Step-step build graphQL in general :

* Create a data access layer for loads database Store
* Create schema file, schema.graphql
* Create Resolver file , resolvers.js
* Create Server and Configure Graphql , Server.js
* Run and Test with Graphiql Application

Dependencies for graphql on general:

"dependencies": {

"apollo-server-express": "^1.4.0",

"body-parser": "^1.18.3",

"cors": "^2.8.4",

"express": "^4.16.3",

"graphql": "^0.13.2",

"graphql-tools": "^3.1.1"

},

GraphQL Server can be deployed by using any of the three methods:

* GraphQL server with connected database
* GraphQL server that integrates existing system
* Hybrid approach

GraphQL has 2 components for front-end and backend:

* Client-side Components : GraphiQL , and ApolloClient
* Server-side Components : Query , Schema , and Resolver

GraphQL - Type System

1. Scalar , Stores a single value:

* **Int** − Signed 32-bit Integer
* **Float** − Signed double precision floating point value
* **String** − UTF - 8-character sequence
* **Boolean** − True or false
* **ID** − A unique identifier, often used as a unique identifier to fetch an object or as the key for a cache.

Like:

type schools

}

FirstName: String;

LastName: String;

Age: 34;

Graduate: true;

}

1. Object Type , represent a group of fields Each field inside an object type maps to another type , thereby allowing nested types.

// --Define an object type--

type Student {

stud\_id:ID

firstname: String

age: Int

score:Float

}

// -- Define a GraphQL schema—

Type Query

{

Stud\_details: [Student]

}

1. Query , A GraphQL query is used to fetch data. It is like requesting a resource in REST-based APIs. GraphQL uses the Schema Definition Language (SDL) to define a Query. Query type is one of the many root-level types in GraphQL.

//-- example of defining a Query –

type Query {

greeting: String

}

1. Mutation

Mutations are operations for to create , update or delete data . These are analogous to the PUT , POST , PATCH , and DELETE . The Query type defines the entry-points for data-fetching operations whereas the Mutation type specifies the entry points for data-manipulation operations.

// --The syntax for defining a Mutation type is given –

Type Mutation {

addStudent(firstName: String , lastName: String ) : Student

}

1. Enum Type

An Enum is similar to a scalar type .  useful in a situation where the value for a field must be from a prescribed list of options.

Following snippet illustrates how an enum type can defined –

Type Days\_of\_Week {

SUNDAY

MONDAY

TUESDAY

WEDNESDAY

THURSDAY

FRIDAY

SATURDAY

}

1. List , can be used to represent an array of values of specific type . defined with a type modifier []  that wraps object types, scalars, and enums.

type Query {

todos: [String]

}

1. Non-Nullable type , To override this default and specify that a field must be defined . By default, each of the core scalar types can be set to null. In other words  the specified type or can have no value.

type Student {

stud\_id:ID!

firstName:String

lastName:String

fullName:String

college:College

}

Resolver

Resolver is a collection of functions that generate response for a GraphQL query. In simple terms, a resolver acts as a GraphQL query handler.

Syntax: fieldName:( root, args, context, info ) => { result }

An example of resolver functions is shown below −

//resolver function with no parameters and returning string

greeting:() => {

return "hello from TutorialsPoint !!!"

}

//resolver function with no parameters and returning list

students:() => db.students.list()

//resolver function with arguments and returning object

studentById:(root,args,context,info) => {

return db.students.get(args.id);

Sr.No. Arguments & Description

1 root

The object that contains the result returned from the resolver on the parent field.

2 args

An object with the arguments passed into the field in the query.

3 context

This is an object shared by all resolvers in a particular query.

4 info

It contains information about the execution state of the query, including the field name, path to the field from the root.

Query

A GraphqQL query is used to read or fetch values while a mutation is used to write or post values.

