Nodejs typescript API Rest

Documentación

Iván Betanzos Macías

Índice:

1- Arquitectura usada

1.1 – Capa de presentación

1.2 – Capa de aplicación

1.3 – Capa de dominio

2 – Estructura de carpetas

3 – Flujo de datos

1 – Arquitectura usada:

**Patrón por servicios:**

Se ha optado por utilizar una arquitectura por capas bien diferenciadas. Con esto nos aseguramos de que cada capa tenga una única responsabilidad. Esto nos ofrece una serie de ventajas mientra la aplicación va creciendo:

* Fácil lectura y mantenimiento de cada componente
* Diferenciación de responsabilidades y posibilidad de abstraer el proceso
* Mayor escalabilidad

Cada capa sera capaz de comunicarse unicamente con su capa mas cercana, sin poder realizar saltos en ellas, y en una única dirección cada vez.

Las responsabilidades, y por tanto, las capas que vamos a encontrar en esta arquitectura seran las siguientes:

1. Capa de presentación (Controllers): Capa de entrada hacia el Rest API. Responsable unicamente de recibir el request por parte del cliente, comunicarse con la siguiente capa y devolver la respuesta al cliente.
2. Capa de aplicación (Services): Encargado de realizar la logica de negocio y comunicarse con la capa de comunicación con la base de datos.
3. Capa de dominio (Models/Entities): Capa encargada de comunicarse con la base de datos y con la capa de aplicación.

Como nota a esta arquitectura, se opto por añadir una documentación de endpoints basada en swagger, a modo de decoradores en el propio controlador. Esto hace que sea necesaria una transformación en el funcionamiento de estas capas, ya que se hace necesaria una estructura para el correcto funcionamiento de la documentación.

Así, en el proyecto encontraremos una responsabilidad cedida al método de comunicación entre el objeto de ruta y el controlador, que no será más que hacer de intermediario entre el controlador y los objetos de Request y Response.

Capa de cliente

Capa de

presentación

Capa de

dominio

Capa de

aplicación

Base de datos

Base de

datos

Modelos

Servicios

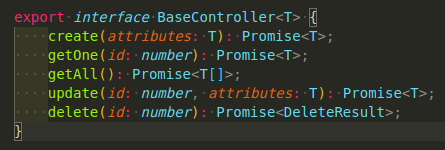
Controladores

Rutas

1.1 – Capa de presentación

En esta capa encontraremos los controladores y rutas hacia los diferentes endpoints de nuestro Rest API.

Tendremos una interfaz base para los controladores, lo que nos asegura la abstracción necesaria para generar los controladores rápidamente.



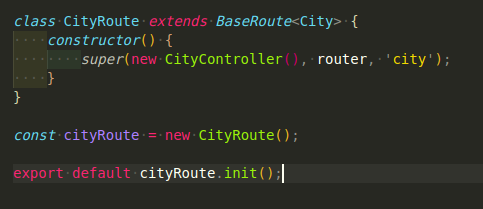
Se implementará la interfaz en cada controlador, añadiendo los decoradores necesarios para generar la documentación del endpoint a través de Swagger:



De cara a la generación de rutas, se trató de abstraer lo más posible la generación de estas. Así, podremos generar rápidamente las clases necesarias para el funcionamiento de cada endpoint:



Haremos que las clases de ruta extiendan esta clase base, para luego inicializarlo llamando a su método init():

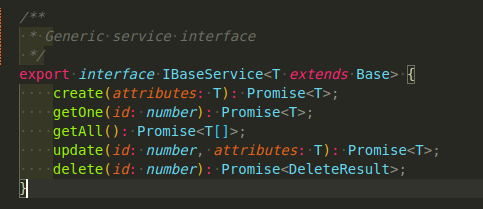


Tan solo con esto estaremos generando todas las rutas necesarias para el correcto funcionamiento del CRUD.

1.2 - Capa de aplicación

Encontramos aquí los servicios de nuestra aplicación, cuya responsabilidad es recibir los datos del request que el controlador provee y comunicarse con la capa de dominio.

Se encontrará una interfaz como base de todos los servicios. Esto hará que podamos crear un tipo base para todos los servicios.



