

Reto técnico pruebas de servicio con Postman

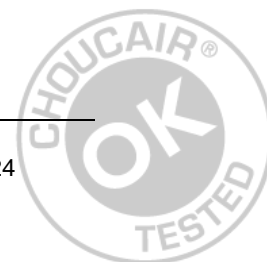
Primera parte

Valentin Despa es un desarrollador de software que está trabajando en un proyecto, en el cual hace un servicio Rest, para poder ver libros disponibles, así como poder hacer ordenes de libros, así como actualización de estas órdenes, y hacer un borrado de una orden en caso de ser necesario. Por este motivo después de terminado el desarrollo del servicio el desarrollador quiere probar el correcto funcionamiento de este y para esto contrata a la empresa Choucair Testing.

Por este motivo el desarrollador le proporciona a Choucair Testing la documentación de la Api en el siguiente link: <https://github.com/vdespa/introduction-to-postman-course/blob/main/simple-books-api.md>.

Ya después de haber leído la documentación, con el cliente se llega a un acuerdo de que las pruebas realizadas se harán en Postman, y que deben estar organizadas en una colección que se divida en subcarpetas en donde nos permitan ver de mejor manera que endpoint se está probando. Por otra parte, lo que el cliente espera que tenga nuestras pruebas en la herramienta de Postman es lo siguiente:

1. Se debe crear un ambiente de pruebas para hacer la ejecución de nuestras pruebas.
2. También se hace necesario el uso de variables desde el ambiente creado.
3. Poder hacer un crud con cada uno de los endpoints del servicio (incluir por lo menos un camino feliz y uno no feliz de cada endpoint).
4. Cada una de las validaciones que se hacen deben estar respaldadas con aserciones que lo justifiquen, no solo validaciones de código de respuesta.



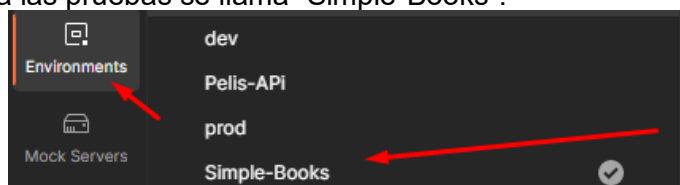
SOLUCIÓN PARTE 1

Para iniciar con la solución de las diferentes pruebas que se puede hacer a los endpoints en las siguientes imágenes se podrá evidenciar el procedimiento realizado con Postman.

Como primera parte se establece un ambiente para las variables y se organizan las colecciones en subcarpetas para contener las peticiones que se harán en los endpoints.

CONFIGURACIÓN DE AMBIENTE Y COLECCIÓN

Ambiente creado para las pruebas se llama "Simple-Books".

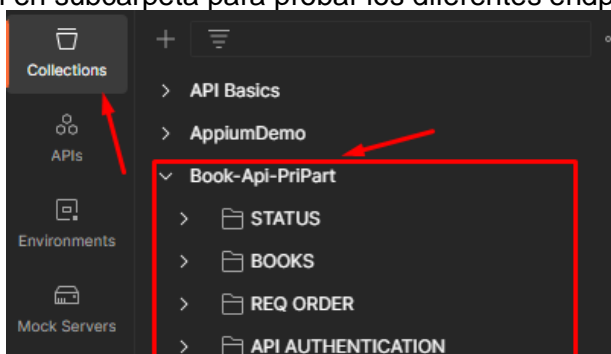


Variables agregadas en el ambiente

A screenshot of the 'Simple-Books' environment variables table in Postman. The table has columns for VARIABLE, TYPE, INITIAL VALUE, CURRENT VALUE, and Persist All. Several variables are listed and checked, including Url_Base, Status, Books, Orders, Clients, user, email, and token. The entire table area is highlighted with a red border.

VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE	Persist All	Reset All
<input checked="" type="checkbox"/> Url_Base	default	https://simple-books-apl...	https://simple-books-apl.glitch.me		
<input checked="" type="checkbox"/> Status	default	/status	/status		
<input checked="" type="checkbox"/> Books	default	/books	/books		
<input checked="" type="checkbox"/> Orders	default	/orders	/orders		
<input checked="" type="checkbox"/> Clients	default	/api-clients	/api-clients		
<input checked="" type="checkbox"/> user	default				
<input checked="" type="checkbox"/> email	default				
<input checked="" type="checkbox"/> token	default				

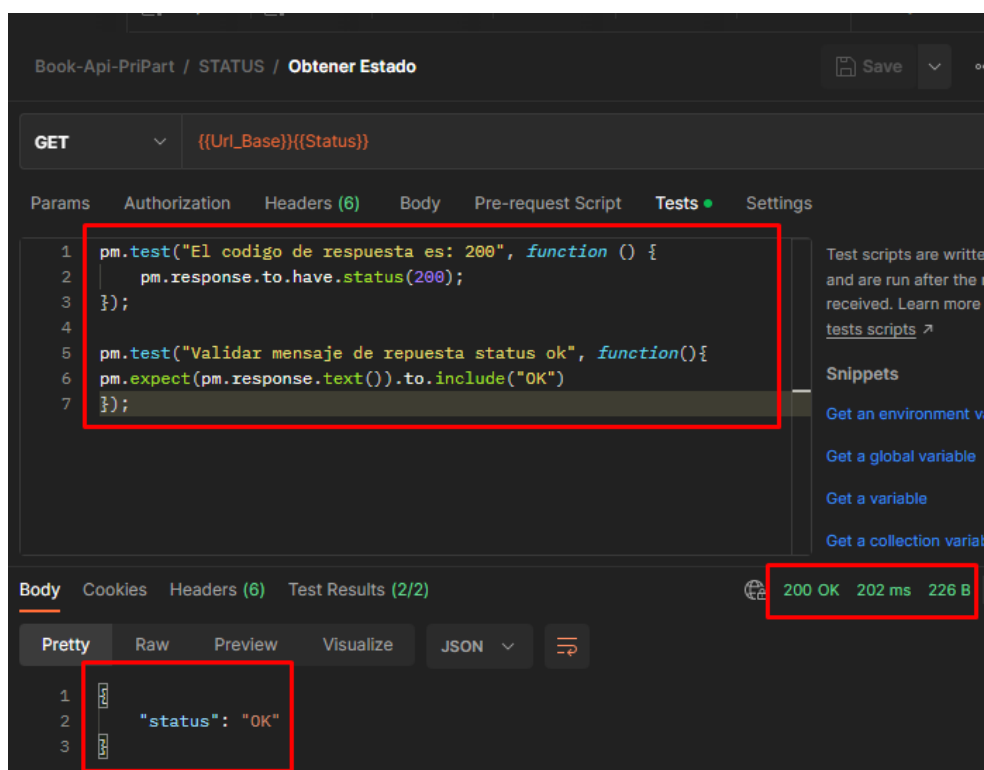
Se organizó la colección en subcarpeta para probar los diferentes endpoints



INICIO DE PRUEBAS A ENDPOINTS

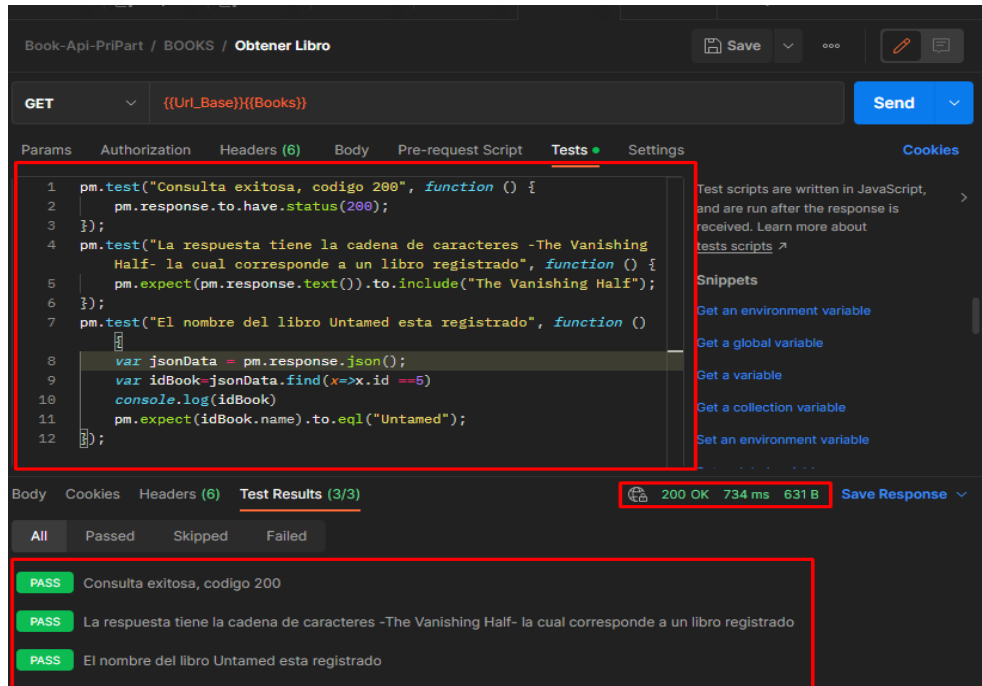
A continuación, después de tener implementando el ambiente y la colección se inician a realizar pruebas a los endpoints de la Api, iniciando con el Status

