**Práctica Sprint 01 AWS.**

**Jose Antonio Ortega Arroyo**

**Ángel Otálora Abellón**

**Juan Carlos Fermosell Romero**

**Victoria Gabriela Navarrete Larive**

1. **Flujo de trabajo de registro de usuario**

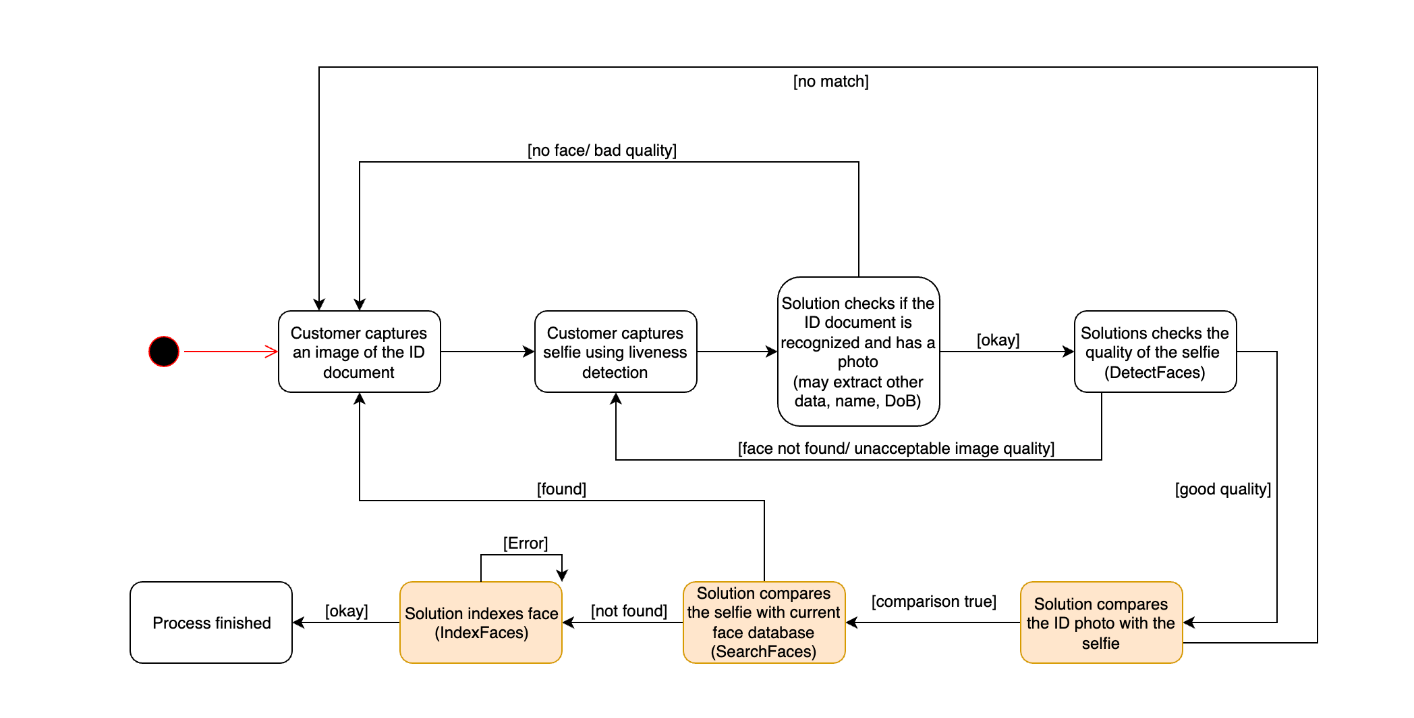
**La siguiente figura muestra un flujo de trabajo de muestra de un nuevo registro de usuario. Los pasos típicos en este proceso son:**

-El usuario captura la imagen de un selfie y la imagen de un documento de identidad.

-Comprobación de calidad de la imagen del selfie.

-Comparación de la imagen del selfie con la imagen del rostro del documento de identidad.

-Comprobación del selfie con una base de datos de rostros de usuarios existentes.



