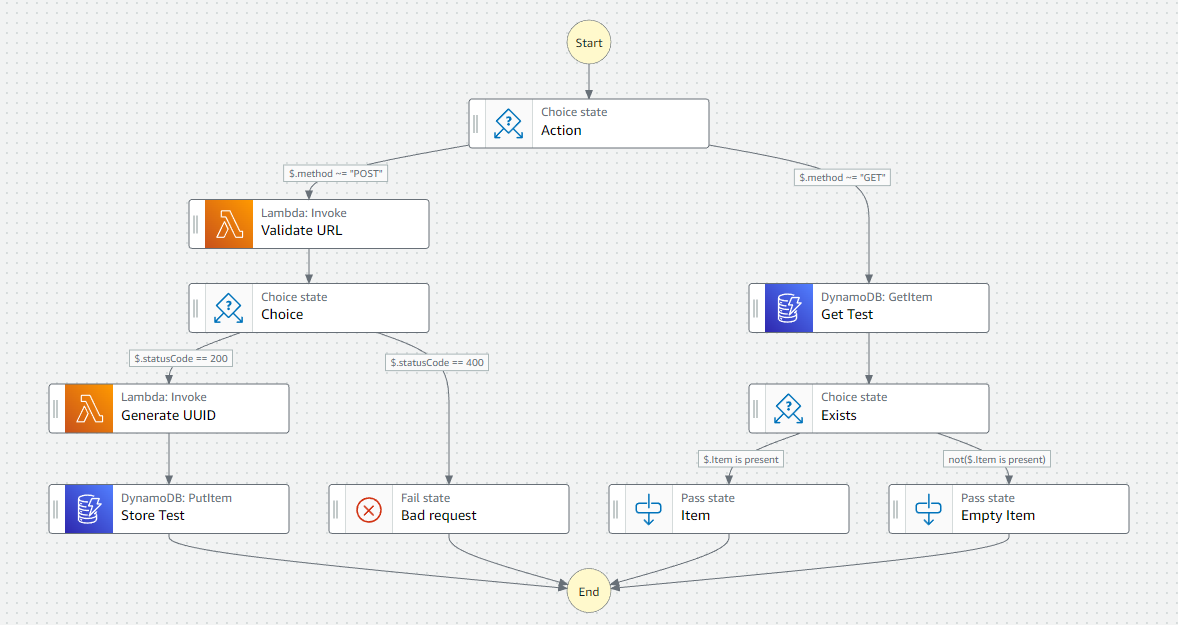
|  |
| --- |
| Christos Papastamos csd4569 HY452 Services Technologies Assignment 1 |

**1. Task A**

**1.1 Structure**

For the first task I created a State Machine based on the description given in the assignment which can be seen below:



When a request arrives to the Step Function the choice state checks its method.

If the request is a **POST** request, the choice state forwards it to the Validate URL lambda function which checks all of the request’s fields. If the fields are valid, the next choice state forwards it to the Generate UUID lambda function where a random number is generated for each new test entry. Finally, DynamoDB stores the test to the table and returns SUCCESS.

If the request is a **GET** request, the choice state forwards the request to DynamoDB to get the test with the requested ID. If the test exists an item containing all the parameters of the test is passed to the State Functions output. If the test does not exist an empty item is generated and passed to the output.

**1.2 Code**

The code of the **Validate URL** lambda function can be seen below:

import json

accepted\_methods=[ #a list of accepted methods

    "GET",

    "HEAD",

    "POST",

    "PUT",

    "DELETE",

    "CONNECT",

    "OPTIONS",

    "TRACE",

    "PATCH"

    ]

def error\_handler(): #a basic error handler

    return {

        "statusCode" : 400

    }

def lambda\_handler(event, context):

    if "url" in event:#check if the url is in the event

        url = event["url"]

    else:

        return error\_handler()

    if "expectedResult" in event: #check if the expected result is in the event

        expectedResult = event["expectedResult"]

    else:

        return error\_handler()

    if "testMethod" in event and (event["testMethod"] in accepted\_methods): #check if the test method is in the event and if it is an accepted method

        testMethod = event["testMethod"]

    else:

        return error\_handler()

    return { #return a success result

        "statusCode" : 200,

        "item":{

            "url" : url ,

            "expectedResult" : expectedResult ,

            "testMethod" : testMethod

        }

    }

The code for the **Generate UUID** lambda function can be seen below:

import json

import random

def lambda\_handler(event, context):

    item = event["item"]

    #check the request's fields

    if not ("url" in item):

        return {

            "statusCode": 400,

            "message": "url not included in json input"

        }

    elif not ("testMethod" in item):

        return {

            "statusCode": 400,

            "message": "testMethod not included in json input"

        }

    elif not ("expectedResult" in item):

        return {

            "statusCode": 400,

            "message": "expectedResult not included in json input"

        }

    return {

        "testID":           str(random.randint(0, 99999)), #generate a random testID

        "testMethod":       item["testMethod"],

        "url":              item["url"],

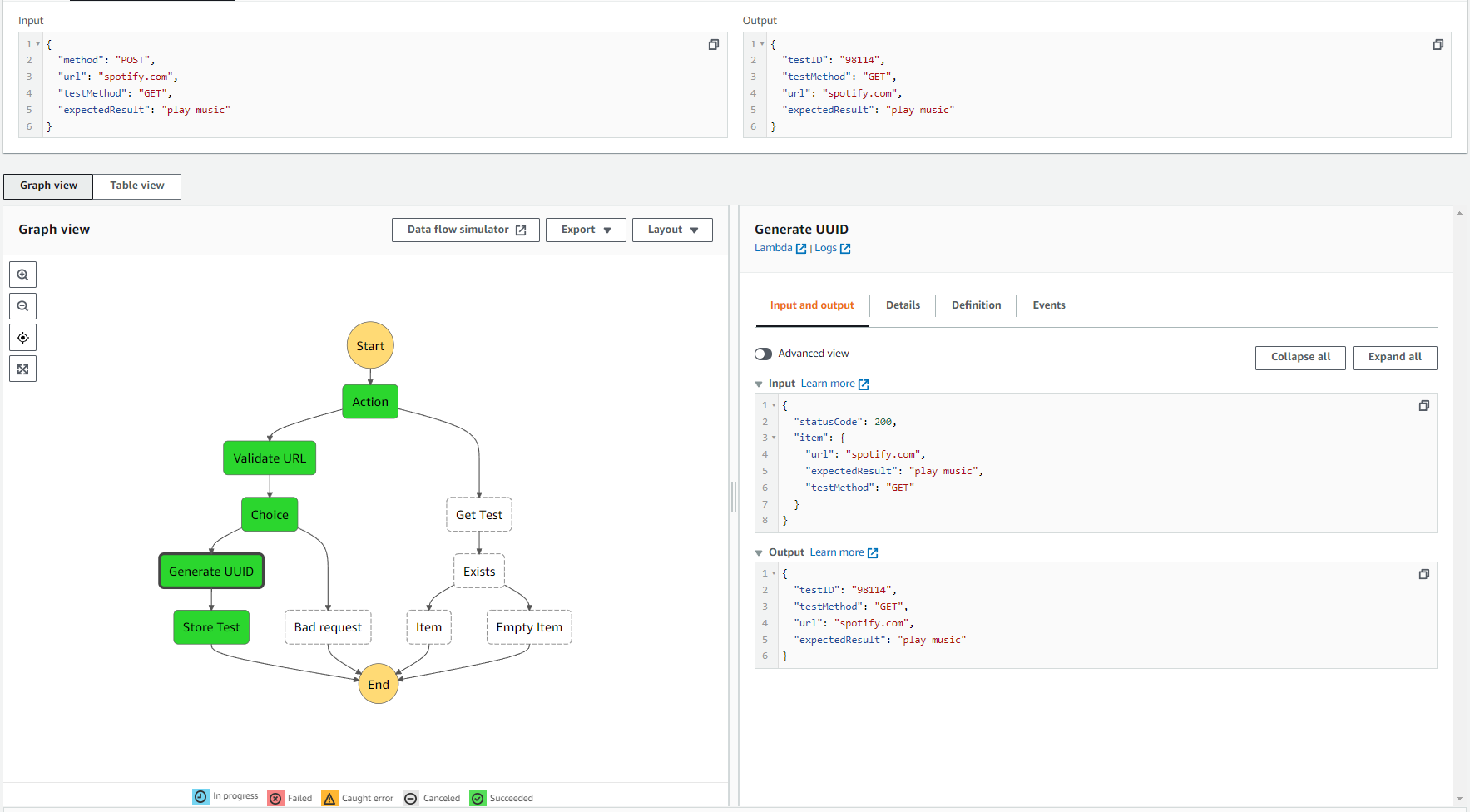
        "expectedResult":   item["expectedResult"]

