Assignment 3 - Search

Summary	To be able to Implement Visual Search algorithms , implement similarity search using elastic search before developing an app for deployment
Author	Rishvita Bhumireddy, Pranathi Manusanipalli , Jugal Sheth
Github Link	https://github.com/digitalmarketingrpj/Search
Codelab Link	https://codelabs-preview.appspot.com/?file_id=15TzOvW4CdU35qpk2XLzej47dlCY8Ha-T-Qovonv99fY#0
Heroku	https://searchheroku.herokuapp.com/

Introduction

Case:

QU analytics has hired you as an Algorithmic marketing analyst. QU is a consulting organization specializing in marketing analytical solutions. Your client is a large e-tailer (CouchSmart) who has millions of products in its catalog. They intend to enhance the user-experience of their clientele by providing rich and engaging interfaces without leaving their couches! They are considering implementing Visual Search and have reached out to QU Analytics assist in prototyping such a solution.

Data:

We are using Cdiscount's Image Classification Challenge dataset from Kaggle. It consists of bson files

```
category_names.csv
sample_submission.csv
test.bson
train.bson
train_example.bson
```

We are using the train.bson file for our analysis. It consists of 3 columns

1. Category_Id
2. Product_Id
3. Image

Ingestion and Pre-Processing:

- We have decoded the bson file using bson package and worked on creating key value pairs for the image pairs for our purpose
- Sampled the data fro 100 products each in 100 categories, and appending these images to the images file in local
- Created a Product_to_Categories csv

```python
data = bson.decode_file_iter(open('C:/Users/jugal/OneDrive/Desktop/Courses neu/Algorithmic dm/Assig
prod_to_category = dict()
image = []
cat_prod_count = dict()
product_count=1
for c, d in enumerate(data):
    product_id = d['_id']
    category_id = d['category_id'] # This won't be in Test data
    if(category_id in prod_count_3):
        if(category_id in cat_prod_count):
            if(cat_prod_count[category_id]<=100):
                product_count = cat_prod_count[category_id]+1
                cat_prod_count[category_id]=product_count
                prod_to_category[product_id] = category_id
                for e, pic in enumerate(d['imgs']):
                    #picture_count+=1
                    picture = imread(io.BytesIO(pic['picture']))
                    picture_file = os.path.join(images_dir, str(product_id) + '_' + str(e) + '.jpg'
                    plt.imsave(picture_file, picture)

        s            image.append({
                    'ImageName':str(product_id) + '_' + str(e),
                    'ProductId': product_id,
                    'CategoryId': category_id})


            else:
                continue
        else:
            product_count=1
            cat_prod_count[category_id]=product_count
            #print(cat_prod_count)

    else:
        #print("no")
        continue
    #if(prod_to_category.shape[0]==10000):
        #break

with open('image.json', 'w') as out:
        json.dump(image, out)

prod_to_category = pd.DataFrame.from_dict(prod_to_category, orient='index')
prod_to_category.index.name = '_id'
prod_to_category.rename(columns={0: 'category_id'}, inplace=True)
```

Image Search By an Artistic Style (model1):

- Here we used the Brute method for similarity search, using the cosine similarity concept
- The vectors/embeddings having closer cosine distance are more similar to each other
- The algorithm has used convolution neural network and VGG to extract granular pattern/data from the images for further cosine distance analysis, I.e similarity search

```
image_paths = glob.glob('C:/Users/rishv/OneDrive/Northeastern/SEM3/Algorithmic Digital Marketing/Assignments/Assignm
print(f'Founnd [{len(image_paths)}] images')

images = {}
for image_path in image_paths:
    image = cv2.imread(image_path, 3) # 3 represent transperency channel
    b,g,r = cv2.split(image)          # get b, g, r
    image = cv2.merge([r,g,b])        # switch it to r, g, b
    image = cv2.resize(image, (200, 200))
    images[ntpath.basename(image_path)] = image

images_sample ={}
for key, value in images.items():
    images_sample[key]=value
    if(len(images_sample)==100):
        break

n_col = 12
n_row = int(len(images_sample)/n_col)
f, ax = plt.subplots(n_row, n_col, figsize=(16, 8))
for i in range(n_row):
    for j in range(n_col):
        ax[i, j].imshow(list(images_sample.values())[n_col*i + j])
        ax[i, j].set_axis_off()
```

```
Founnd [18069] images
```

```
def search_by_style(reference_image, max_results=10):
    v0 = image_style_embeddings[reference_image]
    distances = {}
    for k,v in image_style_embeddings.items():
        d = sc.spatial.distance.cosine(v0, v)
        distances[k] = d

    sorted_neighbors = sorted(distances.items(), key=lambda x: x[1], reverse=False)

    f, ax = plt.subplots(1, max_results, figsize=(16, 8))
    for i, img in enumerate(sorted_neighbors[:max_results]):
        ax[i].imshow(images[img[0]])
        ax[i].set_axis_off()

    plt.show()
```

Output:

```
#plt.show('25_0.jpg')
search_by_style('25_0.jpg')
```

```
search_by_style('0_0.jpg')
```


```
search_by_style('download.jpg')
```

Spotify-Annoy Method (model 2):

