Web aplikacija za pomoć pri samostalnom šišanju

Šlafhauzer, Jakov

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

Sveučilišni prijediplomski studij Računarstvo

WEB APLIKACIJA ZA POMOĆ PRI SAMOSTALNOM ŠIŠANJU

Završni rad

Jakov Šlafhauzer

Osijek, 2024.

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

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Jakov Šlafhauzer
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4719, 28.07.2021.
JMBAG:	0165091938
Mentor:	izv. prof. dr. sc. Mirko Köhler
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Web aplikacija za pomoć pri samostalnom šišanju
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Tema rezervirana za Jakov Šlafhauzer. Potrebno je napraviti web aplikaciju za pomoć pri vlastitom šišanju. Aplikacija mora na osnovu korisnikovog ulaznog podatka predložiti frizuru i upute za šišanje.

Datum prijedloga ocjene završnog rada od strane mentora:	18.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	25.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	26.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 26.09.2024.

Ime i prezime Pristupnika:	Jakov Šlafhauzer
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4719, 28.07.2021.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za pomoć pri samostalnom šišanju**

izrađen pod vodstvom mentora izv. prof. dr. sc. Mirko Köhler

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 pristupnika:

SADRŽAJ

1. UVOD

Unazad nekoliko godina drastično je porasla učestalost šišanja kod muškaraca, što je uzrokovalo velikom potražnjom za terminima u frizerskim salonima do one mjere da ih klijenti moraju rezervirati i do mjesec dana prije samog termina. U razdoblju pandemije COVID-19 frizerski saloni morali su privremeno zatvoriti svoje obrte, što je navelo mnoge muškarce da se okušaju u samostalnom šišanju. Većini to nije pošlo za rukom, ali oni koji su se potrudili i uspjeli savladati ovu vještinu, shvatili su koliko je isplativo – kako organizacijski, tako i financijski. Naime, cijene muških šišanja udvostručile su se u zadnjih pet godina. Upravo to daje ideju izrade ovoga završnog rada kako bi se što više muškaraca motiviralo i odlučilo na šišanje kod kuće.

Nakon uvoda, drugo poglavlje prikazuje postojane aplikacije i web stranice koje pružaju sličnu uslugu pomoći pri samostalnom šišanju. U trećem poglavlju bit će navedeni i detaljno objašnjeni alati i tehnologije koji su bili korišteni za razvoj ove web aplikacije. U četvrtom poglavlju bit će opisan postupak izrade same aplikacije kroz primjere računalnog koda, a u petom poglavlju bit će prikazano sučelje same aplikacije te njene funkcionalnosti, dok će zadnje poglavlje zaključiti temu ovoga rada.

1.1. Zadatak završnog rada

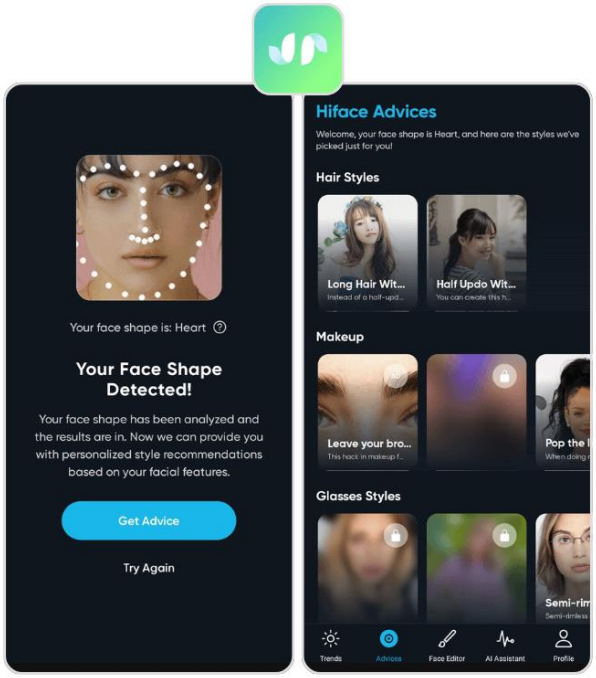
Zadatak završnog rada je izrada web aplikacije koja ima mogućnost prijave korisnika, skeniranja oblika korisnikova lica te na osnovu oblika lica i trenutnih trendova predložiti niz odgovarajućih frizura. Moguće je pogledati opis predloženih frizura te alate potrebne za izvršavanje frizure kako bi korisnici detaljnije mogli znati što trebaju nabaviti za samostalno šišanje.

2. PREGLED PODRUČJA TEME

U ovome poglavlju nabrojana su postojeća rješenja za aplikacije slične tematike.

2.1. HiFace

HiFace je iOS aplikacija koja omogućuje da uz brzo snimanje selfie-a korisnici mogu dobiti detaljnu analizu oblika lica te preporuke za frizure, stiliziranje brade, vrste naočala i šminke. [1] Aplikacija također otkriva s kojom slavnom osobom korisnici dijele sličnosti. HiFace nudi AI asistenta za ljepotu i modu koji pruža savjete o najnovijim trendovima i tehnikama njege kože. Korisnici mogu virtualno isprobavati različite frizure, opcije šminke i estetske tretmane prije nego što donesu konačnu odluku, što je prikazano na slici 2.1.

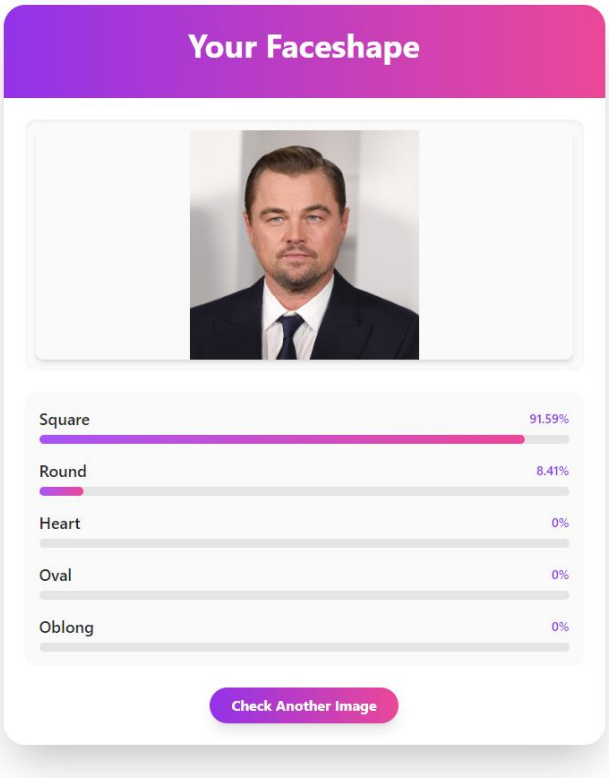


Sl. 2.1. Sučelje aplikacije HiFace

2.2. Face Shape Detector

Face Shape Detector je web aplikacija koja omogućuje korisniku detekciju oblika lica tako što se jednostavno iz računala priloži željena fotografija. Kao što prikazuje slika 2.2. , rezultati se ispisuju odmah u obliku postotaka koliko lice odgovara kojemu obliku. Aplikacija nudi objašnjenje postupka detekcije oblika lica te kako se ova tehnologija razvijala od početaka računala do danas.

Također se mogu pronaći opisi pojedinih lica te ideje kako istaknuti ista različitim frizurama, šminkom i slično. [2]



Sl. 2.2. Rezultat detekcije lica aplikacije Face Shape Detector

2.3. The Good Barbers

The Good Barbers je web aplikacija koja, za razliku od web aplikacije ovoga rada, objašnjava kako odrediti oblik vlastitog lica bez korištenja računalnih tehnologija. Međutim, vrlo detaljno pruža korisne informacije vezane za svaki oblik lica te uz predložene frizure nudi i kako stilizirati odgovarajuću i koje proizvode je najbolje koristiti kako bi se ona postigla. [3] Na slici 2.3. prikazan je blog koji predstavlja vodič za odgovarajuće frizure prema obliku lica korisnika.

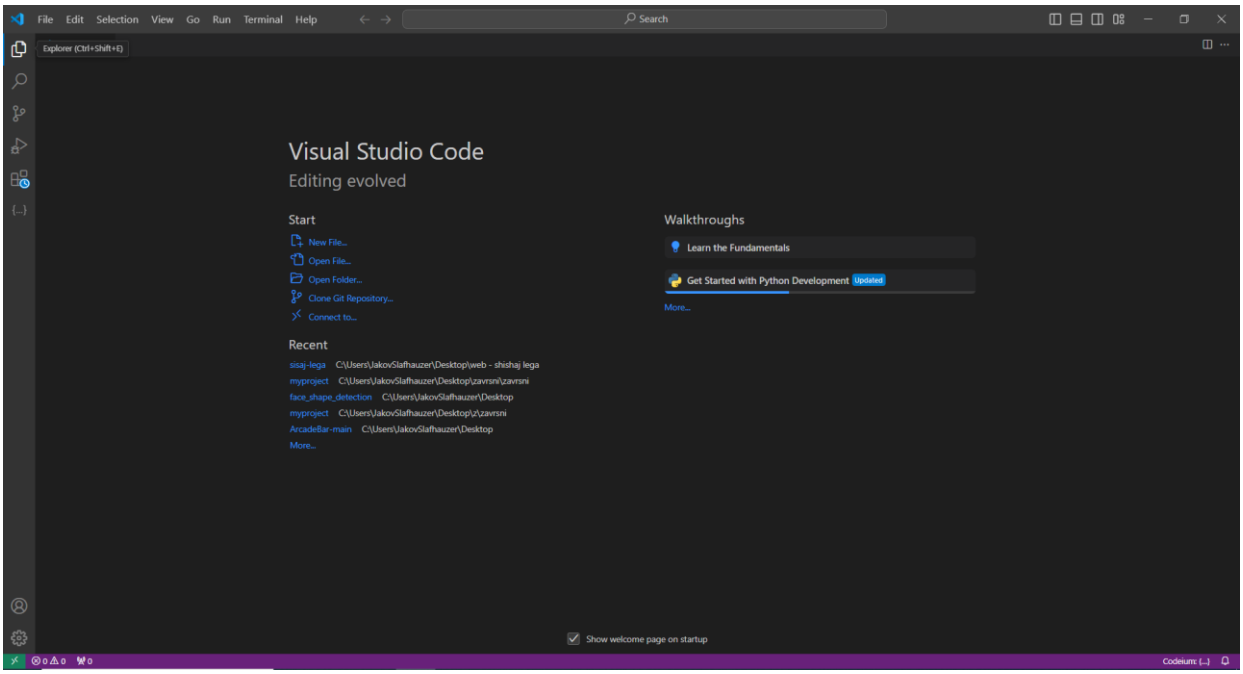
Sl. 2.3. Vodič za frizure u web aplikaciji The Good Barbers

3. KORIŠTENE TEHNOLOGIJE I ALATI

U ovome poglavlju bit će opisane sve tehnologije i alati koji su korišteni u izradi web aplikacije koja je poslužila kao tema za ovaj završni rad. Kao uređivač teksta korišten je Visual Studio Code, aplikacija je izrađena u Django web okviru pomoću programskog jezika Python te opisnog HTML jezika i stilskog jezika CSS koji su zaduženi za opis korisničkog sučelja.

3.1. Visual Studio Code

Visual Studio Code, poznatiji kao VS Code, besplatan je uređivač teksta razvijen od strane Microsofta za pisanje i oblikovanje koda. Olakšava razvoj softvera podržavanjem velikog izbora programskih jezika kao što su Python, C++ ili Java te brojnim značajkama poput otkrivanja pogrešaka, kontrola verzija ili isticanje sintakse. [4] Na slici 3.1. nalazi se početno sučelje uređivača teksta VS Code.



