Curso de Back-End con Node.js (Avanzado)

Clase 06

Temario

Temario

- Relaciones en MongoDB.
  - Anidación de documentos.
  - Referencias a documentos.
- Ejercicios.

Relaciones en MongoDB

Relaciones en MongoDB

Hay dos grandes maneras de establecer relaciones en MongoDB:

1. Colocar (anidar) un documento dentro de otro (*embed*).
   Esto es muy similar colocar un objeto JSON dentro de otro.
   Por ejemplo, un artículo de un blog (`article`) podría tener un atributo `comments` conteniendo un array de comentarios. A su vez, cada uno de estos comentarios puede tener varios atributos.
2. Colocar referencias a documentos de otras colecciones.
   Esto es similar a lo que se realiza en el modelo relacional de BD.
   Por ejemplo, un artículo de un blog (`article`) podría estar vinculado con un autor que está en otra colección.

Relaciones en MongoDB

Anidación de documentos

Anidación de documentos (1/4)

La forma más sencilla de relacionar documentos es anidándolos. Ejemplo:

```
const articleSchema = new Schema({
  title: String,
  author: String,
  content: String,
  comments: [{ body: String, date: Date }],
});
```

En Mongoose, esto se llama *"subdocuments"*. Ver documentación.

Opcionalmente, se podría haber creado un esquema para los comentarios llamado `commentSchema` y luego usarlo de esta manera:

```
  comments: [commentSchema],
```

Anidación de documentos (2/4)

Pero hay que tener cuidado con modelar datos de forma muy anidada.

Hay una premisa que proviene del Python Zen (una serie de “mandamientos” para el programador) que dice:

“Flat is better than nested”

Esta premisa no es sólo válida para programar sino también para modelar datos.

Anidación de documentos (3/4)

Buenas prácticas:

- Sólo anidar información que sea muy intrínseca al modelo.
- En general, no anidar más de un nivel.
- Usar anidación para relaciones “uno-a-pocos”, no para “uno-a-muchos”.
  Ej: una persona puede tener algunas pocas direcciones.
- No anidar documentos que pueden tener “vida propia”, y que probablemente se los quiera acceder de forma independiente.

Por una buena lectura al respecto, ver este link.

Anidación de documentos (4/4)

Para agilizar la manipulación de documentos anidados, darle una mirada a los siguientes operadores:

- `$push`: https://docs.mongodb.com/manual/reference/operator/update/push.
- `$pull`: https://docs.mongodb.com/manual/reference/operator/update/pull.
- `$pop`: https://docs.mongodb.com/manual/reference/operator/update/pop.
- `$elemMatch`: https://docs.mongodb.com/manual/reference/operator/query/elemMatch.
- `$`: https://docs.mongodb.com/manual/reference/operator/projection/positional.
- `$set`: https://docs.mongodb.com/manual/reference/operator/update/set.

Relaciones en MongoDB

Referencias a documentos

Referencias a documentos (1/3)

En lugar de anidar documentos, es posible realizar referencias a documentos de otras colecciones. Eso es útil para relaciones “uno-a-muchos” (1-n).

Ejemplo en Mongoose:

```
const articleSchema = new Schema({
  title: String,
  author: {
    type: Schema.Types.ObjectId,
    ref: "Author",
  },
  content: String,
});
```

En Mongoose, esto se llama “populate”. Ver documentación.

Referencias a documentos (2/3)

Para asignarle un autor a un artículo, se puede hacer lo siguiente:

```
const author = new Author({
  name: "Hack Academy",
  email: "hola@ha.dev",
});
author.save();

const article = new Article({
  title: "Historia de la academia",
  author: author, // También se podría haber puesto `author._id`.
});
article.save();
```

Referencias a documentos (3/3)

Ahora, si se quiere acceder a los datos de un autor a partir de una artículo, es necesario indicarle a Mongoose que “populate” la información:

```
const article = await Article.findOne().populate("author");

console.log(article.author);
```

Este código accede al último artículo de la colección y muestra los datos de su autor.

Ejercicio

Ejercicio

Continuar con el ejercicio de la última clase...

1. Crear un modelo llamado `Goal` que contenga los siguientes atributos:
    a. Equipo que convirtió el gol (referencia a modelo `Team`).
    b. Equipo que recibió el gol (referencia a modelo `Team`).
    c. Nombre del jugador que hizo el gol (`String`).
    d. Minuto en que se hizo el gol (entre 0 y 120).
2. Modificar el modelo `Team` para que ya no contenga un atributo `goals` de tipo `Number` (como se hizo en la última clase) sino que ahora deben existir 2 atributos de tipo array y que contenga una lista de goles (de tipo `Goal`): `goalsScored` y `goalsConceded`.

Ejercicio (cont)

3. Crear los siguientes *endpoints* para la entidad goles:
   a. `GET /goals` – Para retornar la lista de todos los goles realizados.
   b. `POST /goals` – Para crear un gol. Ejemplo de body:

```
{
  "player": "Lionel Messi",
  "minute": 86,
  "teamFor": "ar",
  "teamAgainst": "br"
}
```

   c. `PATCH /goals/:id` – Para modificar un gol (sólo se puede modificar el nombre del jugador y/o el minuto).
   d. `DELETE /goals/:id` – Para eliminar un gol.
      No olvidar eliminar el gol de las listas `goalsScored` y `goalsConceded`, de los equipos correspondientes.

Ejercicio (cont)

4. Modificar el *endpoint* `GET /teams/:code` para que retorne los datos del equipo junto con los datos de los goles realizados y recibidos.

⚠️ Realizar validaciones pertinentes en todos los modelos. En caso de necesitar una validación que no exista en Mongoose “de fábrica”, investigar otras opciones como Validator.js. 👉 No reinventar la rueda.

🤔 Para pensar… ¿Qué sucede si se elimina del sistema un equipo y ya habían goles asociados a dicho equipo?