Petición Get con aserciones utilizadas para validación del test y código de respuesta obtenido en la petición. EndPoint

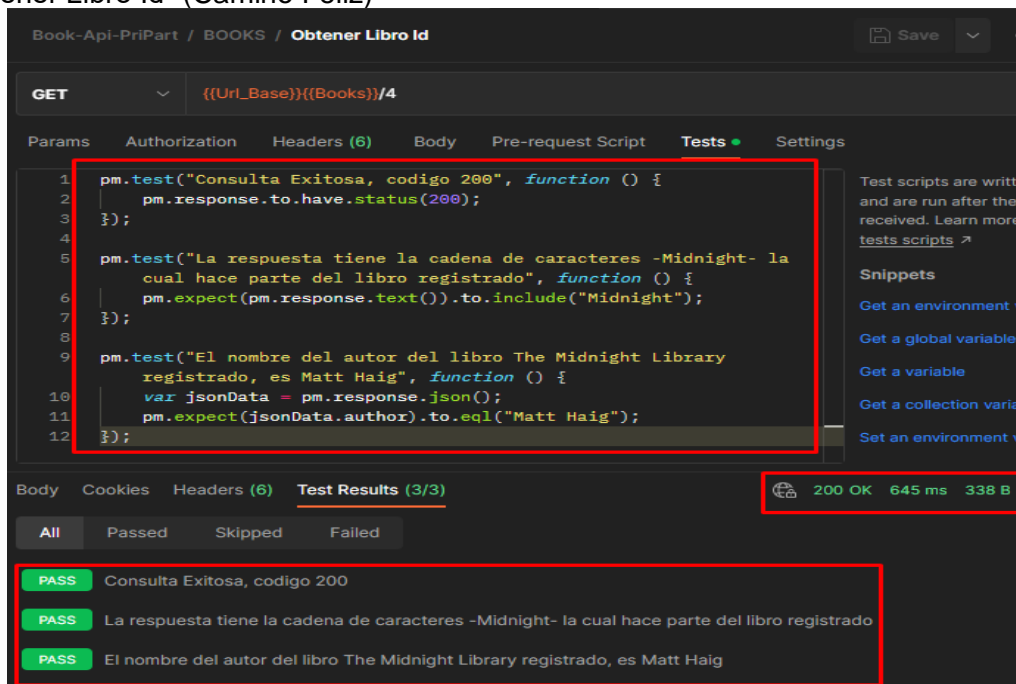


Evaluando endpoint “Books” para obtener los libros y verificar que exista, se implementan 3 aserciones para los test de obtener libros

“Obtener Libro”

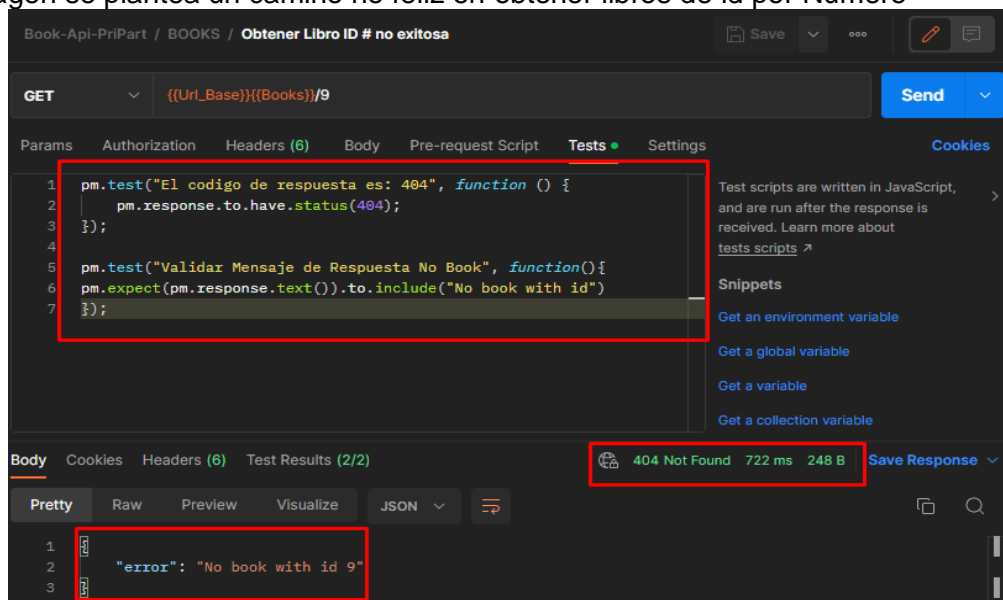


Se continua en el endpoint de obtener libros y ahora se prueba para obtenerlos mediante un id
Se establece 3 aserciones para las pruebas con camino exitoso obteniendo el id correcto
“Obtener Libro Id” (Camino Feliz)

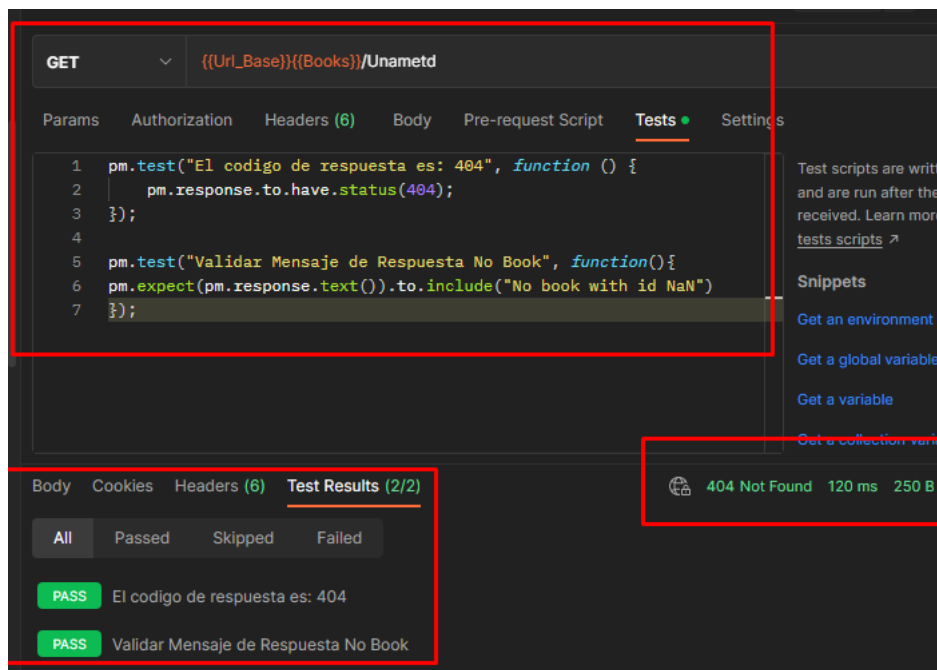


Para las diferentes pruebas en los endpoints se implementa un camino no feliz y verificarlo

En la imagen se plantea un camino no feliz en obtener libros de id por Numero



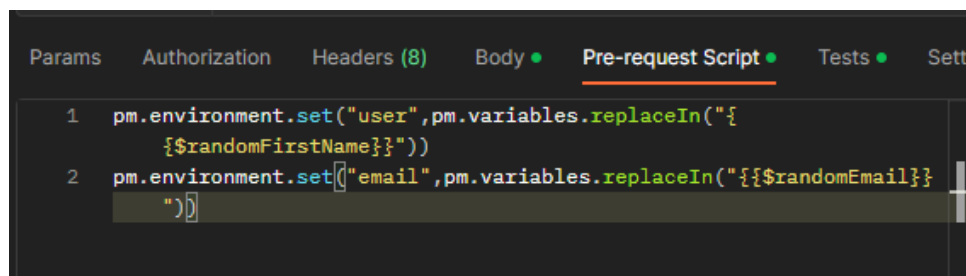
Como ejemplo de otro camino no feliz, se prueba obtener libros con letras y se hacen pruebas



Continuando con las pruebas, la siguiente es poder revisar el token y realizar las pruebas en los diferentes endpoints una vez se obtengan el token.

