ABOUT THE AUTHOR

Zell is a developer from Singapore. He teaches frontend development through his blog. He has also written courses [Learn JavaScript (↦ https://learnjavascript.today/)](https://learnjavascript.today/) and like [Automate Your … (↦ https://automateyourworkflow.com/)](https://automateyourworkflow.com/) [More about Zell Liew … (↦ /author/zellliew)](/author/zellliew)

[(↦ /author/zellliew)](/author/zellliew)

Understanding And Using REST APIs

ant to be able to read API documentations and use them effectively,

erything about REST APIs. Let's get started.

THERE'S A HIGH CHANCE YOU CAME ACROSS THE TERM "REST API" IF YOU'VE thought about getting data from another source on the internet, such as Twitter or Github. But what is a REST API? What can it do for you? How do you use it?

In this article, you'll learn everything you need to know about REST APIs to be able to read API documentations and use them effectively.

What Is A REST API

Let's say you're trying to find videos about Batman on Youtube. You open up Youtube, type "Batman" into a search field, hit enter, and you see a list of videos about Batman. A REST API works in a similar way. You search for something, and you get a list of results back from the service you're requesting from.

An **API** is an application programming interface. It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.

REST determines how the API looks like. It stands for "Representational State Transfer". It is a set of rules that developers follow when they create their API. One of these rules states that you should be able to get a piece of data (called a resource) when you link to a specific URL.

Each URL is called a **request** while the data sent back to you is called a **response**.

The Anatomy Of A Request

It's important to know that a request is made up of four things:

01 The endpoint

02 The method

03 The headers

04 The data (or body)

The endpoint (or route) is the url you request for. It follows this structure:

```
root-endpoint/?
```

The **root-endpoint** is the starting point of the API you're requesting from. The root-endpoint of Github's API *(↦ https://developer.github.com/v3/)* is `https://api.github.com` while the root-endpoint Twitter's API *(↦ https://dev.twitter.com/rest/reference)* is `https://api.twitter.com`.

The **path** determines the resource you're requesting for. Think of it like an automatic answering machine that asks you to press 1 for a service, press 2 for another service, 3 for yet another service and so on.

You can access paths just like you can link to parts of a website. For example, to get a list of all posts tagged under "JavaScript" on Smashing Magazine, you navigate to `https://www.smashingmagazine.com/tag/javascript/`. `https://www.smashingmagazine.com/` is the root-endpoint and `/tag/javascript` is the path.

To understand what paths are available to you, you need to look through the API documentation. For example, let's say you want to get a list of repositories by a certain user through Github's API. The docs *(↦ https://developer.github.com/v3/repos/#list-user-repositories)* tells you to use the the following path to do so:

```
/users/:username/repos
```

Any colons (`:`) on a path denotes a variable. You should replace these values with actual values of when you send your request. In this case, you should replace `:username` with the actual username of the user you're searching for. If I'm searching for my Github account, I'll replace `:username` with `zellwk`.

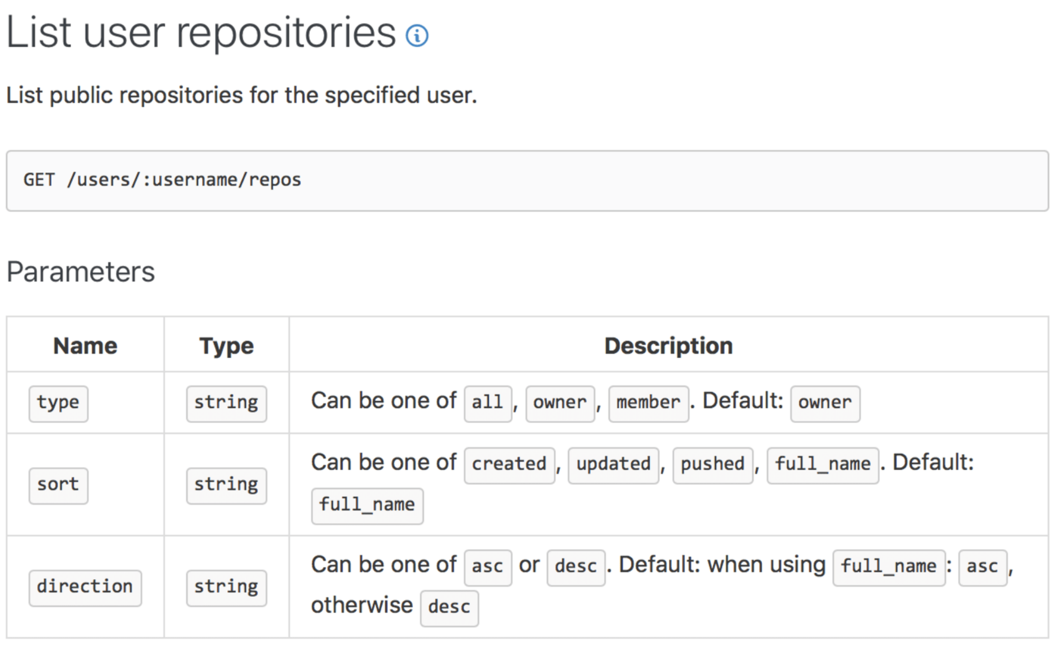
The endpoint to get a list of my repos on Github is this:

```
https://api.github.com/users/zellwk/repos
```

