SpadinaBus Offers Engine Technical Documentation

1. Change Log

#	Date	Author	Summary
1	August 11 2020	Tom Pedron	Initial document skeleton and contents

2. Project Overview

2.1 Summary

The purpose of this project is to construct a cloud-oriented offer serving engine that is highly available, scalable, configurable and maintainable. An endpoint must be exposed for applications and websites to integrate with to retrieve offers for logged in customers.

Offers are assigned to customers using other marketing software that produces an offer-customer whitelist CSV file. This whitelist maps an offer_id to a customer_id along with a score used to prioritize the best offers for each customer. This CSV is loaded into a highly available datastore (AWS DynamoDB) for querying.

Offers are selected based on score and various configurations defining limits on viewing of offers by a customer. When a best offer is selected, a view is logged for it to the offer datastore. The offer data is then returned to the client who will present the best offer based on the offer_id.

2.2 Offer Selection Configuration

The system returns a single best offer for a customer. As mentioned in 2.1, there are several configurations in place to define how many times an offer can be shown to a user before suppressing it in favor of the next best offer. The purpose is to prevent marketing fatigue in customers.

These configurations are defined in `config.json` which is loaded on execution of the GetOffer request. Below is an example of `config.json`:

```
{
  "offer_exclusion": {
    "expired_offers": {
      "enabled": true,
      "max_age_days": 14
    },
    "views_recency_window": {
      "enabled": true,
      "num_hours": 48,
      "max_views": 6
    },
    "views_all_time": {
      "enabled": true,
      "max_views": 10
```

```
    }
  },
  "default_offer": {
      "offer_id": "DefaultOffer1"
  }
}
```

2.2.1 Offer Expiration

You can exclude offers loaded into DynamoDB for a user that were loaded prior to a certain date since they are old and out of date.

`offer_exclusion.expired_offers.enabled`

- Description: Sets whether we exclude older offers from consideration when selecting offers.
- Type: boolean
- Values: `true` or `false`

`Offer_exclusion.expired_offers.max_age_days`

- Description: Sets the limit date timestamp we use to filter out older offers.
    - The limit date is calculated as current_date - max_age_days. This value is compared to the LoadedAt value for offers in DynamoDb.
    - Note that this value is ignored when enabled is false.
- Type: integer
- Values: `values >= 1`
- Example: 21
    - Offers loaded into DynamoDb more than 3 weeks ago will be excluded from consideration.

2.2.2 Recency Window Views

You can exclude offers that have been seen a certain amount of times within a configured recency window.

`offer_exclusion.views_recency_window.enabled`

- Description: Sets whether we exclude offers from consideration that have been seen a certain amount of times in the configured recency window.
- Type: boolean
- Values: `true` or `false`

`offer_exclusion.views_recency_window.num_hours`

- Description: Used to calculate the limit date timestamp we use to determine which offer views occurred during the configured recency window.

    - Note that this value is ignored when enabled is false.
- Type: integer
- Values: `values >= 1`
- Example: 48
    - The system will calculate how many of the offer's views occurred in the last 2 days. If greater than the max_views value then the offer is excluded from consideration.

`offer_exclusion.views_recency_window.max_views`

- Description: Sets the maximum number of views an offer may have within the configured recency window.
    - If the total number of views in the recency window is equal to or greater than the max_views value, then the offer is excluded from selection.
    - Note that this value is ignored when enabled is false.
- Type: integer
- Values: `values >= 1`
- Example: 3
    - Offers with 3 or more views in the recency window will be excluded from consideration.

2.2.3 All Time Views

You can exclude offers that have been seen a certain amount of times, regardless of when they occurred.

`offer_exclusion.views_all_time.enabled`

- Description: Sets whether we exclude offers from consideration that have been seen a certain amount of times in the configured recency window.
- Type: boolean
- Values: `true` or `false`

`offer_exclusion.views_all_time.max_views`

- Description: Sets the maximum number of views an offer may have.
    - If the total number of views is equal to or greater than the max_views value, then the offer is excluded from selection.
    - Note that this value is ignored when enabled is false.
- Type: integer
- Values: `values >= 1`
- Example: 10
    - Offers with 10 or more views will be excluded from consideration.

