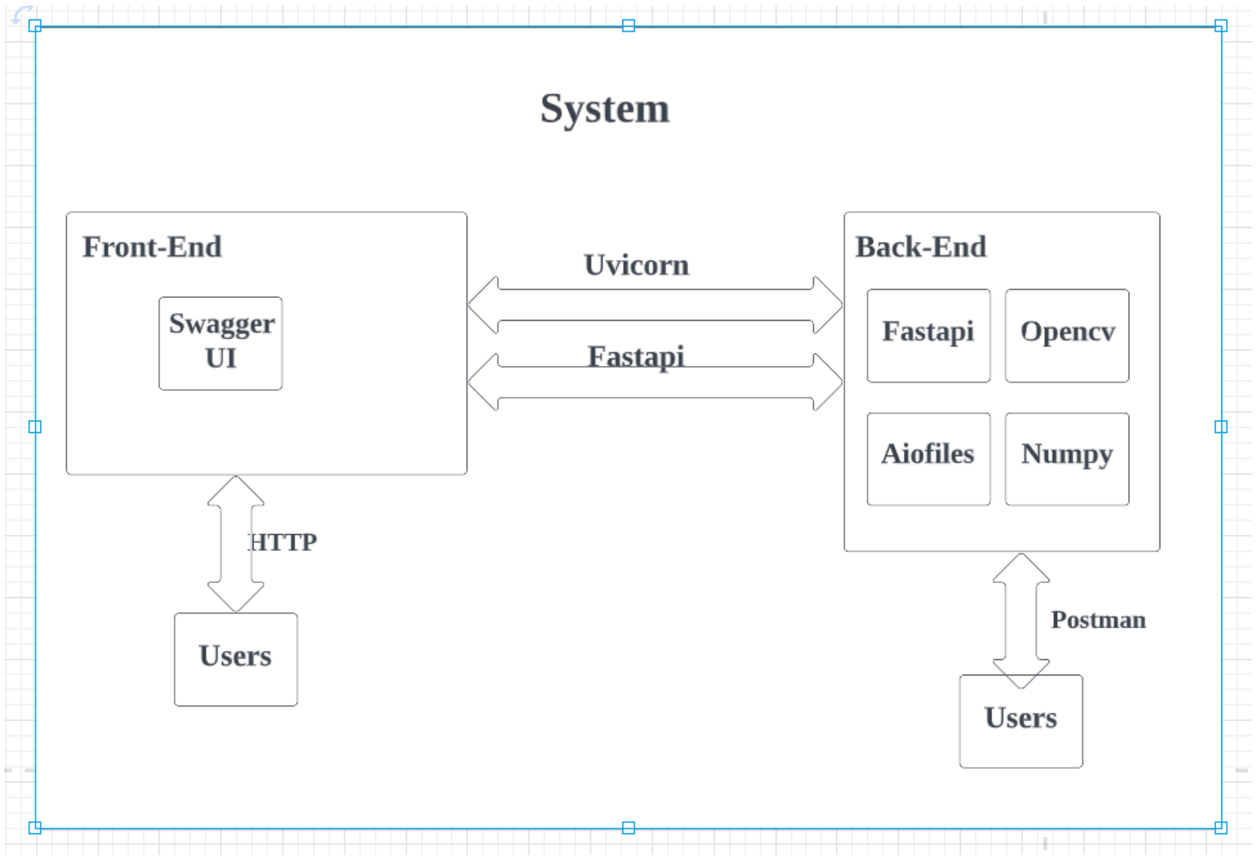
Individual project

Design component:

Overall architecture:



Picture 1 System architecture

The system contains two user interface. The first one is Swagger UI. The second one is postman. Users can upload their images to manipulate via http url or postman get/post methods.

Web UI specifications:

