

ADVANCED PROGAMMING TECHNIQUES

FIZZ BUZZ WEB



**UNIVERSIDAD DE
SAN BUENAVENTURA
CALI**

Juan Pablo Silvestre González

jpsilvestreg@correo.usbcali.edu.co

30000099127

22/4/2024

CHALLENGE DESCRIPTION

Given the Fizz Buzz problem from past activities, the main goal is to create an API that provides the basic CRUD interface. Using Flask as lite server and Postman as a test tool, we will satisfy all requirements specified.

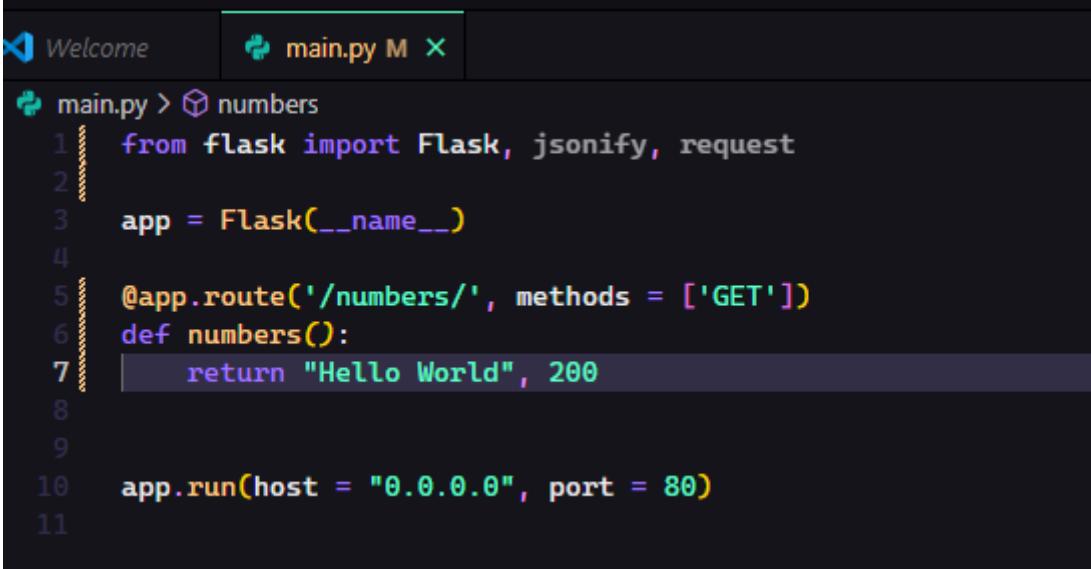
RESTRICTIONS

This report lies its structure on the Test-Driven-Development (TDD) methodology, proving and documenting all the process in each phase (Red, Green and Blue) and what it took to achieve the final product. Also, we will be using the Dependency Inversion Principle (DIP) that guides how we should structure our code to make it more flexible and maintainable.

STRATEGY

In response to each requirement pointed in the challenge, we start off with our main Flask application, then I'm going to define a Postman script for each section in this challenge; these scripts will confirm whether the retrieved data matches the expected response or if it correctly handles any errors. Then we will develop our code's structure, which is going to rely on the DIP; after our structure is defined, we can create our database and the logic that will bridge our database with the requests made from Postman.

RED PHASE



```
Welcome main.py M X
main.py > numbers
1  from flask import Flask, jsonify, request
2
3  app = Flask(__name__)
4
5  @app.route('/numbers/', methods = ['GET'])
6  def numbers():
7      return "Hello World", 200
8
9
10 app.run(host = "0.0.0.0", port = 80)
11
```

We first create our main application where we'll get the response from, now, all it returns is a "Hello World", with the status 200.

The screenshot shows the Postman application interface. At the top, there is a header bar with three dots on the left, followed by an 'HTTP' icon and the URL 'http://127.0.0.1/numbers/'. Below this is a search bar with 'GET' selected and the same URL. A dropdown arrow is next to the search bar.

Below the search bar is a navigation menu with tabs: 'Params' (which is underlined in red), 'Authorization', 'Headers (8)', 'Body •', 'Pre-request Script', 'Tests', and 'Settings'. Under the 'Params' tab, there is a section titled 'Query Params' with a table:

Key	Value
Key	Value

At the bottom of the interface, there is another navigation menu with tabs: 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Body' tab is underlined in red. Below this menu, there are four buttons: 'Pretty' (selected), 'Raw', 'Preview', and 'Visualize'. To the right of these buttons is a dropdown menu set to 'HTML'. On the far right, there is a copy icon.

The main content area shows the response body, which consists of a single line of text: '1 Hello World'.

Now, with a basic setup we can start developing the scripts to validate each section provided in the challenge.

- We start off with 100 numbers stored in the database, those will come from previous results obtained in past projects related to FizzBuzz. We can request the API to retrieve one of those numbers and see what we get:

The screenshot shows the Postman application interface. At the top, there's a search bar and a navigation bar with 'Home', 'Workspaces', 'API Network', and other options like 'Invite' and 'Upgrade'. Below the header, a sidebar on the left lists 'Collections', 'Environments', and 'History'. The main workspace shows a 'GET' request to 'http://127.0.0.1/numbers/551669674'. The 'Tests' tab is selected, displaying the following JavaScript code:

```

1 pm.test("get specific number", function(){
2     pm.response.to.have.body("551669674: 551669674")
3 });
4
5 pm.test("reponse time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 });
8
9 pm.test("response number", function(){
10    | pm.response.to.have.status(200);
11 })

```

Below the tests, the 'Body' tab is selected, showing the raw HTML response:

```

<!DOCTYPE html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

```

The status bar at the bottom indicates 'Status: 404 NOT FOUND Time: 7 ms Size: 388 B'. The bottom section shows the test results with three entries:

- FAIL** get specific number | Assertion Error: expected response body to equal '551669674: 551669674' but got '<!DOCTYPE html>\n<html lang=en>\n<tit...'
- PASS** reponse time below 200ms
- FAIL** response number | Assertion Error: expected response to have status code 200 but got 404

The test result points out the fact that we didn't get the expected response, for the main reason that we haven't created the database and the logic for that request.

- The next section is to retrieve the record of a number that is not stored in the database, for instance we'll try retrieving the number 2.

The screenshot shows the Postman interface with a GET request to `http://127.0.0.1/numbers/2`. The 'Tests' tab is selected, displaying the following PM Test script:

```
1 pm.test("get specific number", function(){
2     pm.response.to.have.body("Not Found")
3 }
4
5 pm.test("reponse time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 }
8
9 pm.test("response code", function(){
10    pm.response.to.have.status(404);
11 })
```

The 'Test Results' section shows 2/3 tests passed:

- FAIL get specific number | Assertion Error: expected response body to equal 'Not Found' but got '<!doctype html><html lang=en><tit...'
- PASS reponse time below 200ms
- PASS response code

Status: 404 NOT FOUND Time: 7 ms Size: 388 B

As shown in the screenshot, 2 out of 3 tests passed, but since there's no actual database we are retrieving our requested number from, it will return the 404 anyway.

- The next section tackles inserting the record of a new number into the database; in case the number is already stored, it will return the HTTP code 409. Otherwise, if the number is present but not active, it will set its status back to active and make it available for future requests, besides returning the HTTP code 200. And finally, if the number is not present, it will return the HTTP code 201.

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1/numbers/551669674`. The 'Tests' tab is selected, displaying the following PM Test script:

```
1 pm.test("Post existing number", function(){
2     pm.response.to.have.body("551669674: 551669674")
3 }
4
5 pm.test("reponse time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 }
8
9 pm.test("response code", function(){
10    pm.response.to.have.status(409);
11 })
```

The 'Test Results' section shows 1/3 tests failed:

- FAIL Post existing number | Assertion Error: expected response body to equal '551669674: 551669674' but got '<!doctype html><html lang=en><tit...'
- PASS reponse time below 200ms
- FAIL response code | Assertion Error: expected response to have status code 409 but got 404

Status: 404 NOT FOUND Time: 12 ms Size: 388 B

First, we try to insert a number already stored (551669674), which response is 404, again, we're getting the same response since there's no database we are inserting or retrieving our data from.

We will get 404 from now on, until we create our first instance of the database.

The screenshot shows a Postman collection named "FizzBuzzWeb" with a single test script for a POST request to "http://127.0.0.1/number/1". The test script includes assertions for response body, time, and status code. The "Test Results" section shows one failure (FAIL) and two passes (PASS). The failed test is for the response body assertion, indicating an unexpected response body.

```

1 pm.test("post new number", function(){
2     pm.response.to.have.body("1: 1")
3 }
4
5 pm.test("response time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 }
8
9 pm.test("response code", function(){
10    | pm.response.to.have.status(201);
11 }
12

```

Test Results (1/3)

- FAIL post new number | Assertion Error: expected response body to equal '1: 1' but got '<!doctype html>\n<html lang=en>\n<tit...'
- PASS response time below 200ms
- FAIL response code | Assertion Error: expected response to have status code 201 but got 404

Same result goes for a number that is not present in the database, in this case, we try the number 1.

- The next section consists on retrieving a range of numbers from the database, which is set by sending both inferior and superior limit as parameters; in this test, we set the range to be between 1773224 and 45597635.

The screenshot shows a Postman collection named "FizzBuzzWeb" with a single test script for a POST request to "http://127.0.0.1/range?low_limit=1&sup_limit=1". The test script includes assertions for response body, time, and status code. The "Test Results" section shows one failure (FAIL) and two passes (PASS). The failed test is for the response body assertion, indicating an unexpected response body.

```

1 numbers = [
2     "1773224", "1773224",
3     "18013025", "18813028",
4     "29900124", "Fizz",
5     "30836800", "Buzz",
6     "48597635", "Buzz"
7 ]
8
9 pm.test("get range", function(){
10     pm.response.to.have.body(numbers);
11
12     if (Object.keys(numbers).length === 0) {
13         pm.response.to.have.status(404);
14     }
15 }
16
17 pm.test("response time below 200ms", function(){

```

Test Results (1/3)

- FAIL get range | Assertion Error: expected response body json to equal { '1773224': '1773224', ...('4') } but got undefined
- PASS response time below 200ms
- FAIL response code | Assertion Error: expected response to have status code 200 but got 404

As expected, no response whatsoever, the status code we get 404 since there's not an initial state of the database.

The screenshot shows the Postman interface with a workspace named 'FizzBuzzWeb'. A POST request titled 'Range Not found' is selected. The URL is `http://127.0.0.1/range/?low_limit=&high_limit=1`. The 'Tests' tab contains the following JavaScript code:

```

1 pm.test("get not found", function(){
2     pm.response.to.have.body("Not Found.");
3 });
4
5 pm.test("response time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 });
8
9 pm.test("response code", function(){
10    pm.response.to.have.status(404);
11 });
12
13 pm.response.to.have.status(404);
14 });

```

The 'Test Results' section shows one failure: 'FAIL | get not found | Assertion: expected response body to equal 'Not Found'' and two passes: 'PASS | response time below 200ms' and 'PASS | response code'.

In this test, we pass the response code for HTTP code 404, which makes sense for the reason there's no database to take the range's information from.

- The next section we are assigned to delete an existing number stored in the database, the expected response is the HTTP code 204 if the number is not stored.

The screenshot shows the Postman interface with a workspace named 'FizzBuzzWeb'. A DELETE request titled 'Delete present number' is selected. The URL is `http://127.0.0.1/numbers/551669674`. The 'Tests' tab contains the following JavaScript code:

```

1 pm.test("get specific number", function(){
2     pm.response.to.have.body("");
3 });
4
5 pm.test("response time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 });
8
9 pm.test("response number", function(){
10    pm.response.to.have.status(204);
11 });
12

```

The 'Test Results' section shows one failure: 'FAIL | get specific number | Assertion: expected response body to equal '' but got '<!doctype html><html lang=en><tit...' and two failures: 'FAIL | response time below 200ms' and 'FAIL | response number | Assertion: expected response to have status code 204 but got 404'.

Same problem as before, the HTTP code returned is 404 due to the absence of a database from which the record can be deleted.

Just like the previous section, the last section involves deleting a non-existing number from the database.

The screenshot shows the Postman application interface. A collection named "FizzBuzzWeb" is selected. Inside it, there is a request titled "Delete number not present" with a DELETE method. The URL is set to `http://127.0.0.1/number/10`. The "Tests" tab contains the following JavaScript code:

```
pm.test("get specific number", function() {
    pm.response.to.have.body("Not Found")
})
pm.test("response time below 200ms", function() {
    pm.expect(pm.response.responseTime).to.be.below(200);
})
pm.test("response code", function() {
    pm.response.to.have.status(404);
})
```

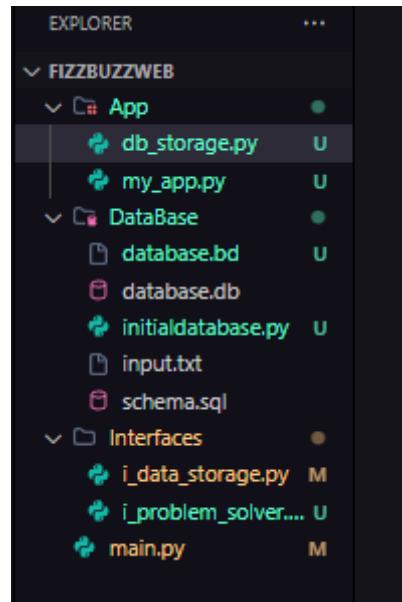
The "Test Results" section shows the following outcomes:

- Failed: 1 - get specific number | Assertion: expected response body to equal "Not Found" but got "<!DOCTYPE html><html lang=en><title>"
- PASSED: 2 - response time below 200ms
- PASSED: 1 - response code

At the bottom of the interface, the status bar indicates "Status: 404 NOT FOUND Time: 6 ms Size: 388 B".

