Best practices for using the REST API

In this article

Follow these best practices when using GitHub's API.

Avoid polling

You should subscribe to webhook events instead of polling the API for data. This will help your integration stay within the API rate limit. For more information, see "Webhooks documentation."

Follow any redirects that the API sends you

GitHub is explicit in telling you when a resource has moved by providing a redirect status code. You should follow these redirections. Every redirect response sets the `Location` header with the new URI to go to. If you receive a redirect, it's best to update your code to follow the new URI, in case you're requesting a deprecated path that we might remove.

We've provided a list of HTTP status codes to watch out for when designing your app to follow redirects.

Don't manually parse URLs

Often, API responses contain data in the form of URLs. For example, when requesting a repository, we'll send a key called `clone_url` with a URL you can use to clone the repository.

For the stability of your app, you shouldn't try to parse this data or try to guess and construct the format of future URLs. Your app is liable to break if we decide to change the URL.

For example, when working with paginated results, it's often tempting to construct URLs that append `?page=<number>` to the end. Avoid that temptation. For more information about dependably following paginated results, see "Using pagination in the REST API."

Dealing with rate limits

The GitHub API rate limit ensures that the API is fast and available for everyone.

If you hit a rate limit, you should stop making requests until after the time specified by

the `x-ratelimit-reset` header. Failure to do so may result in the banning of your integration. For more information, see "Resources in the REST API."

Dealing with secondary rate limits

GitHub may also use secondary rate limits to ensure API availability. For more information, see "Resources in the REST API."

To avoid hitting this limit, you should ensure your application follows the guidelines below.

- Make authenticated requests, or use your application's client ID and secret. Unauthenticated requests are subject to more aggressive secondary rate limiting.
- Make requests for a single user or client ID serially. Do not make requests for a single user or client ID concurrently.
- If you're making a large number of `POST`, `PATCH`, `PUT`, or `DELETE` requests for a single user or client ID, wait at least one second between each request.
- When you have been limited, wait before retrying your request.
    - If the `Retry-After` response header is present, retry your request after the time specified in the header. The value of the `Retry-After` header will always be an integer, representing the number of seconds you should wait before making requests again. For example, `Retry-After: 30` means you should wait 30 seconds before sending more requests.
    - If the `x-ratelimit-remaining` header is `0`, retry your request after the time specified by the `x-ratelimit-reset` header. The `x-ratelimit-reset` header will always be an integer representing the time at which the current rate limit window resets in UTC epoch seconds.
    - Otherwise, wait for an exponentially increasing amount of time between retries, and throw an error after a specific number of retries.

GitHub reserves the right to change these guidelines as needed to ensure availability.

Dealing with API errors

Although your code would never introduce a bug, you may find that you've encountered successive errors when trying to access the API.

Rather than ignore repeated `4xx` and `5xx` status codes, you should ensure that you're correctly interacting with the API. For example, if an endpoint requests a string and you're passing it a numeric value, you're going to receive a `5xx` validation error, and your call won't succeed. Similarly, attempting to access an unauthorized or nonexistent endpoint will result in a `4xx` error.

Intentionally ignoring repeated validation errors may result in the suspension of your app for abuse.

Further reading

- "Best practices for using webhooks"
- "Best practices for creating a GitHub App"