

Banco de dados + Backend

Reprograma - Backend

Prof^a: Kelly Joany de Oliveira Santos



The logo for mongoose, featuring the word "mongoose" in a dark red, lowercase, sans-serif font. The letters are thick and rounded, with a slight shadow effect on the right side of the letters.

elegant **mongodb** object modeling for **node.js**

Setup

```
$ mkdir mongoose-example
```

```
$ cd mongoose-example
```

```
$ npm init
```

```
$ npm install mongoose
```

```
$ touch example.js
```

Connecting to the Database

Criar arquivo “example.js”

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

```
mongoose
```

```
.connect('mongodb://localhost/exampleApp', {useNewUrlParser: true})
```

```
.then(x => console.log(`Connected to Mongo! Database name: "${x.connections[0].name}"`))
```

```
.catch(err => console.error('Error connecting to mongo', err));
```

Criar uma instância e salvar

```
const Cat = mongoose.model('Cat', { name: String });
```

```
const kitty = new Cat({ name: 'kelly' });
```

```
kitty
```

```
.save()
```

```
.then(newCat => console.log(`A new cat is created: ${newCat}!`))
```

```
.catch(err => console.log(`Error while creating a new cat: ${err}`));
```

Local

5 DBS 3 COLLECTIONS

☆ FAVORITE

HOST
localhost:27017CLUSTER
StandaloneEDITION
MongoDB 4.4.1 Community

Filter your data

- > admin
- > config
- > detran
- > exampleApp
 - cats
- > local
- > restaurants
- > video

exampleApp.cats
Documents

exampleApp.cats

DOCUMENTS 1

TOTAL SIZE
47BAVG. SIZE
47B

INDEXES 1

TOTAL SIZE
20.0KBAVG. SIZE
20.0KB

Documents

Aggregations

Schema

Explain Plan

Indexes

Validation

FILTER

OPTIONS

FIND

RESET

...

ADD DATA



VIEW



Displaying documents 1 - 1 of 1



REFRESH

```
{
  "_id": ObjectId("5f91e08761d0a74e3019d837"),
  "name": "Kelly",
  "__v": 0
}
```

Entendendo uma callback

```
Cat.find({}, (err, cats) => {  
  if(err){  
    console.log(`Error occurred during getting cats from DB: ${err}`);  
    return;  
  }  
  console.log('All the CATS!');  
  // cats is an array of Cat instances  
  cats.forEach(cat => console.log(` --> cat: ${cat.name}`));  
});
```

O mesmo código usando Promisses

```
Cat
  .find()
  .then(catsFromDB => {
    // catsFromDB is an array of Cat instances
    catsFromDB.forEach(oneCat => console.log(` --> cat: ${oneCat.name}`));
  })
  .catch(err => console.log(`Error occurred during getting cats from DB: ${err}`));)
```

Nos bastidores, o `find()` do Mongoose está enviando um comando `find` do MongoDB para o banco de dados. Como chamamos o método `find()` usando o modelo `Cat`, o Mongoose está encontrando os documentos resultantes de dentro da coleção `cats`.

Mongoose connection events

Connection event	Description
<code>mongoose.connection.on('connected', callback)</code>	Call <code>callback</code> when Mongoose is connected.
<code>mongoose.connection.on('error', callback)</code>	Call <code>callback</code> when an error happened on connection.
<code>mongoose.connection.on('disconnected', callback)</code>	Call <code>callback</code> when Mongoose is disconnected.
<code>process.on('SIGINT', callback)</code>	Call <code>callback</code> just before stopping Node (can be simulated with <code><Ctrl>-C</code>)

Mongoose | Schemas, Models & Documents

Resumindo, um Schema fornece nossa estrutura de banco de dados. Isso nos ajuda a garantir a consistência em nosso banco de dados, e temos menos probabilidade de:

- Adicionar campos que não deveriam existir (porque você os definiu em seu esquema).
- Esquecer os campos que são obrigatórios.
- Usar o tipo errado em um campo (ou seja, um número dentro de um campo de data ou um booleano em um campo de número).

```
mongoose.model('Cat', {  
  name: String  
})
```

Mongoose | Schemas, Models & Documents

Vamos dar uma olhada em como definir um esquema adicionando alguns campos ao nosso modelo “Cat”. Vamos criar a pasta “models” e adicionar um arquivo Cat.js dentro da pasta models. Vamos adicionar este snippet de código a ele:

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

const catSchema = new Schema({
  name : String,
  color: String,
  age  : Number
});

const Cat = mongoose.model('Cat', catSchema);
module.exports = Cat;
```

Mongoose | Schemas, Models & Documents

Nosso schema é apenas um objeto com chaves para cada um dos nomes dos campos e valores para cada tipo de campo. Existem muitos tipos de schema disponíveis no Mongoose, mas aqui está uma lista rápida de tipos comuns:

- String (ex: kelly)
- Number (ex: 42)
- Date (ex: Date('2020-12-25'))
- Boolean (ex: true)
- Schema.Types.ObjectId: para armazenar ids de outras coleções
- Schema.Types.Mixed: to store anything
- Array or `[]`: para armazenar um array de qualquer tipo (ex: ['foo', 42])
 - [String]: para armazenar um array de strings (ex: ['foo', 'bar'])
 - [Number]: para armazenar um array de numbers (ex: [42, -6])

Mongoose | Schemas, Models & Documents

Freqüentemente, queremos um valor padrão para algum campo, caso contrário, ele será salvo vazio. Por exemplo, quando um usuário se registra em um site e não fornece uma imagem, você pode definir a imagem como padrão para que o usuário obtenha a imagem de avatar padrão. Você pode fazer isso usando a palavra-chave “default”.

```
const userSchema = new Schema({
  email    : { type: String},
  username : { type: String},
  avatarUrl: { type: String, default: 'images/default-avatar.png' }
});
```

Mongoose | Schemas, Models & Documents

Além de especificar o tipo dos campos ou seu valor padrão, podemos definir uma validação mais detalhada definindo mais propriedades:

Field property	Possible values	Description
<code>type</code>	<code>String</code> , <code>Number</code> , ...	Sets a type for the field
<code>default</code>	Anything	Sets a default value
<code>required</code>	<code>true</code>	Adds a required validator
<code>unique</code>	<code>true</code>	Declares an unique index
<code>enum</code>	An array	Adds an enum validator
<code>min</code>	A number	Sets a minimum number validator
<code>max</code>	A number	Sets a maximum number validator
<code>minlength</code>	A number	Sets a minimum length validator
<code>maxlength</code>	A number	Sets a maximum length validator
<code>trim</code>	<code>true</code>	Adds a trim setter
<code>lowercase</code>	<code>true</code>	Adds an lowercase setter
<code>match</code>	A regex	Sets a regex validator
<code>validate</code>	An object	Adds a custom validator (see next part)
<code>set</code>	A function	Adds a custom setter (see next part)

Mongoose | Schemas, Models & Documents

```
const catSchema = new Schema({
  name: { type: String, required: true },
  age: { type: Number, min: 0, max: 30 },
  color: { type: String, enum: ['white', 'black', 'brown'] },
  avatarUrl: { type: String, default: 'images/default-avatar.png' },
  location: {
    address: String,
    city: String
  },
  countryCode: { type: String, match: /^[A-Z]{2}$/ },
  created: {
    type: Date,
    default: Date.now
  }
});
```

Mongoose | Schemas, Models & Documents

Se você não tiver o validador certo, o Mongoose lhe dá a oportunidade de escrever o seu próprio. Para isso, você precisará adicionar “validate” e uma função “validator” e uma mensagem de erro.

