
Customizing and organizing
GraphQL operations
LEARNING
VOYAGE

Customizing & organizing GraphQL operations

This lesson covers

- Using arguments to customize what a request field returns
- Customizing response property names with aliases
- Describing runtime executions with directives
- Reducing duplicated text with fragments
- Composing queries and separating data requirement responsibilities

Customizing fields with arguments

- The fields in a GraphQL operation are similar to functions. They map input to output.
- A function input is received as a list of argument values.
- Just like functions, we can pass any GraphQL field a list of argument values.
- A GraphQL schema on the backend can access these values and use them to customize the response it returns for that field.

Identifying a single record to return

- Example query that asks for information about the user whose email address is jane@doe.name.

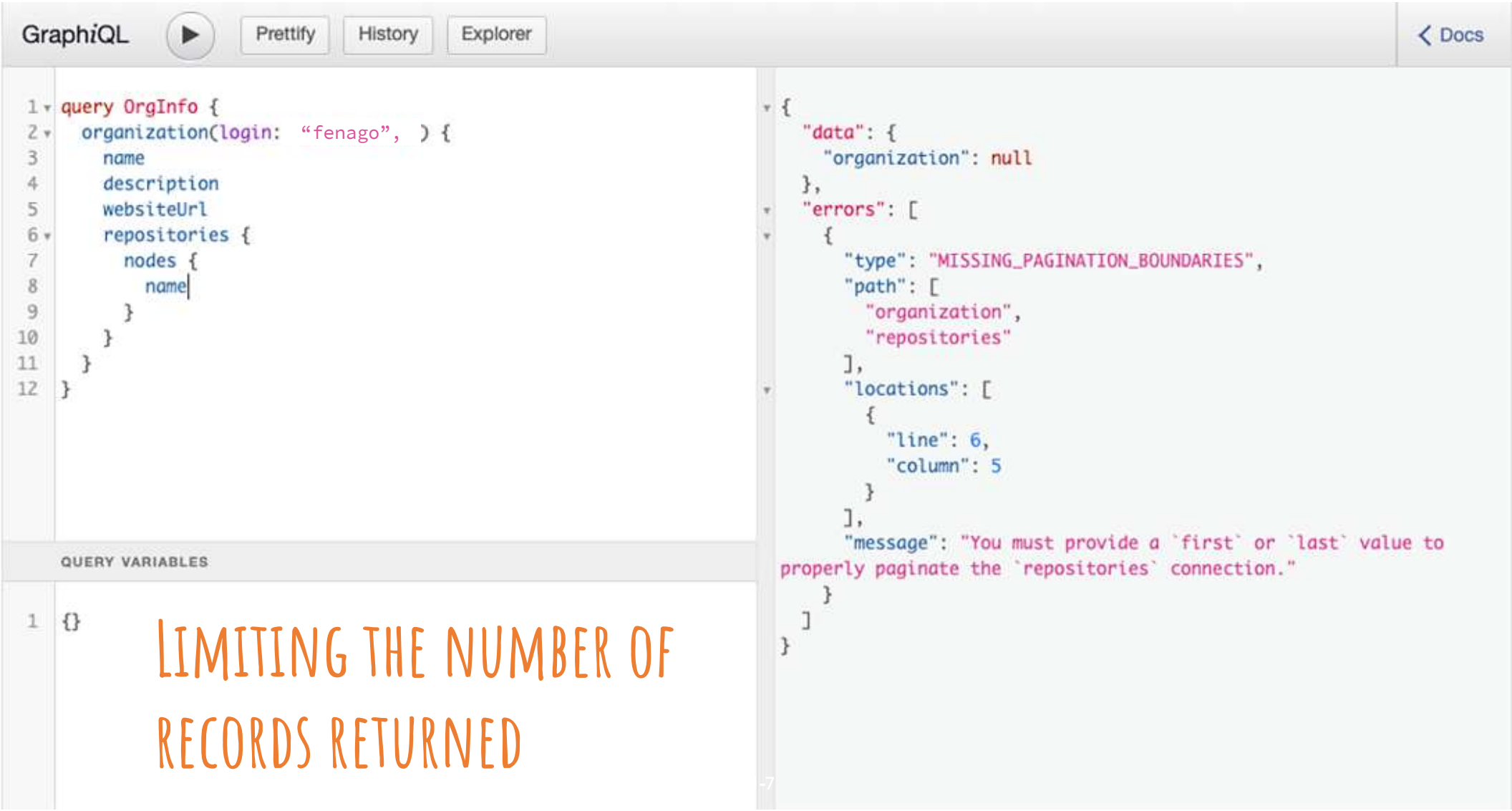
```
query UserInfo {
  user(email: "jane@doe.name") {
    firstName
    lastName
    username
  }
}
```

Identifying a single record to return

```
query NodeInfo {
  node(id: "A-GLOBALLY-UNIQUE-ID-HERE") {
    ...on USER {
      firstName
      lastName
      username
      email
    }
  }
}
```

