


- Back-End(<https://imasters.com.br/back-end>)
- Mobile(<https://imasters.com.br/mobile>)
- Front End(<https://imasters.com.br/front-end>)
- DevSecOps(<https://imasters.com.br/devsecops>)
- Design & UX(<https://imasters.com.br/design-ux>)
- Data(<https://imasters.com.br/data>)
- APIs e Microserviços(<https://imasters.com.br/apis-microservicos>)

## MONGODB

17 JAN, 2023

# Tutorial CRUD em Node.js com driver nativo do MongoDB

 (<https://www.facebook.com/sharer?u=https://imasters.com.br/mongodb/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb>) (<https://twitter.com/share?url=https://imasters.com.br/mongodb/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb>) (<https://www.linkedin.com/shareArticle?url=https://imasters.com.br/mongodb/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb>) COMPARTILHE!

LUIZ FERNANDO DUARTE JUNIOR  
([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))  
Tem 88 artigos publicados com 774400 visualizações desde 2017

LUIZ FERNANDO DUARTE JUNIOR ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))88 

Pós-graduado em computação, trabalha com software desde 2006 nas mais variadas tecnologias. Empreendedor, autor e professor, quando não está ocupado programando, está escrevendo ou gravando sobre programação para seu canal e blog LuizTools.

LEIA MAIS ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))

1 NOV, 2023

JavaScript: Análise e Tratamento de Erros (<https://imasters.com.br/javascript/javascript-analise-e-tratamento-de-erros>)

29 SET, 2023

O que é IPFS – InterPlanetary File System? (<https://imasters.com.br/tecnologia/o-que-e-ipfs-interplanetary-file-system>)

5 SET, 2023

Deploy de aplicação Node.js + PostgreSQL na Amazon AWS (LightSail) (<https://imasters.com.br/banco-de-dados/deploy-de-aplicacao-node-js-postgresql-na-amazon-aws-lightsail>)

**N**ode.js é uma plataforma que permite a você construir aplicações server-side em JavaScript, já falei disso extensivamente [neste artigo \(https://www.luiztools.com.br/post/o-que-e-nodejs-e-outras-5-duvidas-fundamentais/\)](https://www.luiztools.com.br/post/o-que-e-nodejs-e-outras-5-duvidas-fundamentais/).

MongoDB é um banco de dados NoSQL, orientado a documentos JSON(-like), o mesmo formato de dados usado no JavaScript, além de ser utilizado através de funções com sintaxe muito similar ao JS também, diferente dos bancos SQL tradicionais e também já falei disso extensivamente [neste artigo \(https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql/\)](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql/).

Vê algo em comum entre os dois? Pois é, ambos tem o JavaScript em comum e isso facilita bastante o uso destas duas tecnologias em conjunto, motivo de ser uma dupla bem frequente para projetos de todos os tamanhos.

A ideia deste tutorial é justamente lhe dar um primeiro contato prático com ambos, fazendo o famoso CRUD (Create, Retrieve, Update e Delete).

Neste artigo você vai ver:

[Configurando o Node.js \(https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#1\)](https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#1)

[Configurando o MongoDB \(https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#3\)](https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#3)

[Conectando no MongoDB com Node \(https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#4\)](https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#4)

[Cadastrando no banco \(https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#5\)](https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#5)

[Atualizando clientes \(https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#6\)](https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#6)

[Excluindo clientes \(https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#7\)](https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb/#7)

Vamos lá!

## #1 – Configurando o Node.js

Vamos começar instalando o Node.js: apenas clique no link do site oficial (<https://nodejs.org/>) e depois no grande botão verde para baixar o executável certo para o seu sistema operacional (recomendo a versão LTS). Esse executável irá instalar o Node.js e o NPM, que é o gerenciador de pacotes do Node. Caso tenha dificuldade, o vídeo abaixo pode ajudar.