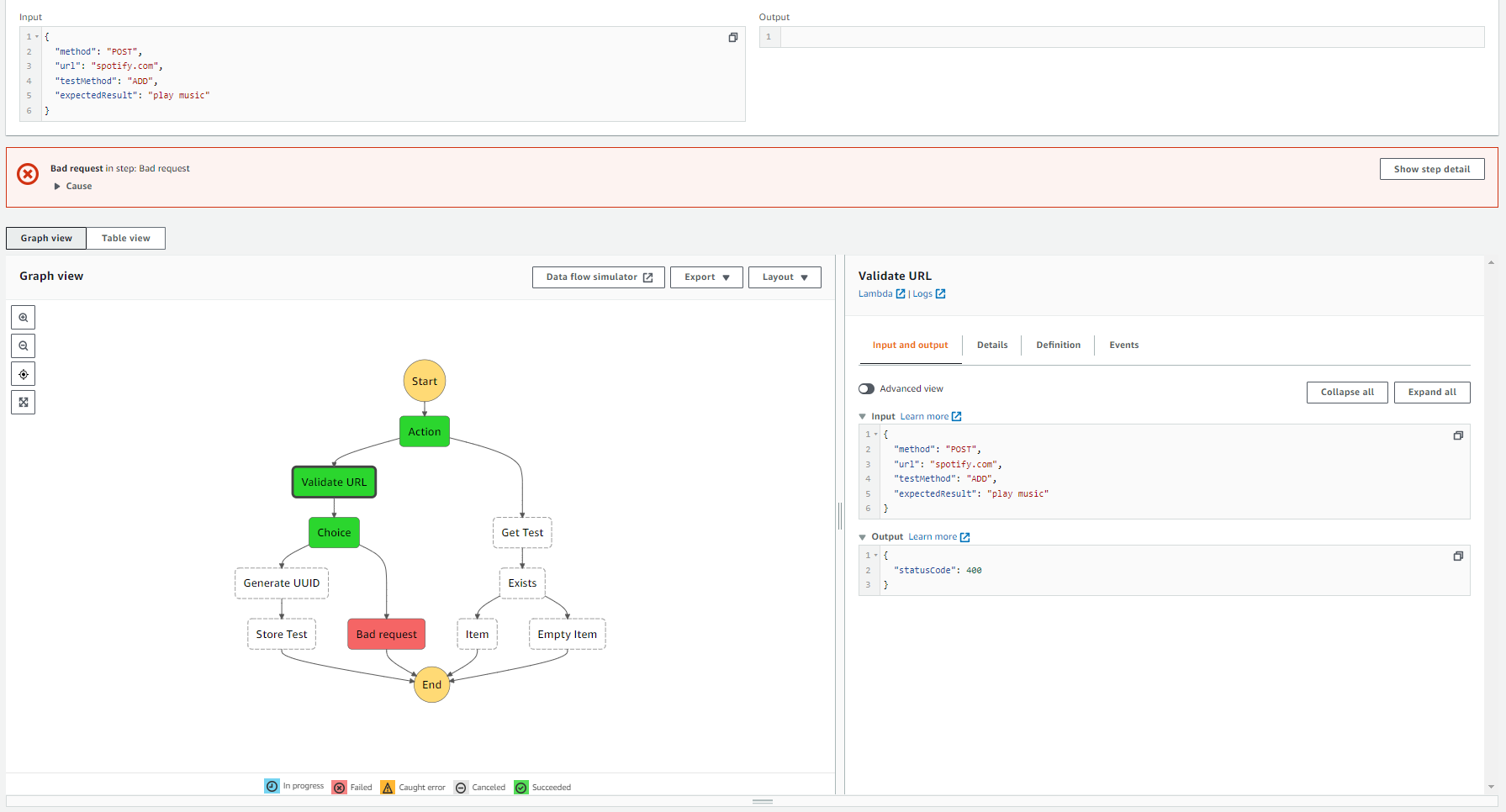
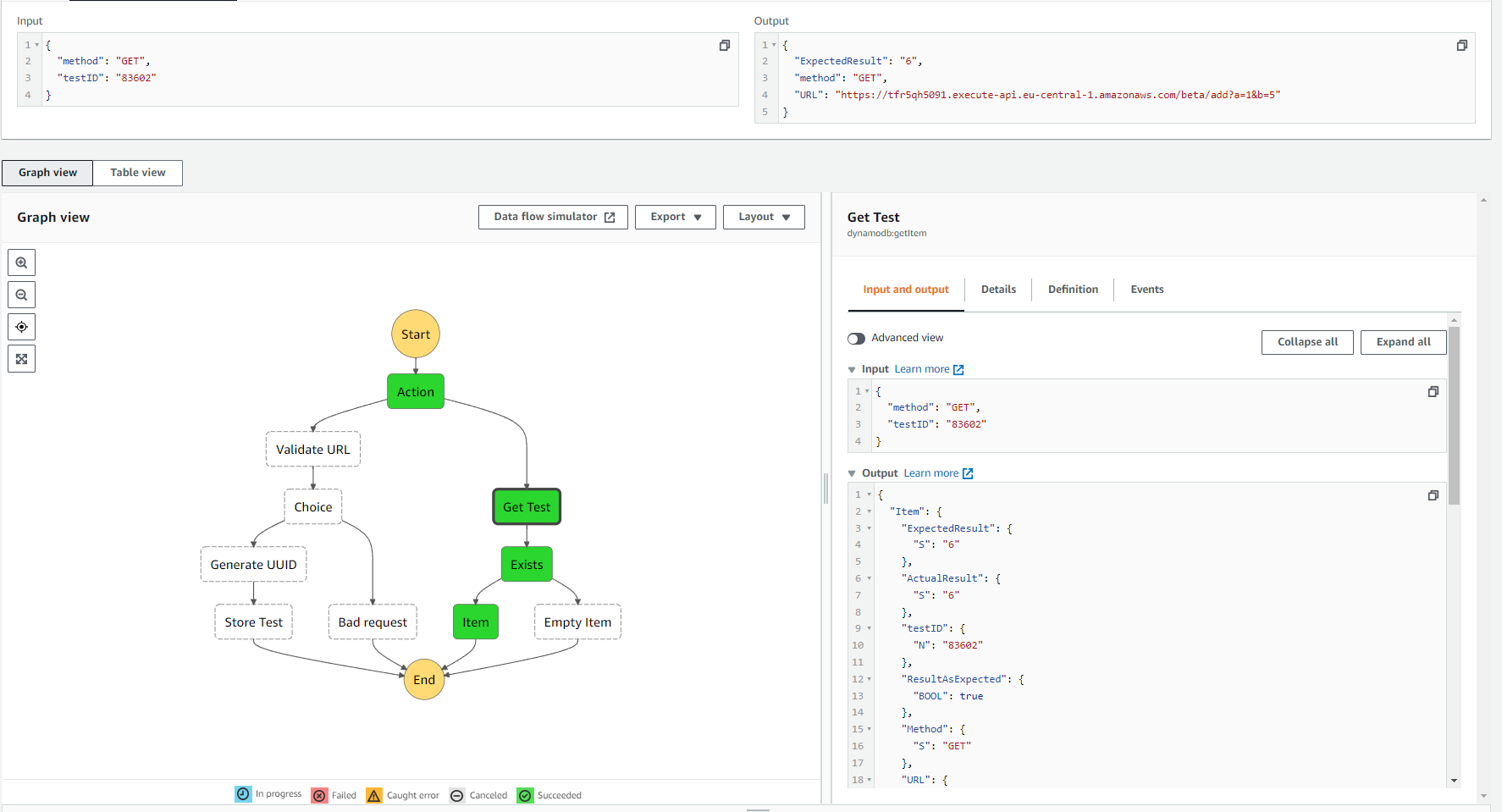
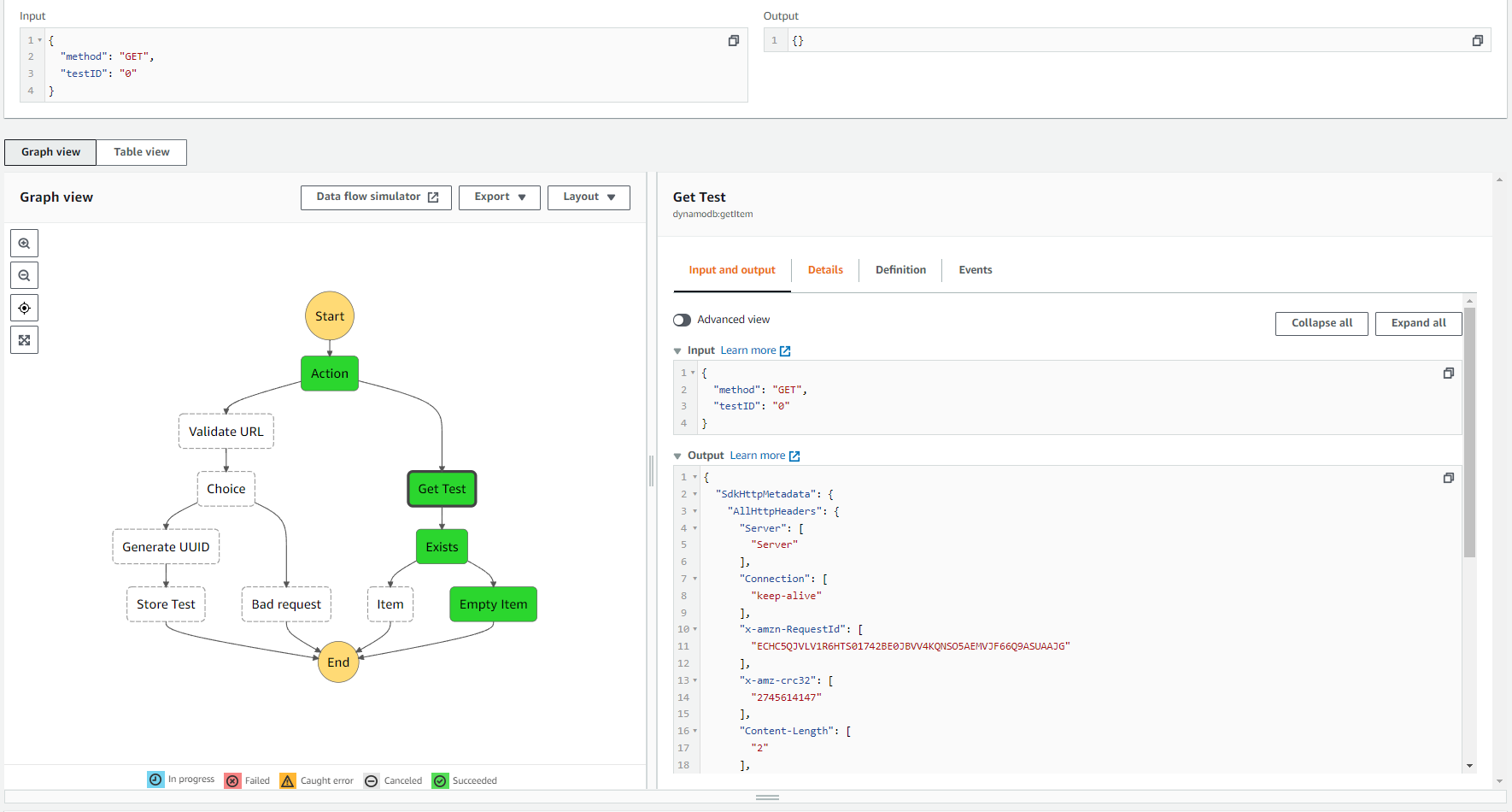
    }

