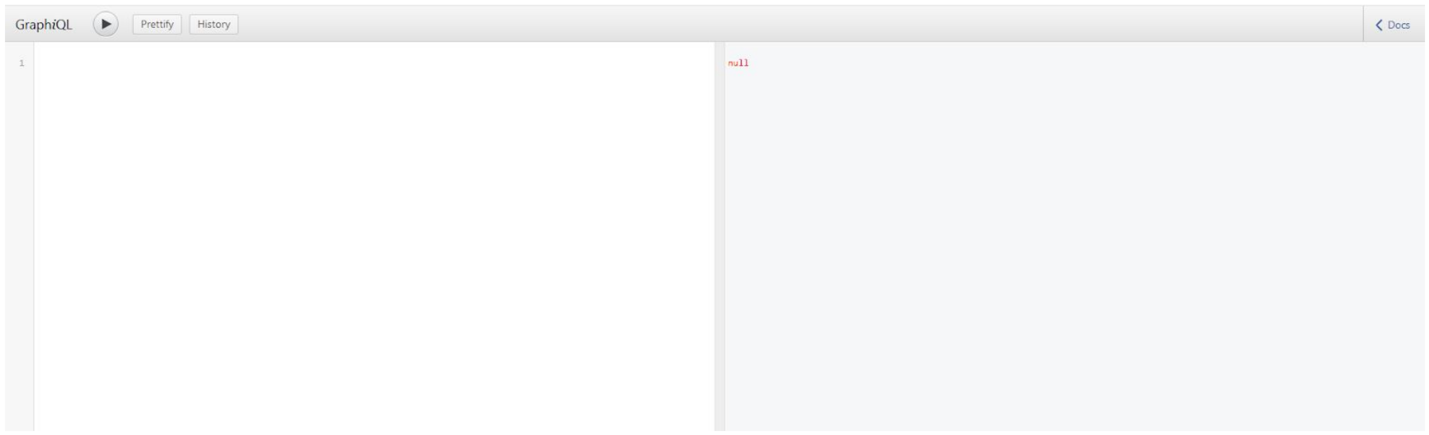
ATTACK DEFENSE

by PentesterAcademy

Name	GraphQL Basics II
URL	https://attackdefense.com/challengedetails?cid=1992
Type	REST: GraphQL

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

When the lab is launched, the GraphiQL console is provided.



```
null
```

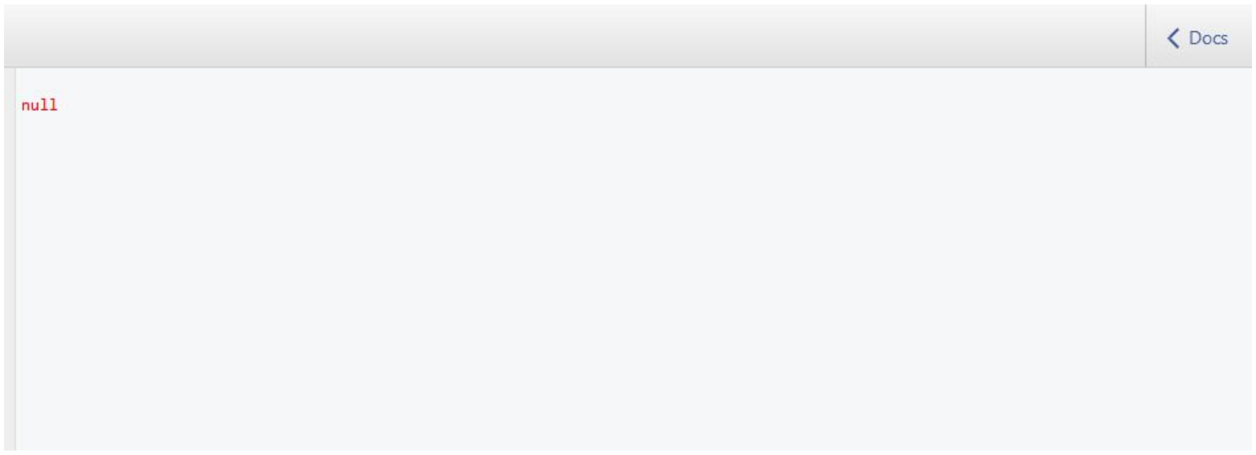
This console could be used to write the GraphQL queries.

Step 1: Exploring the GraphQL schema.