Identifying a single record to return

```
query OrgInfo {
  organization(login: "fenago") {
    name
    description
    websiteUrl
  }
}
```



Limiting the number of records returned

Limiting the number of records returned

```
query First10Repos {
  organization(login: "fenago") {
    name
    description
    websiteUrl
    repositories(first: 10) {
      nodes {
        name
      }
    }
  }
}
```

Ordering records returned by a list field

```
query orgReposByName {
  organization(login: "fenago") {
    repositories(first: 10, orderBy: { field:
NAME, direction: ASC }) {
      nodes {
        name
      }
    }
  }
}
```

Ordering records returned by a list field

```
query OrgPopularRepos {
  organization(login: "fenago") {
    repositories(first: 10, orderBy: { field:
STARGAZERS, direction: DESC }) {
      nodes {
        name
      }
    }
  }
}
```

Paginating through a list of records

```
query OrgRepoConnectionExample {
  organization(login: "fenago") {
    repositories(first: 10, orderBy: { field:
CREATED_AT, direction: ASC }) {
      edges {
        cursor
        node {
          name
        }
      }
    }
  }
}
```

```
query OrgRepoConnectionExample2 {
  organization(login: "fenago") {
    repositories(
      first: 10,
      after:
"Y3Vyc29yOnYyOpK5MjAxNy0wMS0yMVQwODo1NTo0My0wODowMM4Ev4A3",
      orderBy: { field: CREATED_AT, direction: ASC }
    ) {
      edges {
        cursor
        node {
          name
        }
      }
    }
  }
}
```

```
query OrgReposMetaInfoExample {
  organization(login: "fenago") {
    repositories(
      first: 10,
      after: "Y3Vyc29yOnYyOpK5MjAxNy0wMS0yMVQwODo1NTo0My0wODowMM4Ev4A3",
      orderBy: { field: STARGAZERS, direction: DESC }
    ) {
      totalCount
      pageInfo {
        hasNextPage
      }
      edges {
        cursor
        node {
          name
        }
      }
    }
  }
}
```

Searching and filtering

```
query SearchExample {
  repository(owner: "twbs", name: "bootstrap") {
    projects(search: "v4.1", first: 10) {
      nodes {
        name
      }
    }
  }
}
```

Searching and filtering

```
query FilterExample {
  viewer {
    repositories(first: 10, affiliations: OWNER)
{
      totalCount
      nodes {
        name
      }
    }
  }
}
```

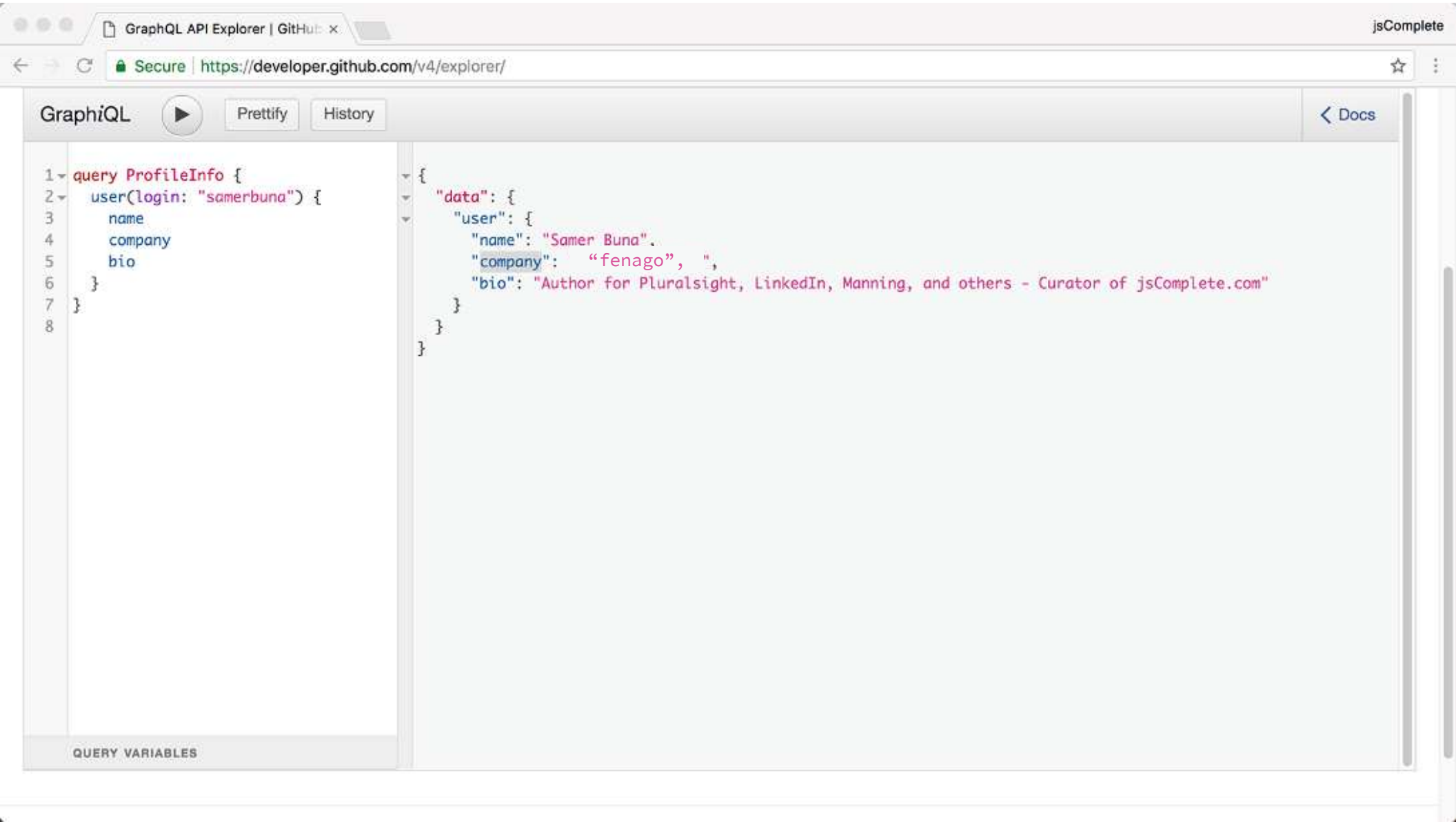
Providing input for mutations

```
mutation StarARepo {
  addStar(input: { starrableId:
"MDEwOlJlcG9zaXRvcnkxMjU2ODEwMDY=" }) {
    starrable {
      stargazers {
        totalCount
      }
    }
  }
}
```