In this section we pass the HTTP code test, but as before, this is a placeholder since the database is not defined.

GREEN PHASE



We start off with 3 directories:

- DataBase: Here, we define the first instance of our data base with the record of the first 100 numbers.

The screenshot shows a Visual Studio Code window with the following details:

- File Explorer:** Shows the project structure under "FIZZBUZZWEB". The "Database" folder contains "database.bd" and "database.db". Inside "database.db", there are "db_storage.py", "initialdatabase.py", and "main.py".
- Code Editor:** The "initialdatabase.py" file is open. It contains Python code for managing a SQLite database. The code includes functions for executing schema files, inserting data from an input file, and printing data.
- Status Bar:** Shows "main" as the active workspace, "Select Postgres Server" as the active connection, and "outline" and "Timeline" as other open panes.
- Bottom Bar:** Shows the current file ("initialdatabase.py"), its status ("U"), the line number (Line 11), column (Col 36), and character (Space: 4). It also shows encoding (UTF-8), line endings (CRLF), and Python version (3.11.4). Other icons include Go Live, Go To Definition, and a search icon.

The goal here is, create our database with the table “numbers” that will come from the “schema.sql” file, then we insert the data from the “input.txt” file, that contains the first 100 numbers and their respective result.

```

CREATE TABLE IF NOT EXISTS numbers (
    num INTEGER PRIMARY KEY,
    num_evaluated TEXT,
    active BOOLEAN
);

```

Here's the schema file that creates the table numbers.

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure for "FIZZBUZZWEB" with files like App, db_storage.py, my_app.py, DataBase, database.bd, database.db, initdatabase.py, input.txt, schema.sql, and Interfaces.
- INPUT.TXT:** A text file containing 100 lines of FizzBuzz results, starting with "551669674 FizzBuzz" and ending with "552373671 FizzBuzz".
- SCHEMA.SQL:** A SQL script to create a "numbers" table with columns num, num_evaluated, and active.
- OUTLINE:** Shows the outline of the current file.
- TIMELINE:** Shows the history of changes made to the file.
- STATUS BAR:** Displays file path ("input.txt - FizzBuzzWeb - Visual Studio Code"), line count (Ln 11, Col 20), spaces (Spaces: 4), encoding (UTF-8), and date/time (9:40 p.m.).

Here's the input file where the 100 numbers and their results come from.

- **Interfaces:** Here we define the interfaces for the data storage type (`i_data_storage`) and the problem solver type (`i_problem_solver`). For the Storage type, we define the methods that will allow us get responses from either a data base or a file in general (if necessary, we will add more methods in the future).

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure for "FIZZBUZZWEB" with files like App, db_storage.py, my_app.py, DataBase, database.bd, database.db, initdatabase.py, input.txt, schema.sql, and Interfaces.
- I_DATA_STORAGE.PY:** A Python file defining an abstract base class `IDataStorage` using ABC (Abstract Base Class). It includes methods for `get_data`, `post_data`, `get_range`, and `delete_data`.
- OUTLINE:** Shows the outline of the current file.
- TIMELINE:** Shows the history of changes made to the file.
- STATUS BAR:** Displays file path ("i_data_storage.py - FizzBuzzWeb - Visual Studio Code"), line count (Ln 19, Col 13), spaces (Spaces: 4), encoding (UTF-8), and date/time (9:47 p.m.).

```
i_problem_solver.py - FizzBuzzWeb - Visual Studio Code
i_problem_solver.py
1 |
```

The screenshot shows the Visual Studio Code interface with the file `i_problem_solver.py` open in the editor. The code is currently empty, with only the first character '1' visible. The project structure on the left includes `App`, `database`, and `Interfaces` folders. The `Interfaces` folder contains `i_data_storage.py` and `i_problem_solver.py`. The status bar at the bottom shows Python 3.11.4 (env/venv) and Go Live.

The `i_problem_solver` interface will remain empty until we have the need of calculating the result of any given number.

- App: Here we define data base storage type (DBStorage) that implements the interface `i_data_storage`; where we will make the logic for retrieving and inserting data into the database.

```
db_storage.py - FizzBuzzWeb - Visual Studio Code
db_storage.py
App > db_storage.py > DBStorage
1 from Interfaces.i_data_storage import IDataStorage
2 import sqlite3
3
4 class DBStorage(IDataStorage):
5     def __init__(self):
6         pass
7
8     @staticmethod
9     def db_connection():
10        connection = sqlite3.connect(r"\DataBase\database.db")
11        connection.row_factory = sqlite3.Row
12        return connection
13
14    def get_data(self, data):
15        connection = self.db_connection()
16        with connection:
17            cursor = connection.cursor()
18            cursor.execute("SELECT num, num_evaluated FROM numbers WHERE num = ? AND active = '1'", (data,))
19            result = cursor.fetchone()
20            connection.close()
21        return result
22
23    def post_data(self, data):
24        pass
25
26
27    def delete_data(self, data):
28        pass
29
30    def get_range(self, low_limit, sup_limit):
31        pass
```

The screenshot shows the Visual Studio Code interface with the file `db_storage.py` open in the editor. The code defines a `DBStorage` class that implements the `IDataStorage` interface. It includes methods for connecting to a SQLite database, retrieving data, posting data, deleting data, and getting a range of data. The status bar at the bottom shows Python 3.11.4 (env/venv) and Go Live.

For our first approach, we work on the “`get_data`” method, because our first Postman test relies on this. We will work on the other methods later on.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "FIZZBUZZWEB".
- Editor:** The "my_app.py" file is open, displaying Python code for a class named "MyApp".
- Status Bar:** Shows "Ln 14, Col 1" and "Python 3.11.4 (env\venv)".

```
my_app.py - FizzBuzzWeb - Visual Studio Code
my_app.py
from .db_storage import DBStorage
class MyApp:
    def __init__(self):
        self.storage = DBStorage()
    def get_number(self, number):
        db_result = self.storage.get_data(number)
        if db_result is None:
            return "Not Found", 404
        else:
            return f'{db_result[0]}', 200
main.py
main.py
main.py M
main.py
```

Then we have “my_app”, which is going to bridge the database to the main program, this is where we will get our responses from depending on the current state of the data base.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "FIZZBUZZWEB".
- Editor:** The "main.py" file is open, displaying Python code for a Flask application.
- Status Bar:** Shows "Ln 14, Col 1" and "Python 3.11.4 (env\venv)".

```
main.py - FizzBuzzWeb - Visual Studio Code
main.py
app = Flask(__name__)
my_app = MyApp()
@app.route('/numbers/<number>', methods = ['GET'])
def numbers(n):
    return my_app.get_number(n)
app.run(host = "*0.0.0.0*", port = 80)
main.py
main.py
main.py M
main.py
```

And last, we have main, where we will run our app and receive the data to validate. Here we instantiate our class MyApp so we can access its methods and do the logic.

- For the first section we get any number from the 100 first declared in our database. As we can appreciate from the screenshot, all our tests pass, confirming the expected outcome.

The screenshot shows a Postman collection named "Get number from 1" with one test step named "Get number from 100". The request URL is `http://127.0.0.1/numbers/551669674`. The "Test Results" tab indicates 3/3 tests passed. The results show three green "PASS" status messages: "get specific number", "reponse time below 200ms", and "response code". The status bar at the bottom shows "Status: 200 OK", "Time: 7 ms", and "Size: 193 B".

The screenshot shows a Postman collection with a single test step named "Get number from 100". The request URL is `http://127.0.0.1/numbers/551669674`. The "Body" tab displays the JSON response: "1 551669674: 551669674".

In addition, here's what we get from the API.

- Next section we try to retrieve the record for the number 1 which is not stored in the database. As we can see from the screenshot, all the tests pass; additionally, the “response code” test passes legitimately since we are getting the HTTP code from our API’s response.

The screenshot shows the Postman application interface. In the left sidebar, there's a 'Collections' section with a 'FizzBuzzWeb' collection containing several requests: 'GET Get number from 100', 'GET get number from 100 and not found', 'POST post existing number', 'POST post new number', 'POST Range', 'DEL Delete present number', and 'DEL delete number not present'. The 'get number from 100 and not found' request is selected. The 'Tests' tab in the top navigation bar is active, showing a JavaScript test script:

```

1 pm.test("get specific number", function(){
2     pm.response.to.have.body("Not Found")
3 });
4
5 pm.test("reponse time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 });
8
9 pm.test("response code", function(){
10    pm.response.to.have.status(404);
11 });

```

Below the script, the 'Test Results' section shows three green 'PASS' status indicators for each test case. At the bottom of the interface, the status bar displays 'Status: 404 NOT FOUND' and 'Time: 8 ms'.

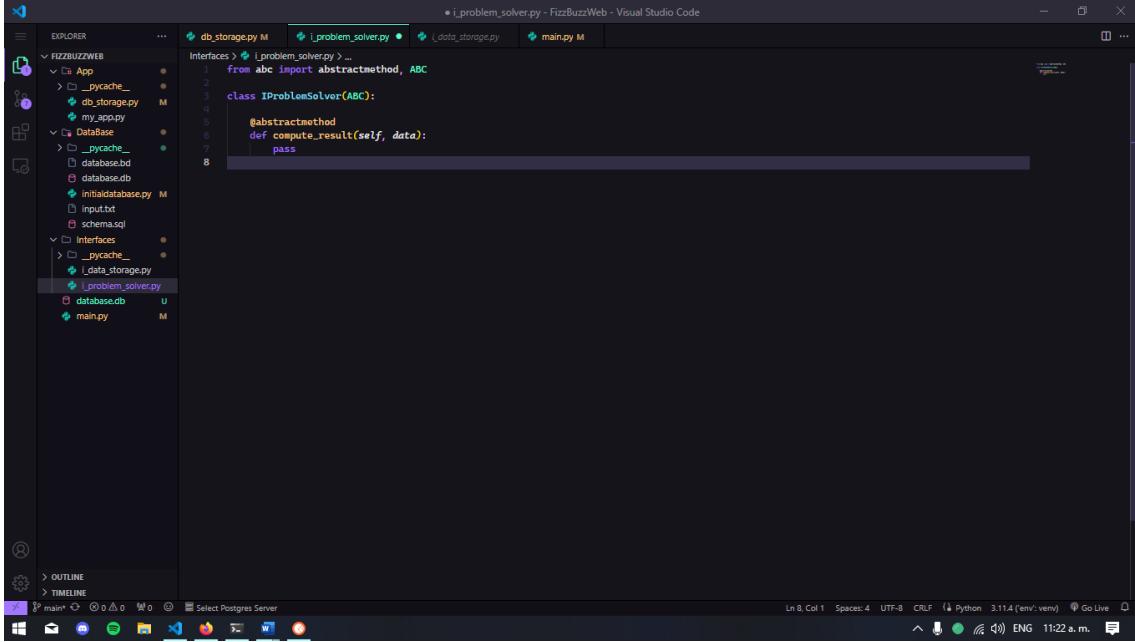
Here we make a slight modification, where we add a new method to the interface. This new method (get_active_data) in addition with “get_data” will make the process easier when it comes to retrieve data that may be active or not.

```

interfaces > i_data_storage.py > ...
1  from abc import ABC, abstractmethod
2
3  class IDataStorage(ABC):
4
5      @abstractmethod
6      def post_data(self, data):
7          pass
8
9      @abstractmethod
10     def get_data(self, data):
11         pass
12
13     @abstractmethod
14     def get_active_data(self, data):
15         pass
16
17     @abstractmethod
18     def delete_data(self, data):
19         pass
20
21     @abstractmethod
22     def get_range(self, low_limit, sup_limit):
23         pass

```

- The next section we try to add a new record of a number into the database, and here's how we are going to achieve it.



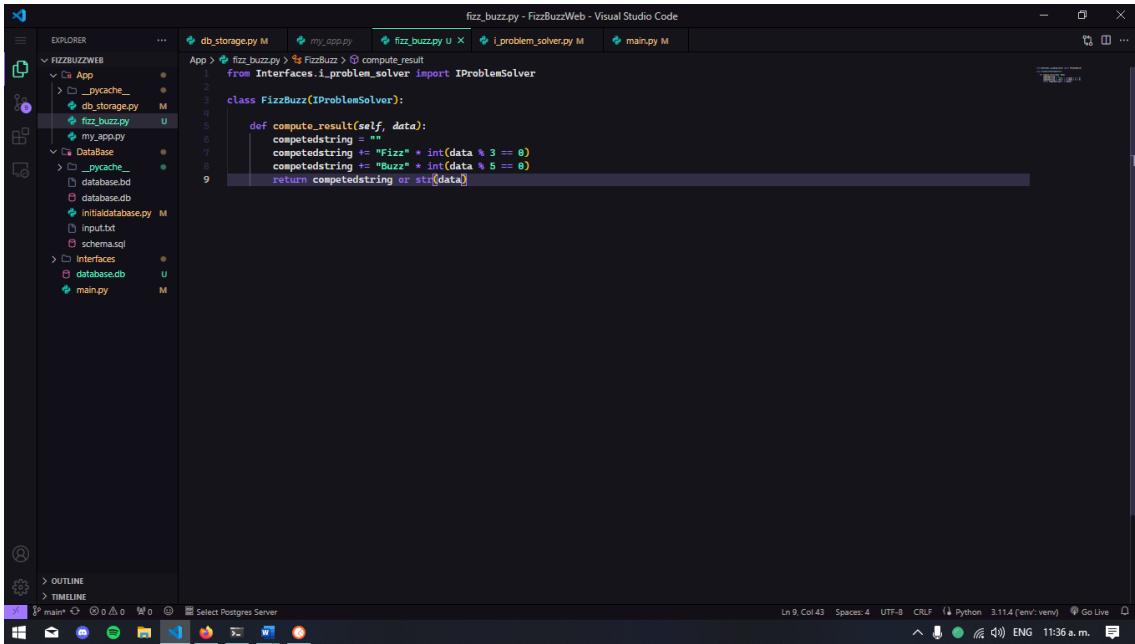
The screenshot shows the Visual Studio Code interface with the following details:

- Project Structure:** The Explorer sidebar shows the project structure under "FIZZBUZZWEB". It includes "App", "Database", and "Interfaces" folders. "App" contains "db_storage.py", "my_app.py", and "pycache_". "Database" contains "database.bd", "database.db", and "pycache_". "Interfaces" contains "i_data_storage.py" and "i_problem_solver.py".
- i_problem_solver.py Content:**

```
from abc import abstractmethod, ABC

class IProblemSolver(ABC):
    @abstractmethod
    def compute_result(self, data):
        pass
```
- Bottom Status Bar:** Shows "Ln 8, Col 1" and other standard status bar information.