- Annoy(Approximate Nearest Neighbor Oh Yeah), is an open-sourced library for approximate nearest neighbor implementation
- An image feature vector is a list of numbers that represents a whole image, typically used for image similarity calculations or image classification tasks.
- We use this Spotify/annoy library and image feature vectors to calculate the image similarity scores which helps us determine the similar images
- We store the output in a json file which consists information about the similarity scores and the product information
- One productid consists of 20 similar productid based on the similarity score

```
""""""""""""""""""""""""""""""""""""""""""""""""""""
def cluster():

    start_time = time.time()

    print("---------------------------------")
    print ("Step.1 - ANNOY index generation - Started at %s" %time.ctime())
    print("---------------------------------")

    # Defining data structures as empty dict
    file_index_to_file_name = {}
    file_index_to_file_vector = {}
    file_index_to_product_id = {}

    # Configuring annoy parameters
    dims = 1792
    n_nearest_neighbors = 20
    trees = 10000

    # Reads all file names which stores feature vectors
    allfiles = glob.glob('C:/Users/rishv/OneDrive/Northeastern/SEM3/Algorithmic Digital Marketing/A

    t = AnnoyIndex(dims, metric='angular')


    for file_index, i in enumerate(allfiles):
        # Reads feature vectors and assigns them into the file_vector
        file_vector = np.loadtxt(i)

        # Assigns file_name, feature_vectors and corresponding product_id
        file_name = os.path.basename(i).split('.')[0]
        file_index_to_file_name[file_index] = file_name # image name
        file_index_to_file_vector[file_index] = file_vector # the npz vector
        file_index_to_product_id[file_index] = match_id(file_name) # product_id in the json for the

        # Adds image feature vectors into annoy index
        t.add_item(file_index, file_vector)

        print("---------------------------------")
        print("Annoy index     : %s" %file_index) # index of the image
        print("Image file name : %s" %file_name) # image name
        print("Product id      : %s" %file_index_to_product_id[file_index]) # product_id
        print("--- %.2f minutes passed ---------" % ((time.time() - start_time)/60))
```

Output:


```
 { "index" : {"_index": "series"} }
{"similarity": 0.7483, "master_pi": 10022, "similar_pi": 16563}
 { "index" : {"_index": "series"} }
{"similarity": 1.0, "master_pi": 10023, "similar_pi": 10023}
 { "index" : {"_index": "series"} }
{"similarity": 0.8264, "master_pi": 10023, "similar_pi": 113016}
 { "index" : {"_index": "series"} }
{"similarity": 0.8264, "master_pi": 10023, "similar_pi": 45245}
 { "index" : {"_index": "series"} }
{"similarity": 0.8264, "master_pi": 10023, "similar_pi": 71091}
 { "index" : {"_index": "series"} }
{"similarity": 0.8132, "master_pi": 10023, "similar_pi": 20839}
 { "index" : {"_index": "series"} }
{"similarity": 0.8099, "master_pi": 10023, "similar_pi": 30207}
 { "index" : {"_index": "series"} }
{"similarity": 0.8048, "master_pi": 10023, "similar_pi": 13555}
 { "index" : {"_index": "series"} }
{"similarity": 0.8047, "master_pi": 10023, "similar_pi": 74277}
 { "index" : {"_index": "series"} }
{"similarity": 0.8009, "master_pi": 10023, "similar_pi": 29394}
 { "index" : {"_index": "series"} }
{"similarity": 0.799, "master_pi": 10023, "similar_pi": 80388}
 { "index" : {"_index": "series"} }
{"similarity": 0.7976, "master_pi": 10023, "similar_pi": 43972}
 { "index" : {"_index": "series"} }
{"similarity": 0.7962, "master_pi": 10023, "similar_pi": 9659}
 { "index" : {"_index": "series"} }
{"similarity": 0.7954, "master_pi": 10023, "similar_pi": 74924}
 { "index" : {"_index": "series"} }
{"similarity": 0.7934, "master_pi": 10023, "similar_pi": 49498}
 { "index" : {"_index": "series"} }
{"similarity": 0.7926, "master_pi": 10023, "similar_pi": 11145}
 { "index" : {"_index": "series"} }
{"similarity": 0.792, "master_pi": 10023, "similar_pi": 63396}
 { "index" : {"_index": "series"} }
{"similarity": 0.792, "master_pi": 10023, "similar_pi": 83563}
 { "index" : {"_index": "series"} }
{"similarity": 0.7915, "master_pi": 10023, "similar_pi": 76803}
 { "index" : {"_index": "series"} }
{"similarity": 0.7887, "master_pi": 10023, "similar_pi": 48240}
```

FAISS (model 3):

FAISS is a C++ Library(with python bindings) that assures faster similarity searching with the number if vectors may go up to millions or billions
At its very heart lies the index.

Faiss has a Handful of features:

- GPU and multithreaded support for index operations
- Dimensionality reduction:vectors with large dimensions can be reduced to smaller dimensions using PCA
- Quantisation: FAISS emphasises in product quantisation for compressing and storing vectors of large dimensions
- Batch processing

```
model = ResNet50(weights='imagenet', include_top=False,
                 input_shape=(180, 180, 3))
def extract_features(img_path, model):
    input_shape = (180, 180, 3)
    img = image.load_img(img_path, target_size=(
        input_shape[0], input_shape[1]))
    img_array = image.img_to_array(img)
    expanded_img_array = np.expand_dims(img_array, axis=0)
    preprocessed_img = preprocess_input(expanded_img_array)
    features = model.predict(preprocessed_img)
    flattened_features = features.flatten()
    normalized_features = flattened_features / norm(flattened_features)
    return normalized_features
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_t
f_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
```

Function to recursively get all the image files under a root directory.

```
extensions = ['.jpg', '.JPG', '.jpeg', '.JPEG', '.png', '.PNG']
def get_file_list(root_dir):
    file_list = []
    counter = 1
    for root, directories, filenames in os.walk(root_dir):
        for filename in filenames:
            if any(ext in filename for ext in extensions):
                file_list.append(os.path.join(root, filename))
                counter += 1
    return file_list
```

Now, let's run the extraction over the entire dataset and time it.

```
root_dir = './'
files = sorted(get_file_list(root_dir))
```

```
feature_list = []
for i in tqdm_notebook(range(len(files))):
    feature_list.append(extract_features(files[i], model))
```

```
feature_list
```

```
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.00303162], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.00891707], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.00411883], dtype=float32),
 array([0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.00385734], dtype=float32),
 array([0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.00669908], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 0.], dtype=float32),
 array([0.        , 0.        , 0.        , ..., 0.        , 0.        ,
```

```
feature_list=np.array(feature_list)
```

```
feature_list.shape
```

