NEST MicroServices GraphQL NOTES U2B

- 16:20 config graphql ms

FIRST STEPS

> nest new

- gateway - npm
- primero haremos un monorepo y lo transformaremos a microservicios
- Entro en el directorio de gateway

> nest g app

- users
- Ahora tengo dentro de apps gateway y users
- Vuelvo a usar nest g app, ahora le indico que cree posts
- Ahora, dentro de gateway uso

> nest g resource

- Elijo users, lo llamaré users y elijo GraphQL code first, genero els endpoints del CRUD
- Esto me genera un boilerplate
- Selecciono todos los archivos generados en users/src/users y los copio en src, reemplazando los anteriores (subo un nivel) y borro el users desde done los he copiado, solo dejo el main y tsconfig.app
- Hago lo mismo con posts
- Copio el tsconfig.app.json de gateway dentro de users y posts, en la raíz
- Hago las instalaciones necesarias en gateway

> npm i @apollo/gateway @apollo/subgraph @nestjs/apollo @nestjs/graphql apollo-server-express graphql

- Configuremos el users.module para usar Apollo

```
import { Module } from '@nestjs/common';
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloFederationDriver, ApolloFederationDriverConfig } from
'@nestjs/apollo';

@Module({
  imports:[
    GraphQLModule.forRoot<ApolloFederationDriverConfig>({
        driver: ApolloFederationDriver,
        autoSchemaFile:{
          federation: 2
        }
    })
```

```
  ],
  providers: [UsersResolver, UsersService],
})
export class UsersModule {}
```

> npm start:dev users

- Hago lo mismo para posts.module (configurar Apollo)
- Para este ejemplo no se hará el update ni el delete del user, por lo que puedo borrarlo y tampoco el update.dto
- Creamos la entidad de User, uso @ObjectType

```
import { ObjectType, Field, ID } from '@nestjs/graphql';

@ObjectType()
export class User {
  @Field(()=> ID)
  id: string

  @Field(()=> String)
  email: string

  @Field(()=> String)
  password: string

}
```

- Pongo las mismas propiedades en el input DTO, uso @InputType

```
import { InputType, Int, Field, ID } from '@nestjs/graphql';

@InputType()
export class CreateUserInput {

  @Field(()=> ID)
  id: string

  @Field(()=> String)
  email: string

  @Field(()=> String)
  password: string

}
```

- En usersService creo un array (no haremnos conexión con la DB)

```
@Injectable()
export class UsersService {

 private readonly users: User[] = []

  create(createUserInput: CreateUserInput) {
     this.users.push(createUserInput)
     return createUserInput
  }

  findAll() {
    return this.users
  }

  findOne(id: string) {
    return this.users.find(user => user.id === id)
  }

}
```

- En el resolver, uso @Resolver
- uso @Mutation cuando modifico data en el método
- En el objeto con la propiedad name, especifico como llamaré al query desde apollo
- Con @Args tomo los argumentos que serán pasados en la query desde apollo

```
import { Resolver, Query, Mutation, Args, Int } from '@nestjs/graphql';
import { UsersService } from './users.service';
import { User } from './entities/user.entity';
import { CreateUserInput } from './dto/create-user.input';
import { UpdateUserInput } from './dto/update-user.input';

@Resolver(() => User)
export class UsersResolver {
  constructor(private readonly usersService: UsersService) {}

  @Mutation(() => User)
  createUser(@Args('createUserInput') createUserInput: CreateUserInput) {
    return this.usersService.create(createUserInput);
  }

  @Query(() => [User], { name: 'users' })
  findAll() {
    return this.usersService.findAll();
  }

  @Query(() => User, { name: 'user' })
  findOne(@Args('id', { type: () => String }) id: string) {
    return this.usersService.findOne(id);
  }
}
```

