GraphQL Variables

In this lesson, we will learn how to access the GraphQL schema and will introduce GraphQL variables and discuss their importance.

WE'LL COVER THE FOLLOWING

Exploring the GraphQL Schema

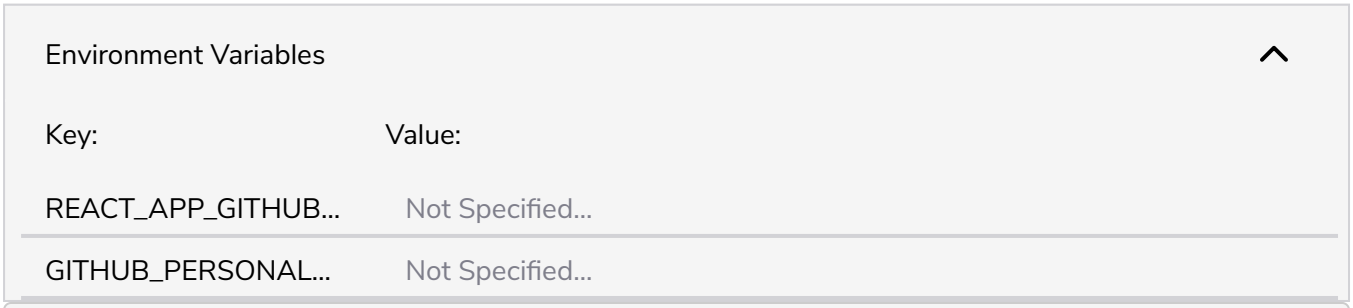
At this point, try opening the GraphQL **schema** by clicking on “Docs” on the right side of your GraphiQL application. This will open the *documentation explorer* giving you access to the GraphQL schema. A schema exposes the GraphQL API used by your GraphiQL application; Github’s GraphQL API in this case. It defines the GraphQL **graph** that is accessible via the GraphQL API using queries and mutations. Since it is a graph, objects and fields can be deeply nested in it which we will certainly encounter as we move along.

Since we’re exploring queries and not mutations at the moment, select “Query” in the “Docs” sidebar. Afterward, traverse the objects and fields of the graph and explore their optional arguments. By clicking them, you can see the accessible fields within those objects in the graph. Some fields are common

GraphQL types such as `String`, `Int` and `Boolean`, while some are custom like the `Organization` type we used. In addition, you can see whether arguments are required when requesting fields on an object; this can be identified by the exclamation point. For instance, a field with a `String!` argument requires you to pass in a `String` argument whereas a field with a simple `String` argument doesn't.

When Do we Use GraphQL Variables?

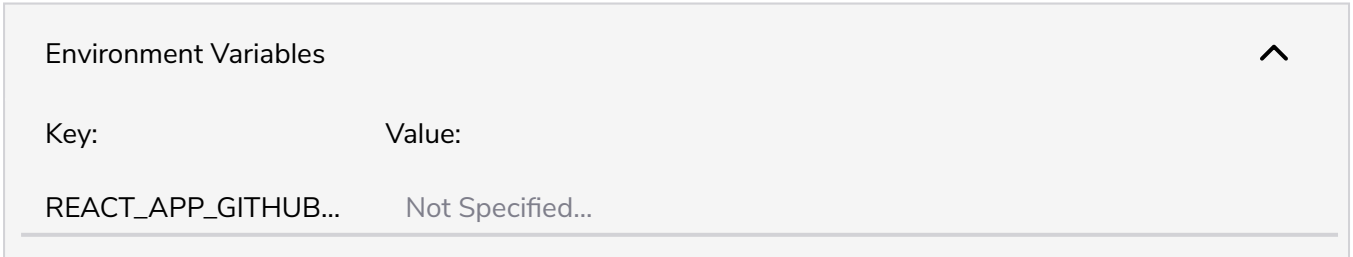
In the previous queries, you provided arguments that identified an organization to your fields, but you inlined these arguments in your query. Think about a query like a function, to which you, have to provide dynamic arguments. This is where the variable in GraphQL comes in as it allows arguments to be extracted as variables from queries. Here's how an organization's `login` argument can be extracted as a dynamic variable:




```
query ($organization: String!) {
  organization(login: $organization) {
    name
    url
  }
}
```

This defines the `organization` argument as a variable using the `$` sign. Furthermore, the argument's type is defined as a `String` and since the argument is required to fulfill the query, the `String` type has an exclamation point.

In the "Query Variables" panel, the variables would have the following content for providing the `organization` variable as an argument for the query:



GITHUB_PERSONAL... Not Specified...

```
{
  "organization": "the-road-to-learn-react"
}
```

Essentially, variables can be used to create dynamic queries. Following the best practices in GraphQL, we don't need manual string interpolation to structure a dynamic query later on. Instead, we provide a query that uses variables as arguments which are available when the query is sent as a request to the GraphQL API. You will see both implementations later on in your React application.

You can also define a **default variable** in GraphQL. It has to be a non-required argument or an error will occur regarding a **nullable variable** or **non-null variable**. For learning about default variables, we will make the `organization` argument non-required by omitting the exclamation point. Afterwards, it can be passed as a default variable.

Environment Variables

Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
query ($organization: String = "the-road-to-learn-react") {
  organization(login: $organization) {
    name
    url
  }
}
```

Try to execute the previous query with two sets of variables: once with the `organization` variable different from the default variable, and once without defining the `organization` variable.

That is **variables** in a nutshell. Let's move on to the structure of a GraphQL query.