



**UNIVERSIDAD DE
SAN BUENAVENTURA**

Andres Felipe Lasso Perdomo

30000097453

andresfelipe.lasso@gmail.com

FizzBuzz web

Técnicas de programación avanzadas

2/05/2024

Problemática

Se necesita realizar una API HTTP web de FizzBuzz la cual recibirá diferentes solicitudes HTTP y generará las correspondientes respuestas al cliente y acciones en una base de datos, esperadas por cada solicitud. En términos generales es una api que permite subir, solicitar, modificar o eliminar un registro en una base de datos y después darle un mensaje de respuesta al cliente. En este problema se trata de una base de datos que almacena registros con dos atributos, un número y su conversión FizzBuzz.

requisitos y restricciones:

1. El servidor de FizzBuzz se apoya en una estrategia de persistencia, en la que inicia con 100 números y sus respectivos valores de FizzBuzz. La persistencia puede ser lograda mediante archivos o una base de datos, pero se espera que se aplique una estrategia de inversión de dependencias.
2. Los 100 números iniciales corresponderán a los empleados en ejercicios previos y disponibles en Replit.
3. Una petición GET para un número presente en el repositorio, y enviado como parte de la URL, retorna su respectivo valor de FizzBuzz y el código HTTP 200. La API sugerida es "fb", pero se tiene la libertad de especificar otra ruta.
4. Una petición GET para un número NO presente en el repositorio, y enviado como parte de la URL, retorna la cadena "Not Found" y el código HTTP 404.

5. Una petición POST, para un número NO presente (o no activo) en el repositorio, y enviado como parte de la URL, agrega el número y su respectivo valor de FizzBuzz al repositorio, retorna el valor de FizzBuzz agregado, y el código HTTP 201. Si al número indicado, se le ha realizado un borrado lógico, este se reactivará, y se retornará el valor de FizzBuzz, acompañado del código HTTP 200.
6. Una petición POST, para un número presente en el repositorio, y enviado como parte de la URL, NO modifica el repositorio, retorna el valor de FizzBuzz respectivo, y el código HTTP 409.
7. Una petición POST, para la ruta "range", y un body indicando límite inferior y límite superior, retornará todos los números y los valores de FizzBuzz en el intervalo [límite inferior, límite superior], acompañado del código HTTP 200. En caso de que no existan valores en el intervalo, se retornará cadena "Not Found" y el código HTTP 404. Se espera que no se emplee un algoritmo trivial o ingenuo para la verificación de la intersección de los números presentes en el repositorio, con la base de datos.
8. Una petición DELETE, acompañado de un número en la URL, para la ruta "fb", y presente en el repositorio, realizará un borrado lógico para el número, retornará la cadena "No Content" y el código HTTP 204.
9. Una petición DELETE, acompañado de un número en la URL, para la ruta "fb", y NO presente en el repositorio, retornará la cadena "Not Found" y el código HTTP 404.
10. Cualquier otra petición, debe retornar un código HTTP 400.

Pruebas:

La estrategia de pruebas elegida es el entorno de pruebas de postman, postman es una herramienta que simula la solicitud de requests con el protocolo HTTP a una API o servicio. Postman envía la solicitud y almacena la respuesta para verificar su contenido y realizar verificaciones de rendimiento, o test para saber si se obtuvo la respuesta u ocurre algún error inesperado.

Se realizó una prueba específica para cada requisito y sus vertientes. Por ejemplo en el requisito número 1, se hizo una prueba para verificar la respuesta de una solicitud Get de un número que existe en el repositorio y otra prueba para un número que no existe en el repositorio. A si se evalúan los dos posibles casos y sus determinadas respuestas.

Algunas pruebas necesitan que se realice una solicitud previa, esta se realiza directamente en la misma prueba por automatización, por ejemplo. Si se quiere probar que se reactiva el estado inactivo de un número previamente eliminado al realizar un POST con el número, primero se debe eliminar dicho número con una solicitud DELETE, ya que la API se inicia con todos los números en estado activo.

Red

Ya con las pruebas realizadas se mostrará el resultado de las pruebas en general, todas fallaran. Luego se mostrará cada prueba en específico. En regla general todas las pruebas verifican que el status de la respuesta sea el esperado y que el body tenga el mensaje esperado.

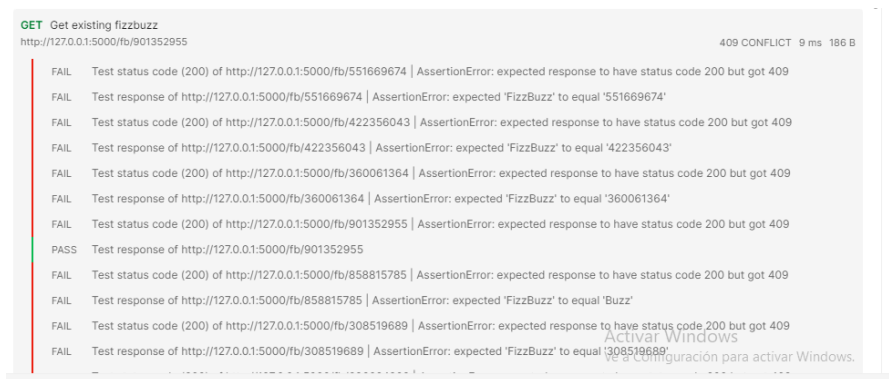
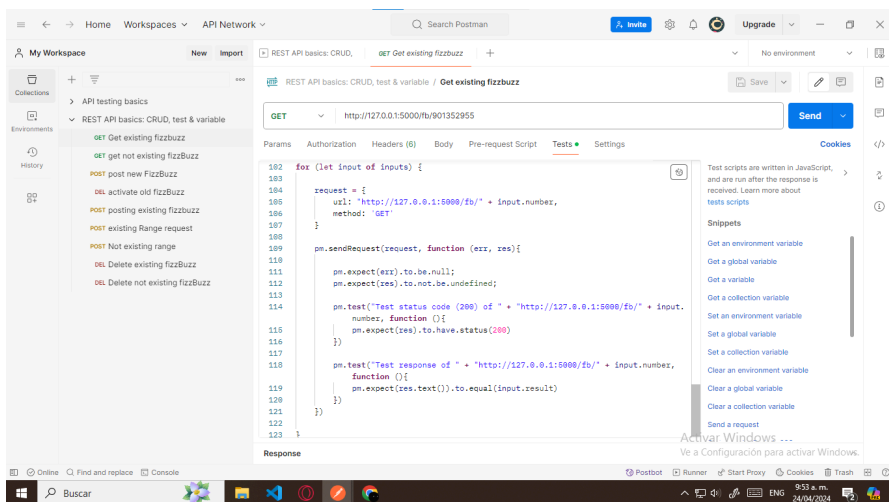
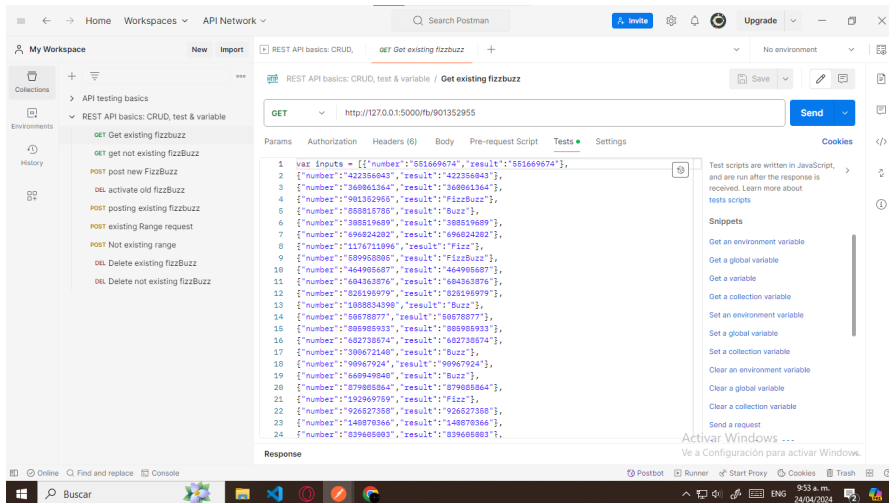
The screenshot shows the Postman interface with a collection named 'REST API basics: CRUD, test...' selected. The 'Run' button has been clicked, and the results are displayed. The 'RUN SUMMARY' table shows that all tests failed (indicated by red 'X' marks).

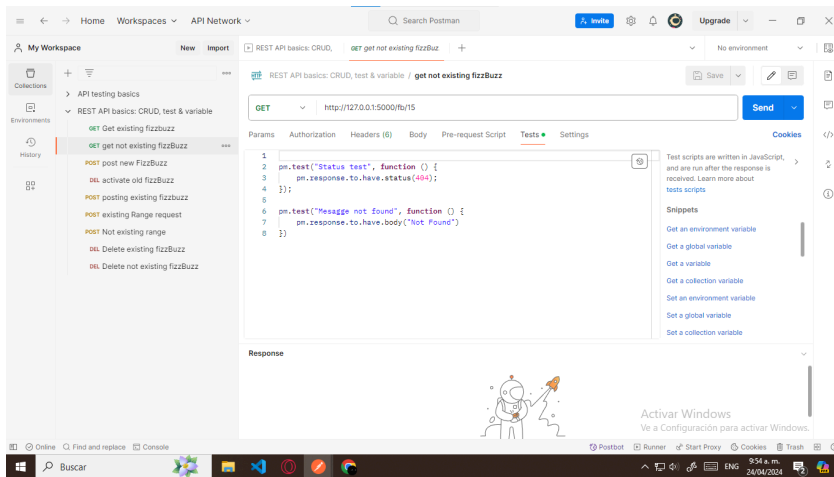
Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	6s 996ms	225	5 ms

Test Name	Pass/Fail
GET Get existing fizzbuzz	0 200 X
GET get not existing fizzBuzz	0 2 X
POST post new FizzBuzz	0 11 X
DELETE activate old fizzBuzz	0 2 X
POST posting existing fizzbuzz	0 2 X
POST existing Range request	0 2 X
POST Not existing range	1 1 X
DELETE Delete existing fizzBuzz	0 2 X
DELETE Delete not existing fizzBuzz	0 2 X

Este es el resumen de todas las pruebas que se realizaron, como se observa todas fallaron. En cada línea se muestra el nombre de la request, el tipo, a la derecha los test que pasaron en esa request y los test totales de esa request, Si se pasan todos los test las x rojas pasarían a verdes. En la request "Not existing range" una prueba paso ya que el código esperado es 404 que es el código por defecto que flask envía cuando el servicio no se encuentra.

