AI Image Extraction Microservices

Date	@December 17, 2021
GitHub @ME	https://github.com/Shreyanshpaliwal02/AI-Image-Extraction
Resources	https://github.com/codingforentrepreneurs/FastAPI-Microservice-for-Django
Source	https://www.youtube.com/watch?v=JxH7cdDCFwE&list=WL&index=1&t=10s

foundation: simple fast app with jinja template

reason: to offload any demand or load on our primary application , a single focused microservice working on only OCR for larger services.

MICROSERVICES: (software development life cycle) SDLC approach in which large applications are built as a collection of small functional modules.

Applications are modeled as collections of services, which are:

- Maintainable and testable
- Loosely coupled
- Independently deployable
- Designed or organized around business capabilities
- Managed by a small team

Working of Microservices:

- **Clients**: Different users send requests from various devices.
- **Identity Provider**: Validate a user's or client's identity and issue security tokens.
- **API Gateway:** Handles the requests from clients.
- **Static Content:** Contains all of the system's content.
- **Management:** Services are balanced on nodes and failures are identified.

- **Service Discovery:** A guide to discovering the routes of communication between microservices.
- **Content Delivery Network:** Includes distributed network of proxy servers and their data centers.
- **Remote Service:** Provides remote access to data or information that resides on networked computers and devices

running uvicorn app cmd:

`uvicorn app.main:app –reload` (first app is the directory name , 2nd would be the module name)

`running pre-commit:`

`pip install pre-commit`

`pre-commit –help`

`pre-commit install #(copy repos and ids)`

`pre-commit run –all-files`

I WAS GETTING A LOT OF ERRORS AS I DIDNT HAVE THE BEST GRASP ON PYTHON → BUT I LEARNED HOW TO DEBUG A LOT FROM THIS PROJECT

I ALSO LEARNED TO ASK FOR HELP

DOCUMENTATION:

Purpose: A way to extract text from images - tesseract OCR-(ML based)

: A good way to learn how to deploy machine learning algorithms into production

digitalOcean/ docker container

requirements: `pip install -r requirements.txt`

vscode;

fastapi

gunicorn

uvicorn

pytest → tests all our python code

requests

environment setup:

new folder- intialize a virtual env - ctrl + ~

`python 3.8 -m venv .`

`source bin/activate  //activating the v env` save workspace

git —version (MAKING SURE ALL UPDATES ARE PUSHED NORMALLY WERE A HASSLE, MULTIPLE TIMES I MADE A WRONG COMMIT IN THE MAIN BRANCH AND DIDNT KNOW HOW TO GO BACK, SO AS NAIVE AS IT SOUNDS, I STARTED FROM SCRATCH A BUNCH OF TIMES UNTIL I JUST GOT IT RIGHT)

VSCode has built in src control `- gitignore python` to skip things i dont wanna push into production

`git push origin main or master` → **ways to switch branch**

FAST API APPLICATION WITH JINJA TEMPLATE

FAST API: fast api is a modern fast web frame work for api dev with python 3.6+;

`pip install fastapi`

an **ASGI Server** such as Uvicorn is reqd for production

`pip install "uvicorn[standard]”`

`uvicorn main:app --reload` //run the live server

`from fastapi import FastAPI`

`app = FastAPI()`

```
@app.get("/")   // PYTHON  DECORATOR?
async def root():
return {"message": "Hello World"}
```

ASGI SERVER ❓: Asynchronous ***Server*** Gateway Interface is **a calling convention for web servers to forward requests to asynchronous-capable Python programming language frameworks, and applications**

the api :

→receives http reqsts in the paths / and /items/{item_id}

→both paths take GET operations

→item_id should be an int

REST API (REPRESENTATIONAL STATE TRANSFER) **//Roy Fielding**

app → app communication

- An API that conforms to the constraints of REST arch. style and allows for interaction with RESTful web services
- its not a protocol or a standard, but a set of architectural constraints

- When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint
- Representation format: JSON, HTML, Python PHP or plain text
- When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint

CRITERIA FOR RESTFULness

- client - server architecture made up of clients, servers, resources - managed through HTTP
- no client info is stored between get requests and **each request is separate and unconnected** (STATELESS CLIENT SERVER COMMUNICATION)
- //STATELESS VS STATEFUL TRANSACTION-read
- **cacheable data** for streamlined interactions
- uniform interface between components so that info is transferred in std form, **requirements:**
    1. resources requested are identifiable and separate from the representations sent to the client.
    2. resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.
    3. hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.
- A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.