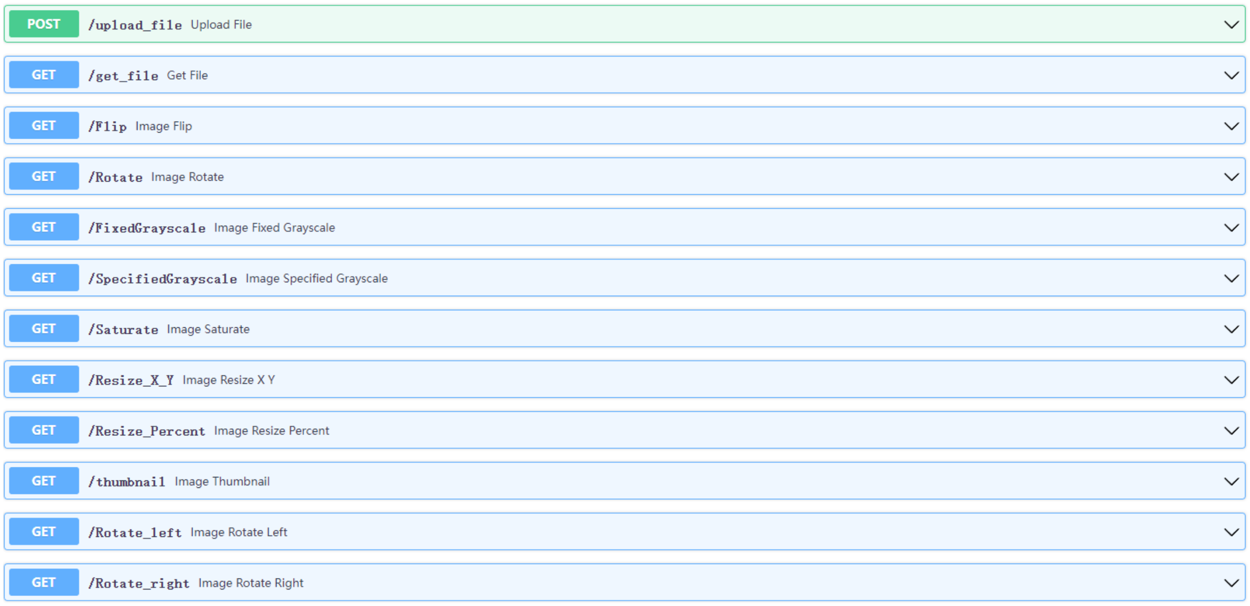
The system can let users manipulate their image by two ways. First, users can upload their image and the system will save the image at disk. Users will get a number of their image. They can choose the way they want and get the result. The organization of it can read from Picture 2. Users should upload their image and get the number of the upload picture. Then tell the system the number of the picture and the way they choose. System will return the result to users. Another way is users send their image to system and system will return the result immediately. System will not save the image data. The logic should be like Picture 3. It looks like parallel method and won't interact with each others.

API specifications:

The fastapi framework combines the UI and API together. When we implemented the back-end logic, fastapi helps me make the swagger UI automatically and it can interact with postman. Same as user specifications. All the user interact way can happen in a same way in Postman. The example shows in later chapter.

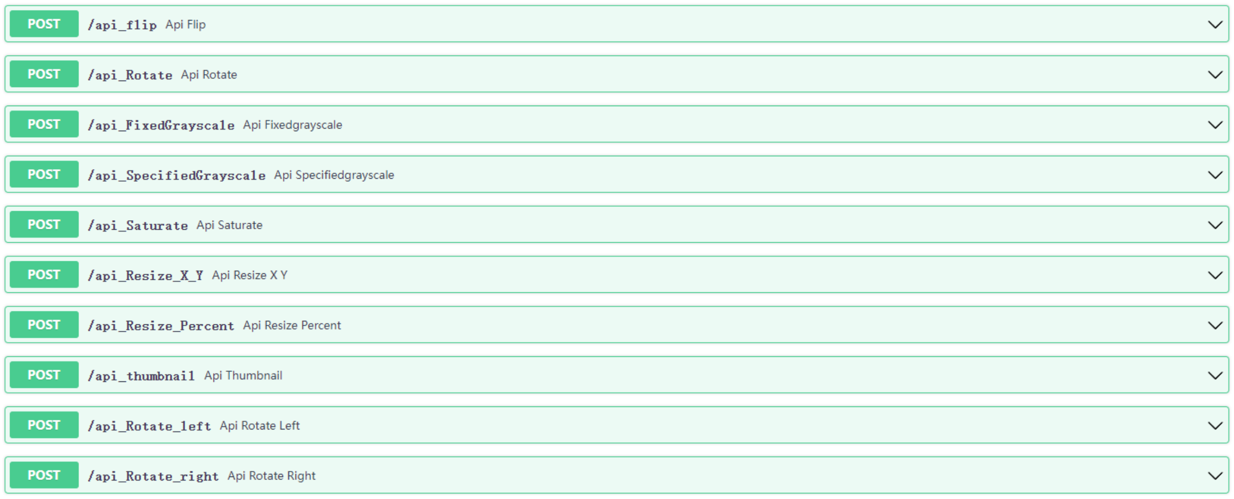
The endpoint descriptions:

The url rule: Postfix with methods name. Plus ? with parameter & input value.
For example: http://127.0.0.1:8000/upload_file
http://127.0.0.1:8000/Resize_X_Y?number=2&X=600&Y=500
http://127.0.0.1:8000/api_Saturate?factor=0.5



POST /upload_file Upload File
GET /get_file Get File
GET /Flip Image Flip
GET /Rotate Image Rotate
GET /FixedGrayscale Image Fixed Grayscale
GET /SpecifiedGrayscale Image Specified Grayscale
GET /Saturate Image Saturate
GET /Resize_X_Y Image Resize X Y
GET /Resize_Percent Image Resize Percent
GET /thumbnail Image Thumbnail
GET /Rotate_left Image Rotate Left
GET /Rotate_right Image Rotate Right