Sl. 3.1. Početno sučelje uređivača teksta VS Code

3.2. Python

Python je programski jezik visoke razine poznat po svojoj čitljivosti i jednostavnosti korištenja. Po prirodi je objektno orijentirani jezik, no podržava i proceduralno te funkcionalno programiranje. Koristi u znanosti o podacima, za razvoj web aplikacija, umjetnu inteligenciju i automatizaciju zbog velikog opsega biblioteka i okvira, što ga čini svestranim.

3.3. Opisivanje sadržaja web stranice

3.3.1. HTML

HTML (engl. *HyperText Markup Language)* je jezik koji se koristi za izradu i strukturiranje web stranica. Definira elemente web stranice, kao što su naslovi, odlomci, poveznice i slike. HTML radi uz CSS i JavaScript za izradu modernih, interaktivnih web stranica. Koristi niz oznaka za organiziranje sadržaja, što ga čini čitljivim web preglednicima. HTML je neophodan za svaki projekt web-razvoja i temelj je njegova sadržaja.

3.3.2. CSS

CSS (engl. *Cascading Style Sheets*) je jezik koji opisuje kako će elementi propisani opisnim jezikom poput HTML-a biti prikazani na mrežnoj stranici. Vrlo je važan jer je ključan za dizajn same stranice, što utječe na korisničko iskustvo. Može kontrolirati dizajn nekoliko stranica u isto vrijeme te tako uskraćuje nepotrebno ponavljanje programskog koda.

3.4. Bootstrap

Bootstrap je najpoznatiji besplatni razvojni okvir koji obuhvaća CSS, HTML i JavaScript s glavnim ciljem razvoja responzivnih web stranica te njihovom prilagođavanju mobilnim uređajima. Integriranjem u zaglavni element HTML dokumenta omogućuje unaprijed izgrađene komponente kao što su gumbi, navigacijske trake, obrasci, ili nizovi fotografija (engl. *carousel*) čineći web razvoj bržim i lakšim. [5]

3.5. Django

Django je razvojni okvir koji olakšava full stack izradu web aplikacija korištenjem Pythona. Omogućuje korištenje već gotovih značajki poput sustava autentifikacije, povezivanja s bazom podataka te upravljanja sadržaja kroz administratorsko sučelje. Django se temelji na MVT (engl. *model-view-template*) arhitekturi. Na slikama 3.2., 3.3. i 3.4. prikazani su primjeri modela, pogleda i predloška u Djangu. Modeli su podaci koje želimo prikazati, a predstavljaju ih entiteti u bazi podataka. Pogled je rukovatelj korisničkih zahtjeva koji uzima modele koje treba prikazati te ih prikazuje pomoću predložaka, dokumenata (najčešće .html dokumenti) koji predstavljaju izgled mrežne stranice.

models.py

detector > models.py > PredictShape > train_model

```python
from django.db import models
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import detector.data.modelData as data

Codeium: Refactor | Explain
class PredictShape:
    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, featureVector) -> None:
        self._new_feature = featureVector

    Codeium: Refactor | Explain | ×
    def train_model(self):
        """
        Training a DTC for prediction classification
        """
        X  = data.X
        y = data.y

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=41)

        model = DecisionTreeClassifier()

        model.fit(X_train, y_train)

        new_feature_vector = self._new_feature

        predictions = model.predict([new_feature_vector])
        y_test_predictions = model.predict(X_test)

        accuracy = accuracy_score(y_test, y_test_predictions)

        return [["accuracy", accuracy * 100], [ "prediction", predictions]]
```

Sl. 3.2. Primjer modela u razvojnom okviru Django

views.py

detector > views.py > ...

```python
import os
from django.conf import settings
from django.core.files.storage import default_storage
from django.shortcuts import render

from detector.face_detector import Main


Codeium: Refactor | Explain | Generate Docstring |
def upload_image(request):
    if request.method == 'POST' and request.FILES.get('image'):
        image = request.FILES['image']

        tmp_dir = os.path.join(settings.MEDIA_ROOT, 'tmp')
        os.makedirs(tmp_dir, exist_ok=True)
        temp_path = os.path.join(tmp_dir, image.name)

        path = default_storage.save(temp_path, image)
        absolute_path = os.path.join(settings.MEDIA_ROOT, path)

        if not os.path.exists(absolute_path) or os.stat(absolute_path).st_size == 0:
            return render(request, 'detector/result.html', {
                'classification_result': 'File upload failed or file is empty',
                'landmarks_result': 'N/A'
            })

        shape_result = Main(absolute_path).run_detection_stillshot()

        split_result = shape_result.split('\n')

        classification_result = split_result[0] if len(split_result) > 0 else "No classification"
        landmarks_result = split_result[1] if len(split_result) > 1 else "No landmarks"

        hairstyle_suggestions = split_result[2:] if len(split_result) > 2 else []

        return render(request, 'detector/result.html', {
            'classification_result': classification_result,
            'landmarks_result': landmarks_result,
            'hairstyle_suggestions': hairstyle_suggestions
        })

    return render(request, 'detector/upload.html')

```

Sl. 3.3. Primjer pogleda u razvojnom okviru Django

result.html

detector > templates > detector > result.html > ...

```html
<html lang="en">
<body>


    <div class="result-container">
        <h2>{{ classification_result }}</h2>
        <h2>{{ landmarks_result }}</h2>

        {% if hairstyle_suggestions %}
            <p>{{ hairstyle_suggestions|join:", " }}</p>
            <p>Za više informacija o ovim frizurama, slobodno istraži malo više na našoj <a href="{% url 'pocetna' %}" class="poc-link">stranici</a>.</p>
        {% else %}
            <p>No hairstyle suggestions available.</p>
        {% endif %}
    </div>


    <a href="{% url 'upload' %}" class="upload-btn">Učitaj novu fotografiju</a>

    <footer>
        <p>&copy; 2024 Šišaj lega!</p>
        <p><a href="#">Privacy Policy</a> | <a href="#">Terms of Service</a></p>
    </footer>

</body>
</html>
```

Sl. 3.4. Primjer predloška u razvojnom okviru Django

3.6. GitLab

GitLab je platforma temeljena na webu koja pruža usluge kontrole verzija te kontinuirane integracije i isporuke. Izgrađen je oko Gita, omogućujući timovima da surađuju na repozitorijima kodova, prate promjene i učinkovito upravljaju softverskim projektima. GitLab nudi integrirane alate za praćenje problema, pregled koda i automatizaciju cjevovoda, što ga čini cjelovitim rješenjem za životni ciklus razvoja softvera. Za razliku od GitHuba koji je temeljen pretežito na oblaku, GitLab nudi i vlastito hostiranje (engl. *self-host*) pružajući time veću kontrolu.

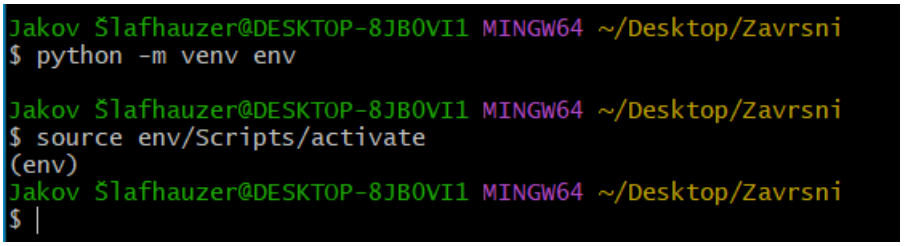
4. IZRADA APLIKACIJE

U ovome poglavlju bit će objašnjeni postupci izrade aplikacije „Šišaj lega!“, korištene kao tema ovoga završnoga rada. Za stvaranje virtualnog okruženja i Django projekta te navigaciju kroz projekt korišten je Git Bash.

4.1. Kreiranje virtualnog okruženja

Virtualno okruženje je alat koji pomaže u održavanju odvojenih ovisnosti (engl. *dependency*) koje zahtijevaju različiti projekti stvaranjem izoliranih Python virtualnih okruženja za njih. Postavljanje virtualnog okruženja u Djangu bitno je za izolaciju ovisnosti vašeg projekta i osiguravanje dosljednog ponašanja u različitim okruženjima. Virtualno okruženje omogućuje lokalno instaliranje paketa bez utjecaja na globalnu instalaciju Pythona. [6]

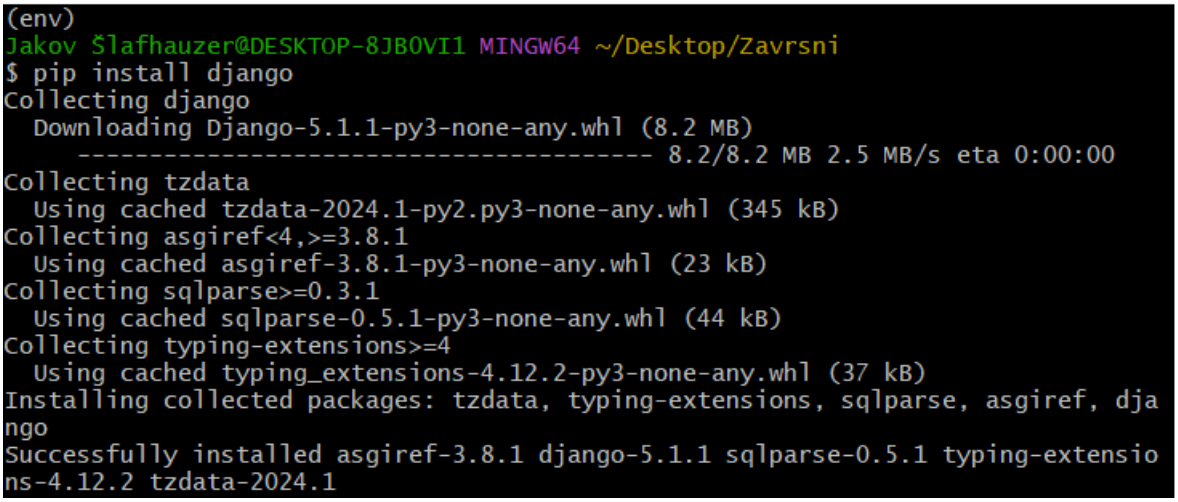
Kada se virtualno okruženje kreira, potrebno ga je i aktivirati te se onda preuzimaju potrebni paketi u svrhu izrade projekta. Nakon aktiviranja, naziv virtualnog okruženja koji se trenutno koristi stoji uz svaku liniju naredbe u terminalu kako bi dalo do znanja korisniku da je aktivirano. Virtualno okruženje u ovome projektu nazvano je env, a postupak njegova stvaranja i aktiviranja prikazan je slikom 4.1.



Sl. 4.1. Stvaranje i aktiviranje virtualnog okruženja

4.2. Kreiranje projekta

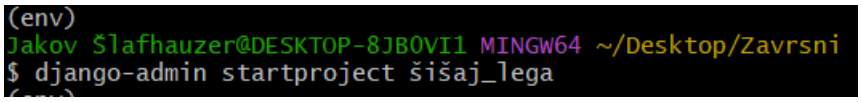
S postavljenim i aktiviranim virtualnim okruženjem moguće je započeti stvaranje Django projekta i aplikacija. Najprije je potrebno preuzeti i instalirati Django paket pomoću pip upravitelja paketa, što je prikazano slikom 4.2.



Sl. 4.2. Instaliranje Django paketa

4.2.1. Kreiranje Django projekta

Nakon uspješnog preuzimanja Django paketa, kreira se projekt koji predstavlja web aplikaciju. Projekt će se sadržavati od nekoliko aplikacija, koje odrađuju određeni posao, odnosno funkcionalnost, kako bi struktura projekta bila preglednija i kako bi projekt učinilo lakšim za upravljati. Prikazano na slici 4.3., projekt se kreira naredbom *django-admin startproject project-name*.