2.2.4 Default Offer

When no best offer is determined for a customer (either they have no offers loaded for them in DynamoDB or they have all been excluded due to the expiration or max views configurations), the offer_id defined here will be returned.

`default_offer.offer_id`

- Description: They offer_id to be returned when a default offer is selected for the customer.
- Type: string
- Example: “DefaultOffer123”

2.3 Offer Selection Logic

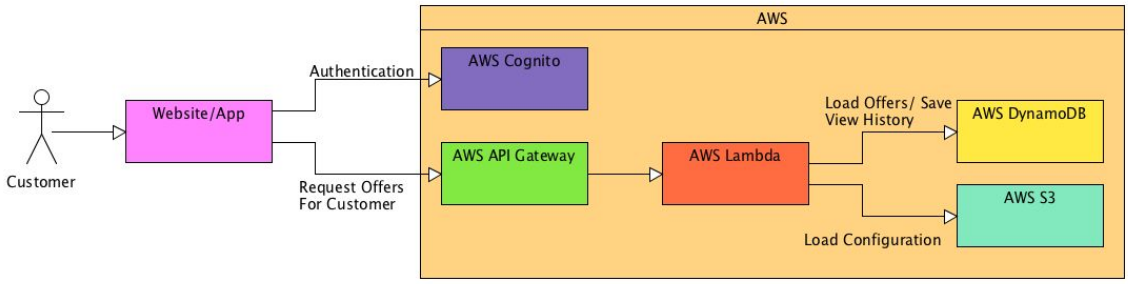
Below is a detailed summary of the logic used to select a best offer for a customer.

1. Retrieve all offers in DynamoDb for the customer.
2. If 0 offers returned than return `default_offer.offer_id`
3. If `offer_exclusion.expired_offers.enabled` = True
    - Filter out offers that have a LoadedAt timestamp > (current time - `offer_exclusion.expired_offers.max_age_days`)
        - I want to keep looking into how to do this best via the Dynamo query but for now this is implemented in the code
4. Sort remaining offers by highest score
5. Set current_best_offer = highest scored offer
6. Validate current_best_offer for exclusion rules. There are 2 rules right now based on the number of times the customer has seen that offer .
    - If `offer_exclusion.views_recency_window.enabled` = True
        - Count the number of views (these are timestamps) for current_best_offer that are greater than (current time - offer_exclusion.views_recency_window.num_hours)
        - Exclude current_best_offer if the count from above >= `offer_exclusion.views_recency_window.max_views`
    - If `offer_exclusion.views_all_time.enabled` = True
        - Count the number of views for the current_best_offer
        - Exclude current_best_offer if count from above >= `offer_exclusion.views_all_time.max_views`
    - If any rules above fail then set current_best_offer to next offer in the list and try again
7. At this point we have determined our best_offer or excluded all offers, in which case we return default_offer.offer_id.
8. Return the best offer.