Ahora se mostrará el código de cada prueba y las respuestas de las pruebas (una por página).





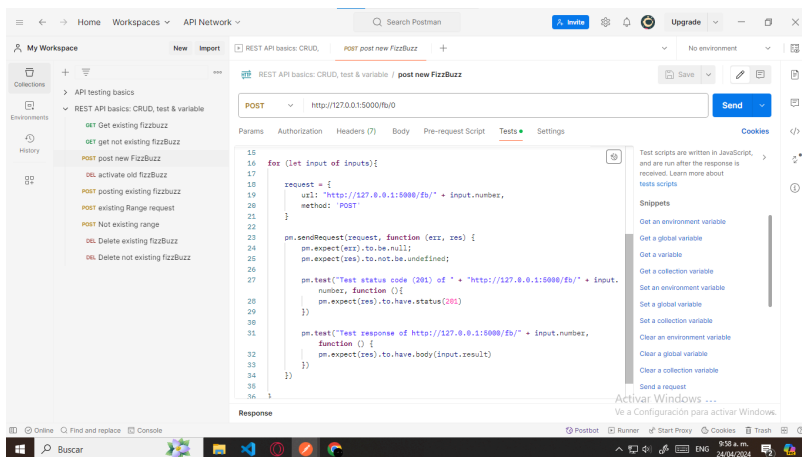
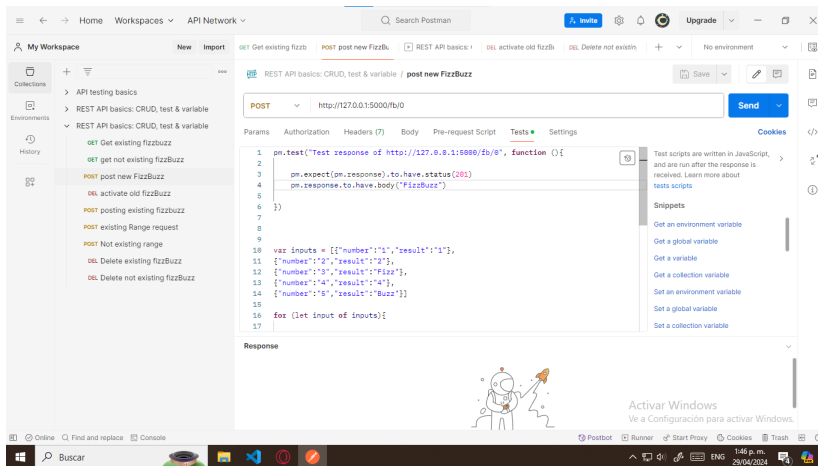
GET get not existing fizzBuzz

http://127.0.0.1:5000/fb/15

409 CONFLICT 6 ms 186 B

FAIL Status test | AssertionError: expected response to have status code 404 but got 409

FAIL Mesagge not found | AssertionError: expected response body to equal 'Not Found' but got 'FizzBuzz'



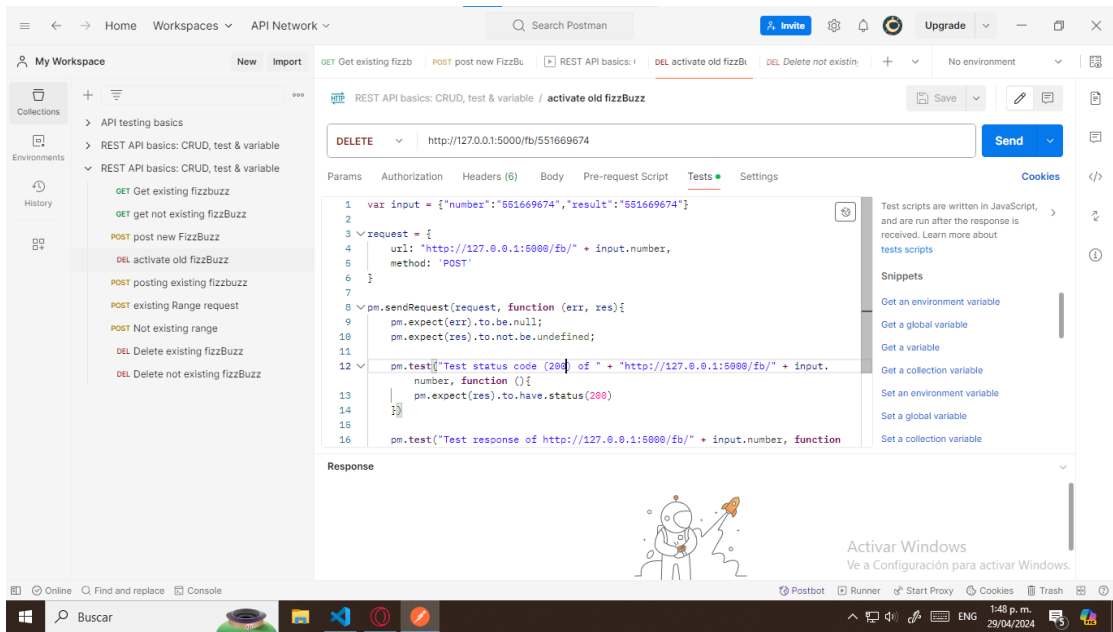
POST post new FizzBuzz

http://127.0.0.1:5000/fb/0

409 CONFLICT 5 ms 186 B

- FAIL Test response of http://127.0.0.1:5000/fb/0 | AssertionError: expected response to have status reason '201' but got 'CONFLICT'
- FAIL Test status code (201) of http://127.0.0.1:5000/fb/1 | AssertionError: expected response to have status code 201 but got 409
- FAIL Test response of http://127.0.0.1:5000/fb/1 | AssertionError: expected response body to equal '1' but got 'FizzBuzz'
- FAIL Test status code (201) of http://127.0.0.1:5000/fb/2 | AssertionError: expected response to have status code 201 but got 409
- FAIL Test response of http://127.0.0.1:5000/fb/2 | AssertionError: expected response body to equal '2' but got 'FizzBuzz'
- FAIL Test status code (201) of http://127.0.0.1:5000/fb/4 | AssertionError: expected response to have status code 201 but got 409
- FAIL Test response of http://127.0.0.1:5000/fb/4 | AssertionError: expected response body to equal '4' but got 'FizzBuzz'
- FAIL Test status code (201) of http://127.0.0.1:5000/fb/5 | AssertionError: expected response to have status code 201 but got 409
- FAIL Test response of http://127.0.0.1:5000/fb/5 | AssertionError: expected response body to equal 'Buzz' but got 'FizzBuzz'
- FAIL Test status code (201) of http://127.0.0.1:5000/fb/3 | AssertionError: expected response to have status code 201 but got 409
- FAIL Test response of http://127.0.0.1:5000/fb/3 | AssertionError: expected response body to equal 'Fizz' but got 'FizzBuzz'

Activar Windows



DELETE activate old fizzBuzz

http://127.0.0.1:5000/fb/551669674

409 CONFLICT 5 ms 186 B

- FAIL Test status code (201) of http://127.0.0.1:5000/fb/551669674 | AssertionError: expected response to have status code 201 but got 409
- FAIL Test response of http://127.0.0.1:5000/fb/551669674 | AssertionError: expected response body to equal '551669674' but got 'FizzBuzz'

REST API basics: CRUD, test & variable / **posting existing fizzbuzz**

POST http://127.0.0.1:5000/fb/292015719

Params Authorization Headers (7) Body Pre-request Script **Tests** Settings

```
1
2 let input = {"number": "292015719", "result": "Fizz"}
3
4 pm.test("Test status code (409) of http://127.0.0.1:5000/fb/292015719", function () {
5   pm.expect(pm.response).to.have.status(409)
6 })
7
8
9 pm.test("Test response of http://127.0.0.1:5000/fb/" + input.number, function () {
10   pm.expect(pm.response).to.have.body(input.result)
11 })
```

Response

Activar Windows
Ve a Configuración para activar Windows.

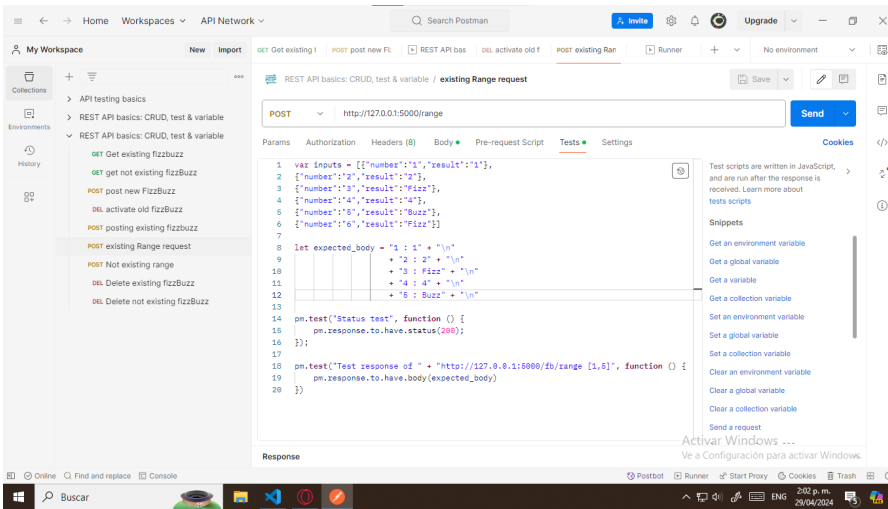
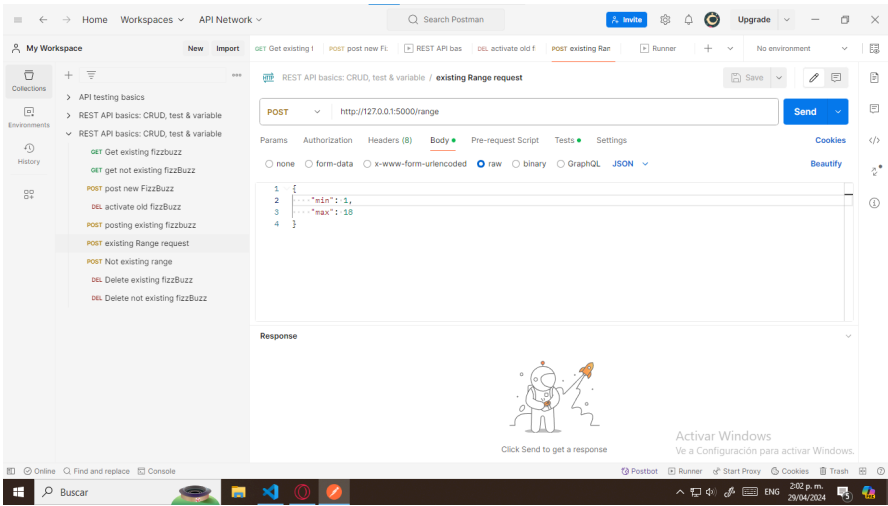
POST posting existing fizzbuzz

