Instructions

Make sure TERMINAL is opened inside assignment2 directory
Example : C:\Users\91892\Desktop\assignment2>

1. Setting up virtualenv (run the below command)

   ```
   cd env/scripts && activate && cd ../..
   ```

2. (faceRecoAPI) on the left ensures a success.

   ```
   (faceRecoAPI) C:\Users\91892\Desktop\assignment2>
   ```

3. Setting up Postgres connection

   ```
   // open terminal inside the assignment2 directory.
   // Create a database and import the dumped sql file
         (contains lfw dataset of ~13000 images)
   psql -U postgres
   CREATE DATABASE face_data;
   \c face_data ;
   \i ./store/store.sql ;
   select count(*) from faces;   [VERIFY : 12572]
   ```

4. Use the config.json file and edit it as per your local hostname and password of postgres.

   ```
   {
    "user": "postgres" ,
    "password": "YOUR_PASSWORD",
    "host": "localhost" ,
    "port": 5432,
    "database": "face_data"
   }
   ```

5. Open a new terminal , Run the FastApi Server

   ```
   //Make sure TERMINAL is connected to virtualenv (step 1 & 2)
   uvicorn main:app -reload
   ```

6. Open a new terminal, Run the Pytest with Coverage

   ```
   //Make sure TERMINAL is connected to virtualenv (step 1 & 2)
   coverage run -m pytest test_main.py
   coverage report -m
   ```

Index (Files strictly adhere to SOLID principles)

- Store/store.sql ⇒ insert commands to create the dumped lfw database.
- Database.py ⇒ Database class that connects to postgres .
    - Connects to Postgresql using SQLAlchemy.
    - Has all database related utility functions like
        - Creating tables and schemas
        - All Relevant queries like adding , updating and searching face datas from the database.
- face_Algorithms.py ⇒ Utility functions related to face_recognition.
- Face_cache.py ⇒ Cache that stores the database memory locally.
    - serves quick runtime for multiple clients.
    - Removes need to load the complete database for multiple search requests during k_BestMatch POST request.
- FileIO.py ⇒ Manages all the Input/Output streams.
    - including downloading zip/image files
    - extracting zip files
    - adding images to a Database from a certain directory
    - File renaming
    - Making directories
    - Deleting Files
    - Deleting File directories
- Main.py ⇒ Contains the root FastApi() server
- test_Main.py ⇒ Contains the unit tests for testing Main.py

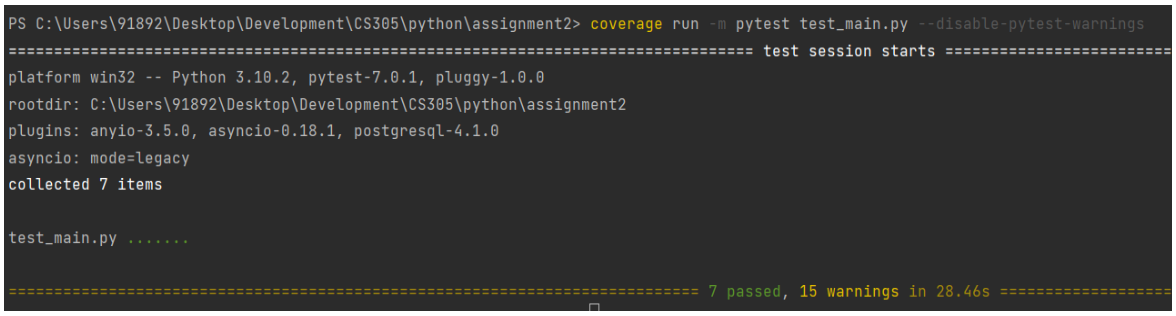
{ Notes }

- Zip File format after extracting should be like ⇒

```
📁Face1_folder
    Face1_1.jpg
    Face1_2.jpg
📁Face2_folder
    Face2_1.jpg
📁Face3_folder
    Face3_1.jpg
    Face3_2.jpg
```

- Images should be in png , jpg or jpeg format.
- Please mail/hangout me at 2019csb1121@iitrpr.ac.in in case there is any trouble during installation.