Uma vez com o NPM instalado, vamos instalar o módulo que será útil mais pra frente. Crie uma pasta para guardar os seus projetos Node no computador (recomendo C:\node) e dentro dela, via terminal de comando com permissão de administrador (sudo no Mac/Linux), rode o comando abaixo:

```
1 npm install -g express-generator
```

O Express é o web framework mais famoso da atualidade para Node.js. Com ele você consegue criar aplicações e APIs web muito rápida e facilmente. O que instalamos é o express-generator, um gerador de aplicação de exemplo, que podemos usar via linha de comando, da seguinte maneira:

```
1 express -e -git node-mongo-crud
```

O “-e” é para usar a view-engine (motor de renderização) EJS, ao invés do tradicional Jade/Pug. Já o “-git” deixa seu projeto preparado para versionamento com Git. Aperte Enter e o projeto será criado (talvez ele peça uma confirmação, apenas digite ‘y’ e confirme).

Depois entre na pasta e mande instalar as dependências com npm install:

1	cd node-mongo-crud
2	npm install

Ainda no terminal de linha de comando e, dentro da pasta do projeto, digite:

1	npm start
---	-----------

Isso vai fazer com que a aplicação default inicie sua execução em localhost:3000, que você pode acessar pelo seu navegador.



## Express

Welcome to Express

### #2 – Configurando o MongoDB

OK, agora que temos a estrutura básica vamos fazer mais alguns ajustes em um arquivo que fica na raiz do seu projeto chamado `package.json` (<https://www.luiztools.com.br/post/o-guia-completo-do-package-json-do-node-js/>). Ele é o arquivo de configuração do seu projeto e determina, por exemplo, quais as bibliotecas que você possui dependência no seu projeto.

Vamos agora acessar o [site oficial do MongoDB](https://www.mongodb.com/) (<https://www.mongodb.com/>) e baixar o Mongo. Clique no menu superior em Products > MongoDB Community Edition > MongoDB Community Server e busque a versão mais recente para o seu sistema operacional. Baixe o arquivo e, no caso do Windows, rode o executável que extrairá os arquivos na sua pasta de Arquivos de Programas, o que é está ok para a maioria dos casos.

Dentro da pasta do seu projeto Node, que aqui chamei de workshoptdc, deve existir uma subpasta de nome data, crie ela agora. Nesta pasta vamos armazenar nossos dados do MongoDB.

Pelo prompt de comando, entre na subpasta bin dentro da pasta de instalação do seu MongoDB e digite (no caso de Mac e Linux, coloque um `./` antes do `mongod` e ajuste o caminho da pasta data de acordo):

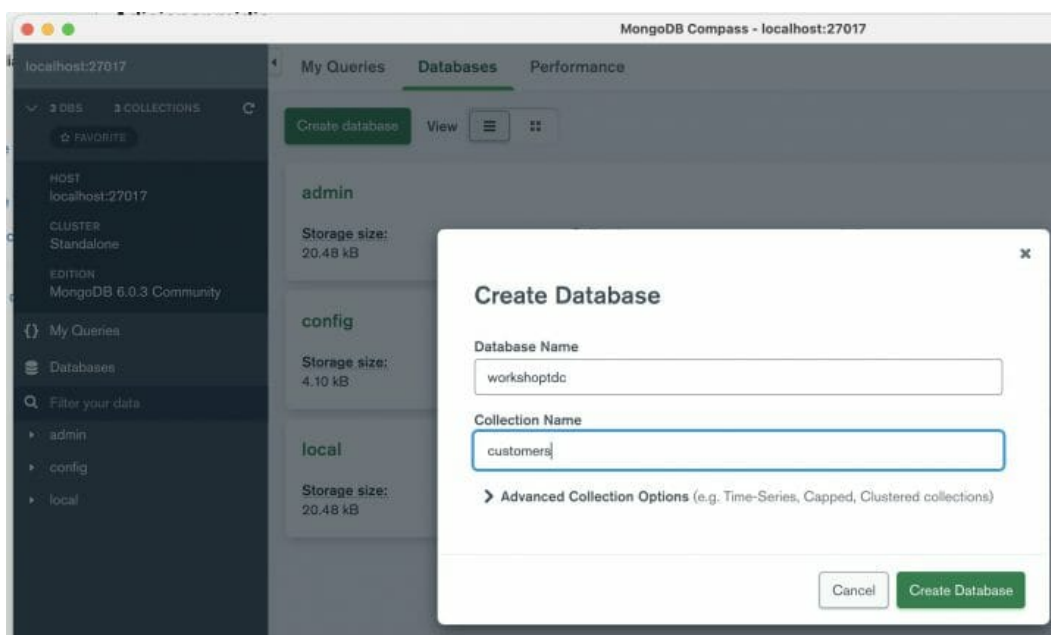
1	mongod --dbpath c:\node-mongo-crud\data
---	---

Isso irá iniciar o servidor do Mongo. Se não der nenhum erro no terminal, o MongoDB está pronto e está executando corretamente.