http://127.0.0.1:5000/fb/292015719

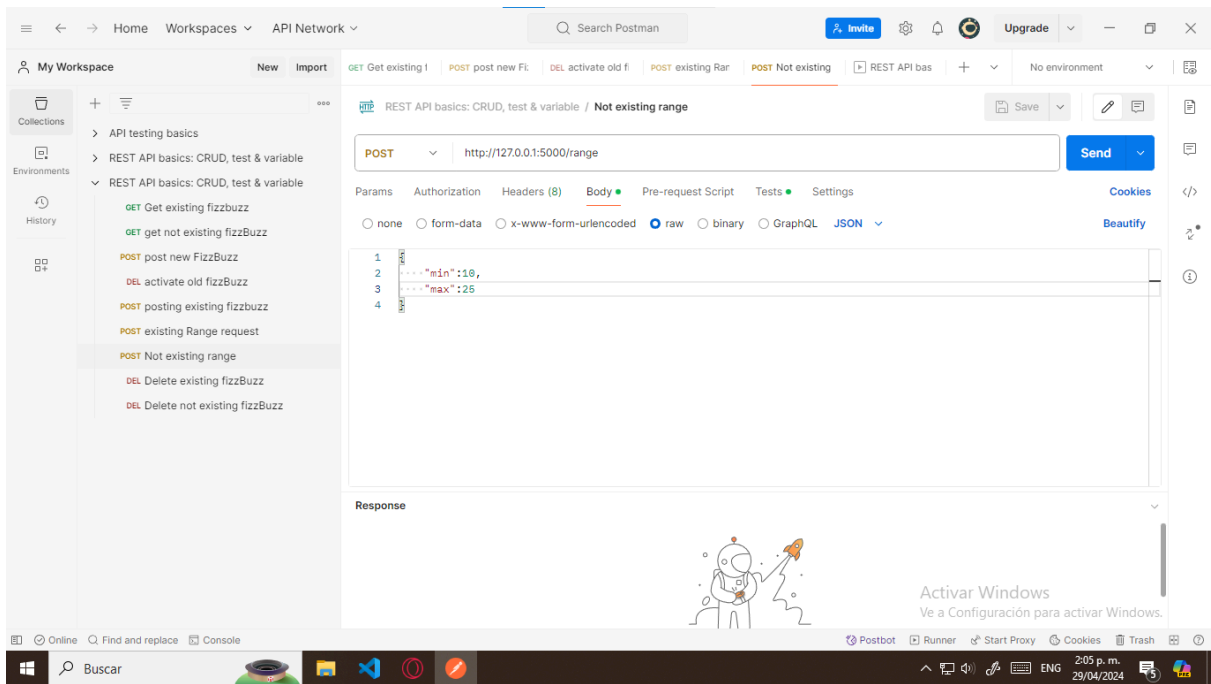
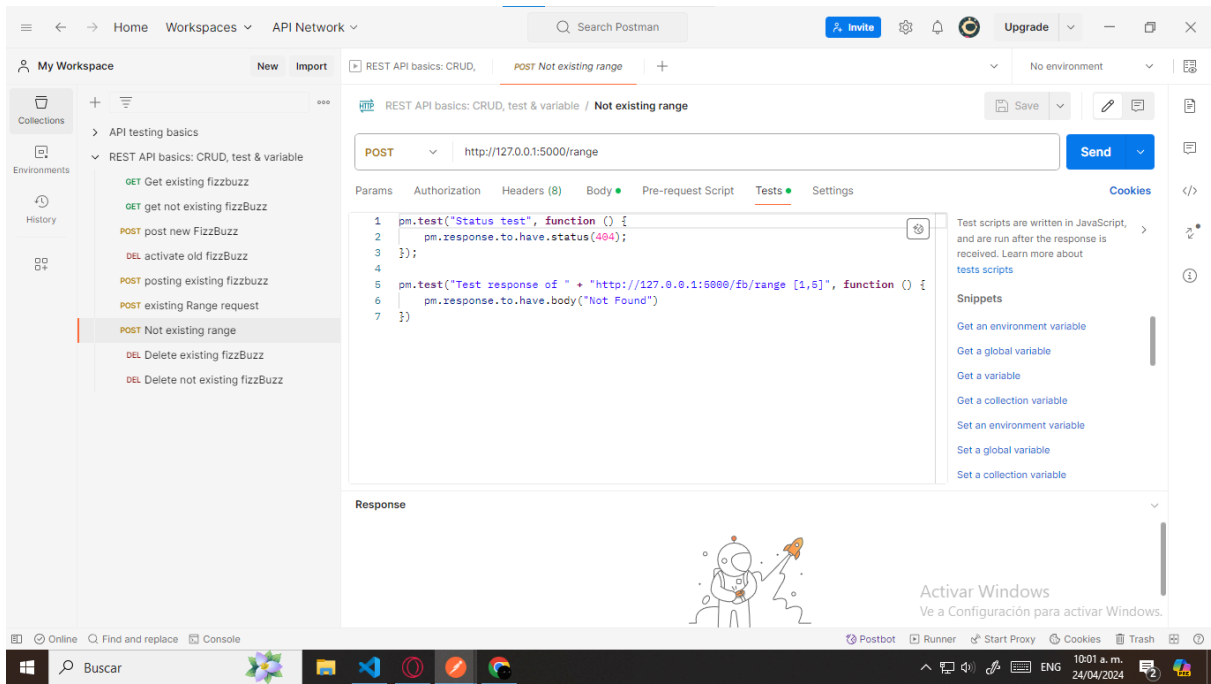
409 CONFLICT 5 ms 186 B

PASS Test status code (409) of http://127.0.0.1:5000/fb/292015719

FAIL Test response of http://127.0.0.1:5000/fb/292015719 | AssertionError: expected response body to equal 'Fizz' but got 'FizzBuzz'



POST existing Range request		http://127.0.0.1:5000/range		404 NOT FOUND	5 ms	184 B
FAIL	Status test	AssertionError: expected response to have status code 200 but got 404				
FAIL	Test response of http://127.0.0.1:5000/fb/range [1,5]	AssertionError: expected response body to equal 'Number: 1, result: 1 Number: 2, re...'				

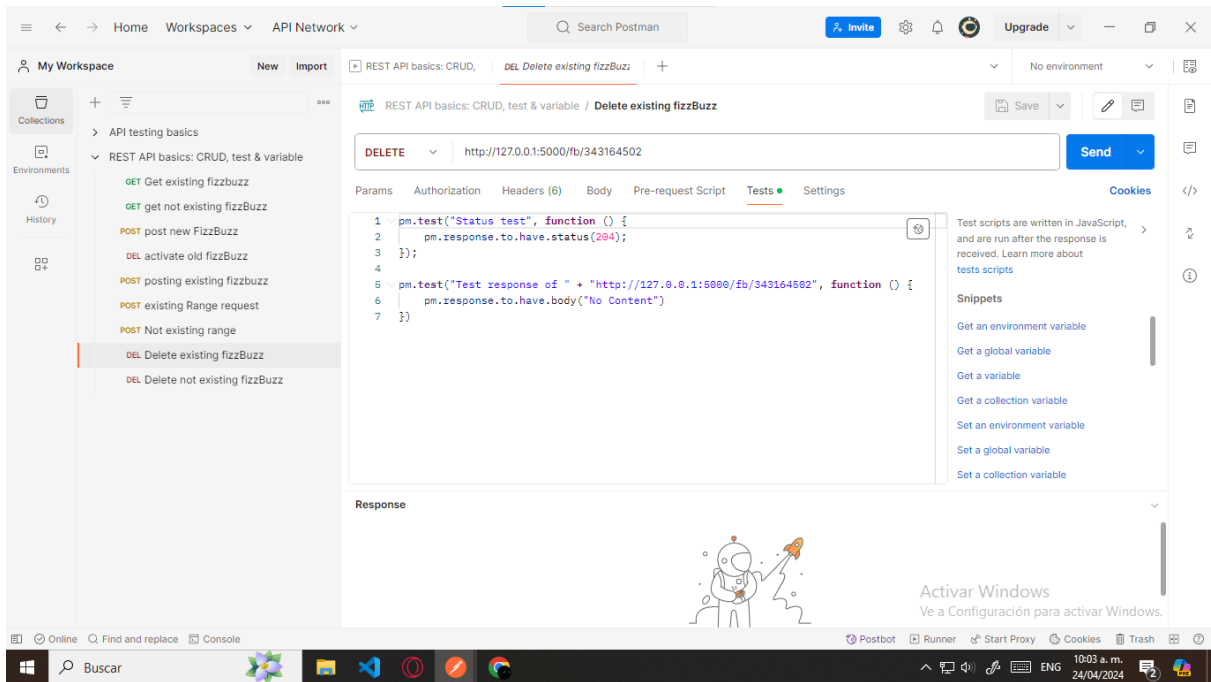


POST Not existing range
http://127.0.0.1:5000/range

404 NOT FOUND 6 ms 184 B

PASS Status test

FAIL Test response of http://127.0.0.1:5000/fb/range [1,5] | AssertionError: expected response body to equal 'Not Found' but got 'Range'



DELETE Delete existing fizzBuzz

http://127.0.0.1:5000/fb/343164502

409 CONFLICT 6 ms 186 B

FAIL Status test | AssertionError: expected response to have status code 204 but got 409

FAIL Test response of http://127.0.0.1:5000/fb/343164502 | AssertionError: expected response body to equal 'No Content' but got 'FizzBuzz'