First, we define our `i_problem_solver` interface which holds the abstract method `compute_result` that will return the evaluation of any given number.



The screenshot shows the Visual Studio Code interface with the following details:

- Project Structure:** The Explorer sidebar shows the project structure under "FIZZBUZZWEB". It includes "App", "Database", and "Interfaces" folders. "App" contains "db_storage.py", "fizz_buzz.py", "my_app.py", and "pycache_". "Database" contains "database.bd", "database.db", and "pycache_". "Interfaces" contains "i_data_storage.py" and "main.py".
- fizz_buzz.py Content:**

```
from Interfaces.I_problem_solver import IProblemSolver

class FizzBuzz(IProblemSolver):
    def compute_result(self, data):
        computedstring = ""
        computedstring += "Fizz" * int(data % 3 == 0)
        computedstring += "Buzz" * int(data % 5 == 0)
        return computedstring or str(data)
```
- Bottom Status Bar:** Shows "Ln 9, Col 43" and other standard status bar information.

Then, we will create or class `FizzBuzz` that contains the method to compute the result of any give number.

```

db_storage.py - FizzBuzzWeb - Visual Studio Code
my_app.py M  db_storage.py M  fizz_buzzby U  i_problem_solver.py M  main.py M  ...
App > db_storage.py:
class MyApp:
    SELECT * FROM numbers WHERE num = ?
    def get_number(self, number):
        db_result = self.storage.get_data(number)
        if db_result is None:
            return "Not Found", 404
        else:
            return f"[db_result[0]]: {db_result[1]}, 200

    def post_number(self, number):
        db_active_result = self.storage.get_active_data(number)
        db_result = self.storage.get_data(number)

        if db_active_result is None:
            if db_result is None:
                self.storage.post_data([number, FizzBuzz.compute_result(number)])
            return f"[number], {FizzBuzz.compute_result(number)}", 201
        else:
            return db_result, 200

    def post_data(self, data):
        connection = self.db.connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("INSERT INTO numbers (num, num_evaluated, active) VALUES (?, ?, ?)", (data[0], data[1]))
        connection.commit()
        connection.close()
        return

    def delete_data(self, data):
        pass

    def get_range(self, low_limit, sup_limit):
        pass

```

```

my_app.py M  db_storage.py M  fizz_buzzby U  i_problem_solver.py M  main.py M  ...
App > db_storage.py > DBStorage > delete_data
class DBStorage(IDataStorage):
    def get_active_data(self, data):
        result = cursor.execute("SELECT num, num_evaluated FROM numbers WHERE num = ? AND active = '1'", (data,)).fetchone()
        connection.close()
        return result

    def get_data(self, data):
        connection = self.db.connection()
        with connection:
            cursor = connection.cursor()
            result = cursor.execute("SELECT num, num_evaluated FROM numbers WHERE num = ?", (data,)).fetchone()
            connection.close()
        return result

    def post_data(self, data):
        connection = self.db.connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("INSERT INTO numbers (num, num_evaluated, active) VALUES (?, ?, ?)", (data[0], data[1]))
        connection.commit()
        connection.close()
        return

    def delete_data(self, data):
        pass

    def get_range(self, low_limit, sup_limit):
        pass

```

Ln 41, Col 26 Spaces: 4 UTF-8 CRLF Python 3.11.4 (env: venv) Go Live

Here, in the red box, we develop the logic to add the new record into the data base, but we have problem, if the number we are trying to add is stored but not active, we haven't declared the method to set its status back to active, so we first need to declare that before proceeding with the logic.

In the Yellow box we implement the modification mentioned earlier, now "get_data" will retrieve a number no matter if it is not active, and "get_active_data" will retrieve the number only if it is active.

```

Places > _data_storage.py > IDatadStorage > update_inactive_data
from abc import ABC, abstractmethod

class IDatadStorage(ABC):

    @abstractmethod
    def post_data(self, data):
        pass

    @abstractmethod
    def get_data(self, data):
        pass

    @abstractmethod
    def get_active_data(self, data):
        pass

    @abstractmethod
    def delete_data(self, data):
        pass

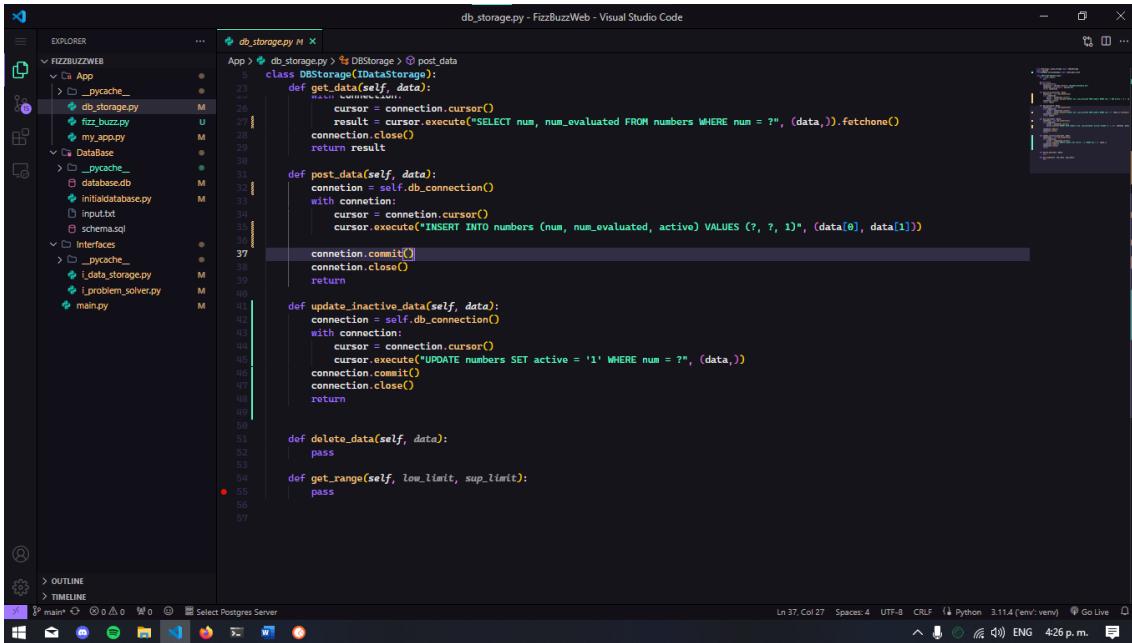
    @abstractmethod
    def get_range(self, low_limit, sup_limit):
        pass

    @abstractmethod
    def update_inactive_data(self, data):
        pass

```

Now we add the new abstract method to our interface to set the status of any given number back to active.

After that we develop the query to set the number's state back to active.



The screenshot shows the Visual Studio Code interface with the file `db_storage.py` open. The code defines a class `DBStorage` with methods for getting, posting, updating, and deleting data from a database. The `update_inactive_data` method is highlighted in the code editor.

```
class DBStorage(IDataStorage):
    def get_inactive_data(self, data):
        cursor = connection.cursor()
        result = cursor.execute("SELECT num, num_evaluated FROM numbers WHERE num = ? ", (data,)).fetchone()
        connection.close()
        return result

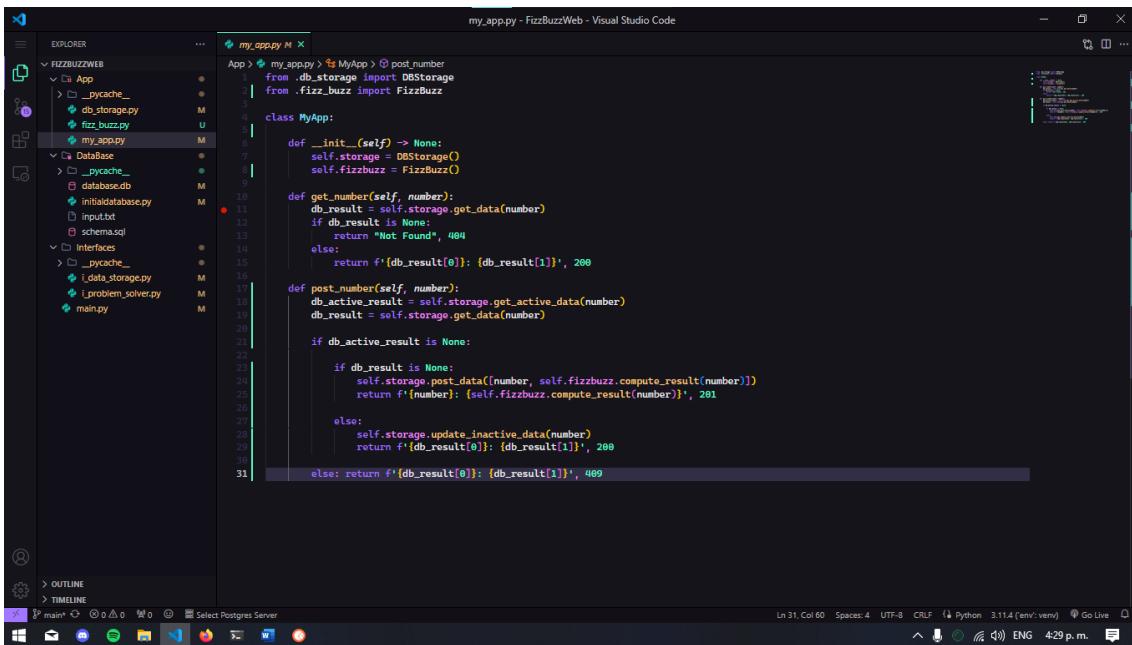
    def post_data(self, data):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("INSERT INTO numbers (num, num_evaluated, active) VALUES (?, ?, 1)", (data[0], data[1]))
        connection.commit()
        connection.close()
        return

    def update_inactive_data(self, data):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("UPDATE numbers SET active = '1' WHERE num = ? ", (data,))
        connection.commit()
        connection.close()
        return

    def delete_data(self, data):
        pass

    def get_range(self, low_limit, sup_limit):
        pass
```

Because we finally developed the logic for updating our number's state, we can continue developing the logic for the post verb, which follows: If the number is not active, we check if it is stored, if not, we add the record into the data base; if it is stored but not active, we set its status back to active, and if it is active, we only return its value.



The screenshot shows the Visual Studio Code interface with the file `my_app.py` open. The code defines a class `MyApp` with methods for getting and posting numbers. The `post_number` method is highlighted in the code editor.

```
class MyApp:
    def __init__(self) -> None:
        self.storage = DBStorage()
        self.fizzbuzz = FizzBuzz()

    def get_number(self, number):
        db_result = self.storage.get_data(number)
        if db_result is None:
            return "Not Found", 404
        else:
            return f'{db_result[0]}: {db_result[1]}', 200

    def post_number(self, number):
        db_active_result = self.storage.get_active_data(number)
        db_result = self.storage.get_data(number)

        if db_active_result is None:
            if db_result is None:
                self.storage.post_data([number, self.fizzbuzz.compute_result(number)])
                return f'{number}: {self.fizzbuzz.compute_result(number)}', 201
            else:
                self.storage.update_inactive_data(number)
                return f'{db_result[0]}: {db_result[1]}', 200
        else:
            if db_result[0] != number:
                self.storage.update_inactive_data(number)
                return f'{db_result[0]}: {db_result[1]}', 409
```

And with that define, we can now add our new record of the number 1.

The screenshot shows the Postman interface with a dark theme. On the left, the 'Collections' sidebar lists several items under 'FizzBuzzWeb': 'Get number from 100', 'Get number from 100 and not f...', 'post post existing number', 'POST post new number' (which is selected), 'POST Range', 'DEL Delete present number', and 'DEL delete number not present'. The main workspace shows a POST request to 'http://127.0.0.1/numbers/1'. The 'Tests' tab contains the following JavaScript code:

```
1 pm.test("post new number", function(){
2     pm.response.to.have.body("1");
3 }
4
5 pm.test("response time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 }
8
9 pm.test("response code", function(){
10    pm.response.to.have.status(201);
11 });
12
```

The 'Test Results' section shows three green 'PASS' status indicators: 'post new number', 'reponse time below 200ms', and 'response code'. At the bottom right, the status bar shows 'Status: 201 CREATED Time: 172 ms Size: 181 B'.