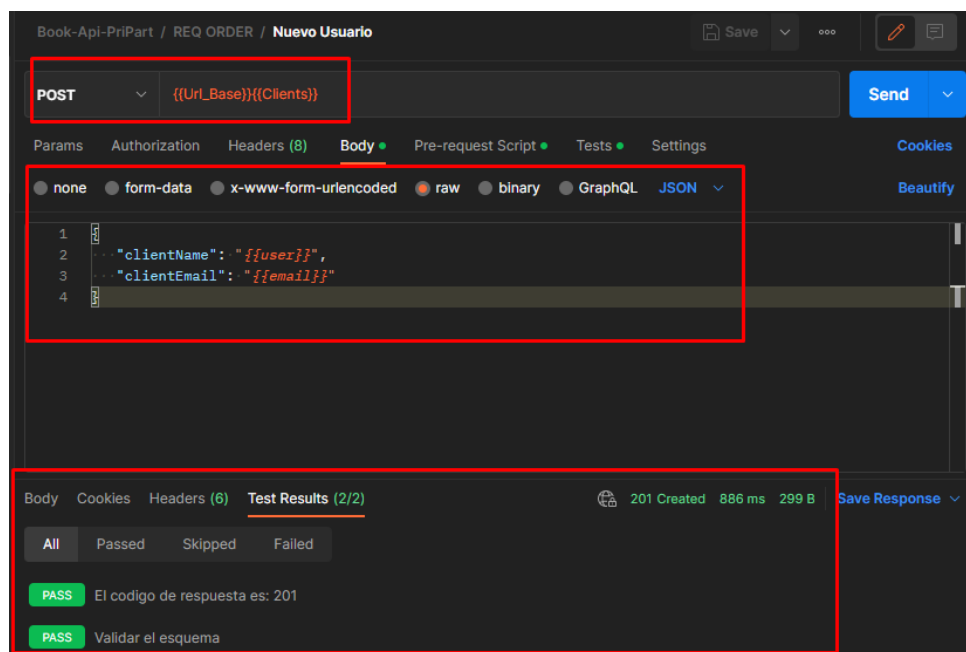
En las siguientes imágenes se obtiene de manera exitosa el token.

Se debe establecer pre-request script para generar los usuarios y correos aleatoriamente como variables

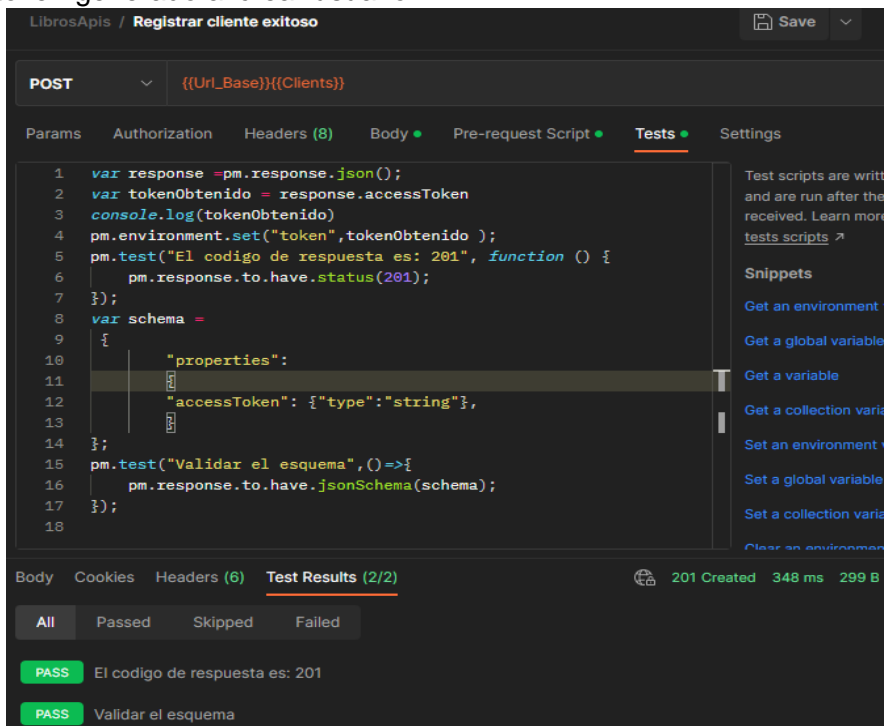


```
1 pm.environment.set("user",pm.variables.replaceIn("{{$randomFirstName}}"))
2 pm.environment.set("email",pm.variables.replaceIn("{{$randomEmail}}"))
```

Se establece una petición post y se envía un schema desde el body a las variables creadas en el ambiente



Finalmente se agregan las aserciones para los test en el código de respuesta y el schema para almacenar el token generado al crear usuario



The screenshot shows the Postman interface for a POST request named "Registrar cliente exitoso". The URL is set to `{{Url_Base}}{{Clients}}`. The "Tests" tab is active, displaying the following JavaScript code:

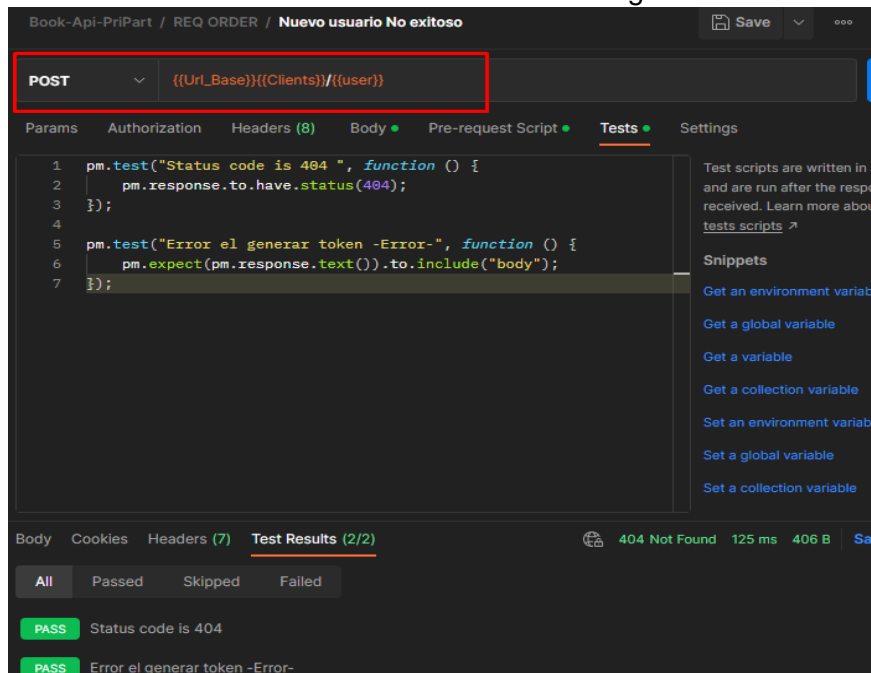
```
1 var response = pm.response.json();
2 var tokenObtenido = response.accessToken
3 console.log(tokenObtenido)
4 pm.environment.set("token",tokenObtenido );
5 pm.test("El codigo de respuesta es: 201", function () {
6     pm.response.to.have.status(201);
7 });
8 var schema =
9 {
10     "properties":
11     {
12         "accessToken": {"type":"string"},
13     }
14 };
15 pm.test("Validar el esquema", ()=>{
16     pm.response.to.have.jsonSchema(schema);
17 });
18
```

The "Test Results" section at the bottom shows two passed tests:

- PASS El codigo de respuesta es: 201
- PASS Validar el esquema

The status bar indicates a 201 Created response with a 348 ms execution time and 299 B of data.

