Articles

What is an API? video explanation: https://www.youtube.com/watch?v=s7wmiS2mSXY

Article for understanding API documentations:
https://idratherbewriting.com/learnapidoc/docapis_resource_descriptions.html

JSON data types: https://www.geeksforgeeks.org/json-data-types/

Postman documentation: https://learning.postman.com/docs/getting-started/introduction/

PetStore API requests (from the below exercise):
https://www.postman.com/mitshex1/workspace/swagger-petstore/example/16308298-eb91945d-b408-4f75-993e-9fa5c4e83d92

Step-by-step exercise

Documentation of the API used in this exercise can be found at the following link:
https://petstore.swagger.io/

Our example that we will test upon

Let’s look at an example from the Swagger Pet Store API:

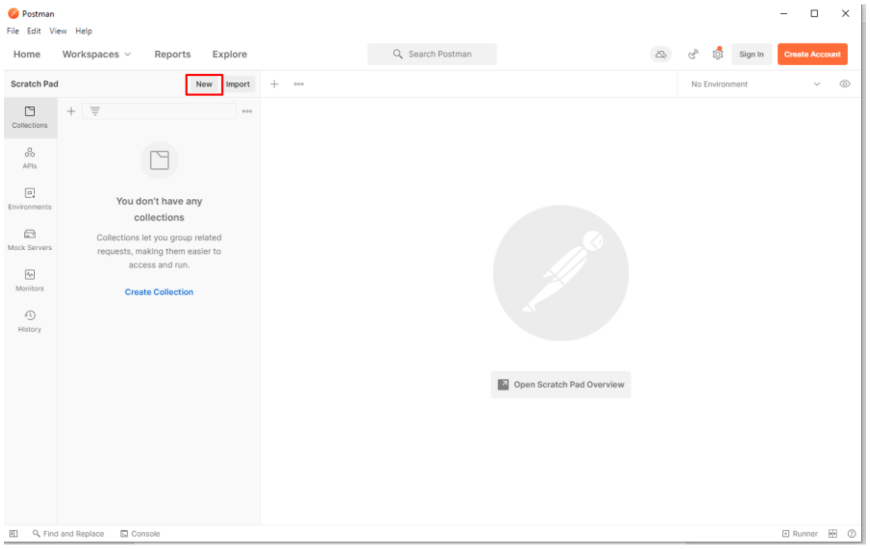
- Sending a GET request to /pet/{petId} would retrieve pets with a specified ID from the database.
- Sending a POST request to /pet would add a new record of a pet in the store.
- Sending a PUT request to /pet would update the attributes of an existing pet, identified by a specified id.
- Sending a DELETE request to /pet/{petId} would delete a specified pet.

So in a nutshell here is what each of these request types map to:

GET	Read or retrieve data
POST	Add new data
PUT	Update data that already exists
DELETE	Remove data

Create a Collection and test

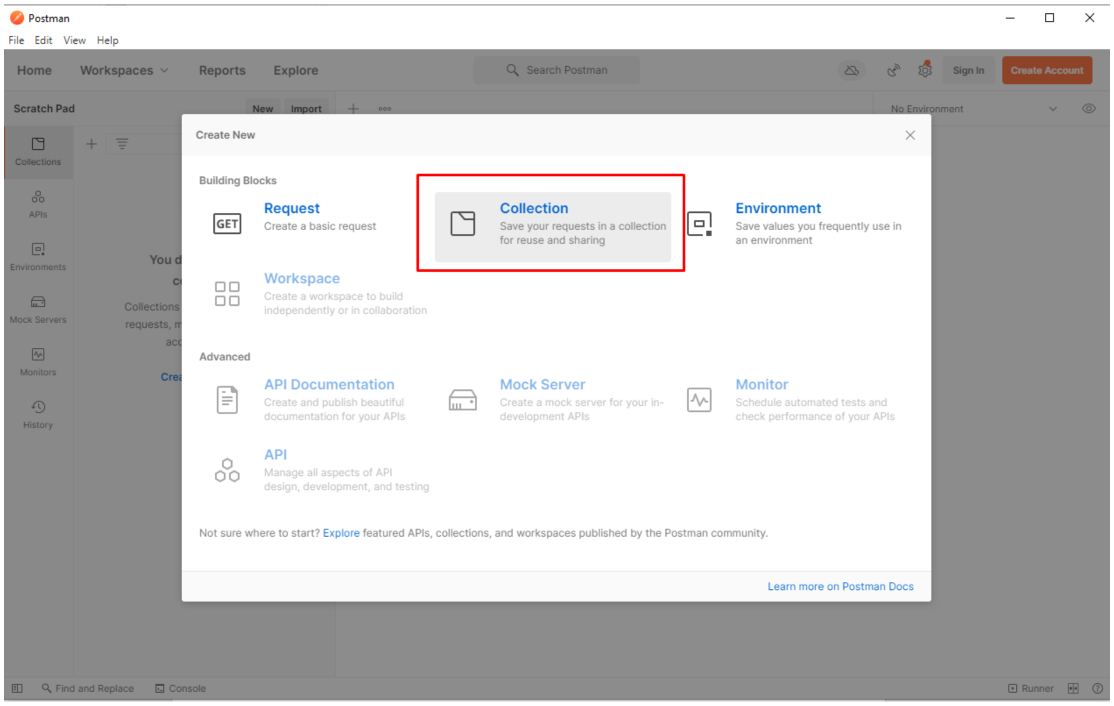
Step 1: In the Collections, click New



You don't have any collections

Collections let you group related requests, making them easier to access and run.

Step 2: Select Collection



Request
Create a basic request

Collection
Save your requests in a collection for reuse and sharing

Environment
Save values you frequently use in an environment

Workspace
Create a workspace to build independently or in collaboration

API Documentation
Create and publish beautiful documentation for your APIs

Mock Server
Create a mock server for your in-development APIs

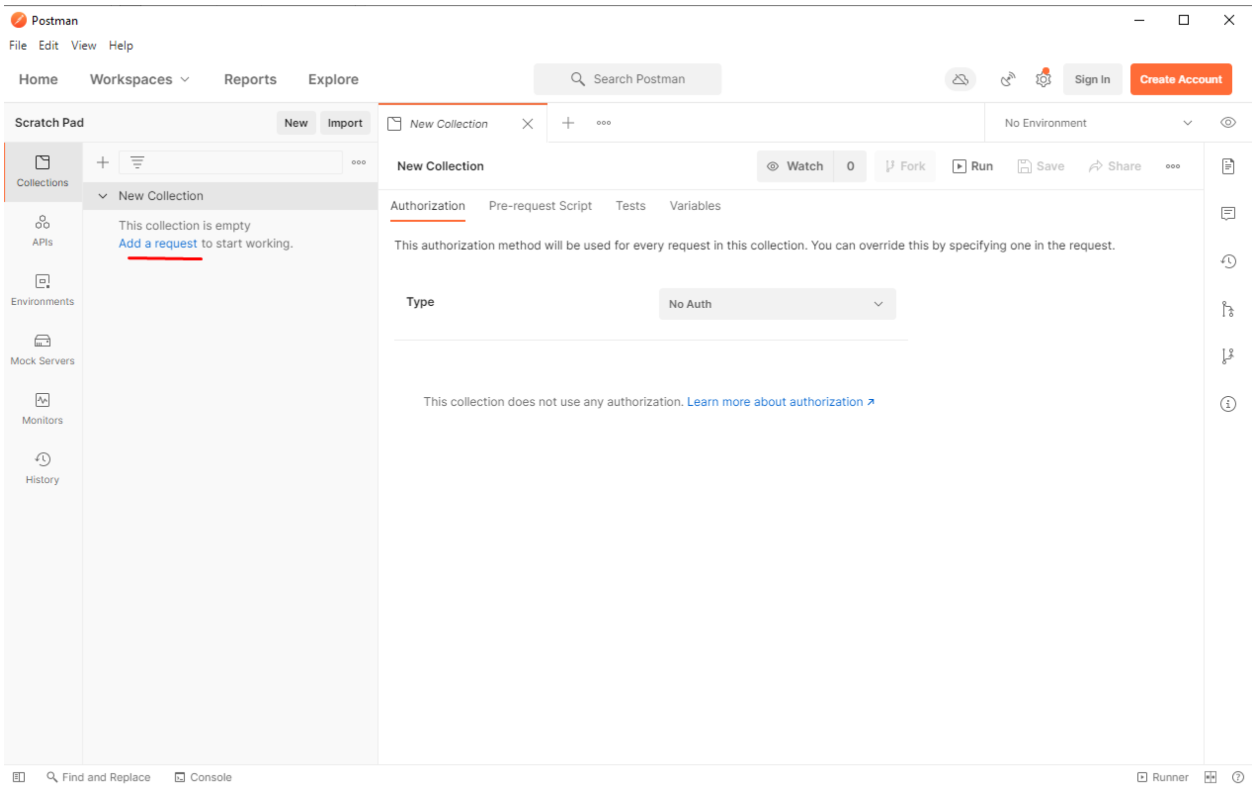
Monitor
Schedule automated tests and check performance of your APIs

API
Manage all aspects of API design, development, and testing

Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.

Learn more on Postman Docs

Step 3: Your collection was created, but it is empty at the moment. Let’s add a request. Click on the ‘Add a request’ blue text



New Collection

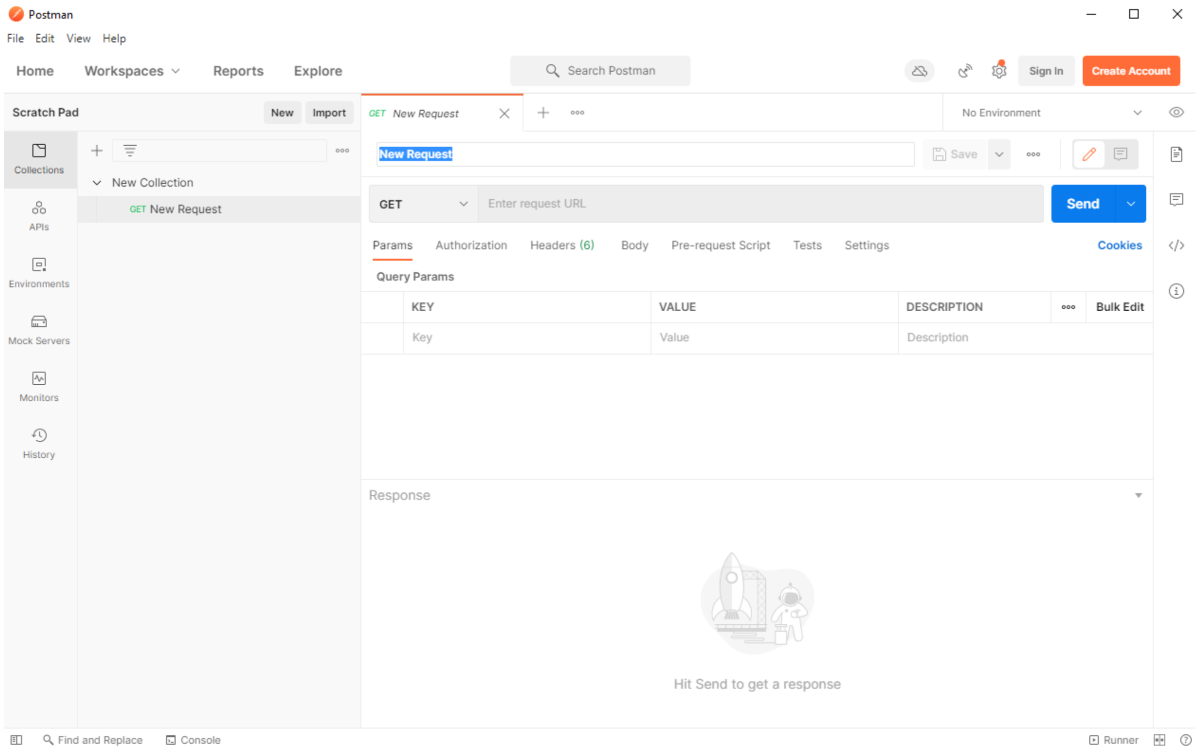
This collection is empty
Add a request to start working.

This authorization method will be used for every request in this collection. You can override this by specifying one in the request.

