# What is a Mutation in GraphQL?

Whenever you want to write data back into the server, mutations are used.

## How is Mutation and Query Different?

**Query** is used when you want to read some data from the server while **mutation** is used when you want to write data back to the server.

But wait. Can’t I go to the resolver in the **query** and do a write operation?

Though it is possible to do a write operation in a **query**, it shouldn’t be done. It is necessary to separate the read the write operations and hence **mutations** are needed.

**Add Movie Mutation**

Let us create a mutation which can be used to add a new movie.

create a new file called as **mutation.js**. Copy the following code into **mutation.js**

**const** { GraphQLObjectType

} **=** require('graphql');

**const** \_ **=** require('lodash');

**const** {movieType} **=** require('./types.js');

**const** {inputMovieType} **=** require('./inputtypes.js');

**let** {movies} **=** require('./data.js');

**const** mutationType **=** **new** GraphQLObjectType({

name: 'Mutation',

fields: {

addMovie: {

type: movieType,

args: {

input: { type: inputMovieType }

},

resolve: **function** (source, args) {

**let** movie **=** {

id: args.input.id,

name: args.input.name,

year: args.input.year,

directorId: args.input.directorId};

movies.push(movie);

**return** \_.find(movies, { id: args.input.id });

}

}

}

});

exports.mutationType **=** mutationType;

You will notice that a mutation looks very similar to a query. The main difference is that the name of the **GraphQLObjectType** is **Mutation**.

Here we have added a mutation called as **addMovie** which has a return type of **movieType** ( *movieType* was covered in the [previous](https://adityasridhar.com/posts/what-is-graphql-and-how-to-use-it) blog )

In args we are mentioning that we need a parameter called **input** which is of type **inputMovieType**

So what is **inputMovieType** here?

**Input Types**

It is possible that multiple mutations need the same input arguments. So it is a good practise to create **Input Types** and reuse the Input Types for all these mutations.

Here we are creating a input type for the movie called as **inputMovieType**.

It is seen that **inputMovieType** in turn comes from **inputtypes.js** file. Let us create this now.

Create a new file called as **inputtypes.js**

Copy the following code into inputtypes.js

**const** {

GraphQLInputObjectType,

GraphQLID,

GraphQLString,

GraphQLInt

} **=** require('graphql');

inputMovieType **=** **new** GraphQLInputObjectType({

name: 'MovieInput',

fields: {

id: { type: GraphQLID },

name: { type: GraphQLString },

year: { type: GraphQLInt },

directorId: { type: GraphQLID }

}

});

exports.inputMovieType **=** inputMovieType;

It is seen that an Input Type looks exactly like any other Type in GraphQL. **GraphQLInputObjectType** is used to create an Input Type, while **GraphQLObjectType** is used to create normal Types.

**Resolve function of a mutation**

The resolve function of a mutation is where the actual write operation happens.

In a real application this can be a Database write operation.

For this example, we are just adding the data to movies array and then returning the added movie back.

resolve: **function** (source, args) {

**let** movie **=** {

id: args.input.id,

name: args.input.name,

year: args.input.year,

directorId: args.input.directorId};

movies.push(movie);

**return** \_.find(movies, { id: args.input.id });

}

The above code in resolve does the following actions

* Gets the input movie parameters from **input** arg.
* Adds the new movie to the movies array
* Returns the new movie which was added, by fetching it from the movies array

**Add Director Mutation**

Let us create a mutation which can be used to add a new director

This would be exactly same as adding the movie Mutation.

**inputtypes.js** with director Mutation added

**const** {

GraphQLInputObjectType,

GraphQLID,

GraphQLString,

GraphQLInt

} **=** require('graphql');

inputMovieType **=** **new** GraphQLInputObjectType({

name: 'MovieInput',

fields: {

id: { type: GraphQLID },

name: { type: GraphQLString },

year: { type: GraphQLInt },

directorId: { type: GraphQLID }

}

});

inputDirectorType **=** **new** GraphQLInputObjectType({

name: 'DirectorInput',

fields: {

id: { type: GraphQLID },

name: { type: GraphQLString },

age: { type: GraphQLInt }

}

});

exports.inputMovieType **=** inputMovieType;

exports.inputDirectorType **=** inputDirectorType;

**mutation.js** after adding **addDirector** mutation

**const** { GraphQLObjectType

} **=** require('graphql');

**const** \_ **=** require('lodash');

**const** {movieType,directorType} **=** require('./types.js');

**const** {inputMovieType,inputDirectorType} **=** require('./inputtypes.js');

**let** {movies,directors} **=** require('./data.js');

**const** mutationType **=** **new** GraphQLObjectType({

name: 'Mutation',

fields: {

addMovie: {

type: movieType,

args: {

input: { type: inputMovieType }

},

resolve: **function** (source, args) {

**let** movie **=** {

id: args.input.id,

name: args.input.name,

year: args.input.year,

directorId: args.input.directorId};

movies.push(movie);

**return** \_.find(movies, { id: args.input.id });

}

},

addDirector: {

type: directorType,

args: {

input: { type: inputDirectorType }

},

resolve: **function** (source, args) {

**let** director **=** {

id: args.input.id,

name: args.input.name,

age: args.input.age};

directors.push(director);

**return** \_.find(directors, { id: args.input.id });

}

}

}

});

exports.mutationType **=** mutationType;

**Enabling the mutations**

Until now we have defined the mutation end points and their functionality. But we haven’t enabled the mutations yet.

To enable them, the mutations have to added to the schema.

