Queries and Organizations

In this lesson, we will start learning how to request data from public organizations.

GitHub Organization

While a good way to learn how to write queries, accessing data about oneself isn't too beneficial. So, how do we request data from other sources? Let's find out using GitHub's public **organizations**.

Organizations are shared accounts where groups of people can collaborate across multiple projects at once.

To specify a GitHub organization, you can pass an **argument** to a field:

Environment Variables		︿
Key:	Value:	
REACT_APP_GITHUB...	Not Specified...	
GITHUB_PERSONAL...	Not Specified...	

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

Identifying an Organization

When using GitHub's API, you can identify an organization using a `login`. If you have previously used GitHub, you might know this is part of the organization URL: `https://github.com/the-road-to-learn-react`.

By providing a `login` to identify the organization, you now have the ability to request its data. In this example, you have specified two fields to access data about the organization's `name` and `URL`. The request should return something similar to the following output:

Environment Variables ⌄

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

```json
{
  "data": {
    "organization": {
      "name": "The Road to learn React",
      "url": "https://github.com/the-road-to-learn-react"
    }
  }
}
```

In the previous query, you passed an argument to a field, as you can imagine, you can add arguments to various fields using GraphQL. This grants a great deal of flexibility for structuring queries because you can make specifications to requests on a field level and arguments can be of different types. With the organization above, you provided an argument with the type `String` however, you can also pass types like enumerations with a fixed set of options, integers, or booleans.

GraphQL Aliases

If you ever want to request data about two identical objects, you would have to use aliases in GraphQL. The following query would not be possible because GraphQL would not know how to resolve the two organization objects in a result:

Environment Variables ⌄

```
{
  organization(login: "the-road-to-learn-react") {
    name
    url
  }
  organization(login: "facebook") {
    name
    url
  }
}
```

You'd see an error such as `Field 'organization' has an argument conflict`. This is where aliases come in; they let you rename the result of a field to anything you want. By using aliases, you can resolve the result into two blocks:

Environment Variables   ⌄

Key: Value:

REACT_APP_GITHUB...   Not Specified...

GITHUB_PERSONAL...   Not Specified...

```
{
  book: organization(login: "the-road-to-learn-react") {
    name
    url
  }
  company: organization(login: "facebook") {
    name
    url
  }
}
```

The result should be similar to the following:

Environment Variables   ⌄

Key: Value:

REACT_APP_GITHUB...   Not Specified...

GITHUB_PERSONAL...   Not Specified...

```
{
  "data": {
```

```
    "book": {
      "name": "The Road to learn React",
      "url": "https://github.com/the-road-to-learn-react"
    },
    "company": {
      "name": "Facebook",
      "url": "https://github.com/facebook"
    }
  }
}
```

GraphQL Fragments

Next, imagine you want to request multiple fields for both organizations. Re-typing all the fields for each organization would make the query repetitive and verbose, so instead, we will use fragments to extract the query's reusable parts. **Fragments** let you construct sets of fields, and then include them in queries where you need to. They are especially useful when your query becomes deeply nested and uses lots of shared fields.

Environment Variables	⌃

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

```
{
  book: organization(login: "the-road-to-learn-react") {
    ...sharedOrganizationFields
  }
  company: organization(login: "facebook") {
    ...sharedOrganizationFields
  }
}

fragment sharedOrganizationFields on Organization {
  name
  url
}
```

As you can see, you have to specify the types of objects the fragment can be used on. In this case, the type is `Organization` which is a custom type defined by GitHub's GraphQL API. This is how you use fragments to extract and reuse parts of your queries.

Now let's open the GraphQL schema and move on to variables.