Renaming fields with aliases

- Let's say you are developing the profile page in GitHub.
- Here is a query to retrieve partial profile information for a GitHub user.

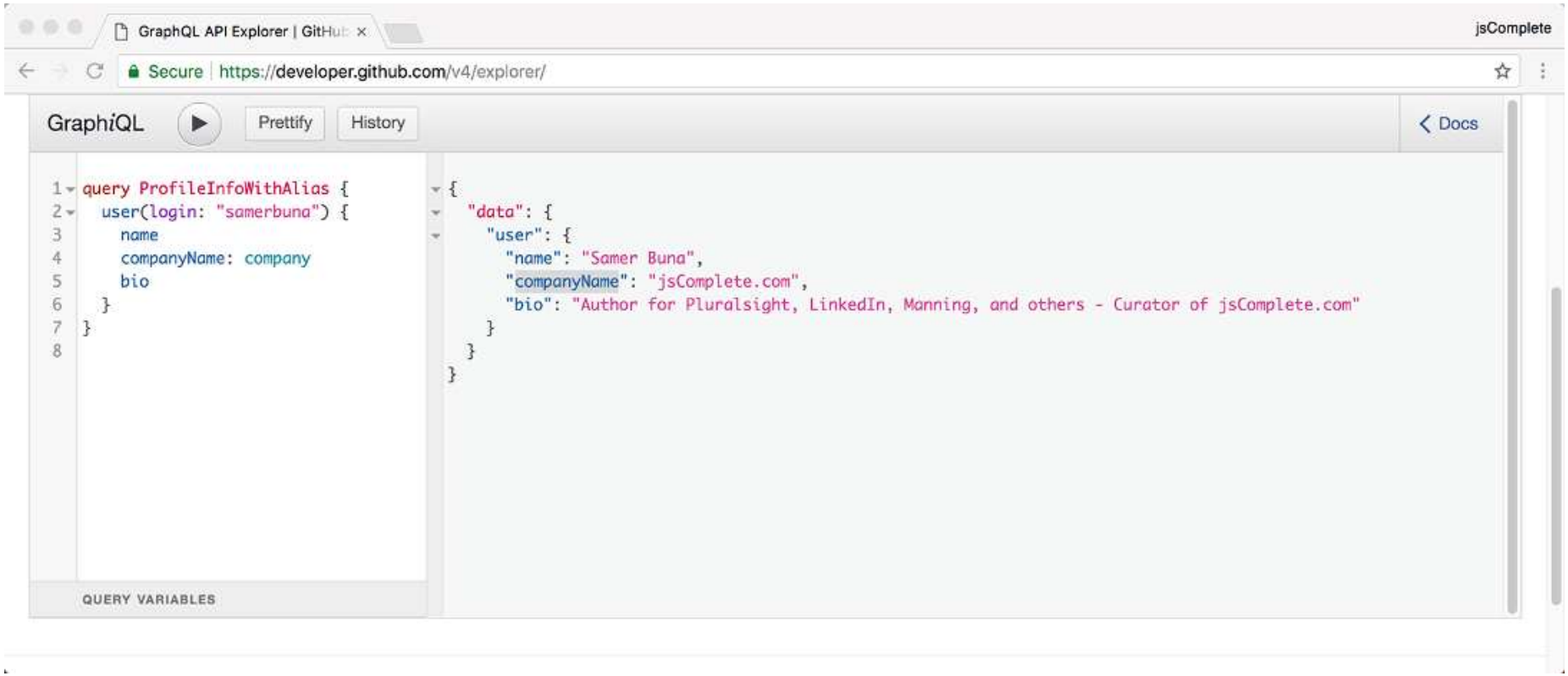
```
query ProfileInfo {
  user(login: "samerbuna") {
    name
    company
    bio
  }
}
```


