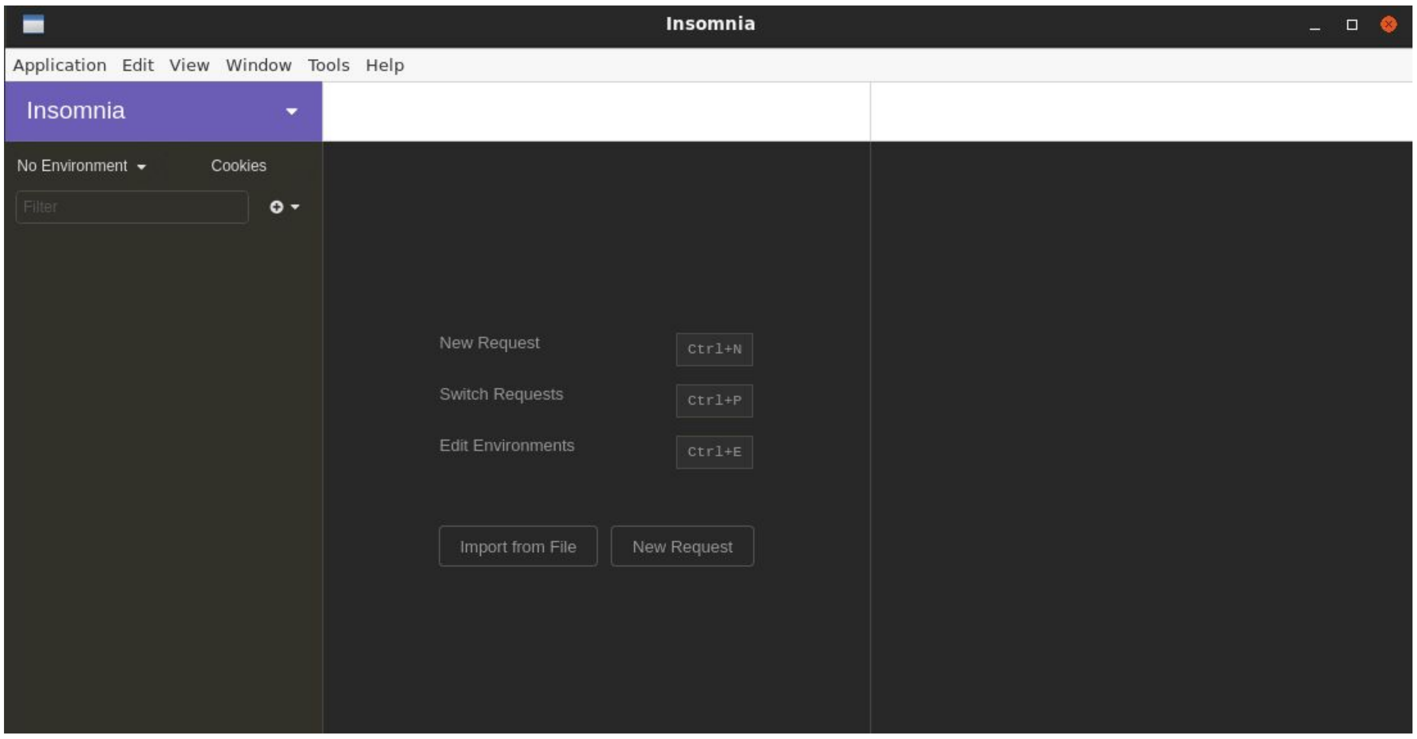
ATTACK DEFENSE

by PentesterAcademy

Name	GraphQL Basics IV
URL	https://attackdefense.com/challengedetails?cid=1993
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.

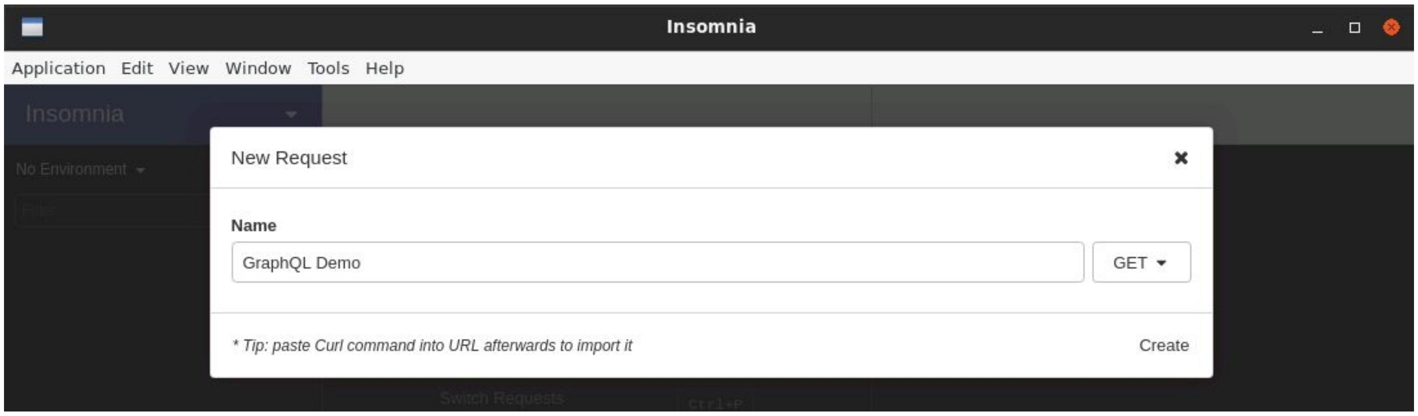
When the lab is launched, the Insomnia app is launched.



This application would be used to write the GraphQL queries in this lab.