The final part of an endpoint is **query parameters**. Technically, query parameters are not part of the REST architecture, but you'll see lots of APIs use them. So, to help you completely understand how to read and use API's we're also going to talk about them. Query parameters give you the option to modify your request with key-value pairs. They always begin with a question mark (`?`). Each parameter pair is then separated with an ampersand (`&`), like this:

```
?query1=value1&query2=value2
```

When you try to get a list of a user's repositories on Github, you add three possible parameters to your request to modify the results given to you:



List user repositories

List public repositories for the specified user.

```
GET /users/:username/repos
```

Parameters

Name	Type	Description
`type`	`string`	Can be one of `all`, `owner`, `member`. Default: `owner`
`sort`	`string`	Can be one of `created`, `updated`, `pushed`, `full_name`. Default: `full_name`
`direction`	`string`	Can be one of `asc` or `desc`. Default: when using `full_name`: `asc`, otherwise `desc`

Github let's you add three parameters to your request

If you'd like to get a list the repositories that I pushed to recently, you can set `sort` to `push`.

```
https://api.github.com/users/zellwk/repos?sort=pushed
```

How do you know if this endpoint works? Well, it's time to give it a try!

Testing Endpoints With Curl

You can send a request with any programming language. JavaScript users can use methods like the Fetch API *(↦ https://css-tricks.com/using-fetch/)* and jQuery's Ajax method *(↦ http://api.jquery.com/jquery.ajax/)*; Ruby users can use Ruby's Net::HTTP class *(↦ http://ruby-doc.org/stdlib-2.4.2/libdoc/net/http/rdoc/index.html)*, Python users can use Python Requests *(↦ http://docs.python-requests.org/en/master/)*; and so on.

For this article, we'll use the command line utility called cURL *(↦ https://curl.haxx.se)*. We use cURL because API documentations are normally written with reference to cURL. If you understand how to use cURL, you'll have no problems understanding API documentations. Then, you can easily perform requests with your preferred language.

Before you continue, you'll want to make sure you have cURL installed on your machine. Open up your Terminal and type `curl -version`. This command checks the version of cURL you have installed on your system.

