

# API Practice



# POSTMAN

1. Go to <https://dummyapi.io/> and click on login. Log in with your Google account and generate an App ID. Create the DUMMYAPI. Go to the DUMMYAPI collection and select the Authorization tab. Select the API Key authorization type. Enter Key and Value. Click Save.

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Test Workspace' with 'Collections' (1\_frist-tests, DUMMYAPI), 'Environments', and 'History'. The main area shows the 'DUMMYAPI' collection under 'Overview'. A red arrow points to the 'Authorization' tab, which is selected. Under 'Type', 'API Key' is chosen. A red box highlights the 'Key' field containing 'app-id' and the 'Value' field containing '658888dd7bcb40db1420b538'. The 'Add to' dropdown is set to 'Header'. The bottom of the screen shows the Postman footer with various icons and links.

2. We take our base URL (<https://dummyapi.io/data/v1/>) according to the documentation and paste it into a new request. Choose the GET method. Send the request.

The screenshot shows the Postman interface with a red arrow pointing from the text below to the 'Send' button. The 'Headers' tab is selected, and the 'Body' tab is visible below it. The response status is 404 Not Found.

3. We receive a 404 status code in the response. This means that the URL is not recognized and the server cannot find the requested resource. Go to the documentation and fix the error.

To get a list of users, we need to add /user at the end of the URL. We get a successful status code 200. Rename the query to User list.

The screenshot shows the Postman interface with a red arrow pointing from the text below to the 'Send' button. The 'Headers' tab is selected, and the 'Body' tab is visible below it. The response status is 200 OK.

4. Create a GET request without authorization. The name of the new request is User list (no auth). Paste the URL: <https://dummyapi.io/data/v1/user>. Click the Send button. We receive a 403 status error code - the client is not authorized.

Test Workspace

DUMMYAPI / User list (no auth)

GET https://dummyapi.io/data/v1/user

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Type No Auth

This request does not use any authorization. Learn more about [authorization](#)

Status: 403 Forbidden Time: 190 ms Size: 616 B Save as example

Pretty Raw Preview Visualize JSON

1 {  
2 "error": "APP\_ID\_MISSING"  
3 }

**5. Check the pagination conditions. Create a new query and name it Pagination. We use the GET method. Paste the URL from the example <https://dummyapi.io/data/v1/user?page=1&limit=10> in the documentation and send the request.**

Test Workspace

DUMMYAPI / Pagination

GET https://dummyapi.io/data/v1/user?page=1&limit=10

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

Query Params

Key	Value	Description	Bulk Edit
page	1		
limit	10		

Status: 200 OK Time: 339 ms Size: 2.22 KB Save as example

Pretty Raw Preview Visualize JSON

66 [ 67 { 68 "id": "60d0fe4f5311236168a109dd", 69 "title": "mr", 70 "firstName": "Miguel", 71 "lastName": "Lima", 72 "picture": "https://randomuser.me/api/portraits/med/men/31.jpg" 73 }, 74 { 75 "total": 99, 76 "page": 1, 77 "limit": 10 }

**6. Check the upper limit. According to the documentation, this value is 50. I get a 200 status code. The server response time and weight have changed as expected.**

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red box highlights the URL field of a GET request to 'https://dummyapi.io/data/v1/user?page=1&limit=50'. Below the URL, the 'Query Params' section shows 'page' set to 1 and 'limit' set to 50. A red arrow points from the 'Send' button to the status bar at the bottom, which displays 'Status: 200 OK'.

Body

```
339 |   "id": "60d0fe4f5311236160a10a2d",
340 |   "title": "mrs",
341 |   "firstName": "Emilie",
342 |   "lastName": "Lambert",
343 |   "picture": "https://randomuser.me/api/portraits/men/93.jpg"
344 |
345 | ],
346 | "total": 99,
347 | "page": 1,
348 | "limit": 50
349 |
350 }
```

## 7. Create a new user using the POST method.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red circle highlights the 'POST Create user' item in the collection tree. A red box highlights the URL field of a POST request to 'https://dummyapi.io/data/v1/user/create'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "firstName": "Alex",
3   "lastName": "Golovko",
4   "email": "domofob133@ubinert.com"
5 }
```

A red arrow points from the 'Send' button to the status bar at the bottom, which displays 'Status: 200 OK'.

Body

```
1 {
2   "id": "658ffb2315ef7a04d6f48bef",
3   "firstName": "Alex",
4   "lastName": "Golovko",
5   "email": "domofob133@ubinert.com",
6   "registerDate": "2023-12-30T11:12:35.487Z",
7   "updatedDate": "2023-12-30T11:12:35.487Z"
8 }
```

The user has been successfully created. His Id: **658ffb2315ef7a04d6f48bef**

8. Modify a newly created user. Change all the data. The email remains the same.

The screenshot shows the Postman interface with a collection named "Test Workspace". A red arrow points from the "PUT Update User" entry in the left sidebar to the "Send" button in the top right of the main panel. The main panel displays a successful response with status 200 OK.

```

PUT https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef
{
  "id": "658ffb2315ef7a04d6f48bef",
  "firstName": "AlexTest",
  "lastName": "Berko",
  "email": "domofob133@ubinert.com",
  "phone": "4578658444",
  "registerDate": "2023-12-30T11:12:35.487Z",
  "updatedDate": "2023-12-30T11:23:07.502Z"
}

```

## 9. Check if the user's data has been updated.

The screenshot shows the Postman interface with the same "Test Workspace" collection. A red arrow points from the "GET Check User after update" entry in the left sidebar to the "Send" button in the top right. The main panel shows a successful response with status 200 OK.

