Mongoose: elegante mongod object modeling for node.js

**Vimos o quão sofrido foi para configurar tudo por drivers do mongoDB. Podemos usar Mongoose!**

# ****Configuração inicial****

* Primeiro, vamos deletar nosso fruitsDB

mongo

use fruitsDB

db.dropDatabase()

touch app.js

npm init

npm install mongodb

npm install mongoose@5.7.1

* A conexão não será por drive, e sim , por mongoose

**const mongoose = require('mongoose');**

**mongoose.connect("mongodb://localhost:27017/fruitsDB", { useNewUrlParser: true , useUnifiedTopology: true });**

// it will look for the port '27017' to connect and stablish and will look for the data Base 'fruitsDB'. If it doesn't exist it will create it

* **Vamos agora esquematizar nossa ‘collection’, passando para o app como serão discriminados os objetos dentro daquela ‘collection’. Importante ressaltar que, nesta etapa, vamos dizer qual a ‘unidade de escala’ de cada ‘atributo’.**

**const fruitsSchema = new mongoose.Schema ({**

**name: String,**

**rating: Number,**

**review: String**

**});**

* **Uma vez que temos um esquema, vamos criar uma ‘collection’, e inserir um objeto nela!**

**const Fruit = mongoose.model("Fruit", fruitssSchema);**

// desta forma eu acabei de criar uma 'collection', com o nome 'fruits', dentro do DataBase 'fruitsDB', que obedece a formatação do 'fruitsSchema' !! – veja que o mongoose muda automaticamente o nome dado de “Fruit” para “fruits -

**const fruit = new Fruit ({**

**name: "Apple",**

**score: 8,**

**review: "good"**

**});**

**fruit.save(); // irá salvar o item dentro da 'collection'**

**cuidado: após a primeira save(), comente essa link para não ficar criando uma superposição desnecessária;**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**const peopleSchema = new mongoose.Schema ({**

**name: String,**

**age: Number**

**});**

**const Person = mongoose.model("Person", peopleSchema);**

Ao conferir no MongoShell, veremos que o nome foi convertido de “Person” para ‘people’

**const person = new Person ({**

**name: "John",**

**age: 37**

**});**

**person.save(); // irá salvar o item dentro da 'collection'**

* + ****Também é possível inserir vários itens:****

**const kiwi = new Fruit({**

**name: "Kiwi",**

**score: 10,**

**review: "The best!!"**

**});**

**const banana = new Fruit({**

**name: "Banana",**

**score: 8,**

**review: "Energy!!"**

**});**

Veja: a função ‘insertMany’ pede como parâmetro os nomes dos objetos inseridos !

**Fruit.insertMany([kiwi, banana], function(err){**

**if (err){**

**console.log(err);**

**} else{**

**console.log("Sucess!!");**

**}**

**});**

# ****Reading and Finding****

**Fruit.find(function(err, fruits){**

**if (err) {**

**console.log(err);**

**} else {**

**console.log(fruits);**

**}**

**});**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-**

* ****Eu também posso fazer um filtro de find, com ‘forEach’:****

**Fruit.find(function(err, fruits){**

**if (err) {**

**console.log(err);**

**} else {**

**fruits.forEach(function(fruta){**

**console.log(fruta.name);**

**});**

**}**

Always close the connection at the end of the code

**mongoose.connection.close();**

**});**

* NÃO SE SEQUEÇA DE DAR O “RUN” !

node app.js

# ****Validating Data****

* ****First, it is a good practice to comment out every part that you created some object, so you don’t repeat them.****
* ****Now let’s see how to validate inputs… for ex. The rating must be a number between 1 and 10****

**const fruitsSchema = new mongoose.Schema ({**

**name: String,**

**rating: {**

**type: Number,**

**min: 1,**

**max: 10**

**},**

**review: String**

**});**

* Agora, vejamos como fazer uma validação para não deixar um item sem nome – adicionamos um required:

****const fruitsSchema = new mongoose.Schema ({****

****name: {****

Instead of true, I can say 1

Required: [1, “Forgot the name]

****type: String,****

****required: [true, "Forgot the name!"]****

****},****

****rating: {****

****….****

****.****

****.****

node app.js

# ****Updating and deleting Data****

* ****The strawberry was added without a rating, because it was not valid. Let’s update it.****
* ****Inside the documentation, I can see all the functions that I can call****

**Fruit.updateOne({\_id: "605613f9b62f3e08e0d21622"}, {rating: 9}, function(err){**

**if(err) {**

**console.log(err);**

**}**

**else {**

**console.log("successfully updated");**

**}**

**mongoose.connection.close();**

**});**

* ****Now, lets delete data, for ex., the duplicated UVA:****

**Fruit.deleteOne({ \_id: "605616469edf371f20c50c9f"}, function(err){**

**if(err) {**

**console.log(err);**

**}**

**else {**

**console.log("successfully deleted");**

**}**

**mongoose.connection.close();**

**});**

* ****AND now the many ‘Johns’****

Person.deleteMany({ name: "John"}, function(err){

if(err) {

console.log(err);

}

else {

console.log("Person successfully deleted");

}

});

# ****Establishing relationships****

* ****We now have one registered person, and we can say which fruit is his favorite. The three blocks of code above will do it: we adapt the schema of People, so it has an embedded document as ‘favorite fruit’. Now we create a new fruit, and a new person that has that fruit as favorite.****

**const peopleSchema = new mongoose.Schema ({**

**name: String,**

**age: Number,**

**favouriteFruit: fruitsSchema**

**});**

**const manga = new Fruit ({**

**name: "Manga",**

**rating: 6,**

**review: "manga é muito bom"**

**});**

**manga.save(); // irá salvar o item dentro da 'collection'**

**const person = new Person ({**

**name: "Amy",**

**favoriteFruit: manga,**

**age: 37**

**});**

**person.save(); // irá salvar o item dentro da 'collection'**

* ****Now, lets update the favorite fruit of Jonh:****

**const pineaple = new Fruit ({**

**name: "Abacaxi",**

**rating: 9.9,**

**review: "Tooop"**

**});**

**pineaple.save(); // irá salvar o item dentro da 'collection'**

**Person.updateOne({name: "John"}, {favoriteFruit: pineaple}, function(err){**

**if(err) {**

**console.log(err);**

**}**

**else {**

**console.log("successfully updated");**

**}**

**});**