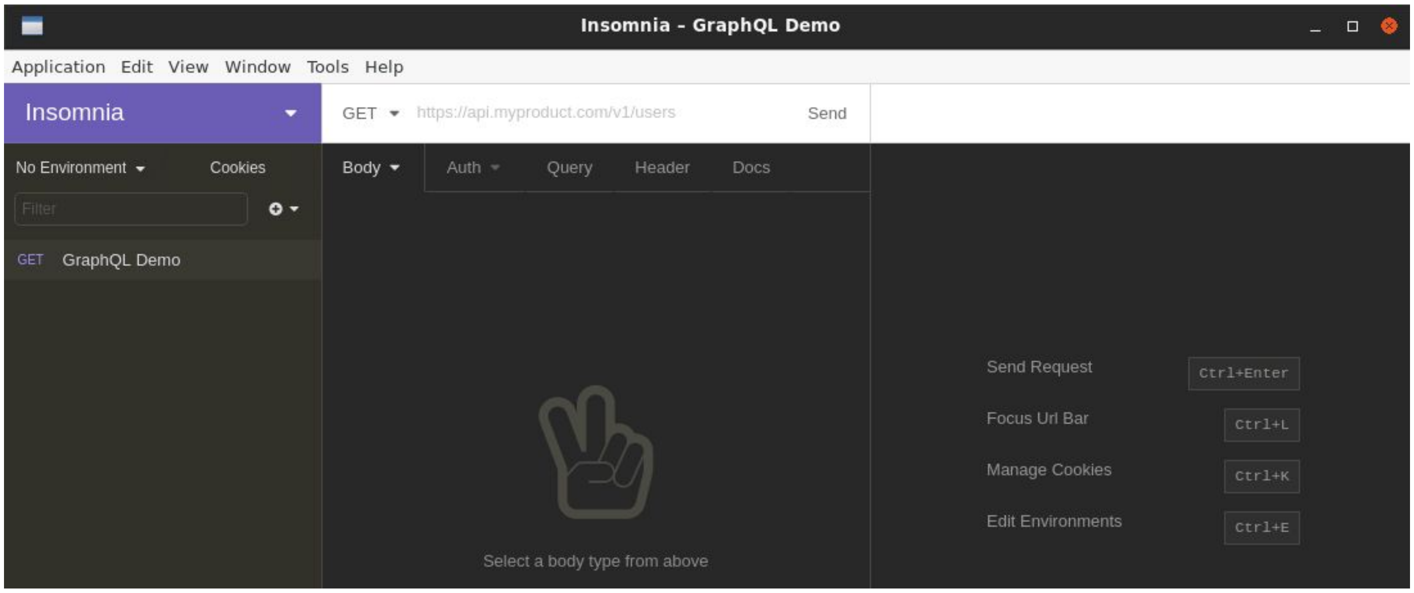
Step 1: Create a New Request.

Click on the New Request button:


Insomnia
Application Edit View Window Tools Help
New Request
Name
GraphQL Demo
GET
* Tip: paste Curl command into URL afterwards to import it
Create

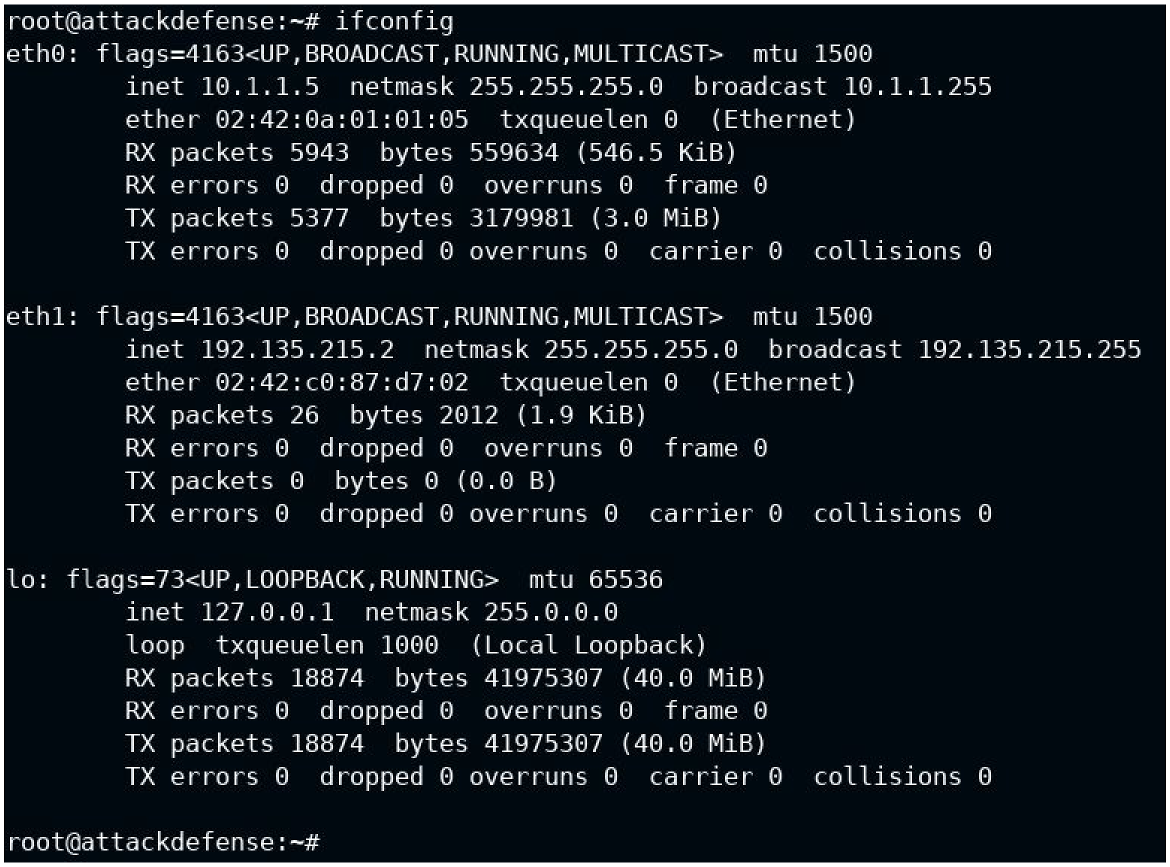
Provide a name to the New Request, eg: GraphQL Demo and then click on Create.

That would lead to the following view:



Step 2: Determine the IP address of the target server running GraphQL.

