educación IT
CODE YOUR FUTURE

Bootcamp
Full Stack Engineer

Fase 3 - Desarrollo Back End

Módulo 35

educación IT
CODE YOUR FUTURE

MongoDB CRUD con Mongoose en consola

MongoDB CRUD con Mongoose en consola

Vamos a trabajar en los conceptos fundamentales del uso de Mongoose para realizar **un CRUD hacia MongoDB utilizando la consola**. Se pueden encontrar más detalles de los comandos que revisaremos en este documento del sitio oficial de Mongoose: Mongoose.

Repaso de conceptos

Mongoose es **una capa de modelado de documentos de objetos** (ODM) ubicada **en la parte superior de la API Node.Js MongoDB**. Si viene de un entorno SQL, entonces **Mongoose es similar a un ORM** (mapeo relacional de objetos).

No es obligatorio usar Mongoose **sobre la API nativa de MongoDB** (MongoDB driver), aunque existen algunos beneficios al hacerlo.

¿Por qué elegir Mongoose sobre MongoDB?

- **Esquemas:** Mongoose **le da estructura a nuestros datos**, esta es conocida como ***esquema***.
- **Validación incorporada:** esto significa que **no tenemos que escribir el código adicional que tenía que escribir con el controlador Mongo DB**. Simplemente incluyendo cosas como **`required:true`** en las definiciones de su esquema, Mongoose proporciona **validaciones internas para sus colecciones** (incluidos los tipos de datos).

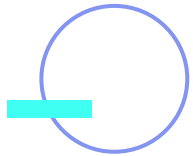
- **Métodos de instancia:** podemos definir **métodos personalizados en un documento con un código mínimo**. Si bien es posible hacer lo mismo en MongoDB, Mongoose facilita la creación y organización de dichos métodos dentro de la definición de su esquema. A su vez, esto significa que tiene más código repetitivo en MongoDB.
- **Devolución de resultados:** la devolución de documentos generados por consultas es generalmente **más fácil en Mongoose**. Un ejemplo son las consultas update. En MongoDB, esta consulta solo devuelve un objeto con un indicador de éxito y la cantidad de documentos modificados.

Mongoose, por otro lado, le proporciona **el documento actualizado para que pueda procesar fácilmente los resultados**.

mongoose

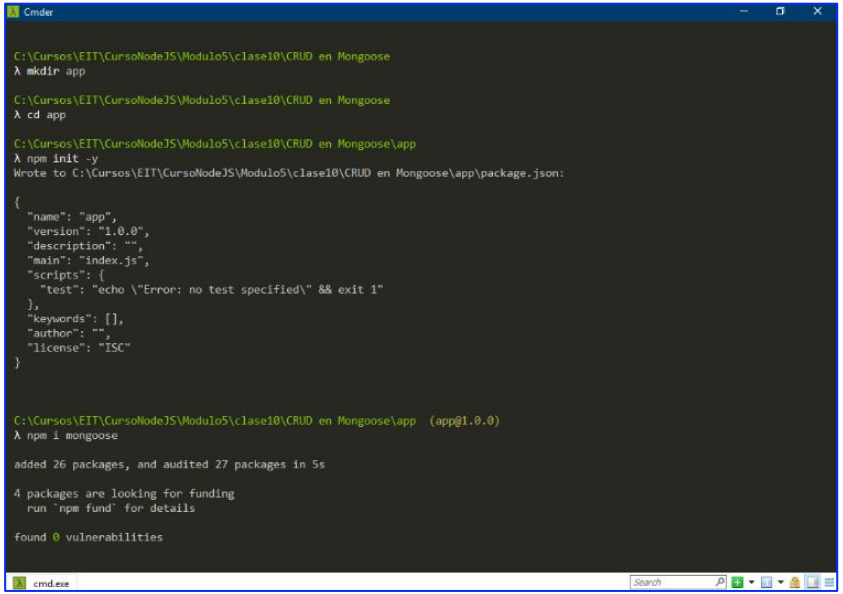
Terminología general

- **Esquemas:** todo en Mongoose comienza con un **esquema**. Cada esquema **se asigna a una colección de MongoDB** y **define la forma de los documentos dentro de esa colección**. Contiene información sobre **propiedades / tipos de campo de documentos** y pueden almacenar información **sobre la validación y los valores predeterminados** y, si se requiere, **una propiedad en particular**. En otras palabras, **son planos de documentos**.
- **Modelo:** un modelo es **una clase con la que construimos documentos**.
- **Documento:** un **registro de datos simple**.
- **Colección: conjunto de documentos**.