```
// Example of custom validation

const userSchema = new Schema({
  linkedinProfile: {
    type: String,
    validate: {
      validator: (text) => {
        return text.indexOf('https://www.linkedin.com/') === 0;
      },
      message: "linkedinProfile must start with 'https://www.linkedin.com/'"
    }
  }
});
```


Mongoose | Schemas, Models & Documents

O Mongoose também lhe dá a oportunidade de escrever seu “setter”. Cada vez que você definir um campo, o Mongoose transformará o valor através da função especificada por “set”.

```
// Example of custom setter

function capitalize (val) {
  if (typeof val !== 'string') {
    val = '';
  }
  return val.charAt(0).toUpperCase() + val.substring(1);
}

const userSchema = new Schema({
  name: {
    type: String,
    set: capitalize // <= here we call the setter we defined earlier
  }
});
```

Mongoose | Schemas, Models & Documents

Primeiro, vamos ver como podemos inserir (create) alguns documentos. Se você der uma olhada na documentação do Mongoose, verá que pode usar a função `Model.create()`. Vamos ver como fica se quisermos criar um novo usuário com base no modelo de usuário já definido. Para esta demonstração, criaremos `User.js` dentro da pasta “models”:

```
// models/User.js

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

const userSchema = new Schema({
  name: { type: String },
  password: { type: String },
  job: { type: String }
}, {
  timestamps: true
});

const User = mongoose.model('User', userSchema);
module.exports = User;
```

Mongoose | Schemas, Models & Documents

```
User.create({ name: 'Alice',  
              password: "2018",  
              job: 'Architect'  
            }, function (err, user) {  
  if (err) {  
    console.log('An error happened:', err);  
  } else {  
    console.log('The user is saved and its value is: ', user);  
  }  
});
```

```
User.create({ name: 'Alice', password: "ironhack2018", job: 'Architect' })  
  .then(user => { console.log('The user is saved and its value is: ', user) })  
  .catch(err => { console.log('An error happened:', err) });
```

Mongoose | Schemas, Models & Documents

Para recuperar vários documentos de um banco de dados, podemos usar `Model.find ()` e estes são alguns dos detalhes de sua sintaxe:

- O primeiro parâmetro é a consulta representada por um objeto. Se você não colocar, encontrará todos os documentos do seu acervo;
- O segundo parâmetro é opcional e é a projeção representada por um objeto ou string com todos os campos a serem exibidos;
- O terceiro parâmetro é opcional e são as opções, representadas por um objeto;
- O último parâmetro é a função de retorno de chamada executada quando a consulta é concluída. Se você não colocar nenhuma função como parâmetro, ele retornará uma promessa.

Mongoose | Schemas, Models & Documents

```
let callback = (err, users) => {});

// Find all users and execute the callback
User.find({}, callback);

// Find all users where (the name is 'Alice' and the age is >= 18)
// and execute the callback
User.find({ name: 'Alice', age: { $gte: 18 }}, callback);

// Find all users where the name is 'Alice' and only selecting
//the "name" and "age" fields, and afterwards, execute the callback
User.find({ name: 'Alice' }, 'name age', callback);

// Find all users and sort them by name descending
User.find({}, null, {sort: { name: -1 }}, callback);

// Find all users where the name contains 'alice' (insensitive case)
// and execute the callback
User.find({ name: /alice/i}, callback);

// Promise version
User.find({ name: 'Alice' })
  .then(successCallback)
  .catch(errorCallback);
```

Mongoose | Schemas, Models & Documents

No caso de quisermos recuperar apenas um documento específico, podemos usar “findOne” ou “findById”

```
User.findOne({ name: "Alice" })  
  .then(successCallback)  
  .catch(errorCallback);  
  
User.findById("someMongoIdGoesHere129")  
  .then(successCallback)  
  .catch(errorCallback);
```

Mongoose | Schemas, Models & Documents

Para atualizar um campo, você pode usar “updateMany”, “updateOne” ou “findByIdAndUpdate”. Em cada caso, você precisa definir primeiro a condição (uma consulta ou um id) e depois a atualização, como no próximo exemplo. Decidimos aqui usar apenas a sintaxe Promise.

