### Nivel 2

#### Objetivos:

* Ampliar el microservicio par que implemente la llamada [POST] /active. El estado, la ciudad y el país se deberá almacenar en una base de datos relacional.
* Modificar el microservicio para que la llamada [GET] /active obtenga sus resultados desde la base de datos.
* Orquestar el funcionamiento del microservicio con el de la base de datos haciendo uso de docker-compose. La base de datos en concreto es indiferente, pero se recomienda utilizar postgres, mysql o mariadb
* Crear una imagen docker que contenga dicho microservicio y publicarla en dockerhub

#### Entregables:

* Documento explicativo del trabajo realizado
* Archivo docker-compose

#### Pruebas a superar:

Arrancar los servicios haciendo:

docker-compose up

Pruebas con curl haciendo un POST a una ciudad de un pais arbitrario y acto seguido obteniendo los valores con el GET correspondiente

Explicacion

1. Se creo un archivo .yml para la automatizacion del orquestado con docker-compose

# Use root/example as user/password credentials

version: '3.1'

services:

mysql:

container\_name: nivel2-db

command: --default-authentication-plugin=mysql\_native\_password

build:

context: .

dockerfile: db/Dockerfile

restart: always

adminer:

image: adminer

restart: always

ports:

- 8080:8080

python:

container\_name: nivel2-app

build:

context: .

dockerfile: Dockerfile

restart: always

ports:

- 8000:8000

2. Se creo un archivo init.sql para llenar con datos la base de datos desde el momento de creacion.

CREATE TABLE country(

id int NOT NULL auto\_increment,

country varchar(2),

PRIMARY KEY (id)

);

CREATE TABLE city(

id int NOT NULL auto\_increment,

city varchar(255),

id\_country int,

PRIMARY KEY (id),

active tinyint(1),

FOREIGN KEY (id\_country) REFERENCES country(id)

);

INSERT INTO `country` (`country`) VALUES ('NI');

INSERT INTO `country` (`country`) VALUES ('US');

INSERT INTO `city` (`city`, `id\_country`, `active`) VALUES ('LEON', '1', '1');

INSERT INTO `city` (`city`, `id\_country`, `active`) VALUES ('CHINANDEGA', '1', '0');

3. Se creo una sub carpeta para definir las caracteristicas de la imagen de la base de datos.

En la sub carpeta llamada db se encuentra un archivo Dockerfile

FROM mysql:latest

ENV MYSQL\_ROOT\_PASSWORD toor

ENV MYSQL\_HOST 127.0.0.1

ENV MYSQL\_DATABASE nivel2-db-mysql

ENV MYSQL\_USER nivel2-user

ENV MYSQL\_PASSWORD nivel2

ADD ./init.sql /docker-entrypoint-initdb.d

EXPOSE 3306

4. Volviendo a la carpeta raiz, tenemos otra sub carpeta llamada nivel2\_app en la cual se encuentran dos archivos: nivel2.py y requirements.txt

nive2.py tiene todos los microservicios para servir los diferentes endpoints con flask

from flask import Flask, jsonify, request, Response

import json

import mysql.connector

#from flask\_cors import CORS, cross\_origin

app = Flask(\_\_name\_\_)

def Connection():

return mysql.connector.connect(user='root', host='nivel2-db', port='3306', password='toor', database='nivel2-db-mysql')

@app.route('/active', methods=['GET', 'POST', 'PUT'])

def active():

if request.method == 'GET':

country = request.args.get("country", default = "no country", type = str)

city = request.args.get("city", default = "no city", type = str)

db = Connection();

try:

sql\_select = "select city.active, city.city, country.country from city inner join country on country.id = city.id\_country where country.country LIKE '%{}%' and city.city LIKE '%{}%'".format(country, city)

cursor = db.cursor()

cursor.execute(sql\_select)

records = cursor.fetchall()

if records == None:

info = {"Response": "No hay parametros o no hay resultados"}

return jsonify(info)

else:

for row in records:

if row[0] == 0:

v\_active = False

else:

v\_active = True

info = {

"active": "{}".format(v\_active),

"country": "{}".format(row[2]),

"city": "{}".format(row[1])

