Introduction

This documentation provides a comprehensive guide to setting up, running, testing, and deploying the OCR API. By following the instructions, you should be able to get the API up and running efficiently.

Overview

The OCR (Optical Character Recognition) API allows you to extract text and bounding boxes from images. It provides two main endpoints: /get-text to extract text and /get-bboxes to extract bounding boxes around recognized text.

File Contents

requirements.txt: Lists the dependencies required for the project.

server.py: Contains the API endpoints and their implementations.

htmlcov: A folder which contains the html format coverage of server.

run_server: A script or command used to start the OCR API server.

test_server.py: This file contains a suite of tests for the HTMLConv Test API using pytest, ensuring the application's endpoints function correctly and handle various edge cases

test_api.py: A test script using pytest to validate the functionality of the OCR API endpoints

img: Directory/folder containing sample images for testing.

outputs: A folder which contains some of the code results which has been executed by me.

OCR API Documentation: Contains the Project Documentation.

Endpoints

1. Get Text from Image

URL: /get-text

Method: POST

Description: Extracts text from the provided image.

Request:

Headers: Content-Type: multipart/form-data

Body:

image (file): The image file from which to extract text.

Response:

Success (200):

Example using Postman

Select POST method.

Enter URL: http://localhost:5000/get-text.

Go to the Body tab.

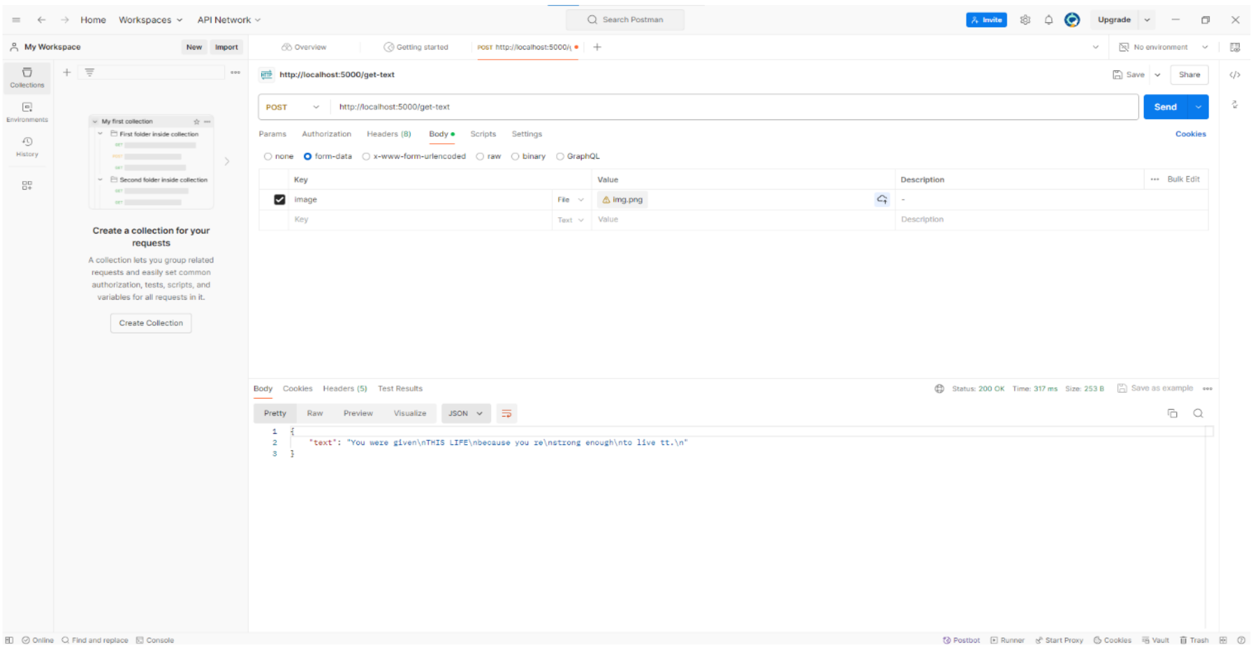
Select form-data.

Add a key with the name image and set the type to File.

Choose a file by clicking the Choose Files button and select your image.

Send the request.