Type: No Auth

This collection does not use any authorization. Learn more about authorization ↗

Step 4: A new window was opened with the New Request that we will create. Give it the name ‘addPet’



New Request

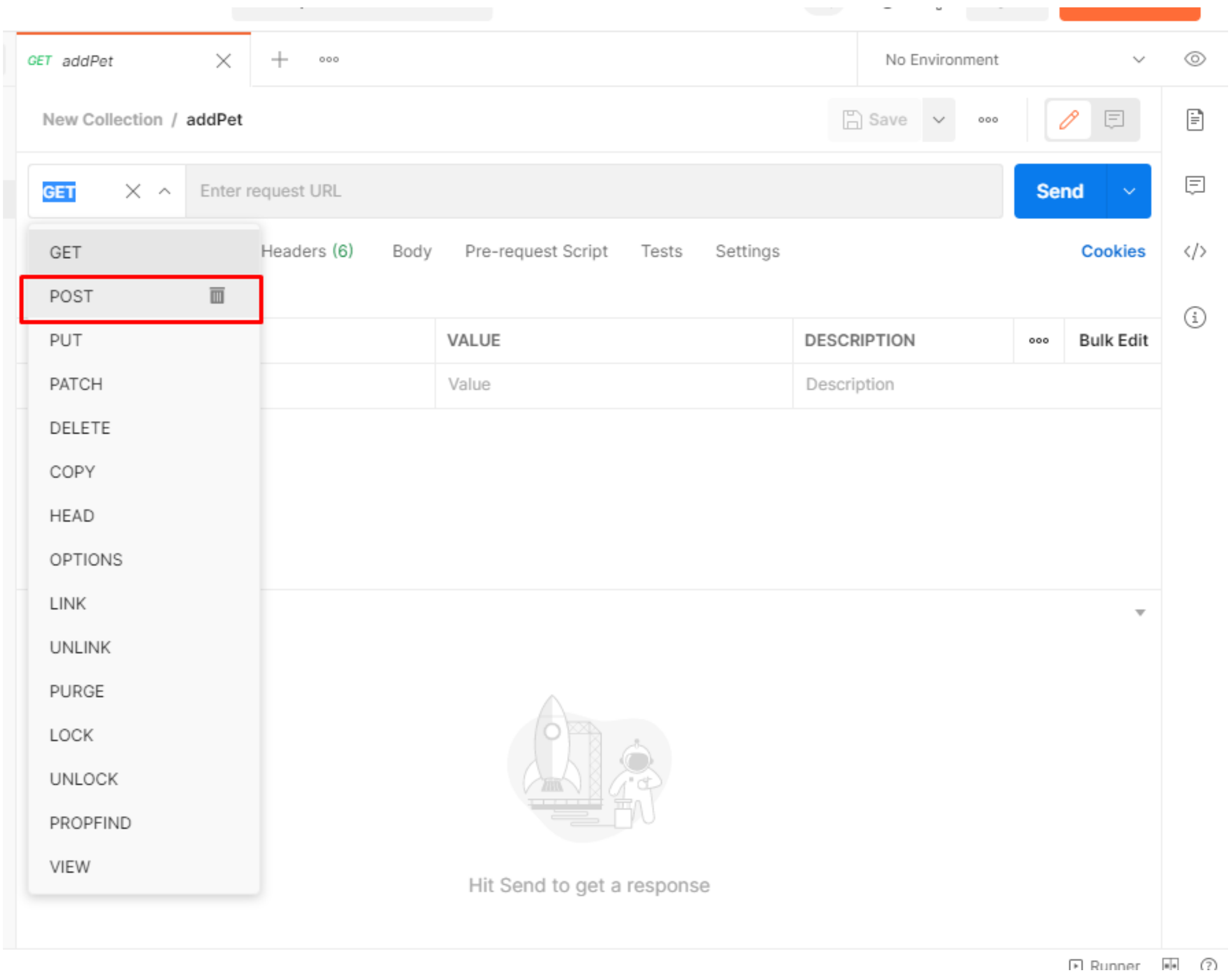
GET Enter request URL

Query Params

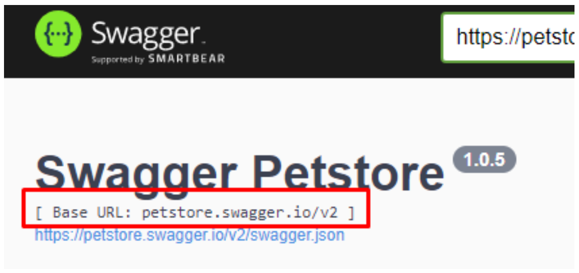
KEY	VALUE	DESCRIPTION
Key	Value	Description

Hit Send to get a response

Step 5: Select the POST method



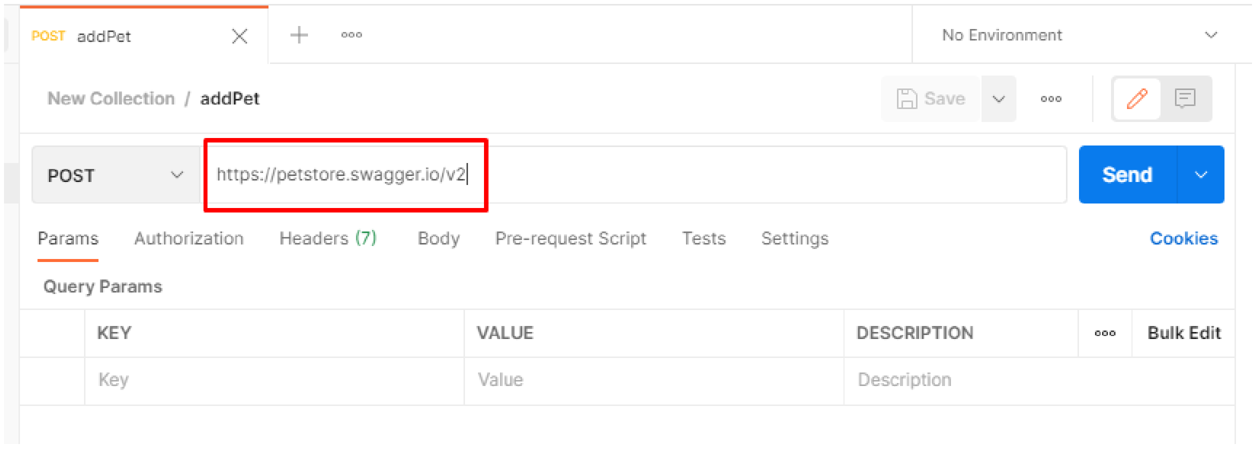
Step 6.1: We need to add the request URL for which we send the requests. Looking over the documentation of the API, the base URL is https://petstore.swagger.io/v2



[Base URL: petstore.swagger.io/v2]

https://petstore.swagger.io/v2/swagger.json

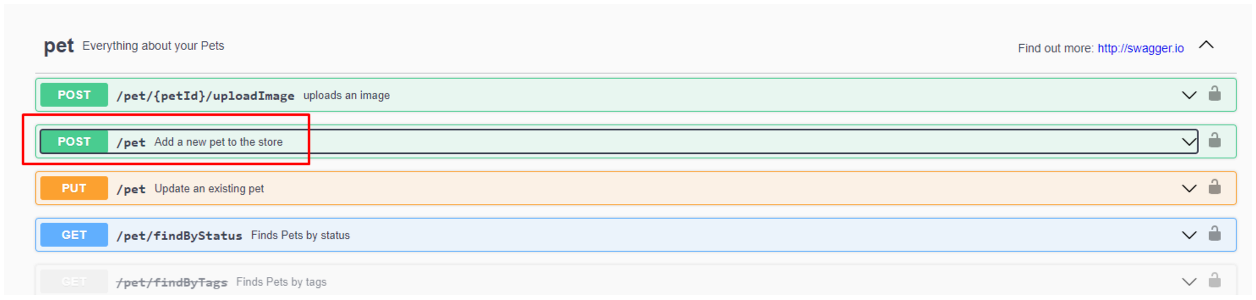
Step 6.2: Take this base URL and add it into the input ‘Enter request URL’ from the ‘addPet’ POST request in Postman



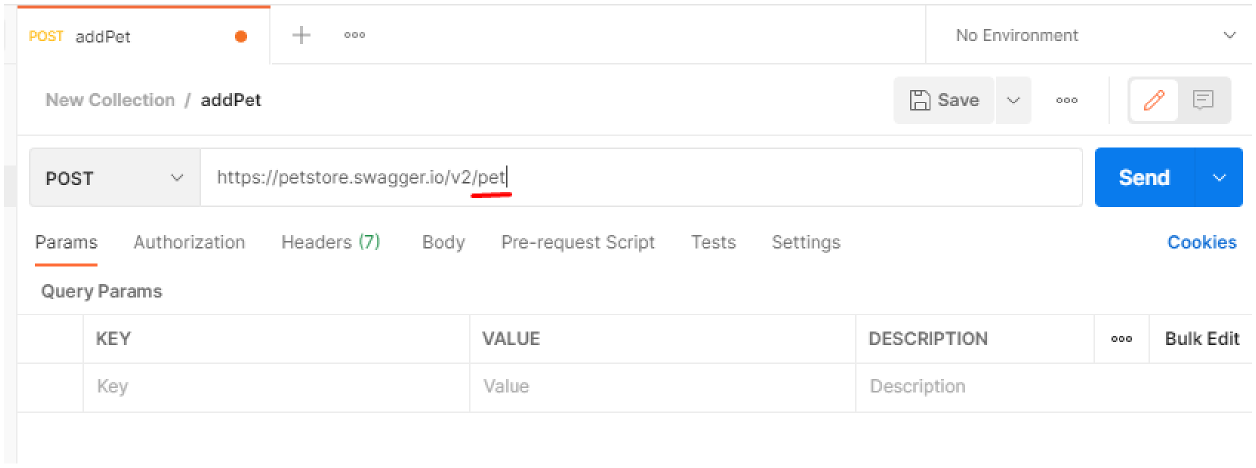
https://petstore.swagger.io/v2

KEY	VALUE	DESCRIPTION
Key	Value	Description

Step 6.3: Go over the documentation of the API and check which is the method for adding a pet. The ‘/pet’ text is the next path that we need to add in the request URL in Postman


pet Everything about your Pets
Find out more: http://swagger.io

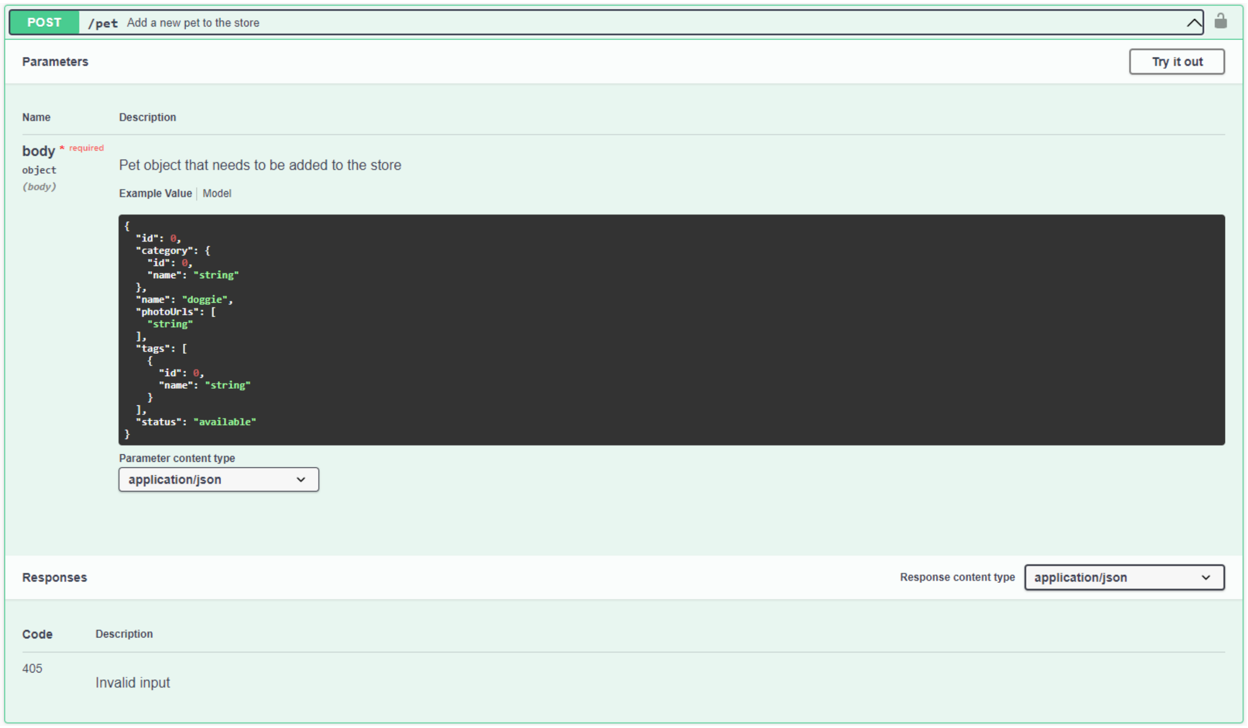
POST /pet/{petId}/uploadImage uploads an image
POST /pet Add a new pet to the store
PUT /pet Update an existing pet
GET /pet/findByStatus Finds Pets by status
GET /pet/findByTags Finds Pets by tags



https://petstore.swagger.io/v2/pet

KEY	VALUE	DESCRIPTION
Key	Value	Description

Step 7: For us to add a pet, we need to send some data about that pet. First we need to check the documentation and see what fields are accepted to be sent. Expand the "POST /pet Add a new pet to the store" from the documentation page of the API



