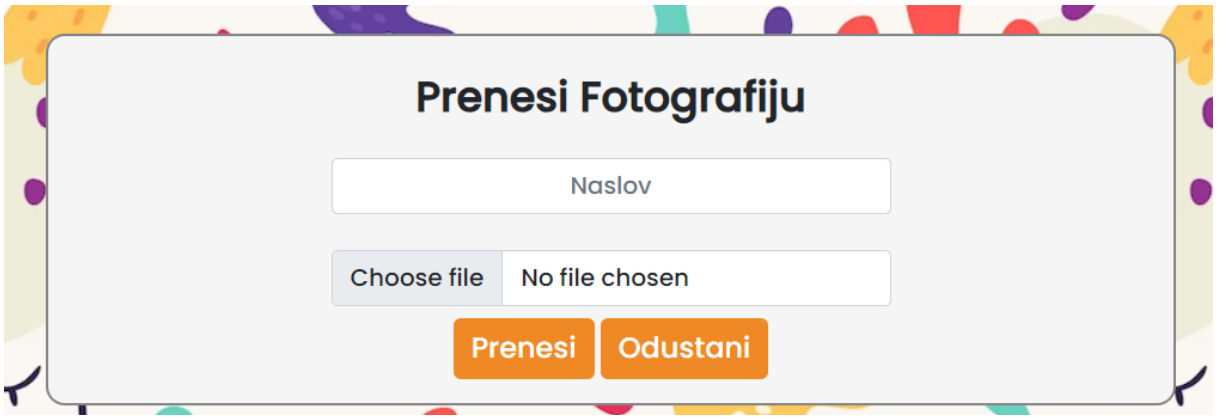
Za prikazivanje i učitavanje fotografija vrtićke skupine korišteni su pogled `foto` i predložak `foto.html`. Prije uređivanja pogleda i predloška kreiran je direktorij `media` i u njemu direktorij `fotografije` u koji će se učitane fotografije pohranjivati. Kako bi se fotografije spremale u stvoreni direktorij u datoteku `settings.py` dodane su varijable `MEDIA_ROOT` i `MEDIA_URL` s putanjom korijenskog direktorija za medijske datoteke `media`. Za spremanje i učitavanje medijskih datoteka kod samog razvoja aplikacije (u lokalnom okruženju) uređen je i URL maper aplikacije u datoteci `urls.py`. U listu uzoraka `urlpatterns` pridodana je i putanja korijenskog direktorija medijskih datoteka.

U pogledu `foto` učitani su atributi grupa i zaposlen korisnika, fotografije skupine (fotografije čiji je atribut grupa jednak atributu grupa korisnika) te obrazac za dodavanje fotografija `FotoForm` (iz datoteke `forms.py`). Radi bržeg učitavanja i preglednijeg prikaza fotografija primijenjena je paginacija za prikaz 10 fotografija po stranici. Navedene varijable proslijeđene su predlošku `foto.html` u kontekstu.

```
def foto(request):
    grupa = getattr(User.objects.get(username=request.user.username), "Grupa")
    Zaposlen =  getattr(User.objects.get(username=request.user.username),
"Zaposlen")
    fotografije =
Fotografija.objects.filter(Grupa__Naziv__contains=grupa).order_by("-
Datum_spremanja")
    paginator = Paginator(fotografije, 10)
    page_number = request.GET.get("page")
    page_obj = paginator.get_page(page_number)
    if request.method == 'POST':
        form = FotoForm(request.POST, request.FILES)
        grupa =  getattr(User.objects.get(username=request.user.username),
"Grupa")
        if form.is_valid():
            obj = form.save(commit=False)
            obj.Datum_spremanja = timezone.now()
            obj.Grupa = grupa
            obj.save()
            form.save()
            return redirect(to='vrticconnect:foto')
    else:
        form = FotoForm()
    context = {'Zaposlen':Zaposlen,'grupa':grupa, 'fotografije': page_obj,
'form':form}
    return render(request, 'vrticconnect/foto.html', context)
```

Programski kod 36. Pogled foto za prikaz i učitavanje fotografija

U predlošku `foto.html` korištene su direktive biblioteke Alpine.js za prikazivanje obrasca za dodavanje fotografija (kao što je prethodno opisano za dodavanje obavijesti u poglavlju 5.2.1. Obavijesti). Pritiskom na gumb za dodavanje fotografija (koji se prikazuje samo odgajateljima) prikazuje se obrazac za učitavanje fotografija sadržan u uključenom parcijalnom predlošku `upload_foto.html`. Obrazac se sastoji od polja za naslov fotografije te polja za učitavanje fotografije. Podnošenjem ispunjenog obrasca pogled `foto` provjerava podatke, objektu dodaje atribute grupa i datum spremanja te sprema objekt.



Slika 14. Obrazac za učitavanje fotografija

Za prikaz fotografija u predlošku korišten je HTML element img kojem je definiran atribut loading s vrijednošću „lazy“ kako bi se fotografije učitale nakon učitavanja svih ostalih dijelova stranice. Za učitavanje fotografija i njihovo raspoređivanje u mrežu napisana je JavaScript skripta koja koristi biblioteke Masonry i Imagesloaded. Za vrijeme učitavanja fotografija i njihovog preslagivanja u mrežu na stranici se prikazuje animacija učitavanja. Nakon završetka preslagivanja pomoću biblioteke imagesLoaded skriva se animacija za učitavanje i prikazuju se fotografije. U stilskom predlošku `style.css` uređen je izgled animacije za učitavanje i broj slika po retku s obzirom na širinu ekrana na kojem se aplikacija prikazuje. Sve korištene fotografije preuzete su s web stranice freepik.com.

```
<script>
  var grid = document.querySelector('.grid');
  var msnry = new Masonry( grid, {
    itemSelector: '.grid-item',
    columnWidth: '.grid-sizer',
    percentPosition: true
  });

  imagesLoaded( grid ).on( 'done', function() {
    // layout Masonry after each image loads
    msnry.layout();
    document.querySelector( "#loader2").style.visibility = "hidden";
    document.querySelector( "#foto").style.visibility = "visible";
  });
</script>
```

Programski kod 37. JavaScript skripta za prikaz i slaganje fotografija u mrežu

Slika 15. Animacija kod učitavanja fotografija

Slika 16. Prikaz fotografija u mreži

5.6. Promjena lozinke

Na stranici korisničkog profila korisnicima je prikazana i poveznica za promjenu lozinke. Klikom na poveznicu korisnik se preusmjerava na putanju `password_change/` koja je povezana s pogledom `ChangePasswordView`. Budući da Django već nudi gotov pogled za promjenu lozinke (s gotovim obrascem `PasswordChangeForm`) korišteni pogled `ChangePasswordView` samo proširuje gotovu klasu pogleda `PasswordChangeView`

(uvezenu iz paketa `django.contrib.auth.views`) i mijenja korišteni predložak te adresu preusmjeravanja kod uspješne promjene lozinke. Za razliku od ostalih pogleda koji su definirani kao funkcije pogled `ChangePasswordView` definiran je kao klasa i u URL maperu na njega je primijenjena funkcija `as_view()` kako bi se koristio kao pogled.

```
class PasswordChangeView(PasswordContextMixin, FormView):
    form_class = PasswordChangeForm
    success_url = reverse_lazy("password_change_done")
    template_name = "registration/password_change_form.html"
    title = _("Password change")

    @method_decorator(sensitive_post_parameters())
    @method_decorator(csrf_protect)
    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super().dispatch(*args, **kwargs)

    def get_form_kwargs(self):
        kwargs = super().get_form_kwargs()
        kwargs["user"] = self.request.user
        return kwargs

    def form_valid(self, form):
        form.save()
        # Updating the password logs out all other sessions for the user
        # except the current one.
        update_session_auth_hash(self.request, form.user)
        return super().form_valid(form)
```