Command: ifconfig



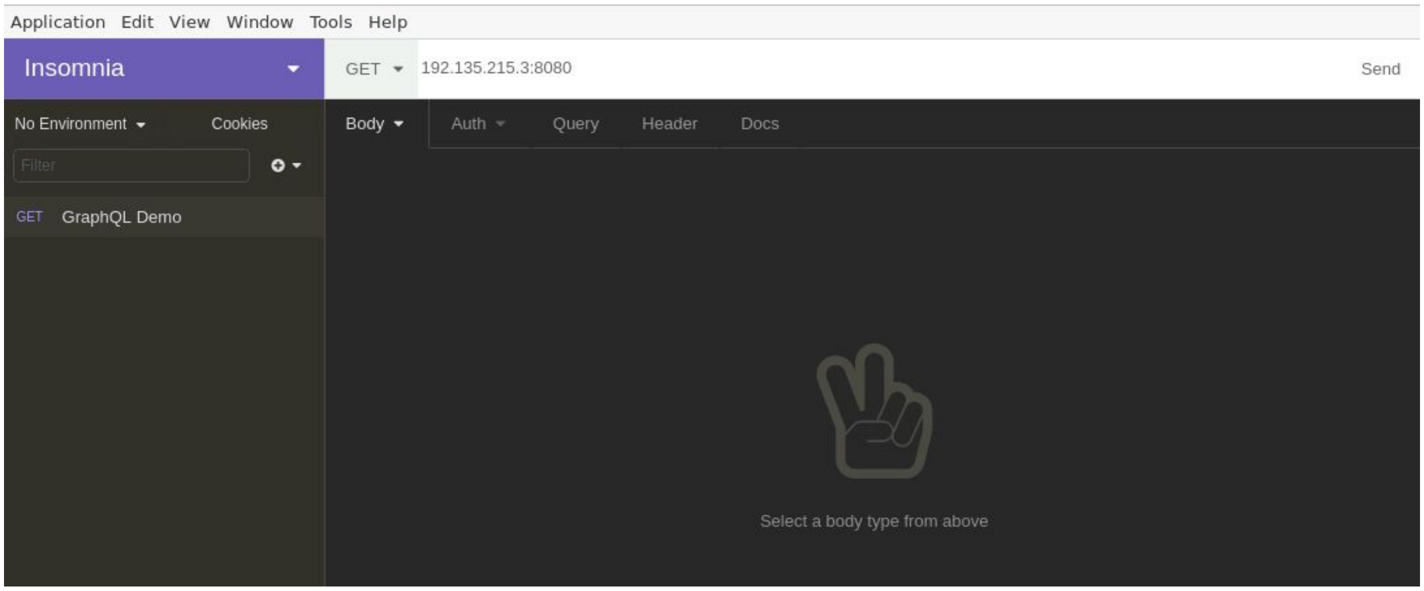
```
root@attackdefense:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.5  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:05  txqueuelen 0  (Ethernet)
        RX packets 5943  bytes 559634 (546.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5377  bytes 3179981 (3.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.135.215.2  netmask 255.255.255.0  broadcast 192.135.215.255
        ether 02:42:c0:87:d7:02  txqueuelen 0  (Ethernet)
        RX packets 26  bytes 2012 (1.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 18874  bytes 41975307 (40.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18874  bytes 41975307 (40.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.135.215.2.

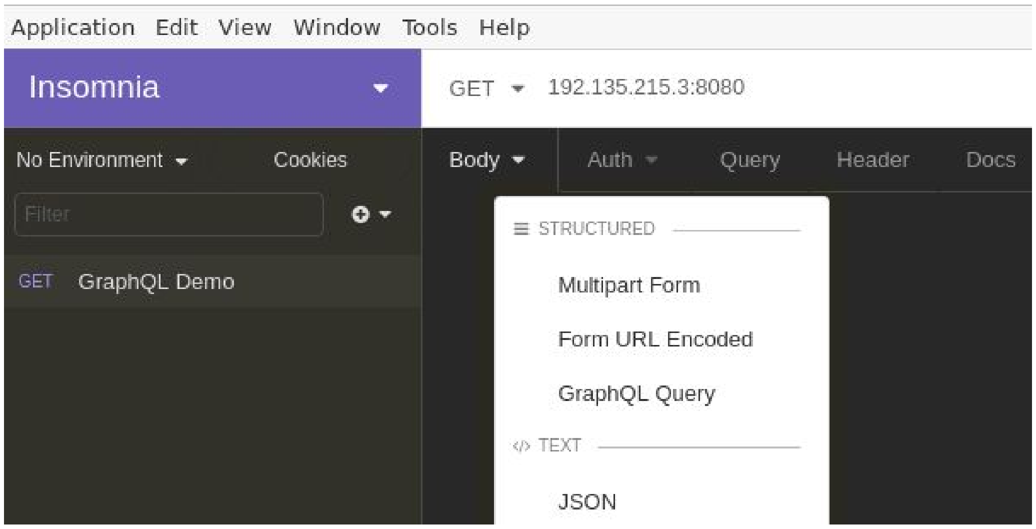
So the target GraphQL endpoint must be located at 192.135.215.3 on port 8080.

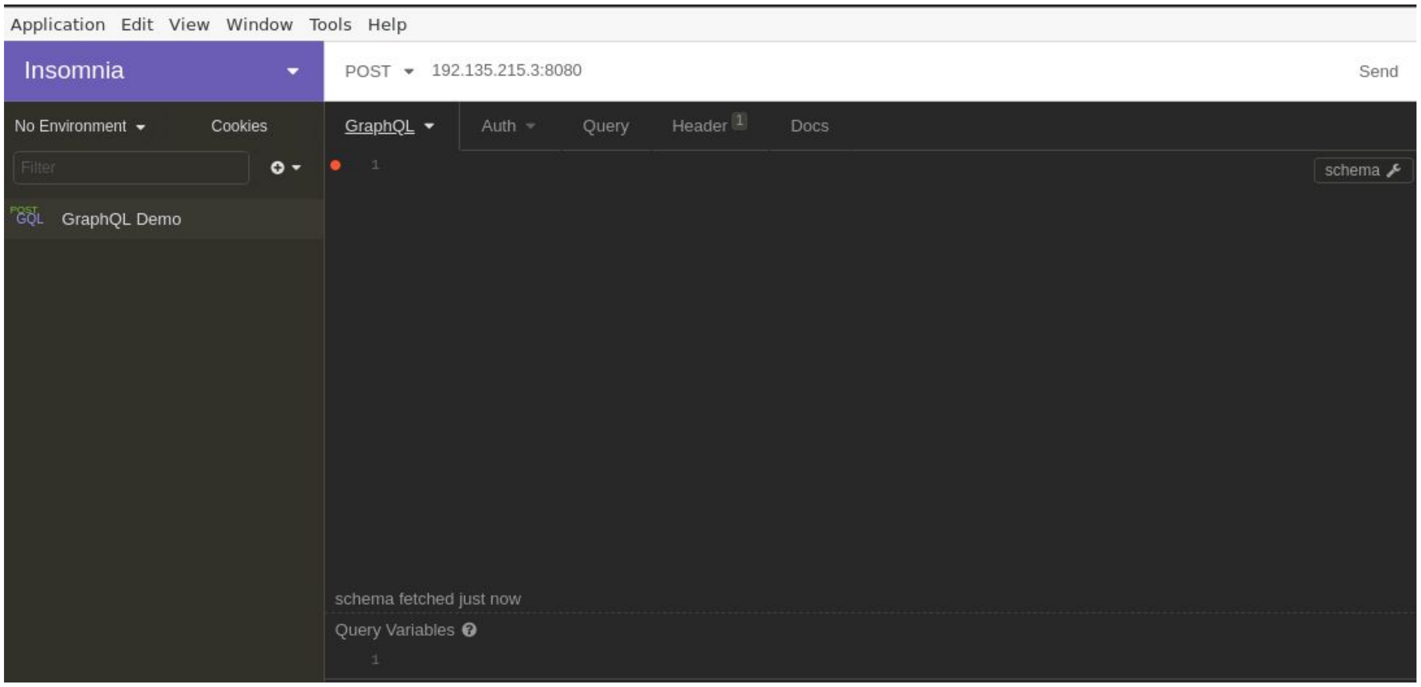
