# You’re My Celebrity

Team 5 - White Paper

## What is the application about?

We wanted to do something fun with AWS services, so we thought of creating a ‘celebrity look alike’ application. The user uploads their image (where their face is visible, not a validation but added this assumption in the user guide) and AWS services bring that to AWS Rekognition. This service looks at the facial features of the user and the 160+ celebrities that we supply to figure out which celebrity the user resembles the most and then returns that celebrity’s image.

## Overview and Goals for the project

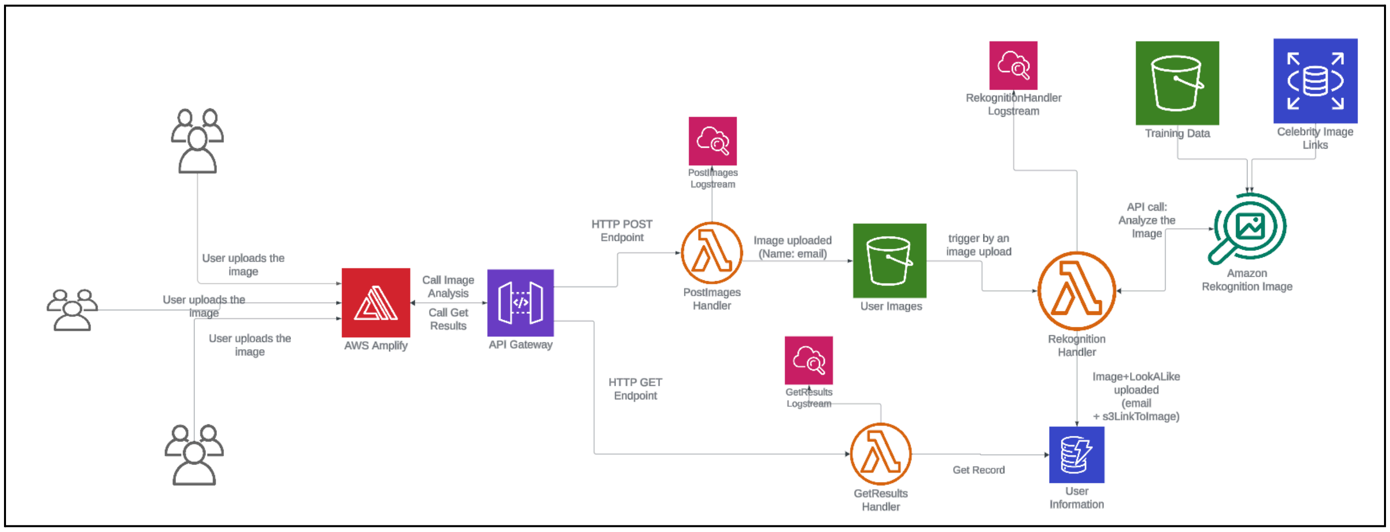
The goals of this project included (based on priority):

1. Our MVP – Users should be able to snap a photo and on submission find a celebrity that they resemble. We can’t guarantee how accurate it would be because we are relying on the internal logic of AWS Rekognition.
2. Our architecture – as a grad team we needed to implement a total of 8 services, and figuring out which piece goes where was important.
3. Keep the entire thing serverless(no EKS or EC2) - allowing us to only pay for what is used via Lambdas.
4. The user should receive at least one celebrity image, even if the similarity percentage is low. For this, we looked at different AWS Rekognition functionalities as well as our cosine similarity model. We found a better performance using AWS Rekognition and therefore went forward with that.
5. The front end should be hosted on Amplify and be accessible through any device.
6. If an existing user logs in, they should see the last celebrity image they matched with and should have the option to retake the image and see new results.

# You’re My Celebrity

Team 5 - White Paper

## Architecture Diagram



## How are the services being used? (This is what we ended up using in the final architecture)

### AWS Amplify

This is used to host the front end using its own set of tools and features that allow us to create full-stack applications easily. We need our website to be accessible using any device so Amplify was a clear choice. Also, we saw it could be connected to our GitHub repositories and as long as we mention the build and deploy path we can get it to work. This can be done through AWS Amplify studio or the command prompt on your local machine.

#### Amplify Studio

This is a visual interface that simplifies the creation of full-stack development applications that need to be scalable. The way we used it earlier before moving to Terraform was that we could link our GitHub repository directly. We can also add build and deploy instructions if we want particular files to be deployed.

#### Command line

Another way is to connect your local application through Amplify using the command line. This was a great way to test stuff in our spikes or proof of concept where we don't have everything figured out.

# You’re My Celebrity

Team 5 - White Paper

## AWS API Gateway

We are using API gateway as a bridge between the Frontend and lambdas functions for communication.

1. Send images to upload it in the S3 bucket:

   In this function, the API Gateway receives the image from the front end, processes the body request as per the template, and triggers the Lambda. This API request also sends the required headers to the browser so they can authenticate the request.