Empezando con el proyecto CRUD

Instalamos el módulo **mongoose** utilizando **npm** dentro de un proyecto inicializado de Node.js:



Inicializando nuestra base de datos

Tras instalar Mongoose en el proyecto, usaremos **`mongoose.connect`** para conectarnos a la base de datos de la siguiente manera:

```
const mongoose = require ('mongoose')
mongoose.connect ('mongodb://localhost/test', {useNewUrlParser: true});
```

En este caso, nos conectamos a una base de datos llamada ***Test***. El primer parámetro es el **`URI`** y el segundo **`options`**. El formato de la URL es

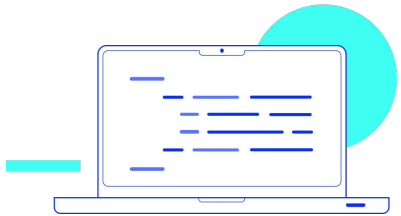
```
type://username:password@host:port/database_name
```

Definiendo un esquema

A continuación, creemos un esquema **BookSchema** que tendrá una propiedad **name** de tipo **String**:

```
const Schema = mongoose.Schema
const BookSchema = new Schema({
    nombre: String
})
```

Esto significa que **los documentos definidos a través de BookSchema tendrán un campo name de tipo String**.



Definición de modelos

1. Con el **`BookSchema`** que acabamos de desarrollar, creemos el modelo Books:

```
const Model = mongoose.model
const Book = Model('Books', BookSchema)
```

El primer parámetro de **`mongoose.model`** es **el nombre de nuestra colección**. El segundo, el **esquema que utilizará el modelo**. En este caso, el nombre de nuestra colección es ***Books*** que utilizará el esquema de **`BookSchema`**.

Ponemos ambas definiciones dentro de un archivo llamado **model.js**, el cual será reutilizado en cada acción CRUD que realicemos en módulos separados.

2. **Model.js** debería verse como en la imagen a continuación:

```
const mongoose = require ('mongoose')

const Schema = mongoose.Schema
const BookSchema = new Schema({
    nombre: String
})

const Model = mongoose.model
const Book = Model('Libros', BookSchema)

module.exports = Book
```

Creación e inserción de documentos

A continuación, crearemos un libro en nuestra base de datos llamado **`NodeJS: Una guía`**.

Ahora, solo resta guardarlo con **`book.save`** ejecutando el código a continuación en el archivo **create.js**:

```
const mongoose = require ('mongoose')
const Book = require('./model')

mongoose.connect ('mongodb://localhost/test', {useNewUrlParser: true}, err => {
    if(err) {
        console.log(`Error en conexión con MongoDB: ${err.message}`)
        process.exit(1)
    }
```

...

```
    console.log('Base de datos conectada')

    const book = new Book({nombre: 'NodeJS: Una guía'})
    book.save((err,result)=>{
        if(err) console.log(err)
        console.log(result);

        mongoose.disconnect()
    })
});
```

En la imagen está la salida del código:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\create.js
Base de datos conectada
{
      nombre: 'NodeJS: Una guía',
      _id: new ObjectId("615c458c81e012955ca8f149"),
      __v: 0
}
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Encontrar y leer documentos

Leer todos los documentos

Para leer documentos, usamos el método **`model.find` *sin especificar una consulta***. Esto nos brinda una lista de todos los documentos disponibles en la colección:

```
model.find ((err, document) => {
   console.log (documento)
})
```

