

# API Workshop Activities

PDF printout of activities

*Tom Johnson, I'd Rather Be Writing*

This booklet contains a PDF of the activities listed in <https://idratherbewriting.com/learnapidoc/workshop.html> in print format. As you're working through the workshop activities, you might find it easier to have the instructions printed out next to your laptop rather than toggling between screens and windows.

**Copyright 2019.** All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law. For permission requests, contact Tom Johnson at [tomjoht@gmail.com](mailto:tomjoht@gmail.com).

# Activities

Activity 1a: Identify your goals.....	2
Activity 2a: Explore OpenWeatherMap API.....	3
Activity 2b: Get OpenWeatherMap authorization keys.....	4
Activity 2c: Make requests with Postman .....	5
Activity 2d: Make requests with curl.....	9
Activity 2e: Make an API request on a web page.....	11
Activity 3a: What's wrong with this API reference topic.....	15
Activity 3b: Evaluate API ref docs to identify core elements.....	19
Activity 4a: Explore Swagger UI through the Petstore Demo.....	21
Activity 4b: Edit an existing OpenAPI specification document .....	25
Activity 4c: Create a SwaggerUI display .....	26
Activity 6a: Judge conceptual content and decide which is best .....	30
Activity 7a: Classify code documentation into one of the five types.....	31
Activity 8a: Set up a GitHub wiki .....	32
Activity 8b: Clone your GitHub repo locally.....	33
Activity 8c: Push local changes to the remote .....	35
Activity 9a: Look at API documentation jobs and requirements .....	38
Activity 9b: Find an open-source project .....	39

# Activity 1a: Identify your goals with API documentation

Identify your goals here and make sure they align with this course. Think about the following questions:

- Why are you taking this course?
- What are your career ambitions related to API documentation?
- Are you in a place where developer documentation jobs are plentiful?
- What would you consider to be a success metric for this course?
- Do you have the technical mindset needed to excel in developer documentation fields?

For live workshops, we typically share responses in a get-to-know-everyone format. But if you're taking this course online, consider jotting down some thoughts in a journal or blog entry.

# Activity 2a: Get familiar with the OpenWeatherMap API

Let's explore the basic sections in the [OpenWeatherMap API \(https://openweathermap.org/api/\)](https://openweathermap.org/api/):

1. Go to the [openweathermap.org \(https://openweathermap.org\)](https://openweathermap.org)
2. Click **API** in the top navigation bar.
3. In the **Current weather data** section, click the **API doc** button.

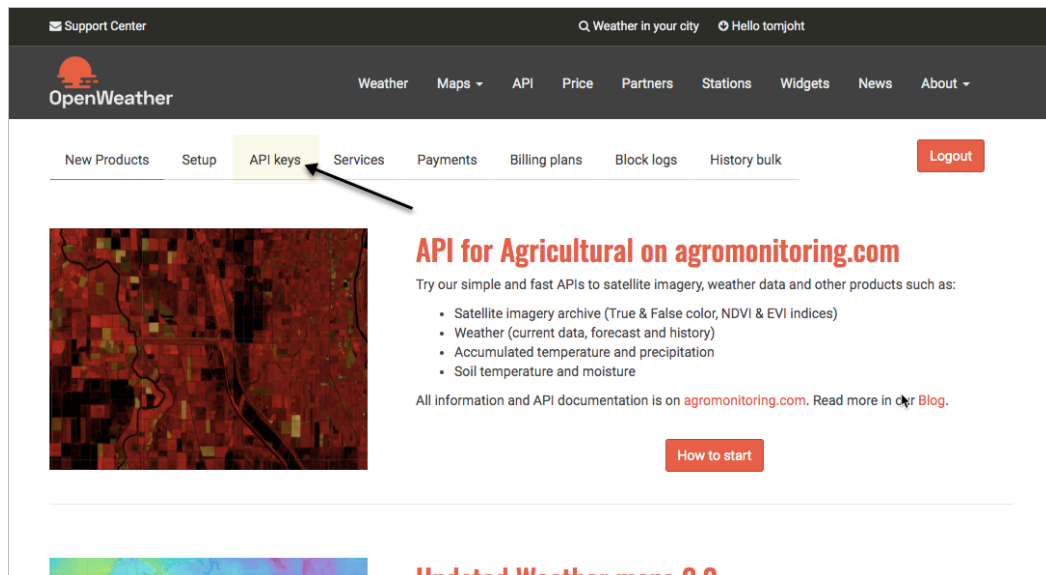
Get a sense of the information this Current Weather Data API provides. The API calls provide developers with ways to pull information into their applications. In other words, the APIs will provide the data plumbing for the applications that developers build.

4. Answer the following questions about the Current Weather Data API endpoint:
  - Does the API provide the information we need about temperature, wind speed, wind direction, and current conditions? (Hint: Look at some of the sample API responses by clicking links under "Examples of API calls.")
  - How many different ways can you specify the location for the weather information?
  - What does a sample request look like?
  - How many endpoints does the API have?
  - What authorization credentials are required to get a response?

# Activity 2b: Get an OpenWeatherMap API key

To get an API key for the OpenWeatherMap API:

1. Go to [openweathermap.org](https://openweathermap.org) (<https://openweathermap.org>).
2. Click **Sign Up** in the top navigation bar and create an account.
3. After you sign up, your API key is sent to the email address you provide. You can also find it on the Developer Dashboard in the site. Return to the OpenWeatherMap homepage and click **Sign in**.
4. After signing in, you'll see the developer dashboard. Click the **API Keys** tab (highlighted in the screenshot below).



API Keys tab on OpenWeatherMap Developer Dashboard

5. Copy the key to a place you can easily find it.

(Note: It can take an hour or so for a new OpenWeatherMap API key to activate. If you have trouble with your key, use one of the keys [listed here](http://idratherbewriting.site/apikeys) (<http://idratherbewriting.site/apikeys>). Put your name next to the key you're using.)

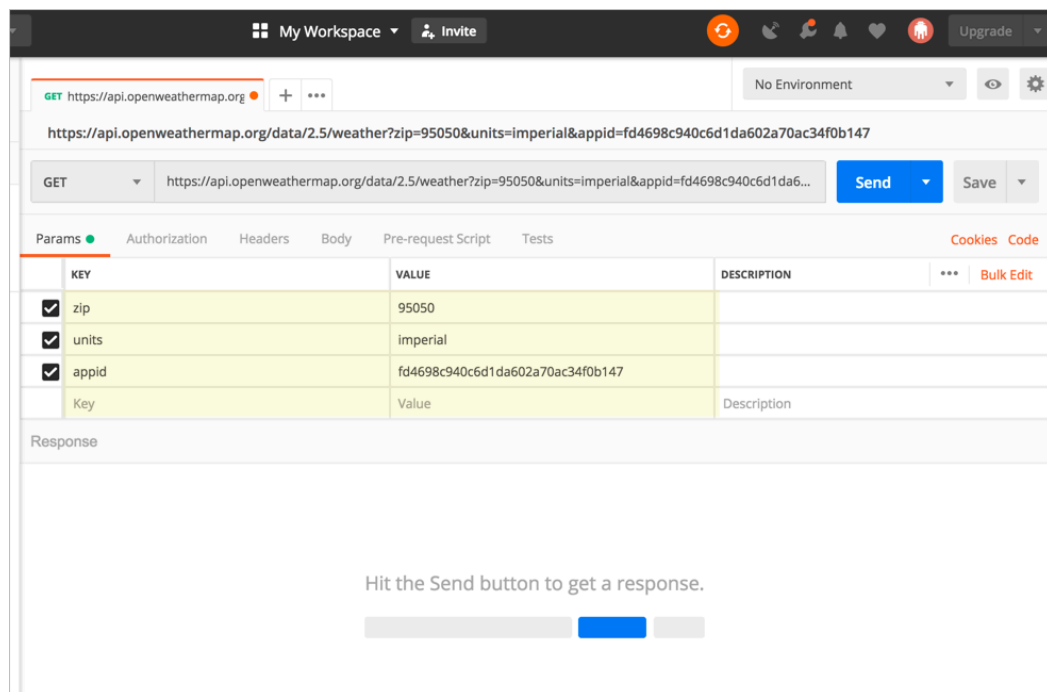
# Activity 2c: Make requests with Postman

## Make a request

In this exercise, you'll use Postman to make a request using OpenWeatherMap's [current weather data API endpoint](https://openweathermap.org/current) (<https://openweathermap.org/current>). To make the request:

1. If you haven't already done so, download and install the Postman app at <https://www.getpostman.com/downloads/> (<https://www.getpostman.com/downloads/>). (Make sure you download the app and not the deprecated Chrome extension.)
2. Start the Postman app.
3. Insert the following endpoint into the box next to **GET**: `https://api.openweathermap.org/data/2.5/weather`.
4. Click the **Params** tab (below the box where you inserted the endpoint) and then add the following three parameters in the **key** and **value** rows:
  - key: `zip` / value: `95050`
  - key: `units` / value: `imperial`
  - key: `appid` / value: <insert your own API key>

For the value for `appid`, use your own API key. (If you didn't [get an API key](https://idratherbewriting.com/learnapidoc/docapis_get_auth_keys.html) ([https://idratherbewriting.com/learnapidoc/docapis\\_get\\_auth\\_keys.html](https://idratherbewriting.com/learnapidoc/docapis_get_auth_keys.html)), use [one of the keys here](https://idratherbewriting.com/apikeys) (<https://idratherbewriting.com/apikeys>.) Your Postman UI should look like this:

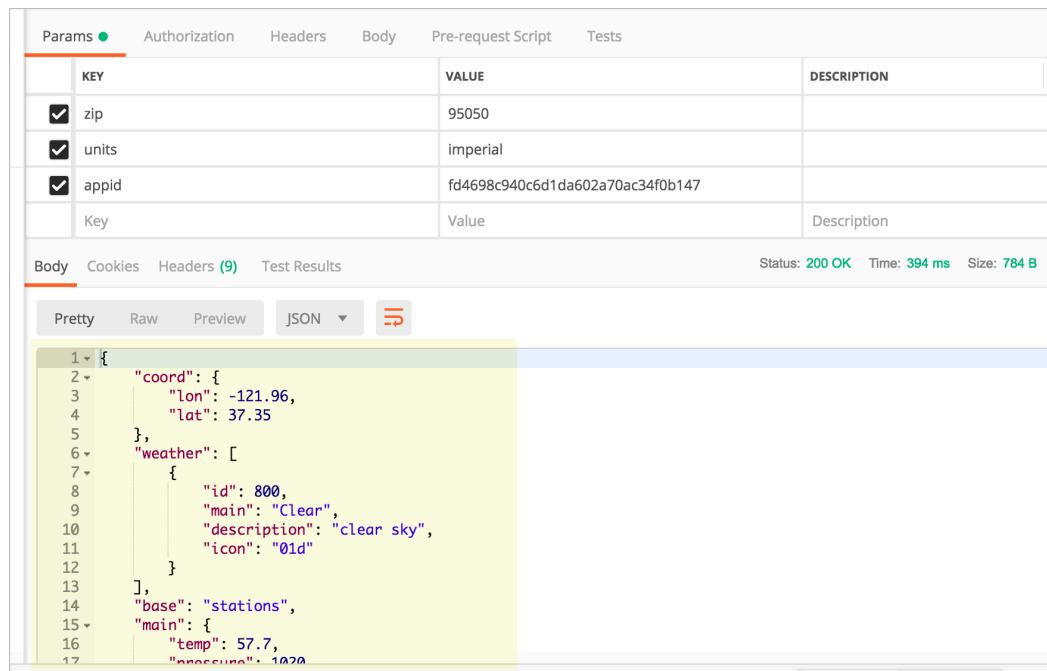


When you add these parameters, they appear as a query string to the endpoint URL in the GET box. For example, your endpoint will now look like this: `https://api.openweathermap.org/data/2.5/weather?zip=95050&units=imperial&appid=fd4698c940c6d1da602a70ac34f0b147` (but with different query string values). Query string parameters appear after the question mark `?` symbol and are separated ampersands `&`. The order of query string parameters doesn't matter.

