Universidad de Ingenieria y Tecnologia
Departamento de Ciencias de la Computación

Ingeniería de Software I

Lab VIII: Testing The Basics
2024 - II

TEAM	**Mayllu**
INTEGRANTES	- Kalos Lazo (Líder) - Lenin Chavez - Matias Castillo - Gabriel Blanco - Valeria Valdez - Jose Tipula
Puntos EC	**2 ptos**

Background:
Unit Test
Integration Test

CASO DE ESTUDIO (level: basic):

1. **FizzBuzz (UNIT TEST):**

- **Baje el repo https://gist.github.com/jaysonrowe/1592775#file-fizzbuzz-py-L3**
- **¿Cuál es el happy path ? ¿Hay algún edge case ?**

El repositorio tiene un algoritmo FizzBuzz el cual evalúa y realiza:

1. Retorna Fizz si el número ingresado es divisible entre 3
2. Retorna Fizz si el número ingresado es divisible entre 5
3. Retorna FizzBuzz si el número ingresado es divisible entre 3 y 5

El happy path es aquel camino que se supone que el usuario seguirá considerando las condiciones perfectas, es la prueba más básica de testeo, todo funciona como se espera. En este caso se le pasa como input un número entero que está en un rango de 1 hasta n, siendo n un número entero.

```python
def fizzbuzz(n):
    if n % 3 == 0 and n % 5 == 0:
        return 'FizzBuzz'
    elif n % 3 == 0:
        return 'Fizz'
    elif n % 5 == 0:
        return 'Buzz'
    else:
        return str(n)

# Happy path
n = 15
print(fizzbuzz(n))
```


```python
# Happy path
n = 15
print(fizzbuzz(n))
```


No obstante, el edge case más fácil de identificar, que posiblemente ocurra en un llamado por un agente externo (sin saber documentación) es que se le pase como dato de entrada un tipo no aceptado, como una cadena, digamos '10', se rompe el algoritmo, pues n%3 no está determinado, lo mismo para distintas estructuras: números flotantes, listas ó booleanos.

```python
# Edge case
n = '10'
print(fizzbuzz(n))
```

En caso se maneje la conversión de tipos resultan nuevos edge cases como los valores vacíos (no se recibe entrada), espacios en blanco en las cadenas ó cadenas con carácteres no aceptados.

- **¿Agregue los unit test necesarios para un coverage del 100% ?**

```python
# fizzbuzz.py
def fizzbuzz(n):
    # 1. Validar que la entrada sea un número
    if not isinstance(n, int):
        return 'Error, el argumento debe ser un número entero'

    # 2. Validar que el número sea positivo
    if n ≤ 0:
        return 'Error, el número debe ser positivo'

    # 3. Lógica del fizzbuzz
    if n % 3 == 0 and n % 5 == 0:
        return 'FizzBuzz'
    elif n % 3 == 0:
        return 'Fizz'
    elif n % 5 == 0:
        return 'Buzz'
    else:
        return str(n)
```

```python
# test_fizzbuzz.py
import pytest
from fizzbuzz import fizzbuzz

def test_fizzbuzz_invalid_type():
    assert fizzbuzz("15") == 'Error, el argumento debe ser un número entero'
    assert fizzbuzz(3.5) == 'Error, el argumento debe ser un número entero'
    assert fizzbuzz([3]) == 'Error, el argumento debe ser un número entero'
    assert fizzbuzz(None) == 'Error, el argumento debe ser un número entero'

def test_fizzbuzz_invalid_numbers():
    assert fizzbuzz(0) == 'Error, el número debe ser positivo'
    assert fizzbuzz(-1) == 'Error, el número debe ser positivo'
    assert fizzbuzz(-15) == 'Error, el número debe ser positivo'

def test_fizzbuzz_divisible_by_three():
    assert fizzbuzz(3) == 'Fizz'
    assert fizzbuzz(6) == 'Fizz'
    assert fizzbuzz(9) == 'Fizz'

def test_fizzbuzz_divisible_by_five():
    assert fizzbuzz(5) == 'Buzz'
    assert fizzbuzz(10) == 'Buzz'
    assert fizzbuzz(20) == 'Buzz'

def test_fizzbuzz_divisible_by_both():
    assert fizzbuzz(15) == 'FizzBuzz'
    assert fizzbuzz(30) == 'FizzBuzz'
    assert fizzbuzz(45) == 'FizzBuzz'

def test_fizzbuzz_not_divisible():
    assert fizzbuzz(1) == '1'
    assert fizzbuzz(2) == '2'
    assert fizzbuzz(4) == '4'
    assert fizzbuzz(7) == '7'
```

```
kaloslazo@archlinux ~/D/Mayllu/semana_10 → pytest --cov=fizzbuzz test_fizzbuzz.py --cov-report term-missing
========== test session starts ==========
platform linux -- Python 3.11.10, pytest-7.4.4, pluggy-1.0.0
rootdir: /home/kaloslazo/Documents/Mayllu/semana_10
plugins: anyio-4.6.2, cov-5.0.0
collected 6 items

test_fizzbuzz.py ......                                                    [100%]

---------- coverage: platform linux, python 3.11.10-final-0 ----------
Name          Stmts   Miss  Cover   Missing
-------------------------------------------
fizzbuzz.py      12      0   100%
-------------------------------------------
TOTAL            12      0   100%


========== 6 passed in 0.02s ==========
```

