Guía GraphQL

¿Qué es?

- Es una especificación **declarativa** para poder obtener datos.
- Provee una interfaz compartida entre un cliente y un servidor para manejo de datos.

Preparando proyecto

```
$ npm init -y
```

Instalamos dependencias

```
$ npm install type-graphql --save
$ npm install typescript @types/node --save-dev
$ npm install reflect-metadata --save
$ npm install @types/ws --save
```

Creamos el siguiente tsconfig.json en la raíz de nuestro proyecto.

```
// tsconfig.json

{
    "compilerOptions": {
      "target": "es2016",
      "module": "commonjs",
      "lib": ["dom", "es2016", "esnext.asynciterable"],
      "moduleResolution": "node",
      "outDir": "./dist",
      "strict": true,
      "strictPropertyInitialization": false,
      "sourceMap": true,
      "emitDecoratorMetadata": true,
      "experimentalDecorators": true
    },
    "include": ["./src/**/*"]
  }
```

Creamos un directorio src y dentro de este directorio, dos directorios más con nombres schemas y data.

Definiendo GrapQL schema

El graphql schema se encarga de definir las entidades que se van a exponer en los servicios graphql.

Creamos dos schemas: Equipo y Calificacion

```
// schemas/Equipo.ts

import { Field, Int, ObjectType } from "type-graphql";

import Calificacion from "./Calificacion"

@ObjectType()
export default class Equipo
{
    @Field(type => Int)
    id : number

    @Field()
    nombre : string

    @Field(type => [Calificacion])
    calificaciones : Calificacion[]
}
```

```
// schemas/Calificacion.ts

import { Field, Int, ObjectType } from "type-graphql";
import Equipo from "./Equipo"

@ObjectType()
export default class Calificacion
{
    @Field(type => Int)
    id : number

    @Field()
    nombre : string

    @Field(type => Equipo)
    equipo : Equipo

    @Field(type => Int)
    nota : number
}
```

Utilizamos los siguientes decorators:

- **ObjectType**: Indica que una clase va a representar a una entidad en graphql.
- **Field**: Indica que cierta propiedad va a ser un campo de la entidad graphql.
- **Int**: Indica el tipo del campo. Nota que en caso que sea un string puede dejar vacio.

Definiendo el acceso a datos

Creamos los tipos de las entidades. Tomar en cuenta que estos tipos serán usados en las comunicaciones del cliente. Para esto se creará el archivo types.ts en una nueva carpeta llamada data dentro de la carpeta src.

```
// data/types.ts

export interface EquipoData
{
    id : number
    nombre : string
}

export interface CalificacionData
{
    id : number
    nombre : string
    equipoId : number
    nota : number
}
```

En esta guía no se está tomando en cuenta el acceso a una base de datos, por lo que solamente se definirán los datos de manera estática. Para esto crearemos el archivo data.ts dentro de la carpeta data anteriormente creada.

```
// data/data.ts

import { EquipoData, CalificacionData } from "./types";

export const equipos : EquipoData[] = [
    { id : 1, nombre : "Los compilas"},
    { id : 2, nombre : "Dale U"}
]

export const calificaciones : CalificacionData[] = [
    {id : 1, nombre : "E1", equipoId : 1, nota : 10},
    {id : 2, nombre : "E1", equipoId : 2, nota : 20},
    {id : 3, nombre : "E2", equipoId : 1, nota : 12},
    {id : 4, nombre : "E2", equipoId : 2, nota : 19},
    {id : 5, nombre : "E3", equipoId : 1, nota : 11},
]
```

Creación de los Resolvers

Los resolvers son clases que se encargarán de definir los servicios que serán utilizados por el cliente.

