Aplikacija za augmentaciju podataka u oblaku

Đuranić, Dominik

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:260743

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

APLIKACIJA ZA AUGMENTACIJU PODATAKA U OBLAKU

Diplomski rad

Dominik Đuranić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 15.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Dominik Đuranić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1200R, 08.10.2021.
OIB studenta:	62257893270
Mentor:	prof. dr. sc. Marijan Herceg
Sumentor:	,
Sumentor iz tvrtke:	Zvonimir Kaprocki
Predsjednik Povjerenstva:	prof. dr. sc. Mario Vranješ
Član Povjerenstva 1:	prof. dr. sc. Marijan Herceg
Član Povjerenstva 2:	izv. prof. dr. sc. Ratko Grbić
Naslov diplomskog rada:	Aplikacija za augmentaciju podataka u oblaku
Znanstvena grana diplomskog rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Za treniranje dubokih neuronskih mreža potrebna je velika količina podataka kako bi se postigli željeni rezultati i izbjegla pretreniranost (engl. overfitting) mreže. Kako bi se postojeći skupovi podatka koji nemaju dovoljnu količinu podataka proširili, koristi se augmentacija podataka. Augmentacija podataka predstavlja tehniku povećanja količine podataka dodavanjem neznatno promijenjenih kopija postojećih podataka. Npr., za augmentacija slika koriste se različite transformacije kao što su translacija, rotacija, zamućenje, horizontalno zrcaljenje, itd., koje od originalne slike kreiraju neznatno promijenjene slike. U okviru ovog diplomskog rada potrebno je korištenjem Django (Python) web radnog okvira napraviti web aplikaciju koja se izvršava na
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 26.09.2023.

Ime i prezime studenta:	Dominik Đuranić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1200R, 08.10.2021.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za augmentaciju podataka u oblaku**

izrađen pod vodstvom mentora prof. dr. sc. Marijan Herceg

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

2. AUGMENTACIJA PODATAKA 2

2.1. Metode augmentacije slika 2

2.2. Postojeće web aplikacije za augmentaciju podataka 3

3. OPIS KORIŠTENIH TEHNOLOGIJA 8

3.1. Tehnologije na poslužiteljskoj strani 8

3.1.1. Python 8

3.1.2. Django radni okvir 9

3.1.3. Microsoft Azure 12

3.2. Tehnologije na klijentskoj strani 12

3.2.1. Bootstrap radni okvir 12

3.2.2. JavaScript 12

3.2.3. AJAX 13

4. STRUKTURA APLIKACIJE 15

4.1. Navigacijska traka i početna stranica 16

4.2. Registracija i prijava korisnika 19

4.3. Učitavanje skupa podataka na poslužitelj 23

4.4. Odabir i primjena augmentacija 27

4.5. Preuzimanje skupa podataka sa poslužitelja 30

5. TESTIRANJE APLIKACIJE 31

5.1. Verifikacija ispravnog rada aplikacije 31

5.2. Anketa o korištenju aplikacije 37

6. ZAKLJUČAK 41

LITERATURA 42

SAŽETAK 44

ABSTRACT 45

ŽIVOTOPIS 46

PRILOZI 47

1. UVOD

U današnjem digitalnom dobu sve je zastupljenije korištenje strojnog učenja. Jedan od ključnih elemenata u postizanju visokih performansi modela u strojnom učenju je kvalitetan skup podataka na kojemu će se model trenirati. Kako bi modeli strojnog učenja bili točni i generalizirani, potrebno je imati velike i raznovrsne skupove podataka koji dobro reprezentiraju stvarni svijet. Međutim, prikupljanje, označavanje i obrada ovih podataka mogu biti vrlo vremenski i resursno zahtjevni procesi. Augmentacija podataka (engl. *data augmentation*) tada može poslužiti kao način za proširivanje postojećeg skupa podataka. U ovom kontekstu, računarstvo u oblaku (engl. *cloud*) igra ključnu ulogu u omogućavanju obrade velikih skupova podataka. Tako uređaj s kojega korisnici vrše obradu podataka može biti slabih performansi i prenosiv, dok je jedini zahtjev brza internetska veza. Usluge u oblaku pružaju korisnicima mogućnost iznajmljivanja resursa prema potrebi, čime omogućuju učinkovitu obradu podataka bez potrebe za ulaganjem u vlastitu infrastrukturu. Sve više različitih platformi za treniranje i razvoj modela strojnog učenja dostupno je putem usluga u oblaku, olakšavajući korisnicima upravljanje skupovima podataka te stvaranje i optimizaciju modela bez zahtjeva za naprednim tehničkim znanjem. S tim je ciljem razvijena i aplikacija za augmentaciju slikovnih skupova podataka, *Augment.io*, koja je tema ovog diplomskog rada.

Ostatak rada strukturiran je na sljedeći način. U drugom poglavlju približeni su pristupi i metode augmentacije podataka. Analizirane su i uspoređene postojeće aplikacije koje imaju mogućnost augmentacije skupova podataka. U trećem su poglavlju opisane korištene tehnologije za razvoj aplikacije. U četvrtom je poglavlju objašnjen tijek razvoja i prikazana struktura aplikacije *Augment.io*. U petom poglavlju je testirana funkcionalnost aplikacije. Provedeni su samostalni testovi kako bi se potvrdilo da aplikacija radi ispravno, a nakon toga provedeno je istraživanje u obliku ankete s korisnicima aplikacije. Na kraju rada iznesen je zaključak.

2. AUGMENTACIJA PODATAKA

Augmentacija podataka je tehnika u području strojnog učenja koja se koristi za proširivanje skupa podataka prije treniranja modela strojnog učenja. Ova tehnika uključuje primjenu različitih transformacija na postojeće podatke kako bi se stvorile nove varijacije istih uzoraka. Augmentacija podataka je jedna od metoda regularizacije modela strojnog učenja, što znači da pridonosi prevenciji prenaučenosti (engl. *overfitting*) modela, jer proširuje skup podataka i omogućuje modelu generalizaciju na novim, neviđenim uzorcima. Cilj augmentacije je poboljšati učinkovitost i robusnost modela tako da model nauči prepoznati objekte ili uzorke iz različitih perspektiva, položaja, veličina ili s različitim svjetlosnim uvjetima. Može se koristiti i za dodavanje podataka pojedinim klasama kada skup podataka nije balansiran, odnosno kada uzoraka za pojedine klase ima više nego za druge klase. Prema [1, str. 215], za učenje dubokih neuronskih mreža potrebna je velika količina podataka koje je nekada teško ili skupo prikupiti. Jedan od načina rješavanja je augmentacija transformacijama koje ne mijenjaju oznake (engl. *labels*) ulaznih podataka. Kod klasifikacijskih problema bi to značilo da podatak nakon augmentacije mora i dalje pripadati istoj klasi. To se postiže proučavanjem skupa podataka i odabirom transformacija koje neće utjecati na klasu podatka. Npr. kod klasifikacije zrelosti jabuka na slici neće se koristiti transformacije koje utječu na boju. Kod segmentacijskih je problema bitno da se ista transformacija primijeni na sliku i na masku kako bi pomaknuti elementi slike i dalje pripadali istom segmentu.

2.1. Metode augmentacije slika

Dva glavna pristupa za augmentaciju slika prema [2, str. 62] su predobrada i dinamička obrada. Metode obrade koje se koriste su jednake, razlika je u tome kada se radi obrada. Predobrada znači da se skup podataka augmentira i spremi na disk prije početka treniranja mreže, dok dinamička znači da se tijekom treniranja obrađuje ulaz u mrežu. Za aplikaciju *Augment.io* razvijenu u ovom radu korištena je pristup predobrade, gdje se obrađeni skup podataka sa poslužitelja preuzima na disk.

Metoda augmetacije je način na koji će ulazni podatak biti izmijenjen. Uobičajeno se koriste transformacije kao:

- zrcaljenje,
- izrezivanje (engl. *crop*),
- rotiranje,
- smicanje (engl. *shear*),

- translacija,
- dodavanje šuma,
- solarizacija,
- posterizacija.

Naprednije metode se stalno razvijaju. Postoje razne ideje kao što su:

- korištenje generativnih suparničkih mreža (engl. *Generative adversarial networks*) za generiranje novih slika [3],
- izrezivanje ili zamjena dijelova slike [4],
- traženje optimalne kombinacije transformacija.

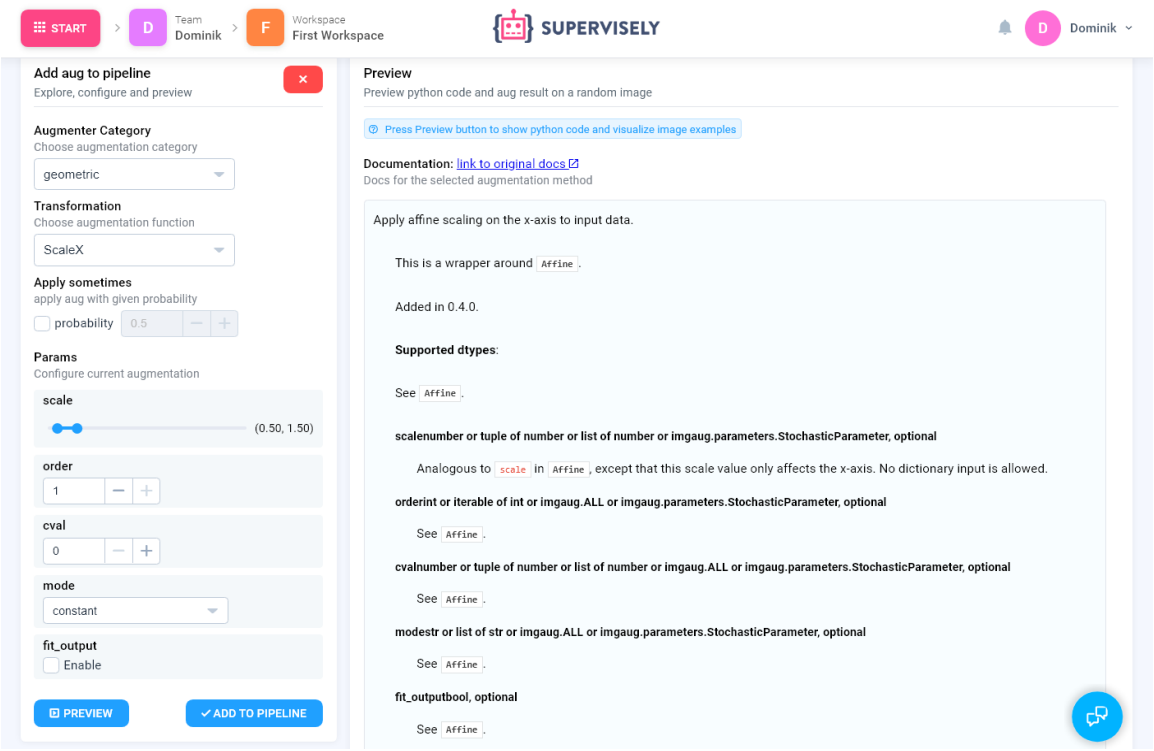
Pošto su naprednije metode često računalno zahtjevne, za implementaciju u aplikaciji *Augment.io* odabrana je metoda *RandAugment* [5] koja koristi samo nasumičnu kombinaciju jednostavnih transformacija uz lako određivanje broja i intenziteta transformacija.

2.2. Postojeće web aplikacije za augmentaciju podataka

Na internetu postoje razne aplikacije i platforme za treniranje modela strojnog učenja, stoga pružaju i pohranu skupova podataka, a neke predobradu i augmentaciju skupa podataka. Većina ih barem pruža uvoz *Python* kôda gdje bi bilo moguće odraditi augmentacija skupa podataka ili urediti slojeve modela da prvi sloj bude sloj za augmentaciju. No, samo su dvije pronađene sa sučeljem za odabir augmentacija: *Supervisely* [6] i *Roboflow* [7].

Supervisely je platforma za anotaciju i razvoj umjetne inteligencije u području računalnog vida. Nazivaju se operacijskim sustavom, no zapravo su ekosustav aplikacija otvorenog kôda koje mogu razvijati sami korisnici. Razne aplikacije omogućuju korisnicima da anotiraju različite vrste podataka, uključujući slike i videozapise, primjenjujući različite vrste oznaka, poput pravokutnika, segmenata, ključnih točaka i tekstualnih oznaka. Dodatno, platforma nudi mogućnost automatske anotacije, čime se ubrzava i pojednostavljuje proces. Korisnici mogu surađivati i timski raditi na anotaciji projekata, olakšavajući dijeljenje podataka i projekata među članovima tima. *Supervisely* je integriran s popularnim alatima za strojno učenje, poput biblioteka *TensorFlow* i *PyTorch*, čime omogućuje korisnicima jednostavno korištenje označenih podataka za treniranje modela strojnog učenja.

Na slici 2.1. je prikazano sučelje za dodavanje transformacije i unos parametara u aplikaciji *ImgAug Studio* [8].



Slika 2.1. Sučelje za odabir augmentacija aplikacije ImgAug Studio [8].

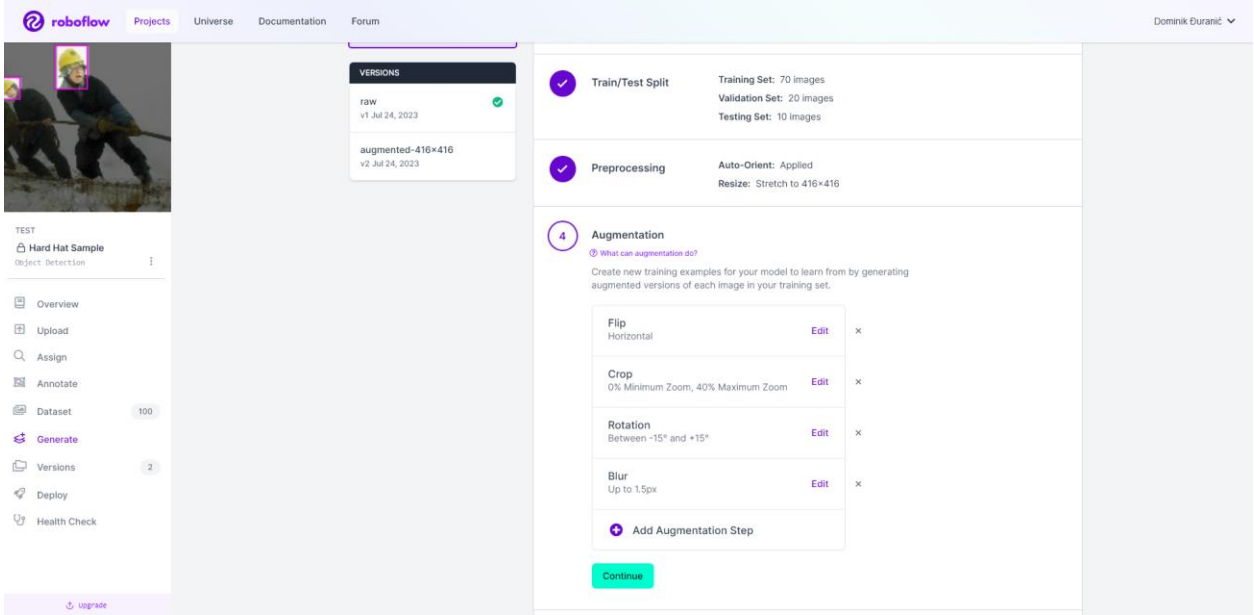
Na slici 2.2 prikazano je sučelje s usporedbom originalne i augmentirane slike, kao i generirani *Python* kôd.

Slika 2.2. generirana slika i Python kôd aplikacije ImgAug Studio [8].

