Integracion

Carrera Programador full-stack

CRUD en TYPEORM

Agenda

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

Repaso - Comparación RAW/ORM

```
let datos = await this.ciudadRepository.query("SELECT * FROM ciudades");
datos.forEach(dato => {
        let ciudad : Ciudad = new Ciudad(dato['idCiudad'], dato['nombre']);
        this.ciudades.push(ciudad);
});
return this.ciudades;
```

```
this.ciudades  : Ciudad[] = await this.ciudadRepository.find();
return this.ciudades;
```

```
let dato = await this.ciudadRepository.query("SELECT * FROM ciudades WHERE idCiudad = "+id);
let ciudad : Ciudad = new Ciudad(dato['idCiudad'], dato['nombre']);
return ciudad;
```

```
const criterio : FindOptions = { where: { idCiudad: id } }
let ciudad : Ciudad = await this.ciudadRepository.find( criterio );
return ciudad;
```

Mucho más sencillo usando el find / findOne

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

CRUD - Read

- Al armar una API REST, es muy común hacer cierto tipo de operaciones, llamadas CRUD
- La segunda operación que comúnmente se usa es READ
- Se asocia con un HTTP GET
- Implica una *lectura* en la base de datos

```
ciudades : Ciudad[] = await this.ciudadRepository.find();
```

Traemos todo

```
let criterio : FindOneOptions = { idCiudad: id };
ciudad : Ciudad = await this.ciudadRepository.findOne(criterio);
```

Buscamos uno

```
let criterio : FindOptions = {
        idCiudad: between ( 1, 10 )
};
ciudades : Ciudad[] = await this.ciudadRepository.findBy(criterio);
```

Traemos todo lo que cumpla con cierta condición

https://orkhan.gitbook.io/typeorm/docs/find-options

DTO

- Para agregar filas a una tabla necesitamos que el FrontEnd envíe la información al backend.
- Establecemos un “contrato” para el formato de los objetos.
- Este contrato se representa como un DTO (Data Transfer Object)
- Tiene valores de sólo lectura que van a ser utilizados para mapear entidades.

```
export class CiudadDTO {
    readonly idCiudad : number,
    readonly nombre : string,
}
```

Si el id no es automático, viene desde el FE y debe estar en el DTO

```
export class CiudadDTO {
    readonly nombre : string
}
```

Si el id es automático, no debe estar en el DTO (ver ejemplos con *)

CRUD - Create (1)

- 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

ciudad.controller.ts

```
@Controller('ciudad')
export class CiudadController {
   constructor(private ciudadService : CiudadService) {}
   …
   @Post()
   private crearCiudad(@Body() ciudad : CiudadDTO) :Promise<Ciudad> {
      return this.ciudadService.addCiudad(ciudad)
   }
   …
}
```

CRUD - Create (2)

ciudad.service.ts

```
@Injectable()
export class CiudadService {
  private ciudades : Ciudad[] = [];

  constructor (@InjectRepository(Ciudad)
  private readonly ciudadRepository : Repository<Ciudad>) {}
  …
  public async addCiudad(ciudadDTO : CiudadDTO) : Promise<Ciudad> {
    try {
      let ciudad : Ciudad = await this.ciudadRepository.save( new Ciudad(
        ciudadDTO.idCiudad, ciudadDTO.nombre
      ) );
      if (ciudad)
        return ciudad;
      else
        throw new Exception('No se pudo crear la ciudad');
    } catch (error) {
        throw new HTTPException( { status : HttpStatus.NOT_FOUND,
            error : 'Error en la creacion de ciudad '+error}, HTTPStatus.NOT_FOUND);
    }
  }
  …
}
```

CRUD - Create (2 *)

ciudad.service.ts

```
@Injectable()
export class CiudadService {
  private ciudades : Ciudad[] = [];

  constructor (@InjectRepository(Ciudad)
  private readonly ciudadRepository : Repository<Ciudad>) {}
  …
  public async addCiudad(ciudadDTO : CiudadDTO) : Promise<Ciudad> {
    try {
      let ciudad : Ciudad = await this.ciudadRepository.save( new Ciudad(
        ciudadDTO.nombre
      ) );
      if (ciudad.getIdCiudad())
        return ciudad;
      else
        throw new Exception('No se pudo crear la ciudad');
    } catch (error) {
        throw new HTTPException( { status : HttpStatus.NOT_FOUND,
            error : 'Error en la creacion de ciudad '+error}, HTTPStatus.NOT_FOUND);
    }
  }
  …
}
```

CRUD - Update (1)

- 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

ciudad.controller.ts

```
@Controller('idCiudad')
export class CiudadController {
   constructor(private ciudadService : CiudadService) {}
   …
   @Put()
   private actualizarCiudad(@Body() ciudad : CiudadDTO) :Promise<Ciudad> {
      return this.ciudadService.updateCiudad(ciudad)
   }
   …
}
```

CRUD - Update (1 *)

- 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

ciudad.controller.ts

```
@Controller('ciudad')
export class CiudadController {
   constructor(private ciudadService : CiudadService) {}
   …
   @Put(':idCiudad')
   private actualizarCiudad(@Param('idCiudad') id : number,
                            @Body() ciudad : CiudadDTO): Promise<Ciudad> {
      return this.ciudadService.updateCiudad(id, ciudad)
   }
   …
}
```