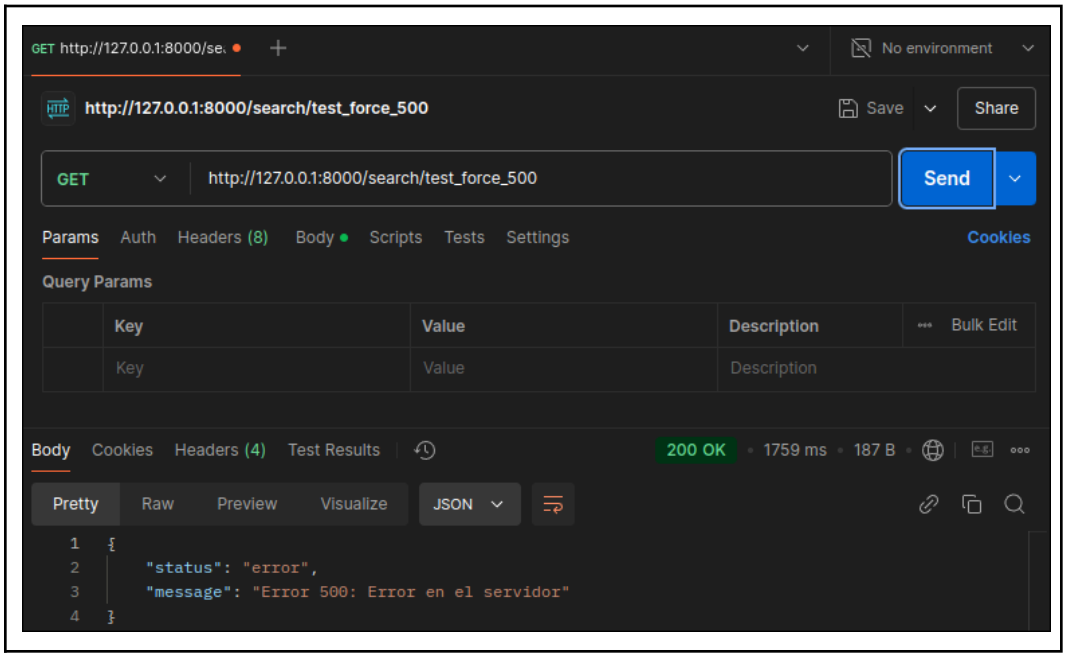
2. Pokedex (INTEGRATION TEST + MOCK)

- Haga una api que permita buscar pokemones por nombres de la siguiente API publica: https://pokeapi.co/ en el lenguaje de su preferencia.

