Lesson Weekend

Intermediate JavaScript (/intermediate-javascript) / Asynchrony and APIs (/intermediate-javascript/asynchrony-and-apis) / Testing API Calls with Postman

Text

In the last lesson, we walked through the process of signing up for a developer account for the OpenWeather API and then getting an API key. We learned some general pointers for walking through API documentation and we even made an API call in the browser. In this lesson, we'll learn how to use a tool called Postman to test our API calls.

Installing Postman

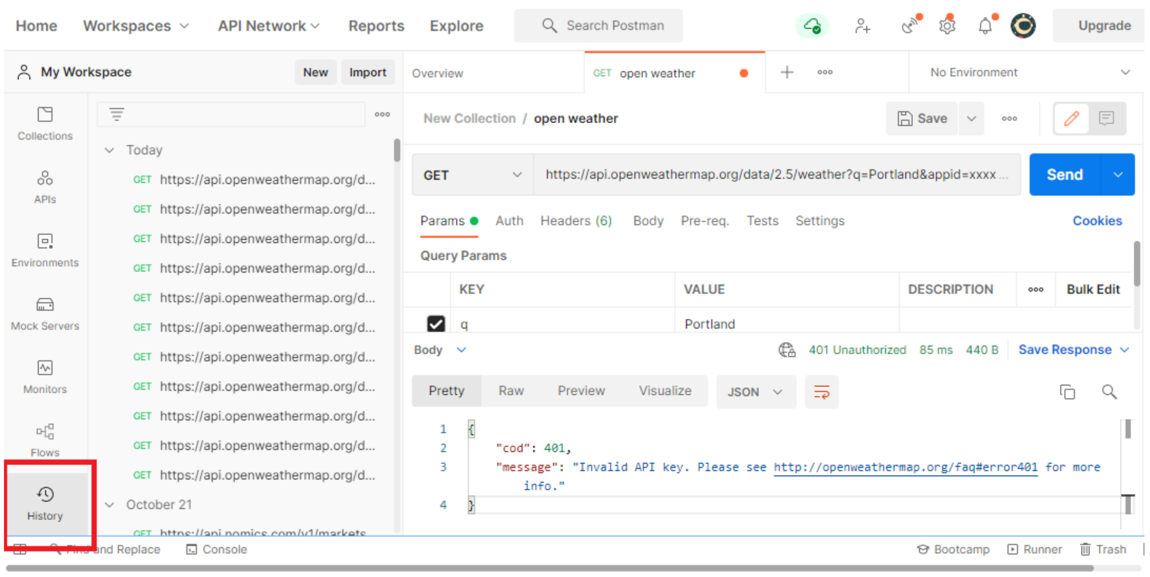
Technically, we could test all of our API calls in the browser. However, Postman is a much better tool for testing API calls and it includes many features that make testing APIs a more visual and inviting process.

To download the Postman app on your personal machine, go to the Postman downloads page (https://www.postman.com/downloads/), and select the correct download for your operating system.

Finish your installation process and then open Postman. While Postman has a lot of nice features, we'll only use the most basic ones here.

Using Postman

To make an API call, we need to create a new request. There's two ways to do this: you can open a new tab or select *Create a request* from the overview tab. In the following image these options are highlighted by a red circle.



GET https://api.openweathermap.org/data/2.5/weather?q=Portland&appid=xxxx ...