Programski kod 38. Gotova klasa PasswordChangeView iz modula django.contrib.auth.views

```
class ChangePasswordView(SuccessMessageMixin, PasswordChangeView):
    template_name = 'vrticconnect/password_change_form.html'
    success_url = reverse_lazy('vrticconnect:korisnik_profil')
```

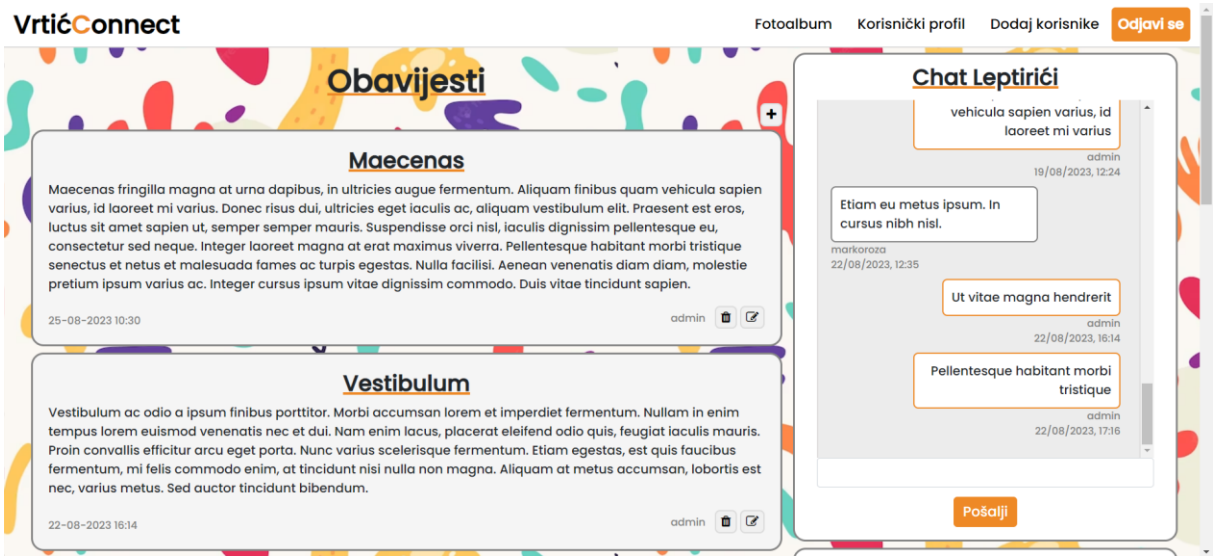
Programski kod 39. Klasa ChangePasswordView koja proširuje klasu PasswordChangeView

S pogledom `ChangePasswordView` korišten je predložak `password_change_form.html`. Predložak nasljeđuje predložak `base.html` i koristi django-widget-tweaks modul za uređivanje i prikaz polja obrasca za promjenu lozinke.

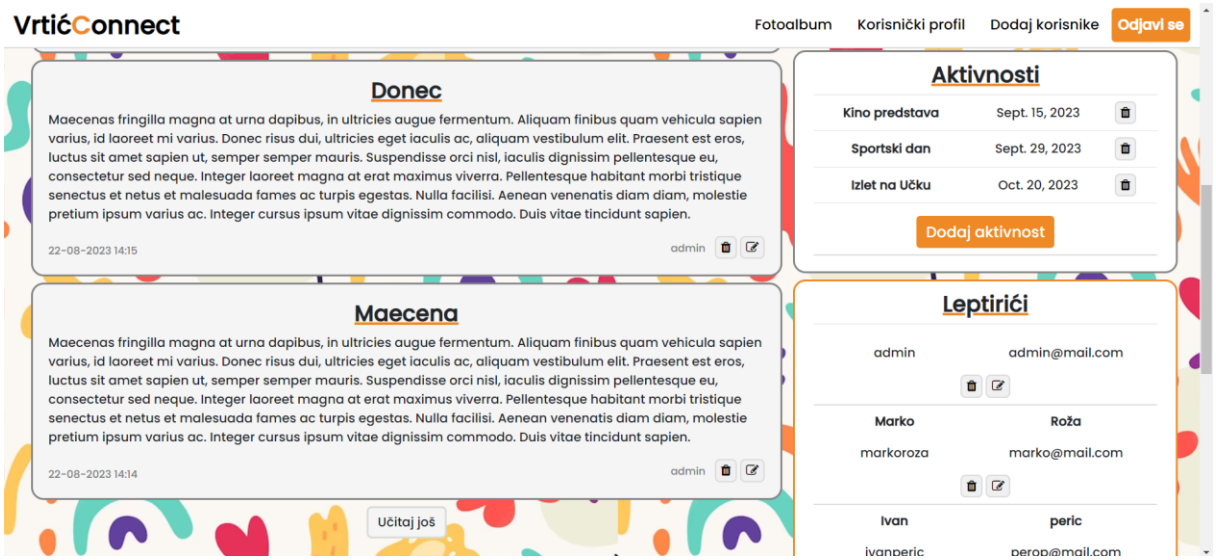
Slika 17. Promjena lozinke korisnika

5.7. Navigacijska traka

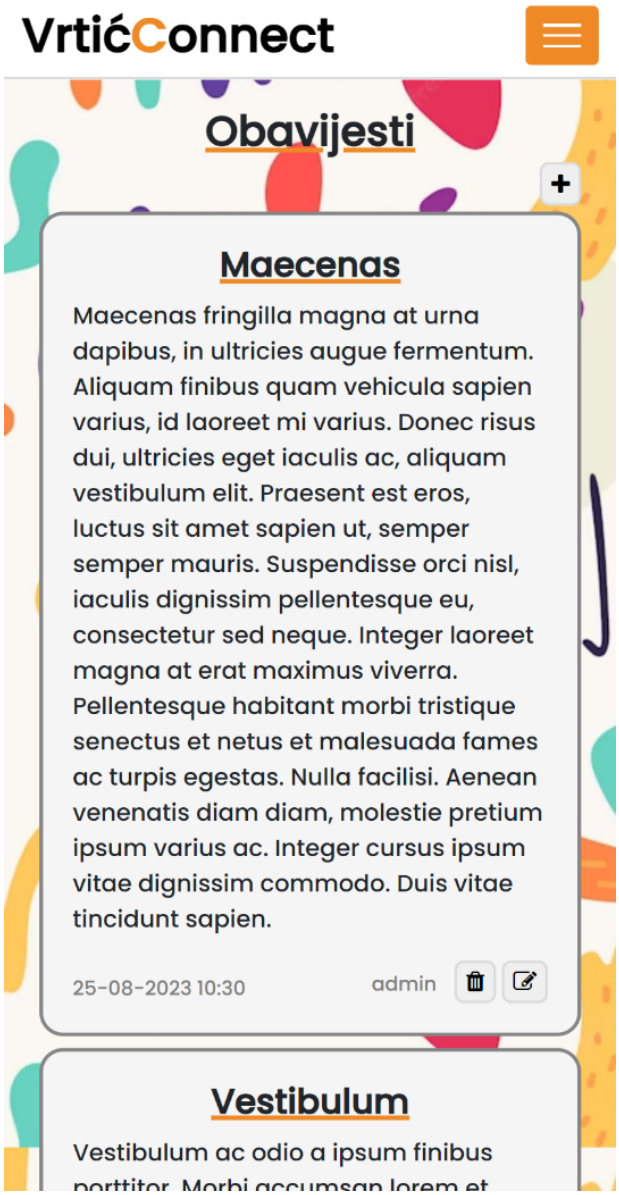
Navigacijska traka uređena je u predlošku `navbar.html` i uključena je u predloške `portal.html`, `korisnik_profil.html`, `foto.html` i `password_change.html`. Sadrži poveznice na početnu stranicu, korisnički profil te stranicu s fotografijama skupine dok je za odgojitelje prikazana i poveznica za dodavanje novih korisnika. Za uređivanje izgleda navigacijske trake i različit prikaz na različitim veličinama ekrana korištene su klase okvira Bootstrap 5.