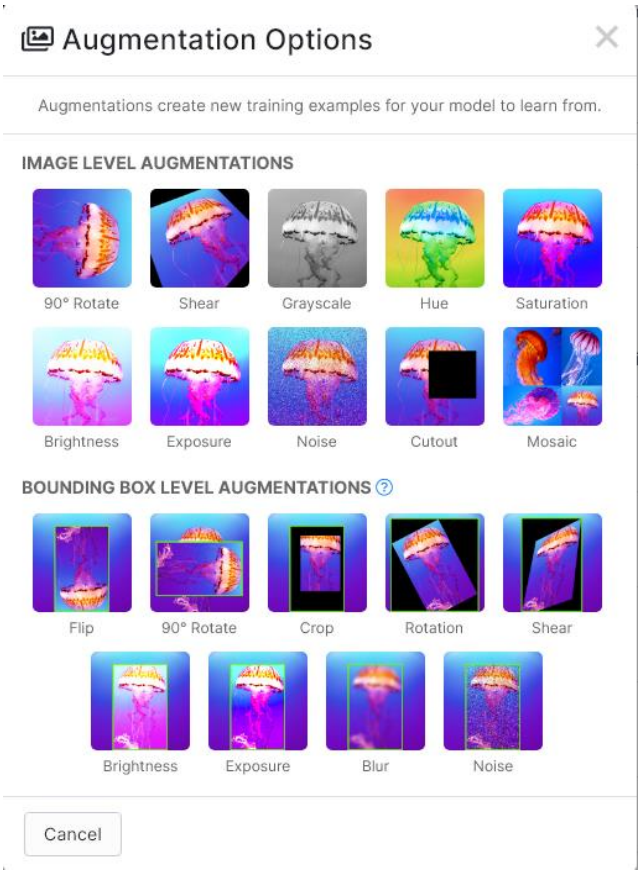
Roboflow je platforma koja se koristi za olakšavanje i ubrzavanje procesa razvoja i implementacije modela za obradu slika, posebno za potrebe strojnog učenja i dubokog učenja. Nudi niz alata i funkcija za prepoznavanje objekata, segmentaciju, detekciju i druge zadatke povezane sa slikama. Njegove ključne značajke uključuju:

- Pretvaranje podataka: Omogućuje obradu i pretvaranje slika i oznaka u različite formate koji su pogodni za treniranje i testiranje modela strojnog učenja.
- Anotacija podataka: omogućuje ručno ili automatsko označavanje objekata na slikama.
- Učenje modela: Platforma omogućuje treniranje vlastitih modela za prepoznavanje uzoraka i objekata koristeći označene skupove podataka ili već postojeće modele.
- Integracija i implementacija: *Roboflow* olakšava integraciju istreniranih modela u aplikacije i web stranice, nudi čak i implementaciju na NVIDIA Jetson platformu.
- Praćenje performansi modela.

Sučelje je intuitivno za korištenje, no neke od funkcionalnosti zahtijevaju dodatno plaćanje. Na slikama 2.3. i 2.4. prikazano je sučelje za dodavanje nove augmentacije.

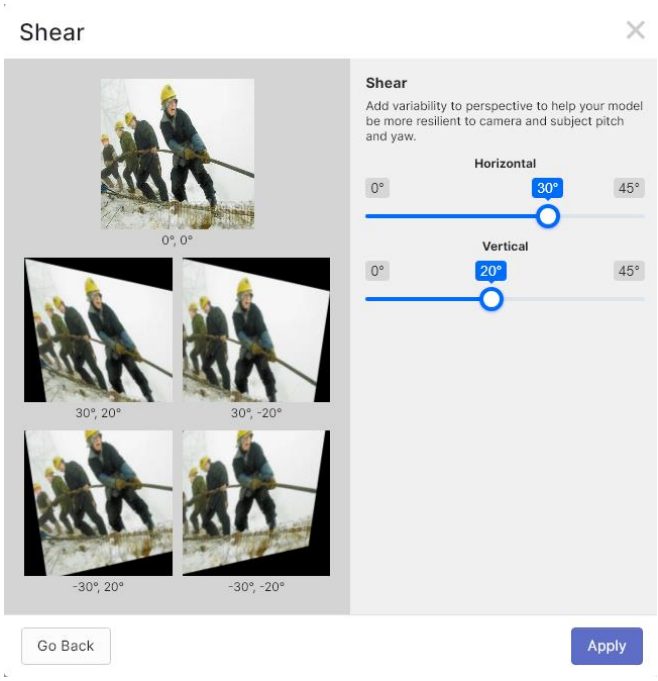


Slika 2.3. Dodavanje koraka augmentacije u aplikaciji Roboflow [7].

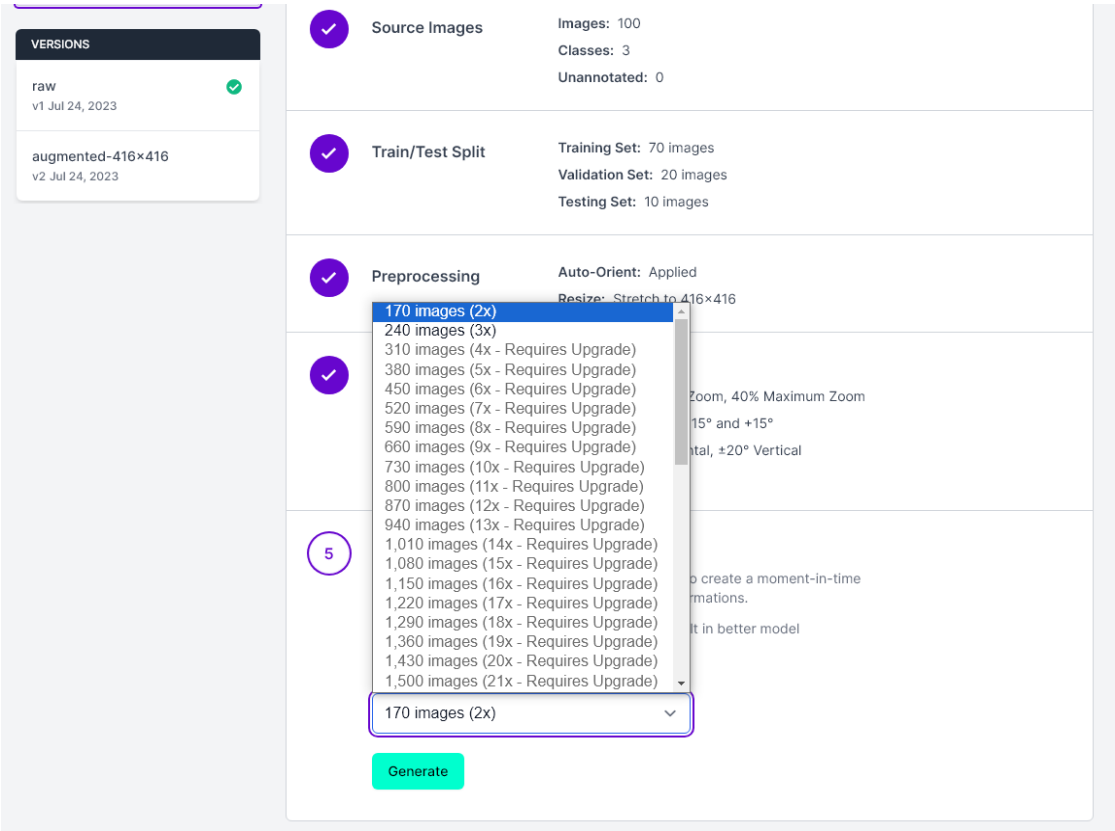


Slika 2.4. Odabir augmentacije u aplikaciji Roboflow [7].

Na slici 2.5. prikazan je pretpregled odabrane augmentacije i odabir parametara. Na slici 2.6. prikazan je odabir broja slika koje će se generirati.



Slika 2.5. Pretpregled i odabir parametara augmentacije u aplikaciji Roboflow [7].



Slika 2.6. Odabir broja generiranih slika u aplikaciji Roboflow [7].

Aplikacije *Roboflow* i *ImgAug Studio* rade s označenim podacima i zadržavaju lokaciju oznaka nakon augmentacije slika. To znači da ako je na slici pravokutnikom (engl. *bounding box*) označena pozicija nekog objekta, i za augmentiranu sliku će biti poznata pozicija pravokutnika. Aplikacija *Augment.io* opisana u ovom radu nema podršku za tako označene podatke, nego samo za slike podijeljene u direktorije po klasama kojim pripadaju ili za skupove podataka za nenadzirano učenje gdje slike nisu označene.

3. OPIS KORIŠTENIH TEHNOLOGIJA

3.1. Tehnologije na poslužiteljskoj strani

3.1.1. Python

Python je objektno-orijentirani, interpretirani programski jezik visoke razine apstrakcije. *Python* je postao izuzetno popularan zbog svoje raznolike primjene, što uključuje web razvoj, znanstveno računanje, strojno učenje, automatizaciju, i zbog toga što je otvorenog kôda i prenosiv. Prenosiv je zato što su interpreteri napisani za razne platforme. Korištenje interpretera omogućuje brzo prototipiranje i interakciju jer se kôd može pokrenuti čim je napisan, bez prevoditelja (engl. *compiler*) koji cijeli program mora prevesti u strojni jezik kako bi se mogao pokrenuti. Sintaksa je jednostavna i slična engleskom jeziku te je često potrebno manje linija kôda za isti program napisan u ostalim programskim jezicima. Kako bi bio čitljiviji daje prednost riječima, uvlakama i novim redcima pred znakovima. Primjer je prikazan na slici 3.1. gdje se kod grananja i petlji koriste uvlake umjesto zagrada, riječi za logičke operacije te kako nema oznaka za kraj retka. *Python* je modularan što znači da se oslanja na razne biblioteke (engl. *libraries*) za obavljanje specifičnih zadataka, a u ovome radu se koriste neke poput *NumPy, imgaug, ZipFile, PIL* i drugih. U ovome radu korištena je *Python* verzija 3.10.11.

Linija ***Kôd***

```
1:      if file.endswith('.jpg') or file.endswith('.png'):
2:          # Open the image using PIL
3:          image =np.array( Image.open(file_path))
4:
5:          for i in range(numberOfImages):
6:              # Apply the augmentations
7:              augmented_image = seq.augment_image(image)
8:              augmented_image_pil = Image.fromarray(augmented_image)
9:              # Save the augmented image in the same directory
10:             augment_path = os.path.join(subdir, f"augment_{i}{file}")
11:             augmented_image_pil.save(augment_path, format='PNG')
```

Slika 3.1. Primjer grananja i petlji s uvučenim blokovima.

Od naprednijih značajki *Pythona* se u ovome radu koriste dekoratori i rukovanje iznimkama. Dekoratori u *Pythonu* su funkcije unutar kojih se poziva željena funkcija [9]. Time se dobije omotač preko kojega se može utjecati na tok izvođenja na način da se ovisno o nekim parametrima

poziva funkcija ili obavlja neki drugi zadatak. Unutar dekoratora je moguće rukovati podacima prije ili nakon poziva funkcije. Tako *Django* radni okvir [10] za web razvoj pruža dekorator koji ovisno o tome je li korisnik prijavljen prikazuje željenu stranicu ili preusmjerava korisnika na stranicu za prijavu. Korištenjem dekoratora postiže se čišći izgled poziva funkcija jer je dekorator često definiran u različitoj datoteci. Na slici 3.2. prikazan je primjer sintakse dekoratora sa znakom „@“, a na slici 3.3 je prikazan ispis pokrenutog primjera.

Linija ***Kôd***

```
1:      def mojDekorator(func):
2:          def omotac():
3:              print("Prije poziva funkcije")
4:              func()
5:              print("Nakon poziva funkcije")
6:          return omotac
7:
8:      @mojDekorator
9:      def pozdrav():
10:         print("Pozdrav svijete!")
11:
12:     pozdrav()
```

Slika 3.2. Primjer dekoratora.

```
PS C:\Users\Dominik\Desktop\Diplomski\primjeri> & "C:/Program Fi
les (x86)/Microsoft Visual Studio/Shared/Python36_64/python.exe"
 c:/Users/Dominik/Desktop/Diplomski/primjeri/primjer.py
Prije poziva funkcije
Pozdrav svijete!
Nakon poziva funkcije
```

Slika 3.3. Ispis poziva funkcije s dekoratorom u terminalu.

3.1.2. Django radni okvir

Django je popularan web radni okvir (engl. *framework*) za razvoj web aplikacija napisan u programskom jeziku *Python*. Njegov glavni cilj je olakšati i ubrzati proces razvoja web aplikacija kroz ponovno korištenje koda, automatsko upravljanje administrativnim dijelom aplikacije, pružanje velikog broja funkcionalnosti i stroge sigurnosne mjere. To što je jednostavan za korištenje ne znači da je ograničen, što je razlog da ga koriste velike stranice kao što su:

- *Instagram,*
- *Mozilla,*
- *Pinterest,*
- *OpenStack,*
- *National Geographic* [11].

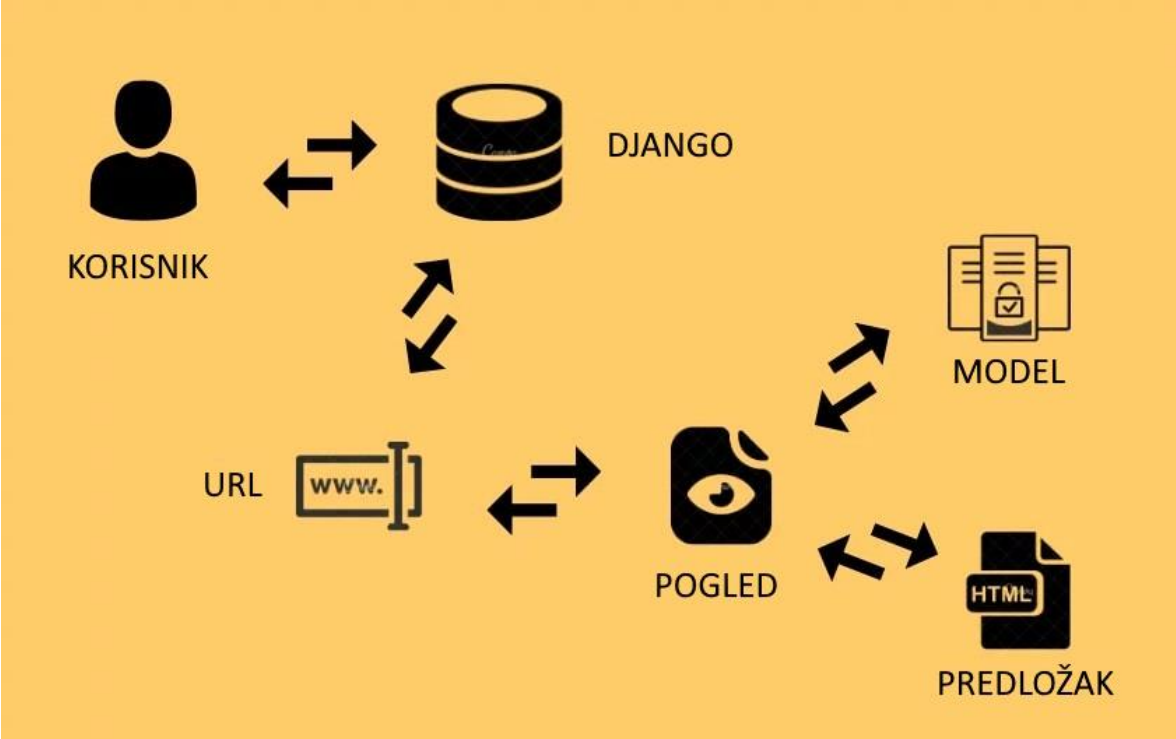
Kôd aplikacije na poslužiteljskoj strani se piše u *Pythonu* pa će imati opisane značajke iz prethodnog poglavlja, no *Django* donosi značajke kao što su:

- URL (engl. *Uniform Resource Locator*) usmjeravanje,
- objektno-relacijsko mapiranje (engl. *Object-relational mapper*),
- administrativno sučelje,
- upravljanje formama,
- generiranje CSRF (engl. *Cross-Site Request Forgery*) tokena,
- modeli, pogledi (engl. *Views*) i predlošci (engl. *Templates*),
- jezik koji se koristi unutar predložaka DTL (engl. *Django Template Language*).