Crearemos la carpeta resolvers dentro de src y también añadiremos el archivo EquipoResolver.ts

```
// resolvers/EquipoResolver.ts

import { Arg, FieldResolver, Query, Resolver, Root } from "type-graphql";
import {EquipoData, CalificacionData} from "../data/types"
import Equipo from "../schemas/Equipo"
import { equipos, calificaciones } from "../data/data";

@Resolver(of => Equipo)
export default class EquipoResolver
{
    @Query(returns => Equipo, {nullable : true})
    equipoPorNombre(@Arg("nombre") name : string) : EquipoData | undefined
    {
        // Codigo para obtener los datos
        return equipos.find(equipo => {
            return equipo.nombre === name
        })
    }

    @Query(returns => [Equipo], {nullable : true})
    getEquipos() : EquipoData[] | undefined
    {
        return equipos
    }

    @FieldResolver()
    calificaciones(@Root() equipoData : EquipoData)
    {
        return calificaciones.filter( calif => {
            return calif.equipoId == equipoData.id
        })
    }
}
```

Como se ve, definimos una clase (con sus decoradores respectivos) así como los métodos que representarán los servicios. Tomar en cuenta que se están decorando los métodos con los siguientes decoradores:

- **Query**: Significa que el método va a ser una consulta de datos.
- **Arg**: Se encarga de definir que el servicio necesita un parámetro y le asigna un nombre.
- **FieldResolver**: Significa que el método va a definir la manera cómo se van a obtener los datos de un campo (en este ejemplo, el campo calificaciones)

Luego definimos el siguiente resolver para la entidad Calificacion.

```
// resolvers/CalificacionResolver.ts

import {Resolver, Query, Arg, FieldResolver, Root, Mutation, Args } from
"type-graphql"
import {CalificacionData} from "../data/types"
```

```
import Calificacion from "../schemas/Calificacion"
import { calificaciones, equipos } from "../data/data"

@Resolver(of => Calificacion)
export default class CalificacionResolver
{
    @Query(returns => [Calificacion], {nullable : true})
    getCalificacionesByNombre(@Arg("nombreCalificacion") nombreCalificacion
: string) : CalificacionData[] | undefined
    {
        // Obtenemos las calificaciones por nombre de calificacion
        return calificaciones.filter(calif => {
            return calif.nombre == nombreCalificacion
        })
    }

    @Mutation(returns => Calificacion)
    ponerNota(@Arg("calificacionId") calificacionId : number, @Arg("nota")
nota : number) : CalificacionData
    {
        const calificacion = calificaciones.find(calif => {
            return calif.id == calificacionId
        })

        // Validamos que no se encuentre una calificacion
        if (!calificacion)
        {
            throw new Error(`No se pudo encontrar una calificacion con el
id: ${calificacionId}`)
        }

        calificacion.nota = nota

        return calificacion
    }

    @FieldResolver()
    equipo(@Root() calificacionData : CalificacionData)
    {
        // Obtenemos el equipo que se tenia definido
        return equipos.find(equipo => equipo.id ==
calificacionData.equipoId)
    }

}
```

En este caso, se ha utilizado un nuevo decorador para el servicio **ponerNota**. Este decorador Mutation indica que este servicio va a realizar un cambio en la data.

Despliegue de servidor Graphql

Instalamos el siguiente servidor Graphql

```
$ npm install graphql-yoga --save-dev
```

Ejecutando servidor

Compilamos nuestro programa. Como se ha definido en el tsconfig.json, el compilado se guardará en la carpeta dist.

```
$ tsc
```

Ejecutamos nuestro servidor

```
$ node ./dist/index.js
```

Ejemplos de querys

Obtenemos la data del equipo de cierto nombre:

```
{
  equipoPorNombre(nombre : "Dale U"){
    id
    nombre
    calificaciones {
      id
    }
  }
}
```

Obtenemos todos los equipos

```
{
  getEquipos {
    id
    nombre
    calificaciones {
      id
    }
  }
}
```

Obtener todas las calificaciones dada un nombre

```
{
  getCalificacionesByNombre(nombreCalificacion : "E1") {
    id
    nombre
    nota
    equipo {
      id
      nombre
    }
  }
}
```

Modificar la nota de una calificacion.

```
mutation {
  ponerNota(nota : 11, calificacionId : 1) {
    id
    nombre
    nota
  }
}
```

Referencias

- https://graphql.org/
- https://blog.pusher.com/getting-up-and-running-with-graphql/
- https://pusher.com/tutorials/graphql-typescript
- https://github.com/prisma-labs/graphql-yoga/blob/master/README.md