Dokumentacija

Autor: Antonio Janach
Datum: 27.2.2023.

Traniranje modela - Pythonu

TensorFlow serviranje omgućava da se trenirani modeli učitaju u TensorFlow kontejnere i da ih se izloži van putem REST API-a. To omogućava da klijenti (web aplikacije, mobilne aplikacije, softver..) mogu poslati podatke modelu koji je serviram putem HTTP zahtjeva i primiti odgovore u JSON formatu. U nastavku biti će objašnjeno kako uspješno provesti TensorFlow serviranje korištenjem REST sučelja.

Za ovaj primjer odabrao sam tranirati model logističke regresije. To je jednostavan model koji dobro funkcionira za primjer jednostavnih predikcija i za ove projektne svrhe.

Prije serviranja samog modela, isti moramo trenirati. Za treniranje modela odlučio sam koristiti, Python. Stoga, uvijek je dobra preporuka kreirati virtualno okruženje za rad u Pythonu.

```
python -m venv env --prompt TFtrain
```

> Glavna svrha Python virtualnih okruženja je stvaranje izoliranog okruženja za različite Python projekte. Na ovaj način može se instalirati određena verzija modula po projektu, bez brige da će to utjecati na druge/ostale Python projekte.

> Ova naredba stvara Python virtualno okruženje s nazivom `env`. Argumentom `--prompt TFtrain` mjenja naziv prompta na `TFtrain`. Ovaj argument omogućava lako prepoznavanje u kojem okruženju radimo.

Sljedeći korak je aktivacija virtualnog okruženja sljedećom naredbom:

```
source env/bin/activate
```

> Ova naredba aktivira virtualno okruženje i postavlja sve varijable okruženja potrebne za rad u virtualnom okruženju.

Sad kad je uspješno aktivirano virtualno kruženje, mogu se instalirati svi potrebni Python moduli.

```
pip install tensorflow
pip install numpy
```

> Prilikom instalacije navedenih paketa/modula pip (sustav za upravljanj paketima napisan u Pythonu) me obavijestio da bi bilo dobro instalirati noviju verziju, stoga sam to i napravio:

> `python.exe -m pip install --upgrade pip`

Dobra preporuka prilikom izrade svakog Python projekta kreirati `requirements.txt` datoteku koja sadrži popis modula/paketa koji su korišteni kako bi Python kôd koji se nalazi u nastavku funkcionirao.

```
pipreqs .
```

> Točka govori to da komanda `pipreqs` spremi sve pakete/module u trenutni radni direktorij. Za instalaciju modula/paketa iz `requirements.txt` datoteke radi se sljedećom komandom `pip install -r requirements.txt`

Može se započeti trenirati model. Kao što je prethodno navedeno odlučio sam tranirati model logističke regresije. U nastavku se nalazi programski kôd koji sam koristio za treniranje logističke regresije i spremio trenirani model u datoteku.

```
import tensorflow as tf
import numpy as np

# Define the logistic regression model
model = tf.keras.Sequential([
  tf.keras.layers.Dense(1, input_shape=(2,), activation='sigmoid')
])

# Define the loss function and optimizer
loss_fn = tf.keras.losses.BinaryCrossentropy()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1)

# Define a training step function to update the model parameters
@tf.function
def train_step(x, y):
    with tf.GradientTape() as tape:
        predictions = model(x)
        loss = loss_fn(y, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

# Generate some sample data for training
x_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_train = np.array([[0], [1], [1], [0]], dtype=np.float32)

# Train the model for a few epochs
for epoch in range(100):
    for x, y in zip(x_train, y_train):
        train_step(np.array([x]), np.array([y]))

# Save the model for serving
tf.saved_model.save(model, "./trainedModel/1/")
```

Kad se Python kôd pokrene `python modelTrain.py` sprema se trenirani model, datoteka ima sljedeću strukturu pohrane:

```
tree /F trainedModel
```

> Koristio sam tree naredbu na Windows OS-u za izlistavanje cijele strukture direktorija, jer mi je tako bilo jednostavnije.

Izlaz navedene naredbe:

```
(TFtrain) PS D:\OneDrive - FIDIT\OneDrive - Sveučilište u
Rijeci\Faks\Diplomski\IPVO\antonio> tree /F .\trainedModel\
Folder PATH listing for volume Posao
Volume serial number is 00000005 3668:1041
D:\ONEDRIVE - FIDIT\ONEDRIVE - SVEUČILIŠTE U
RIJECI\FAKS\DIPLOMSKI\IPVO\ANTONIO\TRAINEDMODEL
└───1
    │   fingerprint.pb
    │   saved_model.pb
    │
    ├───assets
    └───variables
            variables.data-00000-of-00001
            variables.index
```

Sad kad je kreiran trenirani model možemo ga servirati korištenjem TensorFlow kontejnera.

Serviranje treniranog modela korištenjem TensorFlow-a

Prvo je potrebno preuzeti TensorFlow kontejner korištenjem Docker-a. Slika koju sam preuzeo zove se `tensorflow/serving`.

```
docker pull tensorflow/serving
```

Kad imamo sliku kontejnera lako možemo pokrenuti TensorFlow serviranje. Ova naredba pokreće TensorFlow Serving Docker kontejner i servira trenirani model strojnog učenja na određenim portovima.

```
docker run --name "TFserving" -p 8500:8500 -p 8501:8501 --mount
type=bind,source=/home/antonio/Downloads/trainedModel/,target=/models/logistical_regressi
on -e MODEL_NAME=logistical_regression -t tensorflow/serving
```

Evo što svaki dio naredbe radi:

- `--name "TFserving"` dodaje se ime pokrenutom kontejneru
- `docker run`: pokreće Docker kontejner

- `-p 8500:8500 -p 8501:8501`: povezuje TCP portove host stroja s portovima u kontejneru za TensorFlow serviranje
- `--mount type=bind,source=/home/antonio/Downloads/trainedModel/,target=/models/logistical_regression`: montira lokalni direktorij `/home/antonio/Downloads/trainedModel/` u kontejneru na putanju `/models/logistical_regression`, što znači da će model biti dostupan u kontejneru
- `-e MODEL_NAME=logistical_regression`: postavlja varijablu okruženja `MODEL_NAME` na vrijednost `logistical_regression`, što govori TensorFlow serviranju koji model treba servirati
- `-t tensorflow/serving`: odabire sličicu Docker kontejnera za TensorFlow serviranje.