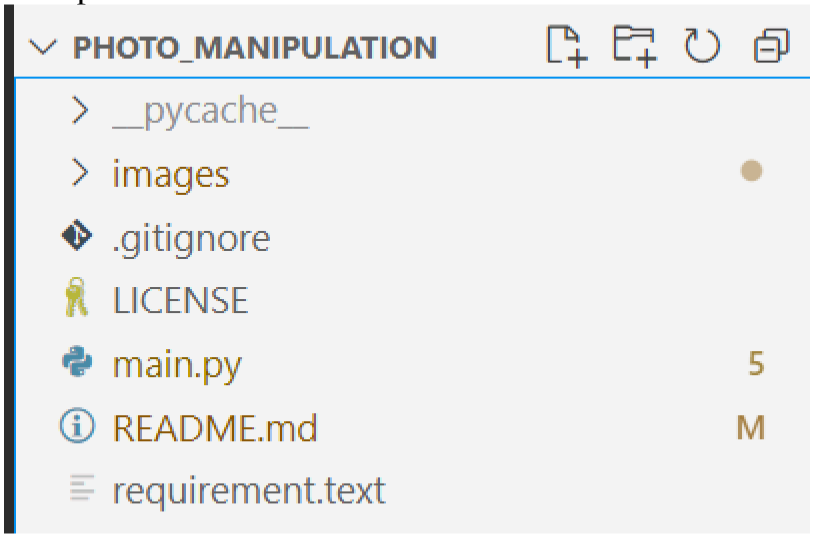
Picture 2 One post image and get result by different get methods



POST /api_flip Api Flip
POST /api_Rotate Api Rotate
POST /api_FixedGrayscale Api Fixedgrayscale
POST /api_SpecifiedGrayscale Api Specifiedgrayscale
POST /api_Saturate Api Saturate
POST /api_Resize_X_Y Api Resize X Y
POST /api_Resize_Percent Api Resize Percent
POST /api_thumbnail Api Thumbnail
POST /api_Rotate_left Api Rotate Left
POST /api_Rotate_right Api Rotate Right

Picture 3 Post image and get the result immediately

Components and Connectors:



PHOTO_MANIPULATION
- __pycache__
- images ●
- .gitignore
- LICENSE
- main.py 5
- README.md M
- requirement.text

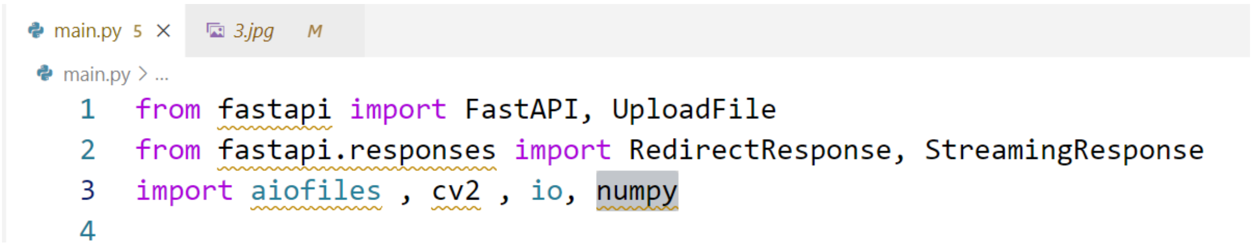
Picture 4 System Organization

The organization is quite simple. Just a python file with all logic and run it. The FastAPI will read all the get/post methods in main.py.

Code component:

FastAPI provides an automatically creating swagger UI when developers implement back-end logic. It is a little inconvenient for developers to change the front-end like use css or js class.

Toolkits:

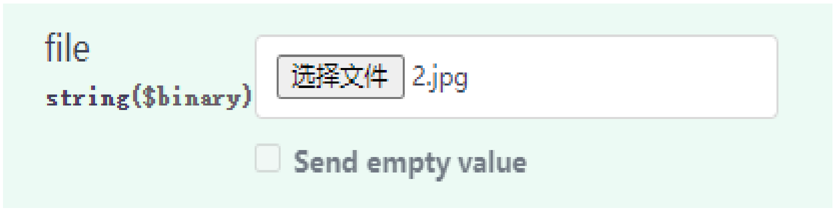


```
main.py 5 ×   3.jpg M
main.py > ...
from fastapi import FastAPI, UploadFile
from fastapi.responses import RedirectResponse, StreamingResponse
import aiofiles , cv2 , io, numpy
```

Picture 5 Tools kit

This is main tools I used to implement the project.

