NEST Microservicios

- Este módulo trata de comunicar dos backends mediante mensajes
- Vamos a controlar desde ambos si hay o no conexión
- Qué es un microservicio?
    - Es un tipo de arquitectura que nos permite conectarnos a diferentes sitios y obtener info4
    - Sirven para comunicar a los backends entre si
    - Desde el microservicio, a través de TCP se enviará y recibirá información entre uno y otro backend
    - Es el microservicio quien realmente se comunica, no el backend en si (aunque haga las tareas)
    - Puedes dividir el peso de la aplicación en diferentes backends, así si falla algo no se cae toda la app

---

Creando el proyecto

- Crearemos dos proyectos con nest new microservicios-1 y 2
- Me abro dos vscode, uno con el backend1 y otro con el 2
- Instalo las dependencias necesarias en ambos backends por separado

> npm i @nestjs/microservices

- Creamos los módulos necesarios en ambos backends. Un módulo con controlador y servicio
- NOTA: Lo hago con res, prefiero borrar lo que no necesito

> nest g res example-communication nest g res microservice-connection

- En el controller de example-communication pongo 'api/v1/microservices-b1' y 'api/v1/microservices-b2' en el del segundo backend
- Borro los .spec y borro tambien app.service, y app.controller, los quito del app.module
- Me aseguro de que ha importado los módulos example-communication y microservice-connection en el imports de app.module

---

Creando nuestros microservicios

- Para crear el microservicio voy al main.ts
- Voy a crear un microservicio que se conectará a mi mismo, que me servirá para que el otro conecte con su respectivo microservicio
- Para algo unidireccinal me puede valer el método .createtMicroservice
- Cuando quiero hacerlo bidireccional, como es el caso, es mejor esta manera
- Le indico el tipo de transporte, y en options el host (a si mismo) y un puerto aleatorio (en este caso 3032)
- Este 3032, cuando hagamos el otro microservicio en el backend 2, **apuntaremos a este 3032**
- No estamos conectando microservicios, de momento. Apunta a si mismo

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

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

  app.connectMicroservice({
    transport: Transport.TCP,
    options: {
      host: '0.0.0.0',
      port: 3032
    }
  })

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

- Copio y pego en el backend2, pongo de puerto 3030
- No se pueden hacer peticiones a este puerto. Lo usará el microservicio para conectarse y pasarse datos

---

Creando la conexión de nuestros microservicios

- Creo el cliente de tipo ClientProxy. Es quién va a hacer la mágia de la conexión
- ClientProxyFactory es el que crea el transporte. Le indico lo mismo que en el main y lo conecto al puerto del microservicio del backend2
- Creo un get para obtener el cliente

```
import { Injectable } from '@nestjs/common';
import { ClientProxy, ClientProxyFactory, Transport } from
'@nestjs/microservices';

@Injectable()
export class MicroserviceConnectionService {
    private client: ClientProxy

    constructor(){
        this.client = ClientProxyFactory.create({
            transport: Transport.TCP,
            options:{
                host: '0.0.0.0',
                port: 3030
            }
        })
    }

    getClient(){
```

```
        return this.client
    }
}
```

- Hago lo mismo en el service del backend2 y lo conecto al microservicio del backend1 (al puerto 3032)

Creando el endpoint

- En ExampleCommunicationController

```
import { Controller, Get } from '@nestjs/common';
import { ExampleCommunicationService } from './example-communication.service';
import { CreateExampleCommunicationDto } from './dto/create-example-
communication.dto';
import { UpdateExampleCommunicationDto } from './dto/update-example-
communication.dto';

@Controller('api/v1/microservices-b1')
export class ExampleCommunicationController {
  constructor(private readonly exampleCommunicationService:
ExampleCommunicationService) {}

  @Get('send-message')
  sendMessage(){
    return this.exampleCommunicationService.sendMessage('hola')
  }

}
```

- En el service

```
import { Injectable } from '@nestjs/common';
import { CreateExampleCommunicationDto } from './dto/create-example-
communication.dto';
import { UpdateExampleCommunicationDto } from './dto/update-example-
communication.dto';

@Injectable()
export class ExampleCommunicationService {

  sendMessage(msg: string){
    return msg
  }
}
```

- Hago lo mismo en el backend2

Patterns 2

- Son los mensajes que tienen que cuadrar en un backend y otro para que sepan que evento están recibiendo
- Sirve para saber adónde tiene que ir (con indicarle el puerto en el Transport no es suficiente)
- Es como una dirección, con la ciudad (el puerto) no es suficiente, necesito saber en que calle
- Creo el archivo example-communication.constants.ts
- Puedo usar cualquier palabra en el objeto, uso controller

```
export const PATTERNS ={
    MESSAGES:{
        SEND_MESSAGE: { controller: 'sendMessage'}
    },
    EVENTS:{
        RECEIVE_MESSAGE: { controller: 'receiveMessage'}
    }
}
```

- Copio el archivo en el backend2

MESSAGE PATTERN Backend1 al Backend2

- Usaremos **MessagePattern** en el service en el ExampleCommunicationService
- En el constructor inyecto el MicroserviceConnectionService
- Debo **importar el módulo** de *MicroserviceConnection* **en el imports** de example-communication.module.ts y **colocar el servicio en provider**
- Uso el servicio **.getClient** con el **send**
- El send es lo que tenemos para comunicar eventos
- Con el send le voy a mandar un pattern y un objeto al controlador del backend2, que tiene que estar escuchando ese patrón, y luego vuelve al backend1
- Uso **firstValueFrom**

```
import { Injectable } from '@nestjs/common';
import { MicroserviceConnectionService } from 'src/microservice-
connection/microservice-connection.service';
import { PATTERNS } from './example-communication.constants';
import { firstValueFrom } from 'rxjs';

@Injectable()
export class ExampleCommunicationService {

  constructor(private microServiceConnection:MicroserviceConnectionService){ }

  sendMessage(msg: string){
    return firstValueFrom(
      this.microServiceConnection
      .getClient()
```

```
      .send(
        PATTERNS.MESSAGES.SEND_MESSAGE,
        {
          msg
        }
      )
      )
  }
}
```

- Ahora en el controller del backend2 usamos **@MessagePattern** y le paso el mismo pattern que he puesto en el send
- En el objeto data lo tipo con el mismo nombre (msg)
- controller backend2

```
import { Controller, Get, Post, Body, Patch, Param, Delete } from
'@nestjs/common';
import { ExampleCommunicationService } from './example-communication.service';
import { MessagePattern } from '@nestjs/microservices';
import { PATTERNS } from './example-communication.constants';

@Controller('api/v1/microservices-b2')
export class ExampleCommunicationController {
  constructor(private readonly exampleCommunicationService:
ExampleCommunicationService) {}

  @Get('send-message')
  sendMessage(){
    return this.exampleCommunicationService.sendMessage('adiós')
  }

  @MessagePattern(PATTERNS.MESSAGES.SEND_MESSAGE)
  receivemessageFromMessagePatternB1(data: {msg: string}){
    console.log("Mensaje de B1 recibido", data.msg)
    return true
  }
}
```

- Si apunto al endpoint del b1 'send-message' enviará el mensaje

> http://localhost:3000/api/v1/microservices-b1/send-message