En la siguiente pantalla veremos el ejemplo en nuestro caso.

```
const mongoose = require ('mongoose')
const Book = require('./model')

mongoose.connect ('mongodb://localhost/test', {useNewUrlParser: true}, err => {
    if(err) {
        console.log(`Error en conexión con MongoDB: ${err.message}`)
        process.exit(1)
    }

    console.log('Base de datos conectada')

    Book.find((err,document)=>{
        if(err) console.log(err)
        console.log(document);

        mongoose.disconnect()
    })
});
```

Esta es la salida aplicada:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\read.js
Base de datos conectada
[
   {
      _id: new ObjectId("615c458c81e012955ca8f149"),
      nombre: 'NodeJS: Una guía',
      __v: 0
   }
]
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Leer documentos consultados

Para usar una consulta, especificamos el argumento **`query`** en el método **`model.find`**:

```
model.find (consulta, (err, documento) =>
{
   consola.log (documento)
})
```

Para ilustrar este ejemplo, añadimos un libro en la base de datos titulado ***Ali & Hussain: Best Friends***.

Ejecutemos una consulta y generemos los resultados. Veamos la siguiente pantalla.

```
const mongoose = require ('mongoose')
const Book = require('./model')

mongoose.connect ('mongodb://localhost/test', {useNewUrlParser: true}, err => {
    if(err) {
        console.log(`Error en conexión con MongoDB: ${err.message}`)
        process.exit(1)
    }
    console.log('Base de datos conectada')

    Book.find({nombre: 'Ali & Hussain: Best Friends'},(err,document)=>{
        if(err) console.log(err)
        console.log(document);

        mongoose.disconnect()
    })
});
```

Resultado devuelto **después de la consulta:**

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\read-query.js
Base de datos conectada
[
   {
      _id: new ObjectId("615c458c81e012955ca8f149"),
      nombre: 'Ali & Hussain: Best Friends',
      __v: 0
   }
]
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Actualización de documentos

Existen varios métodos para **actualizar documentos**, que se describen aquí . El más común es **`model.updateOne`**. Actualizaremos el primer documento que coincida con esta consulta. El método se define de la siguiente manera:

```
model.updateOne (consulta, fieldsToUpdate,
(err, result) => {
   // código
})
```

Con la propiedad **`result`** en esta devolución de llamada obtendremos:

- **La cantidad de documentos que coinciden con esta consulta** (n).
- **El número de documentos modificados** (`nModified`).
- Si la operación **fue exitosa o no** (ok).

En este ejemplo, actualicé un libro con el nombre de ***Ali & Hussain: Best Friends*** a ***Maarij And Hussain: Best Friends***. Veamos el siguiente slide.

```
const mongoose = require ('mongoose')
const Book = require('./model')

mongoose.connect ('mongodb://localhost/test', {useNewUrlParser: true}, err => {
    if(err) {
        console.log(`Error en conexión con MongoDB: ${err.message}`)
        process.exit(1)
    }
    console.log('Base de datos conectada')

    Book.updateOne(
        { nombre: 'Ali & Hussain: Best Friends' },
        { nombre:'Maarij And Hussain: Best Friends' },
        (err,res) => {
            if(err) console.log(err)
            console.log(res);

            mongoose.disconnect()
        }
    )
});
```

La salida será la siguiente:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\update.js
Base de datos conectada
{
      acknowledged: true,
      modifiedCount: 1,
      upsertedId: null,
      upsertedCount: 0,
      matchedCount: 1
}
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Usemos el método `find` para iterar a través de nuestros documentos al final. Esta es la salida:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\read.js
Base de datos conectada
[
   {
      _id: new ObjectId(“615c458c81e012955ca8f149”),
      nombre: ‘NodeJS: Una guía’,
      __v: 0
   },
   {
      _id: new ObjectId(“615d885e45d6d8b7e1987585”),
      nombre: ‘Maarij And Hussain: Best Friends’,
      __v: 0
   }
]
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Eliminar documentos

Para llevar adelante esta acción, usaremos model.deleteOne o model.deleteMany. Ambos tienen parámetros similares:

```
model.deleteOne (consulta, (err,
resultado) => {
  // código
})
```

El parámetro **`result`** en la devolución de llamada proporciona un objeto que indica:

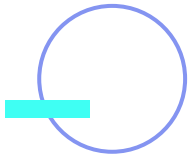
- Cuántos documentos **se eliminaron** (`deletedCount`).
- Cuántos documentos **coincidieron con la consulta** (n).
- Si la operación **fue exitosa o no** (ok).