Step 3: Interacting with the GraphQL endpoint.

Now, since the IP address of the GraphQL endpoint is known, specify the IP address and the port number in the top bar:



Click on the Body tab and select the option "GraphQL Query" from the dropdown.

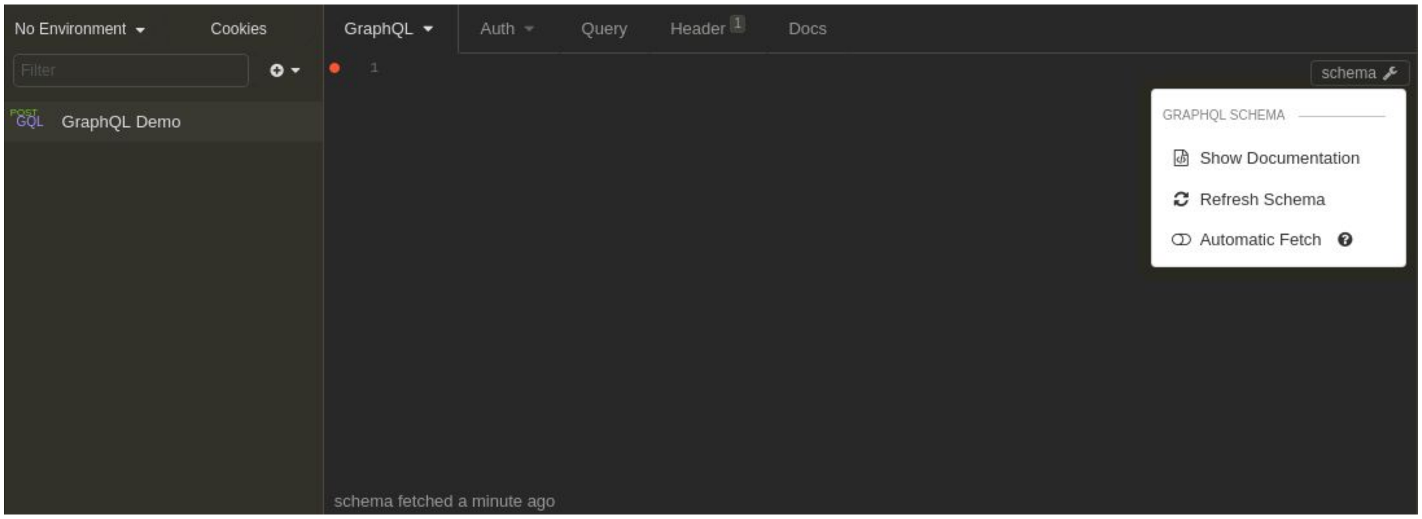




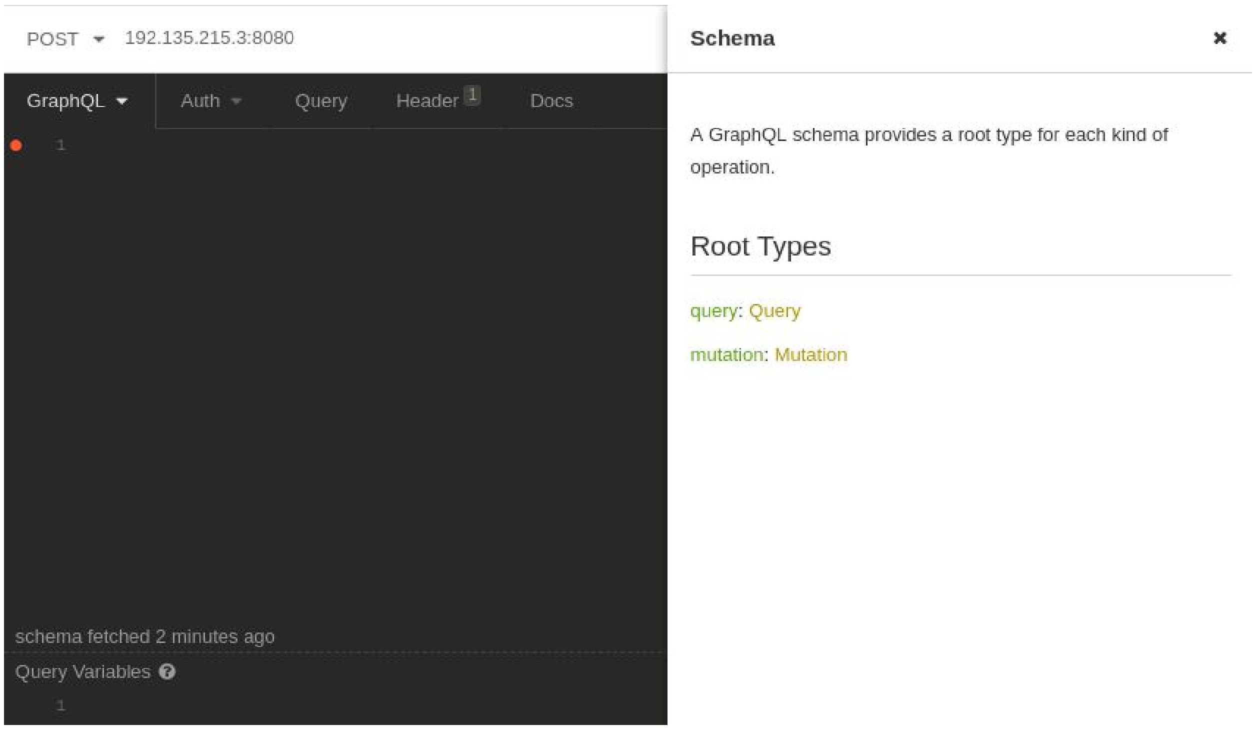
As it is mentioned in the challenge description that the introspection queries are not disabled. That means that the GraphQL schema could be accessed from the console and auto-completion would also work.

Exploring the GraphQL Schema:

Click on the schema button on the top right of the console window.



Click on Show Documentation:



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

Root Types

query: Query

mutation: Mutation

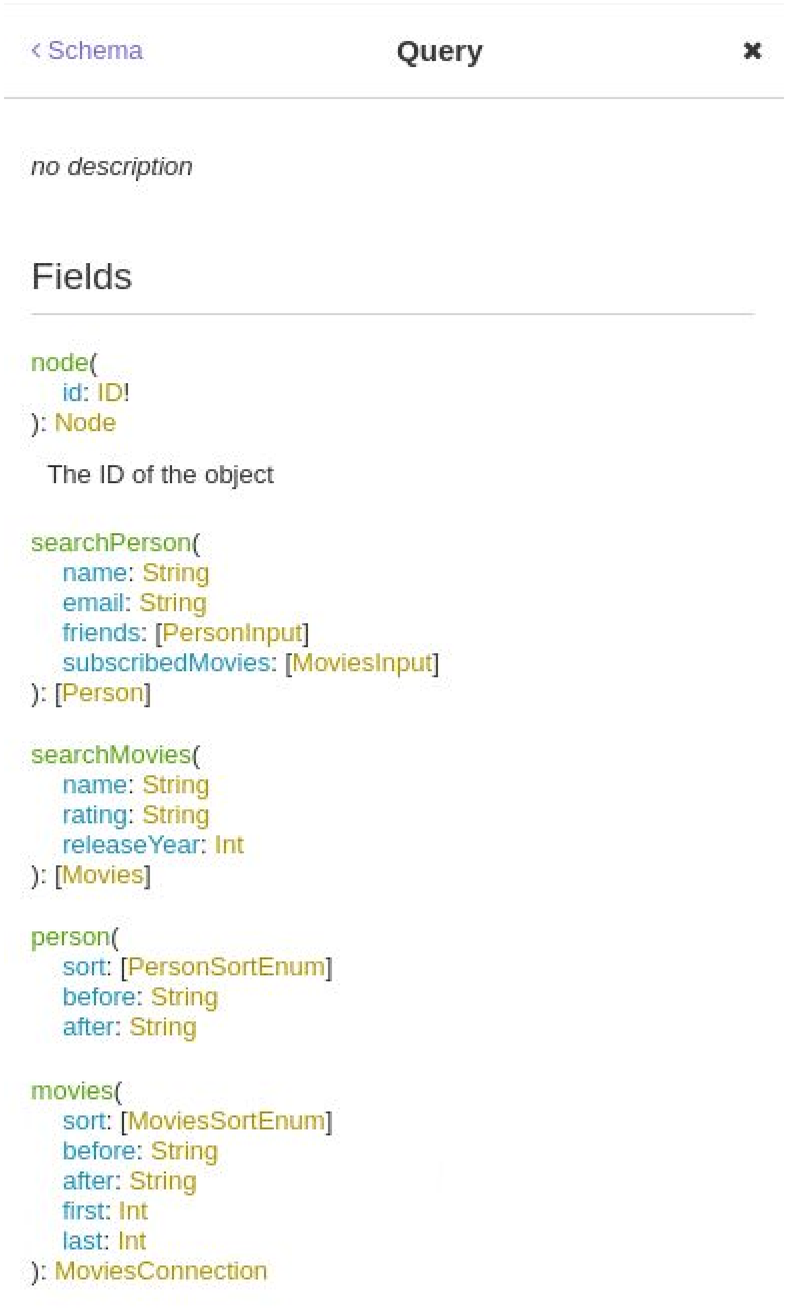
The Schema panel could be used to view the GraphQL schema.

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:

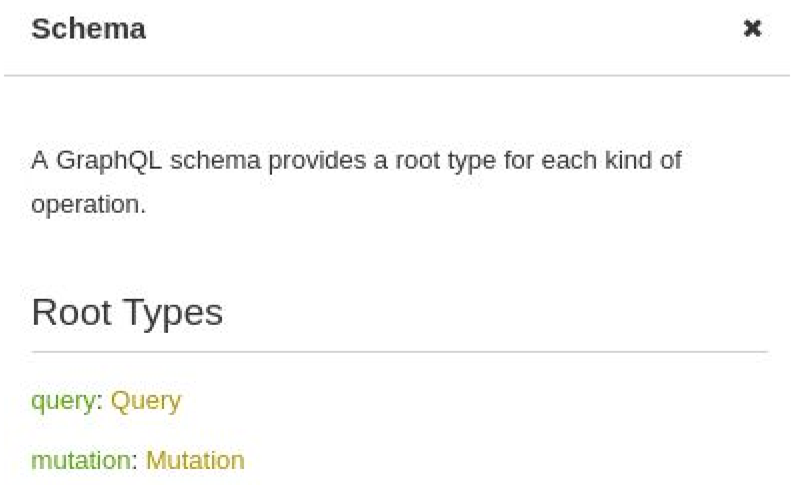


