01 Nest GraphQL - Primeros pasos

- GraphQL me servirá para que el frontend se autoabastezca de lo que necesite haciendo una petición a un único endpoint
- *LINKS DE INTERÉS*
  - graphql.org
  - docs.nestjs.com/graphql
  - fireship.io
- GraphQL **es un lenguaje para leer y mutar data mediante APIs**. Un **Query Language**
- Es agnóstico
- Puedo tipar objetos para que el frontend demande la información que necesita de la manera correcta
- Puedo mezclar el único endpoint de graphQL con unb API REST
- GraphQL hace un query y el backend devuelve la data. Hay menos trabajo en el backend (de código)
- Hay dos formas de trabajar NEST + GraphQL
  - **Schema First** - Es la manera tradicional de GraphQL para la creación de schemas.º
  - **Code First** - Creamos las clases y definiciones en TS y esto automáticamente nos creará el SDL (Schema Definition Language)
- Todo graphQL endpoint tiene al menos un tipo **Query definido**
- Para realizar cambios en la data, existe el tipo **Mutation** el cual sirve para **mutar la data**
- Si habilitamos la opción visual (recomendable) podemos seleccionar campos, la data, y generar el query
- Muy útil para trabajar con autenticación, para validar JWT, etc
- Sabiendo NEST; solo hay que conocer **un par de decoradores** y **un par de conceptos propios de GraphQL**

Proyecto NEST GraphQL

- Aplicación de TODOS
- Si trabajas con el código fuente será necesario actualizar algunos paquetes borrando los del package.json

> npm i @nestjs/apollo @nestjs/common @nestjs/core @nestjs/graphql @nestjs/platform-express apollo-server-core apollo-server-express graphql

- Si generas el proyecto por tu cuenta solo necesitarás estos

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

- Nest trabaja por defecto sobre express, por eso apollo-server-express
- Hay que hacer la configuración del ApolloDriver
- Necesita al menos una consulta
- En el app.module

```
import { join } from 'path';
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
```

```
import { HelloWorldModule } from './hello-world/hello-world.module';
import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';


@Module({
  imports: [
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      autoSchemaFile: join(process.cwd(), 'src/schema.gql'), //process.cwd es la
carpeta donde se ejecuta el proyecto y le paso el schema
      playground: false, //en true habilita la interacción visual en
localhost:3000/graphql
      plugins: [

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

- Con el app.module así me salta un error que dice "Query root type must be provided"
- Creo el módulo HelloWorld nest mo HelloWorld
- hellowrorld.module

```
import { Module } from '@nestjs/common';
import { HelloWorldResolver } from './hello-world.resolver';

@Module({
  providers: [ HelloWorldResolver ]
})
export class HelloWorldModule {}
```

- helloWorld.resolver
- Uso el decorador Resolver
- Uso Query de /graphql (hay otro de /common)
- Tengo que decirle al Query lo que va a devolver. Coloco una función de flecha y le digo que será de tipo String
- En un objeto añado una descripción y el nombre de la query
- En getRandomFromZeroTo tengo @Args para el argumento que quiero recibir
- Hacemos la validación manual (en lugar de usar un dto)
    - Puede ser nulo
    - En type le digo que puede devolver un entero
    - En el valor de vuelta le digo que to es un number y le asignoi 6 por defecto

```
import { Float, Query, Resolver, Int, Args } from '@nestjs/graphql';

@Resolver()
export class HelloWorldResolver {

    @Query( () => String, { description: 'Hola Mundo es lo que retorna', name:
'hello' } )
    helloWorld(): string {
        return 'Hola Mundo';
    }

    @Query( () => Float, { name: 'randomNumber' } )
    getRandomNumber(): number {
        return Math.random() * 100;
    }

    // randomFromZeroTo
    @Query( () => Int, { name: 'randomFromZeroTo', description: 'From zero to
argument TO (default 6)' } )
    getRandomFromZeroTo(
        @Args('to', { nullable: true, type: () => Int } ) to: number = 6
    ): number {
        return Math.floor( Math.random() * to );
    }

}
```

- Creando este resolver "mágicamente" (con el servidor levantado) apareció el schema.gql
- No se modifica jamás (se sobreescribe continuamente)

```
# ------------------------------------------------------
# THIS FILE WAS AUTOMATICALLY GENERATED (DO NOT MODIFY)
# ------------------------------------------------------

type Query {
  """Hola Mundo es lo que retorna"""
  hello: String!
  randomNumber: Float!

  """From zero to argument TO (default 6)"""
  randomFromZeroTo(to: Int): Int!
}
```

- El endpoint por defecto es http://localhost:3000/graphql donde voy a tener el apollo-server en el que escribir querys
- Para poder visualizarlo tiene que estar el playground en true en el app.module
- Escribo mi primera query desde el navegador

```
query{
    helloWorld   //le paso el nombre de la Query (de la función que usé como
Query)
}
```

- Esto nos devuelve

```
{
    "data":{
        "helloWorld": "Hola Mundo"
    }
}
```

- Puedo renombrar la consulta (es lo que mandaría desde el frontend)

```
query{
    hola: helloWorld
}
```

- Que me retornaría hola: "Hola Mundo"
- En el playground tengo en un lateral los Docs con las querys que tengo armadas
- Al haber puesto en el objeto la description y el name, puedo usar el string del name para hacer la query

```
 @Query( () => String, { description: 'Hola Mundo es lo que retorna', name:
'hello' } )
    helloWorld(): string {
        return 'Hola Mundo';
    }
```

- La query

```
query{
    hello
}
```

- Con POSTMAN no tenemos la mejor de las interfaces
- Trabajaremos con **Apollo Studio** que tiene un montón de funciones interesantes, reemplazando el playground por defecto en graphql
- npm i apollo-server-core
- Me ha dado problemas con el tipado en el forRoot, lo quito y añado ApolloServerPluginLandingPageLocalDefault()

```
import { join } from 'path';
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloDriver} from '@nestjs/apollo';
import { HelloWorldModule } from './hello-world/hello-world.module';
import {ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';


@Module({
  imports: [
    GraphQLModule.forRoot({
      driver: ApolloDriver,
      autoSchemaFile: join(process.cwd(), 'src/schema.gql'),
      playground: false,
      plugins: [
        ApolloServerPluginLandingPageLocalDefault
      ]
    }),
    HelloWorldModule,
  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- En el mismo endpoint de 3000/graphql aparece apollo Studio en el browser
- Necesita conexión, puedes usar el playground también

2da PARTE

Todo Resolver y Custom Object Type

- Genero en la carpeta todo el todo.module, el todo.resolver(*nest g r todo*) el todo.service y las carpetas dto, entity y types
- Lo generaremos automáticamente, ahora lo hacemos manual
- El resolver no es más que una simple clase con el decorador @Resolver
- hacemos el CRUD entero a lo GraphQL
- todo.resolver
- Uso @Mutation cuando voy a mutar data

```
import { Resolver, Query, Args, Int, Mutation } from '@nestjs/graphql';
import { TodoService } from './todo.service';
import { Todo } from './entity/todo.entity';

import { CreateTodoInput, UpdateTodoInput, StatusArgs } from './dto';
import { AggregationsType } from './types/aggregations.type';
```

```
@Resolver( () => Todo )
export class TodoResolver {

    constructor(
        private readonly todoService: TodoService
    ){}


    @Query( () => [Todo], { name: 'todos' })
    findAll(
        @Args() statusArgs: StatusArgs
    ): Todo[] {
        return this.todoService.findAll( statusArgs );
    }

    @Query( () => Todo, { name: 'todo' })
    findOne(
        @Args('id', { type: () => Int } ) id: number
    ) {
        return this.todoService.findOne( id );
    }

    @Mutation( () => Todo, { name: 'createTodo' })
    createTodo(
        @Args('createTodoInput') createTodoInput: CreateTodoInput
    ) {
        return this.todoService.create( createTodoInput );
    }

    @Mutation( () => Todo, { name: 'updateTodo' })
    updateTodo(
        @Args('updateTodoInput') updateTodoInput: UpdateTodoInput
    ) {
        return this.todoService.update( updateTodoInput.id, updateTodoInput );
    }

    @Mutation( () => Boolean )
    removeTodo(
        @Args('id', { type: () => Int }) id: number
    ) {
        return this.todoService.delete( id );
    }


    // Aggregations
    @Query( () => Int, { name: 'totalTodos' })
    totalTodos(): number {
        return this.todoService.totalTodos;
    }

    @Query( () => Int, { name: 'pendingTodos' })
    pendingTodos(): number {
        return this.todoService.pendingTodos;
    }
```

```
    @Query( () => Int, { name: 'completedTodos' })
    completedTodos(): number {
        return this.todoService.completedTodos;
    }

    @Query( () => AggregationsType )
    aggregations(): AggregationsType {
        return {
            completed: this.todoService.completedTodos,
            pending: this.todoService.pendingTodos,
            total: this.todoService.totalTodos,
            totalTodosCompleted: this.todoService.totalTodos,
        }
    }
}
```

- El todo.service (como no usamos DB de momento el código es más complejo de lo que debería)
- Uso @Injectable porque lo inyectaré en el resolver

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { CreateTodoInput, UpdateTodoInput } from './dto/inputs';
import { Todo } from './entity/todo.entity';
import { StatusArgs } from './dto/args/status.args';

@Injectable()
export class TodoService {

    private todos: Todo[] = [
        { id: 1, description: 'Piedra del Alma', done: false },
        { id: 2, description: 'Piedra del Espacio', done: true },
        { id: 3, description: 'Piedra del Poder', done: false },
        { id: 4, description: 'Piedra del Tiempo', done: false },
    ];

    get totalTodos() {
        return this.todos.length;
    }

    get pendingTodos() {
        return this.todos.filter( todo => todo.done === false ).length;
    }

    get completedTodos() {
        return this.todos.filter( todo => todo.done === true ).length;
    }



    findAll(  statusArgs: StatusArgs ): Todo[] {
```

```
        const { status } = statusArgs;
        if( status !== undefined ) return this.todos.filter( todo => todo.done ===
status );

        return this.todos;
    }

    findOne( id: number ): Todo {

        const todo = this.todos.find( todo => todo.id === id );

        if ( !todo ) throw new NotFoundException(`Todo with id ${ id } not
found`);

        return todo;
    }

    create( createTodoInput: CreateTodoInput ): Todo {

        const todo = new Todo();
        todo.description = createTodoInput.description;
        todo.id = Math.max( ...this.todos.map( todo=> todo.id ), 0 ) + 1

        this.todos.push( todo );

        return todo;
    }


    update( id: number, updateTodoInput: UpdateTodoInput ) {
        const { description, done } = updateTodoInput;
        const todoToUpdate = this.findOne( id );

        if ( description ) todoToUpdate.description = description;
        if ( done !== undefined ) todoToUpdate.done = done;

        this.todos = this.todos.map( todo => {
            return ( todo.id === id ) ? todoToUpdate : todo;
        });

        return todoToUpdate;

    }

    delete( id: number ):Boolean {
        const todo = this.findOne( id );

        this.todos = this.todos.filter( todo => todo.id !== id );

        return true;
    }
}
```

- @ObjectType en lugar de @Entity para decirle que es mi objeto personalizado de graphQL
    - Puedo añadir en el mismo archivo @Entity para trabajar con mongoose
- Para crear un tipo personalizado acabaremos usando los tipos Int, Float, String, Boolean, ID
- Uso @ObjectType para definirlo como un tipo de graphQL
- Uso @Field para indicarle a graphQL el tipo del campo
- entity/todo.entity

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

@ObjectType()
export class Todo {

    @Field( () => Int )
    id: number;

    @Field( () => String )
    description: string;

    @Field( () => Boolean )
    done: boolean = false;

}
```

- Si quiero consultar las descripciones de todos uso todos

```
query{
    todos {description}
}
```

- Si quiero cambiearle el nombre a tareas

```
query{
    tareas:todos {
        description
        }
}
```

- Cuando consulto el schema en el playground, si aparece Int! es que SIEMPRE voy a recibir un Int
- Lo mismo con los argumentos. Si no pongo el **nullable:true** es que el argumento será obligatorio y aparecerá argumento:Int! (si es un entero)
- Para usar los argumentos (en todo por id, por ejemplo) y retornar algo en especifico
- resolver

```
@Query( () => Todo, { name: 'todo' })
findOne(
```

```
    @Args('id', { type: () => Int } ) id: number
) {
    return this.todoService.findOne( id );
}
```

- Para hacer la consulta le indico el id. Necesito especificarle los campos

```
{
    todo(id:1){
        id
        description
        done
    }
}
```

- Para comentar un campo en la petición (haciendo puebas) uso #
- Puedo dividir el query en varios todos

```
{
  todo1: todo(id:1){
    description
  }
  todo2: todo(id:2){
    description
    done
  }

}
```

- Para no tener que repetir todos los campos (id, description, done) de todos los todos por id que quiero recibir usaré fragments
- Son unidades reutilizables para hacer grupos de campos
- Se escriben fuera del query, se usa con el spread en los campos

```
{
  todo1: todo(id:1){
    ...fields
  }
  todo2: todo(id:2){
    ...fields
  }

}

  fragment fields on Todo {
    description
    done
```

```
    id
  }
```

Mutation e inputs

- Las **MUTATIONS** son querys que sirven para modificar la data almacenada y retornar un valor
- El **tipo Input** en una mutación, es la información que llamariamos body en una petición REST tradicional
- En la mutation, le paso lo mismo, el tipo de retrono y el nombre

```
@Mutation( () => Todo, { name: 'createTodo' })
createTodo(
    @Args('createTodoInput') createTodoInput: CreateTodoInput
) {
    return this.todoService.create( createTodoInput );
}
```

- En dtos/inputs/

```
import { Field, InputType } from '@nestjs/graphql';
import { IsNotEmpty, IsString, MaxLength } from 'class-validator';

@InputType()
export class CreateTodoInput {

    @Field( () => String, { description: 'What needs to be done' })
    @IsString()
    @IsNotEmpty()
    @MaxLength(20)
    description: string;
}
```

Filtros

- Para agregar **FILTROS**, por ejemplo al findAll

```
@Query( () => [Todo], { name: 'todos' })
    findAll(
        @Args() statusArgs: StatusArgs
    ): Todo[] {
        return this.todoService.findAll( statusArgs );
    }
```

- dtos/statusArgs

```
import { ArgsType, Field } from "@nestjs/graphql";
import { IsBoolean, IsOptional } from "class-validator";

@ArgsType()
export class StatusArgs {

    @Field( () => Boolean, { nullable: true })
    @IsOptional()
    @IsBoolean()
    status?: boolean;

}
```

- El query sería

```
{
  pending: todos(status:false){
    ...fields
  }

  completed: todos(status:true){
    ...fields
  }

}

  fragment fields on Todo{
    description
    done
    id
  }
```

- De esta manera en una sola petición tengo los todos en un json dentro de data en dos arreglos diferentes, pending y completed

```
{
  "data": {
    "pending": [
      {
        "description": "Piedra del Alma",
        "done": false,
        "id": 1
      },
      {
        "description": "Piedra del Poder",
        "done": false,
```

```
        "id": 3
      },
      {
        "description": "Piedra del Tiempo",
        "done": false,
        "id": 4
      }
    ],
    "completed": [
      {
        "description": "Piedra del Espacio",
        "done": true,
        "id": 2
      }
    ]
  }
}
```

Agregar conteos como campos adicionales

- Más adelante trabajando con la DB veremos la paginación
- Ahora veremos cómo saber la cantidad de todos totales, pendientes...
- Creo los nuevos Query en el resolver que regresarán un Int

```
// Aggregations
@Query( () => Int, { name: 'totalTodos' })
totalTodos(): number {
    return this.todoService.totalTodos;
}

@Query( () => Int, { name: 'pendingTodos' })
pendingTodos(): number {
    return this.todoService.pendingTodos;
}

@Query( () => Int, { name: 'completedTodos' })
completedTodos(): number {
    return this.todoService.completedTodos;
}
```

- En el servicio creo los getters

```
get totalTodos() {
    return this.todos.length;
}

get pendingTodos() {
    return this.todos.filter( todo => todo.done === false ).length;
```

```
}

get completedTodos() {
    return this.todos.filter( todo => todo.done === true ).length;
}
```

- Puedo hacer el query así

```
{
totalTodos
  completedTodos
  todos(status:true){
    ...fields
  }

}

  fragment fields on Todo{
    description
    done
    id
  }
```

ObjectTypes - Aggregations

-

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


@ObjectType({ description: 'Todo quick aggregations' })
export class AggregationsType {

    @Field( () => Int )
    total: number;

    @Field( () => Int )
    pending: number;

    @Field( () => Int )
    completed: number;

    @Field( () => Int, { deprecationReason: 'Most use completed instead' })
//ejemplo de warning por deprecado
    totalTodosCompleted: number;

}
```

- El totalTodosCompleted aparece en Apollo Studio con un warning. Si clico me dice que *is deprecated*
- El Query aggregations agrupa los getters, devuelve el objeto AggregationsType

```
@Query( () => AggregationsType )
aggregations(): AggregationsType {
    return {
        completed: this.todoService.completedTodos,
        pending: this.todoService.pendingTodos,
        total: this.todoService.totalTodos,
        totalTodosCompleted: this.todoService.totalTodos,
    }
}
```

- Para hacer la consulta

```
{
aggregations{
  completed
}
  todos(status:true){
    ...fields
  }

}

  fragment fields on Todo{
    description
    done
    id
  }
```

02 Nest GraphQL - AnyList (Postgres)

- Actualización del código

> npm i @nestjs/apollo @nestjs/common @nestjs/config @nestjs/core @nestjs/graphql @nestjs/platform-express @nestjs/typeorm apollo-server-core apollo-server-express class-transformer class-validator graphql typeorm pg

- Harermos un CRUD que impacte una DB con graphQL
- Todavía no hay autenticación ni paginación
- Necesito un Schema con al menos un query para poder levantar el server con graphQL
- Para la DB usaremos typeorm
- Si lleva la palabra module es que va en los imports
- En app.module

```
import { join } from 'path';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { ConfigModule } from '@nestjs/config';
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';
import { TypeOrmModule } from '@nestjs/typeorm';

import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';

import { ItemsModule } from './items/items.module';


@Module({
  imports: [

    ConfigModule.forRoot(),

    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      // debug: false,
      playground: true,
      autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
      plugins: [
        //ApolloServerPluginLandingPageLocalDefault
      ]
    }),

    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: +process.env.DB_PORT,
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      database: process.env.DB_NAME,
      synchronize: true,
      autoLoadEntities: true,
    }),

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

- En el main uso GlobalPipes para las validaciones

```
import { NestFactory } from '@nestjs/core';
import { ValidationPipe } from '@nestjs/common';
import { AppModule } from './app.module';
```

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
    whitelist: true,
    forbidNonWhitelisted: true,
    })
  );

  await app.listen(3000);
}
bootstrap();
```

- Creo un CRUD completo con **nest g res items**
    - Le indico GraphQL (code first)
    - Le digo que SI genere los endpoints (en este caso)
- Esto me crea la entidad con @ObjectType, el @InputTyoe para el updateItem, en el módulo me puso el ItemResolver y el ItemService en los providers, entre otras cosas...
- Al tener el resolver automáticamente crea el schema.gql necesario para echar a andar el server

---

Docker - Levantar base de datos

- docker-compose.yml
- *NOTA*: para ver la dfocumentacion acudir a **docker hub** y buscar postgres. Uso el puerto 5434 porque el 5432 cesta ocupado
- Si no encuentra la DB y está levantada en Docker, con el user y password correctos, cambia el puerto!