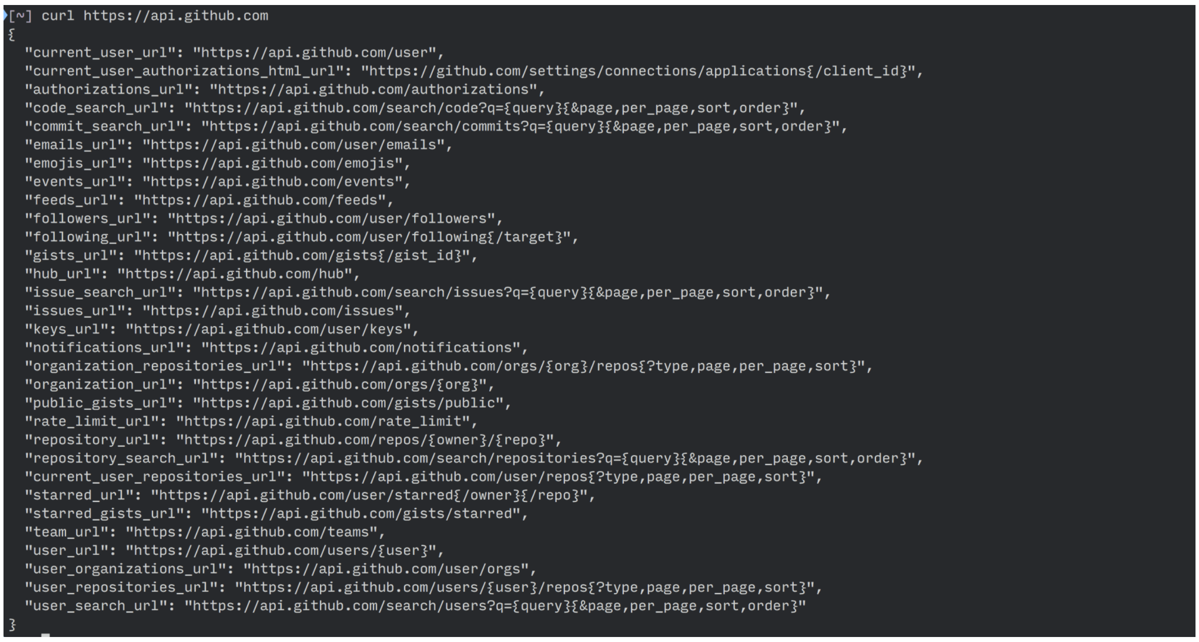
```
curl --version
```

If you don't have cURL installed, you'll get a "command not found" error. If you get this error, you will need to install curl *(↦ https://curl.haxx.se/download.html)* before moving on.

To use cURL, you type `curl`, followed by the endpoint you're requesting for. For example, to get Github's root endpoint, you type the following:

```
curl https://api.github.com
```

Once you hit enter, you should get a response from Github that looks like this:



```
[~] curl https://api.github.com
{
  "current_user_url": "https://api.github.com/user",
  "current_user_authorizations_html_url": "https://github.com/settings/connections/applications{/client_id}",
  "authorizations_url": "https://api.github.com/authorizations",
  "code_search_url": "https://api.github.com/search/code?q={query}{&page,per_page,sort,order}",
  "commit_search_url": "https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}",
  "emails_url": "https://api.github.com/user/emails",
  "emojis_url": "https://api.github.com/emojis",
  "events_url": "https://api.github.com/events",
  "feeds_url": "https://api.github.com/feeds",
  "followers_url": "https://api.github.com/user/followers",
  "following_url": "https://api.github.com/user/following{/target}",
  "gists_url": "https://api.github.com/gists{/gist_id}",
  "hub_url": "https://api.github.com/hub",
  "issue_search_url": "https://api.github.com/search/issues?q={query}{&page,per_page,sort,order}",
  "issues_url": "https://api.github.com/issues",
  "keys_url": "https://api.github.com/user/keys",
  "notifications_url": "https://api.github.com/notifications",
  "organization_repositories_url": "https://api.github.com/orgs/{org}/repos{?type,page,per_page,sort}",
  "organization_url": "https://api.github.com/orgs/{org}",
  "public_gists_url": "https://api.github.com/gists/public",
  "rate_limit_url": "https://api.github.com/rate_limit",
  "repository_url": "https://api.github.com/repos/{owner}/{repo}",
  "repository_search_url": "https://api.github.com/search/repositories?q={query}{&page,per_page,sort,order}",
  "current_user_repositories_url": "https://api.github.com/user/repos{?type,page,per_page,sort}",
  "starred_url": "https://api.github.com/user/starred{/owner}{/repo}",
  "starred_gists_url": "https://api.github.com/gists/starred",
  "team_url": "https://api.github.com/teams",
  "user_url": "https://api.github.com/users/{user}",
  "user_organizations_url": "https://api.github.com/user/orgs",
  "user_repositories_url": "https://api.github.com/users/{user}/repos{?type,page,per_page,sort}",
  "user_search_url": "https://api.github.com/search/users?q={query}{&page,per_page,sort,order}"
}
```

Response from Github's root-endpoint

To get a list of a user's repositories, you modify the endpoint to the correct path, like what we discussed above. To get a list of my repositories, you can use this command:

```
curl https://api.github.com/users/zellwk/repos
```

If you wish to include query parameters with cURL, make sure you prepend a backslash (\ ) before the `?` and `=` characters. This is because `?` and `=` are special characters in the command line. You need to use `\` before them for the command line to interpret them as normal characters:

```
curl https://api.github.com/users/zellwk/repos\?sort\=pushed
```

Try using either commands and perform a request! You'll get a similar response to what you've seen with Github's root-endpont (but with a lot more data).

JSON

JSON (JavaScript Object Notation) a common format for sending and requesting data through a REST API. The response that Github sends back to you is also formatted as JSON.

A JSON object looks like a JavaScript Object. In JSON, each property and value must be wrapped with double quotation marks, like this:

```
{
  "property1": "value1",
  "property2": "value2",
  "property3": "value3"
}
```

Back To The Anatomy Of A Request

You've learned that a request consists of four parts.

