# **Image Service Generator Microservice**

## **Communication Contract**

### **Overview**

This microservice is designed to generate images based on the parameters provided by the user. It supports three key operations:

1. **Random Image Generation**
2. **Specific Image Generation**
3. **Adjusting Image Centering**

The microservice communicates through a **REST API**.

### **Communication Contract**

The microservice uses HTTP GET requests to perform the following operations:

* GET /generate\_random\_image: Returns a random image.
* GET /generate\_specific\_image: Returns a specific image based on the parameters provided (e.g., album name, image format, resolution).
* GET /adjust\_image\_center: Returns an image with the specified centering adjustments.

## **Requesting Data from the Microservice**

### **General Instructions:**

To request data from the image service, you can use HTTP GET requests with the necessary query parameters. The microservice will respond with the requested image in binary format (e.g., PNG or JPEG).

### **Example Call for Requesting Data**

You can use the Python requests library to make a call to the microservice. Below is an example of how to request a **random image**:

import requests  
  
# Define the base URL of the microservice  
BASE\_URL = "<http://127.0.0.1:5000>"  
  
# Request a random image  
response = requests.get(f"{BASE\_URL}/generate\_random\_image", params={  
 "format": "PNG", # Specify the format (e.g., PNG, JPEG)  
 "resolution": "200", # Specify resolution (width x height)  
 "position\_x": "10", # Horizontal axis for centering (in pixels or percentage)  
 "position\_y": "20" # Vertical axis for centering (in pixels or percentage)  
})  
  
# Save the response image to a file  
with open("random\_image.png", "wb") as f:  
 f.write(response.content)  
  
print("Random image saved as 'random\_image.png'")

### **Request Parameters:**

* **action**: Specifies whether the image generation will be "random" or "specific".
* **format**: Specifies the format of the image (e.g., "PNG", "JPEG").
* **resolution**: Specifies the resolution of the image (e.g., "1024x768").
* **position\_x**: Specifies the horizontal position of the image in the frame.
* **position\_y**: Specifies the vertical position of the image in the frame.

### **Example Call for Specific Image Generation:**

To request a **specific image** (e.g., album cover for "Sample Album"):

response = requests.get(f"{BASE\_URL}/generate\_specific\_image", params={  
 "album": "Sample Album", # Specify the album or other specific data  
 "format": "PNG", # Specify the format  
 "resolution": "1024x768", # Specify the resolution  
 "position\_x": "50", # Horizontal position  
 "position\_y": "50" # Vertical position  
})  
  
# Save the response image to a file  
with open("specific\_image.png", "wb") as f:  
 f.write(response.content)  
  
print("Specific image saved as 'specific\_image.png'")

## **Receiving Data from the Microservice**

### **General Instructions:**

Once a request is made to the microservice, it will return the image data as binary content. You can handle this content and save it to a file for display or use in your application.

### **Example Call for Receiving Data**

The response.content will contain the binary data of the image. You can write this content to a file as shown in the above examples, or use it in your application for further processing.

### **Example Response Data:**

The microservice will return an image in binary format. If your request was successful, you will receive image data in the response. If the request fails, the response will include an error message with the appropriate status code.For example, the following Python code receives the image data and writes it to a file:

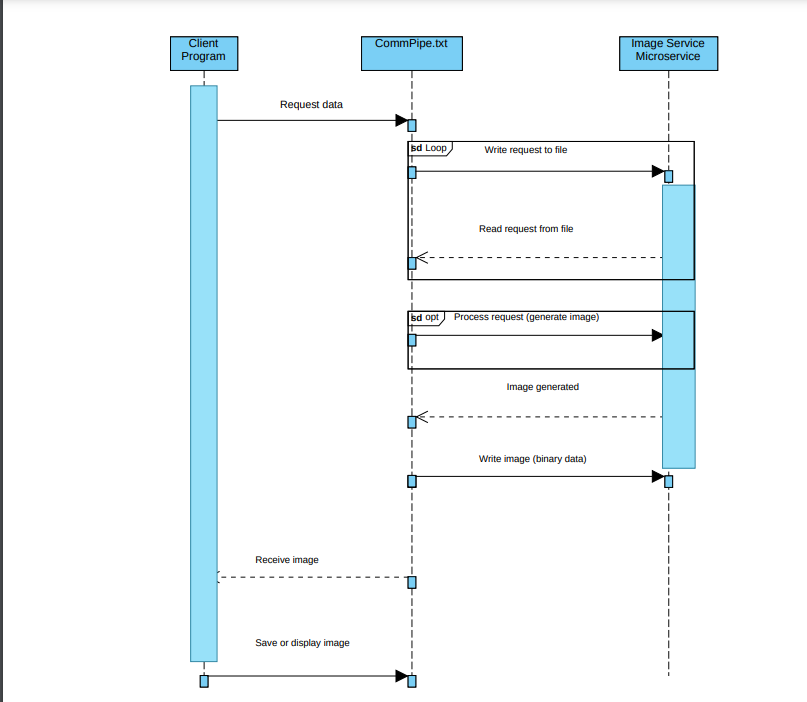
# Assuming `response` is the result of the GET request to the image service  
if response.status\_code == 200:  
 with open("received\_image.png", "wb") as f:  
 f.write(response.content)  
 print("Image received successfully and saved.")  
else:  
 print(f"Error: Unable to fetch image. Status Code: {response.status\_code}")