```
services:
  db:
    image: postgres:14.4
    restart: always
    ports:
      - "5434:5432"
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USERNAME: ${DB_USERNAME}
    container_name: anylistDB
    volumes:
      - ./postgres:/var/lib/postgresql/data
```

- Defino las .env

```
STATE=dev

DB_PASSWORD=123456
```

```
DB_NAME=AnyList
DB_HOST=localhost
DB_PORT=5434
DB_USERNAME=postgres
```

- Uso docker compose. Tiene que estar docker corriendo (si ya tienes la imagen es rápido)

> docker compose up -d

- -d es detouch (desacoplado de la terminal)
- Debe estar el ConfigModule.onRoot() en app.module para usar las variables de entorno

---

Item Entity

- src/items/entities/item.entity
- Combino los decoradores de typeorm y graphql

```
import { ObjectType, Field, ID, Float } from '@nestjs/graphql';
import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm';

@Entity({ name: 'items' })
@ObjectType()
export class Item {

  @PrimaryGeneratedColumn('uuid') //el tipo de id será uuid
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  name: string;

  @Column()
  @Field( () => Float )
  quantity: number;

  @Column({ nullable: true })
  @Field( () => String, { nullable: true } ) //al ser opcional puede ser null
  quantityUnits?: string; // g, ml, kg, tsp

  // stores
  // user
}
```

- En el imports de items.module importo la entidad

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common';
```

```
import { ItemsService } from './items.service';
import { ItemsResolver } from './items.resolver';
import { Item } from './entities/item.entity';

@Module({
  providers: [
    ItemsResolver,
    ItemsService
  ],
  imports: [
    TypeOrmModule.forFeature([ Item ])
  ]
})
export class ItemsModule {}
```

Crear items - Servicio y dtos

- Por ahora vamos a crear los usuarios sin autenticación. Más adelante se hará todo el módulo de autenticación
- items/dtos
- create-item.dto

```
import { InputType, Field, Float } from '@nestjs/graphql';
import { IsNotEmpty, IsOptional, IsPositive, IsString } from 'class-validator';

@InputType()
export class CreateItemInput {

  @Field( () => String )
  @IsNotEmpty()
  @IsString()
  name: string;

  @Field( () => Float )
  @IsPositive()
  quantity: number;

  @Field( () => String, { nullable: true })
  @IsString()
  @IsOptional()
  quantityUnits?: string;

}
```

- updateItem.dto

```
import { CreateItemInput } from './create-item.input';
import { InputType, Field, PartialType, ID } from '@nestjs/graphql';
import { IsUUID } from 'class-validator';

@InputType()
export class UpdateItemInput extends PartialType(CreateItemInput) {

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

}
```

- item.service
- Uso @InjectRepository de @nestjs/typeorm y le paso la entidad.
- Uso **Repository** de typeorm y le paso de tipo la entidad
- Los métodos son async al trabajar con una DB
- Al ser async devuelven una promesa. Especifico el tipo
- El servicio llama al repositorio que he inyectado para interactuar con la DB

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { CreateItemInput, UpdateItemInput } from './dto/inputs';
import { Item } from './entities/item.entity';

@Injectable()
export class ItemsService {

  constructor(
    @InjectRepository( Item )
    private readonly itemsRepository: Repository<Item>,

  ) {}


  async create( createItemInput: CreateItemInput ): Promise<Item> {
    const newItem = this.itemsRepository.create( createItemInput )
    return await this.itemsRepository.save( newItem );
  }

  async findAll(): Promise<Item[]> {
    // TODO: filtrar, paginar, por usuario...
    return this.itemsRepository.find();
  }

  async findOne( id: string ): Promise<Item> {
    const item = await this.itemsRepository.findOneBy({ id })

    if ( !item ) throw new NotFoundException(`Item with id: ${ id } not found`);
```

```
    return item;
  }

  async update(id: string, updateItemInput: UpdateItemInput): Promise<Item> {

    const item = await this.itemsRepository.preload( updateItemInput );

    if ( !item ) throw new NotFoundException(`Item with id: ${ id } not found`);

    return this.itemsRepository.save( item );

  }

  async remove( id: string ):Promise<Item> {
    // TODO: soft delete, integridad referencial
    const item = await this.findOne( id );
    await this.itemsRepository.remove( item );
    return { ...item, id };
  }
}
```

- Deberíamos crear un índice para que cada usuario solo pueda subir un item con el mismo nombre
- Que Fernando pudiera subir un item llamado uvas, que Melissa también, pero solo una vez
- Para crear la peticiópn, que sería POST en una PAI REST uso mutation (no query!)
- Hago la mutation desde el playground o Apollo Server
- Veamos en el resolver como está construido el create

```
@Mutation(() => Item) //la Mutation retorna un item
async createItem(
  @Args('createItemInput') createItemInput: CreateItemInput //como argumento le
paso el createItemInput
): Promise<Item> { //al ser async devuelve una promesa de tipo Item
  return this.itemsService.create(createItemInput); //llamo al servicio
}
```

- La query sería así

```
mutation CreateItem($createItemInput: CreateItemInput!){
  createItem(createItemInput: $createItemInput){
    id
    name
    quantity
    quantityUnits
  }
}
```

- En el apartado variables (en otra terminal abajo de la query) creo el json con el objeto createItemInput

```
{
  "createItemInput": {
    "name": "Cervezas",
    "quantity": 1
  }
}
```

- El create devuelve esto

```
{
  "data": {
    "createItem": {
      "id": "6b5f010e-233a-4858-9944-b50196ca64de",
      "name": "Cervezas",
      "quantity": 1,
      "quantityUnits": null
    }
  }
}
```

- En la query en items.resolver de findAll retorno una promesa de tipo arreglo de Item

```
@Query(() => [Item], { name: 'items' })
  async findAll(): Promise<Item[]> {
    return this.itemsService.findAll();
  }
```

- En la query de findAll, solo necesito pasarle items y los campos a devolver, ya que en el resolver llamé a la query items

```
{
  items{
    id
    name
    quantity
    quantityUnits
  }
}
```

- En el findOne recibimos el id de tipo string. En el resolver será de tipo ID y usaremos el pipe para validarlo
- Es un string al fin y al cabo

```
@Query(() => Item, { name: 'item' }) //llamaré item en la query
async findOne(
  @Args('id', { type: () => ID }, ParseUUIDPipe ) id: string
): Promise<Item> {
  return this.itemsService.findOne(id);
}
```

- En el servicio busco por el id, si no lo encuentra devuelvo una excepción

```
async findOne( id: string ): Promise<Item> {
    const item = await this.itemsRepository.findOneBy({ id })

    if ( !item ) throw new NotFoundException(`Item with id: ${ id } not found`);

    return item;
  }
```

- En la query le paso el id de tipo ID
- En la query puedo nombrar la variable que le paso a item como quiera, pero en item tengo que llamarla igual que la nombré en el resolver
- Luego le paso la variable que declaré en la query

```
               //aquí puedo llamarla como quiera
query QueryItem($idDelItem: ID!){
      //aqui tengo que llamarla igual que en el resolver
 item(id: $idDelItem){
    id
    name
    quantity
    quantityUnits

  }
}
```

- En el apartado de variables (abajo, en otra terminal del playground) le paso el id en un json
- Aquí tengo que pasarle el nombre de la variable de la query

```
{
    "idDelItem": "6b5f010e-233a-4858-9944-b50196ca64de"
}
```

- La respuesta me devuelve un json con el objeto data

```
{
  "data": {
    "item": {
      "id": "6b5f010e-233a-4858-9944-b50196ca64de",
      "name": "Cervezas",
      "quantity": 1,
      "quantityUnits": null
    }
  }
}
```

- Actualizar un item
- En el resolver le paso el dto de updateItem con el id y los campos actualizar
- Por supuesto es una mUtation ya que vamos a cambiar data

```
@Mutation(() => Item)
updateItem(
  @Args('updateItemInput') updateItemInput: UpdateItemInput
):Promise<Item> {
  return this.itemsService.update( updateItemInput.id, updateItemInput ); //le
paso el id que viene en el dto y el dto
}
```

- En el servicio uso .preload
- También podría usar el findOne para buscar y luego actualizar, pero .preload buscará por el id que viene en el dto y lo actualizará

```
async update(id: string, updateItemInput: UpdateItemInput): Promise<Item> {

  const item = await this.itemsRepository.preload( updateItemInput ); //preload
busca por el id y carga la entidad (como viene el id lo buscará)

  if ( !item ) throw new NotFoundException(`Item with id: ${ id } not found`);

  return this.itemsRepository.save( item );// si lo ha encontrado lo salvo

}
```

- Para eliminar, en el resolver recojo el id
- También es una mutation, claro

```
@Mutation(() => Item)
removeItem(
  @Args('id', { type: () => ID }) id: string
): Promise<Item> {
```

```
    return this.itemsService.remove(id);
  }
```

- En el service uso el findOne para encontrarlo y validar que exista, luego lo elimino
- Se buscará hacer un borrado lógico, dónde no se pierda la integridad referencial. Algo como active: false

```
async remove( id: string ):Promise<Item> {
  // TODO: soft delete
  const item = await this.findOne( id );
  await this.itemsRepository.remove( item );
  return { ...item, id };
}
```

03 Nest GraphQL - Autenticación

- Estamos creando una app para manejar listas
- El usuario solo podrá ver sus listas
- El administrador podrá ver todas las listas
- Este es el JSON, se ha añadido bcrypt, passport, passport-jwt, @nestjs/passport y @nestjs/jwt
- tambien instalo como dependencia de desarrollo @types/passport-jwt

```
{
  "name": "anylist",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "prebuild": "rimraf dist",
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-
node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/apollo": "^12.1.0",
```

```
    "@nestjs/common": "^10.3.9",
    "@nestjs/config": "^3.2.2",
    "@nestjs/core": "^10.3.9",
    "@nestjs/graphql": "^12.1.1",
    "@nestjs/platform-express": "^10.3.9",
    "@nestjs/testing": "^10.3.9",
    "@nestjs/typeorm": "^10.0.2",
    "apollo-server-core": "^3.13.0",
    "apollo-server-express": "^3.13.0",
    "bcrypt": "^5.1.0",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.1",
    "graphql": "^16.8.1",
    "passport": "^0.6.0",
    "passport-jwt": "^4.0.0",
    "pg": "^8.12.0",
    "reflect-metadata": "^0.1.13",
    "rimraf": "^3.0.2",
    "rxjs": "^7.2.0",
    "typeorm": "^0.3.20"
  },
  "devDependencies": {
    "@nestjs/cli": "^9.0.0",
    "@nestjs/schematics": "^9.0.0",
    "@types/bcrypt": "^5.0.0",
    "@types/express": "^4.17.13",
    "@types/jest": "28.1.8",
    "@types/node": "^16.0.0",
    "@types/passport-jwt": "^3.0.7",
    "@types/supertest": "^2.0.11",
    "@typescript-eslint/eslint-plugin": "^5.0.0",
    "@typescript-eslint/parser": "^5.0.0",
    "eslint": "^8.0.1",
    "jest": "28.1.3",
    "source-map-support": "^0.5.20",
    "supertest": "^6.1.3",
    "ts-jest": "28.0.8",
    "ts-loader": "^9.2.3",
    "ts-node": "^10.0.0",
    "tsconfig-paths": "4.1.0",
    "typescript": "^4.7.4"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "json",
      "ts"
    ],
    "rootDir": "src",
    "testRegex": ".*\\.spec\\.ts$",
    "transform": {
      "^.+\\.(t|j)s$": "ts-jest"
    },
    "collectCoverageFrom": [
```

```
      "**/*.(t|j)s"
    ],
    "coverageDirectory": "../coverage",
    "testEnvironment": "node"
  }
}
```

- Autenticación (saber quien es el usuario) y autorización (son los permisos que tiene el usuario, habrá ciertos querys bloqueados a ciertos roles)
- Signup y Login normalmente no está en graphQL, porque si no cualquier persona no autorizada tendrían acceso a los endpoints
- Se suele usar REST, u otros tipos de auth
- Crearemos custom decorators, haremos la autenticación, veremos las estrategias para logearnos, validar los tokens, las mutations
- Veremos como bloquear el schema en caso de no tener acceso

User Entity, resolver, Servicio y Auth

- El main está igual

```
import { ValidationPipe } from '@nestjs/common';
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      forbidNonWhitelisted: true,
    })
  );
  await app.listen(3000);
}
bootstrap();
```

- Creo los módulos de Auth (login, signin, revalidación del token) y User(para manejar los usuarios, más en plan admin)
- Veremos la dependencia cíclica de los módulos

> nest g res user

- GraphQl Code first, creo los endpoints
- En el app.module

```
import { join } from 'path';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { ConfigModule } from '@nestjs/config';
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';

import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';

import { ItemsModule } from './items/items.module';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UsersModule } from './users/users.module';
import { AuthModule } from './auth/auth.module';

@Module({
  imports: [
    ConfigModule.forRoot(),
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      playground: true,
      autoSchemaFile: join(process.cwd(), 'src/schema.gql'),
      plugins: [
        //ApolloServerPluginLandingPageLocalDefault
      ]
    }),
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: 'localhost',
      port: +process.env.DB_PORT,
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      database: process.env.DB_NAME,
      synchronize: true,
      autoLoadEntities: true,
    }),
    ItemsModule,
    UsersModule,
    AuthModule,
  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- User.entity queda así
- Aparte de ser una entidad, también es un ObjectType

```
import { ObjectType, Field, ID } from '@nestjs/graphql';
import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm';

@Entity({ name: 'users' })
@ObjectType()
```

```
export class User {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  fullName: string;

  @Column({ unique: true })
  @Field( () => String )
  email: string;

  @Column()
  // @Field(() => String)
  password: string;

  @Column({
    type: 'text',
    array: true,
    default: ['user']
  })
  @Field( () => [ String ])
  roles: string[];

  @Column({
    type: 'boolean',
    default: true
  })
  @Field( () => Boolean )
  isActive: boolean;

  //TODO: relaciones y otras cosas
}
```

- En user.module debo indicar la entidad con forFeature

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common';

import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';
import { User } from './entities/user.entity';

@Module({
  providers: [
    UsersResolver,
    UsersService
  ],
  imports: [
    TypeOrmModule.forFeature([ User ])
```

```
  ],
  exports: [
    // TypeOrmModule,
    UsersService
  ]
})
export class UsersModule {}
```

- La parte de creación de usuarios en el resolver generado no la quiero aquí, la quiero en Auth. Si la tendré en el servicio pero no necesito un endpoint aquí
- El método findAll es async, devuelve un Promise de tipo User[], modifico el resto también
- El resolver de User ha quedado así
- **IMPORTANTE**: asegurarse de que el **Query** importado **es de nestjs/graphql** y no de nestjs/common

```
~~~js
import { Resolver, Query, Mutation, Args, Int, ID } 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) {}

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

  @Query(() => User, { name: 'user' })
  findOne(@Args('id', { type: () => ID }) id: string
  ): Promise<User> {
    //todo:
    throw new Error('No implementado');
    // return this.usersService.findOne(id);
  }

  // @Mutation(() => User)
  // updateUser(@Args('updateUserInput') updateUserInput: UpdateUserInput) {
  //   return this.usersService.update(updateUserInput.id, updateUserInput);
  // }

  @Mutation(() => User)
  blockUser(@Args('id', { type: () => ID }) id: string
  ): Promise<User> {
    return this.usersService.block( id );
  }
}
```

- En el servicio inyecto el repositorio con @InjectRepository y le paso la entidad, luego uso Repository de typeorm de tipo la entidad (User)

```
import { BadRequestException, Injectable, InternalServerErrorException, Logger,
NotFoundException } from '@nestjs/common';
import * as bcrypt from 'bcrypt';

import { User } from './entities/user.entity';

import { CreateUserInput } from './dto/create-user.input';
import { UpdateUserInput } from './dto/update-user.input';

import { SignupInput } from '../auth/dto/inputs/signup.input';
import { Repository } from 'typeorm';
import { InjectRepository } from '@nestjs/typeorm';

@Injectable()
export class UsersService {

  //genero el logger
  private logger = new Logger('UsersService')

  constructor(
    @InjectRepository(User)
    private readonly usersRepository: Repository<User>
  ){}

  async create( signupInput: SignupInput ): Promise<User> {
    try {

      const newUser = this.usersRepository.create({
        ...signupInput,
        password: bcrypt.hashSync( signupInput.password, 10 ) //hasheo el password
       });

      return await this.usersRepository.save( newUser ); //salvo el newUser

    } catch (error) {
      this.handleDBErrors(error);
    }
  }

  async findAll(): Promise<User[]> {
    return [];
  }

  async findOneByEmail( email: string ): Promise<User> {

    try {
      return await this.usersRepository.findOneByOrFail({ email })
    } catch (error) {
```

```
        throw new NotFoundException(`${ email } not found`);

        // this.handleDBErrors({
        //   code: 'error-001',
        //   detail: `${ email } not found`
        // });
      }

    }

    async findOneById( id: string ): Promise<User> {

      try {
        return await this.usersRepository.findOneByOrFail({ id })
      } catch (error) {
        throw new NotFoundException(`${ id } not found`);
      }

    }

    update( id: number, updateUserInput: UpdateUserInput ) {
      return `This action updates a #${id} user`;
    }

    block( id: string ): Promise<User> {
      throw new Error(`block method not implement`);
    }
                                        //never porque nunca regresará un valor,
siempre una excepción
    private handleDBErrors( error: any ): never{

      //cuando tenemos una llave duplicada devuelve este código de error
      if( error.code === '23505' ){   //error.detail empieza el mensaje con 'Key
...', la quito para que quede más bonito el error en el logger
        throw new BadRequestException(error.detail.replace('Key', ''));
      }

      //cuando no encuentra el usuario lanza este error
      if( error.code == 'error-001' ){
        throw new BadRequestException(error.detail.replace('Key', ''));
      }

      this.logger.error( error );

      throw new InternalServerErrorException('Please check server logs');
    }
}
```