Un camino no feliz en la creación de un nuevo token seria la siguiente



The screenshot shows the Postman interface for a POST request named "Nuevo usuario No exitoso". The URL is set to `{{Url_Base}}{{Clients}}/{{user}}`. The "Tests" tab is active, displaying the following JavaScript code:

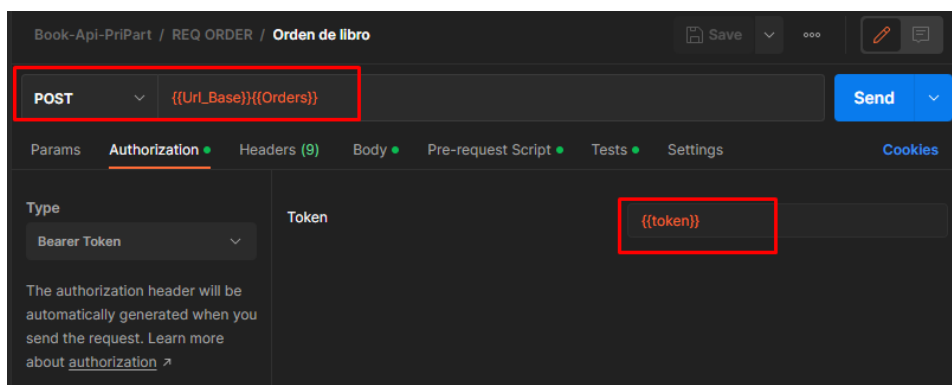
```
1 pm.test("Status code is 404 ", function () {
2     pm.response.to.have.status(404);
3 });
4
5 pm.test("Error el generar token -Error-", function () {
6     pm.expect(pm.response.text()).to.include("body");
7 });
```

The "Test Results" section at the bottom shows two passed tests:

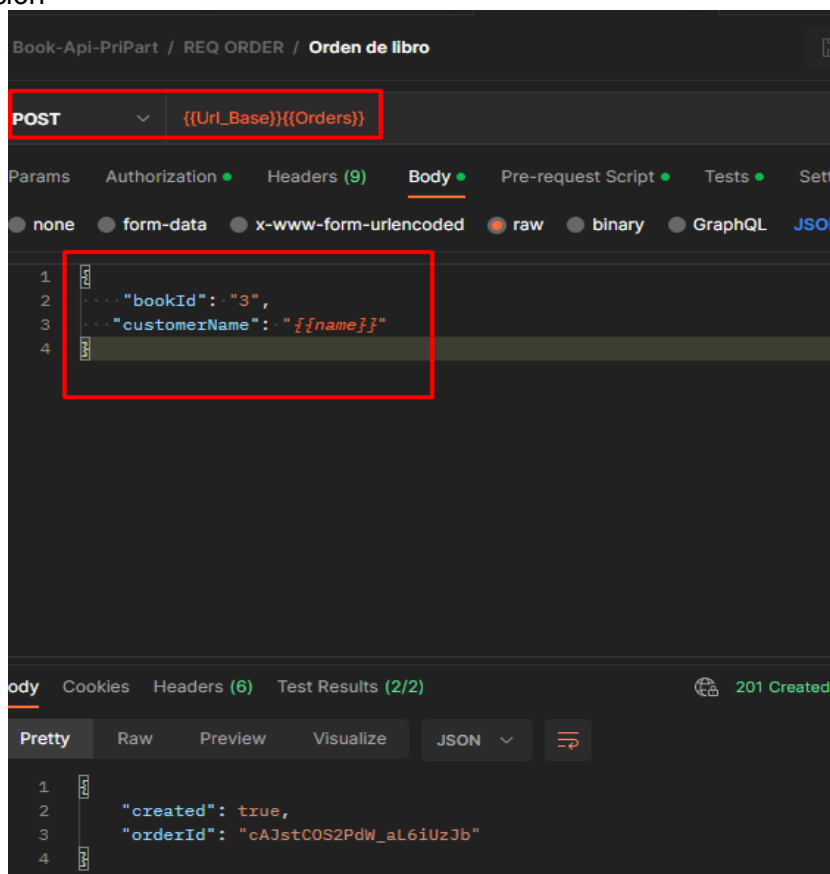
- PASS Status code is 404
- PASS Error el generar token -Error-

The status bar indicates a 404 Not Found response with a 125 ms execution time and 406 B of data.

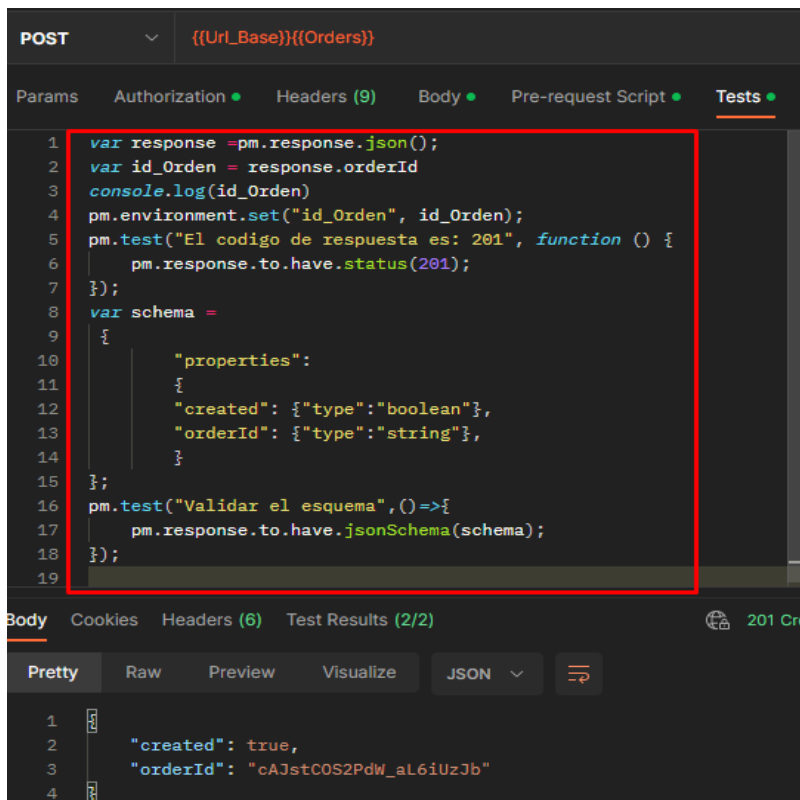
Ordenando un libro de manera exitosa en el endpoint Orders
Se debe agregar el token ya que sin el token no es posible hacer la petición



Una vez agregado el token se hace la petición al endpoint y se agrega desde el body el schema para la petición



Se agregan las aserciones para los test de la orden y que la respuesta sea la esperada junto con el schema



```
POST {{Url_Base}}{{Orders}}

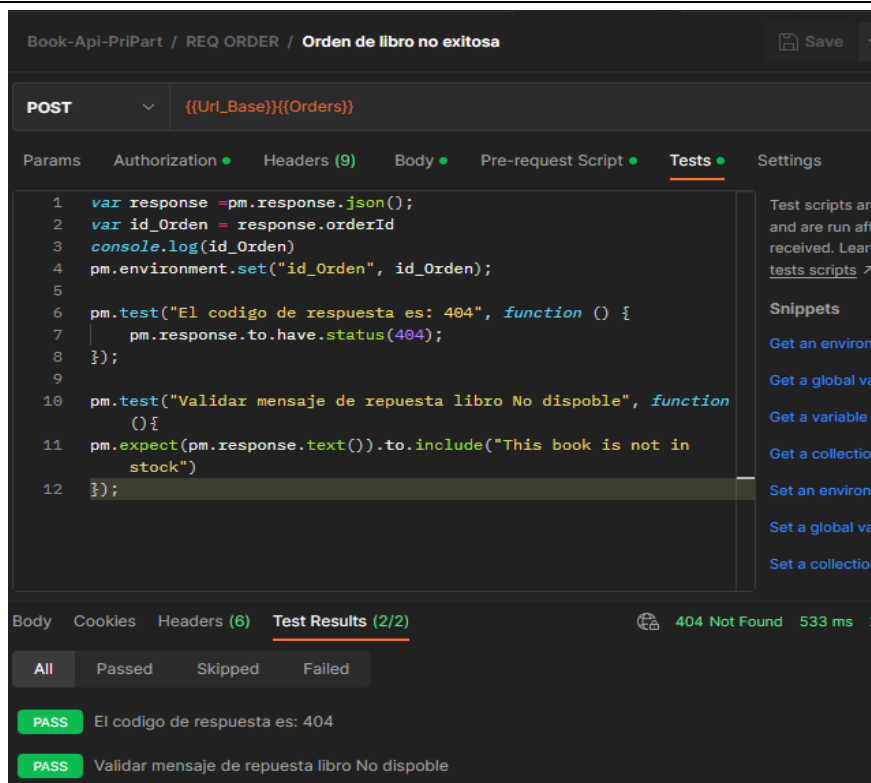
1  var response = pm.response.json();
2  var id_Orden = response.orderId
3  console.log(id_Orden)
4  pm.environment.set("id_Orden", id_Orden);
5  pm.test("El codigo de respuesta es: 201", function () {
6    pm.response.to.have.status(201);
7  });
8  var schema =
9  {
10     "properties":
11     {
12       "created": {"type": "boolean"},
13       "orderId": {"type": "string"},
14     }
15   };
16  pm.test("Validar el esquema", () => {
17    pm.response.to.have.jsonSchema(schema);
18  });
19

Body  Cookies  Headers (6)  Test Results (2/2)  201 Cr

Pretty  Raw  Preview  Visualize  JSON  [icon]

1  [icon]
2  "created": true,
3  "orderId": "cAJstCOS2PdW_aL6iUzJb"
4  [icon]
```

Un Camino no feliz con este endpoint es no encontrar el libro de manera exitosa ya que se hace la petición a un id que no pertenece a un libro.



```
1 var response = pm.response.json();
2 var id_Orden = response.orderId
3 console.log(id_Orden)
4 pm.environment.set("id_Orden", id_Orden);
5
6 pm.test("El codigo de respuesta es: 404", function () {
7     pm.response.to.have.status(404);
8 });
9
10 pm.test("Validar mensaje de repuesta libro No dispoble", function
    () {
11     pm.expect(pm.response.text()).to.include("This book is not in
        stock")
12 });
```

Test Results (2/2)

All	Passed	Skipped	Failed
PASS	El codigo de respuesta es: 404		
PASS	Validar mensaje de repuesta libro No dispoble		

Continuando con el endpoint de Orders, con la siguiente aserción se prueba que lleguen todas las ordenes correctamente.