URL usmjeravanje znači da *Django* povezuje *Python* funkcije s poveznicama. Objektno-relacijsko mapiranje znači da je preko *Python* klasa moguće definirati kako će baza podataka biti strukturirana, a *Django* će se pobrinuti za generiranje baze i upita prema bazi. Tako definirani modeli bit će vidljivi unutar administrativnog sučelja koje *Django* generira. Upravljanje formama znači da *Django* može generirati formu za unos podataka prema definiranom modelu. To je korisno jer će forma na klijentskoj strani automatski raditi provjeru unesenih polja zadovoljavaju li ograničenja definiranih polja modela kao što su maksimalna duljina, nedozvoljeni znakovi i sl. Što se sigurnosti tiče, za svaki HTTP (engl. *Hypertext Transfer Protocol*) POST zahtjev se mora generirati CSRF token kako bi se unos podataka zaštitio od zahtjeva s drugih stranica kojima je vrijednost tokena nepoznata. Već spomenuti modeli i funkcije koje se pozivaju prilikom odlaska na pojedini URL pripadaju MVT (engl. *Model-View-Template*) arhitekturi koju *Django* koristi. MVT arhitektura je slična MVC (engl. *Model-View-Controller*) arhitekturi koja se često koristi za razvoj web stranica u radnim okvirima kao *Spring* za *Javu* [12], *ASP.NET MVC* za *C#* [13] ili *Rails* za *Ruby* [14]. MVT arhitektura sadrži:

1. *Model*: Model predstavlja sloj podataka u *Django* aplikaciji. Ovdje se definiraju klase koje opisuju strukturu podataka u bazi podataka. *Django* automatski stvara SQL tablice na temelju ovih modela i omogućuje komunikaciju s bazom podataka koristeći *Python* objekte pomoću objektno-relacijskog mapiranja. *Django* uključuje bazu *SQLite* koja se po potrebi može zamijeniti.
2. *View*: Pogledi u *Django* radnom okviru predstavljaju poslovnu logiku aplikacije. Svaki pogled je *Python* funkcija koja obrađuje HTTP zahtjev i vraća HTTP ili JSON odgovor. Pogledi određuju kako će se podaci prikazati korisniku i obavljaju potrebne akcije na temelju korisničkih zahtjeva.
3. *Template*: Predlošci se koriste za oblikovanje izlaza koji generiraju pogledi. Oni predstavljaju HTML (engl. *Hypertext Markup Language*) datoteke s ugrađenom DTL sintaksom. Predlošci omogućuju programerima odvajanje dizajna od logike, čime se olakšava izrada dinamičkih HTML stranica koje se temelje na podacima iz baze podataka.

Na slici 3.4. prikazana je shema MVT arhitekture i interakcija s korisnikom.



Slika 3.4. Shema MVT arhitekture [15].

3.1.3. Microsoft Azure

Azure je Microsoftova platforma za računarstvo u oblaku. Namijenjena je za izgradnju, testiranje, razvoj i upravljanje aplikacijama i uslugama preko mreže podatkovnih središta. Pruža različite usluge poput softver kao usluga (engl. *Software as a Service*), platforma kao usluga (engl. *Platform as a Service*) i infrastruktura kao usluga (engl. *Infrastructure as a service*). *Azure* podržava širok spektar programskih jezika, alata i razvojnih okruženja. U ovome je radu korištena usluga platforme, konkretno servis za aplikaciju (engl. *App service*) [16].

3.2. Tehnologije na klijentskoj strani

3.2.1. Bootstrap radni okvir

Bootstrap je besplatni razvojni okvir otvorenog koda za izradu korisničkog sučelja web stranica i web aplikacija. Kao razvojni okvir, *Bootstrap* uključuje osnove responzivnog web razvoja, tako da programeri trebaju samo umetnuti kod u unaprijed definirani sustav rešetke s 12 stupaca [17]. Responzivnost znači da se sadržaj i raspored stranice prilagođava širini ekrana na kojemu se prikazuje. Okvir *Bootstrap* izgrađen je na HTML, CSS (engl. *Cascading Style Sheets*) i JavaScriptu. Koristeći *Bootstrap* moguće je puno brže izraditi web stranice bez trošenja vremena na definiranje osnovnih stilova i funkcija. Korištenjem *Bootstrap* radnog okvira elementima je potrebno samo dodijeliti odgovarajuće klase i biti će primijenjeni stilovi i funkcije.

3.2.2. JavaScript

JavaScript je dinamički programski jezik. Lagan je i najčešće se koristi kao dio web stranica, čije implementacije omogućuju interakciju korisnika i skripte na klijentskoj strani te za stvaranje dinamičnih stranica. Interpretirani je programski jezik s mogućnostima objektno orijentiranog programiranja [18]. Najčešće se koristi za:

- validaciju podataka na strani klijenta,
- dinamičke padajuće izbornike,
- prikazivanje vremena i datuma,
- prikazivanje skočnih prozora,
- ažuriranje dijelova stranice bez ponovnog učitavanja,
- događaje i animacije na prelazak miša, klik i slično.

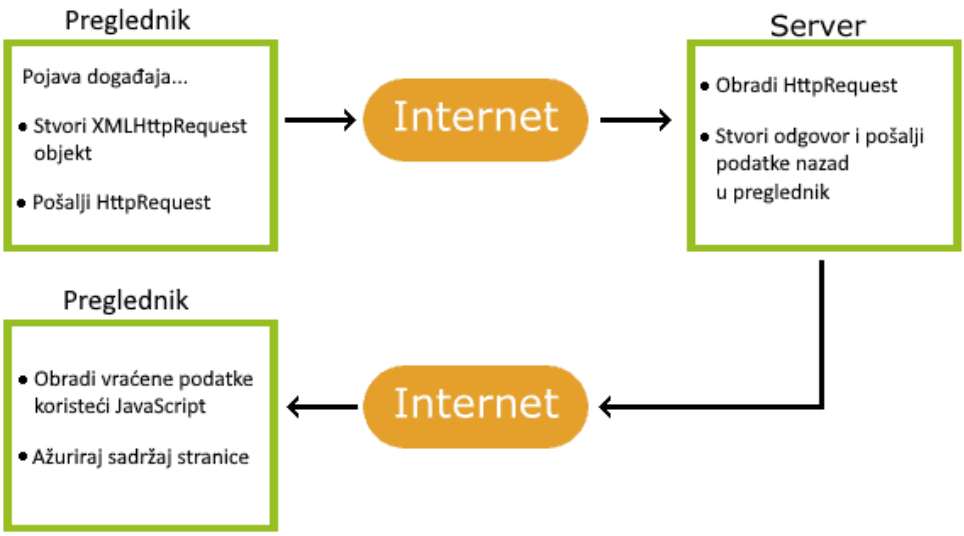
Osim za web stranice može se koristiti i u okruženjima koja nisu unutar preglednika, kao što su *NodeJS* za poslužitelje, *CouchDB* za baze podataka i *Adobe Acrobat*, alat za PDF dokumente [19].

3.2.3. AJAX

AJAX (engl. *Asynchronous JavaScript and XML*) je tehnologija koja omogućuje asinkrono slanje podataka i izmjenu sadržaja između web preglednika i web poslužitelja bez potrebe za ponovnim učitavanjem cijele web stranice. Koristi kombinaciju *JavaScripta*, HTML-a, CSS-a i HTTP za interakciju s web poslužiteljem u pozadini i dinamičko ažuriranje sadržaja na web stranici. Glavne karakteristike i prednosti korištenja AJAX-a su:

- Asinkrono slanje podataka: Podaci se šalju i primaju asinkrono, što znači da korisnik može nastaviti koristiti web stranicu bez prekida tijekom komunikacije s poslužiteljem.
- Dinamičko ažuriranje: Web stranica može ažurirati samo određeni dio sadržaja bez potrebe za ponovnim učitavanjem cijele stranice, što poboljšava brzinu i korisničko iskustvo.
- Bolja interaktivnost: Korisnici mogu obavljati radnje na web stranici bez osvježavanja, kao što su provjera obrazaca, slanje poruka i učitavanje novih podataka
- Manje opterećenje poslužitelja: Pošto se samo određeni dijelovi stranice ažuriraju, zahtjevi za poslužitelj su manji u usporedbi s klasičnim postupkom učitavanja cijele stranice.
- Podrška različitim formatima podataka: Iako naziv sugerira korištenje XML-a, AJAX može koristiti i druge formate podataka poput JSON-a.

Tijek AJAX zahtjeva prikazan je na slici 3.5.



Slika 3.5. Tijek AJAX zahtjeva [20].

AJAX se može koristiti u čistom JavaScriptu, no zbog jednostavnosti je korištena popularna biblioteka *jQuery* [21]. AJAX poziv koristeći *jQuery* je prikazan na slici 3.6.

Linija ***Kôd***

```
1:      $.ajax(
2:          {
3:          type: "POST",
4:          url: "/url",
5:          data: {
6:          'someData':'ABC'
7:          },
8:          success: function () {
9:              console.log("Success");
10:         },
11:         error: function(){
12:             console.log("Error");
13:         }
14:     });
```

Slika 3.6. AJAX POST zahtjev koristeći jQuery.

4. STRUKTURA APLIKACIJE

U ovomu će poglavlju biti opisan tijek razvoja aplikacije *Augment.io* i pojašnjena struktura *Django* projekta. Kompletan programski kôd aplikacije nalazi se u prilogu P.4.1. na elektroničkom mediju priloženom uz ovaj rad. Na početku je napravljeno virtualno okruženje te su instalirane sve potrebne *Python* biblioteke. Stvaranje *Django* projekta izvodi se naredbom u terminalu:

---

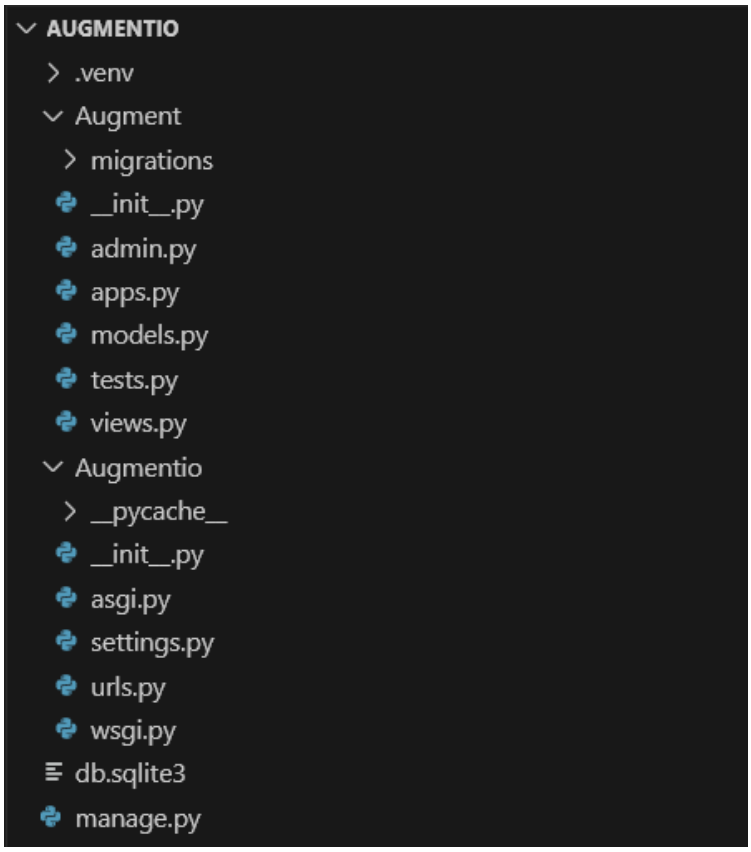
django-admin startproject Augmentio

Zatim je potrebno napraviti aplikaciju unutar projekta. Aplikacija pod nazivom *Augment* je stvorena naredbom:

---

py manage.py startapp Augment

Struktura projekta nakon izvršavanja naredbi je prikazana na slici 4.1.



Slika 4.1. Struktura stvorenog projekta.

U projekt je još prije početka razvoja aplikacije potrebno u aplikaciju *Augment* dodati direktorij pod nazivom *templates* gdje će se spremati predlošci za pojedine dijelove aplikacije.

Također je potrebno navesti poveznice na kojemu će pojedini pogled biti dostupan. To se definira u datoteci *urls.py* koja je prikazana na slici 4.2. Za svaku poveznicu poziva se funkcija *path* s argumentima poveznica, funkcija pogleda i ime.

Linija ***Kôd***

```
1:      from django.urls import path
2:      from . import views
3:
4:      urlpatterns = [
5:          path('',views.home, name='home'),
6:          path('register/', views.registerPage, name='register'),
7:          path('login/', views.loginPage, name='login'),
8:          path('upload/',views.uploadDataset, name='upload'),
9:          path('augment/', views.selectAugmentations, name='augment'),
10:         path('augmentDataset/', views.augmentDataset,
      name='augmentDataset'),
11:         path('zip/', views.zipDataset, name='zip'),
12:         path('download/', views.download, name='download'),
13:         path('delete/', views.deleteDataset, name='delete'),
14:         path('logout/', views.logoutUser, name='logout'),
15:     ]
```

Slika 4.2. Kôd datoteke urls.py.

Svi pogledi nakon *loginPage* zaštićeni su dekoratorom *@login_required* koji je konfiguriran tako da ako korisnik pokuša u adresnu traku unijeti poveznicu do pogleda koji zahtjeva prijavu, a korisnik nije prijavljen preusmjeri se na stranicu za prijavu.

4.1. Navigacijska traka i početna stranica

Za izradu navigacijske trake stvoren je novi predložak *main.html* unutar direktorija *templates.* Kako bi ista navigacijska traka bila prikazana na svakoj stranici aplikacije korišteno je nasljeđivanje predložaka koje pruža *Django*. Primjer strukture glavnog predloška s navigacijom i uključivanjem potrebnih skripti prikazan je na slici 4.3. Na slici 4.3. u linijama 9, 19 i 21 vidljiva je sintaksa *Django* jezika za predloške DTL. Koristeći DTL moguće je definirati blokove koji će se zamijeniti sadržajem u predlošcima koji nasljeđuju glavni predložak. Blok se započinje naredbom *{% block <naziv_bloka> %}*, a završava naredbom *{% endblock <naziv_bloka> %}*.

Linija ***Kôd***

```
1:      !DOCTYPE html>
2:      <html lang="en">
3:      {% load static %}
4:          <head>
5:              <meta charset="UTF-8">
6:              <meta http-equiv="X-UA-Compatible" content="IE=edge">
7:              <meta name="viewport" content="width=device-width,
     initial-scale=1.0">
8:
9:              <title>{% block title %}{% endblock title %}</title>
10:
11:             <!-- Uključivanje svih potrebnih skripti i stilova:
     jQuery, Bootstrap i CSS
12:             <script src=""></script> -->
13:         </head>
14:         <body>
15:             <nav class="navbar navbar-expand-lg navbar-dark"
                  style="background-color: #424242;">
16:                 <!-- Navigacijska traka -->
17:             </nav>
18:
19:             {% block content %}
20:
21:             {% endblock content %}
22:
23:         </body>
24:     </html>
```

Slika 4.3. Primjer kôda predloška main.html kojega će naslijediti svi ostali predlošci.

