Project 2

Seth Williams

SethWilliams2020@fau.edu

GCP Project Name: COT5930-Project-1

GCP Project ID: cot5930-project-1

GCP Project Console: https://console.cloud.google.com/home/dashboard?project=cot5930-project-1

Cloud Run Service Name: flask-image-upload

App URL: flask-image-upload-64129822162.us-east1.run.app

Github Repository Link: SethW411/COT5930-Project-2

Introduction

This project's primary purpose is to teach us how to utilize ChatGPT and other API's in our cloud run applications. This is done by updating our previous assignment to create and store api generated descriptions of the files uploaded by users.

Architecture

Flask Web Application: This project was built using a Flask web application. It allows users to upload and view images. Flask handles the web requests by routing user actions (like uploading a file) to the correct function in the back end.

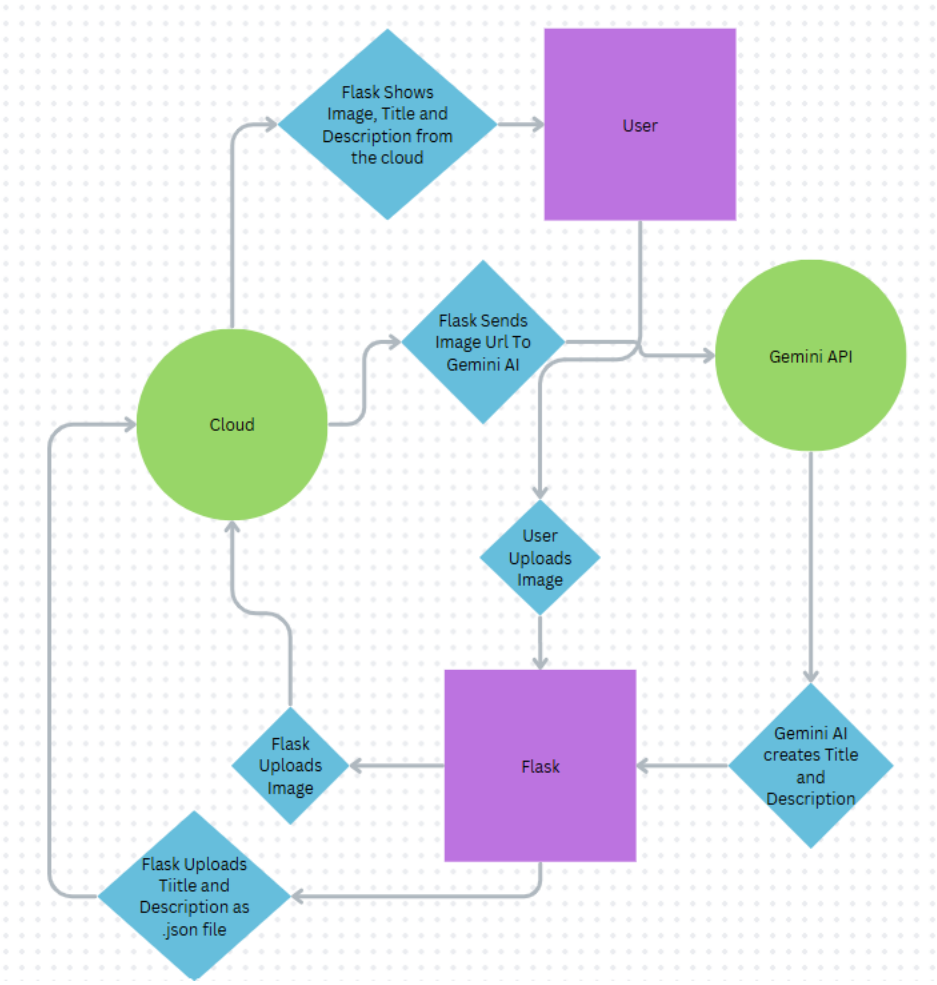
Python: The Python code powers the logic behind the application. It processes user requests received by Flask, handles image uploads, and interacts with Google Cloud Storage. It also retrieves stored images and updates the webpage dynamically to display them.

Google Cloud Run: This allows the application to be accessed online, making it serverless and eliminating the need for manual server management. Cloud Run automatically scales the app based on traffic.