```
// For all users that as "@gmail\.com" in its email, change the company to "reprograma"
User.updateMany({ email: /@gmail\.com/}, { company: "reprograma" })
  .then(successCallback)
  .catch(errorCallback);

// For the first "Alice" found, change the company to "reprograma"
User.updateOne({ name: "Alice"}, { company: "reprograma" })
  .then(successCallback)
  .catch(errorCallback);

// For the user with _id "5a3a7ecbc6ca8b9ce68bd41b", increment the salary by 4200
User.findByIdAndUpdate("5a3a7ecbc6ca8b9ce68bd41b", { $inc: {salary: 4200} })
  .then(successCallback)
  .catch(errorCallback);
```

Mongoose | Schemas, Models & Documents

Para atualizar um campo, você pode usar “deleteMany”, “deleteOne” ou “findByIdAndRemove”, como no próximo exemplo.

```
// Delete all the users that have "@gmail.com" in their email
User.deleteMany({ email: /@gmail\.com/})
  .then(successCallback)
  .catch(errorCallback);

// Delete the first "Alice" found
User.deleteOne({ name: "Alice"})
  .then(successCallback)
  .catch(errorCallback);

// Delete the user with _id "5a3a7ecbc6ca8b9ce68bd41b"
User.findByIdAndRemove("5a3a7ecbc6ca8b9ce68bd41b")
  .then(successCallback)
  .catch(errorCallback);
```


Mongoose | Schemas, Models & Documents

O Mongoose também fornece alguns métodos utilitários. Provavelmente, o que você mais usará é o método “countDocuments”, que permite contar o número de documentos que correspondem a uma condição específica.

```
User.countDocuments({ type: 'student' })  
  .then(count => { console.log(`There are ${count} students`) })  
  .catch(err => { console.log(err) });
```

Mongoose | Schemas, Models & Documents

O Mongoose oferece outra maneira de manipular dados usando os documentos. Podemos usar essa abordagem com:

- Modelos Mongoose, que são classes construtoras que podemos usar para criar novas instâncias do modelo;
- Alguns métodos, como `Model.find ()` em todas as suas formas (`.findOne ()`, `.findById ()`).

Nestes documentos, podemos usar o método `.save ()` para salvar um documento no banco de dados. Podemos usar `.save ()` para criar um novo documento e atualizar um.

Mongoose | Schemas, Models & Documents

```
// ----- CREATE -----  
// Based on already defined User model, create a user Alice ('Architect')  
var myUser = new User({  
  name: 'Alice',  
  password: 'alice2018',  
  job: 'Architect'  
});  
  
myUser.save() // Create a new user and return a promise  
  .then(user => { console.log('The user was created') })  
  .catch(err => { console.log('An error occurred', err) });  
  
// ----- UPDATE -----  
// Find the user with id '5a3a7ecbc6ca8b9ce68bd41b' and update its job and salary  
User.findById('5a3a7ecbc6ca8b9ce68bd41b')  
  .then(user => {  
    user.job = 'Developer';  
    user.salary += 10000;  
    return user.save(); // Update the user '5a3a7ecbc6ca8b9ce68bd41b' and return a promise  
  })  
  .then(user => { console.log('The user was updated: ' + user) })  
  .catch(err => { console.log('An error occurred:', err) });
```

Mongoose | Schemas, Models & Documents

Model method	Description
<code>find</code>	Finds documents
<code>findOne</code>	Finds one document
<code>findById</code>	Finds a single document by its <code>_id</code> field
<code>updateMany</code>	Updates many documents
<code>updateOne</code>	Updates one document
<code>findByIdAndUpdate</code>	Updates a single document based on its <code>_id</code> field
<code>deleteMany</code>	Deletes many documents
<code>deleteOne</code>	Deletes one document
<code>findByIdAndRemove</code>	Removes a single document based on its <code>_id</code> field
<code>countDocuments</code>	Counts number of matching documents in a database collection

Mongoose | Schemas, Models & Documents


Document method	Description
<code>save</code>	Saves this document
<code>toObject</code>	Converts this document into a plain javascript object
<code>toString</code>	Helper for console.log

Mongoose + Backend