Blokovi moraju biti imenovani kako bi se sadržaj mogao točno smjestiti. Blok *title* će u predlošku koji nasljeđuje biti zamijenjen naslovom stranice, npr. „*Home*“, dok će blok *content* biti zamijenjen sadržajem cijele stranice.

Definirane su dvije verzije navigacijske trake: kada je korisnik prijavljen i kada nije. Na slikama 4.4. i 4.5. je prikazana navigacijska traka u oba slučaja.

Slika 4.4. Navigacijska traka kada korisnik nije prijavljen.

Slika 4.5. Navigacijska traka kada je korisnik prijavljen.

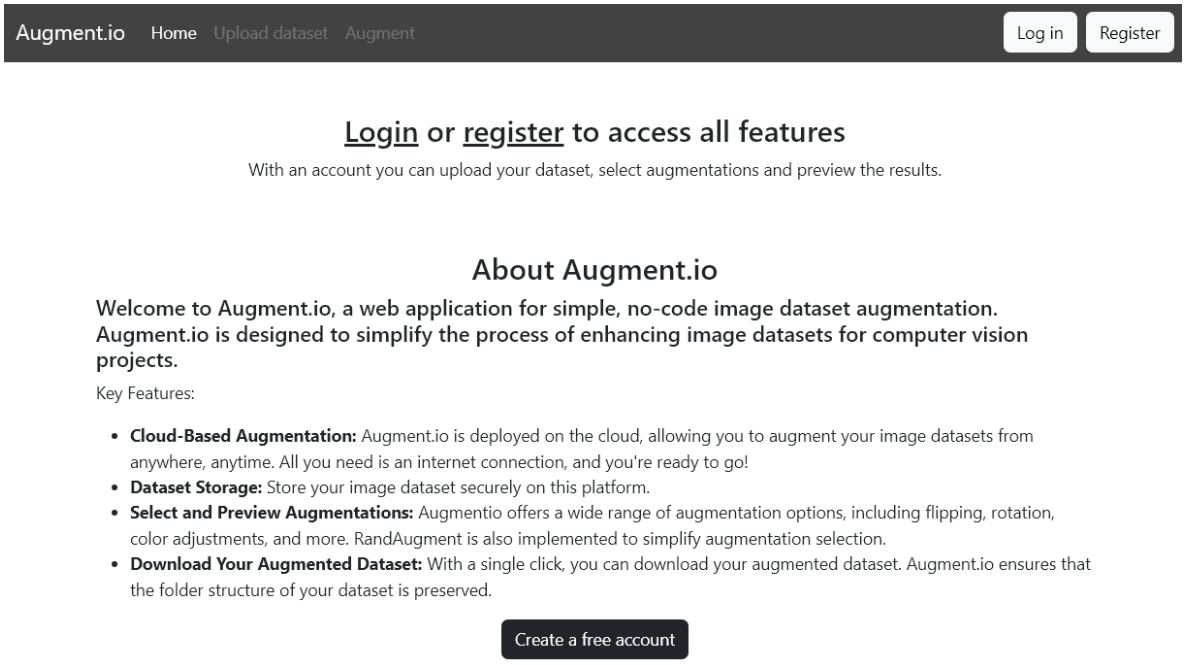
Kada korisnik nije prijavljen prikazane su tipke za prijavu i registraciju, dok kada je prijavljen prikazana je samo tipka za odjavu. Navigacija za učitavanje i za augmentaciju skupa podataka je onemogućena kada korisnik nije prijavljen. Provjera je li korisnik prijavljen se izvodi koristeći DTL i ugrađenu *Django* klasu *User* koja sadrži atribut *is_authenticated*. Provjera korisnika prikazana je na slici 4.6.

Linija ***Kôd***

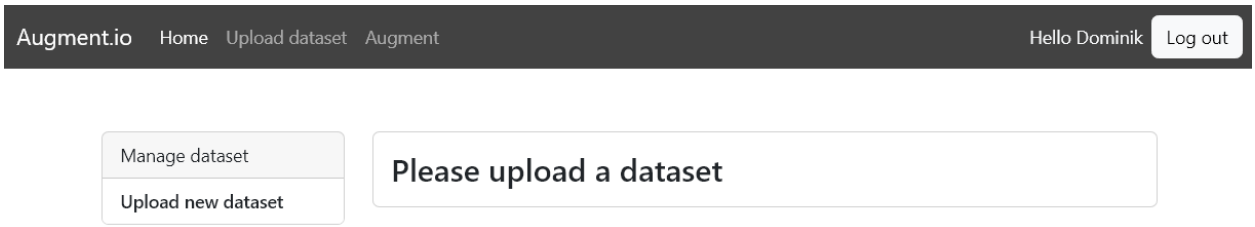
```
1:      {% if user.is_authenticated %}
2:      <div class="text-light me-2">Hello {{user}}</div>
3:      <a href="{% url 'logout' %}" class="btn btn-light" >Log out</a>
4:      {% else %}
5:      <a href="{% url 'login' %}" class="btn btn-light me-2" >Log in</a>
6:      <a href="{% url 'register' %}" class="btn btn-light" >Register</a>
7:      {% endif %}
```

Slika 4.6. Kôd za provjeru je li korisnik prijavljen pomoću DTL.

Na jednak je način napravljena provjera korisnika na početnoj stranici. Tako su korisniku koji nije prijavljen na početnoj stranici prikazane informacije o stranici i obavijest kako se mora prijaviti da bi koristio funkcionalnosti aplikacije, dok je prijavljenom korisniku dostupan dio za upravljanje skupom podataka. Usporedba sadržaja početne stranice ovisno o tome je li korisnik prijavljen prikazana je slikama 4.7. i 4.8.



Slika 4.7. Početna stranica kada korisnik nije prijavljen.



Slika 4.8. Početna stranica kada je korisnik prijavljen.

4.2. Registracija i prijava korisnika

Za upravljanje korisnicima korištena je ugrađena *Django* klasa *User*. Bilo je potrebno je izraditi dva predloška, *login.html* i *register.html* i spremiti ih u direktorij *templates*. Pošto se stranice za registraciju i prijavu koriste samo za predaju forme, koristiti će se *Django forms*. Za definiranje vlastitih formi stvorena je datoteka *forms.py* unutar aplikacije *Augment*. U datoteci *forms.py* definirane su dvije klase: *LoginUserForm* i *CreateUserForm*. Kôd definiranih klasa prikazan je na slikama 4.9. i 4.10.

Za dodjeljivanje odgovarajućih HTML klasa korišten je *Django widget* koji predstavlja *Django* reprezentaciju HTML elementa za unos (engl. *input*) [22]. Klase je potrebno dodijeliti radi stila kojim upravlja *Bootstrap*. Kada su tako definirane forme izrada predloška postaje jednostavna. Kôd predloška za prijavu prikazan je na slici 4.11. Na slici je vidljivo da je forma samo učitana pomoću DTL-a u liniji 14, a predana je predlošku unutar pogleda. Vidljive su HTML klase potrebne za primjenu *Bootstrap* stilova, kao što su *container, mt-5* za gornju marginu, *col-md-6* za definiranje koliko stupaca rešetke će zauzimati *div* element kada je širina ekrana veća od prijelomne točke za *md* klasu. To znači da će na srednje velikim ekranima zauzimati 6 od 12 stupaca, a na manjim ekranima će zauzimati punu širinu ekrana. Na slici je ujedno i vidljivo kako se nasljeđuju predlošci, koristi se naredba *extends* te se definiraju blokovi *title* i *content* koji će biti umetnuti na odgovarajuća mjesta u spomenutom glavnom predlošku *main.html*.

Linija ***Kôd***

```
1:      class LoginUserForm(forms.Form):
2:          username=forms.CharField(max_length=50)
3:          password=forms.CharField(widget=forms.PasswordInput)
4:
5:          model=User
6:          fields=['username', 'password']
7:          def __init__(self, *args, **kwargs):
8:              super(LoginUserForm, self).__init__(*args, **kwargs)
9:
10:             self.fields['username'].widget.attrs['class'] = 'form-
    control'
11:             self.fields['username'].widget.attrs['placeholder'] =
    'Username'
12:             self.fields['password'].widget.attrs['class'] = 'form-
    control'
13:             self.fields['password'].widget.attrs['placeholder'] =
    'Password'
```

Slika 4.9. Definicija klase LoginUserForm.

Linija ***Kôd***

```
1:      class CreateUserForm(UserCreationForm):
2:          class Meta:
3:              model=User
4:              fields = ['username', 'email', 'password1', 'password2']
5:
6:          def __init__(self, *args, **kwargs):
7:              super(CreateUserForm, self).__init__(*args, **kwargs)
8:
9:              self.fields['username'].widget.attrs['class'] = 'form-
    control'
10:             self.fields['username'].widget.attrs['placeholder'] =
    'Username'
11:             self.fields['email'].widget.attrs['class'] = 'form-
    control'
12:             self.fields['email'].widget.attrs['placeholder'] = 'Email'
13:             self.fields['password1'].widget.attrs['class'] = 'form-
    control'
14:             self.fields['password1'].widget.attrs['placeholder'] =
    'Password'
15:             self.fields['password2'].widget.attrs['class'] = 'form-
    control'
16:             self.fields['password2'].widget.attrs['placeholder'] =
    'Confirm password'
```

Slika 4.10. Definicija klase CreateUserForm.

Linija ***Kôd***

```
1:      {% extends 'main.html' %}
2:      {% load static %}
3:      {% block title %} {{ page }} {% endblock title %}
4:
5:      {% block content %}
6:
7:          <div class="container mt-5">
8:
9:              <div class="row justify-content-center">
10:                 <div class="col-md-6">
11:                     <h1>Login</h1>
12:                     <form action="" method="POST">
13:                         {% csrf_token %}
14:                         {{form.as_p}}
15:
16:                         <button type="submit" class="btn btn-dark">Log
     in</button>
17:                         <div>{{ error_message }}</div>
18:                     </form>
19:                 </div>
20:             </div>
21:         </div>
22:
23:     {% endblock content %}
```

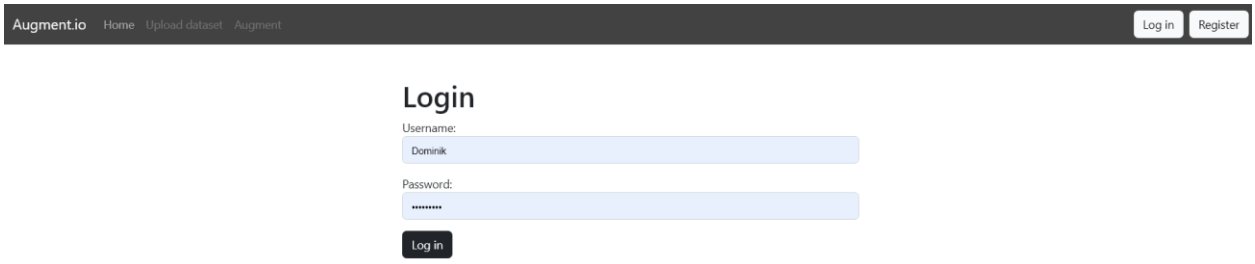
Slika 4.11. Kôd predloška za prijavu.

Unutar pogleda u datoteci *views.py* će se stvoriti objekt klase forme i staviti u rječnik (engl. *dictionary*) kontekst koji se koristi u predlošku pomoću DTL sintakse *{{<naziv_u_rječniku>}}*. Pogled za prijavu prikazan je na slici 4.12. U liniji 2 na slici 4.12. stvoren je objekt klase *LoginUserForm,* koji se sprema u rječnik u liniji 12 ili 15, ovisno je li HTTP zahtjev POST ili GET. HTTP zahtjev je GET kada korisnih dođe na stranicu, a POST kada korisnik klikne tipku za prijavu i pošalje se ispunjena forma. Predložak i pogled stranice za registraciju su jako slični predlošku i pogledu za prijavu. Izgled stranice za registraciju i stranice za prijavu prikazan je na slikama 4.13. i 4.14.

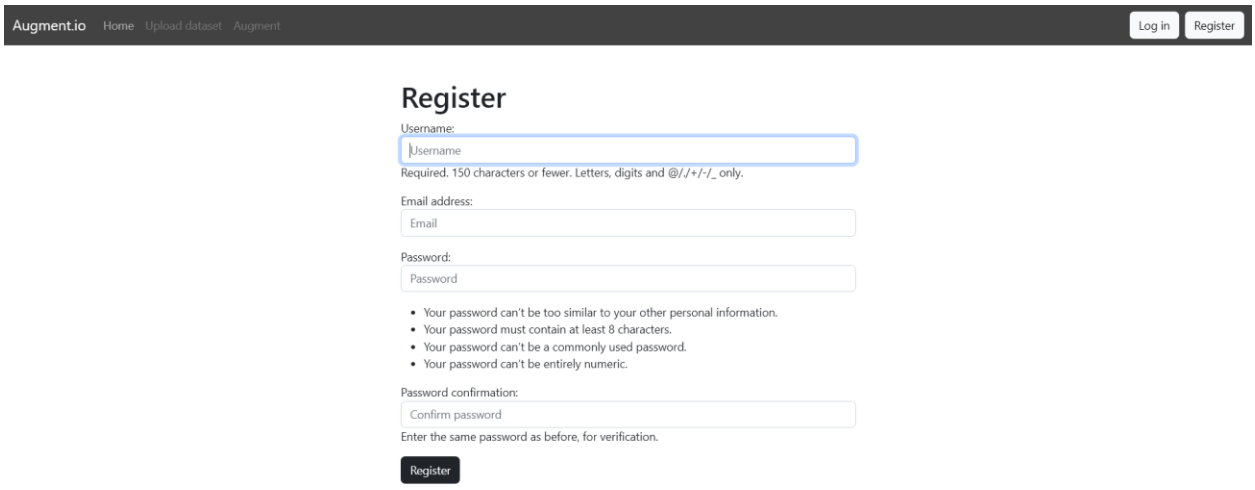
Linija ***Kôd***

```
1:      def loginPage(request):
2:          form=LoginUserForm()
3:          if request.method=="POST":
4:              username = request.POST['username']
5:              password = request.POST['password']
6:              user = authenticate(request, username=username,
      password=password)
7:
8:              if user is not None:
9:                  login(request, user)
10:                 return redirect("home")
11:             else:
12:                 context={'page':'Login', 'error_message': 'Username or
      password entered was incorrect', 'form': form}
13:                 return render(request, "photos/login.html", context)
14:
15:         context={'page':'Login', 'form': form}
16:         return render(request, "photos/login.html", context)
```

Slika 4.12. Kôd pogleda za prijavu u views.py dokumentu.



Slika 4.13. Izgled stranice za prijavu.



Slika 4.14. Izgled stranice za registraciju.

4.3. Učitavanje skupa podataka na poslužitelj

Za praćenje skupova podataka bilo je potrebno kreirati model *Dataset*. Odabran je odnos s korisnikom 1 prema 1, što znači da jedan korisnik može imati samo jedan skup podataka učitan na poslužitelju, a skup podataka pripada jednom korisniku. Naravno, pružena je i funkcionalnost učitavanja novog skupa gdje se stari skup podataka briše sa poslužitelja. U aplikaciju je dodana datoteka *models.py*, s definiranim modelom *Dataset*, prikazana na slici 4.15.

