CS - 305
Assignment-2

Submitted by: AYUSH VERMA
Entry no.: 2019csb1147
Department: CSE
Submitted to: Dr *Balwinder Sodhi*

I have used fast API and postgresSQL

Design of classes/files:

- sqlConnector.py
  This class is basically used to connect to the Postgres database and execute various select and insert functions on the SQL database.
  In my case, the table name was: *imgtable*

  def __init__(self): This constructor specifies various parameters required for proper connection to the Postgres database. In my case, the password was set as `"Ayush@37"`.

  def insertIntoTable(self, fileName, encoding): This function is used to insert an entry into database.

  def selectAll(self): This function is used to select all rows from the table.

  def selectAgainstId(self, id): This function is used to fetch a row corresponding to id provided as a parameter.

  def update_meta_data(self, id, person_name, version, location, date): This function is used to update the metadata field in the table corresponding to the id provided as a parameter.

- faceRecognition.py
  This class is basically used for any functions related to the “face recognition” library.

  def give_encoding(self, img_file): This function returns encoding as a NumPy array for an image file provided as a parameter.

def img_distance(self, source, target): This function returns a list of distances between source and target image encodings by calculating the euclidean distance between them. lower the distance, more closer is the face match

- main.py
  This is the main fast API server program file.
  It contains all the functions corresponding to the post methods as specified by the template given by Sir.
  One extra route is provided *@app.post("/update_meta_data/"),* this route is used for updating the metadata of the image in the table corresponding to supplied id.

- test_main.py
  This file is basically used for unit testing the code.
  assertion is done on response.status_code and response.json()

  def test_1(): This test is used for testing the search_face route of the API.

  def test_2(): This test is used for testing the add_face route of the API.

  def test_3(): This test is used for testing the add_faces_in_bulk route of the API.

  def test_4(): This test is used for testing the get_face_info route of the API.

  def test_5(): This test is used for testing the update_meta_data route of the API.

Solid Principles used

- Single Responsibility Principle: Each class that I have implemented is responsible for handling its own functionality. For example: sqlConnector.py is responsible for connecting to the postgres and running queries of postgres in python, and the faceRecognition.py is responsible for functions related to the face_recognition library. Similarly, main.py and test_main.py handling their respective functionalities.
- Dependency Inversion Principle: Low level class (sqlConnector) closely related to hardware and High level class has been made.

Dependencies to be installed for running the code correctly

- face-recognition library
- Pip install coverage
- FastApi dependencies
- Postgres dependencies
- Numpy array

If you are running on Windows, make sure that you are on the python virtual environment, else face recognition library won't run.

Schema of the table

```
postgres=# \d imgtable
                                  Table "public.imgtable"
   Column   |          Type          | Collation | Nullable |              Default
------------+------------------------+-----------+----------+-----------------------------------
 id         | integer                |           | not null | nextval('imgtable_id_seq'::regclass)
 filename   | character varying(100) |           |          |
 personname | character varying(100) |           |          |
 version    | integer                |           |          |
 location   | character varying(100) |           |          |
 date       | character varying(100) |           |          |
 vector     | double precision[]     |           |          |
```

The schema of the table should be exactly same in your postgres, else error will be prompted.

