Cloud Computing Memory Lane Course Project

Team Members:

Naga Prem Sai Nellure – nnellure2022@fau.edu - **Z23691340**

Keerthana Sandanamaina - ksandanamain2023@fau.edu - **Z23750269**

Sreelekha Potturi - spotturi2023@fau.edu - **Z23745026**

Sai Nitish Kumar Korrapati - skorrapati2023@fau.edu - **Z23709565**

Venkatesh Uppala - vuppala2023@fau.edu - **Z23745027**

Sreedhar Reddy Munagala - munagalas2023@fau.edu - **Z23750119**

Project Repository & Cloud Deployment Specifications:

Repo link: https://github.com/Sreedhar-dev-ux/Cloud-Project

Production Environment: https://project-cicd-vox2grkzha-wl.a.run.app

GCP Project	Upload Image Manager
Cloud Run Service	project-cicd
Cloud Build Trigger	project-cicd-us-cloud
Python Framework Used	FLASK

Google Console : https://console.cloud.google.com/run?hl=en&project=upload-image-manager

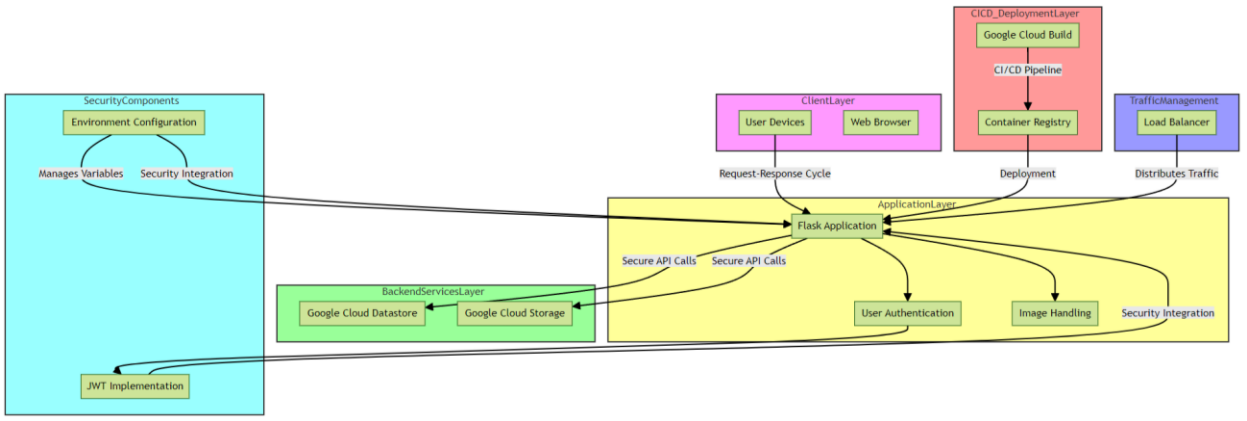
Abstract:

This project showcases the development of "Memory Lane," a robust, cloud-based photo storage and management application. The project was build by using Flask framework and it is deployed on the Google Cloud Platform and the application excels on providing a secure and scalable and user-friendly environment for the storage of the photo. It works on Authorizing the power of Datastore and Cloud Storage of Google cloud, it will handle huge amounts of image data and metadata effectively. The application implements JWT for secure user authentication, ensuring safe user sessions and data privacy. A key feature is its image handling capability, allowing users to upload, view, download, and delete images. It supports various image formats and manages large data volumes with ease. The architecture of the application adheres to modern cloud application standards, using microservices and containers for scalability and portability. The implementation of CI/CD pipelines via Google Cloud Build demonstrates the application's commitment to continuous integration and delivery, ensuring constant updates and seamless deployment. The application stands as a testament to effective cloud-native development

FAU

practices, addressing the core needs of data security, user experience, and scalability in the realm of cloud-based applications.

Architecture Diagram:



Client Layer:

- **User Devices**: These are the entry points for the users, from where they interact with the application. This includes laptops, desktops, smartphones, etc., running a web browser.
- **Web Browser**: The medium through which users send HTTP requests to the Flask application. It renders the user interface and processes user inputs and interactions.

