Overview:

This document will outline the architecture to implement the Image processor.

Problem Statement:

We need to develop a simple image processor application. The end user provides an image in some format and allows the user to perform combinations of the following operations:

- Flip horizontal and vertical
- Rotate +/- in degrees
- Convert to grayscale
- Resize
- Generate a thumbnail
- Rotate left
- Rotate right

The user can specify which operation or operations to perform on the image. Upon completion of the transform the user should have access to the resulting image file. Operations can be applied in an order specified by the caller.

Assumptions:

- We are supporting these 5 image formats on the application- 'png', 'jpeg','tiff', 'bmp','gif'. Supporting other formats would be considered as P1 considering the complexity.
- UI - User must select all the transformations that he wants to apply at once. (Sequence of the transformations).
- We are supporting 15 transformations in a single request not to overhead the processor.
- Maximum size of the image is 3000 x 3000 pixels
- The size of the image should be less than 200 MB. That is to support the feature with a minimal latency to give the best customer experience. Uploading large images like 1GB would overhead the backend processor.

- We have an upper and lower limit to resize the image. The resize parameter can't be more than 3000x3000 and not less than 10x10.

Definitions:

- React JS - React is a free and open-source front-end JavaScript toolkit for creating UI components-based user interfaces.
- Python Flask web framework will be used in the backend.
- The Pillow library will be used for transformations.

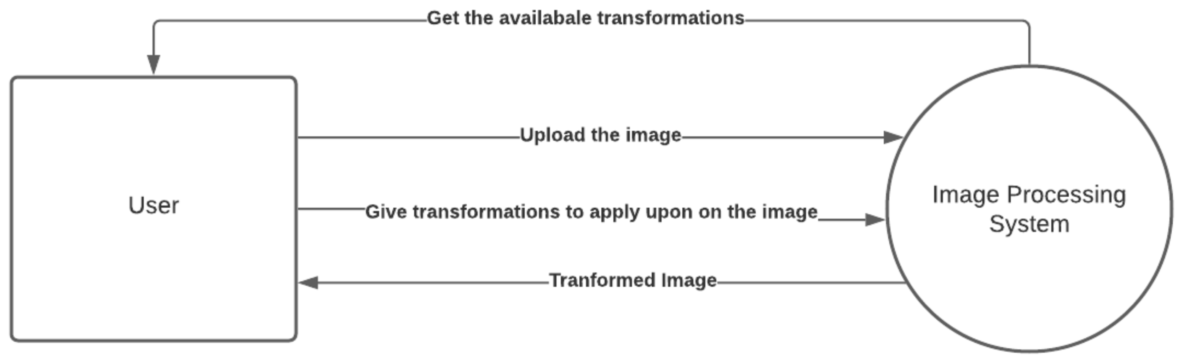
High level architecture:

We need to build an architecture that supports all these features like uploading an image, applying the transformations on the image, and retrieving the final output image.

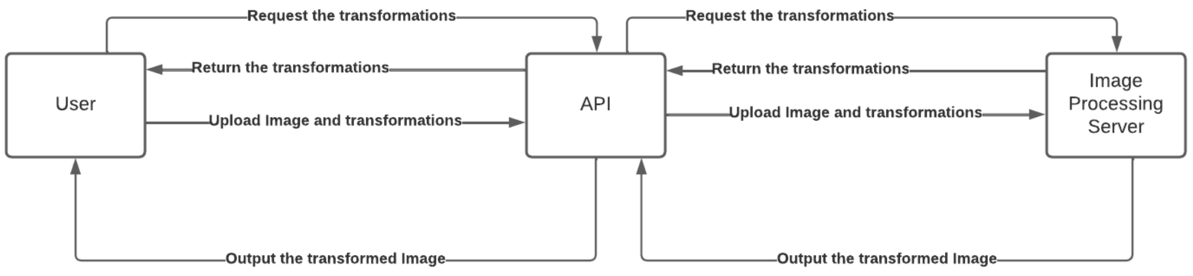
Client side: Client will upload image and select the transformations list to the image processing server.

Server side: Image processing server will take image, image transformations as input, apply the transformations and then hand the transformed image as output.

