

Universidad Autónoma de Chiapas | Campus 01 | Facultad de contaduría y administración.

Docente: Dr. Luis Gutiérrez Alfaro.

Materia: Taller de desarrollo 4.

Nombre del alumno (s):

- Alegría de la Cruz Brayan Fermín | A210519.

Semestre: 6° | Grupo: "M".

Tema: Microservicio con base de datos.

Unidad: 1 | Materia de Taller de Desarrollo IV.

Número de actividad: Actividad 3.1 | Microservicio con base de datos.

Tuxtla Gutiérrez, Chiapas; a 18 de abril de 2024.

Para esta actividad, vamos a utilizar nuestro monorepo, el cual ya tenía configurada una capa de transporte y atendía peticiones a nivel local, sin embargo, no se almacenaban en ninguna base de datos.

En mi caso, utilizaré mysql para la conexión a la base de datos, se instalará el decorador necesario para que funcione, posteriormente se mapeará la base de datos, se definirán DTO's y se lanzarán las queries para inyectar datos, modificar, consultar o eliminar en la base.

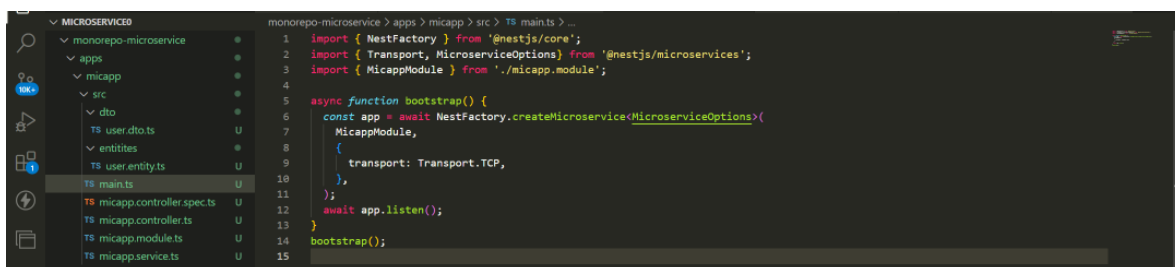
Pasos.

1. MYSQL.

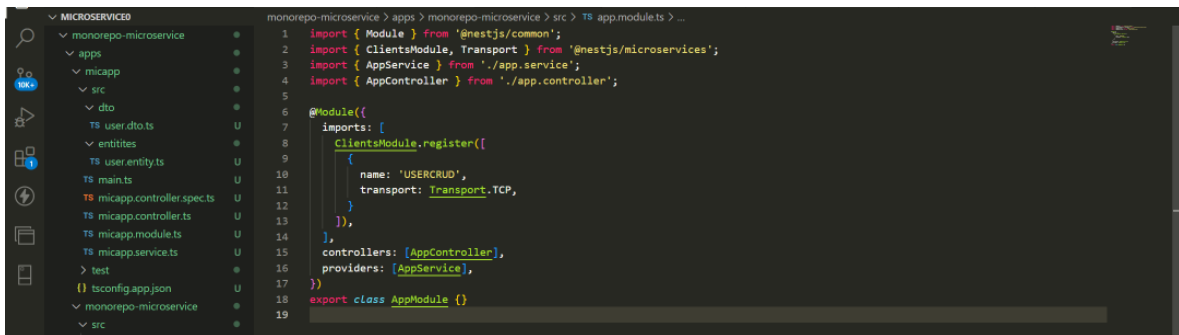
Para hacer uso de MYSQL, deberemos instalar el decorador correspondiente, el cual se hace bajo el siguiente comando: `npm install --save @nestjs/typeorm typeorm mysql2`

2. Microservicio.

Posteriormente, comenzaremos a trabajar con el microservicio, en el main de nuestro microservicio, importaremos el módulo correspondiente y definiremos la capa de transporte correspondiente para convertirlo a microservicio, en mi caso será TCP.