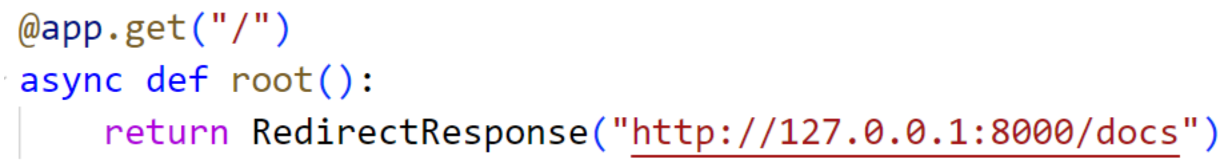
FastAPI is the framework I use.



file
string($binary)
选择文件 2.jpg
Send empty value

Picture 6 UploadFile

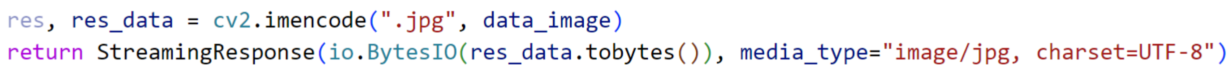
UploadFile is a tool within the FastAPI. Import it for upload file in web page.



```
@app.get("/")
async def root():
    return RedirectResponse("http://127.0.0.1:8000/docs")
```

Picture 7 RedirectResponse to root page

RedirectResponse is a tool within the FastAPI. We can use it to redirect the endpoint.



```
res, res_data = cv2.imencode(".jpg", data_image)
return StreamingResponse(io.BytesIO(res_data.tobytes()), media_type="image/jpg, charset=UTF-8")
```

Picture 8 StreamingResponse to return image data

StreamingResponse is a tool within the FastAPI. We use it for return image data to users. More specific, we need to combine with io to return the image bytes to users.

```python
async with aiofiles.open(file_location, 'wb') as file_object:
    content = await file.read()        # async read
    await file_object.write(content)  # async write
```

Picture 9 aiofiles read file and write file

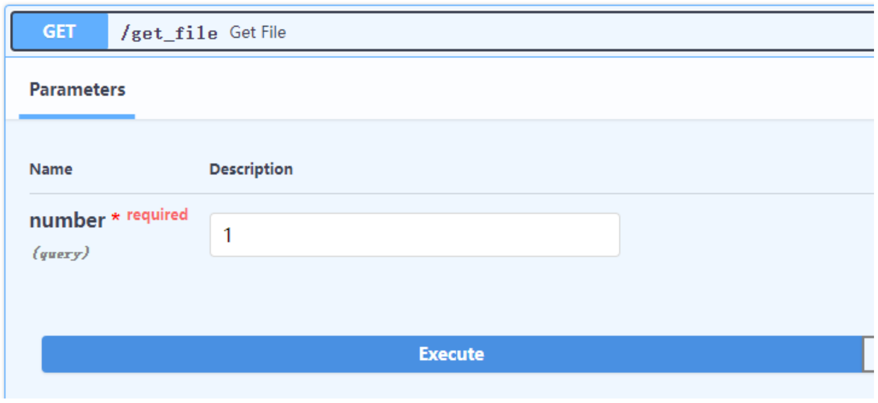
Aiofiles is an external tool for reading and writing image files. Use it for read the upload image and write the image into disk. The original methods of Read/Write are incompatible. It supports asynchronous ways of reading and writing files.

Cv2 is a commonly tool for modifying images.

Get method example:

```python
@app.get("/get_file")
async def get_file(number):
    target_filename = str(number) + ".jpg"
    file_location = f"images/{target_filename}"
    data_image = cv2.imread(file_location)
    res, res_data = cv2.imencode(".jpg", data_image)
    return StreamingResponse(io.BytesIO(res_data.tobytes()), media_type="image/jpg, charset=UTF-8")
```