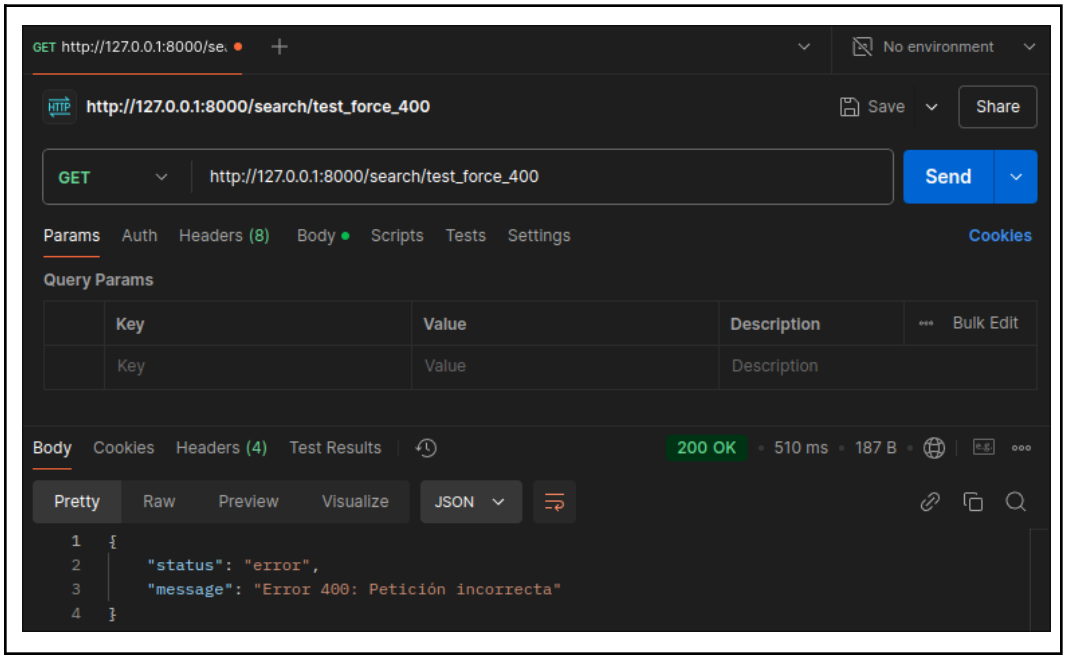
```python
from fastapi import FastAPI
import requests

app = FastAPI()
API_URL = "https://pokeapi.co/api/v2/pokemon/"

@app.get("/")
async def root():
    return {"message": "Utiliza /search/{pokemonNombre} para buscar un pokemon por su nombre"}

@app.get("/search/{pokemon_nombre}")
async def search_pokemon(pokemon_nombre: str):
    try:
        response = requests.get(f"{API_URL}{pokemon_nombre.lower()}")

        # Test -> vamos a forzar errores
        if pokemon_nombre.lower() == "test_force_500":
            response.status_code = 500

        elif pokemon_nombre.lower() == "test_force_400":
            response.status_code = 400

        if response.status_code == 200:
            pokemon_data = response.json()
            # Solo retornamos los datos más relevantes
            return {
                "status": "success",
                "data": {
                    "nombre": pokemon_data["name"],
                    "altura": pokemon_data["height"],
                    "peso": pokemon_data["weight"],
                    "tipos": [tipo["type"]["name"] for tipo in pokemon_data["types"]]
                }
            }

        elif response.status_code == 400:
            return {
                "status": "error",
                "message": "Error 400: Petición incorrecta"
            }
        elif response.status_code == 500:
            return {
                "status": "error",
                "message": "Error 500: Error en el servidor"
            }
        else:
            return {
                "status": "error",
                "message": f"Pokemon no encontrado (Status: {response.status_code})"
            }

    except requests.RequestException:
        return {
            "status": "error",
            "message": "Error al conectar con la API de Pokemon"
        }
```

En el código, se muestra cómo utilizando fastapi manejamos una petición a la API de pokeapi, donde se crea un endpoint /search/{pokemon nombre}, se evalúa de manera de fuerza bruta 3 casos: código 200 cuando todo estuvo OK, código 404 cuando existe una petición incorrecta como un pokemon que no existe ó código 400 cuando la petición es incorrecta, es decir se le introduce algún tipo de dato de entrada que no está definido. Tenemos casos para forzar errores y ver la salida

- **Elabore un postman que testee esta API. Subir el script a su repo.**
    - **¿Qué pasa si su API devuelve 500 ? ¿Cuál es el json body del response ?**