Name	Description
body * required object *(body)*	Pet object that needs to be added to the store

Example Value | Model

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

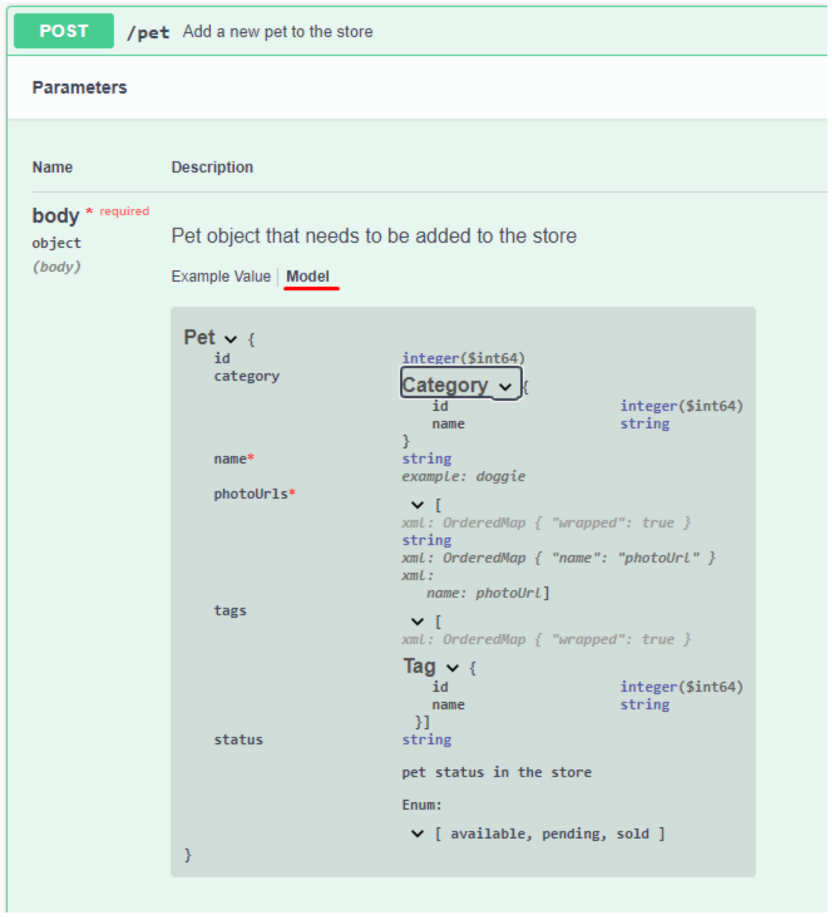
Parameter content type: application/json

Responses — Response content type: application/json

Code	Description
405	Invalid input

Notice that we have Parameters section, which describes what we can send. We need to send a body parameter which is an object data type. See here about JSON data types.

As seen in the image, we have the example value, which includes multiple parameters. If we change to Model, we can see the parameters and the restrictions that should be applied, as well as each data type for the parameters:


POST /pet Add a new pet to the store

Parameters

Name Description

body * required
object
(body)

Pet object that needs to be added to the store

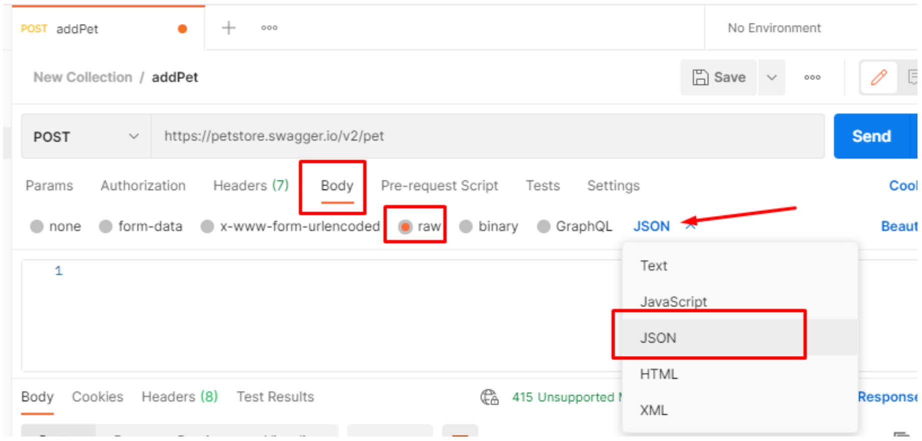
Example Value | Model

```
Pet {
    id                  integer($int64)
    category            Category {
                            id              integer($int64)
                            name            string
                        }
    name*               string
                        example: doggie
    photoUrls*          [
                        xml: OrderedMap { "wrapped": true }
                        string
                        xml: OrderedMap { "name": "photoUrl" }
                        xml:
                          name: photoUrl]
    tags                [
                        xml: OrderedMap { "wrapped": true }
                        Tag {
                            id              integer($int64)
                            name            string
                         }]
    status              string

                        pet status in the store

                        Enum:

                        [ available, pending, sold ]
}
```

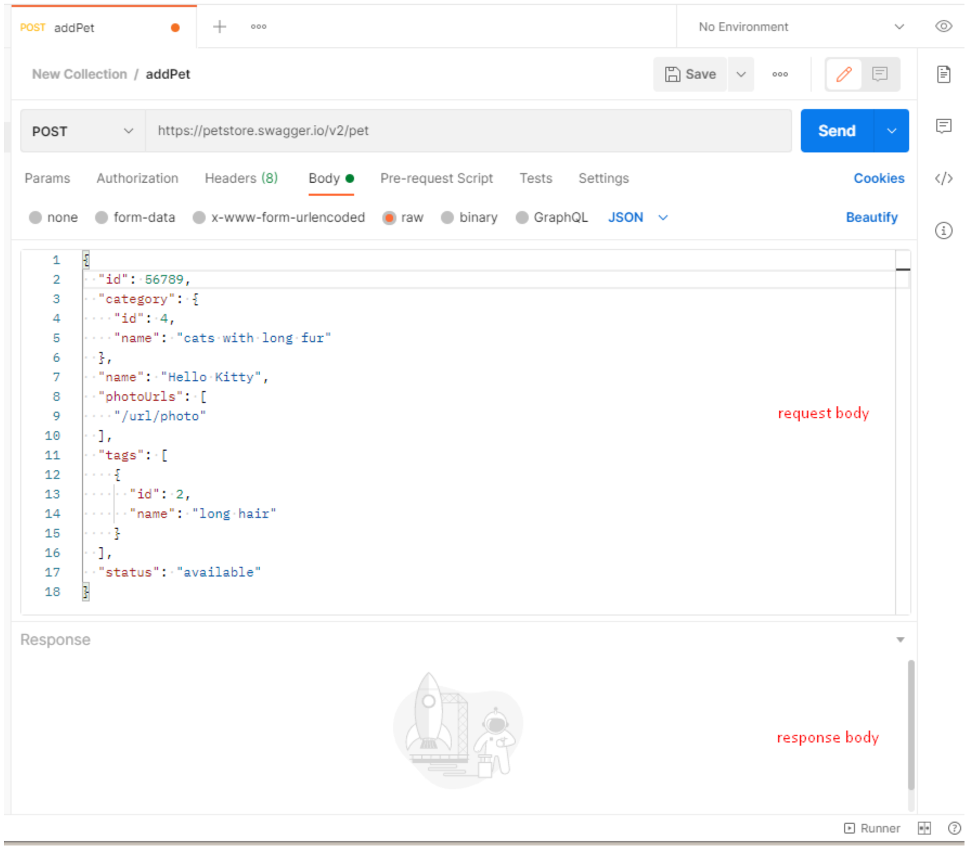
- “id” -> which is an integer, “category” -> which is an object containing two other parameters “id” (integer) and “name” (string), “name” -> which is a string, etc.

Based on these informations we can test the API by giving it invalid data such as a string for “id” or a number for “name”. But first we will send valid data through the API, we can come back later to test the negative scenarios.

Step 8: In Postman, for the 'addPet' request, select the Body window, select the raw option and then select JSON for the request body that we will send

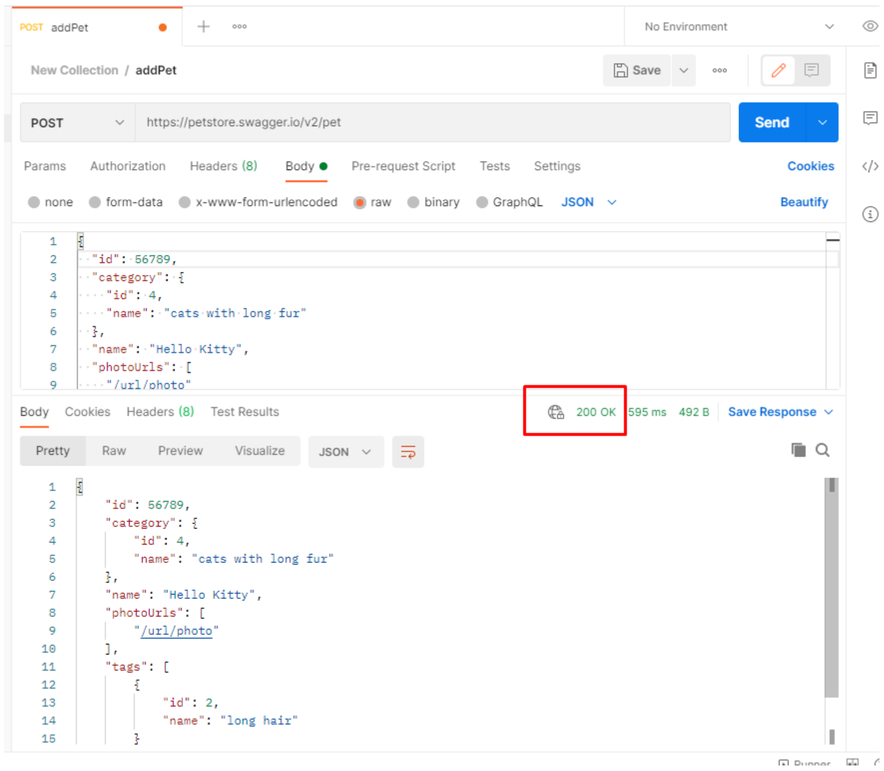


Step 9: Copy the object body from the https://petstore.swagger.io/#/pet/addPet documentation and paste it into the request Body from Postman. You can modify the properties with any values, but most importantly add a unique id.



```
{
  "id": 56789,
  "category": {
    "id": 4,
    "name": "cats with long fur"
  },
  "name": "Hello Kitty",
  "photoUrls": [
    "/url/photo"
  ],
  "tags": [
    {
      "id": 2,
      "name": "long hair"
    }
  ],
  "status": "available"
}
```

Step 10.1: Click on the blue Send button and notice the returned response body and status. The request was send successfully and the pet was added



POST https://petstore.swagger.io/v2/pet

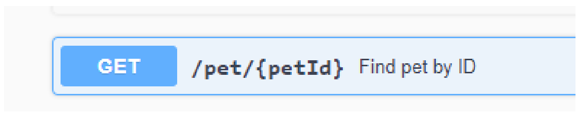
```
{
  "id": 56789,
  "category": {
    "id": 4,
    "name": "cats with long fur"
  },
  "name": "Hello Kitty",
  "photoUrls": [
    "/url/photo"
```

200 OK 595 ms 492 B

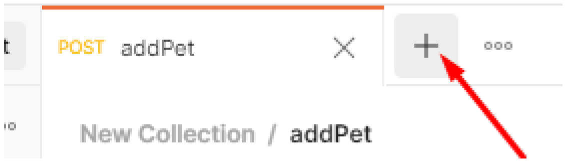
```
{
    "id": 56789,
    "category": {
        "id": 4,
        "name": "cats with long fur"
    },
    "name": "Hello Kitty",
    "photoUrls": [
        "/url/photo"
    ],
    "tags": [
        {
            "id": 2,
            "name": "long hair"
        }
```

Step 10.2: Think about which would be the scenarios we could test for the POST /v2/pet request, by checking the documentation in Swagger and make a list. (check again what was mentioned in step 7)

Step 11: We are not actually checking that the pet was added. Let's add another request in Postman, the GET /pet/{petId} to return the previously created pet. Check the documentation for this method.



