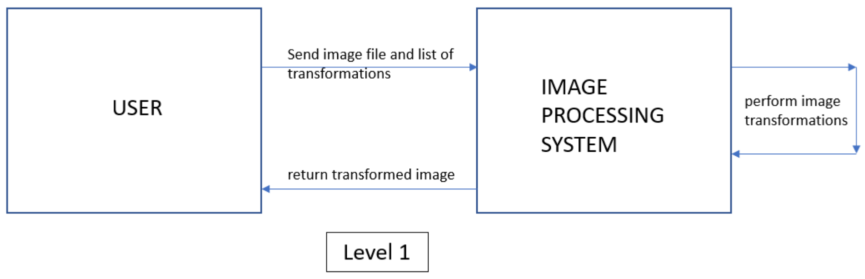
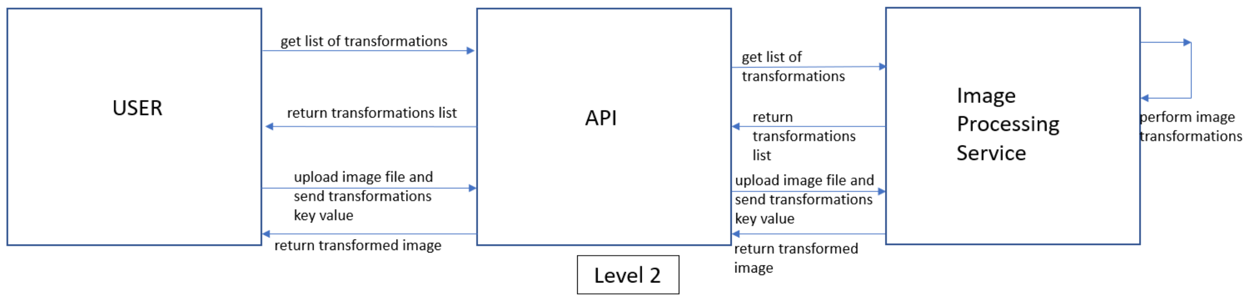
Image Processor

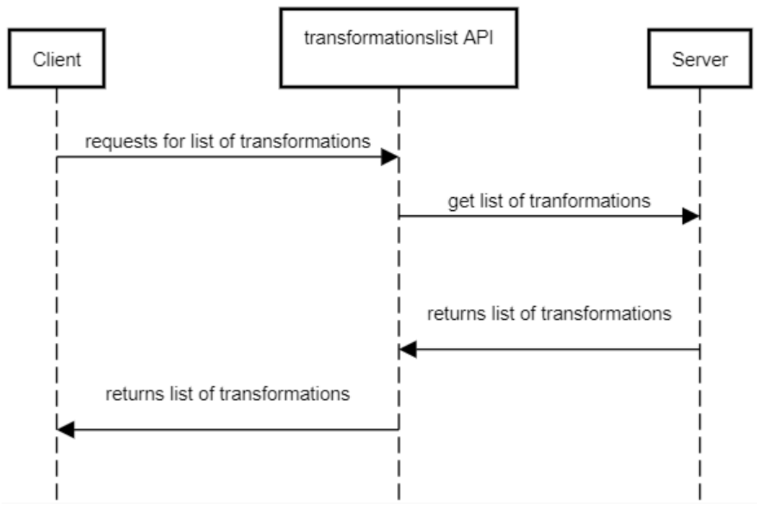
High Level Architecture


USER
Send image file and list of transformations
IMAGE PROCESSING SYSTEM
perform image transformations
return transformed image
Level 1

