Abstract

The Brief understanding of GitHub’s V4 GraphQL API.

Difference between GraphQL and REST

Migrating REST applications into GraphQL

Graphql

GitHub v4 API

Hyperion

CenturyLink

# Get started with graphql

GraphQL is a Declarative data fetching specification language developed by Facebook in 2012. It helped Facebook to get away with the data fetching challenges that had occurred in their native mobile application projects.

After using in their production for all their mobile apps for years, Facebook open sourced GraphQL in 2015. GitHub has early access to GraphQL, and it developed its v4 API using it. It represents an architectural and conceptual shift from the GitHub REST API v3.

Refer the below link for GitHub’s GraphQL Documentation

[developer.github.com/v4/](https://developer.github.com/v4/)

## key Features of GraphQL

* Open Source since 2015
* Language agnostic
* Can use many languages to build a graph server

# REST VS GRAPHQL

There are many different flavours in REST but typicaly we will request and update resources on different URIs.

EG:

Lets say, we have some data of an university[TAMUK].

To fetch all the information of the university we need to call the uri :

*/api/tamuk*

If we want to see the branch information of tamuk

/api/tamuk/cse

If we want data of students in computer sciecne and Electrical engineering courses available in the tamuk.

The urls might be

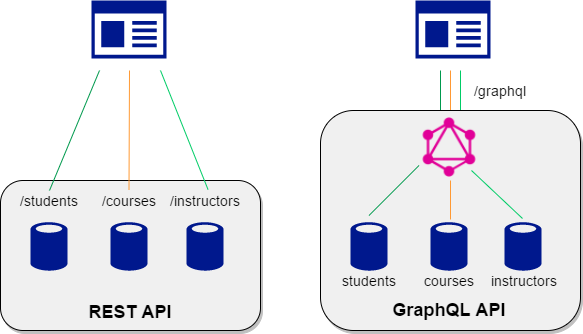
/api/tamuk/cse/students

/api/tamuk/eee/students

If we want to get the information of the international students, then we need to create a custom end point to fetch this data.

/api/tamuk/students/international\_students

Like this, we need to call several requests to fetch the required data. Due to this the respose time can be slower than our expectation.



And alternative to this we have GraphQL on the other side,

With GraphQL we define a Query. A query looks like a JSON object without values. Then we send this query as a string to the GraphQL server and we will get the response similar to the query. With the query we specify what excatly we need and we get nothing more or less than that.

As we want change the data that we want to fetch, we can update the query and fill in the fields. The query is nested and isssued at once.

EG: If we want the data of an univerity [TAMUK]

Query: Response:

{

“data” : {

“University”:{

“Name”: “tamuk”

“capacity“: 11,555,000

“people”: […]

}

}

}

University{

name

capacity

students

}

## Advantages of GraphQL

* Avoids multiple calls to the Database
  + It defines the shape of the desired data and call for it once. So It avoids multiple REST calls to the database.
  + And Solves the probles like Overfetching , under fetching.
  + Imporves the performance of the application.
* Its backward complatible and version-free
  + Meaning you can add new fields to the existing GraphQL server with out breaking the current clients.
  + And Old fields can be depricated and still continue to function.
* Can Wrap around the existing API
  + We can use GraphQL to wrap around the existing API. So we don’t have to set up an start everything from scratch but you can use the existing set up.
* GraphQL is language agnostic
  + Which means we can implement GraphQL solutions in a range of different laguages.

# Public GraphQl API – github v4 api

GitHub is a web-based version control repository and hosting service. After years of using REST for their APIs GitHub open sourced its v4 GraphQL API on 2016 in GitHub Universe.

Refer below article for more information on the GitHub’s shift to GraphQL,

[githubengineering/the-github-graphql-api/](https://github.blog/2016-09-14-the-github-graphql-api/)

Refer below link for GitHub’s GraphQL explorer, which uses GraphiQL in browser IDE to run queries.

<https://developer.github.com/v4/explorer/>

You can also use the GrapQL Playground IDE [Desktop App] to run your queries

<https://github.com/prisma/graphql-playground>

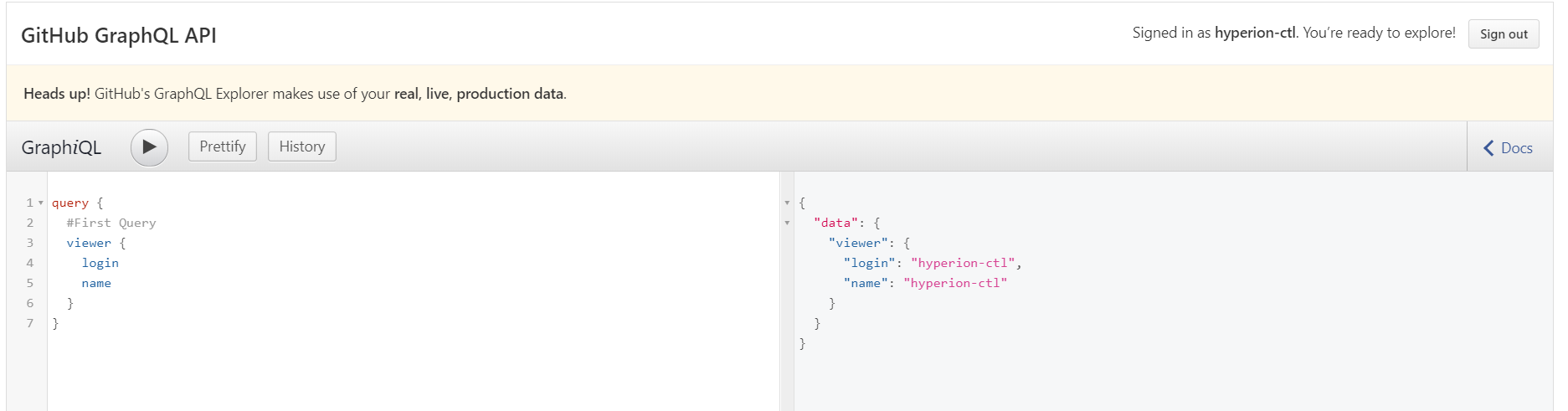
## GraphQL Queries

### Creating basic Queries

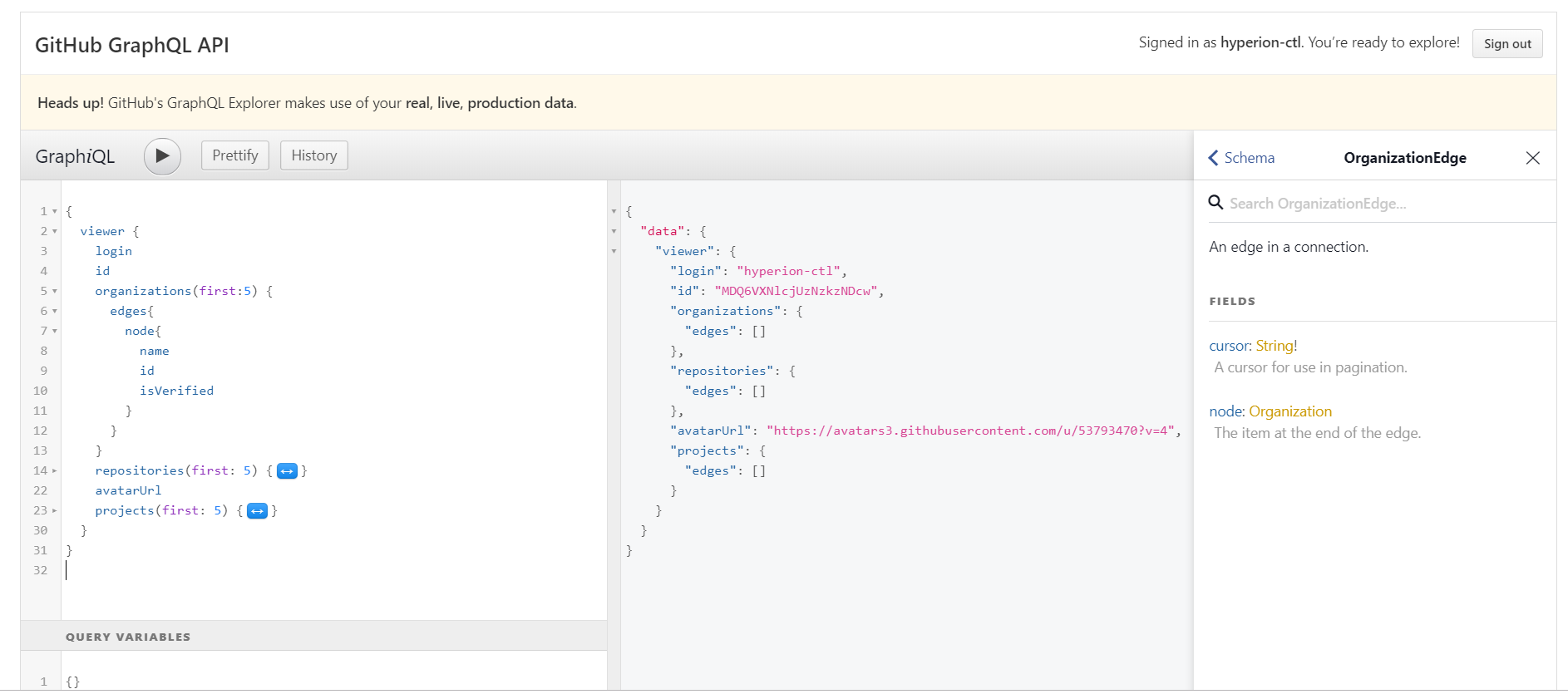
Let see from the below example, we logged in to the GitHub with the common hyperion-ctl account.

So that we can query the data available with the hyperion-ctl account.