REST API basics: CRUD, test & variable / **DEL Delete not existing fizzBuzz**

DELETE http://127.0.0.1:5000/fb/30

Params Authorization Headers Body Pre-request Script **Tests** Settings


```
1 pm.test("Status test", function () {
2   pm.response.to.have.status(404);
3 });
4
5 pm.test("Test response of " + "http://127.0.0.1:5000/fb/343164502", function () {
6   pm.response.to.have.body("Not Found");
7 })
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [test scripts](#)

Snippets

- [Get an environment variable](#)
- [Get a global variable](#)
- [Get a variable](#)
- [Get a collection variable](#)
- [Set an environment variable](#)
- [Set a global variable](#)
- [Set a collection variable](#)

Response



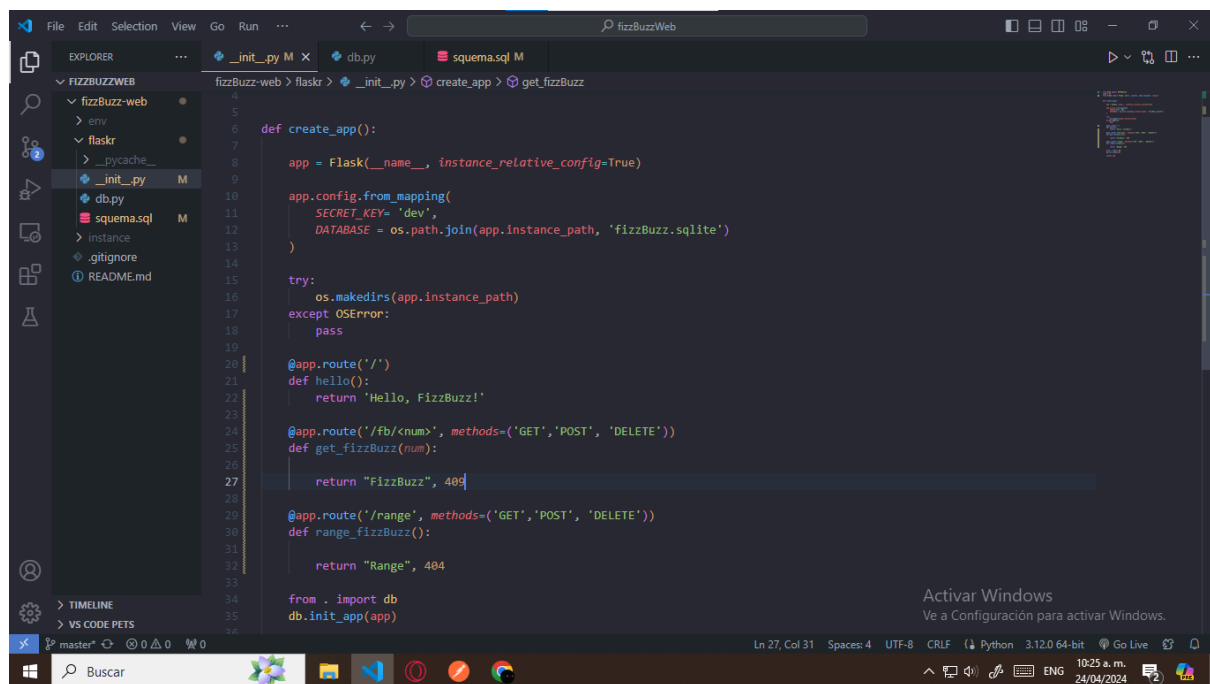
Activar Windows
Ve a Configuración para activar Windows.

DELETE Delete not existing fizzBuzz
http://127.0.0.1:5000/fb/30 409 CONFLICT 5 ms 186 B

FAIL Status test | AssertionError: expected response to have status code 404 but got 409

FAIL Test response of http://127.0.0.1:5000/fb/343164502 | AssertionError: expected response body to equal 'Not Found' but got 'FizzBuzz'

Activar Windows
Ve a Configuración para activar Windows.



```
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

def create_app():
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY="dev",
        DATABASE=os.path.join(app.instance_path, 'fizzBuzz.sqlite')
    )
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    @app.route('/')
    def hello():
        return 'Hello, FizzBuzz!'

    @app.route('/fb/<num>', methods=('GET', 'POST', 'DELETE'))
    def get_fizzBuzz(num):
        return "FizzBuzz", 404

    @app.route('/range', methods=('GET', 'POST', 'DELETE'))
    def range_fizzBuzz():
        return "Range", 404

    from . import db
    db.init_app(app)
```

Este es el código con el que se ejecutaron las pruebas, este código obviamente no pasa las pruebas pero abre la conexión a la api para que postman pueda realizar las solicitudes.

Green

Pasamos a la etapa verde donde el código implementado satisface nuestras pruebas realizadas anteriormente, lo que significa que funciona correctamente y como lo esperado. Antes de repasar el código y mostrar las pruebas se debe saber que se necesitan realizar unos pasos antes de correr el programa para que este y las pruebas funcionen correctamente.

- Se debe activar un entorno virtual de python y instalar el framework “flask”
`pip install Flask`
- Antes de correr el código por primera vez se debe ejecutar un comando que esta dentro del proyecto escribiendo este código en la terminal, en la carpeta donde esta ubicado el directorio flaskr, con el entorno activado:
`flask --app flaskr init-db`
- Este comando activa e inicia la base de datos en su estado predeterminado. Se debe ejecutar el comando cada vez que se vayan a realizar las pruebas, ya que estas tienen un orden de ejecución específico que parte con el estado inicial de la base de datos
- Después de realizar esas dos cosas se puede correr el programa ejecutando esta línea en la terminal, en la misma carpeta mencionada anteriormente.
`flask --app flaskr run --debug`

Pruebas pasadas

The screenshot displays the Postman interface with the 'REST API basics: CRUD, test...' collection selected. The 'Run results' tab is active, showing a summary of 225 tests passed in 6s 735ms. The 'RUN SUMMARY' table lists the following tests:

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	6s 735ms	225	8 ms

The 'RUN SUMMARY' table also includes a detailed list of tests and their results:

Test Name	Pass	Fail
GET Get existing fizzbuzz	200	0
GET get not existing fizzBuzz	2	0
POST post new FizzBuzz	11	0
DELETE activate old fizzBuzz	2	0
POST posting existing fizzbuzz	2	0
POST existing Range request	2	0
POST Not existing range	2	0
DELETE Delete existing fizzBuzz	1	1
DELETE Delete not existing fizzBuzz	2	0

The 'DELETE Delete existing fizzBuzz' test failed. The failure message is: 'Fail Test response of http://127.0.0.1:5000/fb/...'. The status is 'Pass: Status test' and 'Fail: Test response of http://127.0.0.1:5000/fb/...'. The failure is indicated by a red 'X' in the 'Fail' column.

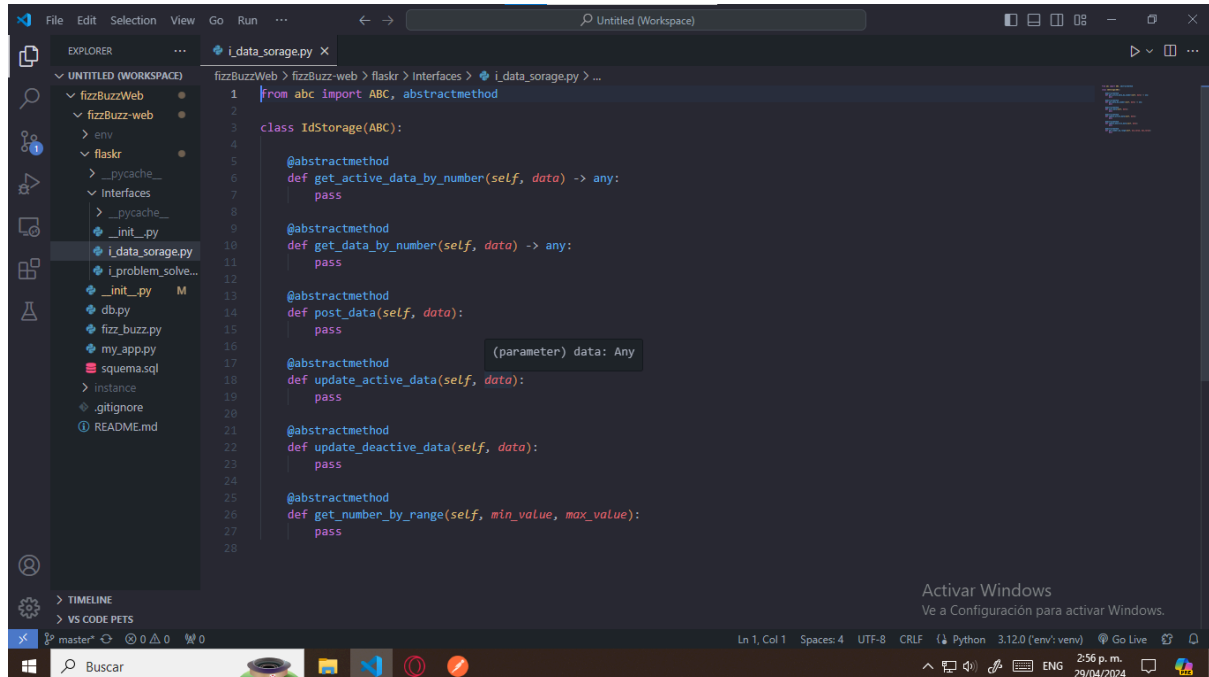
Esta es la ejecución de las pruebas de la colección del programa, se puede observar que todas las pruebas pasaron menos una, en la solicitud “Delete existing fizzBuzz”, esto se debe a que hay un conflicto en los requerimientos. Al menos en flask cuando se retorna un código de estado 204 (No content), debido al significado del código flask espera que el body de la respuesta está vacío, aunque mandemos un body específico este se eliminará ya que el código de estado de la respuesta es un 204 que significa que la respuesta no tiene contenido, o en este caso, body

Código

Estructura de archivos