The mutation is added using the following code in **server.js**

*// Define the Schema*

**const** schema **=** **new** GraphQLSchema(

{

query: queryType,

mutation: mutationType

}

);

Complete Code in **server.js** after adding the mutation

*//get all the libraries needed*

**const** express **=** require('express');

**const** graphqlHTTP **=** require('express-graphql');

**const** {GraphQLSchema} **=** require('graphql');

**const** {queryType} **=** require('./query.js');

**const** {mutationType} **=** require('./mutation.js');

*//setting up the port number and express app*

**const** port **=** 5000;

**const** app **=** express();

*// Define the Schema*

**const** schema **=** **new** GraphQLSchema(

{

query: queryType,

mutation: mutationType

}

);

*//Setup the nodejs GraphQL server*

app.use('/graphql', graphqlHTTP({

schema: schema,

graphiql: **true**,

}));

app.listen(port);

console.log(`GraphQL Server Running at localhost:${port}`);

**Code**

The complete code for this article can be found in [this git repo](https://github.com/aditya-sridhar/graphql-mutations-with-nodejs)

**Testing The mutation End Points**

Run the application using the following command

node server.js

Open your web browser and go to the following url **localhost:5000/graphql**

**Test addMovie Mutation Endpoint**

Input

mutation {

addMovie(input: {**id**: 4,name: "Movie 4", year: 2020,directorId:4}){

id,

name,

year,

directorId

}

}

Output

{

"data": {

"addMovie": {

"id": "4",

"name": "Movie 4",

"year": 2020,

"directorId": "4"

}

}

}

Input

mutation {

addMovie(input: {**id**: 5,name: "Movie 5", year: 2021,directorId:4}){

id,

name,

year,

directorId

}

}

Output

{

"data": {

"addMovie": {

"id": "5",

"name": "Movie 5",

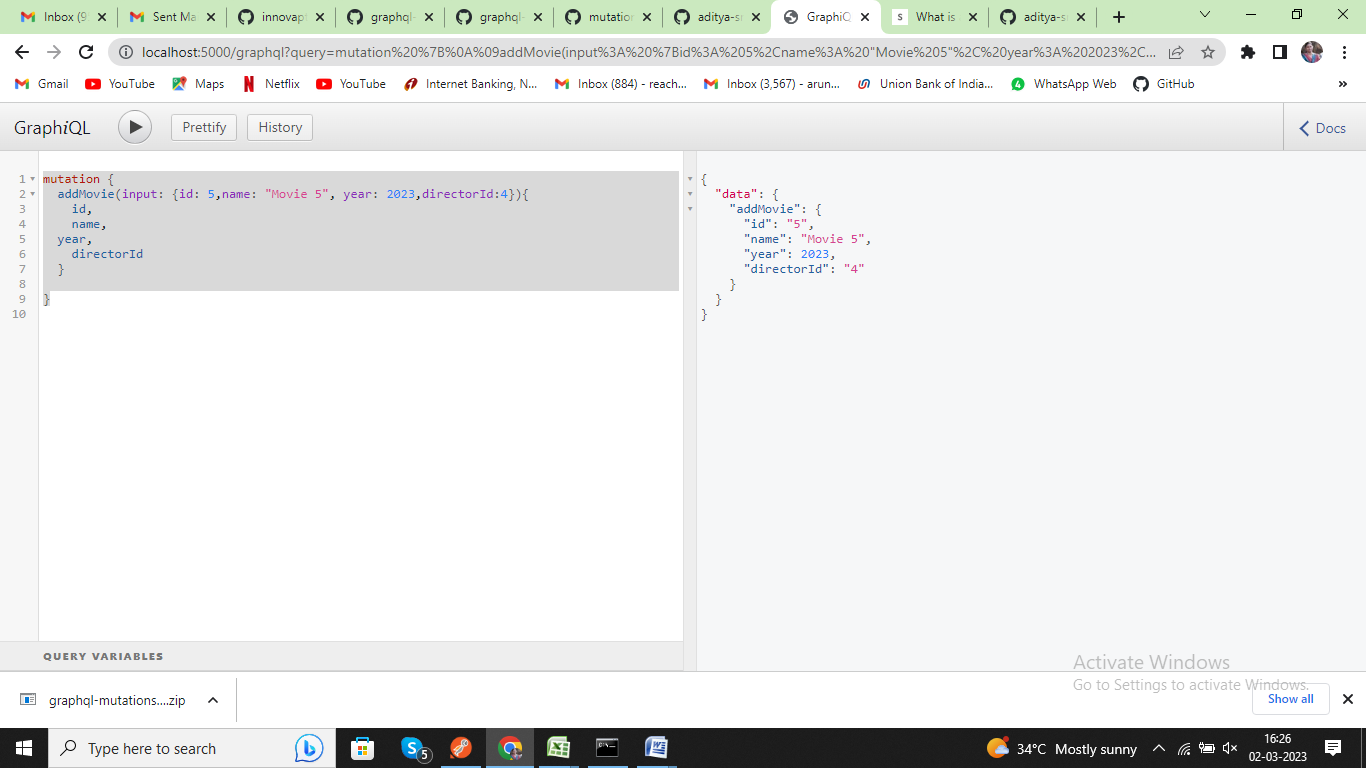
"year": 2021,

"directorId": "4"

}

}

}

****

**Test addDirector Mutation Endpoint**

Input

mutation {

addDirector(input: {**id**: 4,name: "Director 4", age: 30}){

id,

name,

age,

movies{

id,

name,

year

}

}

}

Output

{

"data": {

"addDirector": {

"id": "4",

"name": "Director 4",

"age": 30,

"movies": [

{

"id": "4",

"name": "Movie 4",

"year": 2020

},

{

"id": "5",

"name": "Movie 5",

"year": 2021

}

]

}

}

}