```

{
  "id": "658ffb2315ef7a04d6f48bef",
  "firstName": "AlexTest",
  "lastName": "Berko",
  "email": "domofob133@ubinert.com",
  "phone": "4578658444",
  "registerDate": "2023-12-30T11:12:35.487Z",
  "updatedDate": "2023-12-30T11:23:07.502Z"
}

```

## 10. We are going to undo the user's first post.

The screenshot shows the Postman interface with the "Test Workspace" collection. A red arrow points from the "POST Create Post" entry in the left sidebar to the "Send" button in the top right. The main panel shows a successful response with status 200 OK.

```

POST https://dummyapi.io/data/v1/post/create
{
  "text": "Hello",
  "image": "https://randomuser.me/api/portraits/women/58.jpg",
  "likes": 1000,
  "tags": "manager",
  "owner": "658ffb2315ef7a04d6f48bef"
}

```

Pretty

Raw

Preview

Visualize

JSON



```

1  [
2    "id": "6590097015ef7a37ecf48d9c",
3    "image": "https://randomuser.me/api/portraits/women/58.jpg",
4    "likes": 1000,
5    "link": "",
6    "tags": [
7      "manager"
8    ],
9    "text": "Hello",
10   "publishDate": "2023-12-30T12:13:36.457Z",
11   "updatedDate": "2023-12-30T12:13:36.457Z",
12   "owner": {
13     "id": "658ffb2315ef7a04d6f48bef",
14     "firstName": "AlexTest",
15     "lastName": "Berko"
16   }
17 ]

```

The post was successfully published. Post ID: **6590097015ef7a37ecf48d9c**

**10. We check if our post was created accurately. To do this, we look at all the posts of this user.**

The screenshot shows the Postman interface with the following details:

- Collection:** Test Workspace
- Request:**
  - Method: GET
  - URL: <https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef/post>
  - Params tab (empty)
  - Headers tab (empty)
  - Body tab (empty)
  - Tests tab (empty)
  - Settings tab (empty)
- Response:**
  - Status: 200 OK
  - Time: 290 ms
  - Size: 1.03 KB
  - Save as example
- Preview:** Shows the JSON response for the post with ID 6590097015ef7a37ecf48d9c.

**11. Get list of posts sorted by creation date.**

Home Workspaces API Network

Test Workspace New Import GET Paginate POST Create PUT Update GET Check POST Create GET User list GET Check + No Environment

DUMMYAPI / Check Posts List

GET https://dummyapi.io/data/v1/post

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (19) Test Results Status: 200 OK Time: 267 ms Size: 8.5 KB Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "data": [
3     {
4       "id": "6590097015ef7a37ecf48d9c",
5       "image": "https://randomuser.me/api/portraits/women/58.jpg",
6       "likes": 1000,
7       "tags": [
8         "manager"
9       ],
10      "text": "Hello",
11      "publishDate": "2023-12-30T12:13:36.457Z",
12      "updatedDate": "2023-12-30T12:13:36.457Z",
13      "owner": {
14        "id": "658ffb2315ef7a04d6f48bef",
15        "firstName": "AlexTest",
16      }
17    }
18  ]
19}
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

## 12. Create a comment to the post.

Home Workspaces API Network

Test Workspace New Import POST Create Comment GET Check Posts List + No Environment

DUMMYAPI / Create Comment

POST https://dummyapi.io/data/v1/comment/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Body

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Beautify

```
1 {
2   "message": "comment",
3   "owner": "658ffb2315ef7a04d6f48bef",
4   "post": "6590097015ef7a37ecf48d9c"
5 }
```

Body Cookies Headers (19) Test Results Status: 200 OK Time: 288 ms Size: 923 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "65900de9cb66f2a9258fcc7e",
3   "message": "comment",
4   "owner": {
5     "id": "658ffb2315ef7a04d6f48bef",
6     "firstName": "AlexTest",
7     "lastName": "Berko"
8   },
9   "post": "6590097015ef7a37ecf48d9c",
10  "publishDate": "2023-12-30T12:32:41.768Z"
11}
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

Comment successfully created. Comment ID: **65900de9cb66f2a9258fcc7e**

Home Workspaces API Network

Test Workspace New Import POST Create Comment GET Check comments by User

DUMMYAPI / Check comments by User

GET https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef/comment

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (19) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
  "data": [
    {
      "id": "65900de9cb66f2a9258fcc7e",
      "message": "comment",
      "owner": {
        "id": "658ffb2315ef7a04d6f48bef",
        "firstName": "AlexTest",
        "lastName": "Berko"
      },
      "post": "6590097015ef7a37ecf48d9c",
      "publishDate": "2023-12-30T12:32:41.768Z"
    }
  ],
  "total": 1,
  "page": 0,
  "limit": 20
}
```

Status: 200 OK Time: 256 ms Size: 964 B Save as example

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red circle highlights the 'POST Check comments by User' item in the collection tree. The main panel displays a successful GET request to 'https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef/comment'. The response status is 200 OK, and the response body is a JSON object containing one comment entry.

### 13. Delete the user's comment.

Home Workspaces API Network

Test Workspace New Import POST Create Comment GET Check comments by User DEL Delete comment

DUMMYAPI / Delete comment

DELETE https://dummyapi.io/data/v1/comment/65900de9cb66f2a9258fcc7e

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (19) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
  "id": "65900de9cb66f2a9258fcc7e"
}
```

Status: 200 OK Time: 231 ms Size: 739 B Save as example

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

The screenshot shows the Postman interface with the same 'Test Workspace' collection. A red circle highlights the 'DEL Delete comment' item in the collection tree. The main panel displays a successful DELETE request to 'https://dummyapi.io/data/v1/comment/65900de9cb66f2a9258fcc7e'. The response status is 200 OK, and the response body is a JSON object with a single comment entry.

### 14. Check if the comment was really deleted.

Home Workspaces API Network

Test Workspace New Import POST Create Comment GET Check comments by User DEL Delete comment GET Check comment after delete + - No Environment

Collections Environments History

DUMMYAPI / Check comment after delete

GET https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef/comment

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies Headers (19) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 242 ms Size: 751 B Save as example

