



## **AWS Project:** Image recognition app

# Documentation

By : Abouchiba Mohamed Yassir

To see the files used in this project go to this link: https://github.com/M-Yassir/AWS-Projects/tree/main/Facial Recognition App

#### Introduction



This project demonstrates the development of a fully serverless face recognition application using Amazon Web Services (AWS). The system is designed to provide a secure and efficient way for administrators to register employees and for employees to authenticate themselves using facial recognition technology.

Administrators can register employees by submitting their name and an image, which is then processed and stored using AWS Rekognition for facial recognition. The employee's facial features are indexed in a Rekognition Collection to enable future authentication. Employees can log in by uploading an image, and the system verifies their identity by comparing the uploaded image against the stored employee database. If a match is found, the employee is successfully authenticated.

Through this project, I gained hands-on experience in serverless architecture, facial recognition systems, cloud security, and AWS service integration, making it a valuable exploration of AI-powered authentication solutions.

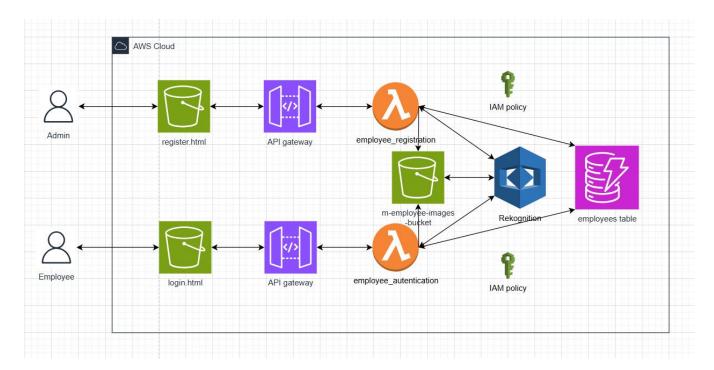
#### **Key Features**

- Employee Registration: Admins can register employees by submitting their name and an image. The image is uploaded to an S3 bucket, and the face is indexed in a Rekognition collection. Employee details (name, ID, and image URL) are stored in a DynamoDB table. **Enhanced security** is ensured by denying registration if an employee's face is already registered, even if the image is different (e.g., a different pose or angle), preventing duplicate registrations.
- Employee Login: Employees can log in by uploading their image, and the system uses AWS Rekognition to match the face against the indexed collection. If a match is found, the employee is authenticated, and their details are displayed.
- Serverless Architecture: The application uses AWS Lambda for backend logic, API Gateway for RESTful APIs, and DynamoDB for data storage. It leverages Amazon S3 for image storage and Amazon Rekognition for facial recognition, ensuring scalability, cost-efficiency, and ease of deployment.

- Static Website Hosting: The frontend, built with HTML, CSS, and JavaScript, is hosted on Amazon S3 for cost-effective and scalable static website hosting.



#### **AWS architecture diagram**



#### - Employee registration workflow:

Admins submit an employee's name and image. The image is temporarily uploaded to S3, and Rekognition checks if the face is already indexed. If no match is found, the image is stored permanently in S3, employee details are saved in DynamoDB, and the face is indexed in Rekognition for future recognition.

#### - Employee login workflow:

Employees upload their image, which is temporarily stored in S3. Rekognition searches for the face in its collection. If a match is found, the employee's details are retrieved from DynamoDB and displayed. If no match is found, an error message is shown.

### Step-by-Step implementation

**NB:** - All the necessary code is on the repository given above.

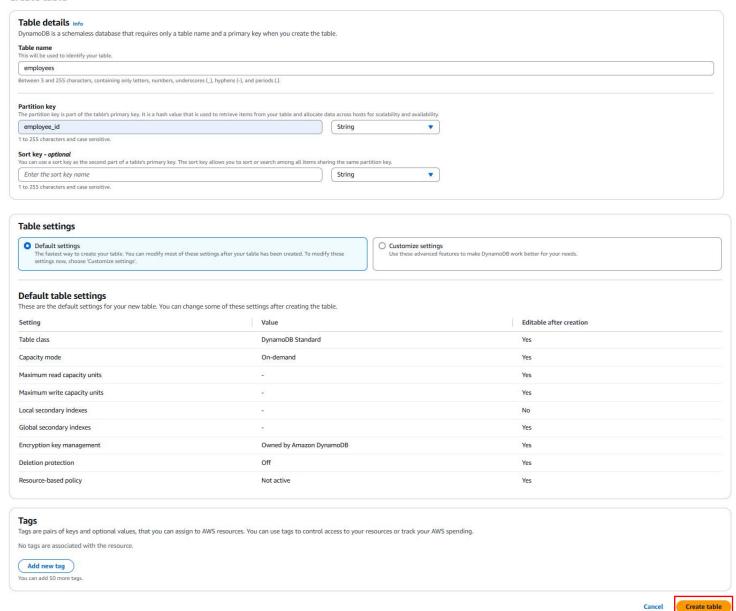
- The creation of the html files isn't covered here as it isn't our topic.