Level 1: High level view of the architecture

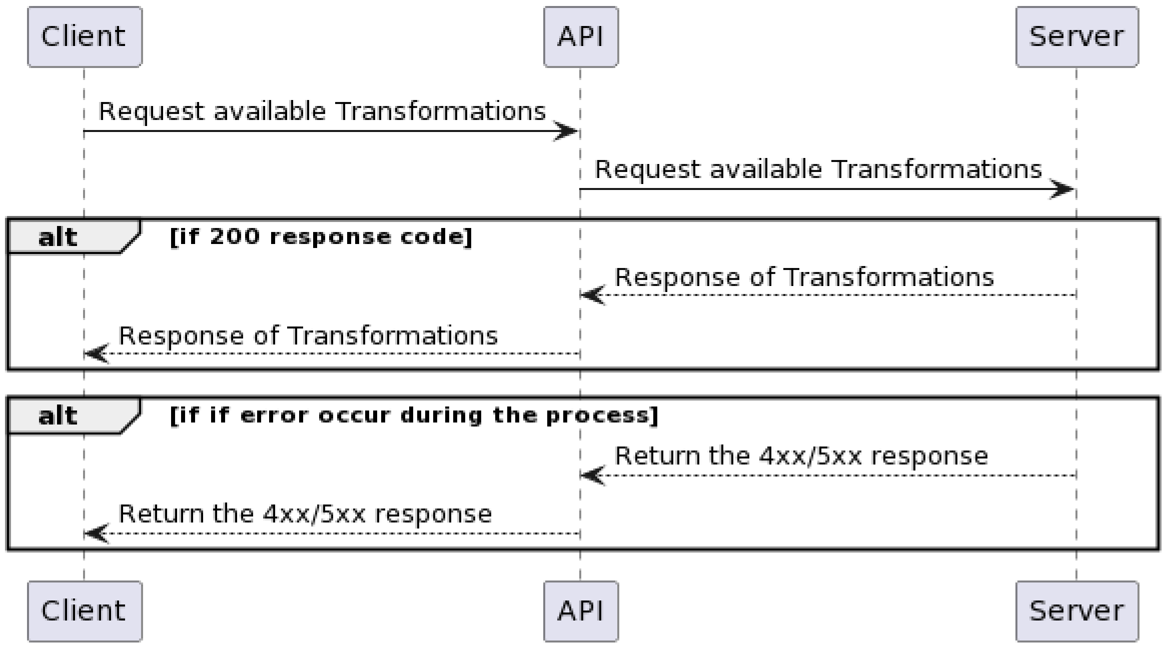

Get the availabale transformations
User
Upload the image
Give transformations to apply upon on the image
Tranformed Image
Image Processing System

Level 2 : Detailed view of the architecture


Request the transformations
Request the transformations
User
Return the transformations
API
Return the transformations
Image Processing Server
Upload Image and transformations
Upload Image and transformations
Output the transformed Image
Output the transformed Image