- Para usar una mutación para crear un usuario, le paso la data al createUserInput como tipo dentro de un objeto
- Le pido que me devuelva id, email, password

```
mutation{
  createUser(createUserInput:{id:"1234", email:"alasjkas@ñasjkaksj.com", password:
"1234"}){
    id
    email
    password
  }
}
```

- Para consultar un user por id

```
query{
  user(id:"1234"){
    id
    email
  }
}
```

- Creo la entidad de Post, usando @ObjectType

```
import { ObjectType, Field, Int } from '@nestjs/graphql';

@ObjectType()
export class Post {
  @Field(() => Int)
  id: string;

  @Field()
  body: string;

  @Field()
  authorId: string;
}
```

- Copio las propiedades en el dto

```
import { ObjectType, Field, Int } from '@nestjs/graphql';

@ObjectType()
export class Post {
```

```
  @Field()
  id: string;

  @Field()
  body: string;

  @Field()
  authorId: string;
}
```

- En el posts.resolver

```
import { Resolver, Query, Mutation, Args, Int } from '@nestjs/graphql';
import { PostsService } from './posts.service';
import { Post } from './entities/post.entity';
import { CreatePostInput } from './dto/create-post.input';


@Resolver(() => Post)
export class PostsResolver {
  constructor(private readonly postsService: PostsService) {}

  @Mutation(() => Post)
  createPost(@Args('createPostInput') createPostInput: CreatePostInput) {
    return this.postsService.create(createPostInput);
  }

  @Query(() => [Post], { name: 'posts' })
  findAll() {
    return this.postsService.findAll();
  }

  @Query(() => Post, { name: 'post' })
  findOne(@Args('id', { type: () => String }) id: string) {
    return this.postsService.findOne(id);
  }

}
```

- El posts.service

```
import { Injectable } from '@nestjs/common';
import { CreatePostInput } from './dto/create-post.input';
import { UpdatePostInput } from './dto/update-post.input';
import { Post } from './entities/post.entity';

@Injectable()
export class PostsService {
```

```
  private readonly posts : Post[] =[]

  create(createPostInput: CreatePostInput) {
    this.posts.push(createPostInput)
    return createPostInput //aquí retornariámos el valor desde la db, en todo caso
  }

  findAll() {
    return this.posts;
  }

  findOne(id: string) {
    return this.posts.find(post => post.id === id);
  }

}
```

- Las querys y mutation serían similares que las de users
- Vamos a conectar los microservicios desde app.module en el módulo de gateway

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloGatewayDriver, ApolloGatewayDriverConfig } from '@nestjs/apollo';
import { IntrospectAndCompose } from '@apollo/gateway';

@Module({
  imports: [
    GraphQLModule.forRoot<ApolloGatewayDriverConfig>({
      driver: ApolloGatewayDriver,
      server: {},
      gateway:{
        supergraphSdl: new IntrospectAndCompose({
          subgraphs: [
            {
              name:'users',
              url: 'http://localhost:3001/graphql'
            },
            {
              name:'posts',
              url: 'http://localhost:3002/graphql'
            }
          ]
        })
      }


    })
  ],
  controllers: [AppController],
```

```
  providers: [AppService],
})
export class AppModule {}
```

- En gateway

> npm run start:dev npm run start:dev users npm run start:dev posts

---

Obtener post de usuario y usuario de post

- En user.entity usaré @Directive y especificaré en la key el campo id

```
import { ObjectType, Field, ID, Directive } from '@nestjs/graphql';

@ObjectType()
@Directive('@key(fields: "id")')
export class User {
  @Field(()=> ID)
  id: string

  @Field(()=> String)
  email: string

  @Field(()=> String)
  password: string

}
```

- En post.entity indico el usuario

```
import { ObjectType, Field, Int } from '@nestjs/graphql';
import { User } from './users.entity';

@ObjectType()
export class Post {
  @Field(() => Int)
  id: string;

  @Field()
  body: string;

  @Field()
  authorId: string;