- El create-user.dto

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

```
@InputType()
export class CreateUserInput {
  @Field(() => Int, { description: 'Example field (placeholder)' })
  exampleField: number;
}
```

- El update-user.dto

```
import { CreateUserInput } from './create-user.input';
import { InputType, Field, Int, PartialType } from '@nestjs/graphql';

@InputType()
export class UpdateUserInput extends PartialType(CreateUserInput) {
  @Field(() => Int)
  id: number;
}
```

- Ahora creo el módulo de Auth
- Necesito métodos accesibles desde fuera como el sign in, sign out

> nest g res auth --no-spec

- GraphQl Code First (sin los endpoints)
- En el resolver debo crear dos Mutations, el sign up y el login
- Hago un tercer método para revalidar el token
- En este momento todavía no tengo el tipo de la respuesta que devolverá la Mutation
- Será src/auth/types/AuthResponse
- El custom decorator @CurrentUser lo veremos más adelante

```
import { UseGuards } from '@nestjs/common';
import { Mutation, Resolver, Query, Args } from '@nestjs/graphql';

import { AuthService } from './auth.service';
import { JwtAuthGuard } from './guards/jwt-auth.guard';

import { SignupInput, LoginInput } from './dto/inputs';
import { AuthResponse } from './types/auth-response.types';
import { CurrentUser } from './decorators/current-user.decorator';
import { User } from '../users/entities/user.entity';
import { ValidRoles } from './enums/valid-roles.enum';

@Resolver()

export class AuthResolver {
  constructor(
    private readonly authService: AuthService
  ) {}

  @Mutation( () => AuthResponse, { name: 'signup' })
```

```
  async signup(
    @Args('signupInput') signupInput: SignupInput
  ): Promise<AuthResponse>{
    return this.authService.signup(signupInput)
  }

  @Mutation( () => AuthResponse, { name: 'login' })
  async login(
    @Args('loginInput') loginInput: LoginInput
  ): Promise<AuthResponse>{
    return this.authService.login(loginInput)
  }

  @Query( () => AuthResponse, { name: 'revalidate' })
  @UseGuards( JwtAuthGuard )
  revalidateToken(
    @CurrentUser( /**[ ValidRoles.admin ]*/ ) user: User
  ): AuthResponse{
    return this.authService.revalidateToken( user );
  }
}
```

- AuthResponse

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

@ObjectType()
export class AuthResponse {

    @Field(() => String)
    token: string;

    @Field(() => User)
    user: User;

}
```

- En los dtos tengo signupInput y LoginInput
- SignupInput

```
import { Field, InputType } from "@nestjs/graphql";
import { IsEmail, IsNotEmpty, MinLength } from "class-validator";

@InputType()
export class SignupInput {

    @Field( () => String )
    @IsEmail()
```

```
    email: string;

    @Field( () => String )
    @IsNotEmpty()
    fullName: string;

    @Field( () => String )
    @MinLength(6)
    password: string;

}
```

- LoginInput

```
import { Field, InputType } from "@nestjs/graphql";
import { IsEmail, MinLength } from "class-validator";

@InputType()
export class LoginInput {

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

    @Field( () => String )
    @MinLength(6)
    password: string;

}
```

Crear usuario

- En el servicio de User

```
@Injectable()
export class UsersService {

  private logger = new Logger('UsersService')

  constructor(
    @InjectRepository(User)
    private readonly usersRepository: Repository<User>
  ){}

  async create( signupInput: SignupInput ): Promise<User> {
    try {

      const newUser = this.usersRepository.create({
```

```
        ...signupInput,
        password: bcrypt.hashSync( signupInput.password, 10 ) //hasheo el password
       });

      return await this.usersRepository.save( newUser ); //salvo el newUser

    } catch (error) {
      this.handleDBErrors(error);
    }
  }
}
```

- En el AuthService, inyecto el UsersService y el JwtService de @nestjs/jwt
- Al incluir el JwtModule en el auth.module tengo disponible el JawtService

```
import { Injectable, BadRequestException, UnauthorizedException } from
'@nestjs/common';
import { JwtService } from '@nestjs/jwt';

import * as bcrypt from 'bcrypt';

import { SignupInput, LoginInput } from './dto/inputs';
import { AuthResponse } from './types/auth-response.types';
import { User } from 'src/users/entities/user.entity';
import { UsersService } from '../users/users.service';

@Injectable()
export class AuthService {

    constructor(
        private readonly usersService: UsersService,
        private readonly jwtService: JwtService,
    ) {}

    private getJwtToken( userId: string ) {
        return this.jwtService.sign({ id: userId }); //genero el token pasándole
el ID como payload
    }

    //para crear el usuario
    async signup( signupInput: SignupInput ): Promise<AuthResponse> {
                            //llamo al usersService
        const user = await this.usersService.create( signupInput )

        const token = this.getJwtToken( user.id ); //uso el método que he creado
para generar el token

        return {token, user} //retrono el token y el user

    }
}
```

- Este token lo voy a estar buscando en los headers de la petición.
- Para que esto funcione, el jwt necesita una secret_key que escribiré en las variables de entorno
- Además en el módulo, debo registrar la estrategia con PassportModule, colocar en providers los servicios, exportarlos, importar ConfigModule y ConfigService de @nestjs/config. Al usar JwtModule.registerAsync() se dá por importado también JwtModule
- **Todo lo que lleve Module va en los imports**
- Uso JwtModule con el método Async, poder usar useFactory, importo el ConfigModule para inyectar el ConfigService y llamar a la variable de entorno en el campo secret
- En signOptions le digo que el token expire en 4 horas en el campo expiresIn
- auth.module

```
import { JwtModule } from '@nestjs/jwt';
import { ConfigModule, ConfigService } from '@nestjs/config';

import { PassportModule } from '@nestjs/passport';
import { Module } from '@nestjs/common';
import { JwtStrategy } from './strategies/jwt.strategy';

import { AuthService } from './auth.service';
import { AuthResolver } from './auth.resolver';

import { UsersModule } from '../users/users.module';

@Module({
  providers: [ AuthResolver, AuthService, JwtStrategy],
  exports: [ JwtStrategy, PassportModule, JwtModule ],
  imports: [

    ConfigModule, //no uso forFeature porque lo quiero usar en el useFactory

    PassportModule.register({ defaultStrategy: 'jwt' }),

    JwtModule.registerAsync({
      imports: [ ConfigModule ],
      inject: [ ConfigService ],
      useFactory: ( configService: ConfigService ) => ({ //uso el return implicito
al englobar la respuesta entre paréntesis

        //console.log(configService.get('JWT_SECRET')) --> me aseguro que tengo la
variable de entorno
        secret: configService.get('JWT_SECRET'),
        signOptions: {
          expiresIn: '4h'
        }
      })
    }),

    UsersModule,
  ]
```

```
})
export class AuthModule {}
```

- En auth/strategies/jwt.strategy
- La clase extiende de PassportStrategy de @nestjs/passport a la que le paso la Strategy que importo de passport-jwt
- Uso @Injectable para hacerlo un servicio (inyectable). Debo incluirlo en los providers de auth.module
- En el constructor inyecto el ConfigService y el AuthService
- En el super (constructor del padre) le paso en un objeto el secretKey usando el ConfigService para obtener la variable de entorno
- Con ExtractJwt de passport-jwt obtengo el token de la Request sin hacer el split y todo aquello para obtenerlo de los headers
- Creo un método validate que devuelve una promesa de tipo User donde extraigo el id con desestructuración del payload que recibe (que tiene una interfaz JwtPayload) y hago la busqueda por id con authService.validateUser
- **Lo que retorne esta función es lo que se añadirá a la request para poder obtener el usuario del AuthGuard con @UseGuards y poderlo extraer en el customDecorator para validar si el rol del usuario tiene permisos o no**
- Desde **AQUI YA ESTÄ EN LA REQUEST**, pues está en la **STRATEGY**

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { ConfigService } from '@nestjs/config';
import { User } from '../../users/entities/user.entity';
import { JwtPayload } from '../interfaces/jwt-payload.interface';

import { AuthService } from '../auth.service';

@Injectable()
export class JwtStrategy extends PassportStrategy( Strategy ){

    constructor(
        private readonly authService: AuthService,

        ConfigService: ConfigService
    ) {
        super({
            secretOrKey: ConfigService.get('JWT_SECRET'),
            jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken()
        })
    }

    //lo que retorne esta función se añadirá a la request, sea el user o la
excepción
    async validate( payload: JwtPayload ): Promise<User> {

        const { id } = payload;
```

```
        const user = await this.authService.validateUser( id );

        return user; //este usuario se añadirá a la request. Esto será req.user

    }
}
```

- En el authService creo el método validate para el que ya tengo el id del user, que me servirá para verificar que el usario está activo para poder acceder a las rutas que yo quiera a través del @UseGuards y el @CurrentUser (mi custom decorator para validar los roles extraidos del payload de la request)

```
 async validateUser( id: string ): Promise<User> {

        const user = await this.usersService.findOneById( id ); //findOneById es
el mismo que findOneByEmail pero con el id

        if( !user.isActive ){
            throw new UnauthorizedException(`User is inactive, talk with an
admin`);
        }

        delete user.password; //aunque esté encriptado lo borro para asegurarme
que no fluye por ahi

        return user;
    }
```

- La interfaz de JwtPayload

```
export interface JwtPayload {
    id: string;
    iat: number; //fecha de creación
    exp: number; //fecha de expiración
}
```

- En el usersService que llamo desde el signup hago uso del repositorio con typeorm para crear el user y lo guardo
- Quédate que el token no lo guardo en la DB

```
async create( signupInput: SignupInput ): Promise<User> {
    try {

      const newUser = this.usersRepository.create({
        ...signupInput,
        password: bcrypt.hashSync( signupInput.password, 10 )
       });
```

```
      return await this.usersRepository.save( newUser );

    } catch (error) {
      this.handleDBErrors(error);
    }
  }
```

- Normalmente queremos el login y el signin fuera de graphQL, en una API REST tradicional porque no queremos que la persona tenga acceso a los endpoints
- Es recomendable

Login

- Para el **login** tengo esto en el auth.resolver, una Mutation que devuelve una AuthResponse, llamo al método login en el objeto
- Uso @Args para indicar que recibirá un objeto de tipo loginInput
- El método devolverá una promesa de tipo AuthResponse
- Llamo al servicio invocando al método login pasándole el loginInput

```
@Mutation( () => AuthResponse, { name: 'login' })
async login(
  @Args('loginInput') loginInput: LoginInput
): Promise<AuthResponse>{
  return this.authService.login(loginInput)
}
```

- En el authService desestructuro el email y password del dto
- Hago la búsqueda por mail usando el método del usersService
- Usando el método compareSync de bcrypt veo si hace match el password
- Si no lo hace mando una excepción
- Si hace match genero el token pasándole el id y lo regreso junto al user

```
 async login( loginInput: LoginInput ): Promise<AuthResponse>{

        const { email, password } = loginInput;
        const user = await this.usersService.findOneByEmail( email );

        if( !bcrypt.compareSync( password, user.password) ){ //
            throw new BadRequestException('Email / Password do not match');
        }

        const token = this.getJwtToken( user.id );

        return {
            token,
            user
```

```
        }
    }
```

- El método findOneByemail del usersService

```
async findOneByEmail( email: string ): Promise<User> {

    try {                                  //en el caso de que no lo encuentre va a
lanzar un error que atraparé con el catch
      return await this.usersRepository.findOneByOrFail({ email })
    } catch (error) {

      throw new NotFoundException(`${ email } not found`);

      // this.handleDBErrors({
      //   code: 'error-001',
      //   detail: `${ email } not found`
      // });
    }
  }
```

- Para validar las rutas creo en el auth.resolver una petición que requiera autenticación
- Siempre queremos regresar algo de tipo Authresponse porque ahí es dónde tenemos el usuario y el token

```
@Query( () => AuthResponse, { name: 'revalidate' })
@UseGuards( JwtAuthGuard ) //no hace falta ejecutarlo porque ya lo estoy invocando
en la construcción
revalidateToken(
  @CurrentUser( /**[ ValidRoles.admin ]*/ ) user: User
): AuthResponse{
  return this.authService.revalidateToken( user );
}
```

- El revalidateToken del auth.service luce así

```
revalidateToken( user: User ): AuthResponse {

    const token = this.getJwtToken( user.id );

    return { token, user }

}
```

- Si uso @UseGuards con el AuthGuard me lanza un error que no puede leer las propiedades de undefined

- Más adelante usaremos este @UseGuard a nivel de resolver porque haremos que todos los querys necesiten uno u otro role
- Crearemos nuestro propio AuthGuard basado en el que ofrece passport
- Le especifico que uso para validar, 'jwt'
- Sobreescribo el método getRequest de AuthGuard y le paso cómo parámetro el ExecutionContext de nestjs/common
- Creo el contexto con gqlExecutionContext (gql de GraphQL) y le paso el context
- Obtengo la request usando el context
- Retorno la request
- En src/auth/guards/jwt-auth.guard

```
import { ExecutionContext } from '@nestjs/common';
import { GqlExecutionContext } from '@nestjs/graphql';
import { AuthGuard } from '@nestjs/passport';

                                //como lo estoy ejecutando aquí no hace falta
ponerle paréntesis en el @UseGuards
export class JwtAuthGuard extends AuthGuard('jwt') {

    //! Override (sobreescribo el getRequest de AuthGuard)
    getRequest( context: ExecutionContext ) {

        const ctx = GqlExecutionContext.create( context ); //creo el context de
graphQL
        const request = ctx.getContext().req; //obtengo la request

        return request; //retorno la request

    }

}
```

- Creo un custom decorator con la función de flecha createParamdecorator, le paso los roles válidos y el context
- Creo de nuevo el context graphQL y obtengo el user de la request
- Debo validar si el rol tiene permisos y si está activo
- En auth/decorators/current-user.decorator

```
import { createParamDecorator, ExecutionContext, ForbiddenException,
InternalServerErrorException } from '@nestjs/common';
import { GqlExecutionContext } from '@nestjs/graphql';
import { ValidRoles } from '../enums/valid-roles.enum';
import { User } from '../../users/entities/user.entity';

export const CurrentUser = createParamDecorator(
    ( roles: ValidRoles[] = [], context: ExecutionContext ) => {

        const ctx = GqlExecutionContext.create( context ) //creo el context de
```

```
graphql
        const user: User = ctx.getContext().req.user; //extraigo el user de la
request que obtengo del strategy con validate

        if ( !user ) {   //si viene nulo será un error del server, porque no
debería ser null
            throw new InternalServerErrorException(`No user inside the request -
make sure that we used the AuthGuard`)
        }

        if ( roles.length === 0 ) return user; //si el role está vacío voy a dejar
pasar al usuario (por defecto todos tienen "user")

        for ( const role of user.roles ) { //recorro el array de roles
            //TODO: Eliminar Valid Roles
            if ( roles.includes( role as ValidRoles ) ) { //valido que el role
esté en ValidRoles
                return user;
            }
        }

        //si no tiene un role válido  devuelvo la excepción
        throw new ForbiddenException(
            `User ${ user.fullName } need a valid role [${ roles }]`
        )

})
```

- auth/enum/valid-roles

```
// TODO: Implementar enum como GraphQL Enum Type
export enum ValidRoles {

    admin = 'admin',
    user = 'user',
    superUser = 'superUser'
}
```

- Podría pasarle así el role al Query

```
@Query( () => AuthResponse, { name: 'revalidate' })
@UseGuards( JwtAuthGuard ) //no hace falta ejecutarlo porque ya lo estoy invocando
en la construcción
revalidateToken(
  @CurrentUser( [ ValidRoles.admin ] ) user: User
): AuthResponse{
  return this.authService.revalidateToken( user );
}
```

- Para hacer la consulta desde el playground, debo añadir crear en el apartado headers Authorization y pasarle el token como Bearer oauihaoshaois
- Bearer(espacio) tokensincomillasninada
- El query revalidateToken es solo para comprobar que funciona el AuthGuard que he creado y el @CurrentUser
- Las querys serían algo asi
- Para crear usuario, en el playground, donde en la query pido que me devuelva el fullName y el token

```
mutation Signup($signUp: SignupInput!){

  signup(signupInput: $signUp){
    user{
      fullName
    }
    token
  }

}
```

- En query variables

```
{
  "signUp": {
    "fullName": "Marta",
   "email": "marta@gmail.com",
    "password": "soclamarta"
  }
}
```

- Me devuelve esto

```
{
  "data": {
    "signup": {
      "user": {
        "fullName": "Marta"
      },
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVjOGIxZGU2LWZhNGQtNGUyNy04ZjU1LTM5
YmY1MTVmOTkzMSIsImlhdCI6MTcxODIyNjA5NSwiZXhwIjoxNzE4MjQwNDk1fQ.-OoEkevWs-
It677KFfrlkOA_-Aqqwn2BbBMsZeChge0"
    }
  }
}
```

- Para el Login (en el caso de que quiera obtener el fullName y el token)

```
mutation Login($loginInput: LoginInput!){

  login(loginInput: $loginInput){
        user{
      fullName
    }
    token
  }

}
```

- En las variables

```
{
  "loginInput": {
   "email": "marta@gmail.com",
    "password": "soclamarta"
  }
}
```

- Debo pasarle el token en el playground en HTTP HEADERS

```
{
  "Authorization": "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVjOGIxZGU2LWZhNGQtNGUyNy04ZjU1LTM5Y
mY1MTVmOTkzMSIsImlhdCI6MTcxODIyNjA5NSwiZXhwIjoxNzE4MjQwNDk1fQ.-OoEkevWs-
It677KFfrlkOA_-Aqqwn2BbBMsZeChge0"
}
```

- Me devuelve esto

```
{
  "data": {
    "login": {
      "user": {
        "fullName": "Marta"
      },
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVjOGIxZGU2LWZhNGQtNGUyNy04ZjU1LTM5
YmY1MTVmOTkzMSIsImlhdCI6MTcxODIyNjQzMiwiZXhwIjoxNzE4MjQwODMyfQ.yU7BxEuF11b_GTDZs5c
VTZ-5-10veSdM2-h_Y8YVbLw"
    }
  }
}
```

- Para la query revalidate

```
query Revalidate{
  revalidate{
    token
    user{
      id
      fullName
      roles
      isActive
    }
  }
}
```

