Group 9 - Course Project

Cloud
Team 9

Caleb Panoch - cpanoch2018@fau.edu
Parul Chauhan - pchauhan2022@fau.edu
Juan Quintero - juanquintero2022@fau.edu
Diego Gama - dgama2021@fau.edu
Kristian Stavri - kstavri2022@fau.edu
Muskan Joshi - joshim2022@fau.edu

Project Name: Group9Fall23

Repository URL: https://github.com/calebpanoch/cloud-team-9/tree/final-project-caleb

Project ID: burnished-block-397422

App URL: https://cloud-team-9-final-project-n4raeziswa-uc.a.run.app/

Introduction

For our final project, we transitioned from Continuous Integration/Continuous Deployment (CI/CD) in Project 3 to adding features such as login functionality to ensure users can only access their images. We implemented a user deletion button, app automatic scaling, and enhanced security across all layers with which the users interact. Additionally, we refined image metadata extraction for cloud storage and its visibility within the app.

We described the components of this app and illustrated the app architecture through a diagram. We also provided detailed instructions for app usage, coding, and the security configurations we established. Regarding the database, we discussed its schema and the organization of objects in storage.

To conclude, each team member presented their reflections on the term, detailing their learning experiences, challenges faced, and provided the appropriate URLs to the cloud configurations, code repository, and web application.

How to use the application:

New Users:

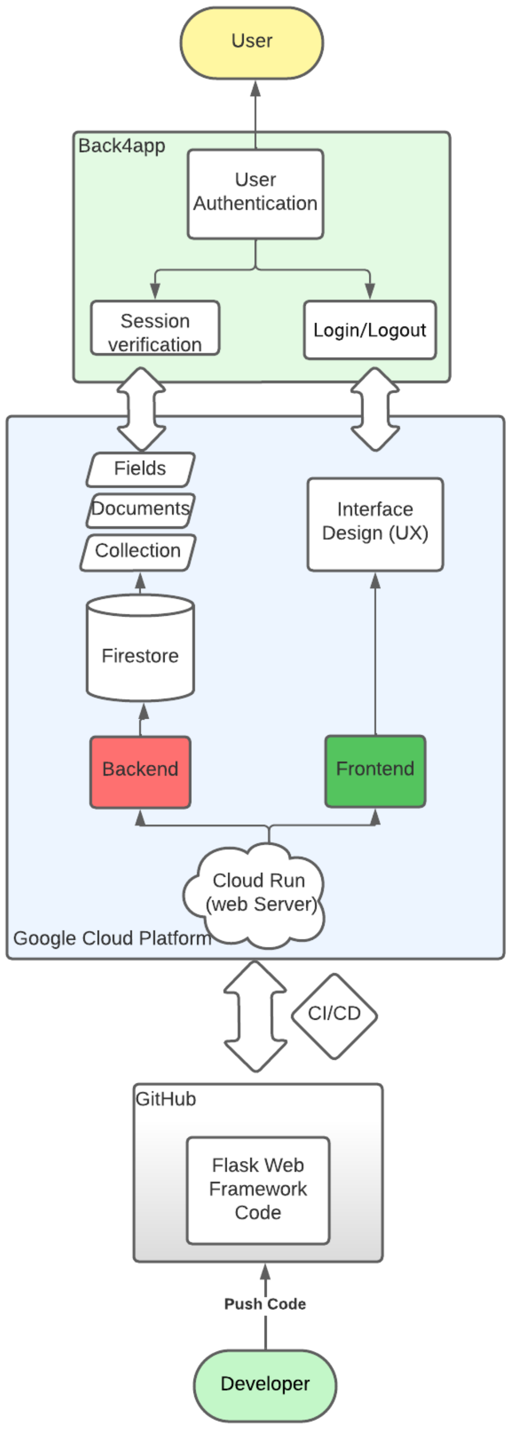
New users should enter a username and password in the fields on the landing page and then click the login button. A pop-up message will appear stating, "Failed Login. Would you like to create an account with this username?" If 'Ok' is clicked, the user will be created with a unique key. All keys contain the phrase: "5up3r_un1qu3_key--user--USERNAME," with "USERNAME" being replaced by the input from the initial step. The user will then be redirected to the home page.

