UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Avanzado

Clase N° 5: Mongoose

Profesor: Ing. Leandro Rodolfo Gil Carrano
Email: leangilutn@gmail.com

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Schema Types

Mongoose – Schema type

Los tipos de datos que se puede declarar en el schema son los siguientes:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array
- Decimal128
- Map

Mongoose – Schema type (Índices)

- index: boolean, establece un indice con el campo especificado.
- unique: boolean, define un indice de tipo unique

```
var schema2 = new Schema({
  test: {
    type: String,
    index: true,
    unique: true // Unique index. If you specify `unique: true`
    // specifying `index: true` is optional if you do `unique: true`
  }
});
```

Mongoose – Schema type (String)

- lowercase: Se aplica minúsculas a toda consulta
- uppercase: Se aplica mayúsculas a toda consulta
- trim: Quita espacios al elemento aplicado en las consultas
- match: Valida la expresion regular especificada
- enum: Crea un validador en base al array especificado
- minlength: Valida que el numero de caracteres sea mayor al especificado
- maxlength: Valida que el numero de caracteres sea menor al especificado

Mongoose – Schema type (String)

```
const schema1 = new Schema({ name: String }); // name will be cast to string
const schema2 = new Schema({ name: 'String' }); // Equivalent

const Person = mongoose.model('Person', schema2);
```

Se puede declarar el tipo con la clase “String” o como ‘String’

El metodo toString() permite convertir a string cualquier valor (excepto arrays)

```
new Person({ name: 42 }).name; // "42" as a string
new Person({ name: { toString: () => 42 } }).name; // "42" as a string

// "undefined", will get a cast error if you `save()` this document
new Person({ name: { foo: 42 } }).name;
```

Mongoose – Schema type (number / date)

- Min: Valida que el numero ingresado sea mayor al establecido
- Max: Valida que el numero ingresado sea menor al establecido

Mongoose – Schema type (number / date)

```
const schema1 = new Schema({ age: Number }); // age will be cast to a Number
const schema2 = new Schema({ age: 'Number' }); // Equivalent

const Car = mongoose.model('Car', schema2);
```

Se puede declarar el tipo con la clase “Number” o como ‘Number’

Modificadores

Mongoose – Modificadores propios

Set

Por ejemplo tenemos el campo sitioweb que deberia comenzar con ‘http://’ o con ‘https://’, pero en lugar de forzar al cliente a agregar esto en la UI, puedes escribir un modificador personalizado que valide la existencia de estos prefijos y los agregue cuando sea necesario. Para agregar tu modificación personalizado necesitaras crear el nuevo campo sitio web con una propiedad set.

Cada usuario creado tendrá una url de un sitio web bien formada que se modifica en tiempo de creacion.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Mongoose – Modificadores propios

```
var UsuarioSchema = new Schema({
    nombre: String,
    apellido: String,
    email: String,
    usuario: {
        type: String,
        trim: true
    },
    password: String,
    creado: {
        type: Date,
        default: Date.now
    },
    sitioweb:{
        type: String,
        set: function(url){
            if(!url){
               return url;
            } else {
               if(url.indexOf('http://') !== 0 && url.indexOf('https://') !== 0){
                  url = 'http://' + url;
               }
               return url;
            }
        }
    }
});

mongoose.model('Usuario', UsuarioSchema);
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Mongoose – Modificadores propios

Get

Los modificadores getter se usan para modificalos los datos existentes antes de enviar los documentos a la siguiente capa. Por ejemplo en nuestro ejemplo previo un modificador getter a veces seria mejor cambiar el documento de usuario ya existenque modificando su campo sitioweb en tiempo de busqueda, en lugar de recorrer toda la coleccion MongoDB actualizando cada componente.

Con el método **UserSchema.set('toJSON', {getters: true});** habilitamos para que todas en todos los json devueltos por las consultas se aplica el getter

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Mongoose – Modificadores propios

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

var UsuarioSchema = new Schema({
    nombre: String,
    apellido: String,
    email: String,
    usuario: {
        type: String,
        trim: true
    },
    password: String,
    creado: {
        type: Date,
        default: Date.now
    },
    sitioweb:{
        type: String,
        get: function(url){
            if(!url){
                return url;
            } else {
                if(url.indoexOf('http://') !== 0 && url.indexOf('https://') !== 0){
                    url = 'http://' + url;
                }
                return url;
            }
        }
    }
});

UserSchema.set('toJSON', {getters: true});
```