- En HTTP HEADERS debo pasarle el Bearer Token del Login

```
{
  "Authorization": "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVjOGIxZGU2LWZhNGQtNGUyNy04ZjU1LTM5Y
mY1MTVmOTkzMSIsImlhdCI6MTcxODIyNjQzMiwiZXhwIjoxNzE4MjQwODMyfQ.yU7BxEuF11b_GTDZs5cV
TZ-5-10veSdM2-h_Y8YVbLw"
}
```

- Me devuelve esto

```
{
  "data": {
    "revalidate": {
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVjOGIxZGU2LWZhNGQtNGUyNy04ZjU1LTM5
YmY1MTVmOTkzMSIsImlhdCI6MTcxODIyNjg5NCwiZXhwIjoxNzE4MjQxMjk0fQ.gFSjMoJ2jyjKjtnrdqJ
alAwmV22lVTl3JjtllDwytZc",
      "user": {
        "id": "5c8b1de6-fa4d-4e27-8f55-39bf515f9931",
        "fullName": "Marta",
        "roles": [
          "user"
        ],
        "isActive": true
      }
    }
  }
}
```

- El env.template

```
STATE=dev
DB_PASSWORD=123456
DB_NAME=AnyList
DB_HOST=localhost
DB_PORT=5434
DB_USERNAME=postgres

JWT_SECRET=por_favor_cambiar_esto
```

04 Nest GraphQL - Usuarios y Enums

- Tengo este docker-compose.yml

```
version: '3'


services:
  db:
    image: postgres:14.4
    restart: always
    ports:
      - "5434:5432"
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    container_name: anylistDB
    volumes:
      - ./postgres:/var/lib/postgresql/data
```

- Con este .env

```
STATE=dev
DB_PASSWORD=123456
DB_NAME=AnyList
DB_HOST=localhost
DB_PORT=5434
DB_USERNAME=postgres

JWT_SECRET=Cambia_esto
```

- Antes de iniciar el server correr Docker y ejecutar

```
docker compose up -d
```

- El main sigue igual

```
import { NestFactory } from '@nestjs/core';
import { ValidationPipe } from '@nestjs/common';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
    whitelist: true,
    forbidNonWhitelisted: true,
    })
  );

  await app.listen(3000);
}
bootstrap();
```

- Este es mi app.module
- Si en lugar de Apollo Studio voy a usar el playground, coloco el playground en true y comento ApolloServerPluginLandingPageLocalDefault
- He importado el AuthModule y he inyectado el JwtService para inyectarlko en el useFactory y usarlo como servicio, por lo que ahora si coloco el forRoot() al ConfigModule
- Con el useFactory tengo el context desde donde puedo extraer la request
- Puedo comprobar si tengo el token y decodificarlo
- De esta manera puedo bloquear el schema con la autenticación

```
import { join } from 'path';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { ConfigModule } from '@nestjs/config';
import { Module } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';

import { GraphQLModule } from '@nestjs/graphql';
import { TypeOrmModule } from '@nestjs/typeorm';

import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';

import { ItemsModule } from './items/items.module';
import { UsersModule } from './users/users.module';
import { AuthModule } from './auth/auth.module';


@Module({
  imports: [

    ConfigModule.forRoot(),

    //esta configuración es para bloquear el schema a través de autenticación JWT
    GraphQLModule.forRootAsync({
```

```
      driver: ApolloDriver,
      imports: [ AuthModule ],
      inject: [ JwtService ],
      useFactory: async( jwtService: JwtService ) => ({
        playground: false,
        autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
        plugins: [
          ApolloServerPluginLandingPageLocalDefault
        ],
        context({ req }) {
          // const token = req.headers.authorization?.replace('Bearer ','');
          // if ( !token ) throw Error('Token needed');

          // const payload = jwtService.decode( token );
          // if ( !payload ) throw Error('Token not valid');

        }
      })
    }),

    // TODO: configuración básica
    // GraphQLModule.forRoot<ApolloDriverConfig>({
    //   driver: ApolloDriver,
    //   // debug: false,
    //   playground: false,
    //   autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
    //   plugins: [
    //     ApolloServerPluginLandingPageLocalDefault
    //   ]
    // }),

    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: +process.env.DB_PORT,
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      database: process.env.DB_NAME,
      synchronize: true,
      autoLoadEntities: true,
    }),

    ItemsModule,

    UsersModule,

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

- Los enumTypes en GraphQl funcionan ingual que en TypeScript
- Registro el enum con la función registerEnumType de nestjs/graphql para poder usarlo como tipo de dato GraphQL
- src/auth/enums/valid-roles.enum

```
import { registerEnumType } from "@nestjs/graphql";

//aqui es un enum de typescript
export enum ValidRoles {
    admin     = 'admin',
    user      = 'user',
    superUser = 'superUser'
}

//aqui hago el enum un tipo de GraphQL
registerEnumType( ValidRoles, { name: 'ValidRoles', description: 'Fiesta en tu
casa a las 3' } )
```

- Creo el @ArgsType para pasarle los roles que me interesa devolver en el findAll
- Uso @IsArray
- Puede ser nulo
- user/dto/args/roles.args

```
import { ArgsType, Field } from '@nestjs/graphql';
import { IsArray } from 'class-validator';
import { ValidRoles } from '../../../auth/enums/valid-roles.enum';

@ArgsType()
export class ValidRolesArgs {
                  //para poder colocar el enum aqui como tipo de dato debo haberlo
registrado con registerEnumType
    @Field( () => [ValidRoles], { nullable: true })
    @IsArray()
    roles: ValidRoles[] = [] //declararlo como un arreglo vacío por defecto indica
que por defecto será nulo


}
```

- El users.resolver queda así
- El Resolver devuelve siempre algo de tipo User
- **Coloco el @UseGuards a nivel de Resolver y le paso el JwtAuthGuard**
- En **findAll** como argumento le paso el tipo que he creado ValidRolesArg
    - En el custom decorator @CurrentUser le digo que solo pueden acceder a la ruta admin y superUser
    - @CurrentUser user devolverá un user de tipo User
    - El método findAll devolverá una promesa de tipo arreglo de User

    - En el servicio crearé un QueryBuilder para recorrer los roles y sacar los usuarios (explicado más adelante)
- En el updateUser y blockUser necesito pasarle también el user
- En updateUser lo necesito para el campo lastUpdateBy
- En blockUser para colocarle el isActive en false

```
import { UseGuards, ParseUUIDPipe } from '@nestjs/common';
import { Resolver, Query, Mutation, Args, Int, ID } 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';
import { ValidRolesArgs } from './dto/args/roles.arg';

import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { CurrentUser } from '../auth/decorators/current-user.decorator';
import { ValidRoles } from '../auth/enums/valid-roles.enum';

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


  @Query(() => [User], { name: 'users' })
  async findAll(
    @Args() validRoles: ValidRolesArgs,
    @CurrentUser([ValidRoles.admin, ValidRoles.superUser ]) user: User
  ):Promise<User[]> {

    const users = await this.usersService.findAll( validRoles.roles );
    console.log(users);
    return this.usersService.findAll( validRoles.roles );
  }

  @Query(() => User, { name: 'user' })
  findOne(
    @Args('id', { type: () => ID }, ParseUUIDPipe ) id: string,
    @CurrentUser([ValidRoles.admin, ValidRoles.superUser ]) user: User
  ): Promise<User> {

    return this.usersService.findOneById(id);
  }

  @Mutation(() => User, { name: 'updateUser' })
  async updateUser(
    @Args('updateUserInput') updateUserInput: UpdateUserInput,
    @CurrentUser([ValidRoles.admin ]) user: User
  ): Promise<User> {
    return this.usersService.update(updateUserInput.id, updateUserInput, user );
  }
```

```
  @Mutation(() => User, { name: 'blockUser' })
  blockUser(
    @Args('id', { type: () => ID }, ParseUUIDPipe ) id: string,
    @CurrentUser([ ValidRoles.admin ]) user: User
  ): Promise<User> {
    return this.usersService.block(id, user );
  }
}
```

- El users.service

```
import { BadRequestException, Injectable, InternalServerErrorException, Logger,
NotFoundException } from '@nestjs/common';
import * as bcrypt from 'bcrypt';


import { User } from './entities/user.entity';

import { CreateUserInput } from './dto/create-user.input';
import { UpdateUserInput } from './dto/update-user.input';
import { ValidRoles } from './../auth/enums/valid-roles.enum';

import { SignupInput } from './../auth/dto/inputs/signup.input';
import { Repository } from 'typeorm';
import { InjectRepository } from '@nestjs/typeorm';


@Injectable()
export class UsersService {

  private logger: Logger = new Logger('UsersService')


  constructor(
    @InjectRepository(User)
    private readonly usersRepository: Repository<User>
  ) {}


  async create( signupInput: SignupInput ): Promise<User> {

    try {

        const newUser = this.usersRepository.create({
          ...signupInput,
          password: bcrypt.hashSync( signupInput.password, 10 )
        });

        return await this.usersRepository.save( newUser );
```

```
    } catch (error) {
      this.handleDBErrors( error );
    }

  }


  async findAll( roles: ValidRoles[] ): Promise<User[]> {

    if ( roles.length === 0 )
      return this.usersRepository.find({
        // TODO: No es necesario porque tenemos lazy la propiedad lastUpdateBy
        // relations: {
        //   lastUpdateBy: true
        // }
      });

    // ??? tenemos roles ['admin','superUser']
    return this.usersRepository.createQueryBuilder()
      .andWhere('ARRAY[roles] && ARRAY[:...roles]')
      .setParameter('roles', roles )
      .getMany();


  }

  async findOneByEmail( email: string ): Promise<User> {
    try {
      return await this.usersRepository.findOneByOrFail({ email });
    } catch (error) {
      throw new NotFoundException(`${ email } not found`);
      // this.handleDBErrors({
      //   code: 'error-001',
      //   detail: `${ email } not found`
      // });
    }
  }

  async findOneById( id: string ): Promise<User> {
    try {
      return await this.usersRepository.findOneByOrFail({ id });
    } catch (error) {
      throw new NotFoundException(`${ id } not found`);
    }
  }

  async update(
    id: string,
    updateUserInput: UpdateUserInput,
    updateBy: User
  ): Promise<User> {

    try {
      const user = await this.usersRepository.preload({
```

```
        ...updateUserInput,
        id
      });

      user.lastUpdateBy = updateBy;

      return await this.usersRepository.save( user );

    } catch (error) {
      this.handleDBErrors( error );
    }


  }

  async block( id: string, adminUser: User ): Promise<User> {

    const userToBlock = await this.findOneById( id );

    userToBlock.isActive = false;
    userToBlock.lastUpdateBy = adminUser;

    return await this.usersRepository.save( userToBlock );

  }


  private handleDBErrors( error: any ): never {


    if (  error.code === '23505' ) {
      throw new BadRequestException( error.detail.replace('Key ','') );
    }

    if ( error.code == 'error-001' ) {
      throw new BadRequestException( error.detail.replace('Key ','') );
    }

    this.logger.error( error );

    throw new InternalServerErrorException('Please check server logs');

  }

}
```

- En la user.entity creo una relación @ManyToOne de User a lastUpdateBy en el campo lastUpdateBy que es opcional
- Le digo que puede ser nulo y coloco el lazy en true
- Le coloco @JoinColumn y le paso el name en un objeto
- Le digo que es de tipo User

```
import { ObjectType, Field, Int, ID } from '@nestjs/graphql';
import { Column, Entity, JoinColumn, ManyToOne, PrimaryGeneratedColumn } from
'typeorm';

@Entity({ name: 'users' })
@ObjectType()
export class User {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  fullName: string;

  @Column({ unique: true })
  @Field( () => String )
  email: string;

  @Column()
  // @Field( () => String )
  password: string;

  @Column({
    type: 'text',
    array: true,
    default: ['user']
  })
  @Field( () => [String] )
  roles: string[]

  @Column({
    type: 'boolean',
    default: true
  })
  @Field( () => Boolean )
  isActive: boolean;

  //TODO: relaciones
  @ManyToOne( () => User, (user) => user.lastUpdateBy, { nullable: true, lazy:
true })
  @JoinColumn({ name: 'lastUpdateBy' })
  @Field( () => User, { nullable: true })
  lastUpdateBy?: User;

}
```

- Añado los campos opcionales al dto de updateUSer, ambos pueden ser nulos

```
import { InputType, Field, Int, PartialType, ID } from '@nestjs/graphql';
import { IsArray, IsBoolean, IsOptional, IsUUID } from 'class-validator';

import { CreateUserInput } from './create-user.input';
import { ValidRoles } from './../../auth/enums/valid-roles.enum';


@InputType()
export class UpdateUserInput extends PartialType(CreateUserInput) {

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

  @Field( () => [ValidRoles], { nullable: true })
  @IsArray()
  @IsOptional()
  roles?: ValidRoles[];

  @Field( () => Boolean, { nullable: true })
  @IsBoolean()
  @IsOptional()
  isActive?: boolean;

}
```

- Para hacer el query de findAll debo estar logeado
- Primero creo el usuario y le pido que me devuelva el fullName y el token

```
mutation CreateUser($signUp: SignupInput!){
  signup(signupInput: $signUp){
    user{
      fullName
    }
    token
  }
}
```

- En las variables le paso los valores a $signUp

```
{
  "signUp": {
    "email": "miguel@gmail.com",
    "fullName": "Miguel",
    "password": "123456"
  }
}
```

- Me devuelve el objeto data

```
{
  "data": {
    "signup": {
      "user": {
        "fullName": "Miguel"
      },
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImM3MjVlNDU1LTdhY2EtNDBlNi1hY2M1LTE0
ZjYxM2YwOGMxZiIsImlhdCI6MTcxODI2NTQxMywiZXhwIjoxNzE4Mjc5ODEzfQ.eNt7pTns8-
1U8eHupL3mz0pKV6yEDDNhXcrNoh6qYJE"
    }
  }
}
```

- Copio el token y hago el login pasándole en HTTP HEADERS el "Authorization": "Bearer token_aqui"

```
mutation LoginUser($loginInput: LoginInput!){
  login(loginInput: $loginInput){
    user{
      fullName
    }
    token
  }
}
```

- En las variables le paso a la variable loginInput el email y password

```
{
  "loginInput": {
    "email": "miguel@gmail.com",
    "password": "123456"
  }
}
```

- En HTTP HEADERS le paso el token del signup

```
{
  "Authorization": "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImM3MjVlNDU1LTdhY2EtNDBlNi1hY2M1LTE0Z
jYxM2YwOGMxZiIsImlhdCI6MTcxODI2NTQxMywiZXhwIjoxNzE4Mjc5ODEzfQ.eNt7pTns8-
1U8eHupL3mz0pKV6yEDDNhXcrNoh6qYJE"
}
```

- Esto me devuelve el objeto data con los campos que le solicité (fullName y el token)


```
{
  "data": {
    "login": {
      "user": {
        "fullName": "Miguel"
      },
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImM3MjVlNDU1LTdhY2EtNDBlNi1hY2M1LTE0ZjYxM2YwOGMxZiIsImlhdCI6MTcxODI2NTc5OSwiZXhwIjoxNzE4MjgwMTk5fQ.QeDFAdelsK_tWb35kTSz1UbtQf_lAXypvA7LralbpRo"
    }
  }
}
```


- Ya puedo usar el token para acceder a los endpoints
- Ahora tengo el role user por defecto, habrá endpoints a los que no podré acceder, creo otro usuario con role admin y nombre admin
- Le cambio el role desde TablePlus
- Para hacer la query, le paso el argumento roles y le pido que me devuelva aquellos que sean user

```
query Users{
  users(roles: user ){
    fullName
  }
}
```

- En HTTP HEADERS le coloco el token extraido del login de un usuario al que le he cambiado el role desde TablePlus a admin


```
{
  "Authorization": "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MGJlZjkwLTQzMGUtNDdmZS1iYThkLTc5OGI4N2YwZjAxOCIsImlhdCI6MTcxODI2NjA2NCwiZXhwIjoxNzE4MjgwNDY0fQ.uv3-yWS-l5LPtaXksK2cbLKG5m7XHTpiEcLtnDJ4q-U"
}
```


- Si miro el tipo ValidRolesArg que he creado con @ArgsType y he colocado en el query findAll, tiene un campo llamado role
- Por eso no ha hecho falta colocar el argumento como string entre los paréntesis de @Args() validRoles: ValidRolesArg en el query del resolver, ya que he declarado el tipo y este ya incluye role
- Para hacer la consulta

```
query Users{
  users(roles: user ){
    fullName
  }
}
```

- Me devuelve esto

```
{
  "data": {
    "users": [
      {
        "fullName": "Miguel"
      }
    ]
  }
}
```

- Miremos el users.service, creo el queryBuilder
- Podría usar el where. Usando andWhere todos los andWhere que vengan detrás **se tienen que cumplir**
- Le indico que busque en un ARRAY según la documentación para hacer querys sobre arreglos

```
'ARRAY[nombre_del_arreglo_en_la_DB]'
```

- Le indico con && (agrego otra condición, debe de estar mi argumento que lo indico con 😃 y esparzo roles con el spread ...