Después, definiremos el microservicio que se esperará en el api-gateway, el cual se definió como “USERCRUD”.



```
1 import { Module } from '@nestjs/common';
2 import { ClientsModule, Transport } from '@nestjs/microservices';
3 import { AppService } from './app.service';
4 import { AppController } from './app.controller';
5
6 @Module({
7   imports: [
8     ClientsModule.register([
9       {
10         name: 'USERCRUD',
11         transport: Transport.TCP,
12       }
13     ]),
14   ],
15   controllers: [AppController],
16   providers: [AppService],
17 })
18 export class AppModule {}
```

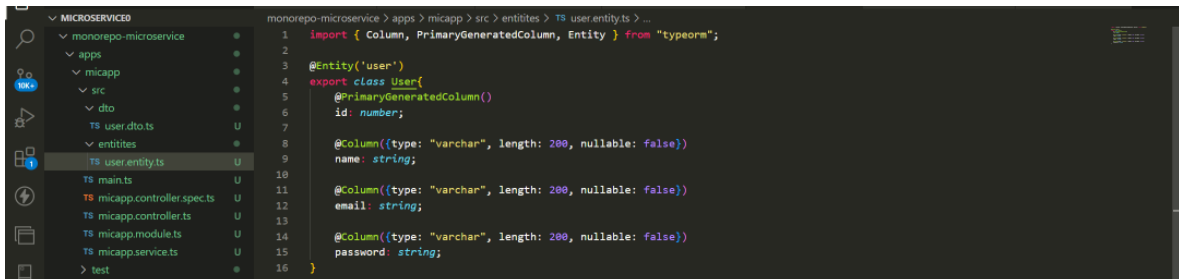
Como siguiente paso, trabajaremos en lo que es el módulo del microservicio y es allí donde haremos la conexión a la base datos, importando el typeorm y escribiendo las credenciales correspondientes a nuestro localhost en mysql.



```
1 import { Module } from '@nestjs/common';
2 import { MicappController } from './micapp.controller';
3 import { MicappService } from './micapp.service';
4 import { TypeOrmModule } from '@nestjs/typeorm';
5 import { User } from './entities/user.entity';
6
7 @Module({
8   imports: [
9     TypeOrmModule.forRoot({
10       type: 'mysql',
11       host: 'localhost',
12       port: 3306,
13       username: 'root',
14       password: '',
15       database: 'service',
16       entities: [User],
17       synchronize: true,
18     }),
19     TypeOrmModule.forFeature([User]),
20   ],
21   controllers: [MicappController],
22   providers: [MicappService],
23 })
24 export class MicappModule {}
```

Allí, se definen las credenciales (tipo de conexión, usuario, puerto, contraseña, base de datos y la entidad a la que estará apuntando).

Consiguiente a ello, definiremos nuestra tabla, la cual se creará al momento de levantar nuestros servicios, dentro de src se crea una carpeta de entidades, junto con un archivo que posee las entidades de la tabla (atributos) y su longitud.



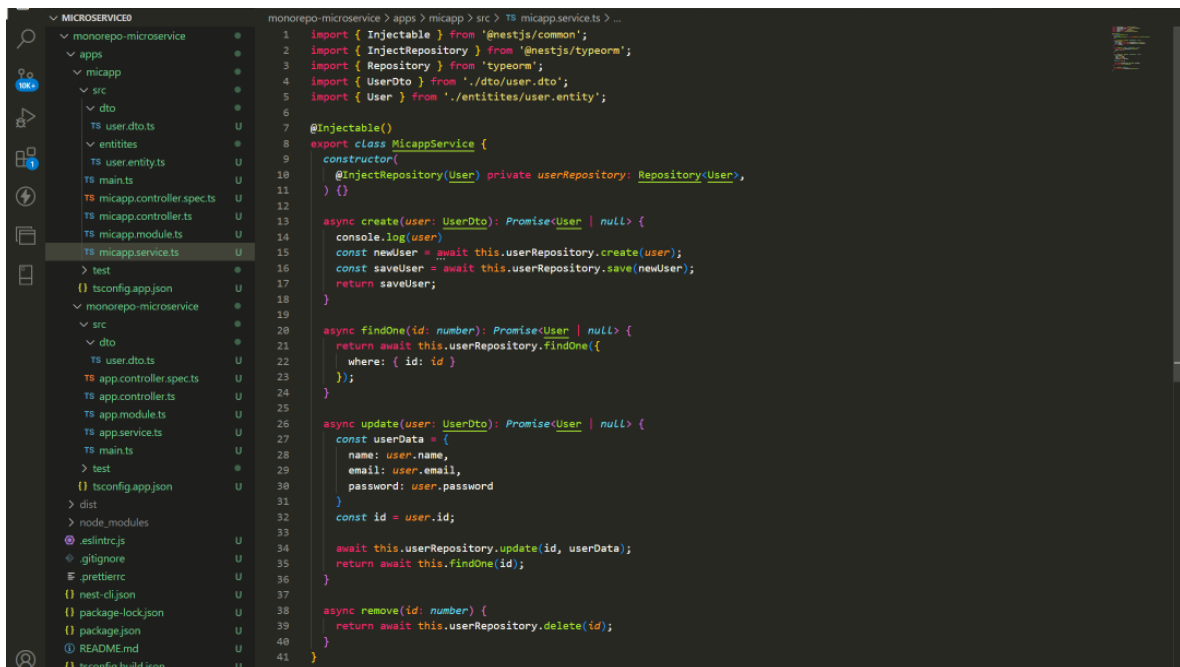
```
1 import { Column, PrimaryGeneratedColumn, Entity } from 'typeorm';
2
3 @Entity('user')
4 export class User {
5   @PrimaryGeneratedColumn()
6   id: number;
7
8   @Column({type: "varchar", length: 200, nullable: false})
9   name: string;
10
11   @Column({type: "varchar", length: 200, nullable: false})
12   email: string;
13
14   @Column({type: "varchar", length: 200, nullable: false})
15   password: string;
16 }
17
```