Atributos Virtuales

Mongoose – Virtuales

Algunas veces puedes queres tener propiedades de los documentos calculadas dinamicamente, las cuales no estan realmente presentes en el documento. A estas propiedades se le llaman atributos virtuales y se pueden usar para obtener requisitos comunes.

Por ejemplo digamos que quieres agregar un nuevo campo nombreCompleto, que represente la concatenacion del nombre y del apellido del usuario. Para ello usaremos el metodo virtual()

Mongoose – Virtuales

```
UsuarioSchema.virtual('nombreCompleto').get(function(){
    return this.nombre + ' ' + this.apellido;
});

UsuarioSchema.set('toJSON', {getters: true, virtuals: true});
```

Retornara en el json un nuevo atributo del documento denominado “nombreCompleto”

Mongoose – Virtuales

Pero los atributos virtuales pueden tambien tener setters para ayudar a tus documentos como prefieras en lugar de solamente agregar mas atributos. En este caso digamos que quieres romper la entrada del campo nombreCompleto en sus campos nombre y apellido.

Mongoose – Virtuales

```
UsuarioSchema.virtual('nombreCompleto')
.get(function(){
    return this.nombre + ' ' + this.apellido;
}).set(function(nombreCompleto){
    var nombreDividido = nombreCompleto.split(' ');
    this.nombre = nombreDividido[0] || '';
    this.apellido = nombreDividido[1] || '';
});
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Índices

Mongoose – Modificadores propios

MongoDB soporta varios tipos de índices para optimizar la ejecución de las búsquedas. Mongoose también soporta la funcionalidad de indexado e incluso nos permite definir índices secundarios.

El ejemplo básico de indexación es el índice único, el cual valida la unicidad de un campo de un documento en una colección. En nuestro ejemplo es común que los nombres de usuario sean únicos, así que vamos a modificar la definición de UsuarioSchmea para realizar esto

Mongoose – Modificadores propios

```
var UsuarioSchema = new Schema({
    ...
usuario: {
    type: String,
    trim: true,
    unique: true
},
    ...
});
```

Mongoose – Modificadores propios

Mongoose también soporta la creación de índices secundarios usando la propiedad index. Así si por ejemplo sabes que tu aplicación tendrá muchas búsquedas que conllevan al campo email, podrás optimizar estas búsquedas creando un índice secundario email

Mongoose – Modificadores propios

```
var UsuarioSchema = new Schema({
    ...
email: {
    type: String,
    index: true
},
    ...
});
```

Populate

Mongoose – Populate

Imaginemos una base de datos relacional de libros. Tendríamos una tabla con los títulos de los libros y otra con los datos de los autores. El campo autor en la tabla de libros, apuntaría a un ID o clave primaria de un autor de la tabla autores.