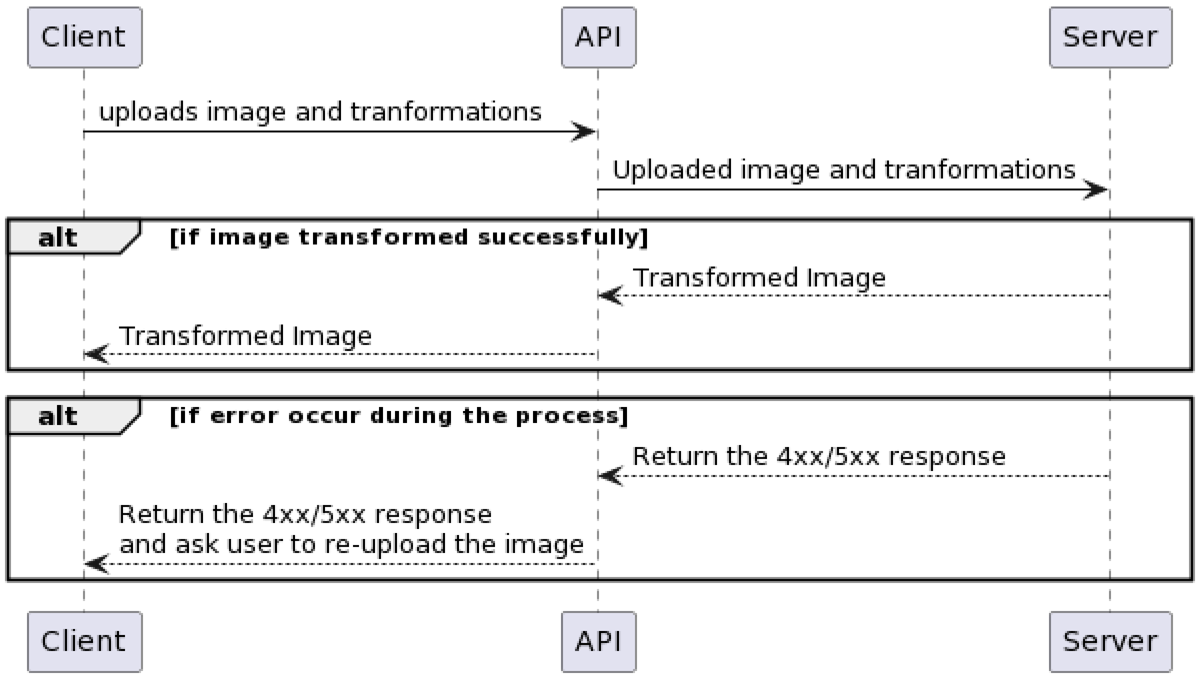
Sequence Diagrams:

Request for transformations:


Client
API
Server
Request available Transformations
Request available Transformations
alt [if 200 response code]
Response of Transformations
Response of Transformations
alt [if if error occur during the process]
Return the 4xx/5xx response
Return the 4xx/5xx response
Client
API
Server

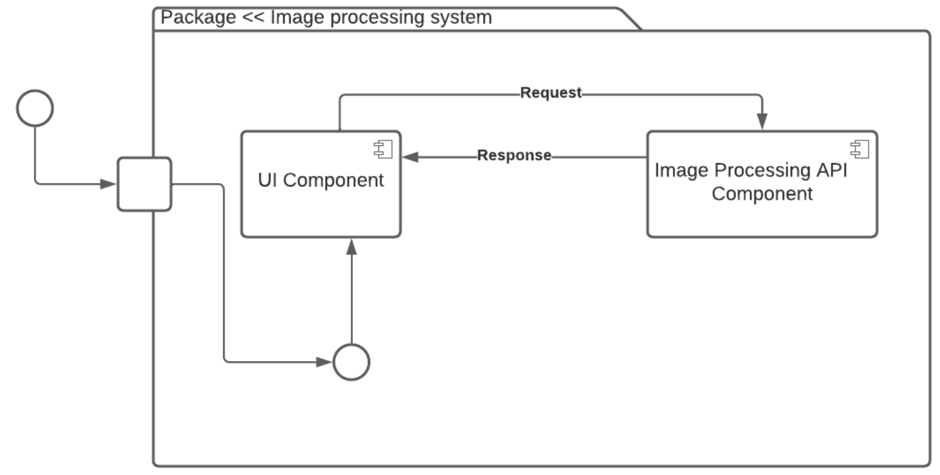
(edit)

