Distributed Systems PS - Final Presentation

Smart Store Project

The Team

Tobias Beiser

Daniel-Sebastian Carp-Popescu

Lukas Geisler

Lukas Eisenstecken

Project Premise

Smart Store

Camera monitoring store entrance

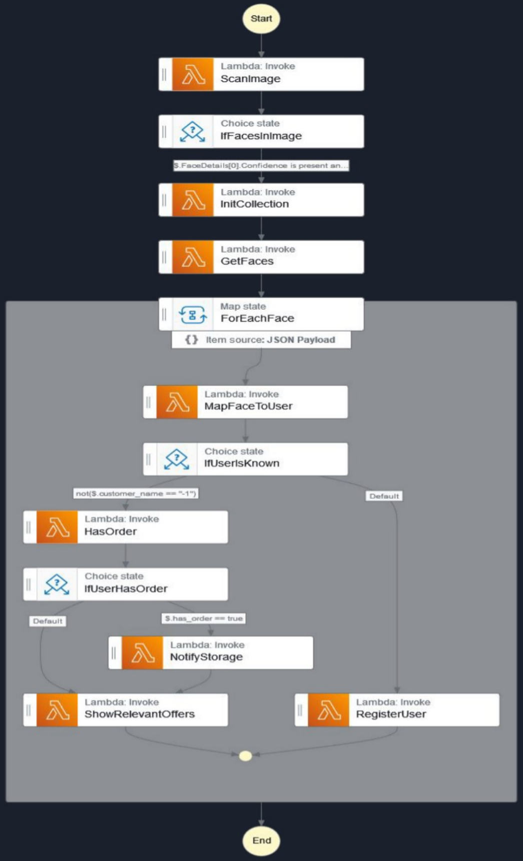
Matches faces of people who enter against database

If they have an active order it will be prepared for pickup.

Recommends additional products based on a customers purchasing history

Photo by Pawel Czerwinski on Unsplash

Workflow


Start
Lambda: Invoke
ScanImage
Choice state
IfFacesInImage
$.FaceDetails[0].Confidence is present an...
Lambda: Invoke
InitCollection
Lambda: Invoke
GetFaces
Map state
ForEachFace
{} Item source: JSON Payload
Lambda: Invoke
MapFaceToUser
Choice state
IfUserIsKnown
not($.customer_name == "-1")
Default
Lambda: Invoke
HasOrder
Choice state
IfUserHasOrder
Default
$.has_order == true
Lambda: Invoke
NotifyStorage
Lambda: Invoke
ShowRelevantOffers
Lambda: Invoke
RegisterUser
End

Lambda: scanImage

- Takes the bytes string of an image as input
- Runs it through Amazon Rekognition
- Returns the number of people in an image and their face details

Lambda: initCollection

- Create a Rekognition collection
- Only executed if collection does not exist (takes ~50s)
- Index all faces of existing customers
    a. Get all customers from database
    b. Each customer has a folder with images of its face (S3 bucket)
    c. External Image Id=Customer Id
- This collection is essential for mapping new images to customers

Lambda: detectFace

- Uses FaceDetails of scanImage function
  - Holds information about bounding box in source image
- Extracts all the faces from the source image
- Return the extracted faces as array of base64 strings

Lambda: mapFaceToUser

- Input: Single cropped face as base64 string
- Use the Rekognition collection created earlier to find best match
  - Similarity must be > 80%
- If match is found then return the Customer Id
- Otherwise return the face string

Lambda: registerCustomer

- If no match was found we register the unknown customer with a temporary name
- Create a new user in database
- Upload the unknown face to S3
- Add image to Rekognition collection

Lambda: hasOrder

- Orders are directly associated with customers
- Checks if a given customer has active orders in the database
- Information is used for the workflows control flow

Lambda: notifyStorage

- If the customer has an order, the next step will be to notify the storage, once the customer enters the store
- It serves as a simulation to notify the employees that they have to prepare an order
- The products ordered by the customer will be marked as delivered and the records in Redis are updated