2.3 Architecture

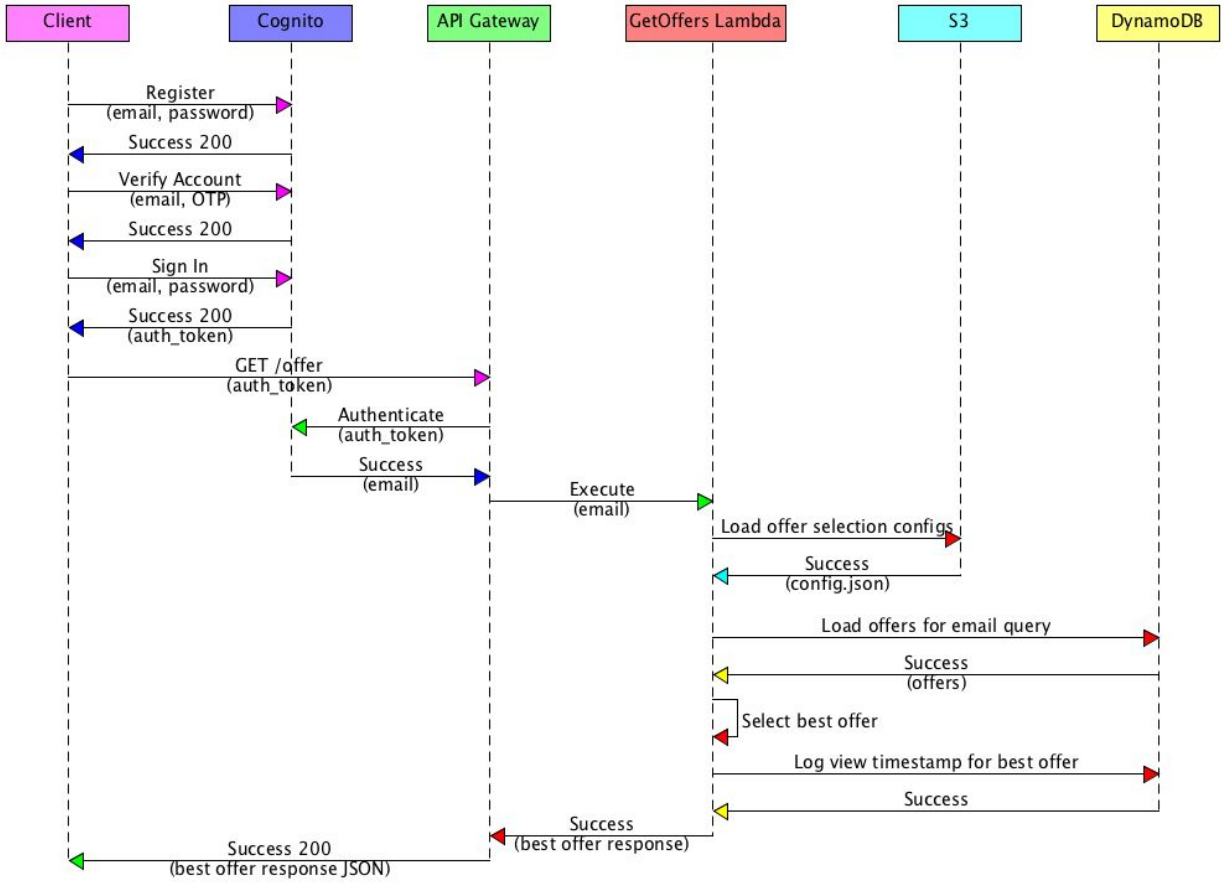
The project leverages a serverless cloud based architecture to allow for a highly scalable and maintainable system.

Below is high level architecture diagram showing the connections between components:



2.4 Sequence Diagram

Below is a sequence diagram showing the intended usage of the system from authentication to retrieving offers.



2.5 Offer Data Structure

The system leverages a single AWS DynamoDB table. Each item in the table represents a single offer mapped to a single user.

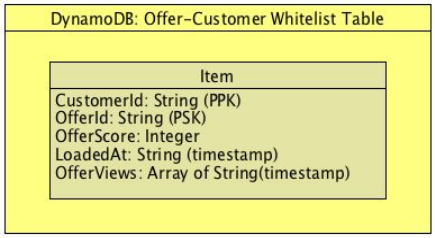
2.5.1 Fields

Below are the fields contained in each item:

- CutomerId
  - Description: The email of the customer this offer is associated with. It must match the customer's account username in AWS Cognito.
  - Type: String
  - Example: `"tom@spadinabus.com"`
- OfferId
  - Description: The id of the offer associated to the customer. Many customers can be mapped to the same OfferId through separate items in the table.
  - Type: String
  - Example: `"OfferA"`
- OfferScore
  - Description: The score of the offer for the customer.
  - Type: Integer
  - Example: `90`
- LoadedAt
  - Description: The timestamp of when the offer was uploaded to DynamoDB.
  - Type: Timestamp String
  - Format: `"%m/%d/%Y, %H:%M:%S"`
  - Example: `"08/11/2020, 14:27:23"`
- OfferViews
  - Description: A list of timestamps representing each view of the offer by the customer. The array is initialized as empty and appended to each time the offer is returned as the best offer.
  - Type: Array of Timestamp Strings
  - Format: Each string in the array is in `"%m/%d/%Y, %H:%M:%S"` format
  - Example: `[ "08/11/2020, 14:27:23" ]`