Pytest Result



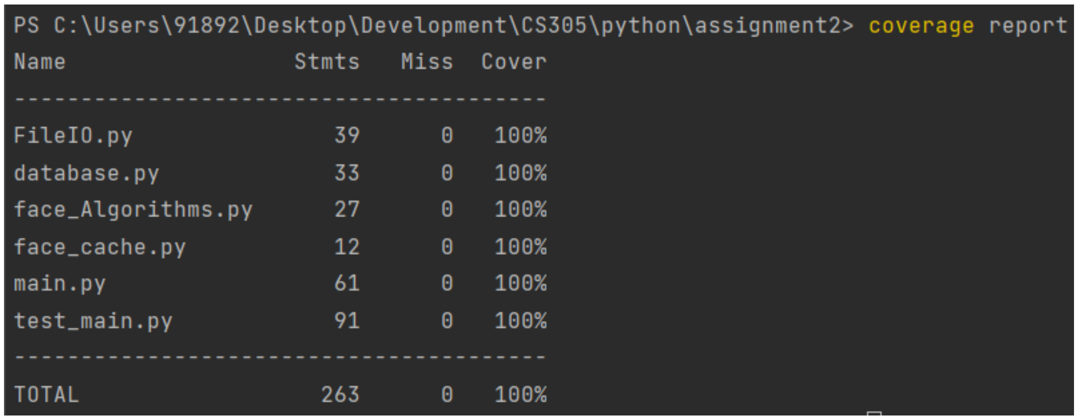
```
PS C:\Users\91892\Desktop\Development\CS305\python\assignment2> coverage run -m pytest test_main.py --disable-pytest-warnings
============================================================================== test session starts ========================
platform win32 -- Python 3.10.2, pytest-7.0.1, pluggy-1.0.0
rootdir: C:\Users\91892\Desktop\Development\CS305\python\assignment2
plugins: anyio-3.5.0, asyncio-0.18.1, postgresql-4.1.0
asyncio: mode=legacy
collected 7 items

test_main.py .......

========================================================================== 7 passed, 15 warnings in 28.46s ==================
```

- Warnings can be ignored , since they are mainly version depreciation issues.
- All 7 tests have passed successfully and the corner cases also have been handled in every possible way.

Coverage Report

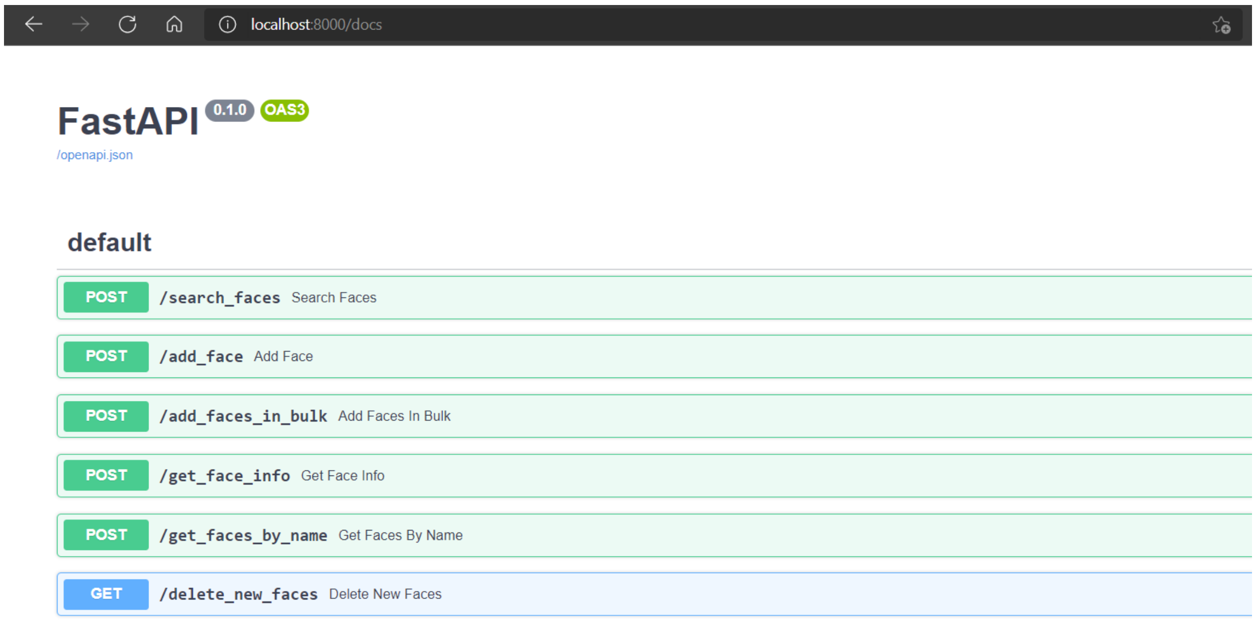


```
PS C:\Users\91892\Desktop\Development\CS305\python\assignment2> coverage report
Name                    Stmts   Miss  Cover
-------------------------------------------
FileIO.py                  39      0   100%
database.py                33      0   100%
face_Algorithms.py         27      0   100%
face_cache.py              12      0   100%
main.py                    61      0   100%
test_main.py               91      0   100%
-------------------------------------------
TOTAL                     263      0   100%
```

- All possible edge cases for each get and post request have been handled carefully and explained with comments in the test_main.py file.
- Each exception and edge case has been handed separately to achieve 100 percent coverage in all files.