Application Layer:

- **Flask Application**: This is the core of the web service. It serves as the backend server that processes requests, manages sessions, authenticates users, handles image uploads, and serves stored images. It's built using Flask, a lightweight and flexible Python web framework.

User Authentication Component:

- **JWT Implementation**: For secure authentication we have used JSON Web Tokens (JWT). When the user logged in, a token is generated and then it is used to perform validate following requests, ensure that user was authenticated and authorized to perform actions in the portal.

Image Handling Component:

- This part of the Flask application deals with the image upload process. It verifies the file type and size before storing the image and ensures that each image has a unique filename for retrieval.

Security Components:

- **Environment Configuration**: Utilizes **python-decouple** to manage environment variables, which helps in securely configuring application settings without hardcoding sensitive information like database credentials or secret keys.

Backend Services Layer:

- **Google Cloud Datastore**: A NoSQL database used to store user credentials and image metadata securely. The metadata includes information like the user's email, filename, display name, image type, image size, and the URL pointing to the image's location in storage.
- **Google Cloud Storage**: An object storage service used for storing the actual image files uploaded by users. It allows for high availability and global retrieval of the images.

CI/CD Deployment Layer:

- **Google Cloud Build**: This is a service that compiles source code, runs tests, and produces software packages that are ready to deploy. It's integrated into the CI/CD pipeline, automating the building and testing of the application upon code commits.
- **Container Registry**: A single place for your team to manage Docker images, perform vulnerability analysis, and decide who can access what with fine-grained access control.

Client Interaction and User Experience: The entry point for any web service is the client layer, where users interact with the application through various devices. In this project, the Flask application serves as the server, handling HTTP requests and responses. The **render_template** function from Flask is pivotal in serving HTML pages, creating a dynamic and responsive user interface.

For instance, upon a successful image upload, the user is immediately notified via Flask's **flash** messages, enhancing the interactive experience. To ensure the service remains user-friendly even when things go wrong, custom error handlers for 404 and 500 HTTP status codes were implemented, providing clear feedback and guidance for users navigating the application.

Secure User Authentication: Security is paramount, especially when handling user authentication. By integrating JWT, the application ensures secure management of user sessions. JWT tokens are generated upon login, which are then used to verify the authenticity of subsequent requests.

```python
from flask import Flask, jsonify, request
from werkzeug.security import safe_str_cmp
import jwt

app = Flask(__name__)
JWT_SECRET_KEY = 'your_secret_key'

@app.route('/login', methods=['POST'])
def login():
    username = request.json.get('username', None)
    password = request.json.get('password', None)


    if authenticate_user(username, password):
        token = jwt.encode({'username': username}, JWT_SECRET_KEY, algorithm='HS256')
        return jsonify({'token': token}), 200
    else:
        return jsonify({'message': 'Invalid credentials'}), 401
```

Backend Storage and Metadata Management: The Google Cloud Platform (GCP) offers robust solutions for backend storage. Google Cloud Datastore is a NoSQL database used for storing user credentials and image metadata securely. This metadata includes the user's email, the unique filename, and other relevant data like the image type and size. Google Cloud Storage is employed for the actual storage of images.

```python
from google.cloud import storage, firestore
from flask import current_app as app

storage_client = storage.Client.from_service_account_json('path_to_service_account.json')
firestore_client = firestore.Client.from_service_account_json('path_to_service_account.json')

def upload_file_to_gcs(file, filename):
    bucket = storage_client.get_bucket(app.config['CLOUD_STORAGE_BUCKET'])
    blob = bucket.blob(filename)
    blob.upload_from_string(file.read(), content_type=file.content_type)
    return blob.public_url

def add_file_metadata_to_firestore(filename, location, size):
    files_collection = firestore_client.collection('files')
    file_metadata = {'filename': filename, 'location': location, 'size': size}
    files_collection.add(file_metadata)
```

Configuration Management: To maintain the integrity and security of the application, sensitive data such as the JWT secret key and GCP bucket names are managed using **python-decouple**. This library abstracts configuration parameters out of the code, allowing for easier management of environment variables and preventing sensitive information from being hardcoded.