Agora o cliente que eu recomendo é o MongoDB Compass, que você também baixa pelo site oficial em Products > Tools > MongoDB Compass, sendo que o vídeo abaixo ilustra bem como funciona a ferramenta, caso queira um rápido overview.



Após a conexão funcionar do MongoDB Compass com o seu servidor de MongoDB, você deve criar uma nova database indo no menu lateral e escolhendo a opção Databases > Create Database. Vou usar como nome da base workshopdc e já vou aproveitar e informar o nome da coleção principal que é customer (clientes em inglês).

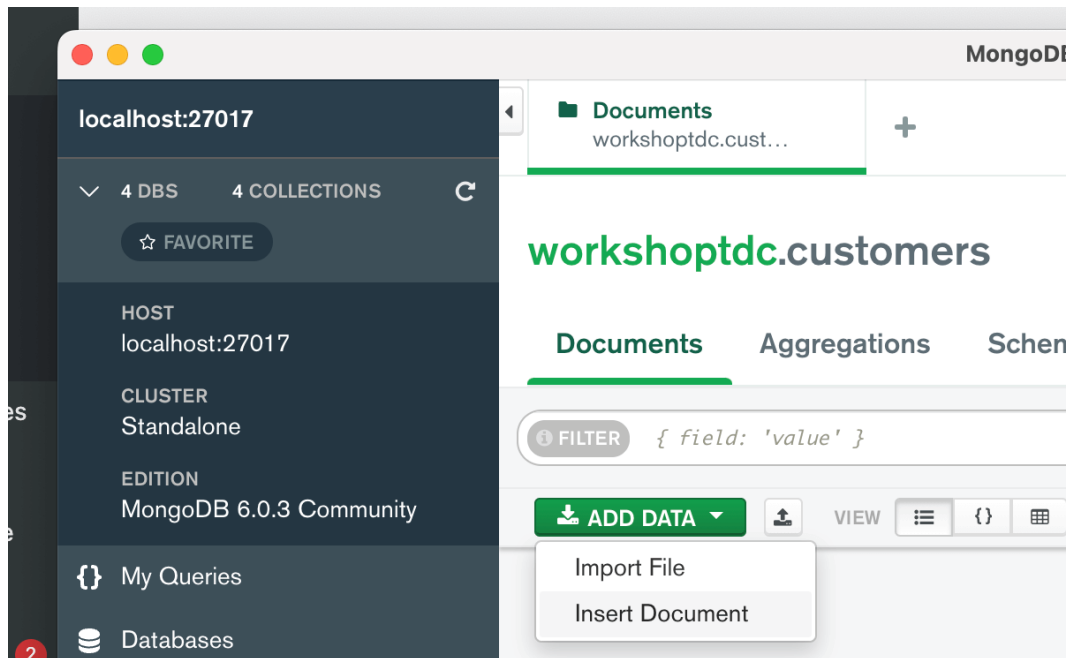


Uma de minhas coisas favoritas sobre MongoDB é que ele usa JSON como estrutura de dados, o que significa curva de aprendizagem zero para quem já conhece o padrão. Caso não seja o seu caso, terá que buscar algum tutorial de JSON na Internet antes de prosseguir.

Vamos adicionar um registro à nossa coleção (o equivalente do Mongo às tabelas do SQL). Para este tutorial teremos apenas uma base de customers (clientes), sendo o nosso formato de dados como abaixo:

1	{
2	"_id" : ObjectId("1234"),
3	"name" : "luiztools",
4	"age" : 32
5	}

O atributo `_id` pode ser omitido, neste caso o próprio Mongo gera um id pra você. No MongoDB Compass, entre na base `workshopdc` e depois na coleção `customers` para ter acesso à opção "Add Data" e em seguida "Insert Document".