```
1 {  
2   "data": [],  
3   "total": 0,  
4   "page": 0,  
5   "limit": 20  
6 }
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

15. Remove the User from the system.

Home Workspaces API Network

Test Workspace New Import POST Create Comment GET Check comment DEL Delete comment GET Check comment DEL Delete User + - No Environment

Collections Environments History

DUMMYAPI / Delete User

DELETE https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies Headers (19) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 233 ms Size: 739 B Save as example

```
1 {  
2   "id": "658ffb2315ef7a04d6f48bef"  
3 }
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

16. Check if the user has really deleted.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red box highlights the URL in the request bar: `https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef`. A red arrow points from the status bar at the bottom right to the response body, which displays the JSON object: 

```
1 {  
2   "error": "RESOURCE_NOT_FOUND"  
3 }
```

.

## 17. Create a new environment for QA.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red box highlights the URL in the request bar: `https://dummyapi.io/data/v1/user/658ffb2315ef7a04d6f48bef`. The environment dropdown on the right is set to 'QA', indicated by a red box around the 'QA' button.

## 18. Drag the data that we will work with in the future (email, user Id) into the environment.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red box highlights the 'Tests' tab in the request settings. A code block in the tests section is highlighted with a red box: 

```
1 var jsonData = pm.response.json();  
2 pm.environment.set("email",jsonData.email);  
3 pm.environment.set("userId",jsonData.id);
```

**19. To see if our snippet is working correctly, go to the Body section and enter a new email domofob13344@ubinert.com**

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A POST request to 'https://dummyapi.io/data/v1/user/create' is selected. The 'Body' tab is active, showing a JSON payload with a red box around the 'email' field containing 'domofob13344@ubinert.com'. The 'Send' button is highlighted with a red arrow. Below the request, the response status is 'Status: 200 OK' and the response body is displayed in a red box, showing a newly created user with an id of '65901d8415ef7a2f9df48edd'.

**User with new email successfully created. A new id has been provided: 65901d8415ef7a2f9df48edd**

**20. Go to Environment and check if the new user's data has been pulled into the QA environment.**

The screenshot shows the Postman interface with the 'QA' environment selected. In the 'Variables' section, there is a red box around a table showing three variables: 'test' (Initial value: qa, Current value: qa), 'email' (Initial value: domofob13344@ubinert.com, Current value: domofob13344@ubinert.com), and 'userId' (Initial value: 65901d8415ef7a2f9df48edd, Current value: 65901d8415ef7a2f9df48edd). The 'Edit' button is highlighted with a red arrow. Below the variables, the 'Globals' section is shown.

**The data has been successfully created.**

**21. Let's try changing the email to domofob13350@ubinert.com again and make sure that the environment data is updated as well.**

The screenshot shows the Postman interface with a successful API call. The 'Body' tab of the request shows the JSON payload with an email field. The response body shows a new user ID '659021e215ef7a188df49019'.

User with new email successfully created. A new id has been provided: **659021e215ef7a188df49019**

The screenshot shows the Postman interface with the QA environment selected. The variables panel shows 'email' set to 'domofob13350@ubinert.com' and 'userId' set to '659021e215ef7a188df49019'.

The new email and Id have been successfully updated in the QA environment.

## 22. The same variables need to be set in two more requests:

- Create Post
- Create Comment

### 22.1. Create variables for a new post.

The screenshot shows the Postman interface with the 'Test Workspace' selected. In the left sidebar, under the 'DUMMYAPI' collection, the 'POST Create Post' request is highlighted with a red circle. In the main panel, the 'Tests' tab is selected, and the following JavaScript code is visible:

```
1 var jsonData = pm.response.json();
2 pm.environment.set("postId",jsonData.id);
```

**22.2. Paste the new user ID (659021e215ef7a188df49019) in the Body section and click Send.**

The screenshot shows the Postman interface with the 'Test Workspace' selected. The 'POST Create Post' request is highlighted with a red circle. The 'Body' tab is selected, and the JSON body is set to:

```
1 {
2   "text": "Hello",
3   "image": "https://randomuser.me/api/portraits/women/58.jpg",
4   "likes": 1000,
5   "tags": "manager",
6   "owner": "659021e215ef7a188df49019"
7 }
```

A red arrow points from the 'Send' button at the top right to the 'owner' field in the JSON body. Another red arrow points from the 'owner' field in the JSON body to the 'id' field in the response body below.

We get a new post ID: **65902c3515ef7a2980f491ad**

**22.3. Check if a new post ID has been added to the QA environment.**

Screenshot of Postman interface showing the 'Create Post' request in the QA environment. The 'Variables' section shows a variable named 'postId' with the value '65902c3515ef7a2980f491ad' highlighted with a red box.

**The new postId was successfully updated in the QA environment.**

## 22.4. Create variables for a new comment.

Screenshot of Postman interface showing the 'Create Comment' request. The 'Tests' tab is selected, containing a JavaScript test script:

```
1 var jsonData = pm.response.json();
2 pm.environment.set("commentId",jsonData.id);
```

**22.5. Paste the new user ID (659021e215ef7a188df49019) and new post ID (65902c3515ef7a2980f491ad) in the Body section and click Send.**

The screenshot shows the Postman interface with a collection named "Test Workspace". A red circle highlights the "POST Create Comment" request. The response body is displayed in JSON format, with the "id" field highlighted by a red box. A red arrow points from the highlighted "id" value in the response to the same value in the "commentId" variable section of the "QA" environment variables.

We get a new comment ID: **6590326a15ef7a6c85f49330**

## 22.6. Check if a new comment ID has been added to the QA environment.

The screenshot shows the Postman interface with the "QA" environment selected. A red circle highlights the "commentId" variable in the "Variables" section. The "Initial value" and "Current value" are both set to "6590326a15ef7a6c85f49330". A red box highlights this value. A red arrow points from the highlighted "commentId" value in the "Variables" section to the same value in the "commentId" variable section of the "QA" environment variables.

**23. We created a collection with variables in the QA environment. Now the main task will be to go through the collection of requests and replace them with a variable format that will allow us to automate tests in the future. In order to no longer enter them manually in the future.**

### 23.1. In the Put method, we entered the User ID manually. We need to change this.

Test Workspace

New Import POST Create user POST Create Post POST Create Comment PUT Update User + QA

PUT DUMMYAPI / Update User https://dummyapi.io/data/v1/user/((userId))

Params Authorization Headers (9) Body

INITIAL CURRENT 659021e215ef7a188d f49019 SCOPE Environment

1 {  
2 "firstName": "AlexTest",  
3 "lastName": "Berko",  
4 "phone": "4578658444"  
5 }

Response

Click Send to get a response

Online Find and replace Console

Postbot Runner Start Proxy Cookies Trash

## We make the same replacement in all other requests:

Test Workspace

New Import POST Create user POST Create Post POST Create Comment PUT Update User GET Check User + QA

GET DUMMYAPI / Check User after update https://dummyapi.io/data/v1/user/((userId))

Params Authorization Headers (9) Body

INITIAL CURRENT 659021e215ef7a188d f49019 SCOPE Environment

1 {  
2 "firstName": "AlexTest",  
3 "lastName": "Berko",  
4 "phone": "4578658444"  
5 }

Response

Click Send to get a response

Online Find and replace Console

Postbot Runner Start Proxy Cookies Trash

Test Workspace

New Import GET Check | DEL Delete | GET Check | DEL Delete | GET Check | GET User list | GET User list POST Create | + QA

DUMMYAPI / Create Post

POST https://dummyapi.io/data/v1/post/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body (JSON)

```
1 {
2   "text": "Hello",
3   "image": "https://randomuser.me/api/portraits/women/58.jpg",
4   "likes": 1000,
5   "tags": "manager",
6   "owner": "#userId"
7 }
```

Status: 200 OK Time: 254 ms Size: 1 KB Save as example

Body Cookies Headers (19) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1 {"id": "659149e8d9001734e4210870", "image": "https://randomuser.me/api/portraits/women/58.jpg", "likes": 1000, "link": "", "tags": ["manager"], "text": "Hello", "publishDate": "2023-12-31T11:00:56.102Z", "updatedDate": "2023-12-31T11:00:56.102Z", "owner": {"id": "65903dfb15ef7a832ff49404", "firstName": "Alex", "lastName": "Golovko"}}
```

Test Workspace

New Import POST Create u | POST Create P | POST Create C | QA | PUT Update U | GET Check Us | GET Check Po | + QA

DUMMYAPI / Check Post

GET https://dummyapi.io/data/v1/user/((userId))/post

Params Authorization Headers (7) Body

Body (JSON)

userId

INITIAL

CURRENT 659021e215ef7a188d f49019

SCOPE Environment

Response

Click Send to get a response

Test Workspace

New Import

POST DUMMYAPI / Create Comment

https://dummyapi.io/data/v1/comment/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body:

```
1 {
2   "message": "comment",
3   "owner": "{{userId}}",
4   "post": "{{postId}}"
5 }
```

Body Cookies Headers (19) Test Results (1/1)

Status: 200 OK Time: 268 ms Size: 921 B Save as example

POST Create Comment

GET Check comments by User

DEL Delete comment

GET Check comment after delete

DEL Delete User

GET Check User after delete

PetStore-Pet

spoonacular API Slava's fork

Test Workspace

New Import

GET DUMMYAPI / Check comments by User

https://dummyapi.io/data/v1/user/{{userId}}/comment

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params:

Key	Value	Description
Key	Value	Description

Response:

Click Send to get a response

Test Workspace

New Import > POST Create PUT Update GET Check GET Check GET Check DEL Delete > + QA Save Send

DUMMYAPI / Delete comment

DELETE https://dummyapi.io/data/v1/comment/{{commentId}}

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response

Click Send to get a response



Postbot Runner Start Proxy Cookies Trash ?

Online Find and replace Console

Test Workspace

New Import > POST Create PUT Update GET Check GET Check GET Check DEL Delete > + QA Save Send

DUMMYAPI / Check comment after delete

GET https://dummyapi.io/data/v1/user/{{userId}}/comment

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response

Click Send to get a response



Postbot Runner Start Proxy Cookies Trash ?

Online Find and replace Console

Test Workspace

New Import > POST Create PUT Update GET Check GET Check GET Check DEL Delete > + QA Save Send

DUMMYAPI / Check comment after delete

GET https://dummyapi.io/data/v1/user/{{userId}}/comment

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response

Click Send to get a response



Postbot Runner Start Proxy Cookies Trash ?

