GraphQL | A query language for your API

What is GraphQL?

GraphQL is a **query language** for your API, and a server side runtime for executing queries using a type system you define for your data. GraphQL is not tied to any specific **database** or storage engine and is instead backed by your **existing code** and data for your application.

Benefits of GraphQL?

1. Less code
2. Avoiding multiple REST calls
3. Flexible
4. Better performance

Difference between GraphQL and REST API

GraphQL and REST are both architectural styles for developing APIs, but there are significant differences between them:

1. ***Query structure:*** REST uses HTTP methods like GET, POST, PUT, DELETE, etc. to manipulate resources, while GraphQL uses a single HTTP endpoint to execute complex queries and mutations.

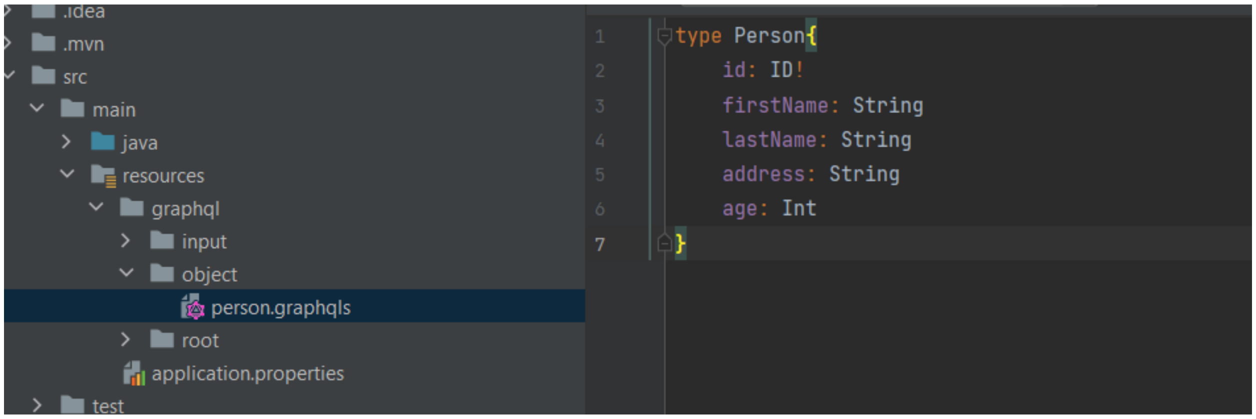
2. ***Data specificity:*** REST returns a complete set of data for a given resource, while GraphQL allows clients to specify exactly which data they are interested in at a given time.
3. ***Performance:*** REST requests usually trigger multiple network calls to fetch related data, while GraphQL allows all necessary data to be fetched in a single query.

But which one to choose?

Ultimately, the choice between REST and GraphQL depends on the specific needs of the application and the preferences of the development team.

How to work with GraphQL?

On the beginning you need to define your schema. GraphQL server uses schema to describe the shape of your available data. The following picture represent the person schema.



```
type Person{
    id: ID!
    firstName: String
    lastName: String
    address: String
    age: Int
}
```

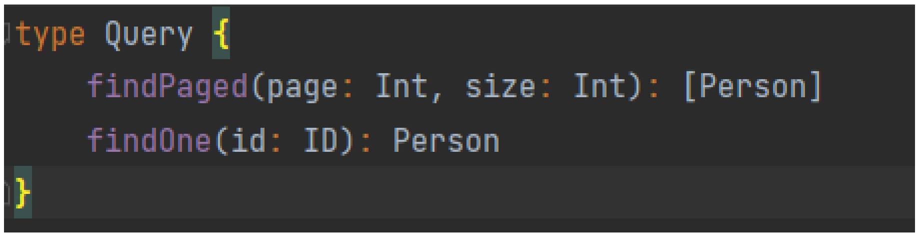
In **GraphQL,** there are several types of data and they are called scalar types.

1. String
2. Int
3. Float
4. Boolean
5. ID (indetificator)

In addition to the standard scalar types, GraphQL users can create custom scalar types to adapt the GraphQL schema to their needs.