2.5.2 ER Diagram

Below is an diagram of the Offer Customer Whitelist table in DynamoDB:



2.5.3 Example

Below is an example of what a single item looks like in the table:


```
{
  "CustomerId": "tom@spadinabus.com",
  "LoadedAt": "08/01/2020, 00:00:00",
  "OfferId": "OfferB",
  "OfferScore": 90,
  "OfferViews": [
    "08/11/2020, 14:27:23",
    "08/11/2020, 14:27:27",
    "08/11/2020, 14:30:51",
    "08/11/2020, 14:31:07",
    "08/11/2020, 14:31:07"
  ]
}
```

3. AWS Services

Since this project architecture is a cloud-based serverless system, many AWS services are leveraged. Below is a summary of each service used along with details around how.

3.1 API Gateway

API Gateway exposes the GET /offers endpoint. It allows AWS to accept the HTTP request, authenticate using the existing Cognito user pool and pass along with event context to the GetOffers Lambda function. It receives a JSON response from the Lambda function and then returns it as an HTTP response to the client along with a 200 OK status code.

Notes:

- CORS must be enabled on the offer resource.
- The Lambda Integration Proxy must be enabled.
- The endpoint must be set as “Edge Optimized”
- An authorizer must be created with a connection to the AWS Cognito User Pool.
- The GET endpoint must have its authorizer set to the authorizer mentioned above.
- The API Source key must be set as HEADER.

3.2 Lambda

Lambda offers the ability to run code snippets in the cloud without the need for provisioning, servers or any other work traditionally required to deploy endpoints. With Lambda, you deploy a piece of code and trigger it to run programmatically.

There are 2 Lambda functions:

- GetOffers Lambda: triggered by API Gateway to select the best offer for a customer.
- UploadOfferData Lambda: triggered by a new file upload to the Whitelist CSV bucket and inserts new items into the Offer-Customer Whitelist table in DynamoDB.

3.2.1 GetOffers Lambda

The Python 3.7 code provided (see `get_offer_lambda.py`) that implements the GetOffers logic described in this document must be deployed as is with the following exceptions:

- There are default configuration values set that are overridden by loading config.json from the S3 bucket. If you wish to change the defaults, please modify the JSON defined near the top of the code.
- The Offer-Customer WhiteList DynamoDB table name is defined in the code. Please change this to the name of your table.

- The S3 Bucket that contains the config.json file is defined in the code. Please change this to the name of your bucket.

Notes:

- The Lambda function must have a IAM Role defined that contains the AWSLambdaBasicExecutionRole and the appropriate permissions for accessing S3 and DynamoDB.
- You must enable Logs and Metrics on the Lambda function to allow for logs to be saved each time the function is executed. They will be accessible via AWS CloudWatch.

3.2.2 UploadOfferData Lambda

The Python 3.7 code provided (see `upload_offers_lambda.py`) that implements the UploadOffers logic described in this document must be deployed as is with the following exceptions:

- The Offer-Customer WhiteList DynamoDB table name is defined in the code. Please change this to the name of your table.
- The S3 Bucket that contains the config.json file is defined in the code. Please change this to the name of your bucket.

Notes:

- The Lambda function must have a IAM Role defined that contains the AWSLambdaBasicExecutionRole and the appropriate permissions for accessing S3 and DynamoDB.
- An S3 trigger must be set up so it executes whenever a new .csv file is uploaded to the Whitelist Bucket.
- You must enable Logs and Metrics on the Lambda function to allow for logs to be saved each time the function is executed. They will be accessible via AWS CloudWatch.

3.3 DynamoDB

This project requires a single DynamoDB table called the Offer-Customer Whitelist table.

The UploadOfferData Lambda function uploads items to the table that map offers to customer emails.

The GetOffer Lambda function queries this table for offers mapped to a customer_id and also updates individual items in the table with view information when offers are selected as best offers.

3.4 S3

There are 2 S3 buckets required for this project.

3.4.1 GetOfferConfigs Bucket

Stores the `config.json` file used to configure the GetOffers Lamba function's offer selection logic.