#### Step1: Create a DynamoDB table and the S3 bucket:

First we go to DynamoDB service then we click on 'Create table'

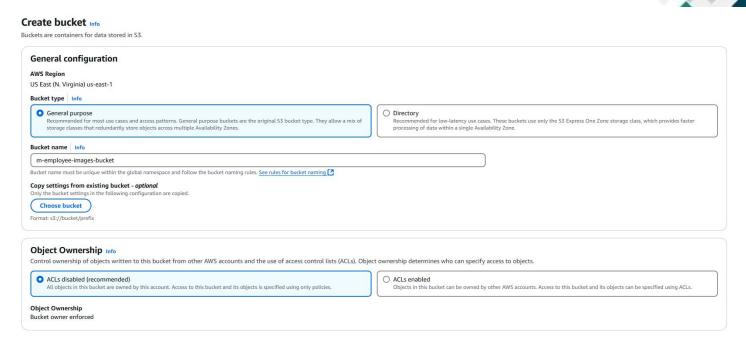
We fill in the necessary informations like it is shown below:

#### Create table

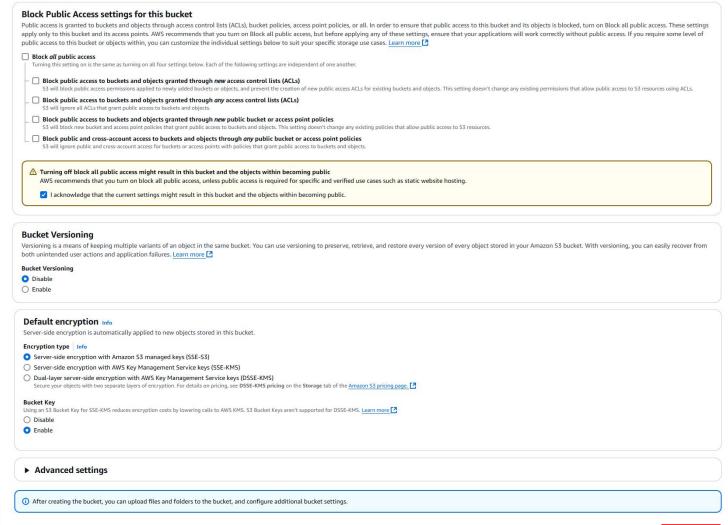


We click on 'Create table'

#### We go to S3 to create our bucket, named 'm-employee-images-bucket'

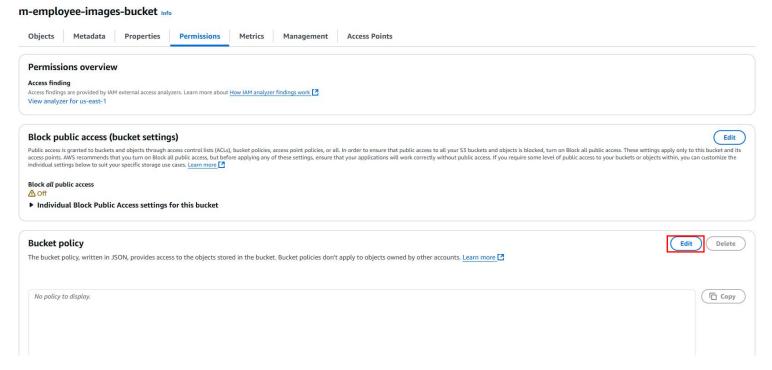


## We need also to **disable** 'Block public access to this bucket' in order to let our lambda functions and Rekognition access it

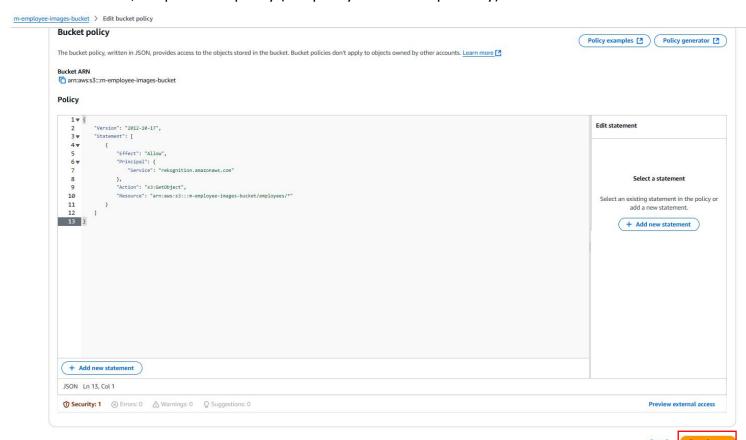


After creating the bucket, we need to assign a policy to it in order to let rekognition access it, so we click on the bucket name, we go to the permissions tab and we click on 'edit' on the policy section





#### Then, we paste the policy (the policy is on the repository)



# Step2: Create a rekognition collection, create the employee\_regitration lambda function and trigger it with an API gateway:

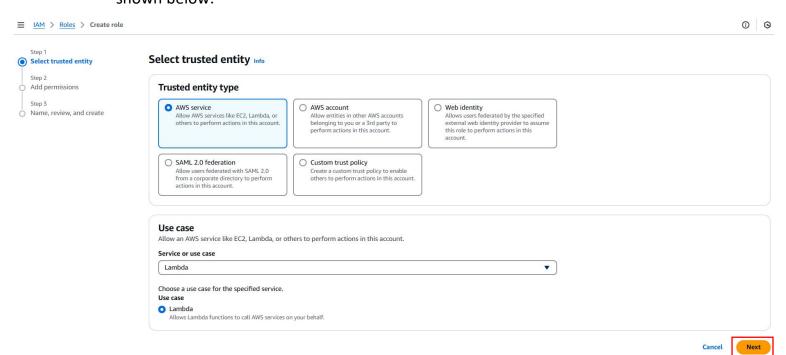
To create a rekognition collection, we go to CloudShell, write the following command:

aws rekognition create-collection --collection-id "employee\_faces"

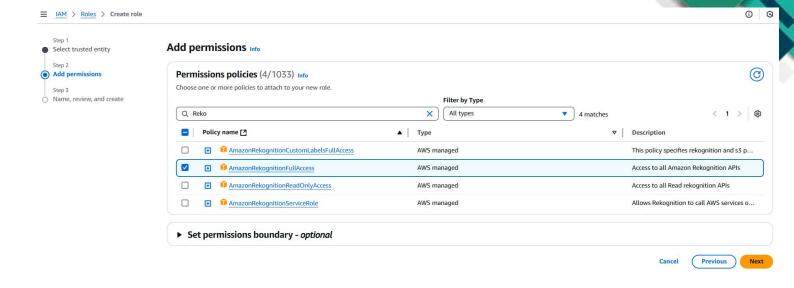
We should get this output:



Before creating the lambda functions, we should create a role because without it, lambda cannot access our resources, so we go to IAM>Roles>Create role, then we fill in the following informations like it is shown below:

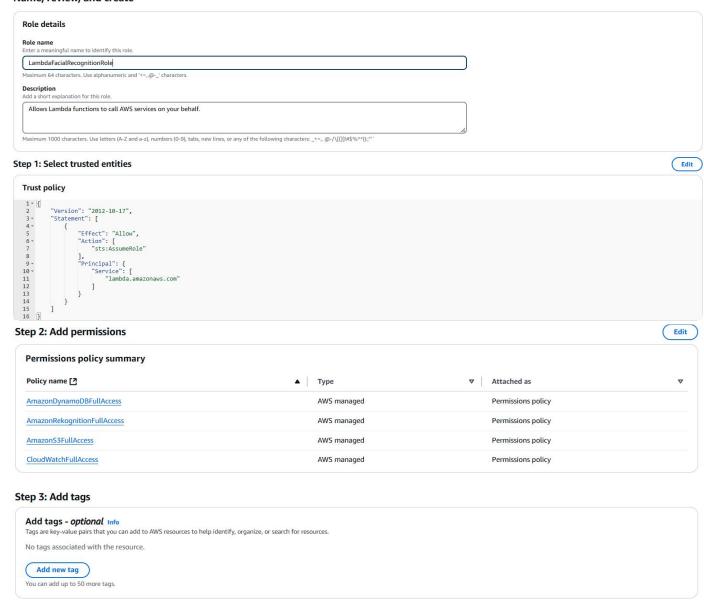


We add the following permissions: AmazonS3FullAccess,
AmazonDynamoDBFullAccess, CloudWatchFullAccess,
AmazonRekognitionFullAccess by searching their names and selecting them like it is shown for example for AmazonRekognitionFullAccess



After adding all these policies, we click on 'next', we should get this:

#### Name, review, and create



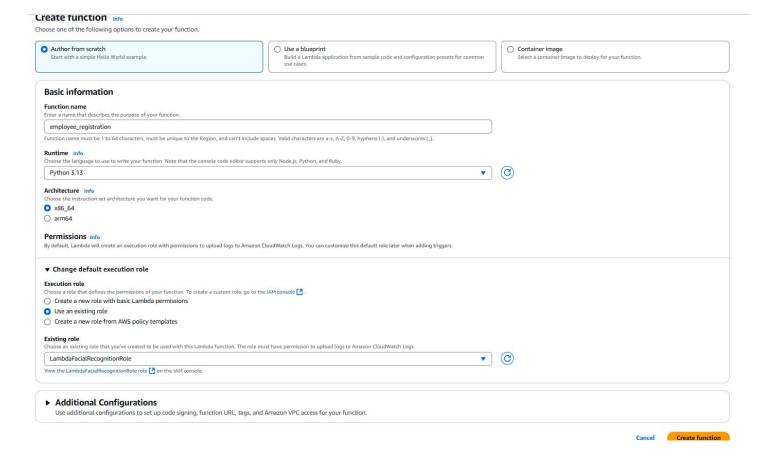
Create role

Previous

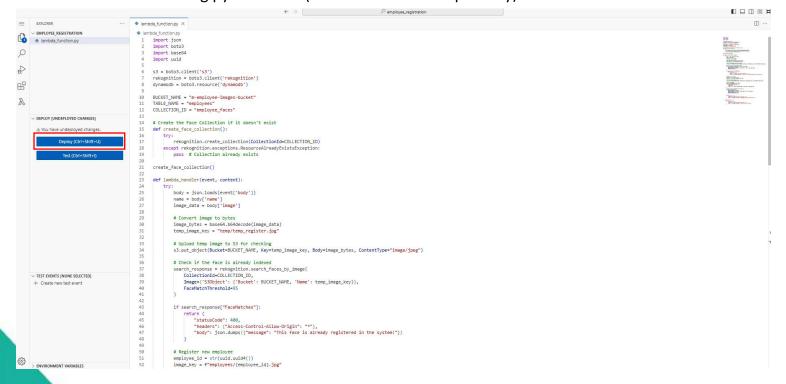
Cancel

We name it LambdaFacialRecognitionRole and we click on 'create role'

Next up, we go to lambda and we create a function named 'employee\_registration' without forgetting to add the LambdaFacialRecognitionRole created before

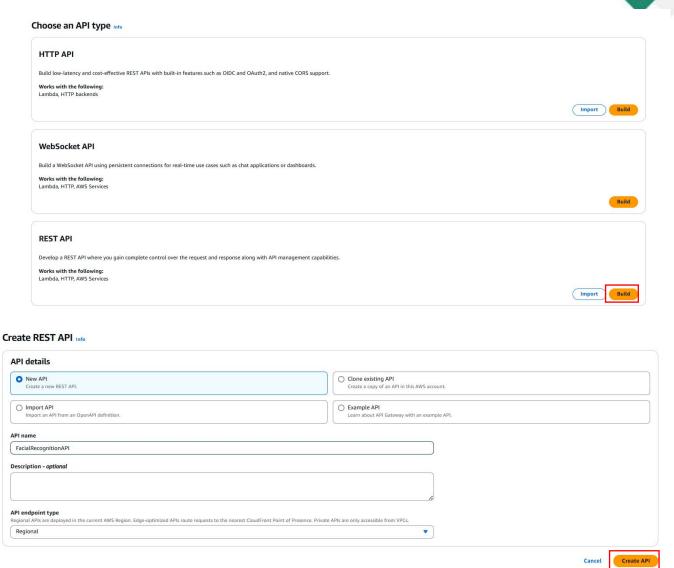


We click on 'create function', after that, we go to 'code source' and we type the following python code (the code is on the repository):



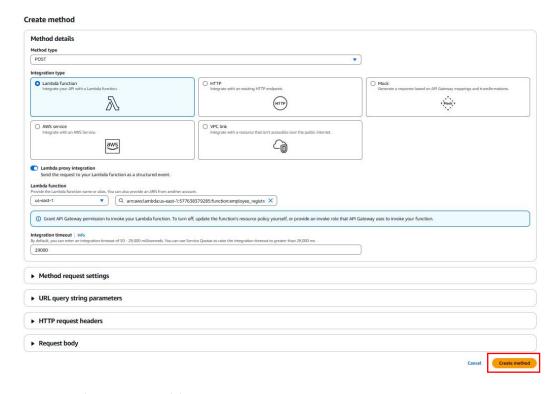
#### We click on 'Deploy'.

#### We go now to create a REST API on the API gateway



We create a new resource named 'register'



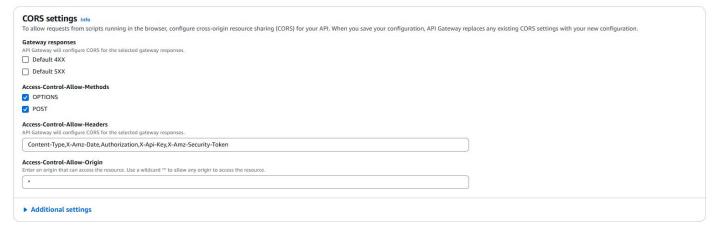




We create a POST method on 'register' and integrate it with our lambda function

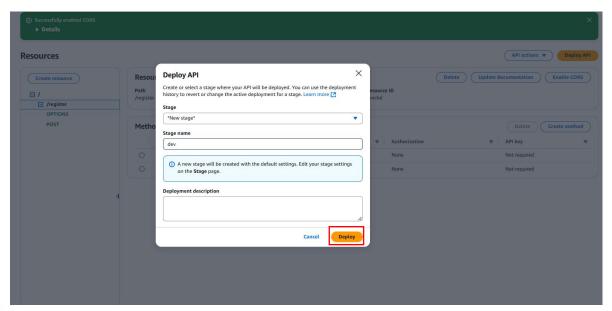
#### Then, we enable CORS:

#### **Enable CORS**

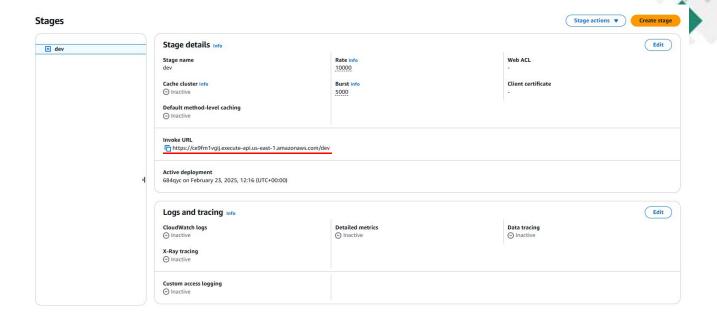


oncel Save

#### and we deploy the API:



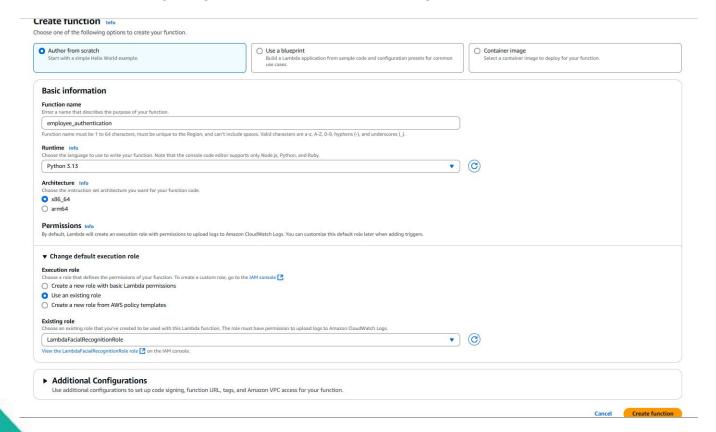
#### After this we should get an invoke URL



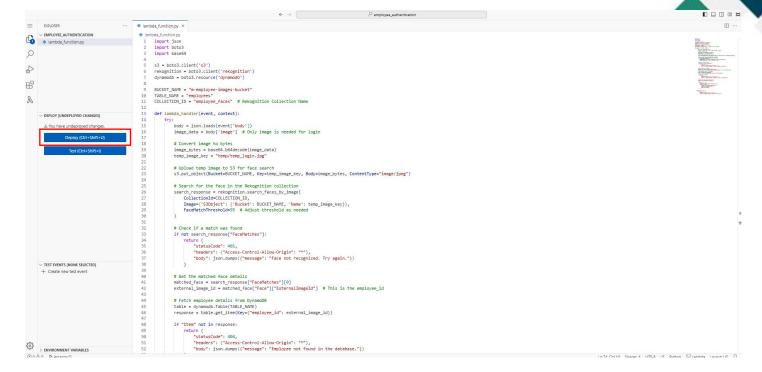
We'll paste in the variable 'apiEndpoint' in the **register.html** and **register.html** files before hosting them on S3.

# Step3: create the employee\_authentication lambda function and trigger it with an API gateway:

We go to lambda and we create a function named 'employee\_authentication' without forgetting to add the **LambdaFacialRecognitionRole** created before



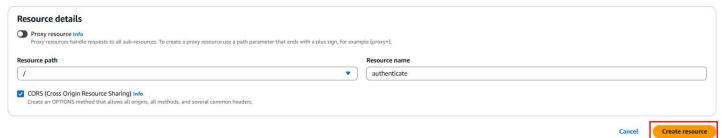
We click on 'create function', after that, we go to 'code source' and we type the following python code (the code is on the repository):



We click on 'Deploy'.

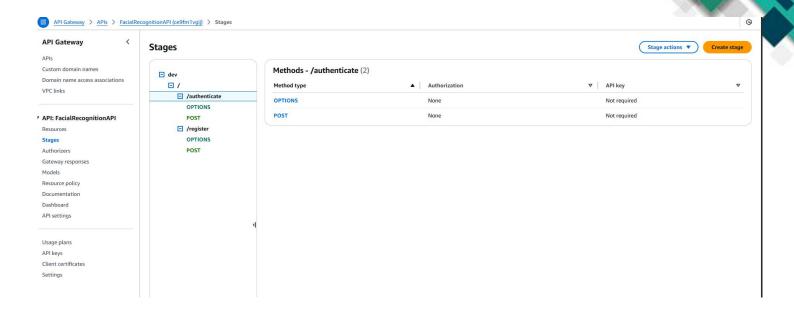
Then, we go to the API gateway and we create another resource named authenticate

#### Create resource



We do the same things we did on the register API except the POST method should be integrated with 'employee\_authentication' lambda function.

