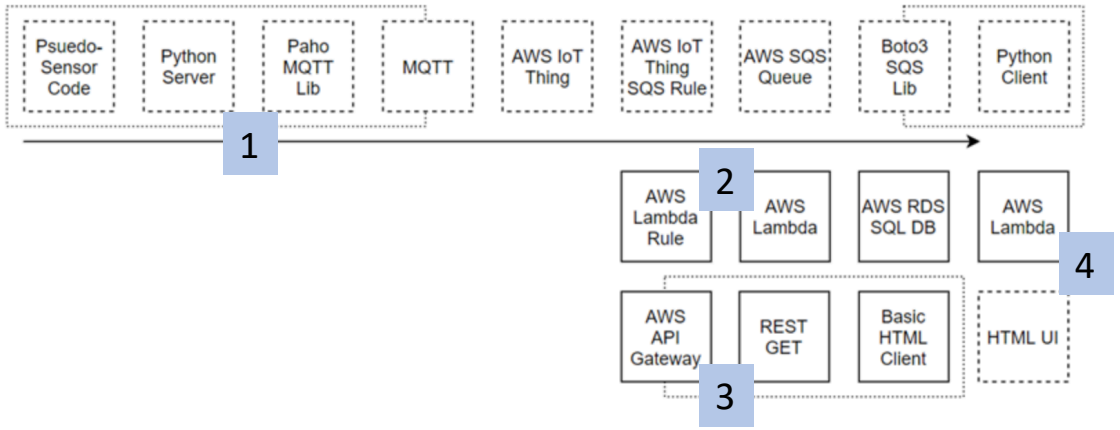
Project 2: Extended AWS IoT Connections

Kaylin Punotai

11 March 2021

Part 1. Implementation/Assumption Notes



1. Python server gets sensor readings and sends to AWS SQS
2. AWS Lambda rule subscribes first lambda to read incoming SQS messages and create entries in DynamoDB
3. A client sends REST GET request to API Gateway
4. API Gateway triggers second lambda to scan DynamoDB and return all entries to client

Part 2. Code: pseudoSensor.py

This code is from Week 1 and generates random humidity and temperature readings

```python
import random

class PseudoSensor:

    h_range = [0, 20, 20, 40, 40, 60, 60, 80, 80, 90, 70, 70, 50, 50, 30, 30, 10, 10]

    t_range = [-20, -10, 0, 10, 30, 50, 70, 80, 90, 80, 60, 40, 20, 10, 0, -10]

    h_range_index = 0

    t_range_index = 0

    humVal = 0

    tempVal = 0


    def __init__(self):

        self.humVal = self.h_range[self.h_range_index]

        self.tempVal = self.t_range[self.t_range_index]


    def generate_values(self):

        self.humVal = self.h_range[self.h_range_index] + random.uniform(0, 10);

        self.tempVal = self.t_range[self.t_range_index] + random.uniform(0, 10);

        self.h_range_index += 1

        if self.h_range_index > len(self.h_range) - 1:

            self.h_range_index = 0

        self.t_range_index += 1

        if self.t_range_index > len(self.t_range) - 1:

            self.t_range_index = 0

        return self.humVal, self.tempVal

```

Each index has a starter magnitude

Generated values are randomized based on the starter magnitude at each index

Part 2. Code: server.py

```python
import sys, time
from datetime import datetime

from psuedoSensor import PseudoSensor
ps = PseudoSensor()

import paho.mqtt.client as paho
import os
import socket
import ssl
from time import sleep
from random import uniform

connflag = False

def on_connect(client, userdata, flags, rc):
    global connflag
    connflag = True
    print("Connection returned result: " + str(rc) )

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

mqttc = paho.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
```