The screenshot shows a Postman interface for a GET request to 'Obtener todas las ordenes'. The URL is {{Url_Base}}{{Orders}}. The 'Tests' tab is active, showing a JavaScript test script that defines a JSON schema and validates the response against it. The schema defines an array of objects with properties: id (string), bookId (number), customerName (string), createdBy (string), quantity (number), and timestamp (number). The test script includes a schema validation and a status code check for 200. The 'Body' tab shows the response in JSON format, with the 'id' field highlighted.

```
1 var schema = {
2   "items":{
3     "type": "object",
4     "properties":
5     {
6       "id": {"type":"string"},
7       "bookId":{"type":"number"},
8       "customerName":{"type":"string"},
9       "createdBy": {"type":"string"},
10      "quantity": {"type":"number"},
11      "timestamp": {"type":"number"},
12    }
13  }
14 };
15 pm.test("Validar el esquema",()=>{
16   pm.response.to.have.jsonSchema(schema);
17 });
18 pm.test("El codigo de respuesta es: 200", function () {
19   pm.response.to.have.status(200);
20 });
```

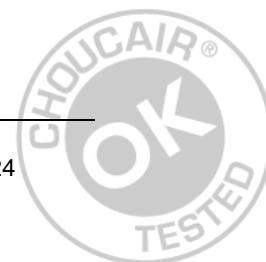
Body: { "id": "Zb1sFw3aAsuYdZ9lc6avY", ... }

Obtener orden por id

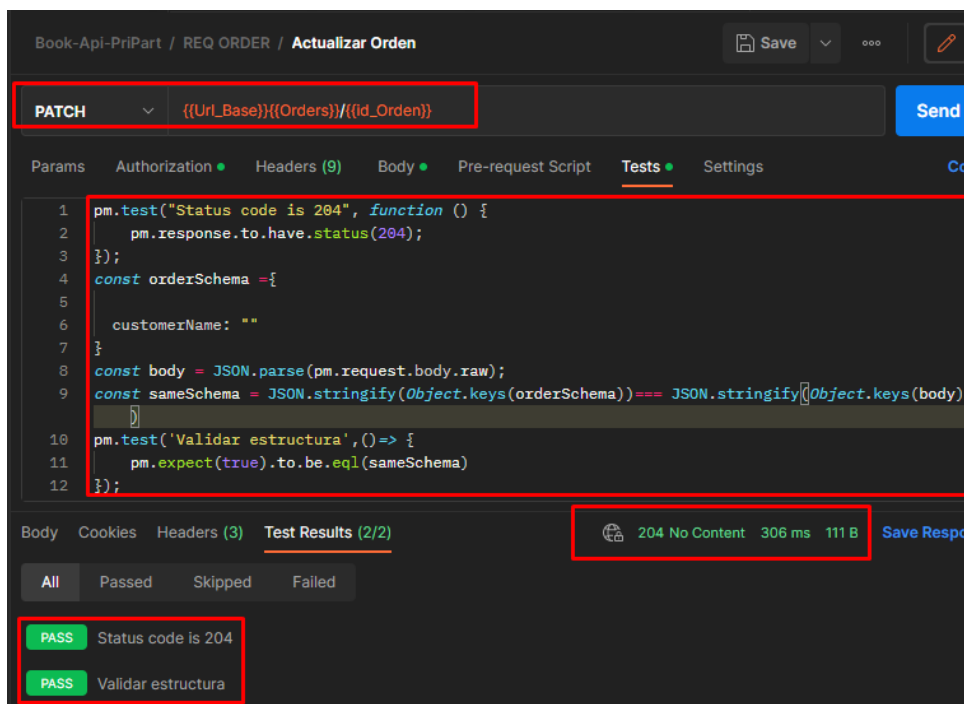
The screenshot shows a Postman interface for a GET request to 'Orden Por ID'. The URL is {{Url_Base}}{{Orders}}/{{id_Orden}}. The 'Tests' tab is active, showing a JavaScript test script that defines a JSON schema and validates the response against it. The schema defines an object with properties: id (string), bookId (number), customerName (string), createdBy (string), quantity (number), and timestamp (number). The test script includes a schema validation and a status code check for 200. The 'Body' tab shows the response in JSON format, with the 'id' field highlighted.

```
3 });
4 var schema = {
5   "items":{
6     "type": "object",
7     "properties":
8     {
9       "id": {"type":"string"},
10      "bookId":{"type":"number"},
11      "customerName":{"type":"string"},
12      "createdBy": {"type":"string"},
13      "quantity": {"type":"number"},
14      "timestamp": {"type":"number"},
15    }
16  }
17 };
18 pm.test("Validar el esquema",()=>{
19   pm.response.to.have.jsonSchema(schema);
20 });
```

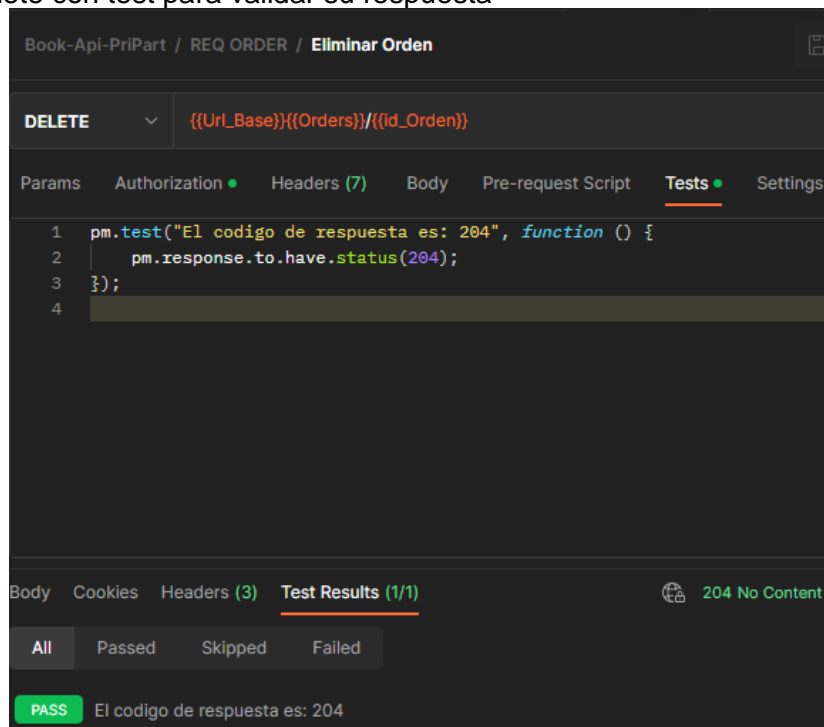
Body: { "id": "Zb1sFw3aAsuYdZ9lc6avY", "bookId": 3, "customerName": "Meagan", "createdBy": "6d4b1c72a7c538c81f21975022604523e337b9c88a643c21315e81f139368773", ... }



En la siguiente imagen se utiliza la petición patch para actualizar la orden de un libro por id, con aserciones que validan su estructura y estatus de respuesta.