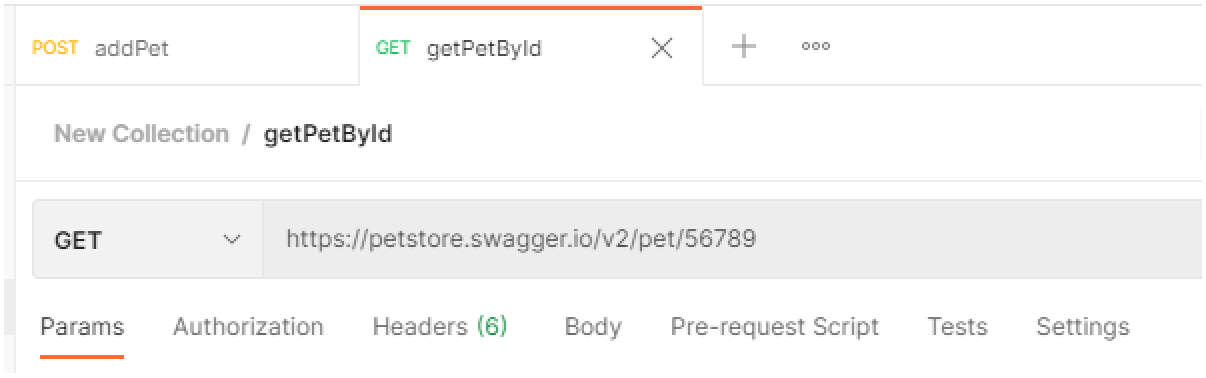
Click on the + right next to the addPet window



Step 12: Add the base URL of the API and the path for the GET pet by Id method

The method has the path /pet/{petId}. {} means that a value need to be added, in our case the petId we just created. In the example above it is `56789`

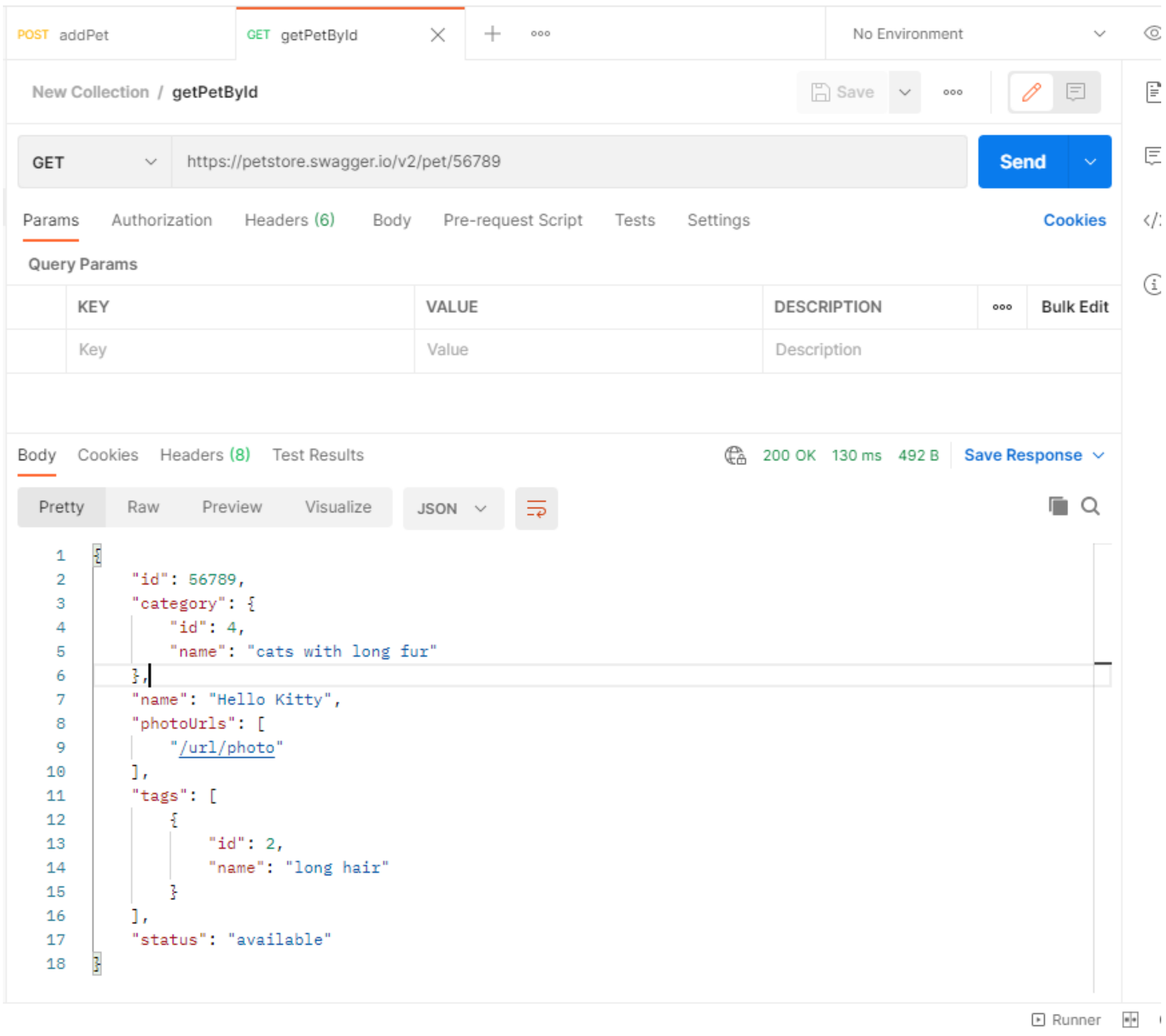
Save the request with the name getPetById.



https://petstore.swagger.io/v2/pet/56789

Step 13.1: Click on the blue Send button. Notice that in the response window of Postman we received the pet details, that we created before.

Additional note: In case “Pet not found” is returned, send the request multiple times. It seems there is a bug with the API.

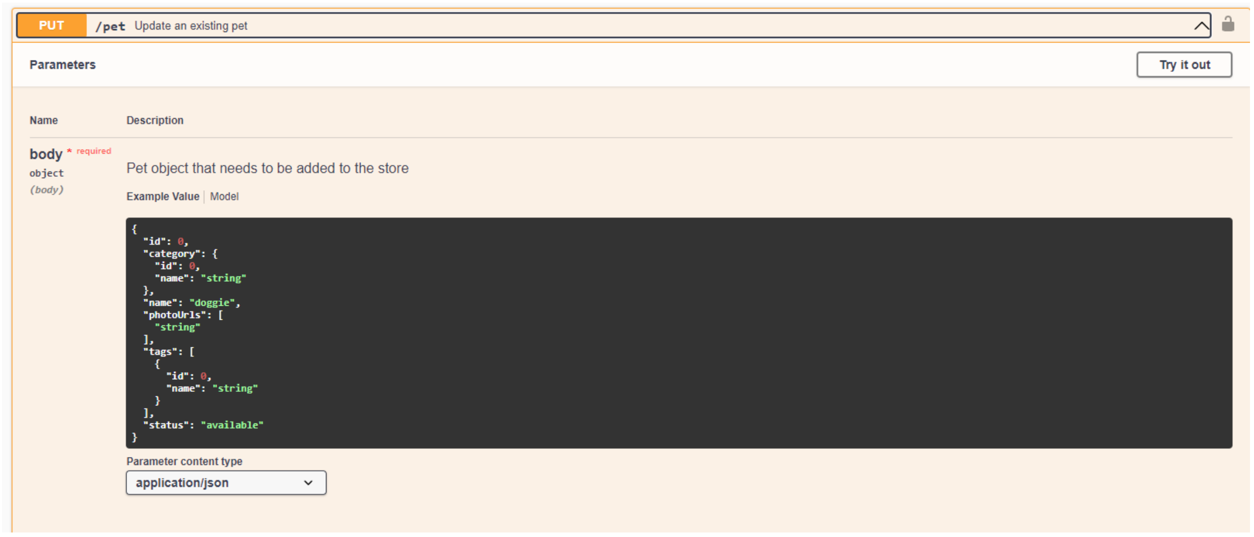


```
{
    "id": 56789,
    "category": {
        "id": 4,
        "name": "cats with long fur"
    },
    "name": "Hello Kitty",
    "photoUrls": [
        "/url/photo"
    ],
    "tags": [
        {
            "id": 2,
            "name": "long hair"
        }
    ],
    "status": "available"
}
```

Step 13.2: **T**hink about which would be the scenarios we could test for the GET /v2/pet/{petId} request, by checking the documentation in Swagger and make a list.

Open https://petstore.swagger.io/, expand the GET /pet/{petId} and select Model in the body parameter

Step 14.1: Now let's learn how we edit the pet. By checking the documentation we can see that for editing a pet using PUT we have the same body that needs to be sent as when adding a pet using POST



body * required
object
(body)

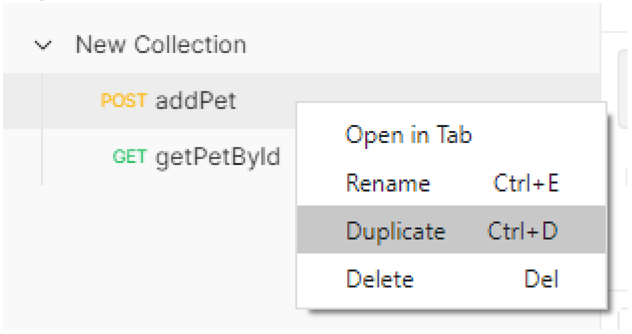
Pet object that needs to be added to the store

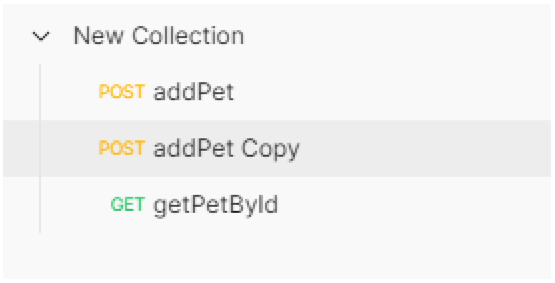
```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Parameter content type: application/json

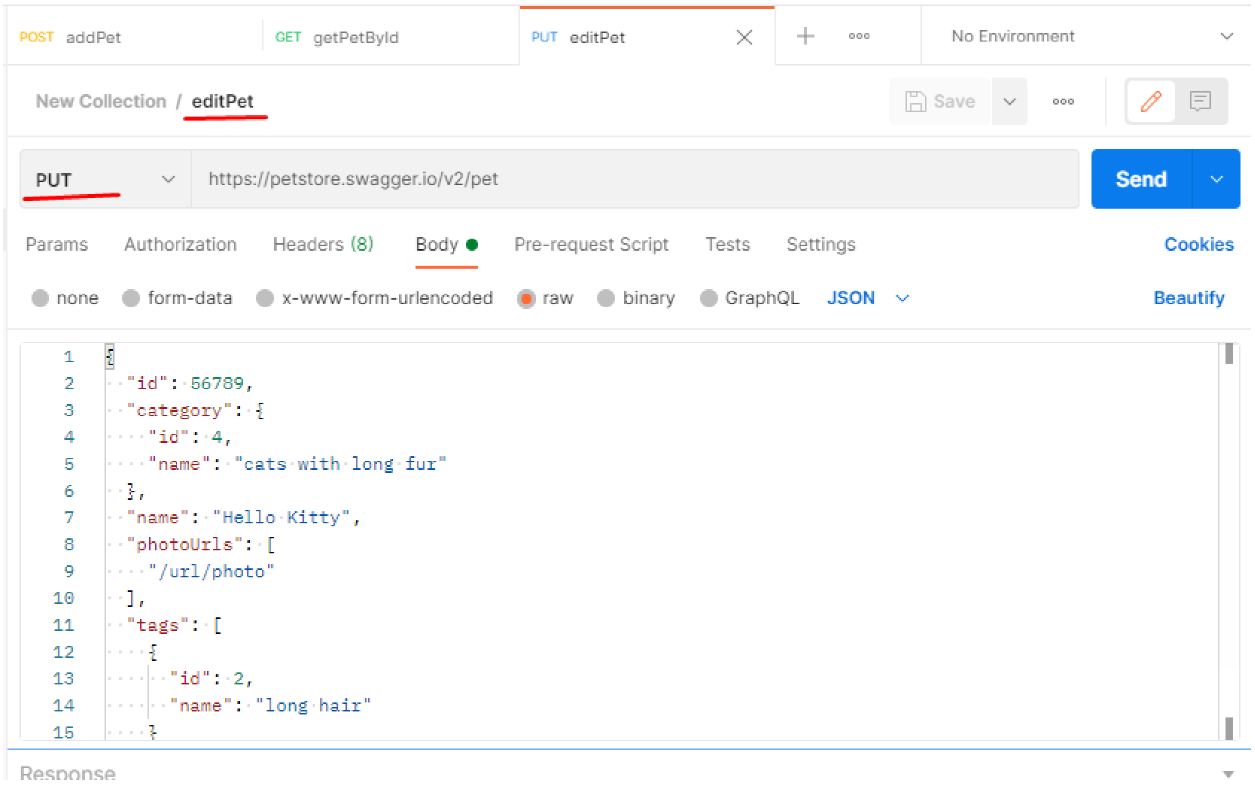
Step 14.2: We need to add the PUT /pet request in Postman. Since we have the same resource path as the POST method, we can just duplicate it and modify it afterwards

Right click on the POST addPet method and select Duplicate => a copy was created





Step 14.3: Select the copy, rename it to editPet and change the method from POST to PUT

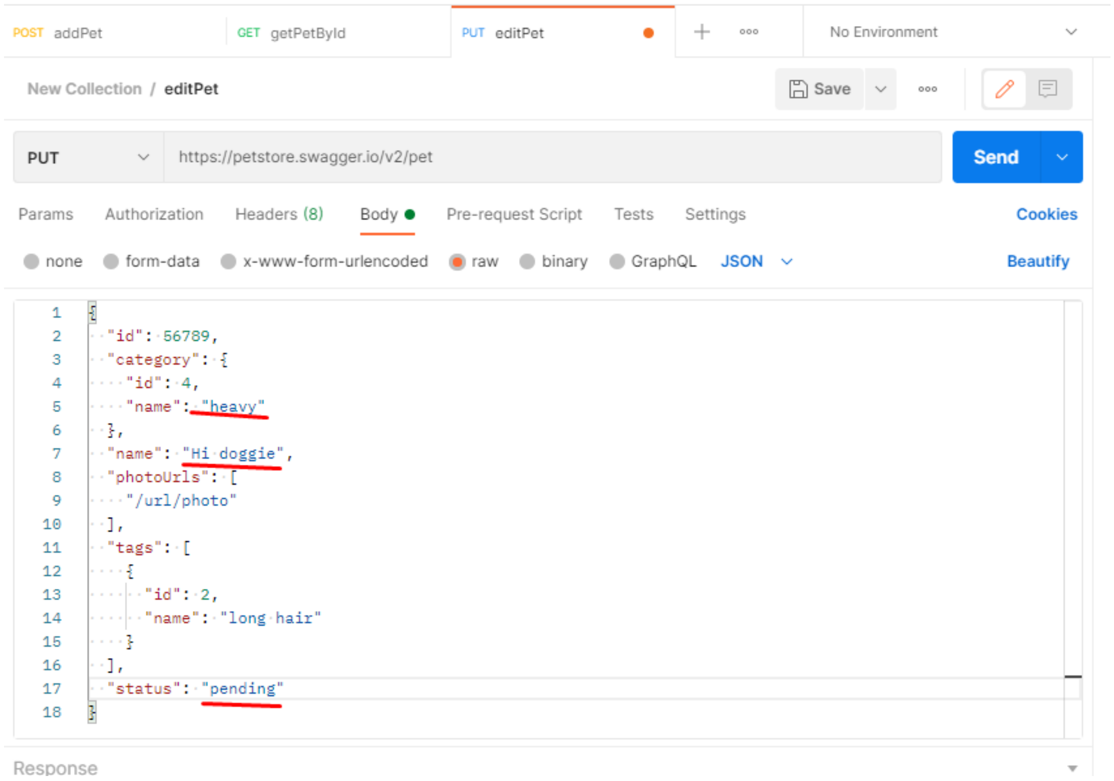


https://petstore.swagger.io/v2/pet