  @Field(()=> User)
  user?: User
}
```

- Uso @ResolveReference en users.resolver para usar reference.id y encontrar el usuario

```
@ResolveReference()
resolveReference(reference: { __typename: string, id: string}): User{
  return this.usersService.findOne(reference.id)
}
```

- En el microservicio de posts creo otra entidad de users.entity

```
import { ObjectType, Field, ID, Directive } from '@nestjs/graphql';
import { Post } from './post.entity';

@ObjectType()
@Directive('@key(fields: "id")')
export class User {
  @Field(()=> ID)
  id: string

  @Field(()=> [Post])
   posts?: Post[]

}
```

- En el posts.resolver uso @ResolveField para obtener el user por el Post

```
import { Resolver, Query, Mutation, Args, Int, ResolveField, Parent } from
'@nestjs/graphql';
import { PostsService } from './posts.service';
import { Post } from './entities/post.entity';
import { CreatePostInput } from './dto/create-post.input';
import { User } from './entities/users.entity';


@Resolver(() => Post)
export class PostsResolver {
  constructor(private readonly postsService: PostsService) {}

  @Mutation(() => Post)
  createPost(@Args('createPostInput') createPostInput: CreatePostInput) {
    return this.postsService.create(createPostInput);
  }

  @Query(() => [Post], { name: 'posts' })
  findAll() {
    return this.postsService.findAll();
  }

  @Query(() => Post, { name: 'post' })
```

```
  findOne(@Args('id', { type: () => String }) id: string) {
    return this.postsService.findOne(id);
  }

  @ResolveField(()=> User)
  user(@Parent() post: Post): any {
    return {typename: 'User', id: post.authorId}
  }
}
```

- Creo también un users.resolver en el microservicio de posts

```
import { Parent, ResolveField, Resolver } from "@nestjs/graphql";
import { User } from "./entities/users.entity";
import { PostsService } from "./posts.service";
import { Post } from "./entities/post.entity";

@Resolver(()=>User)
export class UsersResolver{
    constructor(private readonly postsService: PostsService){
    }

    @ResolveField(()=>[Post])
    posts(@Parent() user: User) : Post[]{
        return this.postsService.forAuthor(user.id)
    }

}
```

- Añado el UsersResolver en providers del post.module

```
import { Module } from '@nestjs/common';
import { PostsService } from './posts.service';
import { PostsResolver } from './posts.resolver';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloFederationDriver, ApolloFederationDriverConfig } from
'@nestjs/apollo';
import { UsersResolver } from './users.resolver';

@Module({
  imports:[
    GraphQLModule.forRoot<ApolloFederationDriverConfig>({
        driver: ApolloFederationDriver,
        autoSchemaFile:{
          federation: 2
        }
    })
  ],
  providers: [PostsResolver, PostsService, UsersResolver],
```

```
})
export class PostsModule {}
```

- Creo el método en el posts.service

```
import { Injectable } from '@nestjs/common';
import { CreatePostInput } from './dto/create-post.input';

import { Post } from './entities/post.entity';
import { User } from './entities/users.entity';

@Injectable()
export class PostsService {

  private readonly posts : Post[] =[]

  create(createPostInput: CreatePostInput) {
    this.posts.push(createPostInput)
    return this.posts
  }

  findAll() {
    return this.posts;
  }

  findOne(id: string) {
    return this.posts.find(post => post.id === id);
  }

  forAuthor(authorId: string){
    return this.posts.filter(post=> post.authorId === authorId) //uso filter, no
tengo disponible el find
  }
}
```

- Si ahora miro en el SCHEMA del playground en localhgost:3000/graphql puedo ver como posts tiene user user tiene posts

