Assignment 6

1. Modify your Lambda function to include the field City for each new record in theStudents DynamoDB table.

To do this, we can use our existing lambda function that was created in assignment 6. We edit our python script to support the “city” field. We can do it as follows:

```
if operation == "CREATE":
    try:
        item = event.get("item")
        if "id" and "full_name" and "personal_website"  and "city" in item:
            res_dynamo = add_item(item=item)
            return {"body": json.dumps(res_dynamo)}
        else:
            return {"body": json.dumps({"message": "This is an invalid format for item"})}
    except ClientError:
        return {"message": json.dumps({"Operation CREATE unsuccessful"})}
```

The rest of the handler function remains the same, just in the CREATE operation, we indicate that a “city” field must be indicated in the item, along with *id*, *full_name* and *personal_website*.

Since we made an edit in our file, we must compress it again as a zip file and update our lambda function.

```
[CV9FCYQ4XQ:tarea5 mxm0822$ zip lambda_michelle.zip CRUD_Michelle.py
  adding: CRUD_Michelle.py (deflated 72%)
```

```
[CV9FCYQ4XQ:tarea5 mxm0822$ aws lambda update-function-code --function-name CRUD_
Michelle2 --zip-file fileb:///Users/mxm0822/Documents/School/6tosemestre/CloudCo
mputing/tarea5/lambda_michelle.zip
```

Output:

```
{
    "FunctionName": "CRUD_Michelle2",
    "FunctionArn": "arn:aws:lambda:us-east-1:292274580527:function:CRUD_Michelle2",
    "Runtime": "python3.9",
    "Role": "arn:aws:iam::292274580527:role/lambda_ice191",
    "Handler": "CRUD_Michelle2.handler_function",
    "CodeSize": 965,
    "Description": "",
    "Timeout": 5,
    "MemorySize": 128,
    "LastModified": "2023-03-20T05:22:55.000+0000",
    "CodeSha256": "uI108pg/MhZCIUN8T4I3b5Zc0mb1O0feRwiPUrzTfuo=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "ab69538d-1245-43a2-a6af-561e36171a37",
    "State": "Active",
    "LastUpdateStatus": "InProgress",
    "LastUpdateStatusReason": "The function is being created.",
    "LastUpdateStatusReasonCode": "Creating",
    "PackageType": "Zip",
    "Architectures": [
        "x86_64"
    ],
    "EphemeralStorage": {
        "Size": 512
    },
    "SnapStart": {
        "ApplyOn": "None",
        "OptimizationStatus": "Off"
    },
    "RuntimeVersionConfig": {
        "RuntimeVersionArn": "arn:aws:lambda:us-east-1::runtime:edb5a058bfa782cb9cedc6d
534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0"
    }
}
```


Having done this, we can test it by creating an item. We must edit our previous json file with the CREATE operation, by adding a “city” field.

Json file used is:

```
{
    "operation": "CREATE",
    "item":
    {
        "id": "1313",
        "full_name": "Mimi",
        "personal_website": "pagina_test.com",
        "city": "Tijuana"
    }
}
```

We must now invoke our function and provide the edited json file:

```
CV9FCYQ4XQ:tarea5 mxm0822$ aws lambda invoke --function-name CRUD_Michelle2 --cli-b
inary-format raw-in-base64-out --payload fileb:///Users/mxm0822/Documents/School/6t
osemestre/CloudComputing/tarea5/test_create.json response2.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

We get the HTTP 200 as StatusCode which indicates us that our request was donde successfully. And the ExecutedVersion indicates us the version of our function that was executed.

If we open our response2.json file, we can se de body of our newly created item:

We can verify it, by entering Students table:

```
    {
        "city": {
            "S": "Tijuana"
        },
        "full_name": {
            "S": "Mimi"
        },
        "id": {
            "S": "1313"
        },
        "personal_website": {
            "S": "pagina_test.com"
        }
    },
```


2. Modify Read in your Lambda function to return the weather of the city assigned to theStudents DynamoDB table record.

A code was provided by the professor that already supported this functionality.

First we have our imported libraries:

```
import json
import boto3
import urllib3
from botocore.exceptions import ClientError
```

In comparison with assignment 5, we have a new library that we are using, “urllib3”. This library helps us make our code more user friendly for our HTTP client, it also helps retrying requests, dealing with HTTP redirects and proxy support for HTTP.

We than have our main l*ambda_handler* function:

```python
def lambda_handler(event, context):
    dynamo = boto3.resource("dynamodb")
    students_table = dynamo.Table("Students")
    matricula = event["id"]

    if matricula:
        try:
            student = students_table.get_item(Key={"id": matricula})
            api_key = get_secret()
            weather = get_weather(student, api_key)
            success_response = {
                "id": matricula,
                "full_name": student["Item"]["full_name"],
                "city": student["Item"]["city"],
                "weather": json.loads(weather)
            }
            return get_response(200, success_response)
        except ClientError as error:
            raise error
    else:
        return get_response(400, {"message": "Missing required field id"})
```

Our lambda_handler function will help us manage all the instructions that will be followed for the trigger event. As same as last assignment, it receives two parameters *event* and *context.*

- Event: It is the request that will be sent by the user. In this case will me *matricula.*
- Context: Provides information to the handler function, on what is going to be executed bases on the location of the execution.

First, if matricula exists while searched through the Students table, we call a function that will return the API key, with the *get_secret()* function. Then with *get_weather* function, we obtain the information given from the student and the api_key. Finally, we have our response, which is the object that stores all the parameters/fields of the search. We manage the statusCode, we obtain 200 as response when successful.

- **Important**: API key is stored as secret.

Now we have our *get_weather* function:

```python
def get_weather(student, api_key):
    base_url = "http://api.openweathermap.org/data/2.5/weather?q={0}&appid={1}"
    if "city" in student["Item"].keys():
        http = urllib3.PoolManager()
        response = http.request('GET', base_url.format(student["Item"]["city"], api_key))
        return response.data
    else:
        return json.dumps("No city assigned to student")
```

This function helps us retrieve the information from an outsider page, which is our Weather API, in this case we are using openweathermap. If city exists in the Students table, we procide to maje an HTTP call with the help of urllib3 library. We then use the GET method to obtain the response and return it. Otherwise, if the city is not found, we show a message indicating that the city is not assigned to any student in the dynamo table.

The *get_secret()* function:

```python
def get_secret():
    secretsmanager = boto3.client(service_name='secretsmanager')
    secret_name = "weather_api_michelle"
    secrets_response = secretsmanager.get_secret_value(SecretId=secret_name)
    return secrets_response['SecretString']
```

As mentioned before, secret is the API key. This functions serves as the retriever of the value of the given key, meaning in this case our Weather API key. For this, we use a secret manager service. We then store the name of our secret in the aws secrets manager and call our APIs service. We then obtain the value of it and return it.

Finally, *get_response* function:

```python
def get_response(code, body):
    return {
        "statusCode": code,
        "body": body
    }
```

This function helps our return the *StatusCode* and the *body* when managing the operations. This depending on the information that was provided.

Having done this, we use aws secrets manager service to create a secret. This service provides encryption to our API key, in this case our Weather API. We indicate the secret string (provided by the teacher: f4edb3afca5c9e19aec9f0210b53735b), which is the value that will be encrypted and stored. Just as follows:

```
CV9FCYQ4XQ:~ mxm0822$ aws secretsmanager create-secret --name api_key_weather_mi
chelle --secret-string f4edb3afca5c9e19aec9f0210b53735b
{
    "ARN": "arn:aws:secretsmanager:us-east-1:292274580527:secret:api_key_weather
_michelle-BZUgvB",
    "Name": "api_key_weather_michelle",
    "VersionId": "7efc3402-6aae-49b6-98f6-42a4505e5025"
}
```

We then run the following command to obtain the value of our recently created secret.

```
CV9FCYQ4XQ:~ mxm0822$ aws secretsmanager get-secret-value --secret-id api_key_we
ather_michelle
{
    "ARN": "arn:aws:secretsmanager:us-east-1:292274580527:secret:api_key_weather
_michelle-BZUgvB",
    "Name": "api_key_weather_michelle",
    "VersionId": "7efc3402-6aae-49b6-98f6-42a4505e5025",
    "SecretString": "f4edb3afca5c9e19aec9f0210b53735b",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": "2023-03-17T18:13:17.084000-07:00"
}
```

It shows the encrypted information of our Weather API.

We must create the function as we did in class and in assignment 5. But an important thing to mention, is that we need to install the newly used library urllib3.

```
CV9FCYQ4XQ:~ mxm0822$ cd /Users/mxm0822/Documents/School/6tosemestre/CloudComput
ing/weather_lambda_michelle
```

```
[CV9FCYQ4XQ:weather_lambda_michelle mxm0822$ pip3 install --target . urllib3
Collecting urllib3
  Downloading urllib3-1.26.15-py2.py3-none-any.whl (140 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 140.9/140.9 kB 2.4 MB/s eta 0:00:00
Installing collected packages: urllib3
Successfully installed urllib3-1.26.15

[notice] A new release of pip available: 22.3.1 -> 23.0.1
[notice] To update, run: python3.10 -m pip install --upgrade pip
```

- **IMPORTANT:** we must create the installation in our folder where our lambda function file is, so when we compress it and create the function, we have the library accessible to our file.

We zip our weather lambda function:

```
[CV9FCYQ4XQ:weather_lambda_michelle mxm0822$ cd ..
[CV9FCYQ4XQ:CloudComputing mxm0822$ zip -r lambda_function.zip weather_lambda_mic
helle
```

Now we run the following command, like in assignment 5, to create the function:

```
[CV9FCYQ4XQ:CloudComputing mxm0822$ aws lambda create-function --function-name we
ather_lambda_michelle --runtime python3.9 --zip-file fileb:///Users/mxm0822/Docu
ments/School/6tosemestre/CloudComputing/lambda_function.zip --handler weather_la
mbda_michelle.main.lambda_handler --role arn:aws:iam::292274580527:role/lambda_i
ce191
```

To test that our lambda was created successfully, we invoke our function and we try to search for an id. As shown bellow, we obtain a statusCode 200.