01 The endpoint

02 The method

03 The headers

04 The data (or body)

Let's go through the rest of what makes up a request.

The Method

The method is the type of request you send to the server. You can choose from these five types below:

- `GET`
- `POST`

- `PUT`
- `PATCH`
- `DELETE`

These methods provide meaning for the request you're making. They are used to perform four possible actions: `Create`, `Read`, `Update` and `Delete` (CRUD).

Method Name	Request Meaning
\`GET\`	This request is used to get a resource from a server. If you perform a \`GET\` request, the server looks for the data you requested and sends it back to you. In other words, a \`GET\` request performs a \`READ\` operation. This is the default request method.
\`POST\`	This request is used to create a new resource on a server. If you perform a \`POST\` request, the server creates a new entry in the database and tells you whether the creation is successful. In other words, a \`POST\` request performs an \`CREATE\` operation.
\`PUT\` and \`PATCH\`	These two requests are used to update a resource on a server. If you perform a \`PUT\` or \`PATCH\` request, the server updates an entry in the database and tells you whether the update is successful. In other words, a \`PUT\` or \`PATCH\` request performs an \`UPDATE\` operation.
\`DELETE\`	This request is used to delete a resource from a server. If you perform a \`DELETE\` request, the server deletes an entry in the database and tells you whether the deletion is successful. In other words, a \`DELETE\` request performs a \`DELETE\` operation.

The API lets you know what request method to use each request. For example, to get a list of a user's repositories, you need a `GET` request:

```
GET /users/:username/repos
```

A GET request is required to get a list of repositories from a user

A GET request is required to get a list of repositories from a user. To [create a new Github repository (↦ https://developer.github.com/v3/repos/#create)](https://developer.github.com/v3/repos/#create), you need a `POST` request:

```
POST /user/repos
```

A POST request is required to create a new repository

You can set the request method in cURL by writing `-X` or `--request`, followed by the request method. This command below tries to create a repository via cURL:

```
curl -X POST https://api.github.com/user/repos
```

Try running this request. You'll get a response that tells you that authentication is required. (More on authentication later).

```
{
  "message": "Requires authentication",
  "documentation_url": "https://developer.github.com/v3"
}
```

The Headers

Headers are used to provide information to both the client and server. It can be used for many purposes, such as authentication and providing information about the body content. You can find a list of valid headers on MDN's HTTP Headers Reference *(↦ https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers)*.

HTTP Headers are property-value pairs that are separated by a colon. The example below shows a header that tells the server to expect JSON content.

```
"Content-Type: application/json". Missing the opening ".
```

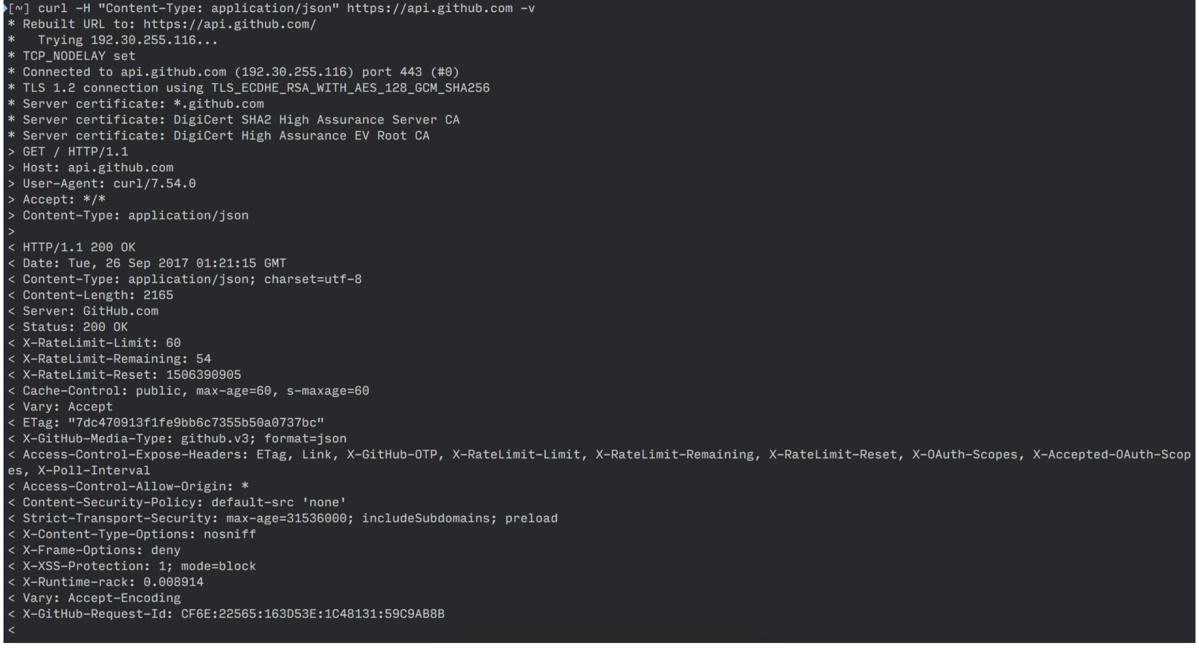
You can send HTTP headers with curl through the `-H` or `--header` option. To send the above header to Github's API, you use this command:

```
curl -H "Content-Type: application/json" https://api.github.com
```