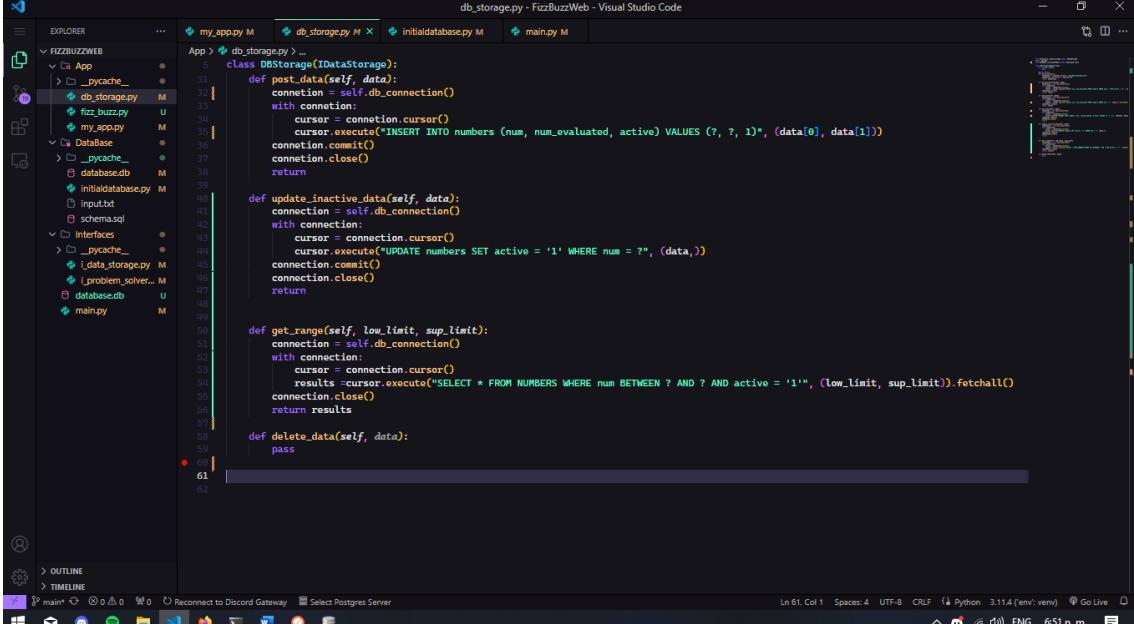
- In this section we try adding a number that is already stored in the data base, in this case, 551669674 is our test subject.

The screenshot shows the Postman interface with a dark theme. The 'Collections' sidebar is identical to the previous screenshot. The main workspace shows a POST request to 'http://127.0.0.1/numbers/551669674'. The 'Tests' tab contains the same JavaScript code as the previous screenshot. The 'Test Results' section shows three green 'PASS' status indicators: 'Post existing number', 'reponse time below 200ms', and 'response code'. However, at the bottom right, the status bar shows 'Status: 409 CONFLICT Time: 8 ms Size: 199 B', indicating a failure due to the number already existing.

We could try adding a number that is not active, but since we haven't yet defined the logic for that, we will test that out later on.

- In this section we try retrieving the range of the records for the limits established back in the report.

First, we define the query logic to get the range.



```

db_storage.py - FizzBuzzWeb - Visual Studio Code

my_app.py M db_storage.py M initialdatabase.py M main.py M

class DBStorage(IDataStorage):
    def post_data(self, data):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("INSERT INTO numbers (num, num_evaluated, active) VALUES (?, ?, ?)", (data[0], data[1]))
            connection.commit()
            connection.close()
        return

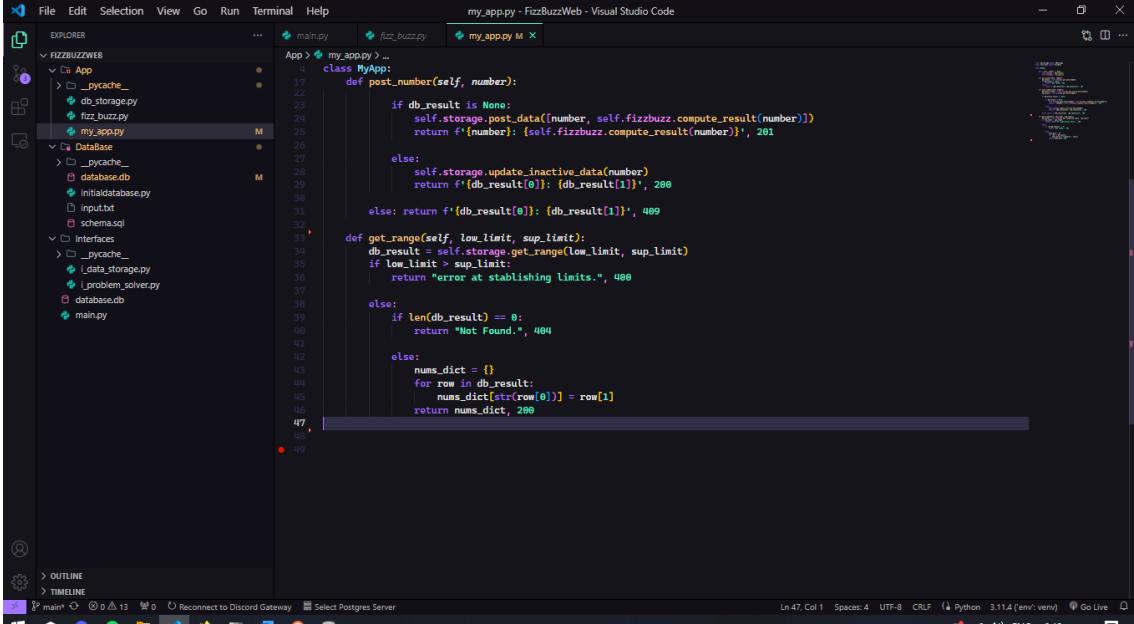
    def update_inactive_data(self, data):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("UPDATE numbers SET active = '1' WHERE num = ?", (data,))
            connection.commit()
            connection.close()
        return

    def get_range(self, low_limit, sup_limit):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            results = cursor.execute("SELECT * FROM NUMBERS WHERE num BETWEEN ? AND ? AND active = '1'", (low_limit, sup_limit)).fetchall()
            connection.close()
        return results

    def delete_data(self, data):
        pass

```

Then we define our API's response depending on the result retrieved from the database.



```

my_app.py - FizzBuzzWeb - Visual Studio Code

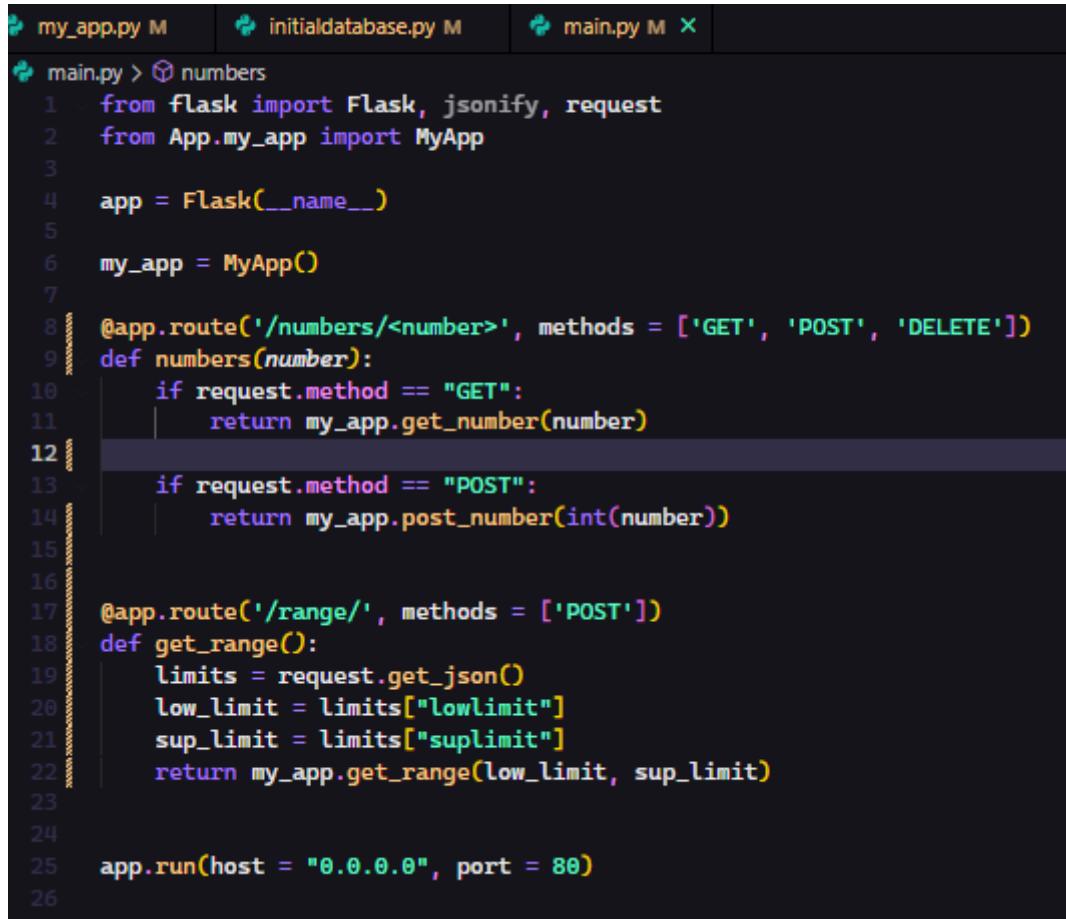
main.py fizz_buzz.py my_app.py M

class MyApp:
    def post_number(self, number):
        if db_result is None:
            self.storage.post_data(number, self.fizzbuzz.compute_result(number))
            return f'{number}: {self.fizzbuzz.compute_result(number)}', 201
        else:
            self.storage.update_inactive_data(number)
            return f'{db_result[0]}: {db_result[1]}', 200
        else:
            return f'{db_result[0]}: {db_result[1]}', 409

    def get_range(self, low_limit, sup_limit):
        db_result = self.storage.get_range(low_limit, sup_limit)
        if low_limit > sup_limit:
            return "error at establishing limits.", 400
        else:
            if len(db_result) == 0:
                return "Not Found.", 404
            else:
                nums_dict = {}
                for row in db_result:
                    nums_dict[str(row[0])] = row[1]
                return nums_dict, 200

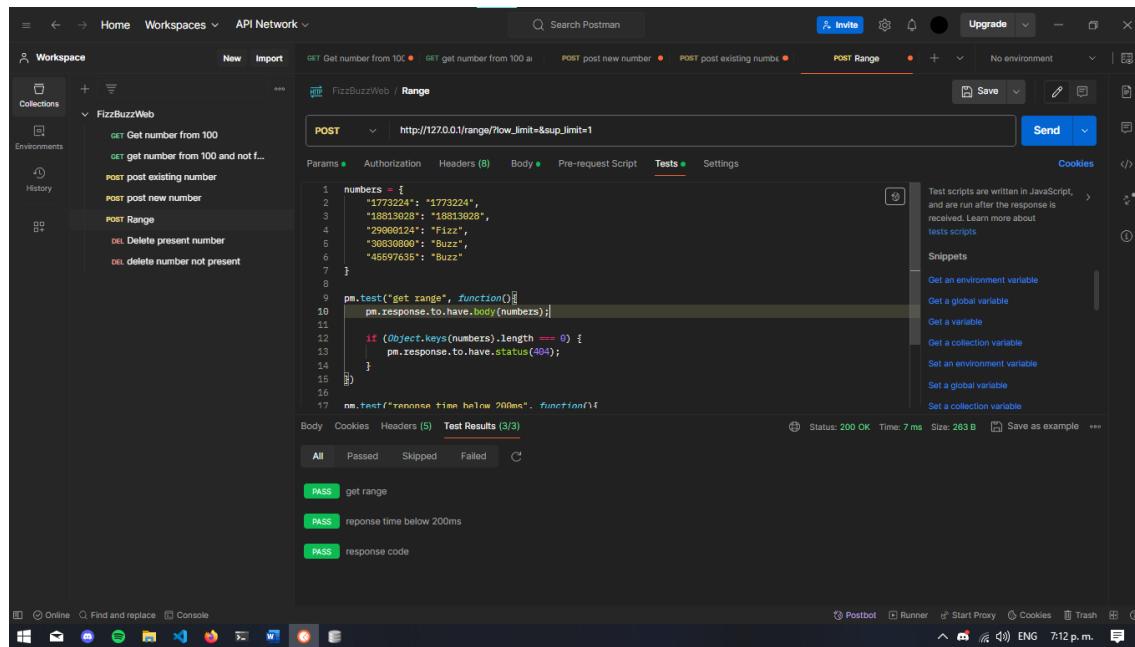
```

The final part consists on developing the logic to receive the request data from Postman and sending it into our API.



```
my_app.py M initialdatabase.py M main.py M X
main.py > ⚡ numbers
1  from flask import Flask, jsonify, request
2  from App.my_app import MyApp
3
4  app = Flask(__name__)
5
6  my_app = MyApp()
7
8  @app.route('/numbers/<number>', methods = ['GET', 'POST', 'DELETE'])
9  def numbers(number):
10     if request.method == "GET":
11         return my_app.get_number(number)
12
13     if request.method == "POST":
14         return my_app.post_number(int(number))
15
16
17  @app.route('/range/', methods = ['POST'])
18  def get_range():
19      limits = request.get_json()
20      low_limit = limits["lowlimit"]
21      sup_limit = limits["suplimit"]
22      return my_app.get_range(low_limit, sup_limit)
23
24
25  app.run(host = "0.0.0.0", port = 80)
26
```

Here's what we get: all the tests pass and we are ready to go to the next section.