Después, definiremos un DTO en nuestro microservicio, se puede realizar a la par en nuestra api-gateway dado que de igual forma haremos uso de él allí, bajo las mismas propiedades.



```
1 export class UserDto {
2   id?: number;
3   name?: string;
4   email?: string;
5   password?: string;
6 }
```

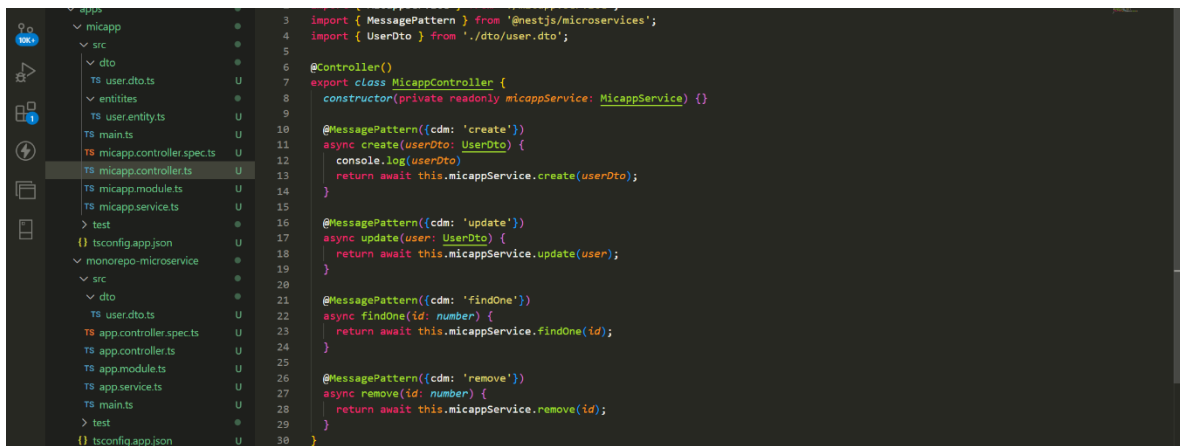
Posteriormente en el service de nuestro microservicio, haremos una instancia de los siguientes parámetros (que funciona como CRUD):



```
1 import { Injectable } from '@nestjs/common';
2 import { InjectRepository } from '@nestjs/typeorm';
3 import { Repository } from 'typeorm';
4 import { UserDto } from '../dto/user.dto';
5 import { User } from '../entities/user.entity';
6
7 @Injectable()
8 export class MicappService {
9   constructor(
10     @InjectRepository(User) private userRepository: Repository<User>,
11   ) {}
12
13   async create(user: UserDto): Promise<User | null> {
14     console.log(user);
15     const newUser = await this.userRepository.create(user);
16     const saveUser = await this.userRepository.save(newUser);
17     return saveUser;
18   }
19
20   async findOne(id: number): Promise<User | null> {
21     return await this.userRepository.findOne({
22       where: { id: id }
23     });
24   }
25
26   async update(user: UserDto): Promise<User | null> {
27     const userData = {
28       name: user.name,
29       email: user.email,
30       password: user.password
31     };
32     const id = user.id;
33
34     await this.userRepository.update(id, userData);
35     return await this.findOne(id);
36   }
37
38   async remove(id: number) {
39     return await this.userRepository.delete(id);
40   }
41 }
```

Ahora, en el controlador del microservicio, se hace una inyección de dependencias al microservicio.