```
no description

Fields

node(
    id: ID!
): Node

  The ID of the object

searchPerson(
    name: String
    email: String
    friends: [PersonInput]
    subscribedMovies: [MoviesInput]
): [Person]

searchMovies(
    name: String
    rating: String
    releaseYear: Int
): [Movies]

person(
    sort: [PersonSortEnum]
    before: String
    after: String

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

So, there are 5 fields: node, searchPerson, searchMovies, person and movies.

For instance, searchMovies accepts movie name, rating and releaseYear.

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


Schema ✖

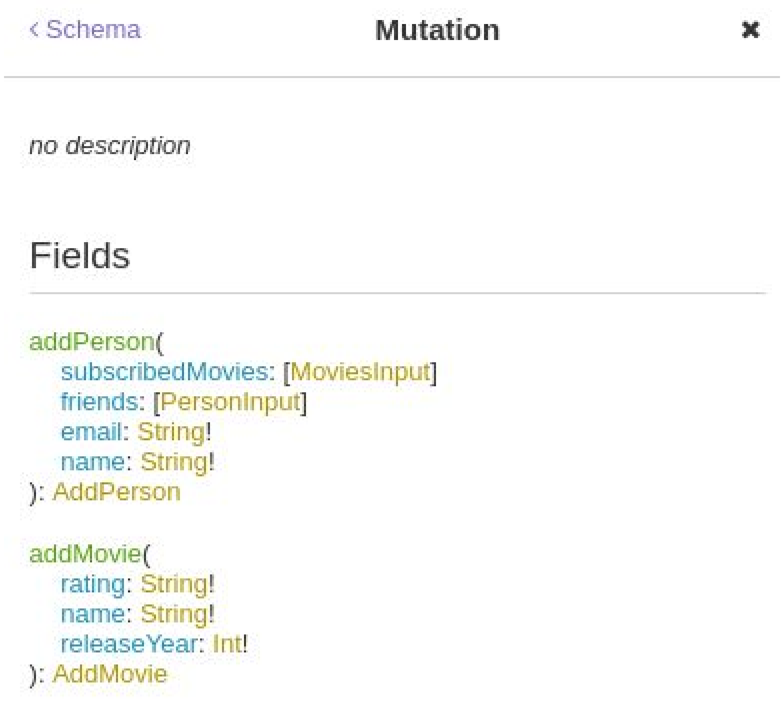
A GraphQL schema provides a root type for each kind of operation.

Root Types

query: Query

mutation: Mutation

Click on Mutations.


‹ Schema 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 4: Making queries to the backend database.

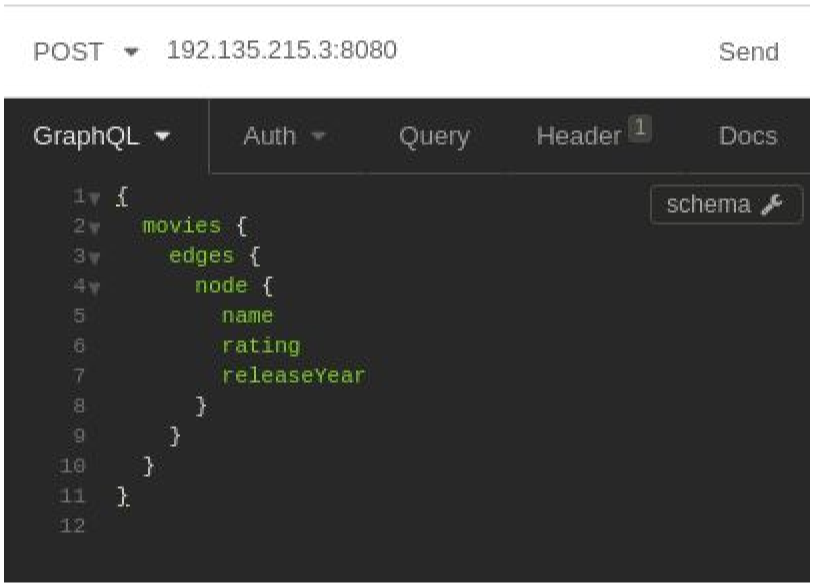
Writing queries in the console window to fetch the desired results from the backend database:

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

Query:

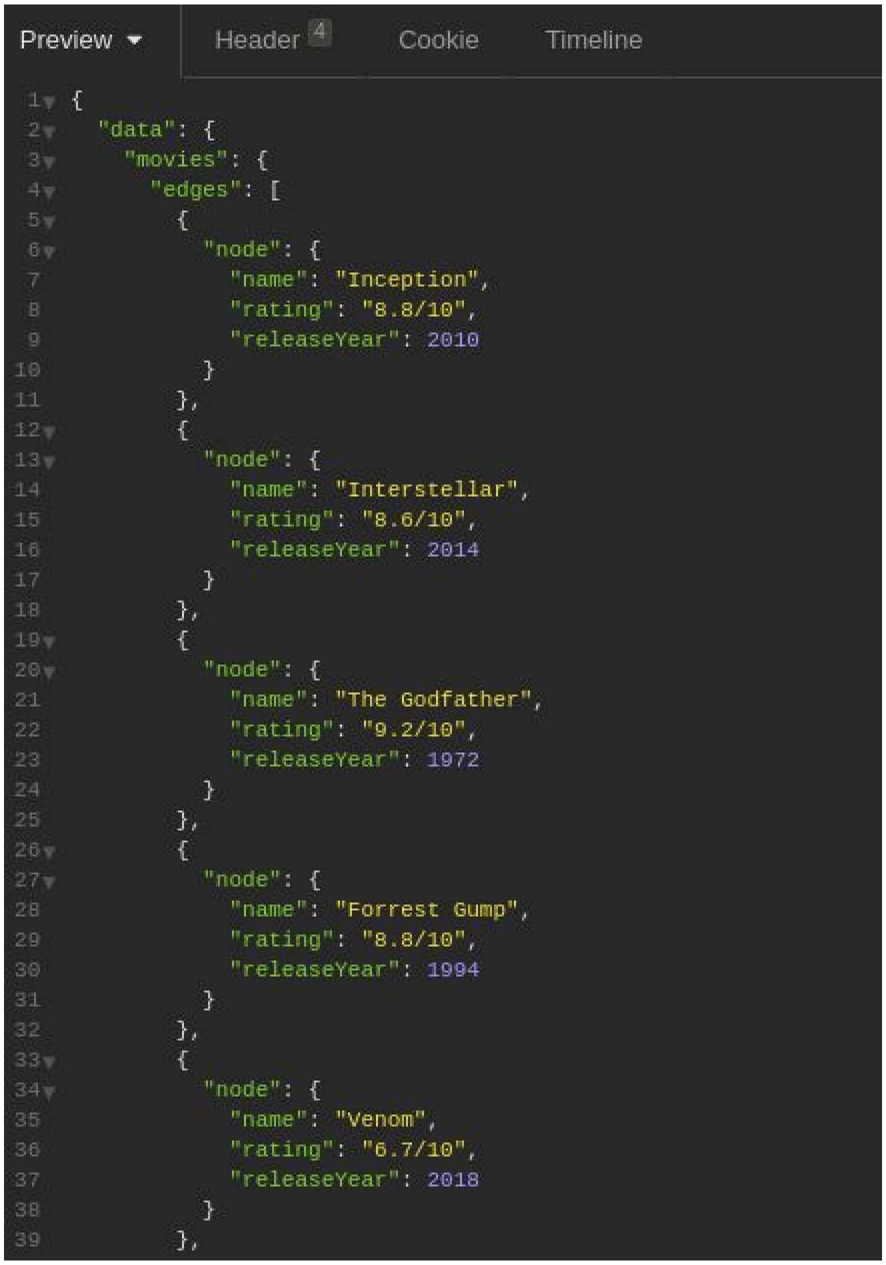
```
{
  movies {
        edges {
        node {
        name
        rating
        releaseYear
        }
        }
  }
}
```

Send the above request by clicking the Send button:



```
{
  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
          }
        },
```

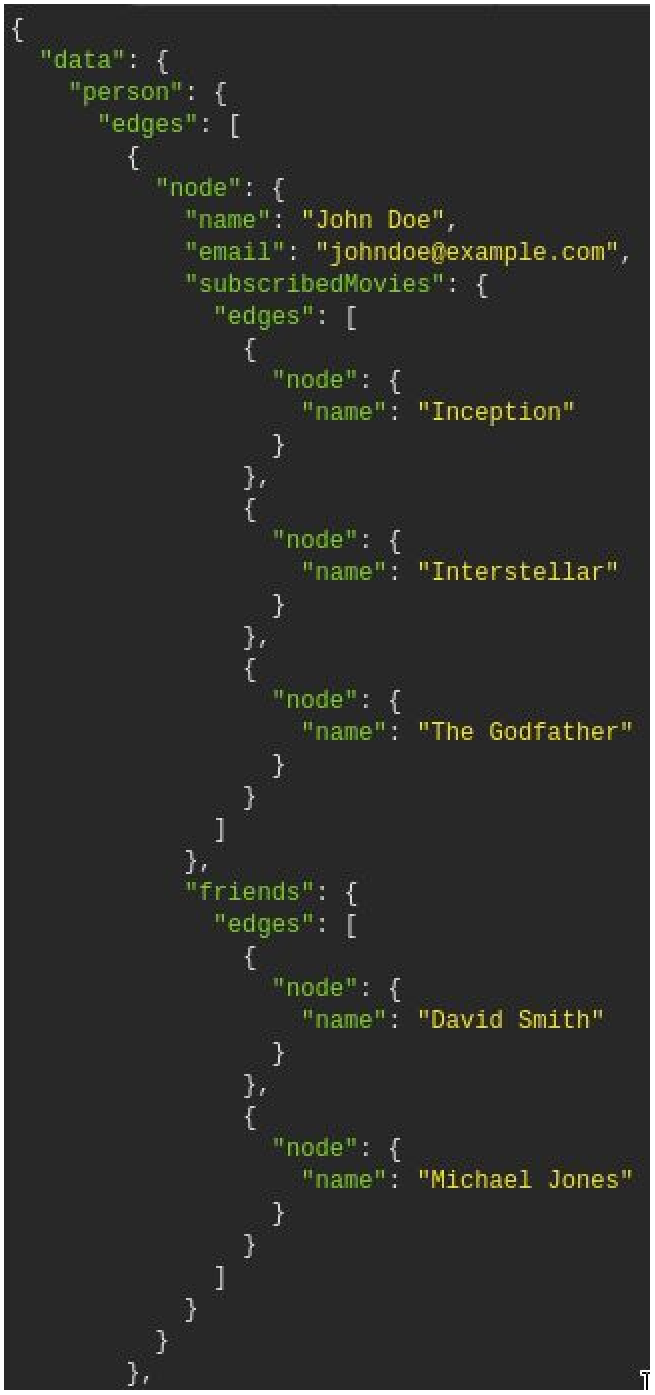
2. Use the following query to fetch all the person objects and display their name, email, name of subscribed movies and the name of their friends.