Online Find and replace Console

Home Workspaces API Network

Test Workspace

New Import

PUT Update GET Check GET Check GET Check DEL Delete GET Check DEL Delete

HTTP DUMMYAPI / Delete User

DELETE https://dummyapi.io/data/v1/user/((userId))

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response

Click Send to get a response



Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

Test Workspace

1\_first-tests DUMMYAPI

- GET User list (no auth)
- GET User list
- GET Pagination
- POST Create user
- PUT Update User
- GET Check User after update
- POST Create Post
- GET Check Post
- GET Check Posts List
- POST Create Comment
- GET Check comments by User
- DEL Delete comment
- GET Check comment after delete
- DEL Delete User
- GET Check User after delete

PetStore-Pet spoonacular API Slava's fork

Home Workspaces API Network

Test Workspace

New Import

PUT Update GET Check GET Check GET Check DEL Delete GET Check DEL Delete GET Check

HTTP DUMMYAPI / Check User after delete

GET https://dummyapi.io/data/v1/user/((userId))

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

Body Options: none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Response

Click Send to get a response



Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

Test Workspace

1\_first-tests DUMMYAPI

- GET User list (no auth)
- GET User list
- GET Pagination
- POST Create user
- PUT Update User
- GET Check User after update
- POST Create Post
- GET Check Post
- GET Check Posts List
- POST Create Comment
- GET Check comments by User
- DEL Delete comment
- GET Check comment after delete
- DEL Delete User
- GET Check User after delete

PetStore-Pet spoonacular API Slava's fork

Test Workspace

New Import < date GET Check GET Check GET Check DEL Delete GET Check DEL Delete GET Check > + QA

HTTP DUMMYAPI / Check User after delete

GET https://dummyapi.io/data/v1/user/((userId))

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

1

Response

Click Send to get a response

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash ?

**24. Now we check whether, for example, when creating a new User, its new ID automatically appears in subsequent requests. New email domofob13368@ubinert.com**

Test Workspace

New Import POST Create user +

HTTP DUMMYAPI / Create user

POST https://dummyapi.io/data/v1/user/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Beautify

1 {  
2 "firstName": "Alex",  
3 "lastName": "Golovko",  
4 "email": "domofob13368@ubinert.com"  
5 }

Status: 200 OK Time: 482 ms Size: 895 B Save as example

Body Cookies Headers (19) Test Results

Pretty Raw Preview Visualize JSON

1 {  
2 "id": "65903df515ef7a832ff49404",  
3 "firstName": "Alex",  
4 "lastName": "Golovko",  
5 "email": "domofob13368@ubinert.com",  
6 "registerDate": "2023-12-30T15:57:41.657Z",  
7 "updatedDate": "2023-12-30T15:57:41.657Z"  
8 }

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash ?

A truly new Id: **65903df515ef7a832ff49404** was automatically pulled up in the next request:

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A PUT request to 'https://dummyapi.io/data/v1/user/1' is selected. In the 'Body' tab, a JSON object is defined:

```

1: {
2:   "userId": 1,
3:   "firstName": "AlexTest",
4:   "lastName": "Berko",
5:   "phone": "4578658444"
6: }

```

The 'userId' field has a value of 'CURRENT 65903df515ef7a832f 149404'.

## 25. Create tests for these requests.

### 25.1. The first test is that the expected status code is 200. Applies to all tests.

#### 25.2.1. Request User list (no auth). The first test failed. It is expected that the status code is 403, because the user is not authorized.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A GET request to 'https://dummyapi.io/data/v1/user' is selected. In the 'Tests' tab, the following JavaScript code is present:

```

1: pm.test("Status code is 200", function () {
2:   pm.response.to.have.status(200);
3: });

```

The 'Status' field in the results panel shows '403 Forbidden' with a red arrow pointing to it. The 'Test Results' section shows 0/1 failed, and the details show a 'FAIL' message: 'Status code is 200 | Assertion Error: expected response to have status code 200 but got 403'.

#### 25.2.2. Request User list (no auth). The second test passed. It is expected that the status code is 403, because the user is not authorized. We expect the status code to be 403 and get the same response.

The screenshot shows the Postman interface with a red arrow pointing from the test script in the 'Tests' tab to the 'Test Results' section. The test script checks for a 403 status code. The results show a single failed test with the message 'Status code is 403'.

Test Workspace

GET User list (no auth)

```
1 pm.test("Status code is 403", function () {  
2 | pm.response.to.have.status(403);  
3 |});
```

Test Results (1/1)

PASS Status code is 403

## 25.3 Request User list (with auth). Test passed.

The screenshot shows the Postman interface with a red arrow pointing from the test script in the 'Tests' tab to the 'Test Results' section. The test script checks for a 200 status code. The results show a single passed test with the message 'Status code is 200'.

Test Workspace

GET User list

```
1 pm.test("Status code is 200", function () {  
2 | pm.response.to.have.status(200);  
3 |});
```

Test Results (1/1)

PASS Status code is 200

## 25.4 Pagination test passed.

Test Workspace

New Import

GET Pagination

HTTP DUMMYAPI / Pagination

GET https://dummyapi.io/data/v1/user?page=0&limit=10

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

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

Cookies

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

Snippets

Get an environment variable  
Get a global variable  
Get a variable

Body Cookies Headers (19) Test Results (1/1)

Status: 200 OK Time: 305 ms Size: 2.22 KB Save as example

All Passed Skipped Failed C

PASS Status code is 200

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

A screenshot of the Postman application interface. On the left, the 'Test Workspace' sidebar shows a collection named 'DUMMYAPI' with several API endpoints listed, including 'GET Pagination' which is circled in red. The main workspace shows a 'GET Pagination' request with a red box highlighting the 'Tests' tab. Inside the 'Tests' tab, there is a block of JavaScript code used for assertions. Below the request, the 'Test Results' section shows a single green 'PASS' status with the message 'Status code is 200'. A red arrow points from the 'Tests' tab in the request panel down to the 'PASS' status in the results panel.