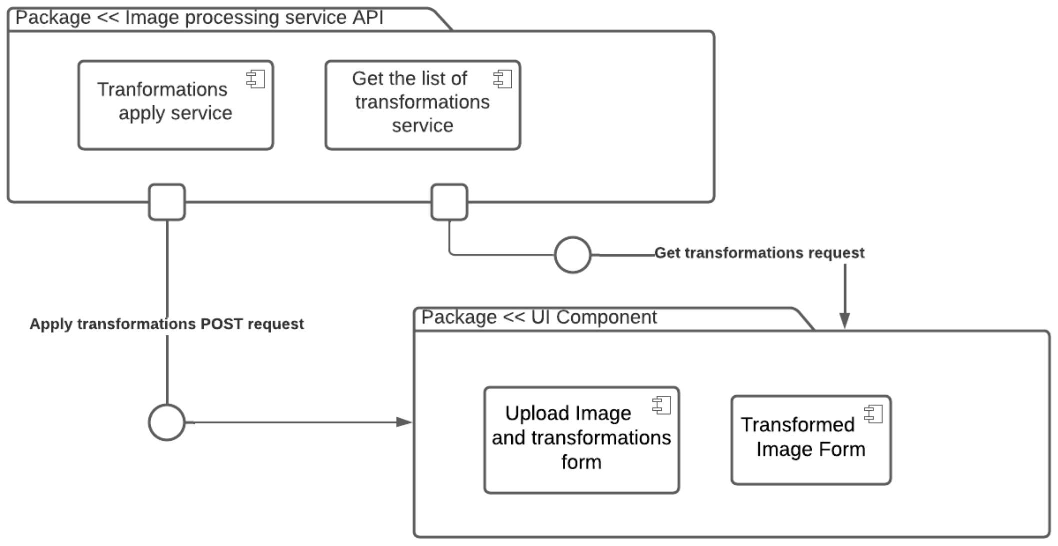
Apply transformations:


Client
API
Server
uploads image and tranformations
Uploaded image and tranformations
alt
[if image transformed successfully]
Transformed Image
Transformed Image
alt
[if error occur during the process]
Return the 4xx/5xx response
Return the 4xx/5xx response
and ask user to re-upload the image
Client
API
Server

(edit)

Component Diagrams:


Package << Image processing system
Request
Response
UI Component
Image Processing API Component


Package << Image processing service API
Tranformations apply service
Get the list of transformations service
Get transformations request
Apply transformations POST request
Package << UI Component
Upload Image and transformations form
Transformed Image Form

Detail Design:

Code design : The whole backend application is built using Python flask and to apply transformation operations , I used the Pillow library. I chose Python, because it's much lighter than JAVA and it’s a preferred language for image processing and it’s faster and easier.

Architectural Style : I would be using client server architectural style as the client will be initiating the action and the server responds back. Client to client communication is not allowed.

Using RPC API over http : As we have the requirement of uploading the image with information of transformations and retrieving the transformed image as output. Hence with this info it seems to be action oriented where the client is initiating the action.

Design Pattern : I am using a factory design pattern. I am creating objects without exposing the creation logic to the client and refer to newly created objects using a common interface.