(Note: the Content-Type header is not a requirement for Github's API to work. This is only an example to illustrate how to use a header with cURL).

To view headers you've sent, you can use the `-v` or `--verbose` option as you send the request, like this:

```
curl -H "Content-Type: application/json" https://api.github.com -v
```



```
[~] curl -H "Content-Type: application/json" https://api.github.com -v
* Rebuilt URL to: https://api.github.com/
*   Trying 192.30.255.116...
* TCP_NODELAY set
* Connected to api.github.com (192.30.255.116) port 443 (#0)
* TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate: *.github.com
* Server certificate: DigiCert SHA2 High Assurance Server CA
* Server certificate: DigiCert High Assurance EV Root CA
> GET / HTTP/1.1
> Host: api.github.com
> User-Agent: curl/7.54.0
> Accept: */*
> Content-Type: application/json
>
< HTTP/1.1 200 OK
< Date: Tue, 26 Sep 2017 01:21:15 GMT
< Content-Type: application/json; charset=utf-8
< Content-Length: 2165
< Server: GitHub.com
< Status: 200 OK
< X-RateLimit-Limit: 60
< X-RateLimit-Remaining: 54
< X-RateLimit-Reset: 1506390905
< Cache-Control: public, max-age=60, s-maxage=60
< Vary: Accept
< ETag: "7dc470913f1fe9bb6c7355b50a0737bc"
< X-GitHub-Media-Type: github.v3; format=json
< Access-Control-Expose-Headers: ETag, Link, X-GitHub-OTP, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset, X-OAuth-Scopes, X-Accepted-OAuth-Scop
es, X-Poll-Interval
< Access-Control-Allow-Origin: *
< Content-Security-Policy: default-src 'none'
< Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
< X-Content-Type-Options: nosniff
< X-Frame-Options: deny
< X-XSS-Protection: 1; mode=block
< X-Runtime-rack: 0.008914
< Vary: Accept-Encoding
< X-GitHub-Request-Id: CF6E:22565:163D53E:1C48131:59C9AB8B
<
```

cURL tells you additional information, which includes headers when you use the verbose option

Here, `*` refers to additional information provided by cURL. `>` refers to request headers, and `<` refers to the response headers.

The Data (Or "Body")

The data (sometimes called "body" or "message") contains information you want to be sent to the server. This option is only used with `POST`, `PUT`, `PATCH` or `DELETE` requests.

To send data through cURL, you can use the `-d` or `--data` option:

```
curl -X POST <URL> -d property1=value1
```

To send multiple data fields, you can create multiple `-d` options:

```
curl -X POST <URL> -d property1=value1 -d property2=value2
```

If it makes sense, you can break your request into multiple lines `\` to make it easier to read:

```
curl -X POST <URL> \
  -d property1=value1 \
  -d property2=value2
```

If you know how to spin up a server, you can make an API and test your own data. If you don't know, but feel courageous enough to try, you can follow this article *(↦ https://zellwk.com/blog/crud-express-mongodb/)* to learn to create a server with Node, Express, and MongoDB

If you don't want to spin up your server, you can go to [Requestbin.com (↦ https://requestbin.com/)](https://requestbin.com/) *(it's free!)* and click on the "create endpoint". You'll be given a URL that you can use to test requests, like `https://requestb.in/1ix963n1` shown in the picture below.