A continuación, eliminaremos un documento que coincide con el título ***NodeJS: A Guide***. Veamos la imagen de la pantalla siguiente.

```
const mongoose = require ('mongoose')
const Book = require('./model')

mongoose.connect ('mongodb://localhost/test', {useNewUrlParser: true}, err => {
    if(err) {
        console.log(`Error en conexión con MongoDB: ${err.message}`)
        process.exit(1)
    }

    console.log('Base de datos conectada')


    Book.deleteOne({name: 'NodeJS : A Guide'},(err,result)=>{
        if(err) console.log(err)
        console.log(result);

        mongoose.disconnect()
    })
});
```

El resultado será el siguiente:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\delete.js
Base de datos conectada
{ deletedCount: 1 }
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```



Usar el método `find` para iterar los resultados finalmente nos da el siguiente resultado:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\read.js
Base de datos conectada
[
   {
      _id: new ObjectId(“615d885e45d6d8b7e1987585”),
      nombre: ‘Maarij And Hussain: Best Friends’,
      __v: 0
   }
]
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Métodos de esquema personalizados

Los métodos **se adjuntan al esquema** y son **de dos tipos**:

Métodos de instancia

Los documentos **derivados de sus respectivos esquemas** pueden utilizar estos métodos. A modo de ejemplo, actualicé nuestro **`BookSchema`** para crear un método que nos proporcione **el nombre del libro y el autor**.

Veamos la siguiente pantalla con la imagen del ejemplo.

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const BookSchema = new Schema({
    name: String,
    author: String
})

BookSchema.methods.getBookName = function () {
    console.log(
    'The name of this book is ' + this.name + ' and author is ' + this.author
  );
}

const Model = mongoose.model
const Book = Model('Libros', BookSchema)

const myBook = new Book(
   { name: 'Ali & Hussain: Best Friends', author: 'Hussain Arif' }
)
myBook.getBookName();
```

Esta es la salida del código:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\metodos-instancia.js
The name of this book is Ali & Hussain: Best Friends and author is Hussain Arif
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Nota: usar **una declaración de función normal para su función estática** en vez de la sintaxis de flecha. Esto permite preservar el significado de `this` de Mongoose dentro de la función.

Métodos estáticos

Se utilizan cuando **no tengamos un objeto en particular o no lo necesitamos**. Los modelos derivados de sus respectivos esquemas pueden usarlos. Por ejemplo, creemos una función que busque libros en la base de datos **consultando sus nombres** (damos por hecho que los libros ya están guardados):

```
const mongoose = require('mongoose')

const Schema = mongoose.Schema
const BookSchema = new Schema({
    name: String,
    author: String
})
```

```
BookSchema.statics.findByName = function (value) {
    this.find({ nombre: value }, (err, document) => {
        if(err) console.log(err)
        console.log(document);
        mongoose.disconnect()
    })
}

const Model = mongoose.model
const Book = Model('Libros', BookSchema)
//additional code to save books.
//...

mongoose.connect ('mongodb://localhost/test', {useNewUrlParser: true}, err => {
    if(err) {
        console.log(`Error en conexión con MongoDB: ${err.message}`)
        process.exit(1)
    }
    console.log('Base de datos conectada')
    Book.findByName('Maarij And Hussain: Best Friends')
});
```

La salida será la siguiente:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\metodos-estaticos.js
Base de datos conectada
[
   {
      _id: new ObjectId(“615d885e45d6d8b7e1987585”),
      nombre: ‘Maarij And Hussain: Best Friends’,
      __v: 0
   }
]
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Hooks

También conocidos como **funciones de middleware**. Son funciones **a las que se les pasa el control sobre la ejecución de funciones asincrónicas**. Entre ellas tenemos:

- `Validate.`
- `Save.`
- `Remove.`
- `UpdateOne.`
- `DeleteOne.`

A continuación, usaremos una función de **pre-hook** por lo que se ejecutará **antes que una función**. Además, comprobaremos **si existe un libro con el mismo nombre**, de ser así, **el libro no se guardará**. En teoría, el libro titulado Maarij And Ali: Best Friends ya se ha guardado y lo guardaremos otra vez. Veamos el siguiente slide.