Users with existing login:

Enter the *username* and *passwor*d, then click 'Ok.' You will be directed to the landing page, where you can *choose* a picture from your documents and click the *submit button*. If pictures have been previously uploaded, the file name will appear alongside a delete option. To view a picture, click on the file name; the picture will be displayed, and its metadata will appear at the bottom.

On the second screen, among other options, you can click the *download button* to save the picture and the *back button* to return to the landing page.

Architecture



Application Components

Application Components Descriptions

User: The person using the application who interacts with the front end to upload images, view images and their metadata, and delete images.

Frontend: The user interface of the application where interactions such as logging in, key creation, uploading images, and viewing images take place.

Key Creation: To save/load/delete images, the application still uses buckets and metadata, except the key is now 5up3r_un1qu3_key--user--USERNAME where USERNAME is replaced with the actual user's username.

Backend: The server-side logic of the application that processes requests, handles authentication, manages sessions, and communicates with the firestore.

Firestore: The database where image metadata and files are stored. It is organized into collections, documents, and fields that allow for efficient data retrieval and storage.

Cloud Run (Web Server): A managed compute platform that automatically scales the backend based on incoming request volume and handles request execution. Cloud Run is containerized in Docker.

User Authentication: For client verification, the application uses the back4app API. It creates the user with JSON and verifies the user in python with the API. Every time a page is loaded, the session key in the URL is verified by the python code. When the user logs out, the session key is deleted in back4app.

CI/CD: Continuous Integration and Continuous Deployment pipeline that automatically tests and deploys code changes to the application, ensuring that new features and fixes are efficiently rolled out from the code repository to Google Cloud.

GitHub: The code repository where the Flask web framework code is stored, version-controlled, and reviewed before being deployed through the CI/CD pipeline.

Developer: The individual or team responsible for writing, updating, and maintaining the codebase of the application.

Back4App: This is an external Backend-as-a-Service (BaaS) platform used for user authentication and session management, interfacing with both the frontend for login/logout and the backend for session verification.

Work/Coding/Configurations

In this project, we developed a Flask-based web application designed to manage images for different users. This application is containerized and deployed using Google's Cloud Run service. The key components of the project include:

Flask Web Framework: Utilized Flask to build the server-side logic of the application, handling routes for user authentication, file upload, viewing, and deletion.

Containerization: The application is containerized for deployment using Docker. This process involved creating a Dockerfile that specifies the environment, dependencies, and how the Flask app is executed.

Deployment on Cloud Run: The containerized application was deployed on Google Cloud's Cloud Run, a serverless platform that scales automatically and handles infrastructure management.

User Authentication: Implemented a user authentication system in collaboration with Back4App's API, ensuring that only authenticated users can access their files.

File Management: We developed functionalities to upload, view, and delete image files. This includes interfacing with buckets for storing and retrieving images and Firestore for storing the metadata of the images. File metadata is collected using the Pillow Python library.

Front-End Development: Created a simple and functional front-end using HTML and CSS, allowing for user interactions with the web application.

Security Description

We incorporated several security measures to ensure the application's integrity and user data protection. These include:

Secure User Authentication: Our web app backend uses Back4App's API for user authentication, with API keys to ensure secure communication.

Secure Session Management: Sessions are securely managed, allowing file additions, viewings, and manipulations only with successful user authentication. Each user also has their own secure and private storage for their files that others cannot view. We also have prevented the user from seeing the URL of the images they view by encoding the images first then sending them to users to view.

Input Validation and Sanitization: Special character checks in filenames are in place to prevent path traversal.

HTTPS Protocol: The app is deployed on Cloud Run, so it benefits from HTTPS, encrypting data transmission between clients and the server for maximum security.

Information Collected and Stored

Our application collects and stores specific types of information for its functionality:

User Data: This includes minimal user information for authentication, like usernames and session tokens.