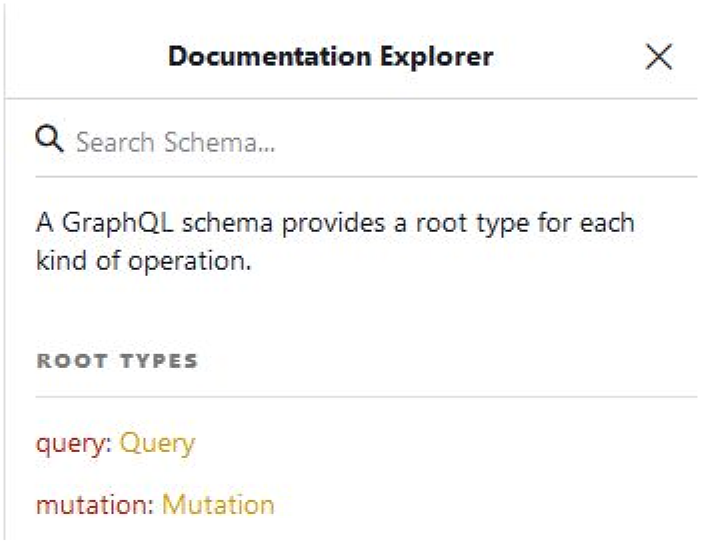
As it is mentioned in the challenge description that the introspection queries are not disabled. That means that the GraphQL schema could be accessed from the console and auto-completion would also work.

1. Exploring the GraphQL Schema:

Click on the Docs link on the top right of the console window.



```
null
```



A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation

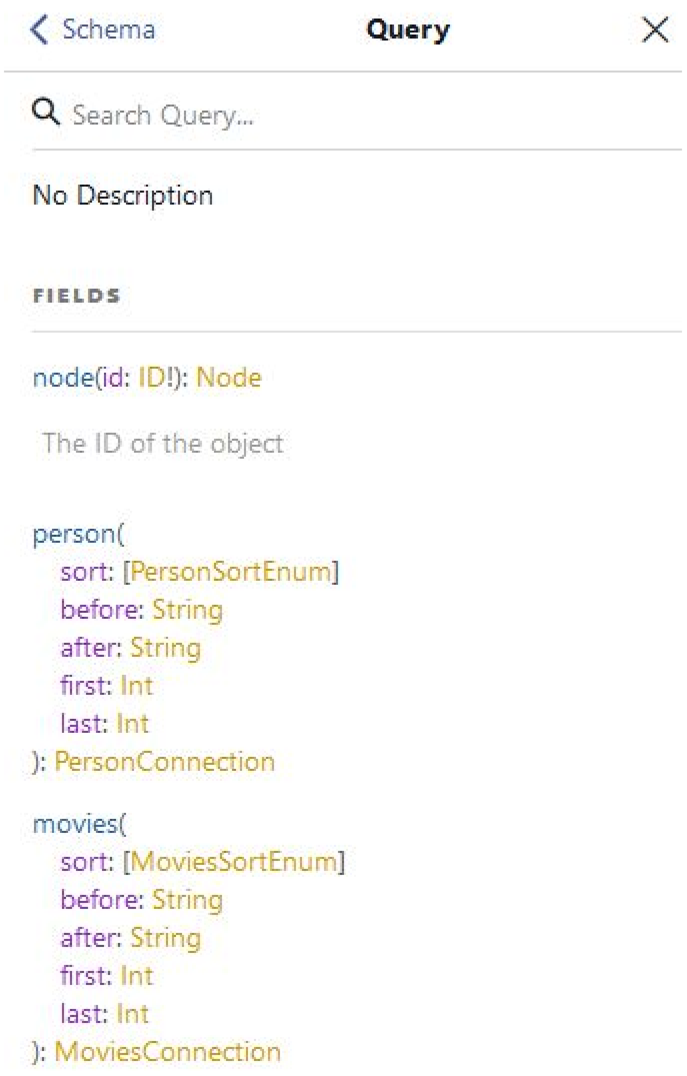
The Documentation Explorer panel could be used to view the GraphQL schema.

Notice that we have Query and Mutation types.

1. Query Type: Used to make queries to get data from the server.
2. Mutations: Used to make changes to server-side data.

Click on Queries.

That leads to the following page:


< Schema Query ×

Search Query...