Integration of Cloud Services: The integration of GCP services within the Flask application framework is a seamless process. Google Cloud's libraries for Python are designed to work smoothly with the Flask framework. This allows for the secure upload of images to Cloud Storage and the addition of metadata to Datastore without disrupting the application flow.

Image Handling and User-Controlled Operations: The Flask application offers users control over their data, allowing them to upload, browse, and delete images. Secure_filename is used to sanitize the file names, ensuring that the uploaded files do not contain harmful paths. Here's how the upload function is structured in our app:

```python
@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        location = upload_file_to_gcs(file, filename)
        add_file_metadata_to_firestore(filename, location, file.size)
        flash('Image uploaded successfully!')
    return redirect('/')
```

Continuous Integration and Deployment: A CI/CD pipeline is established using Google Cloud Build, which listens for GitHub commits and automatically initiates the build process. This ensures that every change to the codebase is automatically tested and deployed, maintaining a continuous flow of integration and delivery. The Container Registry stores the Docker images resulting from these builds, ready for deployment.

Security Aspects of the Setup: At the forefront of the security setup is the use of JSON Web Tokens (JWT). JWT offers a compact, URL-safe means of representing claims to be transferred between two parties. In our application, JWTs ensure that if once a user is authenticated, their subsequent interactions will be securely managed without the need to constantly revalidate or recheck with their credentials. This token-based system will reduce risks, as Cross-Site Request Forgery (CSRF) by ensures each request from a client will be accompanied by a valid and signed token, tying each and every request specifically only to authenticated users.

Environment variables play a critical role in managing sensitive configuration settings. By using **python-decouple**, the application abstracts secrets and other configuration data away from the codebase, mitigating the risk of accidental exposure of secrets in source code repositories. This practice is further enhanced by the use of Google Cloud's secure storage for service account credentials, which are used to authenticate and interact with Google Cloud services programmatically.

Information Collection and Storage: User information and image data constitute the primary categories of information collected by the application. User information is typically limited to what is necessary for authentication and personalization purposes, such as email addresses and securely hashed passwords.

Image data encompasses the files themselves, stored in Google Cloud Storage, and their associated metadata, in which include the attributes such as the filename, display name, content type, and size. Metadata is stored in Google Cloud Datastore, which allows for efficient retrieval and management of the images from user and ensuring that the access to the image files can be controlled and reviewed.

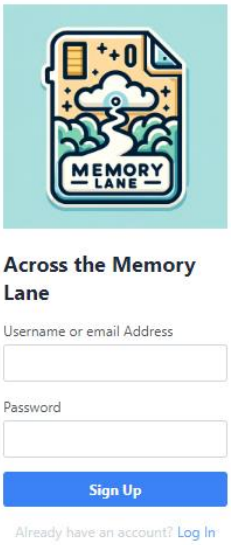
Database Schema: The application employs a NoSQL approach for data management, using Google Cloud Datastore. The schema-less nature of NoSQL databases provides the flexibility to evolve the data model as the application grows. However, in practice, the application enforces a consistent structure for the data it manages.

Users are represented in the Datastore with entities that include attributes such as **email** (String, indexed) and **password** (String, hashed). Images are also represented as entities, with attributes including **user_email** (String, indexed to associate images with users), **filename** (String, unique), **display_name** (String, for user-friendly identification), **image_type** (String, describing the content type), **image_size** (Integer, storing the file size), and **url** (String, storing the secure URL to the image in Cloud Storage).

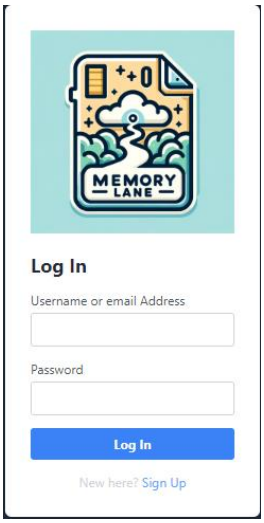
Object Organization in Storage: Google Cloud Storage organizes files in a flat namespace, called a bucket. Each image uploaded by a user is stored in a bucket with a unique filename derived from a combination of user-provided and system-generated data to avoid collisions. This approach simplifies object management and facilitates direct access via URLs. However, direct access is secured through the use of signed URLs, which provide time-limited, read-only access to the storage objects, thus ensuring that users can only access their images through the application's controlled mechanisms.