11]: (780, 73728)

```
dimension = 73728    # dimensions of each vector
n = len(files)    # number of vectors
np.random.seed(1)
db_vec = feature_list #np.random.random((n, dimension)).astype('float32')
```

```
db_vec.shape
```

13]: (780, 73728)

```
nlist = 1  # number of clusters
quantiser = faiss.IndexFlatL2(dimension)
index = faiss.IndexIVFFlat(quantiser, dimension, nlist,   faiss.METRIC_L2)
```

```
print(index.is_trained)   # False
index.train(db_vec)  # train on the database vectors
print(index.ntotal)   # 0
index.add(db_vec)   # add the vectors and update the index
print(index.is_trained)  # True
print(index.ntotal)   # 200
```

```
False
0
True
780
```

```
indi_df = indi_df.replace(dfnew)
indi_df
```

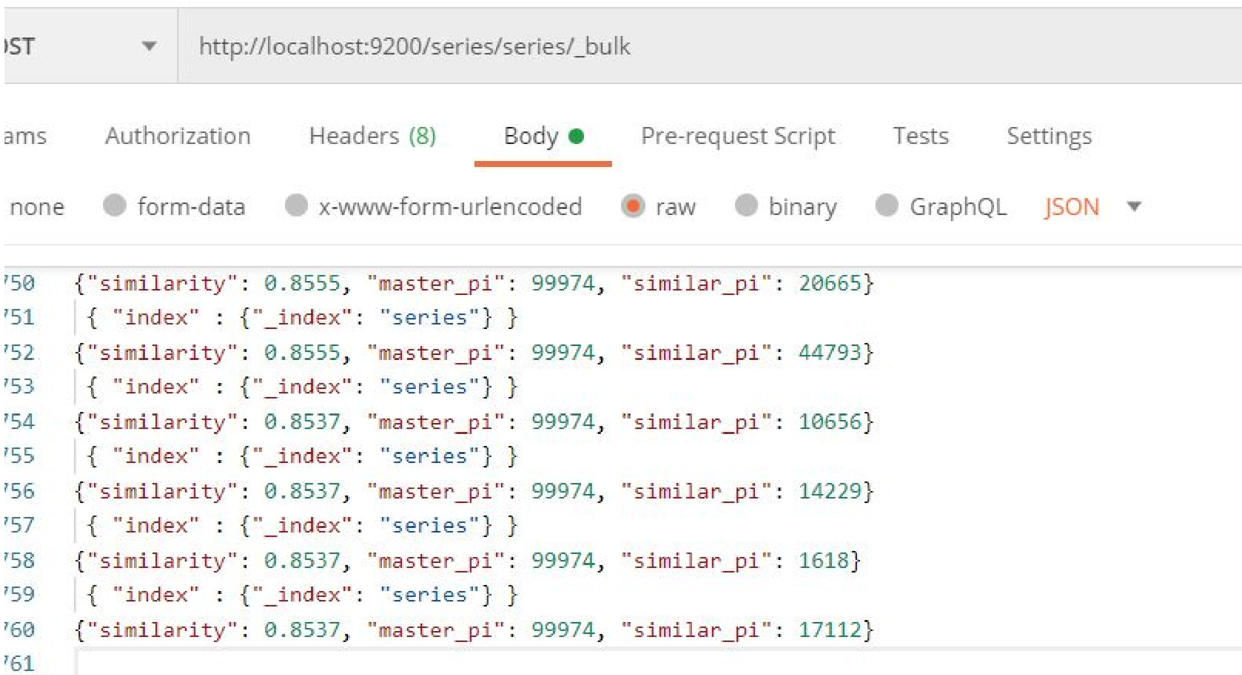
38]:

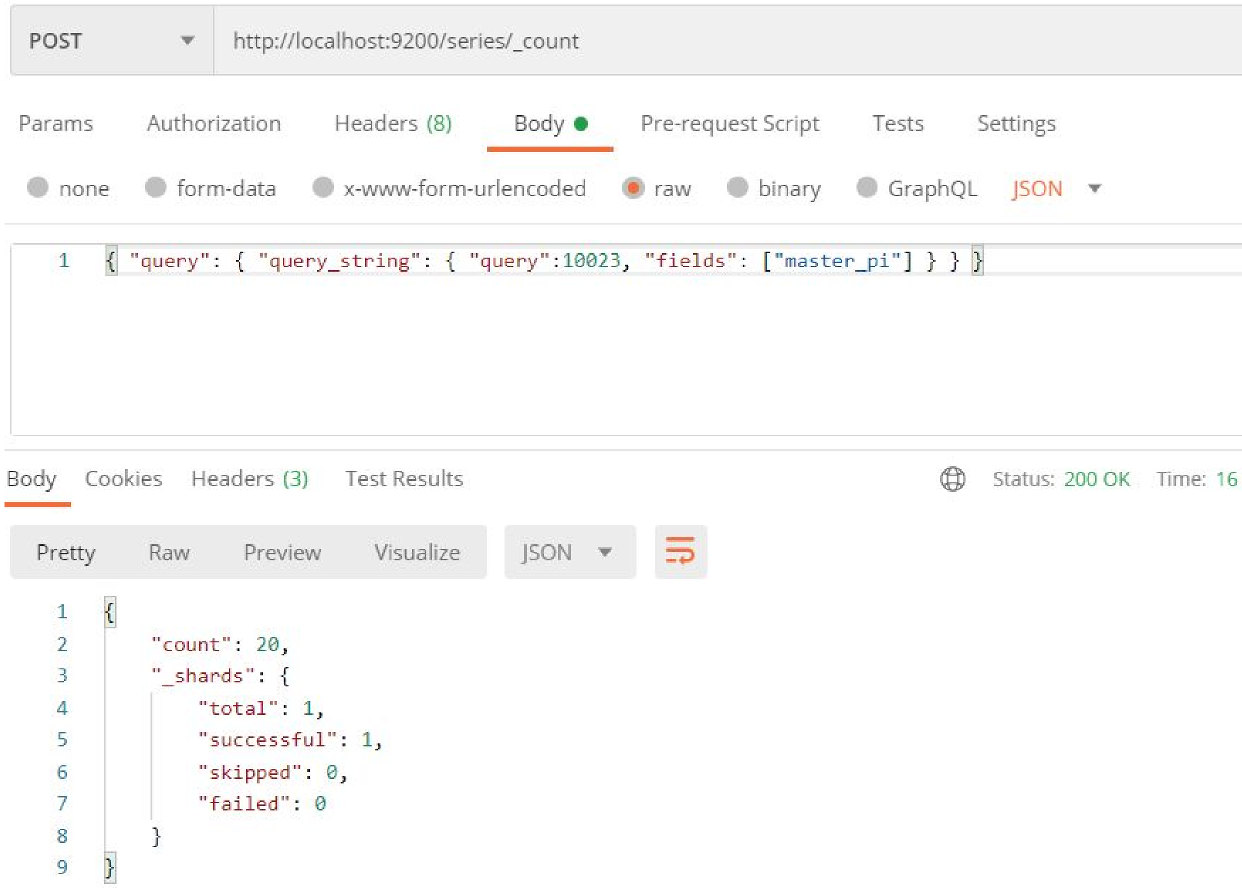
	0	1	2	3	4	5	6	7	8	9
0	1000_0.jpg	203_0.jpg	1399_0.jpg	767_0.jpg	1488_0.jpg	1159_0.jpg	709_0.jpg	575_0.jpg	893_0.jpg	1439_0.jpg
1	1000_1.jpg	1000_0.jpg	1504_0.jpg	1322_0.jpg	893_0.jpg	1439_0.jpg	1399_0.jpg	767_0.jpg	203_0.jpg	1488_0.jpg
2	1000_2.jpg	1467_0.jpg	1048_0.jpg	708_0.jpg	1000_0.jpg	893_0.jpg	1159_0.jpg	709_0.jpg	1149_0.jpg	569_1.jpg
3	1000_3.jpg	1572_2.jpg	219_0.jpg	465_3.jpg	651_0.jpg	685_3.jpg	978_0.jpg	1141_2.jpg	898_2.jpg	1088_3.jpg
4	1002_0.jpg	812_0.jpg	1519_0.jpg	647_0.jpg	752_2.jpg	1070_0.jpg	907_2.jpg	1425_0.jpg	317_0.jpg	440_0.jpg
...	...	...	...	...	...	...	...	...	...	...
775	993_0.jpg	430_0.jpg	814_0.jpg	1426_0.jpg	35_0.jpg	907_3.jpg	410_0.jpg	1009_0.jpg	869_1.jpg	997_0.jpg
776	997_0.jpg	997_3.jpg	733_1.jpg	178_0.jpg	312_0.jpg	177_0.jpg	64_1.jpg	273_0.jpg	898_1.jpg	80_0.jpg
777	997_1.jpg	273_3.jpg	371_3.jpg	177_0.jpg	1242_1.jpg	1170_1.jpg	1180_2.jpg	273_2.jpg	95_0.jpg	371_1.jpg
778	997_2.jpg	973_0.jpg	1166_3.jpg	973_2.jpg	131_0.jpg	124_0.jpg	133_1.jpg	148_2.jpg	79_0.jpg	144_0.jpg
779	997_3.jpg	997_0.jpg	312_0.jpg	64_1.jpg	273_1.jpg	733_1.jpg	273_0.jpg	570_1.jpg	178_0.jpg	280_0.jpg