```
├ flaskr/
│   ├── Interfaces/
│   │   ├── i_data_sorage.py
│   │   ├── i_problem_solver.py
│   │   └── __init__.py
│   ├── i_data_sorage.py
│   ├── i_problem_solver.py
│   ├── __init__.py
│   ├── db.py
│   ├── fizz_buzz.py
│   ├── my_app.py
│   └── __init__.py
│   ├── db.py
│   ├── fizz_buzz.py
│   ├── my_app.py
│   ├── squema.sql
│   └── __init__.py
├ instance/
│   └── fizzBuzz.sqlite
├ .gitignore
└ README.md
```

Flaskr/interfaces/i_data_sorage.py

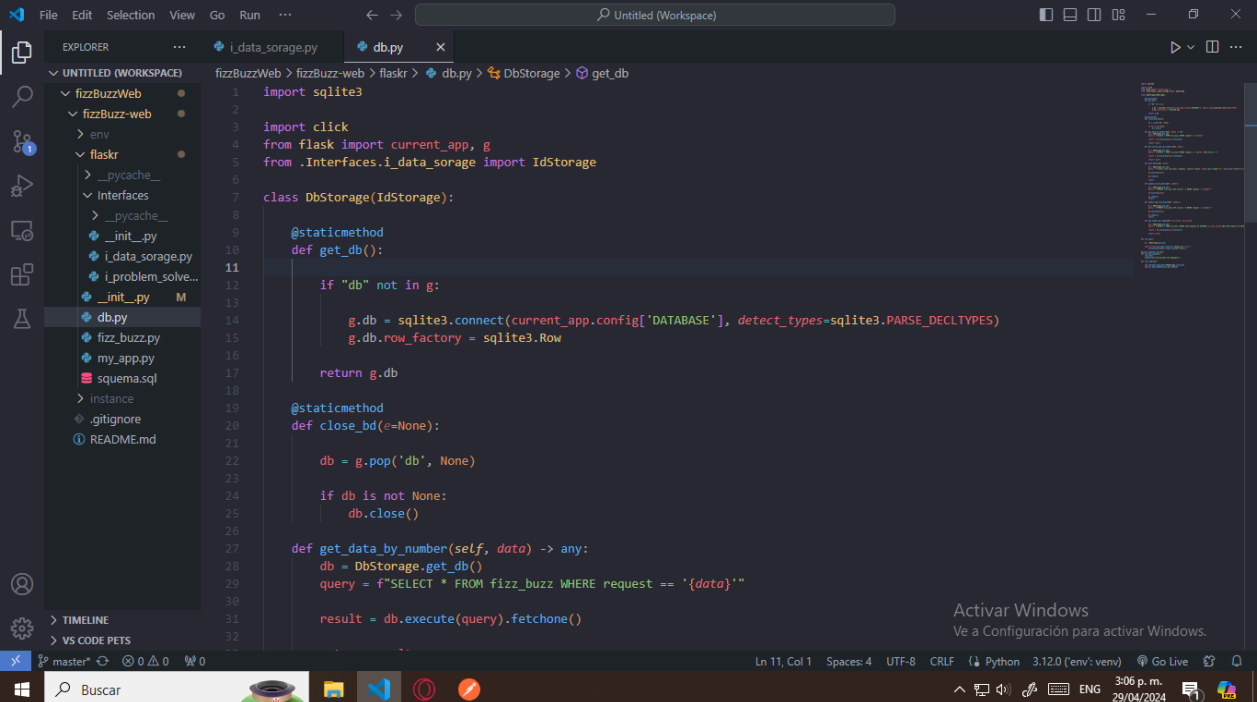


```
1 from abc import ABC, abstractmethod
2
3 class IdStorage(ABC):
4
5     @abstractmethod
6     def get_active_data_by_number(self, data) -> any:
7         pass
8
9     @abstractmethod
10    def get_data_by_number(self, data) -> any:
11        pass
12
13    @abstractmethod
14    def post_data(self, data):
15        pass
16
17    @abstractmethod
18    def update_active_data(self, data):
19        pass
20
21    @abstractmethod
22    def update_deactive_data(self, data):
23        pass
24
25    @abstractmethod
26    def get_number_by_range(self, min_value, max_value):
27        pass
28
```

Esta es la interfaz que implementa el storage de la aplicación, en nuestro caso una clase base de datos. En esta interfaz se definen todos los métodos que la clase de almacenamiento debería implementar para que la aplicación funcione correctamente y el almacenamiento cumpla con las funciones requeridas.

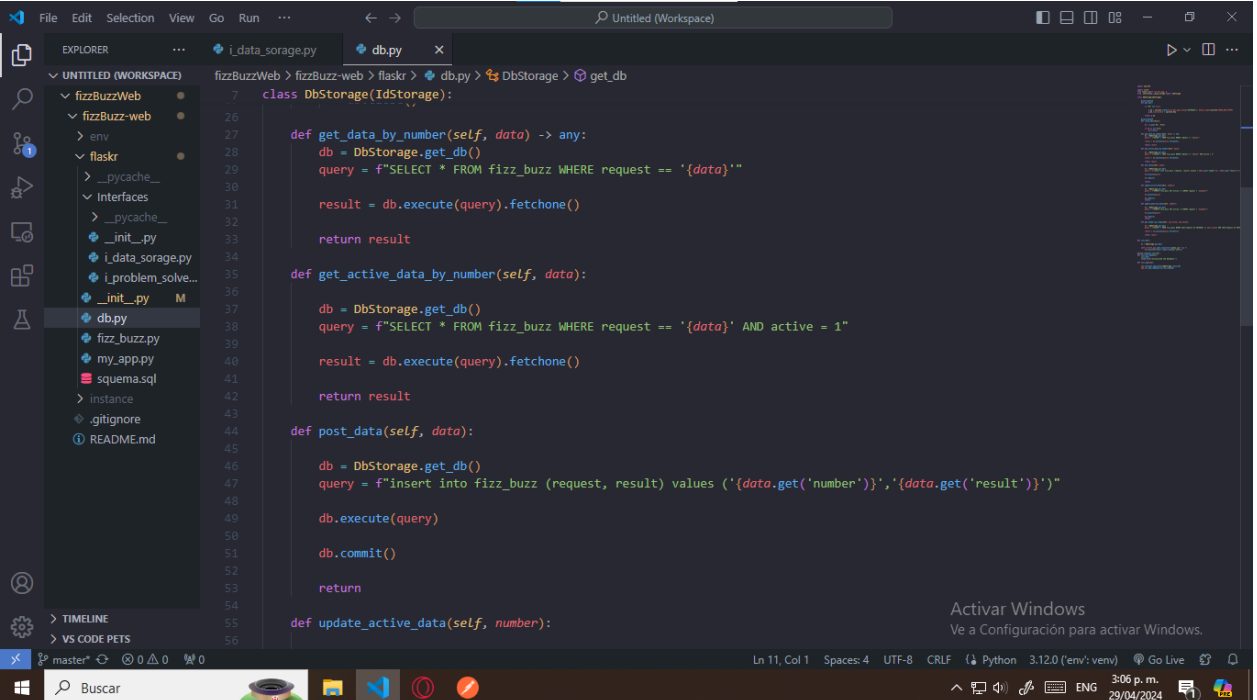
- **get_active_data_by_number(self, data):** Este método implementa una función que retorne los números que estén activados a partir de un filtro, el número. Es un método que obtiene un número activo.
- **get_data_by_number(self, data):** Este metodo tambien obtiene un número buscando por el mismo número, sin embargo este puede estar activo o no, no importa
- **post_data(self, data):** Este método inserta un número dato que se le pasa por el parámetro del método en el sistema de almacenamiento elegido.
- **update_active_data(self, data):** Este método activa un número
- **update_deactive_data(self, data):** Este método desactiva un número
- **get_number_by_range(self, min_value, max_value):** Este método obtiene todos los numeros que esten entre **min_value y max_value**

flaskr/db.py



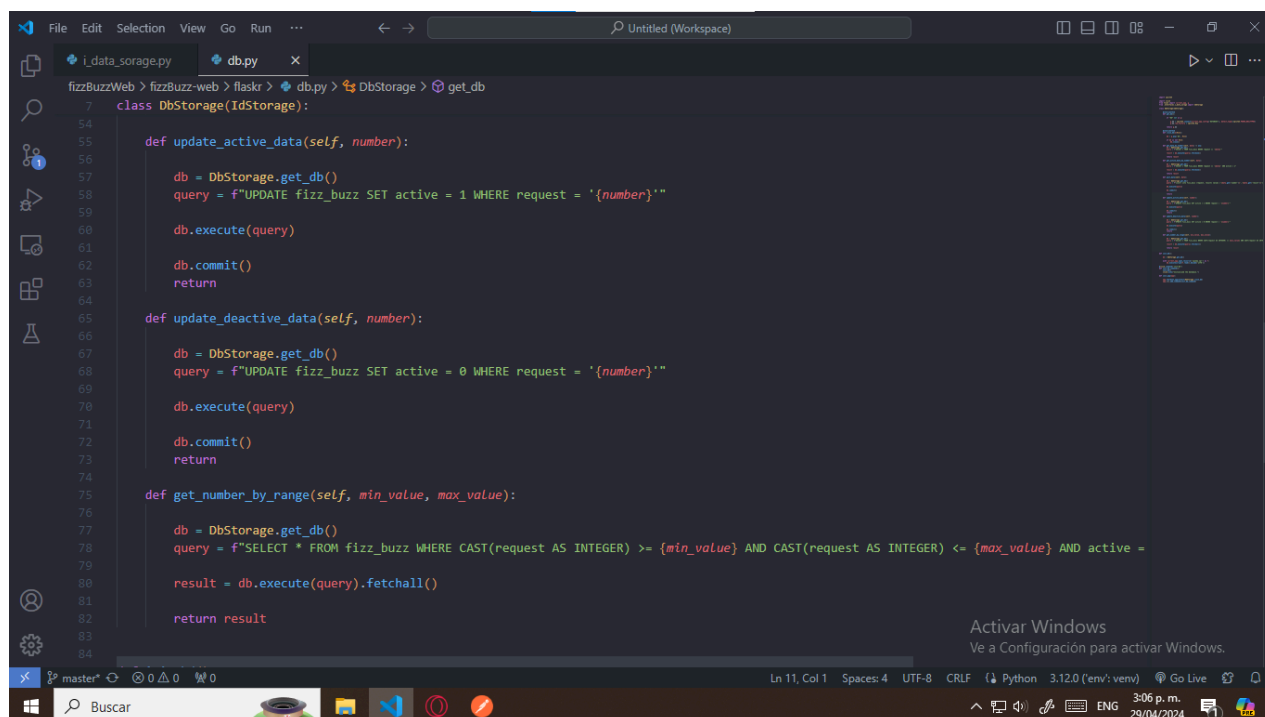
This screenshot shows the Visual Studio Code editor with the file `flaskr/db.py` open. The Explorer sidebar on the left shows the project structure, including `fizzBuzzWeb`, `fizzBuzz-web`, `env`, `flaskr`, `__pycache__`, `Interfaces`, `__pycache__`, `__init__.py`, `i_data_sorage.py`, `i_problem_solve...`, `__init__.py`, `db.py`, `fizz_buzz.py`, `my_app.py`, `schema.sql`, `Instance`, `.gitignore`, and `README.md`. The main editor area displays the code for `db.py`, which includes imports for `sqlite3` and `click`, and a `DbStorage` class that inherits from `IdStorage`. The class has static methods `get_db()`, `close_bd()`, `get_data_by_number()`, `get_active_data_by_number()`, `post_data()`, and `update_active_data()`. The status bar at the bottom indicates the current position is Line 11, Column 1, with 4 spaces, UTF-8 encoding, and CRLF line endings. The Python version is 3.12.0 (env: venv). The Windows taskbar at the bottom shows the search bar and system tray.