```
{
  "id": 56789,
  "category": {
    "id": 4,
    "name": "cats with long fur"
  },
  "name": "Hello Kitty",
  "photoUrls": [
    "/url/photo"
  ],
  "tags": [
    {
      "id": 2,
      "name": "long hair"
    }
```

Step 15: Now we will need to update some properties of the pet object, but the id needs to remain the same.

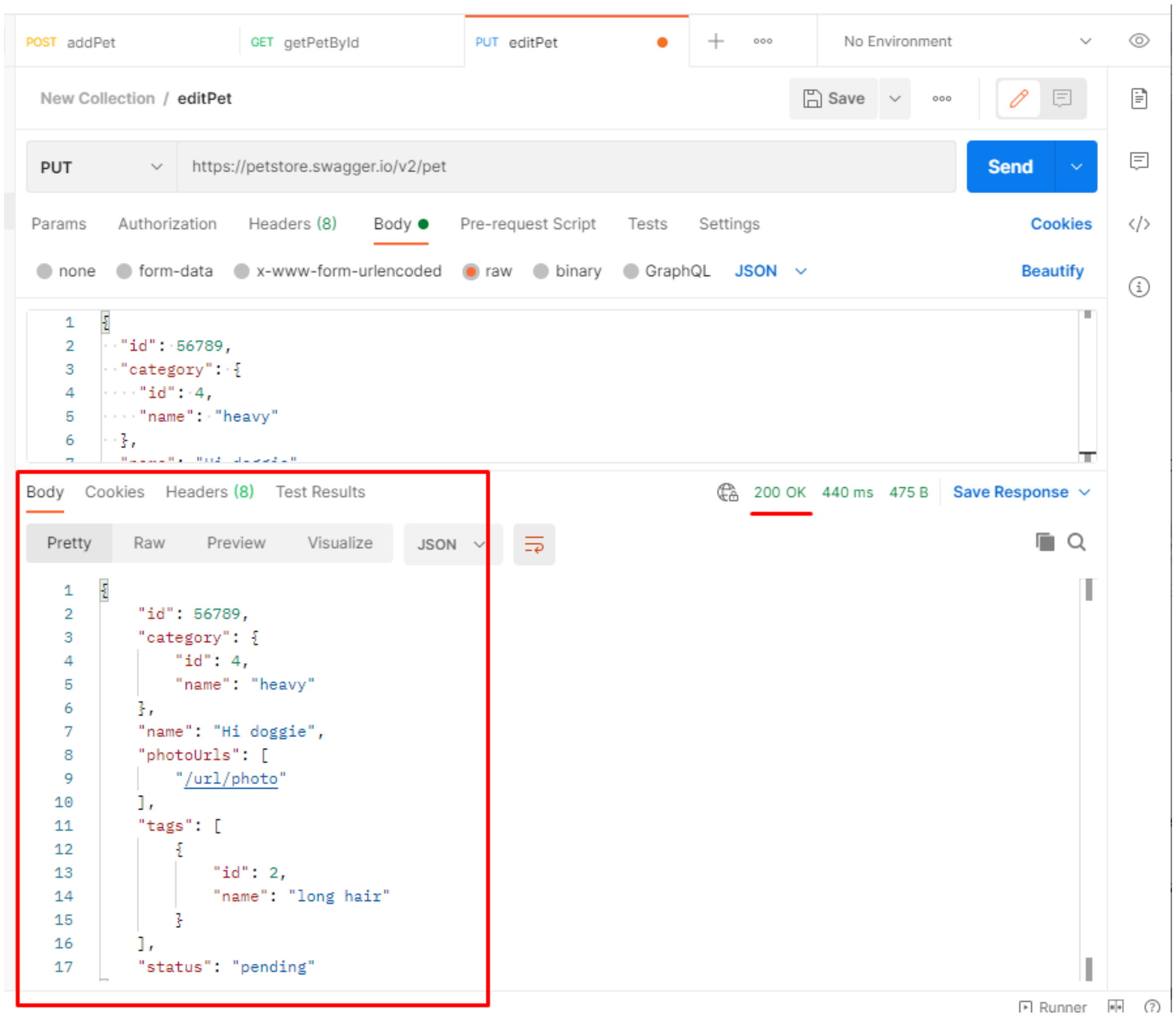
1. Change the body.name property value from “Hello Kitty” to “Hi doggie”
2. Change the body.category.name property from “cats with long fur” to “heavy”
3. Change the body.status from “available” to “pending”



PUT https://petstore.swagger.io/v2/pet

```
{
  "id": 56789,
  "category": {
    "id": 4,
    "name": "heavy"
  },
  "name": "Hi doggie",
  "photoUrls": [
    "/url/photo"
  ],
  "tags": [
    {
      "id": 2,
      "name": "long hair"
    }
  ],
  "status": "pending"
}
```

Step 16.1: Click on the blue Send button. Notice that in the response window of Postman we received the updated pet details.



POST addPet | GET getPetById | PUT editPet | No Environment

New Collection / editPet

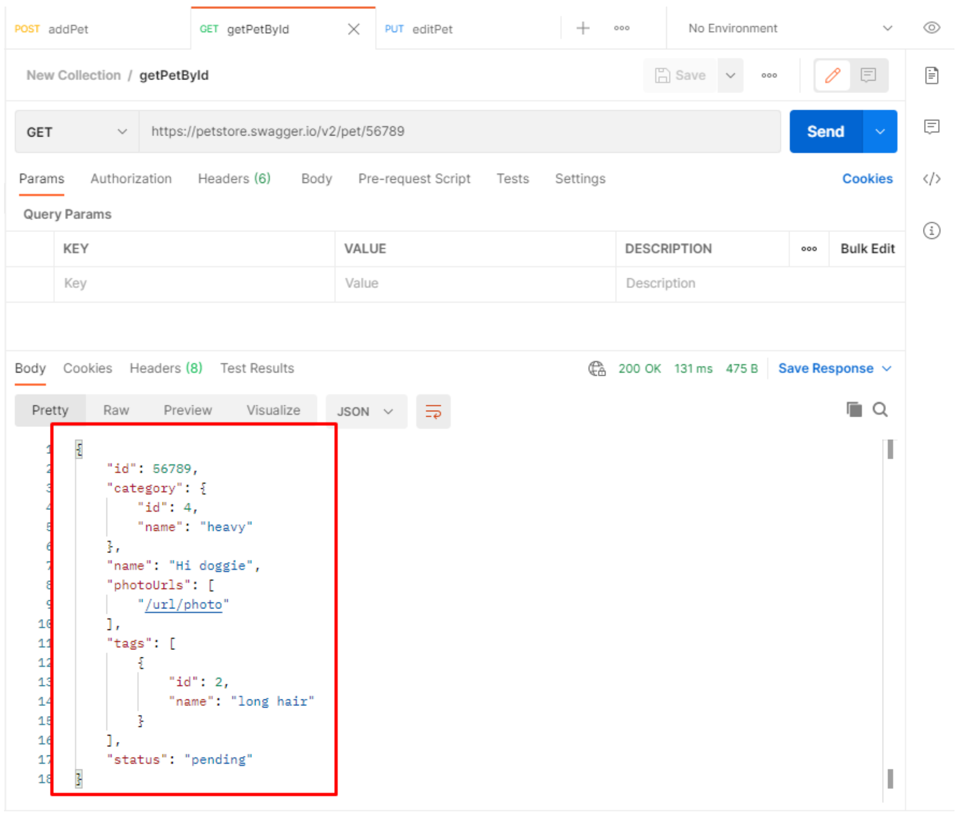
PUT https://petstore.swagger.io/v2/pet

```
{
  "id": 56789,
  "category": {
    "id": 4,
    "name": "heavy"
  },
```

Body Cookies Headers (8) Test Results — 200 OK 440 ms 475 B Save Response