Note that many APIs pass the API key in the header rather than as a query string parameter in the request URL. (If that were the case, you would click the **Headers** tab and insert the required key-value pairs in the header. But OpenWeatherMap passes the API key as a query string parameter.)

5. Click **Send**.

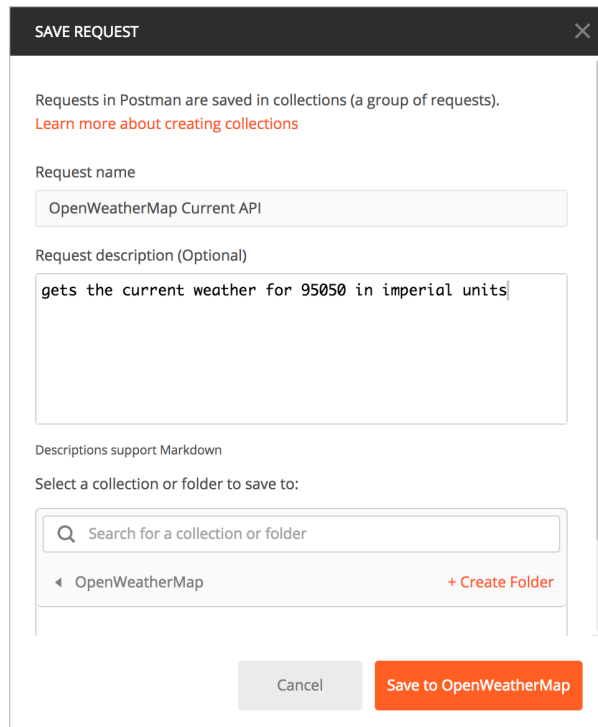
The response appears in the lower pane. For example:



### Save the request


1. In Postman, click the **Save** button (next to Send). The Save Request dialog box appears.
2. In the **Request Name** box, type a friendly name for the request, such as "OpenWeatherMap Current API."
3. In the **Request description (Optional)** field, type a description such as "gets the current weather for 95050 in imperial units."
4. Scroll down a bit and click **Create collection** to create a folder to save the request in. Name your new collection (e.g., "OpenWeatherMap") and click the check mark. Then select the new collection you just created.

After you create the collection, the Save button will be enabled. Your Postman collection should look something like this:



Collection dialog box

5. Click **Save to [collection name]**

Saved endpoints appear in the left side pane under Collections. (If you don't see the Collections pane, click the **Show/Hide Sidebar** button  in the lower-left corner to expand it.)

### Make a request for the OpenWeatherMap 5 day forecast

Now instead of getting the current weather, let's use another OpenWeatherMap endpoint to get the forecast. Enter details into Postman for the [5 day forecast request \(https://openweathermap.org/forecast5\)](https://openweathermap.org/forecast5). In Postman, you can click a new tab, or click the arrow next to Save and choose **Save As**. Then choose your collection and request name.

A sample endpoint for the 5 day forecast, which specifies location by zip code, looks like this:

```
https://api.openweathermap.org/data/2.5/forecast?zip=95050,us
```

Add in the query parameters for the API key and units:

```
https://api.openweathermap.org/data/2.5/forecast?zip=95050&appid=APIKEY&units=imperial
```

(In the above code, replace out **APIKEY** with your own API key.)

Observe how the response contains a **list** that provides the forecast details for five days.

### Make one more OpenWeatherMap API request

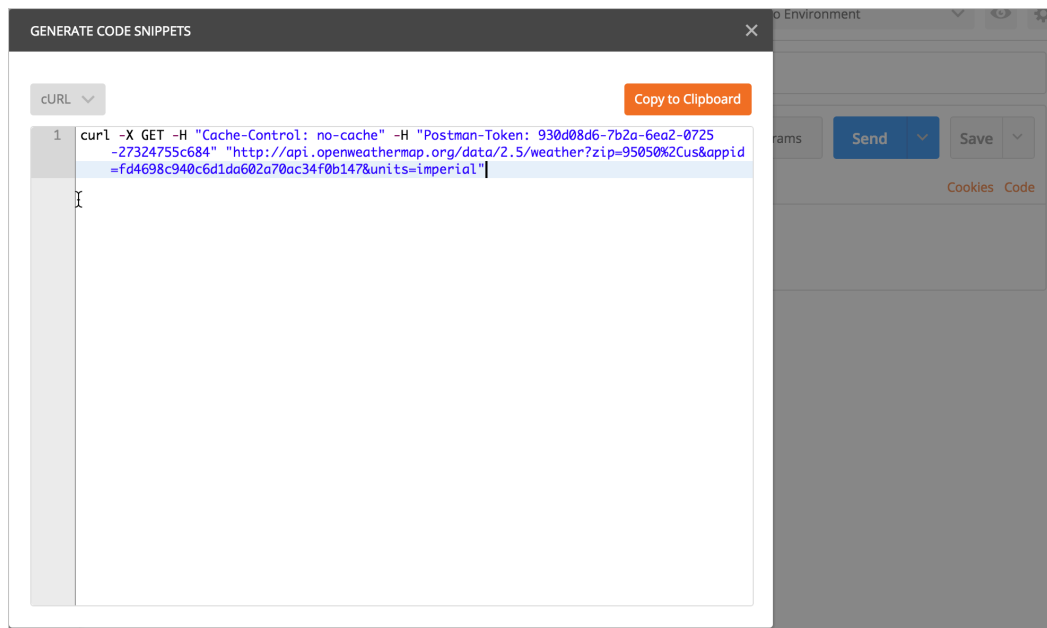
Make one more OpenWeatherMap API request, this time changing the way you specify the location. Instead of specifying the location by zip code, specify the location using `lat` and `lon` geocoordinates instead. For example:

```
https://api.openweathermap.org/data/2.5/weather?lat=37.3565982&lon=-121.9689848&units=imperial&appid=fd4698c940c6d1da602a70ac34f0b147
```

Postman has a lot of other functionality you can use. We'll revisit Postman later in the course for some other activities.

# Activity 2d: Make the OpenWeatherAPI request using curl

1. Assuming you completed the exercises in the [Postman tutorial \(https://idratherbewriting.com/learnapidoc/docapis\\_postman.html\)](https://idratherbewriting.com/learnapidoc/docapis_postman.html), go back into Postman.
2. On any call you've configured, and below the Save button in Postman, click the **Code** link.
3. In the Generate Code Snippets dialog box, select **cURL** from the drop-down list, and then click **Copy to Clipboard**.



The Postman code for the OpenWeatherMap weather request in curl looks as follows:

```
curl -X GET \
  'https://api.openweathermap.org/data/2.5/weather?zip=95050&units=imperial&appid=fd4698c940c6d1da602a70ac34f0b147' \
  -H 'Accept: */*' \
  -H 'Accept-Encoding: gzip, deflate' \
  -H 'Cache-Control: no-cache' \
  -H 'Connection: keep-alive' \
  -H 'Host: api.openweathermap.org' \
  -H 'Postman-Token: 8a9aeae7-f063-42e8-b0e3-09d1a7069bd5,62f56707-3a65-4d68-a774-8e677ef4487e' \
  -H 'User-Agent: PostmanRuntime/7.15.2' \
  -H 'cache-control: no-cache'
```

Postman adds its own header information (designated with **-H**). Do the following:

- Remove all the header (**-H**) tags.
- Remove the backslashes (**\**) (these are just added for readability).

- Put everything on one line
- If you're on Windows, change the single quotation marks to double quotation marks.

Here's the same curl call with these modifications:

```
curl -X GET "https://api.openweathermap.org/data/2.5/weather?zip=95050&units=imperial&appid=fd4698c940c6d1da602a70ac34f0b147"
```

4. curl is available on Mac and Windows 10 by default. (If you're on an older Windows machine that doesn't have curl, see [installing curl here \(http://www.confusedbycode.com/curl/#downloads\)](http://www.confusedbycode.com/curl/#downloads) for details.)
5. Go to your Terminal (Mac) or Command Prompt (Windows).

You can open your Terminal / Command Prompt by doing the following:

- If you're on Windows, go to **Start** and search for **cmd** to open up the command prompt. Paste in the curl request and then press **Enter**. (If you can't paste it in, look for paste options on the right-click menu.)
- If you're on a Mac, open Terminal by pressing **Cmd + spacebar** and typing **Terminal**. (Or go to **Applications > Utilities > Terminal**). (You could also use [iTerm \(https://www.iterm2.com/\)](https://www.iterm2.com/).) Paste in the curl request and then press **Enter**.

The response from the OpenWeatherMap weather request should look as follows:

```
{"coord":{"lon":-121.95,"lat":37.35},"weather":[{"id":802,"main":"Clouds","description":"scattered clouds","icon":"03d"}],"base":"station s","main":{"temp":68.34,"pressure":1014,"humidity":73,"temp_min":63,"temp_max":72},"visibility":16093,"wind":{"speed":3.36},"clouds":{"all":40},"dt":1566664878,"sys":{"type":1,"id":5122,"message":0.0106,"country":"US","sunrise":1566653501,"sunset":1566701346},"timezon e":-25200,"id":0,"name":"Santa Clara","cod":200}
```

This response is minified. You can un-minify it by going to a site such as [JSON pretty print \(http://jsonprettyprint.com/\)](http://jsonprettyprint.com/), or if you have [Python installed \(https://www.python.org/downloads/\)](https://www.python.org/downloads/), you can add `| python -m json.tool` at the end of your cURL request to minify the JSON in the response (see [this Stack Overflow thread \(https://stackoverflow.com/questions/352098/how-can-i-pretty-print-json-in-a-unix-shell-script\)](https://stackoverflow.com/questions/352098/how-can-i-pretty-print-json-in-a-unix-shell-script) for details).