2. Get the images of similar celebrities from the backend and present them on the UI

   During this API call, the front end sends a GET request with the relevant username, which triggers the appropriate Lambda. The return from the lambda is the lookalike celebrity image.

## AWS S3

We use the S3 buckets to store images. Every other component refers/delivers images using the relevant S3 object links which makes data tracking easier. There are two buckets in our architecture diagram:

1. User - Image Bucket

   Every user image on submission gets stored here. When an image is uploaded in the S3 bucket a Lambda function is fired which sends the user’s image (S3 bucket link) to be processed by Rekognition.

2. Celebrity - Image Bucket

   We have about 160+ images of celebrities that we have stored here. On initialization, we load this into our Rekognition collection from this S3 bucket. Later on, we refer to this in our Rekognition logic.

## AWS Lambda

We have 3 Lambda functions

1. Uploading to S3 bucket

   This Lambda function is the first lambda function to be triggered by the API Gateway. It receives an image in base64 format from the front end, decodes it, and then uploads the image to the 'user-image' S3 bucket.

2. Rekognition logic

   When an image is uploaded into the S3 bucket, this Lambda function is fired which has some Rekognition logic(more on this later).

3. Update Results

   This Lambda function is triggered when the user-celebrity link is established. It returns the data to the API Gateway for display. Images are encoded in base64 format before being sent to the frontend.

## AWS Rekognition

The AWS Rekognition is a computer vision based system that automates many ML models so you don’t have to know how the model is coded, but rather focus on which suits you better. Since there wasn’t a pre-made “celebrity look-a-like” algorithm in AWS Rekognition, which meant we had to look at different models - some of Rekognition (compare_faces, search_faces_by_image) and a custom model - cosine similarity.

AWS Rekognition allows us to set our dataset in the form of creating a Rekognition collection. In the initial setup we load the images from the Celebrit-Image bucket into this collection. Each celebrity has only 1 image and they are given a face ID. Why we require this is mentioned in the below service.

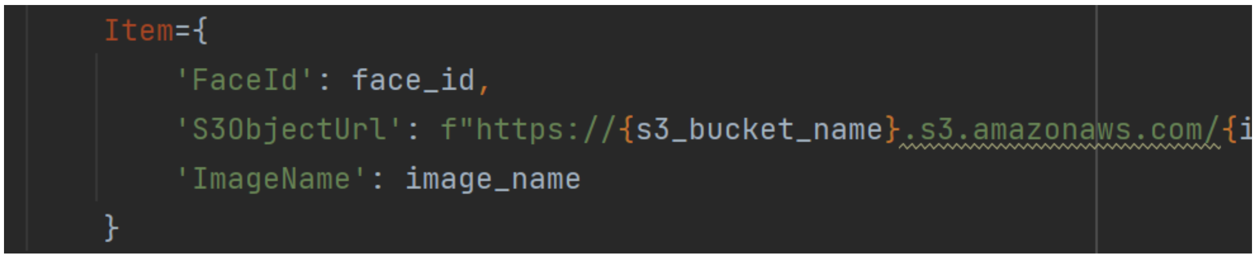
We looked into different models: compare_faces (Rekognition model) and cosine similarity (that checked like the x,y coordinates of facial features such as eye, lips, hair, etc.) were the first two models we looked at. Drawbacks of each: cosine similarity took a long time even on a dataset of only 6 images and we had problems loading this into a Lambda function as it was not allowing us to use the numpy library. On the other hand, compare_faces did the job, but it had to be run in a loop, which meant for every user image, we had to run a loop for all celebrity images. We finally found another image that did the work of compare_faces and did not need to be run in a loop, search_faces_by image function took in a collection ID (Rekognition collection mentioned above) and provided the s3 bucket link of the user’s image. It was here where we found that we could set a threshold of similarity percentage as well as set up max outputs like top 3 or top celebrities to be returned.

## AWS Dynamo DB & AWS RDS

We need storage for two entities:

### Rekognition collection - Celebrity Images (RDS)

When we upload an image into a Rekognition collection it would return as a face ID, so each celebrity had a face ID. It did not however also store the image and we needed to know which image corresponds to which face ID, so we made use of RDS here. Our structure for the Celebrity-Image table looks like:


```
Item={
    'FaceId': face_id,
    'S3ObjectUrl': f"https://{s3_bucket_name}.s3.amazonaws.com/{i
    'ImageName': image_name
}
```

### User-Celebrity -Image (Dynamo DB)

We also needed a place to store the user’s image along with the celebrities they matched with. So once the Rekognition logic was run and we had our list of celebrities(i.e. Their face ID and the similarity score, we simply found the corresponding celebrity s3 bucket image link and added that entry into a table in dynamo db.

### AWS CloudWatch

We made use of Cloud Watch to simply keep track of how our resources are doing as well as use it for when our logic fails.