Import pseudoSensor

Import Paho MQTT library

connflag indicates whether server.py is connected to MQTT

When connected, connflag becomes true and connection is printed to console

MQTT messages print on the client console

```
awshost =
awsport =
clientId =
thingName
caPath = "
certPath =
keyPath =

mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath, cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLS

mqttc.connect(awshost, awsport, keepalive=60)

mqttc.loop_start()

i = 0
while i<10:
    sleep(1)
    if connflag == True:
        h,t = ps.generate_values()
        now = datetime.now()
        timestamp = now.strftime("%m/%d/%Y %H:%M:%S")
        message = "{ \"time\": \"%s\", \"temperature\": \"%.5f\", \"humidity\": \"%.5f\" }" % (timestamp, t, h)
        mqttc.publish("sensor", message, qos=1)
        print("[" + str(i) + "]\n" + message)
    else:
        print("waiting for connection...")
    i += 1

```

AWS connection

Iterate 10 times to send 10 readings

Each reading has a 1 second delay

Generate sensor readings and publish the time, temp, and humidity to MQTT

Part 2. Code: LambdaToDB.js

New SQS messages trigger this Lambda (based on lambda rule), which parses the message and adds them to the “pseudosensor” table

```
const AWS = require('aws-sdk');

AWS.config.update({endpoint: "https://dynamodb.us-west-2.amazonaws.com"});

var dynamo = new AWS.DynamoDB();

exports.handler = (event) => {
    event.Records.forEach(record => {

        var parser = JSON.parse(record.body);

        var params = {
            TableName: "pseudosensor",
            Item: {
                "id": {S: record.messageId},
                "time": {S: parser.time},
                "temperature": {N: parser.temperature},
                "humidity": {N: parser.humidity}
            }
        };

        dynamo.putItem(params, function(err, data) {
            if (err) {
                console.error("Unable to add item. Error JSON:", JSON.stringify(err, null, 2));
            } else {
                console.log("Added item:", JSON.stringify(data, null, 2));
            }
        });

    });
};
```

Message body contains time, temp, and humidity. ID comes from the SQS message ID

Part 2. Code: LambdaToClient.js

This Lambda is triggered by the API Gateway, which sends REST requests to the Lambda

```
1  const AWS = require('aws-sdk');
2
3  const dynamo = new AWS.DynamoDB.DocumentClient();
4
5  exports.handler = async (event, context) => {
6      //console.log('Received event:', JSON.stringify(event, null, 2));
7      let body;
8      let statusCode = '200';
9      const headers = {
10         'Content-Type': 'application/json',
11     };
12
13     try {
14         switch (event.httpMethod) {
15             case 'DELETE':
16                 body = await dynamo.delete(JSON.parse(event.body)).promise();
17                 break;
18             case 'GET':
19                 body = await dynamo.scan({ TableName:"pseudosensor" }).promise();
20                 break;
21             case 'POST':
22                 body = await dynamo.put(JSON.parse(event.body)).promise();
23                 break;
24             case 'PUT':
25                 body = await dynamo.update(JSON.parse(event.body)).promise();
26                 break;
27             default:
28                 throw new Error(`Unsupported method "${event.httpMethod}"`);
29         }
30     } catch (err) {
31         statusCode = '400';
32         body = err.message;
33     } finally {
34         body = JSON.stringify(body);
35     }
36
37     return { body };
38 };
39
```

REST methods

For ”GET” requests, the Lambda returns all entries from the “pseudosensor” table

Part 2. Code: client.py

Client sends the “GET” response and prints all “pseudosensor” entries to the console

```
1  import requests
2
3  URL = "https:/                i.us-west-2.amazonaws.com/default/LambdaToClient"
4
5  response = requests.request("GET", URL)
6
7  print(response.text)
```

