Introduction to GraphQL

Learn useful terminology and concepts for using the GitHub GraphQL API.

In this article

GraphQL terminology

The GitHub GraphQL API represents an architectural and conceptual shift from the GitHub REST API. You will likely encounter some new terminology in the GraphQL API reference docs.

Schema

A schema defines a GraphQL API's type system. It describes the complete set of possible data (objects, fields, relationships, everything) that a client can access. Calls from the client are validatedand executed against the schema. A client can find information about the schema via introspection. A schema resides on the GraphQL API server. For more information, see "Discovering the GraphQL API."

Field

A field is a unit of data you can retrieve from an object. As the official GraphQL docs say: "The GraphQL query language is basically about selecting fields on objects."

The official spec also says about fields:

> All GraphQL operations must specify their selections down to fields which return scalar values to ensure an unambiguously shaped response.

This means that if you try to return a field that is not a scalar, schema validation will throw an error. You must add nested subfields until all fields return scalars.

Argument

An argument is a set of key-value pairs attached to a specific field. Some fields require an argument. Mutations require an input object as an argument.

Implementation

A GraphQL schema may use the term *implements* to define how an object inherits from an interface.

Here's a contrived example of a schema that defines interface `X` and object `Y`:

```
interface X {
  some_field: String!
  other_field: String!
}

type Y implements X {
  some_field: String!
  other_field: String!
  new_field: String!
}
```

This means object `Y` requires the same fields/arguments/return types that interface `X` does, while adding new fields specific to object `Y`. (The `!` means the field is required.)

In the reference docs, you'll find that:

- Each object lists the interface(s) *from which it inherits* under **Implements**.
- Each interface lists the objects *that inherit from it* under **Implementations**.

Connection

Connections let you query related objects as part of the same call. With connections, you can use a single GraphQL call where you would have to use multiple calls to a REST API. For more information, see "Migrating from REST to GraphQL."

It's helpful to picture a graph: dots connected by lines. The dots are nodes, the lines are edges. A connection defines a relationship between nodes.

Edge

Edges represent connections between nodes. When you query a connection, you traverse its edges to get to its nodes. Every `edges` field has a `node` field and a `cursor` field. Cursors are used for pagination.

Node

Node is a generic term for an object. You can look up a node directly, or you can access related nodes via a connection. If you specify a `node` that does not return a scalar, you must include subfields until all fields return scalars. For information on accessing node IDs via the REST API and using them in GraphQL queries, see "Using Global Node IDs."

Discovering the GraphQL API

GraphQL is introspective. This means you can query a GraphQL schema for details about itself.

- Query `__schema` to list all types defined in the schema and get details about each:

```
query {
__schema {
  types {
    name
    kind
    description
    fields {
      name
    }
  }
}
}
```

- Query `__type` to get details about any type:

```
query {
__type(name: "Repository") {
  name
  kind
  description
  fields {
    name
  }
}
}
```

- You can also run an *introspection query* of the schema via a `GET` request:

  ```
  $ curl -H "Authorization: bearer token" https://api.github.com/graphql
  ```

  The results are in JSON, so we recommend pretty-printing them for easier reading and searching. You can use a command-line tool like jq or pipe the results into `python -m json.tool` for this purpose.

  Alternatively, you can pass the `idl` media type to return the results in IDL format, which is a condensed version of the schema:

  ```
  $ curl -H "Authorization: bearer token" -H "Accept: application/vnd.github.v4.idl" \
  https://api.github.com/graphql
  ```

  > **Note**: The introspection query is probably the only `GET` request you'll run in GraphQL. If you're passing a body, the GraphQL request method is `POST`, whether it's a query or a mutation.