Mongoose

Manuel Molino Milla

22 de noviembre de 2017

Índice

1. Introducción

- *Moongose* es un modulo de *node.js*
- Hace de puente entre *node.js* y *mongodb*
- Trabaja como un *ORM* (Object-Relational mapping), que es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos.

1.1. Instalacion

```
npm install mongoose
```

1.2. Uso en node.js

```
var mongoose = require('mongoose');
```

1.3. Conexión a la BD

```
mongoose.mongoose.connect('mongodb://localhost/myappdatabase');
```

1.4. Esquemas

Todo en *mongoose* comienza con un esquema. Cada esquema *mapea* la coleccion definida en *MongoDB* y define la forma de acceder a sus datos. Ejemplo:

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

var blogSchema = new Schema({
  title:  String,
  author: String,
  body:   String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs:  Number
  }
});
var Blog =  mongoose.model('Blog', blogSchema);
//importante Blog mayuscula de blog que
//es la coleccion en la base de datos-
```

Los tipos permitidos en este esquema son:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- Objectid
- Array

Ejemplo:

```
var schema = new Schema({
  name:    String,
  binary:  Buffer,
  living:  Boolean,
  updated: { type: Date, default: Date.now },
  age:     { type: Number, min: 18, max: 65 },
  mixed:   Schema.Types.Mixed,
  _someId: Schema.Types.ObjectId,
  array:      [],
  ofString:   [String],
  ofNumber:   [Number],
  ofDates:    [Date],
  ofBuffer:   [Buffer],
  ofBoolean:  [Boolean],
  ofMixed:    [Schema.Types.Mixed],
  ofObjectId: [Schema.Types.ObjectId],
  ofArrays:   [[]]
  ofArrayOfNumbers: [[Number]]
  nested: {
    stuff: { type: String, lowercase: true, trim: true }
  }
})

// example use

var Thing = mongoose.model('Thing', schema);

var m = new Thing;
m.name = 'Statue of Liberty';
m.age = 125;
m.updated = new Date;
m.binary = new Buffer(0);
m.living = false;
m.mixed = { any: { thing: 'i want' } };
m.markModified('mixed');
m._someId = new mongoose.Types.ObjectId;
m.array.push(1);
m.ofString.push("strings!");
m.ofNumber.unshift(1,2,3,4);
m.ofDates.addToSet(new Date);
m.ofBuffer.pop();
m.ofMixed = [1, [], 'three', { four: 5 }];
m.nested.stuff = 'good';
m.save(callback);
```

```
var schema1 = new Schema({
  test: String // `test` is a path of type String
});

var schema2 = new Schema({
  test: { type: String } // `test` is a path of type string
});
```

```
var schema2 = new Schema({
  test: {
    type: String,
    lowercase: true // Always convert `test` to lowercase
  }
});
```

```
var numberSchema = new Schema({
  integerOnly: {
    type: Number,
    get: v => Math.round(v),
    set: v => Math.round(v),
    alias: 'i'
  }
});

var Number = mongoose.model('Number', numberSchema);

var doc = new Number();
doc.integerOnly = 2.001;
doc.integerOnly; // 2
doc.i; // 2
doc.i = 3.001;
doc.integerOnly; // 3
doc.i; // 3
```

```
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
        `
  }
});
```

1.5. Creando datos

```
var User = require('./user');

var newUser = User({
  id : 2001,
  sexo : 'Male',
  edad : 100,
  nombre : "Nuevo usuario desde  moongose"
});

newUser.save(function(err) {
  if (err) throw err;
  console.log("Usuario creado");
});
```

1.6. Leyendo datos

Obteniendo todos los datos:

```
// get all the users
User.find({}, function(err, users) {
  if (err) throw err;

  // object of all the users
  console.log(users);
});
```

Obteniendo un dato determinado:

```
// get the user starlord55
User.find({ username: 'starlord55' }, function(err, user) {
  if (err) throw err;

  // object of the user
  console.log(user);
});
```

Buscando por *id*

```
// get a user with ID of 1
User.findById(1, function(err, user) {
  if (err) throw err;

  // show the one user
  console.log(user);
});
```

Consultas:

```
User.find({first_name : /^M/, id : {$gt : 800}}, function (err,
    data){
  if (err) throw err;
  console.log(data);
});
```

1.7. Actualizaciones

Buscar y actualizar un registros:

```
// find the user starlord55
// update him to starlord 88
User.findOneAndUpdate({ username: 'starlord55' }, { username: '
    starlord88' }, function(err, user) {
  if (err) throw err;

  // // muestra el antiguo
  console.log(user);
  // con {new: true}
  // muestra el nuevo
});
```

Buscar y actualizar múltiples registros:

```
User.update({first_name : 'Manuel'}, {last_name : 'Multiples
    cambios'},
       {multi : true}, function (err, datos) {
                if (err) throw err;
                console.log(datos);
});
```

Buscar por *id* y actualizar

```
// find the user with id 4
// update username to starlord 88
User.findByIdAndUpdate(4, { username: 'starlord88' }, function(err,
     user) {
  if (err) throw err;

  console.log(user);

});
```

1.8. Borrado de datos

Buscar y eliminar a la vez:

```
User.findOneAndRemove({ username: 'starlord55' }, function(err) {
  if (err) throw err;

  // we have deleted the user
  console.log('User deleted!');
});
```

Buscar por *id* y borrar:

```
// find the user with id 4
User.findByIdAndRemove(4, function(err) {
  if (err) throw err;

  // we have deleted the user
  console.log('User deleted!');
});
```

1.9. Mas ejemplos

consultas

2. Ejercicios

- En la colección usada en el tema de *MongoDB* inserta un nuevo dato con valor de *id* igual a *2000*
- Realizando una búsqueda en la colección, indica cuantas personas tienen 100 años de edad.
- Indica las personas que su edad está comprendida entre 100 y 98.
- Busca aquellos documentos que corresponda a personas que sean menores de edad y su nonbre empieza por vocal.
- Actualiza el documento que le corresponde un *id* igual a 2000 y cambia su edad a 18.
- Finalmente borra este documento.