6. If you want additional practice, make a similar curl request for the 5 day forecast request that you also have in Postman. And another curl request for the third OpenWeatherMap API request you made in Postman.

# Activity 2e: Make an API request on a web page

For this activity, you'll use JavaScript to display the API response on a web page. Specifically, you'll use some auto-generated jQuery code from Postman to create the AJAX request. You'll get the wind speed from the response and print it to the page.

1. Open a text editor such as Sublime Text.
2. Paste in the following code:

```
<html>
  <meta charset="UTF-8">
  <head>
    <title>Sample page</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.1
1.1/jquery.min.js"></script>
    <script>
      var settings = {
        "async": true,
        "crossDomain": true,
        "url": "https://api.openweathermap.org/data/2.5/weather?zi
p=95050&units=imperial&appid=fd4698c940c6d1da602a70ac34f0b147",
        "method": "GET"
      }

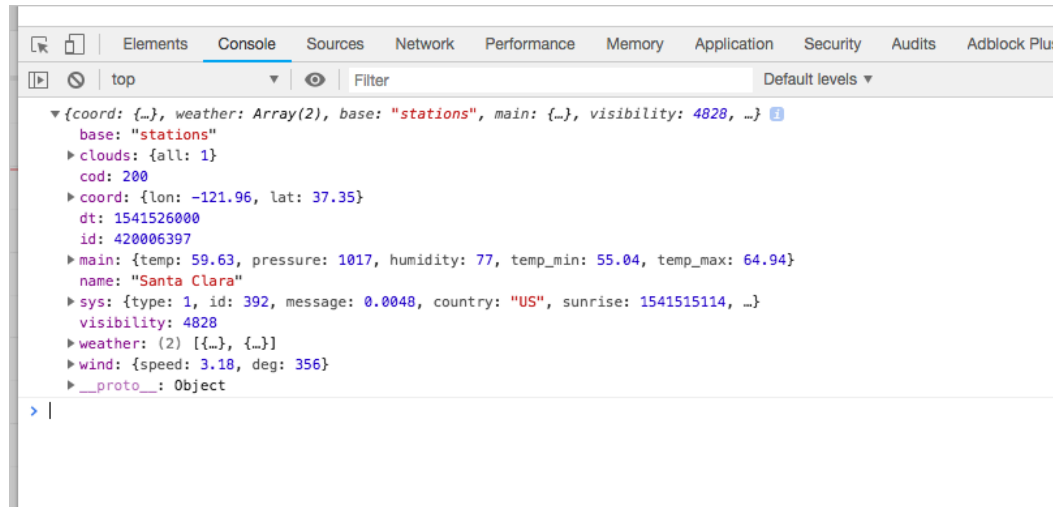
      $.ajax(settings).done(function (response) {
        console.log(response);
        var content = response.wind.speed;
        $("#windSpeed").append(content);
      });

    </script>
  </head>
  <body>
    <h1>Sample Page</h1>
    wind speed: <span id="windSpeed"></span>
  </body>
</html>
```

This code is explained in more detail in a section below.

3. Save the file as an HTML file named `weather.html`.
4. Start **Chrome** and open the JavaScript Console by going to **View > Developer > JavaScript Console**.
5. In Chrome, press **Cmd/Ctrl + O** and select your `weather.html` file.

The weather response should be logged to the JavaScript Console (due to the `console.log(response)` code in the request). If you expand the object returned to the console, it will look as follows:



One of the properties in the response is `wind.speed`. The wind speed is shown on the page as well.

### Step-by-step explanation

The above activity simply had you paste a chunk of prewritten code onto a web page, without much explanation. In this section, we'll step through that code with details about how it was assembled. However, we won't dive too deeply into JavaScript and jQuery here, as this is somewhat beyond the scope of the instruction.

To create the web page code from scratch:

1. In a text editor (such as Sublime Text), create a new HTML file and paste in the following boilerplate template (which contains basic HTML tags and a reference to jQuery):

```

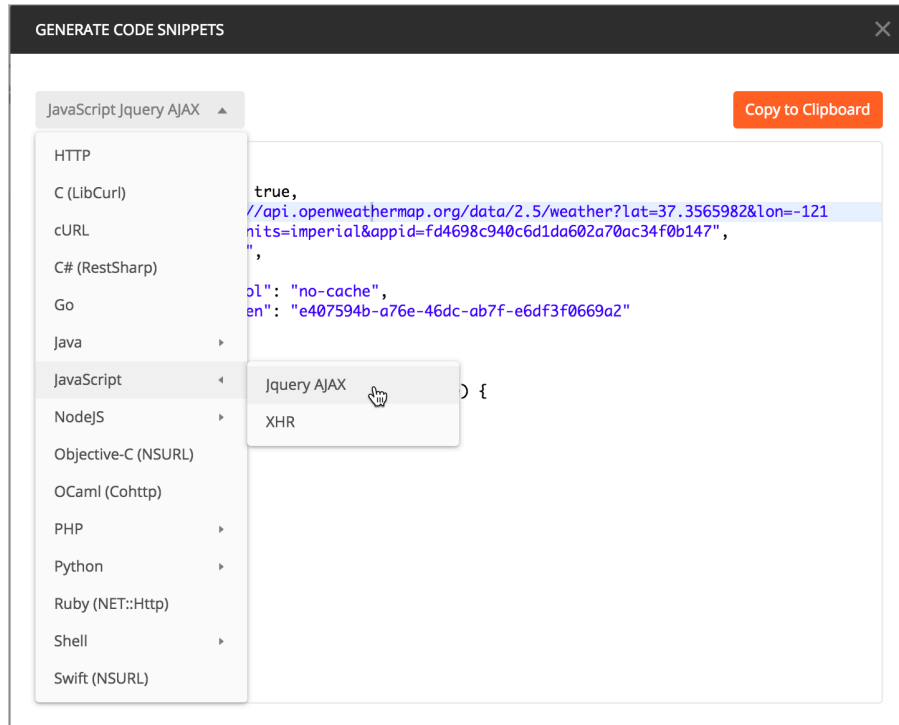
<html>
<meta charset="UTF-8">
  <head>
    <title>Sample page</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.1
1.1/jquery.min.js"></script>
  </head>
<body>
  <h2>Sample page</h2>

</body>
</html>

```

jQuery is necessary because we're using the `ajax` method to make the API request.

2. Save your file (anywhere convenient) with a name such as **weather.html**.
3. Open Postman and go to the Current weather data ( `weather` ) endpoint that you configured in an earlier activity (see [Submit requests through Postman \(https://idratherbewriting.com/learnapidoc/docapis\\_postman.html\)](https://idratherbewriting.com/learnapidoc/docapis_postman.html) for more information).
4. Click the **Code** link (below the Save button), and then select **JavaScript > jQuery AJAX**.



JavaScript Ajax code snippet

The AJAX code looks as follows:

```
var settings = {
  "async": true,
  "crossDomain": true,
  "url": "https://api.openweathermap.org/data/2.5/weather?zip=95050&units=imperial&appid=fd4698c940c6d1da602a70ac34f0b147",
  "method": "GET",
  "headers": {
    "User-Agent": "PostmanRuntime/7.15.2",
    "Accept": "*/*",
    "Cache-Control": "no-cache",
    "Postman-Token": "8a9aeae7-f063-42e8-b0e3-09d1a7069bd5,5468d865-c341-4596-9acc-faba0e0c0c7d",
    "Host": "api.openweathermap.org",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "keep-alive",
    "cache-control": "no-cache"
  }
}

$.ajax(settings).done(function (response) {
  console.log(response);
});
```

- Click **Copy to Clipboard** to copy the code sample.

- In your HTML file, insert the copied code inside a pair of `<script></script>` inside the `header` tags.
- In the jQuery code, remove the entire `headers` object from the Postman code:

```
"headers": {
  "User-Agent": "PostmanRuntime/7.15.2",
  "Accept": "*/*",
  "Cache-Control": "no-cache",
  "Postman-Token": "8a9aeae7-f063-42e8-b0e3-09d1a7069bd5,5468d865-c341-4596-9acc-faba0e0c0c7d",
  "Host": "api.openweathermap.org",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "cache-control": "no-cache"
}
```

- Remove the extra comma after `"method": "GET"` to keep the JSON valid.
- Below `console.log(response);` (but still inside the function's closing brace `});`), create a variable called `content` and set it equal to `response.wind.speed`. Then use the jQuery `append` method to append `content` to an element called `#windSpeed`:

```
$.ajax(settings).done(function (response) {
  console.log(response);
  var content = response.wind.speed;
  $("#windSpeed").append(content);
});
```

When `ajax` retrieves the response from the API, it assigns it to `response`. You can access the properties in the `response` using [dot notation](https://idratherbewriting.com/learnapidoc/docapis_diving_into_dot_notation.html) ([https://idratherbewriting.com/learnapidoc/docapis\\_diving\\_into\\_dot\\_notation.html](https://idratherbewriting.com/learnapidoc/docapis_diving_into_dot_notation.html)). The `append` method in jQuery allows you to append content to a particular element.

- Create an element on the page (below the `h1` tags) with the `id="windSpeed"`:

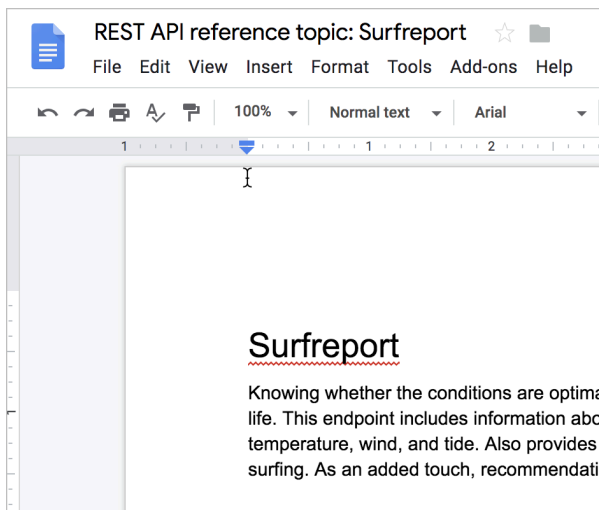
```
<body>
  <h1>Sample Page</h1>
  wind speed: <span id="windSpeed"></span>
</body>
```

- Save the file and open it in Chrome. Open the JS Console to view the object logged.

You can view the file here: [weather-plain.html](https://idratherbewriting.com/learnapidoc/assets/files/weather-plain.html) (<https://idratherbewriting.com/learnapidoc/assets/files/weather-plain.html>).

# Activity 3a: What's wrong with this API reference topic

The following is a sample API reference topic for an endpoint called `surfreport`. There are about 25 things wrong in the topic. A copy of this same content is available in a read-only [Google Doc here](https://idratherbewriting.site/whats-wrong-api) (<https://idratherbewriting.site/whats-wrong-api>). In Google Docs, go to **File > Make a Copy** to create your own instance. Then make comments in Google docs to note as many errors as you can find.