Bin URL

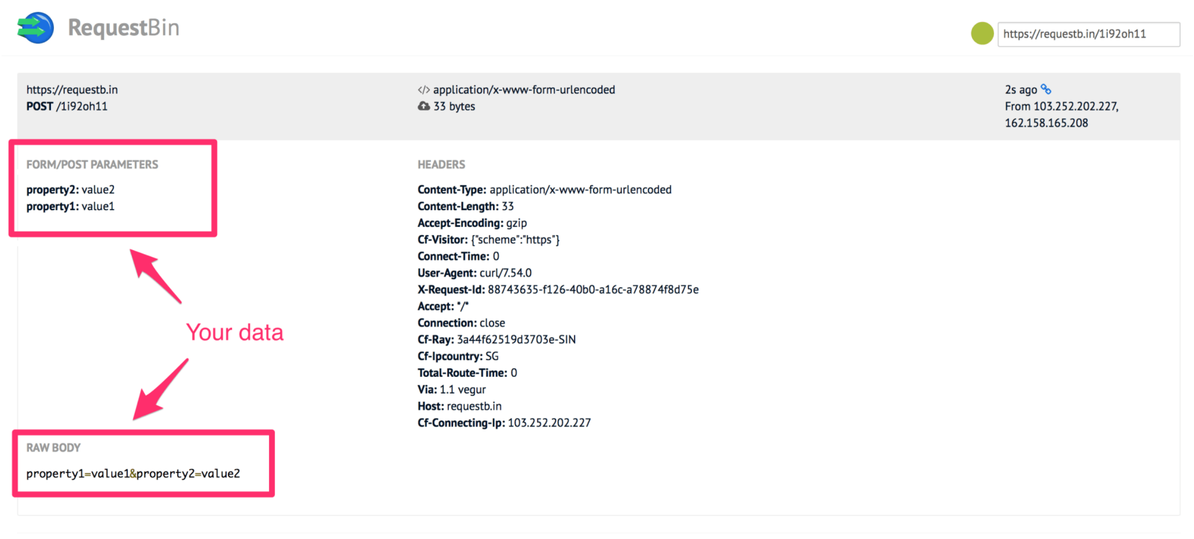
https://requestb.in/1ix963n1

Request bin gives you a unique URL you can use for 48 hours

Make sure you create your own request bin if you want to test your request. Request bins only remains open for 48 hours after its creation. By the time you read this article, the bin I created above will be long gone.

Now, try sending some data to your request bin, then refresh your bin's webpage. You'll see some data, like this:

```
curl -X POST https://requestb.in/1ix963n1 \
  -d property1=value1 \
  -d property2=value2
```



https://requestb.in
POST /1i92oh11

</> application/x-www-form-urlencoded
33 bytes

2s ago
From 103.252.202.227, 162.158.165.208

FORM/POST PARAMETERS

property2: value2
property1: value1

HEADERS

Content-Type: application/x-www-form-urlencoded
Content-Length: 33
Accept-Encoding: gzip
Cf-Visitor: {"scheme":"https"}
Connect-Time: 0
User-Agent: curl/7.54.0
X-Request-Id: 88743635-f126-40b0-a16c-a78874f8d75e
Accept: */*
Connection: close
Cf-Ray: 3a44f62519d3703e-SIN
Cf-Ipcountry: SG
Total-Route-Time: 0
Via: 1.1 vegur
Host: requestb.in
Cf-Connecting-Ip: 103.252.202.227

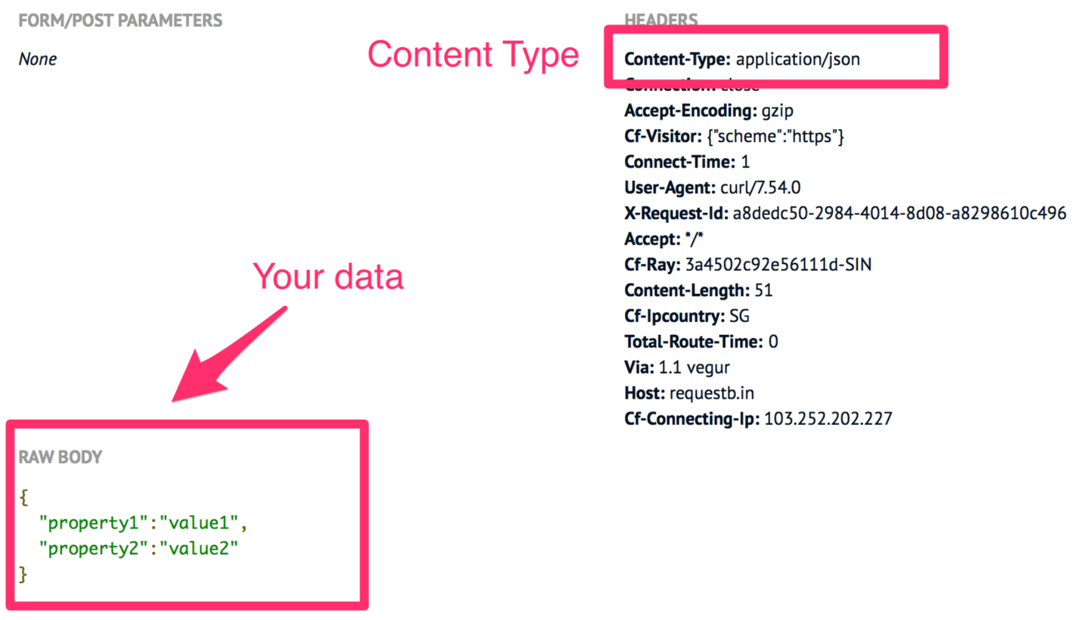
RAW BODY

```
property1=value1&property2=value2
```