```
'ARRAY[roles] && '[:...roles]'
```

- Le estoy diciendo que el arreglo que le voy a mandar de roles como parámetro, al menos uno tiene que hacer match con la tabla de roles
- Por eso uso :...roles. : Para indicar que es un parámetro que introduzco desde fuera (como en los endpoints /:id) y ...roles para esparcir el arreglo en caso de mandar más de un role como parámetro
- Uso setParameter para establecer este parámetro de :roles que he indicado en el andWhere
- También ayuda a escapar caracteres especiales, evitar inyecciones de querys, etc
- El primer 'roles' es el nombre que le puse al parametro :...roles. El segundo parámetro roles es el valor que le estoy pasando como parámetro a findAll(roles) <--- este roles
- Uso getMany para obtener varios resultados

```
async findAll( roles: ValidRoles[] ): Promise<User[]> {

  if ( roles.length === 0 )
```

```
    return this.usersRepository.find()

  //necesito tener el role de admin o superuser
  return this.usersRepository.createQueryBuilder()
    .andWhere('ARRAY[roles] && ARRAY[:...roles]')
    .setParameter('roles', roles )
    .getMany();
}
```

- Para poder acceder a findAll necesito ser admin o superuser

```
@Query(() => [User], { name: 'users' })
async findAll(
  @Args() validRoles: ValidRolesArgs,
  @CurrentUser([ValidRoles.admin, ValidRoles.superUser ]) user: User
):Promise<User[]> {

  //const users = await this.usersService.findAll( validRoles.roles );
  //console.log(users);
  return this.usersService.findAll( validRoles.roles );
}
```

- **findOne**
- user.resolver
- Es un query porque solo queremos traer data, no vamos a impactar la DB

```
@Query(() => User, { name: 'user' })
findOne(
  @Args('id', { type: () => ID }, ParseUUIDPipe ) id: string,
  @CurrentUser([ValidRoles.admin, ValidRoles.superUser ]) user: User
): Promise<User> {

  return this.usersService.findOneById(id);
}
```

- En el users.service uso el findOnByOrFail en un try catch por si lanza el error atraparlo con el catch
- Por eso lo de 'OrFail'

```
async findOneById( id: string ): Promise<User> {
  try {
    return await this.usersRepository.findOneByOrFail({ id });
  } catch (error) {
    throw new NotFoundException(`${ id } not found`);
  }
}
```

```
//el findOneByEmail es igual, lo usaremos en otros lugares
async findOneByEmail( email: string ): Promise<User> {
  try {
    return await this.usersRepository.findOneByOrFail({ email });
  } catch (error) {
    throw new NotFoundException(`${ email } not found`);
    // this.handleDBErrors({
    //   code: 'error-001',
    //   detail: `${ email } not found`
    // });
  }
}
```

- Para hacer la query debo ser admin o superuser
- Le pido que me devuelva el fullName, el email y el role

```
query User($id: ID!){
  user(id: $id ){
    fullName
    email
    roles
  }
}
```

- Le paso un token de algun usuario con role admin
- En variables le paso el id

```
{
  "id": "c725e455-7aca-40e6-acc5-14f613f08c1f"
}
```

- Me devuelve esto

```
{
  "data": {
    "user": {
      "fullName": "Miguel",
      "email": "miguel@gmail.com",
      "roles": [
        "user"
      ]
    }
  }
}
```

Bloquear un usuario - ManyToOne

- Para bloquear un usuario voy a cambiar el estado de activo a false. Eso sería muy sencillo
- Con objetivos didácticos crearemos una nueva columna en la tabla de usuarios llamada lastUpdatedBy
- Nos dirá quien fue la última persona que hizo un cambio en esta tabla
- Esto nos servirá para aprender la relación ManyToOne
- Debe hacerlo un admin, le paso el id, lo valido con el UUIDPipe para asegurarme de que sea un UUID
- user.resolver

```
@Mutation(() => User, { name: 'blockUser' })
blockUser(
  @Args('id', { type: () => ID }, ParseUUIDPipe ) id: string,
  @CurrentUser([ ValidRoles.admin ]) user: User
): Promise<User> {
  return this.usersService.block(id, user );
}
```

- Para añadir la relación ManyToOne en la entidad
- ManyToOne porque el mismo usuario puede estar en muchas actualizaciones, muchas personas y se van a relacionar con una

```
import { ObjectType, Field, Int, ID } from '@nestjs/graphql';
import { Column, Entity, JoinColumn, ManyToOne, PrimaryGeneratedColumn } from
'typeorm';

@Entity({ name: 'users' })
@ObjectType()
export class User {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  fullName: string;

  @Column({ unique: true })
  @Field( () => String )
  email: string;

  @Column()
  // @Field( () => String )
  password: string;

  @Column({
    type: 'text',
    array: true,
    default: ['user']
```

```
  })
  @Field( () => [String] )
  roles: string[]

  @Column({
    type: 'boolean',
    default: true
  })
  @Field( () => Boolean )
  isActive: boolean;
              //User (el ObjectType), como se va a relacionar user con
lastUpdateBy
  @ManyToOne( () => User, (user) => user.lastUpdateBy, { nullable: true, lazy:
true }) //lazy es para que cargue la relación
  @JoinColumn({ name: 'lastUpdateBy' }) //para que typeorm cargue la información
de este campo,
                                        //uso el name para ponerle mi nombre
personalizado a la columna en la tabla
  @Field( () => User, { nullable: true }) //hay que indicarle a graphQL que tipo
de dato va a tener con Field
  lastUpdateBy?: User;

}
```

- En el users.service

```
async block( id: string, adminUser: User ): Promise<User> {

  const userToBlock = await this.findOneById( id );

  userToBlock.isActive = false;
  userToBlock.lastUpdateBy = adminUser;

  return await this.usersRepository.save( userToBlock );

}
```

- Para obtener el lastUpdateBy yo podría hacer esto, pero esto me sigue regresando valores null ya que entra en el array con el queryBuilder

```
async findAll( roles: ValidRoles[] ): Promise<User[]> {

  if ( roles.length === 0 )
    return this.usersRepository.find({
    relations: {
     lastUpdatebY: true
  }});

  //necesito tener el role de admin o superuser
```

```
    return this.usersRepository.createQueryBuilder()
      .andWhere('ARRAY[roles] && ARRAY[:...roles]')
      .setParameter('roles', roles )
      .getMany();
  }
```

- El eager en true funciona (excepto en el query builder) para cargar la relación
- Puedo usar el **lazy en true**, es una forma de decirle cuando se cargue, carga esto también

```
import { ObjectType, Field, Int, ID } from '@nestjs/graphql';
import { Column, Entity, JoinColumn, ManyToOne, PrimaryGeneratedColumn } from
'typeorm';

@Entity({ name: 'users' })
@ObjectType()
export class User {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  fullName: string;

  @Column({ unique: true })
  @Field( () => String )
  email: string;

  @Column()
  // @Field( () => String )
  password: string;

  @Column({
    type: 'text',
    array: true,
    default: ['user']
  })
  @Field( () => [String] )
  roles: string[]

  @Column({
    type: 'boolean',
    default: true
  })
  @Field( () => Boolean )
  isActive: boolean;
              //User (el ObjectType), como se va a relacionar user con
lastUpdateBy
  @ManyToOne( () => User, (user) => user.lastUpdateBy, { nullable: true, lazy:
true }) //lazy es para que cargue la relación
```

```
    @JoinColumn({ name: 'lastUpdateBy' })
  //ya que el eager en true no funciona con el queryBuilder
    lastUpdateBy?: User;
  //que es lo que tengo en el findAll

  }
```

- Entonces no es necesario añadir al find la relación para que la cargue porque usamos el lazy en la relación desde la entidad

```
async findAll( roles: ValidRoles[] ): Promise<User[]> {

  if ( roles.length === 0 )
    return this.usersRepository.find({
      // TODO: No es necesario porque tenemos lazy la propiedad lastUpdateBy
      // relations: {
      //   lastUpdateBy: true
      // }
    });

  // ??? tenemos roles ['admin','superUser']
  return this.usersRepository.createQueryBuilder()
    .andWhere('ARRAY[roles] && ARRAY[:...roles]')
    .setParameter('roles', roles )
    .getMany();


}
```

Update

- Hagamos una mutation en el users.resolver
- Solo el admin puede acceder
- Le paso el id, el dto y el user al servicio

```
  @Mutation(() => User, { name: 'updateUser' })
  async updateUser(
    @Args('updateUserInput') updateUserInput: UpdateUserInput,
    @CurrentUser([ValidRoles.admin ]) user: User
  ): Promise<User> {
    return this.usersService.update(updateUserInput.id, updateUserInput, user );
  }
```

- Recordemos el dto
- Es un @InputType para graphQL ya que lo recibo del body de la petición
- Extiendo la clase con Partial (lo que indica que todos los parámetros son opcionales) de CreateUserDto

- Coloco el id como obligatorio y los otros dos opcionales
- Uso los decoradores para validar

```
import { InputType, Field, Int, PartialType, ID } from '@nestjs/graphql';
import { IsArray, IsBoolean, IsOptional, IsUUID } from 'class-validator';

import { CreateUserInput } from './create-user.input';
import { ValidRoles } from './../../auth/enums/valid-roles.enum';


@InputType()
export class UpdateUserInput extends PartialType(CreateUserInput) {

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

  @Field( () => [ValidRoles], { nullable: true })
  @IsArray()
  @IsOptional()
  roles?: ValidRoles[];

  @Field( () => Boolean, { nullable: true })
  @IsBoolean()
  @IsOptional()
  isActive?: boolean;

}
```

- En el users.service guardo en user usando el repositorio con el método preload y esparzo el dto, para pasarle el id y el body
- preload si no lo encuentra mandará el error al catch
- Actualizo el lastUpdateBY
- Salvo los cambios

```
  async update(
    id: string,
    updateUserInput: UpdateUserInput,
    updateBy: User
  ): Promise<User> {

    try {
      const user = await this.usersRepository.preload({
        ...updateUserInput,
        id
      });

      user.lastUpdateBy = updateBy;

      return await this.usersRepository.save( user );
```

```
    } catch (error) {
      this.handleDBErrors( error );
    }
  }
```

Bloquear GQLSchema

- signup y login no tienen que estar protegidos, porque si no no podríamos ingresar
- Usualmente se crean en un REST API, y el token generado desde ahi se usa para el schema
- Si el usuario no está autenticado con un token válido no va a poder acceder a los endpoints
- Lo que voy a hacer es bloquear la manera en la que el schema se obtiene mcon el JwtService
- Uso **forRootAsync**. El driver es obligatorio.
- Importo el AuthModule donde tengo el JwtService
- Inyecto el JwtService
- Uso el useFactory para inyectar el JwtService
- Le indico el playground en true porque lo quiero usar desde el navegador para hacer las consultas en localhost:3000/graphql
- Le indico el path del schema de graphQL
- Si uso el ApolloPlugin... el playground debe de estar en false, y entonces usar Apollo Studio
- En el useFactory tengo el context, que trae la información de mi schema, del que puedo desestructurar la req
- **Si no dejo un espacio después de Bearer usando el replace** para eliminar esto de la request y quedarme solo con el token, **me quedaría el token con un espacio al principio**, lo que daría error
- Debo ponerle ? al authorization porque puede no venir
- Si el payload es null lanzo el error
- De esta manera **TAMBIÉN ESTOY PIDIENDO UN TOKEN PARA EL SIGNIN Y EL SIGNUP** por lo que **lo voy a dejar comentado de momento**
- app.module

```
import { join } from 'path';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { ConfigModule } from '@nestjs/config';
import { Module } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';

import { GraphQLModule } from '@nestjs/graphql';
import { TypeOrmModule } from '@nestjs/typeorm';

import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';

import { ItemsModule } from './items/items.module';
import { UsersModule } from './users/users.module';
import { AuthModule } from './auth/auth.module';


@Module({
```

```
    imports: [

      ConfigModule.forRoot(),

      GraphQLModule.forRootAsync({
        driver: ApolloDriver,
        imports: [ AuthModule ],
        inject: [ JwtService ],
        useFactory: async( jwtService: JwtService ) => ({
          playground: true,
          autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
          plugins: [
            //ApolloServerPluginLandingPageLocalDefault
          ],
          context({ req }) {              //es importante el espacio despues de
Bearer, si no dará error!!
            // const token = req.headers.authorization?.replace('Bearer ','');
            // if ( !token ) throw Error('Token needed');

            // const payload = jwtService.decode( token );
            // if ( !payload ) throw Error('Token not valid');

          }
        })
      }),

      // TODO: configuración básica
      // GraphQLModule.forRoot<ApolloDriverConfig>({
      //   driver: ApolloDriver,
      //   // debug: false,
      //   playground: false,
      //   autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
      //   plugins: [
      //     ApolloServerPluginLandingPageLocalDefault
      //   ]
      // }),

      TypeOrmModule.forRoot({
        type: 'postgres',
        host: process.env.DB_HOST,
        port: +process.env.DB_PORT,
        username: process.env.DB_USERNAME,
        password: process.env.DB_PASSWORD,
        database: process.env.DB_NAME,
        synchronize: true,
        autoLoadEntities: true,
      }),

      ItemsModule,

      UsersModule,

      AuthModule,
    ],
```

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

05 Nest GraphQL - Items + Usuarios: Peticiones autenticadas

- No vamos a trabajar estando pidiendo el id del usuario
- Trabajaremos basándonos en el dueño/a del token
- Más adelante haremos paginación
- No voy a poder crear items si no se a que usuario pertenece
- Me interesa poder diferenciar estos artículos porque podemos tener cientos de usuarios y algunos el mismo artículo
- Por eso necesito saber de quien es el articulo de manera indexada
- De momento el app.module se queda igual
- app.module

```
import { join } from 'path';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { ConfigModule } from '@nestjs/config';
import { Module } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';

import { GraphQLModule } from '@nestjs/graphql';
import { TypeOrmModule } from '@nestjs/typeorm';

import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';

import { ItemsModule } from './items/items.module';
import { UsersModule } from './users/users.module';
import { AuthModule } from './auth/auth.module';


@Module({
  imports: [

    ConfigModule.forRoot(),

    GraphQLModule.forRootAsync({
      driver: ApolloDriver,
      imports: [ AuthModule ],
      inject: [ JwtService ],
      useFactory: async( jwtService: JwtService ) => ({
        playground: true,
        autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
        plugins: [
          //ApolloServerPluginLandingPageLocalDefault
```

```
        ],
        context({ req }) {
          // const token = req.headers.authorization?.replace('Bearer ','');
          // if ( !token ) throw Error('Token needed');

          // const payload = jwtService.decode( token );
          // if ( !payload ) throw Error('Token not valid');

        }
      })
    }),

    // TODO: configuración básica
    // GraphQLModule.forRoot<ApolloDriverConfig>({
    //   driver: ApolloDriver,
    //   // debug: false,
    //   playground: false,
    //   autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
    //   plugins: [
    //     ApolloServerPluginLandingPageLocalDefault
    //   ]
    // }),

    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: +process.env.DB_PORT,
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      database: process.env.DB_NAME,
      synchronize: true,
      autoLoadEntities: true,
    }),

    ItemsModule,

    UsersModule,

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

- En la entidad de Item debo establecer la relación con los usuarios
- Si fuera una relación uno a uno significaría que solo tenemos 1 item relacionado con 1 usuario
- Muchos a uno significa que muchos artículos pueden estar asociados a una persona
- De uno a muchos significaría que un item puede tener muchos usuarios.
- **Tiene más sentido que muchos items pertenece a un usuario**
- Indico con qué entidad me voy a relacionar, y defino que campo es el que voy a usar para establecer la relación

- Pongo user.items (que todavía no existe)
- Indico que lo quiero indexado con **@Index** y le paso el nombre del campo
- Como puede ser que tenga miles de items y quiero consultarlo de una manera rápida
- Esto añade un índice para que cuando haga una consulta, sabe que voy a usar este campo para hacer la solicitud e ir más rápido
- Se pueden crear índice únicos cvon unique: true
- Se pueden hacer índices compuestos, usando dos o más columnas de la db. Es sencillo y cómodo con typeorm

```
@Index(["firstName", "lastName"], {unique:true})
```

- Debo indicarle el campo a GraphQL con **@Field** y le indico que va a retornar algo de tipo User
- El nullable en true dice que puede ser nulo, y el lazy en true me sirve como el eager para poder traer la info en la query desde apollo
- item.entity

```
import { ObjectType, Field, ID, Float } from '@nestjs/graphql';
import { Column, Entity, Index, ManyToOne, PrimaryGeneratedColumn } from
'typeorm';

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


@Entity({ name: 'items' })
@ObjectType()
export class Item {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  name: string;

   @Column()
   @Field( () => Float )
   quantity: number;


  @ManyToOne( () => User, (user) => user.items, { nullable: false, lazy: true })
  @Index('userId-index')
  @Field( () => User )
  user: User;

}
```

- Debo establecer el campo y la relación en user.entity

- Yo debo saber el usuario propietario de este item
- Del lado del usuario, un usuario puede tener muchos items -** En el primer argumento le indico con una función de flecha con qué se va a relacionar**
- **En el segundo establezco la relación con el campo de la entidad, y le añado en un objeto las opciones**
- Si en este objeto no le digo que es nullable, **siempre va a tener un valor**
- Debo indicar el Field con el tipo para gql
- Coloco Item entre llaves cuadradas porque va a ser un arreglo
- No lo pongo opcional porque siempre voy a tener un valor, si quisiera eso podría devolver un arreglo vacío, pero no es el caso

```
import { ObjectType, Field, Int, ID } from '@nestjs/graphql';
import { Column, Entity, JoinColumn, ManyToOne, OneToMany, PrimaryGeneratedColumn
} from 'typeorm';

import { Item } from './../../items/entities/item.entity';


@Entity({ name: 'users' })
@ObjectType()
export class User {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  fullName: string;

  @Column({ unique: true })
  @Field( () => String )
  email: string;

  @Column()
  // @Field( () => String )
  password: string;

  @Column({
    type: 'text',
    array: true,
    default: ['user']
  })
  @Field( () => [String] )
  roles: string[]

  @Column({
    type: 'boolean',
    default: true
  })
  @Field( () => Boolean )
```

```
  isActive: boolean;
                //el lazy está en true para poder cargar la info desde la query
de user con items anidados
  @ManyToOne( () => User, (user) => user.lastUpdateBy, { nullable: true, lazy:
true })
  @JoinColumn({ name: 'lastUpdateBy' })
  @Field( () => User, { nullable: true })
  lastUpdateBy?: User;

  //Aqui tengo la relación de uno a muchos con item!
  @OneToMany( () => Item, (item) => item.user, { lazy: true })
  @Field( () => [Item] )
  items: Item[];

}
```

- Vamos a tener ciertos problemas porque hemos hecho modificaciones tanto en la parte de usuarios como en la parte de itemsw
- Cuando pedíamos un item, en la parte de los dto/inputs, para crear un item le pedíamso la cantidad
- Ya no se la vamos a pedir, no es parte del item la cantidad
- Si la cantidad de uniddades!

```
import { InputType, Field, Float } from '@nestjs/graphql';
import { IsNotEmpty, IsOptional, IsPositive, IsString } from 'class-validator';

@InputType()
export class CreateItemInput {

  @Field( () => String )
  @IsNotEmpty()
  @IsString()
  name: string;

  // @Field( () => Float )
  // @IsPositive()
  // quantity: number;

  @Field( () => String, { nullable: true })
  @IsString()
  @IsOptional()
  quantityUnits?: string;
}
```

- En la entity item, ya no tengo quantity y debo agregar las unidades

```
import { ObjectType, Field, ID, Float } from '@nestjs/graphql';
import { Column, Entity, Index, ManyToOne, PrimaryGeneratedColumn } from
'typeorm';
```

```
import { User } from './../../users/entities/user.entity';


@Entity({ name: 'items' })
@ObjectType()
export class Item {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  name: string;

  // @Column()
  // @Field( () => Float )
  // quantity: number;

  @Column({ nullable: true })
  @Field( () => String, { nullable: true } )
  quantityUnits?: string; // g, ml, kg, tsp

  // este campo de userId nunca debe de ser nulo, siempre debo poder asociar el
item a un usuario
  @ManyToOne( () => User, (user) => user.items, { nullable: false, lazy: true })
  @Index('userId-index')
  @Field( () => User )
  user: User;
}
```