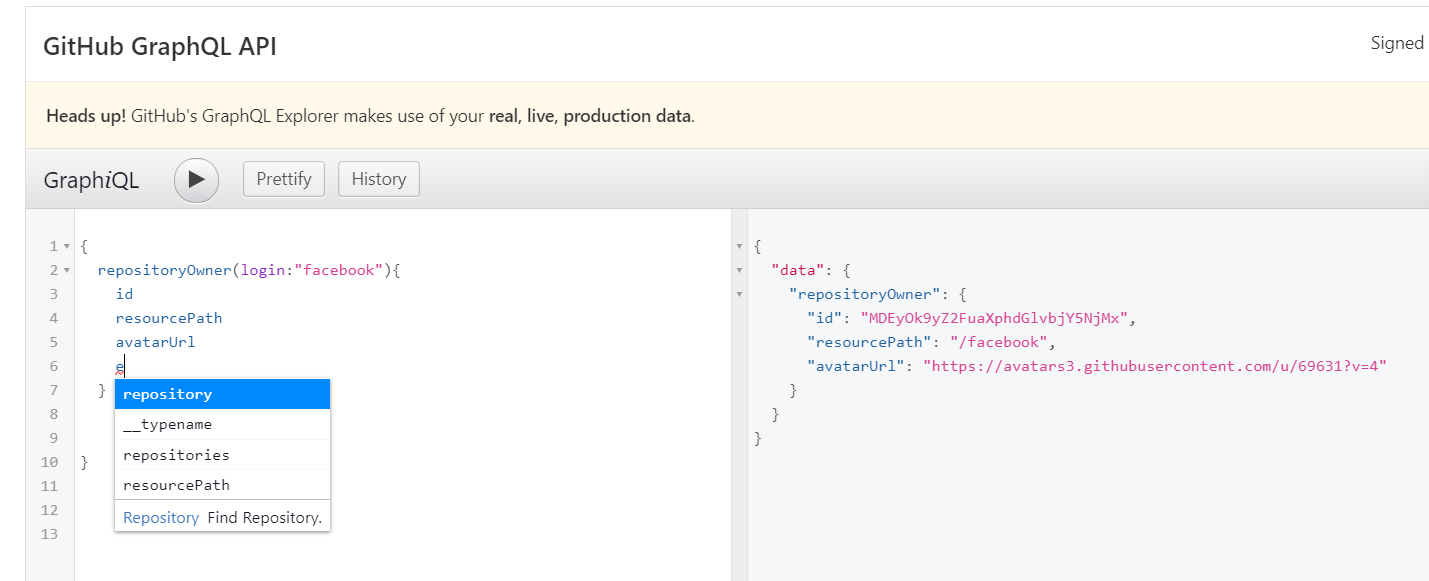
Below is the basic query to fetch the details [name, login] of the logged in user.



### Using multiple fields

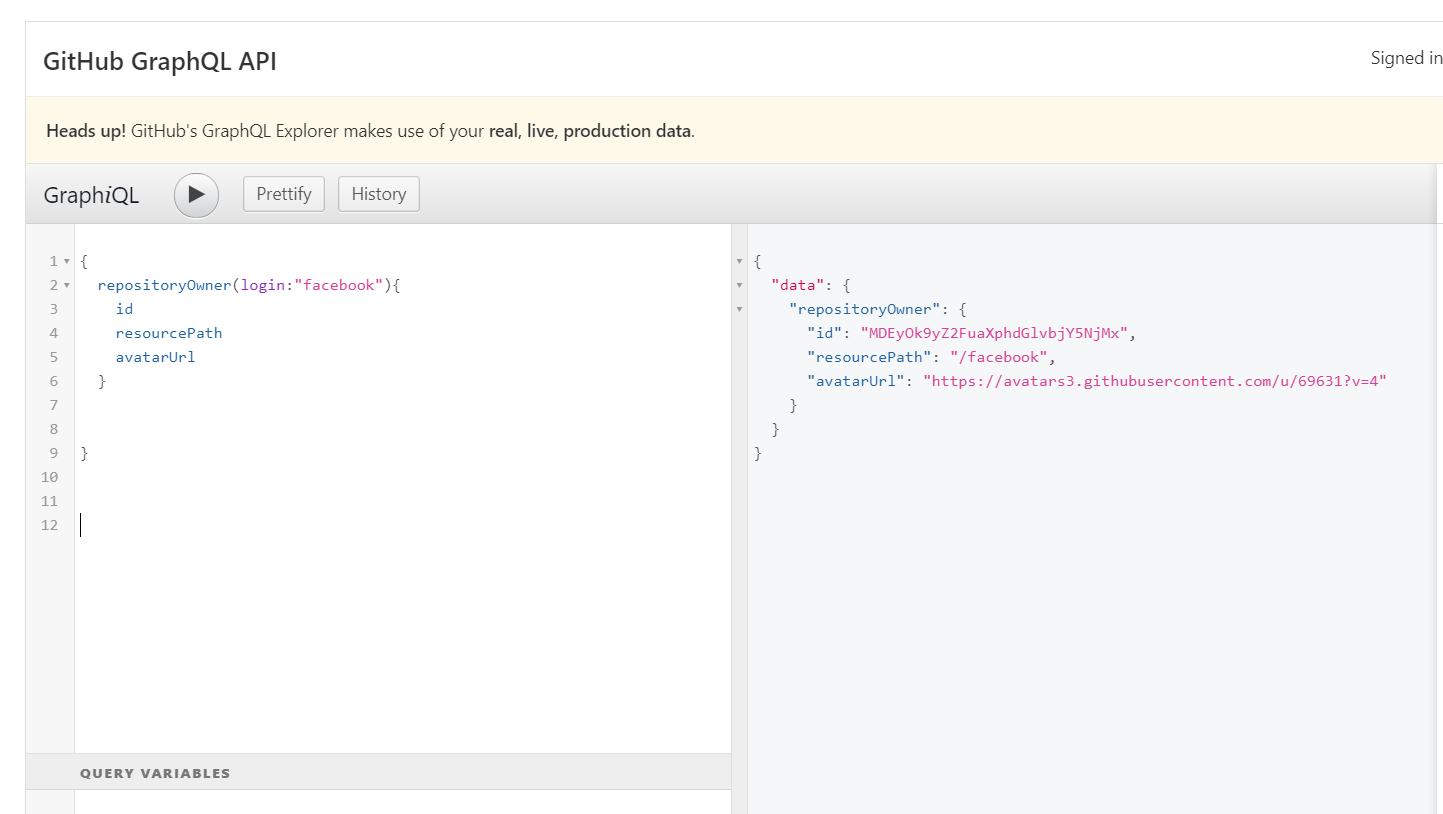


GraphQL will automatically suggest the available fields based on the node.



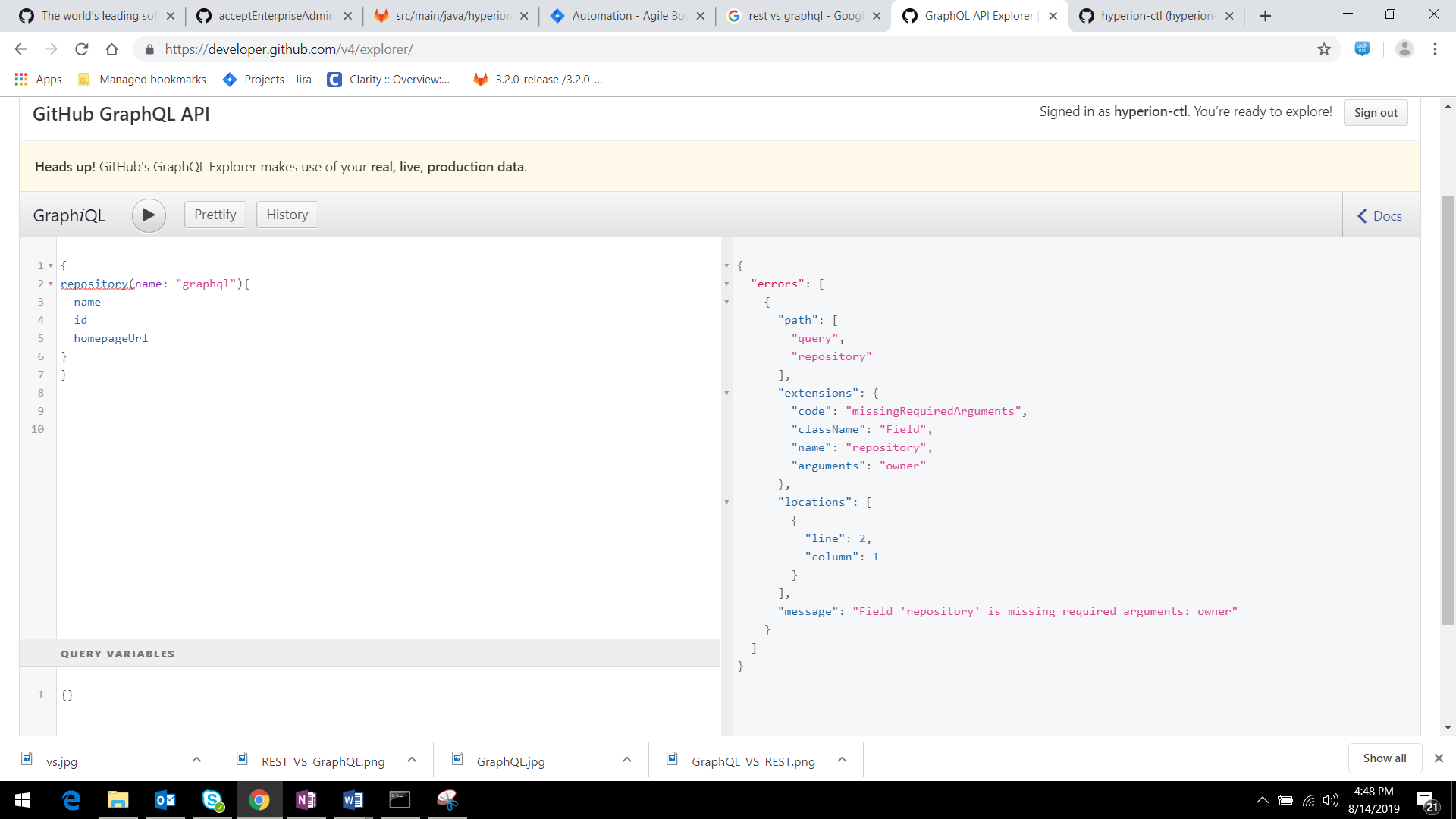
### Passing arguments

We can pass the arguments when ever its required and get the information of it.