Slika 18. Izgled početne stranice aplikacije 1.dio



Slika 19. Izgled početne stranice aplikacije 2.dio



Slika 20. Izgled početne stranice aplikacije na mobilnim uređajima

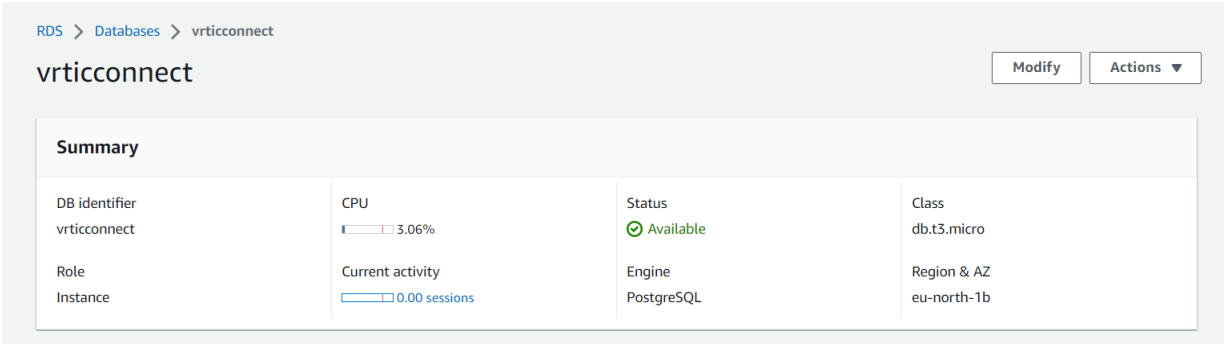
6. Povezivanje aplikacije s AWS uslugama

Za povezivanje aplikacije s AWS uslugama korišteni su sljedeći izvori: (Ridvan 2023) i (Herman 2023).

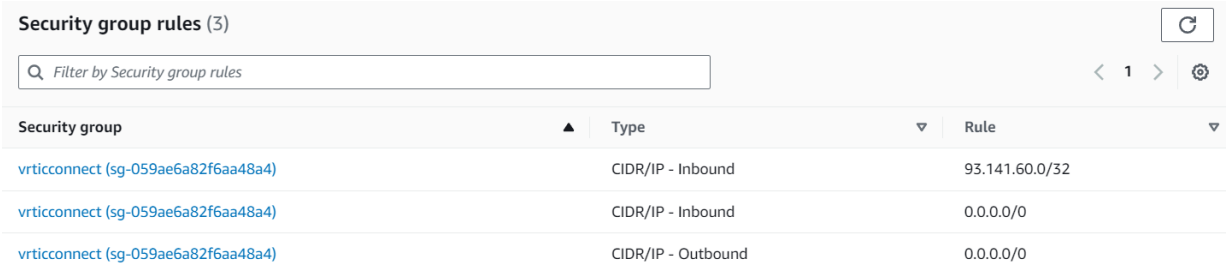
Prije postavljanja aplikacije na web poslužitelj projekt je povezan s AWS uslugama Amazon RDS i Amazon S3. Uslugom Amazon S3 zamijenit će se lokalna baza podataka korištena tokom razvoja aplikacije dok će se usluga Amazon S3 koristiti za pohranjivanje i posluživanje statičkih i medijskih datoteka. Za korištenje AWS usluga otvoren je korisnički račun i odabrane su besplatne inačice usluga.

6.1. Povezivanje na Amazon RDS

U usluzi Amazon RDS kreirana je PostgreSQL baza podataka. Prilikom konfiguriranja usluge odabrane su verzija 15.3 PostgreSQL baze podataka i regija eu-north-1, definirani su podaci potrebni za pristup (naziv baze podataka, naziv glavnog korisnika i njegova lozinka) i dopušten je javni pristup bazi podataka. Budući da je korištena besplatna inačica usluge nije bilo moguće birati instance na kojima će se baza podataka izvoditi već je ponuđena samo instanca db.t3.micro. Nakon kreiranja baze podataka dodano je ulazno pravilo (eng. inbound rule) kako bi se aplikacija mogla spajati.

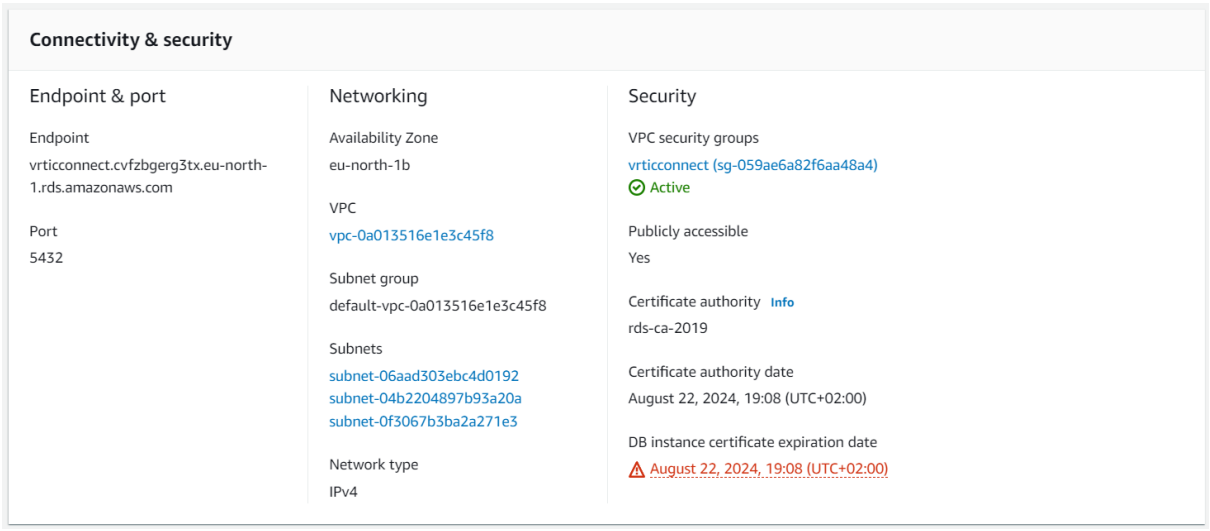


Slika 21. Svojstva baze podataka kreirane na Amazon RDS usluzi



Slika 22. Ulazna i izlazna pravila baze podataka

Budući da su u projektu već korištene varijable okoline za pohranu bitnih vrijednosti nije bilo potrebno mijenjati `settings.py` datoteku projekta već samo zamijeniti podatke lokalne baze podataka u .env datoteci s podacima baze podataka kreirane na RDS usluzi. U .env datoteci definirani su ime, korisnik i lozinka postavljeni kod konfiguriranja baze podataka te domaćin i priključak koji su prikazani u nadzornoj ploči (eng. dashboard) baze podataka. Bitno je napomenuti da iako je baza podataka nazvana „vrticconnect“ kod spajanja aplikacije na bazu potrebno je koristiti ime „postgres“.