Petición Delete con test para validar su respuesta



Segunda parte

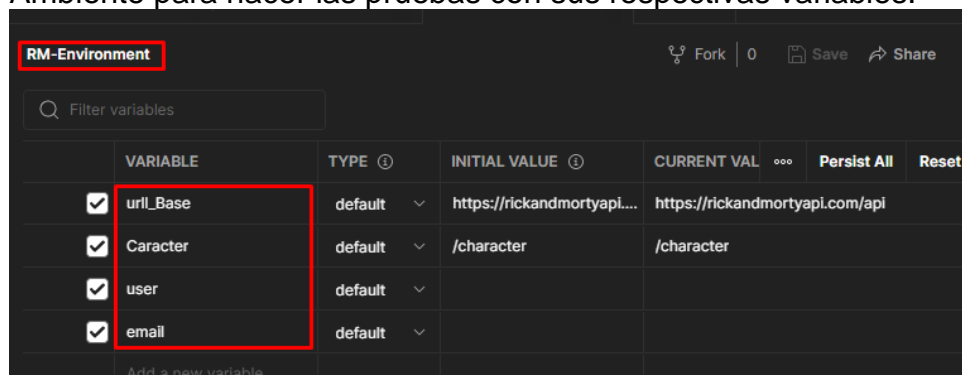
Para esta segunda vamos a trabajar sobre la siguiente Api <https://rickandmortyapi.com/documentation> , la cual es muy usada para hacer pruebas de servicio, para este ejercicio se plantea hacer solicitudes sobre el endpoint de "Character" y se espera puedan hacer lo siguiente con este endpoint:

1. Hacer un listado de todos los personajes, y hacer un test con la información de un personaje que aparezca en esa respuesta (Tener en cuenta que deben hacer una búsqueda entre los diferentes objetos que hay anidados en la respuesta)
2. Hacer un listado de personajes por página (en la documentación se explica cómo hacer paginación), y hacer un test por aparición de personaje en esta página.
3. Traer un personaje por id, y hacer un test correspondiente.
4. Al momento de buscar múltiples Id, ingresando id's que no existen nos da una respuesta, la cual es un array vacío, hacer un test en donde se valide que se tiene una respuesta vacía

SOLUCIÓN PARTE 2

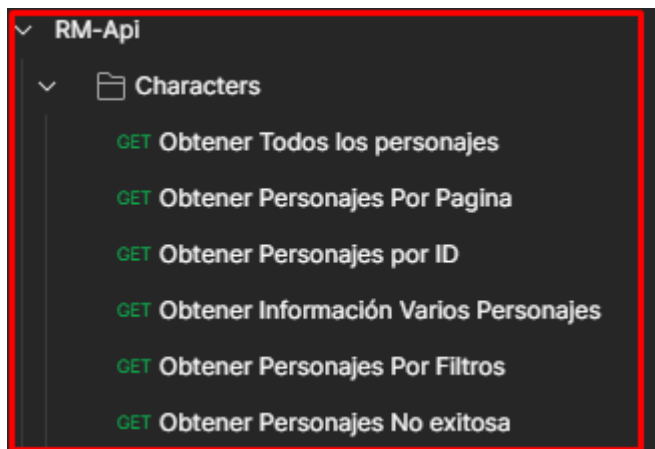
En la solución para la segunda parte del reto se procede a crear el ambiente y colección correspondiente como se puede ver en la siguiente imagen:

Ambiente para hacer las pruebas con sus respectivas variables.



	VARIABLE	TYPE	INITIAL VALUE	CURRENT VAL	Persist All	Reset
<input checked="" type="checkbox"/>	url_Base	default	https://rickandmortyapl...	https://rickandmortyapi.com/api		
<input checked="" type="checkbox"/>	Caracter	default	/character	/character		
<input checked="" type="checkbox"/>	user	default				
<input checked="" type="checkbox"/>	email	default				

Colección con sus peticiones requeridas y carpeta de endpoint a probar



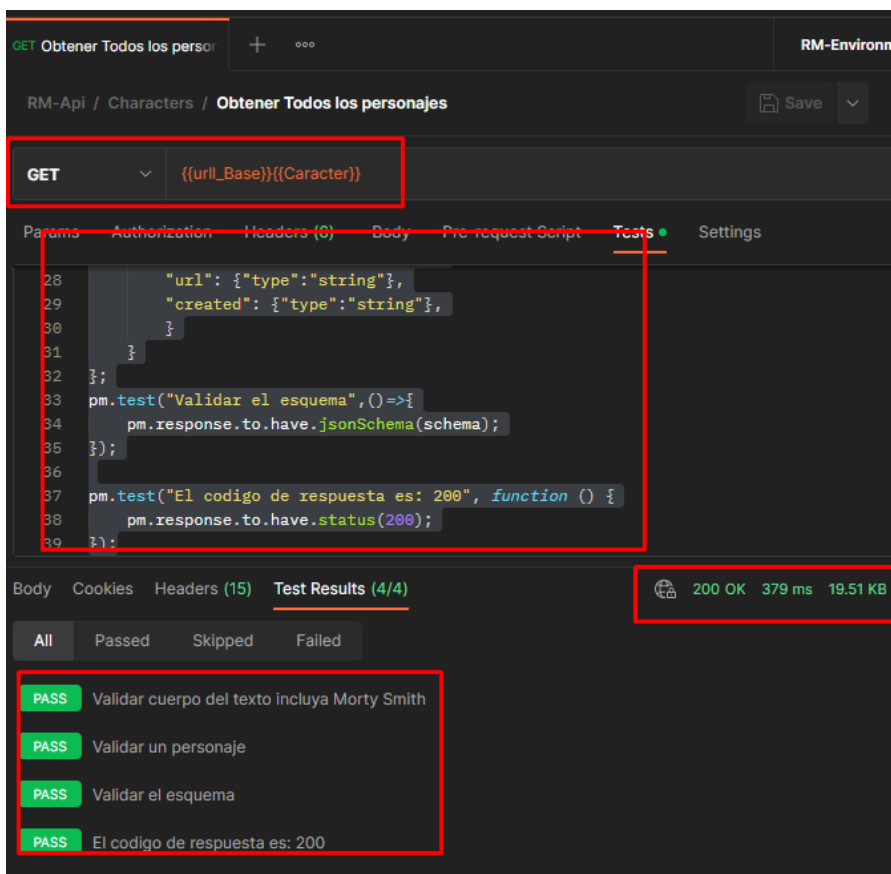
Una vez configurados y creados el ambiente junto con la colección se continua con las peticiones.

Se crea la petición para Obtener todos los personajes y evaluar el cuerpo del texto de personaje, validar la información del personaje junto con el schema y finalmente probar el codigo de respuesta: 200

Asercion utilizada.

```
1 pm.test("Validar cuerpo del texto incluya Morty Smith", function(){
2 pm.expect(pm.response.text()).to.include("Morty Smith")
3 });
4
5 var jsonData = pm.response.json();
6
7 pm.test("Validar un personaje", ()=>{
8   var pers = jsonData.results.find(x => x.id == 3)
9   pm.expect(pers.status).to.equal("Alive");
10  console.log(pers)
11 })
12
13 var schema = {
14   "items":{
15     "type": "object",
16     "properties":
17     {
18       "id": {"type": "number"},
19       "name": {"type": "string"},
20       "status": {"type": "string"},
21       "species": {"type": "string"},
22       "type": {"type": "string"},
23       "gender": {"type": "string"},
24       "origin": {"type": "string"},
25       "location": {"type": "string"},
26       "image": {"type": "string"},
27       "episode": {"type": "string"},
28       "url": {"type": "string"},
29       "created": {"type": "string"},
30     }
31   }
32 };
33 pm.test("Validar el esquema", ()=>{
34   pm.response.to.have.jsonSchema(schema);
35 });
36
37 pm.test("El código de respuesta es: 200", function () {
38   pm.response.to.have.status(200);
39 });
```

Resultado de ejecución



La siguiente petición creada es para obtener personajes por página, enviando un número de página desde la URL, realizando las pruebas para validar un personaje por página, mensaje de respuesta 200 y personaje con las características solicitadas.