Output naredbe:

```
[antonio@lab ~]$ docker run -p 8500:8500 -p 8501:8501 --mount
type=bind,source=/home/antonio/Downloads/trainedModel/,target=/models/logistical_regressi
on -e MODEL_NAME=logistical_regression -t tensorflow/serving
2023-02-20 22:07:14.425645: I tensorflow_serving/model_servers/server.cc:74] Building
single TensorFlow model file config:  model_name: logistical_regression model_base_path:
/models/logistical_regression
2023-02-20 22:07:14.425979: I tensorflow_serving/model_servers/server_core.cc:465]
Adding/updating models.
2023-02-20 22:07:14.425995: I tensorflow_serving/model_servers/server_core.cc:594]
(Re-)adding model: logistical_regression
2023-02-20 22:07:14.573892: I tensorflow_serving/core/basic_manager.cc:739] Successfully
reserved resources to load servable {name: logistical_regression version: 1}
2023-02-20 22:07:14.573931: I tensorflow_serving/core/loader_harness.cc:66] Approving
load for servable version {name: logistical_regression version: 1}
2023-02-20 22:07:14.573965: I tensorflow_serving/core/loader_harness.cc:74] Loading
servable version {name: logistical_regression version: 1}
2023-02-20 22:07:14.574011: I
external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:45] Reading SavedModel from:
/models/logistical_regression/1
2023-02-20 22:07:14.574698: I
external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:89] Reading meta graph with
tags { serve }
2023-02-20 22:07:14.574734: I
external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:130] Reading SavedModel debug
info (if present) from: /models/logistical_regression/1
2023-02-20 22:07:14.574903: I
external/org_tensorflow/tensorflow/core/platform/cpu_feature_guard.cc:193] This
TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use
the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler
flags.
2023-02-20 22:07:14.594115: I
external/org_tensorflow/tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:357]
MLIR V1 optimization pass is not enabled
2023-02-20 22:07:14.595177: I
external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:229] Restoring SavedModel
```

```
bundle.
2023-02-20 22:07:14.607734: I
external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:213] Running initialization
op on SavedModel bundle at path: /models/logistical_regression/1
2023-02-20 22:07:14.612434: I
external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:305] SavedModel load for tags
{ serve }; Status: success: OK. Took 38420 microseconds.
2023-02-20 22:07:14.612920: I
tensorflow_serving/servables/tensorflow/saved_model_warmup_util.cc:62] No warmup data
file found at /models/logistical_regression/1/assets.extra/tf_serving_warmup_requests
2023-02-20 22:07:14.683971: I tensorflow_serving/core/loader_harness.cc:95] Successfully
loaded servable version {name: logistical_regression version: 1}
2023-02-20 22:07:14.684614: I tensorflow_serving/model_servers/server_core.cc:486]
Finished adding/updating models
2023-02-20 22:07:14.684656: I tensorflow_serving/model_servers/server.cc:118] Using
InsecureServerCredentials
2023-02-20 22:07:14.684676: I tensorflow_serving/model_servers/server.cc:383] Profiler
service is enabled
2023-02-20 22:07:14.685886: I tensorflow_serving/model_servers/server.cc:409] Running
gRPC ModelServer at 0.0.0.0:8500 ...
[warn] getaddrinfo: address family for nodename not supported
2023-02-20 22:07:14.686368: I tensorflow_serving/model_servers/server.cc:430] Exporting
HTTP/REST API at:localhost:8501 ...
[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
```

Kako bi testirali ispravnost učitanog modela u TensorFlow koristićemo alat imena `curl`.

```
[antonio@lab ~]$ curl http://localhost:8501/v1/models/logistical_regression
{
 "model_version_status": [
  {
   "version": "1",
   "state": "AVAILABLE",
   "status": {
    "error_code": "OK",
    "error_message": ""
   }
  }
 ]
}
```

> Kao što vidimo model je uspješno učlitan i nalazi se na sljedećem endpointu:
> `http://localhost:8501/v1/models/logistical_regression`

Zatim je potrebno testirati predikcije treniranog modela:

```
[antonio@lab ~]$ curl -d '{"instances": [[0, 1], [1, 0], [1, 1]]}' -X POST
http://localhost:8501/v1/models/logistical_regression:predict -H "Content-Type:
```

```
application/json"
{
    "predictions": [[0.488026142], [0.485000789], [0.49892047]
    ]
}
```

> Zahtjev se šalje s JSON objektom koji sadrži ključ `instances` s vrijednosti popisa dva elementa, pri čemu svaki element predstavlja jedan primjer koji se želi klasificirati pomoću treniranog modela.

> Odgovor na zahtjev je također u JSON formatu i sadrži ključ `predictions` s vrijednostima koje su predviđanja za svaki primjer iz zahtjeva. Te vrijednosti se nalaze u listi, a svaki element u toj listi predstavlja predviđanje za jedan primjer.

> U ovom primjeru, naredba je poslala tri primjera na serviranje logističke regresije, a servis je vratio predviđanja u obliku vjerojatnosti pripadnosti svakom razredu (u ovom slučaju, postoji samo jedan razred, pa su sve vjerojatnosti za isti razred).

Pokretanje MongoDB kontejnera

Kao sustav pohrane odlučio sam koristiti MongoDB, poznatu kao noSQL bazu podataka. MongoDB se pokazuje kao dobar izbor baze podataka u radu s Django framework-om.