File Metadata: Upon file uploads, metadata from images (filename, blob size, key, and EXIF data) are extracted using the pillow library and then stored in Firestore for later retrieval whenever images are viewed.

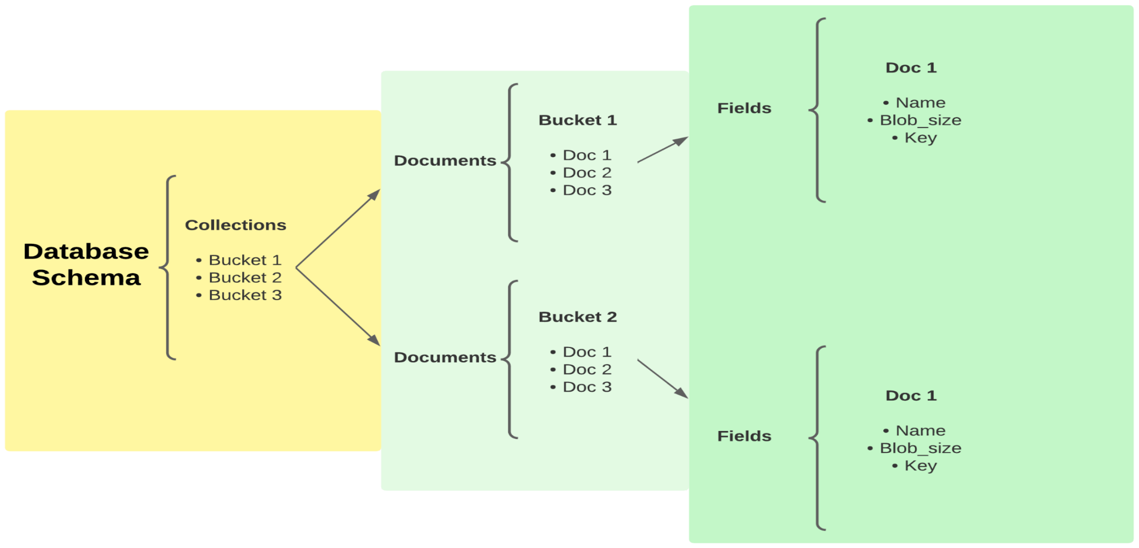
Image Files: Users' uploaded images are stored in a bucket in Google Cloud Firestore storage, and can only be accessed by the user that uploaded them (and the app administrators of course).

Database schema

The database schema is NoSQL-based since it is a non-relational type; this instance is from a Firestore database. The database is organized into Collections, Documents, and Fields.

- **Collections:** These are top-level containers that store documents. Collections contain documents and do not directly contain raw fields with values. Each collection can house a unique set of documents. In our case, collections are determined by the buckets.
- **Documents:** They serve as individual records within a collection, each identified by a unique ID. In our web app, documents correspond to images.
- **Fields:** Data within each document is organized into fields, which are key-value pairs. The key is the field's name, and the value can be any data type supported by Firestore. For our app, some fields represent image metadata, such as name, key, and blob size and also Exifdata. It also depends on the type of image and its features.

This overview provides a clear understanding of the Firestore database schema as it pertains to our web app:



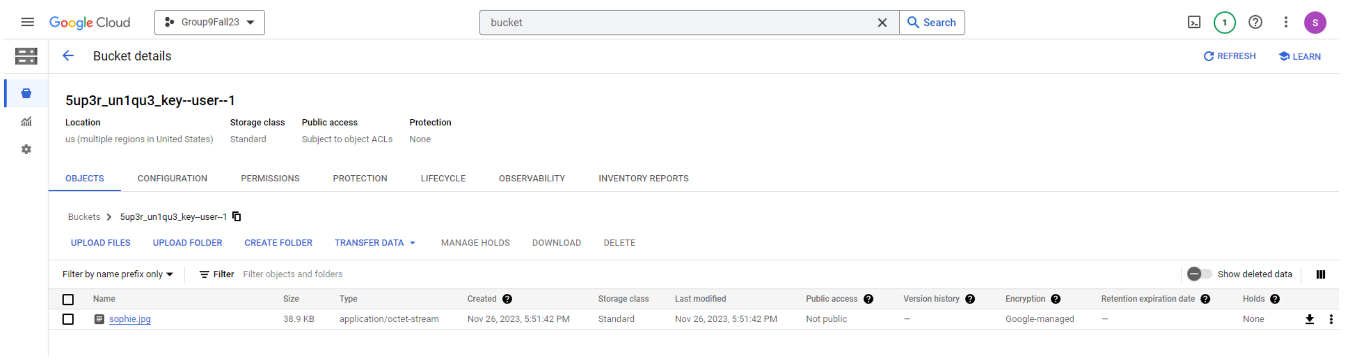
Database schema overview

Object organization in storage

The object organization in storage for our web app is the way that it is physically organized by the database. CGP automatically provides certain file hierarchies and naming conventions. The way our web app data was organized in the storage system of GCP is as follows:

- Name
- Size
- Type
- Creation Date
- Storage Class
- Last Modification
- Public Access
- Version History
- Encryption
- Retention Expiration Date
- Holds

Here is an example extracted from our GCP project:



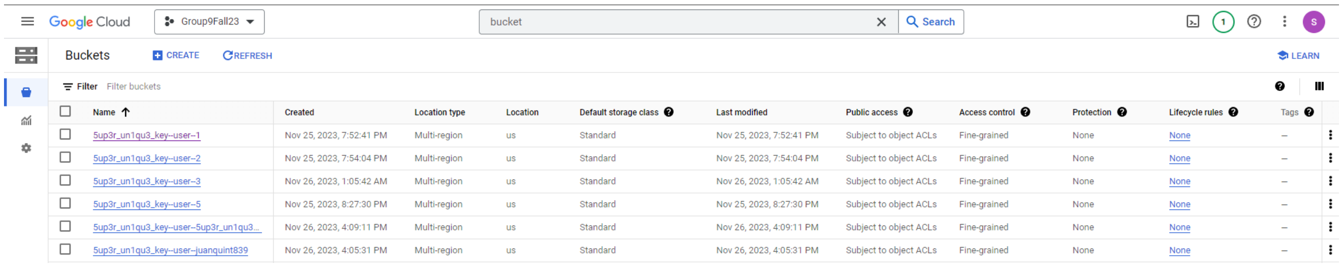
5up3r_un1qu3_key--user--1

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Standard	Subject to object ACLs	None

Buckets > 5up3r_un1qu3_key--user--1

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Retention expiration date	Holds
sophie.jpg	38.9 KB	application/octet-stream	Nov 26, 2023, 5:51:42 PM	Standard	Nov 26, 2023, 5:51:42 PM	Not public	–	Google-managed	–	None

Bucket by the name 5up3r_un1qu3_key–user-1, all the metadata is extracted from the file and new data is created like the “creation date” field. The object storage is organized according to the predefined hierarchy and convention names.



Name	Created	Location type	Location	Default storage class	Last modified	Public access	Access control	Protection	Lifecycle rules	Tags
5up3r_un1qu3_key--user--1	Nov 25, 2023, 7:52:41 PM	Multi-region	us	Standard	Nov 25, 2023, 7:52:41 PM	Subject to object ACLs	Fine-grained	None	None	–
5up3r_un1qu3_key--user--2	Nov 25, 2023, 7:54:04 PM	Multi-region	us	Standard	Nov 25, 2023, 7:54:04 PM	Subject to object ACLs	Fine-grained	None	None	–
5up3r_un1qu3_key--user--3	Nov 26, 2023, 1:05:42 AM	Multi-region	us	Standard	Nov 26, 2023, 1:05:42 AM	Subject to object ACLs	Fine-grained	None	None	–
5up3r_un1qu3_key--user--5	Nov 25, 2023, 8:27:30 PM	Multi-region	us	Standard	Nov 25, 2023, 8:27:30 PM	Subject to object ACLs	Fine-grained	None	None	–
5up3r_un1qu3_key--user--5up3r_un1qu3...	Nov 26, 2023, 4:09:11 PM	Multi-region	us	Standard	Nov 26, 2023, 4:09:11 PM	Subject to object ACLs	Fine-grained	None	None	–
5up3r_un1qu3_key--user--juanquint839	Nov 26, 2023, 4:05:31 PM	Multi-region	us	Standard	Nov 26, 2023, 4:05:31 PM	Subject to object ACLs	Fine-grained	None	None	–

Image showing the object organization storage for the buckets.