Antes de implementar la solución, es importante conocer los siguientes conceptos y descripciones de **API**:

* Colecciones: Amazon Rekognition almacena información sobre rostros detectados en contenedores del lado del servidor conocidos como colecciones. Puede usar la información facial que se almacena en una colección para buscar caras conocidas en imágenes, videos almacenados y transmisión de videos. Puede usar colecciones en una variedad de escenarios. Por ejemplo, puede crear una colección de rostros para almacenar imágenes de credenciales escaneadas mediante la operación IndexFaces. Cuando un empleado ingresa al edificio, se captura una imagen de su rostro y se envía a la operación SearchFacesByImage. Si la coincidencia de caras produce una puntuación de similitud suficientemente alta (digamos 99%), puede autenticar al empleado.
* API DetectFaces: esta API detecta rostros dentro de una imagen proporcionada como entrada y devuelve información sobre los rostros. En un flujo de trabajo de registro de usuarios, esta operación puede ayudarlo a filtrar imágenes antes de pasar al siguiente paso. Por ejemplo, puede verificar si una foto contiene una cara, si la persona identificada está en la orientación correcta y si no usa ningún bloqueador facial, como gafas de sol o gorra.
* API IndexFaces: esta API detecta rostros en la imagen de entrada y los agrega a la colección especificada. Esta operación se utiliza para agregar una imagen filtrada a una colección para consultas futuras.
* API SearchFacesByImage: para una imagen de entrada dada, la API primero detecta la cara más grande en la imagen y luego busca en la colección especificada las caras coincidentes. La operación compara las características de la cara de entrada con las características de la cara en la colección especificada.
* API CompareFaces: esta API compara un rostro en la imagen de entrada de origen con cada uno de los 100 rostros más grandes detectados en la imagen de entrada de destino. Si la imagen de origen contiene varios rostros, el servicio detecta el rostro más grande y lo compara con cada rostro detectado en la imagen de destino. Para nuestro caso de uso, esperamos que tanto la imagen de origen como la de destino contengan una sola cara.
* API DeleteFaces: esta API elimina rostros de una colección. Especifique una ID de colección y una matriz de ID de rostros para eliminar.

La que se empleará es **API Compare Faces, con la librería boto3**. Para usar las imágenes ha sido necesario crear buckets para guardar las imágenes.

aws s3api create-bucket --bucket myfacetestbucket2 --region us-east-1

aws s3 cp roger\_federer.jpg s3://myfacetestbucket1

Abajo se muestra el código empleado con el resultado de una prueba en la que se comparan dos imágenes de la misma persona.

import io

from PIL import Image, ImageDraw, ExifTags

import boto3

if \_name\_ == "\_main\_":

sourceFile='roger\_federer.jpg'

targetFile='roger\_federer\_1.jpg'

client=boto3.client('rekognition')

imageSource=open(sourceFile,'rb')

imageTarget=open(targetFile,'rb')

response=client.compare\_faces(SimilarityThreshold=70,

SourceImage={'Bytes': imageSource.read()},

TargetImage={'Bytes': imageTarget.read()})

############################

#open image and get image data from stream.

image2 = Image.open(open(targetFile,'rb'))

stream = io.BytesIO()

if 'exif' in image2.info:

exif=image2.info['exif']

image2.save(stream,format=image2.format, exif=exif)

else:

image2.save(stream, format=image2.format)

image\_binary = stream.getvalue()

response2 = client.detect\_faces(Image={'Bytes': image\_binary}, Attributes=['ALL'])

##############################

for faceMatch in response['FaceMatches']:

position = faceMatch['Face']['BoundingBox']

confidence = str(faceMatch['Face']['Confidence'])

print('The face at ' +

str(position['Left']) + ' ' +

str(position['Top']) +

' matches with ' + confidence + '% confidence')

width1, height1 = image1.size

width2, height2 = image2.size

draw1 = ImageDraw.Draw(image1)

draw2 = ImageDraw.Draw(image2)

box1 = response1['FaceDetails'][0]['BoundingBox']

box2 = response2['FaceDetails'][0]['BoundingBox']

left1 = width1 \* box1['Left']

top1 = height1 \* box1['Top']

left2 = width2 \* box2['Left']

top2 = height2 \* box2['Top']

draw1.rectangle([left1,top1, left1 + (width1 \* box1['Width']), top1 +(height1 \* box1['Height'])])

image1.show()