*Make a copy of this Google doc and make comments on it.*

Here's the same doc (with problems) displayed on the web:

## Surfreport

Knowing whether the conditions are optimal for surfing is a necessary detail in every surfer's life. This endpoint includes information about surfing conditions, including the surf height, water temperature, wind, and tide. Also provides an overall recommendation about whether to go surfing. As an added touch, recommendations are expressed in surfer lingo. Surf's up!

### Endpoints

GET/POST `surfreport/{:beachId}`

Gets the surf conditions for a specific beach ID.

### Parameters

Parameter	Use	Description	Type of data
-----------	-----	-------------	--------------

<code>{beachId}</code>	Required	Refers to the ID for the beach you want to look up.	Number
<code>days</code>	Optional	The number of days to include in the response. Default is 3. Max 10.	Integer
<code>time</code>	Optional	The time you want the report for.	Integer. ISO 8601 format. Example: <code>20180915T155300+0500</code>

## Sample request

```
https://api.openweathermap.org/data/2.5/surfreport/12345?zip=95050&appid=fd4698c940c6d1da602a70ac34f0b147&days=1
```

## Sample response

```
{
  "surfreport": [
    {
      "beach": "Santa Cruz",
      "monday": {
        "1pm": {
          "tide": 5,
          "wind": 15,
          "watertemp": 80,
          "surf_height": 5,
          "riptide": "moderate",
          "recommendation": "Carve it up, brah! The waves are cran
kin' wild out there."
        },
        "2pm": {
          "tide": -1,
          "wind": 1,
          "watertemp": 50,
          "surf_height": 3,
          "riptide": extreme
          "recommendation": "Waves are foam and frothy but rideabl
e in places. Gravitare to the impact zone, due, and hang loose."
        },
        "3pm": {
          "tide": -1,
          "wind": 10,
          "watertemp": 65,
          "surf_height": 1,
          "recommendation": "Scene is blown out. Bail inland and c
hill on the beach instead or you'll the one who'll be shredded, due."
        }
      }
    }
  ]
}
```

## Response definitions

The following table describes each item in the response.

Response item	Description	Data type
beach	The beach you selected based on the beach ID in the request. The beach name is the official name as described in the National Park Service Geodatabase.	String

<code>{day}</code>	The day(s) of the week requested.	object
<code>{time}</code>	The time for the conditions.	string
<code>tide</code>	The level of tide at the beach for a specific day and time. Tide is the distance inland that the water rises to, and can be a positive or negative number. When the tide is out, the number is negative. When the tide is in, the number is positive. The 0 point reflects the line when the tide is neither going in nor out but is in transition between the two states.	String
<code>wind</code>	The wind speed at the beach. Wind affects the surf height and general wave conditions. Wind speeds of more than 15 make surf conditions undesirable because the wind creates white caps and choppy waters.	Int
<code>watertemp</code>	The temperature of the water. Water temperatures below 70 usually require you to wear a wetsuit. With temperatures below 60, you will need at least a 3mm wetsuit and preferably booties to stay warm.	String
<code>surfheight</code>	The height of the waves, returned in either feet or centimeters depending on the units you specify. A surf height of 3 feet is the minimum size needed for surfing. If the surf height exceeds 10 feet, it is not safe to surf.	Map
<code>recommendation</code>	An overall recommendation based on a combination of the various factors (wind, watertemp, surfheight), etc.	String

## Answers

You can view the answer key here: [What's wrong with this topic answer key \(https://idratherbewriting.com/learnapidoc/whats\\_wrong\\_answer\\_key.html\)](https://idratherbewriting.com/learnapidoc/whats_wrong_answer_key.html).

# Activity 3b: Evaluate API reference docs for core elements

In this activity, you'll review API reference documentation and identify the common elements. To evaluate the API reference docs:

1. Either from an [open-source project \(https://idratherbewriting.com/learnapidoc/docapis\\_find\\_open\\_source\\_project.html\)](https://idratherbewriting.com/learnapidoc/docapis_find_open_source_project.html) you might have identified or from this [list of about 100 API doc sites here \(https://idratherbewriting.com/learnapidoc/pubapis\\_apilist.html#list\\_api\\_doc\\_sites\)](https://idratherbewriting.com/learnapidoc/pubapis_apilist.html#list_api_doc_sites), identify an API documentation site to analyze.
2. In the API documentation, look for the API reference documentation section (the list of endpoints).
3. In the reference documentation, identify each of the following sections:
  - Resource description ([https://idratherbewriting.com/learnapidoc/docapis\\_resource\\_descriptions.html](https://idratherbewriting.com/learnapidoc/docapis_resource_descriptions.html))
  - Endpoints and methods ([https://idratherbewriting.com/learnapidoc/docapis\\_resource\\_endpoints.html](https://idratherbewriting.com/learnapidoc/docapis_resource_endpoints.html))
  - Parameters ([https://idratherbewriting.com/learnapidoc/docapis\\_doc\\_parameters.html](https://idratherbewriting.com/learnapidoc/docapis_doc_parameters.html))
  - Request example ([https://idratherbewriting.com/learnapidoc/docapis\\_doc\\_sample\\_requests.html](https://idratherbewriting.com/learnapidoc/docapis_doc_sample_requests.html))
  - Response example and schema ([https://idratherbewriting.com/learnapidoc/docapis\\_doc\\_sample\\_responses\\_and\\_schema.html](https://idratherbewriting.com/learnapidoc/docapis_doc_sample_responses_and_schema.html))

The section names will probably differ in the API doc sites you find, but they're usually recognizable to some degree. If you're finding it somewhat difficult to locate them, this is part of the wild west of terminology and organization when it comes to API documentation.

4. Assess the API reference documentation by answering the following questions for each section:

**Resource description:**

- Is the description action-oriented?
- Is it a brief 1-3 sentence summary?

**Endpoints and methods:**

- How are the endpoints grouped? (Are they listed all on the same page, or on different pages? Are they grouped by method, or by resource?)
- How are the methods specified for each endpoint?

**Parameters:**

- How many types of parameters are there (header, path, query string, and request body parameters) for the endpoints?
- Are the data types (string, boolean, etc.) defined for each parameter? Are max/min values noted?

**Request example:**

- In what format or language is the request shown (e.g. curl, specific languages, other)?
- How many parameters does the sample request include?

**Response example:**

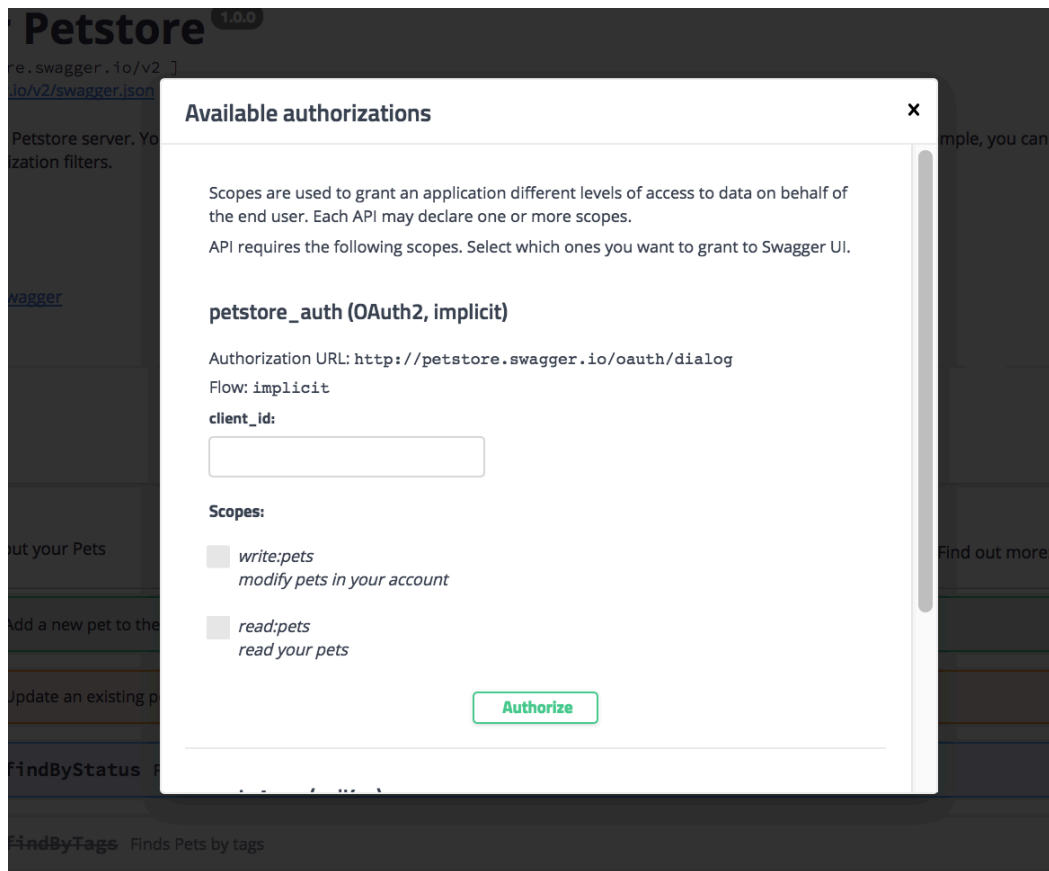
- Is there both a sample response *and* a response schema? (And is each element in the response actually described?)
- How does the doc site handle nested hierarchies in the response definitions?

# Activity 4a: Explore Swagger UI through the Petstore Demo

Let's get some hands-on experience with Swagger UI using the Petstore demo. The Petstore demo provides a good example of how the OpenAPI specification can be rendered visually.

1. Go to the [Swagger Pet Store Demo \(https://petstore.swagger.io/\)](https://petstore.swagger.io/).

As with most Swagger-based outputs, Swagger UI provides a “Try it out” button. To make it work, you must first authorize Swagger by clicking **Authorize** and entering your API key in the Authorization modal. However, the Petstore authorization modal is just for demo purposes. There isn't any real code authorizing those requests, so you can close the Authorization modal or skip it altogether.



Authorization modal in Swagger UI

2. Expand the **Pet** endpoint.
3. Click **Try it out**.

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

**POST** /pet Add a new pet to the store

Parameters

**Name** **Description**

**body** \* required Pet object that needs to be added to the store

(body)

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/xml

Try it out

*Try it out button in Swagger UI*

After you click Try it out, the example value in the Request Body field becomes editable.

4. In the example value, change the first **id** value to a unique (and unlikely repeated) whole number. Change the name **doggie** to a pet name you can remember (e.g., **Bentley**).
5. Click **Execute**.