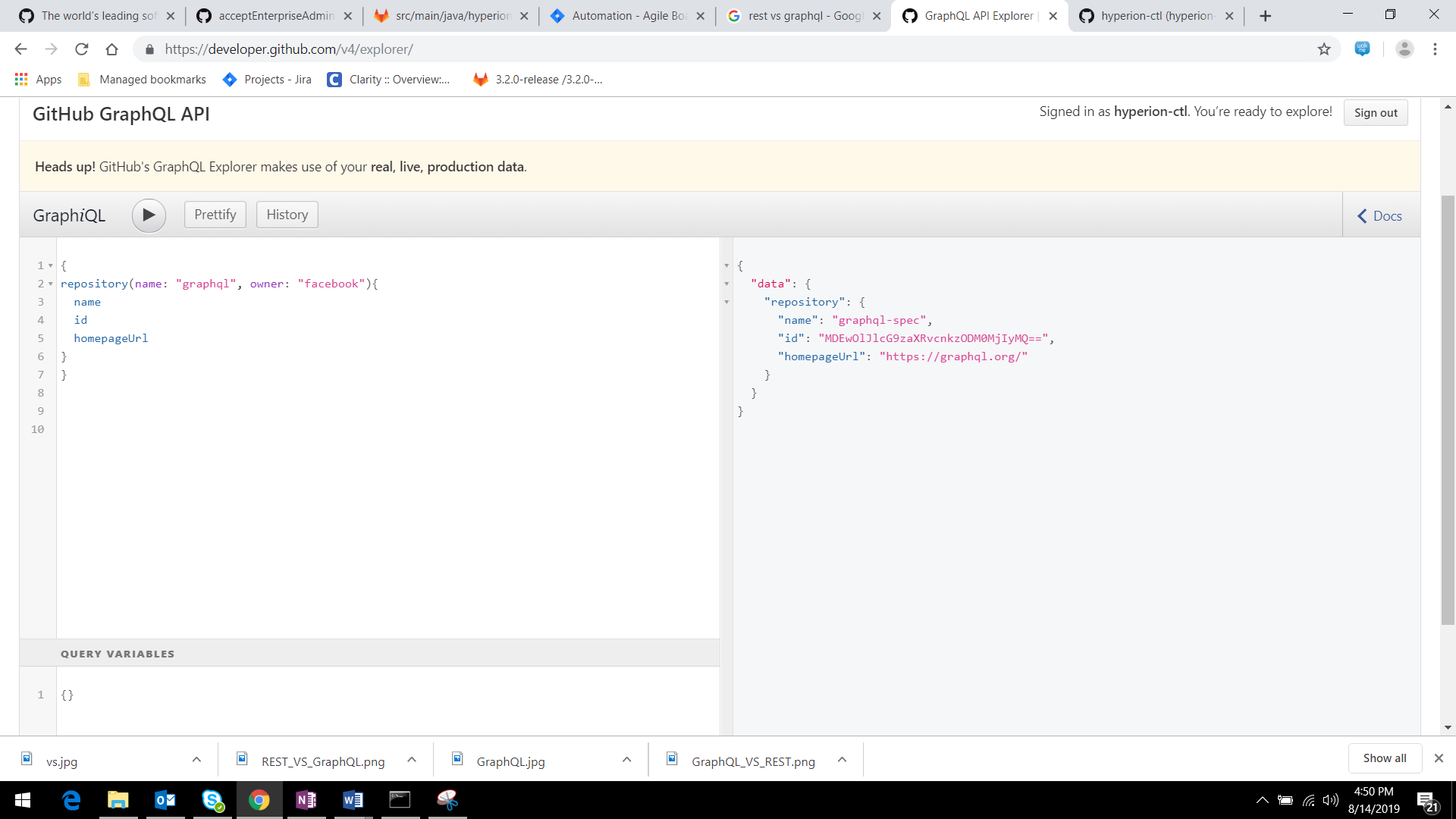


### Required arguments

We must provide the arguments when it is required, otherwise GraphQL will throw error.



If we hover over the error shown in the query, it will show the reason and its schema, so that we can correct it.



And GraphQL is case sensitive, if we provided Facebook instead of facebook as owner it wil throw error.

# Understanding the Schema

## GraphQL Schema

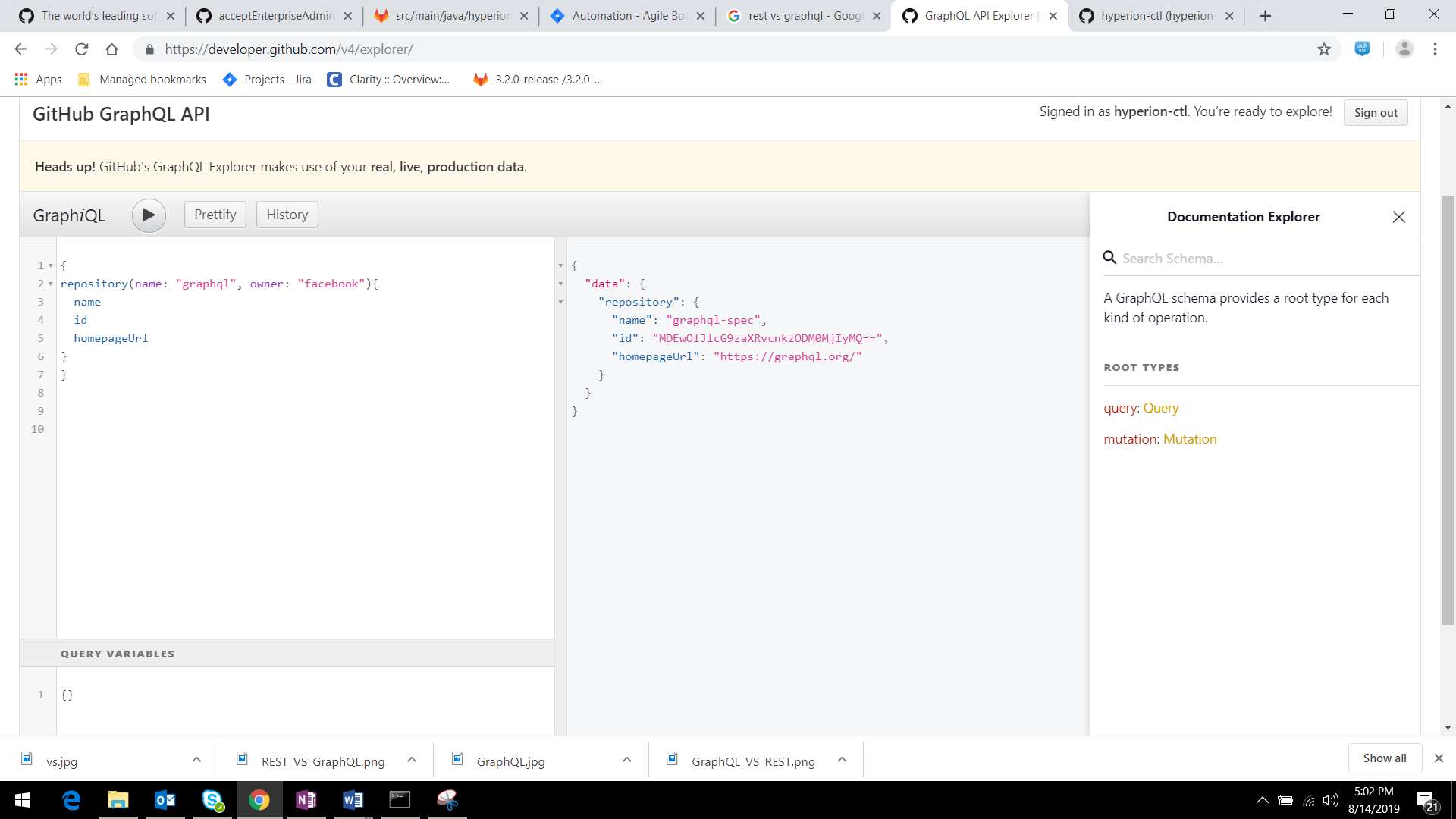
GraphQL schema provides all the object types that are used in our data and it specifies the types for all the values. GraphQL Schema is documented very well in the GraphiQL browser IDE.

If you clicks on the docs option on the right side corner of the online GraphiQL IDE, it will opens up the documentation explorer, once we provide the scema the docuementatiaon will automatically generate.