## 25.5 Create user test passed.

Test Workspace

New Import

GET Pagination

POST Create user

HTTP DUMMYAPI / Create user

POST https://dummyapi.io/data/v1/user/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings

```
1 var jsonData = pm.response.json();  
2 pm.environment.set("email",jsonData.email);  
3 pm.environment.set("userId",jsonData.id);  
4  
5 pm.test("Status code is 200", function () {  
6   pm.response.to.have.status(200);  
7 });
```

Cookies

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

Snippets

Get an environment variable  
Get a global variable  
Get a variable

Body Cookies Headers (19) Test Results (1/1)

Status: 200 OK Time: 210 ms Size: 902 B Save as example

All Passed Skipped Failed C

PASS Status code is 200

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

A screenshot of the Postman application interface. On the left, the 'Test Workspace' sidebar shows a collection named 'DUMMYAPI' with several API endpoints listed, including 'POST Create user' which is circled in red. The main workspace shows a 'POST Create user' request with a red box highlighting the 'Tests' tab. Inside the 'Tests' tab, there is a block of JavaScript code used for assertions. Below the request, the 'Test Results' section shows a single green 'PASS' status with the message 'Status code is 200'. A red arrow points from the 'Tests' tab in the request panel down to the 'PASS' status in the results panel.

## 25.6 Update user test passed.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red arrow points from the 'PUT Update User' test script in the 'Tests' tab to the 'Test Results' section, which displays a single green 'PASS' status with the message 'Status code is 200'.

```
pm.test("Status code is 200", function () {  
  pm.response.to.have.status(200);  
});
```

## 25.7 Check user after update test passed.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red arrow points from the 'GET Check User after update' test script in the 'Tests' tab to the 'Test Results' section, which displays a single green 'PASS' status with the message 'Status code is 200'.

```
pm.test("Status code is 200", function () {  
  pm.response.to.have.status(200);  
});
```

## 25.8 Create post test passed.

The screenshot shows the Postman interface with a 'Test Workspace' selected. A red arrow points from the 'Tests' tab in the 'Create Post' request to the 'Test Results' section, which displays a single green 'PASS' status with the message 'Status code is 200'.

```
1 var jsonData = pm.response.json();
2 pm.environment.set("postId",jsonData.id);
3
4 pm.test("Status code is 200", function () {
5     | pm.response.to.have.status(200);
6 });
```

## 25.8 Check post test passed.

The screenshot shows the Postman interface with a 'Test Workspace' selected. A red arrow points from the 'Tests' tab in the 'Check Post' request to the 'Test Results' section, which displays a single green 'PASS' status with the message 'Status code is 200'.

```
1 pm.test("Status code is 200", function () {
2     | pm.response.to.have.status(200);
3 });
```

## 25.9 Check posts list test passed.

Home Workspaces API Network

Test Workspace New Import POST Create Post GET Check Post GET Check Posts List + QA

HTTP DUMMYAPI / Check Posts List

GET https://dummyapi.io/data/v1/post

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

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about test scripts.

Snippets Get an environment variable Get a global variable Get a variable

Body Cookies Headers (19) Test Results (1/1)

Status: 200 OK Time: 301 ms Size: 8.33 KB Save as example

All Passed Skipped Failed C

PASS Status code is 200

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

A screenshot of the Postman application interface. On the left, the 'Test Workspace' sidebar shows a collection named '1\_first-tests' and a folder named 'DUMMYAPI' containing various API endpoints. The 'GET Check Posts List' endpoint is selected and highlighted with a red oval. In the main workspace, a 'GET' request is made to 'https://dummyapi.io/data/v1/post'. The 'Tests' tab contains a single test script: 'pm.test("Status code is 200", function () { pm.response.to.have.status(200);});'. Below the request, the 'Test Results' section shows a single green 'PASS' status with the message 'Status code is 200'. A red arrow points from the text 'Status code is 200' in the test results back to the corresponding status code in the test script.

## 25.10 Create Comment test passed.

Home Workspaces API Network

Test Workspace New Import POST Create Post GET Check Post GET Check Posts List POST Create Comment + QA

HTTP DUMMYAPI / Create Comment

POST https://dummyapi.io/data/v1/comment/create

Params Authorization Headers (9) Body Pre-request Script Tests Settings

```
1 var jsonData = pm.response.json();  
2 pm.environment.set("commentId",jsonData.id);  
3  
4 pm.test("Status code is 200", function () {  
5   pm.response.to.have.status(200);  
6 });
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about test scripts.

Snippets Get an environment variable Get a global variable Get a variable

Body Cookies Headers (19) Test Results (1/1)

Status: 200 OK Time: 306 ms Size: 923 B Save as example

All Passed Skipped Failed C

PASS Status code is 200

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

A screenshot of the Postman application interface. The 'Test Workspace' sidebar shows a collection named '1\_first-tests' and a folder named 'DUMMYAPI' containing various API endpoints. The 'POST Create Comment' endpoint is selected and highlighted with a red oval. In the main workspace, a 'POST' request is made to 'https://dummyapi.io/data/v1/comment/create'. The 'Tests' tab contains a test script: 'var jsonData = pm.response.json(); pm.environment.set("commentId",jsonData.id); pm.test("Status code is 200", function () { pm.response.to.have.status(200);});'. Below the request, the 'Test Results' section shows a single green 'PASS' status with the message 'Status code is 200'. A red arrow points from the text 'Status code is 200' in the test results back to the corresponding status code in the test script.

## 25.11 Check Comments by user test passed.

Test Workspace

New Import POST Create Post GET Check Post GET Check Posts List POST Create Comment GET Check comment + QA

HTTP DUMMYAPI / Check comments by User

GET https://dummyapi.io/data/v1/user/({userId})/comment

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

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

Body Cookies Headers (19) Test Results (1/1)

Status: 200 OK Time: 271 ms Size: 964 B Save as example

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

Snippets Get an environment variable Get a global variable Get a variable

View Templates

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

A screenshot of the Postman application interface. On the left, the 'Test Workspace' sidebar shows a collection named '1\_first-tests' and a folder named 'DUMMYAPI' containing various API endpoints. A specific endpoint, 'GET Check comments by User', is highlighted with a red oval. The main panel displays a test script in the 'Tests' tab of a request configuration for 'https://dummyapi.io/data/v1/user/({userId})/comment'. The script contains a single assertion: `pm.test("Status code is 200", function () { pm.response.to.have.status(200); });`. Below the request configuration, the 'Test Results' section shows a single green 'PASS' status with the message 'Status code is 200'. A red arrow points from the 'Test Results' text to the 'PASS' status indicator.

## 25.12 Delete Comment test passed.

Test Workspace

New Import POST Create Post GET Check Post GET Check Posts List POST Create Comment GET Check comment DEL Delete comment + QA

HTTP DUMMYAPI / Delete comment

DELETE https://dummyapi.io/data/v1/comment/({commentId})

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

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

Body Cookies Headers (19) Test Results (1/1)

Status: 200 OK Time: 275 ms Size: 743 B Save as example

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

Snippets Get an environment variable Get a global variable Get a variable

View Templates

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

A screenshot of the Postman application interface, similar to the previous one but with a different request configuration. The 'Test Workspace' sidebar shows the same collection and folder structure. The 'DEL Delete comment' endpoint under 'DUMMYAPI' is highlighted with a red oval. The main panel shows a test script in the 'Tests' tab of a request configuration for 'https://dummyapi.io/data/v1/comment/({commentId})'. The script contains a single assertion: `pm.test("Status code is 200", function () { pm.response.to.have.status(200); });`. Below the request configuration, the 'Test Results' section shows a single green 'PASS' status with the message 'Status code is 200'. A red arrow points from the 'Test Results' text to the 'PASS' status indicator.

## 25.13 Check Comment after delete test passed.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red arrow points from the 'Test Results' section of the main panel to the 'Tests' tab in the request configuration. The 'Tests' tab contains the following JavaScript code:

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(200);  
3});
```