```
JS app.js > ...  
const express = require("express")  
const app = express()  
  
const mongoose = require('mongoose');  
mongoose.connect('mongodb://localhost/ToDoList',  
  {  
    useUrlParser: true,  
    useUnifiedTopology: true,  
  });  
  
const index = require("./routes/index")  
const toDoRotas = require("./routes/toDoRoutes")  
  
app.use(express.json())  
  
app.use("/", index)  
app.use("/tarefas", toDoRotas)  
  
module.exports = app
```

Mongoose + Backend

▼  models
  Tarefas.js
 {...} tarefas.json

```
src > models >  Tarefas.js > ...  
 1  const mongoose = require('mongoose');  
 2  
 3  const { Schema } = mongoose;  
 4  
 5  const userSchema = new Schema({  
 6    concluido: { type: Boolean, default: false },  
 7    descricao: String,  
 8    nomeColaborador: { type: String, required: true },  
 9    },  
10    { timestamps: true });  
11  
12  const Tarefas = mongoose.model('Tarefas', userSchema);  
13  
14  module.exports = Tarefas;  
15  
16
```

Mongoose + Backend

```
const { response, request } = require("express")
const mongoose = require('mongoose');
const Task = require('../models/Tarefas');
```


Mongoose + Backend

```
const criarTarefa = (request, response) => {  
  let { descricao, nomeColaborador } = request.body  
  
  /*const novaTarefa = {  
    id: Math.random().toString(32).substr(2,9),  
    dataInclusao: new Date().toString(),  
    concluido: false,  
    descricao: descricao,  
    nomeColaborador: nomeColaborador  
  }  
  
  tarefasModels.push(novaTarefa);  
  
  response.status(201).json(novaTarefa)*/  
  /*const novaTarefa = {  
    id: Math.random().toString(32).substr(2,9),  
    dataInclusao: new Date().toString(),  
    concluido: false,  
    descricao: descricao,  
    nomeColaborador: nomeColaborador  
  }  
  
  tarefasModels.push(novaTarefa); */
```

```
const newTask = new Task({  
  descricao,  
  nomeColaborador,  
});  
  
newTask.save()  
  .then((res) => {  
    response.status(201).json(res);  
  })  
  .catch(err => next(err));
```

Mongoose + Backend

DRAFT

POST

http://localhost:8080/tarefas/cadastro

length: 38 byte(s)

Send

Save as

+ Add an environment

QUERY PARAMETERS

HEADERS ⓘ

Form

☒ Content-Type

:

application/json

×

+ Add header

🔗 Add authorization

🗑

BODY ⓘ

Text

```
1 {
2   "descricao": "teste",
3   "nomeColaborador": "aaaaaa"
4 }
```

Text JSON XML HTML | Format body | ☒ Enable body evaluation

🗑 length: 57 bytes

Mongoose + Backend

Response

Cache Detected - Elapsed Time: 40ms

201 Created

HEADERS

pretty

connection: keep-alive
content-length: 185 bytes
content-type: application/json; charset=utf-8
date: Wed, 28 Oct 2020 20:19:36 GMT
etag: W/"b9-JoB2v/SB1Ige6pHc5DCqVFHuwSk"
x-powered-by: Express

▶ COMPLETE REQUEST HEADERS

BODY

pretty

```
{
  "concluido": false,
  "_id": "5f99d258494fca39c04fb2ca",
  "descricao": "teste",
  "nomeColaborador": "aaaaaa",
  "createdAt": "2020-10-28T20:19:36.531Z",
  "updatedAt": "2020-10-28T20:19:36.531Z",
  "__v": 0
}
```

lines nums

length: 185 bytes

Mongoose + Backend