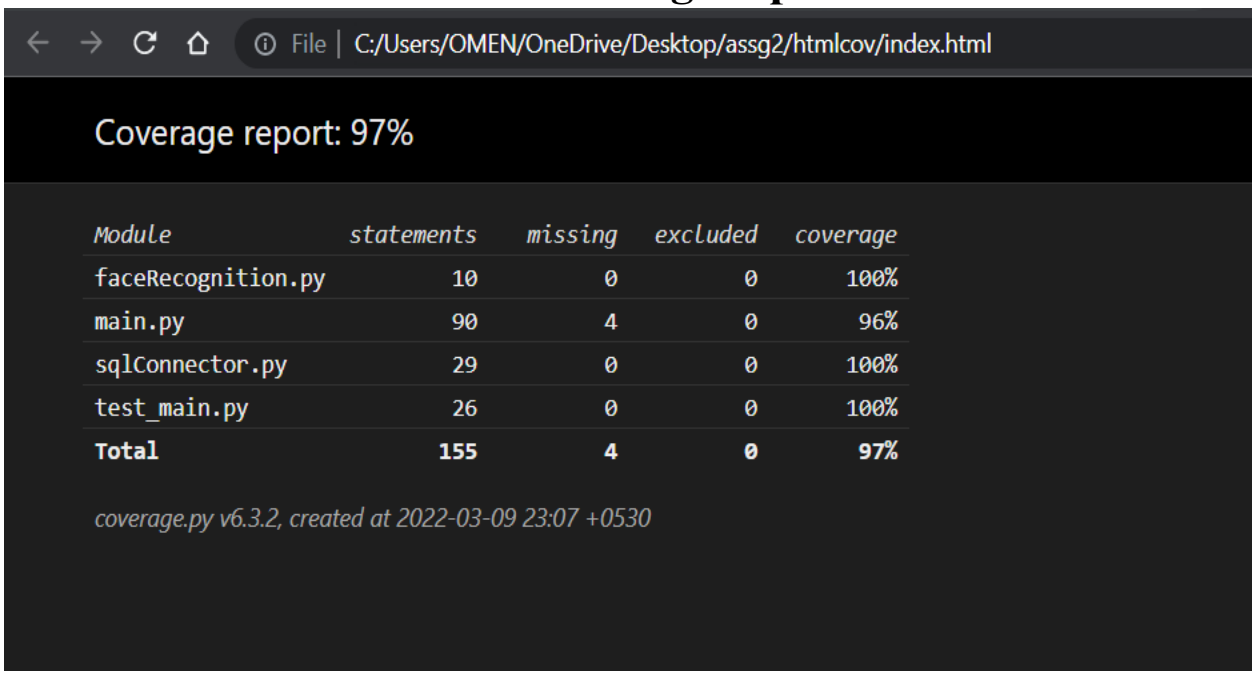
Steps for creating the schema as above:

```
create table imgTable (
  id serial,
  fileName VARCHAR(100),
personName VARCHAR(100),
version int,
location VARCHAR(100),
date VARCHAR(100),
  vector double precision []
 );
```

Unit Testing along with coverage report

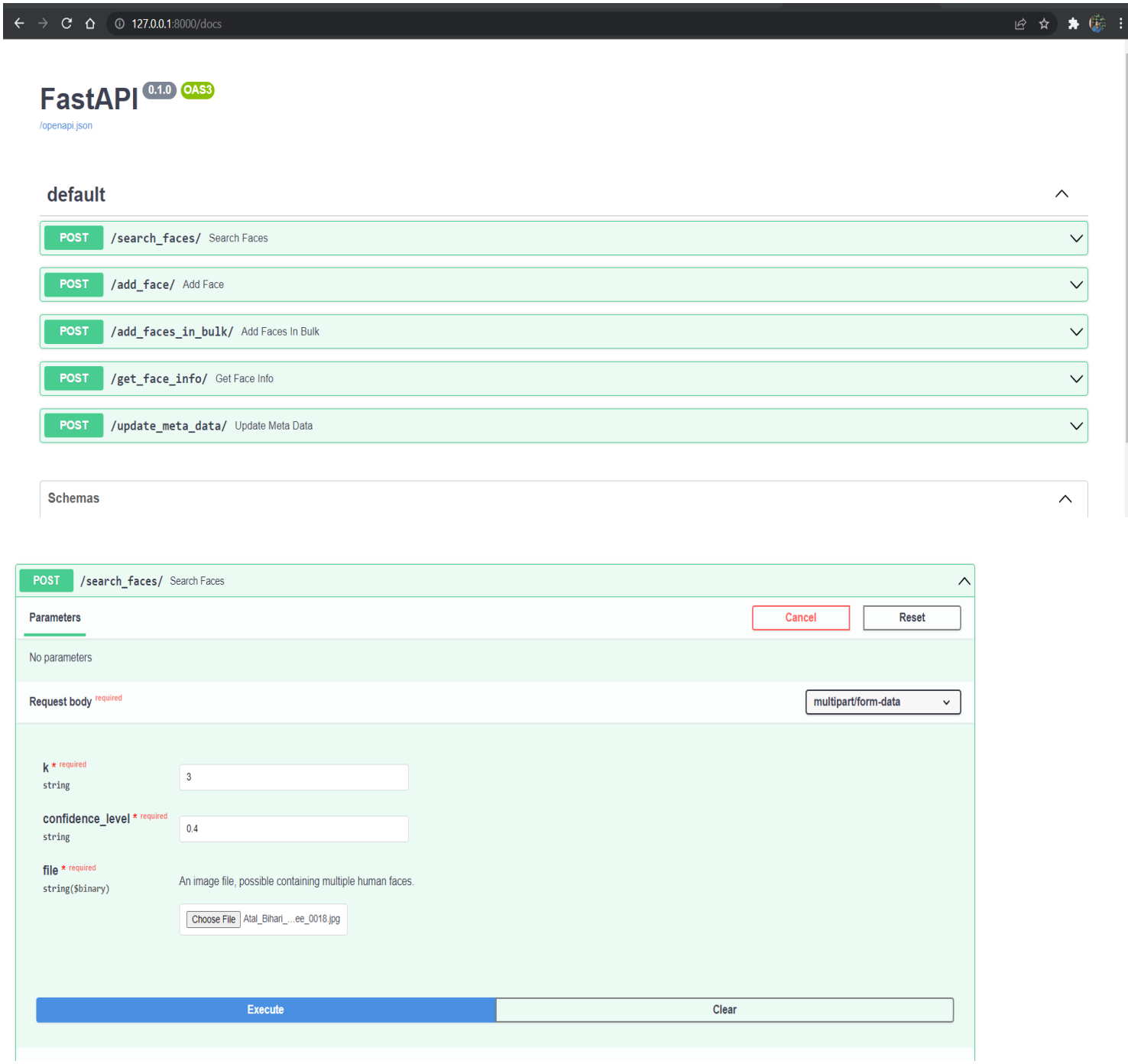
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                                  powershell