GraphQL API Explorer | GitHub
jsComplete
Secure | https://developer.github.com/v4/explorer/
GraphiQL
Prettify
History
Docs
query ProfileInfo {
user(login: "samerbuna") {
name
company
bio
}
}
{
"data": {
"user": {
"name": "Samer Buna",
"company": “fenago”, ",
"bio": "Author for Pluralsight, LinkedIn, Manning, and others - Curator of jsComplete.com"
}
}
}
QUERY VARIABLES

Searching and filtering

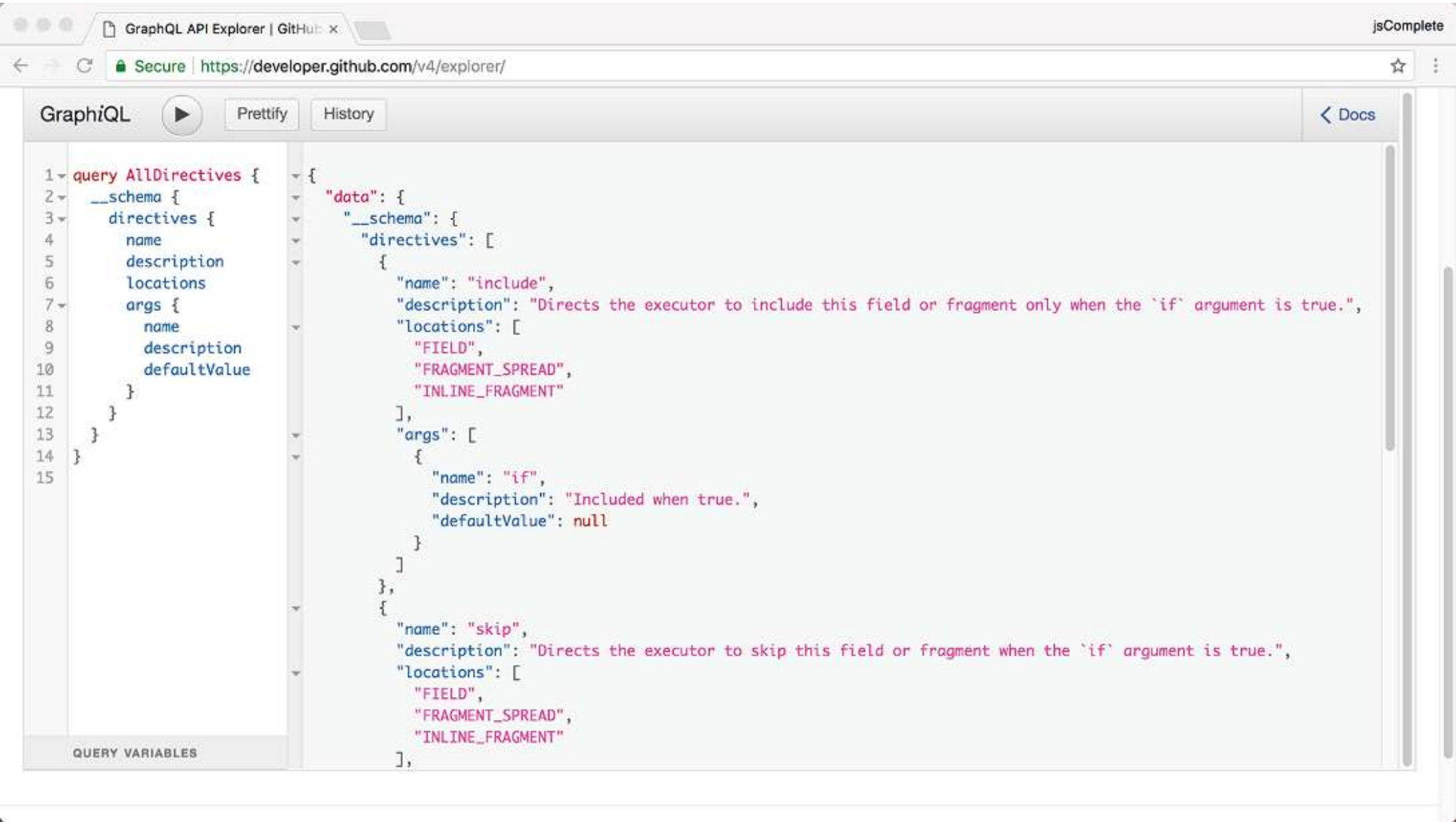
```
query ProfileInfoWithAlias {
  user(login: "samerbuna") {
    name
    companyName: company
    bio
  }
}
```