**1.3 Execution and Testing**

The expected input for this state machine can follow one of the following formats:

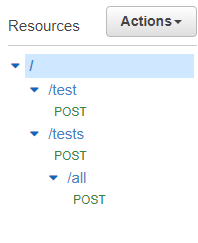
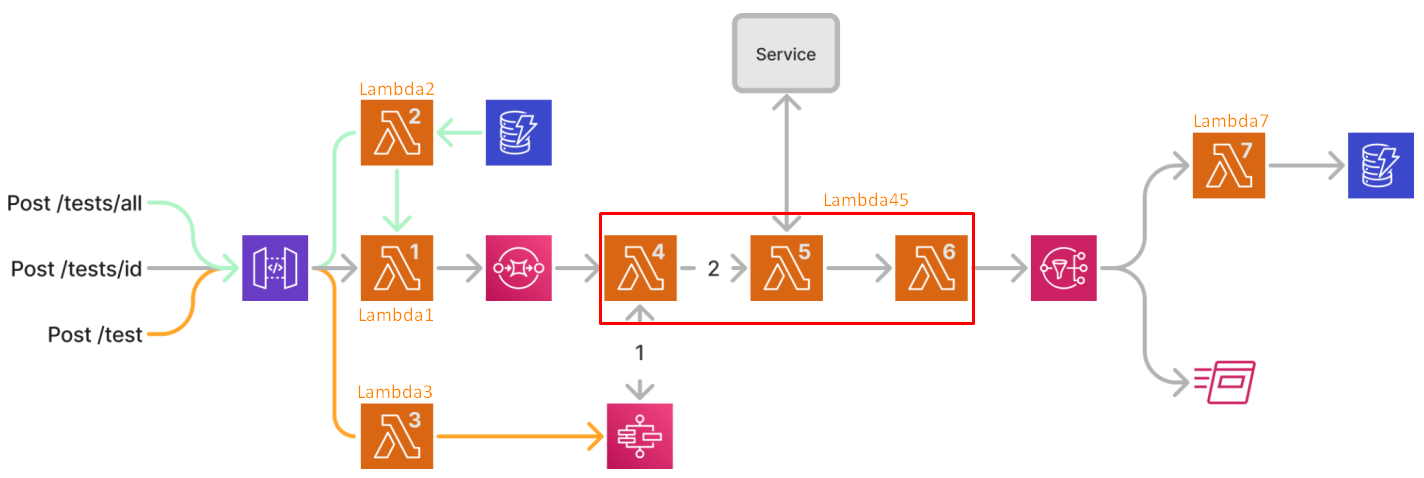
|  |  |
| --- | --- |
| Add a new test | Get a test from a testID |
| {  "method": “POST”,  "url": <String>,  "testMethod": <HTTP method>,  "expectedResult": <String>  } | {  "method" : "GET",  "testID" : <Int>  } |