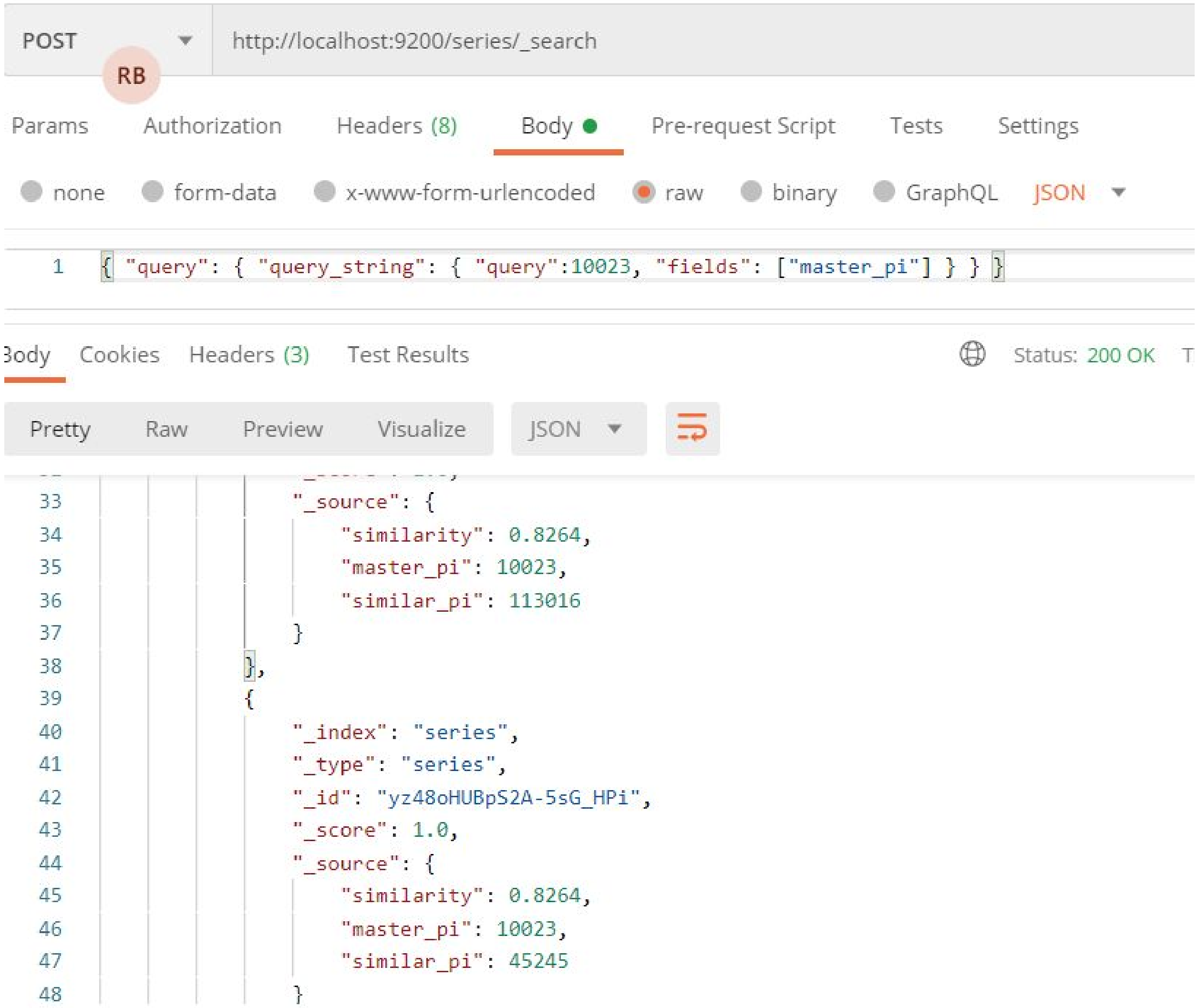
780 rows × 10 columns

ElasticSearch:

- Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time.
- We used the output json file of the Spotify-Annoy Method
- We indexed the json file using POST with the name “series”. We used bulk API to index the data