It is important to make a distinction between database schema and object organization in storage. Database schema is about the structure and organization of data within the database for query and transaction purposes, object organization in storage is about how files and data blobs are stored, retrieved, and managed in a cloud storage service.

Individual Insights

Caleb: In this project, I learned how to use back4app's API to verify that a user is still logged in on the server side. To do this, I passed the session key whenever the user loads a page, and checked to see if the session key was valid. If it was not valid, then the user would be redirected to the home or sign-in/up screen. I also learned how to give the user an image without giving the URL address. To do this, I encoded the image and then sent it to the user to display.

Diego: In this last phase of the project, I gained expertise in implementing authentication within a web application. While exploring various methods, our team decided to employ Back4App, an API that validates user authentication on the server side, proving to be the most efficient choice. From previous experience, I have used Back4App, and it has proved to be really useful for authentication methods, instead of Google Authenticator and such. Since we encountered some issues with other type of account authentications, we opted for the Back4App solution. This project presented both challenges and rewards, increasing my enthusiasm to further enhance my understanding of the Google Cloud management.

Kristian: This overall project taught me a lot that I wanted to know about how to develop on the cloud. It was fun to get to work with NoSQL databases and learn how to integrate these buckets with a web app. This final part of the project was my favorite part, as partitioning it per user and securing the data to each user really tied it all together and made it feel like a real app. I learned many things, but some notable highlights are that I learned what containers are, how to use them, and how to deploy them for others to use as a web app. Learning how to set up CI/CD on a previous part of this project was very useful and I will likely take that forward into my career, as it is very useful to be able to deploy changes in that way. I also never really learned much about front end development, so setting up a UX was a great learning experience.

Juan: In this part of the project, I learned how to implement authentication in a web application. There are several methods available, but we chose to use an API called Back4App, which verifies user authentication on the server side and proved to be the most effective. We also considered using Google Auth to retrieve login information from Gmail accounts; however, I encountered issues with permissions in Google Cloud. Despite the Google login appearing correctly, there was a disconnection between the login process and the web app. Consequently, we opted for the Back4App approach. This project has been both challenging and rewarding, and I am highly motivated to continue expanding my knowledge of the GCP platform for future projects.

Muskan: Starting this project, I dived into figuring out how to make sure users are who they say they are, using back4app's tools. I deepened my understanding of user authentication through back4app's API, emphasizing secure session key handling and implementing features like the user deletion button for data control. The project broadened my expertise in security, privacy, and collaborative development, addressing ethical considerations. In the cloud development phase, I integrated NoSQL databases with the web app, particularly enjoying partitioning data per user and ensuring security. Overall, the project significantly advanced my skills in authentication, security, privacy, and collaborative software development practices.

Parul: Diving into the project was like unraveling a tech mystery. Authenticated user journeys became a focal point, and opting for Back4App's API felt like finding the missing piece. The dance between Flask, Docker, and Google Cloud broadened my tech toolkit, transitioning from a CI/CD rhythm to crafting a feature-rich symphony.Firestore collections, documents, and fields became the building blocks, granting every user a bespoke backstage key.HTTPS adoption and proactive threat monitoring fortified the project as a digital guardian. The technical journey, marked by hurdles and triumphs, sculpted a nuanced understanding of cloud architectures for me. It highlighted the art of seamlessly blending technical finesse with user-centric functionality—a delicate fusion of robust technology and empathetic design principles, which I wish to carry forward in future projects.