Reto Técnico Postman - Daniel Sandoval

En este documento se da solución al reto técnico para las pruebas de servicio con postman

```
1  var          jsonData          =          pm.response.json();
2  pm.test("Validar un personaje de la pagina",()=>{
3      var pers = jsonData.results.find(x => x.id ==81)
4      pm.expect(pers.species).to.equal("Animal");
5      console.log(pers)
6  })
7  pm.test("Mensaje de repuesta 200", function () {
8      pm.response.to.have.status(200);
9  });
10 var
11     "name":          "Cronenberg
12     "status":          Rick",
13     "species":          "unknown",
14     "gender":          "Cronenberg",
15     "gender":          "Male"
16     };
17 pm.test("Personaje con las características solicitadas",()=>{
18     var pers = pm.response.json();
19     var resul = pers.results.find(x => x.id ==82)
20     console.log(resul)
21     pm.expect(resul).to.include(MyTest);
22 })
```

Resultado obtenido por la petición

The screenshot shows the Postman interface for a GET request. The URL is `{{url_Base}}{{Character}}?page=5`. The Tests tab is active, showing the following code:

```
var MyTest={
  "name": "Cronenberg Rick",
  "status": "unknown",
  "species": "Cronenberg",
  "gender": "Male"
};
pm.test("Personaje con las características solicitadas",()=>{
  var pers = pm.response.json();
  var resul = pers.results.find(x => x.id ==82)
  console.log(resul)
  pm.expect(resul).to.include(MyTest);
})
```

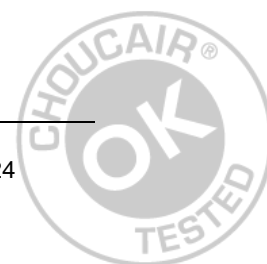
The Test Results tab shows three passed tests:

- PASS Validar un personaje de la pagina
- PASS Mensaje de repuesta 200
- PASS Personaje con las características solicitadas

Reto Técnico Postman - Daniel Sandoval

En este documento se da solución al reto técnico para las pruebas de servicio con postman

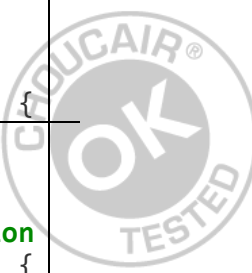
La siguiente petición realizada es Obtener personajes por ID, para las pruebas de esta petición se establecen varias aserciones, para verificar los diferentes atributos de la consulta, el script de prueba utilizado es el siguiente:



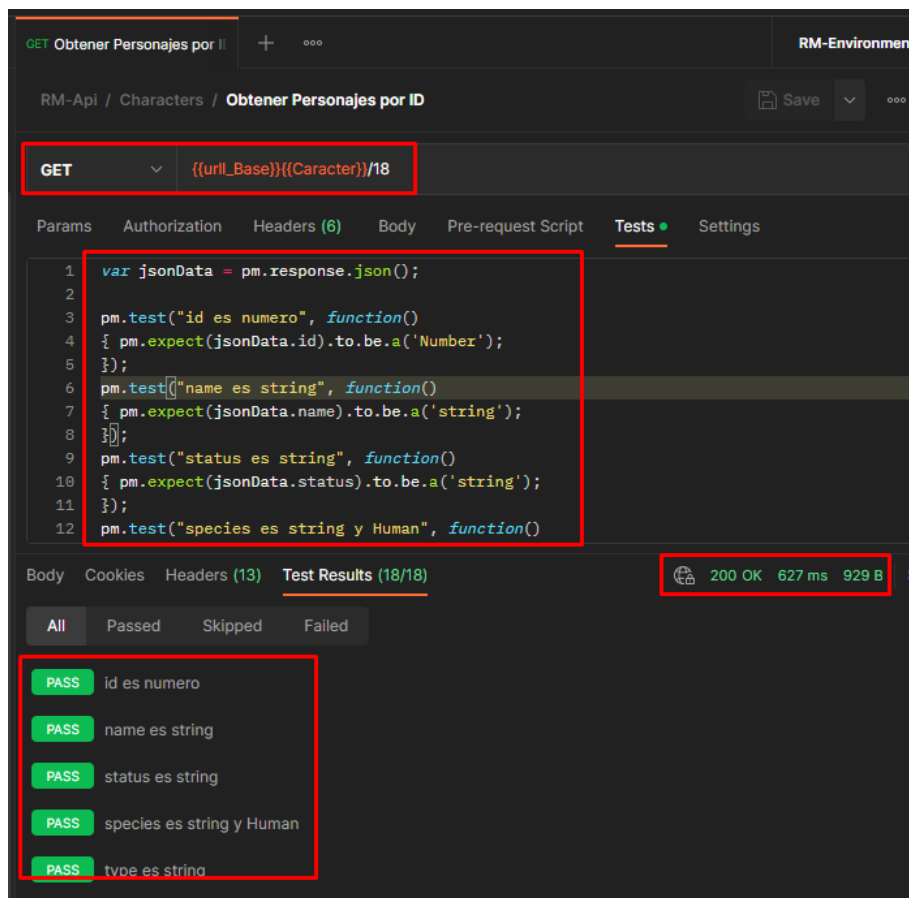
```

1  var jsonData = pm.response.json();
2  pm.test("id es numero", function()
3  {
4      pm.expect(jsonData.id).to.be.a('Number');
5  });
6  pm.test("name es string", function()
7  {
8      pm.expect(jsonData.name).to.be.a('string');
9  });
10 pm.test("status es string", function()
11 {
12     pm.expect(jsonData.status).to.be.a('string');
13 });
14 pm.test("species es string y Human", function()
15 {
16     pm.expect(jsonData.species).to.be.a('string').and.to.be.eql("Human");
17 });
18 pm.test("type es string", function()
19 {
20     pm.expect(jsonData.type).to.be.a('string');
21 });
22 pm.test("gender es string", function()
23 {
24     pm.expect(jsonData.gender).to.be.a('string');
25 });
26 pm.test("Origin es objeto", function()
27 {
28     pm.expect(jsonData.origin).to.be.a('object');
29 });
30 pm.test("location es objeto", function()
31 {
32     pm.expect(jsonData.location).to.be.a('object');
33 });
34 pm.test("image es string", function()
35 {
36     pm.expect(jsonData.image).to.be.a('string');
37 });
38 pm.test("episode es array", function()
39 {
40     pm.expect(jsonData.episode).to.be.a('array');
41 });
42 pm.test("url es string", function()
43 {
44     pm.expect(jsonData.url).to.be.a('string');
45 });
46 pm.test("created es string", function()
47 {
48     pm.expect(jsonData.created).to.be.a('string');
49 });
50 pm.test("Validar especie humana", ()=>{
51     pm.expect(jsonData.species).to.equal("Human");
52     console.log(jsonData)
53 })
54 pm.test("Validar un origen personaje : unknown", ()=>{
55     var Origen = jsonData.origin
56     pm.expect(Origen.name).to.equal("unknown");
57     console.log(Origen)
58 })
59 pm.test("Status mensaje es 200", function () {
60     pm.expect(pm.response.status).to.equal(200);
61 })
62 pm.test("validando propiedades que no debe tener el objeto", function () {
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   //

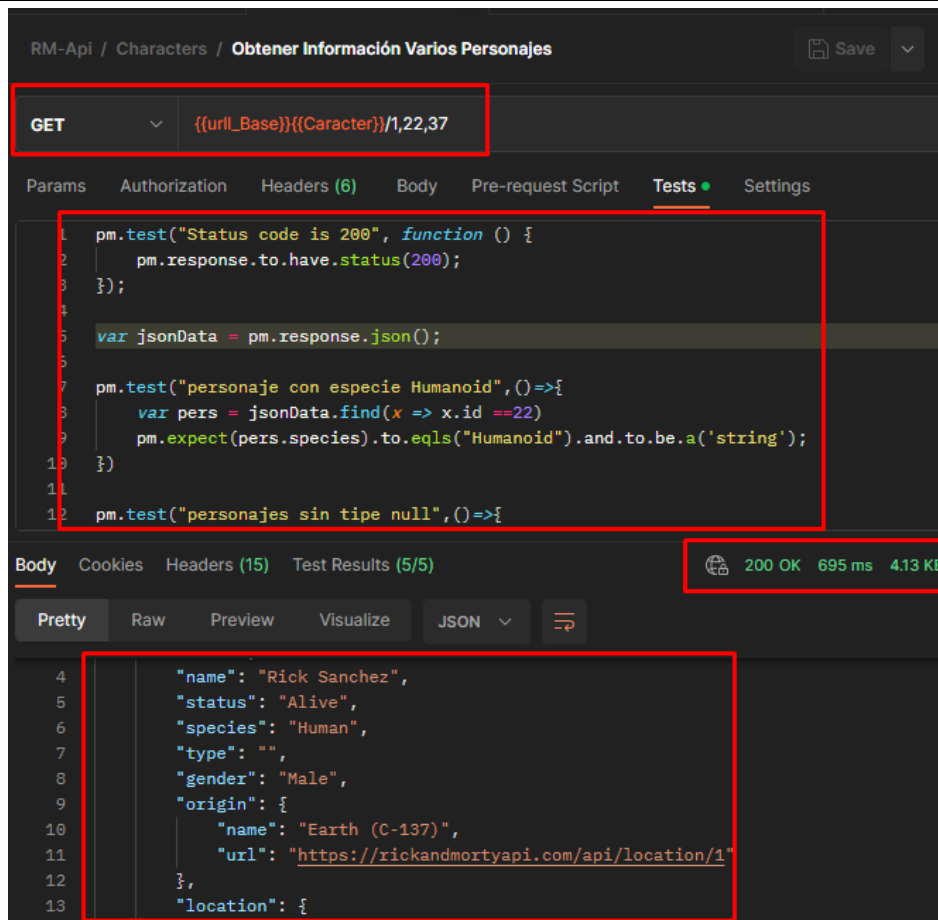
```