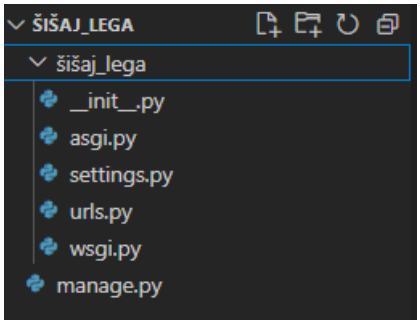


Sl. 4.3. Kreiranje Django projekta s nazivom „šišaj_lega“

Na slici 4.4. prikazana je trenutna struktura projekta prije stvaranja aplikacija, a sastoji se od sljedećih datoteka:

- *__init__.py* – prazna datoteka koja govori Pythonu da direktorij projekta treba smatrati kao Python paket
- *asgi.py* – ulazna točka ASGI-kompatibilnih mrežnih servera
- *settings.py* – konfiguracija postavki za ovaj projekt
- *urls.py* – deklaracije URL veza ovoga projekta

- *wsgi.py* – ulazna točka WSGI-kompatibilnih mrežnih servera
- *manage.py* – uslužni program pomoću kojega se na razne načine upravlja projektom te se jedini ne nalazi unutar direktorija projekta [7]



Sl. 4.4. Struktura direktorija Django projekta

4.2.2. Kreiranje Django aplikacija

Za potrebe web aplikacije ovoga projekta, kreirane su tri Django aplikacije. „Base“ aplikacija predstavlja hardkodirani dio aplikacije, a to su početna stranica i stranice s opisima frizura i alata korištenih za postizanje određenih tipova frizura. Aplikacija „accounts“ upravlja korisničkim računima tako što omogućuje registraciju i prijavu korisnika te spremanje njihovih računa u Django administracijski sustav. Zadnja kreirana aplikacija nazvana je „detector“ te se u njoj odvija logika detekcije oblika lica i predlaganje frizura prema dobivenom rezultatu. Prikazano na slici 4.5., aplikacije se kreiraju naredbom *python manage.py startapp app-name*.

```
(env)
Jakov Šlafhauzer@DESKTOP-8JBOVI1 MINGW64 ~/Desktop/Zavrsni/šišaj_lega
$ python manage.py startapp base
(env)
Jakov Šlafhauzer@DESKTOP-8JBOVI1 MINGW64 ~/Desktop/Zavrsni/šišaj_lega
$ python manage.py startapp accounts
(env)
Jakov Šlafhauzer@DESKTOP-8JBOVI1 MINGW64 ~/Desktop/Zavrsni/šišaj_lega
$ python manage.py startapp detector
```

Sl. 4.5. Kreiranje korištenih Django aplikacija

Nakon što su aplikacije kreirane, potrebno ih je povezati s projektom u datoteci settings.py kako je prikazano na slici 4.6.

```
34  INSTALLED_APPS = [
35      'django.contrib.admin',
36      'django.contrib.auth',
37      'django.contrib.contenttypes',
38      'django.contrib.sessions',
39      'django.contrib.messages',
40      'django.contrib.staticfiles',
41      'base',
42      'detector',
43      'accounts',
44  ]
```

Sl. 4.6. Povezivanje aplikacija u postavkama projekta

Također, unutar urls.py koji predstavlja URL veze na razini projekta, potrebno je povezati URL veze aplikacije, što prikazuje slika 4.7.

```
23  urlpatterns = [
24      path('admin/', admin.site.urls),
25      path('', include('base.urls')),
26      path('', include('detector.urls')),
27      path('accounts/', include('django.contrib.auth.urls')),
28      path('accounts/', include('accounts.urls')),
29  ]
```

Sl. 4.7. Povezivanje URL veza u putanjama projekta

4.3. Modeli

Model u Djangu predstavlja tablicu u SQLite bazi podataka koja se automatski generira prilikom stvaranja projekta. Modeli sadrže podatke koji se koriste unutar web aplikacije. U Pythonu kreirati model znači napisati klasu s atributima i ponašanjima. Modeli kao konkretni objekti mogu biti dodani tijekom rada aplikacije ili kroz Djangovu administracijsku nadzornu ploču.

Za razliku od ostalih Django web aplikacija, za potrebe ovoga projekta nije bilo nužno kreirati modele koji će biti prikazani u Django administraciji.

4.4. Pogledi

Pogledi su Python funkcije koje primaju http zahtjeve i vraćaju http odgovore, najčešće .html dokumente koji se u Djangu zovu predlošci, preusmjeravanja ili greške. Pogledi mogu biti i složeni i jednostavni. Primjer najjednostavnijeg pogleda prikazuje slika 4.8. gdje pogled kao zahtjev (engl. *request*) prima samo dohvaćanje određene stranice te kao odgovor i vraća tu stranicu u obliku HTML dokumenta.

```
6  def pocetna(request):
7      context = []
8      return render(request, 'base/pocetna.html')
9
```

Sl. 4.8. Primjer jednostavnog pogleda

Nešto složeniji pogled može se primijetiti na slici 4.9. gdje je zahtjev predstavlja neku korisničku radnju, što je u ovome slučaju pokušaj izrade korisničkog računa. Funkcija redirect preusmjerava na odgovarajuću stranicu, u ovome slučaju početnu stranicu, ukoliko je forma registracije valjano ispunjena, a funkcija render prikazuje web stranicu registracije prilikom dohvaćanja iste. Unutar vitičastih zagrada kao argument funkcije render predaje se context, rječnik varijabli koje se prosljeđuje navedenom predlošku (stranica registracije) kako bi imao pristup određenim informacijama. Logika autentifikacije korisničkog računa detaljnije je objašnjena u jednom od narednih poglavlja.

```
29  def register_user(request):
30      if request.method == 'POST':
31          form = RegisterUserForm(request.POST)
32          if form.is_valid():
33              form.save()
34              username = form.cleaned_data['username']
35              password = form.cleaned_data['password1']
36              user = authenticate(username=username, password=password)
37              login(request, user)
38              messages.success(request, 'You have successfully registered...')
39              return redirect('pocetna')
40      else:
41          form = RegisterUserForm()
42      return render(request, 'authenticate/register_user.html', {
43          'form': form,
44      })
```

Sl. 4.9. Primjer složenijeg pogleda

4.5. Predlošci

Predlošci su treći i najvažniji dio Djangove MVT strukture. To je .html datoteka napisana pomoću HTML-a, CSS-a i/ili JavaScripta. Django okvir učinkovito obrađuje i generira dinamičke HTML web stranice koje su vidljive krajnjem korisniku. Django uglavnom funkcionira s backendom pa se koriste predlošci kako bi se osiguralo sučelje i pružio izgled web aplikaciji. Postoje dvije metode dodavanja predloška u Django projekt. Može se koristiti jedan direktorij predložaka koji će se proširiti na cijeli projekt ili se svaku aplikaciju našeg projekta stvori drugačiji direktorij predložaka. U ovome projektu u svakoj aplikaciji stvoren je direktorij nazvan templates u kojemu su pohranjeni odgovarajući predlošci te aplikacije. Također su stvoreni direktoriji nazvani static u kojemu se nalaze poddirektoriji za .css datoteke te poddirektoriji za fotografije.

U Django predlošcima koriste se varijable i oznake za postizanje dinamičkog sadržaja. Varijable dohvaćaju informacije iz pogleda kroz context i omeđene su dvostrukim vitičastim zagradama. S druge strane, oznake mogu omogućiti grananja ili petlje, dohvaćati sadržaj iz baze podataka ili omogućiti pristup drugim oznakama, a označavaju se na način prikazan slikom 4.10.

```
38      <div class="grid-left">
39          <h2>Tvoj oblik lica kao rezultat detekcije treniranog modela je: <strong class="result-text">{{ classification_result }}</strong>.</h2>
40          <h2>Tvoj oblik lica prema računanju karakteristika lica je: <strong class="result-text"> {{ landmarks_result }}</strong>.</h2>
41          <h2>{{ shape_description }}</h2>
42          <a href="{% url 'upload' %}" class="upload-btn">Učitaj novu fotografiju</a>
43      </div>
```

Sl. 4.10. Način označavanja varijabli i oznaka u predlošku

4.6. Putanje

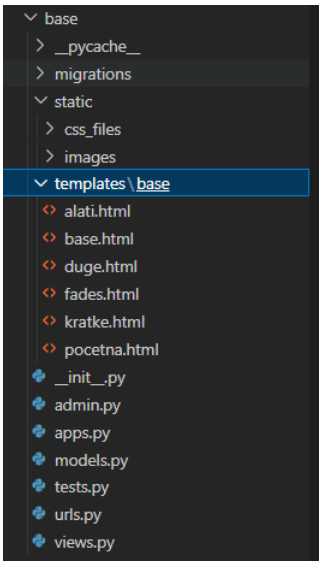
Kako bi aplikacija bile funkcionalna, potrebno je dodati urls.py datoteku koja će, slično datoteci urls.py unutar direktorija projekta, deklarirati URL veze (putanje) za odgovarajuću aplikaciju. Kao što je prikazano slikom 4.11., svaka se putanja definira funkcijom path kojom se, pomoću njenih argumenata, definira naziv putanje, poveže ju s odgovarajućim pogledom te postavlja naziv putanje. Naziv putanje koristi se na primjer, prilikom pisanja oznake predloška koja predstavlja gumb, odnosno vezu do određene stranice.

```python
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.pocetna, name='pocetna'),
    path('kratke/', views.kratke, name='kratke'),
    path('duge/', views.duge, name='duge'),
    path('fades/', views.fades, name='fades'),
    path('alati/', views.alati, name='alati'),
    path('detector/', include('detector.urls')),
    path('accounts/', include('accounts.urls')),
]

```

Sl. 4.11. Putanje base aplikacije

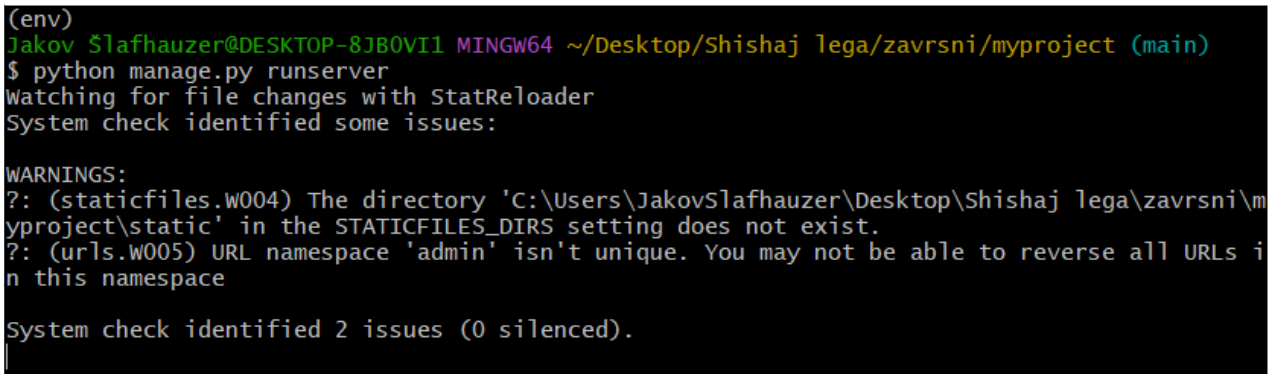
Nakon dodavanja direktorija s predlošcima, direktorija sa stilskim datotekama i fotografijama te datoteke s putanjama, struktura base aplikacije izgleda kako je prikazano na slici 4.12.



