

UNIVERSIDAD DE SAN BUENAVENTURA CALI
FACULTAD DE INGENIERIAS - PROGRAMA DE ING. SISTEMAS

TECNICAS AVANZADAS DE PROGRAMACION

FizzBuzz Web



**UNIVERSIDAD DE
SAN BUENAVENTURA
CALI**

Presentado por: Sebastian Lopez Montenegro
Código estudiantil: 30000097500
Correo institucional: Slopezm4@correo.usbcali.edu.co
Fecha: 27/04/2024

Problema: En este ejercicio se emplearán una serie de operaciones sobre un FizzBuzz Web, empleando Flask como servidor liviano y Postman como herramienta de pruebas.

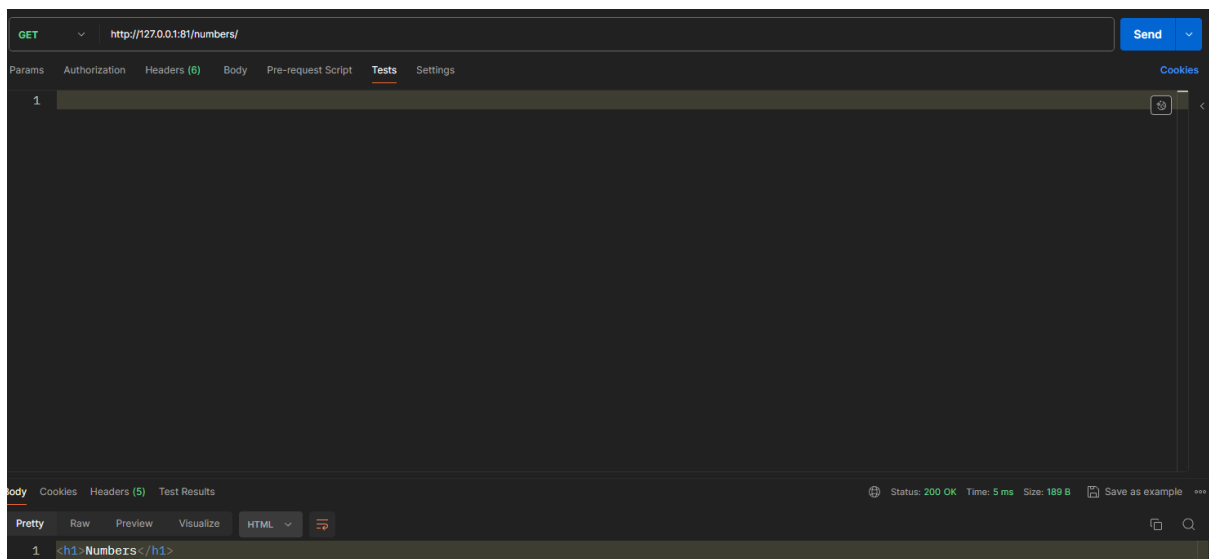
Restricciones: La actividad debe ser resuelta mediante la metodología TDD, especificando cada etapa de esta metodología e implementando el Principio de Inversión de Dependencias para que a futuro pueda ser un código más flexible.

Estrategias: Para la elaboración de esta actividad se necesitará la herramienta Postman como parte de la estructura de pruebas, también se necesitará la librería Flask y SQLite para trabajar directamente con una base de datos local y el manejo de la API, estas se trabajarán desde un entorno virtual por comodidad y no tener conflictos con versiones.

Fase RED

```
main.py > numbers
1  import sqlite3
2  from flask import Flask, request
3
4  app = Flask(__name__)
5
6  @app.route('/numbers/', methods=['GET'])
7  def numbers():
8      return "<h1>Numbers</h1>", 200
9
10 app.run(host='0.0.0.0', port=81)
```

Se iniciará creando el archivo main.py donde se trabajará el funcionamiento de la API gracias a Flask que nos permitirá manejar las rutas que podremos visualizar desde la herramienta Postman como verá a continuación.



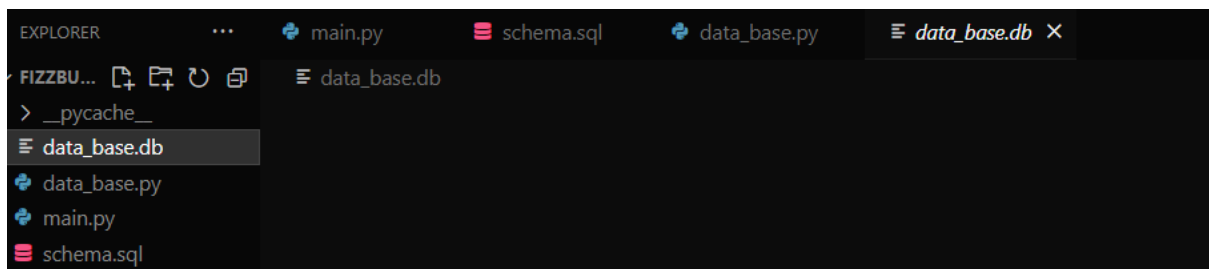
Ya se comprueba que la URL de nuestra API está funcionando correctamente, por lo que ya se podrá continuar al siguiente paso.

Para trabajar con la API, se tomarán 100 números que estarán almacenados en una base de datos local, la cual se necesitará crear mediante SQLite.

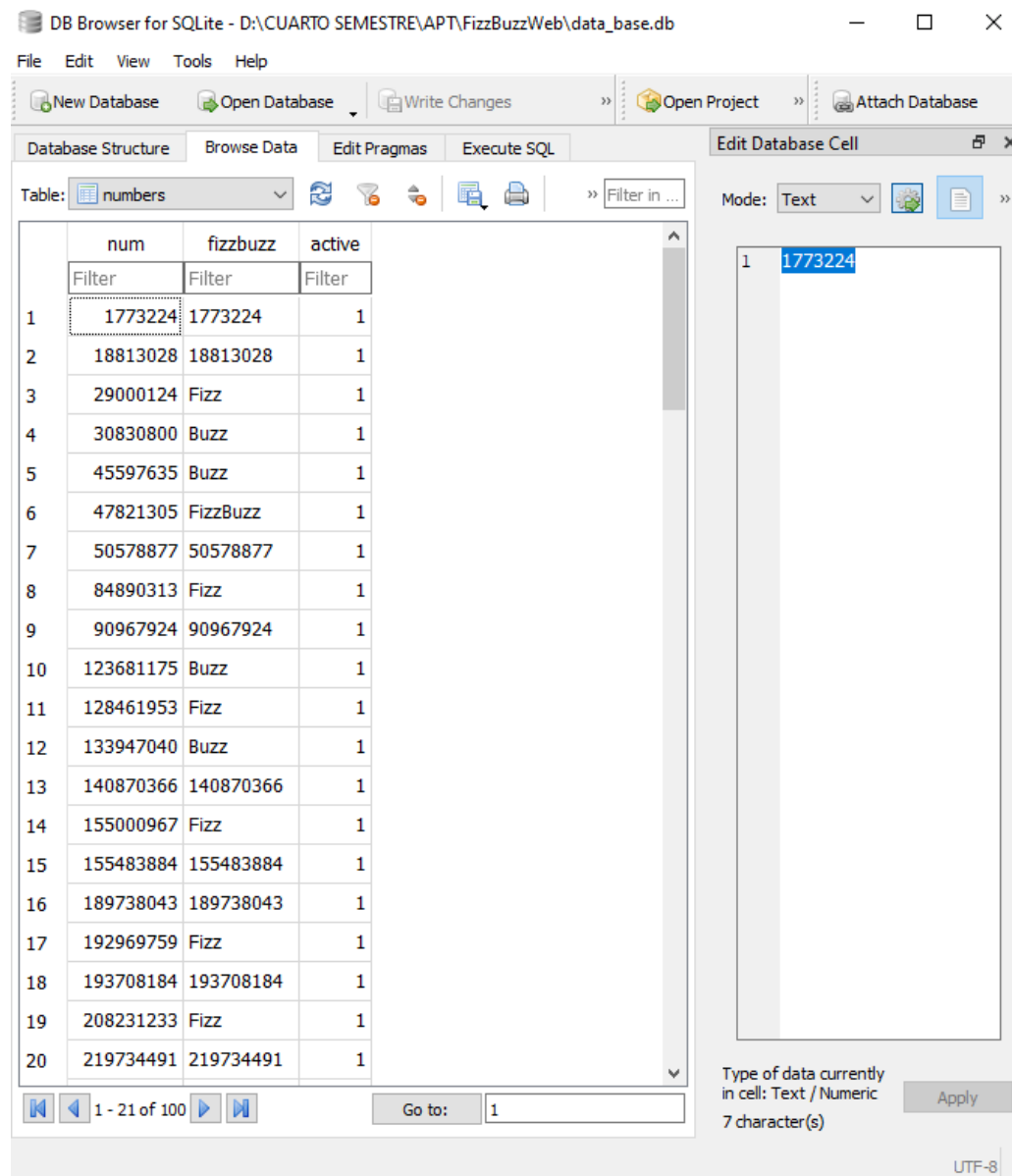
```
main.py schema.sql X
schema.sql
1 CREATE TABLE IF NOT EXISTS numbers (
2     num INTEGER PRIMARY KEY,
3     fizzbuzz TEXT NOT NULL,
4     active INTEGER NOT NULL DEFAULT 1
5 );
6
7 INSERT INTO numbers (num, fizzbuzz) VALUES
8     ('551669674', '551669674'),
9     ('422356043', '422356043'),
10    ('360061364', '360061364'),
11    ('901352955', 'FizzBuzz'),
12    ('858815785', 'Buzz'),
13    ('308519689', '308519689'),
14    ('696024202', '696024202'),
15    ('1176711096', 'Fizz'),
16    ('589958805', 'FizzBuzz'),
17    ('464905687', '464905687'),
18    ('604363876', '604363876'),
19    ('825195979', '825195979'),
20    ('1088834390', 'Buzz'),
21    ('50578877', '50578877'),
22    ('805985933', '805985933'),
23    ('682738574', '682738574'),
24    ('300672140', 'Buzz'),
25    ('90967924', '90967924'),
26    ('660949840', 'Buzz'),
27    ('879085864', '879085864'),
28    ('192969759', 'Fizz'),
29    ('926527358', '926527358'),
30    ('140870366', '140870366'),
31    ('839605003', '839605003'),
32    ('440216775', 'FizzBuzz'),
33    ('873855426', 'Fizz'),
34    ('375853494', 'Fizz'),
35    ('189738043', '189738043'),
36    ('695786561', '695786561'),
37    ('488419594', '488419594'),
38    ('868606882', '868606882'),
39    ('133947040', 'Buzz'),
40    ('921986667', 'Fizz'),

main.py schema.sql data_base.py X
data_base.py > ...
1 import sqlite3
2
3 DATA_BASE_NAME = "data_base.db"
4 SCHEMA_FILE = "schema.sql"
5
6 #Global connection and cursor
7 _connection = sqlite3.connect(DATA_BASE_NAME)
8
9 def execute_schema():
10     with open(SCHEMA_FILE) as file:
11         _connection.executescript(file.read())
12
13 execute_schema()
14 _connection.close()
```

Una vez creado el schema que define la estructura de la tabla y los 100 números a ingresar en ella acompañados de su respectivo FizzBuzz y un campo active que nos servirá como borrado lógico más adelante. Este schema se deberá ejecutar desde el archivo data_base.py para que cree la base de datos con respecto al schema anterior.