Linija ***Kôd***

```
1:      from django.db import models
2:      from django.contrib.auth.models import User
3:
4:      class Dataset(models.Model):
5:          user = models.OneToOneField(User, on_delete=models.CASCADE)
6:          name = models.CharField(max_length=100)
7:          creationDate = models.DateField(auto_now_add=True)
8:
9:          def __str__(self):
10:             return self.name
```

Slika 4.15. Model Dataset u datoteci models.py.

Nakon dodavanja modela potrebno je pokrenuti dvije naredbe u terminalu kako bi se promjene primijenile u bazi podataka. Te naredbe su:

py manage.py makemigrations

py manage.py migrate

Nakon primijenjenih promjena stvoren je novi predložak *upload.html.* Na slici 4.16. prikazan je dio kôda predloška *upload.html* bez *Bootstrap* stupaca i *containera* radi bolje preglednosti.

Linija ***Kôd***

```
1:      {% if hasDataset %}
2:      <div class="warning mb-3">
3:          <img class="float-left" src="{% static 'warning.png' %}"
      alt="Dataset warning">
4:          You already have an uploaded dataset. Uploading a new dataset
      will delete the existing dataset.
5:      </div>
6:      {% endif %}
7:      <h2>Upload new dataset</h2>
8:      <form method="post" enctype="multipart/form-data">
9:          {% csrf_token %}
10:         <div class="form-group">
11:             <div class="mb-3">
12:                 <label class="form-label" for="zip_file">Select your
      zipped dataset</label>
13:                 <div class="input-group custom-file-button">
14:                     <label class="input-group-text"
      for="zip_file">Choose .zip file</label>
15:                     <input class="form-control" type="file"
      name="zip_file" id="zip_file" placeholder="Select .zip file">
16:                 </div>
17:             </div>
18:         </div>
19:         <button class="btn btn-dark m-1" type="submit"
      id="submitButton">Upload</button>
20:     </form>
21:     <div class="progress my-3" style="display: none;">
22:         <div id="bar" class="progress-bar" role="progressbar"
      style="width: 0%;" aria-valuemin="0" aria-valuemax="100">0%</div>
23:     </div>
24:     <div id="success" class="my-3" style="display: none;">
25:         message
26:     </div>
```

Slika 4.16. Dio kôda predloška upload.html.

U predlošku je definirana forma za odabir datoteke. Klikom na tipku *Upload* bi se uobičajeno odmah poslao POST zahtjev i osvježila bi se stranica. Pošto je potrebno provjeriti da je korisnik odabrao datoteku ispravnog formata potrebno je koristeći *JavaScript*. Na slici 4.17. prikazana je funkcija *validateFile* za provjeru formata odabrane datoteke, a na slici 4.18. postavljanje događaja na tipku *Upload*, sprječavanje slanja POST zahtjeva i poziv funkcije *validateFile*.

Linija ***Kôd***

```
1:      function validateFile() {
2:          var fileInput = document.getElementById('zip_file');
3:          var file = fileInput.files[0];
4:          var fileType = file.type;
5:          $("#submitButton").prop("disabled", true)
6:
7:          if (fileType !== 'application/zip' && fileType !==
      'application/x-zip-compressed') {
8:              $("#submitButton").prop("disabled", false)
9:              alert('Please select a ZIP file.');
10:             return false;
11:         }
12:         else{
13:             return true;
14:         }
15:     }
```

Slika 4.17. Funkcija validateFile koja provjerava je li odabrana zip datoteka.

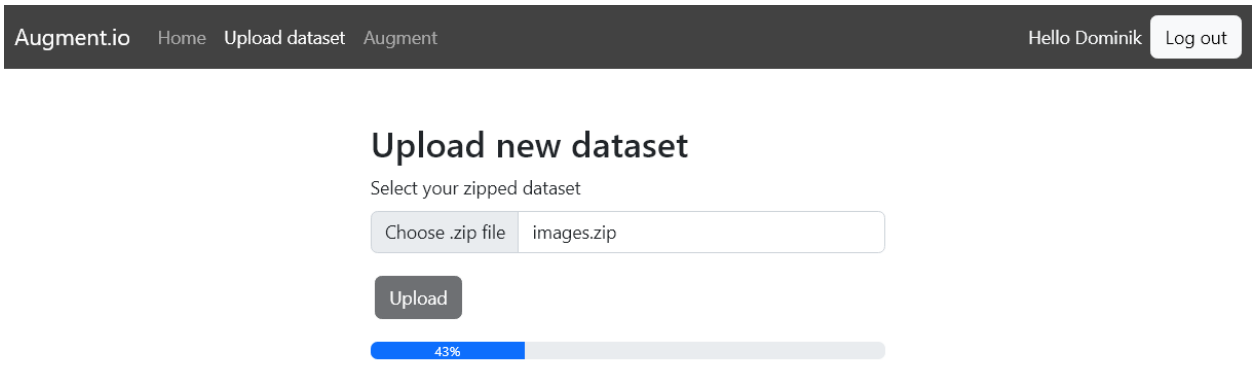
Linija ***Kôd***

```
1:      $("#submitButton").click(function(e) {
2:          e.preventDefault();
3:          if(validateFile()){
4:              var fileInput = document.getElementById("zip_file");
5:              var file = fileInput.files[0];
6:              var formData = new FormData();
7:              formData.append("zip_file", file);
8:              formData.append("csrfmiddlewaretoken",'{{ csrf_token }}');
9:              $.ajax({ //ajax poziv})
```

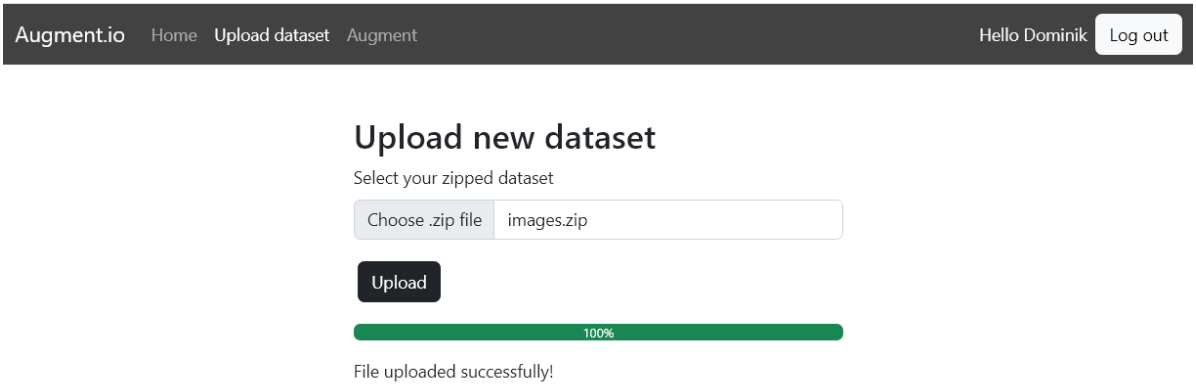
Slika 4.18. Događaj na pritisak tipke Upload i poziv funkcije validateFile.

Time se postiglo ponašanje da ako je odabrana datoteka koja nema nastavak *.zip* izbaciti će se upozorenje da je potrebno odabrati ispravnu datoteku. Umjesto uobičajenog POST zahtjeva koristi se asinkroni pomoću AJAX poziva zato što bi učitavanje datoteke moglo potrajati neko vrijeme, a inače ne bismo imali povratnu informaciju nego bi samo preglednik prikazivao učitavanje stranice. Korištenjem AJAX poziva moguće je računati koliki je dio datoteke poslan i prikazivati traku napretka te po završetku učitavanja poruku. Na poslužitelju će se sadržaj učitane datoteke raspakirati u korisnički direktorij, a ako raspakiranje ne uspije vratiti će grešku da nije učitana ispravna arhiva. Do toga može doći ako se neka druga datoteka preimenuje s nastavkom *.zip* ili ako je datoteka oštećena. Tijekom učitavanja tipka *Upload* je onemogućena kako korisnik ne bi slučajno prekinuo trenutno učitavanje koje je u tijeku i započeo novo. AJAX poziv s računanjem postotka učitavanja nalazi se u prilogu P.4.2. Postotak učitavanja skaliran je do 80 umjesto do 100. Razlog tomu je što će dio vremena zauzeti i raspakiravanje arhive na poslužitelju koje je po procjeni oko 20% vremena. Unutar AJAX poziva se osvježava postotak trake napretka, a po završetku mijenja boja u zeleno ako je uspješno učitavanje, ili u crveno ako datoteka nije uspješno raspakirana. Prikazuje se i poruka ovisno o rezultatu.

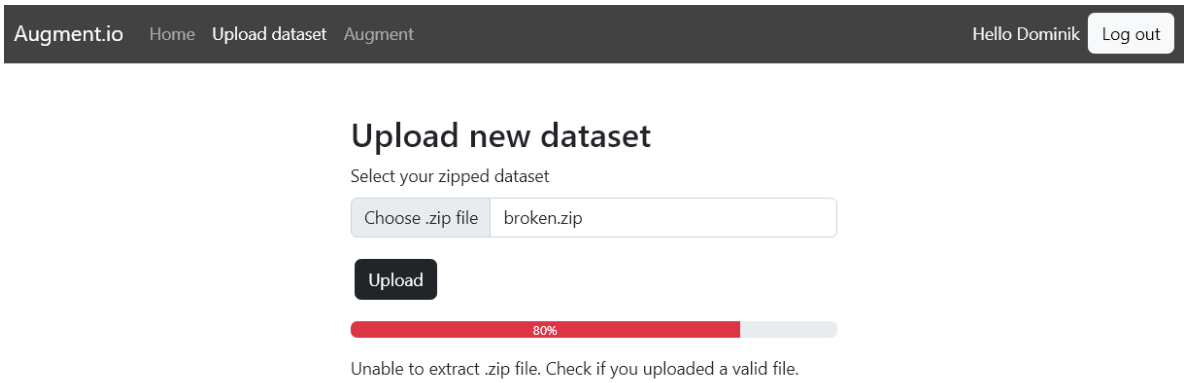
Na slikama 4.19. i 4.20. prikazana je traka napretka u različitim stanjima. Na slici 4.19. je u stanju učitavanja, a na slici 4.20. je prikazano uspješno učitavanje i neuspješno učitavanje.



Slika 4.19. Traka napretka tijekom učitavanja datoteke.



(a)



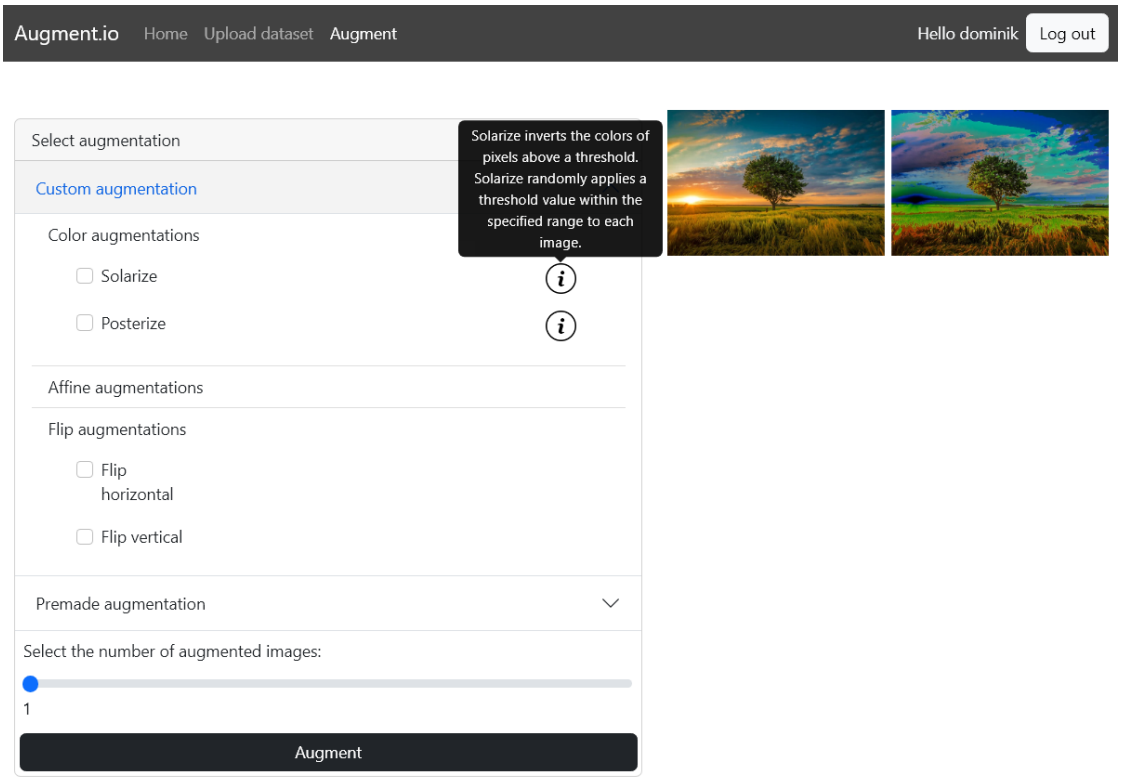
(b)

Slika 4.20. (a) Uspješno učitavanje datoteke; (b) Neuspješno učitavanje datoteke.

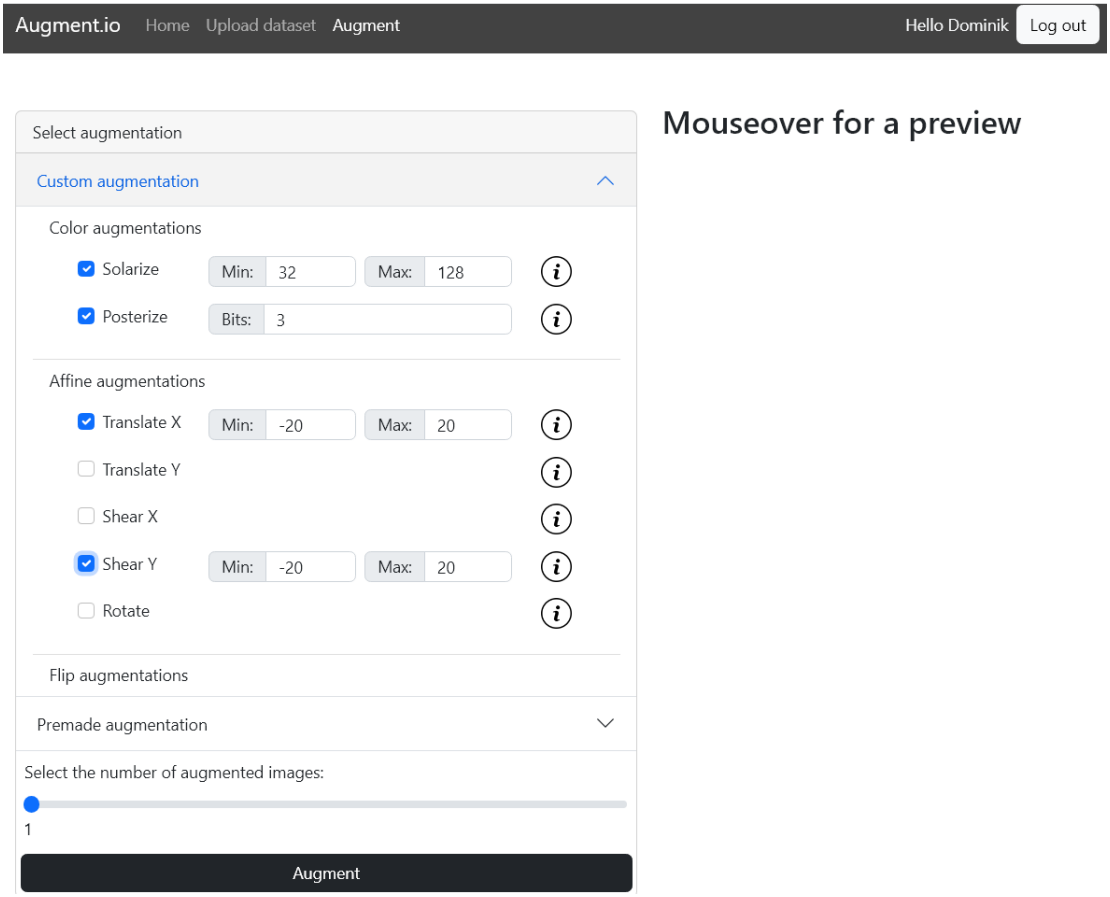
4.4. Odabir i primjena augmentacija

Sučelje za odabir augmentacija je dosta dinamično i interaktivno. Ideja je bila pružiti informacije korisnicima bez da pregled bude zagušen pa se većina informacija pojavljuje na prelazak mišem ili na klik. Za postizanje toga korištene su *Bootstrap* klase poput *collapse*, *accordion* te *tooltip* [23] za informacije na prelazak mišem (ili dodir na mobilnim uređajima). Na slikama 4.21., 4.22. i 4.23. prikazana je interakcija sa sučeljem.