(myenvpy) PS C:\Users\OMEN\OneDrive\Desktop\assg2> coverage run -m pytest test_main.py
============================= test session starts =============================
platform win32 -- Python 3.9.3, pytest-7.0.1, pluggy-1.0.0
rootdir: C:\Users\OMEN\OneDrive\Desktop\assg2
plugins: anyio-3.5.0
collected 5 items

test_main.py .....                                                      [100%]

============================== 5 passed in 4.08s ==============================
(myenvpy) PS C:\Users\OMEN\OneDrive\Desktop\assg2> coverage report -m
Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
faceRecognition.py     10      0   100%
main.py                90      4    96%   19, 61-62, 116
sqlConnector.py        29      0   100%
test_main.py           26      0   100%
-------------------------------------------------
TOTAL                 155      4    97%
(myenvpy) PS C:\Users\OMEN\OneDrive\Desktop\assg2>

Ln 18, Col 21  Spaces: 3  UTF-8  CRLF  {} Python  Go Live  Prettier
```

Code-Coverage report



Running the server only

```
(myenvpy) PS C:\Users\OMEN\OneDrive\Desktop\assg2> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\OMEN\\OneDrive\\Desktop\\assg2']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [9944] using statreload
WARNING:  The --reload flag should not be used in production on Windows.
INFO:     Started server process [16660]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```


127.0.0.1:8000/docs
FastAPI
0.1.0
OAS3
/openapi.json
default
POST /search_faces/ Search Faces
POST /add_face/ Add Face
POST /add_faces_in_bulk/ Add Faces In Bulk
POST /get_face_info/ Get Face Info
POST /update_meta_data/ Update Meta Data
Schemas

POST /search_faces/ Search Faces
Parameters
Cancel
Reset
No parameters
Request body required
multipart/form-data
k * required
string
3
confidence_level * required
string
0.4
file * required
string($binary)
An image file, possible containing multiple human faces.
Choose File
Atal_Bihari_...ee_0018.jpg
Execute
Clear

Request URL

```
http://127.0.0.1:8000/search_faces/
```

Server response

Code	Details
200	Response body


```
{
  "status": "OK",
  "body": {
    "matches": [
      {
        "id": 1008,
        "file_name": "Atal_Bihari_Vajpayee_0018.jpg",
        "person_name": null,
        "version": null,
        "location": null,
        "date": null,
        "distance": 0
      },
      {
        "id": 1013,
        "file_name": "Atal_Bihari_Vajpayee_0023.jpg",
        "person_name": null,
        "version": null,
        "location": null,
        "date": null,
        "distance": 0.3212965238882343
      },
      {
        "id": 1012,
        "file_name": "Atal_Bihari_Vajpayee_0022.jpg",
        "person_name": null,
        "version": null,
        "location": null,
        "date": null,
```


Download

Response headers

```
content-length: 465
content-type: application/json
date: Wed,09 Mar 2022 17:20:37 GMT
server: uvicorn
```