Curso CFP

CFP Programador full-stack

CRUD en TypeORM

Agenda

- Repaso
- CRUD
  - Create
  - Read
  - Update
  - Delete
  - ReadAll
- Demo en Vivo
- Ejercicios

Repaso - Conexión con MySQL

- Para conectarnos a MySQL necesitamos ciertos datos
  - Host
  - Puerto
  - Usuario
  - Contraseña
- Siempre que queramos conectarnos a una base de datos, vamos a necesitar este tipo de información
- En NestJS tenemos dos formas de dar esta data
  - En app.module.ts
  - En un archivo externo → ormconfig.json 👍

Repaso - Entities

```
import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm';

@Entity('photo')
export class Photo {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({
      length: 255
    })
    name: string;

    @Column()
    description: string;

    @Column()
    views: number;

    @Column({
      default: false
    })
    published: boolean;
}
```

Nombre que va a tener la tabla que se cree al levantar la API

Definimos PK

Diferentes formas de definir columnas

Repaso - Comparación RAW/ORM

```
const result = await this.photoRepository.query('select * from photo where id = ' + id);

const photo = new Photo();
photo.id = result[0]['id'];
photo.name = result[0]['name'];
photo.description = result[0]['description'];
photo.filename = result[0]['filename'];
photo.views = result[0]['views'];
photo.published = result[0]['published'];
photo.albumId = result[0]['albumId'];
```

```
const myPhoto = await this.photoRepository.findOne(id);
```

Mucho más sencillo usando el findOne

Además el mapeo del resultado a una variable se hace de forma automática

CRUD - Create

- Al armar una API REST, es muy común hacer cierto tipo de operaciones, llamadas CRUD
- La primera es CREATE
- Se asocia con un HTTP POST
  - La información a crear va en el *body* del request
- Implica una *inserción* en la base de datos

```
const photo = new Photo();
photo.name = dto.name;
photo.description = dto.description;
photo.filename = dto.filename;
photo.published = dto.published;
photo.views = dto.views;
photo.album = album;

await this.photoRepository.save(photo);
```

Creamos una variable del tipo de la entity

Guardamos

CRUD - Read

- La segunda operación que comúnmente se usa es READ
- Se asocia con un HTTP GET
- Implica una *lectura* en la base de datos

```
const photo = await this.photoRepository.findOne(id);
```

Buscamos uno

```
const photos = await this.photoRepository.find();
```

Traemos todo

```
const photos = await this.photoRepository.find({
    where: {
      "description": "abc123"
    }
});
```

Traemos todo lo que cumpla con cierta condición

CRUD - Update

- La tercera operación es UPDATE
- Se asocia con un HTTP PUT
  - La información nueva va en el *body* del request
- Implica una *actualización* en la base de datos
  - Primero hacer una lectura
  - Modificar los campos necesarios
  - Guardar en la base

```
const photo = await this.photoRepository.findOne(id);

if (!photo) {
   throw new HttpException('Photo does not exist!', 404);
}

photo.name = photoDTO.name;
photo.description = photoDTO.description;

await this.photoRepository.save(photo);
```

Obtenemos la fila

Modificamos

Guardamos

CRUD - Delete

- La última es DELETE
- Se asocia con un HTTP DELETE
- Implica una *eliminación* en la base de datos

```
await this.photoRepository.delete(id);
```

```
await this.photoRepository.delete([1, 2, 3]);
```

CRUD - ReadAll

- Otra operación muy empleada normalmente es un READ pero obteniendo todas las filas
- Se asocia con un HTTP GET
- Implica una *lectura* en la base de datos

```
const photos = await this.photoRepository.find();
```

Demo en Vivo

- Hacer un seguimiento de la API
  - Controller
  - Service
  - Repository
- Ver reflejadas los requests en el Workbench
- Tomar como referencia
  https://github.com/francisco-serrano/sample-nestjs-api

Curso CFP

CFP Programador full-stack

Ejercicios

Ejercicios

- Partir de la API de la clase pasada
- Agregar endpoints para hacer CRUD, con la correspondiente operación sobre MySQL