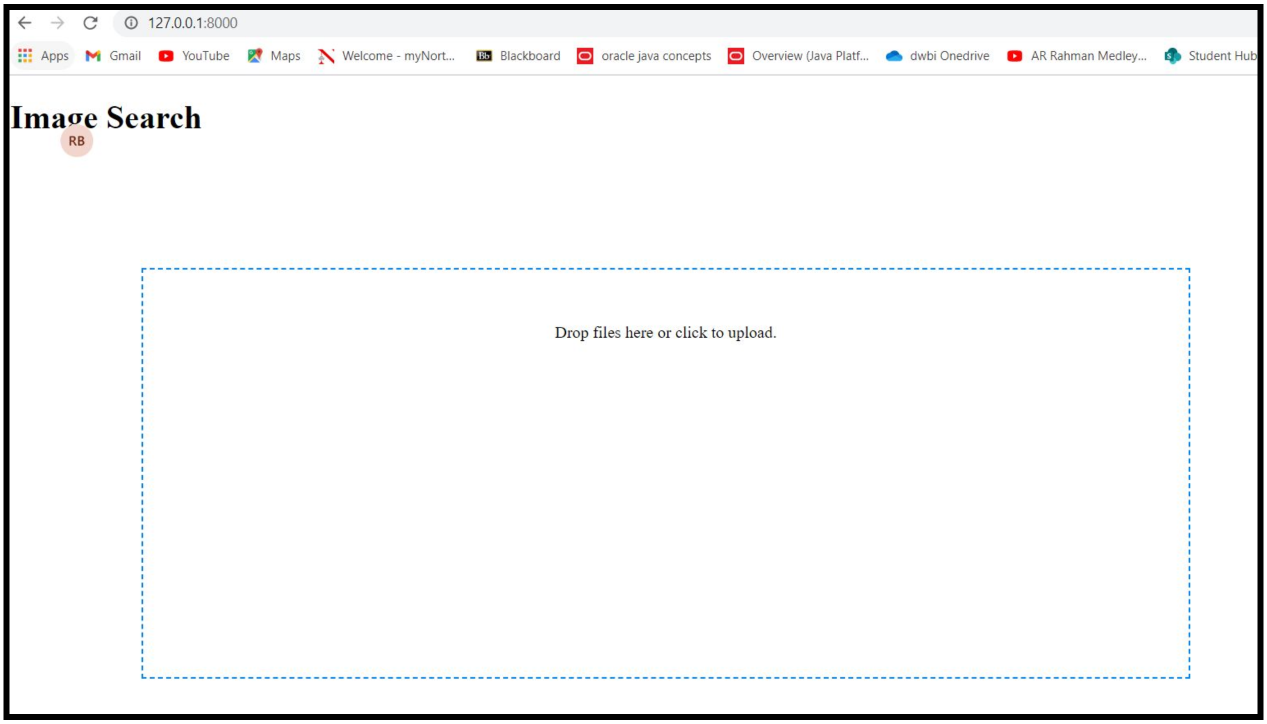


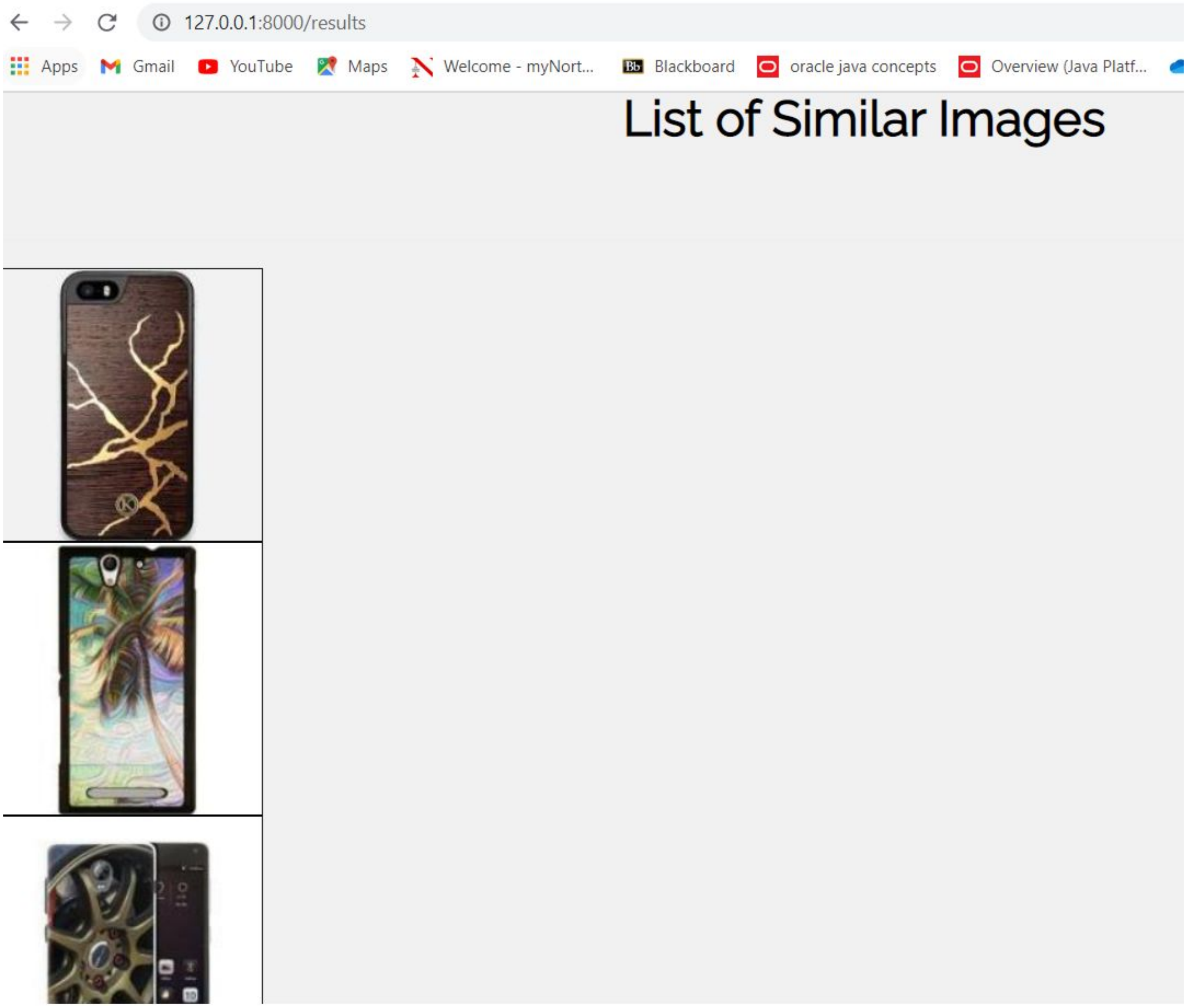

POST
http://localhost:9200/series/_count
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{ "query": { "query_string": { "query":10023, "fields": ["master_pi"] } } }
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Pretty
Raw
Preview
Visualize
JSON
{
"count": 20,
"_shards": {
"total": 1,
"successful": 1,
"skipped": 0,
"failed": 0
}
}



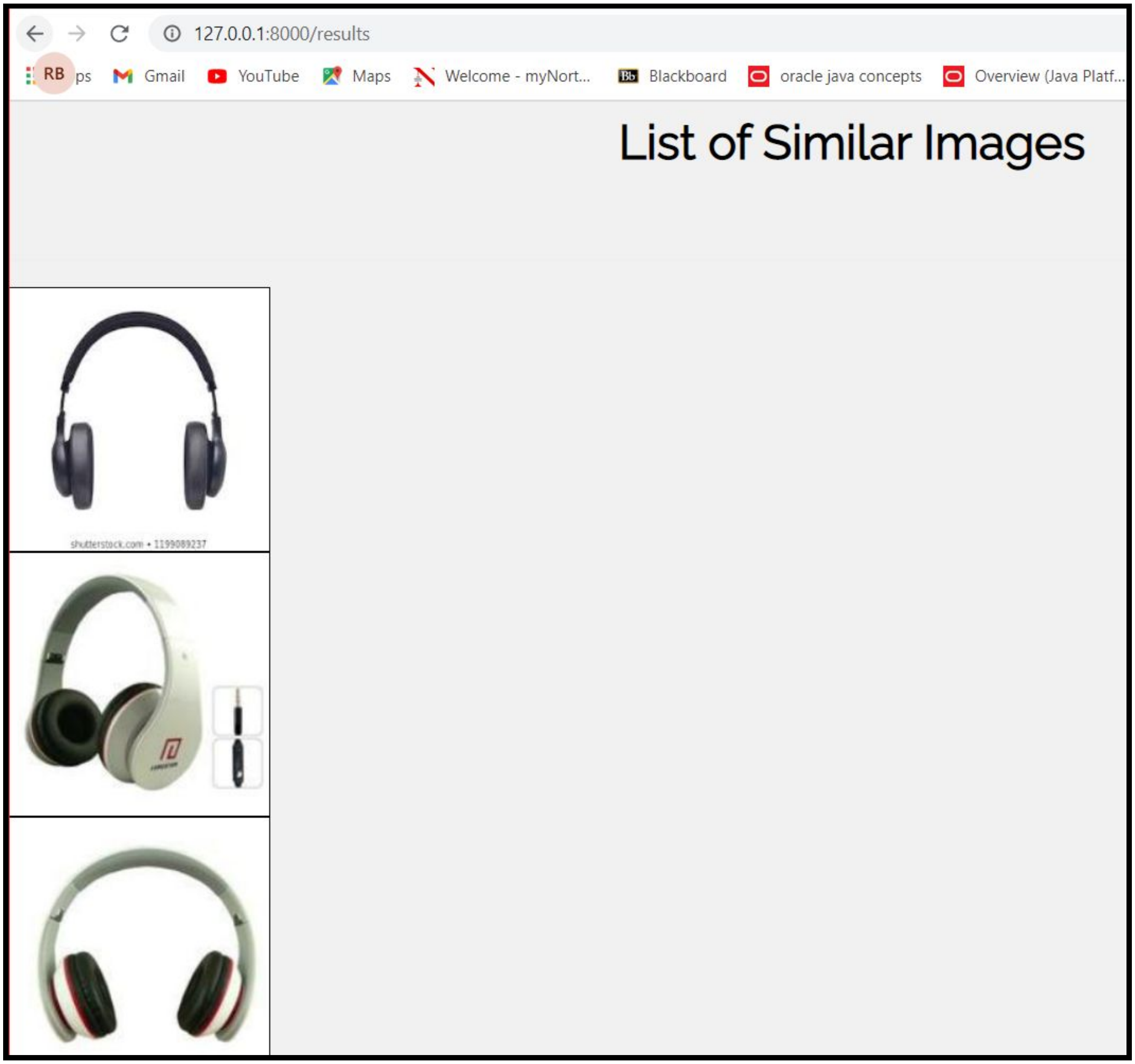
Flask:

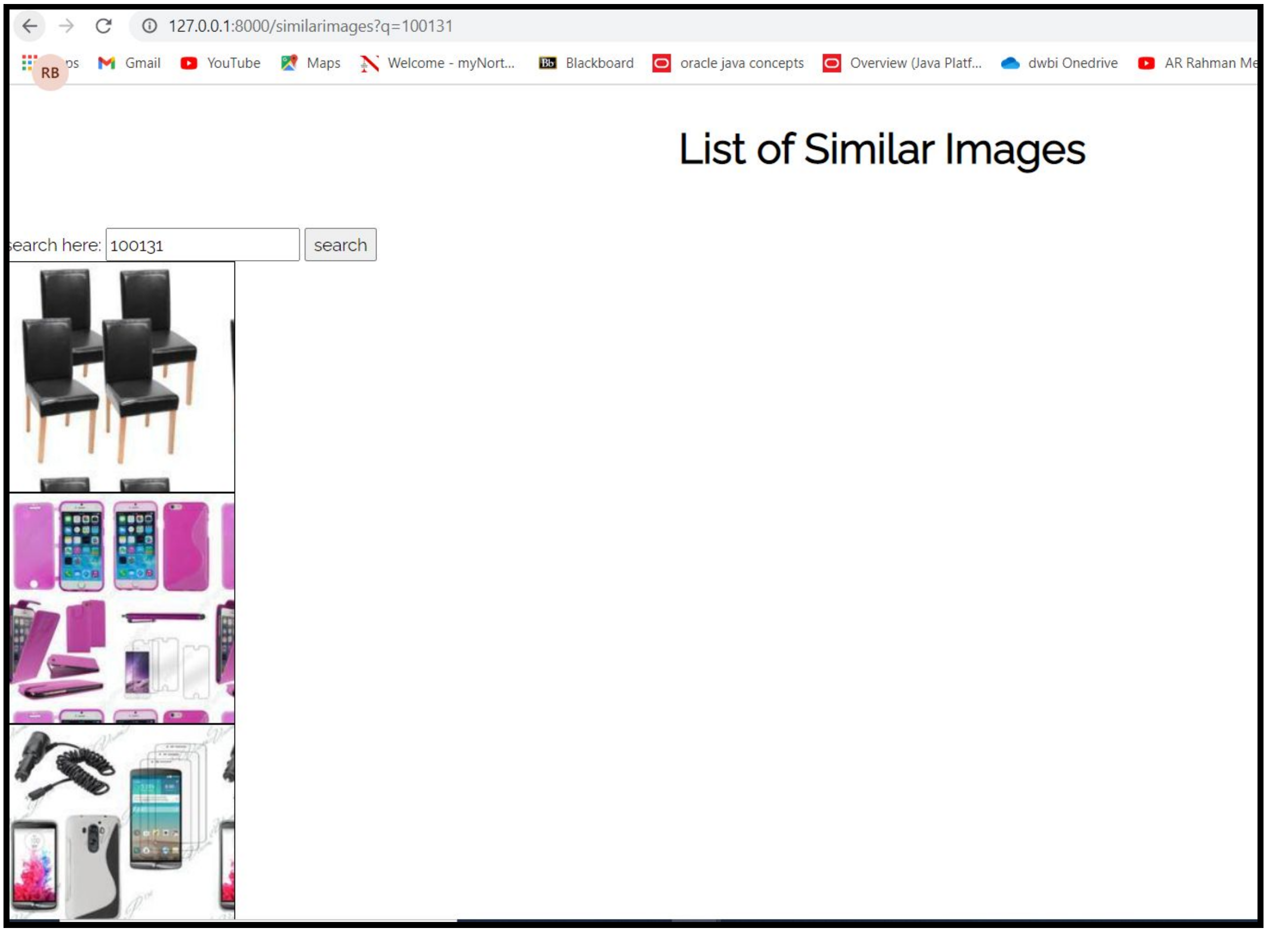
- Flask is an API of Python that allows us to build up web-applications.
- We generated a dashboard for implementing two methods
    a. When a image is uploaded, we retrieve 10 similar images
    b. When a product id is searched, we retrieve 10 similar images