* For the **POST** part of the Step Function a successful test run can be seen below:  On the right of the Step Function we can see the input and output of the Generate UUID lambda function. This function creates a random test ID for each new entry.

* For the **POST** part of the Step Function an unsuccessful test run can be seen below: We can see that the testMethod field has the value “ADD” which is an invalid HTTP method, so the validate URL lambda function returned a status code of 400 which results in a State Machine failiure
* For the **GET** part of the Step Function a successful test run can be seen below (assuming ID 83602 is in the table):  Here we can see that the Step Function outputted the requested fields of the test successfully.
* For the **GET** part of the Step Function an unsuccessful test run can be seen below (ID 0 is not in the table): In this case the output of the Get Test DynamoDB Getitem returned a json which didn’t contain the “item” field in it. That means that the test with the given ID is not in the table so an empty item is created for the output of the Step Function

**2. Task B**

**2.1 Stracture**

For the second task of the assignment I implemented the desired tool by making one change in the stracture.I merged the functionality of the lambda4-6 fucnctions to a single lambda function called Lambda45

Lets take if from the start. For the API gateway I created the resources and methods as seen on the screenshot on the right.

* /test?URL=<u>&Method=<m>&ExpectedResult=<e> (POST) : post a new service to test to the tool
* /tests?id=<ID> (POST) : test a test by ID
* /tests/all (POST) : Test all IDs currently in the table

In order to get the querry part parameters I used a mapping template in the Intergration Request ass seen below

|  |  |
| --- | --- |
| /test?URL=<u>&Method=<m>&ExpectedResult=<e> | /tests?id=<ID> |
|  |  |

Each resource is connected to their respective lambda function. Lambda functions will be listed in the code section.

For the Simple Queue Service I created a simple queue and added lambda45 as a trigger.

On the proccess of making lambda45 send a request and wait for a response from the State Machines I realized that Standard State Machines cannot run synchronously and for that I had to create a new State Machine of type Express with the exact same json code snippet. After that I was able to call the State Machine from lambda45 synchronously using the boto3 library.

For the communication of lamda2 and 7 with the DynamoDB I used the boto3 library as recommended.

Finally for the Simple Notification Service I created a standard topic (since I didn’t want to add another SQS to the mix) and added a subscription to lambda7 and one to my academic email.

**2.1 Code**

A common found function in the code is print\_debug which Is a extremely simple debug function I implemented.

* **Lamda1**

import json

import boto3

sqs = boto3.resource('sqs')

queue = sqs.get\_queue\_by\_name(QueueName="Ask1\_2\_Queue1")

debug = 1

def print\_debug(message): #simple debug function

    if debug:

        print("DEBUG: "+message)

def lambda\_handler(event, context):

    print\_debug("Recieved an event, begining execution ("+str(event)+")")

    if type(event) is dict:

        if "id" in event: #if the event is a dict and has an id, it came from the API

            print\_debug("Request came for single id: ("+str(event["id"])+")")

            response = queue.send\_message(MessageBody=event["id"]) #send the id to the queue

            print\_debug("Sending id "+str(event["id"])+" to SQS")

        else:

            print("ERROR: id not included in the input json object")

            return

    elif type(event) is list: #if the event is a list, it came from lambda2

        print\_debug("Request came for id list: ("+str(event)+")")

        for id in event:

            if type(id) is int:#send each id to the queue

                response = queue.send\_message(MessageBody=str(id))

                print\_debug("Sending id "+str(id)+" to SQS")

            else:

                print("ERROR: non-inetger id detected in list ("+id+")")

    else :

        print("ERROR: input is not type list or json ("+event+")")

        return

* **Lambda2**

import json

import boto3

debug = 1

#resource for DynamoDB and lambda

dynamodb = boto3.resource('dynamodb')

client = boto3.client('lambda')

def print\_debug(message): #simple debug function

    if debug:

        print("DEBUG: "+message)

def lambda\_handler(event, context):

    print\_debug("Recieved event, begining execution ("+str(event)+")")

    table = dynamodb.Table('Ask1Tests')

    response = table.scan() #get all items from table

    items = response['Items']

    ids = []

    for item in items:

        ids.append(int(item["testID"])) #get all testIDs

    print\_debug("Recieved ids from table: "+str(ids))

    print\_debug("Sending ids to Lambda1")

    response = client.invoke( #send ids to Lambda1

        FunctionName='Assignment1\_Ask2\_Lamda1',

        InvocationType='Event',

        LogType='None',

        Payload=str(ids),

    )

    print\_debug("Payload sent, finishing execution")

* **Lambda3** (Some lines are cliped so copy and paste the code to an editor e.g. VS Code)

import json

import boto3

debug = 1

sf\_client = boto3.client('stepfunctions')

lambda\_client = boto3.client('lambda')

def print\_debug(message): #simple debug function

    if debug:

        print("DEBUG: "+message)

def error(message): #simple error function

    print("ERROR: "+message)

def lambda\_handler(event, context):

    print\_debug("Recieved event, begining execution ("+str(event)+")")

    if not("URL" in event): #check if required parameters are in event

        error("URL not specified in the request")

        return

    elif not("Method" in event):

        error("Method not specified in the request")

        return

    elif not("ExpectedResult" in event):

        error("ExpectedResult not specified in the request")

        return

    #Send the data to the SM

    print\_debug("Sending data to State Machine")

    response = sf\_client.start\_sync\_execution(

        stateMachineArn="arn:aws:states:us-east-1:613787790506:stateMachine:ASK1\_2\_Express",

        input='{"method" : "POST","url":"'+event["URL"]+'","testMethod": "'+event["Method"]+'","expectedResult": "'+event["ExpectedResult"]+'"}'

    )

    print\_debug("State Machine returned status:"+response["status"]+" ("+str(response)+")")

    #Get the SM answer

    if response["status"]=="FAILED":

        return {"statusCode": 400, "Description": "Bad Request"}

    #craft the state machine output

    sm\_output = '{"testID" : '+str(json.loads(response["output"])["testID"])+'}'

    print\_debug("State Machine crafted output: "+sm\_output)