Testing out the Application using your custom tests.

- Ensure uvicorn server is running on port 8000 [step 5 in Instructions]
- Goto: http://localhost:8000/docs to open swagger UI.
- Test out the GET and POST requests using the Swagger GUI.



localhost:8000/docs

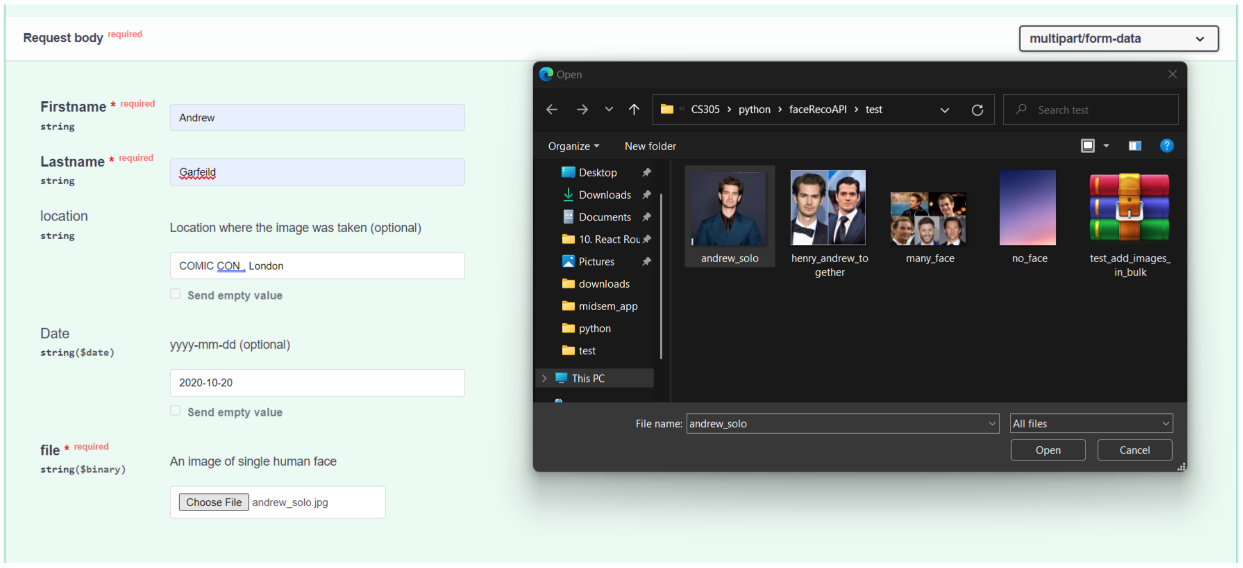
FastAPI 0.1.0 OAS3
/openapi.json

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 /get_faces_by_name Get Faces By Name
GET /delete_new_faces Delete New Faces

add_face POST Request

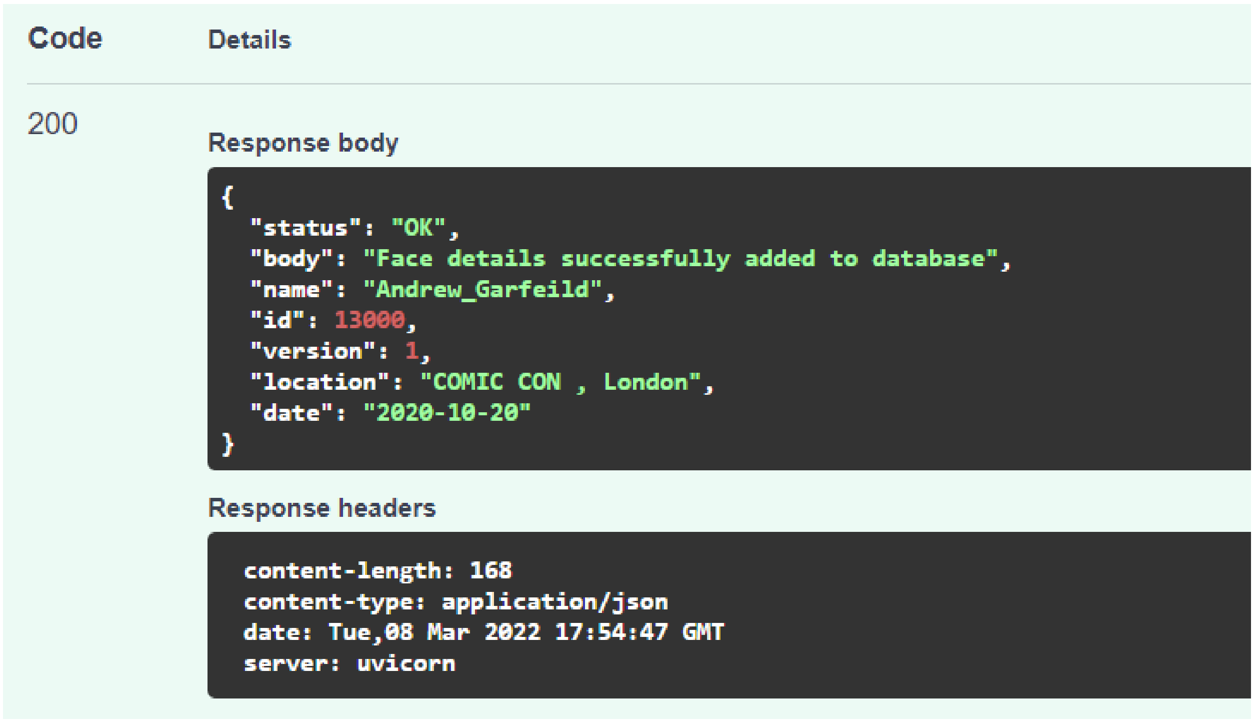
Fill in the details and choose the image file.