Google Cloud Storage: This is where the uploaded images are stored and managed. The application saves files to a storage bucket, making them accessible through public URLs so they can be displayed on the website.

Gemini API: This is where uploaded images are sent to after they are uploaded to the cloud. Gemini API uses these images to generate a short title and short description of the image.

Arch Diagram



Implementation:

Correct Security Issues: Before getting started on the new objectives, I needed to solve the issues of the past objectives. Project 1 suffered from security issue of leaking the image urls to the user. This issue has been corrected by privatizing the urls, and only providing temporary access to those on the webpage.

API Implementation: The next objective was to ensure the API would be installed properly. This was done by enabling the API on google cloud, obtaining the key, and finally recording the key on an environment variable to prevent the user from having access to it.

API Utilization: Now I needed to utilize the Gemini API I now have access to. To ensure every is working properly, the api was used to generate content independently. Afterwards, the code was then updated to record this information onto a .json file and upload it to the google cloud storage. Finally, the html for the webpage would read from that image's correlated .json file and showcase it.

Pros and Cons

Pros:

- API's make it much easier to save space on massive programs and allows programmers to access them remotely.
- They typically have an easy or simple request format and are often designed to be used in a variety of locations.

Cons:

- Sometimes you are limited in the number of requests you can make at a given time.
- Not every API is free to use, with some requiring a fee for their services.

Problems encountered and Solutions

1. **Authentication Failure**: Despite facing this issue in project 1, it came back again. For whatever reason, I could not figure out why, my cloud is unable to identify my google cloud credentials. I tried setting a service account (did not work), I tried uploading the .json file directly onto the cloud (to meet a larger error), and I tried putting console notes around my code to determine where exactly this was happening. I don't know why it was happening, all I do know is that whenever my cloud run tried to access its own cloud, it would return with an error. This error did not occur when I ran my code locally or on a virtual machine, and ran perfectly.
2. **Images display not working at all:** This error occurred when trying to implement the signed urls for security reasons. I struggled to have my code reference the new urls when it came to image generation. I eventually concluded that the issue was in the format of my html. I had the image, description, and title on one ordered list item. Title and description were already giving me issues, so I separated each of them into individual list items, and that solved the image issue.

Application instructions

Step 1: Upload an image from your computer by clicking the "Choose File" button

Step 2: Press Submit

Step 3: Look at your image uploaded under the "Uploaded Images" section and its AI Generated title and description. It will take a minute to upload, but once it is uploaded, it will not take a long time to look at it again.

Lessons Learned

API Usage I did not know much about API's before this assignment. Now I hope to use multiple API's on future programs or applications, assuming I can afford it.

Debugging Debugging is a nightmare. I found the best way to code is to move one step at a time, and constantly checking for issues with the console log. The console log is likely my best tool for determining the cause of errors.