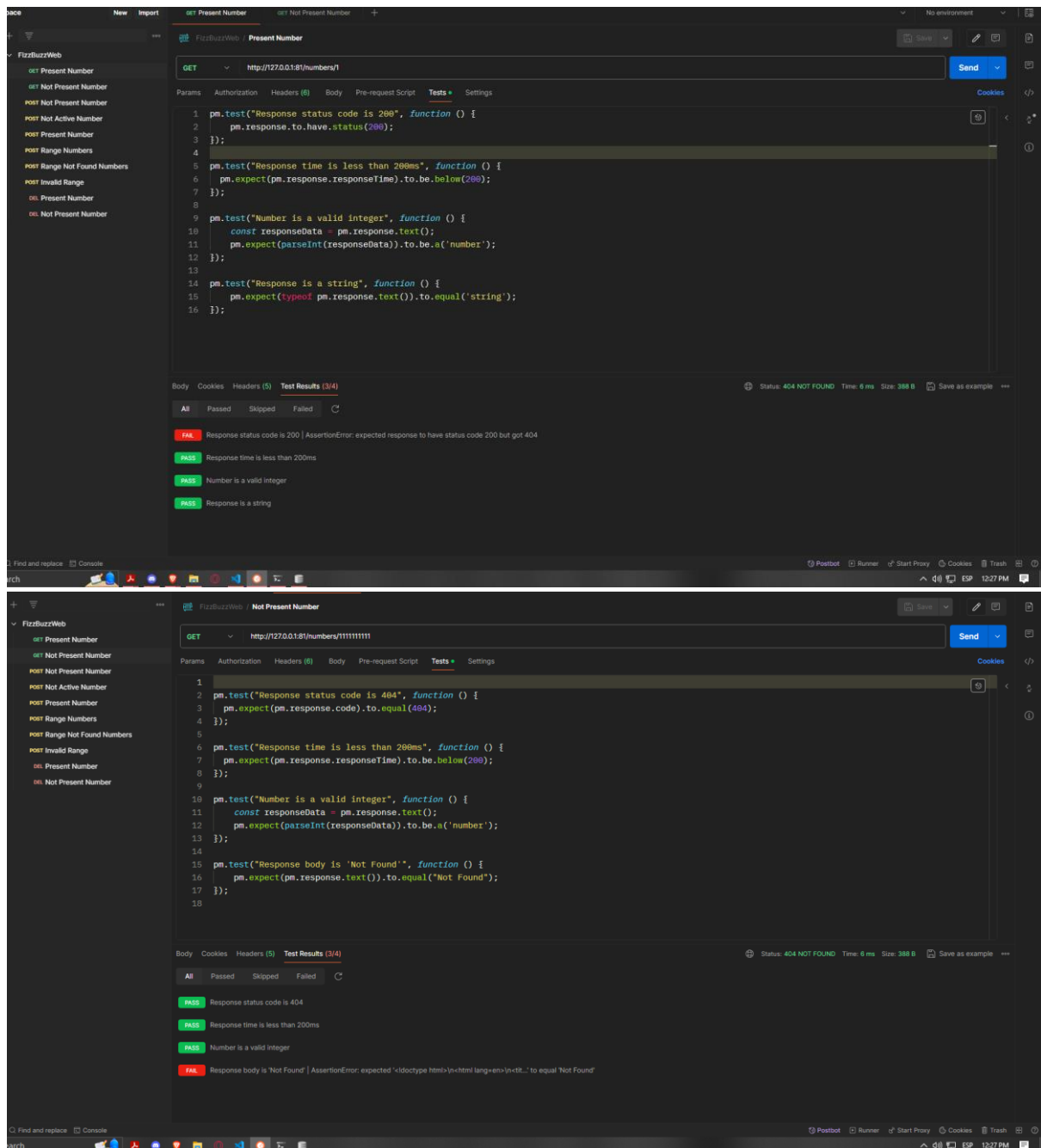


Se ejecutó el archivo da_base.py y se creó la base de datos correctamente, sin embargo, desde Visual Studio Code no se puede visualizar, por lo que se usará la aplicación DB Browser.



Pruebas

Ahora, dentro del Postman se crearán en el apartado de Tests, las pruebas que verificarán el comportamiento de la API.



Postman interface showing a REST client request for the endpoint `http://127.0.0.1:81/numbers/222222222` using the POST method. The request is part of a collection named "FizzBuzzWeb" under the environment "No environment".

The request body contains the following JSON:

```
1
2
3
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
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
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
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

The response status is 404 NOT FOUND. The response time is 10 ms, and the size is 388 B. The response body is empty.

The test results show the following:

- FAIL: Response status code is 201 | AssertionError: expected response to have status code 201 but got 404
- PASS: Response time is less than 200ms
- PASS: Number is a valid integer
- PASS: Response is a string

Postman interface showing a REST client request for the endpoint `http://127.0.0.1:81/numbers/1773224` using the POST method. The request is part of a collection named "FizzBuzzWeb" under the environment "No environment".

The request body contains the following JSON:

```
1
2
3
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
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
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
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

The response status is 404 NOT FOUND. The response time is 6 ms, and the size is 388 B. The response body is empty.

The test results show the following:

- FAIL: Response status code is 200 | AssertionError: expected 404 to equal 200
- PASS: Response time is less than 200ms
- PASS: Number is a valid integer
- PASS: Response is a string

FlizbuzzWeb

- GET Present Number
- GET Not Present Number
- POST Not Present Number
- POST Not Active Number
- POST Present Number
- POST Range Numbers
- POST Range Not Found Numbers
- POST Invalid Range
- ON Present Number
- ON Not Present Number

POST http://127.0.0.1:81/numbers/1234

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

```
1
2 pm.test("Response status code is 409", function () {
3   pm.expect(pm.response.code).to.equal(409);
4 });
5
6
7 pm.test("Response time is less than 200ms", function () {
8   pm.expect(pm.response.responseTime).to.be.below(200);
9 });
10
11 pm.test("Number is a valid integer", function () {
12   const responseData = pm.response.text();
13   pm.expect(parseInt(responseData)).to.be.a('number');
14 });
15
16 pm.test("Response is a string", function () {
17   pm.expect(typeof pm.response.text()).to.equal('string');
18 });
19
```

Body Cookies Headers (5) Test Results (3/4) Status: 404 NOT FOUND Time: 9 ms Size: 368 B Save as example

All Passed Skipped Failed

- FAIL Response status code is 409 | AssertionError: expected 404 to equal 409
- PASS Response time is less than 200ms
- PASS Number is a valid integer
- PASS Response is a string

Find and replace Console

ch

FlizbuzzWeb

- GET Present Number
- GET Not Present Number
- POST Not Present Number
- POST Not Active Number
- POST Present Number
- POST Range Numbers
- POST Range Not Found Numbers
- POST Invalid Range
- ON Present Number
- ON Not Present Number

POST http://127.0.0.1:81/range/

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

```
1
2 pm.test("Response status code is 200", function () {
3   pm.expect(pm.response.code).to.equal(200);
4 });
5
6
7 pm.test("Response time is less than 200ms", function () {
8   pm.expect(pm.response.responseTime).to.be.below(200);
9 });
10
11
12
```

Body Cookies Headers (5) Test Results (1/2) Status: 404 NOT FOUND Time: 7 ms Size: 368 B Save as example

All Passed Skipped Failed

- FAIL Response status code is 200 | AssertionError: expected 404 to equal 200
- PASS Response time is less than 200ms

Find and replace Console

ch

Postman interface showing a REST client request for the endpoint `http://127.0.0.1:81/range/` with a POST method. The request body contains the following JavaScript code:

```
1
2 pm.test("Response status code is 404", function () {
3   pm.expect(pm.response.code).to.equal(404);
4 });
5
6
7 pm.test("Response time is less than 200ms", function () {
8   pm.expect(pm.response.responseTime).to.be.below(200);
9 });
10
11 pm.test("Response body is 'Not Found'", function () {
12   pm.expect(pm.response.text()).to.equal("Not Found");
13 });
14
15
```

The test results show the following status:

- Response status code is 404: **PASS**
- Response time is less than 200ms: **PASS**
- Response body is 'Not Found': **FAIL** | AssertionError: expected '<doctype html>\n<html lang=en>\n<tit...' to equal 'Not Found'

The overall status is **404 NOT FOUND**. Time: 5 ms. Size: 388 B.