Prije svega potrebno je preuzeti MongoDB sliku kontejnera.

```
docker pull mongo:latest
```

Zatim možemo pokrenuti MongoDB korištenjem sljedeće Docker naredbe:

```
docker run --name my_mongo -p 27017:27017 -d mongo
```

> MongoDB prema zadanome koristi port 27017, stoga ga i pokrećemo na tome portu, odnosno mapiramo port 27017 od kontejnera na port 27017 host mašine

Možemo pokrenuti `docker ps` naredbu da bi se uvjerili da je kontejner uspješno pokrenut.

```
[antonio@lab ~]$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS
PORTS                                                  NAMES
1a3460e6372a   mongo                "docker-entrypoint.s…"   20 minutes ago   Up 20
minutes   0.0.0.0:27017->27017/tcp, :::27017->27017/tcp                  mongo
29e341a89251   tensorflow/serving   "/usr/bin/tf_serving…"   21 minutes ago   Up 21
minutes   0.0.0.0:8500-8501->8500-8501/tcp, :::8500-8501->8500-8501/tcp   TFserving
```

> Vidimo da se mongo kontejner pokreće zajedno sa TF kontejnerom.

Možemo testirati da li MongoDB baza radi tako da se spojimo direktno u pokrenuti kontejner u Bash ljusku:

```
docker exec -it mongo bash
```

> na ovaj način imamo pristup MongoDB bash ljusci

Kad smo spojeni na MongoDB kontejner možemo koristiti naredbu `mongosh` koja omogućava da uđemo u MongoDB shell.

```
[root@lab ~]# docker exec -it mongo bash
root@1a3460e6372a:/# mongosh
Current Mongosh Log ID: 63f4da5e46b03d4a8f3da8de
Connecting to:          mongodb://127.0.0.1:27017/?
directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.6.0
Using MongoDB:          6.0.2
Using Mongosh:          1.6.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-02-21T14:12:55.604+00:00: Access control is not enabled for the database. Read
and write access to data and configuration is unrestricted
   2023-02-21T14:12:55.604+00:00: /sys/kernel/mm/transparent_hugepage/enabled is
'always'. We suggest setting it to 'never'
   2023-02-21T14:12:55.604+00:00: vm.max_map_count is too low
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and
display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL
accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command:
db.disableFreeMonitoring()
------

test>
```

Pristupanjem Mongo Shell-u možemo izlistati sve baze podataka koje se u njoj trenutno nalaze.

```
test> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
test> db
test
test> show collections

test>
```

- `show dbs` Prikazuje listu baza podataka MongoDB kontejnera
- `db` Prikazuje trenutnu bazu podataka koja se koristi
- `show collections` prikazuje listu imena i svih detalja u MongoDB

Backend + Frontend

Ovdje će biti opisana Django web aplikacija. Django web aplikacija služi za dohvaćanje i pohranjivanje predikcija modela logističke regresije. Model logistička regresija se servira pomoću TensorFlow serving API-ja. Rezultati se prikazuju na web stranici, te se ujedno pohranjuju u MongoDB noSQL bazu podataka.

Kako bi postavili navedeno okruženje potrebno je kreirati virtualno Python okruženje, isto kao što smo ga kreirali u 1. koraku kad smo trenirali model logističke regresije. Moduli koje je nužno instalirati da bi web aplikacija radila su sljedeći:

```
asgiref==3.6.0
certifi==2022.12.7
charset-normalizer==3.0.1
Django==4.1.7
djongo==1.3.6
dnspython==2.3.0
idna==3.4
pymongo==4.3.3
requests==2.28.2
sqlparse==0.2.4
urllib3==1.26.14
```

Kad je Django framework instaliran unutar Python-a, možemo pokrenuti `startproject` koji će nam služiti kao template za rad na web aplikaciji.

```
django-admin startproject project
```

Stablo direktorija prikazano je na sljedeći način:

```
.
├── project
```

```
│   ├── db.sqlite3
│   ├── manage.py
│   └── project
│       ├── asgi.py
│       ├── __init__.py
│       ├── models.py
│       ├── __pycache__
│       ├── settings.py
│       ├── templates
│       ├── urls.py
│       ├── views.py
│       └── wsgi.py
└── requirements.txt
```

Kratki opis svakog dokumenta:

- db.sqlite3: SQLite baza podataka koju Django koristi za pohranjivanje podataka.
- manage.py: Skripta za upravljanje Django projektom.
- project/asgi.py: Datoteka za ASGI kompatibilan web server.
- project/**init**.py: Inicijalna datoteka Python paketa koji predstavlja Django projekt.
- project/models.py: Definicija Django modela koji se koriste za pohranu podataka u bazu.
- project/settings.py: Postavke projekta koje sadrže informacije o bazi podataka, statickim datotekama i ostalim parametrima.
- project/templates: Mapa u koju se spremaju predlošci (HTML datoteke) za prikaz na web stranici.
- project/urls.py: Definicije URL-ova (linkova) za povezivanje s pogledima (views).
- project/views.py: Pogledi koji opisuju kako se podaci prikazuju korisniku.
- project/wsgi.py: Datoteka za WSGI kompatibilan web server.
- requirements.txt: Datoteka koja sadrži popis Python paketa koji su potrebni za pokretanje projekta.

Ova web aplikacija ima dvije krajnje točke: '/' i '/read.html', gdje krajnja točka '/' koristi se za unos brojčanih vrijednosti, a krajnja točka '/read.html' prikazuje sve pohranjene predikcije u koje su spremljene u bazu podataka. Ako korisnik unese nevaljani tip podataka, aplikacija će prikazati poruku o pogrešci, dok će valjani unos biti pohranjen u bazu podataka s vremenom unosa i vremenom dohvaćanja odgovora.