JINJA, GUNICORN, UVICORN?

jinja : fast api always returns json by default, in order to change that to html we use jinja templates → reason? **THIS CODE IS SIMILAR TO PYTHON SYNTAX**

`pip install jinja2`

uvicorn: basically the server that runs our code one single project for OCR for 1000x of other projects

MACHINE LEARNING MODEL FOR OCR

Tesseract OCR → Documentation

- [ ] **REQUIREMENTS ⭐⭐⭐**

  - **FastAPI**: modern fast web framework for building APIs with python ≥3.6
  - **Tesseract OCR** - open src ocr to extract text from images.
  - **pytesseract** - OCR for python.
  - **pre-commit** - inspect the snapshot that's about to be committed, to see if you've forgotten something, to make sure tests run, or to examine whatever you need to inspect in the code.
  - **pytest** - allows creating marks or custom labels for any test.
  - **Gunicorn** - Python WSGI (Web server gateway interface) HTTP Server(pure python http server).
  - **Uvicorn** - an ASGI web server implementation for Python.
  - **Requests** - Requests allows you to send HTTP/1.1 requests extremely easily.
  - **Docker** - software platform that simplifies the process of building, running, managing and distributing applications
  - **aiofiles** - files for handling local disc files in asyncio applications
  - **pillow** - python image library

- [ ] **FASTAPI APP**
- [ ] **USING JINJA TEMPLATE**
- [ ] **FASTAPI GIT AND PRE-COMMIT**
- [ ] **DEPLOYING TO CLOUD**
- [ ] **TESTING THE APP USING FASTAPI & PYTEST**

requirements: pytest, requests

pytest.ini→ for no recursive directories

test_endpoints.py → for testing the API endpoints

`from app.main import app`

`from fastapi.testclient import TestClient`

`client = TestClient(app`

`def test_get_home()`: #testing the get response from home(or any other sub division)

`response = client.get("/")` #equivalent to requests.get("") #from the python requests package

`assert response.status_code == 200`

`assert "text/html" in response.headers['content-type']`

`def test_post_home()`: #testing the post response from home(or any other sub division)

`response = client.get("/")` #equivalent to requests.post("") #from the python requests package

`assert response.status_code == 200`

`assert "application/json" in response.headers['content-type']`

`assert response.json == {"hello" : "world"}` #testing

☐ HANDLING FILE UPLOAD

similar to post method except it takes an argument for the while which we're going to upload

```
from fastapi import{
    FastAPI,
    File,
    Upload File
}
```

THIS IS NOT AN HTTP RESPONSE LIKE OTHERS ITS A FILE RESPONSE

@`app.post("/img-echo/", response_class = FileResponse)` #http post

#this function runs asynchronously? → **type of programming in which we can execute more than one task without blocking the Main task (function)→ using asyncio**

`import io`

`async def img_echo_view(file:Upload = File(…)):` #ellipses are used for giving arbitrary length

`bytes_string = io.BytesIO(await file.read())`

fname = path of file

fext = fname.suffix # to check whether the format is .jpg, .txt etc

dest = UPLOAD_DIR/f"{uuid.uuid1()}{fext}" dest = UPL

`return file`

→ now we need to save file by setting up a base directory

{go to main function →

```
    BASE_DIR  = pathlib.Path( file ).parent
    UPLOAD_DIR = BASE_DIR/”upload”
```

}

I also created an echo that test function will not run without a boolean flag → the test will only run for environment functions

FILE UPLOADS ARE ALWAYS ASNYCHRONOUS IN NATURE - ANOTHER BENEFIT OF USING UVICORN IS UVICORN IS SET-UP AND READY FOR ASYNCHRONOUS VIEWS

uuid❓ - universally unique identifier (128 But)

rb❓- raw binary

asynchronous programming❓

HTTPexception ❓

☐ **IMAGE UPLOAD VALIDATION**

match the extension of image with valid extensions

from PIL import image

turn echo byte string into image

```
valid_image_extensions = [’png’, ‘jpeg’, ‘jpg’]
def test_echo_epload():
    img_saved_path = BASE_DIR/”images”
    for path in img_saved_path.glob(”*”):
        response = client.post(”/img-echo/”, files = ….)
        fext = str(path.suffix).replace(’.’, ‘’)
        if fext in valid_image_extensions:
            assert fext in response.headers[’content-type’]
            assert response.status_code == 200
```

python PILLOW library❓

-basic image processing functionality (resizing,   rotation, transformation)