```
1 import sqlite3
2
3 import click
4 from flask import current_app, g
5 from .Interfaces.i_data_sorage import IdStorage
6
7 class DbStorage(IdStorage):
8
9     @staticmethod
10     def get_db():
11
12         if "db" not in g:
13
14             g.db = sqlite3.connect(current_app.config['DATABASE'], detect_types=sqlite3.PARSE_DECLTYPES)
15             g.db.row_factory = sqlite3.Row
16
17             return g.db
18
19     @staticmethod
20     def close_bd(e=None):
21
22         db = g.pop('db', None)
23
24         if db is not None:
25             db.close()
26
27     def get_data_by_number(self, data) -> any:
28         db = DbStorage.get_db()
29         query = f"SELECT * FROM fizz_buzz WHERE request == '{data}'"
30
31         result = db.execute(query).fetchone()
```



This screenshot shows the Visual Studio Code editor with the file `flaskr/db.py` open, displaying the continuation of the `DbStorage` class. The Explorer sidebar on the left shows the project structure, including `fizzBuzzWeb`, `fizzBuzz-web`, `env`, `flaskr`, `__pycache__`, `Interfaces`, `__pycache__`, `__init__.py`, `i_data_sorage.py`, `i_problem_solve...`, `__init__.py`, `db.py`, `fizz_buzz.py`, `my_app.py`, `schema.sql`, `Instance`, `.gitignore`, and `README.md`. The main editor area displays the code for `db.py`, which includes imports for `sqlite3` and `click`, and a `DbStorage` class that inherits from `IdStorage`. The class has static methods `get_db()`, `close_bd()`, `get_data_by_number()`, `get_active_data_by_number()`, `post_data()`, and `update_active_data()`. The status bar at the bottom indicates the current position is Line 11, Column 1, with 4 spaces, UTF-8 encoding, and CRLF line endings. The Python version is 3.12.0 (env: venv). The Windows taskbar at the bottom shows the search bar and system tray.

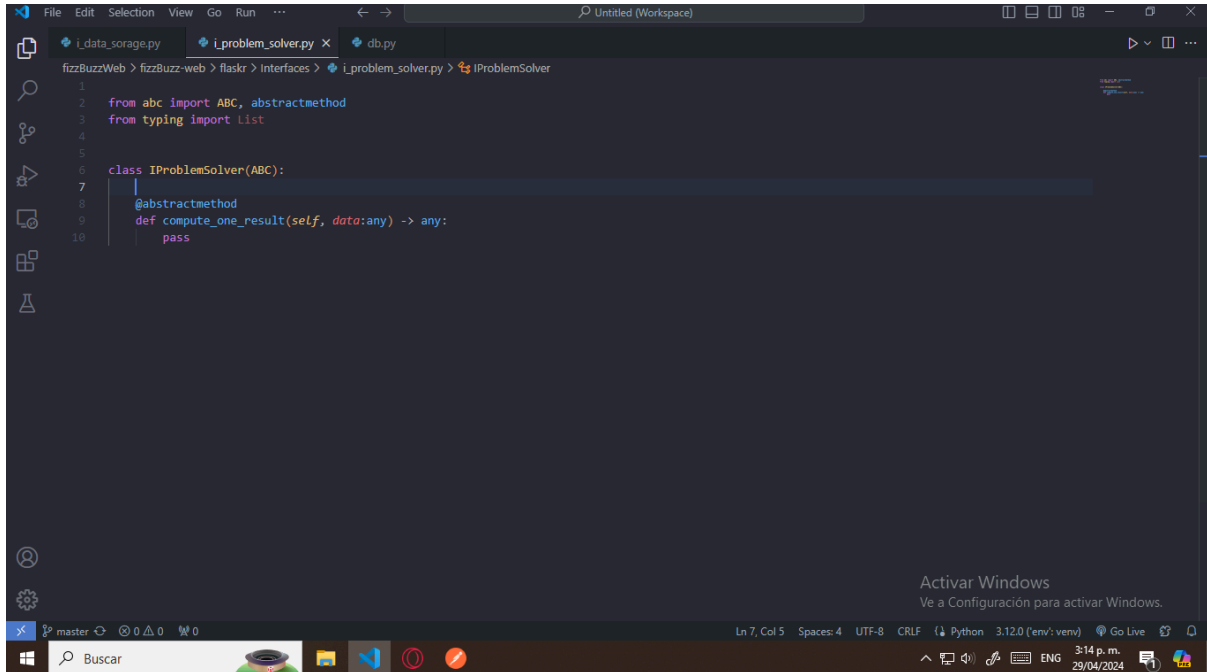
```
32
33
34     def get_data_by_number(self, data) -> any:
35         db = DbStorage.get_db()
36         query = f"SELECT * FROM fizz_buzz WHERE request == '{data}'"
37
38         result = db.execute(query).fetchone()
39
40         return result
41
42     def get_active_data_by_number(self, data):
43
44         db = DbStorage.get_db()
45         query = f"SELECT * FROM fizz_buzz WHERE request == '{data}' AND active = 1"
46
47         result = db.execute(query).fetchone()
48
49         return result
50
51     def post_data(self, data):
52
53         db = DbStorage.get_db()
54         query = f"insert into fizz_buzz (request, result) values ({data.get('number')},{data.get('result')})"
55
56         db.execute(query)
57
58         db.commit()
59
60         return
61
62     def update_active_data(self, number):
```



```
7 class DbStorage(IdStorage):
54
55     def update_active_data(self, number):
56
57         db = DbStorage.get_db()
58         query = f"UPDATE fizz_buzz SET active = 1 WHERE request = '{number}'"
59
60         db.execute(query)
61
62         db.commit()
63         return
64
65     def update_deactive_data(self, number):
66
67         db = DbStorage.get_db()
68         query = f"UPDATE fizz_buzz SET active = 0 WHERE request = '{number}'"
69
70         db.execute(query)
71
72         db.commit()
73         return
74
75     def get_number_by_range(self, min_value, max_value):
76
77         db = DbStorage.get_db()
78         query = f"SELECT * FROM fizz_buzz WHERE CAST(request AS INTEGER) >= {min_value} AND CAST(request AS INTEGER) <= {max_value} AND active = 1"
79
80         result = db.execute(query).fetchall()
81
82         return result
83
84
```

En este archivo está la clase 'DbStorage' que implementa la interfaz "IdStorage". La clase tiene un método para obtener la conexión a la base de datos y otro para cerrar la conexión, también realiza las implementaciones de los métodos de la interfaz.

flaskr/interfaces/i_problem_solver.py

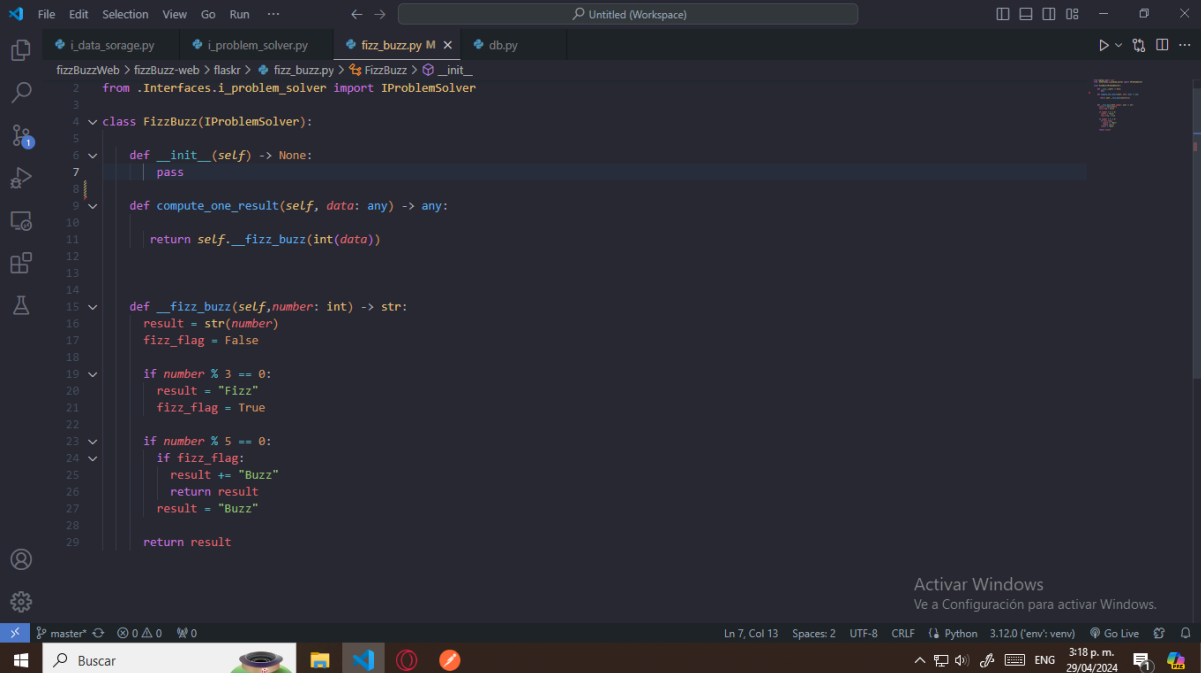