### **Expected Response:**

* **Status Code**: 200 (OK) if the request is successful.
* **Content**: Binary data of the image in the requested format (e.g., PNG or JPEG).

## **UML Sequence Diagram**

Below is a detailed **UML sequence diagram** illustrating how the data is requested and received from the microservice.



### **Sequence Description:**

**1. Client Program Initiates Request**

The sequence begins with the Client Program initiating an HTTP GET request to generate a random image.

The request includes parameters like image format, resolution, and centering positions (position\_x and position\_y).

**2. Write Request to CommPipe.txt**

Upon receiving the request, the Client Program writes the request details to the CommPipe.txt file.

This file acts as a communication medium between the Client Program and the Image Service Microservice.

**3. Read Request from CommPipe.txt**

The Image Service Microservice reads the request details from the CommPipe.txt file.

It processes the request to understand the required image generation specifications.

**4. Process Request (Generate Image)**

The Image Service Microservice processes the request parameters and generates the image accordingly.

Depending on the request type, it may generate a random image, a specific image based on provided data (e.g., album name), or adjust the image centering.

**5. Image Generation**

Once the processing is complete, the Image Service Microservice generates the image.

The generated image is temporarily stored to facilitate the next step of communication.

**6. Write Image to CommPipe.txt**

The Image Service Microservice writes the generated image data (in binary format) back to the CommPipe.txt file.

This step ensures that the image data is available for retrieval by the Client Program.

**7. Receive Image by Client Program**

The Client Program reads the image data from the CommPipe.txt file.

This marks the completion of the data retrieval process.

**8. Save or Display Image**

The Client Program saves the received image data to a file for display or further use.

For example, the image may be saved as random\_image.png or specific\_image.png, depending on the request type.

## **Communication Pipe**

The microservice uses a **REST API** to communicate, ensuring that other programs can make requests via HTTP to interact with it. All responses from the microservice are returned as binary image data.

**4. Microservice A Mitigation Plan**

**A. For which teammate did you implement “Microservice A”?**

I implemented Microservice A for my teammate Nathan Shadnik.

**B. What is the current status of the microservice? Hopefully, it’s done!**

The microservice is complete and fully functional.

**C. If the microservice isn’t done, which parts aren’t done and when will they be done?**

This is not applicable, as the microservice has been fully implemented.

**D. How is your teammate going to access your microservice? Should they get your code from GitHub (if so, provide a link to your public or private repo)? Should they run your code locally? Is your microservice hosted somewhere? Etc.**

Nathan will access the microservice by downloading the code from my GitHub repository. He should run the microservice code locally on his machine. Here is the link to the GitHub repository where he can access it: [GitHub Repository Link].

**E. If your teammate cannot access/call YOUR microservice, what should they do? Can you be available to help them? What’s your availability?**

If Nathan is unable to access or call the microservice, he should contact me for support. I am available on Discord and email, and I’ll respond within 24 hours. My primary availability is on weekdays from 9 AM to 5 PM (PST), but I can also be available on weekends if needed.

**F. If your teammate cannot access/call your microservice, by when do they need to tell you?**

Nathan should notify me of any issues by [specific date, e.g., November 14] to allow time for troubleshooting and resolution before the final project deadline.

**G. Is there anything else your teammate needs to know? Anything you’re worried about? Any assumptions you’re making? Any other mitigations/backup plans you want to mention or discuss with your teammate?**

No additional requirements are needed at this time. I am assuming that Nathan has installed all dependencies listed in the README file and is familiar with using HTTP requests in Python. If he needs further guidance on setup or usage, I am available to assist.