```
# Indicates exactly one field must be supplied and this field must not be `null`.
directive @oneOf on INPUT_OBJECT

input CreatePostInput {
  id: Int!
  body: String!
  authorId: String!
}

input CreateUserInput {
  id: ID!
```

```
  email: String!
  password: String!
}

type Mutation {
  createPost(createPostInput: CreatePostInput!): Post!
  createUser(createUserInput: CreateUserInput!): User!
  removeUser(id: Int!): User!
}

type Post {
  id: Int!
  body: String!
  authorId: String!
  user: User!
}

type Query {
  posts: [Post!]!
  post(id: String!): Post!
  users: [User!]!
  user(id: String!): User!
}

type User {
  id: ID!
  posts: [Post!]!
  email: String!
  password: String!
}
```

- Para crear un usuario desde el playground de Apollo

```
mutation {
  createUser(createUserInput: { id:"123", email: "test@mail.com", password:
"test"}){
    id
    email
    password
  }
}
```

- Para crear un Post debo pasarle el authorId

```
mutation {
  createPost(createPostInput: { id:"234", authorId: "1234", body: "my message"}){

    body
```

```
  }
}
```

- Puedo obtener los posts de los usuarios desde el query de usuarios

```
query {
  users{
    id
    email
    password
    posts{
      id
      authorId
    }
  }
}
```

- Lo mismo users desde posts

Autenticación

- Para implementar la autenticación iremos al app.module de gateway incluiremos una nueva propiedad de contexto
- Para ello crearemos dentro del src de gateway auth.context.ts
- Asumiremos que hemos validado el jwt y obtenemos el user.id

```
import { UnauthorizedException } from "@nestjs/common"

export const authContext = ({req})=>{
    if(req.headers?.authorization){

        //validate jwt
        return {user: {id:'123'}}
     }
        throw new UnauthorizedException('user not valid')

}
```

- En server de app.modeul en gateway añadiremos el context
- Si ahora hago la query me devuelve un error de "Context creation failed: user not valid"
- En el playground de Apollo debo añadir en HTTP Headers Authorization con el id correspondiente

```
{"Authorization": "123"}
```

- Usaremos buildService desestructurando la url, y devolveremos una nueva instancia de remoteGraphQLDataSource a la que le pasaremos la url y la funcion willSendrequest de la que obtendremos la request y el context, con el objeto que emos expuesto en auth.context que era el user: id: "123"

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloGatewayDriver, ApolloGatewayDriverConfig } from '@nestjs/apollo';
import { IntrospectAndCompose, RemoteGraphQLDataSource } from '@apollo/gateway';
import { authContext } from './auth.context';

@Module({
  imports: [
    GraphQLModule.forRoot<ApolloGatewayDriverConfig>({
      driver: ApolloGatewayDriver,
      server: {
        context: authContext
      },
      gateway:{
        supergraphSdl: new IntrospectAndCompose({
          subgraphs: [
            {
              name:'users',
              url: 'http://localhost:3001/graphql'
            },
            {
              name:'posts',
              url: 'http://localhost:3002/graphql'
            }
          ]
        }),
        buildService({url}){
          return new RemoteGraphQLDataSource({
            url,
            willSendRequest({request, context}){
              request.http.headers.set(
                'user', //setearemos un nuevo user
                context.user? JSON.stringify(context.user): null
              )
            }
          })
        }
      }


    })
  ],
  controllers: [AppController],
  providers: [AppService],
```

```
})
export class AppModule {}
```

- Para securizar una ruta crearemos un nuevo decorador, lo llamaremos create-user.decorator

```
import { createParamDecorator } from "@nestjs/common";
import { GraphQLExecutionContext } from "@nestjs/graphql";

export const currentUserDecorator = createParamDecorator(
    (_data: any, ctx: GraphQLExecutionContext)=>{
        try {
            const headers = ctx.getArgs()[2].req.headers

            if(headers.user){
                JSON.parse(headers.user)
            }

        } catch (error) {
            return null
        }
    }
)
```

- Para usarlo, voy a posts.resolver

```
  @Query(() => [Post], { name: 'posts' })
  findAll(@CurrentUser() user: User) {
    console.log(user)
    return this.postsService.findAll();
  }
```