Glavni dio kôda Django web aplikacije (`views.py`):

```python
# project/views.py
from django.shortcuts import render
from django.http import HttpResponse
from datetime import datetime
import requests
import json
from pymongo import MongoClient
import uuid
```

```
# Spajanje na MongoDB bazu
client = MongoClient('mongodb://localhost:27017/') # Spajanje na MongoDB
db = client['predictions_db'] # Korištenje baze podataka "predictions_db"
collection = db['kolekcija'] # Korištenje kolekcije "kolekcija" u bazi podataka
collection.create_index('id', unique=True) # Stvaranje jedinstvenog indexa za 'id'

def index(request):
    if request.method == 'POST': # Provjera je li metoda zahtjeva POST
        value1 = request.POST.get('first') # Dohvaćanje vrijednosti 'first' iz HTTP
zahtjeva
        value2 = request.POST.get('seconde') # Dohvaćanje vrijednosti 'seconde' iz HTTP
zahtjeva
        if not value1 and value2: # Provjera jesu li vrijednosti prazne
            error = "Please enter a value." # Poruka o pogrešci ako vrijednosti nisu
unešene
            return render(request, 'index.html', {'error': error}) # Renderiranje
"index.html" predloška s porukom o pogrešci
        try:
            first = [[int(value1)]]
            seconde = [[int(value2)]]
        except ValueError: # Pokretanje u slučaju pogreške konverzije vrijednosti u broj
            error = "Please enter a valid number."
            return render(request, 'index.html', {'error': error})

        # Upis predviđanja u bazu podataka MongoDB
        headers = {'content-type': 'application/json'}
        data = {"instances": [[float(value1), float(value2)]]}
        resp =
requests.post(f'http://localhost:8501/v1/models/logistical_regression:predict',
json=data, headers=headers)

        if resp.status_code != 200: # Provjera jesu li podaci uspješno dohvaćeni od
TensorFlow Serviranja
            error = "Failed to get predictions from TensorFlow Serving: " + resp.text
            return render(request, 'index.html', {'error': error})

        parsed = resp.json() # Parsiranje JSON podataka iz odgovora
        response = parsed["predictions"][0][0]

        # umetanje podataka u MongoDB bazu
        new_prediction = {
            'id': str(uuid.uuid4()), # Kreiranje jedinstvenog ID-a za predviđanja
            'upit': {'value1': value1, 'value2': value2},
            'odgovor': response,
            'vrijemeUpita': datetime.now(), # Spremanje vremena zahtjeva
            'vrijemeOdgovora': datetime.now() # Spremanje vremena odgovora
        }
        collection.create_index('id', unique=True) # Stvaranje jedinstvenog indexa za
```

```
'id'
        collection.insert_one(new_prediction) # Upisivanje novog predviđanja u bazu
podataka

        return render(request, 'index.html', {'response': response}) # Renderiranje
"index.html" predloška s predviđanjem
    else:
        return render(request, 'index.html') # Ako je metoda requesta 'GET', prikazuje se
samo početna stranica.

# View funkcija za čitanje i prikazivanje svih pohranjenih predikcija u bazi podataka.
def read(request):
    data = collection.find() # Dohvaćanje svih predikcija iz baze podataka.
    print(data) # Ispis predikcija u konzoli u svrhu testiranja.
    return render(request, 'read.html', {'data': data}) # Renderiranje stranice za
prikazivanje svih predikcija.
```

> Programski kôd definira funkcije za obradu HTTP zahtjeva te za spremanje i dohvaćanje podataka iz MongoDB baze. U funkciji "index" provjerava se metoda zahtjeva, dohvaćaju se vrijednosti iz HTTP zahtjeva te se šalju na TensorFlow servis za predviđanje. Nakon dobivanja odgovora od TensorFlow servisa, podaci se spremaju u MongoDB bazu. U funkciji "read" se dohvaćaju sve spremljene predikcije iz baze podataka i prikazuju se na stranici za čitanje.

Programski kôd za mapiranje URL-a .html dokumenata:

```
# project/urls.py

# project URL Configuration
# The `urlpatterns` list routes URLs to views. For more information please see:
# https://docs.djangoproject.com/en/4.1/topics/http/urls/
# Examples:
# Function views
#     1. Add an import:  from my_app import views
#     2. Add a URL to urlpatterns:  path('', views.home, name='home')
# Class-based views
#     1. Add an import:  from other_app.views import Home
#     2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
# Including another URLconf
#     1. Import the include() function: from django.urls import include, path
#     2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))

from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    # Map the root URL of the application to the `index` view
    path('', views.index, name='index'),
```

```
    # Map the `read.html` URL to the `read` view
    path('read.html', views.read, name='read')
]
```

> `urlpatterns` je lista ruta koja se koristi za mapiranje URL-ova. U ovom konkretnom primjeru, prva ruta mapira početnu URL adresu (root) na view `index`, dok se druga ruta odnosi na URL adresu `read.html` koja se mapira na view `read`. Ovaj kod zahtijeva da se pozivajući `views` modul mora importirati. Osim toga, `name` atribut se koristi za identificiranje URL-ova.

HTML programski kôd koji se koristi za prikaz forme na frontendu:

```
<html>
  <head>
    <title>Machine Learning Prediction</title>
  </head>
  <body>
    <h1>Unesite vrijednosti za predikciju:</h1>
    <form action="/" method="post">
      {% csrf_token %}
      <textarea name="first"></textarea>
      <textarea name="seconde"></textarea>
      <br><br>
      <input type="submit" value="Predvidi">
    </form>
    {% if error %}
    <h1>Greška:</h1>
    <pre>{{ error }}</pre>
    {% endif %}
    {% if response %}
    <h1>Predikcija:</h1>
    <pre>{{ response }}</pre>
    {% endif %}

    <a href="read.html"><button>Predikcije</button>
  </body>
</html>
```

> HTML forma s dvije tekstualne okvirne za unos vrijednosti za predikciju, a zatim ima gumb za predviđanje. Ako se dogodi greška tijekom predviđanja, prikazuje se poruka o grešci, a ako se predikcija uspješno dovrši, prikazuje se rezultat predikcije. Na kraju, postoje poveznice koje korisnika vode na drugu stranicu za prikaz svih prethodno pohranjenih predikcija.