```
[CV9FCYQ4XQ:CloudComputing mxm0822$ aws lambda invoke --function-name weather_lam
bda_michelle --cli-binary-format raw-in-base64-out --payload '{"id": "009930"}'
response_michelle.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

We then proceed to create our rest API:

```
[CV9FCYQ4XQ:CloudComputing mxm0822$ aws apigateway create-rest-api --name weather
_api_michelle
{
    "id": "hxtkp0eql7",
    "name": "weather_api_michelle",
    "createdDate": "2023-03-17T18:56:20-07:00",
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "EDGE"
        ]
    },
    "disableExecuteApiEndpoint": false
}
```

We created some resources that were crated in our API, like so:

Some resources created where:

```
[CV9FCYQ4XQ:CloudComputing mxm0822$ aws apigateway get-resources --rest-api-id hx
tkp0eql7
{
    "items": [
        {
            "id": "21hz6g",
            "parentId": "a5tpwt",
            "pathPart": "{id}",
            "path": "/weather/{id}"
        },
        {
            "id": "a5tpwt",
            "parentId": "xwnu0w5obk",
            "pathPart": "weather",
            "path": "/weather"
        },
        {
            "id": "xwnu0w5obk",
            "path": "/"
        }
    ]
}
```

To create a resource we must indicate the parent id (root resource) and the resource id it will be associated to. Just as the examples bellow:

```
CV9FCYQ4XQ:CloudComputing mxm0822$ aws apigateway create-resource --rest-api-id
hxtkp0eql7 --parent-id xwnu0w5obk --path-part weather
{
    "id": "a5tpwt",
    "parentId": "xwnu0w5obk",
    "pathPart": "weather",
    "path": "/weather"
}
```

```
CV9FCYQ4XQ:CloudComputing mxm0822$ aws apigateway create-resource --rest-api-id
hxtkp0eql7 --parent-id a5tpwt --path-part {id}
{
    "id": "21hz6g",
    "parentId": "a5tpwt",
    "pathPart": "{id}",
    "path": "/weather/{id}"
}
```

We then create the integration (proxy) to our lambda function, specifying the endpoints we want to be executed. To do this we run the following command:

```
CV9FCYQ4XQ:CloudComputing mxm0822$ aws apigateway put-integration --rest-api-i
d hxtkp0eql7 --resource-id qm80x0 --http-method GET --integration-http-method
POST --type AWS_PROXY --uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-3
1/functions/arn:aws:lambda:us-east-1:292274580527:function:weather_lambda_mich
elle/invocations
{
    "type": "AWS_PROXY",
    "httpMethod": "POST",
    "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:
aws:lambda:us-east-1:292274580527:function:weather_lambda_michelle/invocations
",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "cacheNamespace": "qm80x0",
    "cacheKeyParameters": []
}
```

As we did in class, we must do this for /students and /weather, each with their corresponding /{id} endpoint.

Having done this, we must deploy our API, since for now is created already created, but in kind of pending mode. To do the deployment, we must run the following command:

```
CV9FCYQ4XQ:CloudComputing mxm0822$ aws apigateway create-deployment --rest-api-i
d hxtkp0eql7 --stage-name dev --description 'Deploy all the things'
```

3. Add authorization to your API Gateway API. Only valid user is admin and passwordabc123!@#.

For this, we can use AWS Cognito service. It will provide us with the functionality to add registration and login specifications, to control who has access to our page.

First, we must create a user pool and indicate the name, by running the following command:

```
CV9FCYQ4XQ:CloudComputing mxm0822$ aws cognito-idp create-user-pool --pool-name
michelle_pool

An error occurred (AccessDeniedException) when calling the CreateUserPool operat
ion: User: arn:aws:iam::292274580527:user/michelleandrea.muniz@cetys.edu.mx is n
ot authorized to perform: cognito-idp:CreateUserPool on resource: * because no i
dentity-based policy allows the cognito-idp:CreateUserPool action
```