-- The syntax to define a query is as follows --

//syntax

query query\_name{ someField }

//syntax

{ someField }

//query with name myQuery

query myQuery{

greeting

}

// query without any name

{

greeting

}

## **Illustration # 1**

Create a Schema first:

type Query {

greeting:String

students:[Student]

studentById(id:ID!):Student

}

type Student {

id:ID!

firstName:String

lastName:String

password:String

collegeId:String

fullName: String

}

Ceate Resolver:

const db = require('./db')

const Query = {

//resolver function for greeting

greeting:( ) => {

return "hello from TutorialsPoint !!!"

},

//resolver function for students returns list

students:( ) => db.students.list(),

//resolver function for studentbyId

studentById:( root,args,context,info ) => {

//args will contain parameter passed in query

return db.students.get(args.id);

}

}

//for each single student object returned,resolver is invoked

const Student = {

fullName:(root,args,context,info) => {

return root.firstName+":"+root.lastName

}

}

module.exports = {Query , Student}

Run Applicion#1

Open the browser and enter the url , http: //localhost: 9000/graphiql

--query on editor graphiql—

{

studentById(id:"S1001") {

id

firstName

lastName

}

}

--The output query –

{

"data": {

"studentById": {

"id": "S1001",

"firstName": "Mohtashim",

"lastName": "Mohammad"

}

}

}

Run Applicion#2

Open the browser and enter the url , http: //localhost: 9000/graphiql

{

students{

id

fullName

}

}

--The output query –

{

"data": {

"students": [

{

"id": "S1001",

"fullName": "Mohtashim:Mohammad"

},

{

"id": "S1002",

"fullName": "Kannan:Sudhakaran"

},

{

"id": "S1003",

"fullName": "Kiran:Panigrahi"

}

]

}

}

Code server

Code on Server.js

const bodyParser = require('body-parser');

const cors = require('cors');

const express = require('express');

const db = require('./db');

const port = 9000;

const app = express();

//loading type definitions from schema file

const fs = require('fs')

const typeDefs = fs.readFileSync('./schema.graphql',{encoding:'utf-8'})

//loading resolvers

const resolvers = require('./resolvers')

//binding schema and resolver

const {makeExecutableSchema} = require('graphql-tools')

const schema = makeExecutableSchema({typeDefs, resolvers})

//enabling cross domain calls and form post

app.use(cors(), bodyParser.json());

//enabling routes

const {graphiqlExpress,graphqlExpress} = require('apollo-server-express')

app.use('/graphql',graphqlExpress({schema}))

app.use('/graphiql',graphiqlExpress({endpointURL:'/graphql'}))

//registering port

app.listen(port, () => console.info(`Server started on port ${port}`));

illustration 3 – Nested Query

create schema :

type College {

id: ID!

name: String

location: String

rating: Float

}

Type Student {

Id: ID!

firstName: String

lastName:String

fullName:String

college: College

}

Modify the resolvers:

const Student = {

fullName: (root , args , context , info) => {

return root.firstName+":"+root.lastName

} ,

college: (root) => {

return db.colleges.get(root.collegeId);

}

}

module.exports = { Query , Student }

How to use Query Variable with Enum:

Step1 – Edit schema.graphql File

Enum , Useful in a situation where you need the user to pick from a prescribed list of options

Enum ColorType {

RED

BLUE

GREEN

}

type Query {

setFavouriteColor(color:ColorType):String

}

Ste2 – Edit resolvers.js file

The resolver function setFavouriteColor takes root and args. The enum value passed to function at runtime

setFavouriteColor:(root,args) => {

return "Your Fav Color is :"+args.color;

}

Step 3 – Declare a Query Variable in GraphiQL

query query\_to\_setColor($color\_variable:ColorType) {

setFavouriteColor(color:$color\_variable)

}

In the query variable section of GraphiQL, type the following code −

{

"color\_variable":"RED"

}

Output:

{

"data": {

"setFavouriteColor": "Your Fav Color is: RED"

}

}