HTML programski kôd koji se koristi za prikaz podataka iz baze podataka.

```
<html>
  <head>
```

```
    <title>Predictions</title>
  </head>
  <body>
    {% if data %}
      <h1>Data:</h1>
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>upit</th>
            <th>odgovor</th>
            <th>vrijemeUpita</th>
            <th>vrijemeOdgovora</th>
          </tr>
        </thead>
        <tbody>
          {% for prediction in data %}
            <tr>
              <td>{{ prediction.id }}</td>
              <td>{{ prediction.upit}}</td>
              <td>{{ prediction.odgovor }}</td>
              <td>{{ prediction.vrijemeUpita }}</td>
              <td>{{ prediction.vrijemeOdgovora }}</td>
            </tr>
          {% endfor %}
        </tbody>
      </table>
    {% endif %}
    <a href="/"><button>Go back.</button></a>
  </body>
</html>
```

> HTML kôd za prikaz svih pohranjenih predikcija iz baze podataka. Ako postoje predikcije, prikazuju se u obliku tablice s ID-om predikcije, ulaznim vrijednostima ('upit'), predviđenom vrijednosti ('odgovor') i vremenima upita i odgovora. U protivnom, ne prikazuje se ništa osim gumba za povratak na početnu stranicu. Ovo se postiže korištenjem Django templating sustava, u kojem se podaci koje je view funkcija poslala u HTML predložak prikazuju koristeći Django sintaksu za varijable i petlje.