```
1 from abc import ABC, abstractmethod
2 from typing import List
3
4
5
6 class IPProblemSolver(ABC):
7     @abstractmethod
8     def compute_one_result(self, data: any) -> any:
9         pass
10
```

The screenshot shows a code editor with the file `i_problem_solver.py` open. The code defines an abstract class `IPProblemSolver` that inherits from `ABC`. It has a single abstract method `compute_one_result` which takes a `data` parameter of type `any` and returns a value of type `any`. The method body is empty, indicated by `pass`. The editor's interface includes a sidebar with icons for Explorer, Search, Run and Debug, Source Control, and Test Explorer. The bottom status bar shows the current line and column (Ln 7, Col 5), encoding (UTF-8), line endings (CRLF), and the Python environment (Python 3.12.0 (env: venv)).

Esta interfaz proporciona los métodos necesarios para calcular o procesar un problema a partir de un dato. Esta es la interfaz que implementa nuestra clase FizzBuzz, que tendrá la implementación de este método

flaskr/fizz_buzz.py



```
1  from Interfaces.i_problem_solver import IProblemSolver
2
3  class FizzBuzz(IProblemSolver):
4
5      def __init__(self) -> None:
6          pass
7
8      def compute_one_result(self, data: any) -> any:
9
10         return self.__fizz_buzz(int(data))
11
12
13
14
15     def __fizz_buzz(self, number: int) -> str:
16         result = str(number)
17         fizz_flag = False
18
19         if number % 3 == 0:
20             result = "Fizz"
21             fizz_flag = True
22
23         if number % 5 == 0:
24             if fizz_flag:
25                 result += "Buzz"
26             return result
27         result = "Buzz"
28
29     return result
```

Esta clase “FizzBuzz” que implementa la interfaz “IProblemSolver” se encarga de convertir un número a su valor FizzBuzz con el metodo “__fizz_buzz” y con el metodo “compute_one_result” se implementa como la clase resuelve o procesa un “problema” en este caso simplemente devuelve el valor del llamado a la función “__fizz_buzz”

flaskr/my_app.py

```
File Edit Selection View Go Run ...
Unsaved (Workspace)

EXPLORER
my_app.py X

UNTITLED (WORKSPACE)
fizzBuzzWeb > fizzBuzz-web > flask > my_app.py > MyApp > get_range

fizzBuzzWeb
  from .db import DbStorage
fizzBuzz-web
  from .fizz_buzz import FizzBuzz
  from .Interfaces_I_data_sorage import IdStorage
  env
  flask
  _pycache_
  interfaces
  init.py
  db.py
  fizz_buzz.py
  my_app.py
  schema.sql
  instance
  .gitignore
  README.md

TIMELINE
VS CODE PETS

Activar Windows
Ve a Configuración para activar Windows.

Ln 38, Col 28 Spaces: 4 UTF-8 CRLF Python 3.12.0 (env: venv) Go Live
3:23 p.m. 29/04/2024
```

The screenshot displays a Windows 10 desktop environment with a Visual Studio Code (VS Code) editor window open. The editor is showing a Python file named `my_app.py` within a project structure. The file explorer on the left lists the following files and directories: `flask`, `pycache_`, `interfaces`, `init.py`, `db.py`, `flizz_buzz.py`, `my_app.py` (selected), `schema.sql`, `instance`, `.gitignore`, and `README.md`. The main editor area contains the following Python code:

```

class MyApp():
    def post_number(self, num):
        data = {'number': num, 'result': self.flizzbuzz.compute_one_result(num)}

        sql_result_exists = self.db.get_data_by_number(num)

        if sql_result_exists is not None:

            if sql_result_exists['active'] == 0:
                self.db.update_active_data(num)
                return sql_result_exists['result'], 200

            else:
                return sql_result_exists['result'], 400

        else:
            self.db.post_data(data)
            return str(data.get('result')), 201

    def delete_number(self, num):
        sql_result_exists = self.db.get_active_data_by_number(num)

        if sql_result_exists is not None:
            self.db.update_deactive_data(num)
            return "", 204
        else:
            return "Not Found", 404

    def get_max_value(self, min_value, max_value):
        sql_result = self.db.get_max_value(min_value, max_value)

```

The status bar at the bottom of the VS Code window indicates the current file is `Ln 5, Col 12 | 5 selected | Spaces 4 | UTF-8 | CRLF | Python 3.12.0 (env: venv) | Go Live`. The Windows taskbar at the bottom shows the Start button, a search bar, and several pinned applications including File Explorer, VS Code, and a terminal window.

The screenshot displays a Windows 10 desktop environment. The primary application is Visual Studio Code (VS Code), which is open to a Python file named `my_app.py`. The Explorer sidebar on the left shows the project's file structure, including a `__pycache__` directory, `interfaces`, `__init__.py`, `db.py`, `hzz_buzz.py`, `my_app.py` (the active file), `request.sql`, `instance`, `.gitignore`, and `README.md`. The main editor window shows the following Python code:

```

class MyApp():
    def get_range(self,min_value,max_value):
        sql_result = self.db.get_number_by_range(min_value, max_value)

        if not sql_result:
            return "Not Found", 404

        print(sql_result)
        result = ""

        for row in sql_result:
            line = f'{row["request"]}: {row["result"]}\n"

            result = result + line

        return result, 200

```

The status bar at the bottom of the VS Code window indicates the current cursor position as 'Ln 5, Col 12 (5 selected)', the file encoding as 'UTF-8', the line endings as 'CRLF', and the active interpreter as 'Python 3.12.0 (env: venv)'. There is also a 'Go Live' button in the status bar.

Esta clase se encarga de hacer el control de la aplicación y la lógica de cada solicitud enviada desde el main, que en el caso de nuestro programa es la aplicación de flask: “flaskr/___init___py”. Esta clase tiene una instancia de la clase de base de datos y la clase fizzBuzz. Los métodos de la clase son las principales acciones o solicitudes que tiene que manejar la aplicación, dentro de ellos se encuentra la lógica y el proceso para llevar a cabo cada solicitud, se hace una consulta a la base de datos y dependiendo de la respuesta se realiza una acción en la base de datos, y/o se retorna un html y un statuscode a la aplicación flask.

- **get_number(self,number):** Maneja la lógica de las request GET. Busca un número activo en la base de datos mediante la instancia interna de la base de datos con su método “get_active_data_by_number” si retorna un valor se devuelve el número y el código de estado 200 si no se obtiene nada se retorna la cadena Not Found y el código 404
- **post_number(self, num):** Maneja la lógica de las request POST. Busca un número activo en la base de datos mediante la instancia interna de la base de datos con su método “get_data_by_number” si se no se obtiene ningún valor se inserta el número y su valor fizzBuzz a la base de datos mediante el método “post_data”, además se retorna el número y el código 201. Si se obtiene un valor se verifica si está activo o no, si lo está se retorna el número y el código 409, si no está activo se cambia su estado a activo a través del método “update_active_data” de la base de datos.
- **delete_number(self, num):** Maneja la lógica de las request DELETE. Busca un número activo en la base de datos mediante la instancia interna de la base de datos con su método “get_active_data_by_number” de la clase de la base de datos. Si se obtiene algo se desactiva el número mediante el método “update_deactive_data” de la base de datos junto con el código de estado 204. Si no se obtiene ningún valor se retorna la cadena “Not Found” y el código de estado 404.
- **get_range(self,min_value,max_value):** Maneja la lógica del requisito range, el cual es una solicitud POST especial, con un body que tiene un número mínimo y un valor máximo. Se llama al método de la base de datos “get_number_by_range” el cual hace la consulta del rango. Si se obtiene un valor se recorre la lista y se construye la respuesta con los valores del número y de fizzBuzz de cada registro obtenido y se retorna la cadena construida y el código de estado 200, si no se obtiene nada se retorna la cadena “Not Found” y el código de estado 404

flaskr/ __init__.py

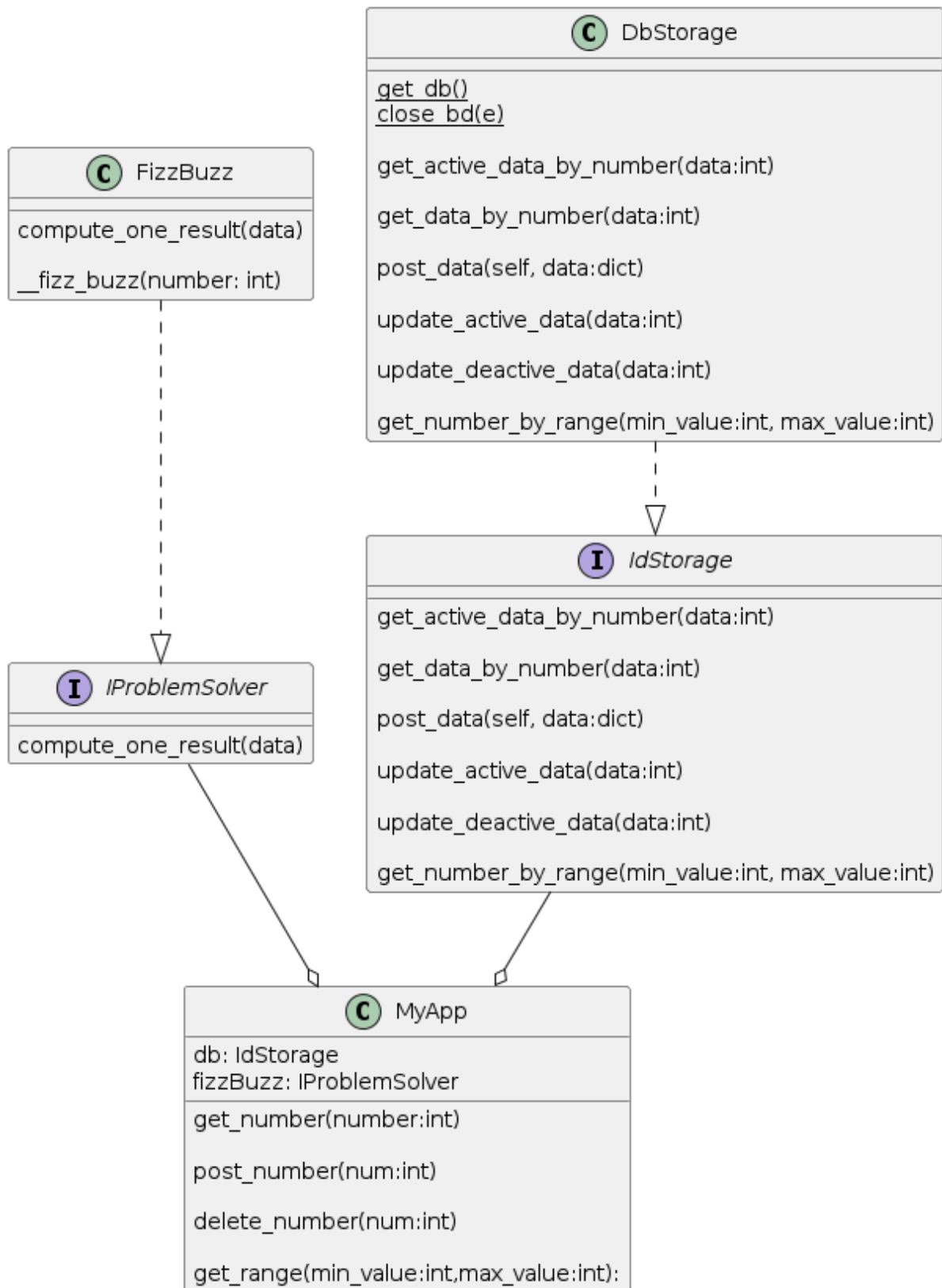
```
1 from http import HTTPStatus
2 import os
3 from flask import Flask, abort, jsonify, make_response, request
4
5 from .my_app import MyApp
6
7
8 def create_app():
9
10     app = Flask(__name__, instance_relative_config=True)
11
12     system = MyApp()
13
14     app.config.from_mapping(
15         SECRET_KEY='dev',
16         DATABASE = os.path.join(app.instance_path, 'fizzBuzz.sqlite')
17     )
18
19     try:
20         os.makedirs(app.instance_path)
21     except OSError:
22         pass
23
24     @app.route('/')
25     def hello():
26         return 'Hello, FizzBuzz!'
27
28     @app.route('/fb<num>', methods=('GET', 'POST', 'DELETE'))
29     def get_fizzBuzz(num):
30
31         if request.method == "GET":
```

