

Getting Started with GitHub's GraphQL API

JUNE 09, 2018 BY ROBIN WIERUCH - [EDIT THIS POST](#)

[Follow on Twitter](#)

17k

[Follow on Facebook](#)



f
t
in

Interested in reading this tutorial as one of many chapters in my GraphQL book? Checkout the entire [The Road to GraphQL](#) book that teaches you to become a fullstack developer with JavaScript.

This tutorial is part 1 of 5 in this series.

[Part 2: GraphQL Tutorial for Beginners](#)

[Part 3: A complete React with GraphQL Tutorial](#)

[Part 4: Apollo Client Tutorial for Beginners](#)

[Part 5: React with Apollo and GraphQL Tutorial](#)

Step-by-step is often the easiest way to learn something new, so it's fortunate that learning GraphQL in JavaScript teaches both the client and the server-side of an application. Seeing

both sides of the web transactions is useful, but the catch is you have to learn two

environments. The step-by-step mentality can be difficult to apply here, so I encourage beginners to start with a client-side application by consuming a third-party GraphQL API before the server side, which uses a GraphQL server.

GitHub is one of the first major tech brands to adopt GraphQL. They even managed to release a public GraphQL API ([official documentation](#)), which is quite popular among developers, because most are familiar enough with GitHub from using it for their own projects.

In this chapter, I hope to cover everything you need to get started with GitHub's GraphQL API, and learning to use GraphQL in JavaScript from a client-side perspective by consuming their API. You should gain understanding about GitHub's terminology, and how to consume account data using its GraphQL API. There are a few applications we will implement with this GraphQL API from a client perspective, so it makes sense to invest time into this section to avoid any fundamental mistakes. Afterward, we will transition to the server-side by implementing our own GraphQL server.



FEEDING THE API WITH DATA ON GITHUB

If you don't have an account on GitHub yet, and don't know much about its ecosystem, follow [this official GitHub Learning Lab](#). If you want to dive deeper into Git and its essential commands, check out [this guide](#) about it. This might come in handy if you decide to share projects with others on GitHub in the future. It is a good way to showcase a development portfolio to potential clients or hiring companies.

For our interactions with GitHub's GraphQL API, you will use your own account with information to read/write from/to this data. Before that, complete your GitHub profile by providing additional information so you can recognize it later when it is read by the API.

Exercises:

- Create a GitHub account if you don't have one
- Provide additional information for your GitHub profile

GitHub Repositories

You can also create repositories on GitHub. In the words of their official glossary: "A

repository is the most basic element of GitHub. They're easiest to imagine as a project's folder. A repository contains all of the project files (including documentation), and stores each file's revision history. Repositories can have multiple collaborators and can be either public or private." GitHub's glossary will explain the key terms--repository, issue, clone, fork, push--which are necessary to follow along with the upcoming chapters to learn about GraphQL. Basically a repository is the place for application source code that can be shared with others. I encourage you to put a few of your projects into GitHub repositories, so you can access them all later with what you've learned about their GraphQL API.

If you don't have any projects to upload, you can always 'fork' repositories from other GitHub users and work on copies of them. A fork is basically a clone of a repository where you can add changes without altering the original. There are many public repositories on GitHub that can be cloned to your local machine or forked to your list so you can get an understanding of the mechanics through experimentation. For example, if you visit [my GitHub profile](#), you can see all my public repositories, though not all of these are mine, because some of them are just forks of others. Feel free to fork these repositories if you'd like to use them as practice, and if you'd like them to be accessible via GitHub's GraphQL API from your own account.



Exercises:



- Create/Fork a couple of GitHub repositories, and verify that they show in your account as copies. Copies are indicated by the username that precedes the repository name in all its titles; for example, a repo called *OriginalAuthor/TestRepo* would be renamed to *YourUserName/TestRepo* once you've forked it.



Paginated Data

GitHub's GraphQL API allows you to request multiple repositories at once, which is useful for pagination. Pagination is a programming mechanic invented to work with large lists of items. For example, imagine you have more than a hundred repositories in your GitHub account, but your UI only shows ten of them. Transferring the whole list across the wire for each request is impractical and inefficient, because only a subset is needed at a time, which pagination allows.