Sl. 4.12. Struktura base aplikacije

5. IZGLED I RAD APLIKACIJE

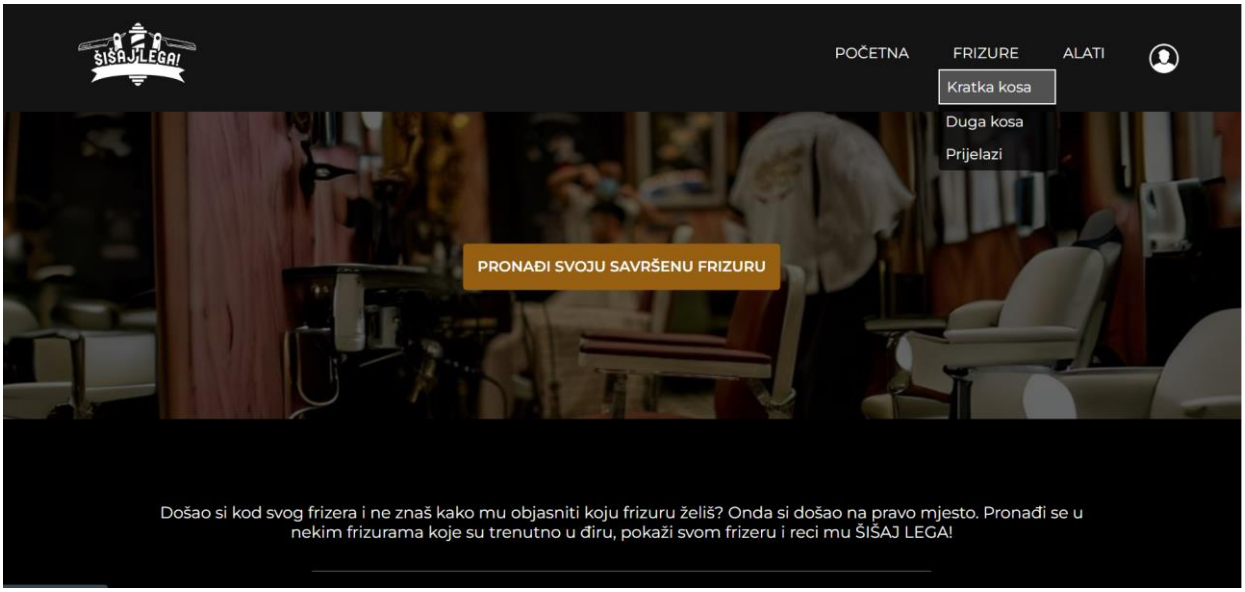
Za prikaz aplikacije u web pregledniku nužno je pokrenuti lokalni server naredbom python manage.py runserver, kao što prikazuje slika 5.1.



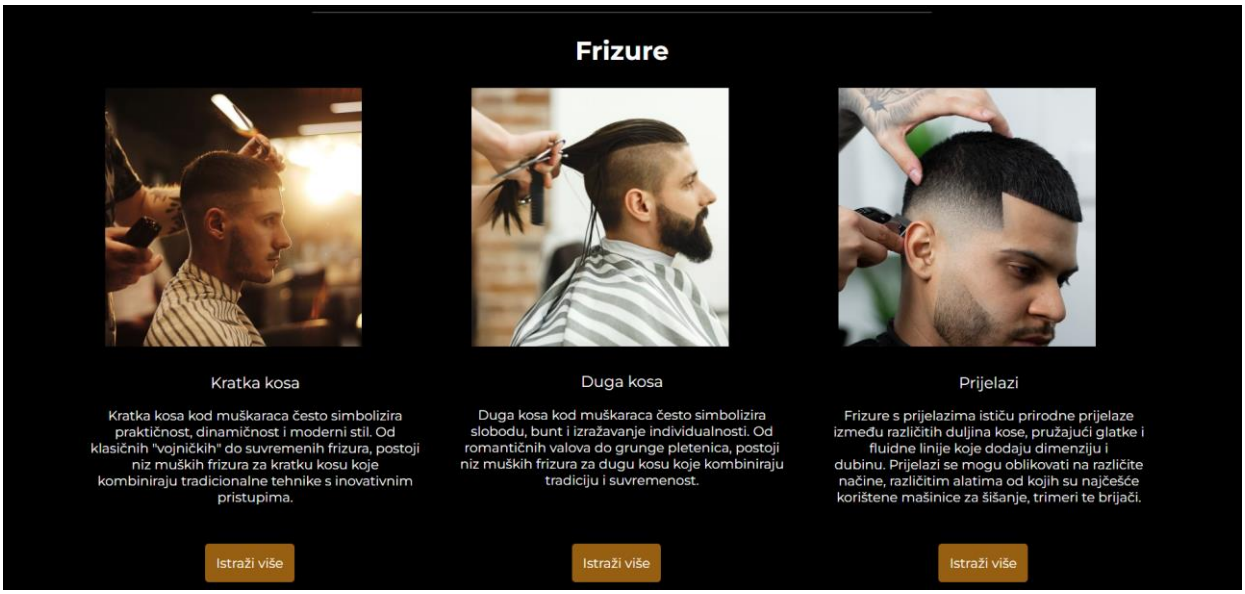
Sl. 5.1. Pokretanje lokalnog servera

5.1. Početno sučelje

Pokretanjem aplikacije u web pregledniku dočekuje nas početna stranica s navigacijskom trakom, gumbom koji vodi do stranice detekcije oblika lica i prijedloga frizura te kratkog opisa tri glavne kategorije frizura. Navigacijska traka pojavljuje se na svakoj stranici aplikacije, a sadrži logo stranice koji je ujedno i poveznica za početnu stranicu, poveznice za početnu stranicu, padajući izbornik sa stranicama kategorija frizura (prikazano na slici 5.2.), poveznica za stranicu s alatima te ikona profila, što predstavlja padajući izbornik za registraciju i prijavu korisnika. Kao što prikazuje slika 5.3. ispod gumba nalaze se kratki opisi glavnih kategorija frizura te gumbovi kao poveznice do svake od stranica, što je alternativa padajućem izborniku navigacijske trake.

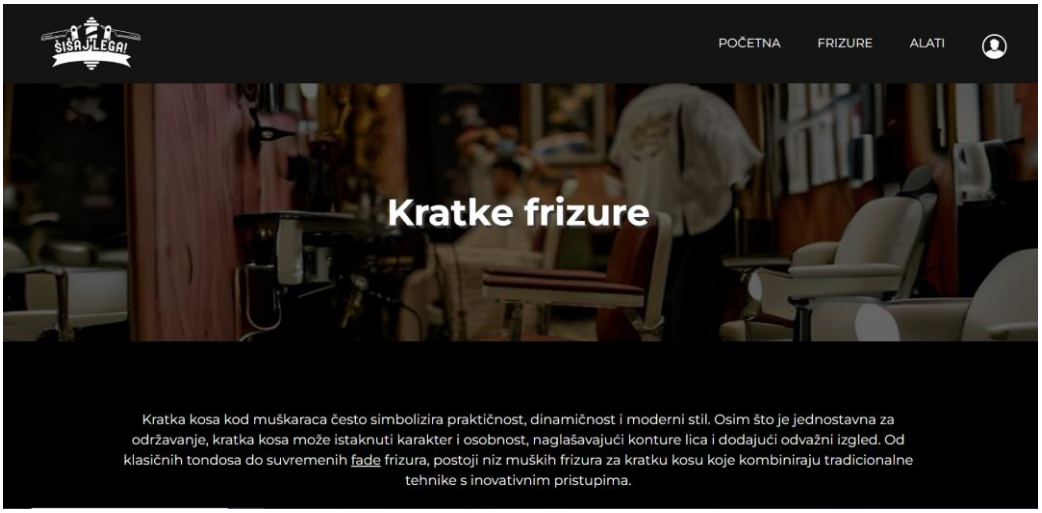


Sl. 5.2. Početna stranica web aplikacije

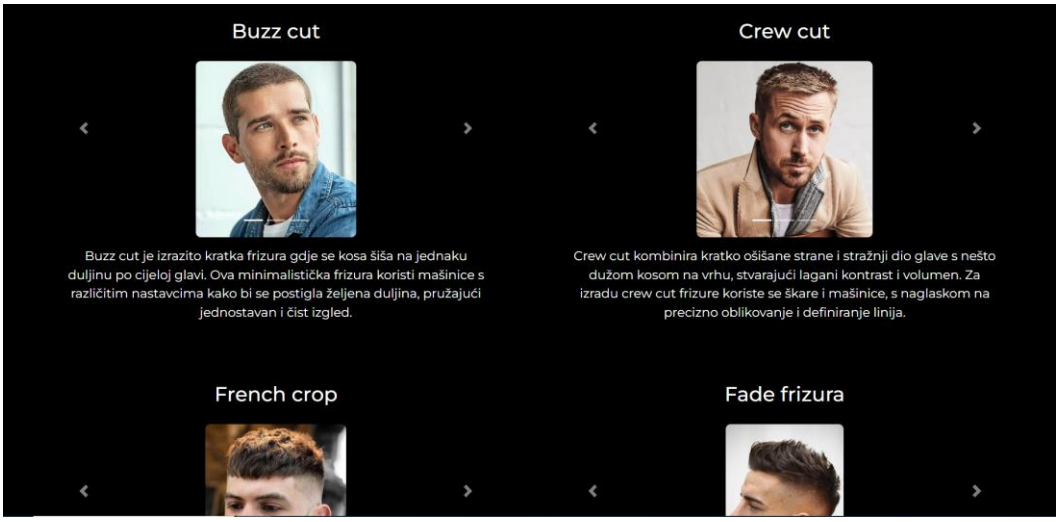


Sl. 5.3. Početna stranica web aplikacije

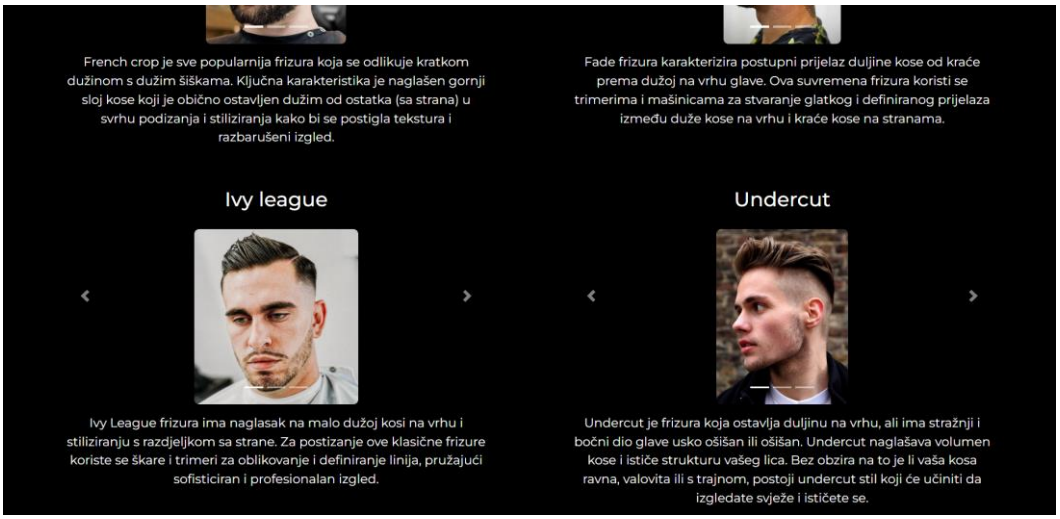
Odlaskom na stranicu Kratka kosa otvara se stranica prikazana slikama 5.4., 5.5. i 5.6. Jednako kao stranice Duga kosa i Prijelazi, ova stranica nudi niz odgovarajućih frizura s pripadajućim opisima i slikama kako bi korisnik mogao dobiti inspiraciju za svoju sljedeću frizuru.