Here we have 2 different types

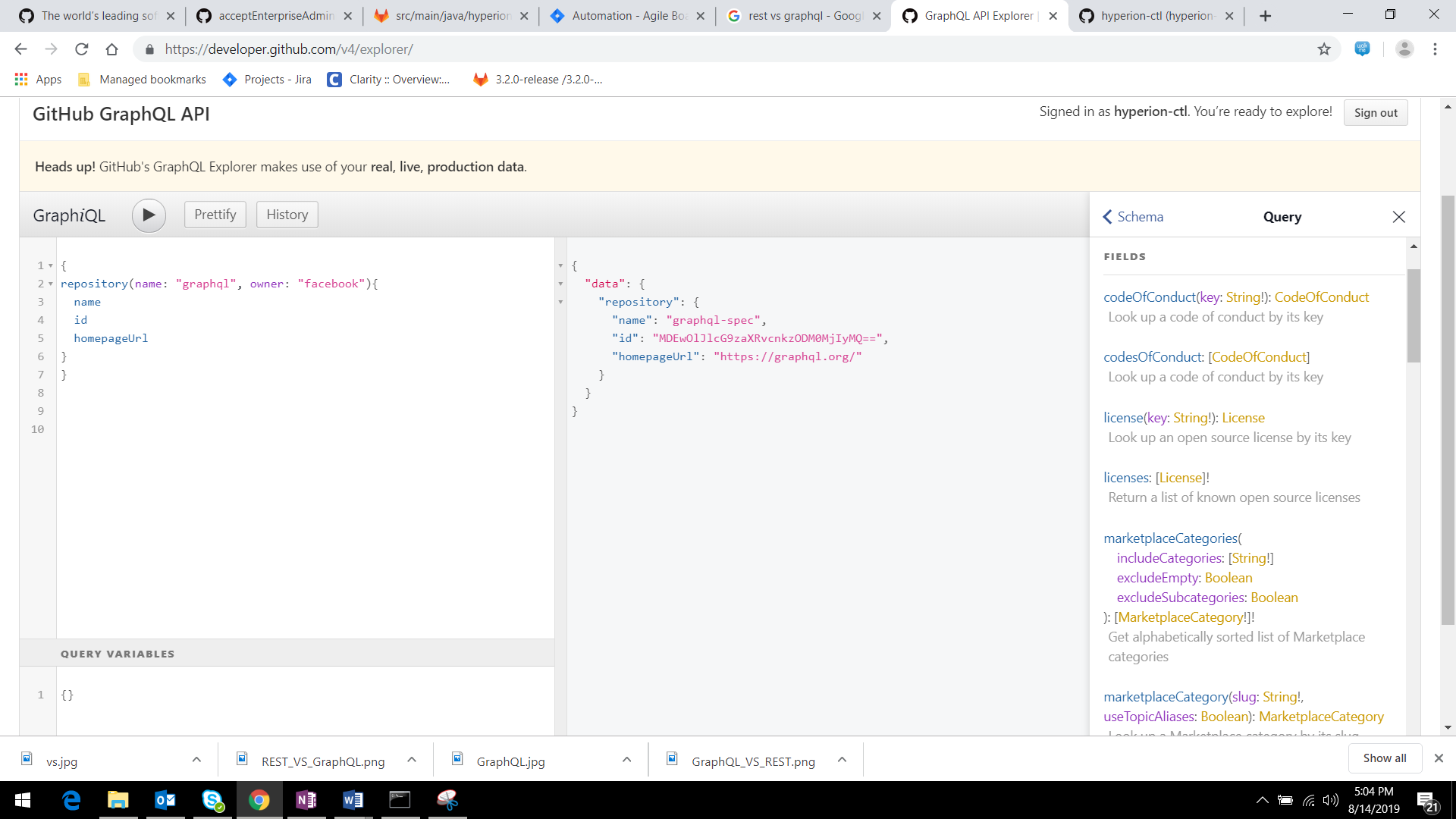
Query : for getting the data

Mutation : for changing data



## Schema types

we can see the types of the objects and the types associated with the fields in the documentation.

Input values type:

* Integer
* Float
* String
* Boolean
* Null
* Enum
* List

If you see any type field ends with “!” which means that field is required.

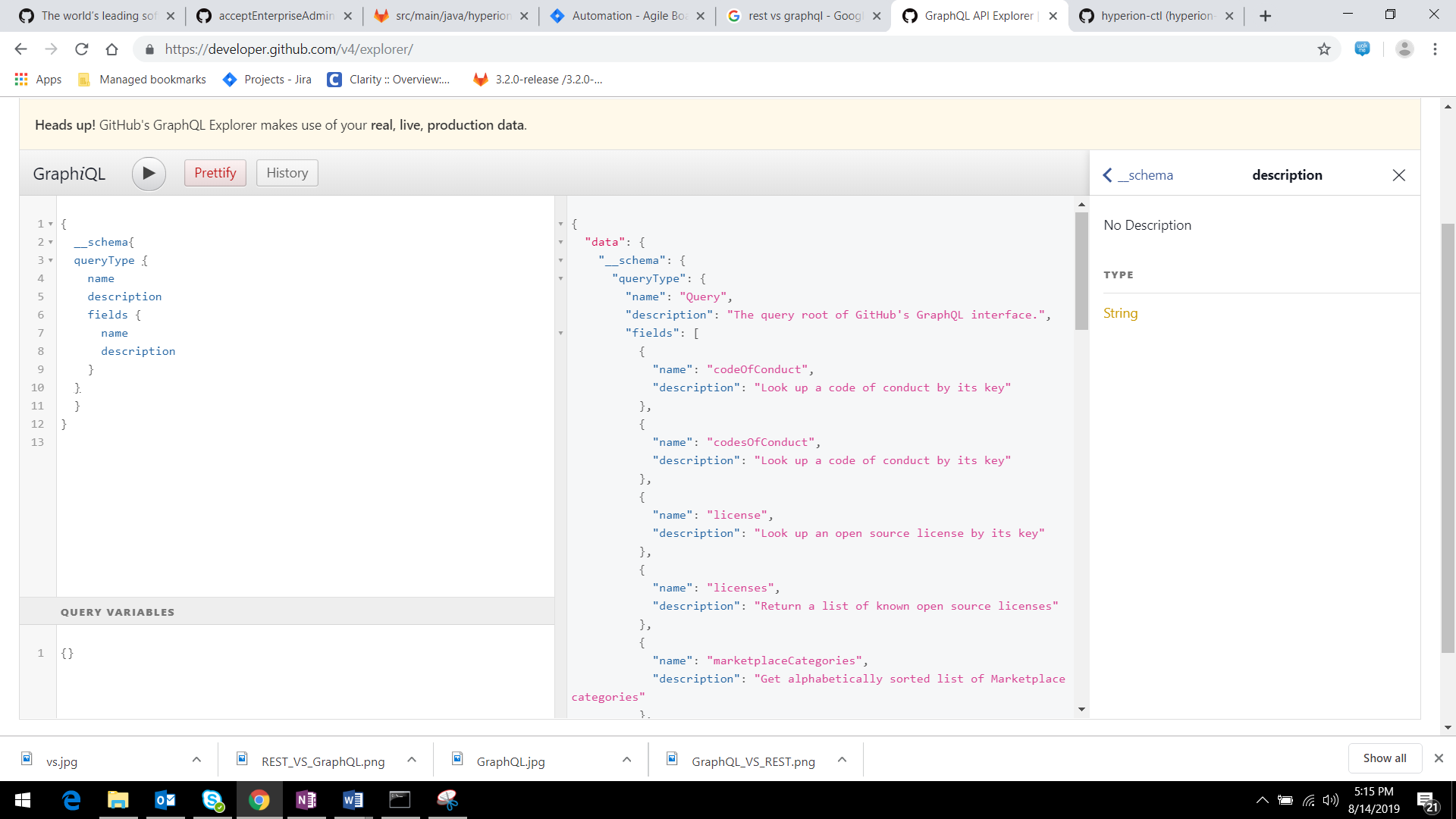
EG: license(key:String!)

Here the key field is required.

## Query the \_\_schema

\_\_schema is the root query of the GraphQL. We can get all the fields available under the root query.

* we can also see the fields which are deprecated in the response along with the non-deprecated field.
* To see whether the fields are deprecated on not we can add isDeprecated filed to the query.