-pull statistical data from the images using histogram method, for automatic constrast enhancement

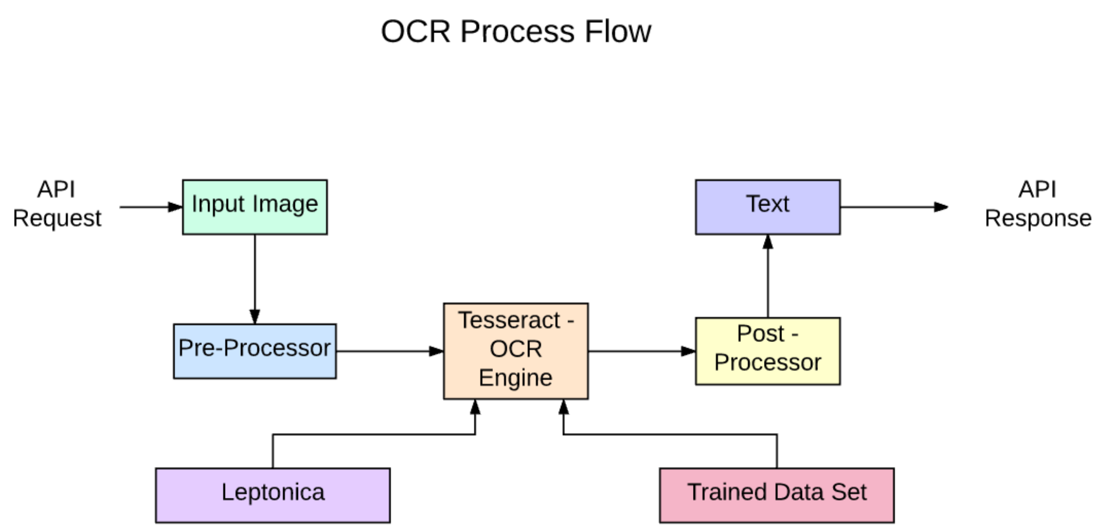
☐ IMPLEMENTING TESSERECT AND PYTESSERECT

implemented independently from fastAPI

OCR Subprocesses

- Preprocessing of the Image
- Text Localization
- Character Segmentation
- Character Recognition
- Post Processing

OCR Process Flow



Tesseract uses Convolutional Neural Network

Languages: tesseract --list-langs

Note - Only languages that have a `.traineddata` file format are supported by tesseract.

Pre- Processing for Tesseract

grayscale→cv2

noise removal→medianBlur

thresholding→threshold

dilate→

erosion→

canny edge detection→

skew correction→

template matching →

CODE:

```
import pathlib
import pytesseract
from PIL import Image


BASE_DIR = pathlib.Path(__file__).parent
IMG_DIR = BASE_DIR / “images”
img_path = IMG_DIR/ “ingredients-1.png”
img = Image.open(img_path)
preds = pytesseract.image_to_string(img)
print(preds)
```

RETURN FORMAT WILL BE: PREDICTED TEXT + ORIGINAL TEXT AS I DONT WANNA DECIDE WHICH DATA WILL BE MOST USEFUL FOR A GIVEN MICROSERVICE

Tesseract limitations summed in the list.

- The OCR is not as accurate as some commercial solutions available to us.
- Doesn't do well with partial occlusion, distorted perspective, and complex background.
- not capable of recognizing handwriting.
- may find gibberish and report this as OCR output.
- If a document contains languages outside of those given in the -l LANG arguments, results may be poor.

- It is not always good at analyzing the natural reading order of documents. For example, it may fail to recognize that a document contains two columns, and may try to join text across columns.
- It does not expose information about what font family text belongs to.

☐ AUTHORIZATION HEADERS

used to protect the prediction view → not same as authentication (does not include pwd, users etc)

ONE APP AUTH VARIABLE THAT'LL BE SHARED IN ENVIRONMENT VARIABLE

☐ AUTHORIZATION TESTS AND PRODUCTION ENDPOINT

```
git add —all
```

```
git commit -m "Production Ready"
```

select server RAM and CPU Cores on Docker

☐ DEPLOYMENT

PYTHON REQUESTS

UPDATE THE DIGITAL OCEAN DEPLOY TEMPLATE EACH TIME I NEED TO DEPLOY IT TO A NEW PROJECT

.do (dockerfile already created in VSCODE)

the-end***********************************