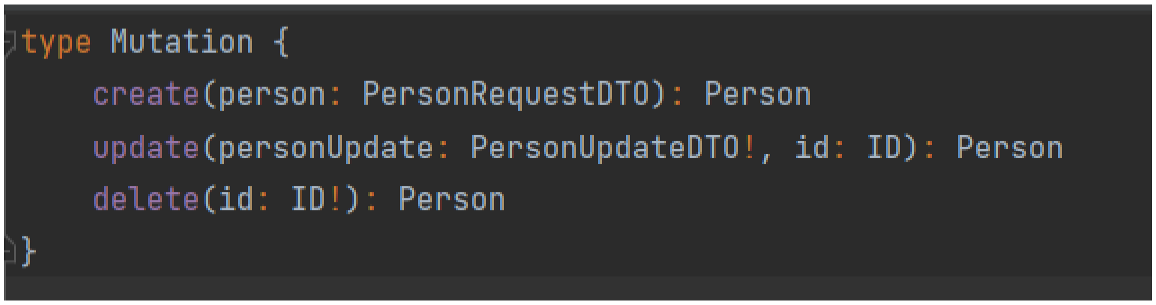
To manipulate with the specific data we use some important **root operation** named **queries**, **mutation** and **subscription.**

Queries are data requests made by the client from the server. Dissimilar to REST where there is an detailed format of data recurred from multiple endpoints, GraphQL only discloses single endpoint, enabling the client to determine what information it actually requires from a predefined framework.



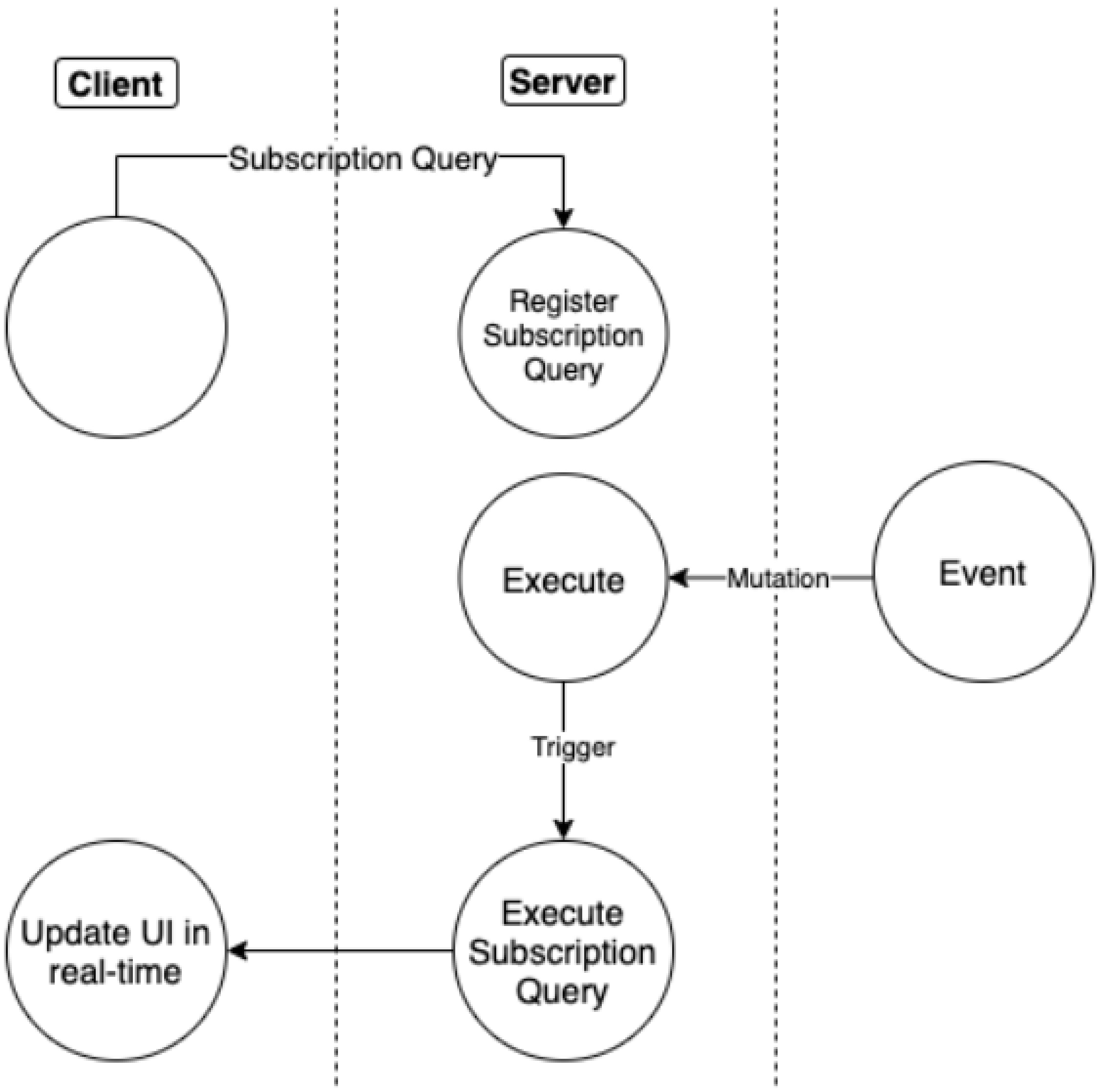
```
type Query {
    findPaged(page: Int, size: Int): [Person]
    findOne(id: ID): Person
}
```