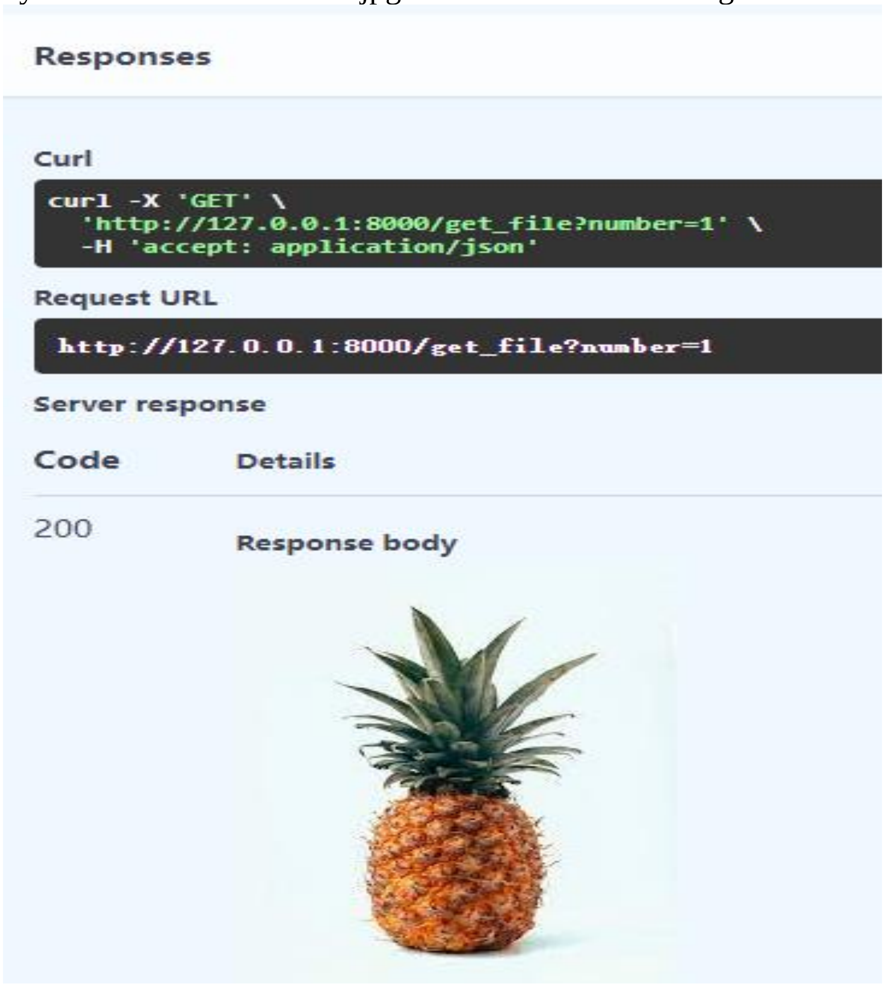
Picture 10 Get method example



Picture 11 Get method Web UI

FastAPI provides a combined way of swagger UI and API logic. We just need to declare there is a get method. It would generate a swagger UI page. When we click every different module, we would redirect to different url. Although they look the same as each other.
For example:
Get file url: http://127.0.0.1:8000/docs#/default/get_file_get_file_get
Image flip url: http://127.0.0.1:8000/docs#/default/image_Flip_Flip_get
When you click different module, it will jump into different url. Postman will get data from these different url.

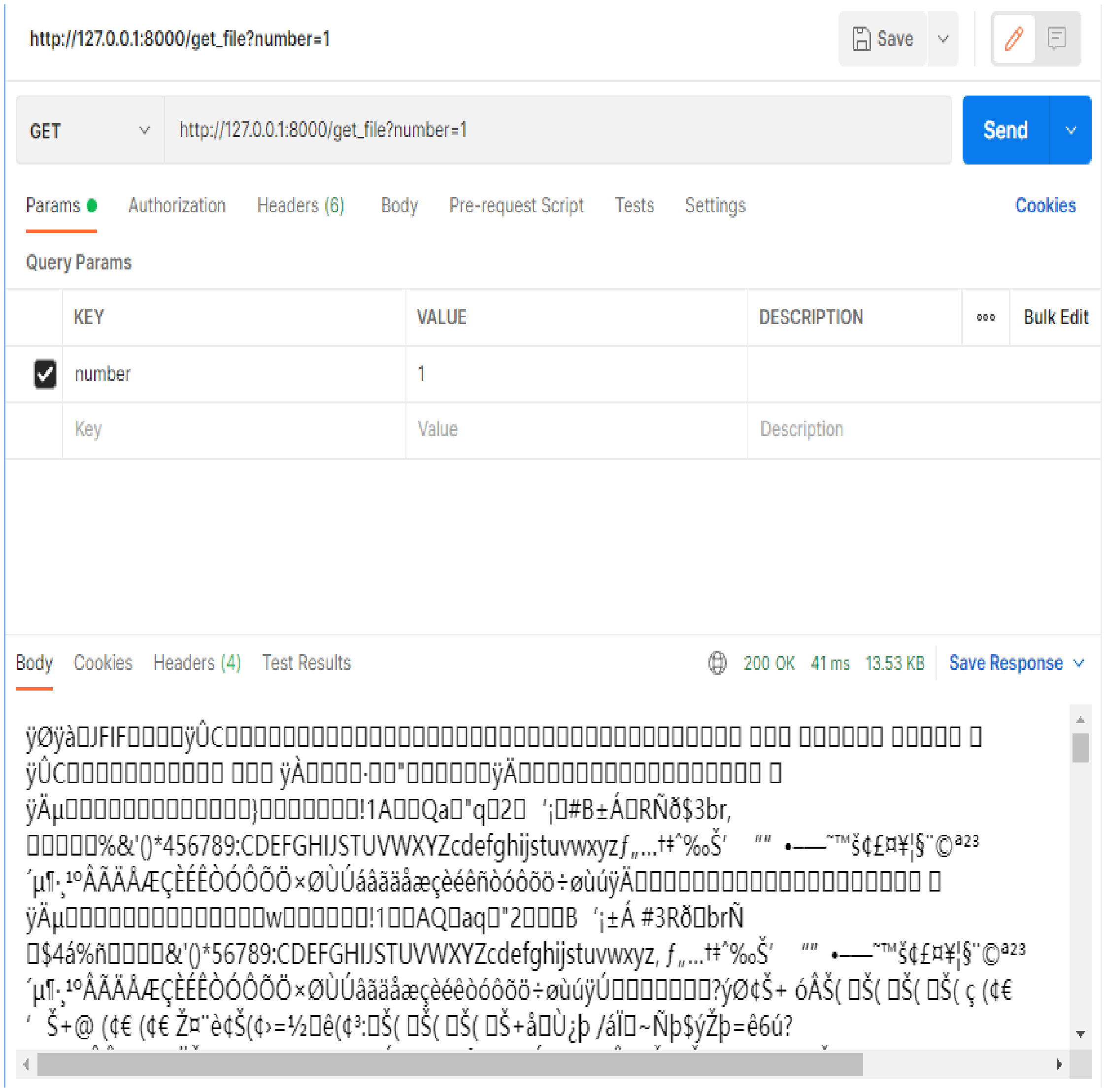
Here are two accessing image ways. One is provide the image number and them the page would display the image data. Another ways is use Postman get method. The Postman would display bytes. Just save the data as .jpg and it will show the image.



```
curl -X 'GET' \
  'http://127.0.0.1:8000/get_file?number=1' \
  -H 'accept: application/json'
```

```
http://127.0.0.1:8000/get_file?number=1
```

