

CSCI 5410 - Serverless Data Processing

Lab Activity 1 Report

Md Samshad Rahman

B00968344

a. Aim:

The aim of the task is to learn how to create multiple AWS Lambda functions and make them work together by integrating DynamoDB and an S3 bucket. To achieve this, I created a basic user registration and login system that allows you to upload and retrieve picture.

Application Functionality:

1. User Registration and Login:

- Users can register with a username and password.
- User credentials are stored securely (MD5 Hash) in DynamoDB. [1]
- Users can log in by verifying their credentials against the stored data in DynamoDB.

2. Profile Picture Upload:

- Authenticated users can upload their profile picture.
- The uploaded image is converted to Base64 format and stored in an S3 bucket.

3. Profile Picture and Image Size Fetching:

- Users can fetch their profile picture, which is retrieved from S3, encoded in Base64, and sent back as a response. This retrieved image is the compressed image.

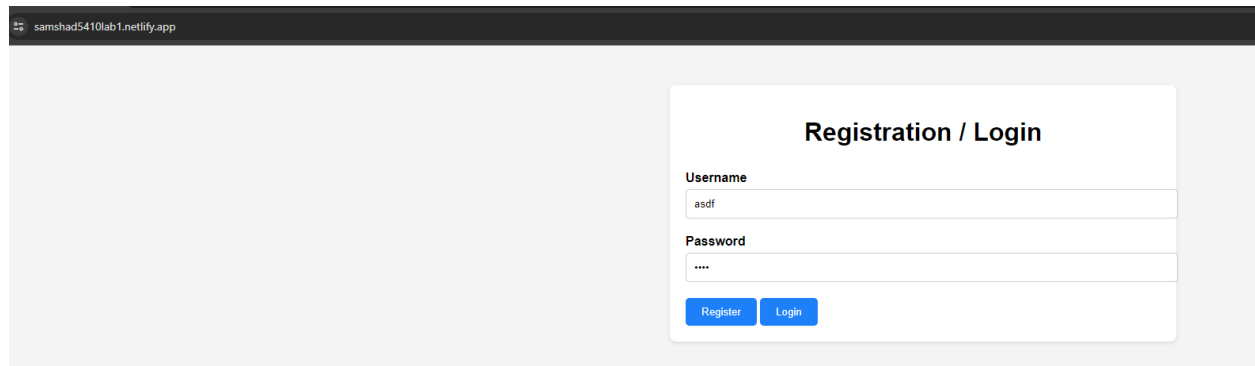
4. Profile Picture Resizing and Compression:

- Images uploaded to a specific S3 bucket are automatically resized and compressed. [2]
- The resized images are stored in another S3 bucket.
- Image size before and after resizing is recorded in DynamoDB.

b. Thought Process and Design Considerations:

1. I have broken down the application into smaller, modular Lambda functions, each responsible for a specific task (e.g., registration, login, image upload, image processing, fetching image).
2. I have used one DynamoDB table to store username and password. Another DynamoDB table stores uploaded image size before compression and after compression to compare the difference.
3. While storing uploaded image to a S3 bucket I could have used compression on runtime and save only the compressed image. Instead I have stored the main uploaded image and used another lambda function on trigger event to do the compression and storing the compressed image into another S3 bucket. By doing this I wanted to learn working with trigger event and as well I want to improve my compression algorithm slowly.
4. While connecting with frontend I faced CORS header issue. I had to add header on my return JSON. [3]
5. Another challenge I am facing in image compression is that not always the algorithm works properly. I have seen some time compressed images having more size than original one. As a result, I am still trying to optimize the algorithm.

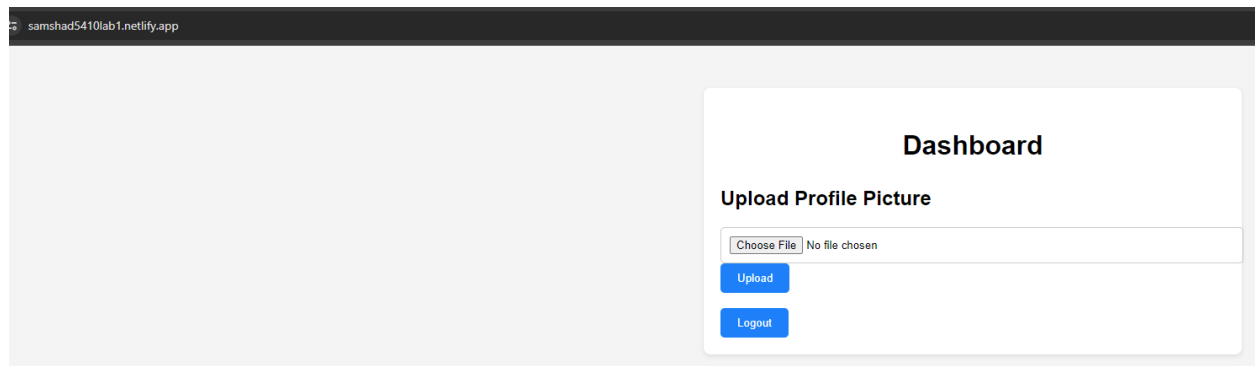
- c. Deployed URL: <https://samshad5410lab1.netlify.app/>
d. Some screenshots given below:



The screenshot shows a web browser window with the address bar displaying "samshad5410lab1.netlify.app". The main content area features a white card titled "Registration / Login". Inside the card, there are two input fields: "Username" with the text "asdf" and "Password" with four dots. Below these fields are two blue buttons labeled "Register" and "Login".