Output:



POST http://localhost:5000/get-text

Body: form-data

Key		Value	Description
image	File	img.png	-

Status: 200 OK Time: 317 ms Size: 253 B

```
{
    "text": "You were given\nTHIS LIFE\nbecause you re\nstrong enough\nto live tt.\n"
}
```

2. Get Bounding Boxes from Image

URL: /get-bboxes

Method: POST

Description: Extracts bounding boxes around recognized text from the provided image.

Request:

Headers: Content-Type: multipart/form-data

Body:

image (file): The image file from which to extract bounding boxes.

Response:

Success (200):

Example using Postman

Select POST method.

Enter URL: http://localhost:5000/get-bboxes.

Go to the Body tab.

Select form-data.

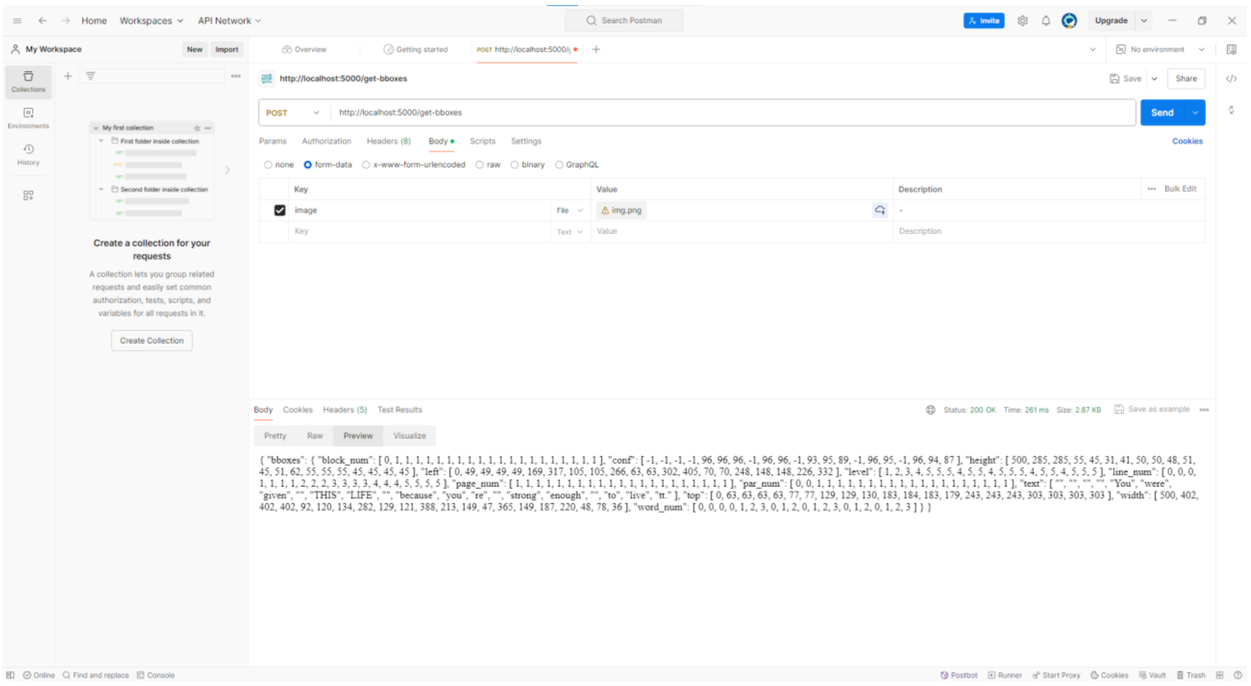
Add a key with the name image and set the type to File.

Choose a file by clicking the Choose Files button and select your image.

Send the request.

Running the Server

Output:



POST http://localhost:5000/get-bboxes

Body: form-data

Key		Value	Description
image	File	img.png	

Status: 200 OK Time: 261 ms Size: 2.87 KB

```
{ "bboxes": { "block_num": [ 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ], "conf": [ -1, -1, -1, -1, 96, 96, 96, -1, 96, 96, -1, 93, 95, 89, -1, 96, 95, -1, 96, 94, 87 ], "height": [ 500, 285, 285, 55, 45, 31, 41, 50, 50, 48, 51,
45, 51, 62, 55, 55, 55, 45, 45, 45, 45 ], "left": [ 0, 49, 49, 49, 49, 169, 317, 105, 105, 266, 63, 63, 302, 405, 70, 70, 248, 148, 148, 226, 332 ], "level": [ 1, 2, 3, 4, 5, 5, 5, 4, 5, 5, 4, 5, 5, 5, 4, 5, 5, 4, 5, 5, 5 ], "line_num": [ 0, 0, 0,
1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5 ], "page_num": [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ], "par_num": [ 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ], "text": [ "", "", "", "", "You", "were",
"given", "", "THIS", "LIFE", "", "because", "you", "re", "", "strong", "enough", "", "to", "live", "tt." ], "top": [ 0, 63, 63, 63, 63, 77, 77, 129, 129, 130, 183, 184, 183, 179, 243, 243, 243, 303, 303, 303, 303 ], "width": [ 500, 402,
402, 402, 92, 120, 134, 282, 129, 121, 388, 213, 149, 47, 365, 149, 187, 220, 48, 78, 36 ], "word_num": [ 0, 0, 0, 0, 1, 2, 3, 0, 1, 2, 0, 1, 2, 3, 0, 1, 2, 0, 1, 2, 3 ] } }
```

Ensure you have the necessary dependencies installed:

1. I have created the requirements.txt you can download it from my github repo else you can create a text file named requirements and add the following content:

```
Flask==2.3.2
pytesseract==0.3.10
Pillow==8.4.0
pytest==6.2.5
pytest-cov==3.0.0
requests==2.28.2
```

2. Next Step is to create a file named setup.py and add the above script content.

3. Run the setup script by opening a terminal or command prompt and navigate to the directory where setup.py and requirements.txt are located. Run the following command:

   python setup.py

This script will:

Install all the required packages listed in requirements.txt

OR

Open terminal directly after the first step and use the below command to install all the dependencies

```
pip install -r requirements.txt
```

Note: I am using tesseract-ocr 5.3.4 version. you can choose any one as per your requirements.
Download Link: https://github.com/tesseract-ocr/tesseract/releases

Flask server code should be as follows:

```
from flask import Flask, request, jsonify
import pytesseract
from PIL import Image
import logging
import os
import io

app = Flask(__name__)

# Set the Tesseract executable path if not in PATH
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'

# Set the TESSDATA_PREFIX environment variable
os.environ['TESSDATA_PREFIX'] = r'C:\Program Files\Tesseract-OCR\tessdata'

@app.route('/')
def index():
    return "Welcome to the OCR API. Use /get-text and /get-bboxes endpoints to
process images.", 200

@app.route('/get-text', methods=['POST'])
def get_text():
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'No image file in request'}), 400
        image_file = request.files['image']
        image = Image.open(io.BytesIO(image_file.read()))
        text = pytesseract.image_to_string(image)
        return jsonify({'text': text})
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/get-bboxes', methods=['POST'])
def get_bboxes():
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'No image file in request'}), 400
        image_file = request.files['image']
        image = Image.open(io.BytesIO(image_file.read()))
        data = pytesseract.image_to_data(image,
output_type=pytesseract.Output.DICT)
        return jsonify({'bboxes': data})
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

Running the server:

```
from server import app

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

Running the test_api file to test the Endpoints:

```
import requests
import logging
import os

# Set up the environment for Tesseract-OCR
os.environ['TESSDATA_PREFIX'] = r'C:\Program Files\Tesseract-OCR\tessdata'

# Set up logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger()

def get_text_from_api(image_path):
    url = 'http://localhost:5000/get-text'
    try:
        with open(image_path, 'rb') as img:
            files = {'image': img}
            response = requests.post(url, files=files)
            response.raise_for_status()
            data = response.json()
            logger.debug(f"Response from /get-text: {data}")
            return data
    except requests.exceptions.RequestException as e:
        logger.error(f"An error occurred in get_text_from_api: {e}")
def get_bboxes_from_api(image_path):
    url = 'http://localhost:5000/get-bboxes'
    try:
        with open(image_path, 'rb') as img:
            files = {'image': img}
            response = requests.post(url, files=files)
            response.raise_for_status()
            data = response.json()
            logger.debug(f"Response from /get-bboxes: {data}")
            return data
    except requests.exceptions.RequestException as e:
        logger.error(f"An error occurred in get_bboxes_from_api: {e}")

if __name__ == "__main__":
    image_path = 'C:/Users/Admin/Desktop/OCR API/img.png'
    text_data = get_text_from_api(image_path)
    bboxes_data = get_bboxes_from_api(image_path)

    if text_data:
        print(f"OCR Text: {text_data.get('text')}")

    if bboxes_data:
        print(f"Bounding Boxes: {bboxes_data.get('bboxes')}")
```