Picture 12 Get method Web Response example


http://127.0.0.1:8000/get_file?number=1

GET http://127.0.0.1:8000/get_file?number=1

Query Params

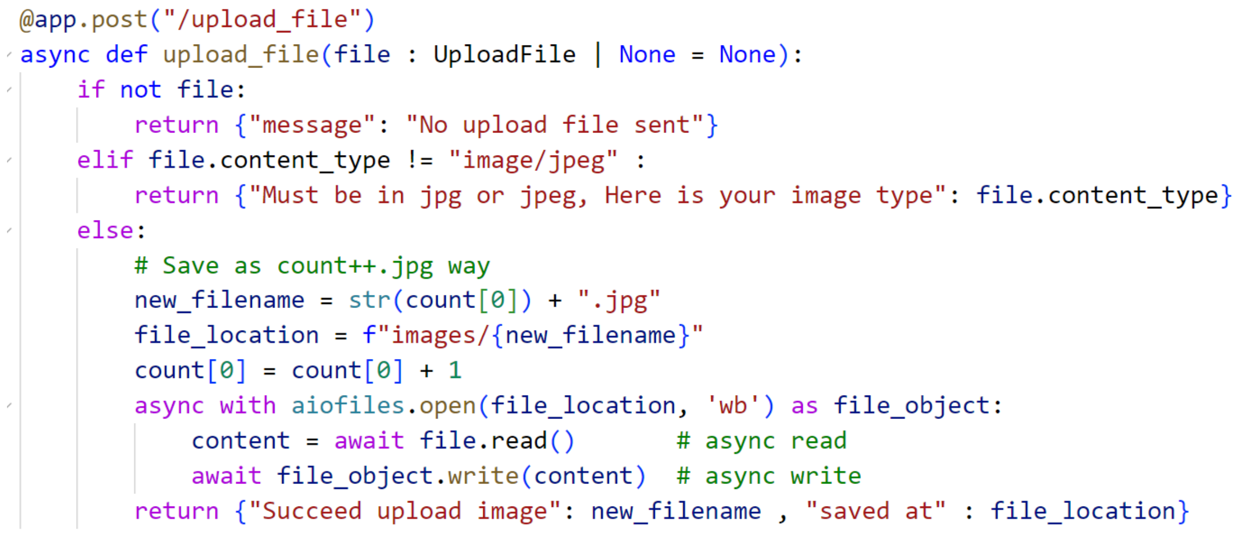
	KEY	VALUE	DESCRIPTION
✓	number	1	
	Key	Value	Description

Body · 200 OK · 41 ms · 13.53 KB

```
ÿØÿà□JFIF□□□□ÿÛC□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□ □□□□□ □
ÿÛC□□□□□□□□□□□□ □□□ ÿÀ□□□□·□□"□□□□□□ÿÄ□□□□□□□□□□□□□□□□□□□□ □
ÿÄµ□□□□□□□□□□□□□□}□□□□□□□□!1A□□Qa□"q□2□  '¡□#B±Á□RÑð$3br,
□□□□□%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyzƒ„…†‡ˆ‰Š'   ""  •–—˜™š¢£¤¥¦§¨©ª²³
´µ¶·¸¹ºÂÃÄÅÆÇÈÉÊÒÓÔÕÖ×ØÙÚáâãäåæçèéêñòóôõö÷øùúÿÄ□□□□□□□□□□□□□□□□□□□□□□□□ □
ÿÄµ□□□□□□□□□□□□□□□w□□□□□□!1□□AQ□aq□"2□□□B  '¡±Á #3Rð□brÑ
□$4á%ñ□□□□&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz, ƒ„…†‡ˆ‰Š'   ""  •–—˜™š¢£¤¥¦§¨©ª²³
´µ¶·¸¹ºÂÃÄÅÆÇÈÉÊÒÓÔÕÖ×ØÙÚâãäåæçèéêòóôõö÷øùúÿÚ□□□□□□□□?ýØ¢Š+ óÂŠ( □Š( □Š( □Š( ç (¢€
'  Š+@ (¢€ (¢€ Ž¤¨è¢Š(¢›=½□ê(¢³:□Š( □Š( □Š( □Š+å□Ù¿þ /áÏ□~Ñþ$ýŽþ=ê6ú?
```