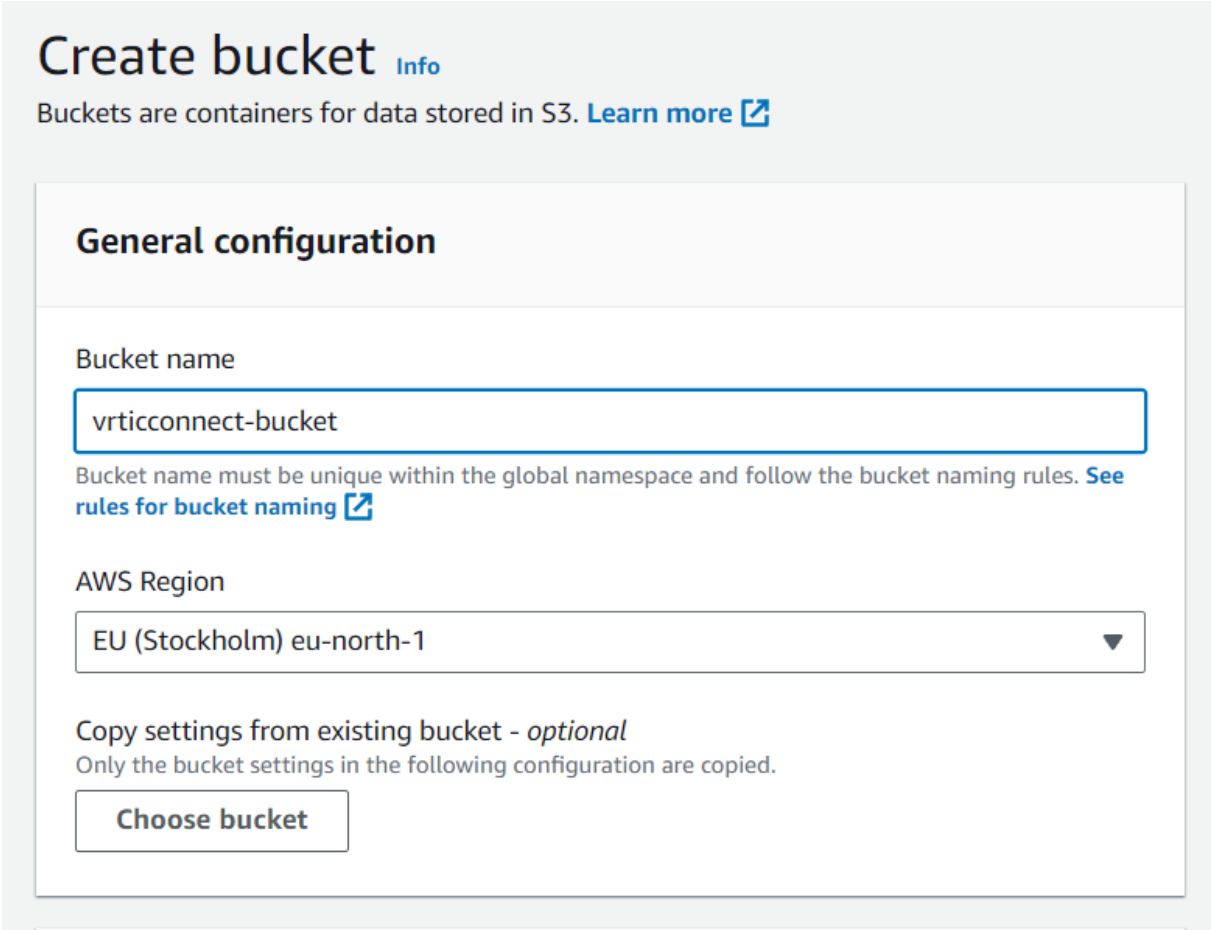


Slika 23. Nadzorna ploča baze podataka u RDS usluzi

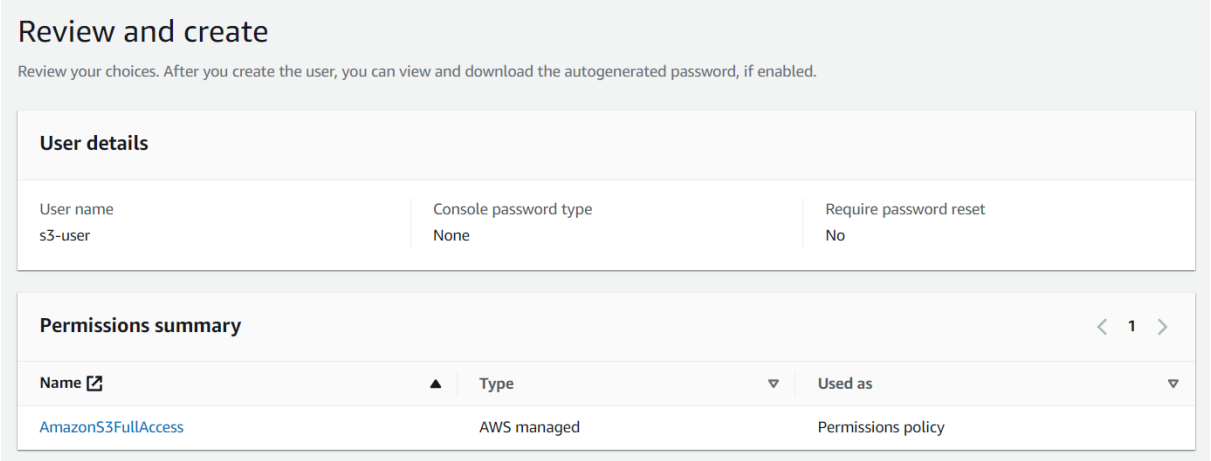
Nakon definiranja podataka za spajanje na bazu u naredbenom retku pokrenute su naredbe `python3 manage.py makemigrations` i `python3 manage.py migrate` kako bi se modeli podataka projekta preslikali u novu bazu podataka. Budući da je aplikacija sada povezana na novu bazu podataka koja ne sadrži nikakve podatke ponovno je kreiran administrator aplikacije naredbom `python3 manage.py createsuperuser` i dodani su podaci za svaki model.