**POST** /pet Add a new pet to the store

Parameters

**Name** **Description**

**body** \* required  
(body)

Pet object that needs to be added to the store

Example Value Model

```
{
  "id": 193844,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Bentley",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Parameter content type  
application/json

Execute

*Executing a sample Petstore request*

Swagger UI submits the request and shows the [curl \(https://idratherbewriting.com/learnapidoc/docapis\\_make\\_curl\\_call.html\)](https://idratherbewriting.com/learnapidoc/docapis_make_curl_call.html) that was submitted. For example, here's the curl Swagger UI sent:

```
curl -X POST "https://petstore.swagger.io/v2/pet" -H "accept: application/xml" -H "Content-Type: application/json" -d "{ \"id\": 1000, \"category\": { \"id\": 0, \"name\": \"string\" }, \"name\": \"Bentley\", \"photoUrls\": [ \"string\" ], \"tags\": [ { \"id\": 0, \"name\": \"string\" } ], \"status\": \"available\"}"
```

Notice that, with the **-d** (data) parameter, the request body parameter is escaped and added directly into the curl command rather than being loaded from a file (as explained in [Common curl commands related to REST \(https://idratherbewriting.com/learnapidoc/docapis\\_understand\\_curl.html#common\)](https://idratherbewriting.com/learnapidoc/docapis_understand_curl.html#common)).

The Responses section in Swagger UI shows the response from the server. By default, the response returns XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Pet>
  <category>
    <id>0</id>
    <name>string</name>
  </category>
  <id>1000</id>
  <name>Bentley</name>
  <photoUrls>
    <photoUrl>string</photoUrl>
  </photoUrls>
  <status>available</status>
  <tags>
    <tag>
      <id>0</id>
      <name>string</name>
    </tag>
  </tags>
</Pet>
```

If you select JSON rather than XML in the “Response content type” drop-down box, you can specify that JSON is returned rather than XML.

The screenshot shows the Swagger UI interface with the 'Responses' tab selected. The 'Response content type' dropdown is set to 'application/json'. The 'Curl' section displays a curl command for a POST request to 'http://petstore.swagger.io/v2/pet' with headers for 'accept' and 'Content-Type' set to 'application/json'. The 'Request URL' is 'http://petstore.swagger.io/v2/pet'. The 'Server response' section shows a 200 status code with the response body as a JSON object:

```
{
  "id": 193844,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Bentley",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

(<http://petstore.swagger.io/>)

- The Petstore is a functioning API, and you have actually created a pet. For fun, expand the GET `/pet/{petId}` endpoint, click **Try it out**, enter the pet `id` you used in the previous operation, and then execute the request. You should see your pet's name returned.

# Activity 4b: Edit an existing OpenAPI specification document

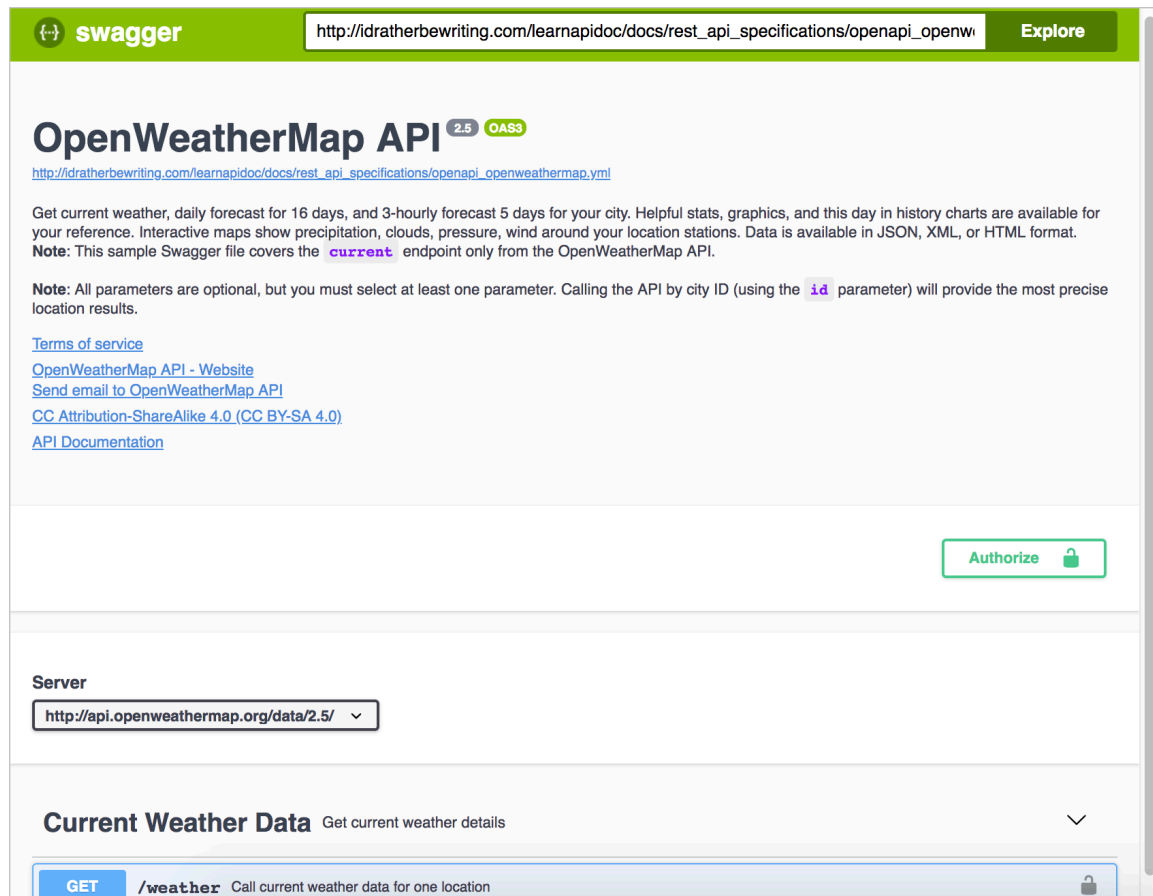
Use this simple [Sunrise and sunset times API](https://sunrise-sunset.org/api) (<https://sunrise-sunset.org/api>) to get more familiar with the process of creating an OpenAPI specification file. This Sunrise and sunset times API doesn't require authentication with requests, so it removes some of the more complicated authentication workflows (the spec file doesn't require the [security object](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step6_security_object.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_step6\\_security\\_object.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step6_security_object.html))). In this activity, you'll edit some of the existing values in an OpenAPI specification document that's already written.

To edit the OpenAPI specification file:

1. Copy the code from this [pre-built OpenAPI specification](https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi_sunrise_sunset.yml) ([https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi\\_sunrise\\_sunset.yml](https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi_sunrise_sunset.yml)).
2. Paste the YAML content into the [Swagger Editor](https://editor.swagger.io/) (<https://editor.swagger.io/>).
3. Identify each of the root-level objects of the OpenAPI spec:
  - [Step 1: The openapi object](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step1_openapi_object.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_step1\\_openapi\\_object.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step1_openapi_object.html))
  - [Step 2: The info object](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step2_info_object.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_step2\\_info\\_object.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step2_info_object.html))
  - [Step 3: The servers object](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step3_servers_object.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_step3\\_servers\\_object.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step3_servers_object.html))
  - [Step 4: The paths object](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step4_paths_object.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_step4\\_paths\\_object.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step4_paths_object.html))
  - [Step 5: The components object](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step5_components_object.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_step5\\_components\\_object.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step5_components_object.html))
  - [Step 8: The externalDocs object](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step8_externaldocs_object.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_step8\\_externaldocs\\_object.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_step8_externaldocs_object.html))
4. In the **info** object (near the top), make some changes to the **description** property and see how the visual display in the right column gets updated.
5. In the **parameters** object, make some changes to one of the **description** properties and see how the visual editor gets updated.
6. Look for the **\$ref** pointer in the **response** object. Identify what it points to in **components**.
7. Change some spacing in a way that makes the spec invalid (such as inserting a space before **info**), and look at the error that appears. Then revert the invalid space.
8. Expand the **Get** section and click **Try it out**. Then click **Execute** and look at the response.

# Create a Swagger UI display with an OpenAPI spec document

In this activity, you'll create a Swagger UI display for an OpenAPI specification document. If you're using one of the pre-built OpenAPI files, you can see a demo of what we'll build here: [OpenWeatherMap Swagger UI](https://idratherbewriting.com/learnapidoc/assets/files/swagger/) (<https://idratherbewriting.com/learnapidoc/assets/files/swagger/>) or [Sunrise/sunset Swagger UI](https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/index.html) (<https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/index.html>).



Demo of Swagger UI rendering an OpenWeatherMap OpenAPI specification document

## To integrate your OpenAPI spec into Swagger UI:

1. Prepare a valid OpenAPI specification document:
  - For instructions on creating an OpenAPI specification document from scratch, follow the [OpenAPI tutorial here](https://idratherbewriting.com/learnapidoc/pubapis_openapi_tutorial_overview.html) ([https://idratherbewriting.com/learnapidoc/pubapis\\_openapi\\_tutorial\\_overview.html](https://idratherbewriting.com/learnapidoc/pubapis_openapi_tutorial_overview.html)).
  - To use a pre-built OpenAPI specification document, you can use the [OpenWeatherMap spec file](https://idratherbewriting.com/learnapidoc/docs/rest_api_specifications/openapi_openweathermap.yml) ([https://idratherbewriting.com/learnapidoc/docs/rest\\_api\\_specifications/openapi\\_openweathermap.yml](https://idratherbewriting.com/learnapidoc/docs/rest_api_specifications/openapi_openweathermap.yml)) or the [Sunrise/sunset API spec file](https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi_sunrise_sunset.yml) ([https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi\\_sunrise\\_sunset.yml](https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi_sunrise_sunset.yml)). (Right-click the link and save the YAML file to your

desktop.)

2. Make sure your OpenAPI specification is valid. Paste your OpenAPI specification code into the [Swagger online editor](http://editor.swagger.io/#/) (<http://editor.swagger.io/#/>) and make sure no warnings appear. The view on the right of the Swagger Editor shows a fully functional Swagger UI display.
3. Go to the [Swagger UI GitHub project](https://github.com/swagger-api/swagger-ui) (<https://github.com/swagger-api/swagger-ui>).
4. Click **Clone or download**, and then click **Download ZIP**. Download the files to a convenient location on your computer and extract the files.

The only folder you'll be working with in the downloaded zip is the `dist` folder (short for distribution). Everything else is used only if you're recompiling the Swagger files, which is beyond the scope of this tutorial.

5. Drag the `dist` folder out of the `swagger-ui-master` folder so that it stands alone. (Then optionally delete the `swagger-ui-master` folder and zip file.)
6. Drag your OpenAPI specification file (from step 1) into the `dist` folder. (If you're using the pre-build OpenAPI files, the file is called either `openapi_openweathermap.yml` or `openapi_sunrise_sunset.yml`.) Your file structure should look as follows:

```
├── dist
│   ├── favicon-16x16.png
│   ├── favicon-32x32.png
│   ├── index.html
│   ├── oauth2-redirect.html
│   ├── swagger-ui-bundle.js
│   ├── swagger-ui-bundle.js.map
│   ├── swagger-ui-standalone-preset.js
│   ├── swagger-ui-standalone-preset.js.map
│   ├── swagger-ui.css
│   ├── swagger-ui.css.map
│   ├── swagger-ui.js
│   ├── swagger-ui.js.map
│   ├── swagger30.yml
│   └── [your openapi specification file]
```

7. Inside your `dist` folder, open `index.html` in a text editor such as [Atom editor](https://atom.io/) (<https://atom.io/>) or [Sublime Text](https://www.sublimetext.com/) (<https://www.sublimetext.com/>).
8. Look for the following code:

```
url: "http://petstore.swagger.io/v2/swagger.json",
```

9. Change the `url` value from `http://petstore.swagger.io/v2/swagger.json` to a relative path to your YAML file, and then save the file. For example:

```
url: "openapi_openweathermap.yml",
```

or

```
url: "openapi_sunrise_sunset.yml",
```

10. View the `index.html` file locally in your browser using a Python simple server. See the

instructions in the next section for doing this.

### View the Swagger UI file in a Python simple server

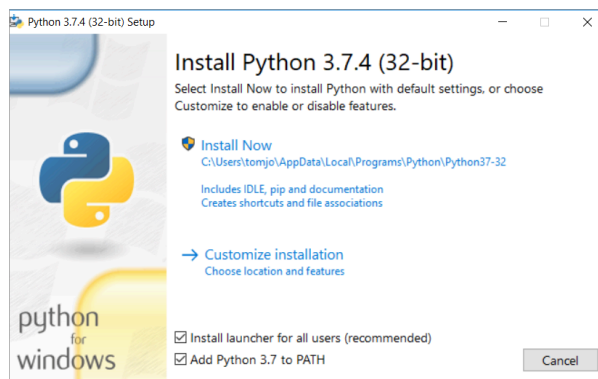
CORS security restrictions in Chrome and Firefox prevent you from viewing the Swagger UI file locally. Swagger UI needs to load on a web server to fulfill the security requirements. You can create a local web server running on your computer through [Python's SimpleHTTPServer module](#) ([https://developer.mozilla.org/en-US/docs/Learn/Common\\_questions/set\\_up\\_a\\_local\\_testing\\_server](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server)). Mac has a system version of Python installed by default, but Windows computers will need to install Python.

#### Windows: Run the Python simpler server:

1. Install Python 3x.

You can check if you already have Python by typing `python --version` on a command prompt. If you don't have Python already, [download and install Python 3x](#) (<https://www.python.org/downloads/>).

When you install Python, be sure to select the check box that says "Add Python 3.7 to PATH." *This check box isn't selected by default. If you don't select it, your command prompt won't recognize the word "python".*



*Add Python 3.7 to PATH*

2. After installing Python, close your command prompt and reopen it.
3. In your command prompt, browse to the Swagger UI `dist` directory.

To browse in the Windows command prompt, type `cd <folder name>` to move into the folder. Type `cd ..` to move up a directory. Type `dir` to see a list of the current directory's contents.

If you're having trouble locating the `dist` directory in the command prompt, try this trick: type `cd`, press the spacebar, and then drag the `dist` folder directly into the command prompt. The path will be printed automatically.

4. After you've navigated into the `dist` folder, launch the server:

```
python -m http.server
```

The server starts:

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

5. Copy the `http://0.0.0.0:8000/` path to your browser's address bar. This is the web server address.

By default, web servers default to the `index.html` file in the directory, so it will show the Swagger UI file automatically. If the browser doesn't direct to `index.html`, add it manually: `http://0.0.0.0:8000/index.html`.

To stop the server, press **Ctrl+C** in your command prompt.

### Mac: Run the Python simple server

1. In your terminal, browse to the Swagger UI `dist` directory.

To browse in your terminal, type `cd <folder name>` to move into the folder. Type `cd ..` to move up a directory. Type `ls` to see a list of the current directory's contents.

If you're having trouble locating the `dist` directory in the command prompt, try this trick: type `cd`, press the spacebar, and then drag the `dist` folder directly into the command prompt. The path will be printed automatically.

2. Since Mac already has Python, you can just run the following in your terminal to launch simple server:

```
python -m http.server
```

The server starts:

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

3. Copy the `http://0.0.0.0:8000/` path to your browser's address bar. This is the web server address.

By default, web servers default to the `index.html` file in the directory, so it will show the Swagger UI file automatically. If the browser doesn't direct to `index.html`, add it manually: `http://0.0.0.0:8000/index.html`.

To stop the server, press **Ctrl+C** in your terminal.

For more details on using the Python simple server, see [How do you set up a local testing server?](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server) ([https://developer.mozilla.org/en-US/docs/Learn/Common\\_questions/set\\_up\\_a\\_local\\_testing\\_server](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server)) for more details.

### Other notes

If you can't get the Python simple server to work, you can simply upload the OpenAPI files to a web server and refer to the path there. If you prefer this approach, you can copy the path to `openapi_openweathermap.yml` ([https://idratherbewriting.com/learnapidoc/docs/rest\\_api\\_specifications/openapi\\_openweathermap.yml](https://idratherbewriting.com/learnapidoc/docs/rest_api_specifications/openapi_openweathermap.yml)) or `openapi_sunrise_sunset.yml` ([https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi\\_sunrise\\_sunset.yml](https://idratherbewriting.com/learnapidoc/assets/files/swagger-sunrise-sunset/openapi_sunrise_sunset.yml)).

When you're ready to publish your Swagger UI file, you just upload the `dist` folder (or whatever you want to call it) to a web server and go to the `index.html` file. For example, if you called your directory `dist` (leaving it unchanged), you would go to `http://myserver.com/dist/`.

For more instructions in working with Swagger UI, see the [Swagger.io docs](https://swagger.io/docs/open-source-tools/swagger-ui/usage/installation/) (<https://swagger.io/docs/open-source-tools/swagger-ui/usage/installation/>).

# Activity 6a: Judge conceptual content and decide which is best

The following are 6 common conceptual topics in API documentation. For each topic, examine the three examples and decide which one is best.

## API overview

- Box (<https://developer.box.com/docs/overview>)
- Hootsuite (<https://developer.hootsuite.com/docs/the-hootsuite-platform>)
- Watson Assistant (<https://cloud.ibm.com/docs/services/assistant?topic=assistant-index#index>)

## API getting started

- Parse (<https://docs.parseplatform.org/parse-server/guide/>)
- Paypal (<https://developer.paypal.com/docs/api/overview/>)
- Google ([https://developers.google.com/adsense/management/getting\\_started](https://developers.google.com/adsense/management/getting_started))

## API authentication and authorization

- Sendgrid ([https://sendgrid.com/docs/User\\_Guide/Settings/api\\_keys.html](https://sendgrid.com/docs/User_Guide/Settings/api_keys.html))
- Twitter (<https://developer.twitter.com/en/docs/basics/authentication/guides/access-tokens.html>)
- AWS (<https://docs.aws.amazon.com/AWSECommerceService/latest/DG/HMACSignatures.html>)

## API status and error codes

- Clearbit (<https://clearbit.com/docs?python#errors-error-types>)
- Twitter (<https://developer.twitter.com/en/docs/basics/response-codes>)
- Mailchimp (<http://developer.mailchimp.com/documentation/mailchimp/guides/error-glossary/>)

## API rate limiting and thresholds

- GitHub ([https://developer.github.com/v3/rate\\_limit/](https://developer.github.com/v3/rate_limit/))
- LinkedIn (<https://docs.microsoft.com/en-us/linkedin/shared/api-guide/concepts/rate-limits?context=linkedin/consumer/context>)
- Bitly ([http://dev.bitly.com/rate\\_limiting.html](http://dev.bitly.com/rate_limiting.html))

## API quick reference

- Eventful (<http://api.eventful.com/docs>)
- Parse (<http://docs.parseplatform.org/rest/guide/#quick-reference>)
- Veracode ([https://help.veracode.com/reader/LMv\\_dtSHyb7ilxAQznC~9w/FhxRdiWf5qeJrtajmjGtpw](https://help.veracode.com/reader/LMv_dtSHyb7ilxAQznC~9w/FhxRdiWf5qeJrtajmjGtpw))

# Activity 7a: Classify code documentation into one of the five types

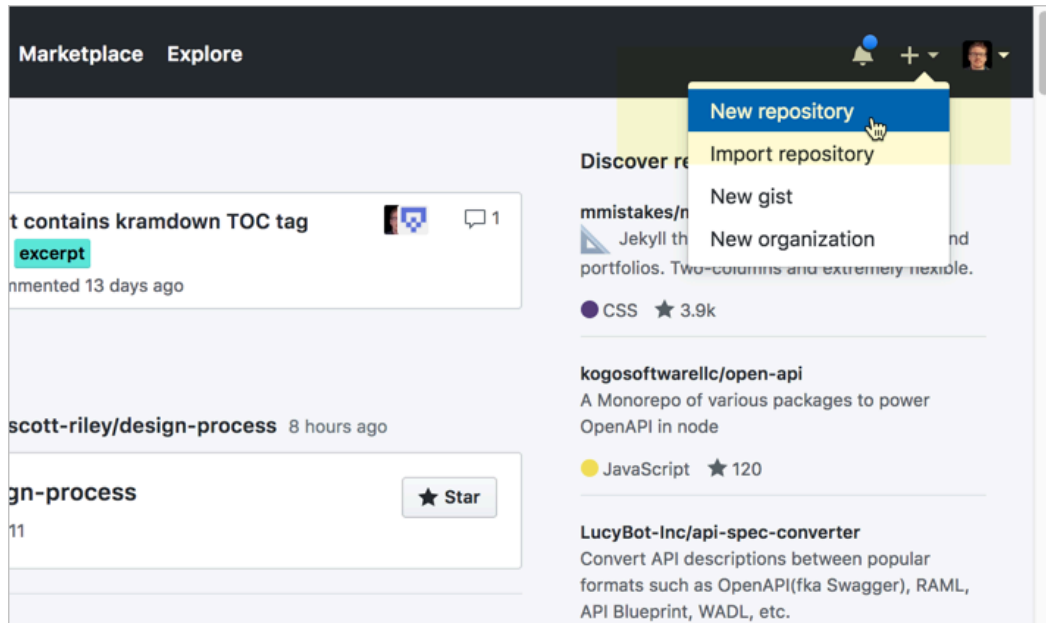
Find a code tutorial within one of the API documentation sites you've been exploring. Or look at one of the API doc sites in this post by Nordic APIs on [5 Examples of Excellent API Documentation \(and Why We Think So\)](https://nordicapis.com/5-examples-of-excellent-api-documentation/) (<https://nordicapis.com/5-examples-of-excellent-api-documentation/>). This post lists five API doc sets: Stripe (<https://stripe.com/docs/api>), Twilio (<https://www.twilio.com/docs/usage/api>), Dropbox (<https://www.dropbox.com/developers/documentation>), GitHub (<https://developer.github.com/v3/guides/getting-started/>), and Twitter (<https://developer.twitter.com/en/docs/tweets/search/overview>).

Find a code tutorial in the documentation. Figure out which approach the code documentation best aligns with.

# Activity 7a: Create a GitHub wiki and publish content on a sample page

In this section, you will create a new GitHub repo and publish a sample file there.

1. Create a GitHub account at [GitHub.com](http://github.com) (<http://github.com>).
2. Go to [GitHub](https://github.com) (<https://github.com>) and sign in. After you're signed in, click the + button in the upper-right corner and select **New repository**.



*Creating a new GitHub repository*

3. Give the repo a **Repository name**, a short **Description**, select **Public**, select **Initialize the repo with a README**, and then click **Create repository**. (Don't worry about selecting the license or gitignore settings for this activity.)
4. Click the **Wiki** tab on the top navigation bar of your new repository.
5. Click **Create the first page**. (Or if your wiki already has pages, click **New Page**.)
6. In the default page ("Home"), insert your own sample documentation content, preferably using Markdown syntax. Or grab the sample Markdown page of a [fake endpoint called surfreport here](https://idratherbewriting.com/learnapidoc/assets/files/surfreportendpointdoc.md) (<https://idratherbewriting.com/learnapidoc/assets/files/surfreportendpointdoc.md>) and insert it into the page.
7. In the **Edit message** box, type a description of what you updated (your commit message).
8. Click **Save Page**.

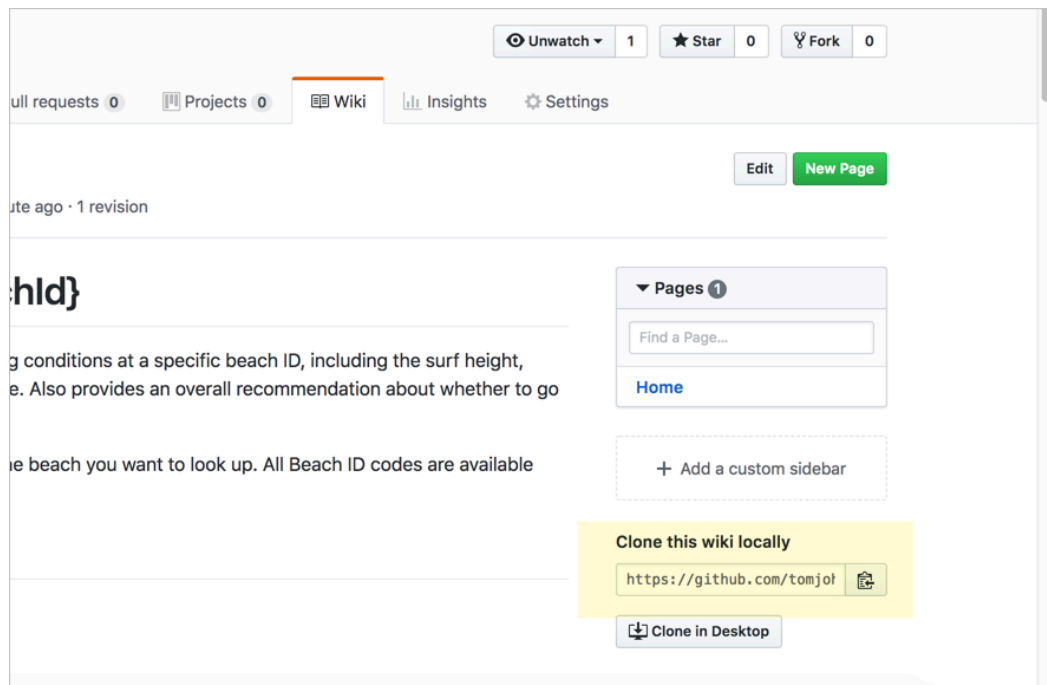
Notice how GitHub automatically converts the Markdown syntax into HTML and styles it in a readable way. You could work with this GitHub wiki entirely in the browser as a way for multiple people to collaborate and edit content. However, unlike other wikis, with GitHub you can also take all the content offline and edit locally, and then commit your changes and push the changes back online.

# Activity 7b: Clone your GitHub repo locally

So far you've been working with GitHub in the browser. Now we'll take the same content and work with it locally. This is what makes the GitHub wiki unique from other wikis — it's a Git repo, so you can manipulate the content the same way as any other Git repo (working locally, pushing, pulling, merging, branching, etc.).

To clone the GitHub repo locally:

1. If you don't already have Git installed, set it up on your computer. (You can check by typing `git --version` in your terminal window. See [Install Git \(https://idratherbewriting.com/learnapidoc/pubapis\\_github\\_wikis.html#git\\_install\)](https://idratherbewriting.com/learnapidoc/pubapis_github_wikis.html#git_install) for more information on installation.)
2. While viewing your the GitHub wiki in your browser, look for the section that says **Clone this wiki locally**. Click the clipboard button. (This copies the clone URL to your clipboard.)



*Clone this wiki locally*

The wiki is a separate clone URL than the project's repository. Make sure you're viewing your wiki and not your project. The clone URL will include `.wiki`.

In contrast to the "Clone this wiki locally" section, the "Clone in Desktop" button launches the [GitHub Desktop client \(https://desktop.github.com/\)](https://desktop.github.com/) and allows you to manage the repository and your modified files, commits, pushes, and pull through the GitHub Desktop client. If you're interested in using the GitHub Client of the command line, see this other activity: [Activity: Use the GitHub Desktop client \(https://idratherbewriting.com/learnapidoc/pubapis\\_github\\_desktop\\_client.html\)](https://idratherbewriting.com/learnapidoc/pubapis_github_desktop_client.html).

3. Open your terminal emulator:

- If you're a Windows user, open the **Git BASH** terminal emulator, which was installed when you [installed Git \(https://gitforwindows.org/\)](https://gitforwindows.org/).
  - If you're a Mac user, go to **Applications > Utilities > Terminal** (or launch [iTerm \(https://iterm2.com/\)](https://iterm2.com/), if you installed it instead).
4. In your terminal, either use the default directory or browse ( `cd` ) to a directory where you want to download your repository.
  5. Type the following, but replace the git URL with your own git URL that you copied earlier (it should be on your clipboard). The command should look something like this:

```
git clone https://github.com/tomjoht/weatherapi.wiki.git
```

When you clone a repo, Git will show something like the following:

```
Cloning into 'weatherapi.wiki'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 6
Unpacking objects: 100% (9/9), done.
```

The folder Git creates in the above example is called `weatherapi.wiki`.

Cloning the wiki gives you a copy of the content on your local machine. Git is distributed version control software, so everyone has his or her own copy. When you clone the repo, you create a copy on your local machine; the version in the cloud on GitHub is referred to as “origin.” Thus, you have two instances of the content.

More than just copying the files, though, when you clone a repo, you initialize Git in the folder where you clone the repo. Initializing Git means Git will create an invisible Git folder in that directory, and Git can start tracking your edits to the files, providing version control. With Git initialized, you can run `pull` commands to get updates from the online repository (origin) pulled down to your local copy. You can also `commit` your changes and then `push` your changes back up to origin.

6. Navigate to the directory where you cloned the repo (either using standard ways of browsing for files on your computer or via the terminal with `cd` ) to see the files you downloaded. For example, type `cd weatherapi.wiki` and then `ls` (Mac) or `dir` (Windows) to see the files.

You don't need to type the full directory name. Just start typing the first few letters and then press your **Tab** key to autocomplete the rest.

You might also want to browse to this folder via Finder (Mac) or Explorer (Windows). If you can view invisible files on your machine (for instructions on making hidden files visible, see one of the following: [Windows \(https://support.microsoft.com/en-us/help/14201/windows-show-hidden-files\)](https://support.microsoft.com/en-us/help/14201/windows-show-hidden-files) or [Mac \(https://ianlunn.co.uk/articles/quickly-showhide-hidden-files-mac-os-x-mavericks/\)](https://ianlunn.co.uk/articles/quickly-showhide-hidden-files-mac-os-x-mavericks/)), you will also see a `git` folder.

# Activity 7c: Push local changes to the remote

1. In a text editor, open the Markdown file you downloaded in the GitHub repository.
2. Make a small change to the content and save it. For example, type your name below the page title.
3. In your terminal, make sure you're in the directory where you downloaded the GitHub project.

To look at the directories under your current path, type `ls` (Mac) or `dir` (Windows). Then use `cd {directory name}` to drill into the folder, or `cd ../` to move up a level.

4. See what files have changed:

```
git status
```

Git responds:

```
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
modified:   Home.md
```

5. Add all the files to your staging area. The staging area contains all files that you want added to your next commit:

```
git add .
```

Git doesn't automatically track all files in the same folder where Git has been initialized. Git tracks modifications only for the files that have been "added" to Git. By typing `git add .` or `git add --all`, you're telling Git to start tracking modifications to all files in this directory. You could also type a specific file name here instead, such as `git add Home.md`, to just add a specific file (rather than all files changed) to Git's tracking.

After you run the `git add` command, Git adds the files into what's called the staging area. As a sports analogy, the staging area is like your on-deck circle. These files are ready to be committed when you run `git commit`.

6. See the changes set in your staging area:

```
git status
```

Git responds with a message indicating which files are on-deck to be committed.