Na tela que vai se abrir, adicione um JSON de cliente com o nome de Luiz e a idade de 34. Ou os dados que julgar melhores, sendo que o `_id` pode ser omitido pois vai ser gerado dinamicamente.

```
1 {
2   "name": "Luiz",
3   "age": 34
4 }
```

Após inserir, automaticamente a listagem da ferramenta se atualiza mostrando o registro recém inserido. Adicione alguns registros para ter uma base mínima para o desenvolvimento e agora sim, vamos interagir de verdade com o web server + MongoDB.

### #3 – Conectando no MongoDB com Node

Agora sim vamos juntar as duas tecnologias-alvo deste post!

Precisamos adicionar uma dependência para que o MongoDB funcione com essa aplicação usando o driver nativo. Usaremos o NPM via linha de comando de novo:

```
1 npm install mongodb dotenv
```

Com isso, duas dependências novas serão baixadas para sua pasta `node_modules` e duas novas linhas de dependências serão adicionadas no `package.json` (<https://www.luiztools.com.br/post/o-guia-completo-do-package-json-do-node-js/>) para dar suporte a MongoDB e a variáveis de ambiente (com `DotEnv` (<https://npmjs.com/package/dotenv>)). As variáveis de ambiente precisam ser a primeira coisa a serem carregadas quando a aplicação subir. Então vá no arquivo `bin/www` e adicione essa linha bem no topo.

```
1 require("dotenv").config();
```

Essa instrução irá procurar um arquivo de variáveis de ambiente na raiz do seu projeto, que você deve criar com o nome `.env` e o seguinte conteúdo.

```
1 MONGO_HOST=mongodb://127.0.0.1:27017
2 MONGO_DATABASE=workshoptdc
```

A variável `MONGO_HOST` é o endereço do seu servidor de MongoDB (use IP, nunca localhost), enquanto que a variável `MONGO_DATABASE` é o nome da sua base de dados, então altere conforme julgar necessário.

Agora, para organizar nosso acesso à dados, vamos criar um novo arquivo chamado db.js na raiz da nossa aplicação Express (workshoptdc). Esse arquivo será o responsável pela conexão e manipulação do nosso banco de dados, usando o driver nativo do MongoDB. Adicione estas linhas:

```
1  const { MongoClient, ObjectId } = require("mongodb");
2  let singleton;
3  async function connect() {
4    if (singleton) return singleton;
5    const client = new MongoClient(process.env.MONGO_HOST);
6    await client.connect();
7    singleton = client.db(process.env.MONGO_DATABASE);
8    return singleton;
9  }
10
11
12
13
```

Estas linhas carregam um objeto cliente de conexão a partir do módulo 'mongodb' e depois fazem uma conexão em nosso banco de dados. Essa conexão é armazenada em uma variável global do módulo. A última linha ignore por enquanto, usaremos ela mais tarde.

A seguir, vamos modificar a nossa rota para que ela mostre dados vindos do banco de dados, usando esse db.js que acabamos de criar.

Para conseguirmos fazer uma listagem de clientes, o primeiro passo é ter uma função que retorne todos os clientes em nosso módulo db.js (arquivos JS que criamos são chamados de módulos se possuírem um module.exports no final), assim, adicione a seguinte função ao seu db.js (substituindo aquela linha do module.exports que tinha antes):

```
1  const COLLECTION = "customers";
2  async function findAll() {
3    const db = await connect();
4    return db.collection(COLLECTION).find().toArray();
5  }
6  module.exports = { findAll }
7
8
```

A consulta aqui é bem direta: usamos a função que criamos antes, connect, para obter a conexão e depois navegamos até a collection de customers e fazemos um find sem filtro algum. O resultado desse find é um cursor, então usamos o toArray para convertê-lo para um array e retorná-lo.