Query:

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

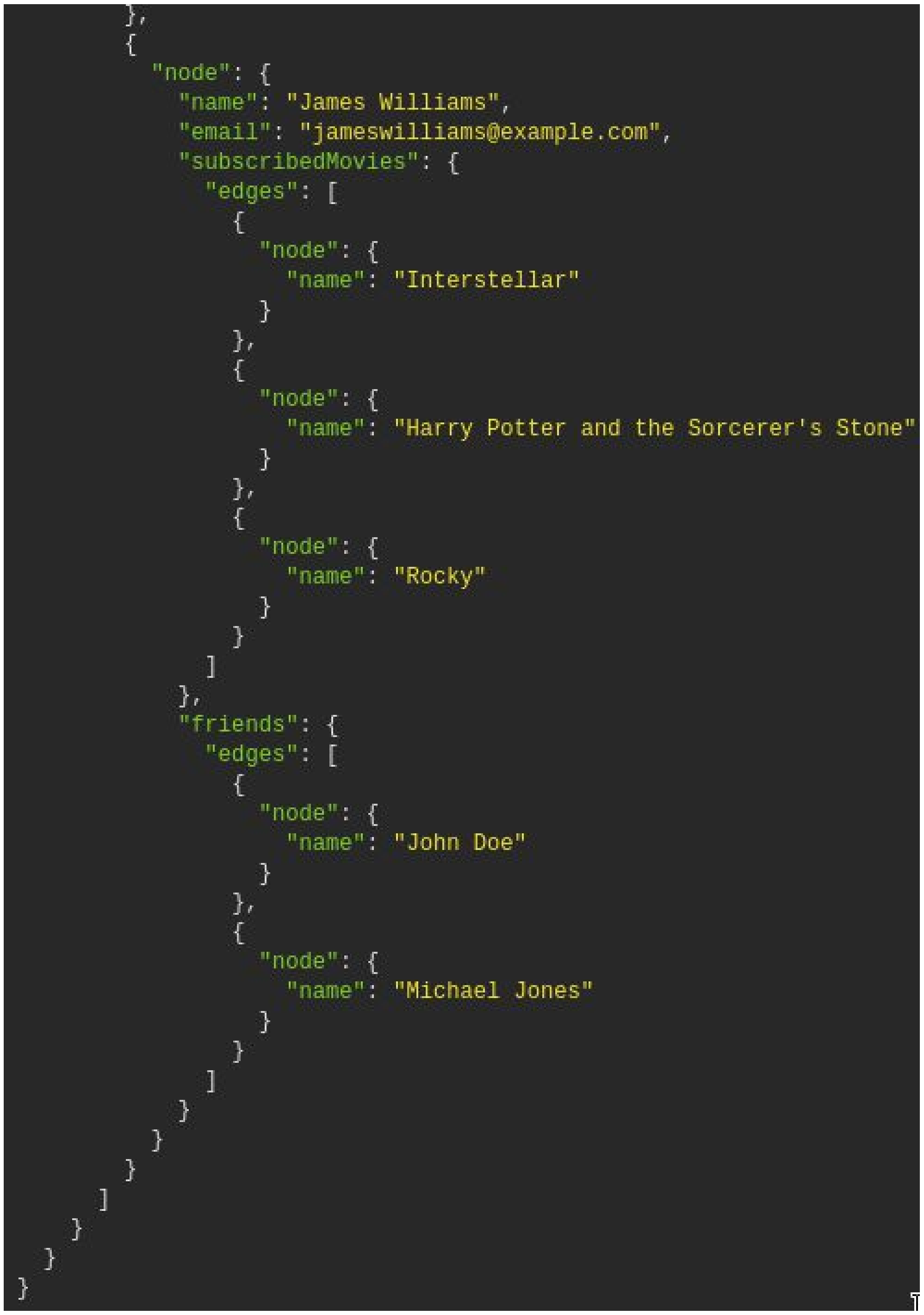
Send the request with this query.

Response:



```
{
  "data": {
    "person": {
      "edges": [
        {
          "node": {
            "name": "John Doe",
            "email": "johndoe@example.com",
            "subscribedMovies": {
              "edges": [
                {
                  "node": {
                    "name": "Inception"
                  }
                },
                {
                  "node": {
                    "name": "Interstellar"
                  }
                },
                {
                  "node": {
                    "name": "The Godfather"
                  }
                }
              ]
            },
            "friends": {
              "edges": [
                {
                  "node": {
                    "name": "David Smith"
                  }
                },
                {
                  "node": {
                    "name": "Michael Jones"
                  }
                }
              ]
            }
          }
        },
```

…



```
        },
        {
          "node": {
            "name": "James Williams",
            "email": "jameswilliams@example.com",
            "subscribedMovies": {
              "edges": [
                {
                  "node": {
                    "name": "Interstellar"
                  }
                },
                {
                  "node": {
                    "name": "Harry Potter and the Sorcerer's Stone"
                  }
                },
                {
                  "node": {
                    "name": "Rocky"
                  }
                }
              ]
            },
            "friends": {
              "edges": [
                {
                  "node": {
                    "name": "John Doe"
                  }
                },
                {
                  "node": {
                    "name": "Michael Jones"
                  }
                }
              ]
            }
          }
        }
      ]
    }
  }
}
```

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

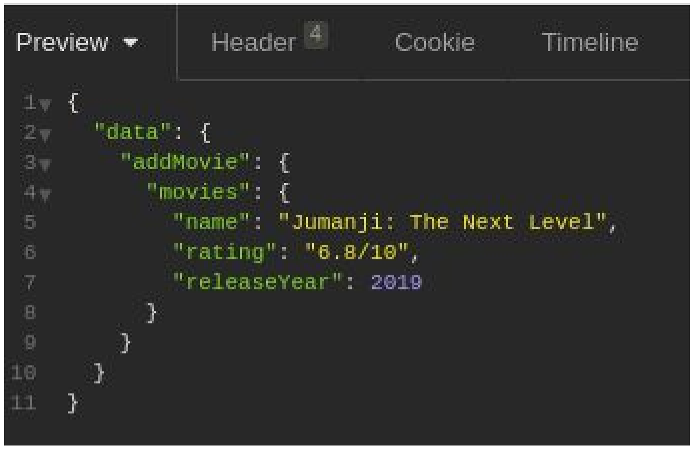
Step 5: Modifying the backend database using mutations.