The screenshot shows the Postman interface with a collection named "FizzBuzzWeb". A POST request to `/range/?low_limit=&sup_limit=1` is selected. The "Tests" tab is active, displaying a JavaScript test script:

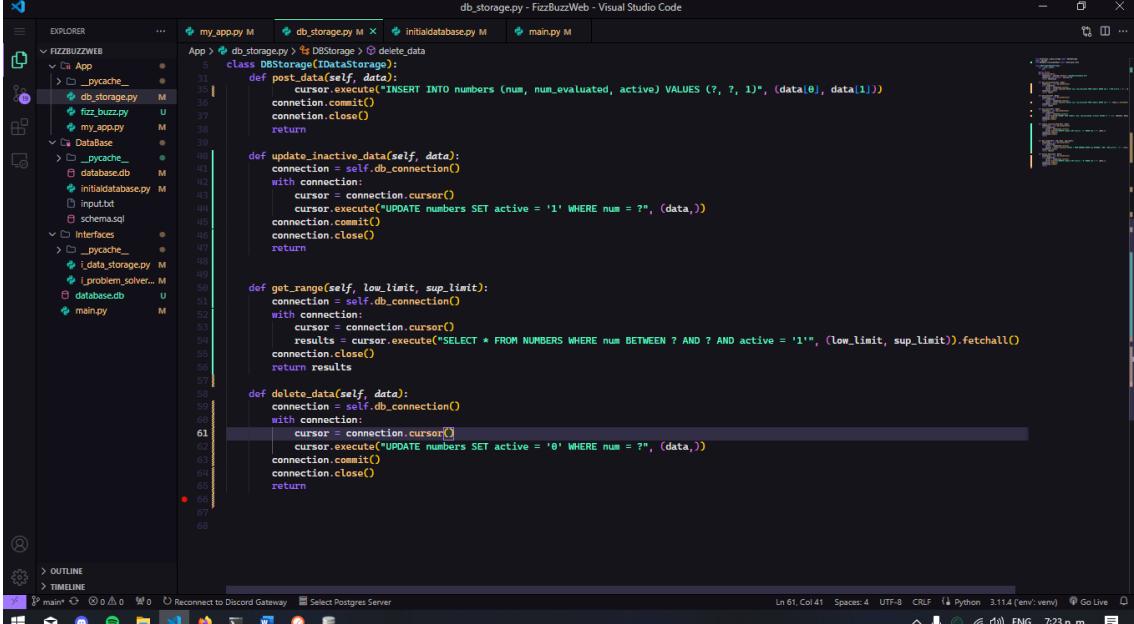
```
1  numbers = [
2      "1773224": "Fizz",
3      "18813628": "Buzz",
4      "29960124": "Fizz",
5      "30830800": "Buzz",
6      "46597635": "Buzz"
7  ]
8
9  pm.test("get range", function(){
10     pm.response.to.have.body(numbers);
11
12     if (Object.keys(numbers).length === 0) {
13         pm.response.to.have.status(404);
14     }
15
16
17     pm.test("response time below 200ms", function(){
18
19     })
20 })
```

The "Test Results" section shows three green "PASS" status indicators:

- PASS get range
- PASS response time below 200ms
- PASS response code

- In this section we try to “delete” the record of a stored number from our database.

First, we define the logic for our SQL query, which is going to perform a soft deletion to our requested number.



```

db_storage.py - FizzBuzzWeb - Visual Studio Code
my_app.py M db_storage.py M initdatabase.py M main.py M
App > db_storage.py > DStorage > delete_data
class DBStorage(IDataStorage):
    def post_data(self, data):
        cursor.execute("INSERT INTO numbers (num, num_evaluated, active) VALUES (?, ?, ?)", (data[0], data[1]))
        connection.commit()
        connection.close()
        return

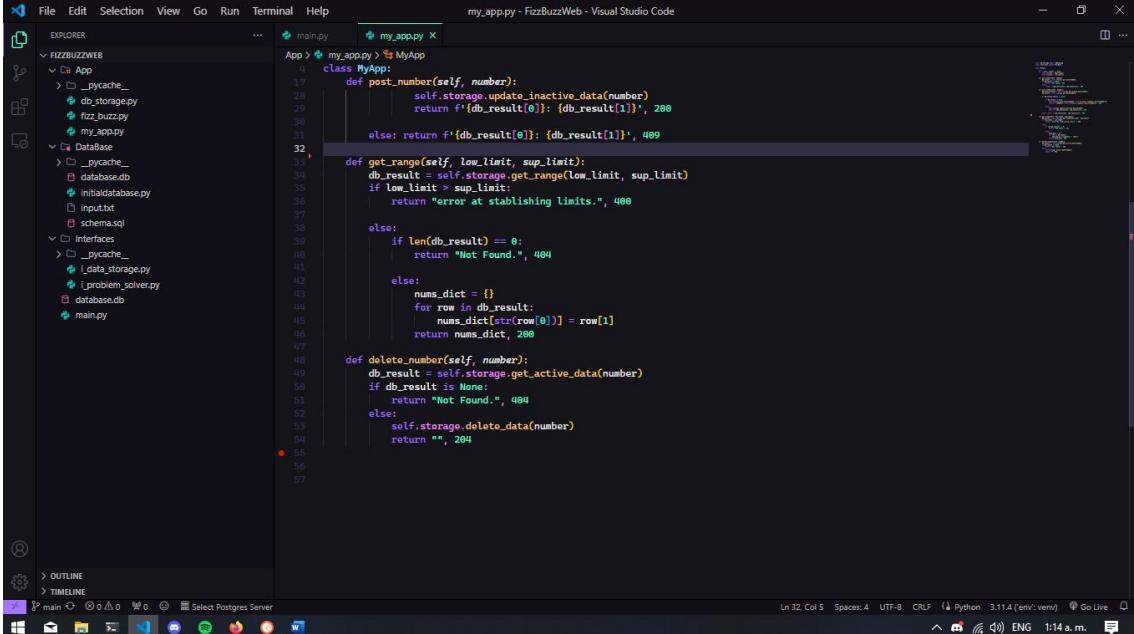
    def update_inactive_data(self, data):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("UPDATE numbers SET active = '1' WHERE num = ?", (data,))
            connection.commit()
            connection.close()
        return

    def get_range(self, low_limit, sup_limit):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            results = cursor.execute("SELECT * FROM NUMBERS WHERE num BETWEEN ? AND ? AND active = '1'", (low_limit, sup_limit)).fetchall()
            connection.close()
        return results

    def delete_data(self, data):
        connection = self.db_connection()
        with connection:
            cursor = connection.cursor()
            cursor.execute("UPDATE numbers SET active = '0' WHERE num = ?", (data,))
            connection.commit()
            connection.close()
        return

```

Then we define our API’s response depending on what we retrieve from the database.



```

File Edit Selection View Go Run Terminal Help my_app.py - FizzBuzzWeb - Visual Studio Code
main.py my_app.py X
App > my_app.py > MyApp
class MyApp:
    def post_number(self, number):
        self.storage.update_inactive_data(number)
        return f'{db_result[0]}: {db_result[1]}, 200'

    else: return f'{db_result[0]}: {db_result[1]}, 409

    def get_range(self, low_limit, sup_limit):
        db_result = self.storage.get_range(low_limit, sup_limit)
        if low_limit > sup_limit:
            return "error at establishing limits.", 400

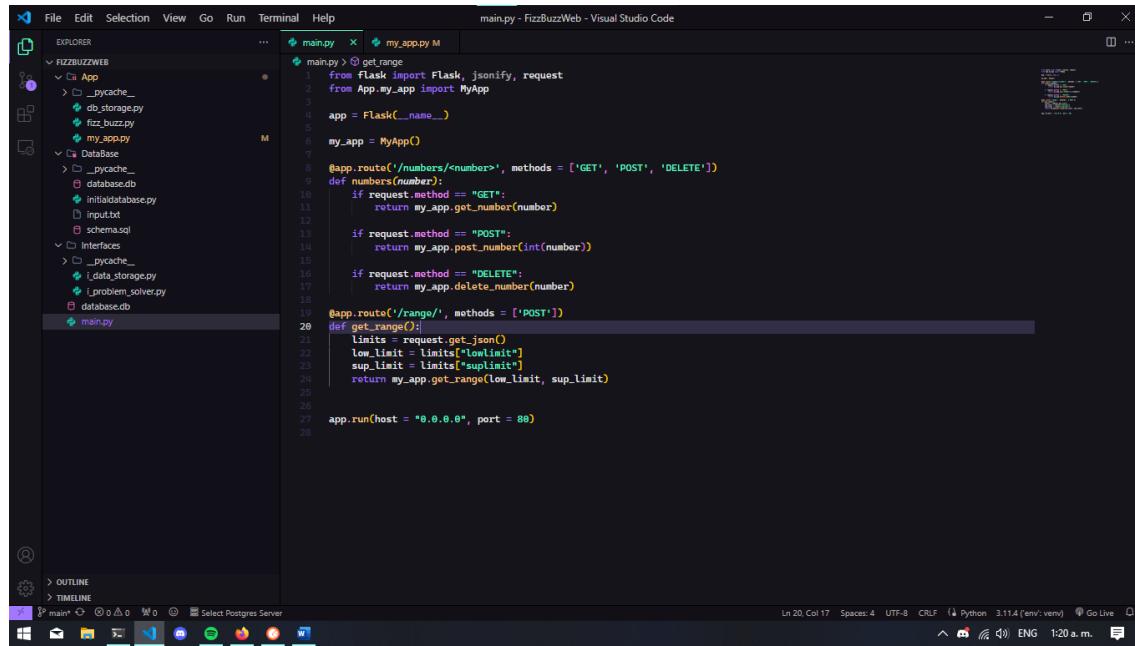
        else:
            if len(db_result) == 0:
                return "Not Found.", 404

            else:
                nums_dict = {}
                for row in db_result:
                    nums_dict[str(row[0])] = row[1]
                return nums_dict, 200

    def delete_number(self, number):
        db_result = self.storage.get_active_data(number)
        if db_result is None:
            return "Not Found.", 404
        else:
            self.storage.delete_data(number)
            return "", 204

```

Last but not least, we define the logic to receive our data.

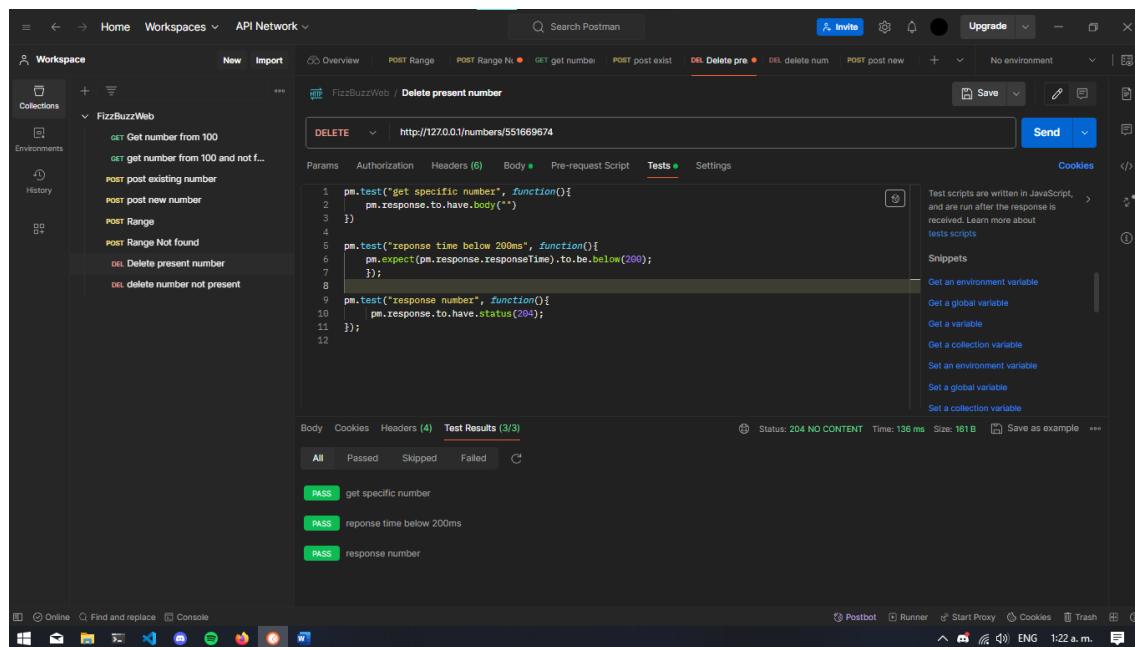


```
File Edit Selection View Go Run Terminal Help main.py - FizzBuzzWeb - Visual Studio Code

EXPLORER
FIZZBUZZWEB
  App
    __pycache__
    db_storage.py
    fizz_buzz.py
    my_app.py
  Database
    __pycache__
    database.db
    initialdatabase.py
    input.txt
    schema.sql
  Interfaces
    __pycache__
    data_storage.py
    problem_solver.py
  database.db
  main.py

main.py
1 from flask import Flask, jsonify, request
2 from App.my_app import MyApp
3
4 app = Flask(__name__)
5
6 my_app = MyApp()
7
8 @app.route('/numbers<number>', methods=['GET', 'POST', 'DELETE'])
9 def numbers(number):
10     if request.method == "GET":
11         return my_app.get_number(number)
12
13     if request.method == "POST":
14         return my_app.post_number(int(number))
15
16     if request.method == "DELETE":
17         return my_app.delete_number(number)
18
19 @app.route('/range/<range>', methods=['POST'])
20 def get_range(range):
21     limits = request.get_json()
22     low_limit = limits["lowlimit"]
23     sup_limit = limits["suplimit"]
24     return my_app.get_range(low_limit, sup_limit)
25
26
27 app.run(host = "0.0.0.0", port = 80)
```

With all the logic defined, we can continue with our tests, first we try deleting the record of a number stored in our data base, and see what we get.