3.4.2 Whitelist CSV Bucket

Stores all the `.csv` files containing mappings of offers to customers. This bucket is configured to run a Lambda function each time a CSV file is uploaded which loads each row in the CSV as an item in the Offer-Customer Whitelist table in DynamoDB.

3.5 Cognito

Cognito is an AWS-manager service to handle User registration, sign in and authorization. Cognito is required for the following reasons:

- To secure the GetOffer endpoint to only registered users and issue an authentication token.
- To identify what user we are requesting offers for by mapping an issued authentication token to the user's email. This allows us to query for offers associated with this email in the Offer-Customer whitelist table in DynamoDB.

3.6 IAM

3.6.1 GetOffers Lambda Role

An IAM Role must be created for the GetOffers Lambda function with the `AWSLambdaBasicExecutionRole` permission.

This role must also have the following permissions defined in an inline Policy associated to allow it to access DynamoDB and S3:

- DynamoDB
  - `"dynamodb:PutItem"`
  - `"dynamodb:GetItem"`
  - `"dynamodb:Query"`
  - `"dynamodb:UpdateItem"`
- S3
  - `"s3:GetObject"`

The JSON that defines this inline policy is provided but will need to be updated with:

- The ARN of your DynamoDB Customer-Offer whitelist table
- The name of your S3 bucket that contains config.json

3.6.2 UploadOfferData Lambda Role

An IAM Role must be created for the UploadOfferData Lambda function with the `AWSLambdaBasicExecutionRole` permission.

This role must also have the following permissions defined in an inline Policy associated to allow it to access DynamoDB and S3:

- DynamoDB
  - `"dynamodb:PutItem"`
- S3
  - `"s3:GetObject"`

The JSON that defines this inline policy is provided but will need to be updated with:

- The ARN of your DynamoDB Customer-Offer whitelist table
- The name of your S3 bucket that contains the whitelist csv

3.7 CloudWatch

Cloudwatch allows you to access execution logs of both Lambda functions which is important for troubleshooting and data tracing.

3.8 Amplify

Amplify is used to host the demo website that integrates with Cognito and makes GET calls for offers. It allows for a simple registration flow, logging in and request offers.

4. API Documentation

Please note that the GetOffers API endpoint JSON response conforms to the JSON API specification, please see https://jsonapi.org

4.1 Request

The HTTP request is of type GET. Below is a sample request (note that a Authorization header containing the IdToken returned by AWS Cognito is also required):

```
GET <API Gateway Offer Resource Invoke URL>/<stage>/offer
```

Here is an example CURL of the request

```
curl --location --request GET
'https://sbiu9g0d4a.execute-api.us-east-1.amazonaws.com/dev/offer' \
--header 'Authorization:
eyJraWQiOiJMZ1N3MkRrV3FTcHB5ZHk2elB3U2tqcTVnRzI4NEg0M3Vlb2d6TkFyVUdjPSIsImF
sZyI6IlJTMjU2In0.eyJzdWIiOiJmYTUzYTg4My1lYWY3LTQzYTAtYjYzZi1jMzgxOWRmNzJmMj
UiLCJhdWQiOiI2cDJ1b2xvNW03ZXNiNzEwZTUwNnVlajNjaSIsImVtYWlsX3ZlcmlmaWVkIjp0c
nVlLCJldmVudF9pZCI6IjgwMzQ3ZjBjLTBhZGEtNDMzMy1hMTEzLTg5YTQ3OGZhMDIyOSIsInRv
a2VuX3VzZSI6ImlkIiwiYXV0aF90aW1lIjoxNTk3MTA5MTY1LCJpc3MiOiJodHRwczpcL1wvY29
nbml0by1pZHAudXMtZWFzdC0xLmFtYXpvbmF3cy5jb21cL3VzLWVhc3QtMV83dUw1VzdaY0siLC
Jjb2duaXRvOnVzZXJuYW1lIjoidG9tQHNwYWRpbmFidXMuY29tIiwiZXhwIjoxNTk3MTEyNzY1L
CJpYXQiOjE1OTcxMDkxNjUsImVtYWlsIjoidG9tQHNwYWRpbmFidXMuY29tIn0.G8DL9haRdT5Q
6dyNf06a2mqLvhmBhG1r2Zdb2uB1HUbOnoAARXAYBZfooMp8CyjAgDBRk2c3ODXKc32XMpb8xDQ
aYWBKSe1gwUv5sV5SZlSe1f5p_I_uDNzeLWtLyzskl_bHHbHLmmTSCD7dvVJIPU6JAX_WKBCPHJ
460DnpYotrGsNNtswuS2NRCpp8UQKpXwD-gPtu3b0FTHppF5pXCNF0HSNu5bEpl6wHt-f4PkZao
VY3xnV4MaSwCWyImEhym1V9PWwc65BUYt7FmAGyUu4S43jfkPkzWlkd0VZoRGhivnrODvYKHc5g
GgLd4JdQeU18O1SkOm8qeDNESu02jQ' \
--header 'Content-Type: application/json'
```