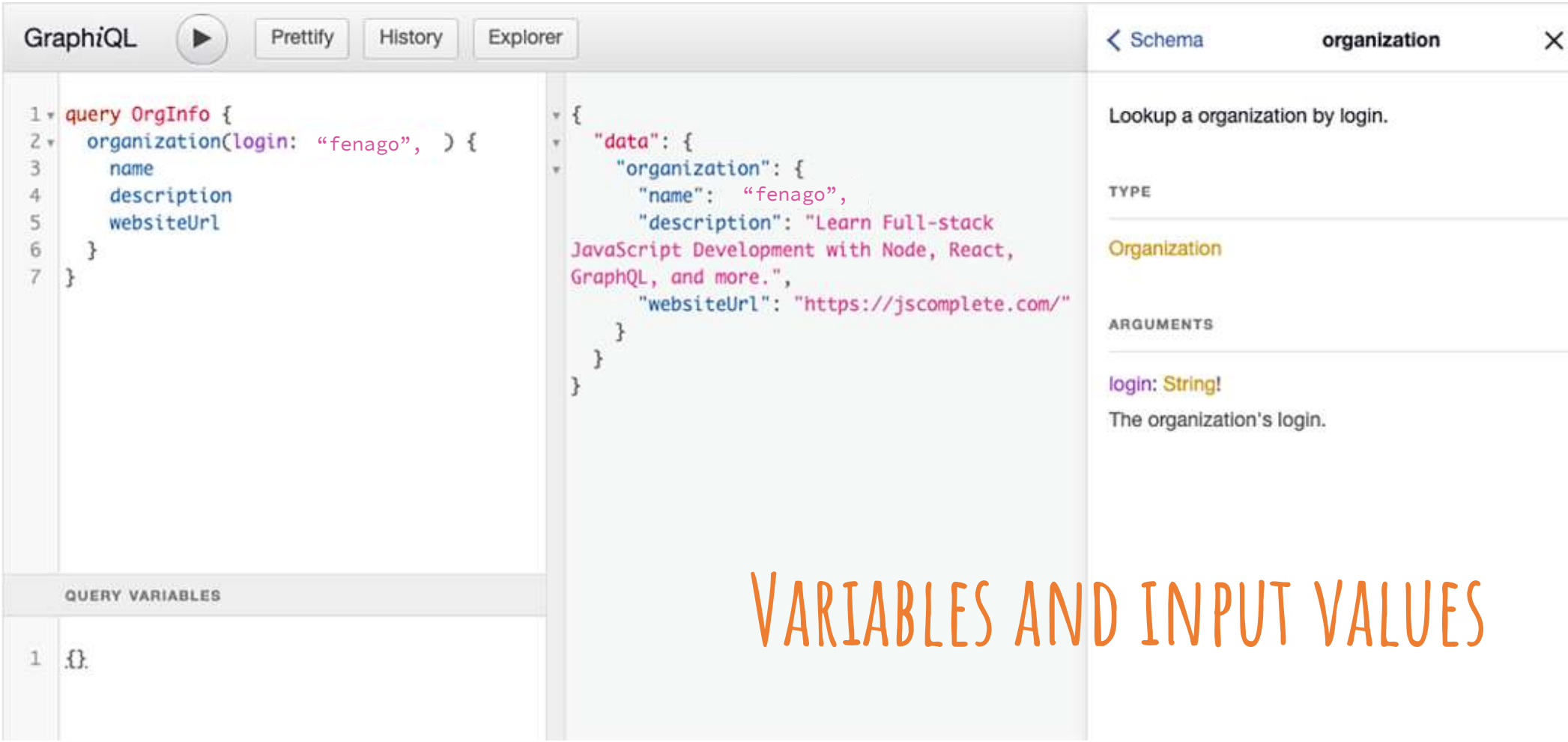
Searching and filtering



Customizing responses with directives

```
query AllDirectives {
  __schema {
    directives {
      name
      description
      locations
      args {
        name
        description
        defaultValue
      }
    }
  }
}
```