No Description

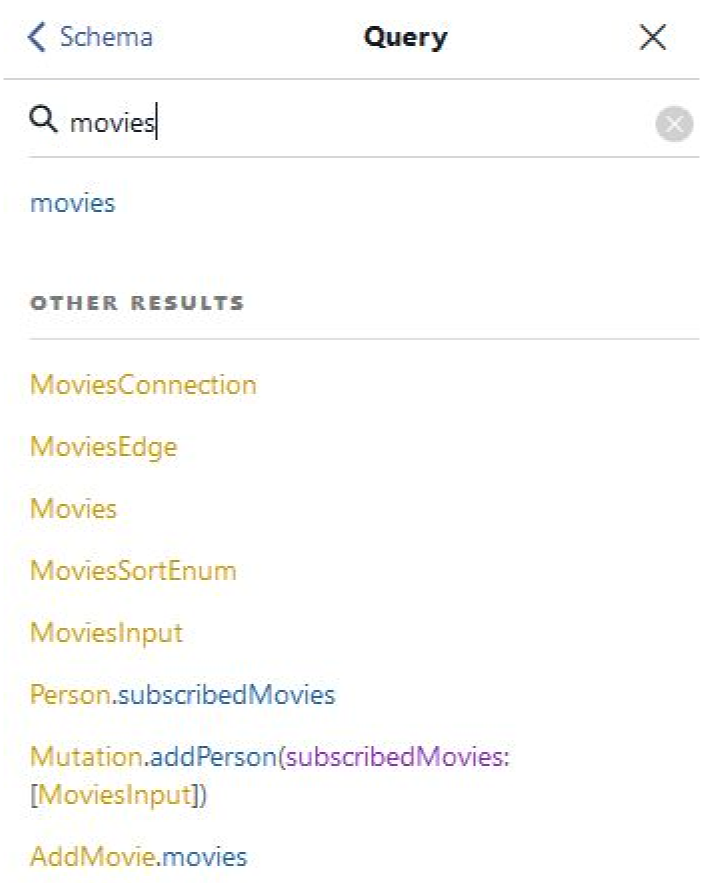
FIELDS

```
node(id: ID!): Node

 The ID of the object

person(
   sort: [PersonSortEnum]
   before: String
   after: String
   first: Int
   last: Int
): PersonConnection

movies(
   sort: [MoviesSortEnum]
   before: String
   after: String
   first: Int
   last: Int
): MoviesConnection
```

So, there are 3 fields: node, person and movies.

We can also make searches using the provided search bar.

Searching for “movies” keyword in the search bar:



< Schema Query ×

movies

movies

OTHER RESULTS

MoviesConnection

MoviesEdge

Movies

MoviesSortEnum

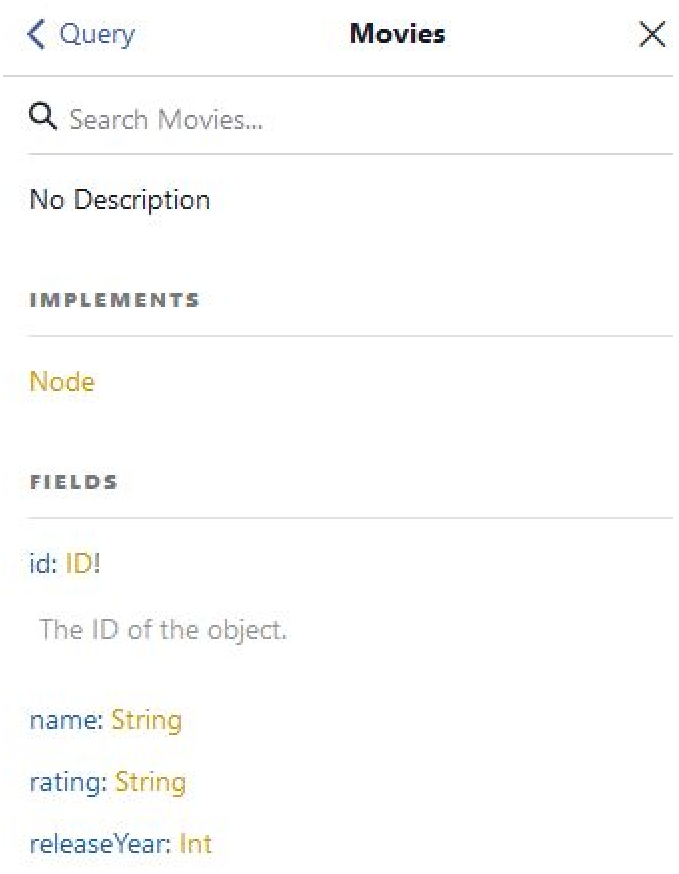
MoviesInput

Person.subscribedMovies

Mutation.addPerson(subscribedMovies: [MoviesInput])

AddMovie.movies

Select the 3rd result: Movies


< Query Movies ×

Search Movies...

No Description

IMPLEMENTS

Node

FIELDS

id: ID!