Mutation are used to create, update or delete data. The structure is almost similar to queries except for the fact that you need to include the word mutation in the beginning.



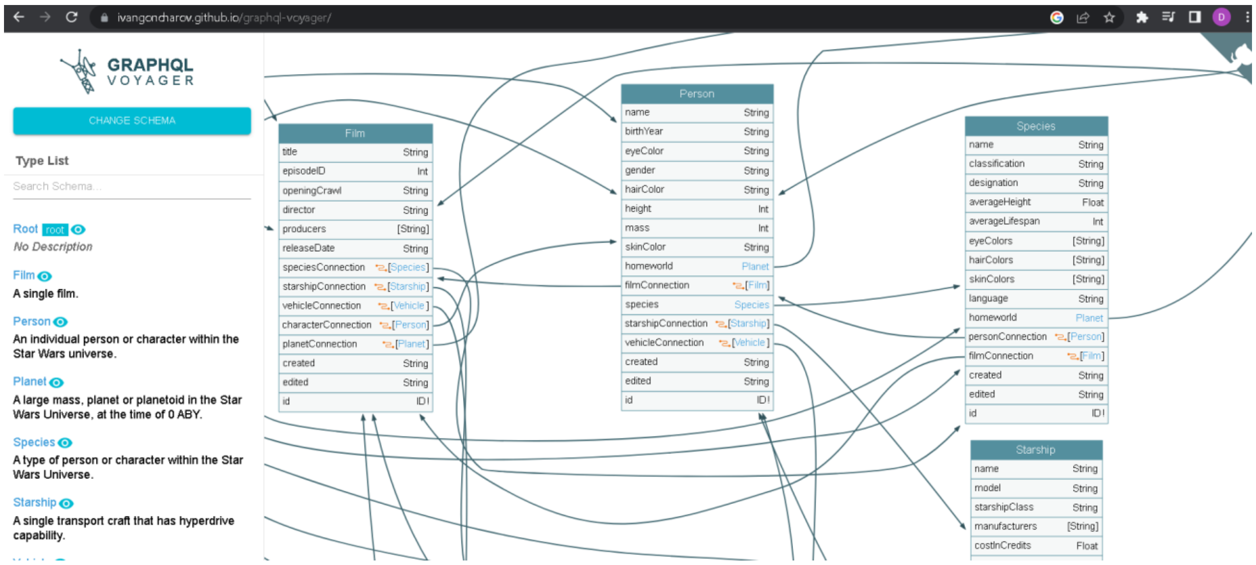
```
type Mutation {
    create(person: PersonRequestDTO): Person
    update(personUpdate: PersonUpdateDTO!, id: ID): Person
    delete(id: ID!): Person
}
```

Subscriptions can be used to set up and retain real-time link to the server. This allow you to get instant information regarding the relevant events. Client need to subscribe to the event in order to get the data.



Interactive tool for visualizing your GraphQL

GraphQL Voyager can be really helpful when discussing and designing your data model are required. UI of this application is in the following image below.


ivangoncharov.github.io/graphql-voyager/

GRAPHQL VOYAGER

CHANGE SCHEMA

Type List

Search Schema...

Root root
No Description

Film
A single film.

Person
An individual person or character within the Star Wars universe.

Planet
A large mass, planet or planetoid in the Star Wars Universe, at the time of 0 ABY.

Species
A type of person or character within the Star Wars Universe.

Starship
A single transport craft that has hyperdrive capability.

Film	
title	String
episodeID	Int
openingCrawl	String
director	String
producers	[String]
releaseDate	String
speciesConnection	[Species]
starshipConnection	[Starship]
vehicleConnection	[Vehicle]
characterConnection	[Person]
planetConnection	[Planet]
created	String
edited	String
id	ID!