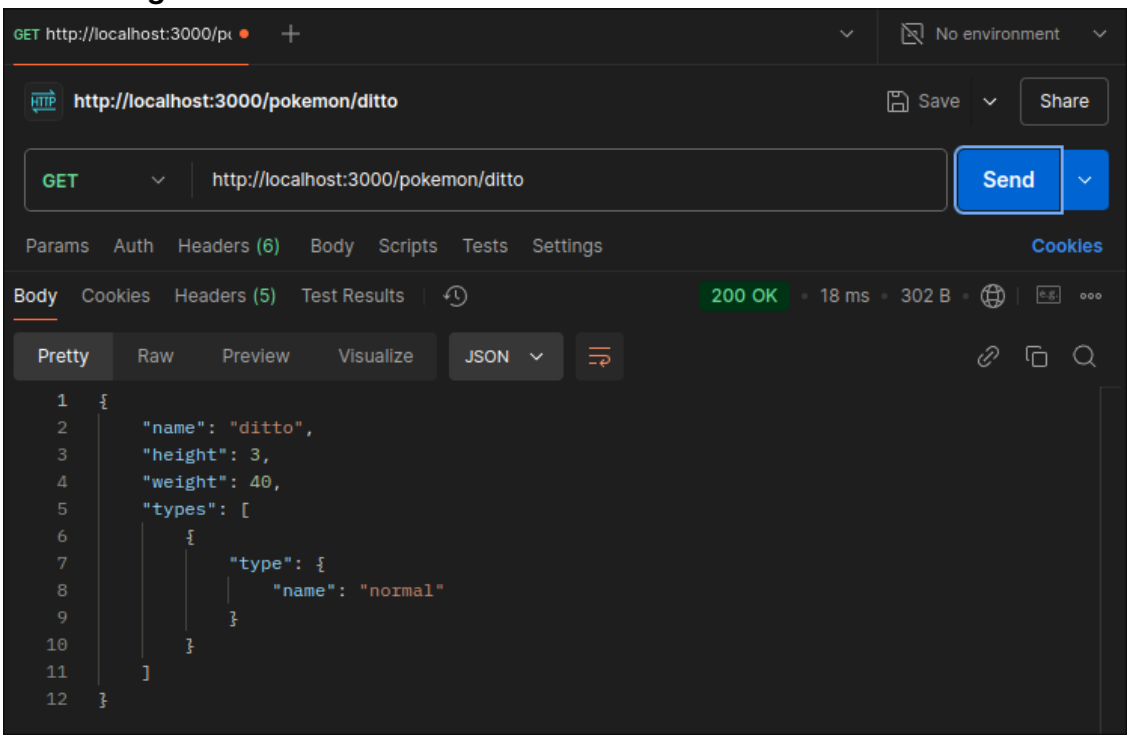
    - **¿Qué pasa si su API devuelve 400 ? ¿Cuál es el json body del response ?**



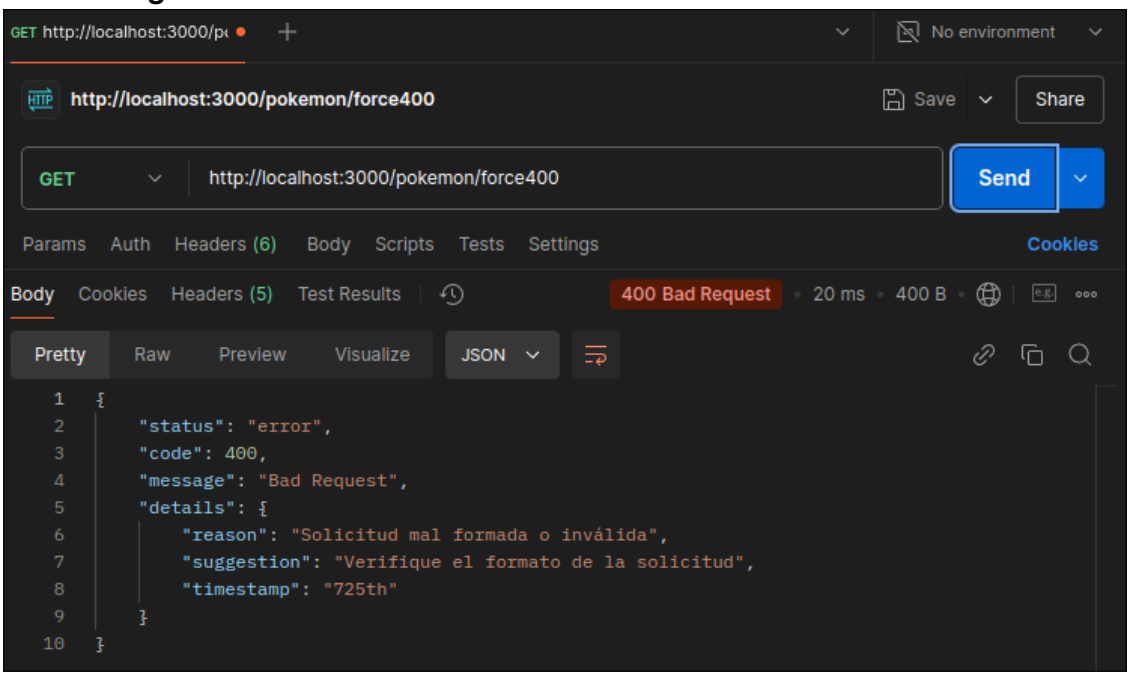
- **Imagine ahora que los viernes Poké API no está disponible por lo que su equipo de QA le sugiere usar Mockoon para implementar mocks: https://mockoon.com/.**
    - Mockear 200, 400 y 500 como respuestas de este mock. En base a estas respuestas cómo responde su API ?