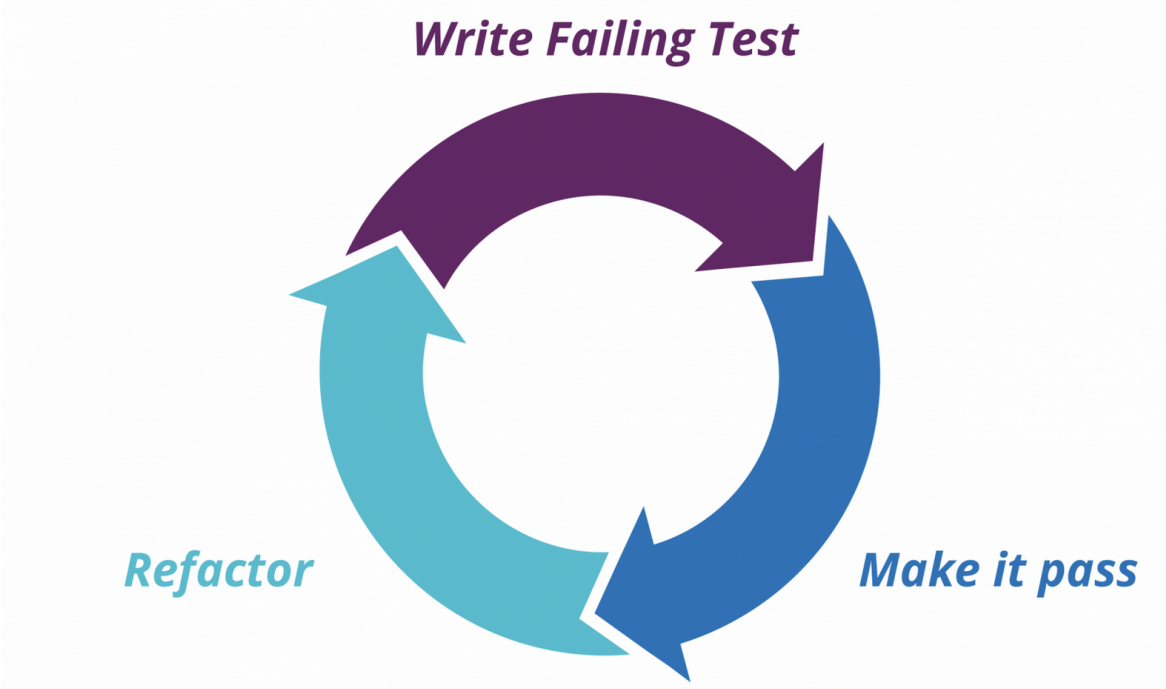
4. Read the Test Driven Development is the best thing that has happened to softwaredesign article and write a summary and opinions about it.

Summary

This article dives into the importance of applying TDD (Test Driven Development) and some key elements that are beneficial for software design.

TDD Cycles

The iterative approach that TTD applies, supports software to evolve into a working and more elegant solution. This process is different from others, since it doesn't take as much time like normal evolutionary processes.



- *Purple phase:* a failing test is written.
- *Dar blue phase:* minimum code to pass the test created prior
- *Blue phase:* the implementation of the test is cleaned

Difference between test driven by code and code driven by tests

1. When tests are driven by code: Tests are made after code is written. Could be tested method-by-method, success scenarios and failure scenarios. Since it passes all the tests made and code is running already in production, you could say its functionality is confirmed. The author says that this approach has a lack of implementation and design.

The principles questions that the test-first approach answers are:

- When should I stop adding new tests?
- Am I sure the implementation is finished?

2. When code is driven by tests: There are multiple benefits for this approach. Firstly, tests are verifying behaviors and not implementation details. Second, code is written always with up-to-date documentation. If there is any doubt about a module, a unit or a component, looking through tests a help clarify any questions. Thi*rd, writing tests before actual implementation forces us to ask the question: What do I expect from the code?*
    - It allows you to determine when to stop creating tests.
    - You introduce a new requirement by adding the new red test. Evolution of the implementation is more natural.

TDD on code impossible to test

- When code is written first, there is no determine way to write a proper unit test. For example, when objects are calling external services.
- In reality there is no safe way to verify functionality.
- When code uses dates and we are using static methods such as LocalDateTime.now() complicated the creation of a test and may failed.
    - Test are nor repeatable.
- When tests are too long in the preparation phase and complicated as well.
    - Can either mean there are too many dependencies or single responsibility principle is broken.

Opinion

I definitely think TDD has enormous benefits for software design, especially because it helps requirements meet the wanted and correct implementation. Even though some people don’t really view tests as important, they really are. Tests are the ones that will give fast and more in debt feedback, even before the firs production code line is written.

From personal experience in my work. TDD is a great approach for code that wasn’t thought at the beginning of writing to be able to do unit tests. Not doing this really complicates things when wanting to create unit tests and have the higher code coverage possible. Also, it can really take a long time trying to understand the code and determine what is the approach that we want to take for the testing.