4.2 Responses

4.2.1 200 OK

Schema

- data (array of object)
  - id (string)

  - `type` (string)
  - `attributes` (object)
    - `score` (integer or null)
    - `rank` (integer or null)
    - `num_views` (integer)
    - `is_default_offer` (boolean)

Notes

- When the default offer is returned, the score and rank are set to `null` and `is_default_offer` is set to true.

Example


```
{
    "data": [
        {
            "id": "OfferB",
            "type": "Offer",
            "attributes": {
                "score": 90,
                "rank": 1,
                "num_views": 2,
                "is_default_offer": false
            }
        }
    ]
}
```


4.2.1 401 Unauthorized

Schema

- `message` (string)

Example


```
{
   "message": "Unauthorized"
}
```

5. Project Setup Guide

1. **Setup the AWS Cognito User Pool**
    a. Navigate to Cognito on the AWS console.
    b. Click on Create a User Pool
    c. Provide a name and click Create.
    d. Note the pool_id returned as it is required for a website to integrate with Cognito for Registration and Sign in.
    e. Navigate to App Clients on the left hand side
    f. Click on Add an app client
    g. Provide a name, uncheck “generate client secret” and click create
    h. Note the appclient_id as it is required for a website to integrate with Cognito for Registration and Sign in.

2. **Create DynamoDB table**
    a. Navigate to DynamoDB on the AWS console.
    b. Click create
    c. Provide a name
    d. Set the Partition Key to “CustomerId” as a String
    e. Set the Sort Key to “OfferId” as a String
    f. Leave the default settings and click Create.
    g. Note the ARN on the summary page as it is required to be set in the policies for the Lambda functions that read and write to/from this table.

3. **Create Config S3 Bucket**
    a. Navigate to S3 in the AWS console
    b. Create a private S3 bucket and provide a name such as GetOffersConfig.
    c. Upload `config.json` file

4. **Create GetOffers IAM Role**
    a. Navigate to IAM in the AWS console and select Roles
    b. Click Create New Role and select Lambda
    c. Add AWSLambdaBasicExecutionRole to the Role
    d. Click next and add a name.
    e. Click create
    f. Add an inline policy to the Role (the JSON for this is provided, see inline_policy_get_offers_lambda.json)
        i. Edit the Dynamo permissions with the correct ARN from step 2g.
        ii. Edit the S3 permissions with the correct bucket name for the Configs S3 bucket from step 3b.

5. **Create GetOffers Lambda function**
    a. Navigate to Lambda in the AWS console
    b. Create a new Lambda function
    c. Provide a name and select Python 3.7 as the runtime
    d. Select the GetOffers IAM role from step 4 as the role.
    e. Click create
    f. Paste in the provided python code (see get_offer_lambda.py) for this function and update the following global variables near the top of the code:
        i. offer_whitelist_table_name: set this string to the name of the DynamoDB table created in step 2
        ii. configs_bucket_name: set this string to the name of the Configs bucket created in step 3.
    g. Click save.

6. **Create OfferWhiteListCSV S3 bucket**
    a. Navigate to S3 in the AWS console
    b. Create a private S3 bucket and provide a name such as OfferWhitelists.