    #Send the state machine output to the lambda45

    print\_debug("Sending "+str(sm\_output)+" to lambda45")

    response = lambda\_client.invoke(

        FunctionName='Assignment1\_Ask2\_Lamda45',

        InvocationType='Event',

        LogType='None',

        Payload=sm\_output

    )

* **Lambda45**

import json

import boto3

import urllib.request

debug = 1

sf\_client = boto3.client('stepfunctions')

sns\_client = boto3.client('sns')

def print\_debug(message): #Simple debug function

    if debug:

        print("DEBUG: "+message)

def format\_result(result): #Simple function to format the result

    print\_debug("Formater: Result contains: "+str(result))

    print\_debug("Formater: Result has type: "+str(type(result))+" .Formatting...")

    output = ""

    if type(result) is str:

        output = result.replace('"','\'')

    elif type(result) is bytes:

        output = str(result, "utf-8").replace('"','\'')

    print\_debug("Formater: Result formating ended. final output string: '"+output+"'")

    return output

def lambda\_handler(event, context):

    print\_debug("Recieved event, begining execution ("+str(event)+")")

    #Simple error case

    if "testID" in event:

        test\_id = event["testID"]

        print\_debug("Call came from Lambda3 for id="+str(event["testID"]))

    elif not ("Records" in event) or not ("body" in event["Records"][0]):

        print("ERROR")

        return

    else:

        test\_id = event["Records"][0]["body"]

        print\_debug("Call came from SQS for id="+str(test\_id))

    #send the ID to the state machine to get furhter details

    print\_debug("Sending ID to State Machine")

    response = sf\_client.start\_sync\_execution(

        stateMachineArn="arn:aws:states:us-east-1:613787790506:stateMachine:ASK1\_2\_Express",

        input='{"method" : "GET","testID" : "'+str(test\_id)+'" }'

    )

    print\_debug("State Machine returned ("+str(response)+")")

    #Get just the SM output

    sm\_output = json.loads(response["output"])

    sm\_output["testID"] = int(test\_id)

    print\_debug("State Machine crafted output: "+str(sm\_output))

    #Validate the URL and also get result

    valid\_URL = 1

    test\_output = sm\_output

    print\_debug("Validating the URL")

    req = urllib.request.Request(url=sm\_output["URL"], method=sm\_output["Method"] ) #Create the request

    try: response = urllib.request.urlopen(req) #Send the request

    except (urllib.error.URLError, ValueError) as e:

        # print\_debug(e)

        valid\_URL=0

        print\_debug("URL invalid, storing error output")

    if valid\_URL:

        print\_debug("URL validated, storing the output")

        test\_output["ActualResult"] = format\_result(response.read())

    else:

        test\_output["ActualResult"] = "Invalid URL"

    print\_debug("URL output stored ("+str(test\_output["ActualResult"])+")")

    #Compare Expected and Actual result

    if test\_output["ActualResult"] == test\_output["ExpectedResult"]:

        test\_output["ResultAsExpected"] = True

    else:

        test\_output["ResultAsExpected"] = False

    # Send the json to the SNS

    print\_debug("Publishing into SNS (Message: "+str(test\_output)+")")

    response = sns\_client.publish(

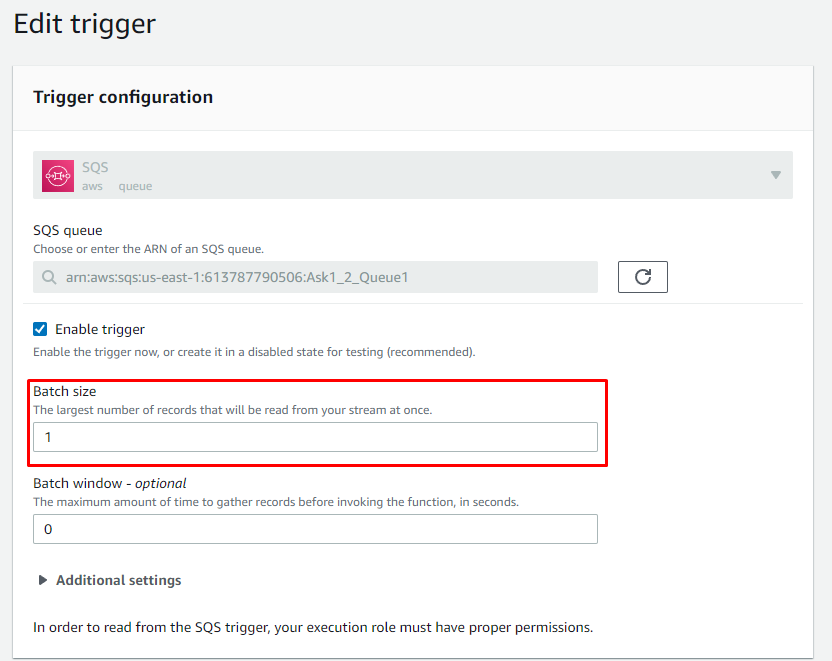
        TopicArn='arn:aws:sns:us-east-1:613787790506:Ask1\_2SNS',

        Message=str(test\_output),

    )

    print\_debug("End of execution, returning ("+str(response)+")")

    return response



The Lambda45 function needed a change in the configuration about its trigger from the SQS. The default batch size was 10 and as a result some triggers were lost. I changed the size to 1 and they seemed to work flawlessly together.

* **Lambda7** (also clipped)

import json

import boto3

import ast

debut = 1

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('Ask1Tests')

def debug(message): #simple debug function

    if debug:

        print("DEBUG: "+message)

def error(message):#simple error function

    print("ERROR: "+message)

def lambda\_handler(event, context):

    if not("Records" in event) or not ("Sns" in event["Records"][0]) or not ("Message" in event["Records"][0]["Sns"]):

        error('Input has wrong format (["Records"][0]["Sns"]["Message"] field not found)')

        return

    #format the test fields to store them into DynamoDB

    debug("event index = "+event["Records"][0]["Sns"]["Message"])

    eventstr = event["Records"][0]["Sns"]["Message"]

    debug("eventstr = "+eventstr)

    item = ast.literal\_eval(eventstr) #i used ast.literal\_eval to convert the string to a dictionary because json.loads wants double quotes

    item["testID"] = int(item["testID"])

    debug("item = "+str(item))

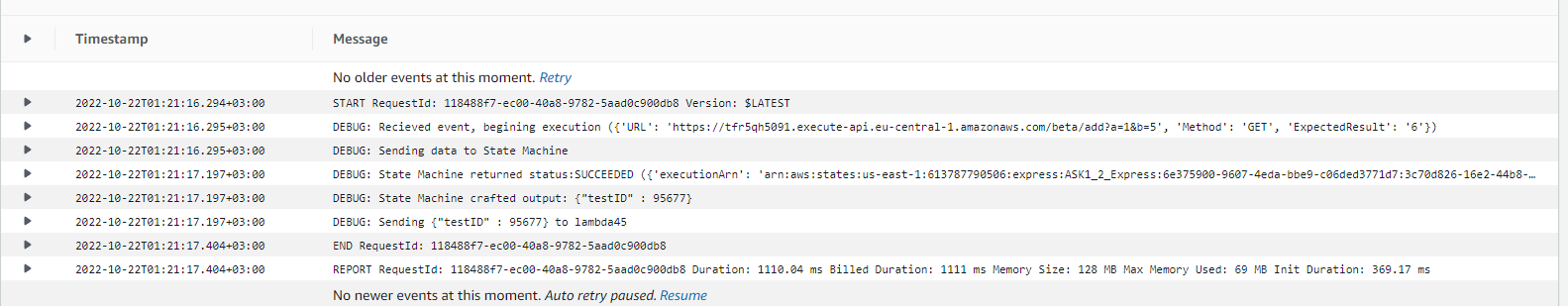
    response = table.put\_item(

        Item= item

    )