The screenshot shows the Postman interface with a collection named "FizzBuzzWeb". A specific test case titled "Delete present number" is selected. The request URL is `http://127.0.0.1/numbers/551669674`. The "Tests" tab contains the following JavaScript code:

```
pm.test("get specific number", function(){
  pm.response.to.have.body("551669674");
});
pm.test("response time below 200ms", function(){
  pm.expect(pm.response.responseTime).to.be.below(200);
});
pm.test("response number", function(){
  pm.response.to.have.status(204);
});
```

The "Test Results" section shows three green "PASS" status indicators for each test step. The status bar at the bottom indicates "Status: 204 NO CONTENT Time: 136 ms Size: 161 B".

Our number 551669674 is now soft deleted, which is what we expected; now we can try setting its status back to active and see what we get:

The screenshot shows the Postman interface with a dark theme. On the left, the 'Workspace' sidebar lists a collection named 'FizzBuzzWeb' containing several requests: 'Get number from 100', 'get number from 100 and not f...', 'post post existing number', 'post post new number', 'POST Range', 'POST Range Not found', 'DEL Delete present number', and 'DEL delete number not present'. The main panel displays a POST request to 'http://127.0.0.1/numbers/551669674'. The 'Tests' tab contains the following JavaScript code:

```
1 pm.test("Post existing number", function(){
2     pm.response.to.have.body("551669674: 551669674")
3 }
4 )
5 pm.test("response time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200)
7 }
8 )
9 pm.test("response code", function(){
10    pm.response.to.have.status(200);
11 }
12 )
```

The 'Test Results' section shows three green 'PASS' status indicators: 'Post existing number', 'response time below 200ms', and 'response code'. The status bar at the bottom indicates 'Status: 200 OK'.

Now the number is active once again!

The screenshot shows the Postman interface with a dark theme. The 'Workspace' sidebar lists the same collection 'FizzBuzzWeb' with the same requests. The main panel displays a DELETE request to 'http://127.0.0.1/numbers/10'. The 'Tests' tab contains the following JavaScript code:

```
1 pm.test("get specific number", function(){
2     pm.response.to.have.body("Not Found.")
3 }
4 )
5 pm.test("response time below 200ms", function(){
6     pm.expect(pm.response.responseTime).to.be.below(200)
7 }
8 )
9 pm.test("response code", function(){
10    pm.response.to.have.status(404);
11 }
```

The 'Test Results' section shows three green 'PASS' status indicators: 'get specific number', 'response time below 200ms', and 'response code'. The status bar at the bottom indicates 'Status: 404 NOT FOUND'.

Our last test involves deleting the record for a number that isn't stored in our database. As shown in the screenshot, all of our tests pass, confirming the desired outcome.

The screenshot shows the Postman interface with a dark theme. On the left, the 'Collections' section lists a single collection named 'FizzBuzzWeb'. Underneath it, several test cases are listed: 'GET Get number from 100', 'GET get number from 100 and not f...', 'POST post existing number', 'POST post new number', 'POST Range', 'POST Range Not found', 'DEL Delete present number', and 'DEL delete number not present'. The main area displays the 'Run results' for a recent run. The summary table shows: Source 'Runner', Environment 'none', Iterations '1', Duration '1s 462ms', All tests '24', and Avg. Resp. Time '51 ms'. Below this is a 'RUN SUMMARY' table with rows for each test case, each showing a green bar indicating success (3|0). At the bottom right of the main window, there are buttons for 'Run Again', 'Automate Run', 'New Run', and 'Export Results'. The bottom of the screen shows the Windows taskbar with various pinned icons.

Now, we can run our collection of tests, with a fresh instance of our data base (the first 100 numbers) all our tests pass; let's try running it one more time.

This screenshot shows the same Postman session after another run. The 'Run results' summary table now shows a duration of '1s 146ms' and an average response time of '29 ms'. In the 'RUN SUMMARY' table, the 'POST post new number' test has failed, indicated by a red 'X' in the status column. Other tests remain successful with green bars. The bottom of the screen shows the Windows taskbar.

After running it once again, we get some errors, which makes sense, for instance:

The “Post new number” test will add a new record of a number into the database, if we run it twice, our API will return a different HTTP code making the test fail, same goes for “Delete present number”, besides the fact if the number is no longer active, our API will return a different HTTP code.

BLUE PHASE (REFACTORY)

We have succeeded on creating our program, now it's time to make it a little bit pleasant. To check our code's indentation and readability, we will use a static code analysis tool for python (Pylint) what will ensure those factors.

Running the command to check one the files, we get 5.65/10 score which is pretty bad. We will try to get at least a 9, so this is going to take a while.

```
Python x Python - py main.py + 

backslash-in-string)
FizzBuzzWeb DataBase\initialdatabase.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb DataBase\initialdatabase.py:4:0: C0103: Constant name "data_base_name" doesn't conform to UPPER_CASE naming style (invalid-name)
FizzBuzzWeb DataBase\initialdatabase.py:7:0: C0103: Constant name "schema_file" doesn't conform to UPPER_CASE naming style (invalid-name)
FizzBuzzWeb DataBase\initialdatabase.py:8:0: C0103: Constant name "data_base_route" doesn't conform to UPPER_CASE naming style (invalid-name)
FizzBuzzWeb DataBase\initialdatabase.py:10:0: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb DataBase\initialdatabase.py:11:9: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
FizzBuzzWeb DataBase\initialdatabase.py:14:0: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb DataBase\initialdatabase.py:26:0: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb DataBase\initialdatabase.py:33:0: C0116: Missing function or method docstring (missing-function-docstring)
***** Module Interfaces.i_data_storage
FizzBuzzWeb\Interfaces\i_data_storage.py:12:0: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\Interfaces\i_data_storage.py:27:0: C0304: Final newline missing (missing-final-newline)
FizzBuzzWeb\Interfaces\i_data_storage.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\Interfaces\i_data_storage.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\Interfaces\i_data_storage.py:6:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\Interfaces\i_data_storage.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\Interfaces\i_data_storage.py:14:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\Interfaces\i_data_storage.py:18:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\Interfaces\i_data_storage.py:22:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\Interfaces\i_data_storage.py:26:4: C0116: Missing function or method docstring (missing-function-docstring)
***** Module Interfaces.i_problem_solver
FizzBuzzWeb\Interfaces\i_problem_solver.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\Interfaces\i_problem_solver.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\Interfaces\i_problem_solver.py:6:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\Interfaces\i_problem_solver.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 5.65/10

(env) D:\My Stuff\Python\Medium\Advanced Programming Techniques>
```

This is the initial evaluation for every file in our project.

```
Python x Python - main.py + ~
-backslash-in-string)
FizzBuzzWeb DataBase\initialdatabase.py:14:9: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
*****
Module Interfaces.i_problem_solver
FizzBuzzWeb\Interfaces\i_problem_solver.py:5:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 9.35/10 (previous run: 9.42/10, -0.06)

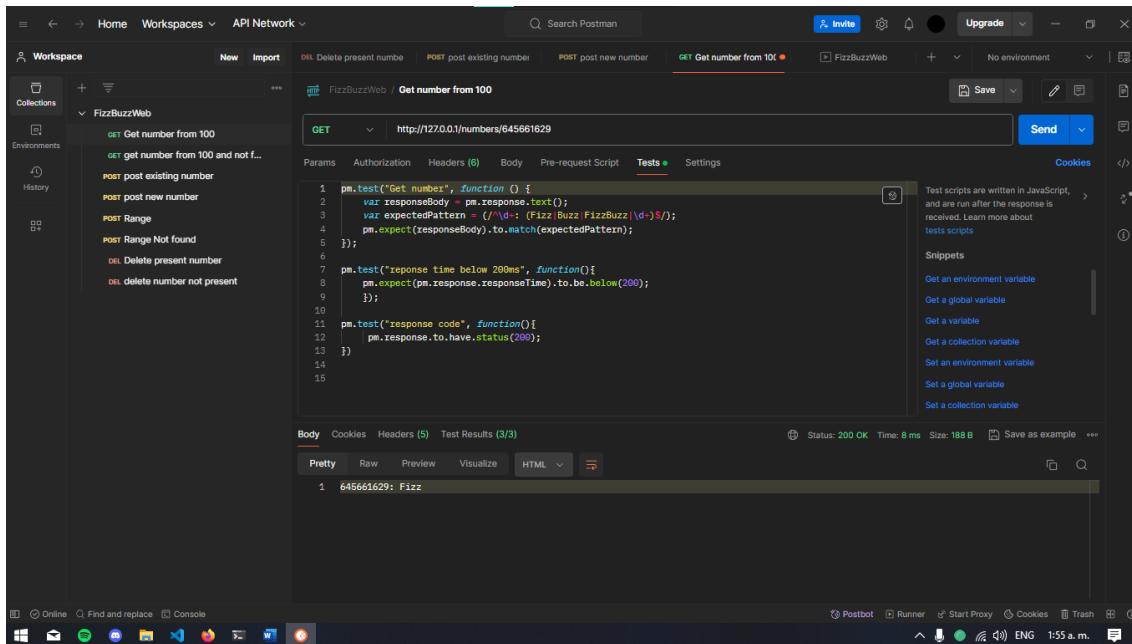
(env) D:\My Stuff\Python\Medium\Advanced Programming Techniques>
(env) D:\My Stuff\Python\Medium\Advanced Programming Techniques> pylint FizzBuzzWeb
*****
Module App.db_storage
FizzBuzzWeb\App\db_storage.py:34:0: C0301: Line too long (108/100) (line-too-long)
FizzBuzzWeb\App\db_storage.py:46:0: C0301: Line too long (111/100) (line-too-long)
FizzBuzzWeb\App\db_storage.py:80:0: C0301: Line too long (104/100) (line-too-long)
*****
Module App.fizz_buzz
FizzBuzzWeb\App\fizz_buzz.py:5:0: R0903: Too few public methods (1/2) (too-few-public-methods)
*****
Module DataBase.initialdatabase
FizzBuzzWeb DataBase\initialdatabase.py:10:20: W1401: Anomalous backslash in string: '\D'. String constant might be missing an r prefix. (anomalous-backslash-in-string)
FizzBuzzWeb DataBase\initialdatabase.py:10:29: W1401: Anomalous backslash in string: '\d'. String constant might be missing an r prefix. (anomalous-backslash-in-string)
FizzBuzzWeb DataBase\initialdatabase.py:14:9: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
*****
Module Interfaces.i_problem_solver
FizzBuzzWeb\Interfaces\i_problem_solver.py:5:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 9.48/10 (previous run: 9.35/10, +0.13)

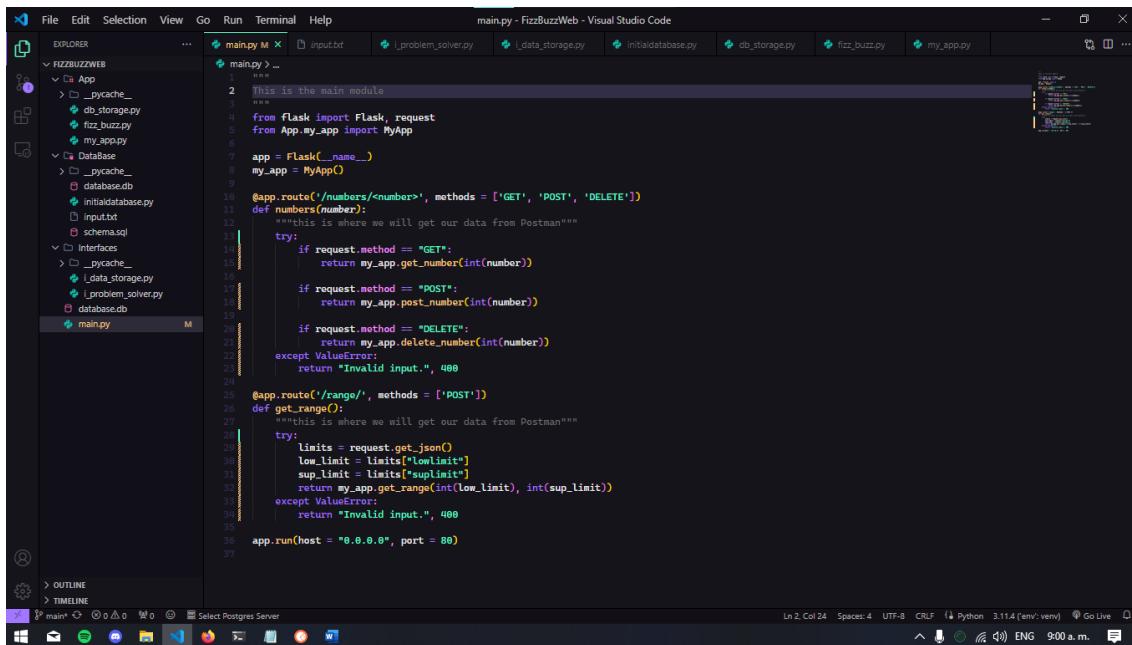
(env) D:\My Stuff\Python\Medium\Advanced Programming Techniques>
```

After some corrections, the max score we get is 9.48/10 which is fairly satisfactory.

AUTOMATIZING TESTS



This slight improvement will make our life easier when it comes to testing different numbers, instead of manually switching the test, now we only need to enter the number in the URL and call it a day.



To handle any non-expected input error, we use a try except block in order to display a message for any bad request.

Last modification will be in our “MyApp” constructor, we set our instances of “DBStorage” and “FizzBuzz” to be their interface type, since they both satisfy their respectively interface.

CONCLUSIONS:

So, we are finally done with our project, this challenge really thought me a lot about HTTP response status code, API's (I wasn't really aware of APIs until this very semester), besides applying the DIP, which at the beginning I was hard struggling to abstract the structure, but the more I spent time on it, the more I got used to it, so at the end I was comfortable working with it, and it really makes the code way cleaner (when I was first messing around with it, it was lots of repetitive code).