Appendix

Main.py

```
import os

# allows interaction with the operating system.

# it allows me to handle files, environmental variables (like API keys), and other system commands.


from flask import Flask, request, render_template, redirect, send_file

# Flask-       is a web framework that allows me to build web applications in Python.

# Request-     allows me to handle incoming user data, such as file uploads.

# Render-      template lets me load and display html files

# Redirect-    lets me send the user to a different page


from google.cloud import storage

# Loads the google cloud storage module allows my app to upload, retrieve and list images from a storage bucket.


import datetime

# allows use to temporarily provide access to download the files


import requests

# allows making HTTP requests, used for fetching the image from a URL


from google import genai

from PIL import Image

import io

import json
```

```
storage_client = storage.Client()

BUCKET_NAME = 'flask-image-storage'

# Cloud Storage Configuration


# 1. Flask Hello World

app = Flask(__name__)

#  Initilizes the app


@app.route('/hello')

#  Defines a URL path that users can visit

#  the first function directly below it is the function that will run when the user accesses the route "/hello"


def hello_world():

  """Return a basic hello world response."""

  return "Hello, World!"

#  Defines a function that will read "Hello, World!"


@app.route('/')

def index():

#  Defines a URL path for the home pagedef index():

  """Display the upload form and list uploaded images from Cloud Storage."""


  image_urls = get_blobs_urls()  # gets all the image URLs from Cloud Storage and stores them in the image_urls variable


  index_html = """

  <form method="post" enctype="multipart/form-data" action="/upload">

    <div>
```

```
    <label for="file">Choose file to upload</label>
    <input type="file" id="file" name="form_file" accept="image/jpeg"/>
  </div>
  <div>
    <button>Submit</button>
  </div>
</form>
<h2>Uploaded Images</h2>
<ul>
"""
# Upload form HTML
# leaves an open <ul> tag to create a list of uploaded images in the next step

new_image_list = "<ul>"  # Start a new list

# Fetch all the blobs (images) from the storage bucket
bucket = storage_client.bucket(BUCKET_NAME)
blobs = bucket.list_blobs()

# Loop through each blob (image)
for blob in blobs:
  # Only process image files (e.g., .jpg, .jpeg, .png)
  if blob.name.endswith(('.jpg', '.jpeg', '.png')):
    # Get signed URL for the image
    signed_url = get_signed_url(blob)

    # Get the corresponding JSON filename (e.g., image-name -> image-name-json.json)
    json_filename = blob.name.rsplit('.', 1)[0] + '-json.json'
```

```
# Try to fetch the corresponding JSON file
try:
    json_blob = bucket.blob(json_filename)
    json_data = json_blob.download_as_string()  # Download JSON as a string
    json_info = json.loads(json_data)  # Convert JSON string to a dictionary

    title = json_info.get("title", "No title found")
    description = json_info.get("description", "No description found")

except Exception as e:
    # If the JSON file is not found or any error occurs, log and set default values
    print(f"Error retrieving JSON for {blob.name}: {e}")
    title = "No title"
    description = "No description"

# Add image to the list using the signed URL
new_image_list += f'''
<li>
    <img src="{signed_url}" alt="Uploaded Image" width="200">
</li>
'''

# Add title to a new list item
new_image_list += f'''
<li>
    <strong>Title: </strong>{title}
</li>
'''
```

```
        # Add description to a new list item
        new_image_list += f'''
        <li>
          <strong>Description: </strong>{description}
        </li>
        '''

        # Add download button in its own list item
        new_image_list += f'''
        <li>
          <form action="/download" method="GET">
            <input type="hidden" name="file_url" value="{signed_url}">
            <button type="submit">Download</button>
          </form>
        </li>
        '''

    new_image_list += "</ul>"  # Close the new list

    # Combine everything together
    index_html += new_image_list

    return index_html



@app.route("/download", methods=["GET"])
def download():
    file_url = request.args.get("file_url")  # retrieves image URL (already a signed URL)
```

```
    # Directly redirect the user to the signed URL
    return redirect(file_url)  # Redirect to the signed URL directly


@app.route("/upload", methods=["POST"])
# 8. Handle image upload
def upload():
    """Handles the image upload."""
    if "form_file" not in request.files:
        return "No file uploaded", 400
    # Checks if form_file exists in the request.files dictionary.

    file = request.files["form_file"]
    # Gets the file from request.files and assigns it to file

    if file.filename == "":
        return "No selected file", 400
    # if file is unnamed, then that means the user did not select a file before clicking submit

    # Upload the image to Google Cloud Storage and get the signed URL
    upload_url = upload_to_gcs(BUCKET_NAME, file)

    # Now call save_info to create the .json file for the image metadata (title and description)
    blob = storage_client.bucket(BUCKET_NAME).blob(file.filename)
    save_info(blob)

    return redirect("/")
    # Redirect to the home page. Updates home page.
```

```
@app.route('/files')
def list_files():