Requests you sent to your bin will appear like this

By default, cURL sends data as if they're sent through "form fields" on a page. If you wish to send JSON data, you'll need to set the `Content-Type` to `application/json`, and you'll need to format your data as a JSON object, like this:

```
curl -X POST https://requestb.in/1ix963n1 \
  -H "Content-Type: application/json" \
  -d '{
  "property1":"value1",
  "property2":"value2"
}'
```



```
FORM/POST PARAMETERS
None

HEADERS
Content-Type: application/json
Connection: close
Accept-Encoding: gzip
Cf-Visitor: {"scheme":"https"}
Connect-Time: 1
User-Agent: curl/7.54.0
X-Request-Id: a8dedc50-2984-4014-8d08-a8298610c496
Accept: */*
Cf-Ray: 3a4502c92e56111d-SIN
Content-Length: 51
Cf-Ipcountry: SG
Total-Route-Time: 0
Via: 1.1 vegur
Host: requestb.in
Cf-Connecting-Ip: 103.252.202.227

RAW BODY
{
  "property1":"value1",
  "property2":"value2"
}
```

Sending data as JSON

And that is (almost!) everything you need to know about the structure of a request.

Now, remember when you tried to send a `POST` request through Github's API, you got a message that says "Requires authentication"? Well, that's because you're not authorized to perform the `POST` request!

Authentication

You wouldn't allow anyone to access your bank account without your permission, would you? On the same line of thought, developers put measures in place to ensure you perform actions only when you're authorized to do. This prevents others from impersonating you.

Since `POST`, `PUT`, `PATCH` and `DELETE` requests alter the database, developers almost always put them behind an authentication wall. In some cases, a `GET` request also requires authentication (like when you access your bank account to check your current balance, for example).

On the web, there are two main ways to authenticate yourself:

01 With a username and password (also called basic authentication)

02 With a secret token

The secret token method includes oAuth *(↦ https://oauth.net)*, which lets you to authenticate yourself with social media networks like Github, Google, Twitter, Facebook, etc.

For this article, you'll only learn to use basic authentication with a username and a password. If you're interested in authenticating with oAuth, I suggest reading "What You Need To Know About OAuth2 And Logging In With Facebook *(↦ https://www.smashingmagazine.com/2017/05/oauth2-logging-in-facebook/)*" by Zack Grossbart *(↦ https://www.smashingmagazine.com/author/zack-grossbart/)*.

To perform a basic authentication with cURL, you can use the `-u` option, followed by your username and password, like this:

```
curl -x POST -u "username:password" https://api.github.com/user/repos
```

Try authenticating yourself with your username and password in the above request. Once you succeed in authentication, you'll see the response change from "Requires authentication" to "Problems parsing JSON."

This is because you've yet to provide any data (which is required by all `POST`, `PUT`, `PATCH` and `DELETE` requests) to the server.

With the knowledge you've learned so far, you should be able to edit the code above to create a Github repository via your cURL. I'd leave you to try it yourself!

Now, let's talk about HTTP Status codes and error messages.

HTTP Status Codes And Error Messages

Some of the messages you've received earlier, like "Requires authentication" and "Problems parsing JSON" are error messages. They only appear when something is wrong with your request. HTTP status codes let you tell the status of the response quickly. The range from 100+ to 500+. In general, the numbers follow the following rules:

01 **200+** means the request has **succeeded**.