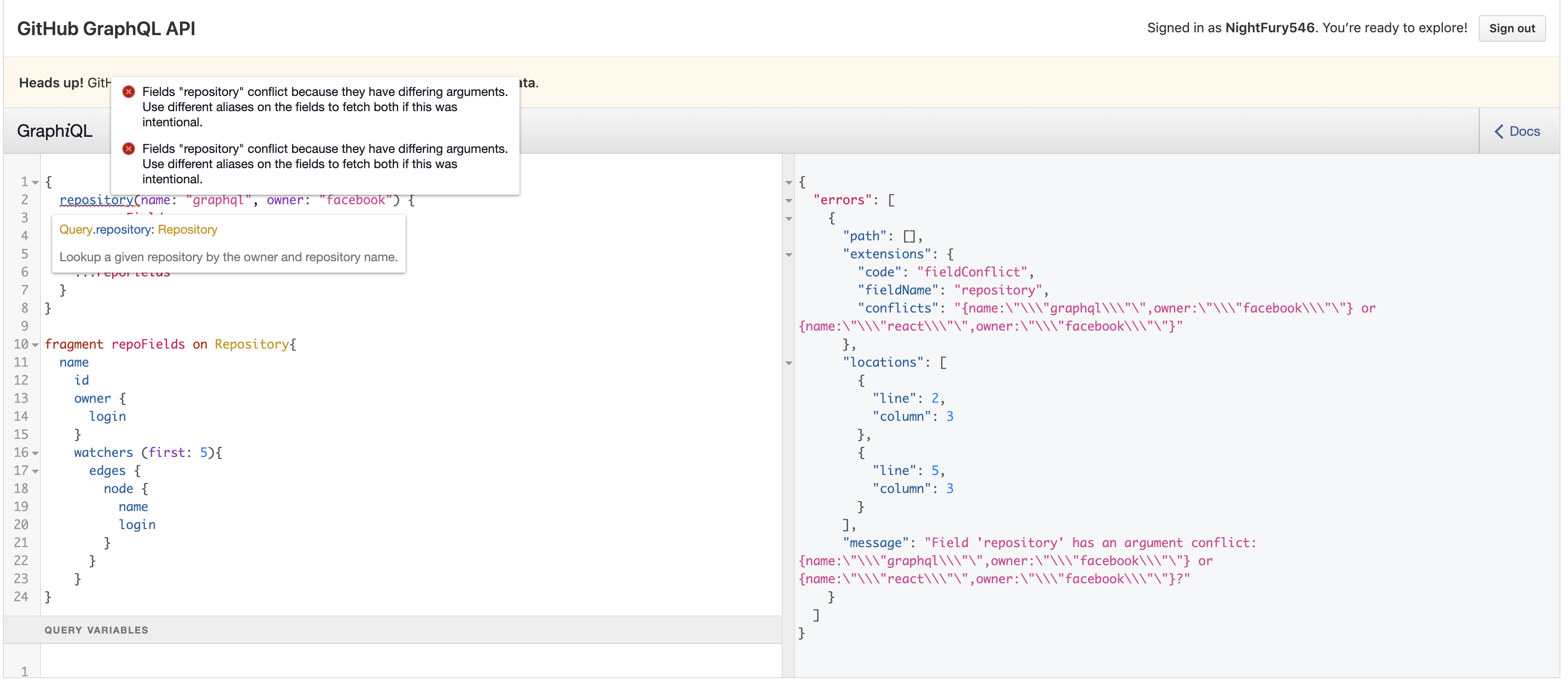


# Handling Data

This is how we handle data in GraphQl

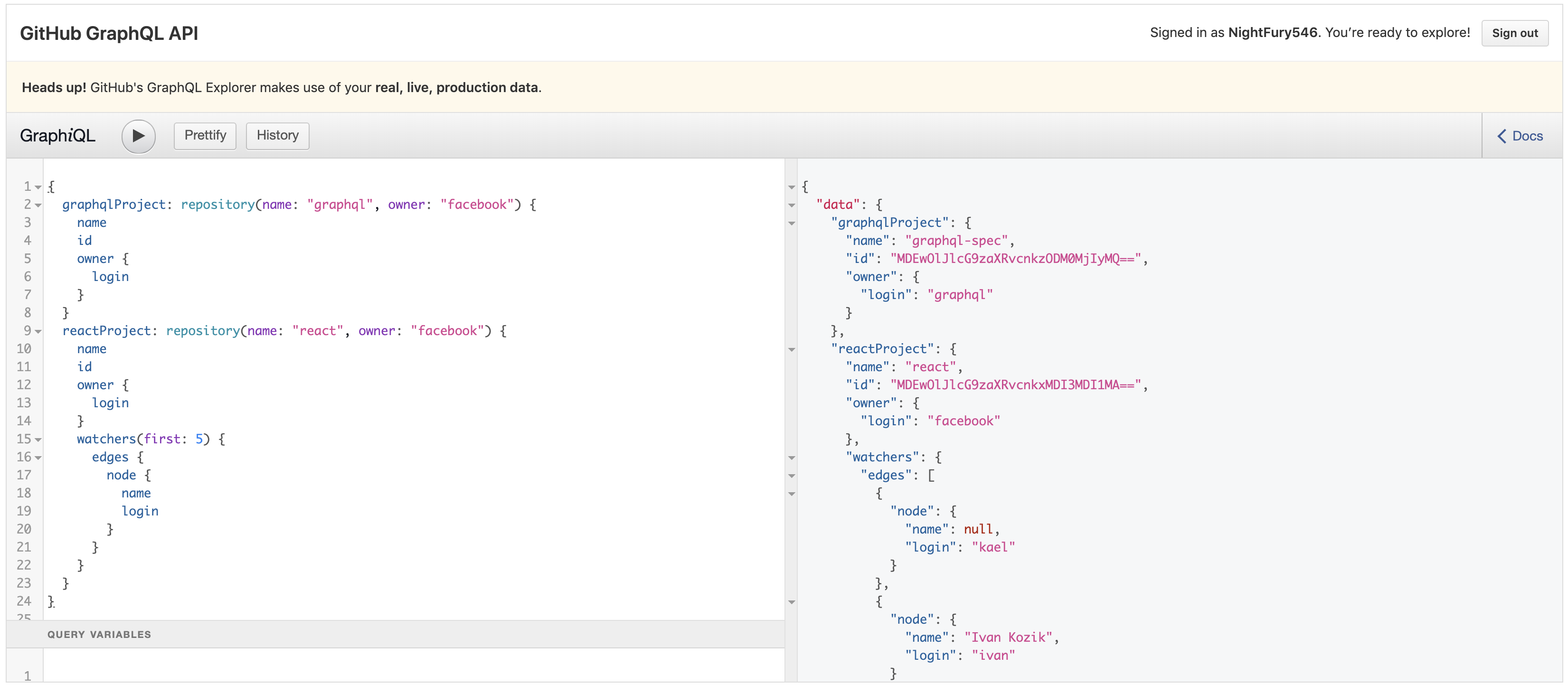
## Aliases

If we want to execute same query with different argument values or different set of fields. Then GraphQL will throw the below error.



we can resolve this kind of errors by aliasing specific names to each query.

See the below image, we assigned two different names to each query. And we are able to see the result.

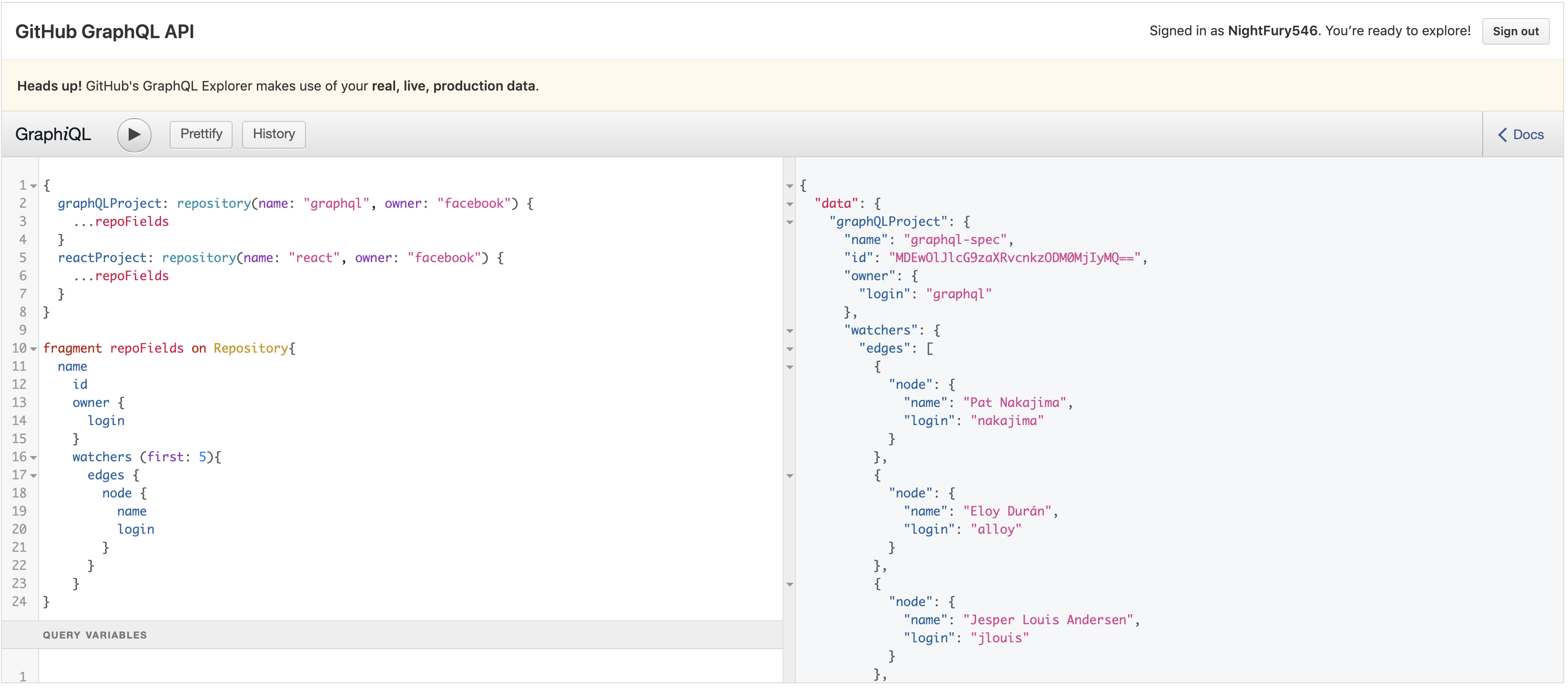


## Fragments

In the Above example, there is a lot of repetition of fields. In an alternative, we can create a fragment to query repeatedly.

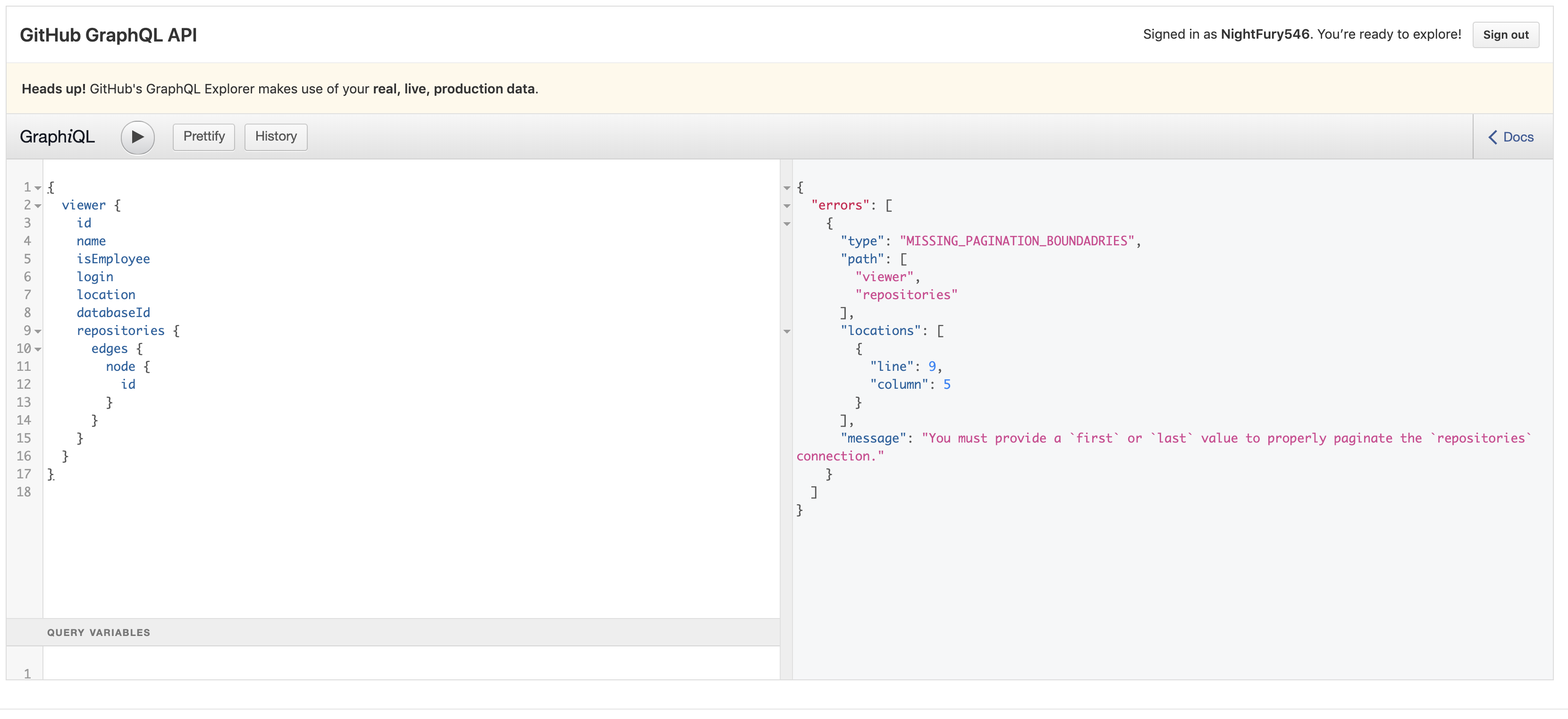
Fragments are reusable sets of fields that can be included in queries as needed.