}

except Exception as e:

print("Error in SQL:\n", e)

finally:

db.close()

return jsonify(info)

if request.method == 'POST':

req\_data = request.get\_json()

country = req\_data['country']

city = req\_data['city']

db = Connection();

try:

sql\_insert = "INSERT INTO `country` (`country`) VALUES ('{}')".format(country)

cursor = db.cursor()

cursor.execute(sql\_insert)

db.commit()

sql\_select = "select id from country where country = '{}'".format(country)

cursor.execute(sql\_select)

records = cursor.fetchall()

for row in records:

id\_v = row[0]

sql\_insert\_c = "INSERT INTO `city` (`city`, `id\_country`, `active`) VALUES ('{}', '{}', '1')".format(city, id\_v)

cursor.execute(sql\_insert\_c)

db.commit()

info = {"Response": "Datos guardados correctamente"}

except Exception as e:

print("Error in SQL:\n", e)

finally:

db.close()

return jsonify(info)

if request.method == 'PUT':

req\_data = request.get\_json()

country = req\_data['country']

city = req\_data['city']

db = Connection();

try:

sql\_select = "select city.id,city.active from city inner join country on country.id = city.id\_country where country.country like '%{}%' and city.city like '%{}%'".format(country, city)

cursor = db.cursor()

cursor.execute(sql\_select)

records = cursor.fetchall()

for row in records:

city\_id = row[0]

active = row[1]

if active == 0:

active\_n = 1

info = {"Response": "Se activo la venta"}

else:

active\_n = 0

info = {"Response": "Se desactivo la venta"}

sql\_update = "update city set active = {} where city.id = {}".format(active\_n, city\_id)

cursor.execute(sql\_update)

db.commit()

except Exception as e:

print("Error in SQL:\n", e)

finally:

db.close()

return jsonify(info)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, host='0.0.0.0', port='8000')

5. El archivo requeriments.txt contien los paquetes necesarios para la ejecucion

Flask==1.1.1

Flask-Migrate==2.5.2

Flask-SQLAlchemy==2.4.0

mysql-connector

**Modo de prueba:**

1. Para arrancar los contenedores nos ubicamos sobre la carpeta raiz en donde se encuentra el archivo nivel2.yml

Ejecutamos la siguiente orden:

sudo docker-compose -f nivel2.yml up

Una vez que arranquen tendremos tres contenedores corriendo: nivel2-db, nivel2-app y adminer

El contenedor nivel2-db estara encargado de tener la base de datos

El contenedor nivel2-app estara encargado de tener la aplicacion Flask

El contenedor adminer tendra una interfaz para administrar el motor de base de datos servidor por nive2-db con las credenciales usuario: root y clave: toor

2. Probamos nuestros endpoints:

El siguiente comando utiliza el motodo GET del endpoint para obtener los datos desde la base de datos:

curl "localhost:8000/active?country=NI&city=LEON"

Devuelve un JSON con la informacion correspondiente a que si la venta esta activa en la ciudad de LEON

{

"active": "True",

"city": "LEON",

"country": "NI"

}

El siguiente comando utiliza el metodo POST del endpoint para guardar los datos en la base de datos:

curl -X POST -H "Content-Type: application/json" -d '{"country":"JP","city":"TOKIO"}' "localhost:8000/active"

Devolviendo un JSON como respuesta satisfactoria.

{

"Response": "Datos guardados correctamente"

}

Una vez con los datos guardados en la base de datos ya podemos consultar el estado de la nueva ciudad y pais.

Para esto utilizamos el siguiente comando:

curl "localhost:8000/active?country=jp&city=tokio"

y nos devolvera un JSON con la siguiente informacion

{

"active": "True",

"city": "TOKIO",

"country": "JP"

}

El siguiente comando utiliza el metodo PUT del endpoint para actualizar el estado de activacion de la venta en una ciudad de un pais.

Para esto utilizamos el siguiente comando para comprobar el estado.

Tomando en cuenta el ejemplo anterior de la ciudad de TOKIO con el pais JP que se encuentra activa usamos:

curl -X PUT -H "Content-Type: application/json" -d '{"country":"JP","city":"TOKIO"}' "localhost:8000/active"

Y nos devolvera un JSON con la informacion siguiente:

{

"Response": "Se desactivo la venta"

}

si volvemos a ejecutar la misma orden con el metodo PUT del endpoint entonces volveremos a activar la venta en la ciudad.