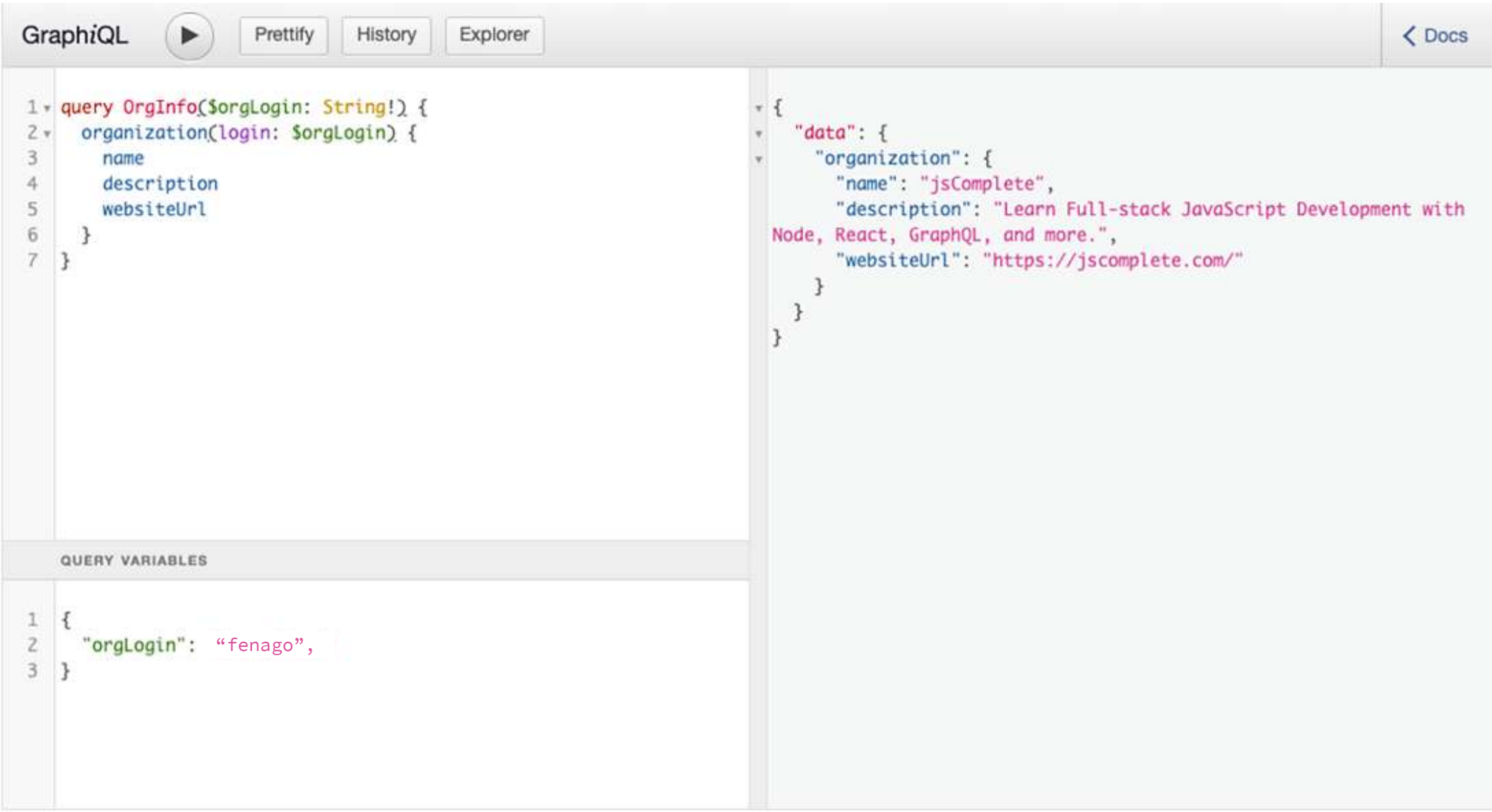

GraphQL API Explorer | GitHub
jsComplete
Secure | https://developer.github.com/v4/explorer/
GraphiQL
Prettify
History
Docs
query AllDirectives {
__schema {
directives {
name
description
locations
args {
name
description
defaultValue
}
}
}
}
QUERY VARIABLES
{
"data": {
"__schema": {
"directives": [
{
"name": "include",
"description": "Directs the executor to include this field or fragment only when the `if` argument is true.",
"locations": [
"FIELD",
"FRAGMENT_SPREAD",
"INLINE_FRAGMENT"
],
"args": [
{
"name": "if",
"description": "Included when true.",
"defaultValue": null
}
]
},
{
"name": "skip",
"description": "Directs the executor to skip this field or fragment when the `if` argument is true.",
"locations": [
"FIELD",
"FRAGMENT_SPREAD",
"INLINE_FRAGMENT"
],



Variables and input values

Variables and input values

```
query OrgInfo($orgLogin: String!) {
  organization(login: $orgLogin) {
    name
    description
    websiteUrl
  }
}
```


GraphiQL
Prettify
History
Explorer
Docs
query OrgInfo($orgLogin: String!) {
organization(login: $orgLogin) {
name
description
websiteUrl
}
}
QUERY VARIABLES
{
"orgLogin": “fenago”,
}
{
"data": {
"organization": {
"name": "jsComplete",
"description": "Learn Full-stack JavaScript Development with Node, React, GraphQL, and more.",
"websiteUrl": "https://jscomplete.com/"
}
}
}