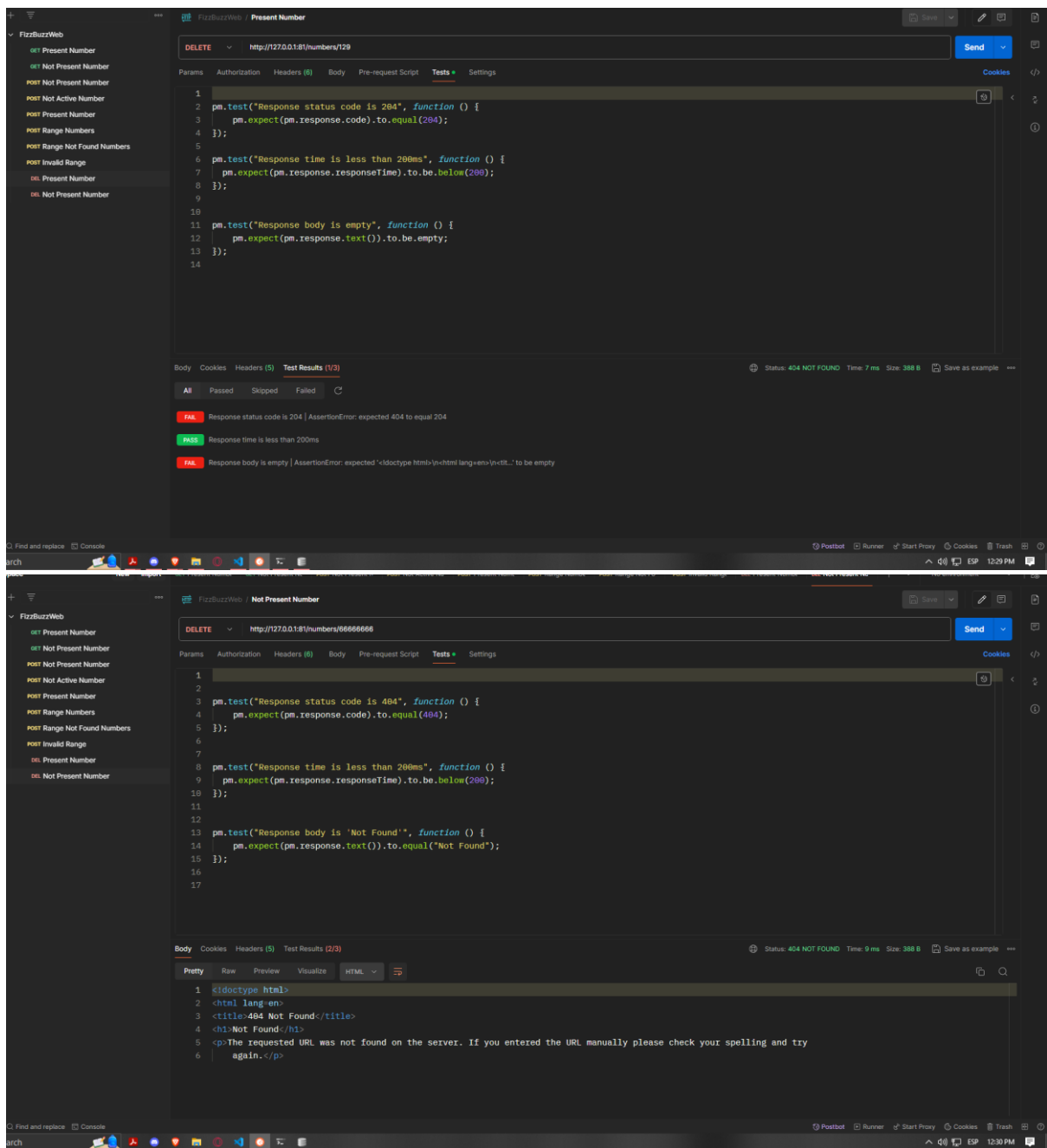
Postman interface showing a REST client request for the endpoint `http://127.0.0.1:81/range/` with a POST method. The request body contains the following JavaScript code:

```
1
2 pm.test("Response status code is 404", function () {
3   pm.expect(pm.response.code).to.equal(404);
4 });
5
6
7 pm.test("Response time is less than 200ms", function () {
8   pm.expect(pm.response.responseTime).to.be.below(200);
9 });
10
11 pm.test("Response body is 'Not Found'", function () {
12   pm.expect(pm.response.text()).to.equal("Not Found");
13 });
14
```

The test results show the following status:

- Response status code is 404: **PASS**
- Response time is less than 200ms: **PASS**
- Response body is 'Not Found': **FAIL** | AssertionError: expected '<doctype html>\n<html lang=en>\n<tit...' to equal 'Not Found'

The overall status is **404 NOT FOUND**. Time: 7 ms. Size: 388 B.



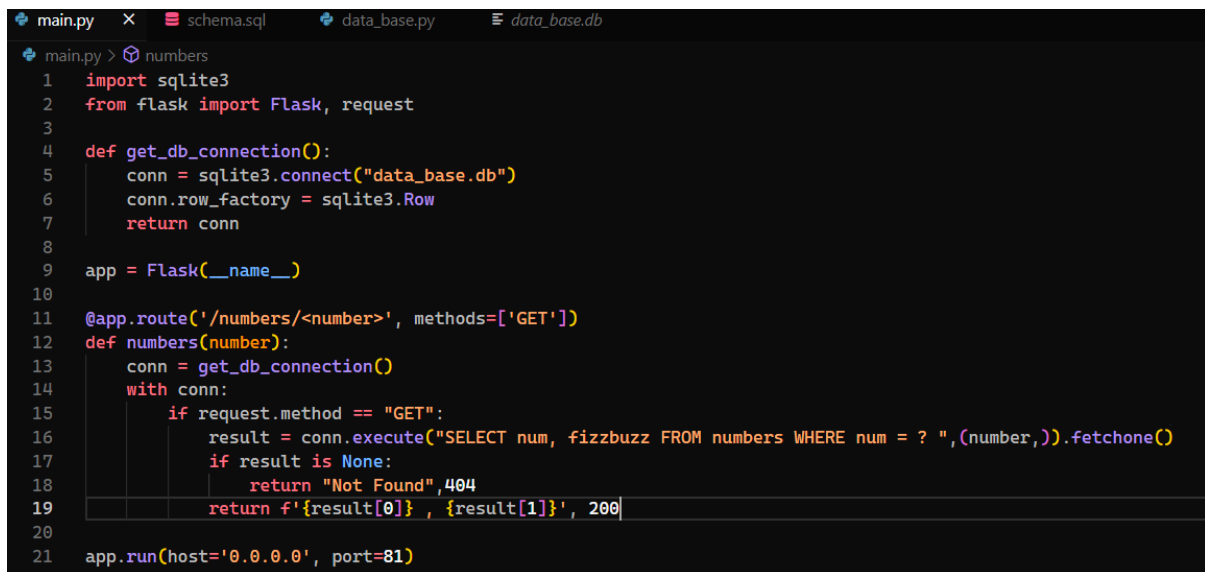
En nuestras pruebas se puede ver que en la mayoría pruebas que se integraron, se refieren al tiempo de respuesta, que código HTTP retorna, y que los valores que entran como parámetro en la URL y salen en el body, sean del tipo esperado. Además de que no se ha establecido la conexión entre Flask y la base de datos.

Fase GREEN

Se iniciará de manera ordenada siguiendo las restricciones del pdf de la actividad.

Peticiones GET para la URL “/numbers/<number>”:

- 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.
- 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.



```
main.py x schema.sql data_base.py data_base.db
main.py > numbers
1 import sqlite3
2 from flask import Flask, request
3
4 def get_db_connection():
5     conn = sqlite3.connect("data_base.db")
6     conn.row_factory = sqlite3.Row
7     return conn
8
9 app = Flask(__name__)
10
11 @app.route('/numbers/<number>', methods=['GET'])
12 def numbers(number):
13     conn = get_db_connection()
14     with conn:
15         if request.method == "GET":
16             result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num = ? ",(number,)).fetchone()
17             if result is None:
18                 return "Not Found",404
19             return f'{result[0]}, {result[1]}', 200
20
21 app.run(host='0.0.0.0', port=81)
```

Se creó la función `get_db_connection` para poder conectar la base de datos con la API.

Se modificó la ruta de `/numbers/` para que ahora se pueda trabajar directamente con los números de la base de datos, por ello se necesitaba la función anterior.

Se definió que para la ruta `/numbers/<numbers>` se puede usar un método GET, y directamente se empieza a trabajar con las restricciones anteriores.

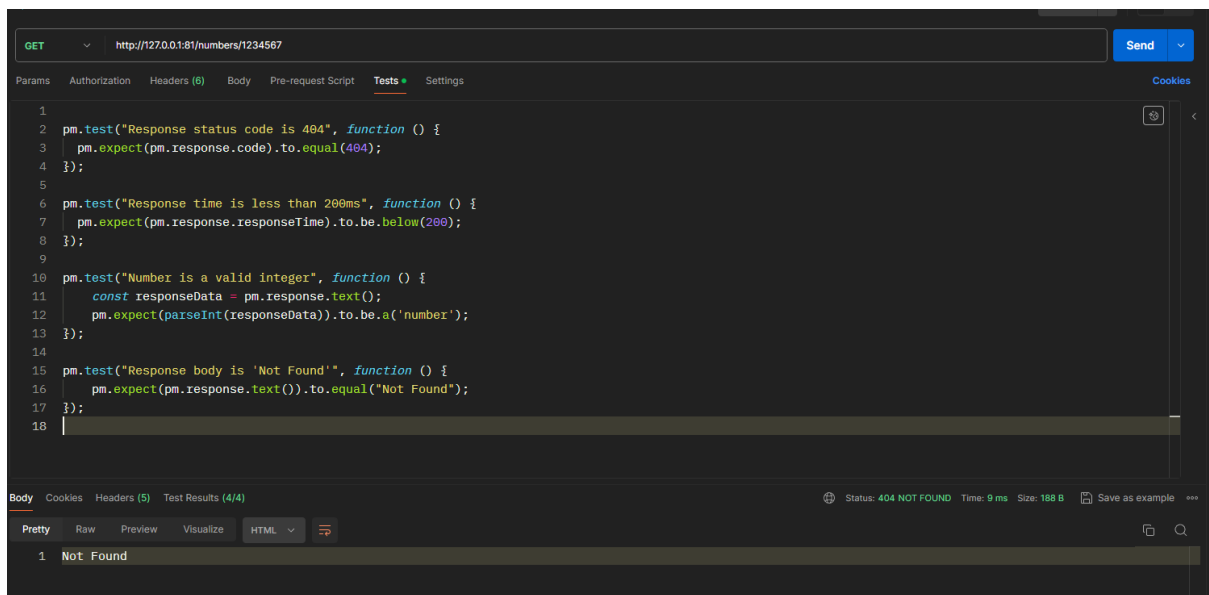
En la línea 16 se envía una sentencia SQL para que busque en la base datos el número que se envía por la URL, si no se encuentra ninguno se retorna un string “Not Found” y también el código HTTP correspondiente. Por otro lado, si se encuentra el numero en la base de datos, se retorna el valor del número y su respectivo fizzbuzz junto con el código HTTP. Cabe aclarar que la variable `result` es un diccionario, por lo que para acceder a sus columnas se llama a la posición de sus columnas, siendo `[0]` la columna `num`, y `[1]` la columna `fizzbuzz`, ya que así se definió en la función `get_db_connection`.