URL for API Gateway

Part 3. Demonstration

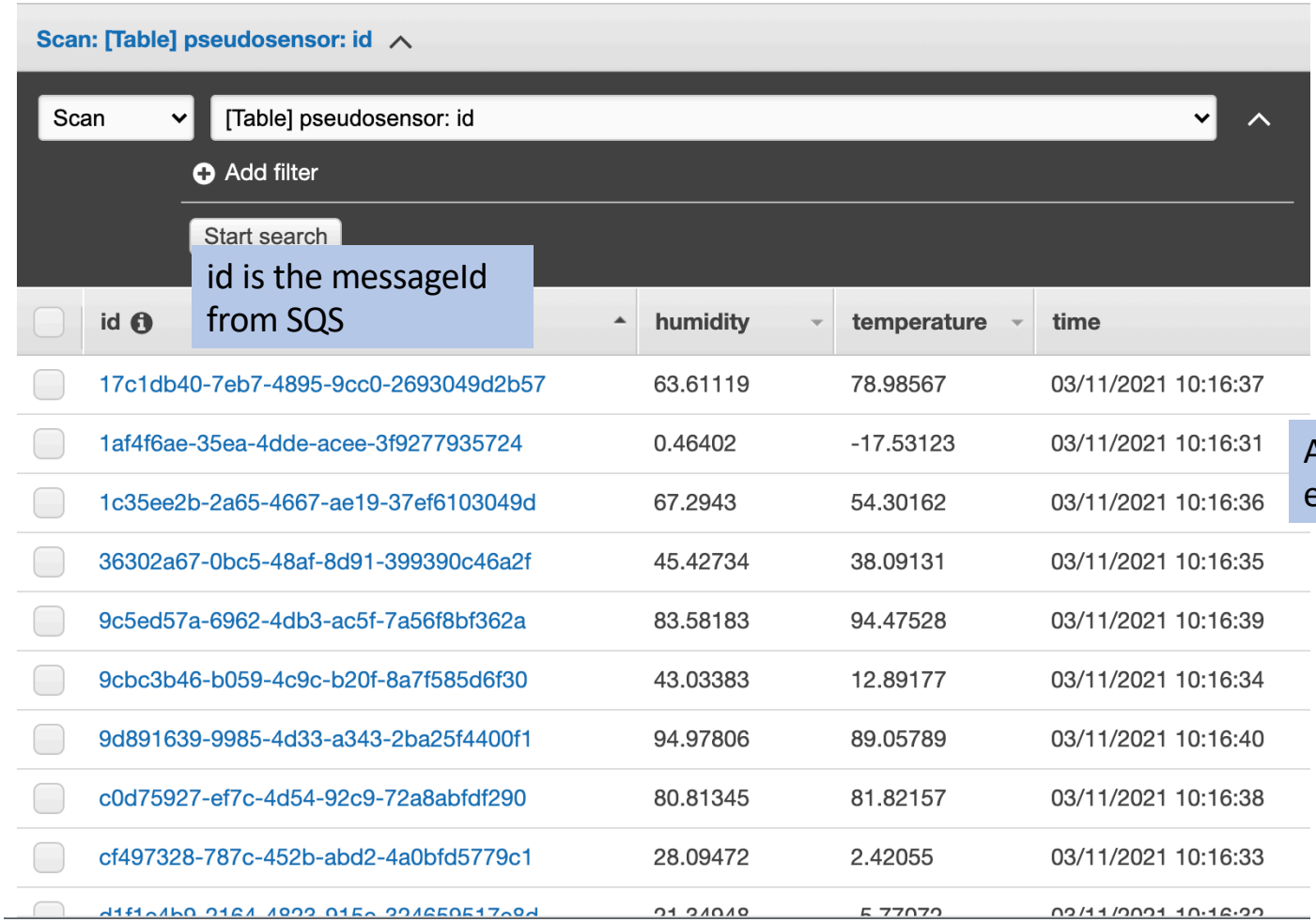
Server.py connecting to AWS and sending JSON messages

```
pi@raspberrypi:~/Desktop/Project 2 $ python server.py
Connection returned result: 0
[0]
{ "time": "03/11/2021 10:16:31", "temperature": "-17.53123", "humidity": "0.46402" }
[1]
{ "time": "03/11/2021 10:16:32", "temperature": "-5.77072", "humidity": "21.34948" }
[2]
{ "time": "03/11/2021 10:16:33", "temperature": "2.42055", "humidity": "28.09472" }
[3]
{ "time": "03/11/2021 10:16:34", "temperature": "12.89177", "humidity": "43.03383" }
[4]
{ "time": "03/11/2021 10:16:35", "temperature": "38.09131", "humidity": "45.42734" }
[5]
{ "time": "03/11/2021 10:16:36", "temperature": "54.30162", "humidity": "67.29430" }
[6]
{ "time": "03/11/2021 10:16:37", "temperature": "78.98567", "humidity": "63.61119" }
[7]
{ "time": "03/11/2021 10:16:38", "temperature": "81.82157", "humidity": "80.81345" }
[8]
{ "time": "03/11/2021 10:16:39", "temperature": "94.47528", "humidity": "83.58183" }
[9]
{ "time": "03/11/2021 10:16:40", "temperature": "89.05789", "humidity": "94.97806" }
pi@raspberrypi:~/Desktop/Project 2 $
```