**2.3 Execution and Testing**

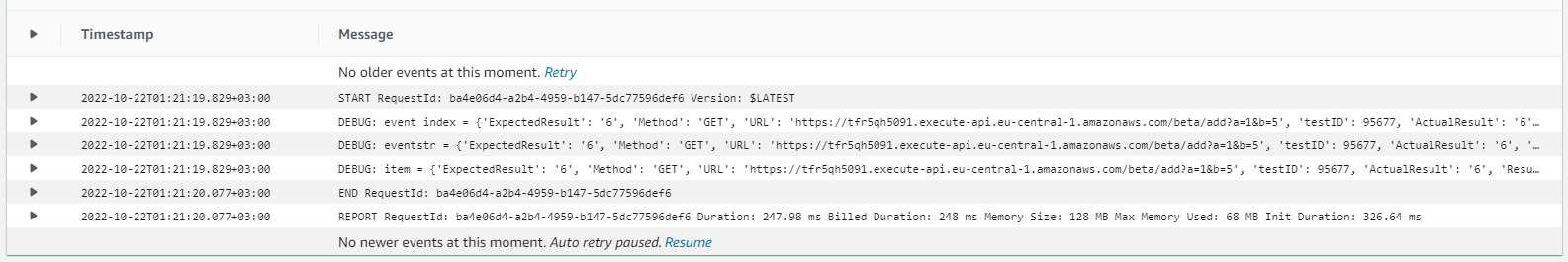
Let’s start with an empty database and test the three API inputs and see some of the logs:

* /test/URL="https://tfr5qh5091.execute-api.eu-central-1.amazonaws.com/beta/add?a=1&b=5"&Method=GET&ExpectedResult=6

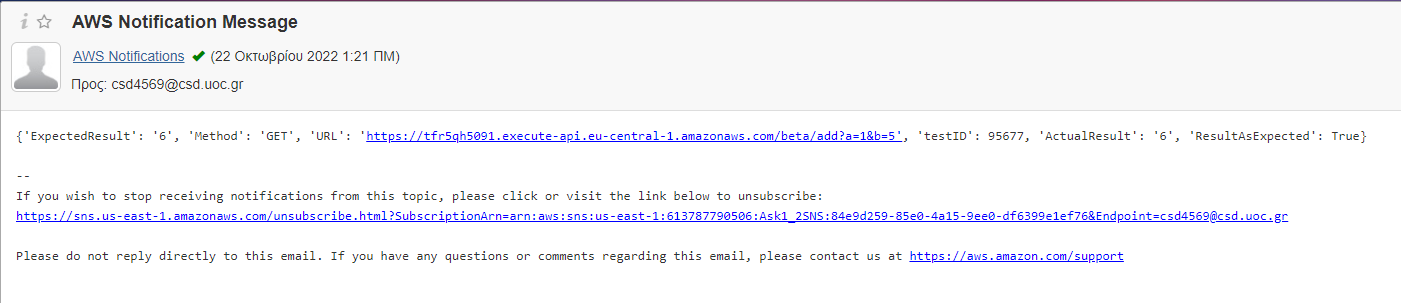
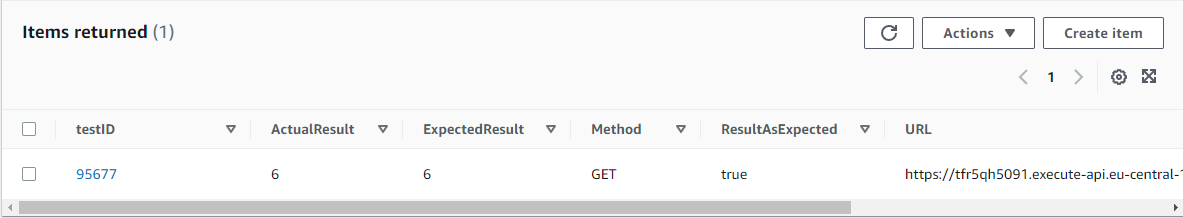
Lambda3 Function Logs

Above we can see through the DEBUG: prints that Lambda3 sent the test parameters to the Step Function and got an ID as a reply. Finally, it sent the new test ID to Lambda45.

Lambda45 Function Logs

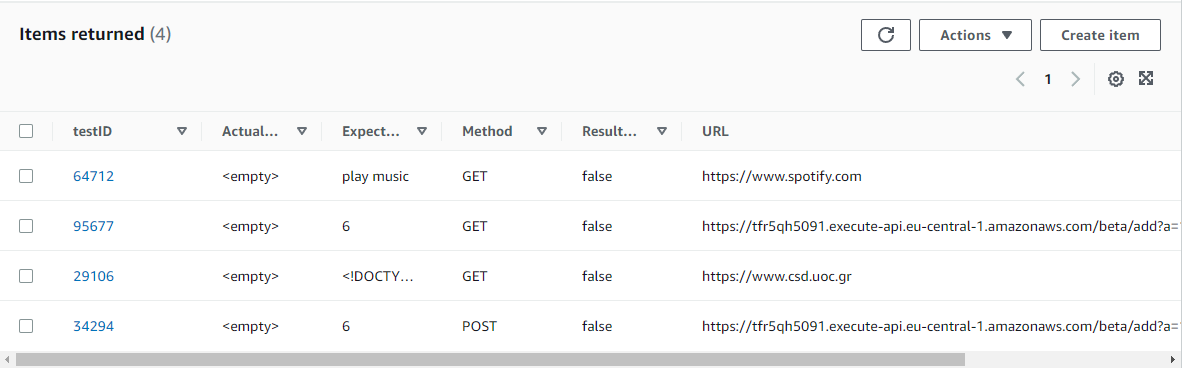
In Lambda 45 function logs we can observe the function getting the test parameters from the Step Function using the test ID. Then it validates the website and gets the response from it. Finally it compares the actual and expected output and sends all the gathered data to the SNS for storing and emailing.

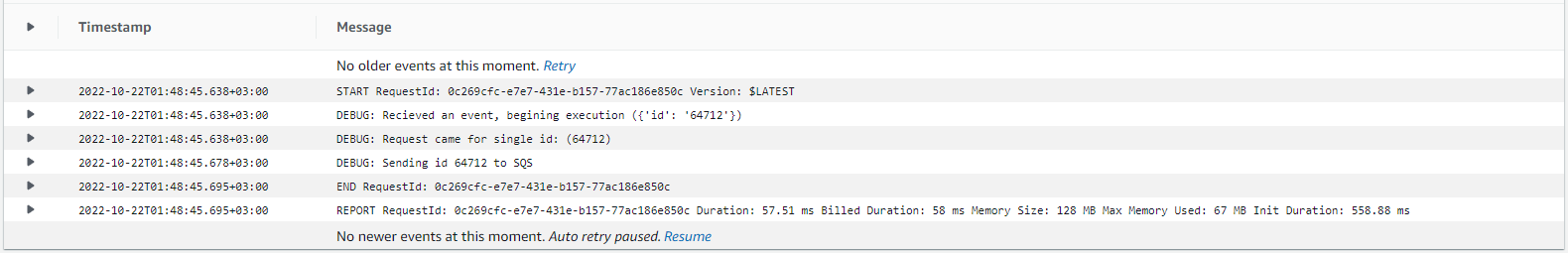
Lambda7 Function Logs

In the screenshot above Lambda7 sends the results to DunamoDB (after some formatting).