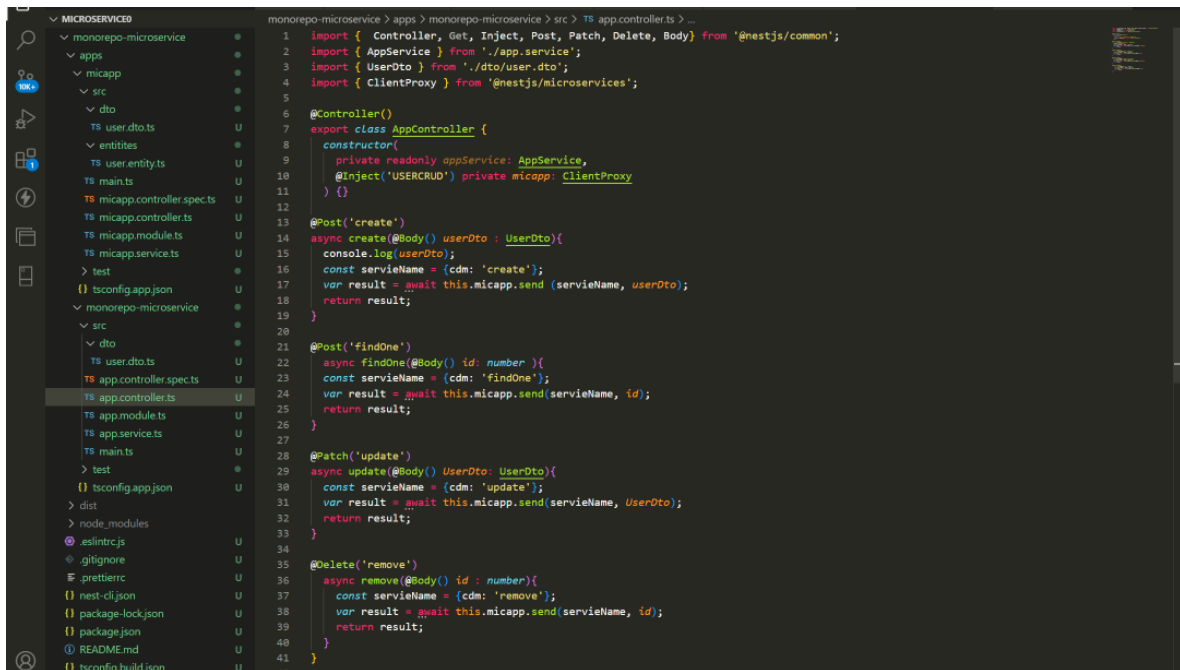
Hecha por mensajes, no por eventos:



```
1 import { Controller } from '@nestjs/common';
2 import { MessagePattern } from '@nestjs/microservices';
3 import { UserDto } from '../dto/user.dto';
4
5 @Controller()
6 export class MicappController {
7   constructor(private readonly micappService: MicappService) {}
8
9   @MessagePattern({cmd: 'create'})
10   async create(userDto: UserDto) {
11     console.log(userDto);
12     return await this.micappService.create(userDto);
13   }
14
15   @MessagePattern({cmd: 'update'})
16   async update(user: UserDto) {
17     return await this.micappService.update(user);
18   }
19
20   @MessagePattern({cmd: 'findOne'})
21   async findOne(id: number) {
22     return await this.micappService.findOne(id);
23   }
24
25   @MessagePattern({cmd: 'remove'})
26   async remove(id: number) {
27     return await this.micappService.remove(id);
28   }
29 }
```

Api-gateway.