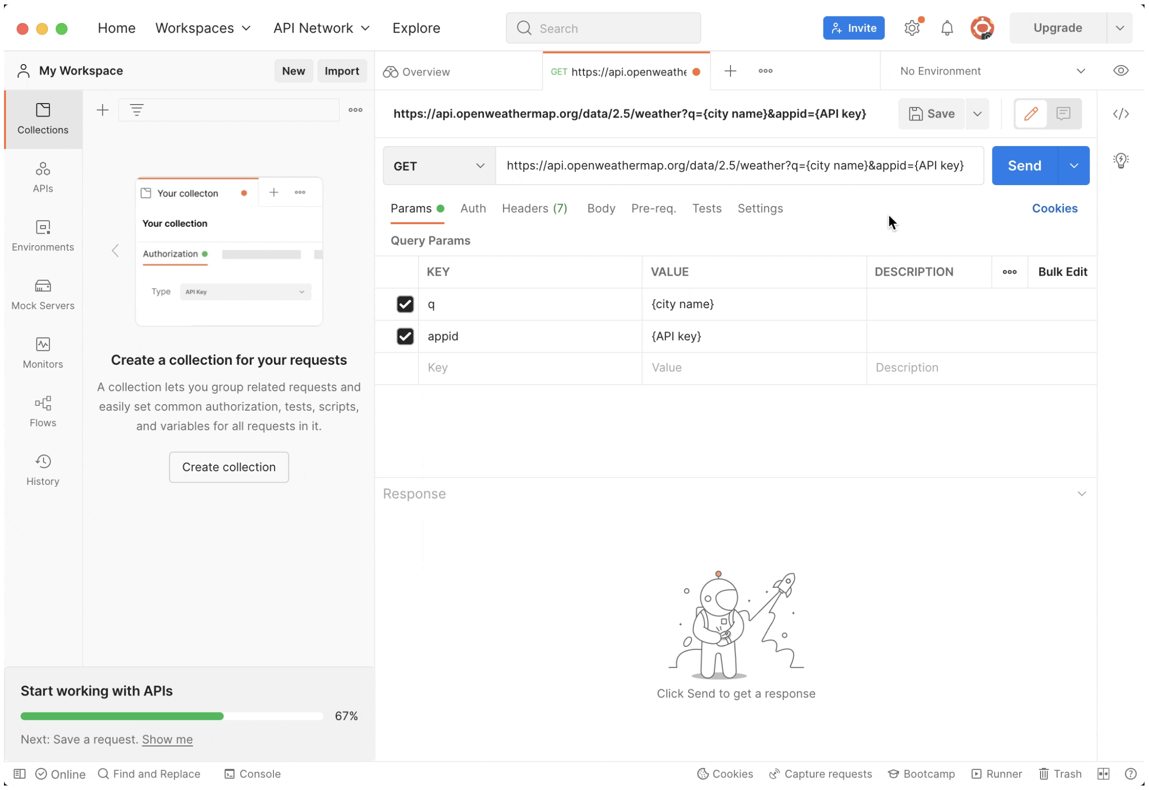
KEY	VALUE	DESCRIPTION
q	Portland	

401 Unauthorized 85 ms 440 B

```
{
    "cod": 401,
    "message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more
        info."
}
```

Next, we need to type in our query into the URL bar.

The default type of API call is a GET request (the request type is just to the left of the URL bar). You will probably only be making GET requests during this section, but if you need to make a POST or other type of request, this is where you'd update the query to reflect that.



https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

GET https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

Params Auth Headers (7) Body Pre-req. Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION
✓	q	{city name}	
✓	appid	{API key}	
	Key	Value	Description

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create collection

Start working with APIs 67%

Next: Save a request. Show me

In the GIF above, we paste in the query we want. The *Params* tab is open and the parameters of the API call are automatically populated if the URL includes any parameters.

Let's take a closer look at our query:

```
https://api.openweathermap.org/data/2.5/weather?q=Portland&
appid=xxxx
```

The format of this URL is exactly the same as the URL format of a webpage after we submit a form. There is:

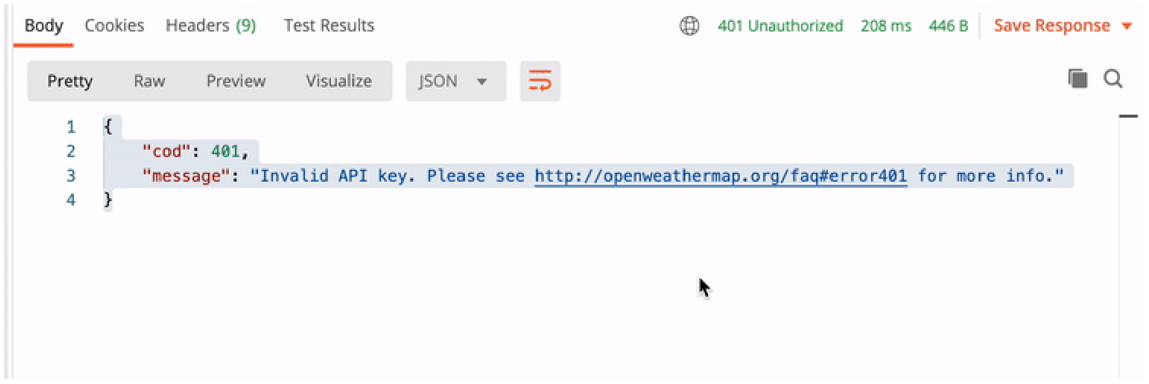
- The protocol `https`
- The domain and path `api.openweathermap.org/data/2.5/weather`
- A `?` to denote it's a query.
- Then finally, the parameters `q` and `appid`; each parameter has a value and is separated by an `&` symbol: `q=Portland&appid=xxxx`

This is a very common format, though the names of the query parameters themselves will vary between APIs.

As we can see, Postman makes it very easy to see our parameters, and also to update them if need be. We can update the parameters (or add more) either in the URL itself or by changing/adding key value pairs in the *Params* tab. The latter is a better idea because then we won't accidentally break the rest of the URL in the process.