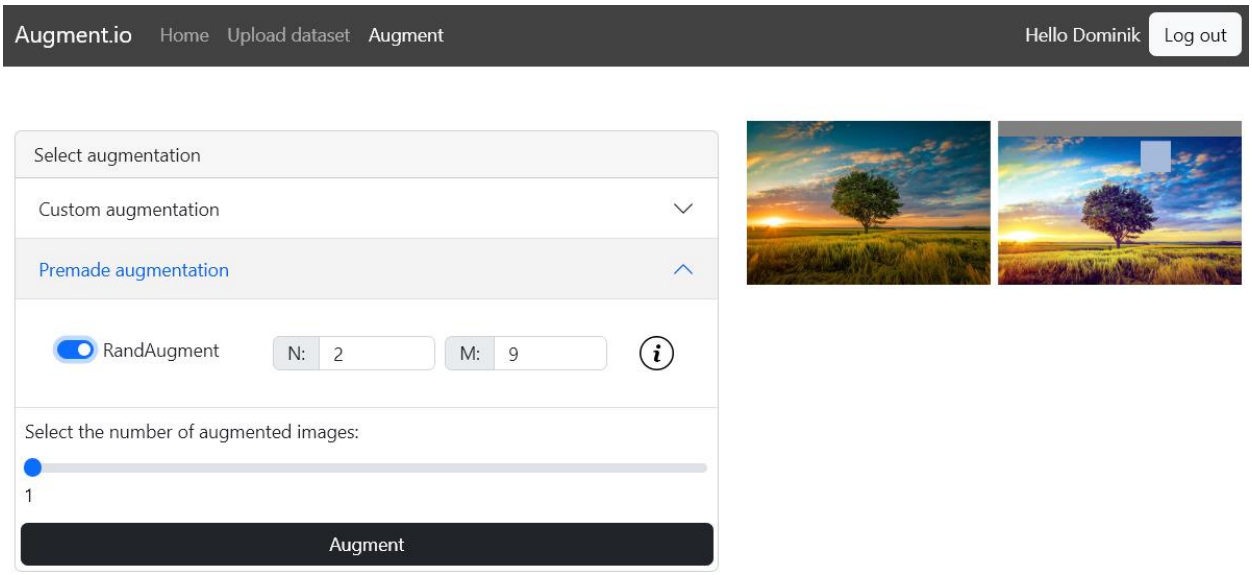
Nakon što su odabrane željene augmentacije sadržaj forme se šalje u obliku POST zahtjeva. Učitava se novi predložak sa kojeg AJAX pozivom započinje obrada slika na poslužitelju. Unutar pogleda se ovisno o odabranim augmentacijama popunjava lista *augmenters*. Taj je dio pogleda *augmentDataset* prikazan u prilogu P.4.3.



Slika 4.21. Prijeđeno mišem preko ikonice za informacije.



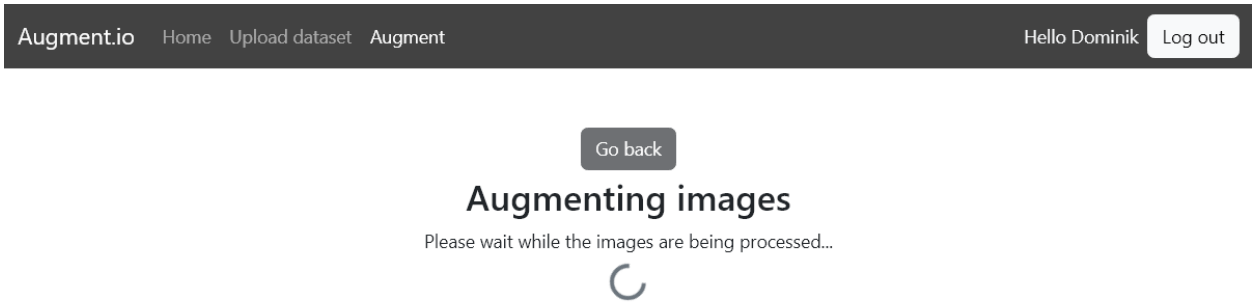
Slika 4.22. Označene augmentacije, pojava unosa parametara.



Slika 4.23. Odabrana druga stavka harmonike (engl. accordion).

Zatim se stvara objekt klase *Sequential* iz biblioteke *imgaug* od popunjene liste *augmenters.* To znači da će odabrane augmentacije biti primijenjene jedna za drugom na istoj slici. Zatim je potrebno obrisati stari augmentirani skup ako postoji te kopirati skup podataka u korisnički direktorij u kojemu će biti augmentirani skup. Nakon toga se prolazi po stablu direktorija te, zadržavajući strukturu direktorija, obrađuje i sprema augmentirana slika s novim nazivom. Taj se dio pogleda *augmentDataset* nalazi u prilogu P.4.4.

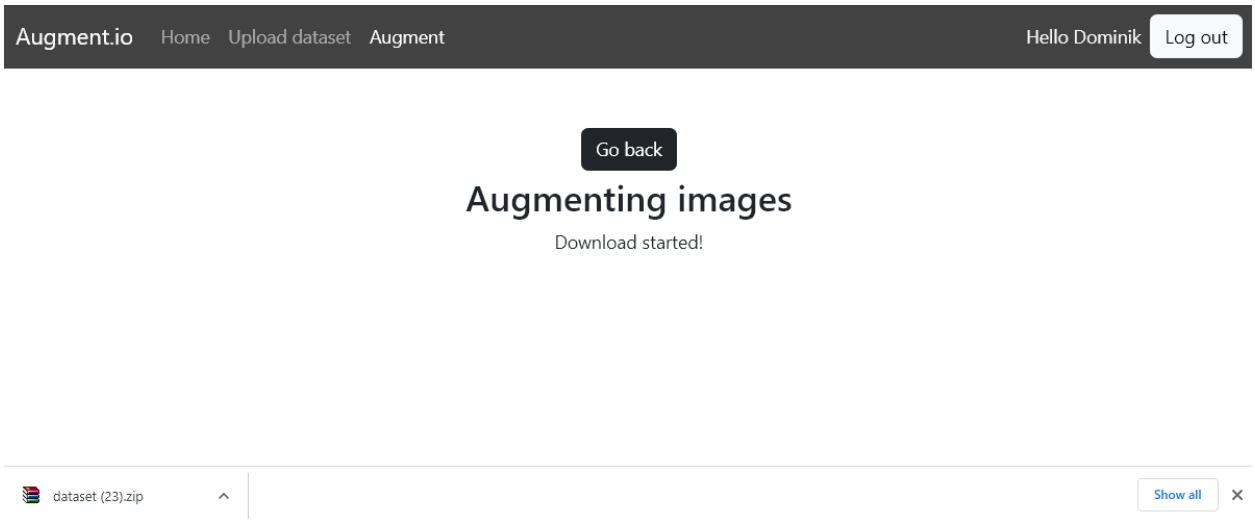
Tijekom obrade korisniku je prikazana poruka da se slike obrađuju i prikazuje se animacija za učitavanje. Ako slučajno dođe do greške, prikazati će se informacija korisniku kako bi pokušao ponovo pokrenuti augmentaciju. Izgled stranice tijekom obrade prikazan je na slici 4.24.



Slika 4.24. Izgled stranice tijekom obrade skupa podataka.

4.5. Preuzimanje skupa podataka sa poslužitelja

Nakon što je augmentacija uspjela sa iste se stranice kao i u prošlom potpoglavlju šalje novi AJAX zahtjev za sažimanje skupa podataka. Kada je sažimanje završeno automatski se započinje preuzimanje *.zip* datoteke skupa podataka unutar preglednika. Preuzimanje je prikazano na slici 4.25.



Slika 4.25. Automatski započeto preuzimanje augmentiranog skupa podataka.

5. TESTIRANJE APLIKACIJE

5.1. Verifikacija ispravnog rada aplikacije

Za verifikaciju da aplikacija ispravno primjenjuje augmentacije korišten je podskup skupa GTSRB (engl. *German Traffic Sign Recognition Benchmark*) [24]. Iz skupa podataka su izdvojene tri klase: 12 (cesta s prednošću prolaska), 13 (raskrižje s cestom s prednošću prolaska), 14 (obavezno zaustavljanje). Za svaku klasu su izabrane 3 slike. Na slici 5.1. prikazana je struktura direktorija i poddirektorija rezultirajućeg skupa podataka.

```
└──train
      ├──12
      │       00012_00011_00029.png
      │       00012_00025_00028.png
      │       00012_00054_00029.png
      │
      ├──13
      │       00013_00014_00021.png
      │       00013_00016_00026.png
      │       00013_00020_00025.png
      │
      └──14
              00014_00013_00017.png
              00014_00017_00025.png
              00014_00018_00024.png
```

Slika 5.1. Struktura direktorija i poddirektorija skupa za verifikaciju rada aplikacije.

Metodologija verifikacije ispravnog rada aplikacija je 12 puta izvršiti augmentaciju opisanog skupa s različitim parametrima i samostalno utvrditi da su augmentacije pravilno primijenjene. Prvih 10 puta su redom odabrane po jedna augmentacija. 11. put je odabrana jedna augmentacija i 3 kao broj augmentiranih slika. Zadnji je test odabrati više augmentacija odjednom.

Od opisanog skupa podataka stvorena je arhiva *train.zip* koja je učitana u aplikaciju. Odabrana je augmentacija „Translate X“ s parametrima (-20, 20). Odabrano je 1 za broj augmentiranih slika. Na slici 5.2. prikazana je struktura augmentiranog skupa preuzetog sa poslužitelja.

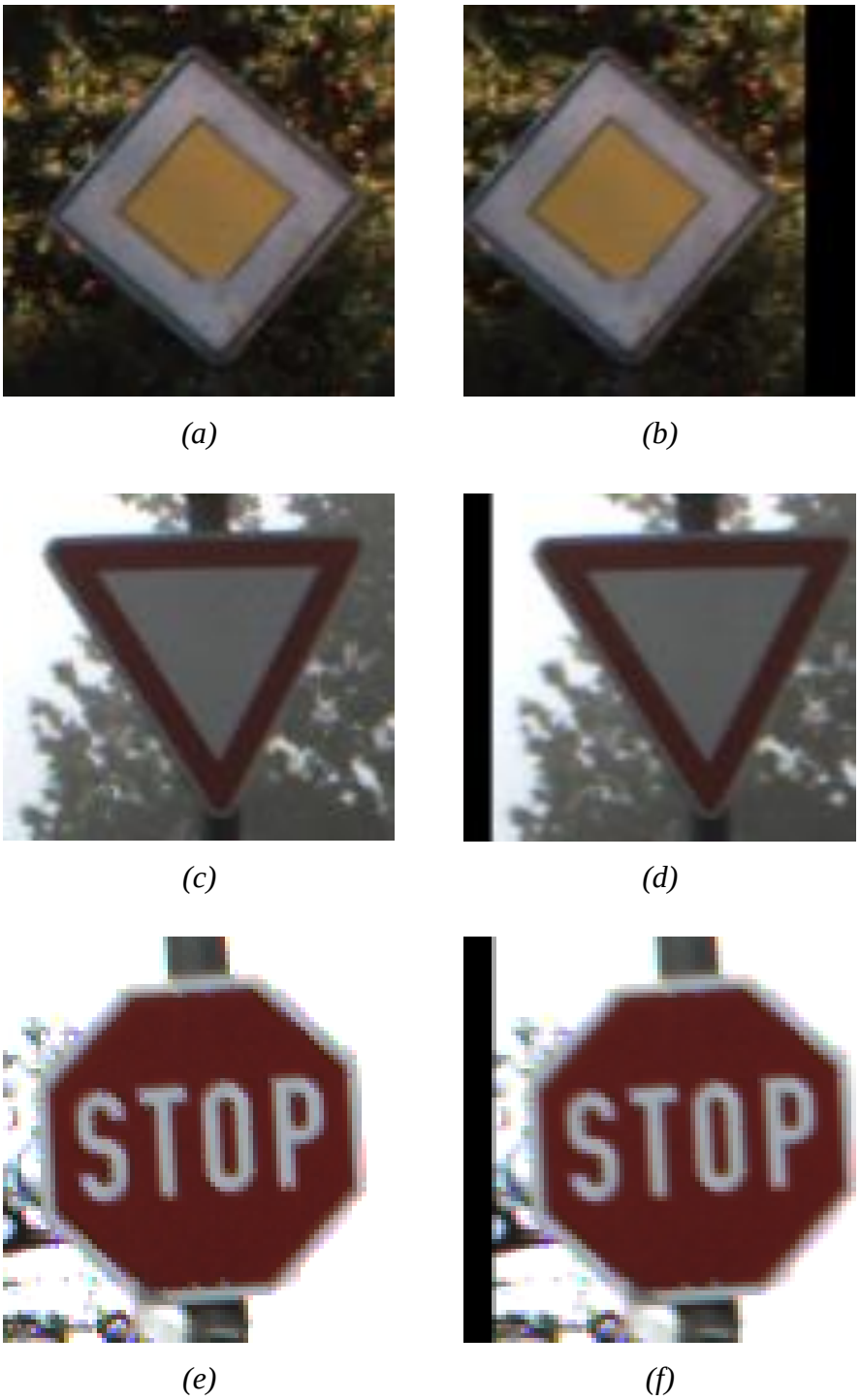
```
└──train
      ├──12
      │       00012_00011_00029.png
      │       00012_00025_00028.png
      │       00012_00054_00029.png
      │       augmented_000012_00011_00029.png
      │       augmented_000012_00025_00028.png
      │       augmented_000012_00054_00029.png
      │
      ├──13
      │       00013_00014_00021.png
      │       00013_00016_00026.png
      │       00013_00020_00025.png
      │       augmented_000013_00014_00021.png
      │       augmented_000013_00016_00026.png
      │       augmented_000013_00020_00025.png
      │
      └──14
              00014_00013_00017.png
              00014_00017_00025.png
              00014_00018_00024.png
              augmented_000014_00013_00017.png
              augmented_000014_00017_00025.png
              augmented_000014_00018_00024.png
```

Slika 5.2. Struktura direktorija i poddirektorija augmentiranog skupa preuzetog sa poslužitelja.