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.

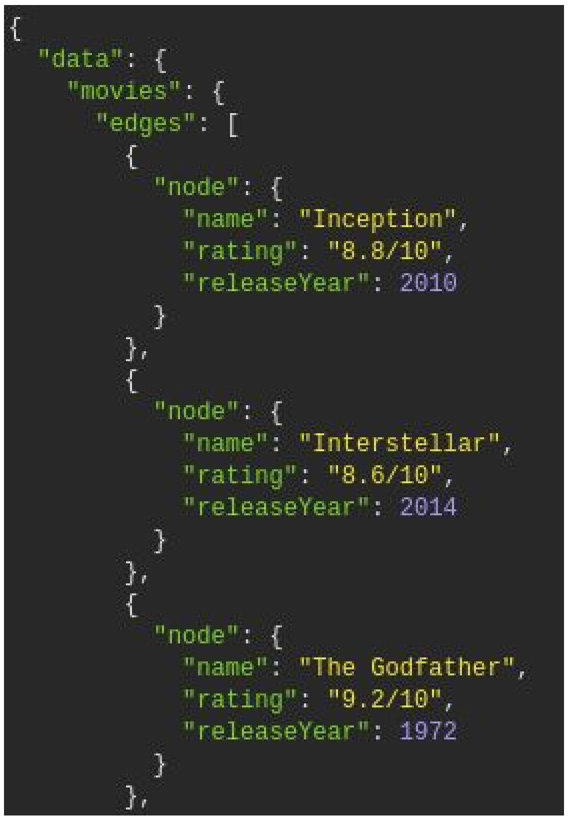
Notice that just like in queries, if the mutation field returns an object type, we can ask for nested fields. This can be useful for fetching the new state of an object without making multiple requests to the backend.

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

Query:

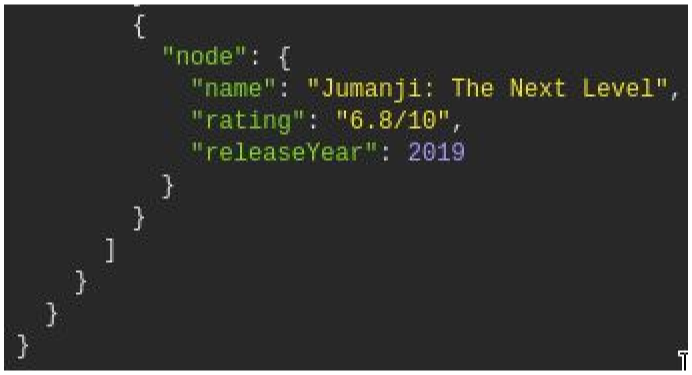
```
{
  movies {
           edges {
           node {
           name
           rating
           releaseYear
           }
           }
  }
}
```

Response:



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

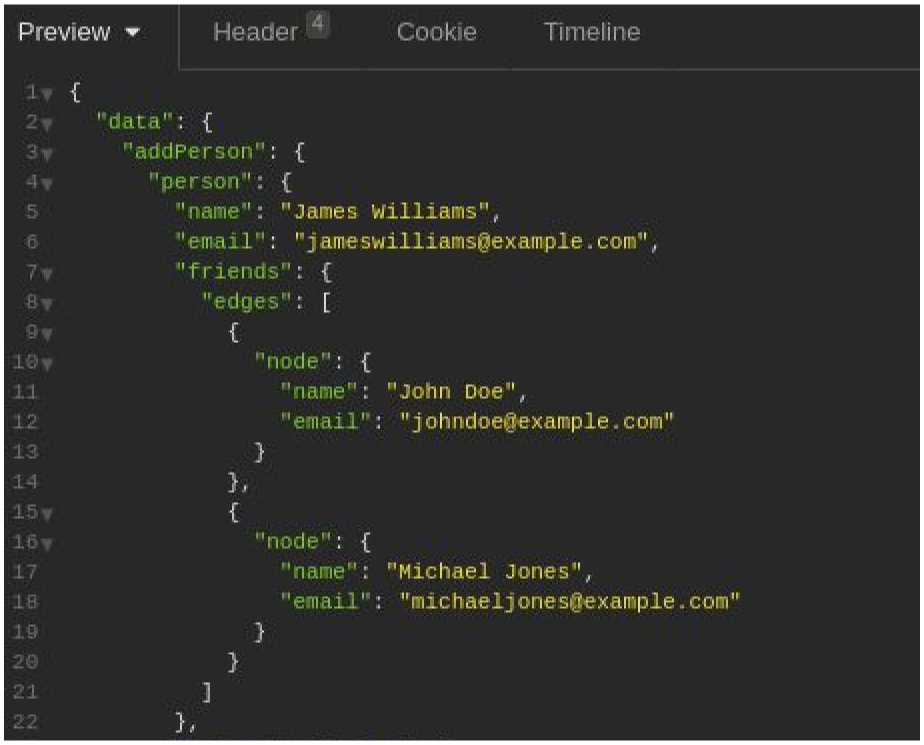
2. Use the following mutation to add a new Person to the backend database, providing a list of friends and a list of subscribed movies:

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
```

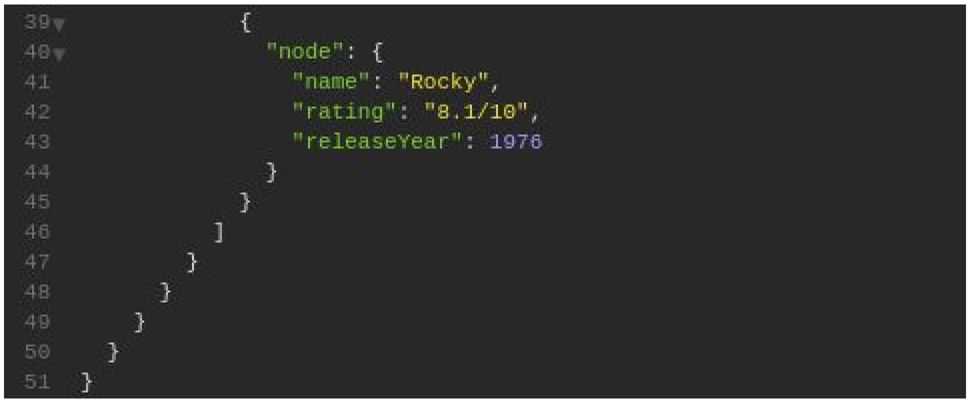
```
            friends {
            edges {
            node {
            name
            email
            }
            }
            }
            subscribedMovies {
            edges {
            node {
            name
            rating
            releaseYear
            }
            }
            }
            }
   }
}
```

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"
              }
            }
          ]
        },
```

...



```
              {
                "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 used "Insomnia" for interacting with the GraphQL endpoint and making different queries and adding objects using mutations.

References:

1. GraphQL (https://graphql.org)
2. Insomnia (https://insomnia.rest)