6.2. Povezivanje na Amazon S3

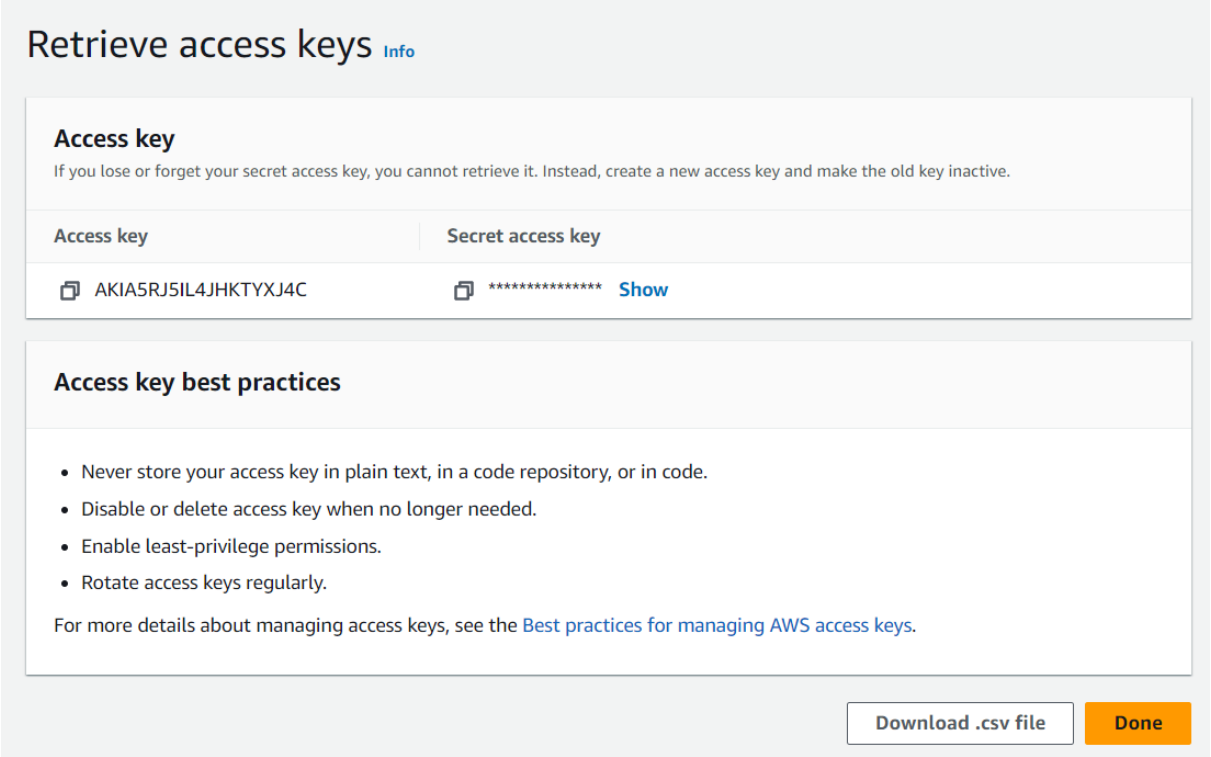
U usluzi Amazon S3 kreiran je spremnik koji će pohranjivati i posluživati statičke i medijske datoteke. Kod konfiguracije spremnika određeno je ime spremnika „vrticconnect-bucket“, regija eu-north-1 i uređen je pristup spremniku (dopušten je javni pristup). Za pristup kreiranom spremniku definiran je i korisnik (pomoću Amazon IAM platforme) preko kojega će se aplikacija spajati na spremnik. Korisnik je nazvan „s3-user“ i dodijeljen mu je potpuni pristup spremniku (pomoću AmazonS3FullAccess skupa dopuštenja). Korisniku je generiran i pristupni ključ s pripadajućim tajnim ključem koji je potreban aplikaciji za povezivanje.



Slika 24. Kreiranje spremnika



Slika 25. Kreirani korisnik (s potrebnim ovlastima) za pristup spremniku



Slika 26. Generiranje pristupnog ključa korisnika

Za povezivanje aplikacije sa spremnikom instalirani su Python moduli django-storages i boto3. Django-storages omogućava povezivanje Django projekta s raznim servisima za pohranu podataka poput usluga Amazon S3, Azure Storage, Digital Ocean, Dropbox i Google Cloud Storage dok je boto3 modul za kreiranje, konfiguraciju i upravljanje AWS uslugama kojeg održava sam Amazon (Boto3 documentation 2023) (django-storages 2023). Povezivanje aplikacije sa spremnikom uređeno je u `settings.py` datoteci koristeći varijable okoline. U .env datoteci definirane su varijable okoline za mjesto pohrane (`STORAGE_DESTINATION`), pristupni ključ korisnika (`AWS_ACCESS_KEY_ID`), tajni ključ korisnika

(AWS_SECRET_ACCESS_KEY) i ime spremnika (AWS_STORAGE_BUCKET_NAME). U datoteci settings.py uključen je modul django-storages u listu instaliranih aplikacija INSTALLED_APPS i konfiguriran je pristup spremniku. Pri konfiguraciji veze na spremnik provjerava se mjesto pohrane, ako je kao mjesto pohrane definirana usluga Amazon S3 (vrijednost varijable okoline STORAGE_DESTINATION je jednaka 's3') slijedi definiranje varijabli i opcija potrebnih za povezivanje na spremnik. Ako je kao mjesto pohrane definirana druga vrijednost aplikacija se usmjerava na lokalne direktorije za statične i medijske datoteke. Kod povezivanja aplikacije sa spremnikom potrebno je definirati varijable s pristupnim i tajnim ključem korisnika preko kojeg se aplikacija povezuje (AWS_ACCESS_KEY_ID i AWS_SECRET_ACCESS_KEY) i zasebno definirati mjesto pohrane za statične datoteke i za medijske datoteke. Kod definiranja mjesta pohrane statičnih datoteka potrebno je odrediti varijable STATICFILES_DIRS, STATIC_URL, STATICFILES_STORAGE i STATIC_ROOT. U varijabli STATIC_URL definirana je URL putanja koja će se koristiti kod dohvaćanja statičnih datoteka. U varijabli STATICFILES_STORAGE određen je mehanizam za pohranjivanje koji će upravljati statičnim datotekama (sadržan u modulu django-storages) dok je u varijabli STATIC_ROOT definirana URL putanja korijenskog direktorija statičnih datoteka. Za usmjeravanje aplikacije prema mjestu pohrane medijskih datoteka određene su varijable MEDIA_URL i DEFAULT_FILE_STORAGE. U varijabli MEDIA_URL određena je URL putanja na kojoj se nalaze medijske datoteke dok je u varijabli DEFAULT_FILE_STORAGE određen mehanizam za pohranjivanje koji će upravljati medijskim datotekama. Kao mehanizam za pohranjivanje navedena je klasa MediaStorage definirana u datoteci storage_backends.py. Navedena klasa proširuje klasu S3Boto3Storage modula django-storages tako što mijenja lokaciju datoteka (u media), onemogućuje pisanje preko postojećih datoteka i omogućuje javno čitanje datoteka. Za sve objekte spremnika (statične i medijske datoteke) definirano je i vrijeme koliko dugo će biti sačuvani u predmemoriji (pomoću varijable AWS_S3_OBJECT_PARAMETERS).

```
STORAGE_DESTINATION = os.getenv('STORAGE_DESTINATION')

if STORAGE_DESTINATION == 's3':
    AWS_ACCESS_KEY_ID = os.getenv('AWS_ACCESS_KEY_ID')
    AWS_SECRET_ACCESS_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')
    AWS_STORAGE_BUCKET_NAME = os.getenv('AWS_STORAGE_BUCKET_NAME')
    AWS_S3_CUSTOM_DOMAIN = f'{AWS_STORAGE_BUCKET_NAME}.s3.amazonaws.com'
    AWS_S3_OBJECT_PARAMETERS = {
        'CacheControl': 'max-age=86400',
```

```
    }
    AWS_LOCATION = 'static'

    STATICFILES_DIRS = [
        'static',
    ]
    STATIC_URL = f'https://{AWS_S3_CUSTOM_DOMAIN}/{AWS_LOCATION}/'
    STATICFILES_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
    STATIC_ROOT = 'https://vrticconnect-bucket.s3.eu-north-
1.amazonaws.com/static/'

    PUBLIC_MEDIA_LOCATION = 'media'
    MEDIA_URL = f'https://{AWS_S3_CUSTOM_DOMAIN}/{PUBLIC_MEDIA_LOCATION}/'
    DEFAULT_FILE_STORAGE = 'diplomski.storage_backends.MediaStorage'
else:
    # Media files
    MEDIA_ROOT = os.path.join(BASE_DIR, 'vrticconnect', 'media')
    MEDIA_URL = 'vrticconnect/media/'

    # Static files (CSS, JavaScript, Images)
    # https://docs.djangoproject.com/en/4.1/howto/static-files/
    STATIC_URL = '/static/'
    STATICFILES_DIRS = [
        os.path.join(BASE_DIR, 'vrticconnect', 'static'),
    ]
```

Programski kod 40. povezivanje aplikacije na Amazon S3 spremnik definirano u settings.py datoteci

```
from storages.backends.s3boto3 import S3Boto3Storage

class MediaStorage(S3Boto3Storage):
    location = 'media'
    file_overwrite = False
    default_acl = 'public-read'
```

Programski kod 41. Proširivanje klase S3Boto3Storage klasom MediaStorage u datoteci storage_backends.py

Nakon uređivanja `settings.py` i `storage_backends.py` datoteka pokrenuta je naredba `python manage.py collectstatic` kako bi se sve statične datoteke prikupile i prenesle u Amazon S3 spremnik.

