GraphQL is a query language for [APIs](https://dzone.com/articles/api-development-an-introductory-guide) that allows clients to request limited data they need, making it possible for clients to gather data in a limited number of requests. GraphQL is a strongly-typed protocol and all data operations are validated against a GraphQL schema.

Example:

<https://dzone.com/articles/a-beginners-guide-to-graphql-with-spring-boot>

## Adding Maven Dependencies

1. graphql-spring-boot-starter is used for enabling GraphQL

servlet, and it becomes available at a path /graphql. It initializes the GraphQLSchema bean.

1. graphql-java allows us to write schema with GraphQL schema language, which is simple to understand.
2. graphiql-spring-boot-starter provides a user interface with which we could test our GraphQL queries and view query definitions.

## GraphQL Schema

GraphQL comes with its own language to write GraphQL Schemas called [Schema Definition Language](https://www.howtographql.com/basics/2-core-concepts/) (SDL). The schema definition consists of all the API functionalities available at an endpoint.

type Vehicle {

id: ID!,

type: String,

modelCode: String,

brandName: String,

launchDate: String

}

type Query {

vehicles(count: Int):[Vehicle]

vehicle(id: ID):Vehicle

}

type Mutation {

createVehicle(type: String!, modelCode: String!, brandName: String, launchDate: String):Vehicle

}

Create a graphql folder under src/main/resources, and create a vehicleql.graphqls file under that folder. Copy the above contents and paste it in the vehicleql.graphqls file.

The Query type represents the query that can be made to the GraphQL server to fetch data

The Mutation type represents the queries that are used to perform write operations on the data.

## Root Query

Query or Mutation objects are root GraphQL objects. They don’t have any associated data class. In such cases, the resolver classes would implement GraphQLQueryResolver or GraphQLMutationResolver. These resolvers will be searched for methods that map to fields in their respective root types.

## Testing the Application

http://localhost:8080/graphiql

**Run,**

Req:

query {

vehicles(count: 1)

{

id,

type,

modelCode

}

}

Res:

{

"data": {

"vehicles": [

{

"id": "1",

"type": "bus",

"modelCode": "XYZ123"

}

]

}

}

Req:

mutation {

createVehicle(type: "car", modelCode: "XYZ0192", brandName: "XYZ", launchDate: "2016-08-16")

{

id

}

}

Res:

{

"data": {

"createVehicle": {

"id": "1"

}

}

}