# Instalar

* yarn add graphql

# Apollo

https://www.apollographql.com/docs/apollo-server/getting-started

* yarn add @apollo/server
* yarn add --save-dev typescript @types/node

crear el tsconfig.json para las configuraciones de TS

{

    "compilerOptions": {

      "rootDirs": ["src"],

      "outDir": "dist",

      "lib": ["es2020"],

      "target": "es2020",

      "module": "esnext",

      "moduleResolution": "node",

      "esModuleInterop": true,

      "types": ["node"]

    }

  }

En package.json hay que agregar

{

  …

  "type":"module",

"scripts": {

   ,"compile":"tsc"

    ,"start":"npm run compile && node ./dist/index.js"

  },

  …

}

# Componentes principales de un proyecto

* origen de datos
* definición de tipos
* Solucionadores
* Mutaciones
* Suscripciones

# Definir tipos de datos

const typeDefs=`

    type TipoDeDato {

atributoId: ID!

        atributo1: Tipo

        atributo2: [Tipo]

    }

type Query {

        nombreQueryLista:[TipoDeDato1]

nombreQueryGet(atributoId: ID!):TipoDeDato1

nombreQueryGet2(

atributoId: ID!

atributo2: String

):TipoDeDato1

    }

type Mutation {

        addTipo1(

            atributo1:String!

            atributo2:String!

        ):TipoDeDato1

    }

`

El tipo puede ser String, un arreglo seria [Tipo], un id seria ID,

Un tipo boligatorio seria Tipo!

## Definir querys querys

Hay que agregarle siempre al final el tipo Query que define como van a ser nuestras consultas, estas consultas son de tipo de obtención, osea no realizan cambios en los datos

Para habilitar una lista de un tipo

nombreQueryLista:[TipoDeDato1]

Para una consulta que solo retorne un solo elemento y pida algún argumento (tipo un búsqueda de un elemento por su id)

nombreQueryGet(atributoId: ID!):TipoDeDato1

## definir mutaciones querys

Las mutaciones son querys que de alguna formar van a modificar los datos

type Mutation {

        addTipo1(

            atributo1:String!

            atributo2:String!

        ):TipoDeDato1

    }

# Definir Resolvers

const resolvers ={

    Query:{

// consultar un bd y retornar esto

        tipos1:()=>datosDeTipos1,

tipo:(parent,args,context,info)=>{

            const id=parseInt(args.id);

            return datosDeTipos1.find((dato)=>dato.id===id);

        }

    }

}

De la manera anterior es si la forma en que están definidos los datos en los typeDefs es igual a como están en los origines de los datos (datosDeTipos1)

En estos es donde se realizan las búsquedas por querys, básicamente hay que implementar cada query

# Declarar el servidor

import { ApolloServer } from "@apollo/server"

import { startStandaloneServer} from '@apollo/server/standalone'

const server =new ApolloServer({

    typeDefs,resolvers

});

const { url } =await startStandaloneServer(server,{

    listen:{port:4000}

})

Va ha estar en http://localhost:4000/

# Lenguaje Petición

Se pueden pedir tipos de datos(en la forma de atributos definidos dentro de resolvers en Query ) ya definidos y de estos atributos ya definidos

## Para pedir una lista

query{

tipos1 {

atributo1

atributo2

}

}

## Para pedir un solo elemento por su id

query  nombreQurey {

  tipo(atributoId:1) {

    atributo1

    atributo2

  }

}

## Variables en querys

Para ponerle variables a las querys para poder usarlas después es declarándolas hantes del cuerpo {} de la query y después del nombre si tiene $nombreVariable: Tipo

query qurey($variable:ID!) {

  tipo(atributoId:$variable) {

    atributo1

    atributo2

  }

}

Esto se ejecuta unido a un json que tiene las variables con sus valores en esta consulta

{

$variable: valor

}