Tendremos una clase abstracta, responsable de implementar la interfaz base:

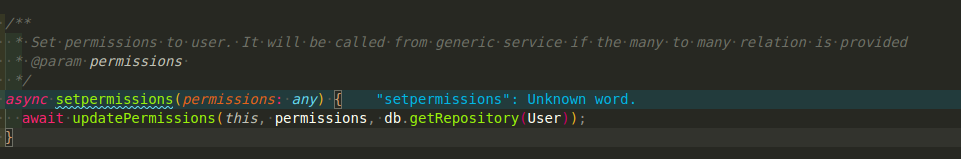


Se añadirá una variable “manyToMany”, en el que daremos los nombres de los campos que se relacionen muchos a muchos en la base de datos. Esto es, de cara a la integración automática de estas relaciones, de manera sencilla.

Para ello, en el update se realizará una verificación de esta variable (manyToMany) y se verificará tanto que venga en el request los ids para su sincronización y que en la clase de entidad exista el método necesario para realizar la sincronización:



Extracto de la entidad User:



1.3 - Capa de dominio

Esta capa se encargará de representar los modelos (tablas) de nuestra aplicación.

ORM usado:

El ORM usado en el proyecto es TypeORM. Se optó por el uso de esta tecnología debido a sus múltiples ventajas:

* Facilidad de uso y estructura amigable. Esto hace que sea muy fácilmente legible, mantenible y escalable en el tiempo. Estructura basada en decoradores.



* Compatibilidad con gran parte de las bases de datos disponibles: Podemos crear conexiones con bases de datos relacionales (por ejemplo, MySQL) o bases de datos documentales (ejemplo, MongoDB).
* Posibilidad de crear varios DataSource, realizando operaciones sobre bases de datos diferentes muy fácilmente.
* Fácil uso de consultas hacia la base de datos: Posibilidad de utilizar los métodos del propio ORM (find, findOne, save,…) o crear sql queries encadenando instrucciones.
* No es necesario generar o crear migraciones: No es necesario crear migraciones de cara a crear o modificar tablas en la base de datos. Esto será realizado a través de la propia clase de entidad. Sin embargo, tenemos la posibilidad de crear migraciones para un mayor control de los cambios realizados a las tablas. Esto es especialmente útil una vez el proyecto está en producción.

2 – Estructura de carpetas

**/**

**|\_\_ \_\_tests\_\_**

**|\_\_ ejemplo\_test.test.ts**

**|\_\_ controllers**

**|\_\_ BaseController.ts**

**|\_\_ index.ts**

**|\_\_ nombre\_modulo**

**|\_\_ ejemplo.controller.ts**

**|\_\_ database**

**|\_\_ db.ts**

**|\_\_ interfaces**

**|\_\_ index.ts**

**|\_\_ nombre\_modulo**

**|\_\_ ejemplo.interface.ts**

**|\_\_ middlewares**

**|\_\_ ejemplo.ts**

**|\_\_ models**

**|\_\_ Base.ts**

**|\_\_ index.ts**

**|\_\_ nombre\_modulo**

**|\_\_ ejemplo.ts**

**|\_\_ routes**

**|\_\_ BaseRoute.ts**

**|\_\_ index.ts**

**|\_\_ nombre\_modulo**

**|\_\_ ejemplo.route.ts**

**|\_\_ Server**

**|\_\_ Server.ts**

**|\_\_ services**

**|\_\_ IBaseService.service.ts**

**|\_\_ BaseService.service.ts**

**|\_\_ index.ts**

**|\_\_ nombre\_modulo**

**|\_\_ ejemplo.service.ts**

**|\_\_ util**

**|\_\_ index.ts**

**|\_\_ ejemplo.ts**

**|\_\_ .env**

**|\_\_ .env.example**

**|\_\_ .gitignore**

**|\_\_ docker-compose.yaml**

**|\_\_ index.ts (Punto de entrada de la aplicación)**

**|\_\_ jest.config.ts**

**|\_\_ package.json**

**|\_\_ README.md**

**|\_\_ tsconfig.json**

**|\_\_ tsao.json**

3 – Flujo de datos