Sl. 5.4. Stranica za kratke frizure

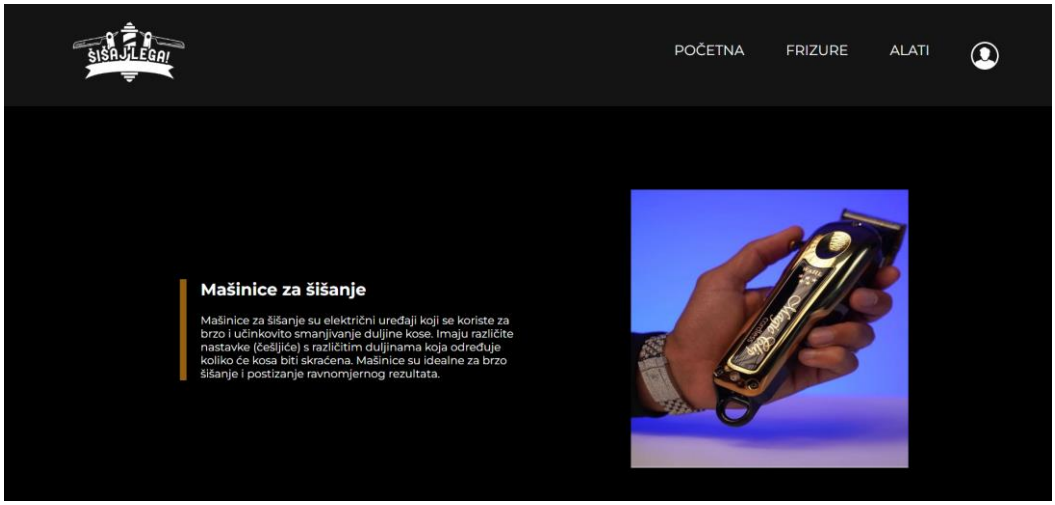


Sl. 5.5. Opis kratkih frizura



Sl. 5.6. Opis kratkih frizura

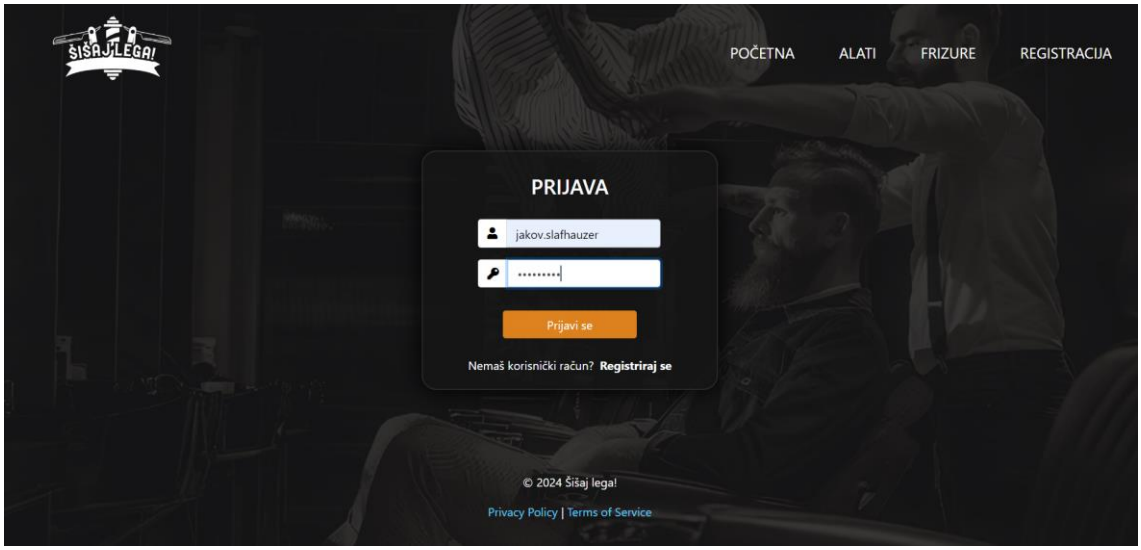
Pritiskom na gumb alati, otvara se stranica prikazana slikom 5.7. na kojoj su opisani alati korišteni za postizanje određenih frizura kako bi korisnik naučio što mu je sve potrebno za samostalno šišanje ili kako bi se mogao lakše dogovoriti s frizerom oko postizanje željene frizure.



Sl. 5.7. Stranica s opisima alata za šišanje kose

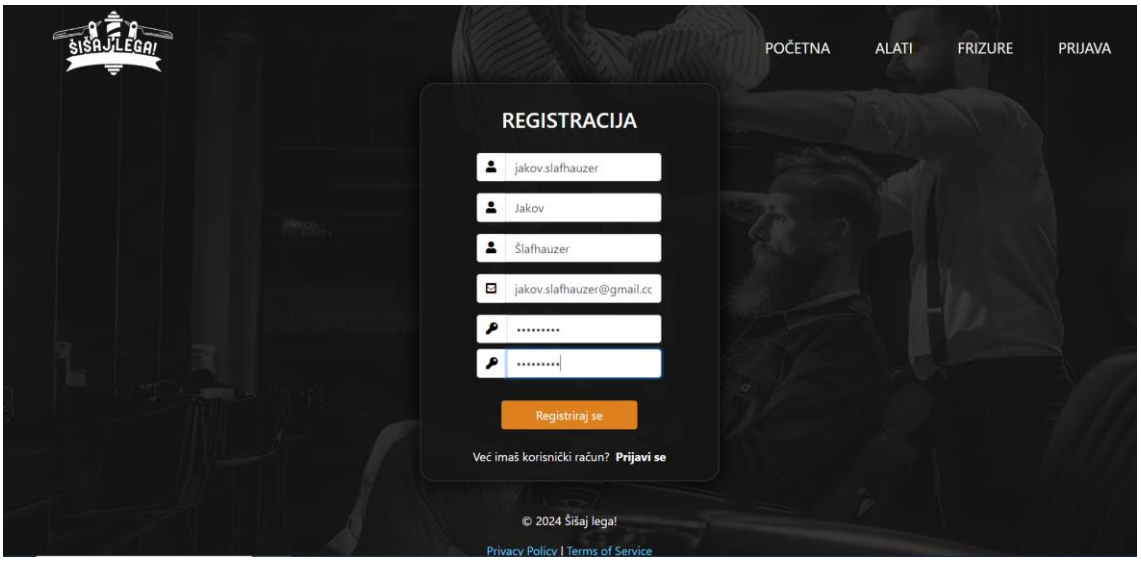
5.2. Django autentifikacija

Kako bi korisnik uspješno obavio detekciju oblika lica, mora najprije biti prijavljen sa svojim korisničkim računom. Ako nije prijavljen, a pokuša otvoriti stranicu detekcije pritiskom na gumb „Pronađi svoju savršenu frizuru“ na početnoj stranici, aplikacija će ga preusmjeriti na stranicu prijave, koja je prikazana slikom 5.8.



Sl. 5.8. Stranica prijave korisničkim računom

Ukoliko korisnik nema postojeći korisnički račun, može ga jednostavno napraviti ispunjavanjem forme na stranici registracije, pokazanu slikom 5.9.



Sl. 5.9. Stranica registracije korisnika

5.2.1. Registracija korisničkog računa

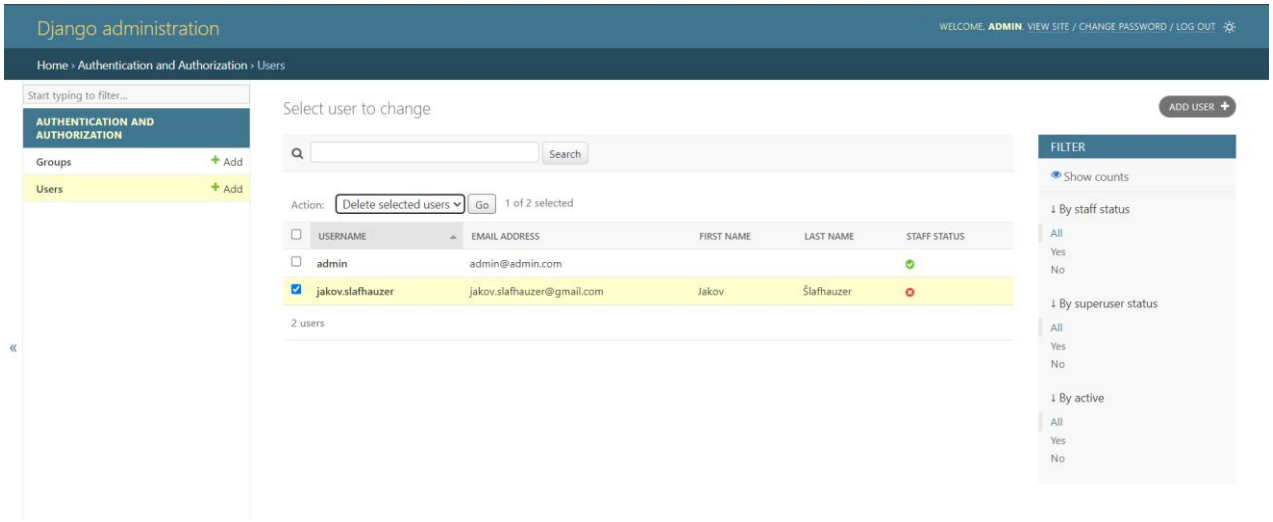
Logika registracije i prijave korisnika postignuta je kroz Djangov autentifikacijski sustav. On potvrđuje da je korisnik ono za što se predstavlja, te određuje što autentificirani korisnik smije učiniti. [8] Pogledom na sliku 5.10. vidi se postupak kreacije korisničkog računa. Forma RegisterUserForm, koja nasljeđuje zadanu UserCreationForm, traži od korisnika unos korisničkog imena (engl. *username*), imena, prezimena, email adresu te lozinku. Zatim se u pogleda funkcijom is_valid() provjerava valjanost ispunjene forme. Ukoliko je forma valjana, korisnički račun se sprema u bazu podataka te se funkcijom authenticate() provjerava autentičnost novostvorenog korisnika. Korisnik se automatski prijavi u sustav te se vraća na početnu stranicu.

```
29  def register_user(request):
30      if request.method == 'POST':
31          form = RegisterUserForm(request.POST)
32          if form.is_valid():
33              form.save()
34              username = form.cleaned_data['username']
35              password = form.cleaned_data['password1']
36              user = authenticate(username=username, password=password)
37              login(request, user)
38              messages.success(request, 'You have successfully registered...')
39              return redirect('pocetna')
40      else:
41          form = RegisterUserForm()
42      return render(request, 'authenticate/register_user.html', {
43          'form': form,
44      })
```

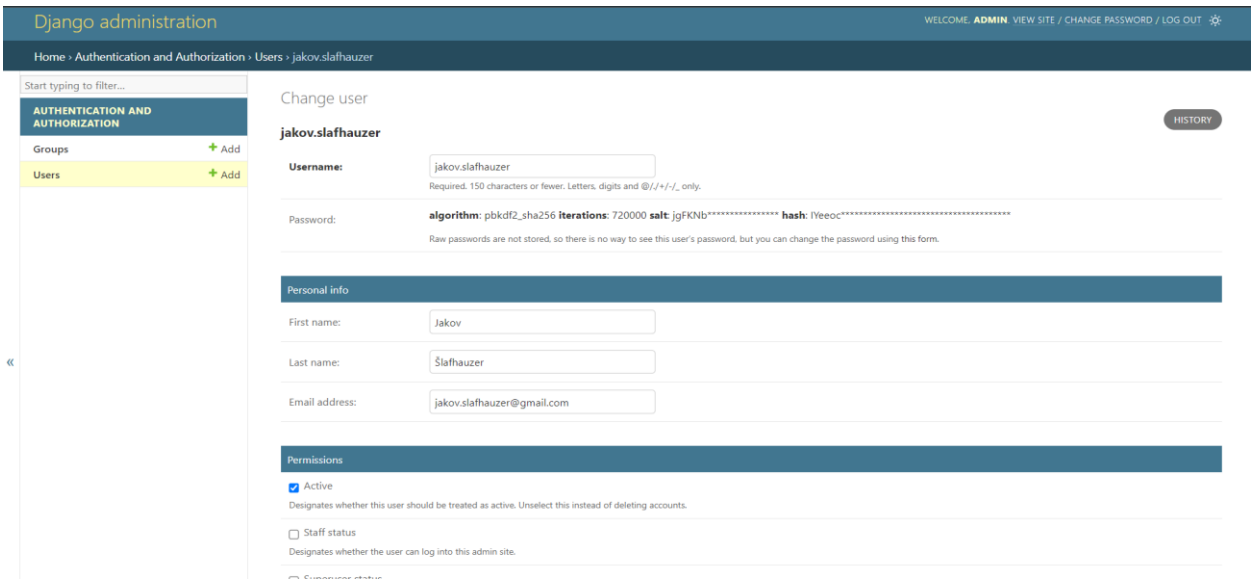

Sl. 5.10. Pogled registracije korisnika

5.2.2. Upravljanje računima kroz administracijsko sučelje

Kreiranje Django superusera (admina) omogućuje upravljanje korisničkim računima kroz administracijsko sučelje. Postupak kreiranja superusera kreće naredbom *python manage.py createsuperuser*. Zatim upišemo željeni naziv superusera (najčešće se za naziv postavi admin), email adresu te lozinku. Nakon prijave pomoću admin računa i ulaska u administracijsko sučelje, odabirom na User poveznicu moguće je vidjeti popis svih registriranih korisničih računa te njihove podatke. Kao što je prikazano slikom 5.11., moguće je odabrati željene račune i obrisati ih. Također, pritiskom gumba Add user u desnom gornjem kutu moguće je ručno kreirati nove korisničke račune. Nadalje, moguće je promijeniti korisničke podatke, koji su prikazani slikom 5.12., pritiskom na željeno korisničko ime.