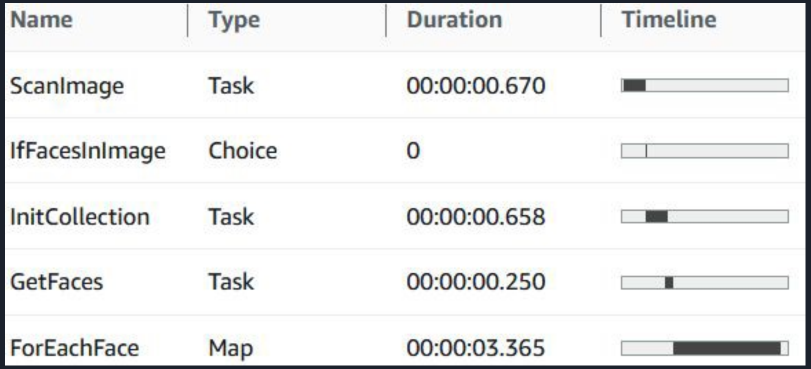
Lambda: showRelevantOffers

- Suggest additional purchases at checkout.
- A customer’s past orders are archived in the database.
- Based on a customer's past purchasing behaviour, additional products will be suggested.
- This projects Implementation simply picks a few random items of a customer’s past orders.
- More sophisticated deployments of this application could use models specifically trained for this purpose.

Demo

Evaluation / Execution Time

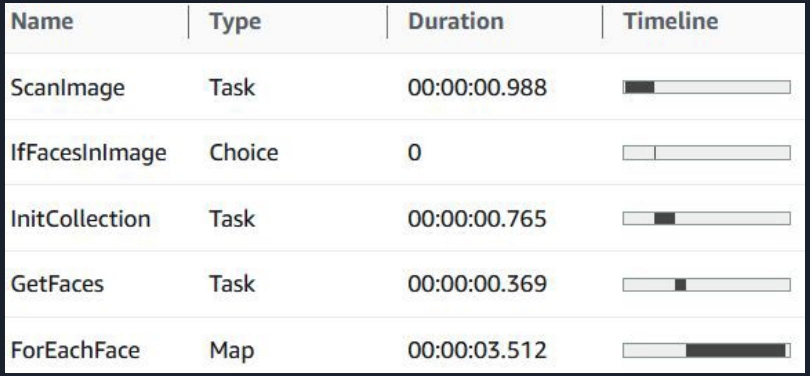
1 Person in Image:



Name	Type	Duration	Timeline
ScanImage	Task	00:00:00.670	
IfFacesInImage	Choice	0	
InitCollection	Task	00:00:00.658	
GetFaces	Task	00:00:00.250	
ForEachFace	Map	00:00:03.365	

Total execution time: 5.191 seconds

4 People in Image:



Name	Type	Duration	Timeline
ScanImage	Task	00:00:00.988	
IfFacesInImage	Choice	0	
InitCollection	Task	00:00:00.765	
GetFaces	Task	00:00:00.369	
ForEachFace	Map	00:00:03.512	

Total execution time: 5.811 seconds

Evaluation / Cost Calculation

Assumption: 1000 Customers per day

Step Functions:
- $0.025 per 1,000 state transitions \ average 25 Transitions per Customer
- Total Cost = 1000 * 25 * 0.025 / 1000 = $0.625

Lambdas
- cost per 1ms = $0.0000000083 \ assume 6s execution time /customer
- Total cost = 1000 * 6000 * 0.0000000083 = $0.0498

EC2 Redis Server:
- cost per hour = $0.0116
- Total cost = 24 * 0.0116 = $0.2784

Rekognition:
- $0.001 per API call \ about 2 API calls per Customer
- Total cost = 1000 * $0.001 *2 = $2

Total cost per day = $2.9532