## Resources Cost BreakDown

| **Service** | **Reasons for costing** | **Cost ( In USD)** | **Documentation** |
|---|---|---|---|
| AWS Amplify | Hosting apps don't cost for 12 months. | 0 | https://AWS.amazon.com/amplify/pricing/?nc=sn&loc=4 |
| | Consumed services by AWS amplify will cost for example EC2, API gateway, Lambda functions | Will need to maintain in their separate sections | |
| API Gateway | POST, GET Calls – 2000 calls are cost free under free tier – After that it costs us $3.50 for first | 3.50 | https://AWS.amazon.com/api-gateway/pricing/ |

| | | | |
|---|---|---|---|
| | 333 million request | | |
| | For Caching of API calls if needed – for example 0.5 GB | 0.02 | |
| AWS Rekognition | For the ML/ CV logic required to figure out the celebrity look alike.<br><br>10$ a month | 10.0 | We are only making use of the Rekognition collection( we know we have around 165 celebrities added) and the search_faces_by_image api (having about 10000 calls per month) |
| AWS Lambda | 1 million requests per month. $0.20 per 1 million requests thereafter, or $0.0000002 per request. | 0.20 | https://docs.aws.amazon.com/whitepapers/latest/how-aws-pricing-works/lambda.html |
| AWS CloudWatch | You can get started with Amazon CloudWatch for free. Most AWS Services (EC2, S3, Kinesis, etc.) send metrics automatically for free to CloudWatch. | | https://aws.amazon.com/cloudwatch/pricing/ |
| AWS Dynamo DB | 25GB of storage, along with 25 provisioned Write and 25 provisioned Read Capacity Units (WCU, RCU) which is enough to handle 200M requests per month. | | https://aws.amazon.com/dynamodb/pricing/ |

| | | | |
|---|---|---|---|
| AWS S3 | 5GB of Amazon S3 storage in the S3 Standard storage class; 20,000 GET Requests; 2,000 PUT, COPY, POST, or LIST Requests; and 100 GB of Data Transfer Out each month. | | |
| AWS RDS | Storage volume (General Purpose SSD (gp2)), Storage amount (10 GB), Nodes (1), Instance Type (db.m1.small), Utilization (On-Demand only) (100 %Utilized/Month), Deployment Option (Multi-AZ), Pricing Model (OnDemand) 111.80 per month | 111.80 | |

## Lessons learned along the way.

### What would you do differently if you had more time or had to do this over?

- With a better grasp on AWS services, we would have made a better design choice from the start, avoiding weeks of probing, debugging, and service-switching.
- We would want to leverage a non-standard database approach, possibly Graph Databases for user and celebrity linkage.
- Create a mobile application instead of a web-based application.
- The ML logic could be improved. Currently, with our limited number of tests, we get a low similarity score. Honing down on this with a better dataset would be an additional goal.
- While Amplify is said to be used for scalability, we would have liked some more time to test the robustness of the application. Creating

# You’re My Celebrity

Team 5 - White Paper

## Why is the cloud better for the project you picked?

- We need the application to be accessible anywhere
- We can make use of the Rekognition logic that is built into AWS.
- We don't have to worry about the maintenance of all resources selected(hosting, storage, computing), rather just pay a fee to the provider.
- Our application is scalable (due to Amplify and built-in concurrency).

## Any issues/challenges with the AWS technologies you selected?

- The issue we are facing with Amplify is the repositories available when we want to connect to a GitHub repository. So the issue is that collaborators cannot view or add the repository. This led to us facing issues when collaborating and having two repositories as one of the collaborators was trying to work with Amplify. We did find workarounds but did not find a concrete solution to this.
- Not an AWS tech issue, but we know the AWS Rekogniiton makes use of the imdb collection. We wanted to use a part of that as our initial dataset but due to copyright restrictions we could not get access to download that data as web scraping is a clear violation for using their site. In the end, we went with a Kaggle dataset which exhibits a lower diversity of celebrities.
- One of the prior services we were considering was AWS SNS. Having worked with in homework we had read an article about how a push notification could be sent to mobile devices. We thought we could do the same with websites (HTTP endpoints), but after a lot of trial and error, we were not able to get this to work. There were limited tutorials and examples about how to get the subscription to be confirmed and therefore we ended up not using this service.
- We had to deliver the image in base64 format across the front end because the image got corrupted if sent in another format via the API gateway. We also had to make various adjustments in the IAM role and trusted connections, as well as design a unique template to transfer the image through API Gateway.

## What other features or enhancements would you like to see added to your project if you had more time?

- We had thought of integrating Gen AI with this, so for the celebrity that you’re matched with, it could generate a short scene of a movie or a poster with your face replacing the face of the celebrity. Of course, this falls into the domain of creating deep fakes.
- We also thought of expanding this into not just celebrities but also characters of popular TV shows, like which character you look like, or into music domains - like which Taylor Swift album would suit you best based on your image.