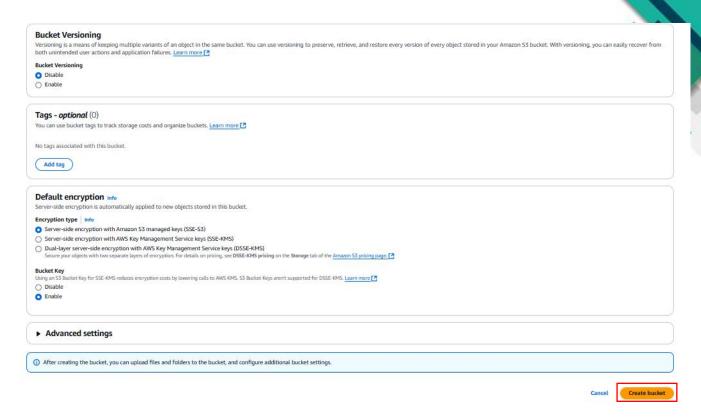
We should get a result like this:



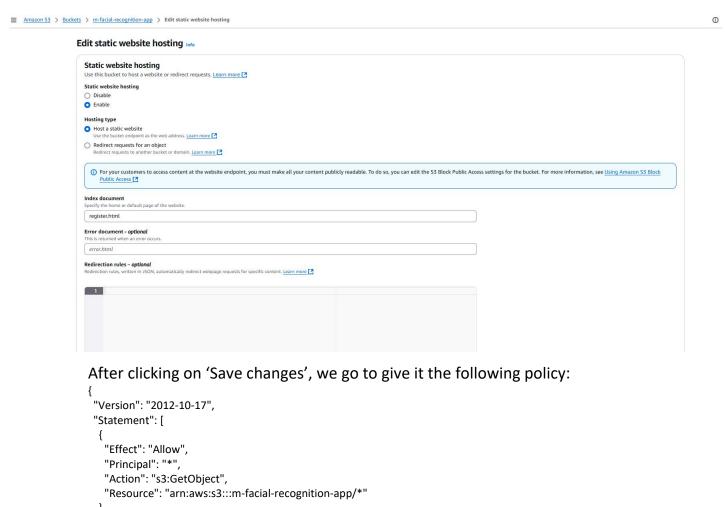
#### Step4: Hosting the 2 html files on S3:

After replacing the 'apiEndpoint' on the register.html file by https://ce9fm1vgij.execute-api.us-east-1.amazonaws.com/dev/register and the 'apiEndpoint' on login.html file by https://ce9fm1vgij.execute-api.us-east-1.amazonaws.com/dev/authenticate We host them on S3, we create a new bucket like it is shown below:

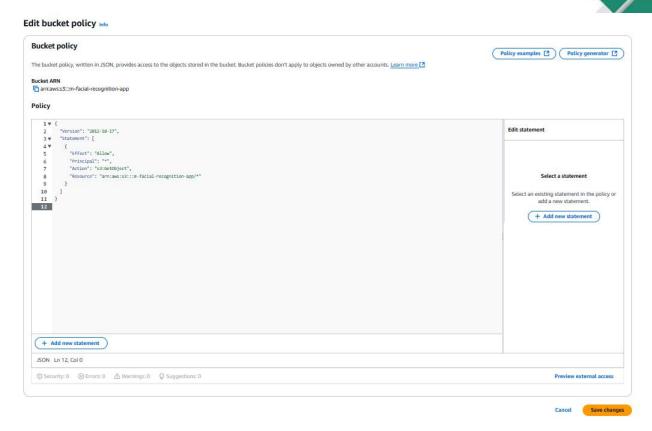
General configuration	
NWS Region	
JS East (N. Virginia) us-east-1	
Sucket type Info	
<ul> <li>General purpose</li> <li>Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix storage classes that redundantly store objects across multiple Availability Zones.</li> </ul>	Directory     Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.
Sucket name Info	
m-facial-recognition-app	
lucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming [2]	
Copy settings from existing bucket - aptional	
Only the bucket settings in the following configuration are copied.	
Choose bucket	
ormat: s3://bucket/prefix	
ACLs disabled (recommended) All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.	Object ownership determines who can specify access to objects.  O ACLs enabled Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.
ACLs disabled (recommended) All objects in this backet are owned by this account. Access to this backet and its objects is specified using only policies.  Object Ownership	○ ACLs enabled
<ul> <li>ACLs disabled (recommended)</li> <li>All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.</li> </ul>	○ ACLs enabled
ACLs disabled (recommended) All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  Object Ownership  bucket owner enforced  Block Public Access settings for this bucket	Oxforms in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.
ACLs disabled (recommended) All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  Diject Ownership  Block Public Access settings for this bucket  Block Public Access settings for this bucket  Under a granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies,	O ACLs enabled Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settinglying any of these settings, ensure that your applications will work correctly without public access. If you require some level or
ACLs disabled (recommended) All objects in this backet are owned by this account. Access to this backet and its objects is specified using only policies.  Deject Ownership  Block owner enforced  Block Public Access settings for this bucket  Unlike access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, poly only to this buckets and its access points. AND recommends that you turn on Block all public access, but before ag	Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settinglying any of these settings, ensure that your applications will work correctly without public access. If you require some level of gets cases. Learn more [2]
ACLs disabled (recommended) All objects in the bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  All object Ownership  Block Public Access settings for this bucket  Block Public Access settings for this bucket  Block Public Access granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies,  Block Block Public Access to buckets and its access points. AND recommends that you turn on Block all public access, but before an  Block all public access  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one anoth  Block public access to buckets and objects granted through new access control lists (ACLs)	Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settinglying any of these settings, ensure that your applications will work correctly without public access. If you require some level of gets cases. Learn more [2]
ACLs disabled (recommended) All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  Deject Ownership Block Public Access settings for this bucket  Block Public Access settings for this bucket  Block Public Access granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies,  Block Public Access to this bucket and its access points. After recommends that you turn on Block all public access, but before ago bublic access to this bucket or objects within, you can customize the individual settings below to suit your specific storal  Block all public access  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another the block public access to buckets and objects granted through new access control lists (ACLs)	O ACLs enabled Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settir plying any of these settings, ensure that your applications will work correctly without public access. If you require some level of ge use cases. Learn more [2]
ACLs disabled (recommended) All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  Diject Ownership  ucket owner enforced  Block Public Access settings for this bucket  ublic access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, ply only to this bucket and its access points. ANY recommends that you turn on Block all public access, but before ag  ublic access to this bucket or objects within, you can customize the individual settings below to suit your specific stora  Block all public access  Turning this atting on is the same as turning on all four settings below. Each of the following settings are independent of one anoth  Block public access to buckets and objects granted through new access control lists (ACLs)  Block public access to buckets and objects granted through new access control lists (ACLs)  Block public access to duckets and objects granted through new access control lists (ACLs)	Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  Or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These setting plying any of these settings, ensure that your applications will work correctly without public access. If you require some level of ge use cases. Learn more [2]  access. These setting buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
ACLs disabled (recommended) All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  All objects on this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  All object Ownership  Libect Ownersh	Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settiplying any of these settings, ensure that your applications will work correctly without public access. If you require some level of the cases. Learn more ?  ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to \$33 resources using ACLs.  y existing policies that allow public access to \$33 resources.
ACLs disabled (recommended) All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.  All object Ownership  ucket owner enforced  Block Public Access settings for this bucket  utilities cases is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies  pply only to this bucket and its access points. ANP recommends that you turn on Block all public access, but before ag  sublic access to this bucket or objects within, you can customize the individual settings below to suit your specific stora  Block public access  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one anoth  Block public access to buckets and objects granted through new access control lists (ACLs)  So will block public access to buckets and objects granted through new access control lists (ACLs)  So will block public access to buckets and objects granted through new access control lists (ACLs)  So will block public access to buckets and objects granted through new public bucket or dependent of new public access  Block public access to buckets and objects granted through new public bucket or access point policies  So will block new bucket and access point policies that given public access to buckets and objects granted through new public bucket or access point policies  So will block new bucket and access point policies that given public access to buckets and objects through any public access to access point policies  Block public and cross-account access to buckets and objects through any public bucket or access point policies	O ACLs enabled Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.  or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These setting plying any of these settings, ensure that your applications will work correctly without public access. If you require some level of gerse cases. Learn more [2]  set.  ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.  y existing policies that allow public access to S3 resources.  cles



We enable static website hosting, we go to properties section and all the way down to it and we click on 'edit', we fill it as it is shown below

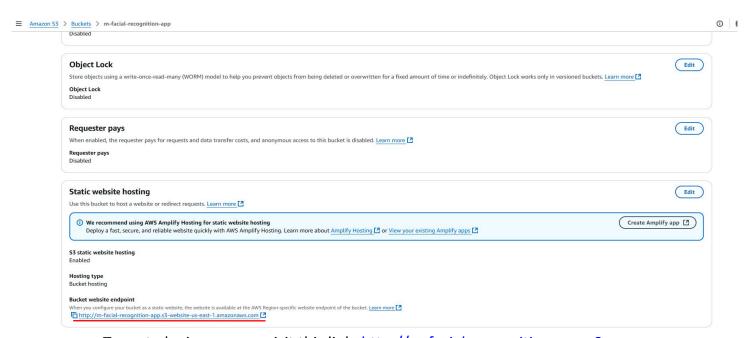


] }



We cick on 'Save changes'

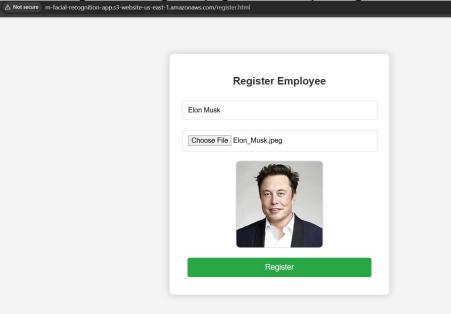
After uploading the **register.html** and the **login.html** files, we go now to get the link in the properties section:

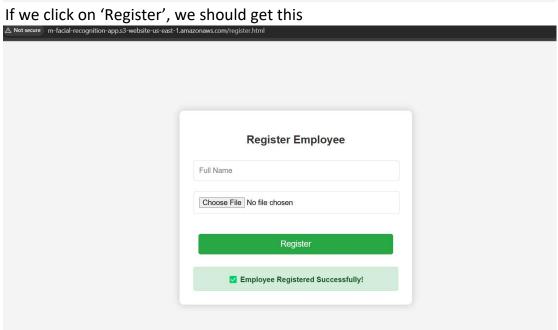


To go to login page we visit this link: <a href="http://m-facial-recognition-app.s3-website-us-east-1.amazonaws.com/register.html">http://m-facial-recognition-app.s3-website-us-east-1.amazonaws.com/register.html</a>