```
{
    "id": 56789,
    "category": {
        "id": 4,
        "name": "heavy"
    },
    "name": "Hi doggie",
    "photoUrls": [
        "/url/photo"
    ],
    "tags": [
        {
            "id": 2,
            "name": "long hair"
        }
    ],
    "status": "pending"
```

Step 17: Now let’s run again the GET /pet/{petId} request to check the updated pet is returned



GET https://petstore.swagger.io/v2/pet/56789

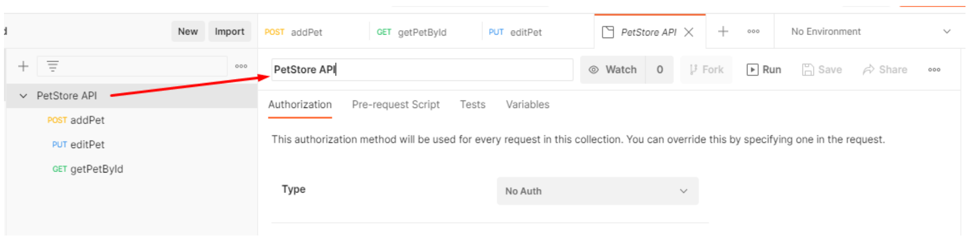
```
{
    "id": 56789,
    "category": {
        "id": 4,
        "name": "heavy"
    },
    "name": "Hi doggie",
    "photoUrls": [
        "/url/photo"
    ],
    "tags": [
        {
            "id": 2,
            "name": "long hair"
        }
    ],
    "status": "pending"
}
```

Step 16.2: Let’s think about which would be the scenarios we could test for the PUT /v2/pet request, by checking the documentation in Swagger and make a list.

Open https://petstore.swagger.io/, expand the PUT /pet and select Model in the body parameter

Exercise by yourself: Add a DELETE request in Postman and run it. Think about what scenarios should be tested. (DELETE /pet/petId)

Step 17: You can name your collection

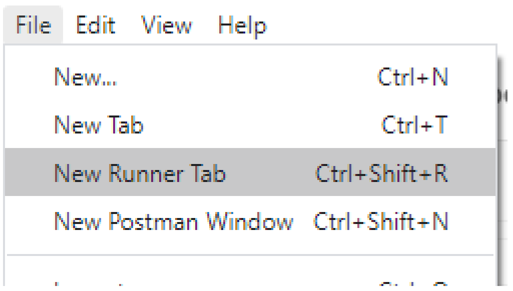


This authorization method will be used for every request in this collection. You can override this by specifying one in the request.

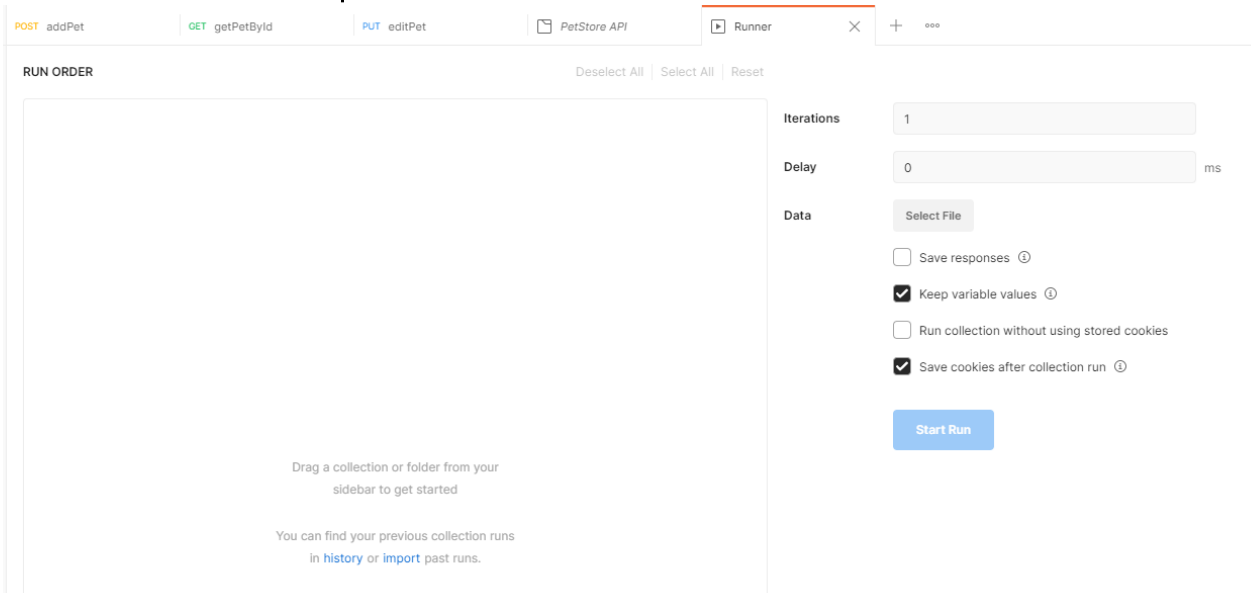
Type: No Auth

Using the collection runner

Step 18: Go to File > New Runner Tab



A new runner tab was opened:



Iterations 1

Delay 0 ms

Data

☐ Save responses

☑ Keep variable values

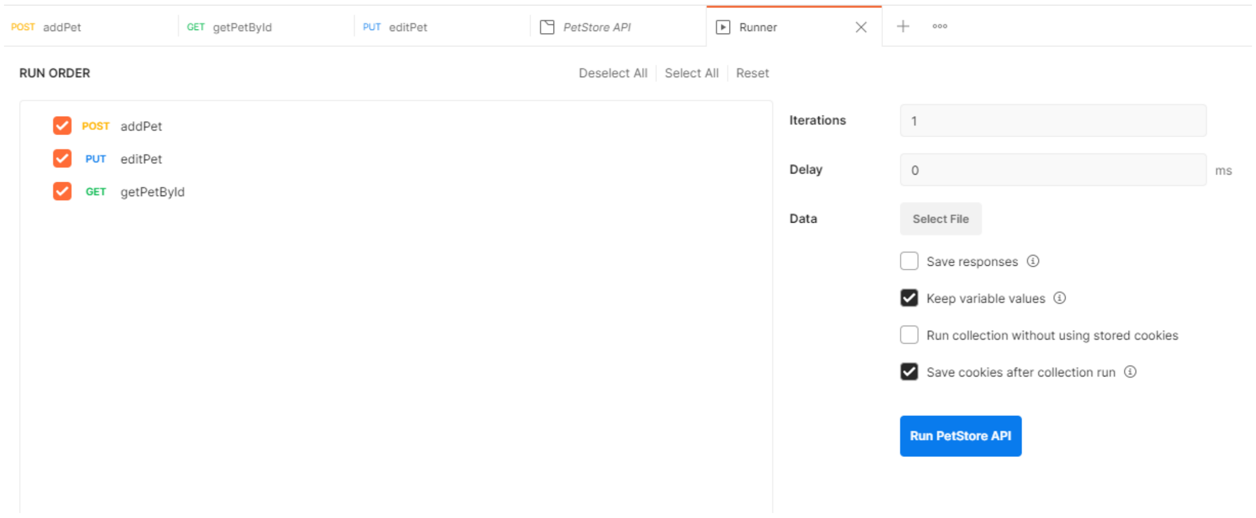
☐ Run collection without using stored cookies

☑ Save cookies after collection run

Drag a collection or folder from your sidebar to get started

You can find your previous collection runs in history or import past runs.

Step 19: Drag and drop the “PetStore API” collection from the left side.



RUN ORDER

Deselect All | Select All | Reset

- ✓ POST addPet
- ✓ PUT editPet
- ✓ GET getPetById

Iterations: 1

Delay: 0 ms

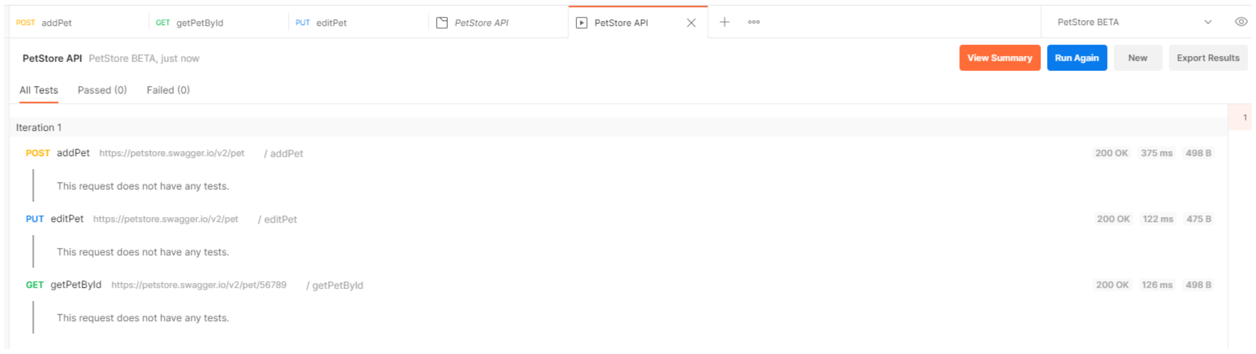
Data: Select File

- ☐ Save responses
- ✓ Keep variable values
- ☐ Run collection without using stored cookies
- ✓ Save cookies after collection run

Run PetStore API

Step 20: Click the blue button “Run PetStore API”

All requests were sent automatically, one after the other. The status code and response time is shown for each request.
As you can see in the runner, for each request we do not have tests. In the next part of the exercise let’s add some tests



PetStore API PetStore BETA, just now

All Tests | Passed (0) | Failed (0)

Iteration 1

POST addPet https://petstore.swagger.io/v2/pet / addPet 200 OK 375 ms 498 B

This request does not have any tests.

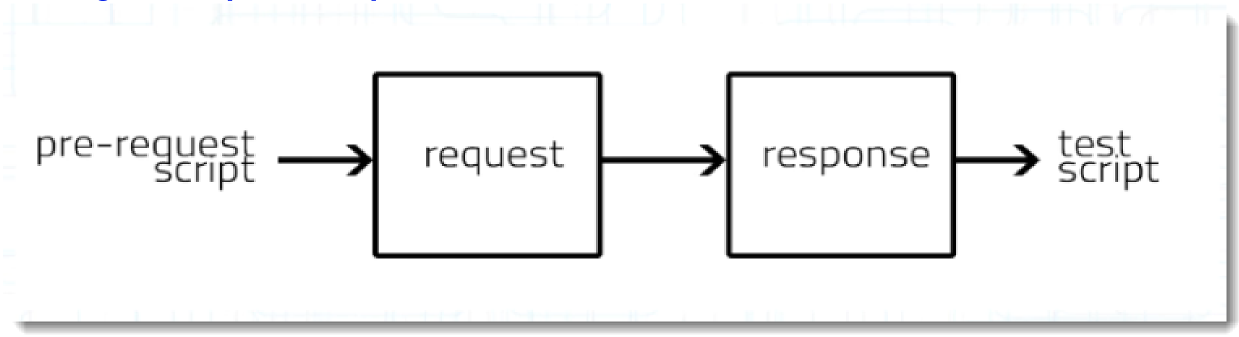
PUT editPet https://petstore.swagger.io/v2/pet / editPet 200 OK 122 ms 475 B

This request does not have any tests.

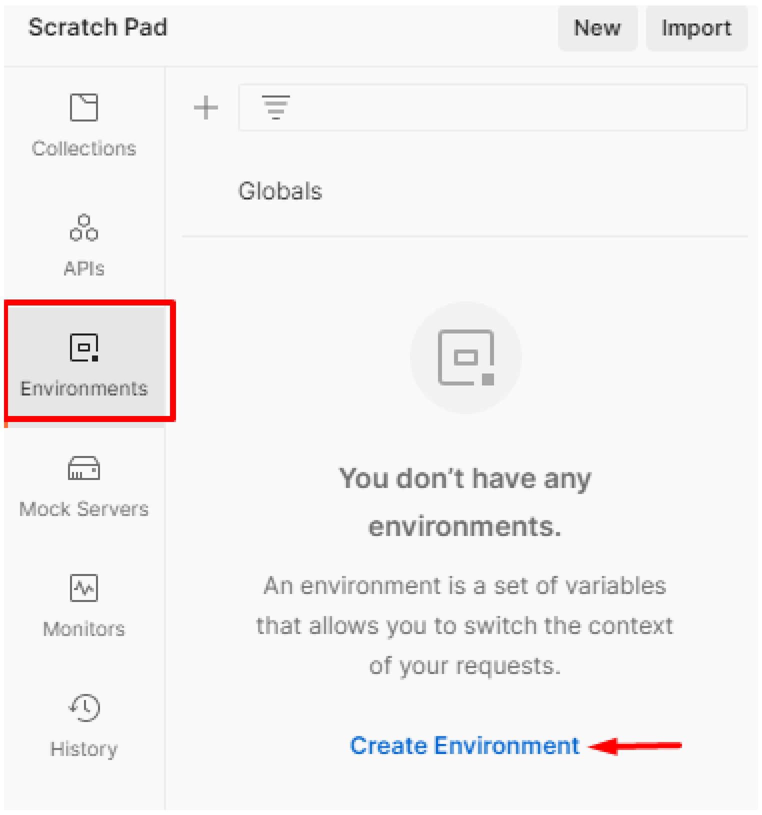
GET getPetById https://petstore.swagger.io/v2/pet/56789 / getPetById 200 OK 126 ms 498 B

This request does not have any tests.

Adding Pre-request Scripts and Tests



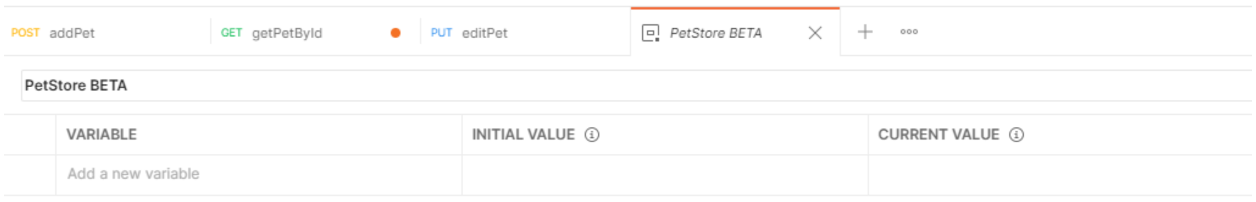
Step 21: First we need to create an environment in Postman, it will be easier to follow our variables. Go to Environments on the left side panel and then click 'Create Environment'



You don't have any environments.

An environment is a set of variables that allows you to switch the context of your requests.

Step 22: You can give it the name “PetStore BETA”

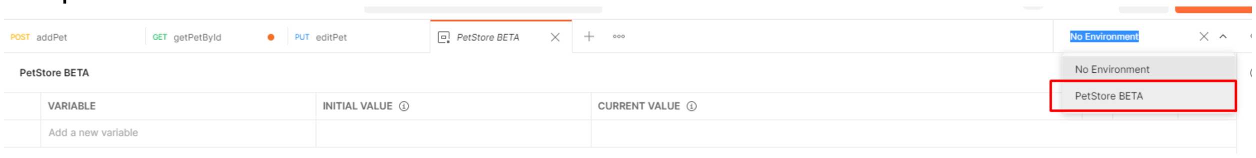


PetStore BETA

VARIABLE	INITIAL VALUE	CURRENT VALUE
Add a new variable		

Step 23: In the right side, where there is a dropdown with the default option “No environment” -> click it and select the newly created environment

Keep the “PetStore BETA” environment selected at all times

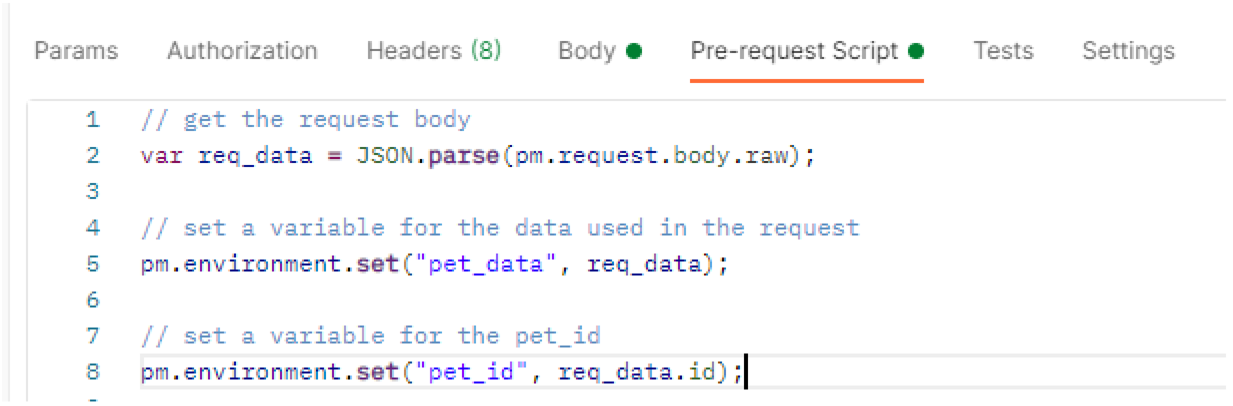


PetStore BETA

VARIABLE	INITIAL VALUE	CURRENT VALUE
Add a new variable		

Step 24: Let’s add a pre-request script for the POST /pet. Since we’ve seen that for this request, the response is the same, we should store the request so that we can compare it in the Tests. Add the following code:

```
// get the request body
var req_data = JSON.parse(pm.request.body.raw);

// set a variable for the data used in the request
pm.environment.set("pet_data", req_data);

// set a variable for the pet_id
pm.environment.set("pet_id", req_data.id);
```



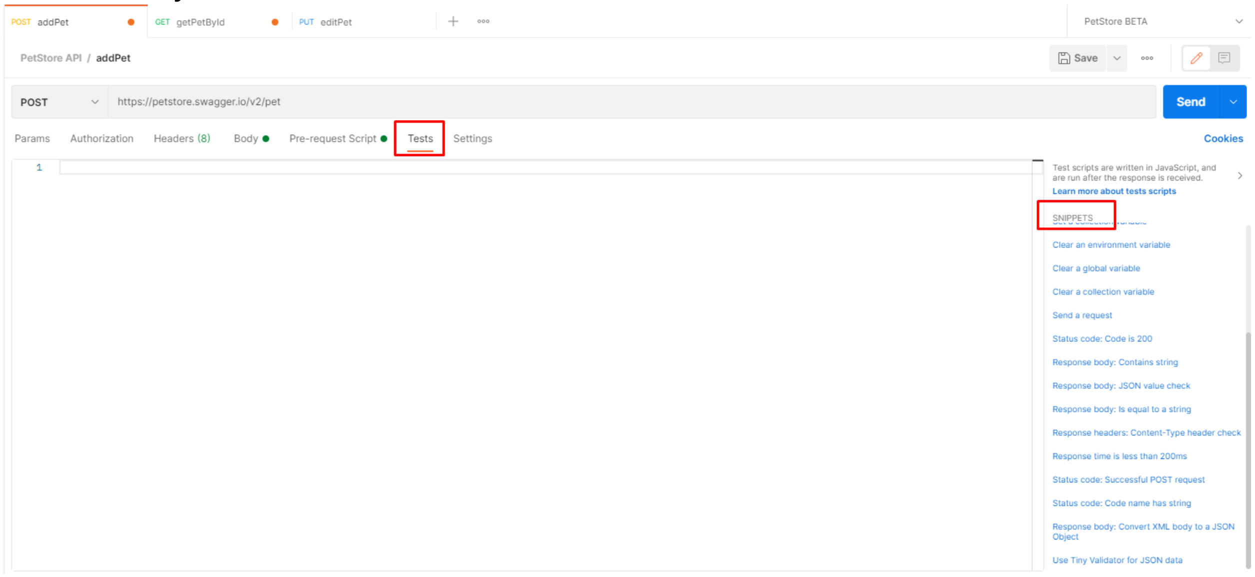
```
// get the request body
var req_data = JSON.parse(pm.request.body.raw);

// set a variable for the data used in the request
pm.environment.set("pet_data", req_data);

// set a variable for the pet_id
pm.environment.set("pet_id", req_data.id);
```

Send the request and then check the “PetStore BETA” environment -> see how the values got saved.

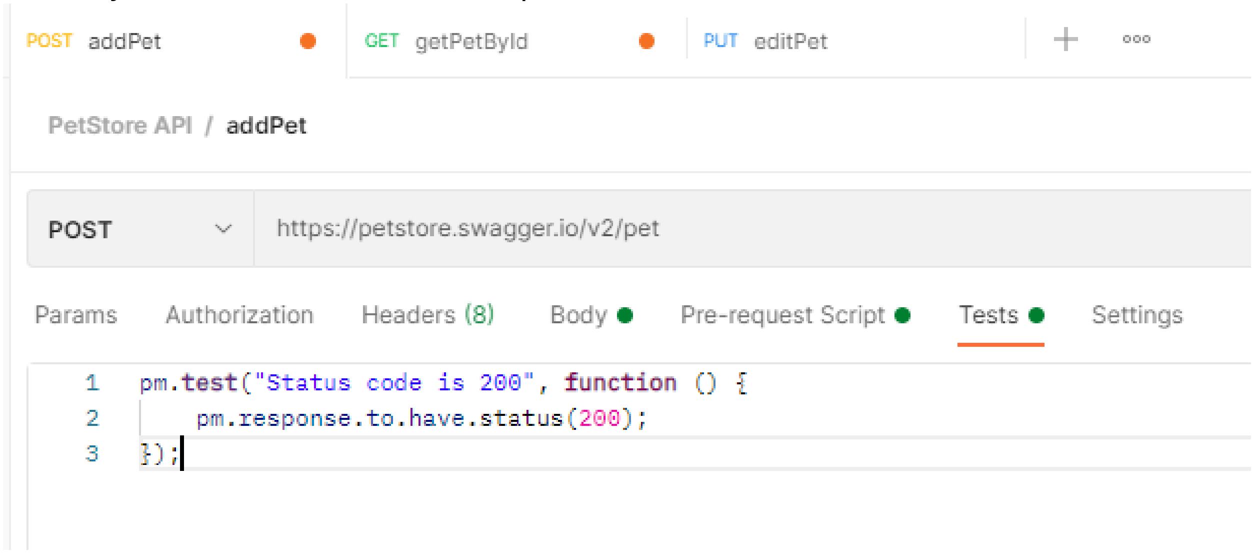
Step 25: Let's add tests for the POST /pet. What should we verify when we receive the response?

- Response status code is successful
- Time to receive the response is not slow
- The response body is the one we expect

On the right side of the Tests area, you will find a SNIPPETS area from which you can automatically add assertions.

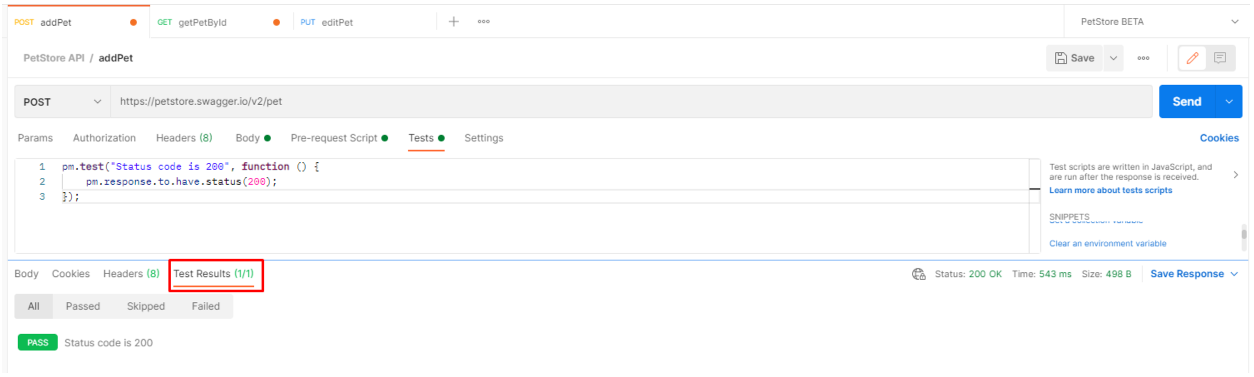

POST addPet
GET getPetById
PUT editPet
PetStore BETA
PetStore API / addPet
Save
POST
https://petstore.swagger.io/v2/pet
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Test scripts are written in JavaScript, and are run after the response is received.
Learn more about tests scripts
SNIPPETS
Clear an environment variable
Clear a global variable
Clear a collection variable
Send a request
Status code: Code is 200
Response body: Contains string
Response body: JSON value check
Response body: Is equal to a string
Response headers: Content-Type header check
Response time is less than 200ms
Status code: Successful POST request
Status code: Code name has string
Response body: Convert XML body to a JSON Object
Use Tiny Validator for JSON data

Let's click on the "Status code: Code is 200". See how code got automatically added to the Tests area. If you send the request right now, there will be an automatic verification made by Postman to check if the response status code is 200 or not.



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

Step 26: The result of the tests can be found next to where the response body is shown, in the “Test Results” section

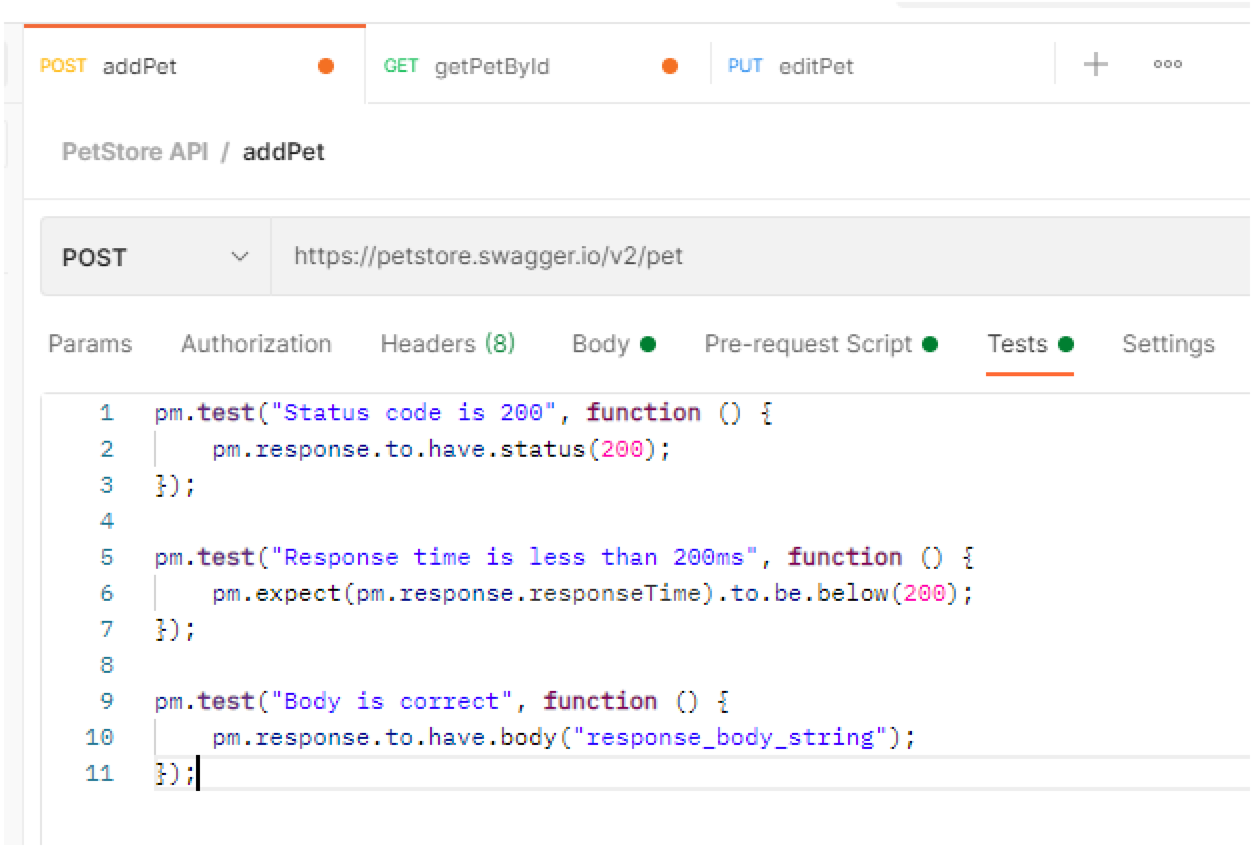


```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Test scripts are written in JavaScript, and are run after the response is received.