```
const mongoose = require('mongoose')

const Schema = mongoose.Schema
const BookSchema = new Schema({
    name: String,
    author: String
})

BookSchema.pre('save', function (next) {
    Book.find({ name: this.name }, (err, document) => {
        if (document.length == 0) {
            return next()
        }
        next('Books like these already exist! Sorry');
        return false;
    })
})
```

...

```
const Model = mongoose.model
const Book = Model('Libros', BookSchema)
const myBook = new Book({ name: 'Maarij And Hussain: Best Friends', author: 'Hussain Arif'
})

//book saving code here
//..

mongoose.connect('mongodb://localhost/test', { useNewUrlParser: true }, err => {
    if (err) {
        console.log(`Error en conexión con MongoDB: ${err.message}`)
        process.exit(1)
    }
    console.log('Base de datos conectada')

    myBook.save((err, document) => {
        if (err) console.log(err)
        console.log(document);
        mongoose.disconnect()
    })
});
```

La salida será la siguiente:

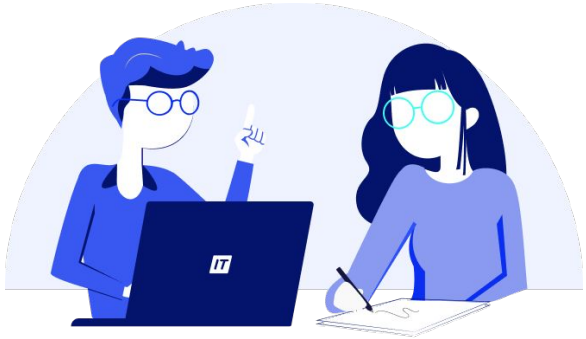
```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\hooks.js
Base de datos conectada
Books like these already exist! Sorry
undefined
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Encontrará todas las funciones de middleware en la página oficial: Mongoose v6.3.5: Middleware.

Campos virtuales

Propiedades que **no necesitan almacenarse en una base de datos**. En general, se utilizan para **propiedades calculadas en documentos**. Por ejemplo, no necesita un campo **`full name`** en una base de datos, simplemente puede combinar **`lastname`** y **`first name`** que están presentes en la base. Los campos virtuales tienen métodos **`get`** y **`set`**.

A continuación, concatenaremos el campo **`author`** y **`name`** y usaremos el método **`get`** para devolverlos. Veamos la siguiente pantalla.



```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const BookSchema = new Schema({
    name: String,
    author: String
})

BookSchema.virtual('fullName')
    .get(function () {
        return this.name + ' , written by: ' + this.author;
    })

const Model = mongoose.model
const Book = Model('Libros', BookSchema)
const myBook = new Book({ name: 'Maarij And Hussain: Best Friends', author: 'Hussain Arif'
})

console.log(myBook.fullName)
```

Esta será la salida del código:

```
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app> node .\campos-virtuales.js
Maarij And Hussain: Best Friends , written by: Hussain Arif
PS C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\CRUD en Mongoose\app>
```

Resumen

Finding

```
model.find(query?,(err,document)=> {
    //code
})
```

Creating

```
document.save((err,document)=>{
     //code
})
```

Updating

```
model.updateOne(query,fieldsToUpdate,(err,result)=>
{
  //code
})
```

Deleting

```
model.deleteOne(query,(err,result)=>{
   //code
})
```

Instance methods

```
schema.methods.functionName = function() {
//code
})
document.functionName()
```

Pre-hook functions

```
schema.pre('functionName' , function() {
//code
})
document.functionName()
```

Statics methods

```
schema.statics.functionName = function() {
//code
})
model.functionName()
```

Virtual fields

```
schema.virtual('fieldName')
       .get() {
        //code
     }.set(value){
       //code
     })
document.fieldName = 'value'
document.fieldName; //returns value
```

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!