Variables and input values

```
query OrgInfoWithDefault($orgLogin: String =
"fenago") {
  organization(login: $orgLogin) {
    name
    description
    websiteUrl
  }
}
```

The @include directive

- The @include directive can be used after fields (or fragments) to provide a condition (using its if argument).
- That condition controls whether the field (or fragment) should be included in the response.
- The use of the @include directive looks like this:

```
fieldName @include(if: $someTest)
```

The @include directive

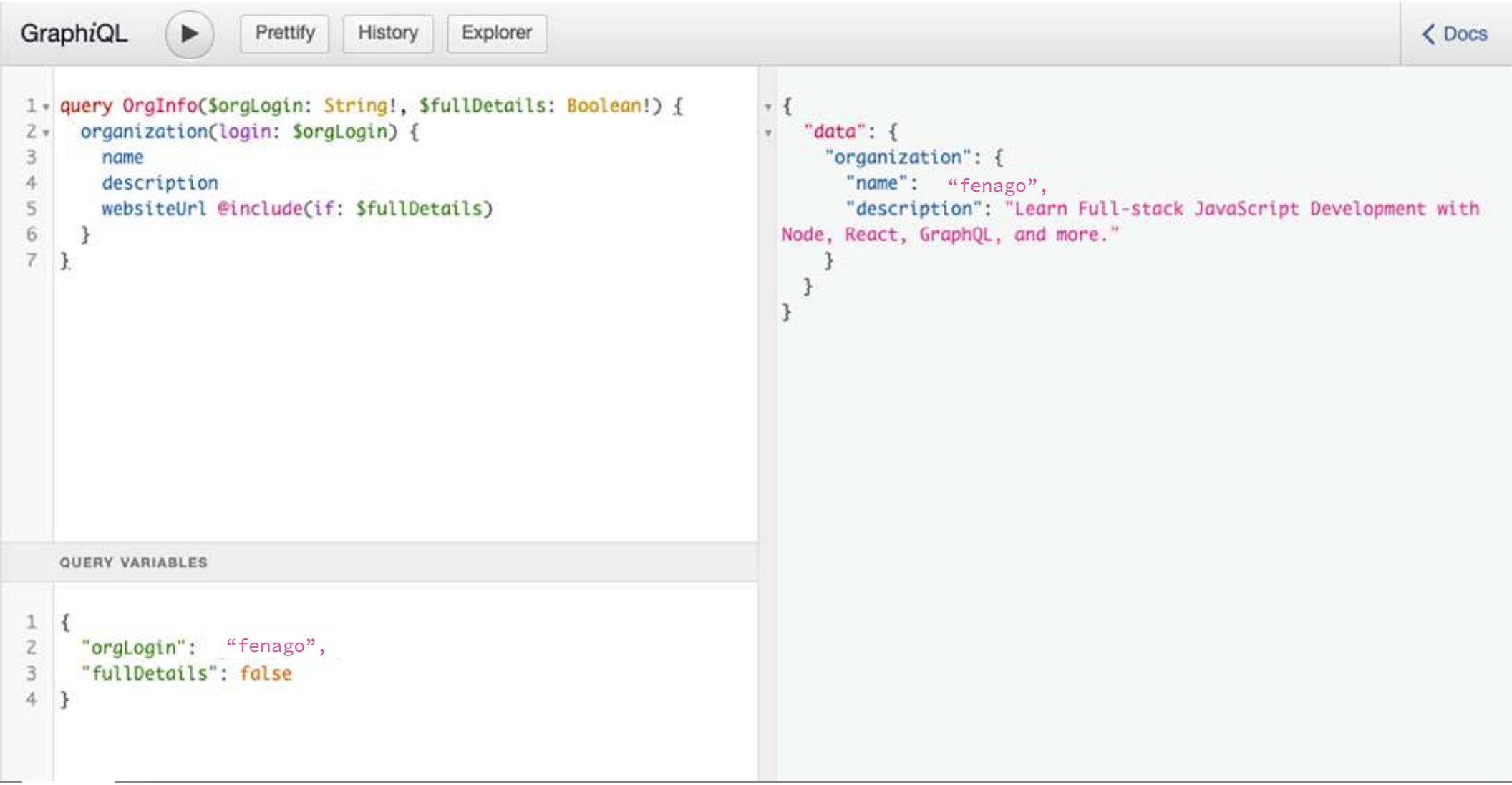
- The first line of the OrgInfo query needs to be changed to add the type of $fullDetails:

```
query OrgInfo($orgLogin: String!, $fullDetails:
Boolean!) {
```

The @include directive

- The if argument value in this case will be the $fullDetails variable. Here is the full query.

```
query OrgInfo($orgLogin: String!, $fullDetails:
Boolean!) {
  organization(login: $orgLogin) {
    name
    description
    websiteUrl @include(if: $fullDetails)
  }
}
```