SQS receives all messages and triggers LambdaToDB to create entries in “pseudosensor” table

This LambdaToDB CloudWatch trace shows entries were successfully added to the table

This is the “pseudosensor” table in DynamoDB

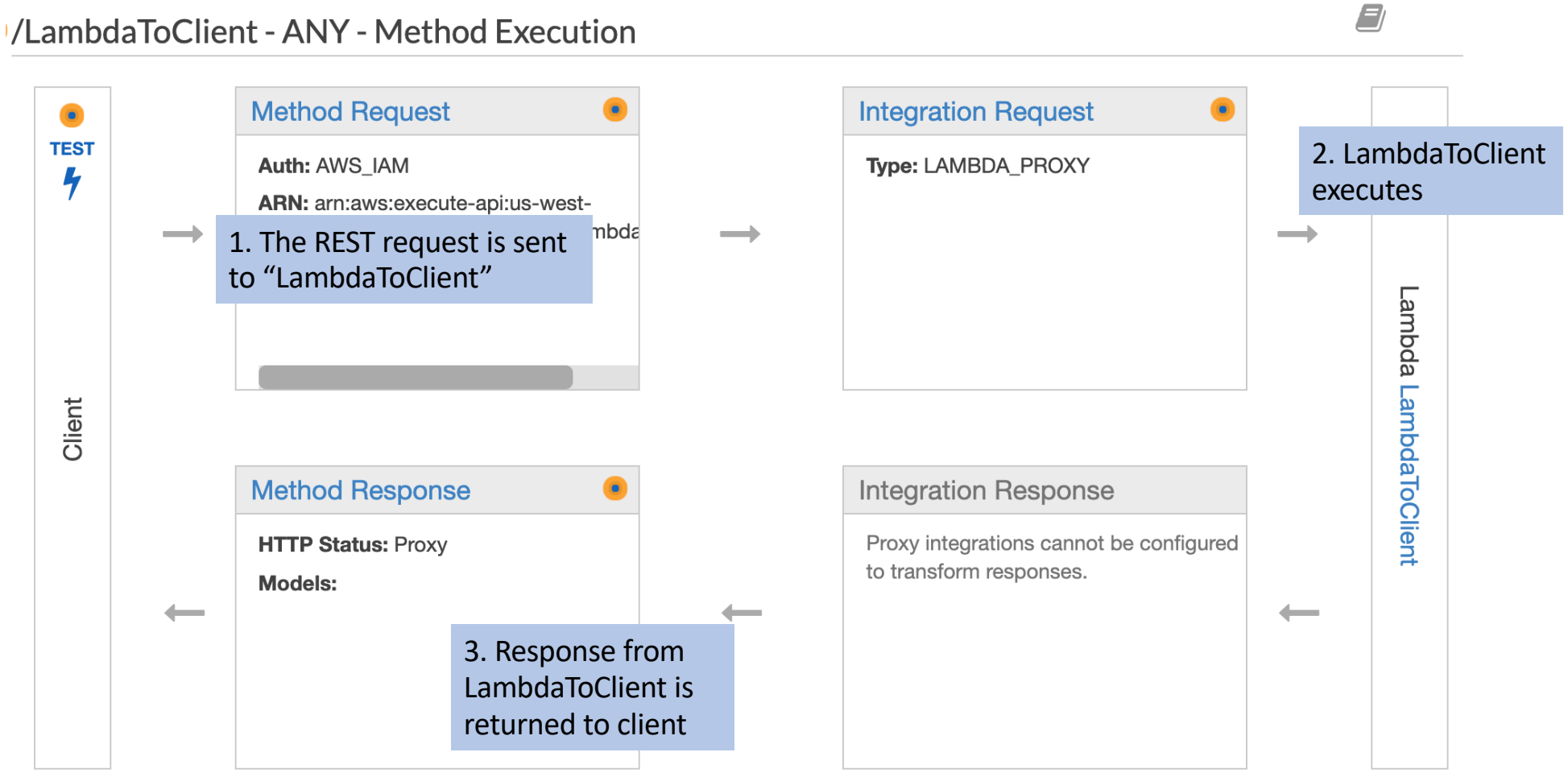


id is the messageId from SQS

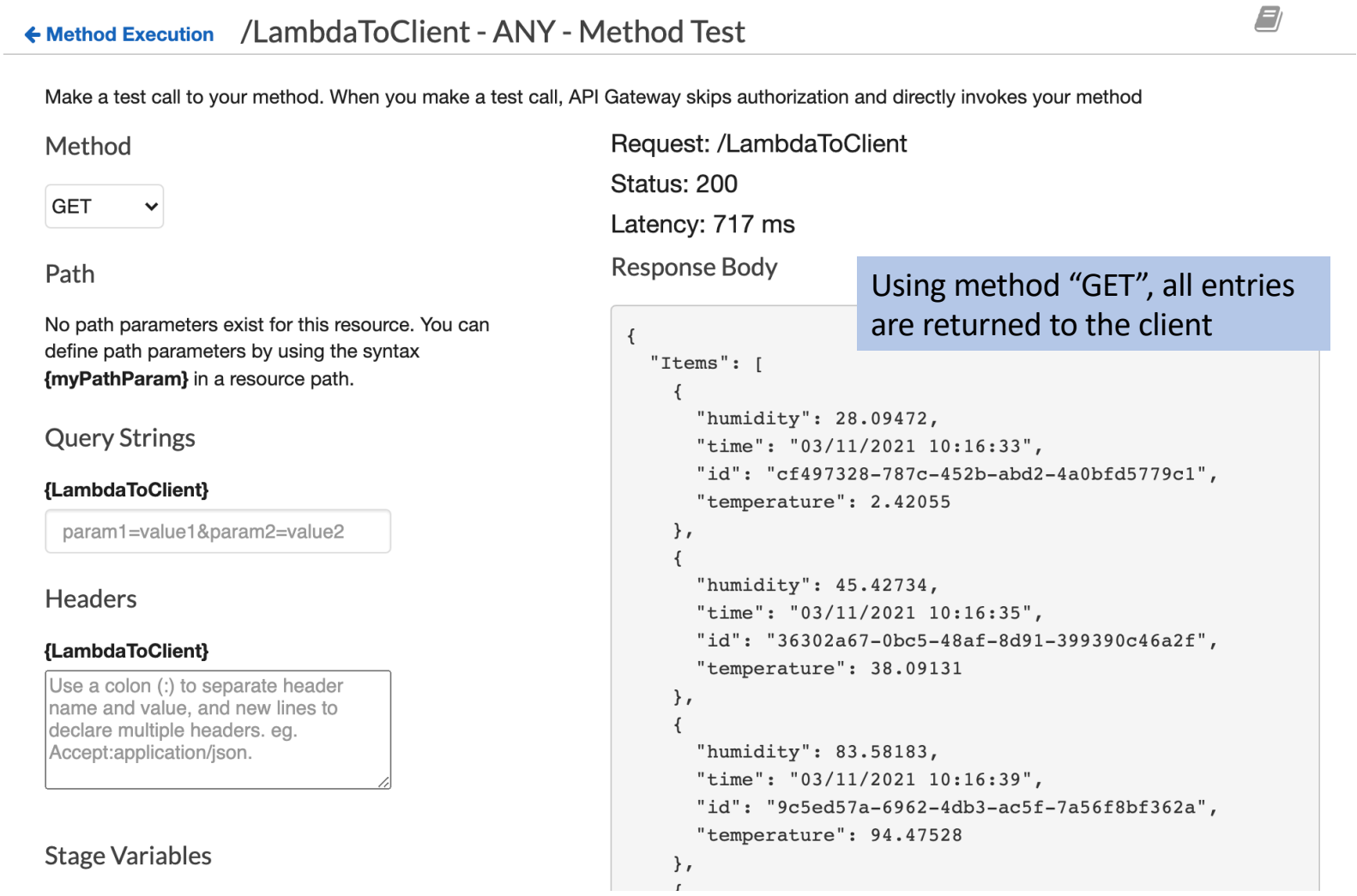
id	humidity	temperature	time
17c1db40-7eb7-4895-9cc0-2693049d2b57	63.61119	78.98567	03/11/2021 10:16:37
1af4f6ae-35ea-4dde-acee-3f9277935724	0.46402	-17.53123	03/11/2021 10:16:31
1c35ee2b-2a65-4667-ae19-37ef6103049d	67.2943	54.30162	03/11/2021 10:16:36
36302a67-0bc5-48af-8d91-399390c46a2f	45.42734	38.09131	03/11/2021 10:16:35
9c5ed57a-6962-4db3-ac5f-7a56f8bf362a	83.58183	94.47528	03/11/2021 10:16:39
9cbc3b46-b059-4c9c-b20f-8a7f585d6f30	43.03383	12.89177	03/11/2021 10:16:34
9d891639-9985-4d33-a343-2ba25f4400f1	94.97806	89.05789	03/11/2021 10:16:40
c0d75927-ef7c-4d54-92c9-72a8abfdf290	80.81345	81.82157	03/11/2021 10:16:38
cf497328-787c-452b-abd2-4a0bfd5779c1	28.09472	2.42055	03/11/2021 10:16:33
d1f1e4b0-2164-4823-915e-324659517e8d	21.34948	-5.77072	03/11/2021 10:16:32