The ID of the object.

name: String

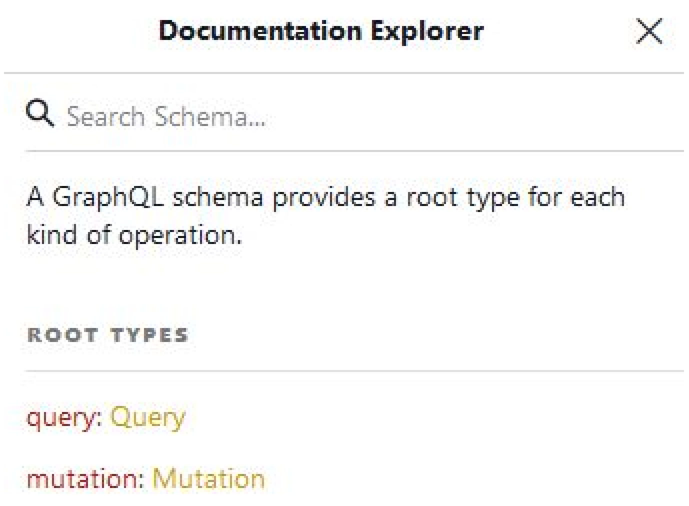
rating: String

releaseYear: Int

That shows the Movies object. It contains the following fields:

id, name, rating, releaseYear

Go back to the starting page of the explorer and explore the mutations:


Documentation Explorer

Search Schema...

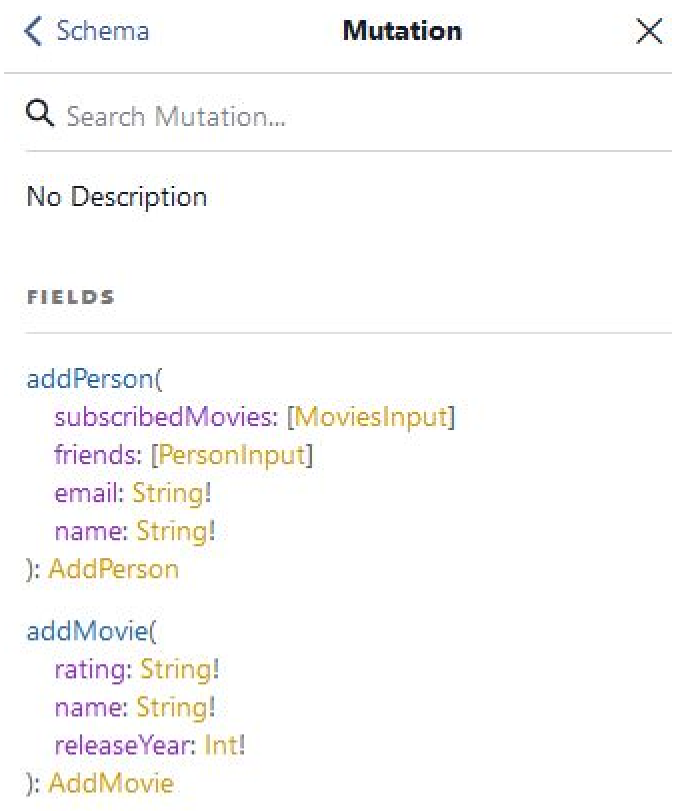
A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation

Click on Mutations.


Schema

Mutation

Search Mutation...

No Description

FIELDS

```
addPerson(
  subscribedMovies: [MoviesInput]
  friends: [PersonInput]
  email: String!
  name: String!
): AddPerson

addMovie(
  rating: String!
  name: String!
  releaseYear: Int!
): AddMovie
```

We have 2 mutations:

addPerson and addMovie to add a new Person and a new movie to the database, respectively.

Step 2: Making queries to the backend database.

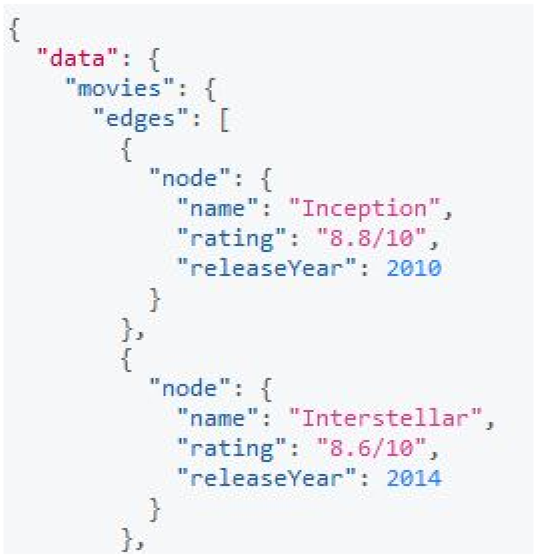
Writing queries in the console window to fetch the desired results from the backend database:

1. Use the following query to fetch all the movie objects and display the name, rating, and releaseYear fields.

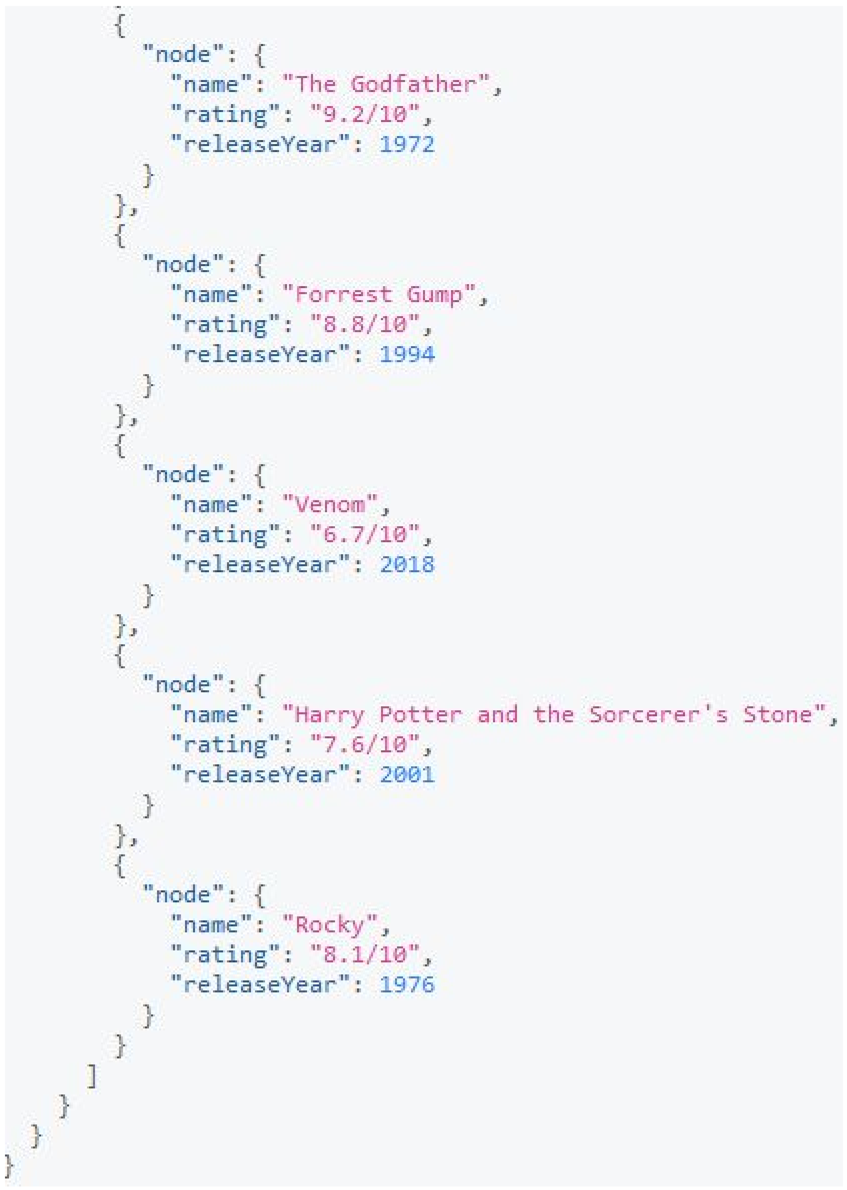
Query:

```
{
  movies {
          edges {
          node {
          name
          rating
          releaseYear
          }
          }
  }
}
```

Response:



```
{
  "data": {
    "movies": {
      "edges": [
        {
          "node": {
            "name": "Inception",
            "rating": "8.8/10",
            "releaseYear": 2010
          }
        },
        {
          "node": {
            "name": "Interstellar",
            "rating": "8.6/10",
            "releaseYear": 2014
          }
        },
```