Database updated

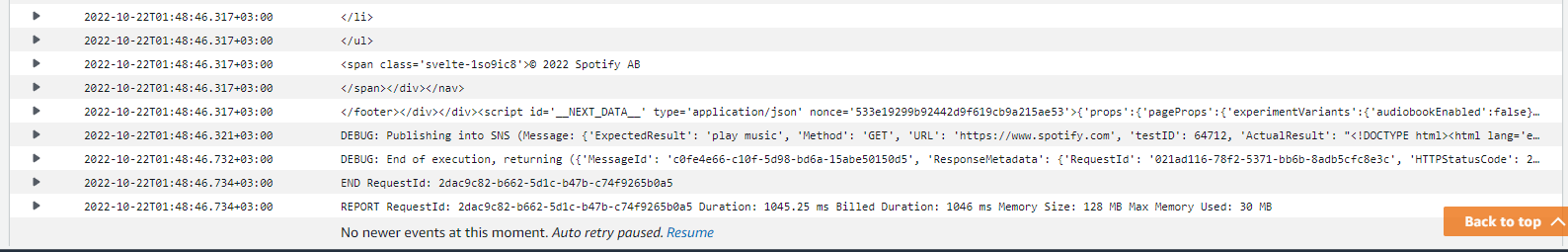
Email Received

**For the next tests we need some additions to the database (I removed all the actual results)**

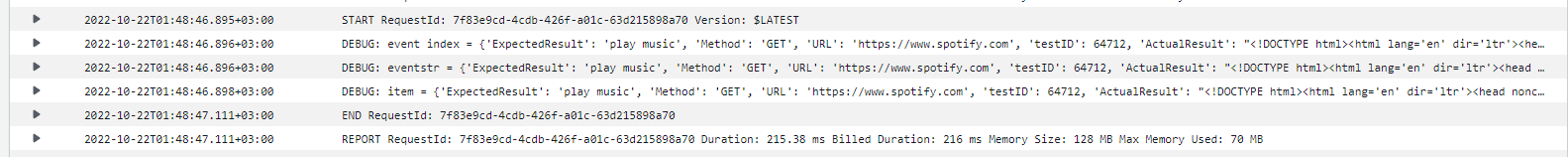
* /tests/64712

Lambda1 Function Logs

When the API resource is called it calls Lambda1 which takes the test ID and just passes it to SQS



Lambda45 Function Logs

The formatter debug output has spammed the logs but with the power of editing I made it somewhat readable. Lambda45 as said in the previous run takes the tests parameters, validates the url, stores the result, compares it with the actual one and finally sends all the findings to SNS

Lambda7 Function Logs

On the Last mile again, Lambda7 sends the results to DynamoDB for storing

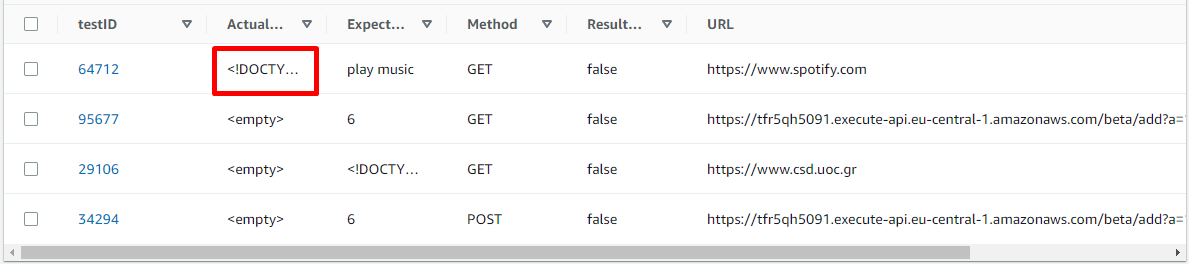
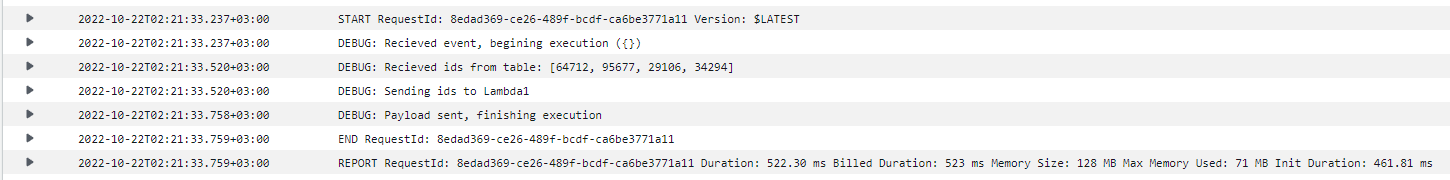