Test for the APIs using a testing framework:

```
import pytest
from server import app  # Import the Flask app from server.py
from io import BytesIO
from PIL import Image
import os

@pytest.fixture
def client():
    with app.test_client() as client:
        yield client

@pytest.fixture
def create_test_image():
    image_path = "C:/Users/Admin/Desktop/OCR API/img/img.png"
    os.makedirs(os.path.dirname(image_path), exist_ok=True)
    # Create a simple image if it doesn't exist
    if not os.path.exists(image_path):
        img = Image.new('RGB', (100, 100), color='red')
        img.save(image_path)
    return image_path
```

```python
def test_index(client):
    response = client.get('/')
    assert response.status_code == 200
    assert b"Welcome to the OCR API" in response.data

def test_get_text_endpoint(client, create_test_image):
    image_path = create_test_image
    with open(image_path, "rb") as img:
        data = {'image': (BytesIO(img.read()), 'img.png')}
        response = client.post('/get-text', content_type='multipart/form-
data', data=data)
        assert response.status_code == 200
        assert 'text' in response.get_json()

def test_get_bboxes_endpoint(client, create_test_image):
    image_path = create_test_image
    with open(image_path, "rb") as img:
        data = {'image': (BytesIO(img.read()), 'img.png')}
        response = client.post('/get-bboxes', content_type='multipart/form-
data', data=data)
        assert response.status_code == 200
        assert 'bboxes' in response.get_json()

def test_get_text_no_image(client):
    response = client.post('/get-text', content_type='multipart/form-data')
    assert response.status_code == 400
    assert 'error' in response.get_json()
    assert response.get_json()['error'] == 'No image file in request'

def test_get_bboxes_no_image(client):
    response = client.post('/get-bboxes', content_type='multipart/form-data')
    assert response.status_code == 400
    assert 'error' in response.get_json()
    assert response.get_json()['error'] == 'No image file in request'

def test_get_text_invalid_image(client):
    data = {'image': (BytesIO(b'not an image'), 'test.txt')}
    response = client.post('/get-text', content_type='multipart/form-data',
data=data)
    assert response.status_code == 500
    assert 'error' in response.get_json()

def test_get_bboxes_invalid_image(client):
    data = {'image': (BytesIO(b'not an image'), 'test.txt')}
    response = client.post('/get-bboxes', content_type='multipart/form-data',
data=data)
    assert response.status_code == 500
    assert 'error' in response.get_json()
def test_get_text_exception(client, monkeypatch):
    def mock_image_to_string(image):
        raise Exception("Mocked exception")
    monkeypatch.setattr("pytesseract.image_to_string", mock_image_to_string)

    image_path = "C:/Users/Admin/Desktop/OCR API/img/img.png"
    with open(image_path, "rb") as img:
        data = {'image': (BytesIO(img.read()), 'img.png')}
        response = client.post('/get-text', content_type='multipart/form-
data', data=data)
        assert response.status_code == 500
        assert 'error' in response.get_json()

def test_get_bboxes_exception(client, monkeypatch):
    def mock_image_to_data(image, output_type):
        raise Exception("Mocked exception")
    monkeypatch.setattr("pytesseract.image_to_data", mock_image_to_data)

    image_path = "C:/Users/Admin/Desktop/OCR API/img/img.png"
    with open(image_path, "rb") as img:
        data = {'image': (BytesIO(img.read()), 'img.png')}
        response = client.post('/get-bboxes', content_type='multipart/form-
data', data=data)
        assert response.status_code == 500
        assert 'error' in response.get_json()
```

Important Commands to run the test for coverage

To run the tests, ensure that you have the **pytest** package installed, then run

1) pytest --cov=server (The --cov=server part specifies that the coverage measurement should be focused on the server module. This means it will collect data on which parts of the server module are executed during the tests.)
2) pytest --cov=server --cov-report=html (This option specifies the coverage report to be in HTML Format and provide a web-based server file in the htmlcov folder.

Error Handling

Ensure the Tesseract executable path and TESSDATA_PREFIX environment variable are set correctly.

Handle cases where the image file is not found in the request with appropriate error messages.

Return detailed error messages in case of any exceptions during processing.