Sl. 5.11. Prikaz registriranih korisničkih računa



Sl. 5.12. Detalji korisničkog računa

5.2.3. Prijava i odjava korisnika

Funkcije prijave i odjave korisnika prikazane su slikom 5.13. Kada korisnik ispuni formu prijave, funkcijom authenticate() provjerava se postoji li korisnički račun s unesenim korisničkim imenom i lozinkom. Ukoliko ima, korisnik je prijavljen u sustav te se vraća na početnu stranicu.

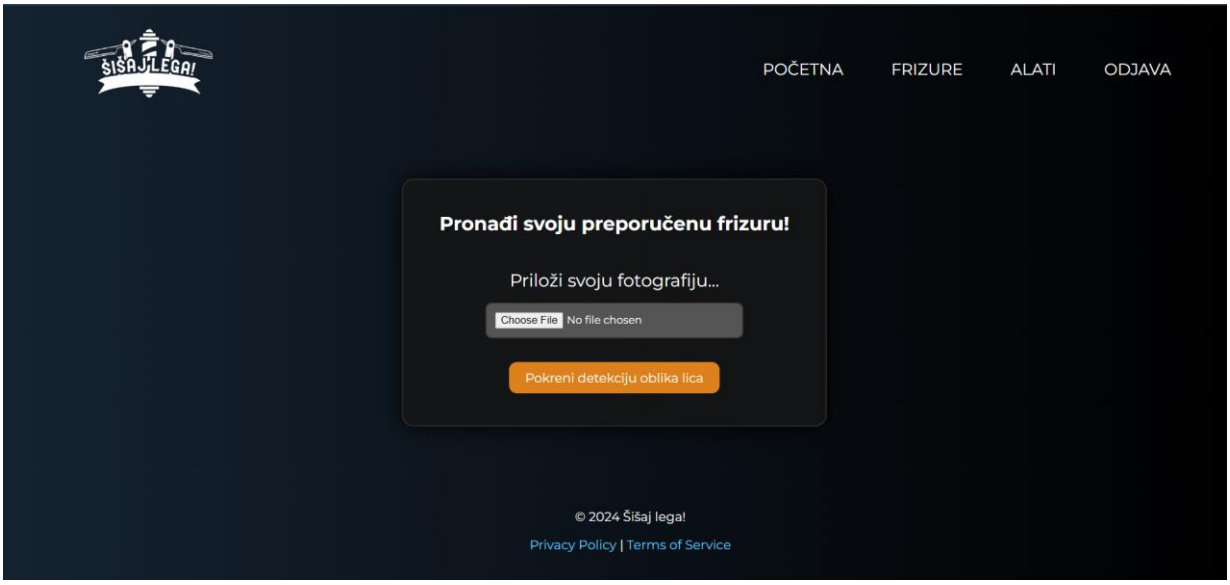
Funkcija odjave nešto je jednostavnija zbog zadane funkcije logout() koja automatski odjavi korisnika na njegov zahtjev.

```
10  def login_user(request):
11      if request.method == 'POST':
12          username = request.POST["username"]
13          password = request.POST["password"]
14          user = authenticate(request, username=username, password=password)
15          if user is not None:
16              login(request, user)
17              return redirect('pocetna')
18          else:
19              messages.success(request, 'There was an error logging in, try again...')
20              return redirect('login')
21
22      return render(request, 'authenticate/login.html', {})
23
    Codeium: Refactor | Explain | Generate Docstring | ×
24  def logout_user(request):
25      logout(request)
26      messages.success(request, 'You have been logged out...')
27      return redirect('pocetna')
```

Sl. 5.13. Funkcije prijave i odjave korisnika

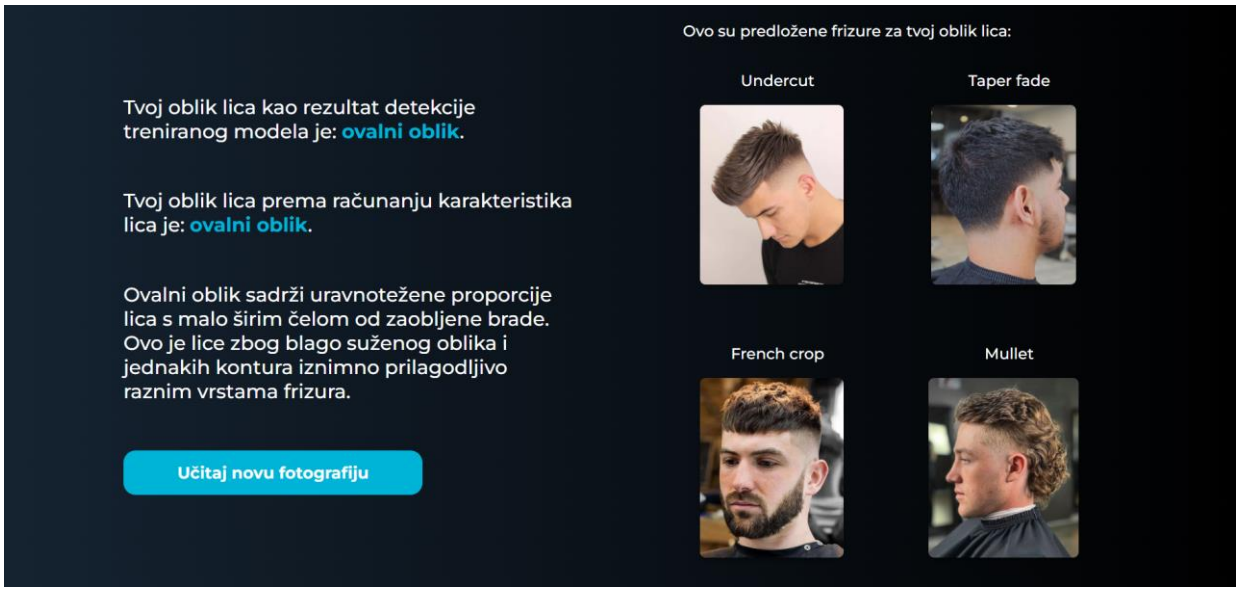
5.3. Prepoznavanje oblika lica

Nakon što se korisnik prijavio u sustav, dopuštena je značajka detekcije oblika lica. Pritiskom gumba na početnoj stranici prikazuje se stranica na kojoj korisnik prilaže svoju fotografiju iz računala, što je prikazano na slici 5.14.



Sl. 5.14. Okvir gdje korisnik prilaže svoju fotografiju

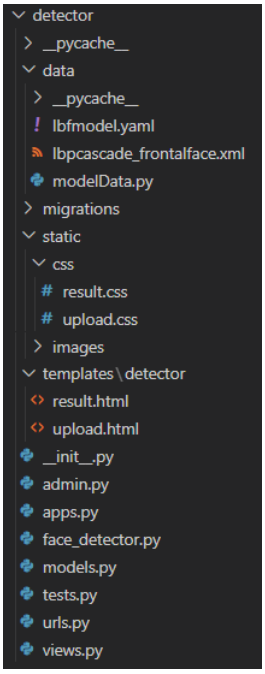
Odabirom željene fotografije i pritiskom na gumb „Pokreni detekciju oblika lica“ započinje glavna funkcionalnost aplikacije. Otvara se stranica s rezultatima detekcije te predložene frizure prema u skladu s odgovarajućim oblikom korisnikova lica. Kao što je prikazano slikom 5.15., detekcija je ostvarena na dva načina, modelom nadziranog učenja te računanjem karakteristika lica priložene fotografije.



Sl. 5.15. Rezultat detekcije oblika lica

5.3.1. Analiza lica učitane fotografije

Za uspješno provođenje detekcija oblika lica i na osnovu treniranog modela i računanjem omjera ključnih točaka (karakteristika) lica, potrebno je uključiti unaprijed trenirane modele za detekciju tih karakteristika lica te OpenCV biblioteku za rukovanje fotografije koje korisnik učita. Za ovu funkcionalnost korišteni su unaprijed trenirani modeli lbpcascade_frontalface.xml i lbfmodel.yaml te trenutno struktura aplikacije detector izgleda kao na slici 5.16.



Sl. 5.16. Struktura aplikacije detector

LBP (engl. Local Binary Patterns) model koristi se za prepoznavanje postojanosti lica u fotografijama, a iznimno je učinkovit u radu u stvarnom vremenu. Treniran je na velikom skupu fotografija koje sadrže i ne sadrže lica. LBP operator uspoređuje intenzitet piksela sa susjednim pikselima i dodjeljuje im binarne vrijednosti na temelju toga jesu li susjedni pikseli svjetliji ili tamniji od središnjeg piksela. Ako je intenzitet susjednog piksela veći od središnjeg, dodijelit će mu se vrijednost 1 te ako je manji, poprimit će vrijednost 0. [9]

LBF (engl. Local Binary Features) model koristi se za određivanje karakterističnih točaka lica kao što su usta, nos, oči, jagodice i slično, kako bi se lakše odredio oblik ili izraz lica. Također radi na LBP principu, što omogućuje prepoznavanje karakterističnih točaka lica u raznim situacijama gdje osvjetljenje ili kut lica nisu idealni. [10] LBF model treniran je na velikom skupu fotografija zvanom 300-W s označenim karakterističnim točkama lica. [11]

Kako bi se učinkovito iskoristili navedeni modeli, kreirana je datoteka face_detector.py koja predstavlja klasu Main sa svim potrebnim računskim funkcijama. U konstruktoru klase postavljeni su LBP i LBF modeli. Funkcija prikazana slikom 5.17. poziva funkcije kreiranja dvaju modela, uzima učitanu fotografiju te ju postavlja na odgovarajuću veličinu kako bi modeli detekcije lica njome mogli rukovati.

```
27      def run_detection_stillshot(self):
28          """
29          Begin face detecion on single image
30          """
31
32          self.create_face_cascade()
33          self.create_LM_detector()
34
35          if self._image is None:
36              return False
37
38          image = cv.imread(self._image)
39
40          target_width = 800
41          ratio = target_width / image.shape[1]
42          image = cv.resize(image, (target_width, int(image.shape[0] * ratio)))
43
44          result = self.detect_and_display(image, self._landmark_detector, "stillshot")
45
46          cv.waitKey(0)
47          return result
48
```

Sl. 5.17. Kod funkcije run_detection_stillshot()

Poziva sljedeću funkciju detect_and_display() koja funkcijama OpenCV biblioteke pretvara fotografiju u sivi spektar te od 68 ključnih točaka koje detektira LBF model uzima najbitnije za izračun potrebnih omjera.