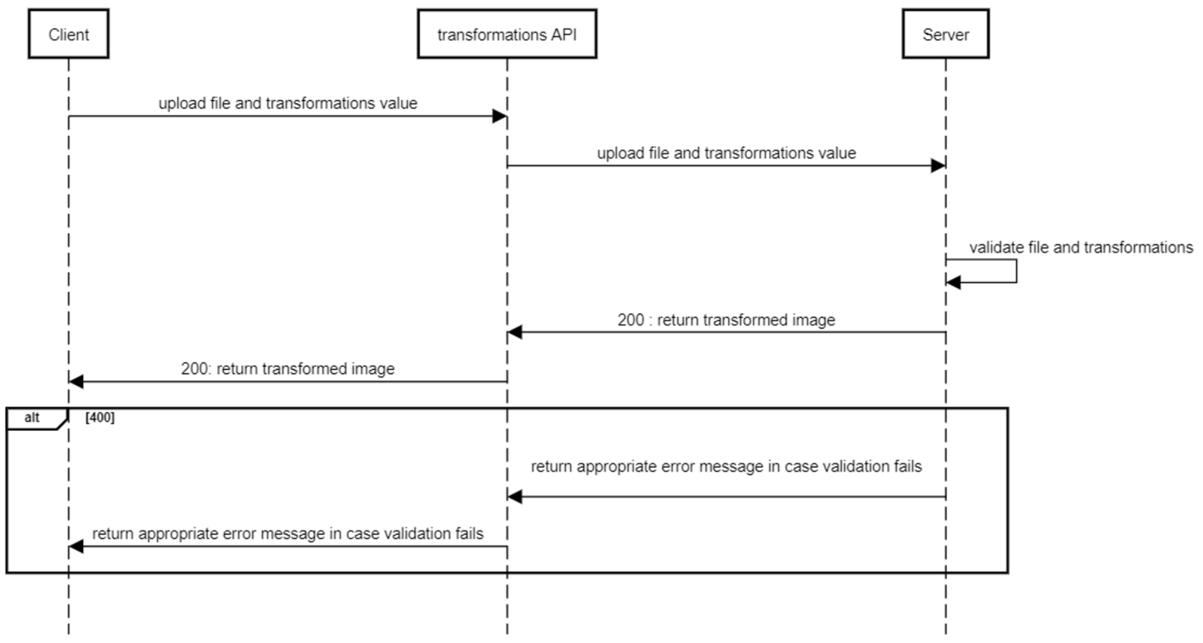

USER
get list of transformations
return transformations list
upload image file and send transformations key value
return transformed image
API
get list of transformations
return transformations list
upload image file and send transformations key value
return transformed image
Image Processing Service
perform image transformations
Level 2

Sequence Diagram

Transformationslist API


Client
transformationslist API
Server
requests for list of transformations
get list of tranformations
returns list of transformations
returns list of transformations

Transformations API


Client
transformations API
Server
upload file and transformations value
upload file and transformations value
validate file and transformations
200 : return transformed image
200: return transformed image
alt
[400]
return appropriate error message in case validation fails
return appropriate error message in case validation fails

API

Architectural Style – client server

- Client initiates actions and server responds back

REST API

- Following a client server architecture where requests are managed through HTTP.
- Communication between client and server is stateless, server is not storing any client information between requests
- Each request is separate and not - connected

API implemented:

- transformationslist (GET)
- transformations (POST)

Language:

- ➢ Python – 3.10.2

Framework:

- ➢ Flask

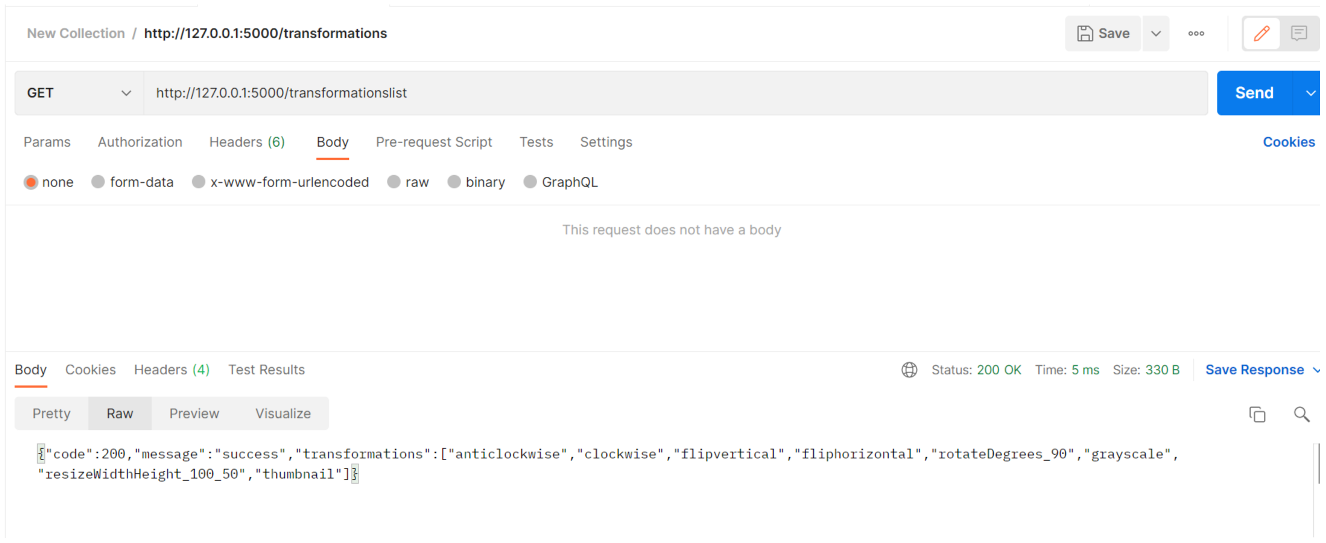
Libraries:

- ➢ Pillow

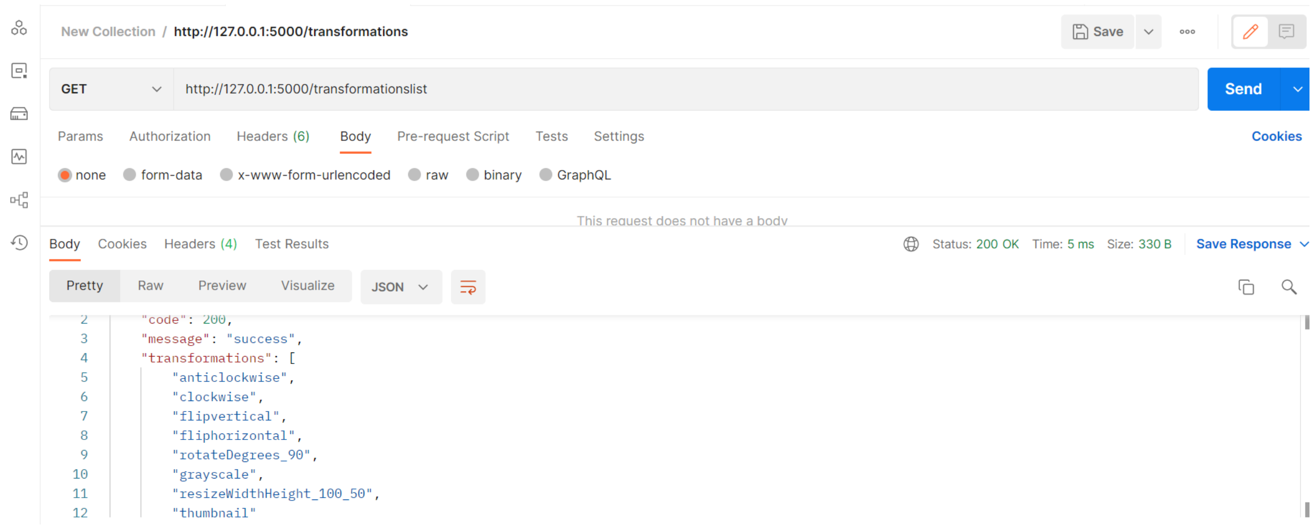
TransformationsList API

- **Url:** http://127.0.0.1:5000/transformationslist
- **Request Type**: Get
- **Keys**: None
- **Description**: This API gets the list of transformations that can be performed by the image processor sever. It also gives the list of keys to be provided for rotate and resize transformations.

Sample Response:


New Collection / http://127.0.0.1:5000/transformations
Save
GET
http://127.0.0.1:5000/transformationslist
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL
This request does not have a body
Body Cookies Headers (4) Test Results
Status: 200 OK Time: 5 ms Size: 330 B
Save Response
Pretty Raw Preview Visualize

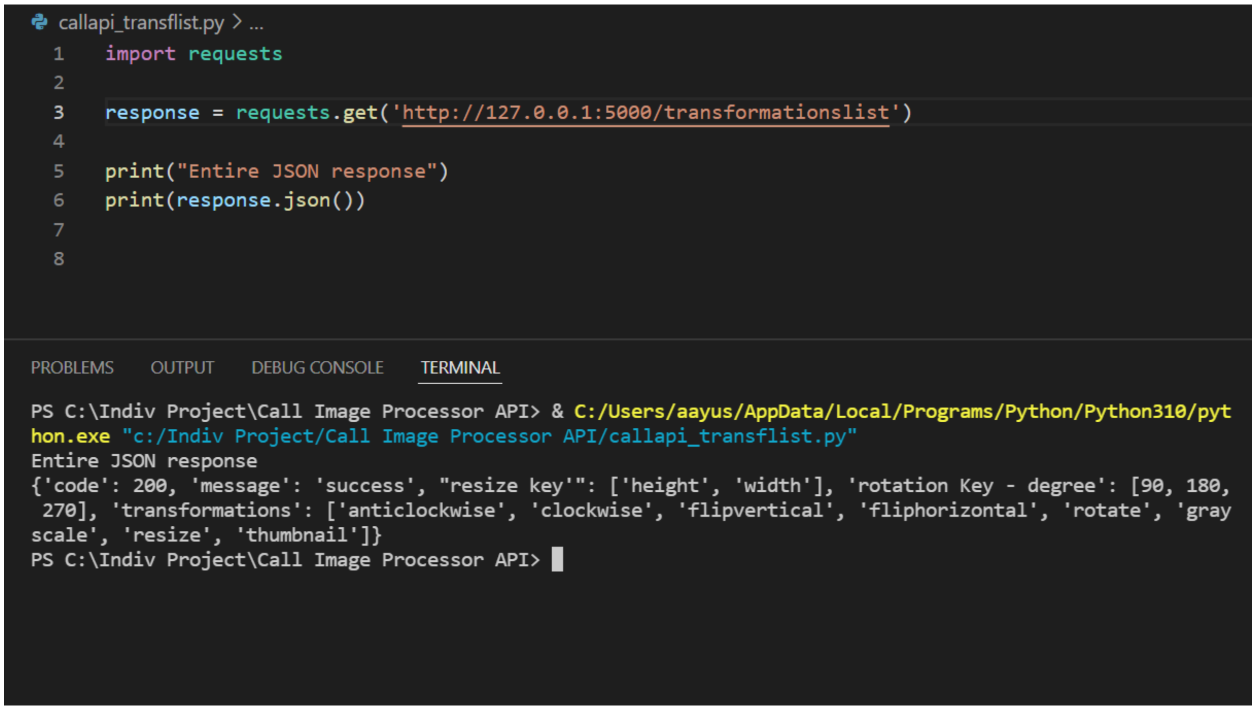
```
{"code":200,"message":"success","transformations":["anticlockwise","clockwise","flipvertical","fliphorizontal","rotateDegrees_90","grayscale",
"resizeWidthHeight_100_50","thumbnail"]}
```