```
const mongoose = require('../bin/mongodb');

var Schema = mongoose.Schema;

var autorSchema = new Schema({
    nombre: String,
    biografia: String,
    fecha_de_nacimiento: Date,
    nacionalidad: String
});

module.exports = mongoose.model('Autor', autorSchema);
```

Mongoose – Populate

supongamos un modelo sencillo para libro de la siguiente manera:

```
const mongoose = require('../bin/mongodb');

var Schema = mongoose.Schema;
var Autor = mongoose.model('Autor');

var libroSchema = new Schema({
    titulo: String
    paginas: Number,
    isbn: String,
    autor: { type: Schema.ObjectId, ref: "Autor" }
});

module.exports = mongoose.model("Libro", libroSchema);
```

Mongoose – Populate

supongamos un modelo sencillo para libro de la siguiente manera:

```
const mongoose = require('../bin/mongodb');

var Schema = mongoose.Schema;
var Autor = mongoose.model('Autor');

var libroSchema = new Schema({
    titulo: String
    paginas: Number,
    isbn: String,
    autor: { type: Schema.ObjectId, ref: "Autor" }
});

module.exports = mongoose.model("Libro", libroSchema);
```

Si nos fijamos, para el campo autor en el modelo libro hemos usado el tipo Schema.ObjectId y la referencia al modelo Autor. Esto nos permitirá establecer la relación entre un campo de una tabla y otra.

Mongoose – Populate

Para consultar los datos lo haremos de la siguiente manera:

```
app.get("/libros", function(req, res) {
    Libro.find({}, function(err, libros) {
        res.status(200).send(libros)
    });
});
```

Luego aplicamos método populate dentro del callback de libros

```
app.get("/libros", function(req, res) {
    Libro.find({}, function(err, libros) {
        Autor.populate(libros, {path: "autor"},function(err, libros){
            res.status(200).send(libros);
        });
    });
});
```

Mongoose – Populate

Para consultar los datos lo haremos de la siguiente manera:

```
app.get("/libros", function(req, res) {
    Libro.find({}, function(err, libros) {
        res.status(200).send(libros)
    });
});
```

Luego aplicamos método populate dentro del callback de libros

```
app.get("/libros", function(req, res) {
    Libro.find({}, function(err, libros) {
        Autor.populate(libros, {path: "autor"},function(err, libros){
            res.status(200).send(libros);
        });
    });
});
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Mongoose – Populate

La línea Autor.populate(libros, {path: "autor"},...); toma el array de objectos libros y le indica que en la ruta autor lo "popule" con los datos del modelo Autor. Quedando una respuesta más completa como este ejemplo:


```
[{
    "_id": "547db17cbe9956a000001",
    "__v": 0
    "titulo": "Juego de Tronos",
    "paginas": 150,
    "isbn": "0-553-57340-4",
    "autor": {
        "_id": "547db17cbe9958b000001",
        "__v": 0,
        "nombre": "George R. R. Martin",
        "biografia": "American novelist...",
        "fecha_de_nacimiento": "1948-09-20T00:00:00.000Z",
        "nacionalidad": "USA"
    }
},
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Mongoose – Populate

En el caso de querer querer guardar datos se debe asociar al campo autor de un libro un object id valido, por ejemplo:

```
app.post("/libros", function(req, res) {
    Autor.findOne({'Nombre':'Pablo'}, function(err, autor) {

        Libros.create({
            titulo: "Test"
            paginas: 5,
            isbn: "123456",
            autor: autor._id
        }, function (err, result) {
            if (err)
            next(err);
            //next('route');
            else
            res.status(200).json({status: "success", message: "Libro added successfully!!!", data: result});

        });

    });
});
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Mongoose – Populate

Utilizando async / await

```
var productModel = require("../models/productsModel")
var categoriasModel = require("../models/categoriasModel")

module.exports = {
    getAll: async function(req, res, next) {
        var producto = await productModel.find({});
        var productoCompleto = await categoriasModel.populate(producto,{path:'categoria'});
        console.log(productoCompleto);
        res.status(200).json({status: "success", message: "ok", data: productoCompleto});

    },
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Mongoose – Populate

Utilizando async / await

```
});
save: async function(req, res, next) {
    var result = await productModel.create({
        name: req.body.name,
        sku: req.body.sku,
        price: req.body.price,
        categoria:req.body.categoria
    });
    res.status(200).json({status: "success", message: "Product added successfully!!!", data: result});
}
```