Request body required — multipart/form-data

Firstname * required (string)	Andrew
Lastname * required (string)	Garfeild
location (string)	Location where the image was taken (optional): COMIC CON , London; Send empty value
Date (string($date))	yyyy-mm-dd (optional): 2020-10-20; Send empty value
file * required (string($binary))	An image of single human face: Choose File andrew_solo.jpg

A successful response is obtained and image data is added to the postgres database.



Code Details

200

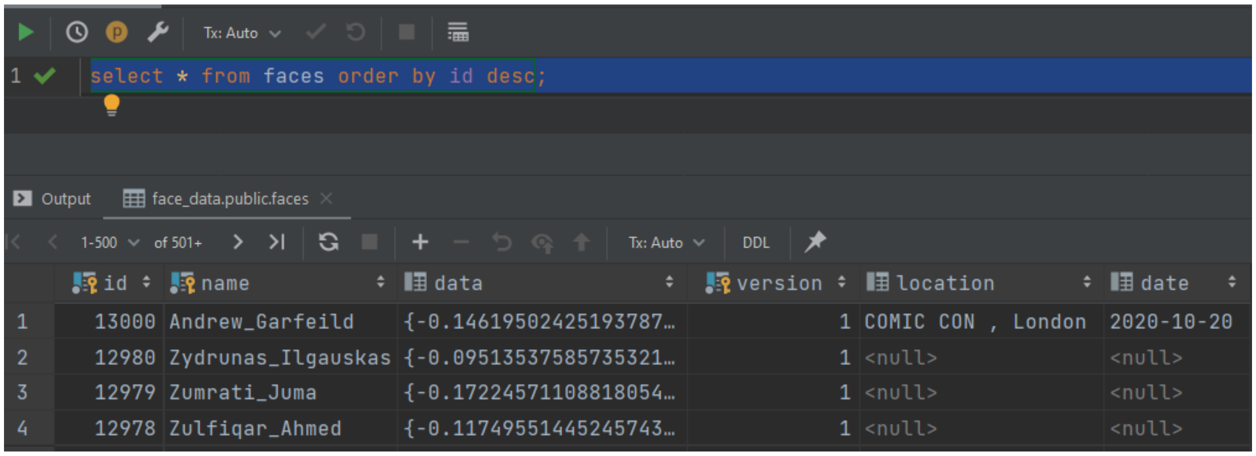
Response body

```
{
  "status": "OK",
  "body": "Face details successfully added to database",
  "name": "Andrew_Garfeild",
  "id": 13000,
  "version": 1,
  "location": "COMIC CON , London",
  "date": "2020-10-20"
}
```

Response headers

```
content-length: 168
content-type: application/json
date: Tue,08 Mar 2022 17:54:47 GMT
server: uvicorn
```

New row is successfully inserted into the database with its face_encodings.



```
select * from faces order by id desc;
```

	id	name	data	version	location	date
1	13000	Andrew_Garfeild	{-0.14619502425193787…	1	COMIC CON , London	2020-10-20
2	12980	Zydrunas_Ilgauskas	{-0.09513537585735321…	1	<null>	<null>
3	12979	Zumrati_Juma	{-0.17224571108818054…	1	<null>	<null>
4	12978	Zulfiqar_Ahmed	{-0.11749551445245743…	1	<null>	<null>

You can try out other custom tests using the SWAGGER UI onto other GET and POST requests similarly.