The 'Test Results' section shows one test result: 'Status code is 200' with a green 'PASS' status.

## 25.14 Delete User test passed.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red arrow points from the 'Test Results' section of the main panel to the 'Tests' tab in the request configuration. The 'Tests' tab contains the same JavaScript code as the previous screenshot:

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(200);  
3});
```

The 'Test Results' section shows one test result: 'Status code is 200' with a green 'PASS' status.

**25.14.1 Check User after delete request. The first test failed. It is expected that the status code is 404, because the user no longer exists.**

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red arrow points from the text below to the 'Tests' tab of a request labeled 'Check User after delete'. The test script contains:

```
1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200);
3 });
```

The test results show a failure: 'Status code is 200 | Assertion Error: expected response to have status code 200 but got 404'.

**25.14.2 Check User after delete request. The second test passed. It is expected that the status code is 404, because the requested resource does not exist. We expect the status code to be 404 and get the same response.**

The screenshot shows the Postman interface with the same collection 'Test Workspace'. A red arrow points from the text below to the 'Tests' tab of the same request. The test script is identical:

```
1 pm.test("Status code is 404", function () {
2     pm.response.to.have.status(404);
3 });
```

The test results show a pass: 'Status code is 404'.

**26. Check for the presence of a string in a response with a specific content.**

**26.1 Test the Create user request. Check that the response contains a string with user's Id. Test passed.**

The screenshot shows the Postman interface with the 'Test Workspace' selected. In the left sidebar, under the 'DUMMYAPI' collection, the 'POST Create user' endpoint is highlighted with a red circle. The main panel displays a POST request to 'https://dummyapi.io/data/v1/user/create'. The 'Tests' tab is active, showing the following JavaScript code:

```

1 var jsonData = pm.response.json();
2 pm.environment.set("email",jsonData.email);
3 pm.environment.set("userId",jsonData.id);
4
5 pm.test("Status code is 200", function () {
6     pm.response.to.have.status(200);
7 });
8
9 pm.test("Body matches string", function () {
10    pm.expect(pm.response.text()).to.include("id");
11 });

```

A red box highlights the last two lines of the test script. Below the script, the 'Test Results' section shows two green 'PASS' status indicators: 'Status code is 200' and 'Body matches string'. A red arrow points from the text 'Body matches string' to the 'PASS' status indicator.

## 26.2 Test the Create user request. Check that the response contains a string with with arbitrary numbers 111. Test failed. Because there are no such numbers in the request.

This screenshot is similar to the previous one but shows a failed test. The 'Test Results' section now has a single red 'FAIL' status indicator for the 'Body matches string' check. The message next to it reads: 'Body matches string | Assertion Error: expected '{"id":"6591602cd900174899210971","fir...' to include '111''. A red arrow points from the text 'Assertion Error' to the 'FAIL' status indicator.

## 27. Test the Create post request. Check if the header contains the Content-Type. Test passed.

The screenshot shows the Postman interface with a collection named 'Test Workspace'. A red arrow points from the 'Tests' tab in the top navigation bar to the 'Tests' section of the request configuration. The 'Tests' section contains the following JavaScript code:

```
1 var jsonData = pm.response.json();
2 pm.environment.set("postId",jsonData.id);
3
4 pm.test("Status code is 200", function () {
5     | pm.response.to.have.status(200);
6 });
7
8 pm.test("Content-Type is present", function () {
9     | pm.response.to.have.header("Content-Type");
10});
```

The 'Test Results' section shows two green 'PASS' status entries: 'Status code is 200' and 'Content-Type is present'. The status bar at the bottom indicates a 'Status: 200 OK'.

The screenshot shows the Postman interface with the same collection 'Test Workspace'. A red arrow points from the 'Headers' tab in the top navigation bar to the 'Headers' section of the request configuration. The 'Headers' section lists the following headers:

Key	Value
Connection	keep-alive
Content-Length	313
Access-Control-Allow-Origin	*
Cache-Control	private
Content-Type	application/json; charset=utf-8
Etag	W/"139-YI53z4dCaUcz1D4QsamzlcAfDcQ"

## 28. Test the Create post request. Check if the header contains the Cache-Control. Test passed.

The screenshot shows the Postman interface with a collection named "Test Workspace". A red circle highlights the "POST Create Post" item in the list. The main panel displays a POST request to "https://dummyapi.io/data/v1/post/create". The "Tests" tab is selected, showing the following JavaScript code:

```
1 var jsonData = pm.response.json();
2 pm.environment.set("postId",jsonData.id);
3
4 pm.test("Status code is 200", function () {
5     pm.response.to.have.status(200);
6 });
7
8 pm.test("Cache-Control is present", function () {
9     pm.response.to.have.header("Cache-Control");
10});
```

A red box highlights the second test block. Below the tests, the "Test Results" section shows two "PASS" results: "Status code is 200" and "Cache-Control is present". The status bar at the bottom indicates "Status: 200 OK Time: 223 ms Size: 1 KB".

## 29. Test the Create post request. Negative testing. Check if the header contains the AAA. Test failed.

The screenshot shows the Postman interface with the same setup as the previous screenshot, but the second test block has been modified to check for the header "AAA" instead of "Cache-Control".

```
1 var jsonData = pm.response.json();
2 pm.environment.set("postId",jsonData.id);
3
4 pm.test("Status code is 200", function () {
5     pm.response.to.have.status(200);
6 });
7
8 pm.test("Cache-Control is present", function () {
9     pm.response.to.have.header("AAA");
10});
```

A red box highlights the second test block. The "Test Results" section shows one "PASS" result ("Status code is 200") and one "FAIL" result ("Cache-Control is present | AssertionError: expected response to have header with key 'AAA'"). The status bar at the bottom indicates "Status: 200 OK Time: 242 ms Size: 1 KB".

## 30. Make sure the response time is no more than 400ms. Test passed.

Test Workspace

**DUMMYAPI / Pagination**

```

1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4
5 pm.test("Response time is less than 400ms", function () {
6   pm.expect(pm.response.responseTime).to.be.below(400);
7 });
  
```

Body Cookies Headers (19) Test Results (2/2)

All Passed Skipped Failed

PASS Status code is 200  
PASS Response time is less than 400ms

### 31. Run the entire collection of tests.

Test Workspace

**Runner**

- Run manually
- Schedule runs
- Automate runs via CLI

Run configuration

Iterations: 1

Delay: 0 ms

Data: Select File

Persist responses for a session

Advanced settings

Stop run if an error occurs  
 Keep variable values  
 Run collection without using stored cookies  
 Save cookies after collection run

**Run DUMMYAPI**

The 5 tests passed, 13 tests failed.

Screenshot of Postman showing the DUMMYAPI collection. The 'All Tests' section shows 5 passed, 13 failed, and 0 skipped tests. A red circle highlights the 'All Tests' link.

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA	1	5s 553ms	18	233 ms

**Iteration 1**

- GET User list (no auth)** https://dummyapi.io/data/v1/user  
Status code is 403
- GET User list** https://dummyapi.io/data/v1/user  
Status code is 200
- GET Pagination** https://dummyapi.io/data/v1/user?page=1&limit=50  
Status code is 200  
Response time is less than 400ms
- POST Create user** https://dummyapi.io/data/v1/user/create  
Status code is 200 | Assertion Error: expected response to have status code 200 but got 400

**The errors started happening with the Create user request. This is because the email was not changed. It's impossible to create a new user with the same email.**

**32. To avoid manually entering new user data every time, you can use urlencoded. Create random values for First Name, Last Name, and email.**

Screenshot of Postman showing the POST Create user request configuration. The 'Body' tab is selected, showing three form-data fields: email, lastName, and firstName, each with a value of \$randomEmail, \$randomLastName, and \$randomFirstName respectively. A red box highlights the 'Body' tab and the three form-data fields.

**33. Run the test collection again. Passed 18 tests out of 18.**

The screenshot shows the Postman interface with the 'DUMMYAPI' collection selected. The 'Run results' section displays 18 passed tests. A red box highlights the 'Passed (18)' status.

DUMMYAPI - Run results

Ran today at 16:09:16 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA	1	5s 632ms	18	244 ms

All Tests Passed (18) Failed (0) Skipped (0) [View Summary](#)

Iteration 1

GET User list (no auth)  
https://dummyapi.io/data/v1/user  
PASS Status code is 403

GET User list  
https://dummyapi.io/data/v1/user  
PASS Status code is 200

GET Pagination  
https://dummyapi.io/data/v1/user?page=1&limit=50  
PASS Status code is 200

POST Create user  
https://dummyapi.io/data/v1/user/create  
PASS Status code is 200

### 34. We test the system load. Run 10,000 iterations with a delay of 5 seconds. We study the moment when the server will fail.

The screenshot shows the Postman interface with the 'DUMMYAPI' collection selected for a runner. The 'Functional' tab is active, and the 'Run configuration' section is highlighted with a red box. The 'Iterations' field is set to 10000 and the 'Delay' field is set to 5000 ms.

Test Workspace

New Import

DUMMYAPI

Runner

Run order

Deselect All Select All Reset

Functional Performance

Choose how to run your collection

Run manually  
Run this collection in the Collection Runner.

Schedule runs  
Periodically run collection at a specified time on the Postman Cloud.

Automate runs via CLI  
Configure CLI command to run on your build pipeline.

Run configuration

Iterations: 10000

Delay: 5000 ms

Data: Select File

Persist responses for a session ⓘ

Advanced settings

Stop run if an error occurs

Keep variable values ⓘ

Run collection without using stored cookies

Save cookies after collection run ⓘ