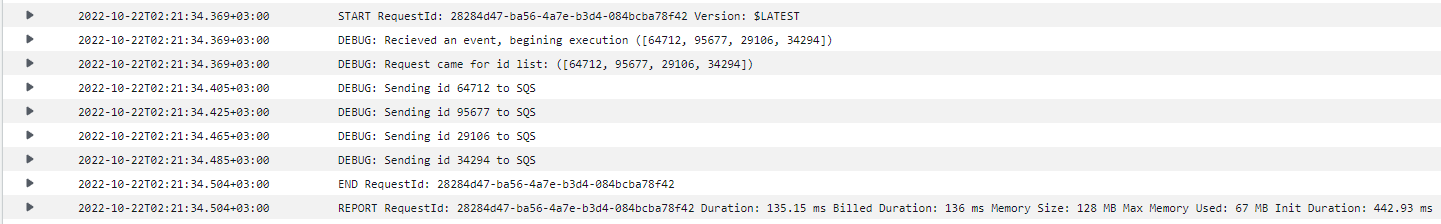
Table Contents after the test run

* /tests/all

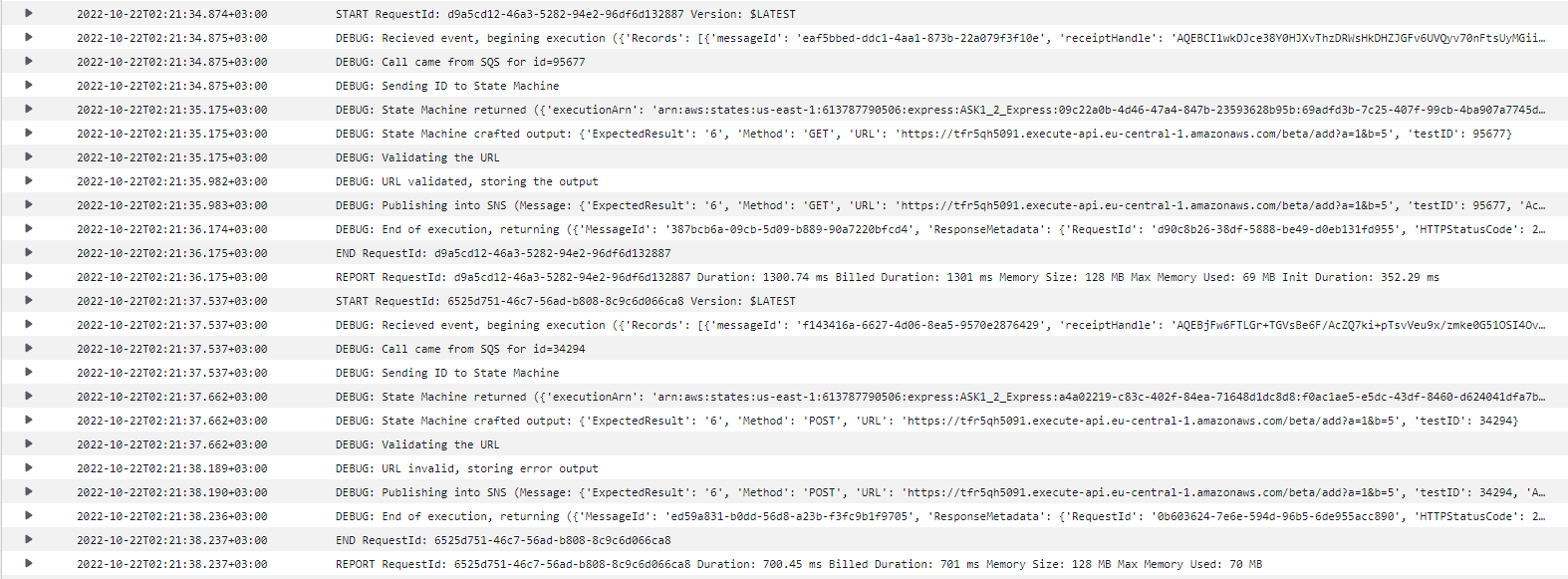
With the database as seen above we send a request to the resource



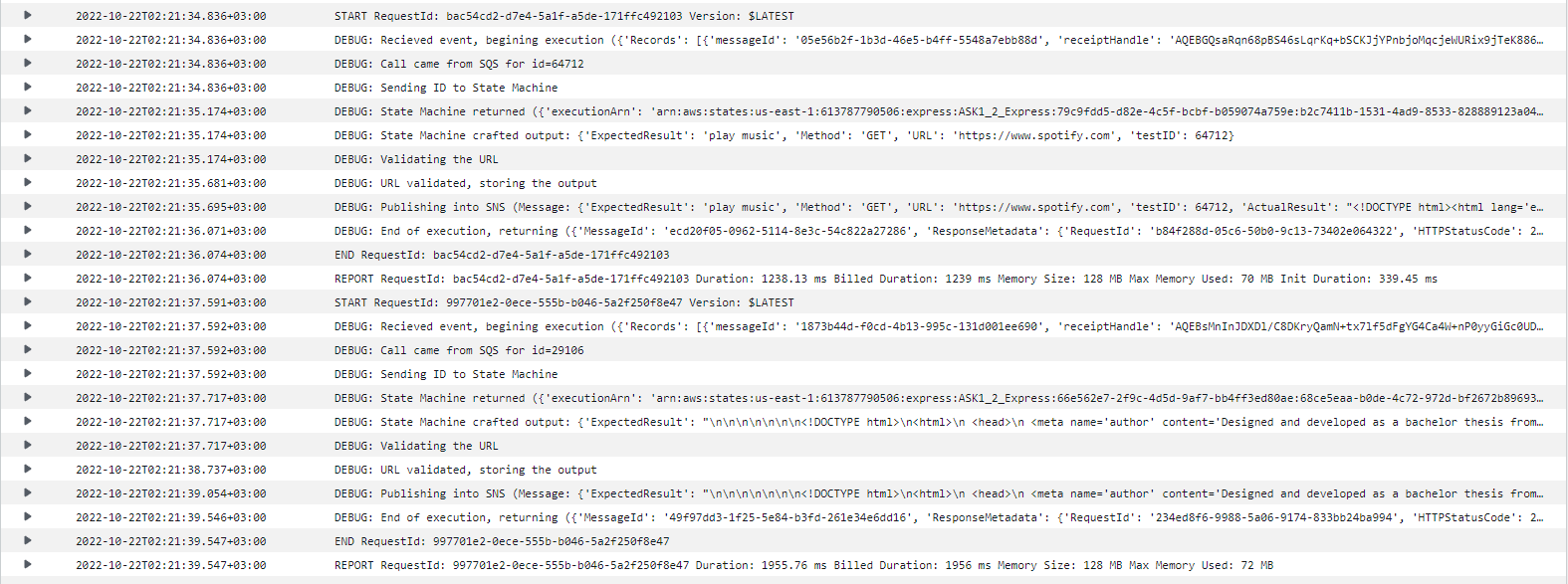
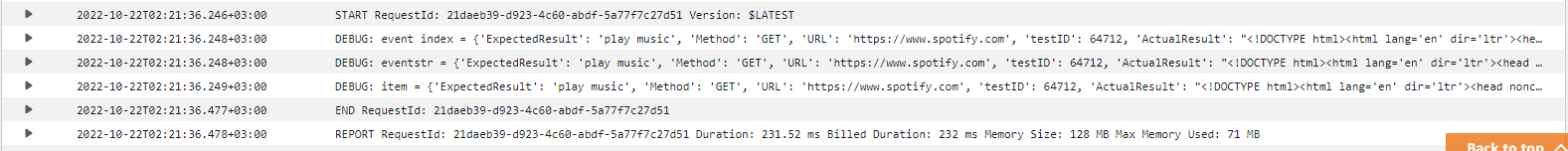
Lambda2 Function Logs

Lambda2 gathers all test IDs from DynamoDB table and adds them to a list. Then forwards it to Lambda1

Lambda1 Function Logs

Lambda1 gets the ID list and sends every ID individually to SQS

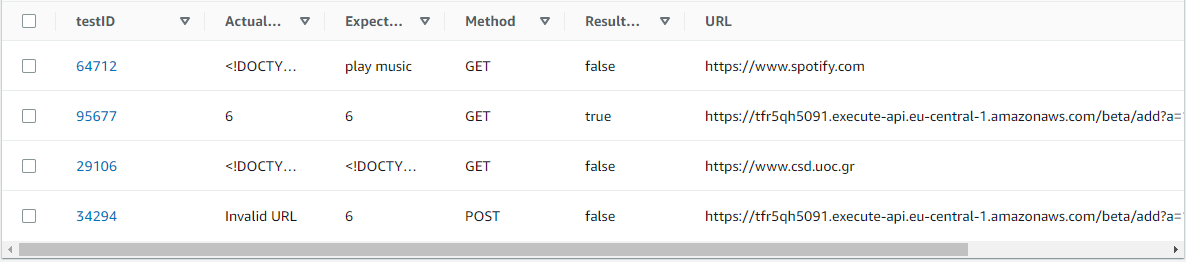
Lambda45 Function Logs (1/2)

On Lambda45 we can see that all 4 test IDs are processed and passed to SNS (I removed some debug prints because the spam became uncontrollable)

Lambda7 Function Logs (2-4/4)

Lambda7 Function Logs (1/4)

Lambda45 Function Logs (2/2)

Lambda7 takes all notifications and stores them to DynamoDB as can be seen below