7. Postavljanje aplikacije na Heroku platformu

Kod postavljanja aplikacije na Heroku platformu korišteni su sljedeći izvori: (Zaczyński n.d.) i (Kaplan-Moss 2016).

Postavljanje aplikacije na Heroku platformu može se izvesti na više načina i za ovaj projekt odabrano je postavljanje pomoću Git repozitorija. U korijenskom direktoriju projekta inicijaliziran je lokalni Git repozitorij naredbom `git init`. U istom direktoriju uređena je datoteka .gitignore u kojoj su navedeni direktoriji i datoteke koji se neće pohranjivati u Git repozitorij. Zatim su u lokalni repozitorij dodane datoteke projekta naredbama `git add .` i `git commit` te je lokalni repozitorij povezan s repozitorijem na servisu GitHub. Nakon povezivanja, lokalni repozitorij prenesen je na Github servis naredbom `git push -u origin master`. Repozitoriju projekta može se pristupiti na adresi https://github.com/markoR-19/Vrticconnect .

Za potrebe postavljanja aplikacije otvoren je korisnički račun na Heroku platformi i odabran je „Eco Dynos“ plan za posluživanje aplikacije. Zatim je instalirano Heroku sučelje za naredbeni redak (eng. Command line interface, CLI) kojim će se aplikacija postaviti na poslužitelj. Putem Heroku CLI-a izvedena je prijava na Heroku platformu i primijenjena je naredba `heroku create vrticconnect` kojom je kreirana aplikacija vrticconnect-66bb4e19c6c4. Prilikom stvaranja aplikacije lokalni Git repozitorij se povezuje i s udaljenim repozitorijem platforme Heroku (na adresi https://git.heroku.com/vrticconnect.git) što će biti potrebno za implementiranje Django projekta u stvorenu Heroku aplikaciju. Za implementaciju Django projekta u Heroku aplikaciju u samom projektu su definirane datoteke `requirements.txt`, `runtime.txt` i `Procfile`. Prilikom postavljanja projekta Heroku instalira sve Python module navedene u datoteci `requirements.txt` koji su potrebni za ispravno funkcioniranje aplikacije i aplikaciju pokreće s verzijom Python programskog jezika navedenom u datoteci `runtime.txt`. U datoteci `Procfile` navedeni su tipovi Dyno kontejnera i naredbe koje oni pokreću prilikom pokretanja aplikacije. Za pokretanje aplikacije „Vrtić connect“ korištena su dva Dyno kontejnera, prvi kontejner je tipa „web“ i pokreće Daphne poslužitelj za ASGI aplikaciju dok je drugi kontejner tipa „worker“ i koristi se za pokretanje potrošača i obradu asinkronih podataka (chat funkcionalnost aplikacije).

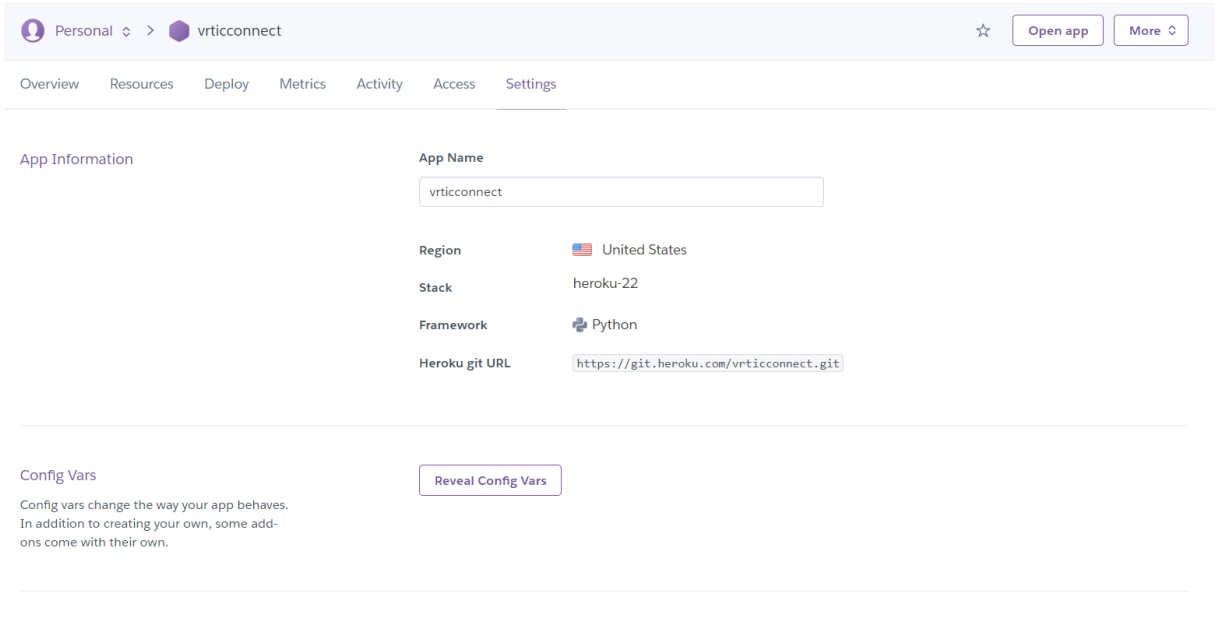
```
web: daphne diplomski.asgi:application --port $PORT --bind 0.0.0.0 -v2
worker: python3 manage.py runworker vrticconnect.consumers.grupaConsumer -v2
```

Programski kod 42. Procfile za pokretanje projekta na Heroku platformi

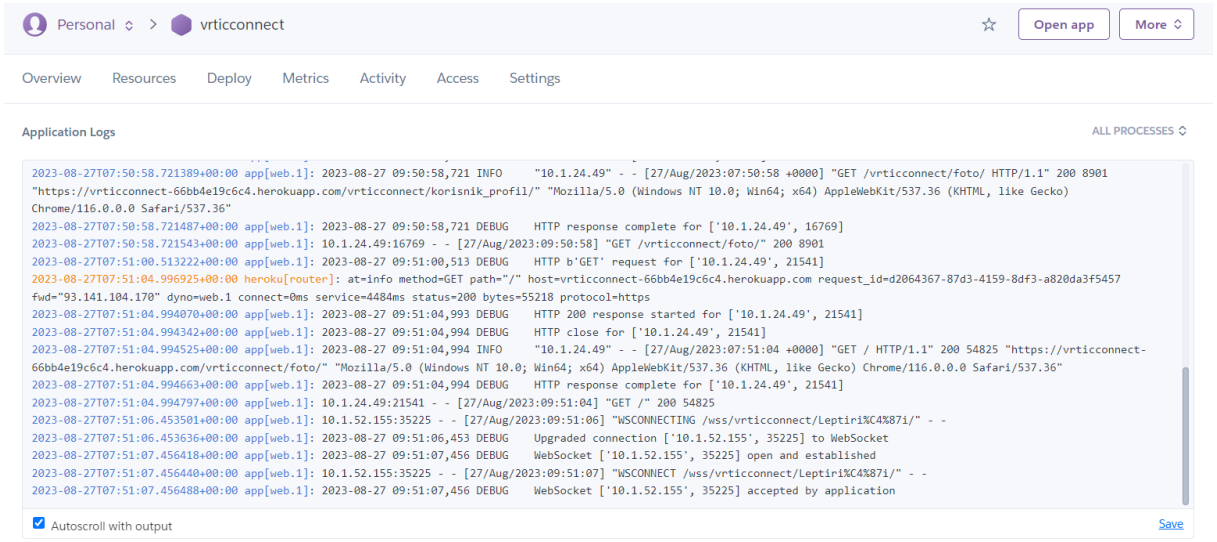
```
channels==4.0.0
daphne==4.0.0
python-dotenv==0.21.0
Django==4.1.7
django-widget-tweaks==1.4.12
Unidecode==1.3.6
psycopg2-binary==2.9.5
Twisted==22.10.0
attrs==23.1.0
boto3==1.28.21
django-storages==1.13.2
django-on-heroku==1.1.2
Pillow==9.0.0
```