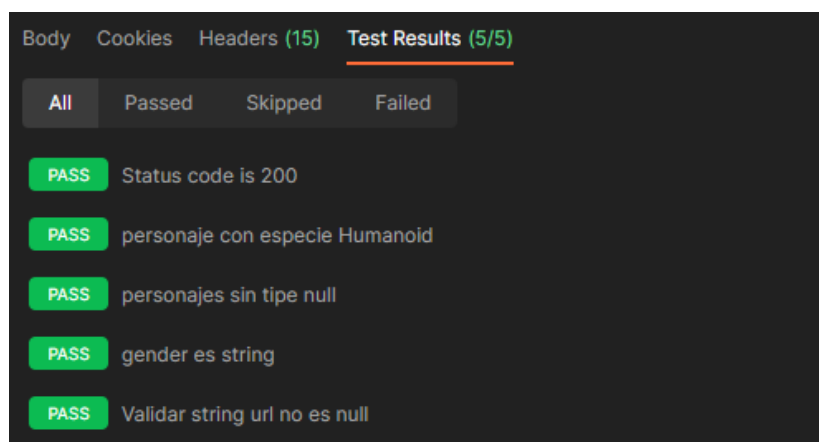
En la imagen se podrá observar el resultado de la petición una vez se implementa el script para las pruebas del endpoint para obtener personajes por id:



Se continúan probando las peticiones a la Api y la siguiente es Obtener información de Varios personajes por su id desde la url, se envían varias id consecutivamente en la url y se verifica mediante varios test.



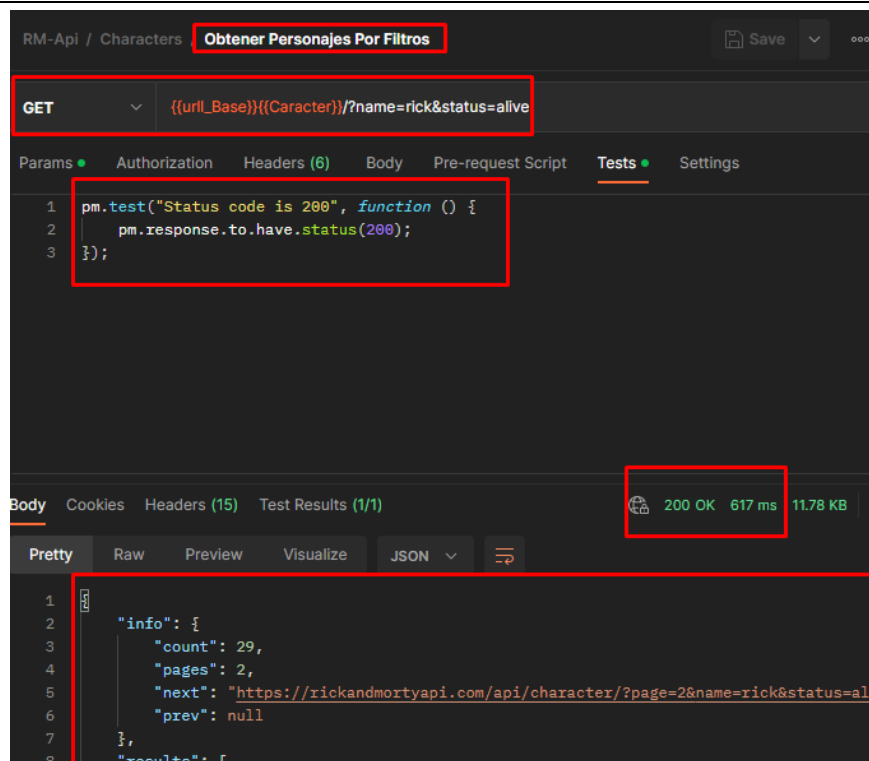
Los test implementados son los siguientes:



El Script que se implementó para elaborar los test a esta petición se compone de las siguientes aserciones:

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4
5 var jsonData = pm.response.json();
6
7 pm.test("personaje con especie Humanoid", ()=>{
8   var pers = jsonData.find(x => x.id ==22)
9   pm.expect(pers.species).to.eqls("Humanoid").and.to.be.a('string');
10 })
11
12 pm.test("personajes sin tipe null", ()=>{
13   var perspn = jsonData.find(x => x.id ==1)
14   pm.expect(perspn.type).to.be.a("string")
15 })
16
17 pm.test("gender es string", ()=>{
18   var pers = jsonData.find(x => x.id ==22)
19   pm.expect(pers.gender).to.be.equal("Male").and.to.be.a('string');
20   console.log(pers)
21 })
22
23 pm.test("Validar string url no es null", ()=>{
24   var perspn = jsonData.find(x => x.id ==22)
25   var Origen = perspn.origin
26   pm.expect(Origen.url).to.be.a("string")
27 })
```

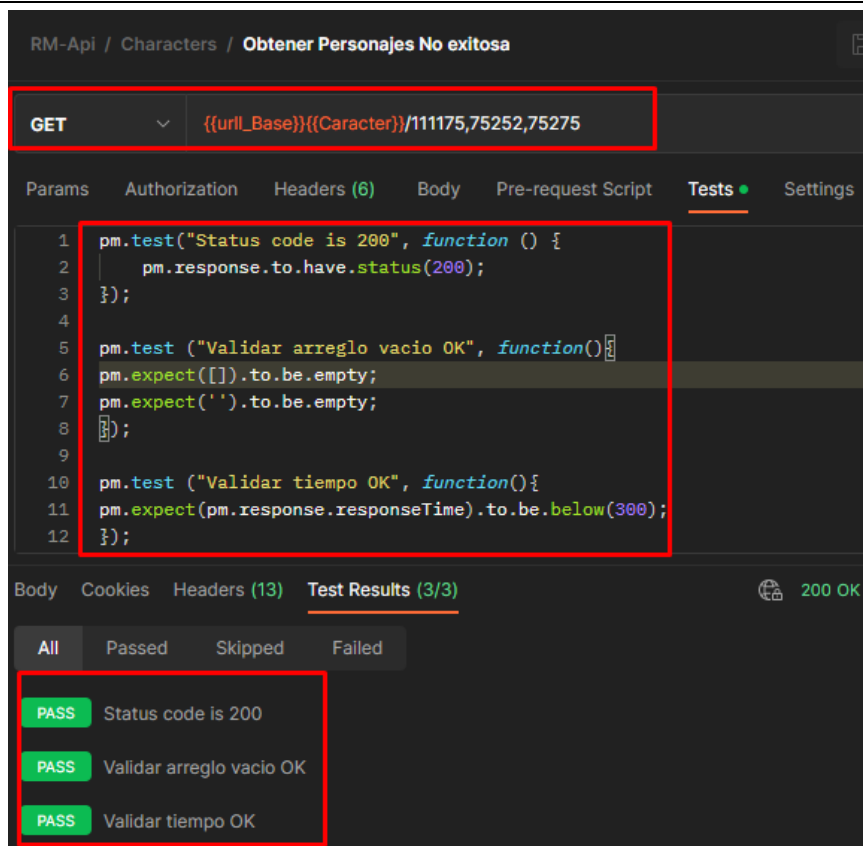
También se utiliza una petición para poder Obtener personajes por Filtros, mediante la url se envían filtros para obtener únicamente los personajes que se implementan en el filtro en este caso por su nombre y estatus.



El único test implementado fue para validar el código de respuesta como se observa en la imagen y como se hizo anteriormente.

Finalmente, como última petición para el endpoint de Characters se establece un camino no feliz al momento de obtener personajes por id, enviando muchos números en el parámetro de la URL

Las pruebas que se utilizaron fueron para validar el código de respuesta, que el resultado sea un arreglo vacío ya que no existirían personajes e igualmente se valida el tiempo de respuesta.



El script utilizado para las aserciones son las siguientes:

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4
5 pm.test ("Validar arreglo vacio OK", function(){
6   pm.expect([]).to.be.empty;
7   pm.expect('').to.be.empty;
8 });
9
10 pm.test ("Validar tiempo OK", function(){
11   pm.expect(pm.response.responseTime).to.be.below(300);
12 });
```