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

Clase 05

Temario

Temario

- Borrar documentos en Mongoose.
- Editar documentos en Mongoose.
- Update Operators
- Ejercicios.

Borrar documentos en Mongoose

Borrar documentos en Mongoose (1/4)

Borrar un (1) documento, dada su instancia:

```
unArticle
  .remove()
  .then((articleBorrado) => console.log(articleBorrado));
```

Nota 1: el método `remove` aplicado a un **modelo** está *deprecado*. Sin embargo, el método `remove` aplicado a una **instancia** sigue vigente 🤷‍♂️. Esto último es lo que está mostrado en esta diapositiva.

Nota 2: En este caso, también se podría haber hecho: `unArticle.deleteOne()` 👉 Ver documentación.

👉 Ver documentación.

Versión con `async/await`:

```
const articleBorrado = await unArticle.remove();
console.log(articleBorrado);
```

Borrar documentos en Mongoose (2/4)

Borrar documentos que cumplan cierto criterio, a partir de un modelo:

```
const articlesBorrados = await Article.deleteMany({ author: "Bill Gates" });

console.log(articlesBorrados);
```

👉 Ver documentación.

Los criterios de búsqueda son los mismos que cuando se utiliza `find`.

Borrar documentos en Mongoose (3/4)

Borrar un (1) documento que cumpla cierto criterio, dado su modelo:

```
const articleBorrado = await Article.findOneAndDelete({ author: "Bill Gates" });

console.log(articleBorrado);
```

👉 Ver documentación.

👉 También se podría haber usado el método `deleteOne` (ver documentación).

Los criterios de búsqueda son los mismos que cuando se utiliza `find`.

Borrar documentos en Mongoose (4/4)

Borrar un (1) documento que tenga el ID correspondiente, dado su modelo:

```
const articleBorrado = await Article.findByIdAndDelete("5b4d2d0957616e188ef34eb0");

console.log(articleBorrado);
```

👉 Ver documentación.

Editar documentos en Mongoose

Editar documentos en Mongoose (1/4)

Editar un (1) documento, dado su modelo:

```
await Article.updateOne({ title: "Hola Mundo" }, { published: false });
```

👉 Ver documentación.

Editar un (1) documento, dada su instancia:

```
const unArticle = await Article.findOne({ title: "Hola Mundo" });
unArticle.published = false;
unArticle.save();
```

👉 Ver documentación.

Editar documentos en Mongoose (2/4)

Editar documentos que cumplan cierto criterio, dado su modelo:

```
const articlesEditados = await Article.updateMany(
  { author: "Bill Gates" },
  { published: false }
);
console.log(articlesEditados);
```

👉 Ver documentación.

En este caso se editan todos los artículos del autor “Bill Gates” (se despublican).

Editar documentos en Mongoose (3/4)

Editar un (1) documento que cumpla cierto criterio, dado su modelo:

```
const articleEditado = await Article.findOneAndUpdate(
  { author: "Bill Gates" },
  { published: false }
);
console.log(articleEditado);
```

👉 Ver documentación.

👉 Notar que podrían existir varios artículos del autor “Bill Gates”. En este caso, se actualiza sólo uno.

Editar documentos en Mongoose (4/4)

Editar un (1) documento que tenga el ID correspondiente, dado su modelo:

```
const articleEditado = await Article.findByIdAndUpdate(
  "5b4d2d0957616e188ef34eb0",
  { published: false },
);
console.log(articleEditado);
```

👉 Ver documentación.

⚠️ Cuidado: al usar `findByIdAndUpdate` el documento retornado no contiene los nuevos valores. Sin embargo, no sucede lo mismo con `findOneAndUpdate`. Ver docs.

Update Operators

Update Operators (1/2)

Problema: ¿Qué pasa si necesitamos hacer un cambio relativo al valor actual almacenado en la base de datos?

Ej: necesitamos incrementar el campo `likes` de un artículo de un blog.

Con el teórico visto hasta ahora, primero sería necesario buscar el artículo en la BD para obtener su valor actual de `likes`. Y luego habría que actualizarlo con el nuevo valor. Son dos interacciones con la BD para una simple modificación. 👎

Afortunadamente, esto se puede hacer de una forma más eficiente y más segura (de forma atómica).

Update Operators (2/2)

Notar que en Sequelize también existe algo similar. Ver ejemplo.

Los Update Operators son modificadores para usar en operaciones de actualización.

Todos estos operadores llevan el prefijo: `$`.

Ejemplo, el operador de incremento `$inc`:

```
const articleEditado = await Article.findByIdAndUpdate(
  "5b4d2d0957616e188ef34eb0",
  { $inc: { likes: 1 } }
);
```

Por otros operadores, consultar los docs.

Ejercicio

Ejercicio

- Completar la API que se comenzó en clases anteriores, agregando soporte para los métodos HTTP `PUT`, `PATCH` y `DELETE`, aplicando los conceptos aprendidos en esta clase.
- Agregar un campo `goals` al schema de `Team`, de tipo `Number`.
- Crear un *endpoint* `teams/:code/goal` con método `PATCH` que, sin utilizar el `body`, ni hacer *queries*, incremente en 1 el valor de `goals` del equipo con el `code` correspondiente.
- Crear un *endpoint* `teams/:code/goal` con método `DELETE` que decremente la cantidad de goles del equipo.