CRUD - Update (2)

ciudad.service.ts

```
@Injectable()
export class CiudadService {
  private ciudades : Ciudad[] = [];

  constructor (@InjectRepository(Ciudad)
  private readonly ciudadRepository : Repository<Ciudad>) {}
  ...
  public async updateCiudad(ciudadDTO : CiudadDTO) : Promise<Ciudad> {
    try {
      let criterio : FindOneOptions = { idCiudad: ciudadDTO.idCiudad };
      let ciudad : Ciudad = await this.ciudadRepository.findOne(criterio);
      if (!ciudad)
        throw new Exception('No se encuentra la ciudad');
      else
        ciudad.setNombre(ciudadDTO.nombre);
      ciudad = await this.ciudadRepository.save(ciudad);
      return ciudad;
    } catch (error) {
        throw new HTTPException( { status : HttpStatus.NOT_FOUND,
            error : 'Error en la actiualizacion de ciudad '+error}, HTTPStatus.NOT_FOUND);
    }
  }
  ...
}
```

CRUD - Update (2 *)

ciudad.service.ts

```
@Injectable()
export class CiudadService {
  private ciudades : Ciudad[] = [];

  constructor (@InjectRepository(Ciudad)
  private readonly ciudadRepository : Repository<Ciudad>) {}
  ...
  public async updateCiudad(id: number, ciudadDTO : CiudadDTO) : Promise<Ciudad> {
    try {
      let criterio : FindOneOptions = { idCiudad: id };
      let ciudad : Ciudad = await this.ciudadRepository.findOne(criterio);
      if (!ciudad)
        throw new Exception('No se encuentra la ciudad');
      else
        ciudad.setNombre(ciudadDTO.nombre);
      ciudad = await this.ciudadRepository.save(ciudad);
      return ciudad;
    } catch (error) {
        throw new HTTPException( { status : HttpStatus.NOT_FOUND,
            error : 'Error en la actiualizacion de ciudad '+error}, HTTPStatus.NOT_FOUND);
    }
  }
  ...
}
```

CRUD - Delete (1)

- La última es DELETE
- Se asocia con un HTTP DELETE
- Implica una *eliminación* en la base de datos
- Siempre debemos asegurarnos que el objeto con ese id exista, antes de eliminar

ciudad.controller.ts

```
@Controller('ciudad')
export class CiudadController {
   constructor(private ciudadService : CiudadService) {}
   …
   @Delete()
   private eliminarCiudad(@Body() ciudad : CiudadDTO): Promise<boolean> {
      return this.ciudadService.deleteCiudad(ciudad)
   }
   …
}
```

CRUD - Delete (1 *)

- La última es DELETE
- Se asocia con un HTTP DELETE
- Implica una *eliminación* en la base de datos
- Siempre debemos asegurarnos que el objeto con ese id exista, antes de eliminar

ciudad.controller.ts

```
@Controller('ciudad')
export class CiudadController {
   constructor(private ciudadService : CiudadService) {}
   …
   @Delete(':idCiudad')
   private eliminarCiudad(@Param('idCiudad') id : number) :Promise<boolean> {
      return this.ciudadService.deleteCiudad(id);
   }
   …
}
```

CRUD - Delete (2)

ciudad.service.ts

```
@Injectable()
export class CiudadService {
  private ciudades : Ciudad[] = [];

  constructor (@InjectRepository(Ciudad)
  private readonly ciudadRepository : Repository<Ciudad>) {}
  …
  public async deleteCiudad(ciudadDTO : CiudadDTO) : Promise<boolean> {
    try {
      let criterio : FindOneOptions = { idCiudad: ciudadDTO.idCiudad };
      let ciudad : Ciudad = await this.ciudadRepository.findOne(criterio);
      if (!ciudad)
        throw new Exception('No se encuentra la ciudad');
      else
        await this.ciudadRepository.delete(id);
      return true;
    } catch (error) {
        throw new HTTPException( { status : HttpStatus.NOT_FOUND,
            error : 'Error en la eliminacion de ciudad '+error}, HTTPStatus.NOT_FOUND);
    }
  }
  …
}
```

CRUD - Delete (2 *)

ciudad.service.ts

```
@Injectable()
export class CiudadService {
  private ciudades : Ciudad[] = [];

  constructor (@InjectRepository(Ciudad)
  private readonly ciudadRepository : Repository<Ciudad>) {}
  ...
  public async deleteCiudad(id : number) : Promise<boolean> {
    try {
      let criterio : FindOneOptions = { where:{idCiudad: id} };
      let ciudad : Ciudad = await this.ciudadRepository.findOne(criterio);
      if (!ciudad)
        throw new Exception('No se encuentra la ciudad');
      else
        await this.ciudadRepository.delete({idCiudad:id});
      return true;
    } catch (error) {
        throw new HTTPException( { status : HttpStatus.NOT_FOUND,
            error : 'Error en la eliminacion de ciudad '+error}, HTTPStatus.NOT_FOUND);
    }
  }
  ...
}
```

Integracion

Carrera Programador full-stack

Ejercicios

Ejercicios

- Crear DTO y completar servicio y controller para las entidades Escuelas, Estudiantes y Profesores creadas como tarea de la clase anterior.
- Se deben proveer los siguientes endpoints:
  - /[entidad]/add (**POST** de nueva entidad)
  - /[entidad]/upd/:id: (**PUT** para actualizar una entidad)
  - /[entidad]/del/:id: (**DELETE** de una entidad)
- Por ahora no tener en cuenta las relaciones de la tabla.
- Una vez testeados los endpoints, escribir el Front End correspondiente.