Programación Web Full Stack

Relaciones y CRUD

Práctica integradora

Objetivo

Algo muy común en el mundo del desarrollo web es realizar las funcionalidades para dejar listo un **CRUD** completo: creación de recursos, lectura de los mismos, actualización y borrado.

Es importante tener presente que también podemos referirnos a CRUD como **ABM**, por las siglas en español: Altas, Bajas y Modificaciones de recursos en una base de datos. Este será el foco del trabajo, junto a las posibles relaciones que tengan las tablas en la base de datos.

¡Buena suerte! 😎👍✨

Requisitos previos:

- **Paquete Sequelize instalado:** para lograr el objetivo es fundamental lograr instalar Sequelize y los paquetes relacionados necesarios.
- **Conexión con base de datos:** el paquete Sequelize no solamente precisa ser instalado sino que también precisa ser inicializado y configurado correctamente.

Micro desafío - Paso 1:

Utilizando de base el siguiente **proyecto creado con Express** (no hay que olvidarse de ejecutar la instrucción **npm install**). Además, utilizaremos la base de datos **movies_db**.

Una vez realizada la instalación de todas las dependencias del proyecto, debemos efectuar lo siguiente:

1. Realizar en los modelos ya creados (**Movie, Genre, Actor**), la definición de las siguientes **relaciones**:
    - Una película tiene un género.
    - Un género tiene muchas películas.
    - Una película tiene muchos actores.
    - Un actor tiene muchas películas.

Micro desafío - Paso 2:

Para la construcción de esta versión del sitio web, el cliente espera contar con la posibilidad de acceso a las siguientes URLs, por medio de las cuales estaríamos construyendo el **CRUD**:

- /movies/add (GET)
    - La misma muestra el formulario para la creación de una película.
    - Previamente, debe agregarse en la **vista detalle de películas** (**moviesDetail.ejs**) un botón de "**Agregar**" que envíe a esta URL y un botón de "**Listado de Películas**", el cual conduzca a la ruta (**/movies**).
    - El formulario de creación de una película debe contar con un **<select>** que permita elegir el género de la película.
    - Desde el controlador (**moviesController.js**), se le debe enviar a la vista ya existente (**moviesAdd.ejs**), los géneros de las películas.
- /movies/create (POST)
    - El controlador deberá utilizar la conexión a la base de datos y el modelo de Película ya creado. El método **create** permitirá la creación del nuevo registro. Finalmente, el controlador debe redirigir al **listado de películas** (**/movies**).
    - **Optativo**: idealmente la información recibida debe estar validada con **express-validator**.
- /movies/edit/:id (GET)
    - Previamente, debe agregarse en la vista detalle de la película (**moviesDetail.ejs**) un botón de "**Modificar**" que envíe a esta URL.
    - El formulario de edición de una película (**moviesEdit.ejs**) entre las opciones del **<select>**, por medio del cual podemos elegir el género de la misma, debe figurar por default el género que ya tenía la película.
    - El controlador deberá utilizar la conexión a la base de datos y el modelo de Película ya creado. Mediante el método **findByPk** y el id obtenible mediante **req.params**, se enviará a la vista la información que ya tiene la película. Esa variable se comparte con la vista y, mediante los atributos **value** de los campos del formulario, se carga la información.

- /movies/update/:id (POST) --- **Idealmente PUT (Optativo)**
    - Previamente, debe agregarse en el detalle de la película (**moviesDetail.ejs**), un botón de "**Borrar**" que envíe a esta URL.
    - Esta ruta recibe información del formulario (**moviesEdit.ejs**), recuperando los datos mediante **req.body**. El controlador deberá utilizar la conexión a la base de datos y el modelo de Película (**Movie**) ya creado.
    - Mediante el método **update**, se modifica la información de la base de datos. Luego, el controlador debe redirigir a la vista del listado de películas (**/movies**).
    - **Optativo**: idealmente, la información recibida debe estar validada con **express-validator**.
    - **Optativo**: utilizar los paquetes y modificaciones necesarias para que el método responda al pedido por **PUT**.
- /movies/delete/:id (GET) --- **Idealmente DELETE (Optativo)**
    - Previamente, debe agregarse en el detalle de la película (**moviesDetail.ejs**), un botón de "**Borrar**" que envíe a esta URL.
    - El controlador (**moviesController.js**) recupera el id de la URL mediante **req.params**.
    - El controlador debe devolver a la vista (**moviesDelete.ejs**), el nombre de la película a eliminar.
    - Finalmente, si se desea eliminar la película, debe llamar a la ruta (**/movies/delete/:id**).
    - El controlador deberá utilizar la conexión a la base de datos y el modelo de Película (**Movie**) ya creado.
    - Mediante el método **destroy** se elimina el registro y luego se redirige al listado de películas (**/movies**).

Bonus Track:

Si logramos realizar toda la práctica, una buena idea es replicar el proceso, pero con el **modelo Actores**.

Conclusión

El mapeo relacional de objetos (ORM) es una técnica que asigna objetos de software a tablas de bases de datos. Sequelize es una herramienta ORM popular y estable que se utiliza junto con Node.js.

Con ese conocimiento, procedimos a escribir una aplicación Node.js/Express simple que usa Sequelize para persistir los datos en el modelo a la base de datos. Usando los métodos heredados, así como la construcción de las relaciones entre las tablas, hemos realizado operaciones CRUD en la base de datos.

¡Hasta la próxima!