Using pagination with GitHub's GraphQL API lets you adjust the numbers to your own needs, so make sure to adjust the numbers (e.g. limit, offset) to your personal requirements (e.g. available repositories of your GitHub account or available repositories of a GitHub organization). You at least want to have enough repositories in your collection to see the pagination feature in action, so I recommend more than twenty (20), assuming each page will display ten (10), or use five(5) repositories when displaying two (2).

Issues and Pull Requests

Once you dive deeper into GitHub's GraphQL API and you start to request nested relationships (e.g. issues of repositories, pull requests of repositories), make sure that the repositories have a few issues or pull requests. This is so you'll see something when we implement the feature to show all the issues in a repository. It might be better to request repositories from a GitHub organization where there will be plenty of issues and pull requests.

Exercises:

- Read more about the different terms in [GitHub's glossary](#). Consider these questions:
 - What is a GitHub organization and GitHub user?
 - What are repositories, issues and pull requests?
 - What are GitHub repository stars and GitHub repository watchers?
- Create or fork enough repositories to use the pagination feature.
- Create pull requests and issues in a few of your GitHub repositories.



READ/WRITE DATA WITH GITHUB'S PERSONAL ACCESS TOKEN

To use GitHub's GraphQL API, you need to generate a personal access token on their website. The access token authorizes users to interact with data, to read and write it under your username. [Follow their step by step instructions](#) to obtain the personal access token, and be sure to check the necessary scopes (permissions) for it, as you will need them to implement a well-rounded GitHub client later.

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> admin:org	Full control of orgs and teams
<input checked="" type="checkbox"/> write:org	Read and write org and team membership
<input checked="" type="checkbox"/> read:org	Read org and team membership
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys

<input type="checkbox"/>	admin:repo_hook	Full control of repository hooks
<input type="checkbox"/>	write:repo_hook	Write repository hooks
<input type="checkbox"/>	read:repo_hook	Read repository hooks
<input type="checkbox"/>	admin:org_hook	Full control of organization hooks
<input type="checkbox"/>	gist	Create gists
<input checked="" type="checkbox"/>	notifications	Access notifications
<input checked="" type="checkbox"/>	user	Update all user data
<input checked="" type="checkbox"/>	read:user	Read all user profile data
<input checked="" type="checkbox"/>	user:email	Access user email addresses (read-only)
<input checked="" type="checkbox"/>	user:follow	Follow and unfollow users
<input type="checkbox"/>	delete_repo	Delete repositories
<input type="checkbox"/>	write:discussion	Read and write team discussions
<input type="checkbox"/>	read:discussion	Read team discussions
<input type="checkbox"/>	admin:gpg_key	Full control of user gpg keys (Developer Preview)
<input type="checkbox"/>	write:gpg_key	Write user gpg keys
<input type="checkbox"/>	read:gpg_key	Read user gpg keys



Later, the personal access token can be used to interact with GitHub's GraphQL API. Be careful not to share these authorizations with any third parties.



INTERACTING WITH GITHUB'S GRAPHQL API

There are two common ways to interact with the GitHub GraphQL API without writing any source code for it. First, you can use GitHub's [GraphQL Explorer](#). You only need to sign up with your GitHub account to perform a query or mutation to their GraphQL API, and its a good way to simplify your first experience. Second, you can use a generic client in the form of an application. GraphiQL is a client that makes GraphQL requests as an integration or as a standalone application. The former can be accomplished [by setting up GraphiQL directly in your application](#); the latter may be more convenient for you by [using GraphiQL as a standalone application](#). It's a lightweight shell around GraphiQL that can be downloaded and installed manually or by the command line.

GitHub's GraphQL Explorer knows about your credentials, since you need to sign up using it, but the GraphiQL application needs to know about the personal access token you created. You can add it in your HTTP header for every request in the headers configuration.