5.3.2. Računanje karakteristika lica

Nakon pronađenih ključnih točaka lica, poziva se funkcija calculate_face_shape() koja kao parametre prima navedene točke i koristi ih za izračun omjera pojedinih karakteristika lica. Na slici 5.18. prikazani su izračuni omjera te predikcija odgovarajućih oblika lica na osnovu granica spomenutih omjera. Također, napravljen je rječnik naziva „hairstyles“ koji za ključeve prima rezultate predikcije, dok vrijednosti predstavljaju predložene frizure za odgovarajuće oblike. Može se primjetiti kako ova metoda detekcije oblika lica nudi više mogućnosti od treniranog modela pa tako, osim spomenuta četiri oblika lica, može detektirati i dijamantni oblik te oblik srca.

```python
def calculate_face_shape(self, cheek, jaw, forehead, chin, head_length, frame, jaw_angle, method):

    cheek_ratio = cheek / head_length
    jaw_ratio = jaw / head_length
    forehead_ratio = forehead / head_length
    chin_ratio = chin / head_length
    head_ratio = head_length / cheek

    result = "Loading..."


    hairstyles = {
        "okrugli oblik": ["crew cut", "ivy League", "srednji prijelaz s većom dužinom na vrhu", "french crop"],
        "ovalni oblik": ["undercut", "taper fade", "french crop", "mullet"],
        "kvadratni oblik": ["buzz cut", "frizure s prijelazom", "undercut", "boho valovi"],
        "izduženi oblik (pravokutnik)": ["ivy League", "french crop", "kraća valovita kosa", "samurai rep"],
        "oblik srca": ["boho valovi", "kraća valovita kosa", "frizure s prijelazom", "taper fade"],
        "dijamantni oblik": ["duža razdijeljena", "burst fade", "undercut", "crew cut"]
    }

    # Round Face
    if (
        0.8 <= cheek_ratio <= 1.0 and
        0.7 <= jaw_ratio <= 0.8 and
        0.6 <= forehead_ratio <= 0.8 and
        0.3 <= chin_ratio <= 0.4 and
        head_ratio <= 1.25 and jaw_angle <= 50.0
    ):
        result = "okrugli oblik"

    # Oval Face
    elif (
        0.5 <= cheek_ratio <= 0.8 and
        0.5 <= jaw_ratio <= 0.7 and
        0.5 <= forehead_ratio <= 0.7 and
        0.2 <= chin_ratio <= 0.4 and
        1.25 <= head_ratio <= 1.6 and jaw_angle > 50.0
    ):
        result = "ovalni oblik"

    # Rectangle Face
    elif (
        0.5 <= cheek_ratio <= 0.8 and
        0.5 <= jaw_ratio <= 0.8 and
        0.5 <= forehead_ratio <= 0.8 and
        0.3 <= chin_ratio <= 0.4 and
        head_ratio >= 1.30 and jaw_angle > 55
    ):
        result = "izduženi oblik (pravokutnik)"

    # Square Face
    elif (
        0.7 <= cheek_ratio <= 0.99 and
        0.7 <= jaw_ratio <= 0.8 and
        0.6 <= forehead_ratio <= 0.99 and
        0.3 <= chin_ratio <= 0.5 and
        head_ratio <= 1.29 and jaw_angle < 55
    ):
        result = "kvadratni oblik"

```

Sl. 5.18. Isječak koda funkcije calculate_face_shape()

5.3.3. Model nadziranog učenja

Kao alternativa klasifikaciji oblika prema fiksno definiranim granicama omjera, izračunati omjeri karakteristika prosljeđuju se treniranom modelu za složenije odlučivanje o klasifikaciji oblika lica. Tijekom pozivanja modela kroz konstruktor mu se predaje niz vrijednosti koje predstavljaju omjere obraza, čeljusti, čela, brade i glave s licem te kut koji čeljust zatvara s horizontalnom osi. Ovaj niz odgovara ulaznim podacima prema kojima je model treniran.

Nadzirano učenje vrsta je strojnog učenja u kojoj se model uči na temelju podataka koji već imaju ulazno-izlazne parove, odnosno za svaki ulaz postoji točno određeni izlaz. Proces nadziranog učenja obuhvaća dvije faze: treniranje i testiranje. U fazi treniranja model se prilagođava na temelju poznatih ulazno-izlaznih parova, dok se u fazi testiranja ocjenjuje koliko dobro model predviđa točne izlaze za nove primjere, pritom koristeći znanje stečeno u fazi treniranja. [12]

Na slikama 5.19. i 5.20. prikazan je dio ulazno-izlaznih parova korištenih za treniranje modela. Ulazi predstavljaju vektore (nizove) već spomenutih omjera karakteristika i kuta koji zatvara čeljust, dok izlazi predstavljaju oblike lica za odgovarajuće vektore u obliku stringa. Može se primjetiti kako ulazi pravilnim redoslijedom odgovaraju, kroz konstruktor, predanom nizu. Ulazni skup podataka izrađen je i prilagođen je prema velikom skupu fotografija celebA. [13] Ovaj model može detektirati četiri vrste oblika lica, a to su: okrugli, ovalni, kvadratni te izduženi oblik lica (oblik pravokutnika).

```
X = [
        [0.8473684210526315, 0.7894736842105263, 0.6789473684210526,
            0.38421052631578945, 1.1801242236024845, 46.54812217260336],
        [0.8387096774193549, 0.7870967741935484, 0.6774193548387096,
            0.38064516129032255, 1.1923076923076923, 47.032069363440364],
        [0.8387096774193549, 0.7870967741935484, 0.6774193548387096,
            0.38064516129032255, 1.1923076923076923, 47.032069363440364],
        [0.8387096774193549, 0.7870967741935484, 0.6774193548387096,
            0.38064516129032255, 1.1923076923076923, 47.032069363440364],
        [0.8315789473684211, 0.7631578947368421, 0.6684210526315789,
            0.37894736842105264, 1.2025316455696202, 44.02197871108006],
        [0.8315789473684211, 0.7631578947368421, 0.6684210526315789,
            0.37894736842105264, 1.2025316455696202, 44.02197871108006],
        [0.875, 0.8, 0.7125, 0.38333333333333336,
            1.1428571428571428, 42.328156473656186],
        [0.875, 0.8, 0.7125, 0.38333333333333336,
            1.1428571428571428, 42.328156473656186],
        [0.875, 0.8, 0.7125, 0.38333333333333336,
            1.1428571428571428, 42.328156473656186],
        [0.8653061224489796, 0.7918367346938775, 0.7387755102040816,
            0.363265306122449, 1.1556603773584906, 40.222310313531715],
```

Sl. 5.19. Ulazni podaci

```
[0.75, 0.7045454545454546, 0.6306818181818182, 0.3806818181818182, 1.3333333333333333, 58.29023163773467],
[0.7570621468926554, 0.711864406779661, 0.6271186440677966, 0.3954802259887006, 1.3208955223880596, 58.891077489471485],
[0.7555555555555555, 0.7111111111111111, 0.6333333333333333, 0.38333333333333336, 1.3235294117647058, 56.76058056789746],
[0.7486338797814208, 0.7213114754098361, 0.6284153005464481, 0.3879781420765027, 1.3357664233576643, 55.19008803832226],
[0.7365591397849462, 0.7043010752688172, 0.6236559139784946, 0.3870967741935484, 1.3576642335766422, 55.74940425160953],
[0.7473118279569892, 0.7096774193548387, 0.6236559139784946, 0.3924731182795699, 1.3381294964028776, 58.03804324364676],
[0.7446808510638298, 0.7180851063829787, 0.6223404255319149, 0.3829787234042553, 1.3428571428571427, 55.34543090725295],
      ]

y = ["okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
            "okrugli oblik",
```

Sl. 5.20. Izlazni podaci

Budući da je u aplikaciji kao izlaz potreban oblik lica, korištena je klasifikacija kao tip nadziranog učenja. Za razliku od regresije koja predviđa kontinuirane vrijednosti, klasifikacijski modeli imaju cilj svrstati izlazne podatke u neku od unaprijed definiranih kategorija (klasa) što su u ovome slučaju oblici lica. Na slici 5.21. prikazan je postupak treniranja modela.

```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import detector.data.modelData as data

Codeium: Refactor | Explain
class PredictShape:
    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, featureVector) -> None:
        self._new_feature = featureVector

    Codeium: Refactor | Explain | ×
    def train_model(self):
        """
        Training a DTC for prediction classification
        """
        X  = data.X
        y = data.y

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=41)

        model = DecisionTreeClassifier()

        model.fit(X_train, y_train)

        new_feature_vector = self._new_feature

        predictions = model.predict([new_feature_vector])
        y_test_predictions = model.predict(X_test)

        accuracy = accuracy_score(y_test, y_test_predictions)

        return [["accuracy", accuracy * 100], [ "prediction", predictions]]

```

Sl. 5.21. Kod treniranja stabla odluke

Kao klasifikator korišteno je stablo odluke. Ono analizira ulazne podatke i dijeli ih na osnovu pojedinačnih karakteristika te tako stvara granice odluke koje vode do predikcije. Svaki čvor u stablu predstavlja odluku na temelju određenih uvjeta, a listovi predstavljaju konačne klasifikacijske odluke. Proces obilaska stabla ponavlja se slijedeći grane stabla sve dok se ne dođe do lista – konačne klase.

Ulazni podaci dohvaćeni su iz datoteke s podacima za treniranje i označeni s X, dok su izlazi spremljeni u varijablu y. Funkcijom train_test_split() dijelimo podatke na 80% trening skupa i 20% testnog skupa. Model je postavljen kao klasifikator stablo odluke te se predviđanje sprema u varijablu predictions pomoću funkcije predict() kojoj se kao parametar predaje new_feature_vector - vektor ulaznih podataka. Rezultat se dalje ispisuje zajedno s rezultatom računanja karakteristika lica.

Ovisno o rezultatu predikcije, web aplikacija korisniku daje opis njegova oblika lica, što je također provedeno u istoj funkciji, a kod ove funkcionalnosti prikazan je slikom 5.22. Ako je prepoznat oblik metodom računanja karakteristika lica, opis oblika bit će vođen rezultatom ove metode jer nudi i podvrste oblika, kao što je dijamantni oblik podvrsta ovalnog oblika lica. Ukoliko ova metoda ne prepozna oblik lica, tada se za opis oblika koristi rezultat treniranog modela.