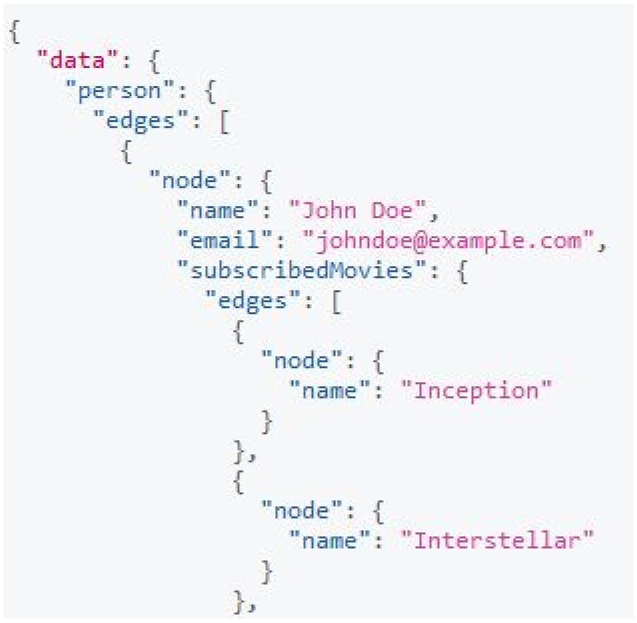
```
        {
          "node": {
            "name": "The Godfather",
            "rating": "9.2/10",
            "releaseYear": 1972
          }
        },
        {
          "node": {
            "name": "Forrest Gump",
            "rating": "8.8/10",
            "releaseYear": 1994
          }
        },
        {
          "node": {
            "name": "Venom",
            "rating": "6.7/10",
            "releaseYear": 2018
          }
        },
        {
          "node": {
            "name": "Harry Potter and the Sorcerer's Stone",
            "rating": "7.6/10",
            "releaseYear": 2001
          }
        },
        {
          "node": {
            "name": "Rocky",
            "rating": "8.1/10",
            "releaseYear": 1976
          }
        }
      ]
    }
  }
}
```

2. Use the following query to fetch all the person objects and display their name, email, name of subscribed movies and the name of their friends.

Query:

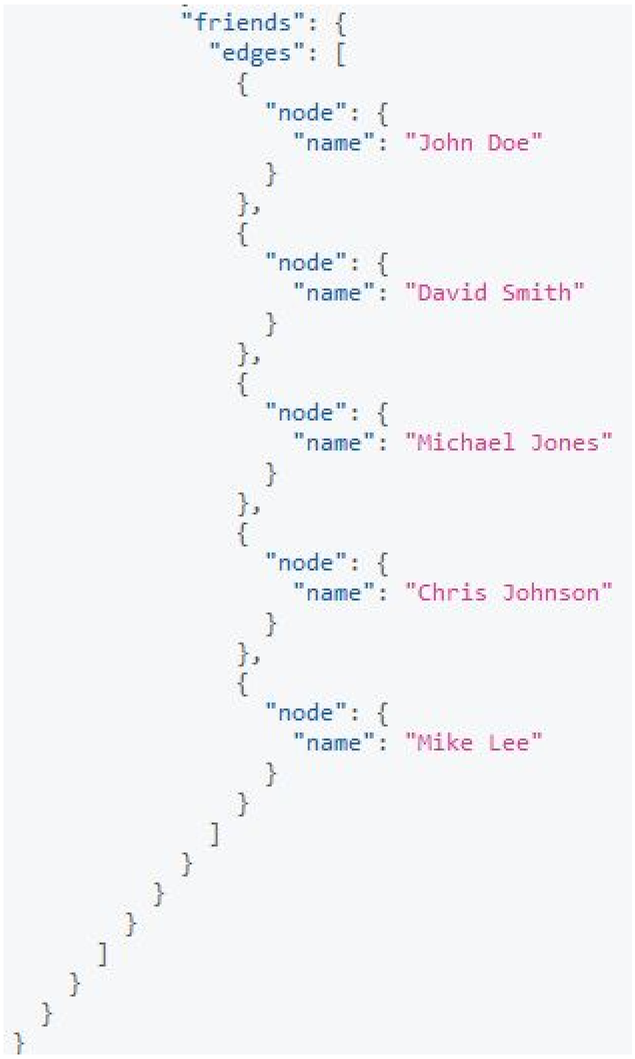
```
{
 person {
          edges {
          node {
          name
          email
          subscribedMovies {
          edges {
          node {
          name
          }
          }
          }
          friends {
          edges {
          node {
          name
          }
          }
          }
          }
          }
 }
}
```

Response:



```
{
  "data": {
    "person": {
      "edges": [
        {
          "node": {
            "name": "John Doe",
            "email": "johndoe@example.com",
            "subscribedMovies": {
              "edges": [
                {
                  "node": {
                    "name": "Inception"
                  }
                },
                {
                  "node": {
                    "name": "Interstellar"
                  }
                },
```

…



```
            "friends": {
              "edges": [
                {
                  "node": {
                    "name": "John Doe"
                  }
                },
                {
                  "node": {
                    "name": "David Smith"
                  }
                },
                {
                  "node": {
                    "name": "Michael Jones"
                  }
                },
                {
                  "node": {
                    "name": "Chris Johnson"
                  }
                },
                {
                  "node": {
                    "name": "Mike Lee"
                  }
                }
              ]
            }
          }
        }
      ]
    }
  }
}
```

Note: While typing the queries, the editor would provide suggestions (auto-completion) since the Introspection Queries are not disabled (in default settings).