Programski kod 43. requirements.txt datoteka s listom Python modula korištenih u projektu

Heroku platforma ne koristi .env datoteke za pohranu varijabli okoline već se svaka varijabla okoline postavlja u aplikaciju pomoću naredbe `heroku config:set`. Nakon postavljanja varijabli okoline u `settings.py` datoteci promijenjena je vrijednost varijable `DEBUG` na `False` kako se prilikom grešaka ne bi prikazivao detaljan opis greške na web stranici. Promjenom varijable `DEBUG` na `False` potrebno je urediti i varijable `ALLOWED_HOSTS` i `CSRF_TRUSTED_ORIGINS`. U varijabli `ALLOWED_HOSTS` definiran je popis domena koje Django aplikacija može posluživati dok je u varijabli `CSRF_TRUSTED_ORIGINS` naveden popis domena od kojih se prihvaćaju CSRF tokeni. Na kraju su promijene primijenjene na lokalni repozitorij i projekt je implementiran na Heroku platformu. Implementiranje projekta izvršeno je prenošenjem lokalnog repozitorija na udaljeni Heroku repozitorij naredbom `git push heroku master` i aplikacija je postavljena na adresu https://vrticconnect-66bb4e19c6c4.herokuapp.com/ (za pregled aplikacije mogu se koristiti korisničko ime „diplomskirad“ i lozinka „Fidit2023!“). Kod implementiranja aplikacije automatski se pokreće samo „web“ Dyno kontejner pa je potrebno primijeniti naredbu `heroku ps:scale web=1:free worker=1:free` kako bi oba Dyno kontejnera bila aktivna.



Slika 27. Opće informacije aplikacije u nadzornoj ploči na Heroku platformi



Slika 28. Prikaz log zapisa u nadzornoj ploči aplikacije

8. Zaključak

U ovom radu prikazan je razvoj aplikacije „Vrtić connect“ pomoću razvojnog okvira Django uz upotrebu HTMX biblioteke i usluga u oblaku Amazon RDS, Amazon S3 i Heroku. U sklopu rada opisan je i način rada razvojnog okvira Django, njegova arhitektura i mogućnosti koje on nudi. Opisane su i funkcionalnosti koje biblioteka HTMX pruža za postizanje modernog i responzivnog korisničkog iskustva. Isto tako objašnjene su AWS usluge u oblaku i platforma Heroku.

U tijeku razvoja aplikacije detaljno su objašnjeni svi koraci razvoja od početnog postavljana projekta do implementiranja aplikacije na web poslužitelju. Svrha aplikacije „Vrtić connect“ je olakšati komunikaciju roditelja i odgojitelja u vrtićkim skupinama pružajući jedinstvenu platformu koja će sadržavati sve informacije vezane uz odgojnu skupinu. Aplikacija omogućava korisnicima komunikaciju preko chata te prikaz obavijesti, nadolazećih aktivnosti i fotografija skupine.

S obzirom na sve prikazano u radu može se zaključiti kako Django razvojni okvir zaista omogućava brz i jednostavan razvoj web aplikacija dok AWS usluge pojednostavljuju upravljanje vanjskim servisima. Također se može uvidjeti kako korištenje usluga Heroku platforme bitno olakšava postavljanje aplikacije na web.

Literatura

n.d. </> *htmx* . Pokušaj pristupa 10. 8 2023. https://htmx.org/.

n.d. *AlpineJS.* Pokušaj pristupa 15. 8 2023. https://alpinejs.dev/.

2023. *Amazon Relational Database Service.* Pokušaj pristupa 12. 8 2023. https://aws.amazon.com/rds/.

2023. *Amazon S3.* Pokušaj pristupa 12. 8 2023. https://aws.amazon.com/s3/.

2023. *Amazon web services.* Pokušaj pristupa 11. 8 2023. https://www.aboutamazon.com/what-we-do/amazon-web-services.

2023. *Applications.* Pokušaj pristupa 12. 8 2023. https://docs.djangoproject.com/en/4.2/ref/applications/.

Asaolu, Elijah. 2021. *Htmx: The newest old way to make web apps.* 4. 6. Pokušaj pristupa 11. 8 2023. https://blog.logrocket.com/htmx-the-newest-old-way-to-make-web-apps/.

2023. *AWS - What is PostgreSQL?* Pokušaj pristupa 3. 8 2023. https://aws.amazon.com/rds/postgresql/what-is-postgresql/.

Barney, Nick. 2022. *Amazon Web Services (AWS).* 10. Pokušaj pristupa 11. 8 2023. https://www.techtarget.com/searchaws/definition/Amazon-Web-Services.

2023. *Boto3 documentation.* Pokušaj pristupa 18. 8 2023. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

Brewster, Cordenne. 2021. *9 Examples of Companies Using Django in 2023.* 18. 3. Pokušaj pristupa 2. 8 2023. https://www.trio.dev/blog/django-applications.

2023. *Cloud computing with AWS.* Pokušaj pristupa 11. 8 2023. https://aws.amazon.com/what-is-aws/.

2023. *Design philosophies.* Pokušaj pristupa 2. 8 2023. https://docs.djangoproject.com/en/dev/misc/design-philosophies/.

n.d. *Django (web framework).* Pokušaj pristupa 2. 8 2023. https://en.wikipedia.org/wiki/Django_(web_framework).

2022. *Django channels - databases.* Pokušaj pristupa 14. 8 2023. https://channels.readthedocs.io/en/stable/topics/databases.html.

2022. *Django channels Tutorial Part 2: Implement a Chat Server.* Pokušaj pristupa 14. 8 2023. https://channels.readthedocs.io/en/stable/tutorial/part_2.html.

n.d. *Django Channels WebSockets Quickstart Tutorial.* Pokušaj pristupa 16. 8 2023. https://appliku.com/post/django-channels-websockets-quickstart-and-deploy.

2023. *Django Documentation - Customizing authentication in Django.* Pokušaj pristupa 14. 8 2023. https://docs.djangoproject.com/en/4.2/topics/auth/customizing/.

2023. *Django Documentation - Databases.* Pokušaj pristupa 13. 8 2023. https://docs.djangoproject.com/en/4.2/ref/databases/#postgresql-notes.

2023. *Django Documentation - Model field reference.* Pokušaj pristupa 12. 8 2023. https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.ImageFiel d.

2023. *Django Documentation - Model instance reference.* Pokušaj pristupa 12. 8 2023. https://docs.djangoproject.com/en/4.2/ref/models/instances/#django.db.models.Model. __str__.

2023. *Django Documentation - Models.* Pokušaj pristupa 3. 8 2023. https://docs.djangoproject.com/en/4.2/topics/db/models/.

2023. *Django Documentation - Pagination.* Pokušaj pristupa 16. 8 2023. https://docs.djangoproject.com/en/4.2/topics/pagination/.

2023. *Django Documentation - Settings.* Pokušaj pristupa 13. 8 2023. https://docs.djangoproject.com/en/4.2/ref/settings/.

2023. *Django Documentation - Templates.* Pokušaj pristupa 3. 8 2023. https://docs.djangoproject.com/en/4.2/topics/templates/.