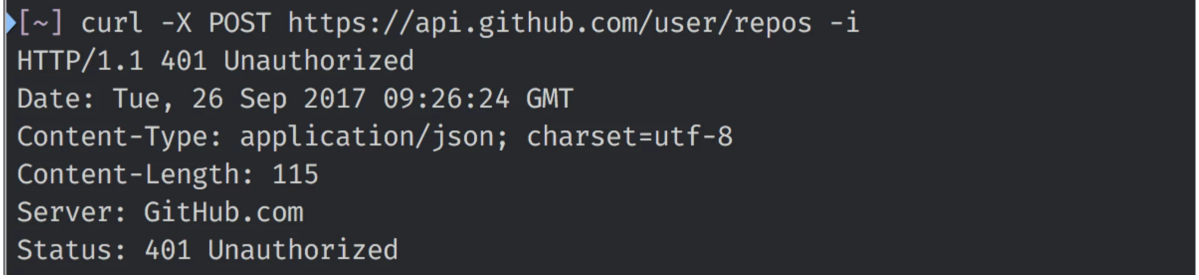
02 **300+** means the request is **redirected** to another URL

03 **400+** means an **error that originates from the client** has occurred

04 **500+** means an **error that originates from the server** has occurred

You can debug the status of a response with the verbose option (`-v` or `--verbose`) or the head option (`-I` or `--head`).

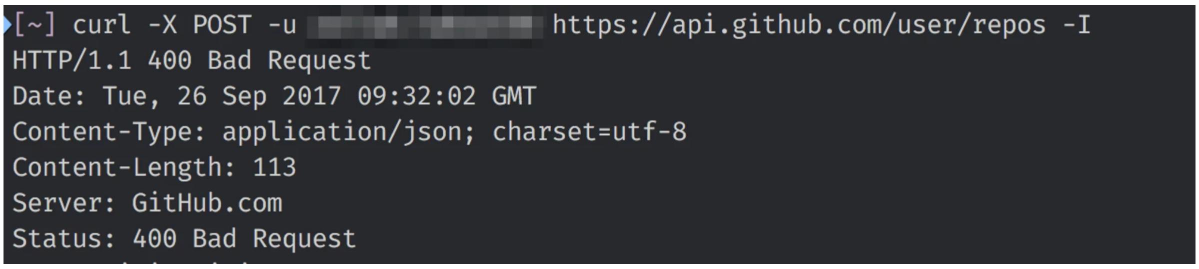
For example, if you tried adding `-I` to a `POST` request without providing your username and password, you'll get a 401 status code (Unauthorized):



```
[~] curl -X POST https://api.github.com/user/repos -i
HTTP/1.1 401 Unauthorized
Date: Tue, 26 Sep 2017 09:26:24 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 115
Server: GitHub.com
Status: 401 Unauthorized
```

Example of an unauthorized request

If your request is invalid because your data is wrong or missing, you usually get a 400 status code (Bad Request).



```
[~] curl -X POST -u [redacted] https://api.github.com/user/repos -I
HTTP/1.1 400 Bad Request
Date: Tue, 26 Sep 2017 09:32:02 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 113
Server: GitHub.com
Status: 400 Bad Request
```

Example of a bad request

To get more information about specific HTTP status codes, you may want to consult MDN's [HTTP Status Reference](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) *(↦ https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)*.

API Versions

Developers update their APIs from time to time. Sometimes, the API can change so much that the developer decides to upgrade their API to another version. If this happens, and your application breaks, it's usually because you've written code for an older API, but your request points to the newer API.

You can request for a specific API version in two ways. Which way you choose depends on how the API is written.

These two ways are:

01 Directly in the endpoint

02 In a request header

Twitter, for example, uses the first method. At the time of writing, Twitter's API is at version 1.1, which is evident through its endpoint:

```
https://api.twitter.com/1.1/account/settings.json
```

Github, on the other hand, uses the second method. At the time of writing, Github's API is at version 3, and you can specify the version with an `Accept` header:

```
curl https://api.github.com -H Accept:application/vnd.github.v3+json
```

Wrapping Up

In this article, you learned what a REST API is and how to use cURL to perform a request with `GET`, `POST`, `PUT`, `PATCH` and `DELETE` methods. In addition, you also learned how to authenticate your requests with the `-u` option, and what HTTP statuses mean.

I hope this article has helped you learn enough about REST APIs, and you can use them fluently as you create your applications. Feel free to pop over to my blog *(↦ https://zellwk.com/newsletter/smashing-magazine)* or leave your comments below if you have any questions.

(rb, ra, il)