Mockon nos permite definir qué datos retornar, simular códigos de estado o añadir latencia, esto nos sirve para testear nuestro backend. Es interesante pues sólo necesita un archivo de configuración que tiene respuestas posibles y se comparte de manera sencilla. Se tomo como referencia este link: https://raw.githubusercontent.com/mockoon/mock-samples/main/mock-apis/data/botschaftlocal.json

Mock código 200

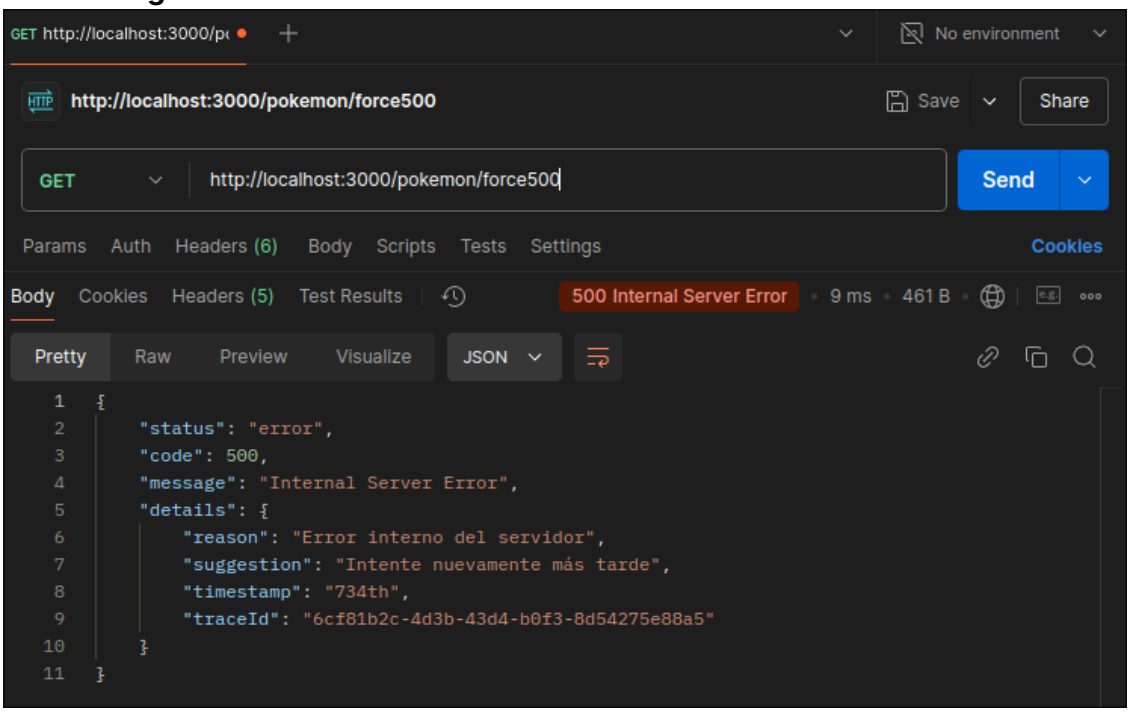


Mock código 400




```
{"app":"mockoon-server","environmentName":"Pokemon Mock API","environmentUUID":"0a49957d-a44e-4641-85dd-c14190199cff","level":"info","message":"Transaction recorded","requestMethod":"GET","requestPath":"/pokemon/force400","requestProxied":false,"responseStatus":400,"timestamp":"2024-10-28T03:23:22.875Z"}
```


Mock código 500




```
{"app":"mockoon-server","environmentName":"Pokemon Mock API","environmentUUID":"0a49957d-a44e-4641-85dd-c14190199cff","level":"info","message":"Transaction recorded","requestMethod":"GET","requestPath":"/pokemon/force500","requestProxied":false,"responseStatus":500,"timestamp":"2024-10-28T03:23:32.296Z"}
```