Once we're ready to make our API call, we click the *Send* button.

We intentionally put in a bogus API key here but we still get an informative response from the API. We'll see that response in the bottom tab of Postman.

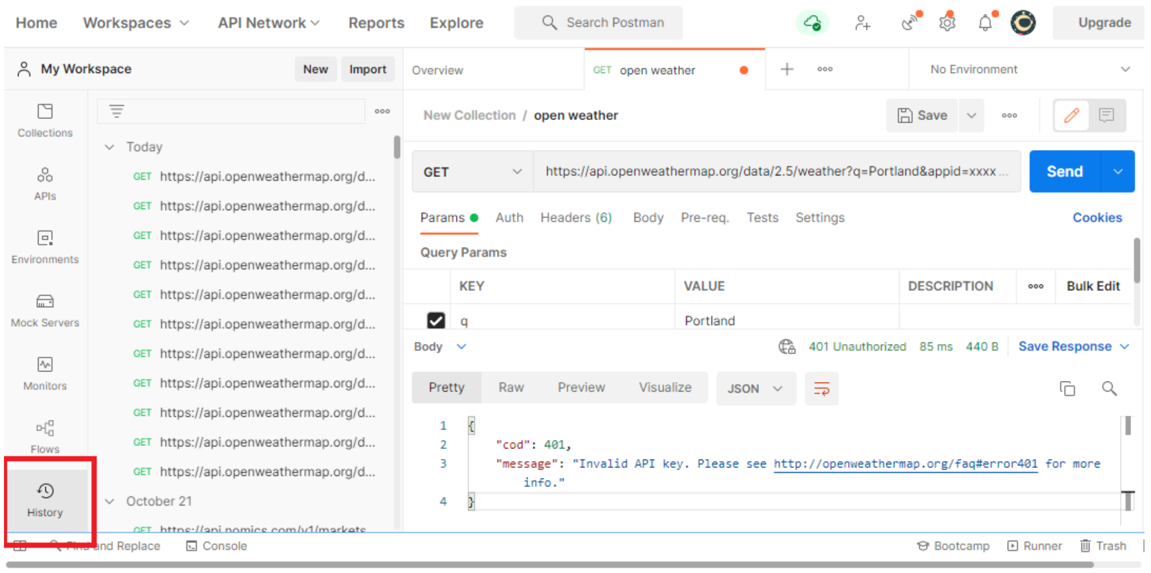


```
{
    "cod": 401,
    "message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info."
}
```

Every API response comes with a **response code**. If all goes well, we'll get a *200 OK*. A 200 code means the query was successful. In this case, though, we get a *401* code, which means the query was unauthorized. The message tells us exactly why: "Invalid API key." That's to be expected. We intentionally didn't put in a valid key. We can hover over the response code in the upper right corner to see what it means. Postman also has tabs so we can look at the response in different ways — but as you can see from the GIF, the *Pretty* option is the best way to go.

One other really nice thing about Postman is the *History* tab in the left-hand pane, which is highlighted in red in the image below. Just as our browser keeps a history of pages we've visited, making it easy to visit them again, Postman does the same.

We can also open a new tab (above the URL bar) if we want to be able to toggle between testing multiple requests at once.



GET https://api.openweathermap.org/data/2.5/weather?q=Portland&appid=xxxx ...

KEY	VALUE	DESCRIPTION
q	Portland	

401 Unauthorized 85 ms 440 B

```
{
    "cod": 401,
    "message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more
        info."
}
```

We've just scratched the surface of what Postman can do, but this functionality is likely all you'll need for making and testing API calls while you're a student at Epicodus. Feel free to explore Postman further — the documentation (https://learning.postman.com/docs/getting-started/introduction/) is extensive.

Always Test your APIs

Whenever you want to integrate an API call in your code, **always** test it in Postman first. There are several reasons why this is important:

- If the API endpoint doesn't work, you can verify that *before* wasting your time building code based on it. Most API endpoints *will* work, but if you're working with an API that might be outdated or has weak documentation, you might find that some endpoints aren't working correctly.
- More likely, and more importantly, you want to make sure that you've correctly formatted the API call before you integrate it

into your code. That way, if it's not working correctly in your code, you know that your code is the problem, *not* the API call. If you don't test the call first, you may have a hard time distinguishing whether the problem is coming from your code or the API query itself.

At this point, we are *almost* ready to start building out a JavaScript application. Before we do, though, let's take a closer look at the JSON format — and how to pull data from it.

Previous (/intermediate-javascript/asynchrony-and-apis/api-documentation-and-keys)
Next (/intermediate-javascript/asynchrony-and-apis/parsing-json)
Lesson 5 of 33
Last updated more than 3 months ago.

disable dark mode

Epicodus (http://www.epicodus.com)