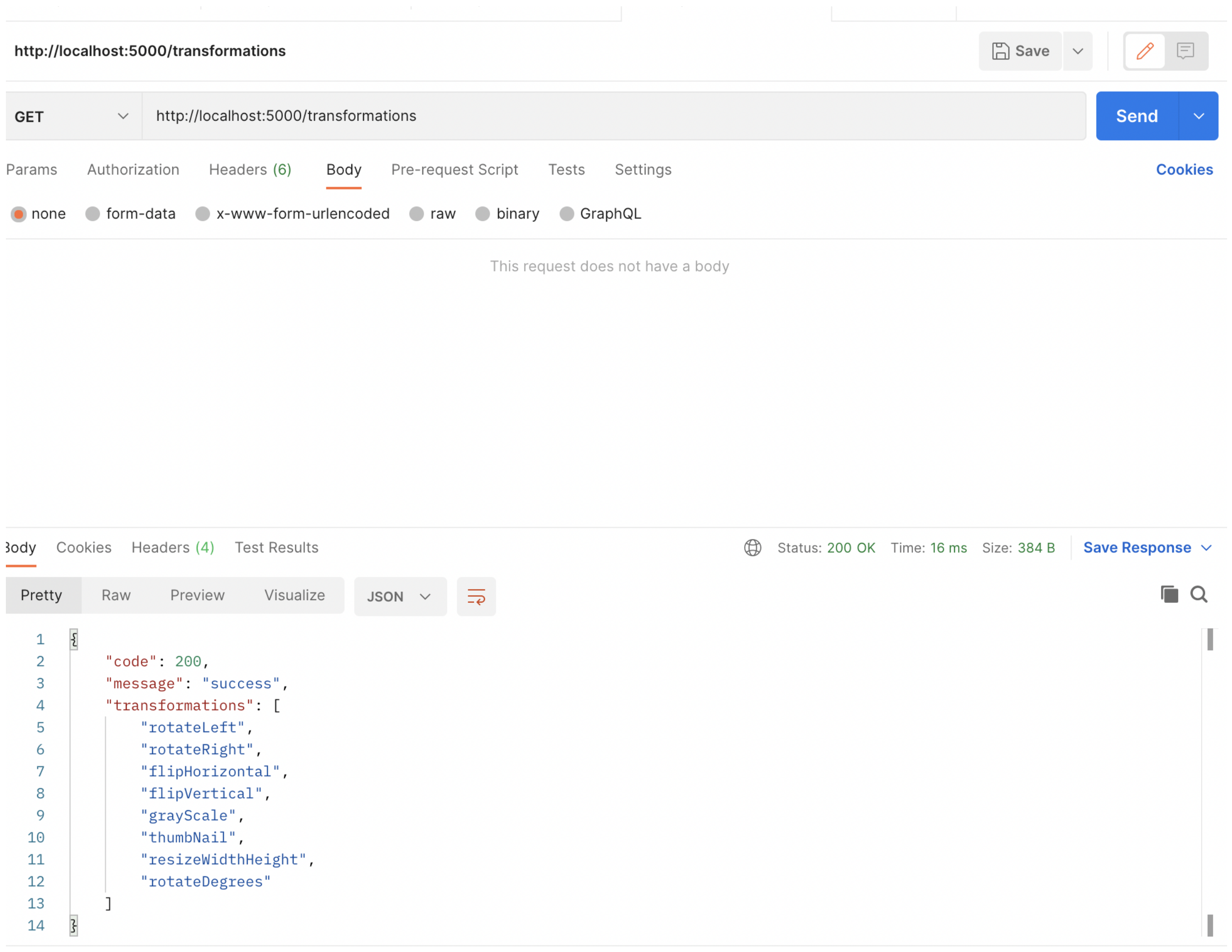
Little Language : This application takes image and transformations JSON as input and gives transformed image as the output. I would be supporting the list of transformations input