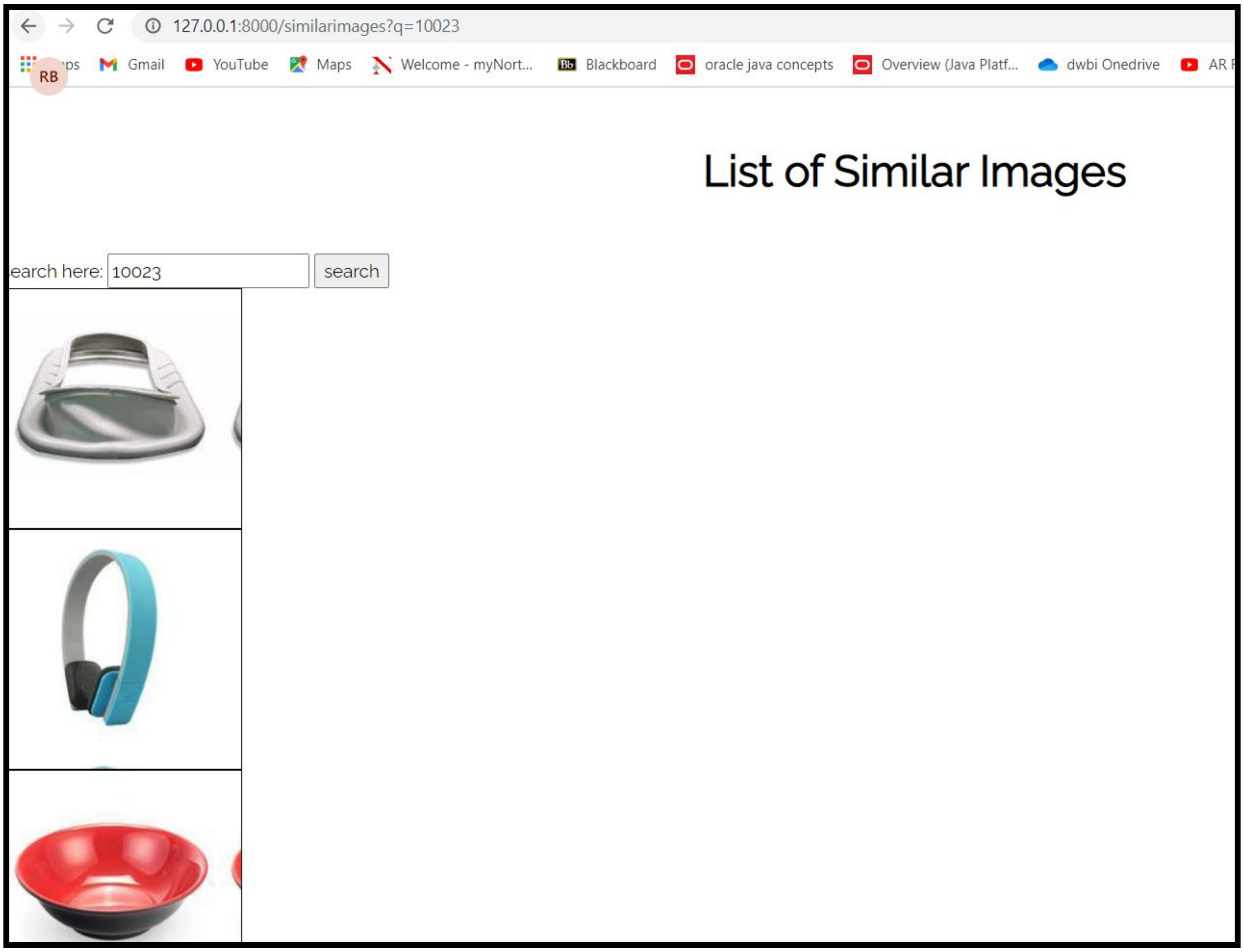

127.0.0.1:8000
Apps
Gmail
YouTube
Maps
Welcome - myNort...
Blackboard
oracle java concepts
Overview (Java Platf...
dwbi Onedrive
AR Rahman Medley...
Student Hub
Image Search
RB
Drop files here or click to upload.



127.0.0.1:8000/results
Apps
Gmail
YouTube
Maps
Welcome - myNort...
Blackboard
oracle java concepts
Overview (Java Platf...
List of Similar Images


127.0.0.1:8000/results
RB
ps
Gmail
YouTube
Maps
Welcome - myNort...
Blackboard
oracle java concepts
Overview (Java Platf...
List of Similar Images
shutterstock.com • 1199089237


127.0.0.1:8000/similarimages?q=100131
Gmail
YouTube
Maps
Welcome - myNort...
Blackboard
oracle java concepts
Overview (Java Platf...
dwbi Onedrive
List of Similar Images
earch here: 100131
search


127.0.0.1:8000/similarimages?q=10023
Gmail
YouTube
Maps
Welcome - myNort...
Blackboard
oracle java concepts
Overview (Java Platf...
dwbi Onedrive
RB
List of Similar Images
earch here:
10023
search

Streamlit:

Streamlit is an open source app framework specifically designed for ML engineers working with Python. It allows you to create a stunning looking application with only a few lines of code.

A few of the advantages of using Streamlit tools like Dash and Flask:

- It embraces Python scripting No HTML knowledge is needed!
- Less code is needed to create a beautiful application
- No callbacks are needed since widgets are treated as variables
- Data caching simplifies and speeds up computation pipelines.

```python
import streamlit as st
import os
import pandas as pd
import numpy as np

#add_selectbox=st.sidebar.radio("Select the type of Search Method ",("FAISS"))

#os.chdir('C:/Users/rishv/OneDrive/Northeastern/SEM3/Algorithmic Digital Marke

#if add_selectbox=='FAISS':
st.title("FACEBOOK ADD SIMILARITY SEARCH")
st.write("-----------------------------------------------------")
def get_data():
    return pd.read_csv('faiss.csv')

n=1
df=get_data()
images=df['0'].unique()
st.subheader("Select an image from drop down menu :")
pic=st.selectbox('Choices:',images)
st.write("**You selected**")
st.image(pic,width=None)

#Displaying output
z=st.slider('How many images do you want to see?',1,10,5)
st.write("----------------------------------------------------------")
st.subheader("Output:")
st.write('**Images similar to the image selected by you: **')
for index,row in df.iterrows():
    if row['0']==pic:
        while n<z+1:
            st.image(row[n],use_column_width=None,caption=row[n])
            n+=1
```

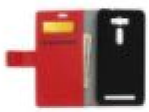
FACEBOOK ADD SIMILARITY SEARCH

Select an image from drop down menu :

Choices:

1486_3.jpg

You selected



How many images do you want to see?

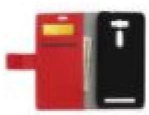
5

1

10

Output:

Images similar to the image selected by you:



1486_2.jpg

875_1.jpg

217_1.jpg

Heroku:

Heroku is a cloud platform as a service (PaaS) supporting several programming languages. One of the first cloud platforms, Heroku has been in development since June 2007, when it supported only the Ruby programming language, but now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go.For this reason, Heroku is said to be a polyglot platform as it has features for a developer to build, run and scale applications in a similar manner across most languages. Heroku was acquired by Salesforce.com in 2010