Example:



Like this we can create a fragment and reuse it in queries using “…<fragment\_name>”

## Nested FIELDS, Paginaton and Connections



Look at the above example, Repositories fields has some nested fields such as id, name. to query those we need to mention the pagination and connections.

### Pagination:

Repositories field requires an argument such as (first, last) to query the results. This will act as a filter. We have a list of filters available.

If we mention first: 5, it will query the first 5 repositories available for the user.

If we mention last: 5, it will query the last 5 repositories available for the user.

### Connections:

We also need to mention the connection between the repositories and the nested fields.

Node

Edges

Repository

Repositories

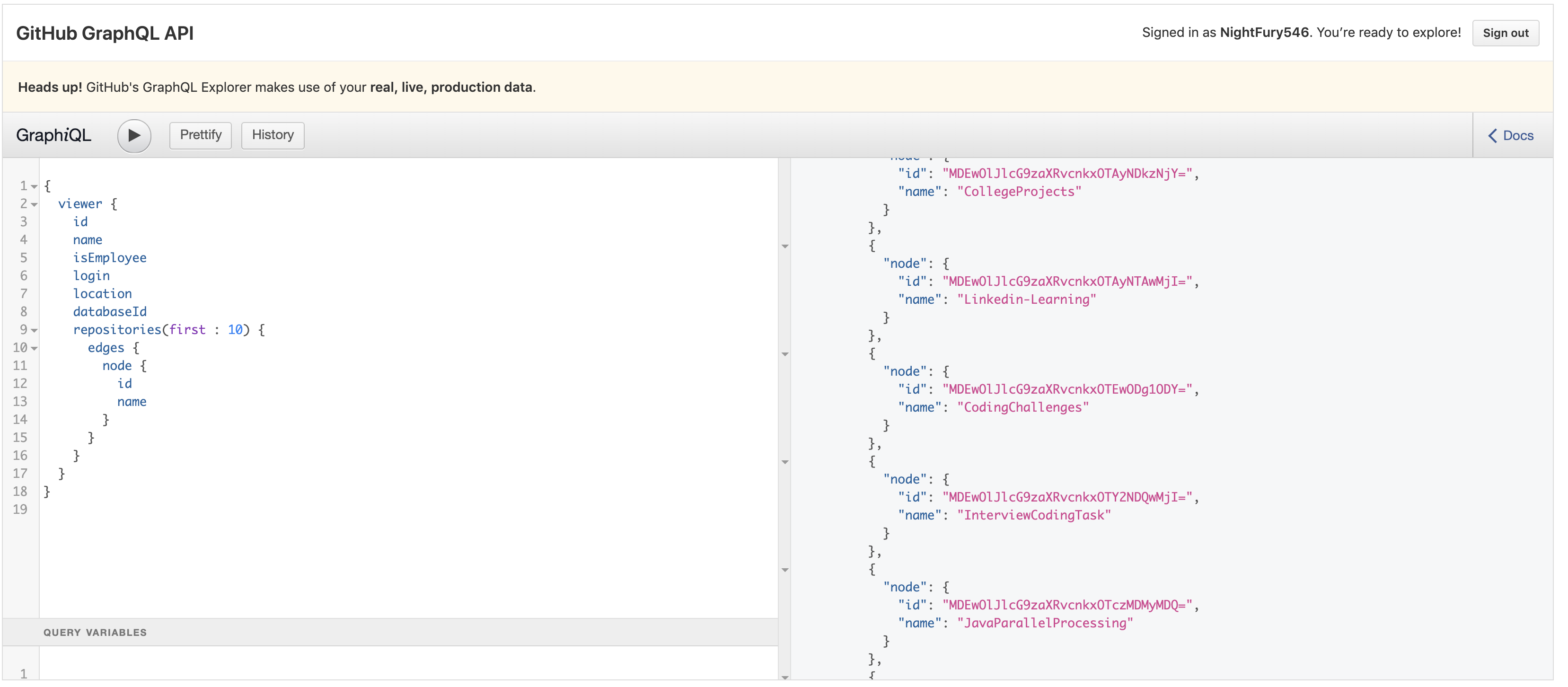
Node

Node

Repository

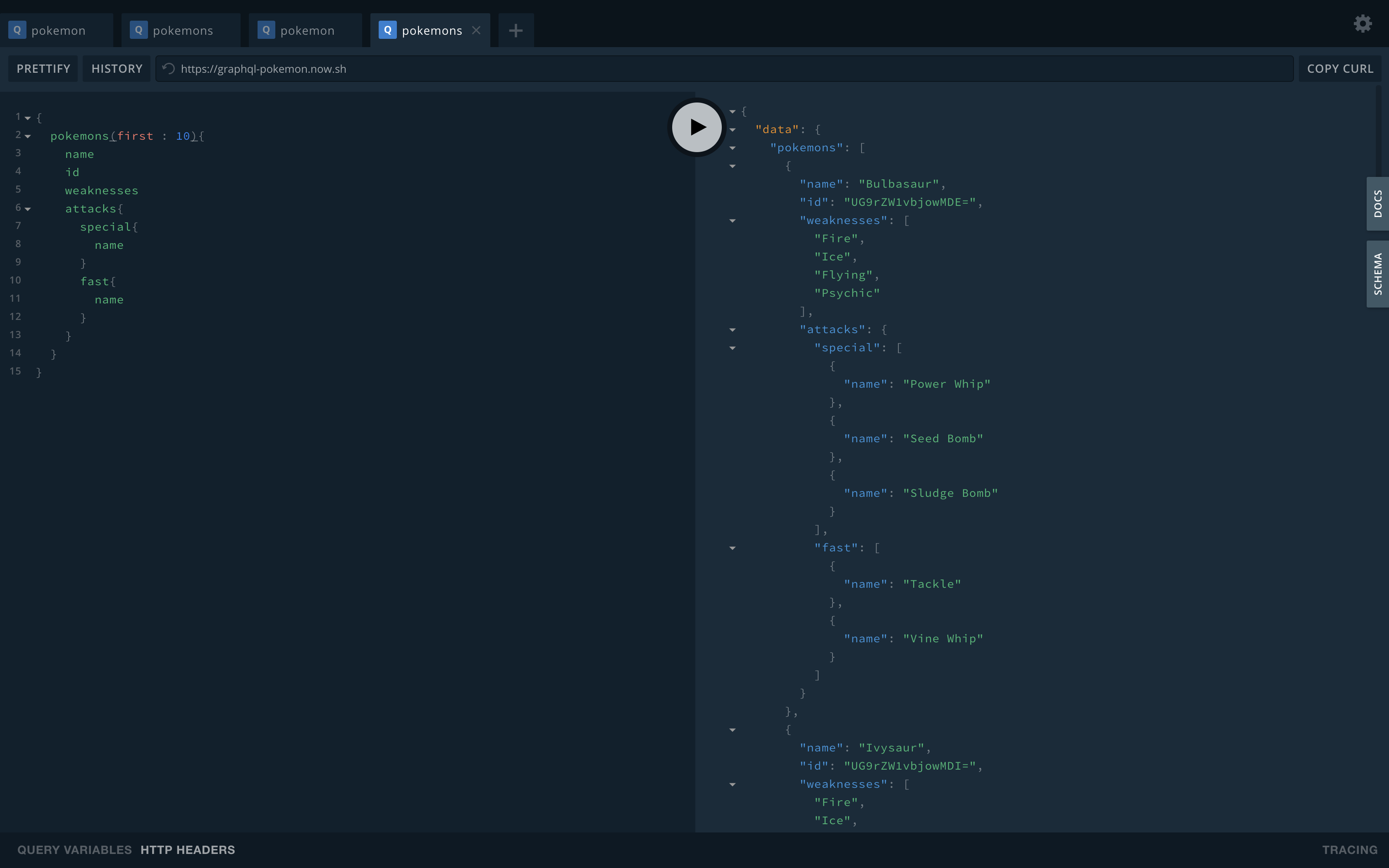
Repository

So, to query the repositories we need to mention the edges and its nodes.



# Operations and Variables

The queries can run without mentioning its names just like anonymous functions. But like below we can mention name the operations, variables.



## Operation names

Like this we can define the query/operation names and select the particular query to execute.



Here I used, Pokémon Database example from GraphQL playground desktop IDE.