    """Lists uploaded image files from Google Cloud Storage."""
    return get_blobs_urls()
# 9 & 10. List uploaded image files from Cloud Storage


def upload_to_gcs(bucket_name, file):
    """Upload file to Google Cloud Storage."""
    bucket = storage_client.bucket(bucket_name)
    # Connects "bucket" to the bucket_name in the function syntax (connects from me to the storage)
    blob = bucket.blob(file.filename)
    # Creates a reference file to the uploaded file selected in the function syntax


    file.seek(0)
    # Move the file stream pointer to the beginning to prevent reading the end of the file, when it is
accessed again


    blob.upload_from_file(file)
    # Upload file


    signed_url = get_signed_url(blob)


    return signed_url  # Return the public URL of the uploaded file


def get_blobs_urls():
    """Fetch signed URLs of all uploaded images from Google Cloud Storage."""
    print("Getting image URLs...")
    bucket = storage_client.bucket(BUCKET_NAME)
```

```
    blobs = bucket.list_blobs()

    # Generate signed URL for each image
    image_urls = []
    for blob in blobs:
        signed_url = get_signed_url(blob)  # Pass the blob object directly
        image_urls.append(signed_url)

    return image_urls
import os

def get_signed_url(blob, expiration_minutes=2):
    filename = blob.name.split('/')[-1]
    print(f"Extracted filename for signed URL: {filename}")  # Log the filename for debugging

    # Log the environment variable to check if the credentials are set properly
    credentials_path = os.getenv('GOOGLE_APPLICATION_CREDENTIALS')
    print(f"GOOGLE_APPLICATION_CREDENTIALS is set to: {credentials_path}")

    if not credentials_path:
        raise Exception("GOOGLE_APPLICATION_CREDENTIALS environment variable is not set!")  # Error if credentials not found

    # Now, generate the signed URL
    url = blob.generate_signed_url(
        version="v4",
        expiration=datetime.timedelta(minutes=expiration_minutes),
        method="GET",
    )
```

```
    return url

api_key = os.getenv("GOOGLE_API_KEY")

def generate_title_description(blob):
    """Generates title and description for the given image using Gemini API."""

    filename = blob.name  # Use the 'name' attribute to get the filename of the blob
    print(f"--- Generating title and description for image: {filename} ---")

    if not api_key:
        print("API key is missing!")
        return "Error", "Error"

    # Initialize the Gemini client with the API key
    client = genai.Client(api_key=api_key)

    # Generate signed URL for the blob
    signed_url = get_signed_url(blob)
    print(f"Generated signed URL: {signed_url}")  # Console log to verify the URL

    # Fetch the image using the signed URL
    response = requests.get(signed_url)

    if response.status_code == 200:
        print("Successfully fetched the image.")

        # Open the image from the response content (use in-memory bytes)
        image = Image.open(io.BytesIO(response.content))
```

```
        # Call the Gemini API for title and description generation
        title_response = client.models.generate_content(
            model="gemini-2.0-flash",
            contents=[image, "Generate a single, short title for this image."])

        description_response = client.models.generate_content(
            model="gemini-2.0-flash",
            contents=[image, "Generate a short, one-sentence description of this image."]
        )

        title = title_response.text  # Capture the generated title
        description = description_response.text  # In this case, title and description are the same

        print(f"Generated title: {title}")
        print(f"Generated description: {description}")
        return title, description

    else:
        print(f"Failed to fetch the image. Status code: {response.status_code}")  # Log error if failed
        return "Error fetching title", "Error fetching description"

def save_info(blob):
    """Generates title and description, creates a JSON file, and uploads it to GCS."""

    # Get the title and description using the generate_title_description function
    title, description = generate_title_description(blob)

    # Define the name of the JSON file based on the blob's name, appending '-json' to make it distinct
```

```
    json_filename = blob.name.rsplit('.', 1)[0] + '-json.json'

    # Create the info to be saved as JSON
    info = {
        "title": title,
        "description": description
    }

    # Convert the dictionary to a JSON string
    json_data = json.dumps(info)

    # Upload the JSON string as a .json file to Cloud Storage
    bucket = storage_client.bucket(BUCKET_NAME)
    json_blob = bucket.blob(json_filename)
    json_blob.upload_from_string(json_data, content_type='application/json')

    print(f"Info saved as {json_filename} in Cloud Storage.")

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
# 11. Run the Flask app
```

Requirements.txt

```
blinker==1.9.0
cachetools==5.5.1
certifi==2024.12.14
charset-normalizer==3.4.1
click==8.1.8
colorama==0.4.6
Flask
```

google-api-core==2.24.1

google-auth==2.38.0

google-cloud-core==2.4.1

google-cloud-storage==2.19.0

google-crc32c==1.6.0

google-resumable-media==2.7.2

googleapis-common-protos==1.66.0

idna==3.10

itsdangerous==2.2.0

Jinja2==3.1.5

MarkupSafe==3.0.2

proto-plus==1.26.0

protobuf==5.29.3

pyasn1==0.6.1

pyasn1_modules==0.4.1

requests==2.32.3

rsa==4.9

urllib3==2.3.0

Werkzeug==3.1.3

google-genai==0.3.0