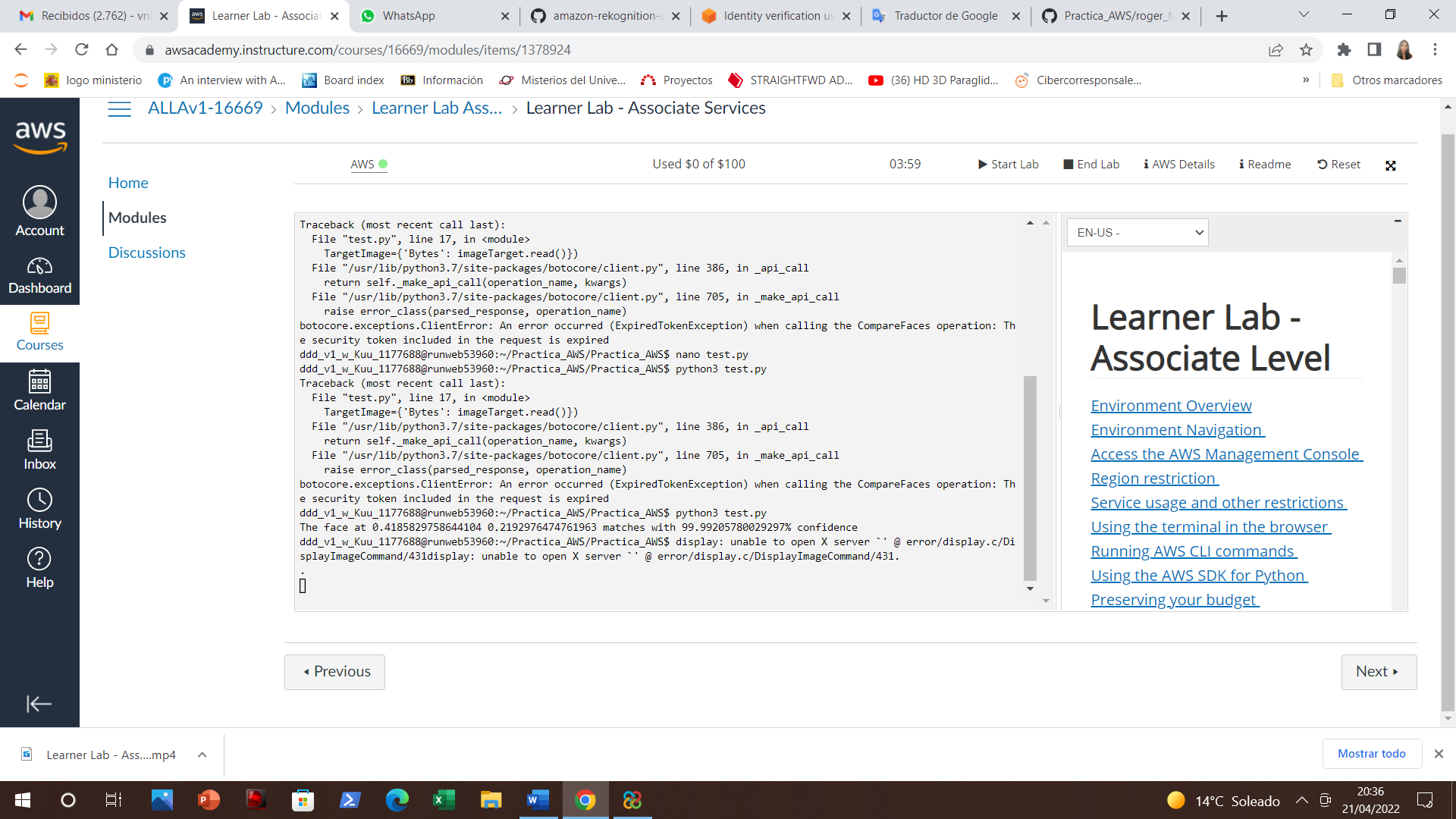
draw2.rectangle([left2,top2, left2 + (width2 \* box2['Width']), top2 +(height2 \* box2['Height'])])

image2.show()

#imageSource.close()

#imageTarget.close()

Aquí se muestra el resultado.



Se ve que compara correctamente las imágenes.

El objetivo es afinarlo para poder hacer una comparativa desde una base de datos con la suficiente precisión.