Na slici 5.3. prikazana je usporedba originalnih slika i augmentiranih za odabranu augmentaciju „Translate X“ s parametrima (-20, 20). Na slici je vidljivo da su pojedine slike pomaknute u lijevo, a pojedine u desno jer su postavljeni parametri bili (-20, 20), te se iz tog intervala nasumično uzima vrijednost pomaka kao postotak širine za svaku sliku.

Zatim je odabrana augmentacija „Translate Y“ s parametrima (10, 40). Rezultat je prikazan na slici 5.4. Na slici je vidljivo da su pojedine slike pomaknute samo prema dolje jer se pomak nasumično uzimao iz intervala (10, 40), a pozitivan pomak je prema dolje.

Isti je postupak ponovljen 8 puta za preostale augmentacije, te je samostalno utvrđeno da su augmentacije pravilno primijenjene. Također, testirana je kombinacija više augmentacija istovremeno. Odabrane su augmentacije „Posterize“ s parametrom 3, „Rotate“ s parametrima (-45, 45) i „Flip vertical“. Rezultat je prikazan na slici 5.5.



Slika 5.3. (a, c, e) Originalne slike;
(b, d, f) Slike nakon primjene augmentacije „Translate X“ s parametrima (-20, 20).

Za posljednji test je odabrano je 3 za broj augmentiranih slika. Na slici 5.6. prikazana je struktura datoteka unutar direktorija te je vidljivo da je nazivu datoteka dodan prefiks *augmented_<broj_slike>*, gdje *broj_slike* poprima vrijednosti od 0 do 2 što znači da je za svaku

originalnu sliku stvoreno 3 augmentirane. To potvrđuje ispravan rad odabira broja slika. Svi opisani testovi dali su očekivane rezultate.

(a)

(b)

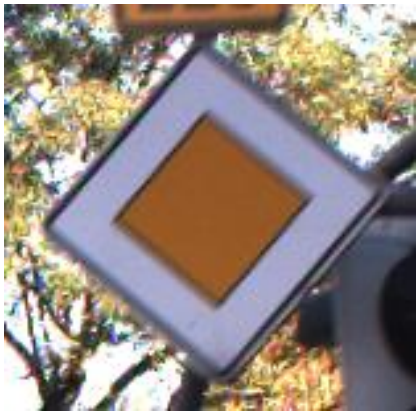
(c)

(d)

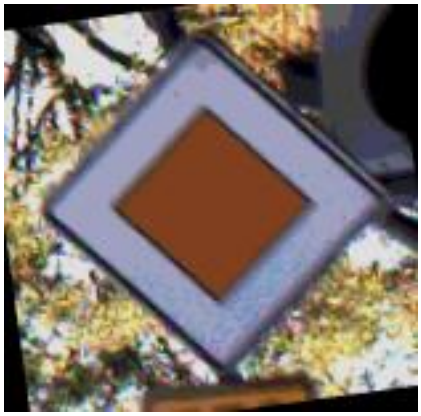
(e)

(f)

Slika 5.4. (a, c, e) Originalne slike;
(b, d, f) Slike nakon primjene augmentacije „Translate Y“ s parametrima (10, 40).

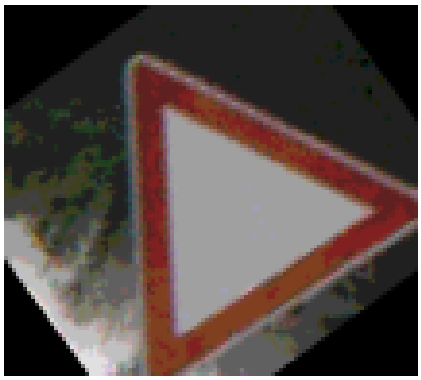


(a)



(b)

(c)



(d)

(e)

(f)

Slika 5.5. (a, c, e) Originalne slike;
(b, d, f) Slike nakon primjene augmentacija „Posterize“ s parametrom 3, „Rotate“ s parametrima (-45, 45) i „Flip vertical“

```
└──train
      ├──12
      │       00012_00011_00029.png
      │       00012_00025_00028.png
      │       00012_00054_00029.png
      │       augmented_000012_00011_00029.png
      │       augmented_000012_00025_00028.png
      │       augmented_000012_00054_00029.png
      │       augmented_100012_00011_00029.png
      │       augmented_100012_00025_00028.png
      │       augmented_100012_00054_00029.png
      │       augmented_200012_00011_00029.png
      │       augmented_200012_00025_00028.png
      │       augmented_200012_00054_00029.png
      │
      ├──13
      │       00013_00014_00021.png
      │       00013_00016_00026.png
      │       00013_00020_00025.png
      │       augmented_000013_00014_00021.png
      │       augmented_000013_00016_00026.png
      │       augmented_000013_00020_00025.png
      │       augmented_100013_00014_00021.png
      │       augmented_100013_00016_00026.png
      │       augmented_100013_00020_00025.png
      │       augmented_200013_00014_00021.png
      │       augmented_200013_00016_00026.png
      │       augmented_200013_00020_00025.png
      │
      └──14
              00014_00013_00017.png
              00014_00017_00025.png
              00014_00018_00024.png
              augmented_000014_00013_00017.png
              augmented_000014_00017_00025.png
              augmented_000014_00018_00024.png
              augmented_100014_00013_00017.png
              augmented_100014_00017_00025.png
              augmented_100014_00018_00024.png
              augmented_200014_00013_00017.png
              augmented_200014_00017_00025.png
              augmented_200014_00018_00024.png
```

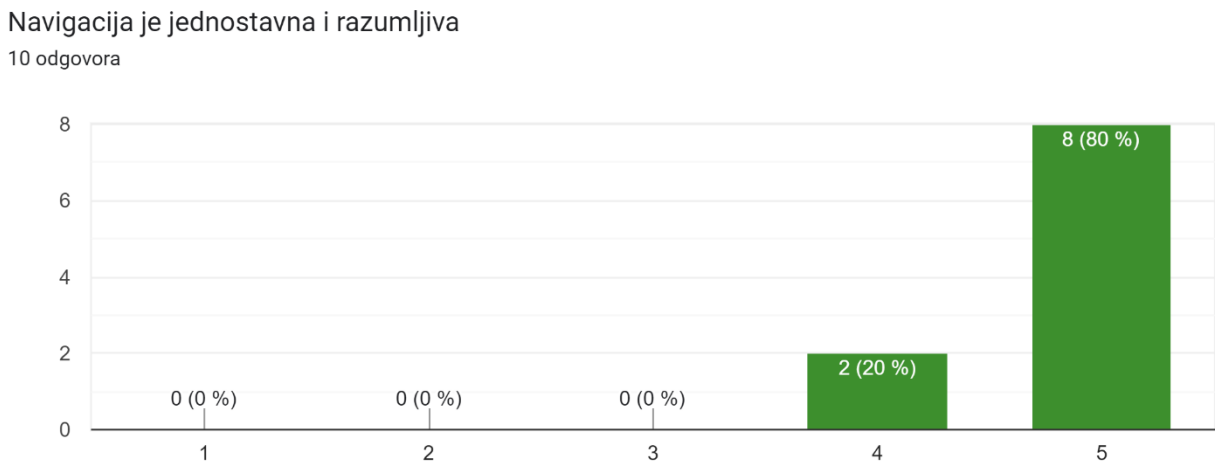
Slika 5.6. Struktura direktorija i poddirektorija augmentiranog skupa s odabrane 3 augmentirane slike.

5.2. Anketa o korištenju aplikacije

Ispitivanje o iskustvu korištenja razvijene aplikacije provedeno je u srpnju 2023. godine na uzorku od 10 ispitanika. Uzorak je prikupljen ciljnim pozivanjem ispitanika jer je potrebno da su ispitanici upoznati sa strojnim učenjem. Cilj anketnog istraživanja koje je opisano u nastavku bio je dobiti povratnu informaciju korisnika o radu i dizajnu razvijene aplikacije. Metoda ispitivanja bila je anonimna online anketa koju su ispitanici popunjavali nakon korištenja aplikacije bez vremenskog ograničenja.

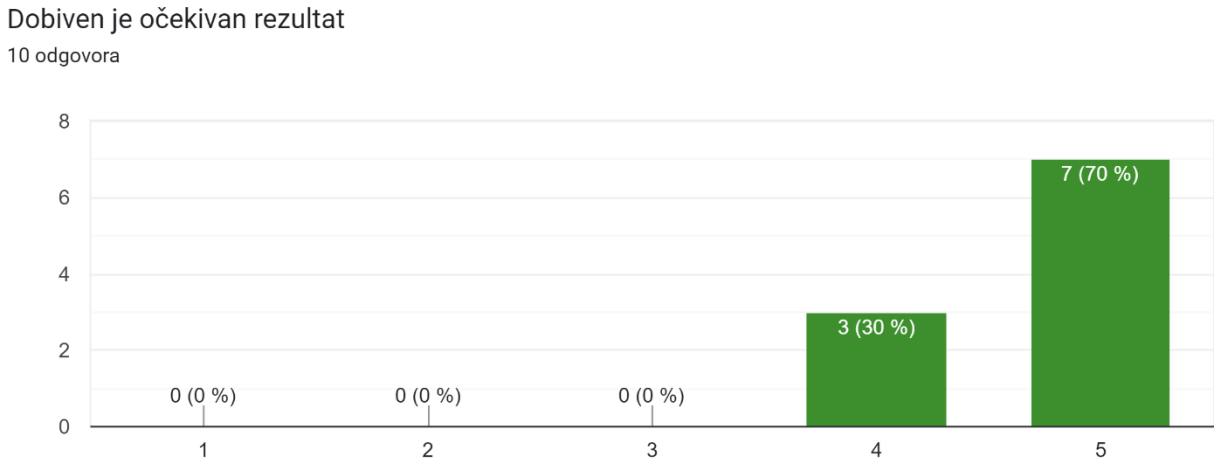
U sljedeća četiri pitanja ispitanici su trebali na skali od 1 do 5 označiti u kojoj se mjeri slažu s ponuđenom izjavom, gdje 1 označava da se uopće ne slažu, dok 5 znači da se u potpunosti slažu s izjavom.

Velika većina ispitanika se slagala s izjavom da je navigacija u aplikaciji jednostavna i razumljiva. (Slika 5.7.)



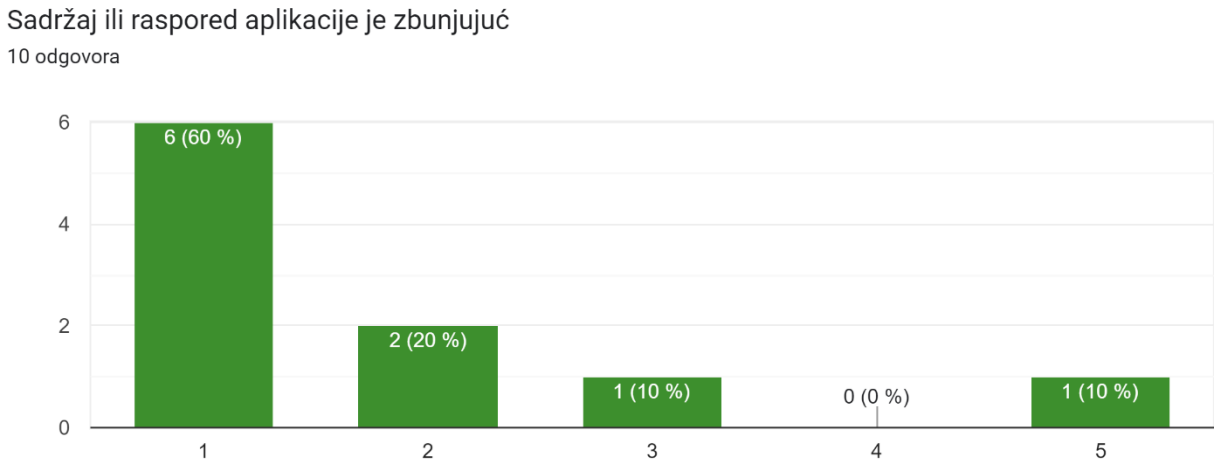
Slika 5.7. Grafički prikaz odgovora korisnika o navigaciji.

70% ispitanika se u potpunosti slagalo s izjavom da su korištenjem aplikacije dobili očekivani rezultat, dok se 30% ispitanika djelomično slagalo s izjavom (Slika 5.8.).



Slika 5.8. Grafički prikaz odgovora korisnika o rezultatu.

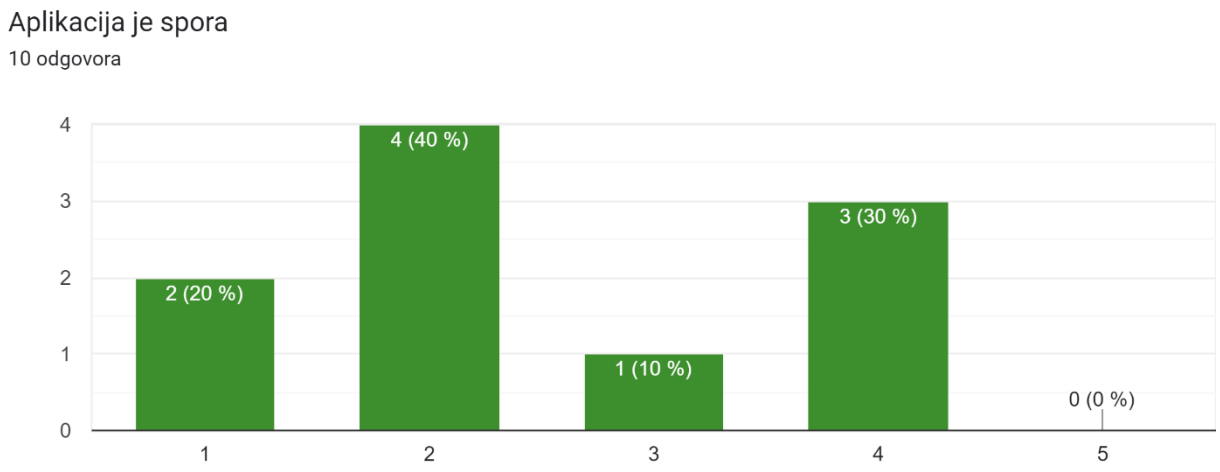
S izjavom da je sadržaj ili raspored aplikacije zbunjujuć se u potpunosti složio jedan ispitanik, jedan ispitanik se niti složio niti ne, dok se preostalih 8 ispitanika nije složilo s izjavom (Slika 5.9.). Moguć razlog zbunjujućeg sadržaja ili rasporeda je što kod odabira augmentacije informacije i pretpregled nisu vidljivi dok korisnik ne prijeđe mišem preko pojedine augmentacije. To znači da je možda teško započeti interakciju jer nije jasno što je potrebno odabrati.



Slika 5.9. Grafički prikaz odgovora korisnika o zbunjujućem sadržaju ili rasporedu.