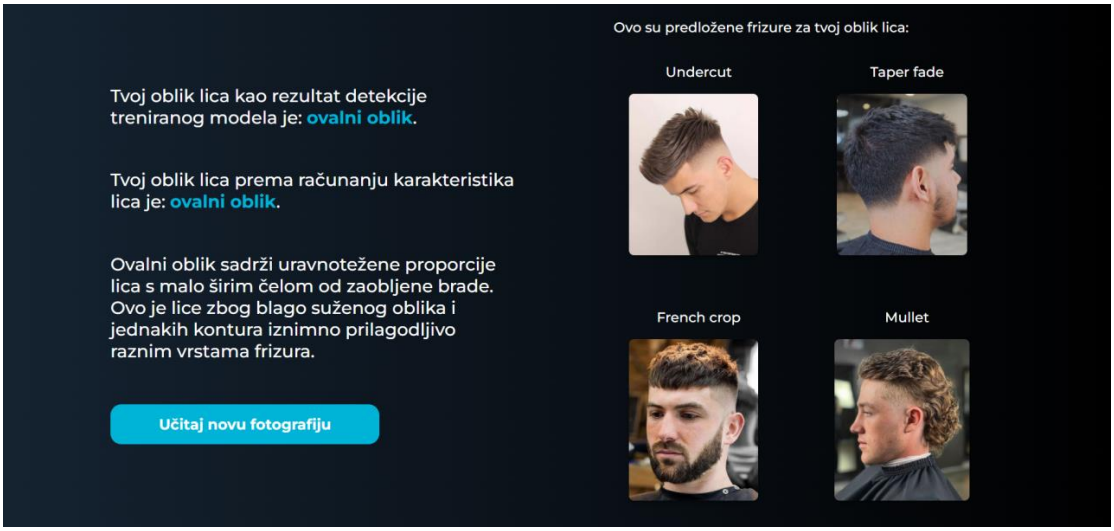
```
if result == "okrugli oblik":
    opis_oblika = " Okrugli oblik lica podrazumijeva mekane i zaobljene crte lica. \
                  Dužina i širina su uglavnom jednake veličine, \
                  dok su jagodice najširi dio lica, \
                  davajući licu zaobljeni oblik bez oštrih kutova."

elif result == "ovalni oblik":
    opis_oblika = " Ovalni oblik sadrži uravnotežene proporcije lica s \
                  malo širim čelom od zaobljene brade. Ovo je lice zbog \
                  blago suženog oblika i jednakih kontura iznimno \
                  prilagodljivo raznim vrstama frizura."
```

Sl. 5.22. Postavljanje logike za ispis opisa oblika lica

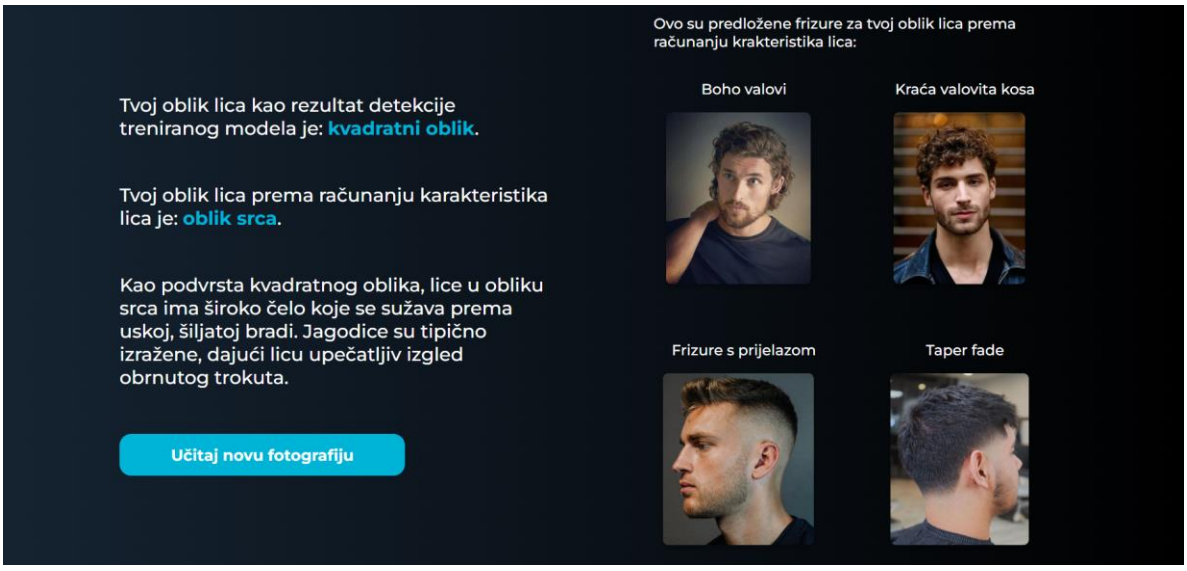
5.4. Pregled predloženih frizura - odlučivanje

Slično kao kod prikazivanja opisa detektiranog oblika lica, prikaz predloženih frizura ovisi o rezultatima dvaju metoda detekcije. Najčešći je slučaj da obje metode ispišu isti rezultat, što je slučaj kao na slici 5.23.



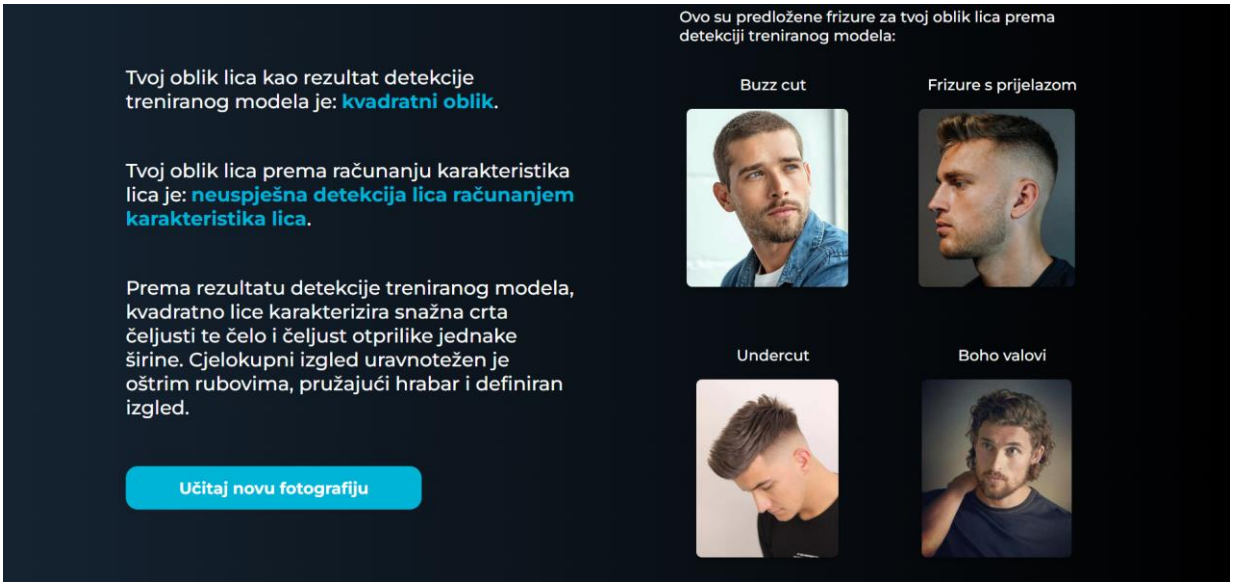
Sl. 5.23. Predložene frizure za jednake rezultate obaju metoda

Međutim, zna se dogoditi da rezultati nisu jednaki, što je uzrokovano većom preciznošću metode računanja karakteristika lica. U tom slučaju se, kao kod opisa oblika, prikazuju predložene frizure prema preciznijoj metodi, kao što je prikazano na slici 5.24.



Sl. 5.24. Predložene frizure prema računanju karakteristika lica

Zadnji slučaj koji se može dogoditi je da metoda računanja ne uspije detektirati oblik lica. Tada se ispisuje prikladna poruka, a frizure će biti predložene prema rezultatu treniranog modela, kao na slici 5.25.



Sl. 5.25. Predložene frizure prema rezultatu treniranog modela

6. ZAKLJUČAK

Ovaj završni rad prikazuje razvoj web aplikacije koja korisnicima omogućuje detekciju oblika lica i predlaganje frizura prema rezultatu metoda detekcije. Kroz primjenu Django razvojnog okvira, aplikacija nudi jednostavno korisničko sučelje i funkcionalnosti poput prijave i registracije korisnika. Implementirane su dvije metode za detekciju oblika lica: model nadziranog učenja i ručno računanje karakteristika lica, a zaključeno je da je trenirani model stabilniji, dok ručno računanje, unatoč nestabilnošću, pruži preciznije rezultate.

Ovaj projekt nastoji olakšati muškarcima samostalno odlučivanje o frizurama, što može biti korisno u vremenu kada su frizerske usluge sve skuplje i teško dostupne. Budući razvoj mogao bi uključivati proširenje aplikacije ženskim frizurama, kako bi ova aplikacija mogla biti korisna i ženama, ali i mogućnost spremanja željene frizure u listu omiljenih frizura.

LITERATURA

[1] HiFace, "Find your face shape," HiFace, [Online]. Available: https://www.hifaceapp.com/face-shape-detector/. [Accessed 28. 6. 2024.].

[2] F. S. Detector, "Accurately Detect Face Shape in Seconds," Face Shape Detector, [Online]. Available: https://faceshapedetectors.com/. [Accessed 28. 6. 2024.].

[3] T. G. Barbers, "Best Haircut and Beard Shape for Your Face – Your ultimate guide," The Good Barbers, [Online]. Available: https://www.thegoodbarbers.ch/post/best-haircut-and-beard-for-your-face. [Accessed 28. 6. 2024.].

[4] V. S. Code, "Why Visual Studio Code," Visual Studio Code, [Online]. Available: https://code.visualstudio.com/docs/editor/whyvscode. [Accessed 6. 9. 2024.].

[5] W3Schools, "Bootstrap Get Started," W3Schools, [Online]. Available: https://www.w3schools.com/bootstrap/bootstrap_get_started.asp. [Accessed 6. 9. 2024.].

[6] GeeksforGeeks, "Python Virtual Environment," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/python-virtual-environment/. [Accessed 10. 9. 2024.].

[7] Django, "Django documentation," Django, [Online]. Available: https://docs.djangoproject.com/en/5.1/intro/tutorial01/. [Accessed 11. 9. 2024.].

[8] Django, "User authentification in Django," Django, [Online]. Available: https://docs.djangoproject.com/en/5.1/topics/auth/. [Accessed 13. 9. 2024.].

[9] A. B. Shetty, Bhoomika, Deeksha, J. Rebeiro and Ramyashree, "Facial recognition using Haar cascade and LBP classifiers," ScienceDirect, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666285X21000728. [Accessed 19. 9. 2024.].

[10] Y. Duan, J. Lu, J. Feng and J. Zhou, "Context-Aware Local Binary Feature Learning for Face Recognition," IEEE, 1. 5. 2018.. [Online]. Available: https://ieeexplore.ieee.org/document/7936534. [Accessed 19. 9. 2024.].

[11] C. Sagonas, E. Antonakos, G. Tzimiropoulos, S. Zafeiriou and M. Pantić, "300 Faces In-The-Wild Challenge: database and results," iBUG, [Online]. Available: https://ibug.cod.ic.ac.uk/resources/300-W/. [Accessed 19. 9. 2024.].

[12] IBM, "What is supervised learning?," IBM, [Online]. Available: https://www.ibm.com/topics/supervised-learning. [Accessed 13. 9. 2024.].

[13] Z. Liu, P. Luo, X. Wang and X. Tang, "Large-scale CelebFaces Attributes (CelebA) Dataset," Multimedia Laboratory, The Chinese University of Hong Kong, [Online]. Available: https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. [Accessed 19. 9. 2024.].

SAŽETAK

U ovome radu prikazane su i objašnjene tehnologije potrebne za izradu web aplikacije za pronalazak odgovarajuće frizure prema obliku lica korisnika. Za izgradnju korisničkog sučelja korišteni su HTML i CSS uz Bootstrap okvir, dok je za razvoj pozadinskog i serverskog dijela aplikacije korišten Django mrežni okvir s programskim jezikom Python. Aplikacija zahtjeva od korisnika učitavanje fotografije iz računala te na osnovu treniranog modela i računanja karakteristika lica nudi niz predloženih frizura na temelju detektiranog oblika lica.

Ključne riječi: Django, frizure, nadzirano strojno učenje, oblik lica, web aplikacija

ABSTRACT

Title: Web application for self-haircut assistance

This final paper presents and explains the technologies required for developing a web application that finds suitable hairstyles based on the user's face shape. HTML and CSS, along with the Bootstrap framework, were used for building the user interface, while the Django web framework with the Python programming language were used for developing the backend and server-side of the application. The application requires users to upload a photo from their computer and based on the trained model and face landmarks calculations, it offers a set of suggested hairstyles according to the detected face shape.

Keywords: Django, haircuts, supervised machine learning, face shape, web application

PRILOZI

Poveznica na GitLab repozitorij projekta: https://gitlab.com/jakov.slafhauzer/zavrsni