GraphiQL
Prettify
History
Explorer
Docs
query OrgInfo($orgLogin: String!, $fullDetails: Boolean!) {
organization(login: $orgLogin) {
name
description
websiteUrl @include(if: $fullDetails)
}
}
{
"data": {
"organization": {
"name": "fenago",
"description": "Learn Full-stack JavaScript Development with Node, React, GraphQL, and more."
}
}
}
QUERY VARIABLES
{
"orgLogin": "fenago",
"fullDetails": false
}

The @skip directive

- This directive is simply the inverse of the @include directive.
- Just like the @include directive, it can be used after fields (or fragments) to provide a condition (using its if argument).
- The condition controls whether the field (or fragment) should be excluded in the response. The use of the @skip directive looks like this:

```
fieldName @skip(if: $someTest)
```

The @skip directive

```
query OrgInfo($orgLogin: String!, $partialDetails:
Boolean!) {
  organization(login: $orgLogin) {
    name
    description
    websiteUrl @skip(if: $partialDetails)
  }
}
```

The @skip directive

```
query OrgInfo($orgLogin: String!, $partialDetails:
Boolean!) {
  organization(login: $orgLogin) {
    name
    description
    websiteUrl @skip(if: $partialDetails)
@include(if: false)
  }
}
```

The @deprecated directive

- The following is the GraphQL's schema language representation of a type that has a deprecated field.

```
type User {
  emailAddress: String
  email: String @deprecated(reason: "Use
'emailAddress'.")
}
```

GraphQL fragments

Why fragments?

- To build anything complicated, the truly helpful strategy is to split what needs to be built into smaller parts and then focus on one part at a time.
- Ideally, the smaller parts should be designed in a way that does not couple them with each other.
- They should be testable on their own, and they should also be reusable.

Defining and using fragments

- For example, let's take the simple GitHub organization information query example:

```
query OrgInfo {
  organization(login: "fenago") {
    name
    description
    websiteUrl
  }
}
```

Defining and using fragments

- To make this query use a fragment, you first need to define the fragment.

```
fragment orgFields on Organization {
  name
  description
  websiteUrl
}
```

Defining and using fragments

- To use the fragment, you “spread” its name where the fields were originally used in the query.

```
query OrgInfoWithFragment {
  organization(login: "fenago") {
    ...orgFields
  }
}
```

Defining and using fragments

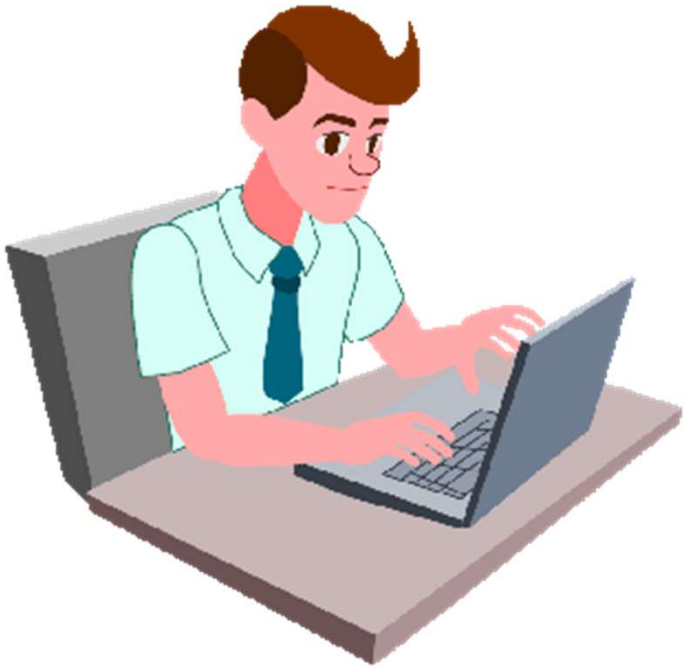
```
query MyRepos {
  viewer {
    ownedRepos: repositories(affiliations: OWNER, first: 10) {
      nodes {
        nameWithOwner
        description
        forkCount
      }
    }
    orgsRepos: repositories(affiliations: ORGANIZATION_MEMBER, first: 10) {
      nodes {
        nameWithOwner
        description
        forkCount
      }
    }
  }
}
```

Defining and using fragments

```
query MyRepos {
  viewer {
    ownedRepos: repositories(affiliations: OWNER, first: 10) {
      ...repoInfo
     }
    orgsRepos: repositories(affiliations: ORGANIZATION_MEMBER, first: 10) {
      ...repoInfo
    }
  }
}
fragment repoInfo on RepositoryConnection {
  nodes {
    nameWithOwner
    description
    forkCount
  }
}
```

Summary

- You can pass arguments to GraphQL fields when sending requests.
- GraphQL servers can use these arguments to support features like identifying a single record, limiting the number of records returned by a list field, ordering records and paginating through them, searching and filtering, and providing input values for mutations.



"Complete Lab"