Primer caso GET



Se envió como parámetro un numero de la base de datos, y efectivamente, devolvió su valor de fizzbuzz. Se puede ver también que en la parte de Test Results todas las pruebas fueron aprobadas.

Segundo caso GET



Se envió como parámetro en la URL un número que no existe en la base de datos, y este devolvió correctamente el mensaje de "Not Found" y pasó todas las pruebas en Test Results.

Peticiones POST para la URL “/numbers/<number>”:

- 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.
- 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.

```
main.py > compute_fizzbuzz
4 def get_db_connection():
5     conn = sqlite3.connect('data_base.db')
6     conn.row_factory = sqlite3.Row
7     return conn
8
9 def compute_fizzbuzz(n):
10     output = "Fizz" * (int(n) % 3 == 0) + "Buzz" * (int(n) % 5 == 0)
11     return output or str(n)
12
13 app = Flask(__name__)
14
15 @app.route('/numbers/<number>', methods=['GET'])
16 def numbers(number):
17     conn = get_db_connection()
18     with conn:
19         if request.method == "GET":
20             result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num = ?", (number,)).fetchone()
21             if result is None:
22                 return "Not Found", 404
23             return f'{result[0]}, {result[1]}', 200
24         elif request.method == "POST":
25             result_active = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=? AND active='1'", (number,)).fetchone()
26             result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=?", (number,)).fetchone()
27             if result_active is None:
28                 if result is None:
29                     conn.execute("INSERT INTO numbers (num, fizzbuzz, active) VALUES (?, ?, 1)", (number, compute_fizzbuzz(number)))
30                     conn.commit()
31                     return f'{number}, {compute_fizzbuzz(number)}', 201
32                 else:
33                     conn.execute("UPDATE numbers SET active = '1' WHERE num = ?", (number,))
34                     conn.commit()
35                     return f'{result[0]}, {result[1]}', 200
36             else:
37                 return f'{result[0]}, {result[1]}', 409
38
39 app.run(host='0.0.0.0', port=81)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\CUARTO SEMESTRE\APT\FizzBuzzWeb> & D:/env/Scripts/python.exe "d:/CUARTO SEMESTRE\APT\FizzBuzzWeb\data_base.PY"

PS D:\CUARTO SEMESTRE\APT\FizzBuzzWeb>

Se crearon las posibles situaciones que pueden pasar dentro del método POST (línea 24), para el POST se requiere tener presente que un número que quiera ser subido a la base de datos puede no estar en la base de datos, puede estar desactivado o incluso puede ya estar en la base de datos.

Entonces se toman las condiciones:

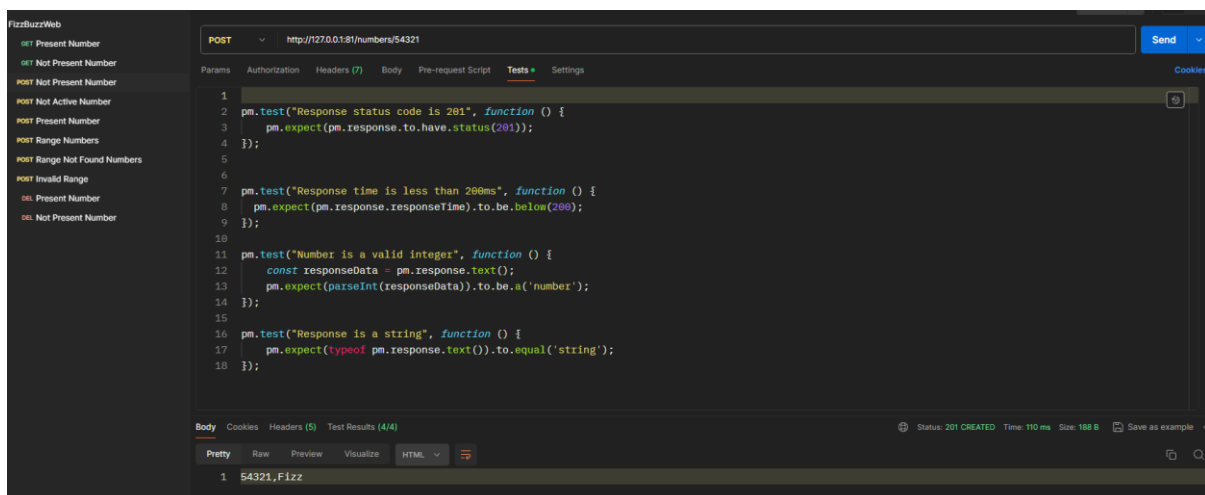
- Si un número no está activo y ni siquiera creado, se crea en la base de datos mediante una sentencia SQL INSERT y se calcula el valor de fizzbuzz en el momento de crearlo. Este devuelve el código HTTP 201(CREATED).
- Si un número no está activo, pero si está creado, se va a hacer un UPDATE del registro en la base de datos para que cambie el valor de active de 0 a 1. y devuelve el código HTTP 200 (OK).
- Si un numero está activo, quiere decir que solo se hace la consulta para saber su fizzbuzz mediante un SELECT y devuelve el código HTTP 409 (CONFLICT).

Se creo también la función `compute_fizzbuzz` para calcular el fizzbuzz de un número que se esté ingresando a por primera vez a la base de datos (línea 9).

```
@app.route('/numbers/<number>', methods=['GET', 'POST'])
def numbers(number):
```

Había olvidado agregar en methods el método POST.

Primer caso POST



Se puso como parámetro un número que no existe en la base de datos, entonces se creó el numero en la base de datos junto con su fizzbuzz y retornando el código HTTP 201. También se puede ver que las pruebas se pasaron con éxito.

Segundo caso POST

Para el segundo caso se necesita un número que esté desactivado, por lo que se usó DB Browser para modificar el campo active de un número y poder realizar la prueba.

The image shows two screenshots. The top screenshot is from 'DB Browser for SQLite' showing a table named 'numbers' with columns 'num', 'fizzbuzz', and 'active'. The table contains 8 rows. The value '0' in the 'active' column for the row with 'num' 29000124 is highlighted. The bottom screenshot is from 'Postman' showing a POST request to 'http://127.0.0.1:81/numbers/29000124'. The request body is '29000124, Fizz'. The response status is 200 OK, and the response body is '29000124, Fizz'.

	num	fizzbuzz	active
1	54321	Fizz	1
2	1773224	1773224	1
3	18813028	18813028	1
4	29000124	Fizz	0
5	30830800	Buzz	1
6	45597635	Buzz	1
7	47821305	FizzBuzz	1
8	50578877	50578877	1

```
POST http://127.0.0.1:81/numbers/29000124
1
2 pm.test("Response status code is 200", function () {
3   pm.expect(pm.response.code).to.equal(200);
4 });
5
6
7 pm.test("Response time is less than 200ms", function () {
8   pm.expect(pm.response.responseTime).to.be.below(200);
9 });
10
11 pm.test("Number is a valid integer", function () {
12   const responseData = pm.response.text();
13   pm.expect(parseInt(responseData)).to.be.a('number');
14 });
15
16 pm.test("Response is a string", function () {
17   pm.expect(typeof pm.response.text()).to.equal('string');
18 });
19
```

Body: 29000124, Fizz

Se envió como parámetro un número que está en la base de datos, pero no está activo, por lo que se hace un update a la base de datos y para activarlo, se trae el valor de fizzbuzz y devuelve el código HTTP 200. Y paso exitosamente todas las pruebas.

Tercer caso POST

Database Structure			
Browse Data			
Edit Pragmas			
Execute SQL			
Edit Database Cell			
Table: numbers			
	num	fizzbuzz	active
	Filter	Filter	Filter
1	54321	Fizz	1
2	1773224	1773224	1
3	18813028	18813028	1
4	29000124	Fizz	1
5	30830800	Buzz	1

FizzBuzzWeb

POST http://127.0.0.1:81/numbers/30830800

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