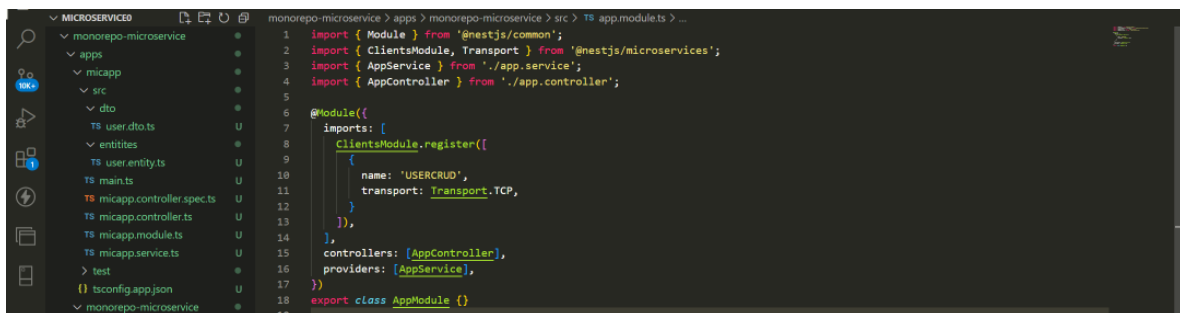
Aquí, dentro de módulo instanciamos y posteriormente declaramos la comunicación con el microservicio “USERCRUD” usando una variable de transacción y la definición de los métodos correspondientes:



```
monorepo-microservice > apps > monorepo-microservice > src > TS app.controller.ts > ...
1 import { Controller, Get, Inject, Post, Patch, Delete, Body } from '@nestjs/common';
2 import { AppService } from '../app.service';
3 import { UserDto } from '../dto/user.dto';
4 import { ClientProxy } from '@nestjs/microservices';
5
6 @Controller()
7 export class AppController {
8   constructor(
9     private readonly appService: AppService,
10    @Inject('USERCRUD') private micapp: ClientProxy
11  ) {}
12
13  @Post('create')
14  async create(@Body() userDto: UserDto) {
15    console.log(userDto);
16    const serviceName = {cdm: 'create'};
17    var result = await this.micapp.send(serviceName, userDto);
18    return result;
19  }
20
21  @Post('findOne')
22  async findOne(@Body() id: number) {
23    const serviceName = {cdm: 'findOne'};
24    var result = await this.micapp.send(serviceName, id);
25    return result;
26  }
27
28  @Patch('update')
29  async update(@Body() UserDto: UserDto) {
30    const serviceName = {cdm: 'update'};
31    var result = await this.micapp.send(serviceName, UserDto);
32    return result;
33  }
34
35  @Delete('remove')
36  async remove(@Body() id: number) {
37    const serviceName = {cdm: 'remove'};
38    var result = await this.micapp.send(serviceName, id);
39    return result;
40  }
41 }
```

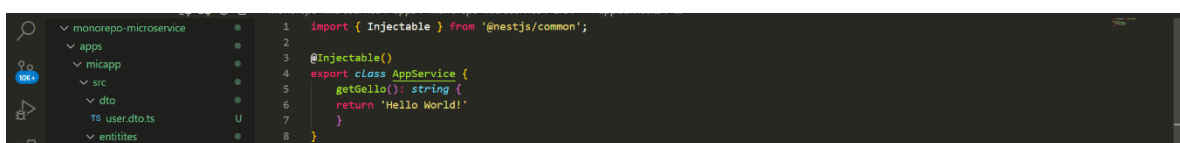
Y los demás archivos quedarán de la siguiente forma:

Módulo:



```
monorepo-microservice > apps > monorepo-microservice > src > TS app.module.ts > ...
1 import { Module } from '@nestjs/common';
2 import { ClientsModule, Transport } from '@nestjs/microservices';
3 import { AppService } from '../app.service';
4 import { AppController } from './app.controller';
5
6 @Module({
7   imports: [
8     ClientsModule.register([
9       {
10        name: 'USERCRUD',
11        transport: Transport.TCP,
12      }
13     ]),
14   ],
15   controllers: [AppController],
16   providers: [AppService],
17 })
18 export class AppModule {}
19
```

Service:



```
monorepo-microservice > apps > monorepo-microservice > src > TS app.service.ts > ...
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class AppService {
5   getHello(): string {
6     return 'Hello World!';
7   }
8 }
```