CreateItem

- Miremos la query del items.resolver
- Ahora createItemInput solo me pide el name y quantityUnits
- Tengo que estar autenticado para hacer uso de ella!!
- El user llega al resolver a través de lo que hicimos con JwtAuthGuard y @CurrentUser usando el @UseGuards a nivel de resolver
- Para trabajar con items no necesito ser admin ni super-user, si estar logueado siendo user por default es suficiente
- Lo único que debo añadir al createItemInput es el user, por lo que desestructuro el createItemInput para esparcirlo (ya que el user no viene en él) y le paso el user

```
import { ParseUUIDPipe, UseGuards } from '@nestjs/common';
import { Resolver, Query, Mutation, Args, Int, ID } from '@nestjs/graphql';
import { JwtAuthGuard } from './../auth/guards/jwt-auth.guard';

import { ItemsService } from './items.service';
```

```
import { Item } from './entities/item.entity';
import { User } from './../users/entities/user.entity';

import { CreateItemInput, UpdateItemInput } from './dto/inputs';
import { CurrentUser } from '../auth/decorators/current-user.decorator';

@Resolver(() => Item)
@UseGuards( JwtAuthGuard )
export class ItemsResolver {
  constructor(private readonly itemsService: ItemsService) {}

  @Mutation(() => Item, { name: 'createItem' })
  async createItem(
    @Args('createItemInput') createItemInput: CreateItemInput,
    @CurrentUser() user: User
  ): Promise<Item> {
    return this.itemsService.create( createItemInput, user );
  }


}
```

- Puedo hacer una mutation y pedir toda la información del user (email, fullName) porque es @CurrentUser quien se encarga de pasarlo a la request
- Echémosle un ojo a auth/decorators/CurrentUser

```
import { createParamDecorator, ExecutionContext, ForbiddenException,
InternalServerErrorException } from '@nestjs/common';
import { GqlExecutionContext } from '@nestjs/graphql';
import { User } from '../../users/entities/user.entity';
import { ValidRoles } from '../enums/valid-roles.enum';



export const CurrentUser = createParamDecorator(
    ( roles: ValidRoles[] = [], context: ExecutionContext  ) => {


        const ctx = GqlExecutionContext.create( context );
        const user: User = ctx.getContext().req.user;

        if ( !user ) {
            throw new InternalServerErrorException(`No user inside the request -
make sure that we used the AuthGuard`);
        }

        if ( roles.length === 0 ) return user;

        for ( const role of user.roles ) {

            if ( roles.includes( role as ValidRoles ) ) {
```

```
                return user; //aquí lo establecemos en la request
            }
        }

        throw new ForbiddenException(
            `User ${ user.fullName } need a valid role [${ roles }]`
        )

})
```

- En el items.service

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { CreateItemInput, UpdateItemInput } from './dto/inputs';
***
import { User } from './../users/entities/user.entity';
import { Item } from './entities/item.entity';

@Injectable()
export class ItemsService {

  constructor(
    @InjectRepository( Item )
    private readonly itemsRepository: Repository<Item>,

  ) {}


  async create( createItemInput: CreateItemInput, user: User ): Promise<Item> {

    const newItem = this.itemsRepository.create({ ...createItemInput, user })
    return await this.itemsRepository.save( newItem );
  }

}
```

- Para **findAll** en items.resolver

```
@Query(() => [Item], { name: 'items' })
async findAll(
  @CurrentUser() user: User
): Promise<Item[]> {
  return this.itemsService.findAll( user );
}
```

- En el items.service busco los items que pertenecen al usuario

- Uso el where para establecer la condición y le digo que el id de user es el user.id

```
async findAll( user: User ): Promise<Item[]> {
  // TODO: filtrar, paginar, por usuario...
  return this.itemsRepository.find({
    where: {
      user: {
        id: user.id
      }
    }
  });
}
```

- Para **findOne** y **removeItem**
- No deberíamos poder borrar un item que sea parte de una lista o si se borra un item borrar la lista (luego lo veremos)
- En el remove todavía no he hecho el borrado lógico (soft delete), ya se hará

```
@Query(() => Item, { name: 'item' })
async findOne(
  @Args('id', { type: () => ID }, ParseUUIDPipe ) id: string,
  @CurrentUser() user: User
): Promise<Item> {
  return this.itemsService.findOne(id, user );
}

@Mutation(() => Item)
removeItem(
  @Args('id', { type: () => ID }) id: string,
  @CurrentUser() user: User
): Promise<Item> {
  return this.itemsService.remove(id, user);
}
```

- En el items.service para el findOne le paso el id, y el id del user

```
 async findOne( id: string, user: User ): Promise<Item> {

    const item = await this.itemsRepository.findOneBy({
      id,          //de esta manera tienen que cumplirse las dos condiciones
      user: {
        id: user.id  //también podría usar el andwhere
      }
    });

    if ( !item ) throw new NotFoundException(`Item with id: ${ id } not found`);

    // item.user = user;
```

```
    return item;
  }

  async remove( id: string, user: User ):Promise<Item> {

    // TODO: soft delete, integridad referencial
    const item = await this.findOne( id, user );
    await this.itemsRepository.remove( item );
    return { ...item, id };
  }
```

- Puedo traerme la información del user cuando hago la query desde apollo, porque tengo el lazy en true desde la entidad
- En **updateItem** de items.resolver

```
@Mutation(() => Item)
updateItem(
  @Args('updateItemInput') updateItemInput: UpdateItemInput,
  @CurrentUser() user: User
):Promise<Item> {
  return this.itemsService.update( updateItemInput.id, updateItemInput, user );
}
```

- En items.service, uso findOne, que lanzará unba excepción si no encuentra el item por el id de ese usuario
- Si el flujo del código continua, es que tengo el item, con lo cual le paso el updateItemInput
- Con el preload no puedo establecer el where ni nada, pero si le paso los campos hace la búsqueda automáticamente
- Uso .save para salvar los cambios

```
async update(id: string, updateItemInput: UpdateItemInput, user: User ):
Promise<Item> {

  await this.findOne( id, user );
  //? const item = await this.itemsRepository.preload({ ...updateItemInput, user
}); si no usara el lazy en true podría hacerlo así
  const item = await this.itemsRepository.preload( updateItemInput );

  if ( !item ) throw new NotFoundException(`Item with id: ${ id } not found`);

  return this.itemsRepository.save( item ); //salvo el item!

}
```

- El dto sigue igual

```
import { CreateItemInput } from './create-item.input';
import { InputType, Field, PartialType, ID } from '@nestjs/graphql';
import { IsUUID } from 'class-validator';

@InputType()
export class UpdateItemInput extends PartialType(CreateItemInput) {

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

}
```

- Podría crear consultas para hacer la paginación, buscar el más caro de los productos, todo en un solo query
- Los querys pueden ser muy grandes en graphQL
- Por ejemplo

```
{
  items(to:0, limit:120){
    name
  }
  items(to:120, limit:220){
    name
  }
  //buscar el más caro
  //hacer excepciones
  //etc
}
```

Items por usuario

- Tiene sentido que cree un itemCount desde los users para poder medir cuantos items quiero mostrar, ya que el usuario podría tener miles y eso podría ser demasiado pesado
- No tenemos paginaciones todavía
- En la entidad le coloco el lazy en la relación cOneToMany con los items, algo que deberemos deshabilitar después

```
@OneToMany( () => Item, (item) => item.user, { lazy: true })
@Field( () => [Item] )
items: Item[];
```

ResolveField con información del padre

- Para poder limitar el numero de items puedo crear un itemCount
- Sabiendo el usuario sería un simple SELECT * la_tabla WHERE user.id == USER
- Cómo lo quiero agregar a la query de usuarios, tiene sentido colocarlo en el users.resolver
- Con **@ResolveField** estoy modificando mi esquema diciéndole que voy a tener un nuevo campo, y **este es el método que va a usarse en este campo cuando sea solicitado**
- **@Parent** nos permite tener acceso a la información del padre (User)
- users.resolver

```
@ResolveField( () => Int, { name: 'itemCount' })
async itemCount(
  @CurrentUser([ ValidRoles.admin ]) adminUser: User,
  @Parent() user: User
): Promise<number> {
  return this.itemsService.itemCountByUser( user )
}
```

- Este resolveField está amarrado a mi usuario, si voy a la definición de Usuario en DOCS del Schema voy a tener itemCount
- En cualquier punto que tenga acceso al usuario puedo saber el itemCount
- Tiene más sentido que el conteo de los items esté en itemsService
- Uso el método count

```
async itemCountByUser( user: User ): Promise<number> {

  return this.itemsRepository.count({
    where: {
      user: {
        id: user.id
      }
    }
  })

}
```

- Para poder inyectar itemsService en users.resolver debo exponerllo en el módulo de items e imporarlo en el de users
- En items.module exporto el servicio

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common';

import { ItemsService } from './items.service';
import { ItemsResolver } from './items.resolver';
import { Item } from './entities/item.entity';

@Module({
```

```
  providers: [
    ItemsResolver,
    ItemsService
  ],
  imports: [
    TypeOrmModule.forFeature([ Item ])
  ],
  exports: [
    ItemsService,
    TypeOrmModule,
  ]
})
export class ItemsModule {}
```

- En users.module importo el módulo
- Si quisiera inyectar el repositorio debería importar el TypeOrmModule

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common';

import { ItemsModule } from './../items/items.module';

import { User } from './entities/user.entity';
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';

@Module({
  providers: [UsersResolver, UsersService],
  imports: [
    TypeOrmModule.forFeature([ User ]),
    ItemsModule,
  ],
  exports: [
    // TypeOrmModule, si quisiera inyectar el repositorio
    UsersService
  ]
})
export class UsersModule {}
```

06 Nest gRaphQL - Seed

- La variable de entorno STATE (ahora en dev) nos servirá para que cuando estemos en producción no podamos ejecutar el seed y nos destruya la DB
- Vamos a llenar la DB con usuarios e items
- Se agradece tener un seed y una documentación es bastante valioso

---

Seed Resolver

- Creamos el módulo de seed
- El seed se podría hacer como una query, pero tecnicamente es una mutación

> nest g res seed --no-spec

- Selecciono GraphQL code first sin endpoints
- Dentro de seed creo la carpeta data con la data a insertar

```
export const SEED_USERS = [
    {
        fullName: 'Fernando Herrera',
        email: 'fernando@google.com',
        password: '123456',
        roles: ['admin','superUser','user'],
        isActive: true
    },
    {
        fullName: 'Melissa Flores',
        email: 'melissa@google.com',
        password: '123456',
        roles: ['user'],
        isActive: true
    },
    {
        fullName: 'Hernando Vallejo',
        email: 'hernando@google.com',
        password: '123456',
        roles: ['user'],
        isActive: false
    },
]

export const SEED_ITEMS = [
    {
        name: "Chicken breast (skinless,boneless)",
        quantityUnits: "lb",
        category: "meat"
    },
    {
        name: "Chicken thighs (skinless,boneless)",
        quantityUnits: "box",
        category: "meat"
    },
    {
        name: "Fish filets",
        quantityUnits: "unit",
        category: "meat"
    },
    {
        name: "Ground turkey or chicken",
        quantityUnits: "lb",
        category: "meat"
```

```
    },
    {
        name: "Lean ground beef",
        quantityUnits: "pound",
        category: "meat"
    },
    {
        name: "Veggie burgers",
        quantityUnits: "box",
        category: "meat"
    },
    {
        name: "Chicken breast (skinless,boneless)",
        quantityUnits: "unit",
        category: "meat"
    },
    {
        name: "Chicken thighs (skinless,boneless)",
        quantityUnits: "box",
        category: "meat"
    },
    {
        name: "Chicken salad (made with lower calorie mayo)",
        quantityUnits: null,
        category: "meat"
    },
    {
        name: "Tuna salad (made with lower calorie mayo)",
        quantityUnits: null,
        category: "meat"
    },
    {
        name: "Egg salad (made with lower calorie mayo)",
        quantityUnits: "unit",
        category: "meat"
    },
    {
        name: "Lean ground beef",
        quantityUnits: "pound",
        category: "meat"
    },
    {
        name: "Ground turkey or chicken",
        quantityUnits: "pound",
        category: "meat"
    },
    {
        name: "Mixed vegetables",
        quantityUnits: "bag",
        category: "vegetables"
    },
    {
        name: "Brussels sprouts",
        quantityUnits: null,
```

```
            category: "vegetables"
        },
        {
            name: "Arugula",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Asparagus",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Broccoli",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Bell peppers (green, red, orange, yellow or roasted in a jar)",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Cabbage (green or red)",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Carrots",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Cauliflower",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Celery",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Corn",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Cucumber",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Eggplant",
```

```
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Endive",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Garlic",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Ginger",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Green beans (not canned)",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Green beans",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Green onion",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Jalapeños",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Kale",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Leeks",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Lettuce (iceberg, romaine)",
        quantityUnits: null,
        category: "vegetables"
    },
    {
```

```
            name: "Onions",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Parsley",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Peas (not canned)",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Peas",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Radicchio",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Radishes",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Shiitake mushrooms",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Summer squash (yellow)",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Turnip",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Turnip greens",
            quantityUnits: null,
            category: "vegetables"
        },
        {
            name: "Watercress",
            quantityUnits: null,
            category: "vegetables"
        },
```

```
    {
        name: "Zucchini",
        quantityUnits: null,
        category: "vegetables"
    },
    {
        name: "Apples",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Blackberries",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Blueberries",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Cherries",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Fruit cocktail (not packed in syrup)",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Grapes",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Lemons",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Limes",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Peaches (not packed in syrup)",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Pears",
        quantityUnits: null,
        category: "fruits"
```

```
    },
    {
        name: "Pineapple",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Plums",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Raspberries",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Strawberries",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Tangerine",
        quantityUnits: null,
        category: "fruits"
    },
    {
        name: "Rice: Brown, basmati, or jasmine",
        quantityUnits: null,
        category: "grains"
    },
    {
        name: "Cereals: corn flakes, chex, rice",
        quantityUnits: null,
        category: "grains"
    },
    {
        name: "krispies, puffed rice, puffed",
        quantityUnits: null,
        category: "grains"
    },
    {
        name: "Couscous",
        quantityUnits: null,
        category: "grainsits"
    },
    {
        name: "Oatmeal",
        quantityUnits: null,
        category: "grainsuits"
    },
    {
        name: "Cream of wheat",
        quantityUnits: null,
```

```
            category: "grains"
        },
        {
            name: "Grits",
            quantityUnits: null,
            category: "grains"
        },
        {
            name: "Crackers (unsalted and without added phosphorus)",
            quantityUnits: null,
            category: "grains"
        },
        {
            name: "Pasta (whole wheat or white)",
            quantityUnits: null,
            category: "grains"
        },
        {
            name: "English muffins",
            quantityUnits: null,
            category: "bread"
        },
        {
            name: "Polenta",
            quantityUnits: null,
            category: "bread"
        },
        {
            name: "Whole wheat breads",
            quantityUnits: null,
            category: "bread"
        },
        {
            name: "Whole grain breads",
            quantityUnits: null,
            category: "bread"
        },
        {
            name: "Rye bread",
            quantityUnits: null,
            category: "bread"
        },
        {
            name: "Tortillas (without added phosphorus)",
            quantityUnits: null,
            category: "bread"
        },
        {
            name: "Sourdough bread",
            quantityUnits: null,
            category: "bread"
        },
        {
            name: "Parsley",
```

```
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Basil",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Oregano",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Garlic powder (not garlic salt)",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Black pepper",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Red pepper flakes",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Cayenne",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "No salt added chili powder",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Old Bay",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Cumin",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
            name: "Coriander",
            quantityUnits: null,
            category: "dried herbs and spices"
        },
        {
```

```
        name: "Thyme",
        quantityUnits: null,
        category: "dried herbs and spices"
    },
    {
        name: "Turmeric",
        quantityUnits: null,
        category: "dried herbs and spices"
    },
    {
        name: "Cinnamon",
        quantityUnits: null,
        category: "dried herbs and spices"
    },
    {
        name: "Curry powder",
        quantityUnits: null,
        category: "dried herbs and spices"
    },
    {
        name: "Chives",
        quantityUnits: null,
        category: "dried herbs and spices"
    },
    {
        name: "Ginger",
        quantityUnits: null,
        category: "dried herbs and spices"
    },
    {
        name: "Water",
        quantityUnits: 'gl',
        category: "beverages"
    },
    {
        name: "Coffee",
        quantityUnits: "bag",
        category: "beverages"
    },
    {
        name: "Tea",
        quantityUnits: "box",
        category: "beverages"
    },
    {
        name: "Sodas (Pepsi, Coke)",
        quantityUnits: 'cans',
        category: "beverages"
    },
    {
        name: "Canola oil or olive oil",
        quantityUnits: "bottle",
        category: "other"
    },
```

```
    {
        name: "Mayonnaise (low calorie)",
        quantityUnits: "bottle",
        category: "other"
    },
    {
        name: "Balsamic",
        quantityUnits: "bottle",
        category: "other"
    },
]
```

- En el seed.module importo el ItemsModule y el UsersModule

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';

import { ItemsModule } from './../items/items.module';
import { UsersModule } from './../users/users.module';

import { SeedService } from './seed.service';
import { SeedResolver } from './seed.resolver';

@Module({
  providers: [SeedResolver, SeedService],
  imports: [
    ConfigModule,
    ItemsModule,
    UsersModule,
  ]
})
export class SeedModule {}
```

- En el seed.resolver

```
import { Mutation, Resolver } from '@nestjs/graphql';
import { SeedService } from './seed.service';

@Resolver()
export class SeedResolver {

  constructor(private readonly seedService: SeedService) {}


  @Mutation( () => Boolean, { name: 'executeSeed', description: 'Ejecuta la
construcción de la base de datos' })
  async executeSeed(): Promise<boolean> {

    return this.seedService.executeSeed();
```

```
  }
}
```

- En el seedService inyecto los rerpositorios y los servicios
- Por ello tengo que exportar el service de items (y el TypeOrmModule si quiero inyectar el repositorio) y el de users
- Entonces, exporto los servicios, importo los módulos
- items.module

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common';

import { ItemsService } from './items.service';
import { ItemsResolver } from './items.resolver';
import { Item } from './entities/item.entity';

@Module({
  providers: [
    ItemsResolver,
    ItemsService
  ],
  imports: [
    TypeOrmModule.forFeature([ Item ])
  ],
  exports: [
    ItemsService,
    TypeOrmModule,
  ]
})
export class ItemsModule {}
```

- En user.module

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common';

import { ItemsModule } from './../items/items.module';

import { User } from './entities/user.entity';
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';

@Module({
  providers: [UsersResolver, UsersService],
  imports: [
    TypeOrmModule.forFeature([ User ]),
    ItemsModule,
  ],
  exports: [
```

```
    TypeOrmModule,
    UsersService
  ]
})
export class UsersModule {}
```