New Collection / http://127.0.0.1:5000/transformations
Save
GET
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL
This request does not have a body
Body Cookies Headers (4) Test Results
Status: 200 OK Time: 5 ms Size: 330 B
Save Response
Pretty Raw Preview Visualize JSON

```
http://127.0.0.1:5000/transformationslist
```

```
    "code": 200,
    "message": "success",
    "transformations": [
        "anticlockwise",
        "clockwise",
        "flipvertical",
        "fliphorizontal",
        "rotateDegrees_90",
        "grayscale",
        "resizeWidthHeight_100_50",
        "thumbnail"
```

Sample code to access API:



```python
import requests

response = requests.get('http://127.0.0.1:5000/transformationslist')

print("Entire JSON response")
print(response.json())
```

```
PS C:\Indiv Project\Call Image Processor API> & C:/Users/aayus/AppData/Local/Programs/Python/Python310/pyt
hon.exe "c:/Indiv Project/Call Image Processor API/callapi_transflist.py"
Entire JSON response
{'code': 200, 'message': 'success', "resize key'": ['height', 'width'], 'rotation Key - degree': [90, 180,
 270], 'transformations': ['anticlockwise', 'clockwise', 'flipvertical', 'fliphorizontal', 'rotate', 'gray
scale', 'resize', 'thumbnail']}
PS C:\Indiv Project\Call Image Processor API>
```

Transformations API

- **Url:** http://127.0.0.1:5000/transformations
- **Request Type:** Post
- **Keys:** file, transformations
- **Description:** This API takes an image file, a list of parameters and sends those to the Image Processor server. The server first validates the request file and mandatory keys required for a successful transformation. Upon successful validation it performs transformation operations in the same order as specified in the request. After a successful transformation, it returns the transformed image to the end user. They can view the response and save it.