The screenshot shows a Visual Studio Code interface with two open files: `main.py` and `schema.sql`. The `main.py` file contains Python code for a Flask application. The `schema.sql` file contains an SQL script to create a table named `numbers` with columns `num`, `num_evaluated`, and `active`. A red box highlights the `schema.sql` file in the center of the editor.

```
main.py
16 |     return count > 0
17 |
18 | app = Flask(__name__)
19 |
20 | @app.route('/')
21 | def index():
22 |     return "Home"
23 |
24 | #GET
25 | @app.route('/numbers/', methods=['GET'])
26 | def numbers_get():
27 |     connection = data_base_connection()
28 |     number = request.args.get('num')
29 |     cursor = connection.cursor()
30 |
31 |     cursor.execute('SELECT num, num_evaluated FROM numbers WHERE num = ?',
32 |                   (number,))
33 |     result = cursor.fetchone()
34 |     if result is not None:
35 |         return f'{result[0]}: {result[1]}'
36 |     else:
37 |         return "Number not found.", 404
38 |
39 | #POST
40 | @app.route('/numbers/', methods=['POST'])
41 | def numbers_post():
42 |     connection = data_base_connection()
43 |     number = request.get_json()['num']
44 |     cursor = connection.cursor()
45 |     evaluated_number = fizz_buzz.check_numbers(number)
46 |
47 |     if number_in_data_base(number):
48 |         return "number already in data base", 408
49 |     else:
50 |         cursor.execute("INSERT INTO numbers (num, num_evaluated, active)
51 |                         VALUES (?, ?, ?)", (number, evaluated_number, True))
52 |         connection.commit()
53 |         connection.close()
54 |         return f'{number} has been added with success.', 201
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100 |
101 |
102 |
103 |
104 |
105 |
106 |
107 |
108 |
109 |
110 |
111 |
112 |
113 |
114 |
115 |
116 |
117 |
118 |
119 |
120 |
121 |
122 |
123 |
124 |
125 |
126 |
127 |
128 |
129 |
130 |
131 |
132 |
133 |
134 |
135 |
136 |
137 |
138 |
139 |
140 |
141 |
142 |
143 |
144 |
145 |
146 |
147 |
148 |
149 |
150 |
151 |
152 |
153 |
154 |
155 |
156 |
157 |
158 |
159 |
160 |
161 |
162 |
163 |
164 |
165 |
166 |
167 |
168 |
169 |
170 |
171 |
172 |
173 |
174 |
175 |
176 |
177 |
178 |
179 |
180 |
181 |
182 |
183 |
184 |
185 |
186 |
187 |
188 |
189 |
190 |
191 |
192 |
193 |
194 |
195 |
196 |
197 |
198 |
199 |
200 |
201 |
202 |
203 |
204 |
205 |
206 |
207 |
208 |
209 |
210 |
211 |
212 |
213 |
214 |
215 |
216 |
217 |
218 |
219 |
220 |
221 |
222 |
223 |
224 |
225 |
226 |
227 |
228 |
229 |
230 |
231 |
232 |
233 |
234 |
235 |
236 |
237 |
238 |
239 |
240 |
241 |
242 |
243 |
244 |
245 |
246 |
247 |
248 |
249 |
250 |
251 |
252 |
253 |
254 |
255 |
256 |
257 |
258 |
259 |
259 |
260 |
261 |
262 |
263 |
264 |
265 |
266 |
267 |
268 |
269 |
270 |
271 |
272 |
273 |
274 |
275 |
276 |
277 |
278 |
279 |
279 |
280 |
281 |
282 |
283 |
284 |
285 |
286 |
287 |
288 |
289 |
289 |
290 |
291 |
292 |
293 |
294 |
295 |
296 |
297 |
298 |
299 |
299 |
300 |
301 |
302 |
303 |
304 |
305 |
306 |
307 |
308 |
309 |
309 |
310 |
311 |
312 |
313 |
314 |
315 |
316 |
317 |
318 |
319 |
319 |
320 |
321 |
322 |
323 |
324 |
325 |
326 |
327 |
328 |
329 |
329 |
330 |
331 |
332 |
333 |
334 |
335 |
336 |
337 |
338 |
339 |
339 |
340 |
341 |
342 |
343 |
344 |
345 |
346 |
347 |
348 |
349 |
349 |
350 |
351 |
352 |
353 |
354 |
355 |
356 |
357 |
358 |
359 |
359 |
360 |
361 |
362 |
363 |
364 |
365 |
366 |
367 |
368 |
369 |
369 |
370 |
371 |
372 |
373 |
374 |
375 |
376 |
377 |
378 |
379 |
379 |
380 |
381 |
382 |
383 |
384 |
385 |
386 |
387 |
388 |
389 |
389 |
390 |
391 |
392 |
393 |
394 |
395 |
396 |
397 |
398 |
399 |
399 |
400 |
401 |
402 |
403 |
404 |
405 |
406 |
407 |
408 |
409 |
409 |
410 |
411 |
412 |
413 |
414 |
415 |
416 |
417 |
418 |
419 |
419 |
420 |
421 |
422 |
423 |
424 |
425 |
426 |
427 |
428 |
429 |
429 |
430 |
431 |
432 |
433 |
434 |
435 |
436 |
437 |
438 |
439 |
439 |
440 |
441 |
442 |
443 |
444 |
445 |
446 |
447 |
448 |
449 |
449 |
450 |
451 |
452 |
453 |
454 |
455 |
456 |
457 |
458 |
459 |
459 |
460 |
461 |
462 |
463 |
464 |
465 |
466 |
467 |
468 |
469 |
469 |
470 |
471 |
472 |
473 |
474 |
475 |
476 |
477 |
478 |
479 |
479 |
480 |
481 |
482 |
483 |
484 |
485 |
486 |
487 |
488 |
489 |
489 |
490 |
491 |
492 |
493 |
494 |
495 |
496 |
497 |
498 |
499 |
499 |
500 |
501 |
502 |
503 |
504 |
505 |
506 |
507 |
508 |
509 |
509 |
510 |
511 |
512 |
513 |
514 |
515 |
516 |
517 |
518 |
519 |
519 |
520 |
521 |
522 |
523 |
524 |
525 |
526 |
527 |
528 |
529 |
529 |
530 |
531 |
532 |
533 |
534 |
535 |
536 |
537 |
538 |
539 |
539 |
540 |
541 |
542 |
543 |
544 |
545 |
546 |
547 |
548 |
549 |
549 |
550 |
551 |
552 |
553 |
554 |
555 |
556 |
557 |
558 |
559 |
559 |
560 |
561 |
562 |
563 |
564 |
565 |
566 |
567 |
568 |
569 |
569 |
570 |
571 |
572 |
573 |
574 |
575 |
576 |
577 |
578 |
579 |
579 |
580 |
581 |
582 |
583 |
584 |
585 |
586 |
587 |
588 |
589 |
589 |
590 |
591 |
592 |
593 |
594 |
595 |
596 |
597 |
598 |
599 |
599 |
600 |
601 |
602 |
603 |
604 |
605 |
606 |
607 |
608 |
609 |
609 |
610 |
611 |
612 |
613 |
614 |
615 |
616 |
617 |
618 |
619 |
619 |
620 |
621 |
622 |
623 |
624 |
625 |
626 |
627 |
628 |
629 |
629 |
630 |
631 |
632 |
633 |
634 |
635 |
636 |
637 |
638 |
639 |
639 |
640 |
641 |
642 |
643 |
644 |
645 |
646 |
647 |
648 |
649 |
649 |
650 |
651 |
652 |
653 |
654 |
655 |
656 |
657 |
658 |
659 |
659 |
660 |
661 |
662 |
663 |
664 |
665 |
666 |
667 |
668 |
669 |
669 |
670 |
671 |
672 |
673 |
674 |
675 |
676 |
677 |
678 |
679 |
679 |
680 |
681 |
682 |
683 |
684 |
685 |
686 |
687 |
688 |
689 |
689 |
690 |
691 |
692 |
693 |
694 |
695 |
696 |
697 |
698 |
698 |
699 |
699 |
700 |
701 |
702 |
703 |
704 |
705 |
706 |
707 |
708 |
709 |
709 |
710 |
711 |
712 |
713 |
714 |
715 |
716 |
717 |
718 |
719 |
719 |
720 |
721 |
722 |
723 |
724 |
725 |
726 |
727 |
728 |
729 |
729 |
730 |
731 |
732 |
733 |
734 |
735 |
736 |
737 |
738 |
739 |
739 |
740 |
741 |
742 |
743 |
744 |
745 |
746 |
747 |
748 |
749 |
749 |
750 |
751 |
752 |
753 |
754 |
755 |
756 |
757 |
758 |
759 |
759 |
760 |
761 |
762 |
763 |
764 |
765 |
766 |
767 |
768 |
769 |
769 |
770 |
771 |
772 |
773 |
774 |
775 |
776 |
777 |
778 |
779 |
779 |
780 |
781 |
782 |
783 |
784 |
785 |
786 |
787 |
788 |
789 |
789 |
790 |
791 |
792 |
793 |
794 |
795 |
796 |
797 |
798 |
798 |
799 |
799 |
800 |
801 |
802 |
803 |
804 |
805 |
806 |
807 |
808 |
809 |
809 |
810 |
811 |
812 |
813 |
814 |
815 |
816 |
817 |
818 |
819 |
819 |
820 |
821 |
822 |
823 |
824 |
825 |
826 |
827 |
828 |
829 |
829 |
830 |
831 |
832 |
833 |
834 |
835 |
836 |
837 |
838 |
839 |
839 |
840 |
841 |
842 |
843 |
844 |
845 |
846 |
847 |
848 |
849 |
849 |
850 |
851 |
852 |
853 |
854 |
855 |
856 |
857 |
858 |
859 |
859 |
860 |
861 |
862 |
863 |
864 |
865 |
866 |
867 |
868 |
869 |
869 |
870 |
871 |
872 |
873 |
874 |
875 |
876 |
877 |
878 |
878 |
879 |
879 |
880 |
881 |
882 |
883 |
884 |
885 |
886 |
887 |
888 |
888 |
889 |
889 |
890 |
891 |
892 |
893 |
894 |
895 |
896 |
897 |
898 |
898 |
899 |
899 |
900 |
901 |
902 |
903 |
904 |
905 |
906 |
907 |
908 |
909 |
909 |
910 |
911 |
912 |
913 |
914 |
915 |
916 |
917 |
918 |
919 |
919 |
920 |
921 |
922 |
923 |
924 |
925 |
926 |
927 |
928 |
929 |
929 |
930 |
931 |
932 |
933 |
934 |
935 |
936 |
937 |
938 |
939 |
939 |
940 |
941 |
942 |
943 |
944 |
945 |
946 |
947 |
948 |
949 |
949 |
950 |
951 |
952 |
953 |
954 |
955 |
956 |
957 |
958 |
959 |
959 |
960 |
961 |
962 |
963 |
964 |
965 |
966 |
967 |
968 |
969 |
969 |
970 |
971 |
972 |
973 |
974 |
975 |
976 |
977 |
978 |
978 |
979 |
979 |
980 |
981 |
982 |
983 |
984 |
985 |
986 |
987 |
988 |
988 |
989 |
989 |
990 |
991 |
992 |
993 |
994 |
995 |
996 |
997 |
998 |
998 |
999 |
999 |
1000 |
1001 |
1002 |
1003 |
1004 |
1005 |
1006 |
1007 |
1008 |
1009 |
1009 |
1010 |
1011 |
1012 |
1013 |
1014 |
1015 |
1016 |
1017 |
1018 |
1019 |
1019 |
1020 |
1021 |
1022 |
1023 |
1024 |
1025 |
1026 |
1027 |
1028 |
1029 |
1029 |
1030 |
1031 |
1032 |
1033 |
1034 |
1035 |
1036 |
1037 |
1038 |
1039 |
1039 |
1040 |
1041 |
1042 |
1043 |
1044 |
1045 |
1046 |
1047 |
1048 |
1049 |
1049 |
1050 |
1051 |
1052 |
1053 |
1054 |
1055 |
1056 |
1057 |
1058 |
1059 |
1059 |
1060 |
1061 |
1062 |
1063 |
1064 |
1065 |
1066 |
1067 |
1068 |
1069 |
1069 |
1070 |
1071 |
1072 |
1073 |
1074 |
1075 |
1076 |
1077 |
1078 |
1078 |
1079 |
1079 |
1080 |
1081 |
1082 |
1083 |
1084 |
1085 |
1086 |
1087 |
1088 |
1088 |
1089 |
1089 |
1090 |
1091 |
1092 |
1093 |
1094 |
1095 |
1096 |
1096 |
1097 |
1097 |
1098 |