## Variable names

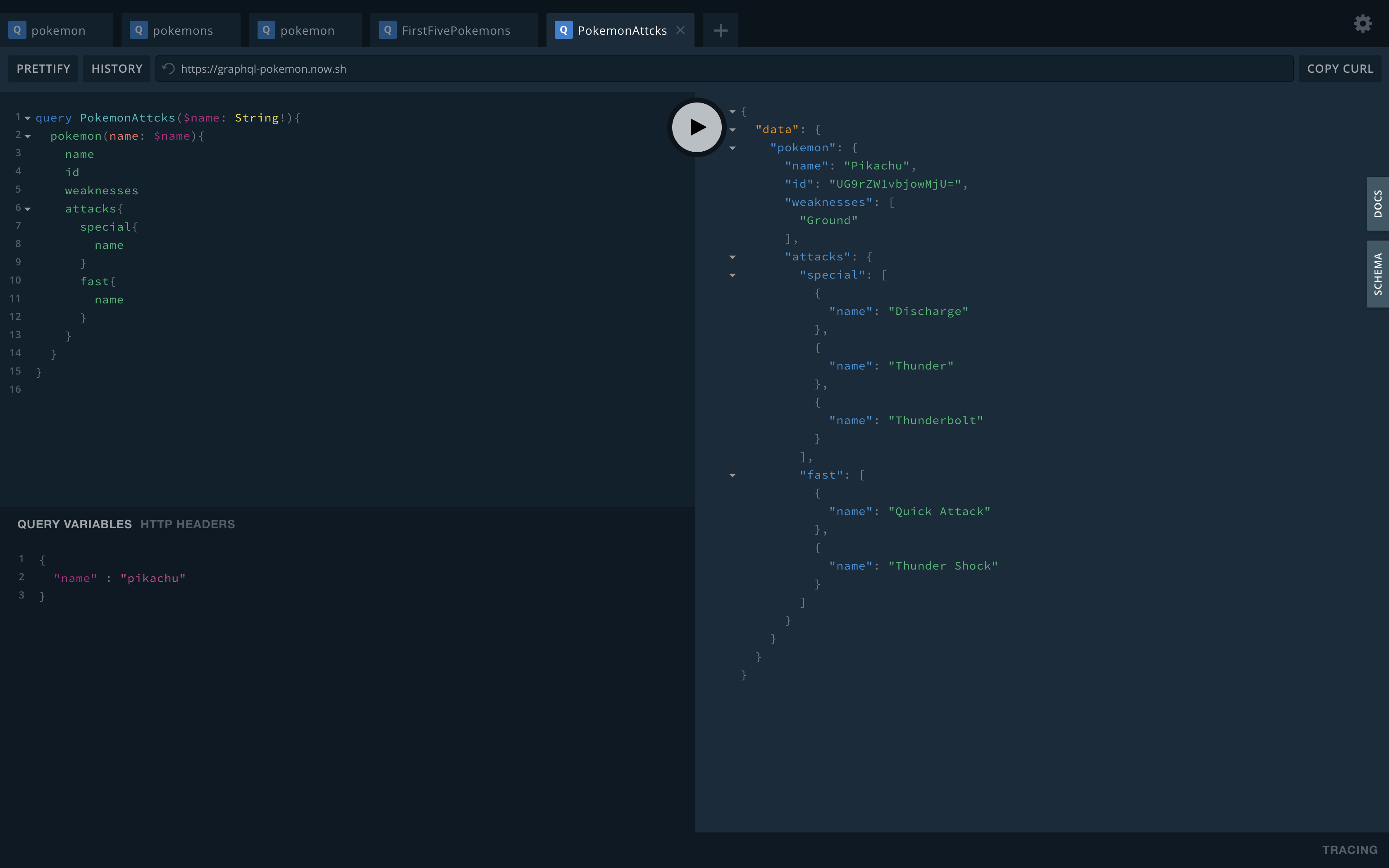
Here is the below example, we created a variable name and passed argument to the pokemon() query used that variable name.

Created $name variable

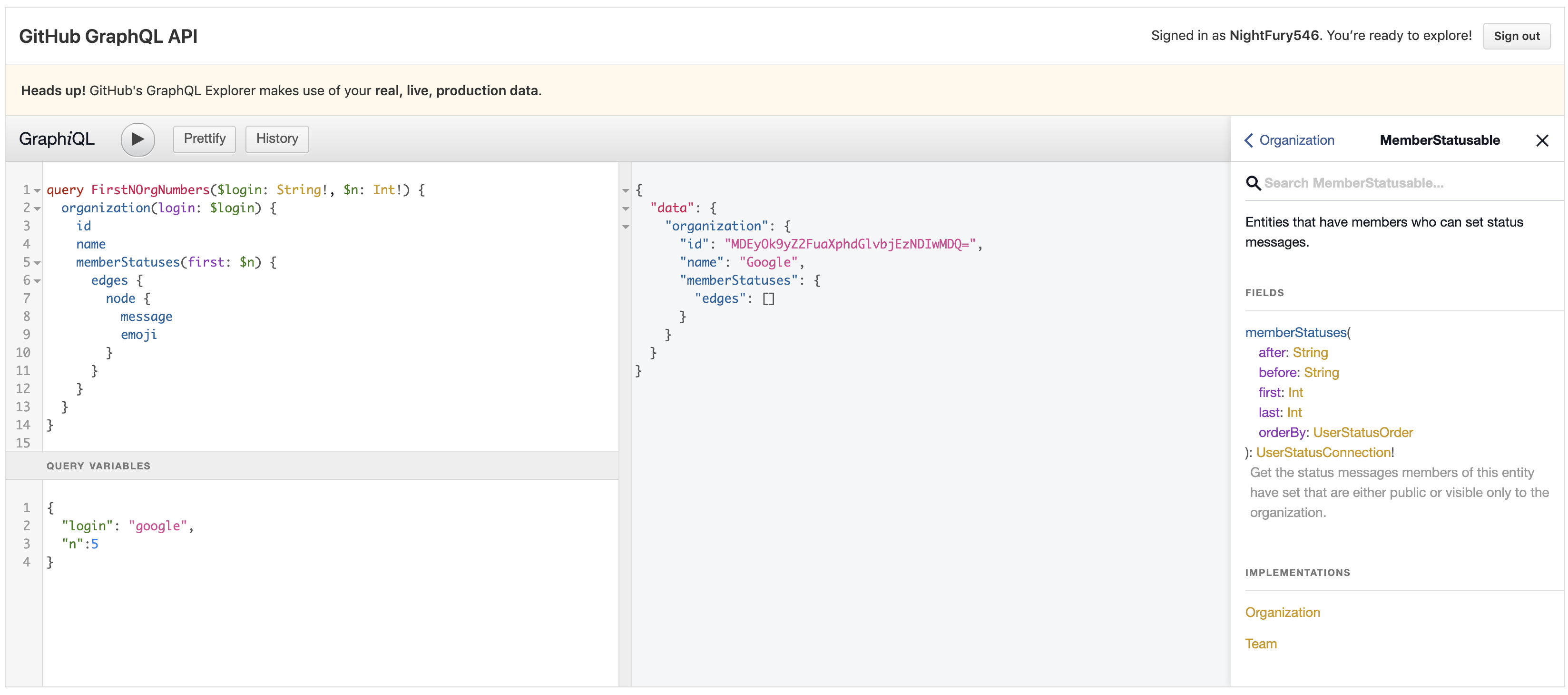
assigned “name”: “Pikachu” in Query variables

passed pokemon(name: $name)

So, the query pulled the information of pokemon name Pikachu from the database.



## Multiple variable definitions



Like above we can define and pass multiple variables to a query and use where ever it required.

## Mutation

Using Mutations, we can modify the data in the database.

* Mutations are similar to PUT or DELETE in REST.
* We send the data as a payload in a mutation.
* GraphQL changes the dataset behind the schema.
* API defines which Mutations are allowed.

If create a GraphQL server, we need to define the mutations.

Here are few Mutations allowed by GitHub

createProject(input: CreateProjectInput!): CreateProjectPayload

Creates a new project.

createPullRequest(input: CreatePullRequestInput!): CreatePullRequestPayload

Create a new pull request

createRef(input: CreateRefInput!): CreateRefPayload

Create a new Git Ref.

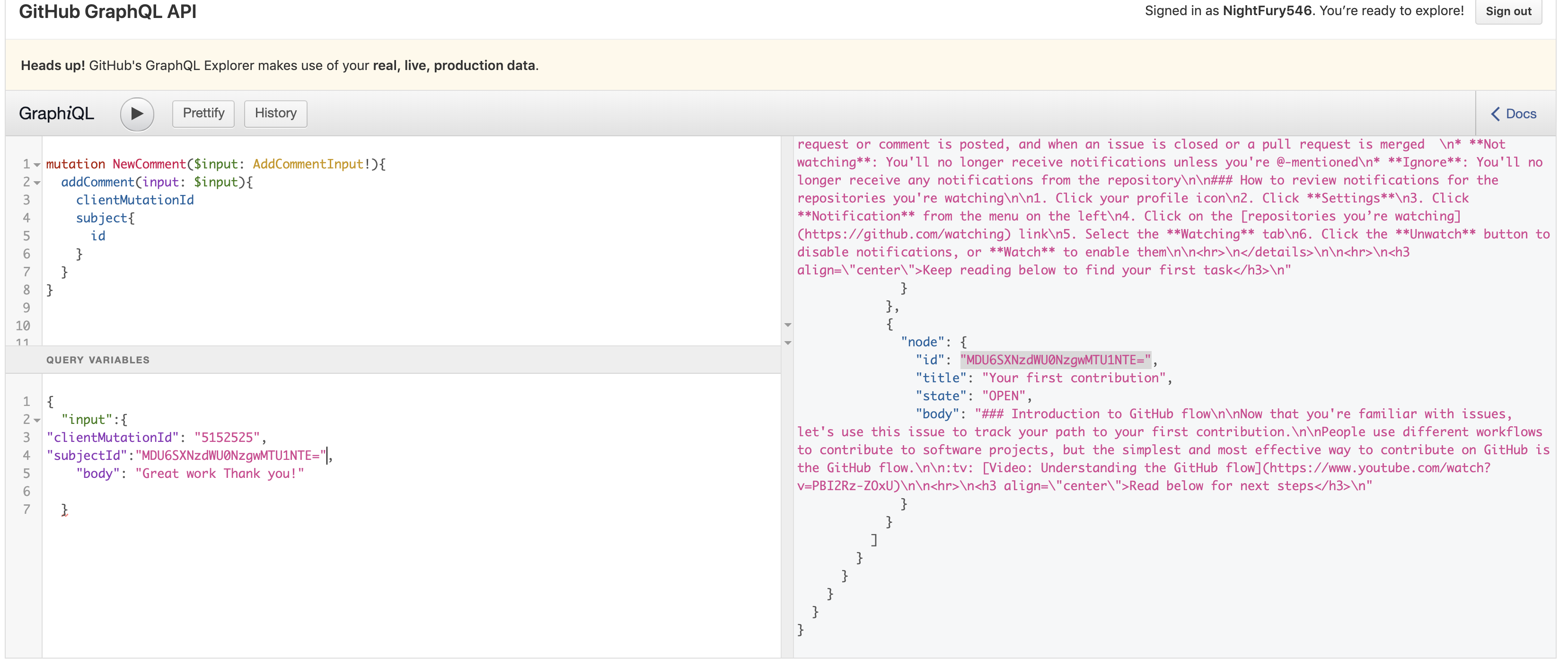
createRepository(input: CreateRepositoryInput!): CreateRepositoryPayload

Create a new repository.