PASS Status code is 200

Step 27: Let’s add some more assertions in the Tests section. Click on the “Response time is less than 200ms” and “Response body: is equal to a string” from the SNIPPETS area. More code for asserting the response was added.

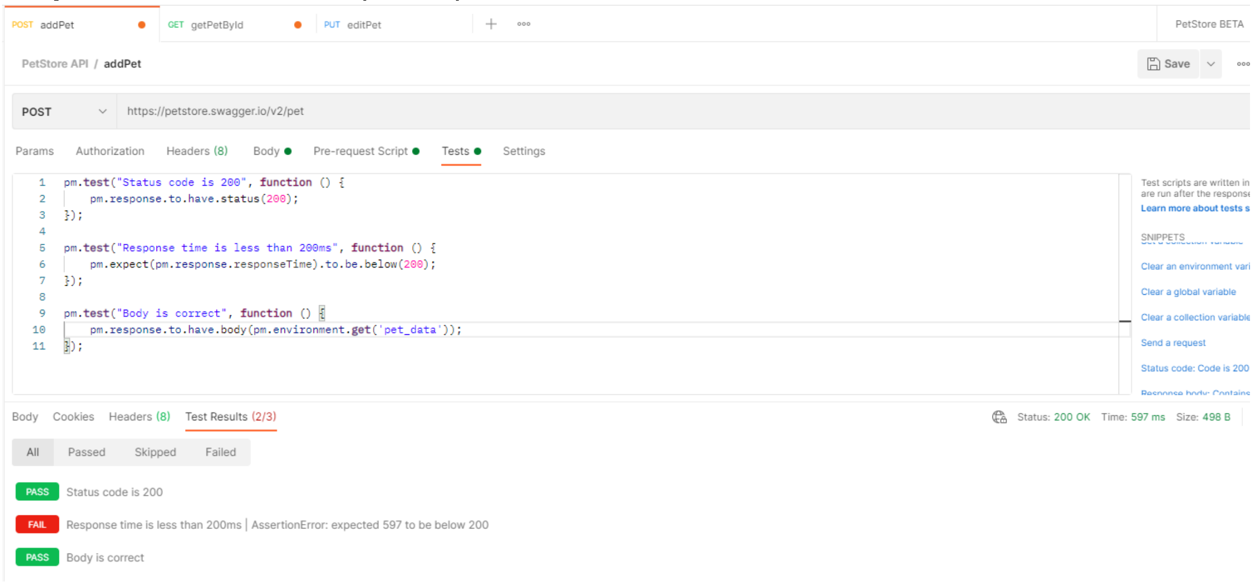


```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Body is correct", function () {
    pm.response.to.have.body("response_body_string");
});
```

Step 28: The test for the checking if the response body is correct is not yet finished. We need to modify the code and get the value of the environment variable "pet_data" that we saved in step 21.

```
pm.test("Body is correct", function () {
    pm.response.to.have.body(pm.environment.get('pet_data'));
});
```

Step 29: Run the POST /pet request now and check the Test Results



https://petstore.swagger.io/v2/pet

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

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Body is correct", function () {
    pm.response.to.have.body(pm.environment.get('pet_data'));
});
```

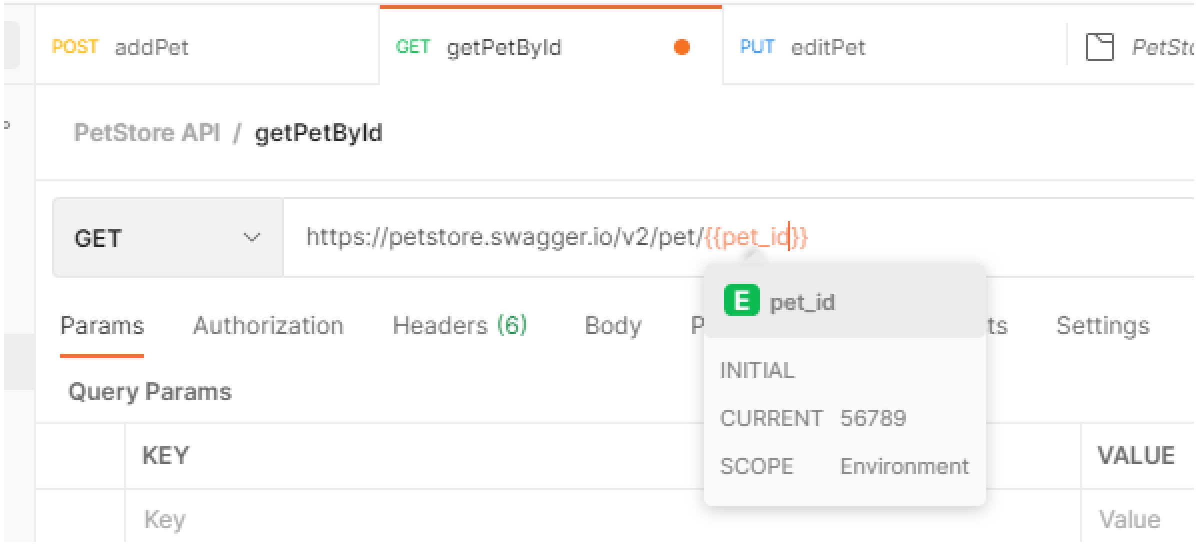
Test scripts are written in
are run after the respons
Learn more about tests s

Status: 200 OK Time: 597 ms Size: 498 B

PASS Status code is 200
FAIL Response time is less than 200ms | AssertionError: expected 597 to be below 200
PASS Body is correct

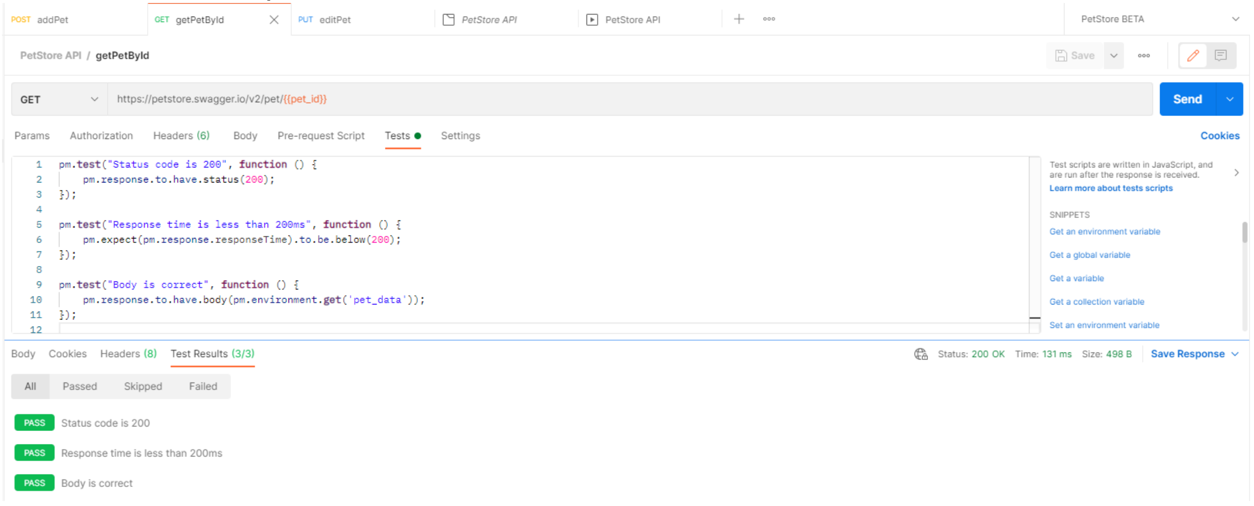
Step 30: Open the GET /pet/{petId} request. Since we've also stored the "pet_id" in the environment variables, let's use that to send the id to the PetStore API.

We need to add the variable in the path of the URL. You can do this by replacing the manually written id "56789" with {{pet_id}}



Step 31: Now run the GET /pet/{petId} request. Notice that the details for out pet were returned.

Step 32: Add tests for GET /pet/{petId} request and run it. We can use the same that we had for POST /pet.



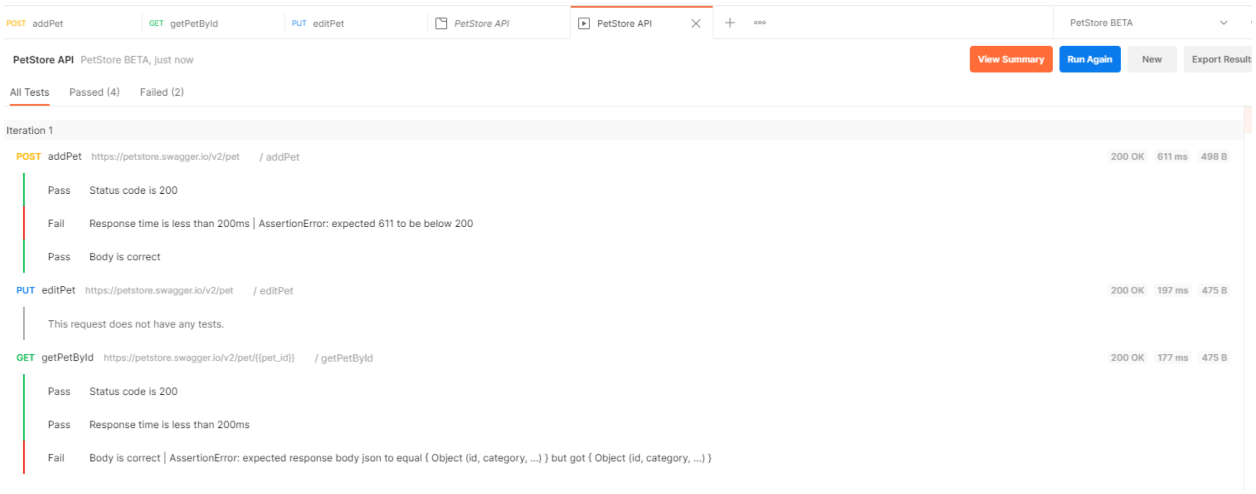
```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Body is correct", function () {
    pm.response.to.have.body(pm.environment.get('pet_data'));
});
```

```
PASS Status code is 200
PASS Response time is less than 200ms
PASS Body is correct
```

Exercise by yourself: Add tests and pre-request scripts (if necessary) for the PUT /pet/{petId} and DELETE /pet/{petId} requests

Using the collection runner with tests

Step 33: Open the collection runner again, add the collection and run it. Notice how this time we also have some failed/passed statuses.



PetStore API PetStore BETA, just now

```
Iteration 1
POST  addPet  https://petstore.swagger.io/v2/pet  / addPet                                        200 OK  611 ms  498 B
    Pass  Status code is 200
    Fail  Response time is less than 200ms | AssertionError: expected 611 to be below 200
    Pass  Body is correct
PUT  editPet  https://petstore.swagger.io/v2/pet  / editPet                                        200 OK  197 ms  475 B
    This request does not have any tests.
GET  getPetById  https://petstore.swagger.io/v2/pet/{{pet_id}}  / getPetById                       200 OK  177 ms  475 B
    Pass  Status code is 200
    Pass  Response time is less than 200ms
    Fail  Body is correct | AssertionError: expected response body json to equal { Object (id, category, ...) } but got { Object (id, category, ...) }
```