The screenshot shows the GraphiQL interface with a toolbar at the top containing 'GraphiQL', 'Prettify', 'History', and a 'Docs' link. Below the toolbar, there's a section titled 'Edit HTTP Headers' with a table. The table has two columns: 'Header name' and 'Header value'. A single row is present with 'Authorization' in the 'Header name' column and 'bearer eb4d1' in the 'Header value' column. To the right of the table are 'Edit' and 'Delete' buttons. At the bottom right of the 'Edit HTTP Headers' section is a '+ Add Header' button.

Header name	Header value
Authorization	bearer eb4d1

+ Add Header

Edit Delete

In the next step, we add a new header with a name and value to your GraphiQL configuration. To communicate with GitHub's GraphQL API, fill in the header name with "Authorization" and the header value with "bearer [your personal access token]". Save this new header for your GraphiQL application. Finally, you are ready to make requests to GitHub's GraphQL API with your GraphiQL application.

Edit HTTP Headers

+ Add Header

Header name	Header value	
Authorization	bearer eb4d1	Edit Delete

If you use your own GraphiQL application, you'll need to provide the GraphQL endpoint for GitHub's GraphQL API: <https://api.github.com/graphql>. For GitHub's GraphQL API, use the [POST HTTP method](#) for queries and mutations, and to transfer data as a payload to your GraphQL endpoint.



This section provided you with two ways to interact with GitHub's GraphQL API. Where GitHub's GraphQL Explorer can only be used for GitHub's API, GraphiQL integrated into an application or standalone can be used for any GraphQL API. The difference is that it requires a bit more setup. The GitHub GraphQL Explorer is really nothing more than a hosted standalone GraphiQL application tailored to use GitHub's GraphQL API.



After you've set up GitHub to use their GraphQL API to learn about GraphQL, you should be ready to implement your first GraphQL client interactions. Follow along and create your first GraphQL client-side application with the tools you have just set up but also with React.

This tutorial is part 1 of 5 in this series.

[Part 2: GraphQL Tutorial for Beginners](#)

[Part 3: A complete React with GraphQL Tutorial](#)

[Part 4: Apollo Client Tutorial for Beginners](#)

[Part 5: React with Apollo and GraphQL Tutorial](#)

KEEP READING ABOUT STARTER >

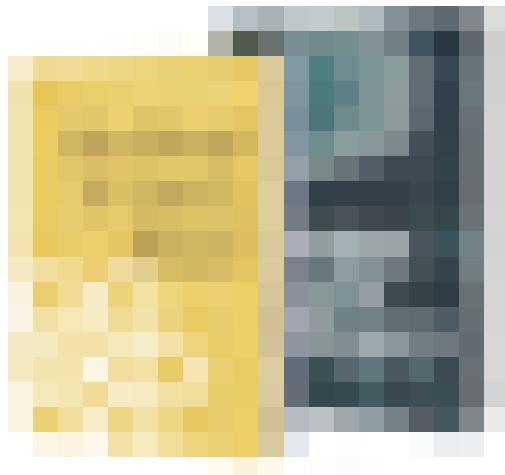
MOCKING A GRAPHQL SERVER FOR APOLLO CLIENT

Often you run into the case where you have to mock your GraphQL server for your GraphQL client application. It can be for testing your GraphQL client or when your GraphQL server is not (always...)

GRAPHQL SERVER TUTORIAL WITH APOLLO SERVER AND EXPRESS



In this chapter, you will implement server-side architecture using GraphQL and Apollo Server. The GraphQL query language is implemented as a reference implementation in JavaScript by Facebook, while...



THE ROAD TO REACT

Learn React by building real world applications. No setup configuration. No tooling. Plain React in 200+ pages of learning material. Learn React like **50.000+ readers.**

[GET THE BOOK](#)

Get it on Amazon.



TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.



✓ Join 50.000+ Developers

✓ Learn Web Development

✓ Learn JavaScript

✓ Access Tutorials, eBooks and Courses

✓ Personal Development as a Software Engineer

[SUBSCRIBE >](#)

[View our Privacy Policy.](#)

PORFOLIO

Online Courses

Open Source

Tutorials

ABOUT

About me

What I use

How to work with me

How to support me

© Robin Wieruch



[Contact Me](#) [Privacy & Terms](#)