- En el seedService importo también el ConfigModule porque usaré la variable de entorno para saber si estoy en producción y no ejecutar el seed
- Debo borrar primero los items por la integridad referencial
- Para el seed, los items necesitan los usuarios para ser insertados
- Necesito los repositorios para crear los builders para impactar en la DB, i borrar los registros
- Para insertar usuarios SEED_USERS lo recorro con un for y uso el push para meterle cada usuario usando el servicio
- Para insertar items importo SEED_ITEMS, no voy a usar el await y usaré el Promise.all

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';

import { SEED_USERS, SEED_ITEMS } from './data/seed-data';

import { Item } from './../items/entities/item.entity';
import { User } from './../users/entities/user.entity';

import { ItemsService } from '../items/items.service';
import { UsersService } from './../users/users.service';


@Injectable()
export class SeedService {

    private isProd: boolean;

    constructor(
        private readonly configService: ConfigService,

        @InjectRepository(Item)
        private readonly itemsRepository: Repository<Item>,

        @InjectRepository(User)
        private readonly usersRepository: Repository<User>,

        private readonly usersService: UsersService,
        private readonly itemsService: ItemsService,

    ) {
        this.isProd = configService.get('STATE') === 'prod'; //obtengo la variable
isProd si esta es igual a prod
    }
```

```
    async executeSeed() {

        if ( this.isProd ) {
            throw new UnauthorizedException('We cannot run SEED on Prod');
        }

        // Limpiar la base de datos BORRAR TODO
        await this.deleteDatabase();

        // Crear usuarios
        const user = await this.loadUsers();

        // Crear items
        await this.loadItems( user );



        return true;
    }

    async deleteDatabase() {

        // borrar items
        await this.itemsRepository.createQueryBuilder()
            .delete()
            .where({})
            .execute();

        // borrar users
        await this.usersRepository.createQueryBuilder()
            .delete()
            .where({})
            .execute();

    }

    async loadUsers(): Promise<User> {

        const users = [];

        for (const user of SEED_USERS ) {
            users.push( await this.usersService.create( user ) )
        }

        return users[0]; //He dicho que devolvería una promesa de tipo User!
retorno el primer resultado del arreglo

    }


    async loadItems( user: User ): Promise<void> { //Aquí devuelvo una Promesa
vacía (no hay return)
```

```
        const itemsPromises = [];

        for (const item of SEED_ITEMS ) {
            itemsPromises.push( this.itemsService.create( item, user ) ); //al no
usar el await usaré el Promise.all
        }

        await Promise.all( itemsPromises );
    }
}
```

- Para ejecutar el seed

```
mutation executeSeed{
  executeSeed
}
```

README

````
<p align="center">
  <a href="http://nestjs.com/" target="blank"><img
src="https://nestjs.com/img/logo-small.svg" width="200" alt="Nest Logo" /></a>
</p>

# Dev

1. Clonar el proyecto
2. Copiar el ```env.template``` y renombar a ```.env```
3. Ejecutar
```
yarn install
```
4. Levantar la imagen (Docker desktop)
```
docker-compose up -d
```


5. Levantar el backend de Nest
```
yarn start:dev
```


6. Visiar el sitio
```
localhost:3000/graphql
```
````

```
7. Ejecutar la __"mutation"__ executeSeed, para llenar la base de datos con
información
```

07 Nest GraphQL - Paginaciones

- Vamos a trabajar con maestro detalles, muy relacionado con índices, paginaciones, llaves únicas, straingths compuestos
- Es el ejercicio más complejo que hemos hecho hasta ahora, porque hay muchas inserciones, actualizaciones, maneras de trabajar
- Les daremos un montón de flexibilidad a nuestros querys
- Trabajaremos con un usuario con role de admin autenticado

```
mutation login($loginInput: LoginInput!){
  login(loginInput: $loginInput){
    user{
      fullName
    }
    token
  }
}
```

- En las variables:

```
{
  "loginInput": {
    "email": "fernando@google.com",
    "password": "123456"
  }
}
```

- Copio el token y lo pego en HTTP Headers del playground de Apollo

```
{
  "Authorization": "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjFjMTAyMTJhLTIwNzctNGIyMS04YTYxLTE0Y
WI4MzE4MTExYyIsImlhdCI6MTcxODUxODY2NSwiZXhwIjoxNzE4NTMzMDY1fQ.GSXFcnVdUMv0AA8JntMN
X816CbXUMhV1i1KGtHMpN1g"
}
```

Paginar resultados

- No queremos un volcado completo de la data, pierde el sentido de graphQL

- Uso limit para indicar el número de resultados y offset para decirle el número de items que quiero que se salte primero
- Esto viene por los @Args
- No le pongo el ? para hecrlo opcional para TypeScript porque siempre va a tener un valor (por defecto)
- Creo la carpeta common/dto/pagination.args

```
import { ArgsType, Field, Int } from '@nestjs/graphql';
import { IsOptional, Min } from 'class-validator';


@ArgsType()
export class PaginationArgs {

    @Field( () => Int, { nullable: true })
    @IsOptional()
    @Min(0)
    offset: number = 0;

    @Field( () => Int, { nullable: true })
    @IsOptional()
    @Min(1)
    limit: number = 10;

}
```

- Vamos al items.resolver

```
@Query(() => [Item], { name: 'items' })
async findAll(
@CurrentUser() user: User,
@Args() paginationArgs: PaginationArgs,
): Promise<Item[]> {

return this.itemsService.findAll( user, paginationArgs);
}
```

- En el servicio, - El user.id es un campo compuesto que hay que especificar como se detalla aqui

```
async findAll( user: User, paginationArgs: PaginationArgs, searchArgs: SearchArgs
): Promise<Item[]> {

const { limit, offset } = paginationArgs;
const { search } = searchArgs;

const queryBuilder = this.itemsRepository.createQueryBuilder()
    .take( limit ) //take toma una cantidad de registros
    .skip( offset ) //uso el skip para saltar registros y le mando el offset
    .where(`"userId" = :userId`, { userId: user.id }); //le digo que el userId es
```

```
igual al parámetro que le paso
                                                        //luego le indico que el
valor de este argumento :userId es igual al del id del user que he recibido como
parámetro


return queryBuilder.getMany();
}
```

- El query sería

```
query findAll{
    items (limit:200, offset:0){
    name
  }
}
```

- El dto searchArgs me servirá para buscar por alguna palabra en concreto

```
import { ArgsType, Field } from '@nestjs/graphql';
import { IsOptional, IsString } from 'class-validator';

@ArgsType()
export class SearchArgs {

    @Field( ()=> String, { nullable: true })
    @IsOptional()
    @IsString()
    search?: string;
}
```

- Lo agrego al items.resolver

```
@Query(() => [Item], { name: 'items' })
async findAll(
@CurrentUser() user: User,
@Args() paginationArgs: PaginationArgs,
@Args() searchArgs: SearchArgs,
): Promise<Item[]> {

return this.itemsService.findAll( user, paginationArgs, searchArgs );
}
```

- Si coloco el serachArgs antes que el paginationArgs me salta un error.
- Es por algo del class-validator que voy a tener que desactivar en el main

- El forbidNonWhitelisted sirve para que ignorar cuando me mandan más información de la que yo espero en el endpoint
- GraphQl se va a encargar de hacer esta validación por mi
- main.ts

```
import { NestFactory } from '@nestjs/core';
import { ValidationPipe } from '@nestjs/common';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      // forbidNonWhitelisted: true, hay que comentarla
    })
  );

  await app.listen(3000);
}
bootstrap();
```

- En el items.service uso el Like para que el search incluya o sea similar al parámetro
- Para usar este Like (de la consulta que hay comentada) debería crear un índice especializado
- Los comodines % son para que no importe lo que hay antes y después de la palabra que haya en search
- O grabo todo en lowerCase en la DB o formateo
- Puedo hacerlo como un queryBuilder que es mejor opción
- En el queryBuilder uso LOWER para reducir a minúsculas el name, uso el like para asemejarlo al parámetro de search que quiero como name con :name. Lo manejo como una variable porque necesito estar seguro de que no lo estoy inyectando directamente en la sentencia SQL
- En un objeto le paso en un template string el search que paso a minúsculas y coloco los comodines

```
async findAll( user: User, paginationArgs: PaginationArgs, searchArgs: SearchArgs
): Promise<Item[]> {

const { limit, offset } = paginationArgs;
const { search } = searchArgs;

const queryBuilder = this.itemsRepository.createQueryBuilder()
    .take( limit )
    .skip( offset )
    .where(`"userId" = :userId`, { userId: user.id }); //le digo que el userId es
igual al parámetro que le paso
                                                       //luego le indico que el
valor de este argumento :userId es igual al del id del user que he recibido como
parámetro
```

```
if ( search ) {    //
    queryBuilder.andWhere('LOWER(name) like :name', { name: `%${
search.toLowerCase() }%` });
}

return queryBuilder.getMany();
// return this.itemsRepository.find({
//   take: limit,
//   skip: offset,
//   where: {
//     user: {
//       id: user.id
//     },
//     name: Like(`%${ search.toLowerCase() }%`) quiero que el name sea algo
similar o incluya algo como el search, por eso uso Like
//   }
// });
}
```

- Si quiero hacer la consulta con el search

```
query findAll{
    items (limit:200, offset:0, search: "Rice"){
    name
  }
}
```

- Puedo hacer consultas incluyendo el user

```
query findAll{
    items (limit:200, offset:0, search: "Rice"){
    name
    user{
      itemCount
    }
  }
}
```

- Si quiero hacer la consulta de items desde el users ya no funciona por la relación que tenemos
- Nuestros usuarios tienen el campo items que está asociado con la DB
- Quiero romper esa relación automática y definir la forma en la que quiero que estos items se construyan y no decirle a typeorm que lo haga
- Ahora quiero que desde usuarios, automáticamente me cargue un número de items (una paginación)
- de esta manera, en graphQL creamos querys y mutaciones, con campos que podemos ir añadiendo
- Añado paginationArgs y searchArgs al método de users.resolver
- users.resolver

```
  @ResolveField( () => [Item], { name: 'items' })
  async getItemsByUser(
    @CurrentUser([ ValidRoles.admin ]) adminUser: User,
    @Parent() user: User,
    @Args() paginationArgs: PaginationArgs,
    @Args() searchArgs: SearchArgs,
  ): Promise<Item[]> {
    return this.itemsService.findAll( user, paginationArgs, searchArgs );
  }
```

- En el items.service usaré el findAll

Nest GraphQL - Entidad para el manejo de listas Maestro Detalle

- Creo el módulo Lists y armo el lists.resolver
- lists.resolver

```
import { UseGuards, ParseUUIDPipe } from '@nestjs/common';
import { Resolver, Query, Mutation, Args, Int, ID, ResolveField, Parent } from
'@nestjs/graphql';

import { ListsService } from './lists.service';
import { ListItemService } from './../list-item/list-item.service';

import { JwtAuthGuard } from './../auth/guards/jwt-auth.guard';

import { List } from './entities/list.entity';
import { ListItem } from './../list-item/entities/list-item.entity';
import { User } from './../users/entities/user.entity';

import { CurrentUser } from '../auth/decorators/current-user.decorator';

import { PaginationArgs, SearchArgs } from './../common/dto/args';
import { CreateListInput } from './dto/create-list.input';
import { UpdateListInput } from './dto/update-list.input';

@Resolver(() => List)
@UseGuards( JwtAuthGuard )
export class ListsResolver {

  constructor(
    private readonly listsService: ListsService,
    private readonly listItemsService: ListItemService
  ) {}

  @Mutation(() => List)
```

```
  async createList(
    @Args('createListInput') createListInput: CreateListInput,
    @CurrentUser() user: User
  ):Promise<List> {
    return this.listsService.create( createListInput, user );
  }

  @Query(() => [List], { name: 'lists' })
  async findAll(
    @CurrentUser() user: User,
    @Args() paginationArgs: PaginationArgs,
    @Args() searchArgs: SearchArgs,
  ):Promise<List[]> {
    return this.listsService.findAll(user, paginationArgs, searchArgs );
  }

  @Query(() => List, { name: 'list' })
  async findOne(
    @Args('id', { type: () => ID }, ParseUUIDPipe ) id: string,
    @CurrentUser() user: User
  ): Promise<List> {
    return this.listsService.findOne( id, user );
  }

  @Mutation(() => List)
  updateList(
    @Args('updateListInput') updateListInput: UpdateListInput,
    @CurrentUser() user: User
  ): Promise<List> {
    return this.listsService.update(updateListInput.id, updateListInput, user );
  }

  @Mutation(() => List)
  removeList(
    @Args('id', { type: () => ID }) id: string,
    @CurrentUser() user: User
  ) {
    return this.listsService.remove( id, user );
  }

  @ResolveField( () => [ListItem], { name: 'items' } )
  async getListItems(
    @Parent() list: List,
    @Args() paginationArgs: PaginationArgs,
    @Args() searchArgs: SearchArgs,
  ): Promise<ListItem[]> {

    return this.listItemsService.findAll( list, paginationArgs, searchArgs );
  }

  @ResolveField( () => Number, { name: 'totalItems' } )
  async countListItemsByList(
    @Parent() list: List,
  ): Promise<number> {
```

```
    return this.listItemsService.countListItemsByList( list );
  }

}
```

- create-list.input

```
import { InputType, Int, Field } from '@nestjs/graphql';
import { IsNotEmpty, IsString } from 'class-validator';

@InputType()
export class CreateListInput {

  @Field(() => String )
  @IsString()
  @IsNotEmpty()
  name: string;

}
```

- update-list.input

```
import { CreateListItemInput } from './create-list-item.input';
import { InputType, Field, Int, PartialType, ID } from '@nestjs/graphql';
import { IsUUID } from 'class-validator';

@InputType()
export class UpdateListItemInput extends PartialType(CreateListItemInput) {

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

}
```

- Necesitaré también el list-items. Creo el módulo (GraphQl Code first).
- En lists.module importo el listItemModule para usar el servicio. Exporto el TypeOrmModule opara inyectar su repositorio en otro módulo

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common';
import { ListsService } from './lists.service';
import { ListsResolver } from './lists.resolver';

import { ListItemModule } from './../list-item/list-item.module';

import { List } from './entities/list.entity';
```

```
@Module({
  providers: [ListsResolver, ListsService],
  imports: [
    TypeOrmModule.forFeature([ List ]),
    ListItemModule,
  ],
  exports: [
    TypeOrmModule,
    ListsService,
  ]
})
export class ListsModule {}
```

- Para inyectar los servicios debo importa también los módulos en el list-item.module y exportar los servicios en los otros módulos

```
import { TypeOrmModule } from '@nestjs/typeorm';
import { Module } from '@nestjs/common'
;
import { ListItem } from './entities/list-item.entity';

import { ListItemService } from './list-item.service';
import { ListItemResolver } from './list-item.resolver';

@Module({
  providers: [ListItemResolver, ListItemService],
  imports: [
    TypeOrmModule.forFeature([ ListItem ])
  ],
  exports: [
    ListItemService, TypeOrmModule
  ]
})
export class ListItemModule {}
```

- Este es el resolver de list-item
- list-item.resolver

```
import { Resolver, Query, Mutation, Args, Int } from '@nestjs/graphql';
import { UseGuards, ParseUUIDPipe } from '@nestjs/common';

import { JwtAuthGuard } from './../auth/guards/jwt-auth.guard';

import { ListItemService } from './list-item.service';
import { ListItem } from './entities/list-item.entity';

import { CreateListItemInput } from './dto/create-list-item.input';
import { UpdateListItemInput } from './dto/update-list-item.input';
```

```
@Resolver(() => ListItem)
@UseGuards( JwtAuthGuard )
export class ListItemResolver {

  constructor(private readonly listItemService: ListItemService) {}

  @Mutation(() => ListItem)
  createListItem(
    @Args('createListItemInput') createListItemInput: CreateListItemInput,
    //! Todo pueden pedir el usuario para validarlo
  ): Promise<ListItem> {
    return this.listItemService.create(createListItemInput);
  }

  // @Query(() => [ListItem], { name: 'listItem' })
  // findAll() {
  //   return this.listItemService.findAll();
  // }

  @Query( () => ListItem, { name: 'listItem' })
  async findOne(
    @Args('id', { type: () => String }, ParseUUIDPipe ) id: string
  ): Promise<ListItem> {
    return this.listItemService.findOne(id);
  }

  @Mutation(() => ListItem)
  async updateListItem(
    @Args('updateListItemInput') updateListItemInput: UpdateListItemInput
  ): Promise<ListItem> {
    return this.listItemService.update( updateListItemInput.id,
updateListItemInput );
  }

  // @Mutation(() => ListItem)
  // removeListItem(@Args('id', { type: () => Int }) id: number) {
  //   return this.listItemService.remove(id);
  // }
}
```

- La entidad de Lists es esta
- list.entity

```
import { ObjectType, Field, Int, ID } from '@nestjs/graphql';
import { Column, Entity, Index, ManyToOne, OneToMany, PrimaryGeneratedColumn }
from 'typeorm';

import { ListItem } from './../../list-item/entities/list-item.entity';
import { User } from './../../users/entities/user.entity';
```

```
@Entity({ name: 'lists' })
@ObjectType()
export class List {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column()
  @Field( () => String )
  name: string;

  // Relación, index('userId-list-index')
  @ManyToOne( () => User, (user) => user.lists, { nullable: false, lazy: true  })
  @Index('userId-list-index')
  @Field( () => User )
  user: User;

  @OneToMany( () => ListItem, (listItem) => listItem.list, { lazy: true })
  // @Field( () => [ListItem] )
  listItem: ListItem[];

}
```

- La entity de list-item
- Un constrain es una regla de validación.
- Necesitamos el ListId para poder insertar un ListItem

```
import { ObjectType, Field, Int, ID } from '@nestjs/graphql';
import { Column, Entity, ManyToOne, PrimaryGeneratedColumn, Unique } from
'typeorm';

import { Item } from './../../items/entities/item.entity';
import { List } from './../../lists/entities/list.entity';

@Entity('listItems')
@Unique('listItem-item', ['list','item']) //los constrains son reglas de
validación para la DB, primero se añade la entidad y luego el decorador
@ObjectType()
export class ListItem {

  @PrimaryGeneratedColumn('uuid')
  @Field( () => ID )
  id: string;

  @Column({ type: 'numeric' })
  @Field( () => Number )
  quantity: number;

  @Column({ type: 'boolean' })
  @Field( () => Boolean )
```

```
  completed: boolean;


  // Relaciones
  @ManyToOne( () => List, (list) => list.listItem, { lazy: true })
  @Field( () => List )
  list: List;

  @ManyToOne( () => Item, (item)=> item.listItem, { lazy: true })
  @Field( () => Item )
  item: Item;

}
```