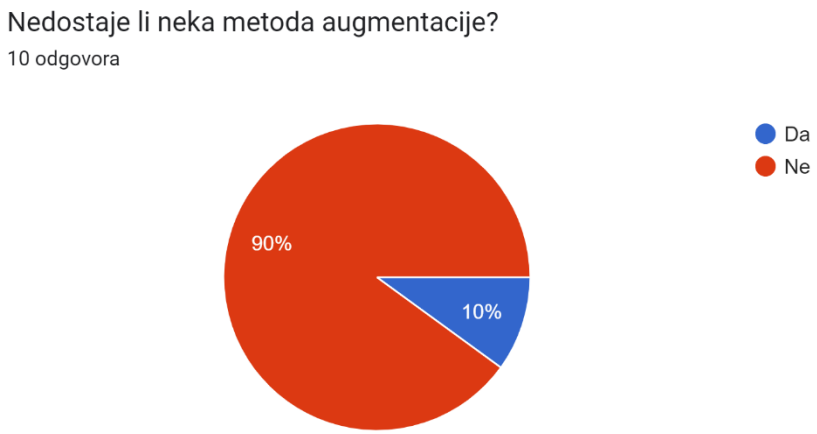
S izjavom da je aplikacija spora djelomično se složila 30% ispitanika (Slika 5.10.). Prosječan odgovor za ovu izjavu bio je 2.5, što znači da aplikacija nije ni spora ni brza. Još lošije iskustvo bi korisnici imali kada bi aplikaciju koristio veći broj korisnika zbog opterećenja poslužitelja. Jedan

od razloga tomu je korištenje osnovnog plana u *Azure* usluzi. Povećanjem razine usluge povećale bi se performanse poslužitelja, a time i brzina aplikacije.



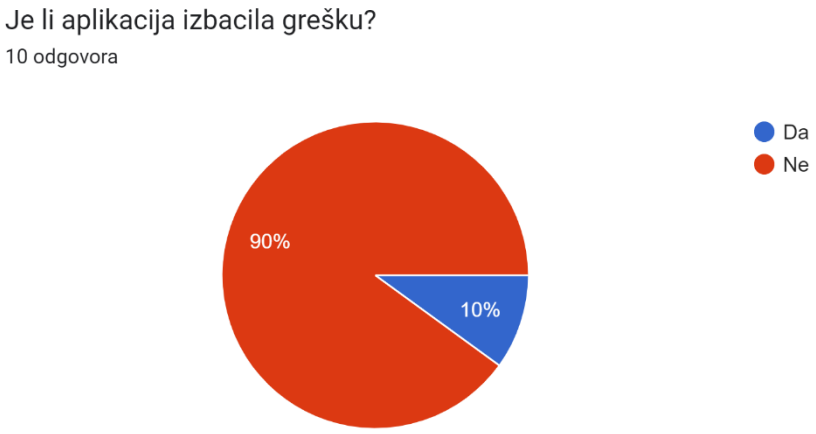
Slika 5.10. Grafički prikaz odgovora korisnika o brzini aplikacije.

Na pitanje „Nedostaje li neka metoda augmentacije?“ 90% ispitanika odgovorilo je da ne nedostaje (Slika 5.11.). Jedan ispitanik je na sljedeće pitanje „Ako nedostaje, koja?“ odgovorio da nedostaje metoda zamućenja (engl. *blur*). Kada bi se aplikacija izrađivala za tržište bilo bi poželjno provesti dodatno istraživanje o zahtjevima korisnika za dodatne metode augmentacije i dodati ih u aplikaciju.



Slika 5.11. Grafički prikaz odgovora korisnika o nedostajanju metoda augmentacije.

Posljednje pitanje je bilo vezano za moguće greške u aplikaciji koje su promaknule prilikom samostalnog testiranja. Jedan je korisnik na pitanje „Je li aplikacija izbacila grešku?“ odgovorio s da (Slika 5.12.), te je uvidom u zapise terminala na poslužitelju pronađena povremena greška vezana uz verzije *Python* biblioteke. Izmjenom *Python* verzije poslužitelja i verzije biblioteke te ponovnim učitavanjem izvornog kôda na uslugu u oblaku je greška riješena.



Slika 5.12. Grafički prikaz odgovora korisnika o pojavi greške.

6. ZAKLJUČAK

U ovome radu opisan je razvoj vlastite web aplikacije za augmentaciju podataka u oblaku. Aplikacija je postavljena na uslugu u oblaku, te je testirana njena funkcionalnost. Također, provedeno je istraživanje o korisničkom iskustvu s radom aplikacije. Opisani su pristupi i metode augmentacije podataka i uspoređene postojeće aplikacije. Korištene su suvremene tehnologije koje su pobliže objašnjene i pojašnjene na primjerima. Prilikom razvoja aplikacije pridržavalo se smjernica za korištenje i dokumentaciji korištenih tehnologija.

Testiranjem je potvrđen ispravan rad aplikacije. Istraživanje o korisničkom iskustvu je pokazalo da su korisnici zadovoljni s dizajnom i navigacijom unutar aplikacije, a korištenjem su dobili očekivane rezultate.

Moguće poboljšanje aplikacije bila bi traka napretka tijekom obrade slika te odvojen poslužitelj baze podataka kako bi podaci o korisnicima i skupovima podataka bili odvojeni od poslužitelja web stranice. Prednost toga bi bili sigurniji podaci i trajnost baze u slučaju ponovnog učitavanja na poslužitelj. Moguće je implementirati dodatne metode augmentacije, neke od kojih su i korisnici naveli da nedostaju. Također, moguće je platiti bolju razinu usluge u oblaku radi većih performansi aplikacije.

U konačnici, razvijena web aplikacija predstavlja koristan alat za ljude koji se bave problemima strojnog učenja i omogućuje učinkovit i praktičan pristup augmentaciji podataka u oblaku. Korisničko iskustvo i rezultati testiranja su zadovoljavajući. Ova tehnologija ima potencijal olakšati i približiti augmentaciju podataka ljudima koji su zainteresirani za strojno učenje, te im omogućiti pristup i sa slabijih ili mobilnih uređaja.

LITERATURA

[1] W. Di, A. Bhardwaj, i J. Wei, *Deep learning essentials: your hands-on guide to the fundamentals of deep learning and neural network modeling*. Birmingham, UK: Packt Publishing, 2018.

[2] D. Haba, *DATA AUGMENTATION WITH PYTHON enhance deep learning accuracy with data augmentation methods for image, text, audio, and tabular data*, 1st edition. England: PACKT PUBLISHING LIMITED, 2023.

[3] H. Xu *i ostali*, „An Enhanced Framework of Generative Adversarial Networks (EF-GANs) for Environmental Microorganism Image Augmentation With Limited Rotation-Invariant Training Data“, *IEEE Access*, sv. 8, str. 187455–187469, 2020, doi: 10.1109/ACCESS.2020.3031059.

[4] S. Yun *i ostali*, „CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features“. arXiv, 07. kolovoz 2019. [online]. Dostupno na: http://arxiv.org/abs/1905.04899 [05. kolovoz 2023]

[5] E. D. Cubuk, B. Zoph, J. Shlens, i Q. V. Le, „Randaugment: Practical automated data augmentation with a reduced search space“, u *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA: IEEE, lip. 2020, str. 3008–3017. doi: 10.1109/CVPRW50498.2020.00359.

[6] „Supervisely: unified OS for computer vision“ [online]. Dostupno na: https://supervisely.com/ [01. kolovoz 2023.].

[7] „Roboflow: Give your software the power to see objects in images and video“ [online]. Dostupno na: https://roboflow.com/ [01. kolovoz 2023.].

[8] „ImgAug Studio“ [online], *Supervisely*. Dostupno na: https://ecosystem.supervisely.com/apps/imgaug-studio [01. kolovoz 2023.].

[9] „Primer on Python Decorators“ [online], *Real Python*. Dostupno na: https://realpython.com/primer-on-python-decorators/ [27. srpanj 2023.].

[10] „Django“ [online], *Django Project*. https://www.djangoproject.com/ [23. kolovoz 2023.].

[10] B. Shaw, S. Badhwar, C. Guest, B. Chandra K S „Web Development with Django: A definitive guide to building modern Python web applications using Django 4“, Packt Publishing Pvt Ltd, 2023.

[11] „Getting Started | Serving Web Content with Spring MVC“ [online], *Spring*. Dostupno na: https://spring.io/guides/gs/serving-web-content/ [27. srpanj 2023.].

[12] „ASP.NET MVC Pattern | .NET“ [online], *Microsoft*. https://dotnet.microsoft.com/en-us/apps/aspnet/mvc [27. srpanj 2023.].

[13] „Ruby on Rails“ [online], *Ruby on Rails*. Dostupno na: https://rubyonrails.org/ [27. srpanj 2023.].

[14] V. Nishant, „Django MVT Architecture: A Fresh Take on Classic MVC“ [online], *AskPython*, 20. kolovoz 2020. Dostupno na: https://www.askpython.com/django/django-mvt-architecture [31. srpanj 2023.].

[15] „App Service — Build and Host Web Apps | Microsoft Azure“ [online], *Microsoft*. Dostupno na: https://azure.microsoft.com/en-us/products/app-service [31. srpanj 2023.].

[16] „What is a Bootstrap and how does it work?“ [online], *WhatIs.com*. Dostupno na: https://www.techtarget.com/whatis/definition/bootstrap [30. srpanj 2023.].

[17] „JavaScript Overview“ [online], *Tutorialspoint*. Dostupno na: https://www.tutorialspoint.com/javascript/javascript_overview [31. srpanj 2023.].

[18] „JavaScript | MDN“ [online], *Mozilla*. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript [31. srpanj 2023.].

[19] „AJAX Introduction“ [online], *W3Schools*. Dostupno na: https://www.w3schools.com/js/js_ajax_intro.asp [31. srpanj 2023.].

[20] „jQuery“ [online]. Dostupno na: https://jquery.com/ [31. srpanj 2023.].

[21] „Django - Widgets“ [online], *Django Project*. Dostupno na: https://docs.djangoproject.com/en/4.2/ref/forms/widgets/ [04. kolovoz 2023.].

[22] „Bootstrap Tooltips“ [online], *Bootstrap*. Dostupno na: https://getbootstrap.com/docs/5.0/components/tooltips/ [04. kolovoz 2023.].

[23] „GTSRB - German Traffic Sign Recognition Benchmark“ [online]. Dostupno na: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign [02. kolovoz 2023.]

SAŽETAK

U radu je opisan razvoj vlastite web aplikacije za augmentaciju podataka u oblaku. Aplikacija je implementirana na platformi u oblaku, testirana i istraženo je korisničko iskustvo. Prikazani su pristupi i metode augmentacije te uspoređene postojeće aplikacije. Korištene su suvremene tehnologije koje su pobliže objašnjene i pojašnjene na primjerima. Testiranje je potvrdilo ispravnost rada, a istraživanje korisničkog iskustva pokazalo zadovoljstvo dizajnom i navigacijom te postignutim rezultatima. Razvijena aplikacija koristan je alat za probleme strojnog učenja, omogućujući učinkovitu augmentaciju podataka u oblaku. S pozitivnim korisničkim iskustvom i rezultatima, tehnologija ima potencijal olakšati i približiti augmentaciju podataka ljudima zainteresiranim za strojno učenje, čak i na slabijim uređajima.

Ključne riječi: *augmentacija podataka, Django, platforma u oblaku, Azure, korisničko iskustvo*

ABSTRACT

DATA AUGMENTATION APPLICATION IN CLOUD

This paper describes the development of a data augmentation web application in the cloud. The application was implemented on a cloud platform. The application was tested and a user experience survey was conducted. An overview of augmentation approaches and methods was provided, and existing applications were compared. It utilizes contemporary technologies, whose features were explained and demonstrated using examples. Testing verified that the application works correctly, and the user experience survey demonstrated user satisfaction with design and navigation, as well as the achieved outcomes. The developed application is a useful tool for machine learning problems, enabling efficient data augmentation in the cloud. With positive user experience and results, the technology has the potential to simplify and make data augmentation more accessible to individuals interested in machine learning, even on less powerful devices.

Key words: *data augmentation, Django, cloud platform, Azure, user experience*

ŽIVOTOPIS

Dominik Đuranić rođen je 12. svibnja 1998. godine u Osijeku. Završio je Osnovnu školu Ladimirevci te opću gimnaziju u Srednjoj školi Valpovo. Tijekom srednje škole sudjelovao je na brojnim županijskim natjecanjima iz fizike, engleskog jezika te na državnom natjecanju iz logike. Preddiplomski studij računarstva završio je na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Na istom fakultetu nastavlja diplomski sveučilišni studij Računarstva, smjer Informacijske i podatkovne znanosti.

_____

Potpis autora

PRILOZI

Prilog P.4.1: Kompletan programski kod aplikacije (elektronički prilog)

Prilog P.4.2: AJAX poziv za učitavanje datoteke

```
$.ajax({
    xhr: function() {
        var xhr = new window.XMLHttpRequest();
        $(".progress").show();
        document.getElementById("bar").className = "progress-bar bg-primary";
        xhr.upload.addEventListener("progress", function(evt) {
        if (evt.lengthComputable) {
            var percentComplete = (evt.loaded / evt.total) * 80;
            percentComplete = parseInt(percentComplete);
            console.log(percentComplete);

            $("#bar").css('width',percentComplete+'%');
            $("#bar").text(percentComplete+'%')
            if (percentComplete === 80) {
                $("#success").html("Extracting the files...");
                $("#success").show();
            }
        }
        }, false);

        return xhr;
    },
    url: "{% url 'upload' %}", type: "POST",
    data: formData, processData: false, contentType: false,
    success: function(result) {
        $("#bar").css('width','100%');
        $("#bar").text('100%')
        $("#success").html(result["message"]);
        $("#success").show();
        $("#bar").addClass("bg-success");
        $("#submitButton").prop("disabled", false)
    },
    error: function(result){
        $("#success").html(result.responseJSON.error);
        $("#success").show();
        document.getElementById("bar").className = "progress-bar bg-danger";
        $("#submitButton").prop("disabled", false);
    }
});
```

Prilog P.4.3: Dio pogleda augmentDataset za popunjavanje liste augmenters

```
augmenters=[]
if(premade=="True"):
    augmenters.append(iaa.RandAugment(ranges[7][0], ranges[7][1]))
else:
for augmentation in augmentations:
    match (augmentation):
        case 'solarize':
            augmenters.append(iaa.Solarize(1, threshold=ranges[0]))
        case 'posterize':
            augmenters.append(iaa.Posterize(ranges[1]))
        case 'translatex':
            augmenters.append(iaa.TranslateX(percent=ranges[2]))
        case 'translatey':
            augmenters.append(iaa.TranslateY(percent=ranges[3]))
        case 'shearx':
            augmenters.append(iaa.ShearX(ranges[4]))
        case 'sheary':
            augmenters.append(iaa.ShearY(ranges[5]))
        case 'flipx':
            augmenters.append(iaa.Fliplr(1))
        case 'flipy':
            augmenters.append(iaa.Flipud(1))
        case 'rotate':
            augmenters.append(iaa.Affine(rotate=ranges[6]))
        case _:
            print("illegal state")
```

Prilog P.4.4: Dio pogleda augmentDataset za obradu i spremanje slika.

```
seq = iaa.Sequential(augmenters)
dataset_folder = os.path.join(settings.MEDIA_ROOT, request.user.username,
"dataset")
augmented_folder = os.path.join(settings.MEDIA_ROOT,
request.user.username, "augmented")
try:
    shutil.rmtree(augmented_folder)
    shutil.copytree(dataset_folder, augmented_folder)
    for subdir, dirs, files in os.walk(augmented_folder):
        for file in files:
            # Create the full file path
            file_path = os.path.join(subdir, file)
            # Check if the file is an image
            if file.endswith('.jpg') or file.endswith('.png'):
```

```
                    # Open the image using PIL
                    image =np.array( Image.open(file_path))
                    for i in range(numberOfImages):
                        # Apply the augmentations
                        augmented_image = seq.augment_image(image)
                        augmented_image_pil = Image.fromarray(augmented_image)
                        # Save the augmented image in the same directory
                        augmented_file_path = os.path.join(subdir,
f"augmented_{i}{file}")
                        augmented_image_pil.save(augmented_file_path,
format='PNG')
  except Exception as e:
      return JsonResponse({"message": "An error occured. Please try
again."})
```