7. **Create UploadOfferData IAM role**
    a. Navigate to IAM in the AWS console and select Roles
    b. Click Create New Role and select Lambda
    c. Add AWSLambdaBasicExecutionRole to the Role
    d. Click next and add a name.
    e. Click create
    f. Add an inline policy to the Role (the JSON for this is provided, see inline_policy_upload_offers_lambda.json)
        i. Edit the Dynamo permissions with the correct ARN from step 2g.
        ii. Edit the S3 permissions with the correct bucket name for the OfferWhitelists S3 bucket from step 6b.

8. **Create UploadOfferData Lambda Function & Trigger**
    a. Navigate to Lambda in the AWS console
    b. Create a new Lambda function
    c. Provide a name and select Python 3.7 as the runtime
    d. Select the GetOffers IAM role from step 7 as the role.
    e. Click create
    f. Paste in the provided python code (see upload_offers_lambda.py) for this function and update the following global variables near the top of the code:
        i. offer_whitelist_table_name: set this string to the name of the DynamoDB table created in step 2.
        ii. whitelist_bucket_name: set this string to the name of the Offer Whitelists bucket created in step 6.

g. Click save.
h. Click on Add Trigger near the top of the page
    i. Select S3 Trigger type
    ii. Select the bucket name from step 6
    iii. Set EventType = ObjectCreated
    iv. Suffix = .csv
    v. Check Enable trigger
    vi. Click Add

9. Upload whitelist csv to OfferWhitelist S3 bucket

a. Upload your whitelist csv file to the bucket. This will trigger the Lambda created in step 8
b. Navigate to the Items view for your DynamoDB Offer Whitelist table from step 2 and you will see the items from your whitelist csv uploaded to the table.

10. Deploy API Gateway

a. Navigate to API Gateway in the AWS Console
b. Click Create API
c. Provide a name such as GetOffersApi
d. Select Edge Optimized and click Create.
e. Create Authorizer
    i. Enter a name such as GetOffersAuthorizer
    ii. Select Cognito as the type
    iii. Set Region to be the same region as the one that the Cognito User Pool from Step 1 was created in.
    iv. Set Cognito User Pool to the name of the User pool (step 1c)
    v. Set Token source to Authorization
f. Create Offers resource
    i. Under Resources select Create Resource
    ii. Enter offer as the name and ensure the resource path is the same
    iii. Ensure you select Enable API Gateway CORS
    iv. Click create
g. Create GET offer method
    i. Under the new created /offer resource, click Create Method
    ii. Select GET and click the checkmark
    iii. Set Integration Type to Lambda Function
    iv. Set the region to the same one as the GetOffers Lambda Function from Step 5
    v. Set Lambda Function to the name of the GetOffers Lambda function from step 5
    vi. Click Save and click ok if prompted to provide permission to API Gateway.

vii. Click on the method request card, click edit (the pencil icon) next to Authorization and select the Cognito User Pool Authorizer from step 10e.

h. Deploy the API
   i. Select Deploy API
   ii. Enter a stage (dev/staging/prod).
   iii. Click deploy
   iv. Note the Invoke URL provided on successful deployment.

11. Deploy test website

a. Clone the provided test website respiratory
b. Update config values in `js/config.js`
   i. Cognito User Pool Id: Set this to the value from step 1d
   ii. Cognito Application Client Id: Set this to the value from 1h.
   iii. AWS Region: Set this to the same region used in step 1.
   iv. GetOffers Invoke URL: Set this to the value from step 10h
c. Commit the updated test website repo to a Github repository.
d. Deploy to AWS Amplify
   i. Navigate to AWS Amplify in the AWS console.
   ii. Click Get Started and then connect your Github repository from Step 11c.
   iii. Select your repository and branch from step 11c.
   iv. Use the default settings provided and click next.
   v. Click Save and Deploy
e. Your website will deploy in a few minutes and you will be provided with a URL.
f. Test
   i. Navigate to the URL from Step 11e in your browser
   ii. Register with an email that has offers mapped to it in the Offer Whitelist table in DynamoDB. You will be emailed an OTP code to verify your account.
   iii. Sign In with your username and password.
   iv. Navigate to the Get Offers page and click the request button.
   v. Enjoy.