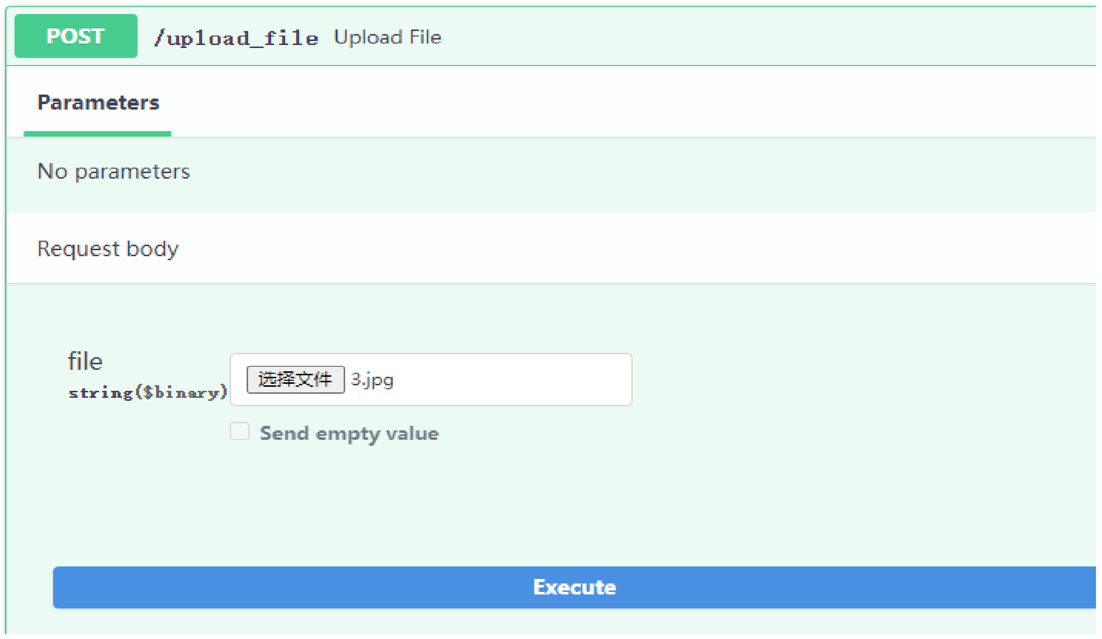
Picture 13 Get method Postman example

Post method example:



```
@app.post("/upload_file")
async def upload_file(file : UploadFile | None = None):
    if not file:
        return {"message": "No upload file sent"}
    elif file.content_type != "image/jpeg" :
        return {"Must be in jpg or jpeg, Here is your image type": file.content_type}
    else:
        # Save as count++.jpg way
        new_filename = str(count[0]) + ".jpg"
        file_location = f"images/{new_filename}"
        count[0] = count[0] + 1
        async with aiofiles.open(file_location, 'wb') as file_object:
            content = await file.read()        # async read
            await file_object.write(content)  # async write
        return {"Succeed upload image": new_filename , "saved at" : file_location}
```