```
1 pm.test("Response status code is 409", function () {
2   pm.expect(pm.response.code).to.equal(409);
3 });
4
5
6
7 pm.test("Response time is less than 200ms", function () {
8   pm.expect(pm.response.responseTime).to.be.below(200);
9 });
10
11 pm.test("Number is a valid integer", function () {
12   const responseData = pm.response.text();
13   pm.expect(parseInt(responseData)).to.be.a('number');
14 });
15
16 pm.test("Response is a string", function () {
17   pm.expect(typeof pm.response.text()).to.equal('string');
18 });
19
```

Body Cookies Headers (5) Test Results (4/4)

All Passed Skipped Failed

Response status code is 409

Response time is less than 200ms

Number is a valid integer

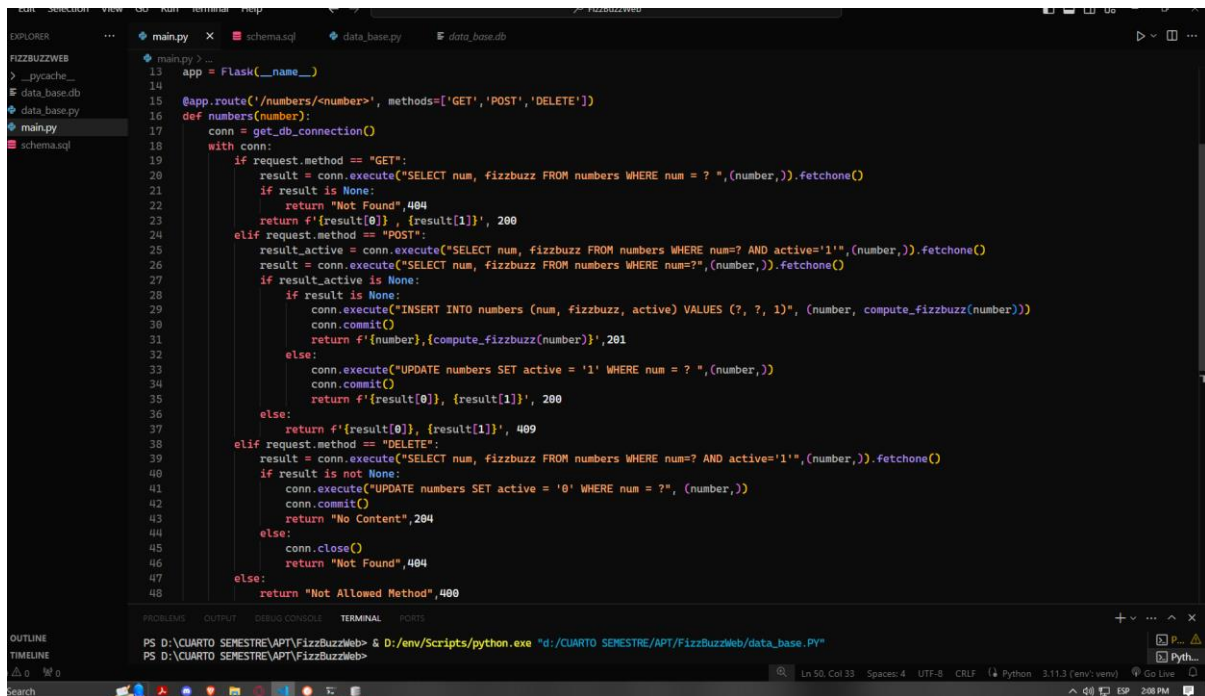
Response is a string

Status: 409 CONFLICT Time: 7ms Size: 103 B Save as example

Se envió como parámetro un numero de la base de datos que está activo (30830800), por ende, el programa solo trae su fizzbuzz y devuelve el código HTTP 409 (Conflict), aquí existe un conflicto porque se quiere crear un número que ya está creado, sin embargo, se muestra su fizzbuzz.

Peticiones DELETE para la URL “/numbers/<number>”:

- 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.
- 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.

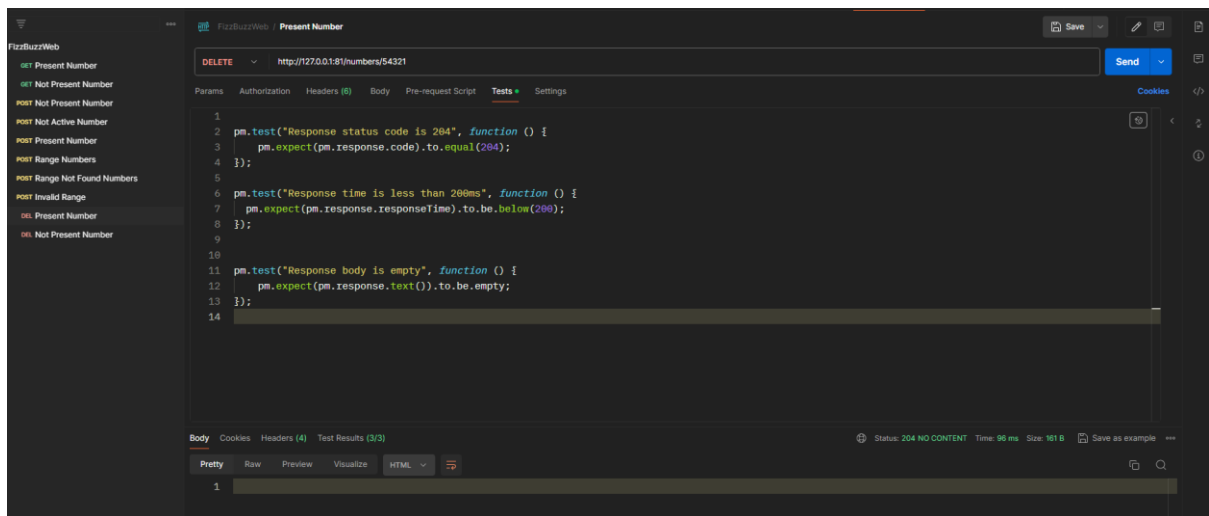


```
13 app = Flask(__name__)
14
15 @app.route('/numbers/<number>', methods=['GET','POST','DELETE'])
16 def numbers(number):
17     conn = get_db_connection()
18     with conn:
19         if request.method == "GET":
20             result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num = ?", (number,)).fetchone()
21             if result is None:
22                 return "Not Found", 404
23             return f'{result[0]}, {result[1]}', 200
24         elif request.method == "POST":
25             result_active = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=? AND active='1'", (number,)).fetchone()
26             result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=?", (number,)).fetchone()
27             if result_active is None:
28                 if result is None:
29                     conn.execute("INSERT INTO numbers (num, fizzbuzz, active) VALUES (?, ?, 1)", (number, compute_fizzbuzz(number)))
30                     conn.commit()
31                     return f'{number}, {compute_fizzbuzz(number)}', 201
32                 else:
33                     conn.execute("UPDATE numbers SET active = '1' WHERE num = ?", (number,))
34                     conn.commit()
35                     return f'{result[0]}, {result[1]}', 200
36             else:
37                 return f'{result[0]}, {result[1]}', 409
38         elif request.method == "DELETE":
39             result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=? AND active='1'", (number,)).fetchone()
40             if result is not None:
41                 conn.execute("UPDATE numbers SET active = '0' WHERE num = ?", (number,))
42                 conn.commit()
43                 return "No Content", 204
44             else:
45                 conn.close()
46                 return "Not Found", 404
47         else:
48             return "Not Allowed Method", 400
```

Se agregó a los métodos que maneja la ruta, el método DELETE.

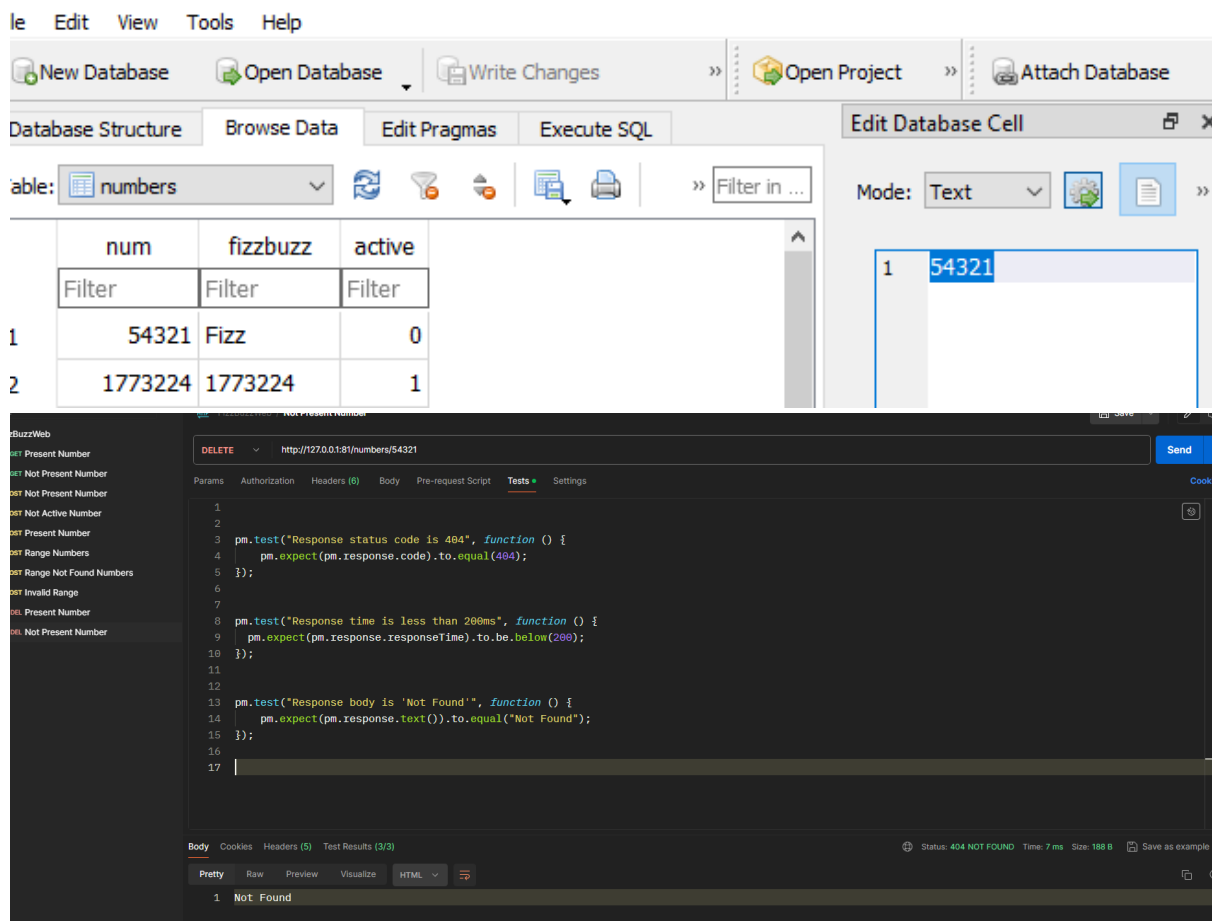
Para el método DELETE tendremos dos situaciones, una en la que se “borrará” un numero presente en la base de datos que está activo mediante un UPDATE para desactivarlo y devuelve un código HTTP 204(No Content), y en el otro caso, un número que no está presente o activo en la base de datos y que, al no encontrarlo, se devuelve el mensaje “Not Found” con el código HTTP 404.

Primer caso DELETE

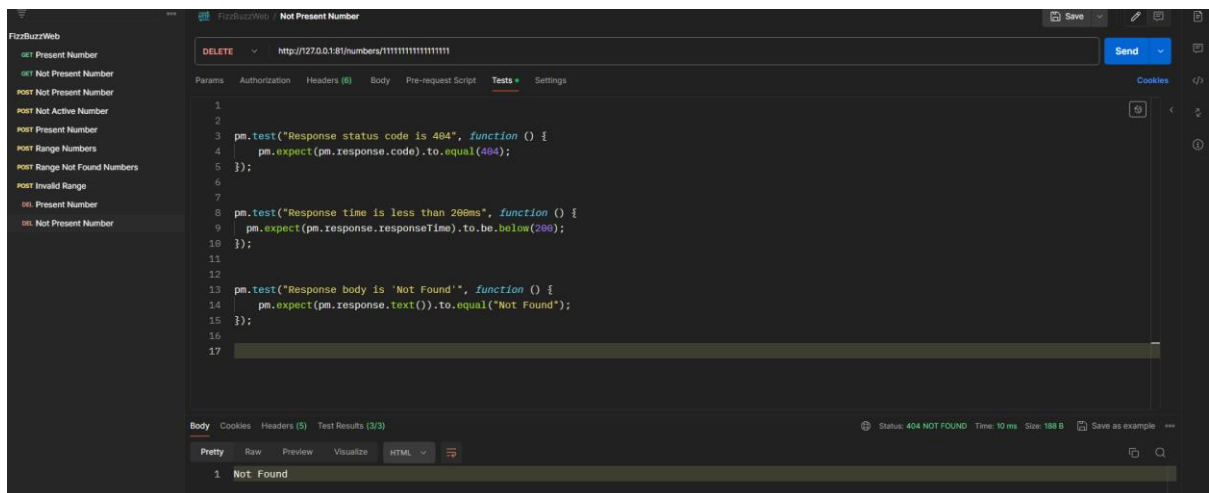


Se envió un numero presente en la base de datos y se le hizo un borrado lógico. Retornando entonces un HTTP 204 (No Content)

Segundo caso DELETE



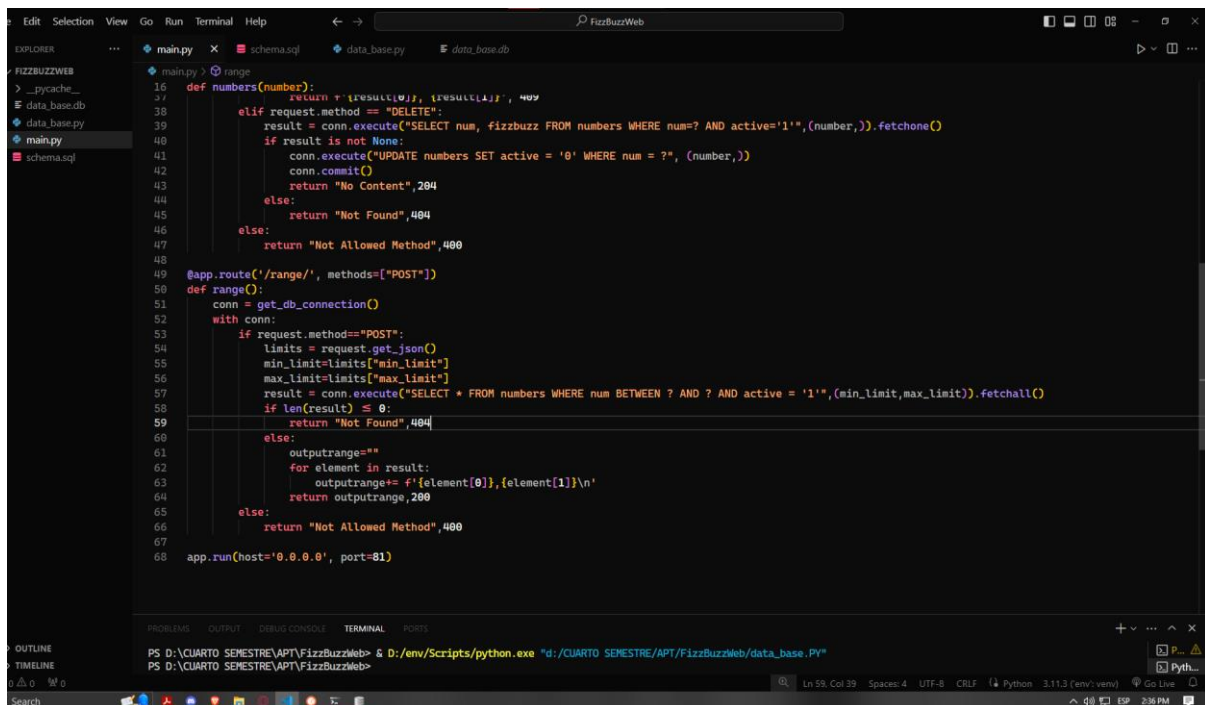
Como en el ejemplo anterior se borró ese número, ahora se intentó borrar de nuevo y devolvió "Not Found" y HTTP 404 porque ya fue borrado.



Se envió un número que no existe en la base de datos, y como no se encontró, el programa devolvió "Not Found" con el código HTTP 404.

Peticiones POST para la URL “/range/”:

- 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.