in any arbitrary order. (Validation of input will be on the front end side).

Resources:

https://flask.palletsprojects.com/en/2.0.x/
https://pillow.readthedocs.io/en/stable/

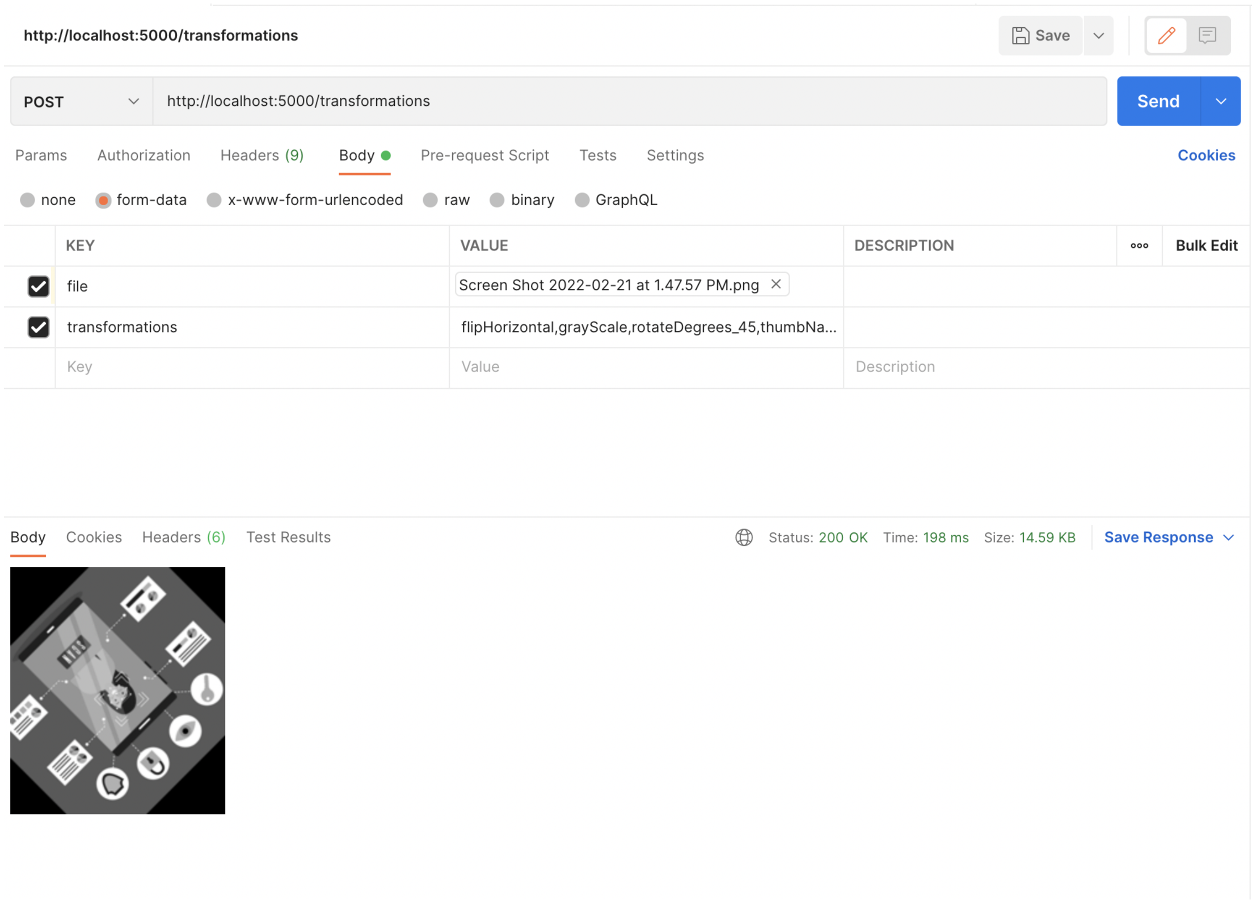
APIs:

GET/ transformations : Will give the list of available transformations



```json
{
    "code": 200,
    "message": "success",
    "transformations": [
        "rotateLeft",
        "rotateRight",
        "flipHorizontal",
        "flipVertical",
        "grayScale",
        "thumbNail",
        "resizeWidthHeight",
        "rotateDegrees"
    ]
}
```

POST/ transformations : uploads image and transformation operation or operations to be performed as input and output will be the transformed image.


http://localhost:5000/transformations

POST http://localhost:5000/transformations

Body: form-data

KEY	VALUE	DESCRIPTION
✓ file	Screen Shot 2022-02-21 at 1.47.57 PM.png	
✓ transformations	flipHorizontal,grayScale,rotateDegrees_45,thumbNa...	

Status: 200 OK Time: 198 ms Size: 14.59 KB

Swagger Definition:

openapi: 3.0.1
info:
  title: Image processor public API
  description: 'Sahana barki - Image processor public API'
  termsOfService: http://sahana.barki.imageprocessor.com/
  contact:
    email: sbarki@seattleu.edu
  license:
    name: Private
  version: 0.8.1
servers:
- **url:** http://sahana.barki.imageprocessor.com/
tags:
- **name:** Image transformation processor
paths:
  /v1/transformations:

```
  get:
   tags:
   - Image transformation processor
   summary: This will return the all available  transformation.
   responses:
     default:
      $ref: '#/components/responses/DefaultHeaders'
     200:
      description: OK
       content:
       application/json:
          schema:
           type: string
            example:
{"transformations":['rotateLeft','rotateRight','flipHorizontal','flipVertical','grayScale','thumbNail','re
sizeWidthHeight','rotateDegrees']}
     500:
      $ref: '#/components/responses/InternalServerError'
  post:
   tags:
   - Image transformation processor
   summary: This will apply transformation to the  input image.
   requestBody:
    content:
      multipart/form-data:
       schema:
        x-body-name: v1_transformations_body
        type: object
        properties:
         transformations:
          type: string
           description: list the of the transformations  with comma separator
           example:
"flip_horizontal,flip_vertical,rotate_plus_90,rotate_minus_270,gray_scale,resize_2046_1028,thumb
nail,rotate_left,rotate_right"
          fileName:
           type: string
           description: Binary data of the  image
           format: binary
   responses:
    default:
      $ref: '#/components/responses/DefaultHeaders'
    201:
     description: Return the binary data of the  image.
     content:
      multipart/form-data:
       schema:
        type: string
        format: binary
    400:
     description: Bad request
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/Error'
       examples:
        Validation Failed:
         value:
          message: 'The validation failed  - XXXX'
          code: '0f7b5003-345f-4267-9c29-08857634255b'
          description: 'The validation failed  for XXXX'
          details:
           -
            message: 'The transformations  list is in-valid or resize parameter is not the
boundary limit'
            field: 'The list value or  resize parameter'
            value: 'null'
    500:
     $ref: '#/components/responses/InternalServerError'
components:
 schemas:
  Error:
   type: object
   properties:
    message:
     type: string
     description: Message describing the current  error
     example: 'The validation failed - XXXX'
    code:
     type: string
     format: uuid
     description: Correlation uuid specific to  the request (same as X-Correlation-Id)
     example: "defe5527-49db-45fc-81e9-7b64b21cb5fe"
    description:
        type: string
        description: More details about the current  error
        example: 'The transformations JSON cant  be empty.'
      details:
        type: array
        items:
          properties:
            message:
              type: string
              description: More details about the  current error
              example: 'The transformations JSON  cant be empty.'
            field:
              type: string
              example: 'transformations JSON'
            value:
              type: string
              example: 'null'
    required:
      - message
      - code
headers:
  X-Correlation-Id:
    schema:
      type: string
      format: uuid
      example: 'ae031729f836e32bae0e4d7096e26c20'
    description: Correlation id of the response
responses:
  InternalServerError:
    description: Internal Server Error
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
        example:
          message: 'XXXX - Internal Server Error.'
          code: 'fd142bbb-c57c-4de7-879f-1f813c075164'
  DefaultHeaders:
    headers:
      X-Correlation-Id:
        $ref: '#/components/headers/X-Correlation-Id'
```

description: A unique identifier value that is attached to requests and messages that allow reference to a particular transaction or event chain.

Usage of "X-Correlation-Id" : A unique identifier value that is attached to requests and messages that allow reference to a particular transaction or event chain.

Deployment Procedure:

1. Download the zip file
2. Execute the requirements.txt file with command "pip3 install -r requirements.txt"
3. To run the application execute "python3 -m swagger_server"
4. Now the application is hosted.

Sample requests and responses in case of success scenarios :

1. API 1 : Get the list of transformations that are supported.

- **Input :** GET : http://localhost:5000/transformation(No input parameters)
- **Output :** {

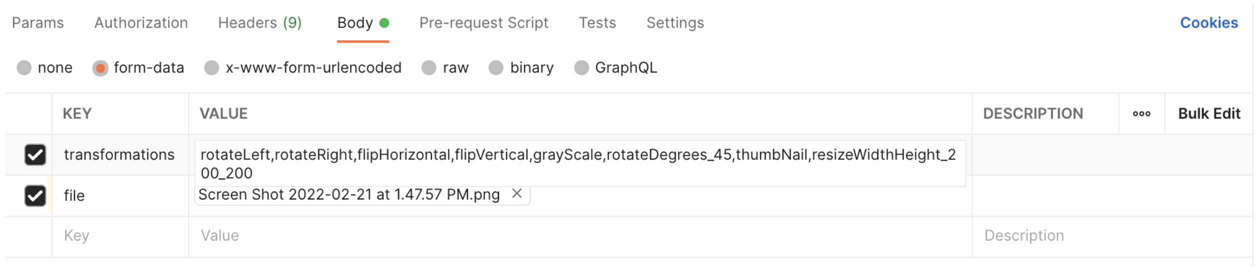
```
"code": 200,
"message": "success",
 "transformations": [
 "rotateLeft",
 "rotateRight",
  "flipHorizontal",
  "flipVertical",
  "grayScale",
  "thumbNail",
  "resizeWidthHeight",
  "rotateDegrees" ]}
```

This API will provide the supported transformations to list our front end application i.eReact JS. The user has to select the width and height size for resizeWidthHeight operation and degrees for rotateDegrees operation. Our frontend application forwards these parameters to POST/ transformations API.