Picture 14 Post method example



POST /upload_file Upload File

Parameters

No parameters

Request body

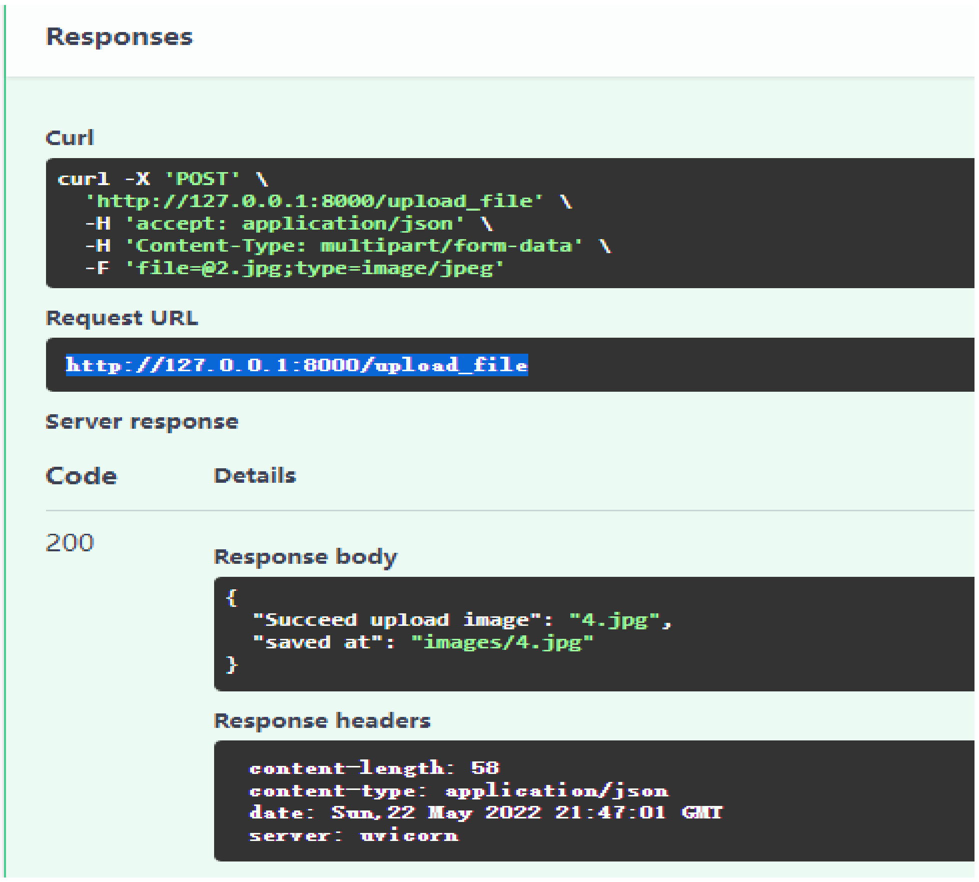
file
string($binary)
选择文件 3.jpg

Send empty value

Execute

Picture 15 Post method Web UI

Same as get method. We can upload the image by web UI or postman. The post url shows in every module.


Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/upload_file' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@2.jpg;type=image/jpeg'
```

Request URL

```
http://127.0.0.1:8000/upload_file
```

Server response

Code Details

200

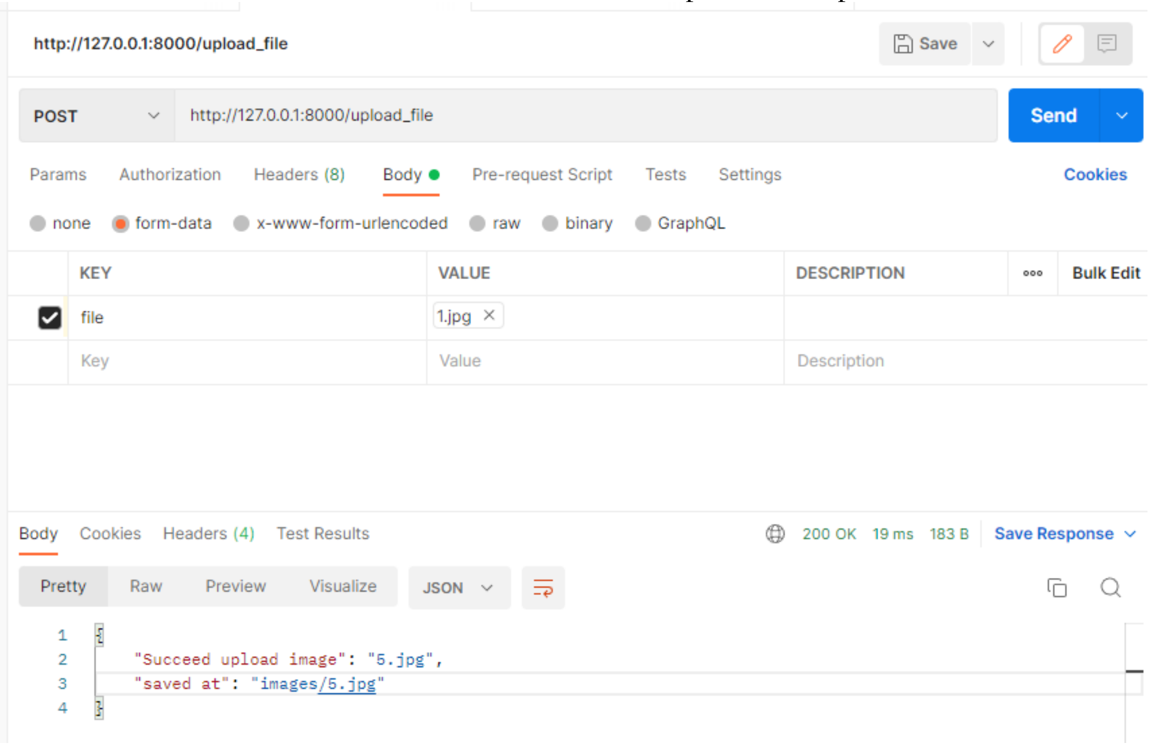
Response body

```
{
  "Succeed upload image": "4.jpg",
  "saved at": "images/4.jpg"
}
```

Response headers

```
 content-length: 58
 content-type: application/json
 date: Sun,22 May 2022 21:47:01 GMT
 server: uvicorn
```

Picture 16 Post method Web Response example



http://127.0.0.1:8000/upload_file

POST http://127.0.0.1:8000/upload_file

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE	DESCRIPTION
✓	file	1.jpg	
	Key	Value	Description

Body Cookies Headers (4) Test Results 200 OK 19 ms 183 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "Succeed upload image": "5.jpg",
    "saved at": "images/5.jpg"
}
```

Picture 17 Post method Postman example