```
16 def numbers(number):
17     return f'{result[0]}, {result[1]}, 400'
18
19 elif request.method == "DELETE":
20     result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=? AND active='1'", (number,)).fetchone()
21     if result is not None:
22         conn.execute("UPDATE numbers SET active = '0' WHERE num = ?", (number,))
23         conn.commit()
24         return "No Content", 204
25     else:
26         return "Not Found", 404
27     else:
28         return "Not Allowed Method", 400
29
30 @app.route('/range/', methods=["POST"])
31 def range():
32     conn = get_db_connection()
33     with conn:
34         if request.method == "POST":
35             limits = request.get_json()
36             min_limit=limits["min_limit"]
37             max_limit=limits["max_limit"]
38             result = conn.execute("SELECT * FROM numbers WHERE num BETWEEN ? AND ? AND active = '1'", (min_limit,max_limit)).fetchall()
39             if len(result) <= 0:
40                 return "Not Found", 404
41             else:
42                 outputrange=""
43                 for element in result:
44                     outputrange+= f'{element[0]}, {element[1]}\n'
45                 return outputrange, 200
46             else:
47                 return "Not Allowed Method", 400
48
49 app.run(host='0.0.0.0', port=81)
```

Se creo una nueva ruta para la URL “/range/” con el método POST.

Se creo un objeto limits que guardará el archivo JSON que está en el body de POSTMAN y lo convierte en un diccionario.

Luego min_limit y max_limit toman los valores que están en el diccionario limits mediante su “key”, para definir el límite inferior y el límite superior del rango a buscar.

Luego en result se ejecutará una sentencia SQL SELECT que tomará todos los números de la base de datos que estén entre min_limit y max_limit, y que obviamente estén activos.

Luego encontraremos dos situaciones, la primera es que result no haya encontrado ningún número, por ende, devolverá “Not Found” con código HTTP 404. Por otro lado, que result si haya encontrado números, entonces se hará un string que contenga todos los números con sus respectivos fizzbuzz y finalmente retorna el string creado con el código HTTP 200.

Primer caso POST (range)

```
7 pm.test("Response time is less than 200ms", function () {
8   pm.expect(pm.response.responseTime).to.be.below(200);
9 });
10
11 pm.test("Body format", function () {
12   var formatoEsperado = /^\\d+\\s*,\\s*(Fizz|Buzz|FizzBuzz|\\d+)\\s*$/;
13   var cuerpoRespuesta = pm.response.text();
14   var lineas = cuerpoRespuesta.split('\\n');
15   lineas.forEach(function (linea) {
16     if(linea.trim() !== '') {
17       pm.expect(linea).to.match(formatoEsperado);
18     }
19   });
20 });
21
22
```

Para la siguiente prueba se agregó el siguiente test para verificar el formato con que llega la respuesta.

The screenshot shows the Postman interface. At the top, a POST request is configured to `http://127.0.0.1:81/range/`. The 'Body' tab is selected, and the request body is raw JSON: `{ "min_limit": 123681175, "max_limit": 735477762 }`. Below the request, the 'Test Results' section shows the response body in 'Pretty' format. The response is a list of 17 lines, each containing a number followed by 'Fizz', 'Buzz', or 'FizzBuzz' based on the FizzBuzz rules. The status bar at the bottom indicates a successful response with status 200 OK, time 8 ms, and size 1.08 KB.

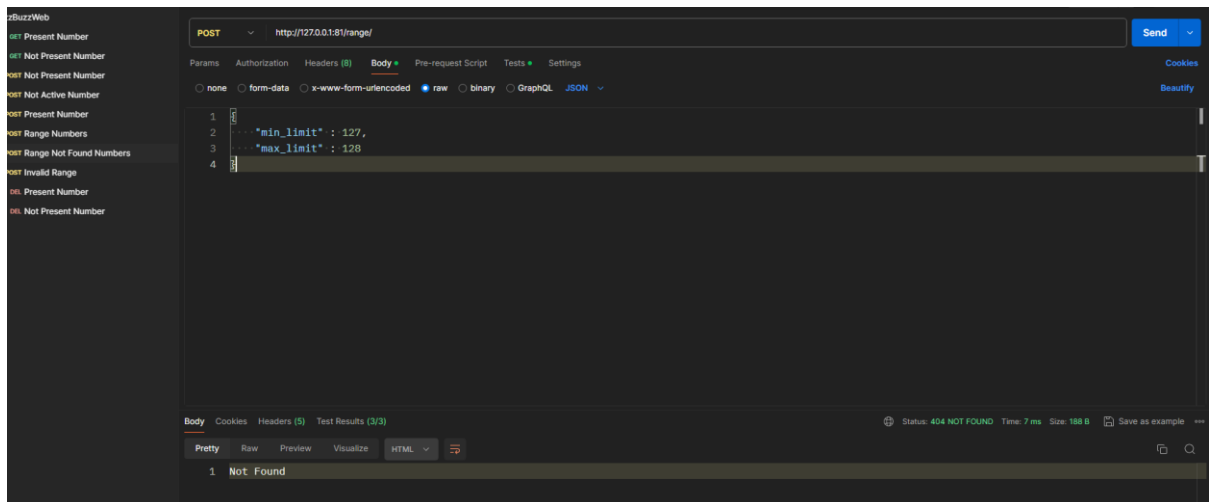
```
1 {
2   "min_limit": 123681175,
3   "max_limit": 735477762
4 }
```

Body Cookies Headers (5) Test Results (3/3) Status: 200 OK Time: 8 ms Size: 1.08 KB Save as example

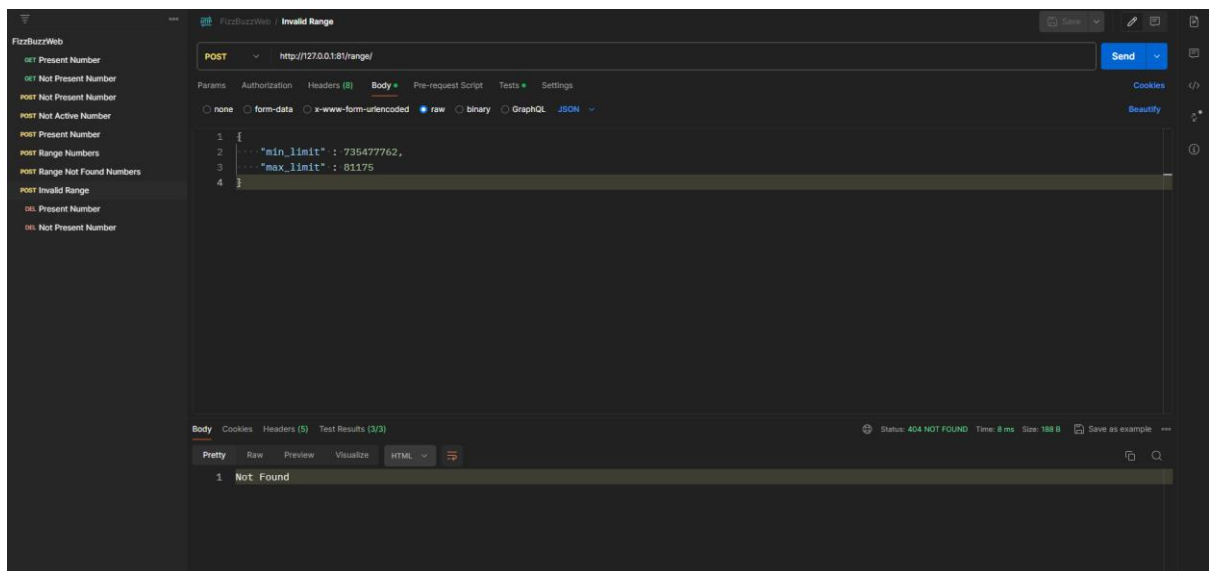
Pretty Raw Preview Visualize HTML