Person	
name	String
birthYear	String
eyeColor	String
gender	String
hairColor	String
height	Int
mass	Int
skinColor	String
homeworld	Planet
filmConnection	[Film]
species	Species
starshipConnection	[Starship]
vehicleConnection	[Vehicle]
created	String
edited	String
id	ID!

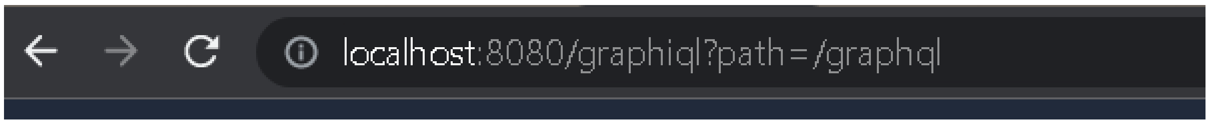
Species	
name	String
classification	String
designation	String
averageHeight	Float
averageLifespan	Int
eyeColors	[String]
hairColors	[String]
skinColors	[String]
language	String
homeworld	Planet
personConnection	[Person]
filmConnection	[Film]
created	String
edited	String
id	ID!

Starship	
name	String
model	String
starshipClass	String
manufacturers	[String]
costInCredits	Float

Where to test the endpoints?

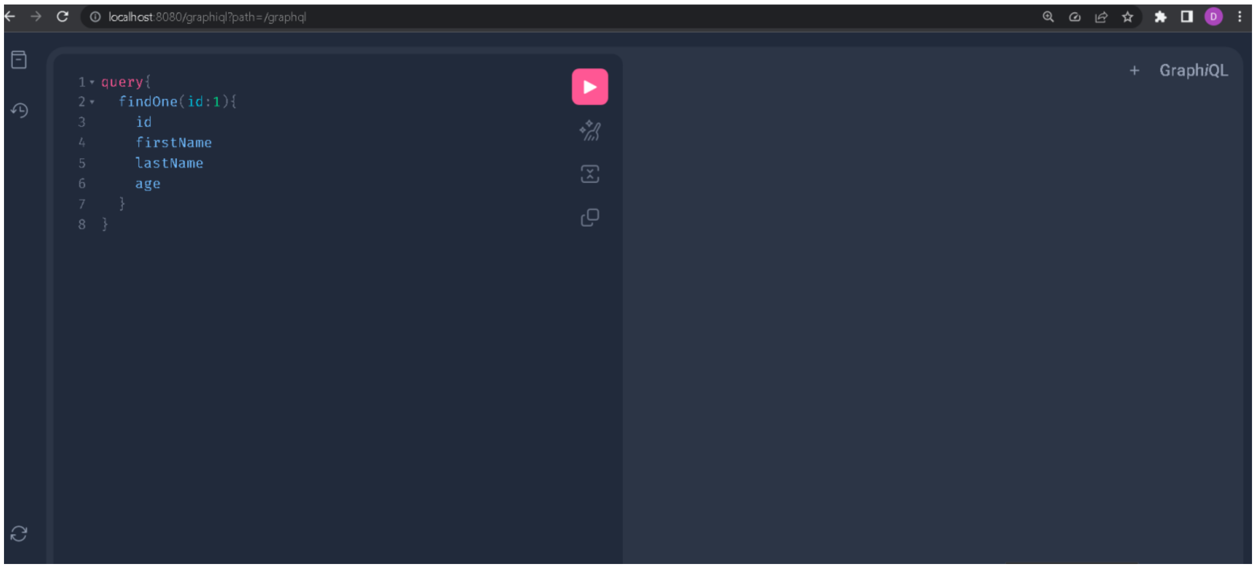
We are going to test the GraphQl endpoints in the client named GraphiQL.

To access to this client you need to run your application and hit the url in the following image.


localhost:8080/graphiql?path=/graphql

Link: http://localhost:8080/graphiql?path=/graphql

On the following link you will access the GraphQL client for testing your endpoints.


localhost:8080/graphiql?path=/graphql

```
query{
  findOne(id:1){
    id
    firstName
    lastName
    age
  }
}
```

Learning resource

1. Youtube channel of Dan Vega: https://www.youtube.com/@DanVega
2. Official documentation of GraphQL: https://graphql.org/