All entries were successfully entered in table

This is the API test for LambdaToClient



This is the result of the API test



When the client Python runs, it invokes the API and prints the response to the console

```
pi@raspberrypi:~/Desktop/Project 2 $ python client.py
{"Items":[{"humidity":28.09472,"time":"03/11/2021 10:16:33","id":"cf497328-787c-
452b-abd2-4a0bfd5779c1","temperature":2.42055},{"humidity":45.42734,"time":"03/1
1/2021 10:16:35","id":"36302a67-0bc5-48af-8d91-399390c46a2f","temperature":38.09
131},{"humidity":83.58183,"time":"03/11/2021 10:16:39","id":"9c5ed57a-6962-4db3-
ac5f-7a56f8bf362a","temperature":94.47528},{"humidity":67.2943,"time":"03/11/202
1 10:16:36","id":"1c35ee2b-2a65-4667-ae19-37ef6103049d","temperature":54.30162},
{"humidity":0.46402,"time":"03/11/2021 10:16:31","id":"1af4f6ae-35ea-4dde-acee-3
f9277935724","temperature":-17.53123},{"humidity":80.81345,"time":"03/11/2021 10
:16:38","id":"c0d75927-ef7c-4d54-92c9-72a8abfdf290","temperature":81.82157},{"hu
midity":43.03383,"time":"03/11/2021 10:16:34","id":"9cbc3b46-b059-4c9c-b20f-8a7f
585d6f30","temperature":12.89177},{"humidity":63.61119,"time":"03/11/2021 10:16:
37","id":"17c1db40-7eb7-4895-9cc0-2693049d2b57","temperature":78.98567},{"humidi
ty":21.34948,"time":"03/11/2021 10:16:32","id":"d1f1e4b9-2164-4823-915e-32465951
7e8d","temperature":-5.77072},{"humidity":94.97806,"time":"03/11/2021 10:16:40",
"id":"9d891639-9985-4d33-a343-2ba25f4400f1","temperature":89.05789}],"Count":10,
"ScannedCount":10}
```