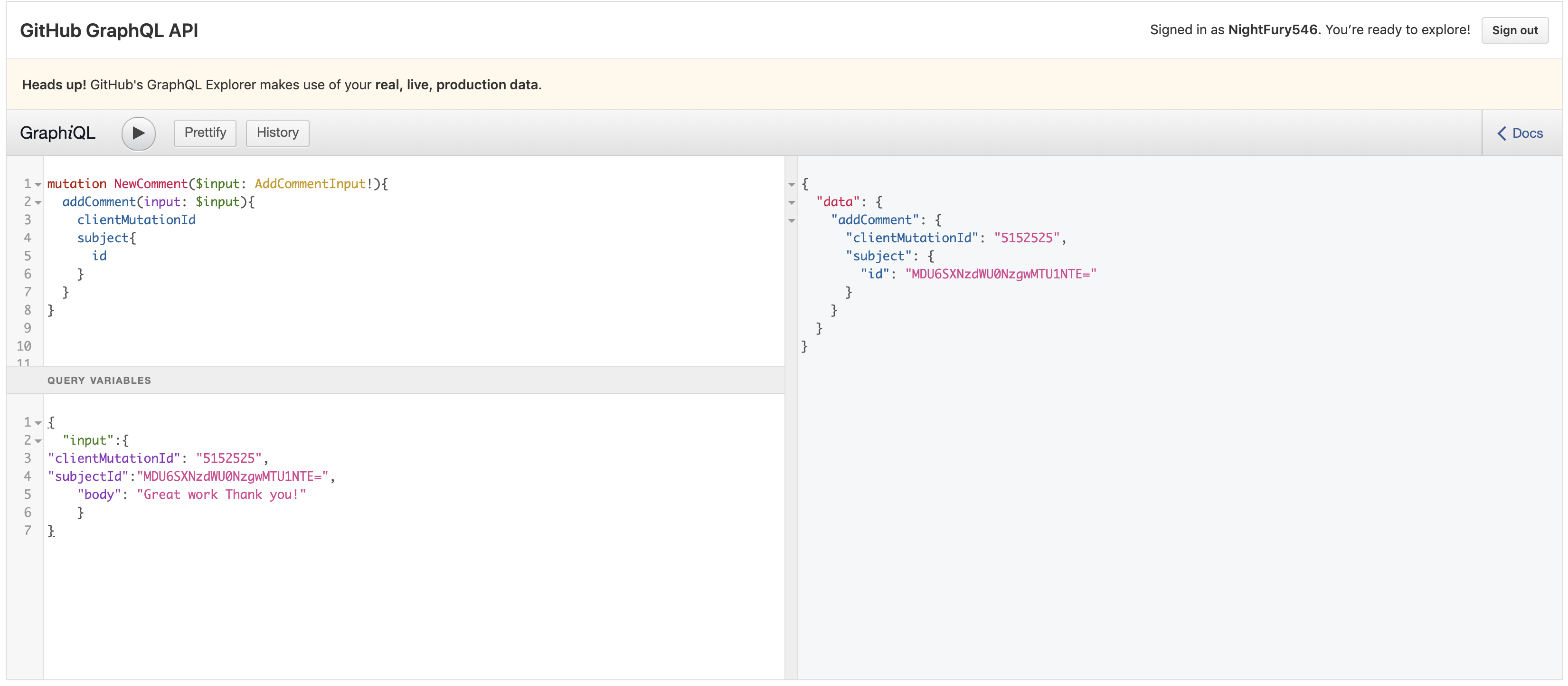
You can see the entire list of mutations and queries available in the GitHub in their documentation.

## Create mutations

The below example shows how to create mutations.

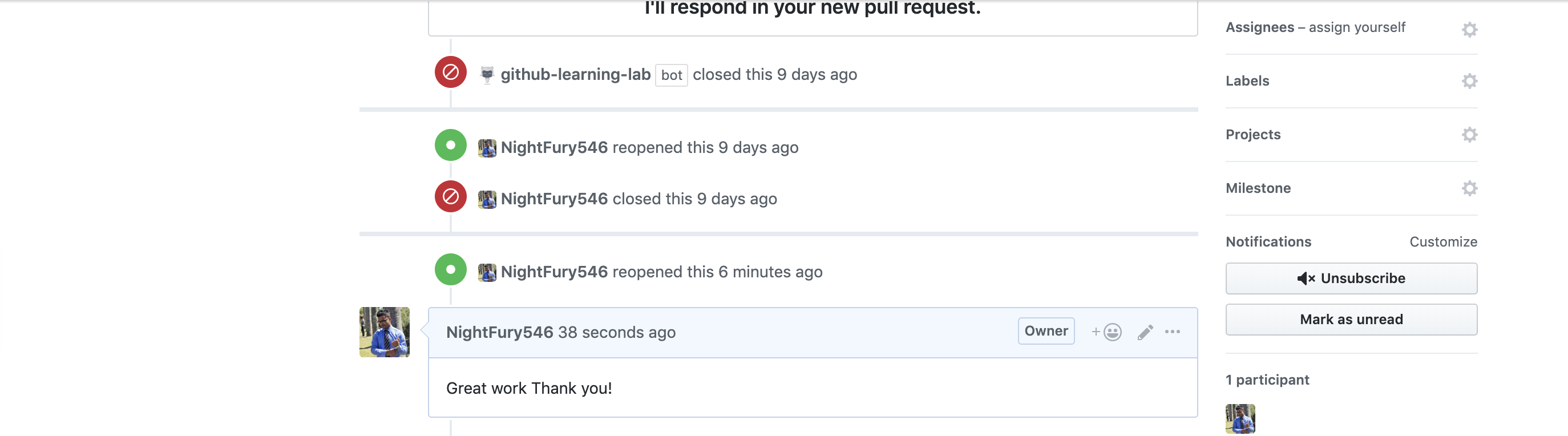


we passed the comment body to the JSON. And it will add that comment to the issue with given subject ID.



See the response here.

Below the newly added comment.

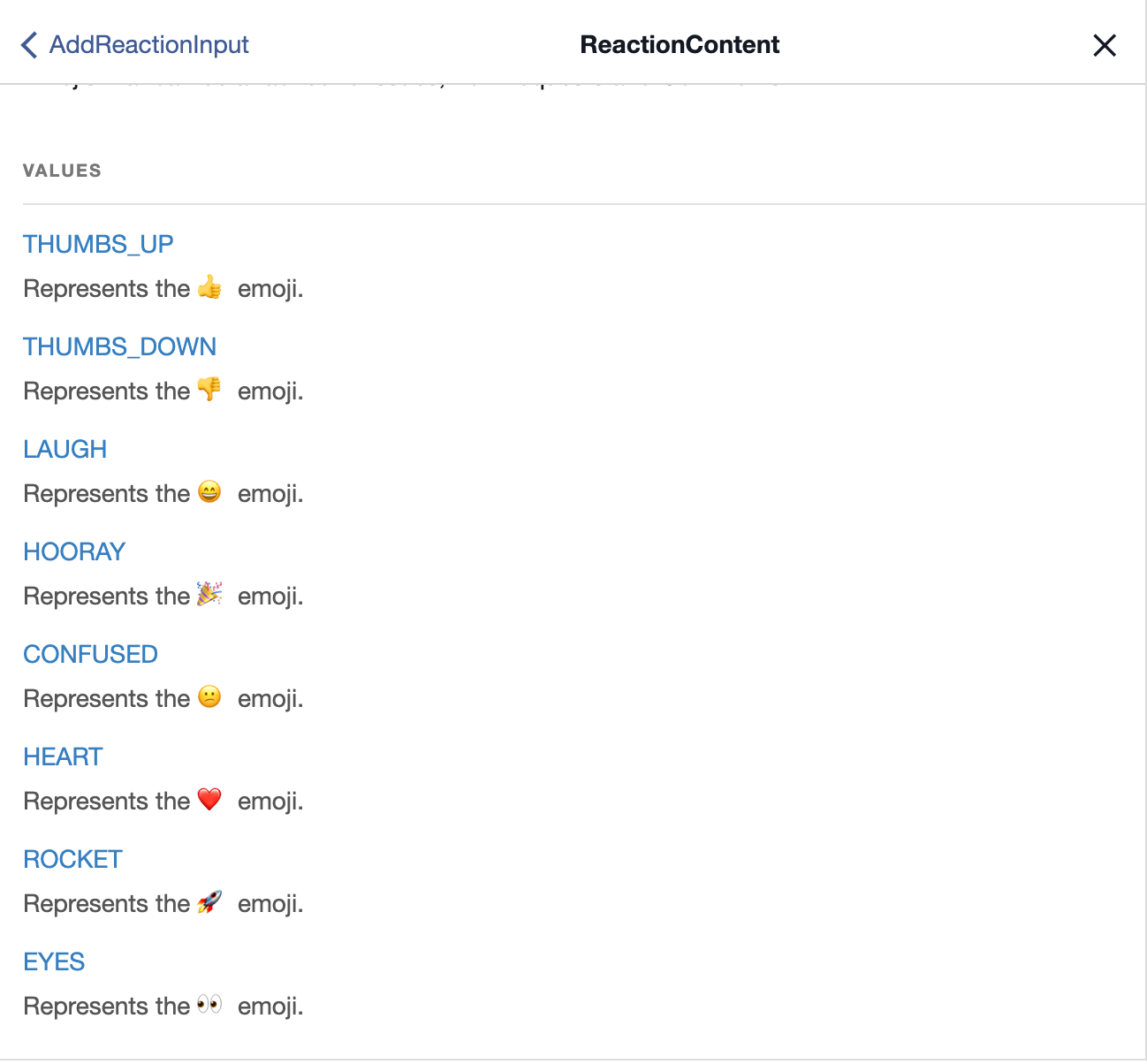


## Add reaction to mutation

We can also add reactions to the mutations.

We have a list of mutations available in the GitHub.

Those are



# additional References

If you want to learn more about GraphQL further with various combinations of front-end and Back-end Technologies, you can refer the below link

[howtographql.com](https://www.howtographql.com/)