```
const getAll = (request, response) => {  
  // response.status(200).send(tarefasModels)  
}
```

```
const getAll = (request, response) => {  
  Task.find()  
    .then((tasks) => {  
      response.status(200).json(tasks);  
    })  
    .catch(err => next(err));  
}
```

Mongoose + Backend

DRAFT

Save as

METHOD

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

GET

http://localhost:8080/tafeas

length: 29 byte(s)

► QUERY PARAMETERS

HEADERS ?

Form

► BODY ?

+ Add header

Add authorization

XHR does not allow payloads for GET request.

Response

Cache Detected - Elapsed Time: 29ms

200 OK

HEADERS ?

pretty

X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 738 bytes
ETag: W/"2e2-F0yh7R5ZzR0s0CKInfpIwan1rt8"
Date: Wed, 28 Oct 2020 20:44:28 GMT
Connection: keep-alive

► COMPLETE REQUEST HEADERS

BODY ?

pretty

```
[
  {
    concluido: false,
    _id: "5f987677abbc8046c4dd5e3e",
    descricao: "teste",
    nomeColaborador: "bla",
    createdAt: "2020-10-27T19:35:19.864Z",
    updatedAt: "2020-10-27T19:35:19.864Z",
    __v: 0
  },
  {
    concluido: false,
    _id: "5f9876e6ccf13741bcf8bcae",
    descricao: "teste",
    nomeColaborador: "sssss",
    createdAt: "2020-10-27T19:37:10.812Z",
    updatedAt: "2020-10-27T19:37:10.812Z",
    __v: 0
  }
],
```

Mongoose + Backend

```
const deletarTarefa = (request, response) => {  
  const { id } = request.params  
  /*const tarefaFiltrada = tarefasModels.find(tarefa => tarefa.id == id)  
  
  const index = tarefasModels.indexOf(tarefaFiltrada)  
  tarefasModels.splice(index, 1)  
  
  response.json({mensagem: "Tarefa deletada com sucesso"}) */  
}
```

```
const deletarTarefa = (request, response) => {  
  const { id } = request.params  
  
  Task.findByIdAndDelete(id)  
    .then(() => {  
      response.status(200).json('task deleted');  
    })  
    .catch((err) => {  
      throw new Error(err);  
    });  
}
```

Mongoose + Backend

DRAFT

METHOD

DELETE

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:8080/tarefas/5f9876e6ccf13741bcf8bcae

length: 54 byte(s)

Send

QUERY PARAMETERS

HEADERS

Form

+ Add header

Add authorization

BODY

XHR does not allow payloads for DELETE request.

Response

Cache Detected - Elapsed Time: 24ms

200 OK

HEADERS

pretty

X-Powered-By: Express

Content-Type: application/json; charset=utf-8

Content-Length: 14 bytes

ETag: W/"e-75c1RMquRi73ixC1vxuvfHF84iE"

Date: Wed, 28 Oct 2020 20:53:48 GMT

Connection: keep-alive

COMPLETE REQUEST HEADERS

BODY

pretty

"task deleted"

length: 14 bytes

Top

Bottom

Collapse

Open

2Request

Copy

Download

Mongoose + Backend

```
const atualizarTarefa = (request, response) => {  
  const { id } = request.params //pega o ID na URL  
  /*const { concluido, descricao, nomeColaborador } = request.body //pega os dados enviados pelo usuário no body  
  
  const tarefaAtualizada = tarefasModels.find(tarefa => tarefa.id == id) //procura a tarefa q será atualizada  
  
  const novaTarefa = { //construir o novo objeto editado  
    id: tarefaAtualizada.id, //manter o id que já existe  
    dataInclusao: tarefaAtualizada.dataInclusao, //manter a data que já existe  
    concluido: concluido, //adicionando o valor "concluido" que foi mandado pelo usuario  
    descricao: descricao, //adicionando o valor "descricao" que foi mandado pelo usuario  
    nomeColaborador: nomeColaborador //adicionando o valor "nomeColaborador" que foi mandado pelo usuario  
  }  
  