2. **API 2 : Upload the image and transformation list in any arbitrary order.**

   Input : POST : http://localhost:5000/transformation

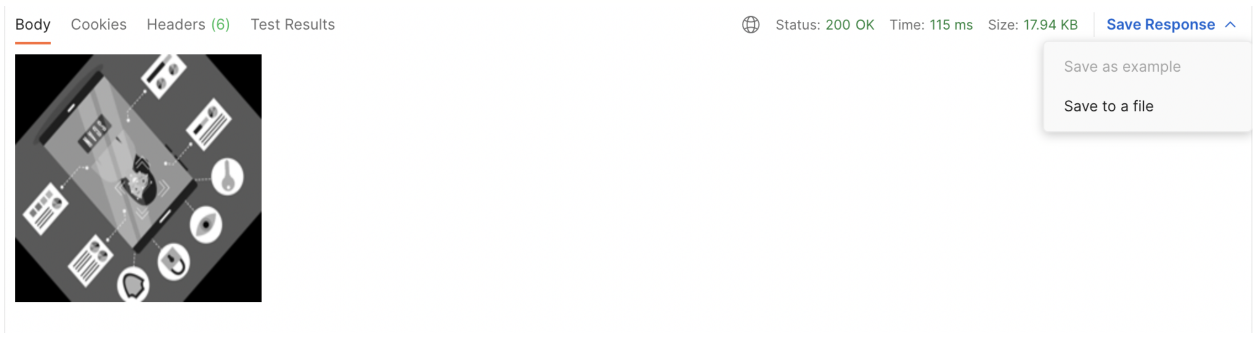
   Parameters : Will be sent as shown in image.



	KEY	VALUE	DESCRIPTION
✓	transformations	rotateLeft,rotateRight,flipHorizontal,flipVertical,grayScale,rotateDegrees_45,thumbNail,resizeWidthHeight_200_200	
✓	file	Screen Shot 2022-02-21 at 1.47.57 PM.png	
	Key	Value	Description

Output : Transformed image.

Sample Output :



Sample requests and responses in case of Error scenarios :

1. **When the image format is not supported -**

curl --location --request POST 'http://localhost:5000/transformations' \
--form 'file=@"/Users/sahanabarki/Desktop/a.txt"' \
--form 'transformations="randomOperation,rotateLeft,rotateRight"'

Response:

Error code : 400

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

2. When the transformation list is invalid -

```
curl --location --request POST 'http://localhost:5000/transformations' \
--form 'file=@"/Users/sahanabarki/Desktop/Screen Shot 2022-02-21 at 1.47.57 PM.png"' \
--form 'transformations="randomOperation,rotateLeft,rotateRight"'
```

Response:

Error code : 400

```
{
  "message": "The transformation list is invalid"
}
```

3. When the resize limit is not in the bound 3000x3000 and 10x10 -

```
curl --location --request POST 'http://localhost:5000/transformations' \
--form 'file=@"/Users/sahanabarki/Desktop/Screen Shot 2022-02-21 at 1.47.57 PM.png"' \
--form 'transformations="resizeWidthHeight_200_20000"'
```

Response:

Error code : 400

```
{
  "message": "The resizeWidthHeight value should  be in the range of 3000x3000 and 10x10"
}
```

4. When the resize limit values are not integers -

```
curl --location --request POST 'http://localhost:5000/transformations' \
--form 'file=@"/Users/sahanabarki/Desktop/Screen Shot 2022-02-21 at 1.47.57 PM.png"' \
--form 'transformations="resizeWidthHeight_20ab0_20aa"'
```

Response:

Error code : 400

```
{
  "message": "Please provide the integer values for  resizeWidthHeight"
}
```

5. **When the rotate degrees value is not a integer -**

```
curl --location --request POST 'http://localhost:5000/transformations' \
--form 'file=@"/Users/sahanabarki/Desktop/Screen Shot 2022-02-21 at 1.47.57 PM.png"' \
--form 'transformations="rotateDegrees_4a5b"'
```

Response:

Error code : 400

```
{
  "message": "Please provide the integer value for  rotateDegrees"
}
```