2023. *Django Documentation - Using the Django authentication system.* Pokušaj pristupa 15. 8 2023. https://docs.djangoproject.com/en/4.2/topics/auth/default/.

2023. *Django Documentation - Working with forms.* Pokušaj pristupa 14. 8 2023. https://docs.djangoproject.com/en/4.2/topics/forms/.

2023. *Django Documentation - Writing your first Django app.* Pokušaj pristupa 12. 8 2023. https://docs.djangoproject.com/en/4.2/intro/tutorial01/.

2023. *Django Documentation - Writing your first Django app, part 2.* Pokušaj pristupa 13. 8 2023. https://docs.djangoproject.com/en/4.2/intro/tutorial02/.

2023. *Django Documentation - Writing your first Django app, part 3.* Pokušaj pristupa 15. 8 2023. https://docs.djangoproject.com/en/4.2/intro/tutorial03/.

2023. *Django Documentation - Writing your first Django app, part 4.* Pokušaj pristupa 14. 8 2023. https://docs.djangoproject.com/en/4.2/intro/tutorial04/.

2023. *Django introduction.* 3. 7. Pokušaj pristupa 2. 8 2023. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction.

2023. *django-storages.* Pokušaj pristupa 18. 8 2023. https://django-storages.readthedocs.io/en/latest/.

2023. *Dynos and the Dyno Manager.* 6. 2. Pokušaj pristupa 10. 8 2023. https://devcenter.heroku.com/articles/dynos.

2023. *EC2.* Pokušaj pristupa 12. 8 2023. https://aws.amazon.com/ec2/.

2023. *FAQ: General.* Pokušaj pristupa 2. 8 2023. https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names.

2023. *Glossary.* Pokušaj pristupa 12. 8 2023. https://docs.djangoproject.com/en/4.2/glossary/#term-project.

2022. *Guide to Django Channels: What it is, pros and cons and use cases.* 7. 12. Pokušaj pristupa 14. 8 2023. https://ably.com/topic/what-is-django-channels#:~:text=Django%20Channels%20consists%20of%20several,prefixing%20to%20determine%20WebSocket%20vs.

Herman, Michael. 2023. *Storing Django Static and Media Files on Amazon S3.* 29. 1. Pokušaj pristupa 15. 8 2023. https://testdriven.io/blog/storing-django-static-and-media-files-on-amazon-s3/.

2023. *Heroku.* Pokušaj pristupa 10. 8 2023. https://www.heroku.com/what#.

2023. *Heroku Dynos.* Pokušaj pristupa 10. 8 2023. https://www.heroku.com/dynos.

Hibbard, James. 2023. *An Introduction to htmx, the HTML-focused Dynamic UI Library.* 8. 8. Pokušaj pristupa 10. 8 2023. https://www.sitepoint.com/htmx-introduction/.

2023. *How Heroku Works.* 8. 3. Pokušaj pristupa 11. 8 2023. https://devcenter.heroku.com/articles/how-heroku-works.

2023. *How to customize Django forms using Django Widget Tweaks ?* 16. 1. Pokušaj pristupa 13. 8 2023. https://www.geeksforgeeks.org/how-to-customize-django-forms-using-django-widget-tweaks/.

n.d. *htmx - Documentation.* Pokušaj pristupa 14. 8 2023. https://htmx.org/docs/.

Jandhyala, Gayatri. 2022. *What is Django ORM?* 2. 9. Pokušaj pristupa 3. 8 2023. https://www.tutorialspoint.com/what-is-django-orm.

Jha, Shyamli. 2023. *What Is AWS? Benefits, Applications of AWS, and More!* 1. 6. Pokušaj pristupa 11. 8 2023. https://www.simplilearn.com/tutorials/aws-tutorial/what-is-aws.

Kaplan-Moss, Jacob. 2016. *Finally, Real-Time Django Is Here: Get Started with Django Channels.* 17. 3. Pokušaj pristupa 18. 8 2023. https://blog.heroku.com/in_deep_with_django_channels_the_future_of_real_time_apps_in_django.

Kaul, Karan. 2023. *What is 'WSGI' & 'ASGI'? — Exploring their Key Differences.* 9. 8. Pokušaj pristupa 14. 8 2023. https://python.plainenglish.io/what-is-wsgi-asgi-exploring-their-key-differences-9c9afc768571.

2023. *MVC.* 8. 6. Pokušaj pristupa 3. 8 2023. https://developer.mozilla.org/en-US/docs/Glossary/MVC.

2023. *PostgreSQL - About.* Pokušaj pristupa 3. 8 2023. https://www.postgresql.org/about/.

2023. *PostgreSQL 15.4 Documentation.* Pokušaj pristupa 12. 8 2023. https://www.postgresql.org/docs/15/.

2021. *psycopg2 - Python-PostgreSQL Database Adapter.* Pokušaj pristupa 2. 8 2023. https://www.psycopg.org/docs/.

n.d. *python-dotenv.* Pokušaj pristupa 10. 8 2023. https://pypi.org/project/python-dotenv/.

Ridvan, Yusuf. 2023. *How to Configure a Django Application with S3 Buckets for File Storage.* 26. 4. Pokušaj pristupa 16. 8 2023. https://blog.devgenius.io/how-to-configure-a-django-application-with-s3-buckets-for-file-storage-9cea315316a4.

2023. *SPA (Single-page application).* 8. 6. Pokušaj pristupa 11. 8 2023. https://developer.mozilla.org/en-US/docs/Glossary/SPA.

2023. *Top 25 AWS Services List 2023.* 28. 3. Pokušaj pristupa 11. 8 2023. https://www.geeksforgeeks.org/top-aws-services/.

2023. *URL Dispatcher.* Pokušaj pristupa 3. 8 2023. https://docs.djangoproject.com/en/4.2/topics/http/urls/.

2023. *webtechsurvey - Django.* 8. Pokušaj pristupa 2. 8 2023. https://webtechsurvey.com/technology/django.

2023. *What is Amazon EC2.* Pokušaj pristupa 11. 8 2023. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

2023. *What is Amazon Relational Database Service (Amazon RDS)?* Pokušaj pristupa 12. 8 2023. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html.

2023. *What is Amazon S3?* Pokušaj pristupa 12. 8 2023. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

n.d. *What Is HTMX.* Pokušaj pristupa 10. 8 2023. https://vegibit.com/what-is-htmx/.

2023. *What is PostgreSQL?* Pokušaj pristupa 3. 8 2023. https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/.

2023. *Why Django?* Pokušaj pristupa 2. 8 2023. https://www.djangoproject.com/start/overview/.

n.d. *WSGI vs ASGI.* Pokušaj pristupa 14. 8 2023. https://www.tutorialspoint.com/python_falcon/python_falcon_wsgi_vs_asgi.htm.

Yasar, Kinza. 2023. *Amazon S3 bucket.* 4. Pokušaj pristupa 12. 8 2023. https://www.techtarget.com/searchaws/definition/AWS-bucket.

Yenigün, Okan. 2022. *Step by Step Django Channels.* 29. 11. Pokušaj pristupa 14. 8 2023. https://levelup.gitconnected.com/step-by-step-django-channels-b17a58141de1.

Zaczyński, Bartosz. n.d. *Hosting a Django Project on Heroku.* Pokušaj pristupa 18. 8 2023. https://realpython.com/django-hosting-on-heroku/.

Popis slika

Popis dijagrama

Popis programskih kodova

Prilozi

1. Web aplikacija Vrtić connect: https://vrticconnect-66bb4e19c6c4.herokuapp.com/
2. Repozitorij projekta na web stranici Github: https://github.com/markoR-19/Vrticconnect