```
27
28     @app.route('/fb<num>', methods=('GET', 'POST', 'DELETE'))
29     def get_fizzBuzz(num):
30
31         if request.method == "GET":
32
33             sql_result = system.get_number(num)
34             return sql_result
35
36         if request.method == "POST":
37
38             sql_result = system.post_number(num)
39             return sql_result
40
41         if request.method == "DELETE":
42             sql_result = system.delete_number(num)
43             return sql_result
44
45     @app.route('/range', methods=('POST',))
46     def range_fizzBuzz():
47
48         request_body = request.json
49
50         min_value = request_body['min']
51         max_value = request_body['max']
52
53         if request.method == "POST":
54             print("entre")
55             sql_result = system.get_range(min_value, max_value)
56             return sql_result
```

Aquí se define la “application factory” del programa la cual es la que flask busca para correr la aplicación, también se realizan algunas configuraciones y se definen las diferentes vistas. En este caso se implementaron dos vistas importantes, “get_fizzBuzz” con la ruta “/fb<num>”, cuando en el navegador se ingrese esta ruta se renderizar esta vista, <num> es una variable de url que se pasa en la ruta url, por ejemplo “fb/12”, el 12 se envía como argumento de la vista. La otra vista es range_fizzBuzz que tiene como ruta “/range” y recibe solicitudes post. Por último se debe aclarar que se realiza una instancia a la clase MyApp para pasar los números en cada tipo de request, y así obtener el body y el código de estado correspondiente.

Diagramas

Diagrama de clases



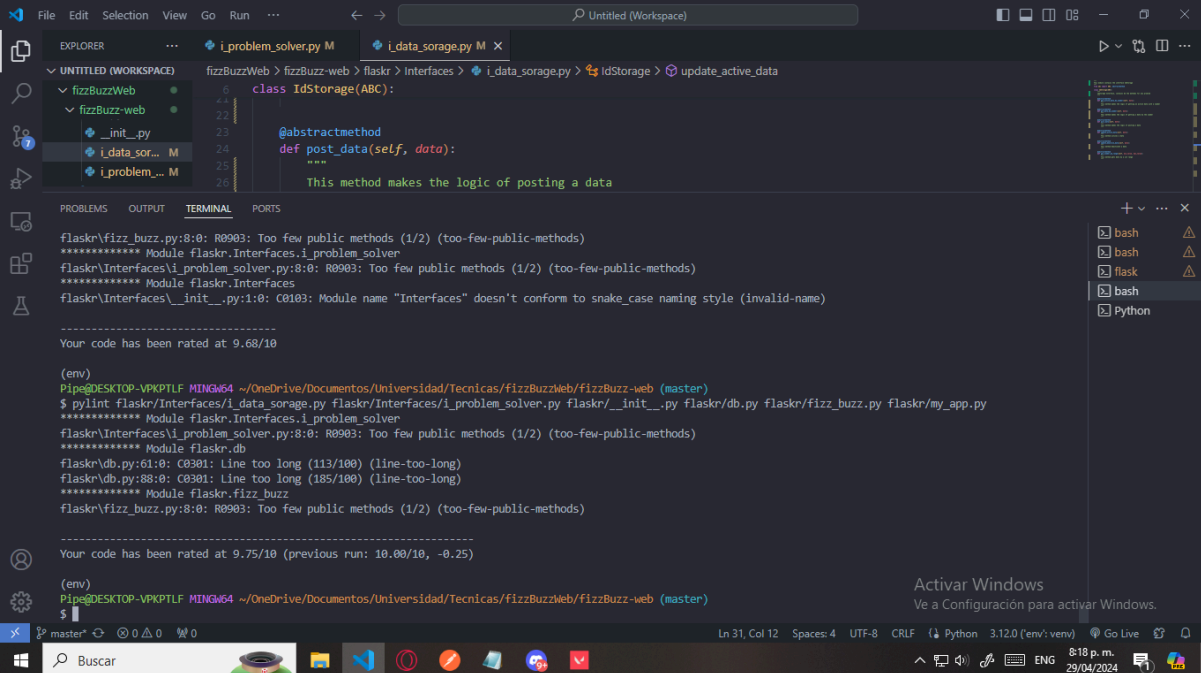
Análisis estático de código

mypy

```
File Edit Selection View Go Run ...  
Explorer  
  UNTITLED (WORKSPACE)  
  fizzBuzzWeb  
    > env  
    > flask  
    > _pycache_  
    > interfaces  
  my_app.py  
  i_problem_solver.py M  
  i_data_sorage.py M  
  fizz_buzz.py M  
  db.py M X  
  classes.txt  
  fizzBuzzWeb > fizzBuzz-web > flask > db.py > DbStorage > get_data_by_number  
    7 class DbStorage(IdStorage):  
    27     def get_data_by_number(self, data):  
    28         query = f"SELECT * FROM TIZZ_BUZZ WHERE request == {data}"  
    30  
    31         result = db.execute(query).fetchone()  
    32  
    33         return result  
    34  
Terminal  
(env)  
Pipe@DESKTOP-VPKPTLF MINGW64 ~/OneDrive/Documentos/Universidad/Tecnicas/fizzBuzzWeb/fizzBuzz-web (master)  
$ mypy flask/_init_.py  
Success: no issues found in 1 source file  
(env)  
Pipe@DESKTOP-VPKPTLF MINGW64 ~/OneDrive/Documentos/Universidad/Tecnicas/fizzBuzzWeb/fizzBuzz-web (master)  
$ mypy flask/db.py  
Success: no issues found in 1 source file  
(env)  
Pipe@DESKTOP-VPKPTLF MINGW64 ~/OneDrive/Documentos/Universidad/Tecnicas/fizzBuzzWeb/fizzBuzz-web (master)  
$ mypy flask/fizz_buzz.py  
Success: no issues found in 1 source file  
(env)  
Pipe@DESKTOP-VPKPTLF MINGW64 ~/OneDrive/Documentos/Universidad/Tecnicas/fizzBuzzWeb/fizzBuzz-web (master)  
$ mypy flask/my_app.py  
Success: no issues found in 1 source file  
(env)  
Pipe@DESKTOP-VPKPTLF MINGW64 ~/OneDrive/Documentos/Universidad/Tecnicas/fizzBuzzWeb/fizzBuzz-web (master)  
$  
Activar Windows  
Ve a Configuración para activar Windows.  
Ln 27, Col 39 Spaces: 4 UTF-8 CRLF Python 3.12.0 (env: venv) Go Live ENG 6:40 p.m. 29/04/2024
```

Se utilizó mypy como herramienta de análisis de código estático. Se revisaron todos los archivos .py con mypy y todos pasaron.

Pylint



The screenshot shows a Visual Studio Code editor with a Python file open. The file contains a class `IdStorage(ABC)` with an abstract method `post_data(self, data)`. The terminal window displays the output of a Pylint command, showing several errors and a code quality score of 9.75/10.

```
flaskr\ fizz_buzz.py:8:0: R0903: Too few public methods (1/2) (too-few-public-methods)
***** Module flaskr.Interfaces.i_problem_solver
flaskr\Interfaces\i_problem_solver.py:8:0: R0903: Too few public methods (1/2) (too-few-public-methods)
***** Module flaskr.Interfaces
flaskr\Interfaces\__init__.py:1:0: C0103: Module name "Interfaces" doesn't conform to snake_case naming style (invalid-name)

-----
Your code has been rated at 9.68/10

(env)
Pipe@DESKTOP-VPKPTLF MINGW64 ~/OneDrive/Documentos/Universidad/Tecnicas/fizzBuzzWeb/fizzBuzz-web (master)
$ pylint flaskr/Interfaces/i_data_storage.py flaskr/Interfaces/i_problem_solver.py flaskr/__init__.py flaskr/db.py flaskr/fizz_buzz.py flaskr/my_app.py
***** Module flaskr.Interfaces.i_problem_solver
flaskr\Interfaces\i_problem_solver.py:8:0: R0903: Too few public methods (1/2) (too-few-public-methods)
***** Module flaskr.db
flaskr\db.py:61:0: C0301: Line too long (113/100) (line-too-long)
flaskr\db.py:88:0: C0301: Line too long (185/100) (line-too-long)
***** Module flaskr.fizz_buzz
flaskr\ fizz_buzz.py:8:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 9.75/10 (previous run: 10.00/10, -0.25)

(env)
Pipe@DESKTOP-VPKPTLF MINGW64 ~/OneDrive/Documentos/Universidad/Tecnicas/fizzBuzzWeb/fizzBuzz-web (master)
$
```

También se utilizó pylint, corrigiendo la mayor parte de “errores” se obtuvo una puntuación de 9.75. Pylint ayuda mucho a seguir buenas prácticas de programación como, indentación, comentarios, documentación mínima. Básicamente ayuda a tener un código mucho mas limpio y agradable.