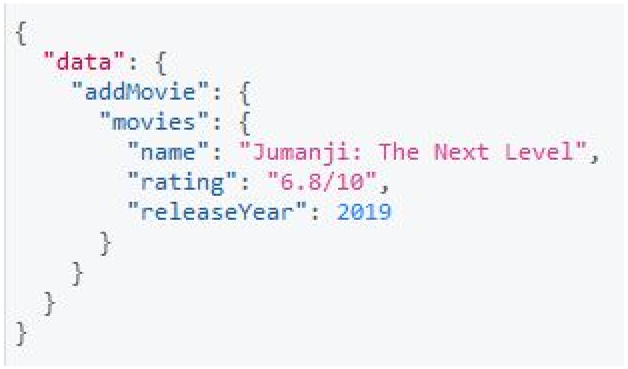
Step 3: Modifying the backend database using mutations.

1. Use the following mutation to add a new movie to the backend database:

Mutation:

```
mutation {
  addMovie(name: "Jumanji: The Next Level", rating: "6.8/10", releaseYear: 2019) {
        movies {
        name
        rating
        releaseYear
        }
  }
}
```

Response:


```
{
  "data": {
    "addMovie": {
      "movies": {
        "name": "Jumanji: The Next Level",
        "rating": "6.8/10",
        "releaseYear": 2019
      }
    }
  }
}
```

The movie got successfully added to the list of movies.

Notice that just like in queries, if the mutation field returns an object type, we can ask for nested fields. This can be useful for fetching the new state of an object without making multiple requests to the backend.

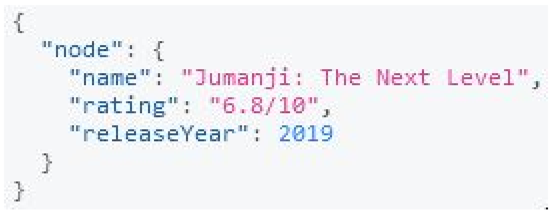
Making a query to fetch all the movies would confirm that the above movie was added to the database:

Query:

```
{
  movies {
            edges {
            node {
            name
            rating
            releaseYear
            }
            }
  }
}
```

Response:

The last record contains the newly added movie object:

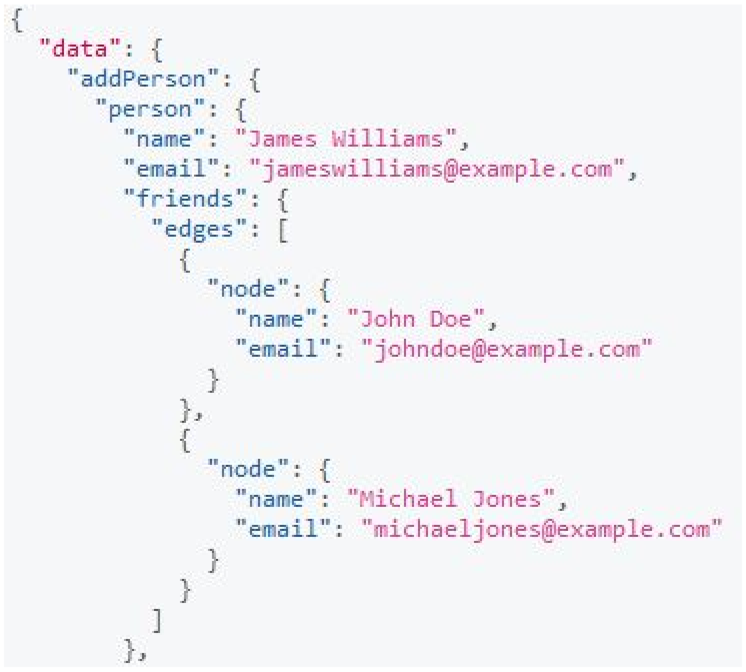

```
{
  "node": {
    "name": "Jumanji: The Next Level",
    "rating": "6.8/10",
    "releaseYear": 2019
  }
}
```