  const index = tarefasModels.indexOf(tarefaAtualizada) //procuro a posição dentro do JSON do objeto que será atualizado  
  
  tarefasModels[index] = novaTarefa //atribuindo a antiga tarefa a nova que construímos  
  
  response.status(200).json(tarefasModels[index])*/  
}
```


Mongoose + Backend

```
const atualizarTarefa = (request, response) =>{
  const { id } = request.params //pega o ID na URL

  if (!mongoose.Types.ObjectId.isValid(id)) {
    response.status(400).json({ message: 'Specified id is not valid' });
    return;
  }
  Task.findByIdAndUpdate(id, request.body)
    .then(() => {
      response.status(200).json({ message: ` ${request.params.id} is updated successfully.` });
    })
    .catch((err) => {
      response.json(err);
    });
}
```

Mongoose + Backend

METHOD: PUT SCHEME: // HOST: [":" PORT] [PATH ["?" QUERY]]

length: 61 byte(s)

Send

QUERY PARAMETERS

HEADERS 1 2

Form

☒ Content-Type : application/json

+ Add header Add authorization

BODY 2

Text

```
1 {
2   "descricao": "teste",
3   "nomeColaborador": "aaaaaa",
4   "concluido": true
5 }
```

Text JSON XML HTML | Format body | Enable body evaluation

length: 79 bytes

Response

Cache Detected - Elapsed Time: 46ms

200 OK

HEADERS 2

pretty

X-Powered-By: Express

Content-Type: application/json; charset=utf-8

Content-Length: 64 bytes

ETag: W/"40-Lj+WV6i14oT/xV4daIncXSF/U"

Date: Wed, 28 Oct 2020 21:03:47 GMT

Connection: keep-alive

BODY 2

pretty

```
{
  "message": "5f99d258494fca39c04fb2ca is updated successfully."
}
```

lines nums

Top Bottom Collapse Open 2Request Copy Download

Mongoose + Backend

```
/* const tarefa = tarefasModels.find(tarefa => tarefa.id == id)//encontrando a tarefa referente ao ID

tarefa.concluido = concluido//atualizando o campo "concluido" no nosso JSON

response.status(200).json({
  mensagem: "Tarefa concluida",
  tarefa
}) */
```

```
const concluirTarefa = (request, response)=>{
  const { id } = request.params //pegando o valor do ID mandado na URL
  const { concluido } = request.body //pegando o valor de "concluido" enviado no Body

  Task.findByIdAndUpdate(id, { $set: { concluido } })
    .then((task) => {
      response.status(200).json({ message: `${request.params.id} task is finished.`});
    })
    .catch((err) => {
      response.json(err);
    });
}
```

Mongoose + Backend

METHOD: PATCH SCHEME: // HOST: localhost:8080 PATH: /tarefas/naoconcluidas/5f987677abb8046c4dd5e3e

length: 68 byte(s)

QUERY PARAMETERS

HEADERS: Content-Type: application/json

+ Add header Add authorization

Form

BODY

```
1 {
2   "descricao": "teste",
3   "nomeColaborador": "aeaeaea",
4   "concluido": true
5 }
```

Text JSON XML HTML | Format body | Enable body evaluation

length: 79 bytes

Response

Cache Detected - Elapsed Time: 38ms

200 OK

HEADERS: X-Powered-By: Express Content-Type: application/json; charset=utf-8 Content-Length: 56 bytes ETag: W/"38-U41IYyHY9EEIbzFH2Tc3mAZJjk8" Date: Wed, 28 Oct 2020 21:40:40 GMT Connection: keep-alive

pretty

BODY

```
{
  message: "5f987677abb8046c4dd5e3e task is finished."
}
```

lines nums

Top Bottom Collapse Open 2Request Copy Download