```
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

modified:   Home.md
```

The staging area lists all the files that have been added to Git that you have modified in some way. It's a good practice to always type `git status` before committing files because you might realize that by typing `git add .`, you might have accidentally added some files you didn't intend to track (such as large binary files). If you want to remove this file from the staging area, you can type `git reset HEAD Home.md` to unstage it.

7. Commit the changes:

```
git commit -m "updated some content"
```

When you commit the changes, you're creating a snapshot of the files at a specific point in time for versioning.

The `git commit -m` command is a shortcut for committing and typing a commit message in the same step. It's much easier to commit updates this way.

If you type `git commit` only, you'll be prompted with another window to describe the change. On Windows, this new window will probably be a Notepad window. Describe the change on the top line, and then save and close the Windows file.

On a Mac, a new window doesn't open. Instead, the [Vim editor](https://en.wikipedia.org/wiki/Vim) (<https://en.wikipedia.org/wiki/Vim>) mode opens up within the terminal. ("vi" stands for visual and "m" for mode, but it's not a very visual editor.) I don't recommend using Vim. If you get stuck in this mode and need to escape, press your **Escape** key. Then type `q` to quit. (See [Vim commands](http://www.cs.rit.edu/~cslab/vi.html) (<http://www.cs.rit.edu/~cslab/vi.html>) here.) Normally, you want an external editor such as Sublime Text to open from your terminal. See [Associating text editors with Git](https://help.github.com/articles/associating-text-editors-with-git/) (<https://help.github.com/articles/associating-text-editors-with-git/>) for details.

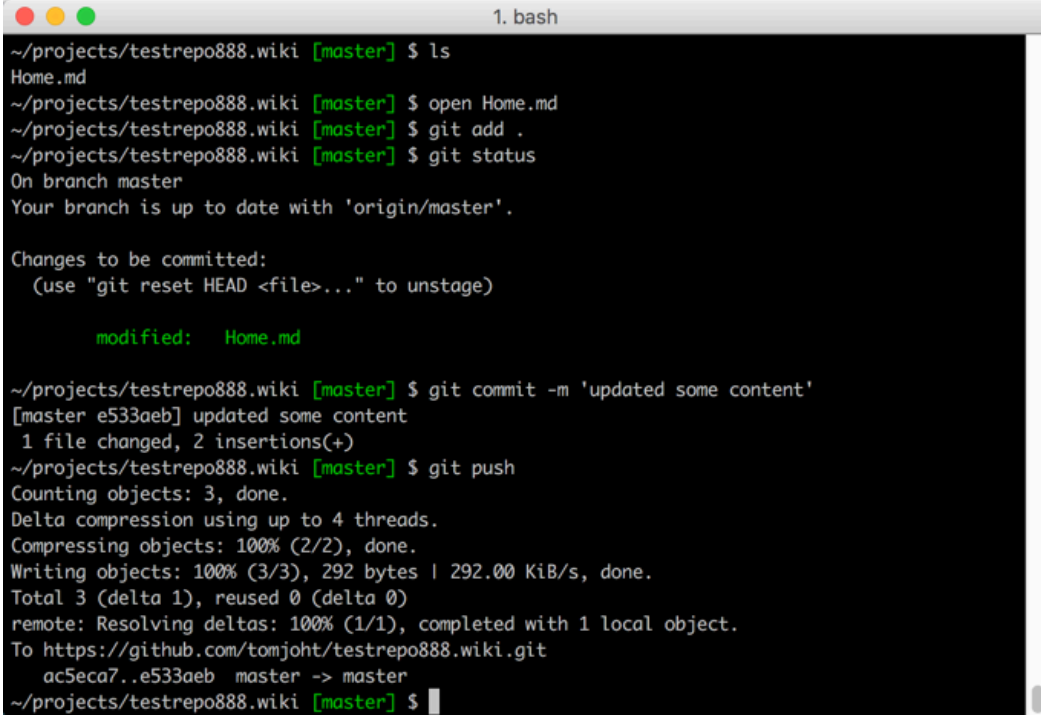
8. Push the changes to your repository:

```
git push
```

Unless you [set up automatic GitHub authentication](https://idratherbewriting.com/learnapidoc/pubapis_github_wikis.html#set-up-automatic-github-authentication) ([https://idratherbewriting.com/learnapidoc/pubapis\\_github\\_wikis.html#set-up-automatic-github-authentication](https://idratherbewriting.com/learnapidoc/pubapis_github_wikis.html#set-up-automatic-github-authentication)), you will be prompted for your GitHub username and password. (Note that your username is your GitHub login ID, such as "jdoe," not your friendly name, such as "John Doe.")

Note that when you type `git push` or `git pull` and don't specify the branch, GitHub uses the default branch from origin. The default branch on GitHub is called `master`. Thus the command actually passed is `git push origin master` (which means "push these changes to the remote origin[al] repository, in the `master` branch"). Some developers prefer to specify the repository and branch to ensure they are interacting with the right repositories and branches.

Your terminal window probably looks something like this:

A terminal window titled "1. bash" with a dark background and light green text. The window shows a series of git commands and their outputs. The user is in the directory ~/projects/testrepo888.wiki on the master branch. They list files, open Home.md, add it to the staging area, check status, commit it with the message "updated some content", and push it to the remote repository. The push is successful, showing object counts and compression details.

```
~/projects/testrepo888.wiki [master] $ ls
Home.md
~/projects/testrepo888.wiki [master] $ open Home.md
~/projects/testrepo888.wiki [master] $ git add .
~/projects/testrepo888.wiki [master] $ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   Home.md

~/projects/testrepo888.wiki [master] $ git commit -m 'updated some content'
[master e533aeb] updated some content
 1 file changed, 2 insertions(+)
~/projects/testrepo888.wiki [master] $ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 292 bytes | 292.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/tomjoht/testrepo888.wiki.git
   ac5eca7..e533aeb  master -> master
~/projects/testrepo888.wiki [master] $
```

Terminal window with git commands

9. Now verify that your changes took effect. Browse to your GitHub wiki repository and look to see the changes.

Although there are many options for authoring and publishing tools with developer docs, at their core is a probably a Git workflow. The Git workflow can be more powerful and complex than any authoring tool. Interacting with Git might also be key for interacting with engineering repos to make edits to documentation that appears inside code.

# Activity 9a: Look at API documentation jobs and requirements

In this activity, you'll get a sense of the skills needed for the jobs in your location, and then draw up a plan.

1. Go to [indeed.com](https://www.indeed.com/) (<https://www.indeed.com/>).
2. In the **Where** field, type your desired location.
3. Search for “API technical writer” or some combination of API + technical writer + developer documentation jobs.
4. Read the descriptions of 5 jobs.
5. Note a few of the salient requirements for these jobs.
6. Assess where you're currently at with the following:
  - Portfolio with writing samples that include developer documentation
  - Technical knowledge related to developer domain
  - Experience writing developer documentation
7. Make a plan for how you'll match up your portfolio, tech knowledge, and experience related to what these job descriptions are asking for. You might need to dedicate more time to an [open-source documentation project](https://idratherbewriting.com/learnapidoc/docapis_find_open_source_project.html) ([https://idratherbewriting.com/learnapidoc/docapis\\_find\\_open\\_source\\_project.html](https://idratherbewriting.com/learnapidoc/docapis_find_open_source_project.html)) to build up the needed skills.

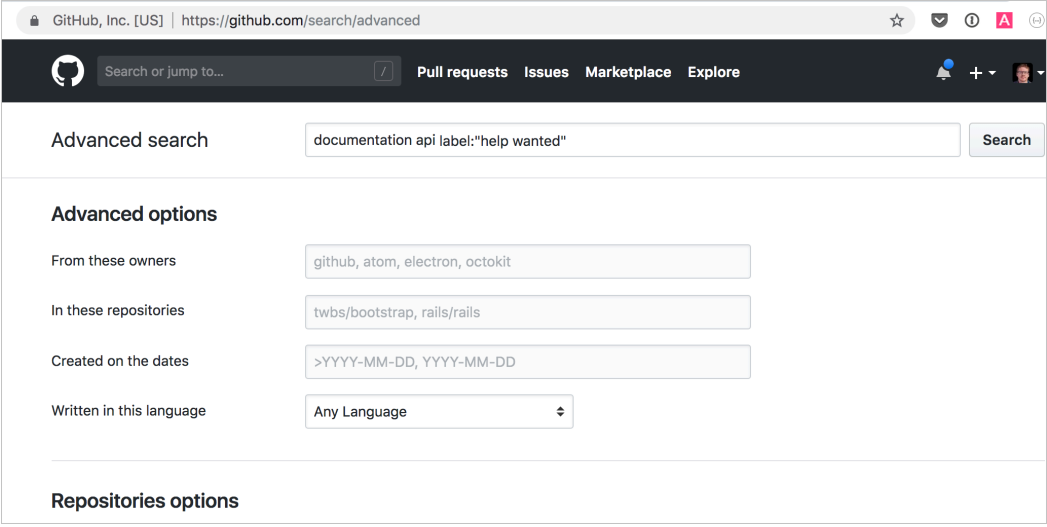
# Activity 9b: Find an open-source project with API doc needs

To find an open-source project with API doc needs:

1. Go to the [GitHub Advanced Search](https://github.com/search/advanced) (<https://github.com/search/advanced>).
2. Under the **Issues Options** section, in the **With the labels** row, type `help wanted`. This is a standard tag that teams use to attract volunteers to their project (but some teams that need help might not use it).

Scroll to the top and notice that `label: "help wanted"` automatically populates in the field.

3. In this Advanced Search box at the top, add some additional keywords (such as `documentation` and `rest api`) as well:



The screenshot shows the GitHub Advanced Search page. The search bar at the top contains the query: `documentation api label:"help wanted"`. Below the search bar, the 'Advanced options' section is expanded, showing several filters: 'From these owners' with the value 'github, atom, electron, octokit'; 'In these repositories' with the value 'twbs/bootstrap, rails/rails'; 'Created on the dates' with the value '>YYYY-MM-DD, YYYY-MM-DD'; and 'Written in this language' with the value 'Any Language'. The 'Repositories options' section is also visible at the bottom of the form.

4. Click **Search** and browse [the results](http://idratherbewriting.site/githubsearchchops) (<http://idratherbewriting.site/githubsearchchops>).

In the results, you might want to look for a *REST API* project (rather than a *native-library API* ([https://idratherbewriting.com/learnapidoc/nativelibraryapis\\_overview.html](https://idratherbewriting.com/learnapidoc/nativelibraryapis_overview.html)) such as a Java API). It might be hard to distinguish between the two, so include “REST” in your query. If you see doc requests related to Java, C++, JavaScript, or some other programming-specific framework, it’s probably not a REST API.

As you browse the results, are there any projects that look interesting or promising? If so, great. If not, adjust some of the keywords and keep looking.

5. If searching GitHub doesn’t yield any appropriate projects, try the following resources:
  - [Trending GitHub projects](https://github.com/trending) (<https://github.com/trending>)
  - [Crowdforge](https://crowdforge.io/) (<https://crowdforge.io/>)
  - [Up for Grabs](http://up-for-grabs.net/#/) (<http://up-for-grabs.net/#/>)
  - [Bus Factor](https://libraries.io/experiments/bus-factor) (<https://libraries.io/experiments/bus-factor>)
  - [Code Triage](https://www.codetriage.com/) (<https://www.codetriage.com/>)
  - [Changelog](https://changelog.com/) (<https://changelog.com/>)

- [24-hour Pull Requests \(https://24pullrequests.com\)](https://24pullrequests.com)
- [Programmableweb.com API directory \(https://www.programmableweb.com/category/all/apis\)](https://www.programmableweb.com/category/all/apis)

**Note:** You could spend a long time evaluating and deciding on open source projects. For this activity, it's okay if you focus on a project that looks only mildly interesting. You don't need to commit to it. You can always change it later.

6. After selecting a project, make notes on the following:

- Does the project involve a REST API?
- How does the project tag documentation-related issues? For example, does it use the “documentation” label?
- Identify the current state of the project's documentation. Are the docs robust, skimpy, nonexistent, extensive?
- How active is the project? (What is the frequency of commits?)
- How many contributors does the project have?

You don't have to actually reach out or interact with the team yet. You're just gathering information and analyzing documentation needs here.