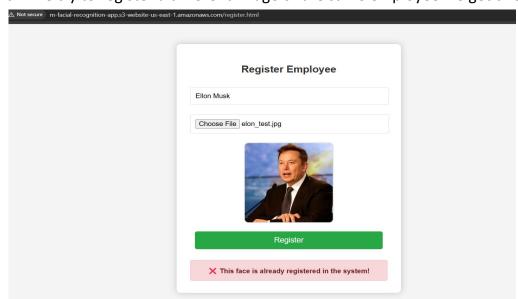
To go to register page we visit this link: <a href="http://m-facial-recognition-app.s3-website-us-east-1.amazonaws.com/login.html">http://m-facial-recognition-app.s3-website-us-east-1.amazonaws.com/login.html</a>

So now we go to the register page and we try to register a user for example:

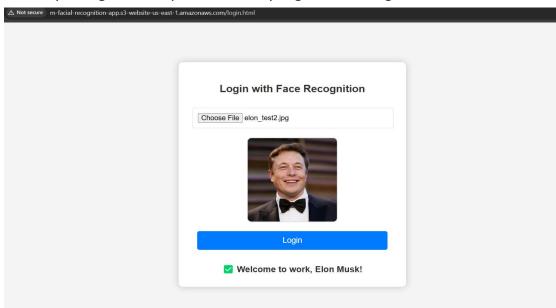


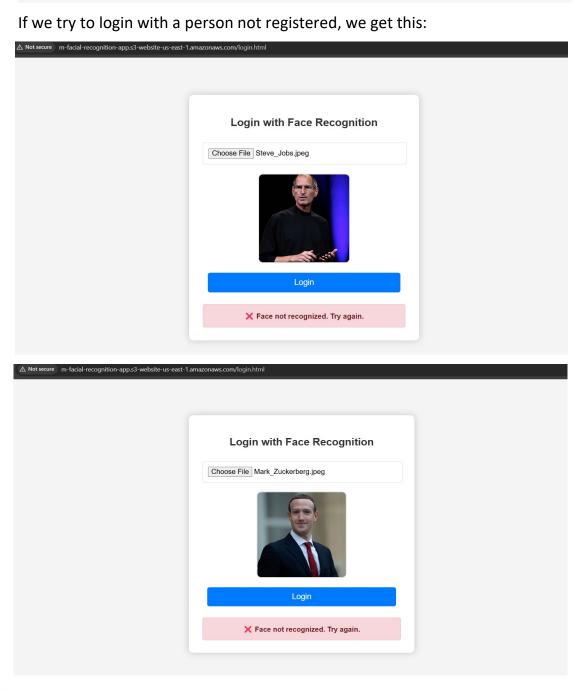


And if we try to register a different image of the same employee we get this



If we try to login with a person already registered, we get this:





#### Conclusion

This project has been an insightful journey into building a fully serverless facial recognition web application using AWS services. By leveraging AWS Lambda, API Gateway, DynamoDB, Rekognition, and S3, I successfully implemented a scalable, cost-efficient, and automated authentication system within the AWS Free Tier.

#### Challenges Faced & Lessons Learned:

1) Preventing Duplicate Registrations with the Same Person's Image A major challenge was ensuring that two different users could not register using the same person's image. This required implementing face comparison logic before storing new employees.

I developed a system that checks for existing faces in Rekognition Collections before registration, preventing unauthorized duplicate enrollments.

- 2) Managing S3 Bucket Permissions for Image Storage Ensuring that images were stored securely while remaining accessible to AWS Rekognition and Lambda required fine-tuned S3 permissions. Without proper IAM policies, Rekognition and Lambda failed to access stored images.
- 3) DynamoDB Query Optimization for Fast Authentication When an employee logs in, the system retrieves their data from DynamoDB based on the recognized face. Ensuring fast and efficient lookups was crucial, especially as the database grows.

I learned to structure DynamoDB tables effectively, use primary keys (employee\_id) for quick lookups, and ensure that API responses remain fast and scalable.

#### Final Takeaways:

Through this project, I gained hands-on experience in serverless architecture and AWS facial recognition while building a real-world authentication system. Key skills I developed include:

- ✓ Implementing face recognition authentication using AWS Rekognition
- $\operatorname{\mathscr{C}}$  Configuring IAM roles and permissions for secure AWS service communication
- arphi Handling S3 bucket policies for secure image storage and retrieval

This project has deepened my understanding of serverless cloud development, showcasing its advantages in terms of scalability, security, and cost-efficiency. Moving forward, I am excited to explore enhanced biometric security features, multi-factor authentication, and more advanced AI-driven recognition systems in cloud-based solutions.