Access to both Datastore and Cloud Storage is managed through the application's service account, which has the minimum required permissions to perform necessary operations. This follows the principle of least privilege, reducing the risk surface and ensuring that the application can only access resources essential for its operation.In conclusion, the application's security setup is a testament to a design that prioritizes user privacy, data integrity, and secure access. By integrating JWT for authentication, employing environment variables for configuration management, and utilizing Google Cloud's secure infrastructure, the application stands as a robust and secure full-stack web service.

User Registration/Login:



- Users can create a new account on the "Sign Up" page by entering a username or email address and a password. The application ensures the confidentiality of these credentials, storing them securely in the backend.



- For existing users, the "Log In" page requires the username or email and password to access the user's personal space within the application.

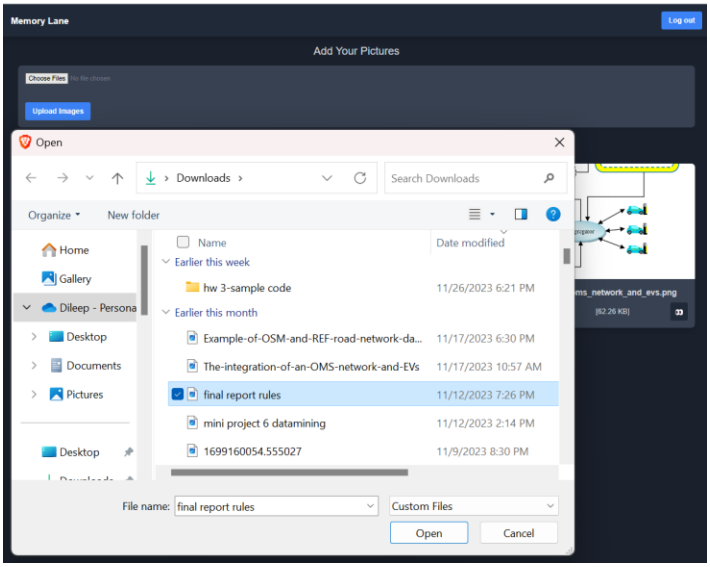


Image Upload:

- After the log in, user will be taken to the main interface of the application where user can upload the images. In application we can view the Choose Files button which will allow users to select picture files from their PC or their device.

- The image formats which are supported to upload are jpg, png, jpeg and gif which should be less than 4mb which should be verified by server for maintain security and consistency.
- If once a file is chosen then users can upload the image to their gallery by just clicking the "Upload Images" button which is founded on application. Then the uploading process include transferring the file to secure cloud storage and then recording its metadata in a database.

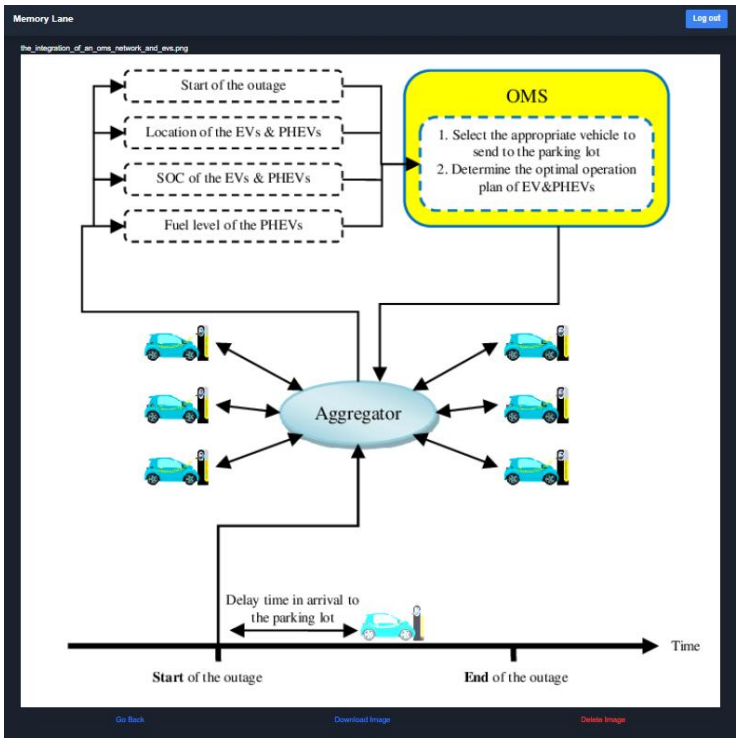


Image Deletion:

Suppose if the user wants to delete an image from the gallery, now the user need to select the eyes icon which opens the total image then at the interface user can find Delete image option now user can do by clicking the Delete Image. This action will permanently delete the image file from cloud storage and the corresponding metadata from the database will also be deleted, ensuring that the user's data is managed with according of their preferences.

Image Download:

- The user's gallery displays all uploaded images with options to download or delete them.
- If we want to download an image, now users can click on the Download image below the image where can we can download the image . Then the application will provide the image file, firmly extracted from the cloud storage,and then it will be saved on the user PC or their device.

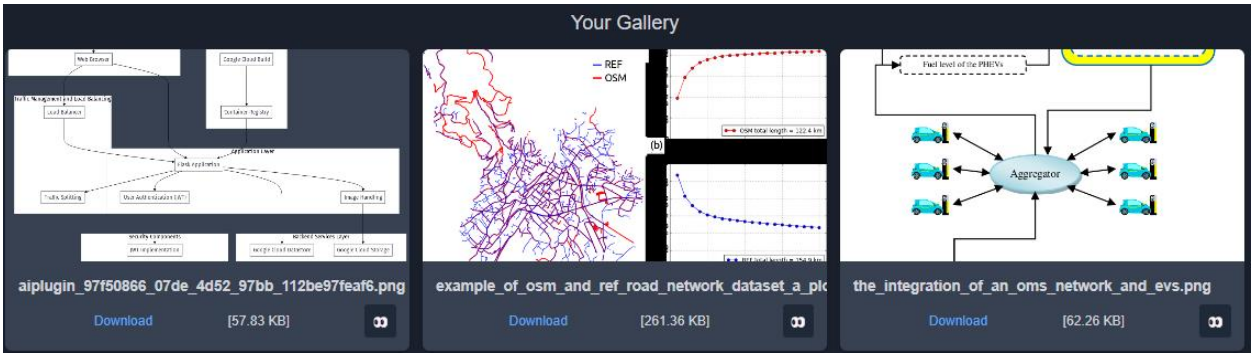


Image Metadata Display and Visualization:

- At gallery, each image is backed by a set of metadata which provides information detailly about the file. This metadata includes several factors of the images which are filename, date when image was uploaded, size of the file, and format of the image. This information helps users to organize and recognize their images effectively.
- In every image, we can see an eye symbol or eyes icon, when we click the icon it will trigger the visualization of the image feature. This function in application allow the users to view the image in full within the browser without needing to download the image for viewing full image. It is generally useful for a quick review or when the user need to find the specific image in instead browsing according to the Total collection of the images.
- When an image is visualized, users can often see additional metadata, potentially including the resolution of the image and the date when photo was captured and when the device was used to capture image. At this level image detail enriches the user experience, providing context and background for the memories captured in the photographs.

Conclusion:

The **Memory Lane** project successfully culminates as a testament to the efficient use of cloud technologies and modern software engineering practices. The application, through its Flask-based architecture and integration with Google Cloud services, offers a highly scalable and secure environment for photo storage and management. Its ability to handle large datasets, thanks to the robust infrastructure of Google Cloud Datastore and Storage, is particularly noteworthy. The application not only provides fundamental features such as image upload, download, and deletion but also enhances user experience with features like image metadata display and secure access through JWT authentication. The project highlights the importance of CI/CD in maintaining a consistent and error-free deployment process, a crucial aspect of modern software development. The use of microservices and containers further speaks to the application's readiness for scalable cloud deployment. Overall, "Memory Lane" stands as a comprehensive solution in the cloud application landscape, effectively balancing functionality, security, and user experience. This project serves as a valuable model for cloud-native application development, demonstrating the potential and efficiency of combining Flask, Google Cloud Platform, and best practices in software development.