```
1 123681175,Buzz
2 128461953,Fizz
3 133947848,Buzz
4 148878366,148878366
5 155809967,Fizz
6 155483884,155483884
7 189738843,189738843
8 192969759,Fizz
9 193788184,193788184
10 288231233,Fizz
11 219734491,219734491
12 226848788,Buzz
13 226685737,226685737
14 266892455,Buzz
15 292815719,Fizz
16 388672148,Buzz
17 386528696,386528696
```

En esta prueba se le pasa como body al post un archivo JSON, con los valores numéricos de los límites del rango a buscar. Se puede evidenciar que efectivamente devolvió los números con el respectivo fizzbuzz y cumpliendo las pruebas de formato. Pasó todas las pruebas y retorno el código HTTP 200.



En esta prueba se definió un rango donde no hay números para verificar que retorna el mensaje "Not Found" con el código HTTP 404.

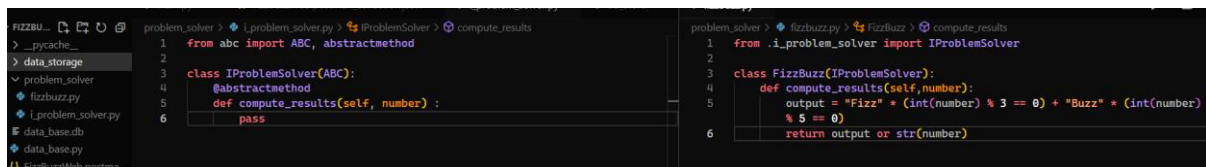


En esta prueba se crearon los límites de manera incorrecta, esperando entonces que el mensaje que retorne sea "Not Found" con el código HTTP 404.

Fase BLUE

En esta parte del reporte se mostrará el proceso de refactoring, adaptando el código al Principio de Inversión de Dependencias.

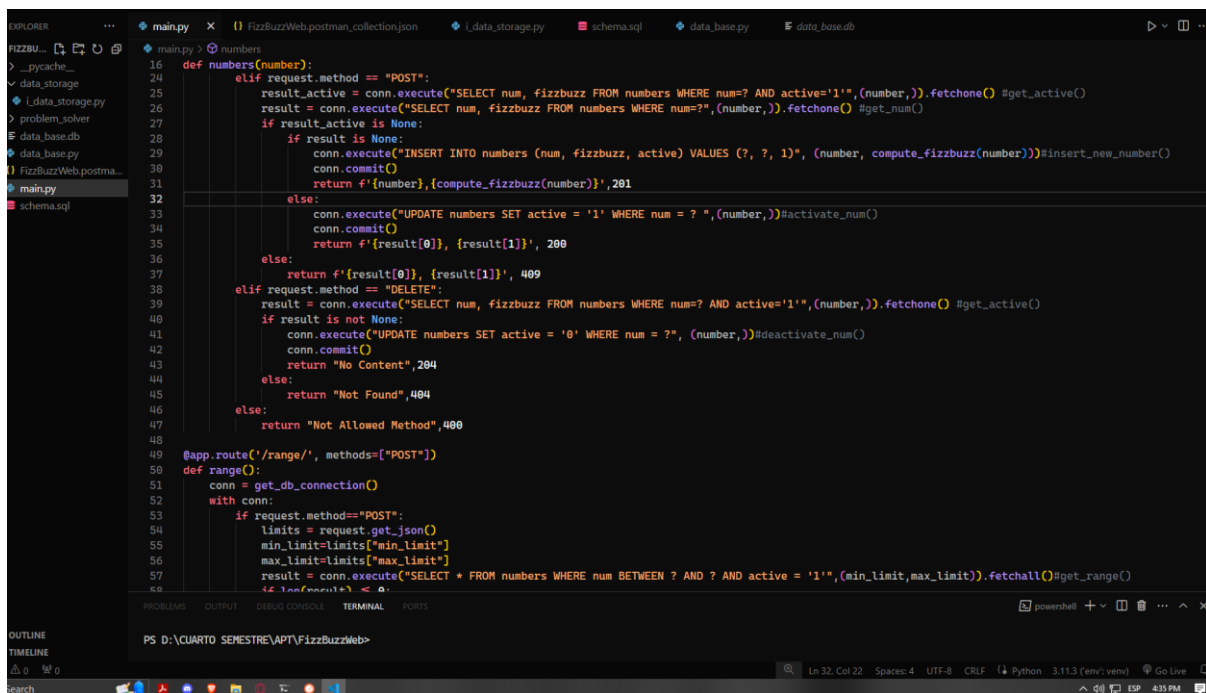
Primeramente, se creará la carpeta `problem_solver` y dentro de ella estará la interface `i_problem_solver` y el archivo `fizzbuzz.py` que se encargará de todo el proceso que tenga que ver con el fizzbuzz



```
1 from abc import ABC, abstractmethod
2
3 class IProblemSolver(ABC):
4     @abstractmethod
5     def compute_results(self, number):
6         pass
```

```
1 from .i_problem_solver import IProblemSolver
2
3 class FizzBuzz(IProblemSolver):
4     def compute_results(self, number):
5         output = "Fizz" * (int(number) % 3 == 0) + "Buzz" * (int(number)
6             % 5 == 0)
7         return output or str(number)
```

Ahora se creará la carpeta `data_storage` que tendrá la interface `i_data_storage` con los métodos para hacer sentencias SQL en la base de datos, por lo que tocará reconocer que métodos se pueden usar y que incluso, se repitan.



```
16 def numbers(number):
17     if request.method == "POST":
18         result_active = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=? AND active='1',(number,)).fetchone() #get_active()
19         result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=?", (number,)).fetchone() #get_num()
20         if result is None:
21             conn.execute("INSERT INTO numbers (num, fizzbuzz, active) VALUES (?, ?, 1)", (number, compute_fizzbuzz(number))) #insert_new_number()
22             conn.commit()
23             return f'{number},{compute_fizzbuzz(number)}', 201
24         else:
25             conn.execute("UPDATE numbers SET active = '1' WHERE num = ?", (number,)) #activate_num()
26             conn.commit()
27             return f'{result[0]},{result[1]}', 200
28         else:
29             return f'{result[0]},{result[1]}', 409
30     elif request.method == "DELETE":
31         result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=? AND active='1',(number,)).fetchone() #get_active()
32         if result is not None:
33             conn.execute("UPDATE numbers SET active = '0' WHERE num = ?", (number,)) #deactivate_num()
34             conn.commit()
35             return "No Content", 204
36         else:
37             return "Not Found", 404
38     else:
39         return "Not Allowed Method", 400
40
41 @app.route('/range/', methods=["POST"])
42 def range():
43     conn = get_db_connection()
44     with conn:
45         if request.method=="POST":
46             limits = request.get_json()
47             min_limit=limits["min_limit"]
48             max_limit=limits["max_limit"]
49             result = conn.execute("SELECT * FROM numbers WHERE num BETWEEN ? AND ? AND active = '1',(min_limit,max_limit)).fetchall() #get_range()
50             if len(result) < 1:
51                 return "No Content", 204
```

En la imagen anterior se comentó las líneas de código que se van a volver métodos de `i_data_storage`.

```

zzBuzzWeb.postman_collection.json  i_data_storage.py  ...
_storage > i_data_storage.py > IDataStorage > deactivate_num
from abc import ABC, abstractmethod

class IDataStorage(ABC):
    @abstractmethod
    def get_num(self, number):
        pass

    @abstractmethod
    def get_active(self, number):
        pass

    @abstractmethod
    def insert_new_number(self, number):
        pass

    @abstractmethod
    def activate_num(self, number):
        pass

    @abstractmethod
    def deactivate_num(self, number):
        pass

    @abstractmethod
    def get_range(self, number):
        pass

```

En la captura anterior se muestra como quedó definido la interface i_data_storage.

Ahora en el archivo db_storage se debe implementar la interface i_data_storage y tomar las funciones anteriormente identificadas y crearlas en este archivo db_storage.