  <u>Request Parameters:</u>

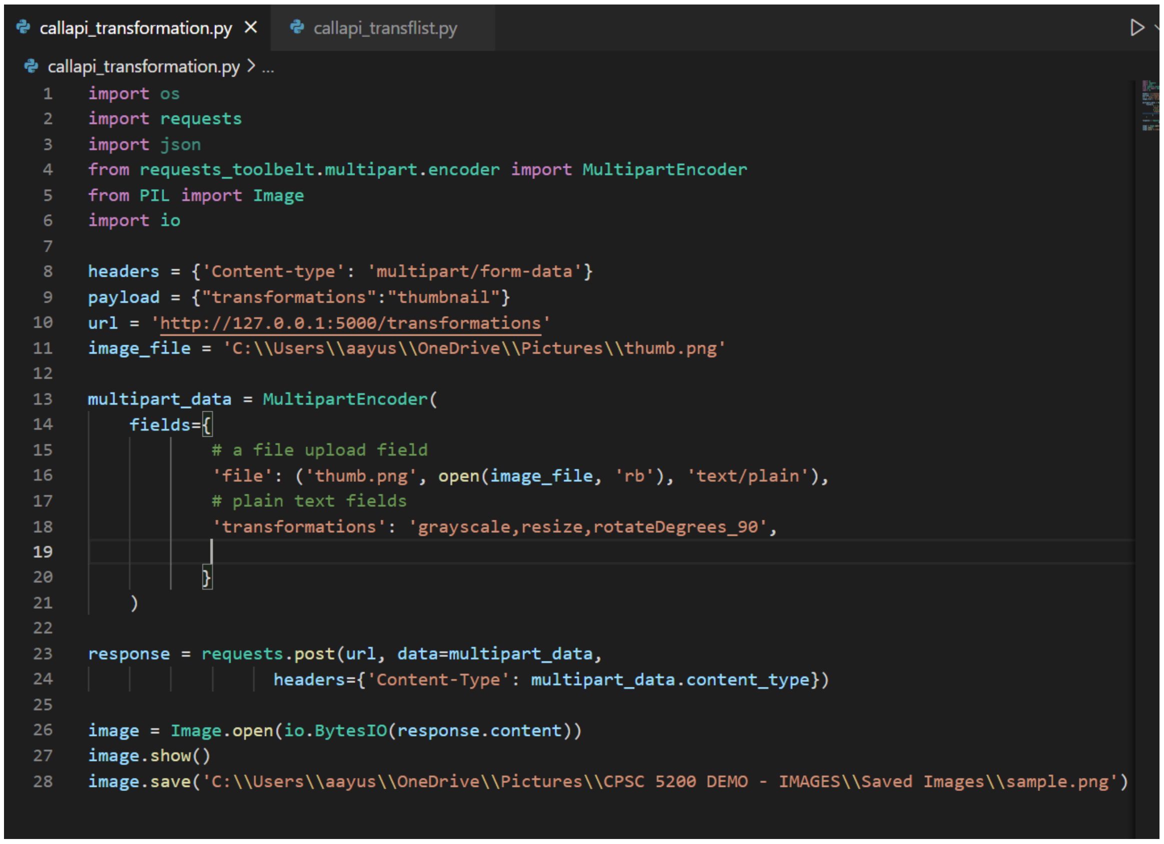
- ➢ file - Allowed file types are png, jpg, jpeg
- ➢ transformations: Allowed transformations are anticlockwise, clockwise, fliphorizontal, flipvertical,resizeWidthHeight_100_50,rotateDegrees_90,grayscale,thumbnail
- ➢ resize – width and height should be passed as an integer value separated by "_". Example: resizeWidthHeight_100_50
- ➢ rotate – degree should be passed an integer value separated by "_". Example: rotateDegrees_90

- **<u>Response:</u>**
- Image file which is accessible to the end user. They can choose to save it.

<u>Error Handling</u>

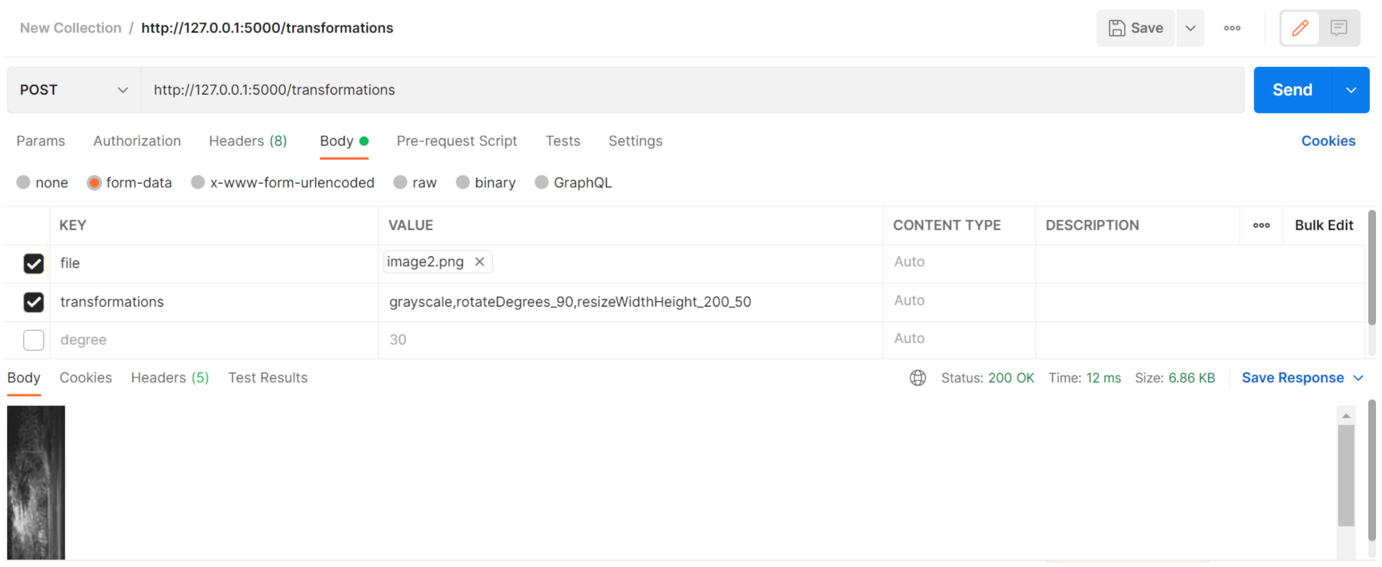
Scenario	Server Response
No file in the request	Status Code – 400; Message: No file part in the request
File key is there but value is missing	Status Code – 400; Message: No file selected for uploading
Uploaded file is not an image	Status Code – 400; Message: Allowed file types are png, jpg, jpeg
Missing transformations key	Status Code – 400; Message: please enter required transformations.
Transformation key is there but value is invalid	Status Code – 400; Message: please enter valid transformation(s).
Non integer height or width for resize	Status Code – 400; Message: Please provide valid integers for height and width.
Non integer degrees for rotation	Status Code – 400; Message: Please provide valid integers for degree.