1099 |
1099 |
1100 |
1101 |
1102 |
1103 |
1104 |
1105 |
1106 |
1107 |
1108 |
1109 |
1109 |
1110 |
1111 |
1112 |
1113 |
1114 |
1115 |
1116 |
1117 |
1118 |
1119 |
1119 |
1120 |
1121 |
1122 |
1123 |
1124 |
1125 |
1126 |
1127 |
1128 |
1129 |
1129 |
1130 |
1131 |
1132 |
1133 |
1134 |
1135 |
1136 |
1137 |
1138 |
1139 |
1139 |
1140 |
1141 |
1142 |
1143 |
1144 |
1145 |
1146 |
1147 |
1148 |
1149 |
1149 |
1150 |
1151 |
1152 |
1153 |
1154 |
1155 |
1156 |
1157 |
1158 |
1159 |
1159 |
1160 |
1161 |
1162 |
1163 |
1164 |
1165 |
1166 |
1167 |
1168 |
1169 |
1169 |
1170 |
1171 |
1172 |
1173 |
1174 |
1175 |
1176 |
1177 |
1178 |
1178 |
1179 |
1179 |
1180 |
1181 |
1182 |
1183 |
1184 |
1185 |
1186 |
1187 |
1188 |
1188 |
1189 |
1189 |
1190 |
1191 |
1192 |
1193 |
1194 |
1195 |
1196 |
1196 |
1197 |
1197 |
1198 |
1199 |
1199 |
1200 |
1201 |
1202 |
1203 |
1204 |
1205 |
1206 |
1207 |
1208 |
1209 |
1209 |
1210 |
1211 |
1212 |
1213 |
1214 |
1215 |
1216 |
1217 |
1218 |
1219 |
1219 |
1220 |
1221 |
1222 |
1223 |
1224 |
1225 |
1226 |
1227 |
1228 |
1229 |
1229 |
1230 |
1231 |
1232 |
1233 |
1234 |
1235 |
1236 |
1237 |
1238 |
1239 |
1239 |
1240 |
1241 |
1242 |
1243 |
1244 |
1245 |
1246 |
1247 |
1248 |
1249 |
1249 |
1250 |
1251 |
1252 |
1253 |
1254 |
1255 |
1256 |
1257 |
1258 |
1259 |
1259 |
1260 |
1261 |
1262 |
1263 |
1264 |
1265 |
1266 |
1267 |
1268 |
1269 |
1269 |
1270 |
1271 |
1272 |
1273 |
1274 |
1275 |
1276 |
1277 |
1278 |
1278 |
1279 |
1279 |
1280 |
1281 |
1282 |
1283 |
1284 |
1285 |
1286 |
1287 |
1288 |
1288 |
1289 |
1289 |
1290 |
1291 |
1292 |
1293 |
1294 |
1295 |
1296 |
1297 |
1297 |
1298 |
1298 |
1299 |
1299 |
1300 |
1301 |
1302 |
1303 |
1304 |
1305 |
1306 |
1307 |
1308 |
1309 |
1309 |
1310 |
1311 |
1312 |
1313 |
1314 |
1315 |
1316 |
1317 |
1318 |
1319 |
1319 |
1320 |
1321 |
1322 |
1323 |
1324 |
1325 |
1326 |
1327 |
1328 |
1329 |
1329 |
1330 |
1331 |
1332 |
1333 |
1334 |
1335 |
1336 |
1337 |
1338 |
1339 |
1339 |
1340 |
1341 |
1342 |
1343 |
1344 |
1345 |
1346 |
1347 |
1348 |
1349 |
1349 |
1350 |
1351 |
1352 |
1353 |
1354 |
1355 |
1356 |
1357 |
1358 |
1359 |
1359 |
1360 |
1361 |
1362 |
1363 |
1364 |
1365 |
1366 |
1367 |
1368 |
1369 |
1369 |
1370 |
1371 |
1372 |
1373 |
1374 |
1375 |
1376 |
1377 |
1378 |
1378 |
1379 |
1379 |
1380 |
1381 |
1382 |
1383 |
1384 |
1385 |
1386 |
1387 |
1388 |
1388 |
1389 |
1389 |
1390 |
1391 |
1392 |
1393 |
1394 |
1395 |
1396 |
1397 |
1398 |
1398 |
1399 |
1399 |
1400 |
1401 |
1402 |
1403 |
1404 |
1405 |
1406 |
1407 |
1408 |
1409 |
1409 |
1410 |
1411 |
1412 |
1413 |
1414 |
1415 |
1416 |
1417 |
1418 |
1419 |
1419 |
1420 |
1421 |
1422 |
1423 |
1424 |
1425 |
1426 |
1427 |
1428 |
1429 |
1429 |
1430 |
1431 |
1432 |
1433 |
1434 |
1435 |
1436 |
1437 |
1438 |
1439 |
1439 |
1440 |
1441 |
1442 |
1443 |
1444 |
1445 |
1446 |
1447 |
1448 |
1449 |
1449 |
1450 |
1451 |
1452 |
1453 |
1454 |
1455 |
1456 |
1457 |
1458 |
1459 |
1459 |
1460 |
1461 |
1462 |
1463 |
1464 |
1465 |
1466 |
1467 |
1468 |
1469 |
1469 |
1470 |
1471 |
1472 |
1473 |
1474 |
1475 |
1476 |
1477 |
1478 |
1479 |
1479 |
1480 |
1481 |
1482 |
1483 |
1484 |
1485 |
1486 |
1487 |
1488 |
1489 |
1489 |
1490 |
1491 |
1492 |
1493 |
1494 |
1495 |
1496 |
1497 |
1498 |
1498 |
1499 |
1499 |
1500 |
1501 |
1502 |
1503 |
1504 |
1505 |
1506 |
1507 |
1508 |
1509 |
1509 |
1510 |
1511 |
1512 |
1513 |
1514 |
1515 |
1516 |
1517 |
1518 |
1519 |
1519 |
1520 |
1521 |
1522 |
1523 |
1524 |
1525 |
1526 |
1527 |
1528 |
1529 |
1529 |
1530 |
1531 |
1532 |
1533 |
1534 |
1535 |
1536 |
1537 |
1538 |
1539 |
1539 |
1540 |
1541 |
1542 |
1543 |
1544 |
1545 |
1546 |
1547 |
1548 |
1549 |
1549 |
1550 |
1551 |
1552 |
1553 |
1554 |
1555 |
1556 |
1557 |
1558 |
1559 |
1559 |
1560 |
1561 |
1562 |
1563 |
1564 |
1565 |
1566 |
1567 |
1568 |
1569 |
1569 |
1570 |
1571 |
1572 |
1573 |
1574 |
1575 |
1576 |
1577 |
1578 |
1578 |
1579 |
1579 |
1580 |
1581 |
1582 |
1583 |
1584 |
1585 |
1586 |
1587 |
1588 |
1588 |
1589 |
1589 |
1590 |
1591 |
1592 |
1593 |
1594 |
1595 |
1596 |
1597 |
1598 |
1598 |
1599 |
1599 |
1600 |
1601 |
1602 |
1603 |
1604 |
1605 |
1606 |
1607 |
1608 |
1609 |
1609 |
1610 |
1611 |
1612 |
1613 |
1614 |
1615 |
1616 |
1617 |
1618 |
1619 |
1619 |
1620 |
1621 |
1622 |
1623 |
1624 |
1625 |
1626 |
1627 |
1628 |
1629 |
1629 |
1630 |
1631 |
1632 |
1633 |
1634 |
1635 |
1636 |
1637 |
1638 |
1639 |
1639 |
1640 |
1641 |
1642 |
1643 |
1644 |
1645 |
1646 |
1647 |
1648 |
1649 |
1649 |
1650 |
1651 |
1652 |
1653 |
1654 |
1655 |
1656 |
1657 |
1658 |
1659 |
1659 |
1660 |
1661 |
1662 |
1663 |
1664 |
1665 |
1666 |
1667 |
1668 |
1669 |
1669 |
1670 |
1671 |
1672 |
1673 |
1674 |
1675 |
1676 |
1677 |
1678 |
1678 |
1679 |
1679 |
1680 |
1681 |
1682 |
1683 |
1684 |
1685 |
1686 |
1687 |
1688 |
1688 |
1689 |
1689 |
1690 |
1691 |
1692 |
1693 |
1694 |
1695 |
1696 |
1697 |
1698 |
1698 |
1699 |
1699 |
1700 |
1701 |
1702 |
1703 |
1704 |
1705 |
1706 |
1707 |
1708 |
1709 |
1709 |
1710 |
1711 |
1712 |
1713 |
1714 |
1715 |
1716 |
1717 |
1718 |
1719 |
1719 |
1720 |
1721 |
1722 |
1723 |
1724 |
1725 |
1726 |
1727 |
1728 |
1729 |
1729 |
1730 |
1731 |
1732 |
1733 |
1734 |
1735 |
1736 |
1737 |
1738 |
1739 |
1739 |
1740 |
1741 |
1742 |
1743 |
1744 |
1745 |
1746 |
1747 |
1748 |
1749 |
1749 |
1750 |
1751 |
1752 |
1753 |
1754 |
1755 |
1756 |
1757 |
1758 |
1759 |
1759 |
1760 |
1761 |
1762 |
1763 |
1764 |
1765 |
1766 |
1767 |
1768 |
1769 |
1769 |
1770 |
1771 |
1772 |
1773 |
1774 |
1775 |
1776 |
1777 |
1778 |
1778 |
1779 |
1779 |
1780 |
1781 |
1782 |
1783 |
1784 |
1785 |
1786 |
1787 |
1788 |
1788 |
1789 |
1789 |
1790 |
1791 |
1792 |
1793 |
1794 |
1795 |
1796 |
1797 |
1798 |
1798 |
1799 |
1799 |
1800 |
1801 |
1802 |
1803 |
1804 |
1805 |
1806 |
1807 |
1808 |
1809 |
1809 |
1810 |
1811 |
1812 |
1813 |
1814 |
1815 |
1816 |
1817 |
1818 |
1819 |
1819 |
1820 |
1821 |
1822 |
1823 |
1824 |
1825 |
1826 |
1827 |
1828 |
1829 |
1829 |
1830 |
1831 |
1832 |
1833 |
1834 |
1835 |
1836 |
1837 |
1838 |
1839 |
1839 |
1840 |
1841 |
1842 |
1843 |
1844 |
1845 |
1846 |
1847 |
1848 |
1849 |
1849 |
1850 |
1851 |
1852 |
1853 |
1854 |
1855 |
1856 |
1857 |
1858 |
1859 |
1859 |
1860 |
1861 |
1862 |
1863 |
1864 |
1865 |
1866 |
1867 |
1868 |
1869 |
1869 |
1870 |
1871 |
1872 |
1873 |
1874 |
1875 |
1876 |
1877 |
1878 |
1878 |
1879 |
1879 |
1880 |
1881 |
1882 |
1883 |
1884 |
1885 |
1886 |
1887 |
1888 |
1888 |
1889 |
1889 |
1890 |
1891 |
1892 |
1893 |
1894 |
1895 |
1896 |
1897 |
1898 |
1898 |
1899 |
1899 |
1900 |
1901 |
1902 |
1903 |
1904 |
1905 |
1906 |
1907 |
1908 |
1909 |
1909 |
1910 |
1911 |
1912 |
1913 |
1914 |
1915 |
1916 |
1917 |
1918 |
1919 |
1919 |
1920 |
1921 |
1922 |
1923 |
1924 |
1925 |
1926 |
1927 |
1928 |
1929 |
1929 |
1930 |
1931 |
1932 |
1933 |
1934 |
1935 |
1936 |
1937 |
1938 |
1939 |
1939 |
1940 |
1941 |
1942 |
1943 |
1944 |
1945 |
1946 |
1947 |
1948 |
1949 |
1949 |
1950 |
1951 |
1952 |
1953 |
1954 |
1955 |
1956 |
1957 |
1958 |
1959 |
1959 |
1960 |
1961 |
1962 |
1963 |
1964 |
1965 |
1966 |
1967 |
1968 |
1969 |
1969 |
1970 |
1971 |
1972 |
1973 |
1974 |
1975 |
1976 |
1977 |
1978 |
1978 |
1979 |
1979 |
1980 |
1981 |
1982 |
1983 |
1984 |
1985 |
1986 |
1987 |
1988 |
1988 |
1989 |
1989 |
1990 |
1991 |
1992 |
1993 |
1994 |
1995 |
1996 |
1997 |
1998 |
1998 |
1999 |
1999 |
2000 |
2001 |
2002 |
2003 |
2004 |
2005 |
2006 |
2007 |
2008 |
2009 |
2009 |
2010 |
2011 |
2012 |
2013 |
2014 |
2015 |
2016 |
2017 |
2018 |
2019 |
2019 |
2020 |
2021 |
2022 |
2023 |
2024 |
2025 |
2026 |
2027 |
2028 |
2029 |
2029 |
2030 |
2031 |
2032 |
2033 |
2034 |
2035 |
2036 |
2037 |
2038 |
2039 |
2039 |
2040 |
2041 |
2042 |
2043 |
2044 |
2045 |
2046 |
2047 |
2048 |
2049 |
2049 |
2050 |
2051 |
2052 |
2053 |
2054 |
2055 |
2056 |
2057 |
2058 |
2059 |
2059 |
2060 |
2061 |
2062 |
2063 |
2064 |
2065 |
2066 |
2067 |
2068 |
2069 |
2069 |
2070 |
2071 |
2072 |
2073 |
2074 |
2075 |
2076 |
2077 |
2078 |
2078 |
2079 |
2079 |
2080 |
2081 |
2082 |
2083 |
2084 |
2085 |
2086 |
2087 |
2088 |
2088 |
2089 |
2089 |
2090 |
2091 |
2092 |
2093 |
2094 |
2095 |
2096 |
2097 |
2098 |
2098 |
2099 |
2099 |
2100 |
2101 |
2102 |
2103 |
2104 |
2105 |
2106 |
2107 |
2108 |
2109 |
2109 |
2110 |
2111 |
2112 |
2113 |
2114 |
2115 |
2116 |
2117 |
2118 |
2119 |
2119 |
2120 |
2121 |
2122 |
2123 |
2124 |
2125 |
2126 |
2127 |
2128 |
2129 |
2129 |
2130 |
2131 |
2132 |
2133 |
2134 |
2135 |
2136 |
2137 |
2138 |
2139 |
2139 |
2140 |
2141 |
2142 |
2143 |
2144 |
2145 |
2146 |
2147 |
2148 |
2149 |
2149 |
2150 |
2151 |
2152 |
2153 |
2154 |
2155 |
2156 |
2157 |
2158 |
2159 |
2159 |
2160 |
2161 |
2162 |
2163 |

```