Web aplikacija se pokreće pokretanjem naredbe `python project/manage.py runserver` iz terminala unutar Visual Studio Code-a. Kad se aplikacija pokrene, ona je dostupna na: `http://127.0.0.1:8000/`.

```
(djangoApp) [antonio@lab djangoApp]$ python project/manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
```

```
February 24, 2023 - 10:23:57
Django version 4.1.7, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

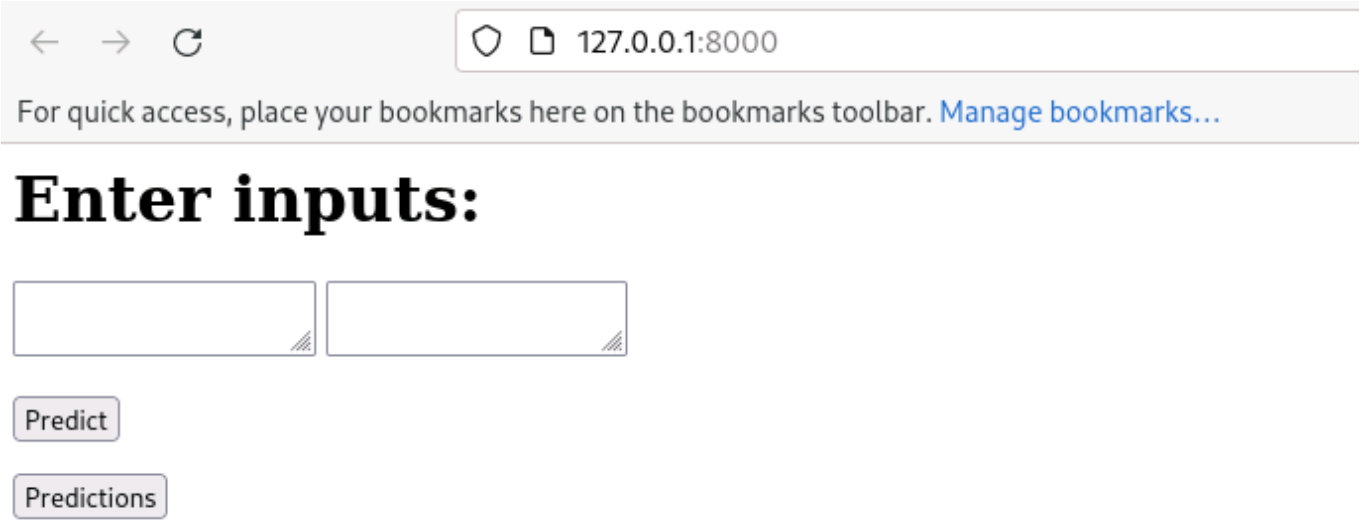
Testiranje rada web aplikacije, serviranja TensorFlow-a te upisa i ispisa iz MongoDB baze podataka

Proces testiranja funkcionalnosti web aplikacije. Testiranje se sastoji od tri glavna koraka: testiranje rada web aplikacije, testiranje TensorFlow serviranja modela i testiranje upisa/ispisa MongoDB baze podataka.

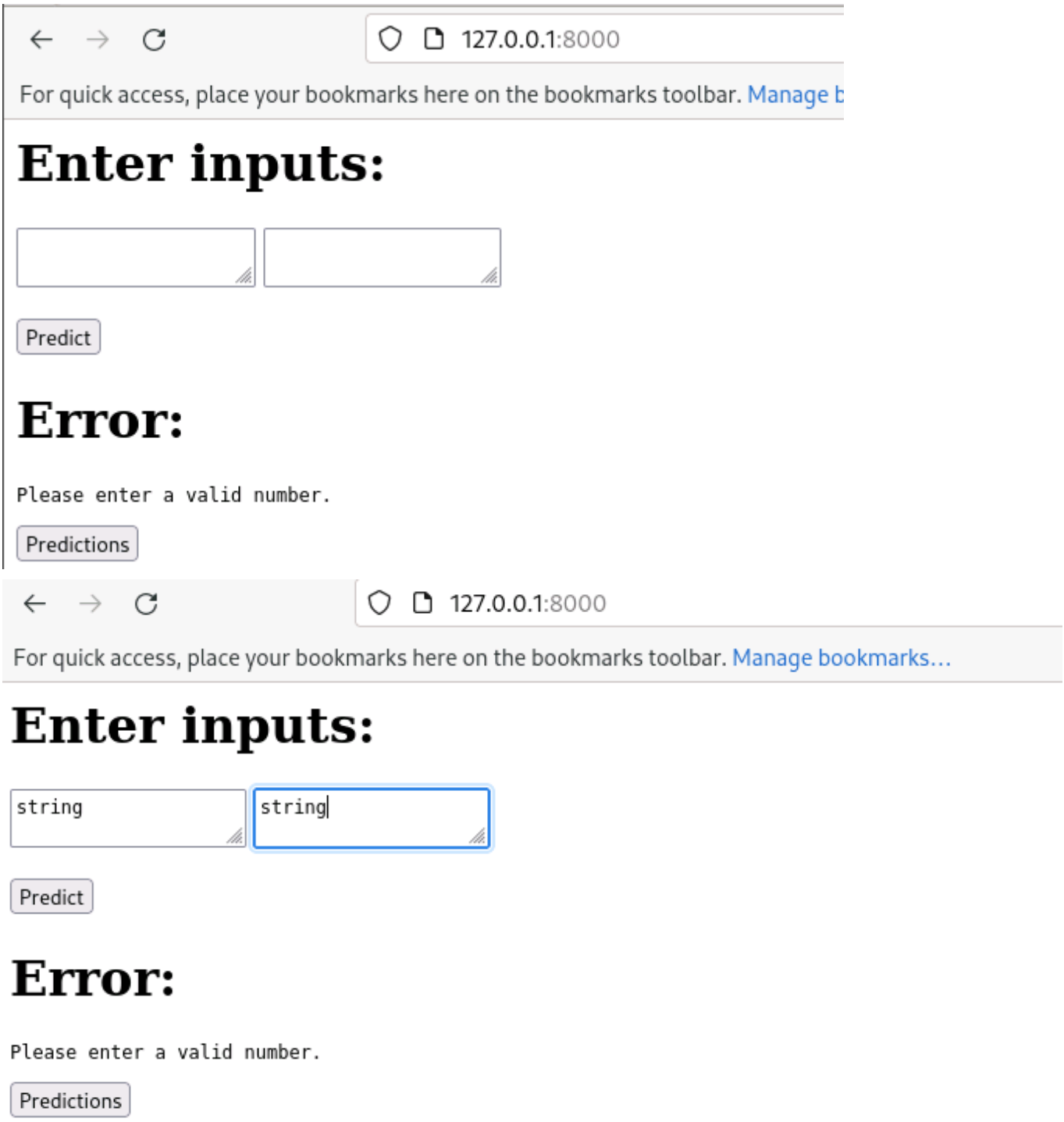
Testiranje rada web aplikacije: Ovaj korak testiranja uključuje sljedeće:

- Sučelja web aplikacije
- Praznog i pogriješnog unosa
- Prikaz predikcija na temelju danog ulaza
- Upis podataka u bazu (upit, odgovor, vrijemeUpita, vrijemeOdgovora)
- Čitanja zapisanih podataka iz baze

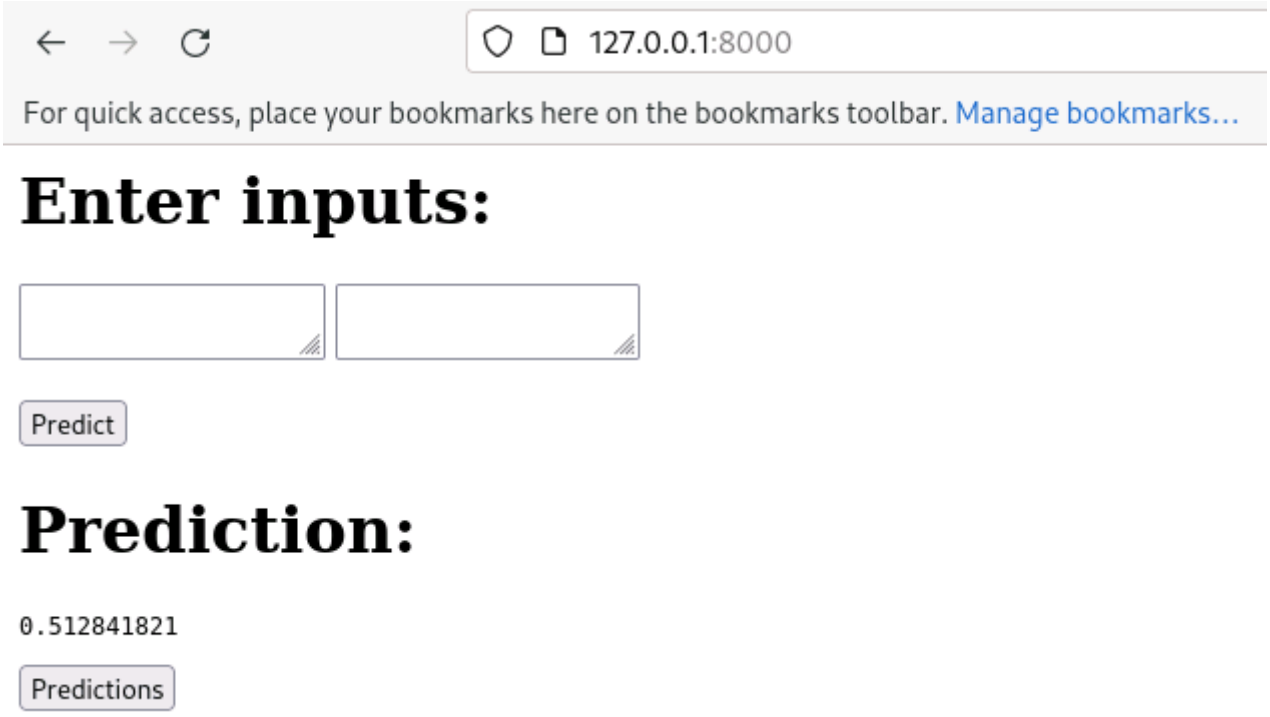
Prikaz sučelja web aplikacije: Sučelje web aplikacije prikazuje se korisniku u svrhu pregledavanja i interakcije s aplikacijom. Korisniku su na raspolaganju određene funkcionalnosti, poput unošenja ulaza i dobivanja odgovora/izlaza.



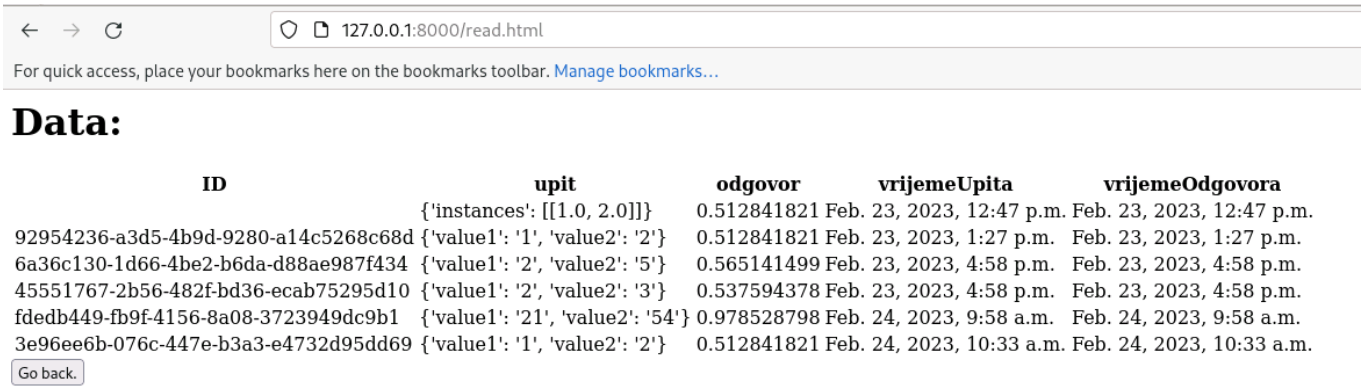
Testiranje praznog i pogriješnog unosa: Cilj ovog testiranja je provjeriti kako web aplikacija reagira na prazan ili pogrešan unos korisnika. Očekuje se da aplikacija korisnika upozori na pogrešan unos i pruži relevantne informacije o tome kako ispraviti pogrešku.



Prikaz predikcija modela na temelju danog ulaza: Ova aktivnost uključuje prikaz predikcija dvaju modela na temelju danog ulaza. Očekuje se da će web aplikacija uspješno predvidjeti odgovor na temelju unesenog ulaza.



Testiranje upisa u bazu podataka: Cilj ovog testiranja je provjeriti funkcionalnost upisa podataka u MongoDB bazu podataka. Očekuje se da će aplikacija uspješno zapisati sve relevantne podatke (upit, odgovor, vrijemeUpita, vrijemeOdgovora) u bazu podataka i da će korisnik moći kasnije čitati te podatke.



Prikaz zapisa prilikom spajanja na MongoDB kontejner:

```
predictions_db> db.getCollectionNames().forEach((c) => { db[c].find().forEach((d) => {
print(c); printjson(d); }); })
predictions_collection
{
  _id: ObjectId("63f75fe8d65f3b34b38def1f"),
  upit: { instances: [ [ 1, 2 ] ] },
  odgovor: 0.512841821,
  vrijemeUpita: ISODate("2023-02-23T12:45:28.399Z"),
  vrijemeOdgovora: ISODate("2023-02-23T12:45:28.399Z")
}
```

```
kolekcija
{
  _id: ObjectId("63f7604cb17695f4211fd9b7"),
  upit: { instances: [ [ 1, 2 ] ] },
  odgovor: 0.512841821,
  vrijemeUpita: ISODate("2023-02-23T12:47:08.136Z"),
  vrijemeOdgovora: ISODate("2023-02-23T12:47:08.136Z")
}
kolekcija
{
  _id: ObjectId("63f769d8bec289a3a87b0b25"),
  id: '92954236-a3d5-4b9d-9280-a14c5268c68d',
  upit: { value1: '1', value2: '2' },
  odgovor: 0.512841821,
  vrijemeUpita: ISODate("2023-02-23T13:27:52.077Z"),
  vrijemeOdgovora: ISODate("2023-02-23T13:27:52.077Z")
}
```

Pokretanje TensorFlow, MongoDB i Django aplikacije pomoću Docker file-a i Docker compose-a

Dockerfile je tekstualna datoteka koja opisuje kako se gradi Docker kontejner. Ona sadrži niz naredbi koje se izvršavaju kada se kontejner gradi, uključujući odabir osnovne slike, kopiranje datoteka u kontejner, pokretanje skripti, postavljanje varijabli okoline i otvaranje mrežnih portova.

Korištenje Dockerfile-a omogućuje programerima i timovima da konzistentno reproduciraju i dijele aplikacije u različitim okruženjima, olakšava implementaciju, distribuciju i upravljanje aplikacijama, i omogućuje izgradnju samodostatnih kontejnera koji se mogu pokrenuti gotovo bilo gdje.

TensorFlow serviranje - dockerfile i docker compose

Kreiranje dockerfile-a za TensorFlow serviranje

```
touch tf_serving_image
vim tf_serving_image
```

```
# Base image
FROM tensorflow/serving

# Set the working directory
WORKDIR /models/logistical_regression

# Copy the trained model to the container
COPY trainedModel/ /models/logistical_regression/

# Set the environment variable for the model name
ENV MODEL_NAME logistical_regression
```

```
# Expose the ports used by TF Serving
EXPOSE 8501
EXPOSE 8501

# Start TF Serving when the container starts
CMD tensorflow_model_server --port=8500 --rest_api_port=8501 --model_name=$MODEL_NAME --
model_base_path=/models/$MODEL_NAME/
```

Dockerfile kôd koristi baznu sliku `tensorflow/serving` za pokretanje TensorFlow Serving servera unutar Docker kontejnera.

Nakon definiranja bazne slike, definiraju se sljedeće naredbe:

- `WORKDIR /models/logistical_regression` - postavlja radni direktorij na `/models/logistical_regression`.
- `COPY trainedModel/ /models/logistical_regression/` - kopira trenirani model sa staze `trainedModel/` u kontejner u direktorij `/models/logistical_regression/`.
- `ENV MODEL_NAME logistical_regression` - postavlja varijablu okruženja `MODEL_NAME` na vrijednost `logistical_regression`.
- `EXPOSE 8500` i `EXPOSE 8501` - izlaže porte `8500` i `8501` na kojima će biti dostupan TensorFlow Serving.
- `CMD tensorflow_model_server --port=8500 --rest_api_port=8501 --model_name=$MODEL_NAME --model_base_path=/models/$MODEL_NAME/` - pokreće TensorFlow Serving server s konfiguracijom da koristi port `8500` za gRPC komunikaciju i port `8501` za RESTful API. Također, definira se varijabla `MODEL_NAME` kao naziv modela koji je ranije postavljen putem `ENV` naredbe i definira se staza do direktorija s modelom koji je kopiran putem `COPY` naredbe.

> Ovaj dockerfile kôd omogućuje da se TensorFlow model spremi u kontejner i pokrene TensorFlow Serving server unutar kontejnera za posluživanje tog modela na definiranim portovima.

Build Docker slike pomoću navedenog DockerFile-a:

```
docker build -t tf_serving_image  -f tf_serving_image .
```

Provjera da li stvarno ta slika kontejnera postoji:

```
[antonio@lab Downloads]$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
tf_serving_image    latest    8ff8bdb6ab34   25 minutes ago   459MB
postgres            latest    680aba37fd0f   13 days ago      379MB
tensorflow/serving  latest    3d0c9b293f81   3 months ago     459MB
mongo               latest    1cca5cf68239   4 months ago     695MB
redis               latest    f8528f17261c   4 months ago     117MB
httpd               latest    d16a51d08814   4 months ago     145MB
```

```
curlimages/curl      latest    dddbb581f872   5 months ago     15.2MB
hello-world          latest    feb5d9fea6a5   17 months ago    13.3kB
```

> postoji

Pokretanje Docker slike na temelju dockerfile-a:

```
docker run -p 8500:8500 -p 8501:8501 --name tfserviranje tf_serving_image
```

Također, kreiran je i `docker-compose.yml` datoteka koja izgleda ova:

```
version: '3'
services:
  tfserviranje:
    image: tf_serving_image
    ports:
      - "8500:8500"
      - "8501:8501"
```

> Docker compose datoteka pokreće se s naredbom `docker compose up`

Provjera da li je navedeni Docker kontejner pokrenuti i da li je omogućeno serviranje treniranog modela:

```
[antonio@lab Downloads]$ docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS
PORTS                                                         NAMES
2e551b4cedde   tf_serving_image   "/usr/bin/tf_serving…"   7 minutes ago   Up 7 minutes
0.0.0.0:8500-8501->8500-8501/tcp, :::8500-8501->8500-8501/tcp   tfserviranje
1a3460e6372a   mongo              "docker-entrypoint.s…"   3 days ago      Up 3 days
0.0.0.0:27017->27017/tcp, :::27017->27017/tcp                   mongo
```

```
[antonio@lab Downloads]$ curl http://localhost:8501/v1/models/logistical_regression
{
 "model_version_status": [
  {
   "version": "1",
   "state": "AVAILABLE",
   "status": {
    "error_code": "OK",
    "error_message": ""
   }
  }
 ]
}
```

> vidimo da sve radi

Isto treba ponoviti za MongoDB i Django framework.

MongoDB - dockerfile

Za MongoDB nije potrebno raditi dockerfile i kreirati image jer ništa ne kopiramo u kontejner već koristimo postojeći image.

No, možemo kreirati `docker-compose` datoteku:

```
version: "3"

services:
  mymongo:
    image: mongo
    ports:
      - "27017:27017"
```

> docker-compose.yaml možemo pokrenuti sa naredbom `docker compose up`

Djagno framework - dockerfile

```
[antonio@lab djangoApp]$ cat dockerfile
FROM python:3.9

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Expose port 8000 for the Django app to run on
EXPOSE 8000

# Start the Django app when the container starts
CMD ["python", "project/manage.py", "runserver", "0.0.0.0:8000"]
```

> dockerfile

```
docker build -t django -f dockerfile .
```

> build slike

```
[antonio@lab djangoApp]$ docker images
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
```

```
django               latest    a1ece8c10b76   4 minutes ago    1.17GB
tf_serving_image     latest    8ff8bdb6ab34   43 minutes ago   459MB
postgres             latest    680aba37fd0f   13 days ago      379MB
tensorflow/serving   latest    3d0c9b293f81   3 months ago     459MB
mongo                latest    1cca5cf68239   4 months ago     695MB
redis                latest    f8528f17261c   4 months ago     117MB
httpd                latest    d16a51d08814   4 months ago     145MB
curlimages/curl      latest    dddbb581f872   5 months ago     15.2MB
hello-world          latest    feb5d9fea6a5   17 months ago    13.3kB
```

> Docker slike kontejnera

Možemo odmah kreirati i docker-compose:

```
[antonio@lab djangoApp]$ cat docker-compose.yml
version: "3"


services:
  web:
    image: django
    ports:
      - "8000:8000"
    volumes:
      - ./djangoApp:/app
    command: python project/project/manage.py runserver 0.0.0.0:8000
```

Zatim pokretanje slike izvršavamo sa naredbom `docker compose up`

Provjera da li su sva tri kontejnera pokrenuta

```
[antonio@lab ~]$ docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED             STATUS
PORTS                                                          NAMES
b055d8ed0c42   django             "python project/mana…"   37 seconds ago      Up 36
seconds      0.0.0.0:8000->8000/tcp, :::8000->8000/tcp                     djangoapp-
web-1
2e551b4cedde   tf_serving_image   "/usr/bin/tf_serving…"   About an hour ago   Up About
an hour   0.0.0.0:8500-8501->8500-8501/tcp, :::8500-8501->8500-8501/tcp   tfserviranje
1a3460e6372a   mongo              "docker-entrypoint.s…"   3 days ago          Up 3 days
0.0.0.0:27017->27017/tcp, :::27017->27017/tcp                 mongo
[antonio@lab ~]$
```

> Vidimo da su sva tri kontejnera pokrenuta te su sve usluge TF serviranja, Djanga i MongoDB-a dostupne.

> Napomena, kontejneri koriste zadanu mrežu