Main:



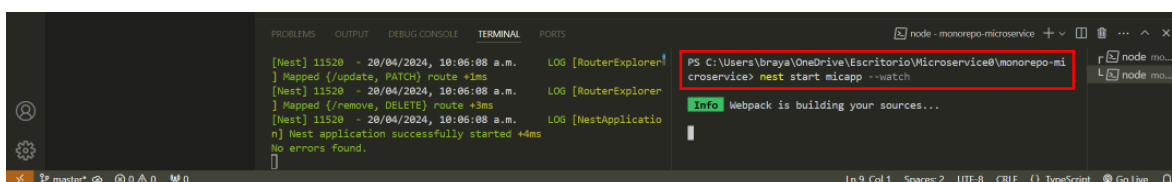
```
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app.module';
3
4 async function bootstrap() {
5   const app = await NestFactory.create(AppModule);
6   await app.listen(3001);
7 }
8 bootstrap();
```

Querys a la base de datos:

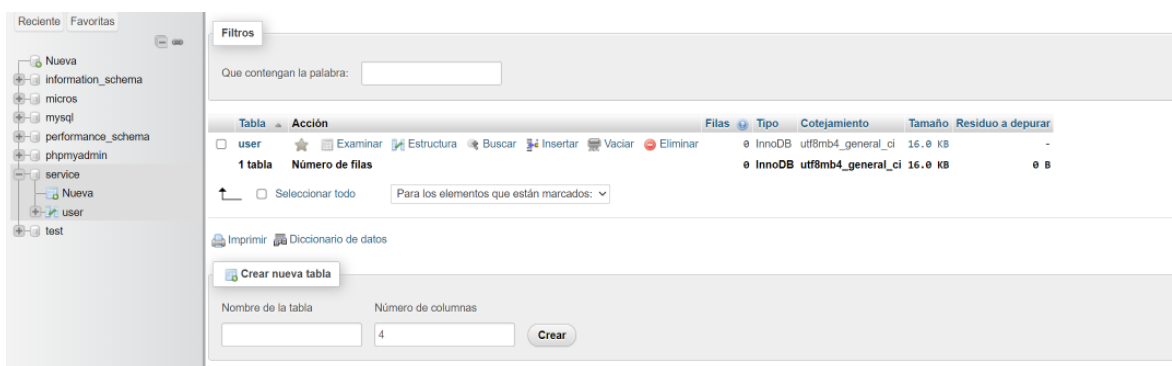
Como vemos en localhost, nuestra base a la que apuntamos es “service” sin embargo, no posee ninguna tabla y evidentemente ningún atributo:



Ahora, veamos que, al momento de levantar nuestro microservicio, se creará la tabla de usuarios:

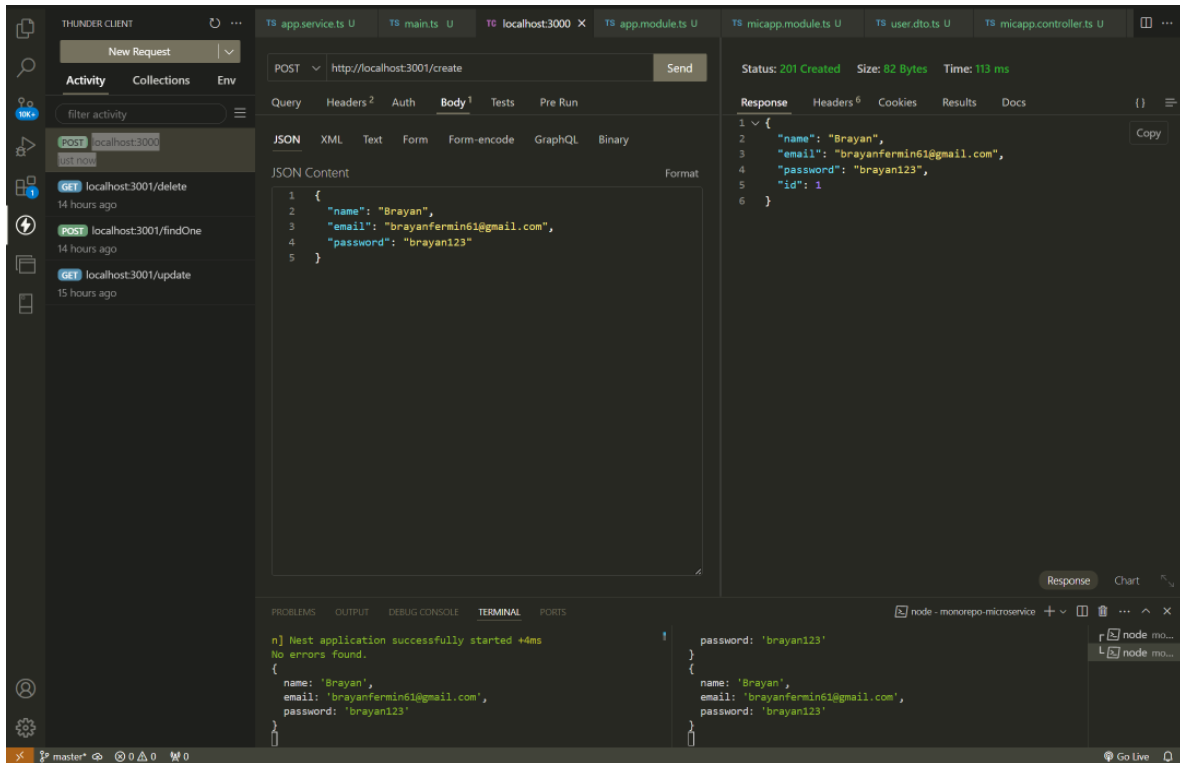


Actualizamos la tabla en localhost:



Y vemos que la tabla vacía ha sido creada.

Ahora, podemos comenzar a insertar usuarios, consultarlos, eliminarlos o modificarlos. Para ello, haré uso de thunder client especificando el tipo de query que lanzaré y la estructura json que corresponde:

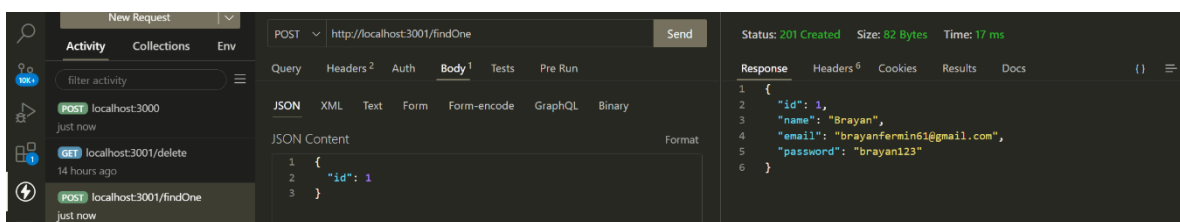


Vemos que la query se lanzó correctamente y verificamos en nuestra tabla la inserción de nuestro usuario:

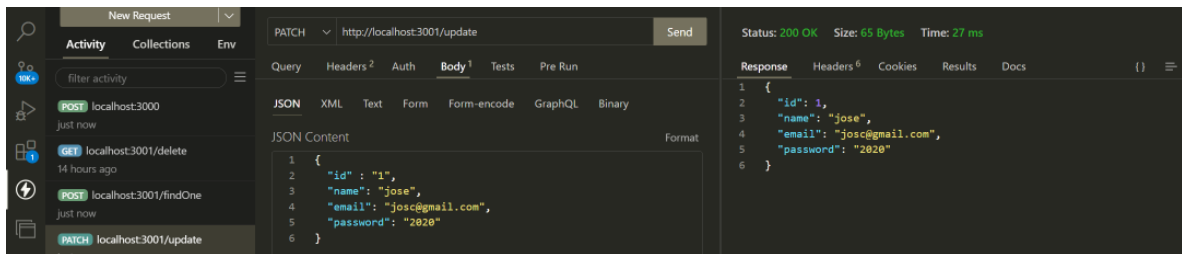


Ahora las demás queries:

Buscar un usuario por id:



Actualizar usuario:



Activity Collections Env

Filter activity

POST localhost:3000 just now

GET localhost:3001/delete 14 hours ago

POST localhost:3001/findOne just now

PATCH localhost:3001/update just now

Send

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content

```
1 {
2   "id": "1",
3   "name": "jose",
4   "email": "josc@gmail.com",
5   "password": "2020"
6 }
```

Status: 200 OK Size: 65 Bytes Time: 27 ms

Response Headers Cookies Results Docs

```
1 {
2   "id": 1,
3   "name": "jose",
4   "email": "josc@gmail.com",
5   "password": "2020"
6 }
```



Reciente Favoritas

Nueva

- information_schema
- micros
- mysql
- performance_schema
- phpmyadmin
- service
- Nueva
- user
- test

Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,0003 segundos.)