Sample Code to call transformationsAPI



```python
import os
import requests
import json
from requests_toolbelt.multipart.encoder import MultipartEncoder
from PIL import Image
import io

headers = {'Content-type': 'multipart/form-data'}
payload = {"transformations":"thumbnail"}
url = 'http://127.0.0.1:5000/transformations'
image_file = 'C:\\Users\\aayus\\OneDrive\\Pictures\\thumb.png'

multipart_data = MultipartEncoder(
    fields={
            # a file upload field
            'file': ('thumb.png', open(image_file, 'rb'), 'text/plain'),
            # plain text fields
            'transformations': 'grayscale,resize,rotateDegrees_90',
            
        }
    )

response = requests.post(url, data=multipart_data,
                         headers={'Content-Type': multipart_data.content_type})

image = Image.open(io.BytesIO(response.content))
image.show()
image.save('C:\\Users\\aayus\\OneDrive\\Pictures\\CPSC 5200 DEMO - IMAGES\\Saved Images\\sample.png')
```

Example of server response:

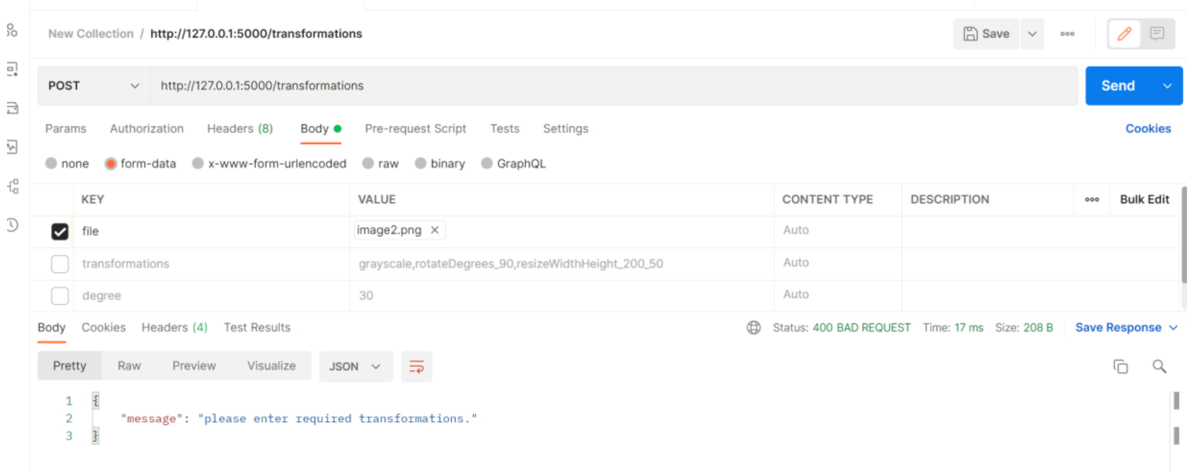


POST http://127.0.0.1:5000/transformations

Body: form-data

	KEY	VALUE	CONTENT TYPE	DESCRIPTION
✓	file	image2.png	Auto	
✓	transformations	grayscale,rotateDegrees_90,resizeWidthHeight_200_50	Auto	
	degree	30	Auto	

Status: 200 OK Time: 12 ms Size: 6.86 KB transformations are not added in the request:



New Collection / http://127.0.0.1:5000/transformations

POST http://127.0.0.1:5000/transformations

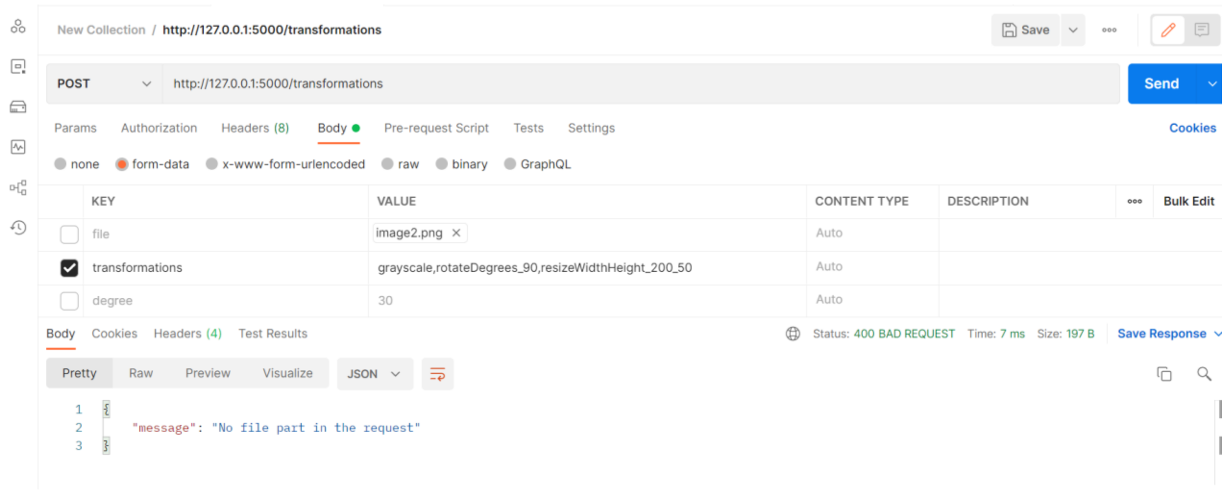
Body: form-data

	KEY	VALUE	CONTENT TYPE	DESCRIPTION
✓	file	image2.png	Auto	
	transformations	grayscale,rotateDegrees_90,resizeWidthHeight_200_50	Auto	
	degree	30	Auto	

Status: 400 BAD REQUEST Time: 17 ms Size: 208 B

```
{
    "message": "please enter required transformations."
}
```

file is not added:



New Collection / http://127.0.0.1:5000/transformations

POST http://127.0.0.1:5000/transformations

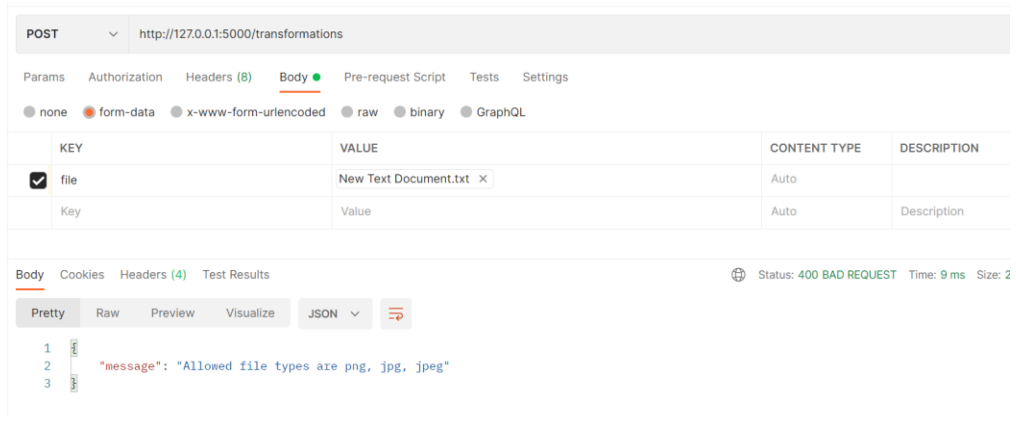
Body: form-data

	KEY	VALUE	CONTENT TYPE	DESCRIPTION
	file	image2.png	Auto	
✓	transformations	grayscale,rotateDegrees_90,resizeWidthHeight_200_50	Auto	
	degree	30	Auto	

Status: 400 BAD REQUEST Time: 7 ms Size: 197 B

```
{
    "message": "No file part in the request"
}
```

Incorrect file type:



POST http://127.0.0.1:5000/transformations

Body: form-data

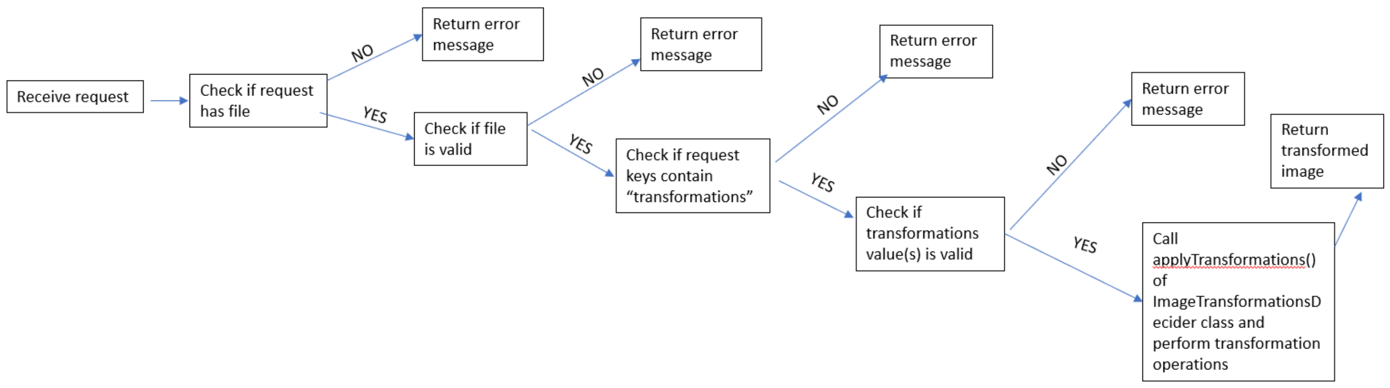
	KEY	VALUE	CONTENT TYPE	DESCRIPTION
✓	file	New Text Document.txt	Auto	
	Key	Value	Auto	Description