2. Use the following mutation to add a new Person to the backend database, providing a list of friends and a list of subscribed movies:

Mutation:

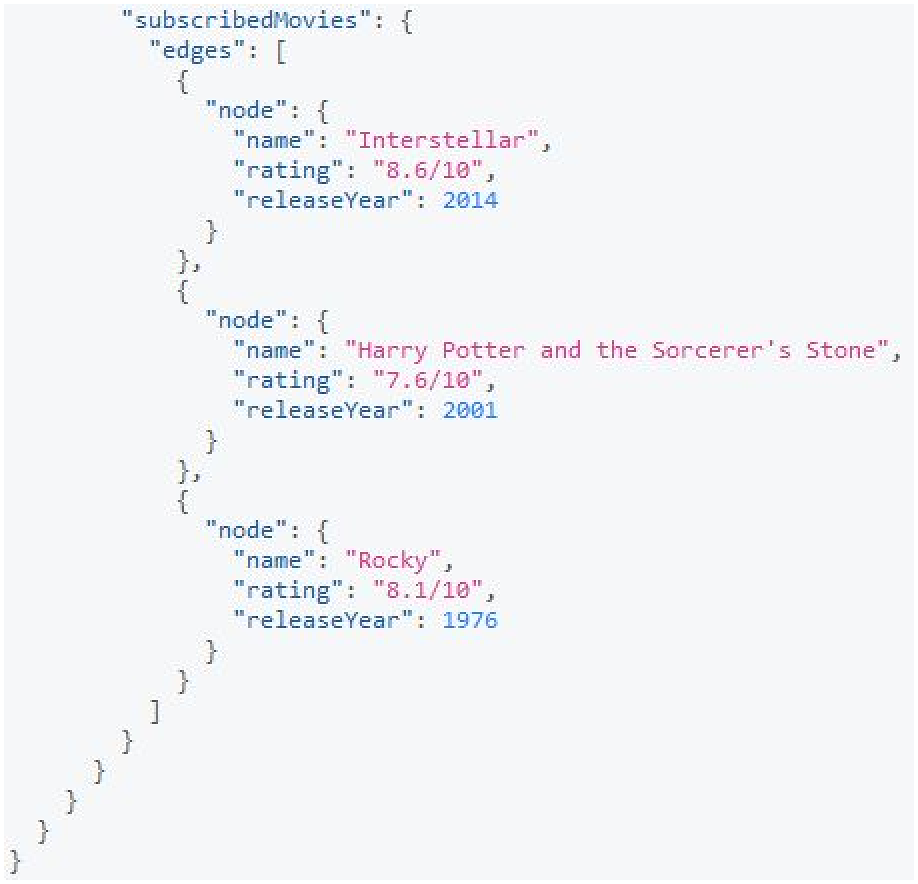
```
mutation {
 addPerson(
        name: "James Williams",
        email: "jameswilliams@example.com",
        friends: [
        {
        name:"John Doe"
        },
        {
        email: "michaeljones@example.com"
        }
        ],
        subscribedMovies: [
        {
        name: "Rocky"
        },
        {
        name: "Interstellar"
        },
        {
        name: "Harry Potter and the Sorcerer's Stone"
        }
        ]
 ) {
        person {
        name
        email
        friends {
        edges {
        node {
        name
        email
        }
        }
        }
        subscribedMovies {
        edges {
        node {
        name
        rating
        releaseYear
        }
        }
        }
        }
  }
}
```

Response:



```
{
  "data": {
    "addPerson": {
      "person": {
        "name": "James Williams",
        "email": "jameswilliams@example.com",
        "friends": {
          "edges": [
            {
              "node": {
                "name": "John Doe",
                "email": "johndoe@example.com"
              }
            },
            {
              "node": {
                "name": "Michael Jones",
                "email": "michaeljones@example.com"
              }
            }
          ]
        },
```

...



```
        "subscribedMovies": {
          "edges": [
            {
              "node": {
                "name": "Interstellar",
                "rating": "8.6/10",
                "releaseYear": 2014
              }
            },
            {
              "node": {
                "name": "Harry Potter and the Sorcerer's Stone",
                "rating": "7.6/10",
                "releaseYear": 2001
              }
            },
            {
              "node": {
                "name": "Rocky",
                "rating": "8.1/10",
                "releaseYear": 1976
              }
            }
          ]
        }
      }
    }
  }
}
```

A new Person object has been added to the database and its details are also displayed in the response.

Conclusion: In this lab we have dived deeper into GraphQL, mainly making different queries and adding objects using mutations.

References:

1. GraphQL (https://graphql.org)