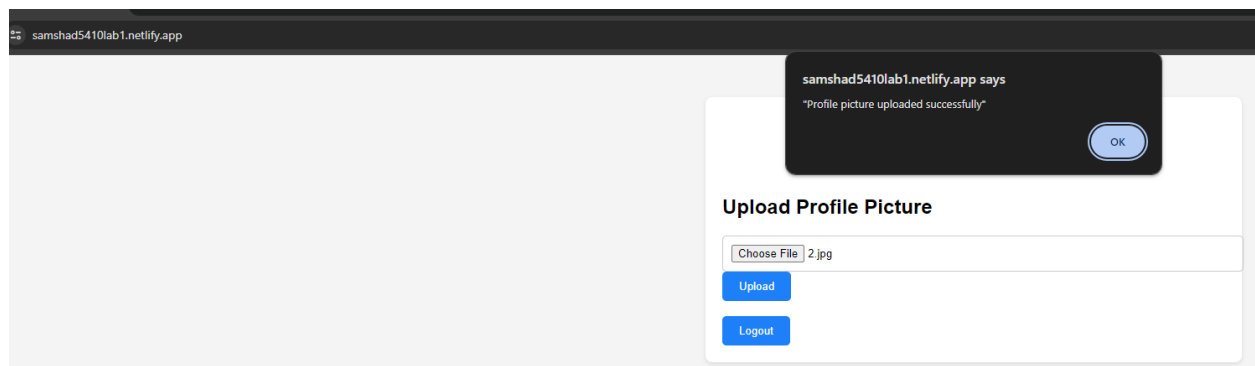
Figure 1 Register user

After successful registration it redirects to Dashboard and user can upload a picture.



The screenshot shows the "Dashboard" page. It has a white card with the title "Dashboard". Below the title is the section "Upload Profile Picture". This section contains a file selection area with a "Choose File" button and the text "No file chosen". Below this are two blue buttons: "Upload" and "Logout".

Figure 2 Dashboard



This screenshot shows the same "Dashboard" page as Figure 2, but with a dark notification box at the top. The notification box contains the text "samshad5410lab1.netlify.app says" and "Profile picture uploaded successfully", with an "OK" button. Below the notification, the "Upload Profile Picture" section now shows "2.jpg" next to the "Choose File" button. The "Upload" and "Logout" buttons remain visible.

Figure 3 Picture uploaded successfully

After successfully uploading picture it takes time for backend to compress the image and send back again to frontend. As I am using vanilla JS I was having issue with await and promise to auto reload the delay. As a result, user has to logout and log in again after uploading picture to see the image and image size data.

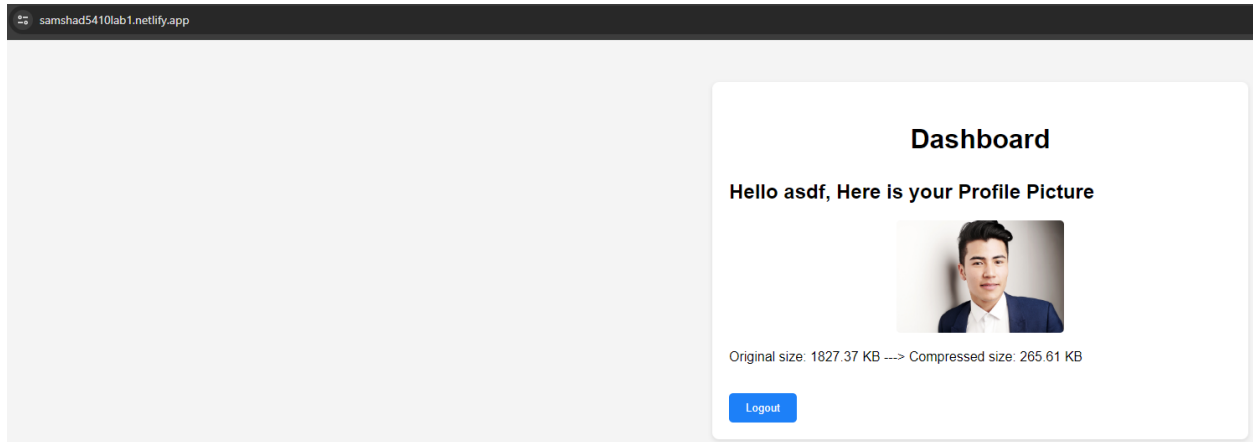


Figure 4 Compressed image with size comparison

e. References:

- [1] "MD5 hash in Python", *geeksforgeeks.org*. [Online]. Available: <https://www.geeksforgeeks.org/md5-hash-python/> [Accessed: June 6, 2024]
- [2] "Pillow in Python", *pillow.readthedocs.io*. [Online]. Available: <https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.thumbnail> [Accessed: June 6, 2024]
- [3] "CORS, API Gateway and Python Lambda", *medium.com*. [Online]. Available: <https://medium.com/my-adventures-with-aws-serverless-development/cors-api-gateway-and-python-lambda-7c53ef7ce06b> [Accessed: June 6, 2024]