```

db_storage.py x main.py
data_storage > db_storage.py > DBStorage > get_num
1  import sqlite3
2  from .i_data_storage import IDataStorage
3
4  class DBStorage(IDataStorage):
5      @staticmethod
6      def get_db_connection():
7          conn = sqlite3.connect("data_base.db")
8          conn.row_factory = sqlite3.Row
9          return conn
10
11     def get_num(self, number):
12         conn=self.get_db_connection()
13         result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num = ? ",(number,)).fetchone()
14         conn.close()
15         return result
16
17     def get_active(self, number):
18         conn=self.get_db_connection()
19         result = conn.execute("SELECT num, fizzbuzz FROM numbers WHERE num=? AND active='1',(number,)).fetchone()
20         conn.close()
21         return result
22
23     def insert_new_number(self, number):
24         conn=self.get_db_connection()
25         conn.execute("INSERT INTO numbers (num, fizzbuzz, active) VALUES (?, ?, 1)", (number[0],number[1]))
26         conn.commit()
27         conn.close()
28
29     def activate_num(self, number):
30         conn=self.get_db_connection
31         conn.execute("UPDATE numbers SET active = '1' WHERE num = ? ",(number,))
32         conn.commit()
33         conn.close()
34
35     def deactivate_num(self, number):
36         conn=self.get_db_connection
37         conn.execute("UPDATE numbers SET active = '0' WHERE num = ?", (number,))
38         conn.commit()
39         conn.close()
40
41     def get_range(self, min_limit,max_limit):
42         conn=self.get_db_connection
43         result = conn.execute("SELECT * FROM numbers WHERE num BETWEEN ? AND ? AND active = '1',(min_limit,max_limit)).fetchall()
44         conn.close()
45         return result
46

```

Luego, en el archivo my_app.py se debe organizar para que los métodos funcionen como un solo bloque cada uno, teniendo en cuenta que el único diferente es el POST de un rango. Es decir, trabajar todo lo relacionado con GET, POST y DELETE cada una en un método independiente dentro de my_app.py.

```

my_app.py > MyApp
from data_storage.db_storage import DBStorage
from data_storage.i_data_storage import IDataStorage
from problem_solver.fizzbuzz import FizzBuzz
from problem_solver.i_problem_solver import IProblemSolver

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

    def get(self,number):
        result = self.storage.get_num(number)
        if result is None:
            return "Not Found",404
        return f'{result[0]} , {result[1]}', 200

    def post(self,number):
        result_active = self.storage.activate_num(number)
        result = self.storage.get_num(number)
        if result_active is None:
            if result is None:
                self.storage.insert_new_number([number,self.fizzbuzz.compute_results(number)])
                return f'{number},{self.fizzbuzz.compute_results(number)}',201
            else:
                self.storage.activate_num(number)
                return f'{result[0]} , {result[1]}', 200
        else:
            return f'{result[0]} , {result[1]}', 409

    def delete(self,number):
        result=self.storage.get_active(number)
        if result is None:
            return "Not Found",404
        self.storage.deactivate_num(number)
        return "",204

    def range_post(self,min_limit,max_limit):
        result = self.storage.get_range(min_limit,max_limit)
        if len(result)==0:
            return "Not Found",404
        else:
            outputrange=""
            for element in result:
                outputrange+= f'{element[0]} , {element[1]}\n'
            return outputrange,200

```

Por último, se hacen las modificaciones al main.py ya implementando todas las dependencias creadas en my_app.py, las carpetas data_storage y problem_solver.

```
main.py × db_storage.py data_base.py i_data_storage.py my_app.py
main.py > ...
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(int(number))
12     elif request.method == "POST":
13         return my_app.post(int(number))
14     elif request.method == "DELETE":
15         return my_app.delete(int(number))
16     else:
17         return "Not Allowed Method",400
18
19 @app.route('/range/', methods=["POST"])
20 def range():
21     if request.method=="POST":
22         limits = request.get_json()
23         min_limit=limits["min_limit"]
24         max_limit=limits["max_limit"]
25         return my_app.range_post(int(min_limit),int(max_limit))
26     else:
27         return "Not Allowed Method",400
28
29 app.run(host='0.0.0.0', port=81)
```

Pylint

Ahora se mostrará también como parte de la fase BLUE la calificación de todo el proyecto FizzBuzzWeb, posterior a la primera calificación se modificará cada archivo .py por separado para mejorar la calidad del producto.

Calificación General Proyecto:

```
***** Module data_storage.db_storage
FizzBuzzWeb\data_storage\db_storage.py:15:0: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\data_storage\db_storage.py:18:102: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\data_storage\db_storage.py:18:0: C0301: Line too long (102/100) (line-too-long)
FizzBuzzWeb\data_storage\db_storage.py:21:0: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\data_storage\db_storage.py:24:114: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\data_storage\db_storage.py:24:0: C0301: Line too long (114/100) (line-too-long)
FizzBuzzWeb\data_storage\db_storage.py:27:0: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\data_storage\db_storage.py:30:0: C0301: Line too long (107/100) (line-too-long)
FizzBuzzWeb\data_storage\db_storage.py:48:0: C0301: Line too long (130/100) (line-too-long)
FizzBuzzWeb\data_storage\db_storage.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\data_storage\db_storage.py:5:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\data_storage\db_storage.py:11:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\data_storage\db_storage.py:46:4: W0221: Number of parameters was 2 in 'IDataStorage.get_range' and is now 3 in overr
***** Module data_storage.i_data_storage
FizzBuzzWeb\data_storage\i_data_storage.py:26:0: C0304: Final newline missing (missing-final-newline)
FizzBuzzWeb\data_storage\i_data_storage.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\data_storage\i_data_storage.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\data_storage\i_data_storage.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\data_storage\i_data_storage.py:9:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\data_storage\i_data_storage.py:13:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\data_storage\i_data_storage.py:17:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\data_storage\i_data_storage.py:21:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\data_storage\i_data_storage.py:25:4: C0116: Missing function or method docstring (missing-function-docstring)
***** Module problem_solver.fizzbuzz
FizzBuzzWeb\problem_solver\fizzbuzz.py:1:44: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\problem_solver\fizzbuzz.py:4:37: C0303: Trailing whitespace (trailing-whitespace)
FizzBuzzWeb\problem_solver\fizzbuzz.py:6:0: C0304: Final newline missing (missing-final-newline)
FizzBuzzWeb\problem_solver\fizzbuzz.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\problem_solver\fizzbuzz.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\fizzbuzz.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)
***** Module problem_solver.i_problem_solver
FizzBuzzWeb\problem_solver\i_problem_solver.py:6:0: C0304: Final newline missing (missing-final-newline)
FizzBuzzWeb\problem_solver\i_problem_solver.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 3.18/10
```

Calificación General después de organizar data_base.py

```
FizzBuzzWeb\problem_solver\fizzbuzz.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\fizzbuzz.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)
***** Module problem_solver.i_problem_solver
FizzBuzzWeb\problem_solver\i_problem_solver.py:6:0: C0304: Final newline missing (missing-final-newline)
FizzBuzzWeb\problem_solver\i_problem_solver.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
```

Your code has been rated at 3.64/10 (previous run: 3.18/10, +0.47)

(env) D:\CUARTO SEMESTRE\APT>

Calificación General después de organizar db_storage.py

```
FizzBuzzWeb\problem_solver\ fizzbuzz.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)
***** Module problem_solver.i_problem_solver
FizzBuzzWeb\problem_solver\i_problem_solver.py:6:0: C0304: Final newline missing (missing-final-newline)
FizzBuzzWeb\problem_solver\i_problem_solver.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 4.57/10 (previous run: 3.64/10, +0.93)

(env) D:\CUARTO SEMESTRE\APT>
```

Calificación General después de organizar i_data_storage.py

```
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 5.04/10 (previous run: 4.57/10, +0.47)

(env) D:\CUARTO SEMESTRE\APT>
```

Calificación General después de organizar my_app.py

```
FizzBuzzWeb\problem_solver\i_problem_solver.py:1:0: C0114: Missing module docstring (missing-module-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: C0115: Missing class docstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: R0903: Too few public methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 7.07/10 (previous run: 5.04/10, +2.03)

(env) D:\CUARTO SEMESTRE\APT>
```

Calificación General después de organizar main.py

```
docstring (missing-module-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: C0115: Missing class d
ocstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:5:4: C0116: Missing functio
n or method docstring (missing-function-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: R0903: Too few public
methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 9.02/10 (previous run: 7.07/10, +1.94)

(env) D:\CUARTO SEMESTRE\APT>
```

Calificación General después de organizar fizzbuzz.py

```
(env) D:\CUARTO SEMESTRE\APT>pylint FizzBuzzWeb
***** Module problem_solver.i_problem_solver
FizzBuzzWeb\problem_solver\i_problem_solver.py:6:0: C0304: Final newline m
issing (missing-final-newline)
FizzBuzzWeb\problem_solver\i_problem_solver.py:1:0: C0114: Missing module
docstring (missing-module-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: C0115: Missing class d
ocstring (missing-class-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:5:4: C0116: Missing functio
n or method docstring (missing-function-docstring)
FizzBuzzWeb\problem_solver\i_problem_solver.py:3:0: R0903: Too few public
methods (1/2) (too-few-public-methods)

-----
Your code has been rated at 9.59/10 (previous run: 9.02/10, +0.57)

(env) D:\CUARTO SEMESTRE\APT>
```

Calificación General después de organizar i_problem_solver.py

```
(env) D:\CUARTO SEMESTRE\APT>pylint FizzBuzzWeb

-----
Your code has been rated at 10.00/10 (previous run: 9.59/10, +0.41)

(env) D:\CUARTO SEMESTRE\APT>
```

Finalmente se obtiene el mejor puntaje para calificar el proyecto, cabe recalcar que algunas reglas de pylint eran innecesarias para nuestro código por lo que se omitieron algunas que vienen por defecto.

Ya para terminar con nuestra fase BLUE, se hará una prueba general de todo el proyecto desde postman con todas las pruebas.

GET Present Number	Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
GET Not Present Number	Runner	none	1	1s 406ms	35	34 ms
POST Not Present Number	RUN SUMMARY					
POST Not Active Number						
POST Present Number						1
POST Range Numbers	▶ GET Present Number					4 0
POST Range Not Found Numbers	▶ GET Not Present Number					4 0
POST Invalid Range	▶ POST Not Present Number					4 0
DEL Present Number	▶ POST Not Active Number					4 0
DEL Not Present Number	▶ POST Present Number					4 0
	▶ POST Range Numbers					3 0
	▶ POST Range Not Found Numbers					3 0
	▶ POST Invalid Range					3 0
	▶ DELETE Present Number					3 0
	▶ DELETE Not Present Number					3 0

Metacognición

Para esta actividad aprendí que es mejor plantear desde un inicio el tema del DIP, para no tener que meter tanta mano a la modificación del código.

Aprendí cómo funcionan las APIs y las URL mucho mejor, y cómo funciona en general los request que se le pueden hacer.

En un aspecto que me ayudó una AI, fue al crear la pruebas en Postman, pues Postman tiene un Bot para generar Test.

Por último, aprendí a usar nuevas herramientas como Flask, Postman y DB Browser.