- El servicio de list
- list.service

```
import { Injectable, NotFoundException } from '@nestjs/common';

import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';

import { List } from './entities/list.entity';
import { User } from './../users/entities/user.entity';

import { CreateListInput } from './dto/create-list.input';
import { UpdateListInput } from './dto/update-list.input';
import { PaginationArgs, SearchArgs } from '../common/dto/args';

@Injectable()
export class ListsService {

  constructor(
    @InjectRepository( List )
    private readonly listsRepository: Repository<List>
  ) {}

  async create(createListInput: CreateListInput, user: User ): Promise<List> {

    const newList = this.listsRepository.create({ ...createListInput, user })
    return await this.listsRepository.save( newList );

  }

  async findAll( user: User, paginationArgs: PaginationArgs, searchArgs:
SearchArgs ): Promise<List[]> {

    const { limit, offset } = paginationArgs;
    const { search } = searchArgs;

    const queryBuilder = this.listsRepository.createQueryBuilder()
```

```
      .take( limit )
      .skip( offset )
      .where(`"userId" = :userId`, { userId: user.id });

    if ( search ) {
      queryBuilder.andWhere('LOWER(name) like :name', { name: `%${
search.toLowerCase() }%` });
    }

    return queryBuilder.getMany();
  }

  async findOne( id: string, user: User ): Promise<List> {

    const list = await this.listsRepository.findOneBy({
      id,
      user: { id: user.id }
    });

    if ( !list ) throw new NotFoundException(`List with id: ${ id } not found`);

    return list;
  }

  async update(id: string, updateListInput: UpdateListInput, user: User ):
Promise<List> {

    await this.findOne( id, user );

    const list = await this.listsRepository.preload({ ...updateListInput, user });

    if ( !list ) throw new NotFoundException(`List with id: ${ id } not found`);

    return this.listsRepository.save( list );

  }

  async remove(id: string, user: User ): Promise<List> {

     const list = await this.findOne( id, user );
     await this.listsRepository.remove( list );
     return { ...list, id };
  }

  async listCountByUser( user: User ): Promise<number> {

    return this.listsRepository.count({
      where: {
        user: { id: user.id }
      }
    });

  }
```

```
}
```

- El servicio de list-item

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';

import { List } from './../lists/entities/list.entity';
import { ListItem } from './entities/list-item.entity';

import { PaginationArgs, SearchArgs } from './../common/dto/args';

import { CreateListItemInput } from './dto/create-list-item.input';
import { UpdateListItemInput } from './dto/update-list-item.input';

@Injectable()
export class ListItemService {

  constructor(

    @InjectRepository( ListItem )
    private readonly listItemsRepository: Repository<ListItem>,

  ) {}


  async create(createListItemInput: CreateListItemInput): Promise<ListItem> {

    const { itemId, listId, ...rest } = createListItemInput;

    const newListItem = this.listItemsRepository.create({
      ...rest,
      item: { id: itemId },
      list: { id: listId }
    });

    await this.listItemsRepository.save( newListItem );

    return this.findOne( newListItem.id );
  }

  async findAll( list: List, paginationArgs: PaginationArgs, searchArgs:
SearchArgs ): Promise<ListItem[]> {

    const { limit, offset } = paginationArgs;
    const { search } = searchArgs;

    const queryBuilder = this.listItemsRepository.createQueryBuilder('listItem')
// <-- Nombre para las relaciones
      .innerJoin('listItem.item','item') // <--- Lo añadí después, fue un problema
```

```
que no grabé
      .take( limit )
      .skip( offset )
      .where(`"listId" = :listId`, { listId: list.id });

    if ( search ) {
      queryBuilder.andWhere('LOWER(item.name) like :name', { name: `%${
search.toLowerCase() }%` });
    }

    return queryBuilder.getMany();

  }

  async countListItemsByList( list: List ): Promise<number> {
    return this.listItemsRepository.count({
      where: { list: { id: list.id }}
    });
  }

  async findOne(id: string): Promise<ListItem> {
    const listItem = await this.listItemsRepository.findOneBy({ id });

    if ( !listItem ) throw new NotFoundException(`List item with id ${ id } not
found`);

    return listItem;
  }

  async update(
    id: string, updateListItemInput: UpdateListItemInput
  ): Promise<ListItem> {

    const { listId, itemId, ...rest } = updateListItemInput;

    const queryBuilder = this.listItemsRepository.createQueryBuilder()
      .update()
      .set( rest )
      .where('id = :id', { id });

    if ( listId ) queryBuilder.set({ list: { id: listId } });
    if ( itemId ) queryBuilder.set({ item: { id: itemId } });

    await queryBuilder.execute();

    return this.findOne( id );

  }

  remove(id: number) {
    return `This action removes a #${id} listItem`;
  }
}
```

- create-list-item.input

```
import { InputType, Int, Field, ID } from '@nestjs/graphql';
import { IsBoolean, IsNumber, IsOptional, IsUUID, Min } from 'class-validator';

@InputType()
export class CreateListItemInput {

  @Field( () => Number, { nullable: true })
  @IsNumber()
  @Min(0)
  @IsOptional()
  quantity: number = 0;

  @Field( () => Boolean, { nullable: true })
  @IsBoolean()
  @IsOptional()
  completed: boolean = false;

  @Field( () => ID )
  @IsUUID()
  listId: string;

  @Field( () => ID )
  @IsUUID()
  itemId: string;
}
```

- update-list-item.input

```
import { CreateListItemInput } from './create-list-item.input';
import { InputType, Field, Int, PartialType, ID } from '@nestjs/graphql';
import { IsUUID } from 'class-validator';

@InputType()
export class UpdateListItemInput extends PartialType(CreateListItemInput) {

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

}
```

Dockerizar

- El objetivo de dockerizar la aplicación es tener la imagen lista para correr como si estuviera instalada

- Se puede hacer mediante un docker-compose.yml o un Dockerfile
- Vamos a tomar un linux, instalar Linux, instalar los paquetes de la aplicación, el build...
- Vamos a usar un gist de Fernando dónde tiene lo necesario para construir imágenes de Node
- docker-compose.prod.yml

```
version: '3'


services:
  db:
    image: postgres:14.4
    restart: always
    ports:
      - "${DB_PORT}:${DB_PORT}"
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    container_name: anylistDB
    volumes:
      - ./postgres:/var/lib/postgresql/data

  anylistapp:
    depends_on:
      - db
    build:
      context: .
      dockerfile: Dockerfile  # le indico el Dockerfile para el build
    image: nest-graphql
    container_name: AnylistApp
    restart: always # reiniciar el contenedor si se detiene
    ports:
      - "${PORT}:${PORT}"

    environment:
      DB_PASSWORD: ${DB_PASSWORD}
      DB_NAME: ${DB_NAME}
      DB_HOST: ${DB_HOST}
      DB_PORT: ${DB_PORT}
      DB_USERNAME: ${DB_USERNAME}
      JWT_SECRET: ${JWT_SECRET}
      PORT: ${PORT}
```

- DockerFile

```
# Install dependencies only when needed
# usamos la imagen de node de 5MB y le ponemos el nombre de deps
FROM node:18-alpine3.15 AS deps
# Check https://github.com/nodejs/docker-
node/tree/b4117f9333da4138b03a546ec926ef50a31506c3#nodealpine to understand why
libc6-compat might be needed.
```

```
# libc6-compat es para ciertos paquetes/procedimientos internos (opcional)
RUN apk add --no-cache libc6-compat
# trabajaremos en este directorio . Cualquier path relativo va a ser en este
directorio
WORKDIR /app

# copiamos el json de nuestro proyecto
COPY package.json yarn.lock ./
# congelamos el yarn-lock
RUN yarn install --frozen-lockfile


# Build the app with cache dependencies contenedor solo para construir la
aplicación
# otro FROm es otra etapa
FROM node:18-alpine3.15 AS builder
# trabajamos en /app
WORKDIR /app
# desde deps instalamos en el path app/node_modules los node-modules
COPY --from=deps /app/node_modules ./node_modules
# una vez copiados los modulos de node, le digo que copie todo lo que se encuentra
en nuestro proyecto
# hay que hacer algo para evitar que copie literalmente todo lo que hay en el
proyecto, porque eso no me interesa
# nop quiero que copie dist, los módulos de node por lo que creo un .dockerignore
COPY . .
# ejecuto el build
RUN yarn build


# Production image, copy all the files and run next
# usamos una nueva imagen de node limpia
FROM node:18-alpine3.15 AS runner

# Set working directory
# le indico el directorio
WORKDIR /usr/src/app

# copio el pckage.json y el yarn.lock y lo pegamos en el working directory
COPY package.json yarn.lock ./
# ejecuto el yarn install
RUN yarn install --prod
# copio  desde la imagen builder (anterior) el /app/dist en el directorio ./dist
COPY --from=builder /app/dist ./dist

# el comando para levantarlo (tambien se podria usar nest start)
CMD [ "node","dist/main" ]
```

- .dockerignore

```
dist/
node_modules/
```

```
postgres/

.git/
```

- Usaremos libc6-compat como librería para Node
- El Dockerfile esta dividio en tres etapas
    - Instalación de dependencias
    - Un contenedor solamente para construir la aplicación
    - El runner que hará correr la aplicación que es lo que terminamos ejecutando
- De esta manera, si no tenemos cambios en nuestras dependencias es una construcción mucho más rápida
- Para construir uso el markdown de BUILD

Build

docker-compose -f docker-compose.prod.yml --env-file .env.prod up --build

Run

docker-compose -f docker-compose.prod.yml --env-file .env.prod up

Nota

Por defecto, **docker-compose** usa el archivo `.env`, por lo que si tienen el archivo .env y lo configuran con sus variables de entorno de producción, bastaría con

```
docker-compose -f docker-compose.prod.yml up --build
```

Cambiar nombre

```
docker tag <nombre app> <usuario docker hub>/<nombre repositorio>
```

Ingresar a Docker Hub

```
docker login
```

Subir imagen

```
docker push <usuario docker hub>/<nombre repositorio>
```

Dockerizar

- El objetivo de dockerizar la aplicación es tener la imagen lista para correr como si estuviera instalada
- Se puede hacer mediante un docker-compose.yml o un Dockerfile
- Vamos a tomar un linux, instalar Linux, instalar los paquetes de la aplicación, el build...
- Vamos a usar un gist de Fernando dónde tiene lo necesario para construir imágenes de Node
- docker-compose.prod.yml

```
version: '3'


services:
  db:
    image: postgres:14.4
    restart: always
    ports:
      - "${DB_PORT}:${DB_PORT}"
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    container_name: anylistDB
    volumes:
      - ./postgres:/var/lib/postgresql/data

  anylistapp:
    depends_on:
      - db
    build:
      context: .
      dockerfile: Dockerfile  # le indico el Dockerfile para el build
    image: nest-graphql
    container_name: AnylistApp
    restart: always # reiniciar el contenedor si se detiene
    ports:
      - "${PORT}:${PORT}"

    environment:
      STATE: ${STATE}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_NAME: ${DB_NAME}
      DB_HOST: ${DB_HOST}
      DB_PORT: ${DB_PORT}
      DB_USERNAME: ${DB_USERNAME}
      JWT_SECRET: ${JWT_SECRET}
      PORT: ${PORT}
```

- DockerFile

```
# Install dependencies only when needed
# usamos la imagen de node de 5MB y le ponemos el nombre de deps
FROM node:18-alpine3.15 AS deps
```

```
# Check https://github.com/nodejs/docker-
node/tree/b4117f9333da4138b03a546ec926ef50a31506c3#nodealpine to understand why
libc6-compat might be needed.
# libc6-compat es para ciertos paquetes/procedimientos internos (opcional)
RUN apk add --no-cache libc6-compat
# trabajaremos en este directorio . Cualquier path relativo va a ser en este
directorio
WORKDIR /app

# copiamos el json de nuestro proyecto
COPY package.json yarn.lock ./
# congelamos el yarn-lock
RUN yarn install --frozen-lockfile


# Build the app with cache dependencies contenedor solo para construir la
aplicación
# otro FROm es otra etapa
FROM node:18-alpine3.15 AS builder
# trabajamos en /app
WORKDIR /app
# desde deps instalamos en el path app/node_modules los node-modules
COPY --from=deps /app/node_modules ./node_modules
# una vez copiados los modulos de node, le digo que copie todo lo que se encuentra
en nuestro proyecto
# hay que hacer algo para evitar que copie literalmente todo lo que hay en el
proyecto, porque eso no me interesa
# nop quiero que copie dist, los módulos de node por lo que creo un .dockerignore
COPY . .
# ejecuto el build
RUN yarn build


# Production image, copy all the files and run next
# usamos una nueva imagen de node limpia
FROM node:18-alpine3.15 AS runner

# Set working directory
# le indico el directorio
WORKDIR /usr/src/app

# copio el pckage.json y el yarn.lock y lo pegamos en el working directory
COPY package.json yarn.lock ./
# ejecuto el yarn install
RUN yarn install --prod
# copio  desde la imagen builder (anterior) el /app/dist en el directorio ./dist
COPY --from=builder /app/dist ./dist

# el comando para levantarlo (tambien se podria usar nest start)
CMD [ "node","dist/main" ]
```

- .dockerignore

```
dist/
node_modules/
postgres/

.git/
```

- Usaremos libc6-compat como librería para Node
- El Dockerfile esta dividio en tres etapas
    - Instalación de dependencias
    - Un contenedor solamente para construir la aplicación
    - El runner que hará correr la aplicación que es lo que terminamos ejecutando
- De esta manera, si no tenemos cambios en nuestras dependencias es una construcción mucho más rápida
- Para construir uso el markdown de BUILD
- Puedo crear un .env.prod para tener variables de producción independientes (lo añado al .gitignore)

Build

docker-compose -f docker-compose.prod.yml --env-file .env.prod up --build

Run

docker-compose -f docker-compose.prod.yml --env-file .env.prod up

Nota

Por defecto, **docker-compose** usa el archivo `.env`, por lo que si tienen el archivo .env y lo configuran con sus variables de entorno de producción, bastaría con

```
docker-compose -f docker-compose.prod.yml up --build
```

Cambiar nombre

```
docker tag <nombre app> <usuario docker hub>/<nombre repositorio>
```

Ingresar a Docker Hub

```
docker login
```

Subir imagen

```
docker push <usuario docker hub>/<nombre repositorio>
```

Resolver bcrypt y aceptar conexiones

- Hubo un error con bcrypt
- Desinstalamos bcrypt y @types/bcrypt
- Esto da error en la aplicación en users.service y auth.service
- Instalamos bcryptjs y @types/bcryptjs
- En auth.service cambio la importación de bcrypt por bcryptjs
- Ya puedo usar la imagen con docker-compose
- Si quiero cambiar las variables de entorno lo hago desde el archivo docker-compose.prod.yml
- Si elimino la imagen de postgres del docker-compose.prod.yml saltará un error como "no encryption..."
- Digital Ocean está esperando una conexión SSL como definimos en app.module

```
import { join } from 'path';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { ConfigModule } from '@nestjs/config';
import { Module } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';

import { GraphQLModule } from '@nestjs/graphql';
import { TypeOrmModule } from '@nestjs/typeorm';

import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';

import { ItemsModule } from './items/items.module';
import { UsersModule } from './users/users.module';
import { AuthModule } from './auth/auth.module';
import { SeedModule } from './seed/seed.module';
import { CommonModule } from './common/common.module';
import { ListsModule } from './lists/lists.module';
import { ListItemModule } from './list-item/list-item.module';


@Module({
  imports: [

    ConfigModule.forRoot(),

    GraphQLModule.forRootAsync({
      driver: ApolloDriver,
      imports: [ AuthModule ],
      inject: [ JwtService ],
      useFactory: async( jwtService: JwtService ) => ({
        playground: false,
        autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
        plugins: [
          ApolloServerPluginLandingPageLocalDefault
        ],
        context({ req }) {
```

```
            // const token = req.headers.authorization?.replace('Bearer ','');
            // if ( !token ) throw Error('Token needed');

            // const payload = jwtService.decode( token );
            // if ( !payload ) throw Error('Token not valid');

          }
        })
      }),

      // TODO: configuración básica
      // GraphQLModule.forRoot<ApolloDriverConfig>({
      //   driver: ApolloDriver,
      //   // debug: false,
      //   playground: false,
      //   autoSchemaFile: join( process.cwd(), 'src/schema.gql'),
      //   plugins: [
      //     ApolloServerPluginLandingPageLocalDefault
      //   ]
      // }),

      TypeOrmModule.forRoot({
        type: 'postgres',
        ssl:(process.env.STATE === 'prod')
        ?{
          rejectUnauthorized: false,
          sslmode: 'require'

        }
        : false as any,
        host: process.env.DB_HOST,
        port: +process.env.DB_PORT,
        username: process.env.DB_USERNAME,
        password: process.env.DB_PASSWORD,
        database: process.env.DB_NAME,
        synchronize: true,
        autoLoadEntities: true,
      }),

      ItemsModule,

      UsersModule,

      AuthModule,

      SeedModule,

      CommonModule,

      ListsModule,

      ListItemModule,
    ],
    controllers: [],
```

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

- Debo colocar la variable STATE en el docker-compose.prod.yml

```
environment:
  STATE: ${STATE}
  DB_PASSWORD: ${DB_PASSWORD}
  DB_NAME: ${DB_NAME}
  DB_HOST: ${DB_HOST}
  DB_PORT: ${DB_PORT}
  DB_USERNAME: ${DB_USERNAME}
  JWT_SECRET: ${JWT_SECRET}
  PORT: ${PORT}
```

- Ahora tenemos una imagen creada

Usar la imagen y regenrarla sin compose

- No tengo porqué desplegar la imagen en DockerHub
- Para hacer el build, puedo renombrarla a nest-graphql-prod

> docker build -t nest-grapql-prod docker run nest-graphql-prod

- El comando docker compose establece las variables de entorno, docker build no
- Puedo usar -e

> docker run -e DB_PORT=5300 nest-graphql-prod

- Para no especificar una por una, también mapear el puerto 8080 de mi computadora con el 4000 del contenedor

> docker run --env-file=.env.prod -p 8080:4000 nest-graphql-prod

- En la consola dirá que corre en el puerto 4000 pewro para mi será el 8080
- En mi navegador apuntaré a localhost:8080
- Si quiero subir la imagen en dockerHub creo un repositorio y subo la imagen