SELECT * FROM `user`

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

Mostrar todo Número de filas: 25 Filtrar filas: Buscar en esta tabla

Opciones extra

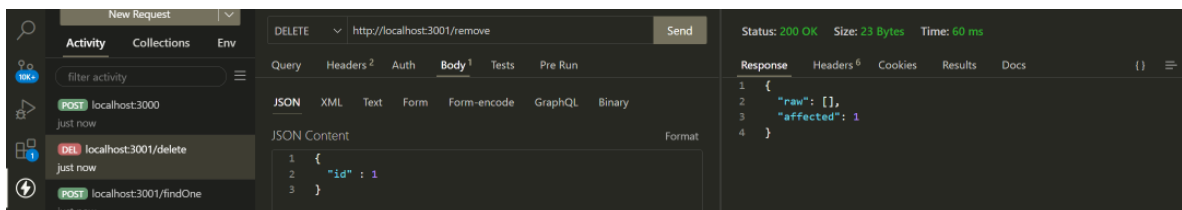
id	name	email	password
1	jose	josc@gmail.com	2020

Editar Copiar Borrar

Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

Mostrar todo Número de filas: 25 Filtrar filas: Buscar en esta tabla

Borrar usuario:



Activity Collections Env

Filter activity

POST localhost:3000 just now

DEL localhost:3001/delete just now

POST localhost:3001/findOne just now

Send

Query Headers Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

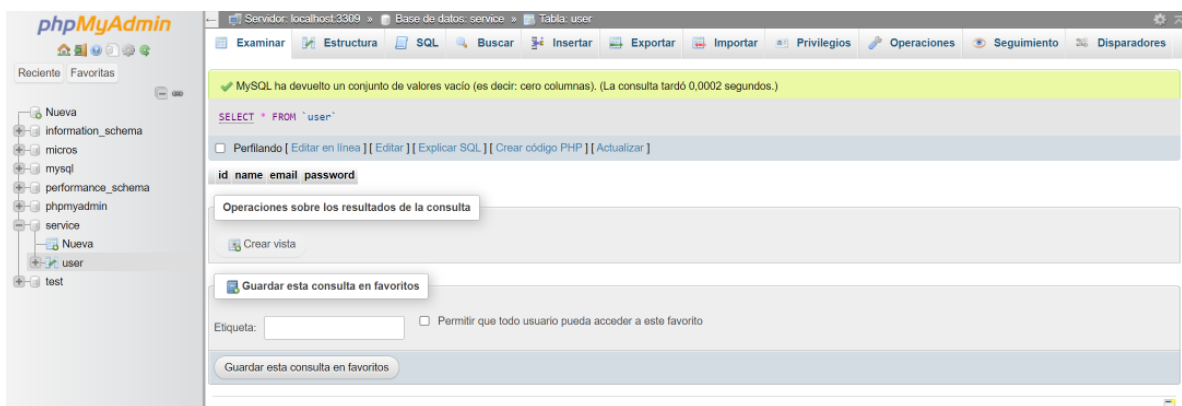
JSON Content

```
1 {
2   "id": 1
3 }
```

Status: 200 OK Size: 23 Bytes Time: 60 ms

Response Headers Cookies Results Docs

```
1 {
2   "raw": [],
3   "affected": 1
4 }
```



phpMyAdmin

Reciente Favoritas

Nueva

- information_schema
- micros
- mysql
- performance_schema
- phpmyadmin
- service
- Nueva
- user
- test

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0002 segundos.)

SELECT * FROM `user`

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

id	name	email	password
----	------	-------	----------

Operaciones sobre los resultados de la consulta

Crear vista

Guardar esta consulta en favoritos

Etiqueta: Permitir que todo usuario pueda acceder a este favorito

Guardar esta consulta en favoritos