Status: 400 BAD REQUEST Time: 9 ms Size: 2

```
{
    "message": "Allowed file types are png, jpg, jpeg"
}
```

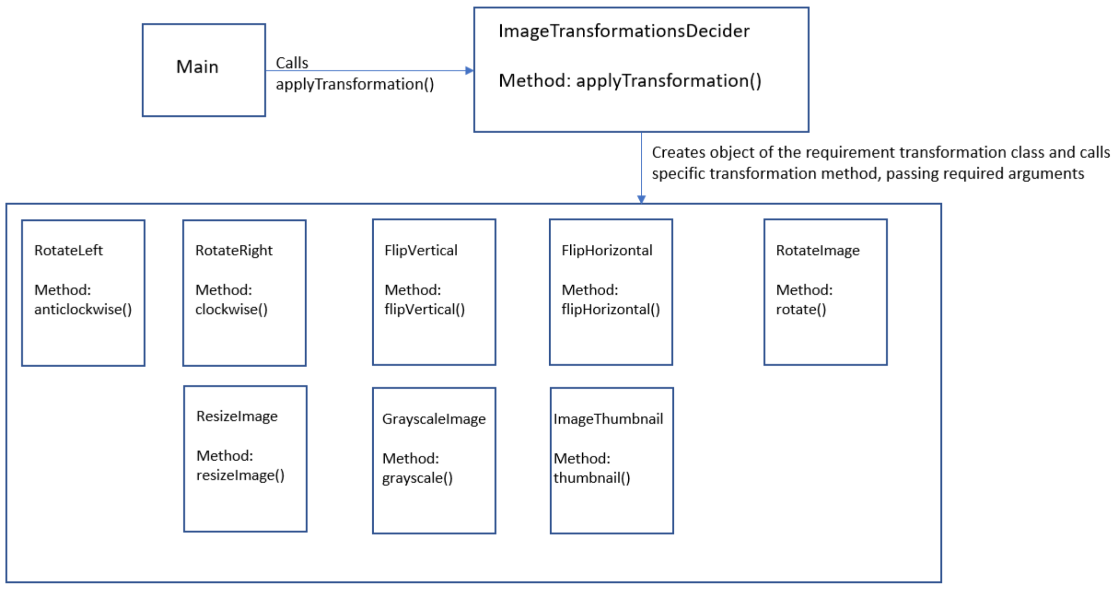
Image Processor Service

Image Processor receives request and checks if the file is present in the request or not. If not, it returns an error message. If file is present, then it validates file type ensure it is one of the allowed image files. In case uploaded file is not correct, it will return an error message. If the uploaded image file is valid, then it will further check if the request form key has transformations or not. If it is present, then it verifies if the value is a valid transformation operation. It returns an error message if transformations are not present in the request or if it contains an invalid value. If transformation values are correct, then it calls applyTransformations() by passing image and transformation values.



Receive request
Check if request has file
NO
Return error message
YES
Check if file is valid
NO
Return error message
YES
Check if request keys contain "transformations"
NO
Return error message
YES
Check if transformations value(s) is valid
NO
Return error message
YES
Call applyTransformations() of ImageTransformationsDecider class and perform transformation operations
Return transformed image

Design Pattern: Factory Design Pattern



Learnings

- I first started with OpenCV library, and later upon receiving feedback switched to Pillow. It helped a lot as the suggested library for Image Processing in Python is Pillow. OpenCV is basically a computer vision library.
- I also messed up my little language by passing values for height, width and degrees separately as individual key value pairs. I updated my little language after receiving feedback and now values for height, width and degree are passed along with the transformation type, separated by an underscore ("_").
- Initially I created individual APIs for transformations. Later on, I followed Factory Design Pattern approach and merged all into one API and created individual classes (end points) for different transformations.

Trade Offs

- Though OpenCV is faster, I used Pillow as it is the one of the best libraries available for Image Processing.
- Converting jpg images to png internally because underlying library was giving KeyError for jpg format.