Repare na última instrução, ela diz que a nossa função findAll poderá ser usada em outros pontos da nossa aplicação que importem nosso módulo db:

```
1  module.exports = { findAll }
```

Agora vamos programar a lógica que vai usar esta função. Abra o arquivo ./routes/index.js no seu editor de texto (sugiro o Visual Studio Code (<https://code.visualstudio.com/download>)). Dentro dele temos apenas a rota default, que é um get no path raiz. Vamos editar essa rota da seguinte maneira (atenção ao carregamento do módulo db.js no topo):

```
1 const db = require("../db");
2 /* GET home page. */
3 router.get('/', async (req, res, next) => {
4   try {
5     const docs = await db.findAll();
6     res.render('index', { title: 'Lista de Clientes', docs });
7   } catch (err) {
8     next(err);
9   }
10 })
11
```

`router.get` define a rota que trata essas requisições com o verbo GET. Quando recebemos um GET /, a função de callback dessa rota (aquela com `req`, `res` e `next`) é disparada e com isso usamos o `findAll` que acabamos de programar. Como esta é uma função assíncrona que vai no banco de dados, precisamos usar a palavra reservada `await` antes dela, ter um `try/catch` para tratar seus possíveis erros e o callback do router tem de ter a palavra `async` antes dos parâmetros da função.

Foge um pouco do escopo deste tutorial explicar programação assíncrona em Node.js, então recomendo o vídeo abaixo se quiser entender mais do assunto.

### Callbacks, Promises e Async-Await no Node.js



Agora vamos arrumar a nossa view para listar os clientes. Entre na pasta `./views/` e edite o arquivo `index.ejs` para que fique desse jeito:

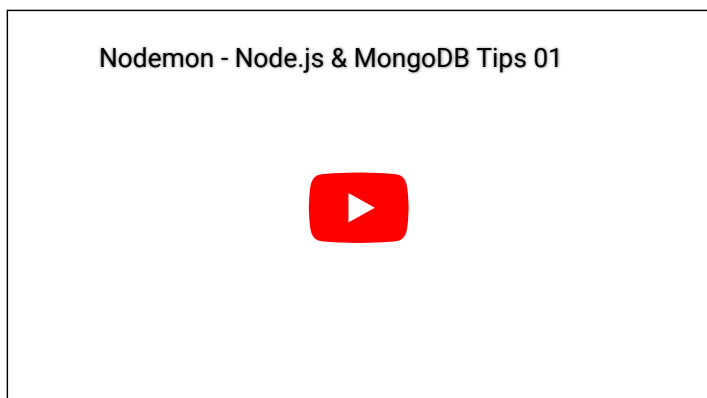
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title><%= title %></title>
5     <link rel='stylesheet' href='/stylesheets/style.css' />
6   </head>
7   <body>
8     <h1><%= title %></h1>
9     <ul>
10      <% docs.forEach(function(customer){ %>
11        <li>
12          <%= customer.name %>
13        </li>
14      <% }) %>
15    </ul>
16  </body>
17 </html>
```

Aqui estamos dizendo que o objeto `docs`, que será retornado pela rota que criamos no passo anterior, será iterado com um `forEach` e seus objetos utilizados um-a-um para compor uma lista não-ordenada com seus nomes.

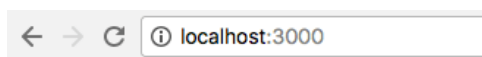
Isto é o bastante para a listagem funcionar. Para não ter que ficar derrubando e subindo novamente a sua aplicação toda hora, modifique o script de start no seu `package.json` para:

1	"scripts": {
2	"start": "npx nodemon ./bin/www"
3	},

Agora suba com `npm start` novamente e toda vez que fizer novas alterações, elas serão carregadas automaticamente por causa da extensão `nodemon`, que eu falo mais a respeito no vídeo abaixo.



Abra seu navegador, acesse <http://localhost:3000/userlist> e maravilhe-se com o resultado.



## Lista de Clientes

- Luiz
- Fernando

Se você viu a página acima é porque sua conexão com o banco de dados está funcionando!

Agora vamos seguir adiante com coisas mais incríveis em nosso projeto de CRUD!

### Parte 4 – Cadastrando no banco

Listar dados é moleza e salvar dados no MongoDB não é algo particularmente difícil. Essencialmente precisamos definir uma rota para receber um POST, ao invés de um GET.

Primeiro vamos criar a nossa tela de cadastro de usuário com dois clássicos e horríveis campos de texto à moda da década de 90. Dentro da pasta `views`, crie um `new.ejs` com o seguinte HTML dentro:

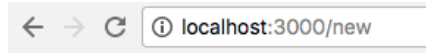
1	<!DOCTYPE html>
2	<html>
3	<head>
4	<title><%= title %></title>
5	<link rel='stylesheet' href='/stylesheets/style.css' />
6	</head>
7	<body>
8	<h1><%= title %></h1>
9	<form action="/new" method="POST">
10	<p>Nome:<input type="text" name="name"/></p>
11	<p>Idade:<input type="number" name="age" /></p>
12	<input type="submit" value="Salvar" />
13	</form>
14	</body>
15	</html>



Agora vamos voltar à pasta routes e abrir o nosso arquivo de rotas, o index.js onde vamos adicionar duas novas rotas. A primeira, é a rota GET para acessar a página new quando acessarmos /new no navegador:

```
1 router.get('/new', (req, res, next) => {  
2   res.render('new', { title: 'Novo Cadastro' });  
3 });
```

Se você reiniciar seu servidor Node e acessar <http://localhost:3000/newuser> verá a página abaixo:



## Novo Cadastro

Nome:

Idade:

Salvar

Se você preencher esse formulário agora e clicar em salvar, dará um erro 404. Isso porque ainda não criamos a rota que receberá o POST desse formulário!.

Então, vamos alterar nosso db.js para incluir uma nova função, desta vez para inserir clientes usando a conexão global e, novamente, incluindo esta função no module.exports:

```
1 async function insert(customer) {  
2   const db = await connect();  
3   return db.collection(COLLECTION).insertOne(customer);  
4 }  
5 module.exports = { findAll, insert }  
6
```

Agora vamos criar uma rota para que, quando acessada via POST, nós chamaremos o objeto global db para salvar os dados no Mongo. A rota será a mesma /new, porém com o verbo POST. Então abra novamente o arquivo /routes/index.js e adicione o seguinte bloco de código logo após as outras rotas e antes do modules.export:

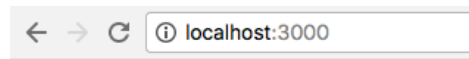
```
1 router.post('/new', async (req, res, next) => {  
2   const name = req.body.name;  
3   const age = parseInt(req.body.age);  
4   try {  
5     const result = await db.insert({ name, age });  
6     console.log(result);  
7     res.redirect('/');  
8   } catch (err) {  
9     next(err);  
10  }  
11 })  
12
```

Obviamente no mundo real você irá querer colocar validações ([ensino aqui \(https://www.luiztools.com.br/post/tutorial-de-validacao-de-input-de-dados-em-nodejs-expressjs/\)](https://www.luiztools.com.br/post/tutorial-de-validacao-de-input-de-dados-em-nodejs-expressjs/)), mas aqui, apenas pego os dados que foram postados no body da requisição HTTP usando o objeto req (request/requisição). Crio um JSON com essas duas variáveis e envio para função insert que criamos agora a pouco.

Novamente, idas ao banco são assíncronas no Node.js, então isso exige o uso de `async` na função que a chamada estiver, o uso de `await` para recebermos o seu retorno e um `try/catch` para tratar erros. Se tudo der certo, a rota redireciona para a `index` novamente para que vejamos a lista atualizada. Apenas para finalizar, edite o `views/index.ejs` para incluir um link para a página `/new`:

```
1 <hr />
2   <a href="/new">Cadastrar novo cliente</a>
3 </body>
4 </html>
5
```

Resultado:



## Lista de Clientes

- Luiz
- Fernando
- teste

[Cadastrar novo cliente](#)

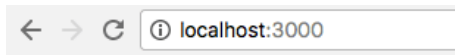
### #5 – Atualizando clientes

Para atualizar clientes (o U do CRUD) não é preciso muito esforço diferente do que estamos fazendo até agora. No entanto, são várias coisas menores que precisam ser feitas e é bom tomar cuidado para não deixar nada pra trás.

Primeiro, vamos editar nossa `views/index.ejs` para que quando clicarmos no nome do cliente, joguemos ele para uma tela de edição. Fazemos isso com uma âncora ao redor do nome, construída no EJS:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title><%= title %></title>
5     <link rel='stylesheet' href='/stylesheets/style.css' />
6   </head>
7   <body>
8     <h1><%= title %></h1>
9     <ul>
10      <% docs.forEach(function(customer){ %>
11        <li>
12          <a href="/edit/<%= customer._id %>">
13            <%= customer.name %>
14          </a>
15        </li>
16      <% }) %>
17    </ul>
18    <hr />
19    <a href="/new">Cadastrar novo cliente</a>
20  </body>
21 </html>
```

Note que este link aponta para uma rota `/edit` que ainda não possuímos, e que após a rota ele adiciona o `_id` do `customer`, que servirá para identificá-lo na página seguinte. Com esse `_id` em mãos, teremos de fazer uma consulta no banco para carregar seus dados no formulário permitindo um posterior `update`. Por ora, apenas mudou a aparência da tela de listagem:



## Lista de Clientes

- [Luiz](#)
- [Fernando](#)
- [teste](#)

---

[Cadastrar novo cliente](#)

Sendo assim, vamos começar criando uma nova função no db.js que retorna apenas um cliente, baseado em seu \_id:

```
1 const ObjectId = require("mongodb").ObjectId;
2 async function findOne(id) {
3   const db = await connect();
4   return db.collection(COLLECTION).findOne(new ObjectId(id));
5 }
6 module.exports = { findAll, insert, findOne }
7
```

Como nosso filtro do find será o id, ele deve ser convertido para ObjectId, pois virá como string na URL e o Mongo não entende Strings como \_ids. Não esqueça de incluir esta nova função no module.exports, que está crescendo.

Agora vamos criar a respectiva rota GET em nosso routes/index.js que carregará os dados do cliente para edição no mesmo formulário de cadastro:

```
1 router.get('/edit/:id', async (req, res, next) => {
2   const id = req.params.id;
3   try {
4     const doc = await db.findOne(id);
5     res.render('new', { title: 'Edição de Cliente', doc, action: '/edit/' + doc._id });
6   } catch (err) {
7     next(err);
8   }
9 })
10
```

Esta rota está um pouco mais complexa que o normal. Aqui nós pedimos ao db que encontre o cliente cujo id veio como parâmetro da requisição (req.params.id). Após ele encontrar o dito cujo, mandamos renderizar a mesma view de cadastro, porém com um model inteiramente novo contendo apenas um documento (o cliente a ser editado) e a action do form da view 'new.ejs'.

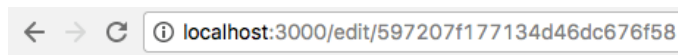
Mas antes de editar a view new.ejs, vamos editar a rota GET em /new para incluir no model dela o cliente e a action, mantendo a uniformidade entre as respostas:

```
1 router.get('/new', (req, res, next) => {
2   res.render('new', { title: 'Novo Cadastro', doc: { "name": "", "age": "" }, action: '/new' });
3 });
```

Agora sim, vamos na new.ejs e vamos editá-la para preencher os campos do formulário com o model recebido, bem como configurar o form com o mesmo model:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title><%= title %></title>
5     <link rel='stylesheet' href='/stylesheets/style.css' />
6   </head>
7   <body>
8     <h1><%= title %></h1>
9     <form action="<%= action %>" method="POST">
10      <p>Nome:<input type="text" name="name" value="<%= doc.name %>" /></p>
11      <p>Idade:<input type="number" name="age" value="<%= doc.age %>" /></p>
12      <input type="submit" value="Salvar" />
13    </form>
14  </body>
15 </html>
```

Agora, se você mandar rodar e clicar em um link de edição, deve ir para a tela de cadastro mas com os campos preenchidos com os dados daquele cliente em questão:



## Edição de Cliente

Nome:

Idade:

Agora estamos muito perto de terminar a edição. Primeiro precisamos criar um nova função no db.js para fazer update, como abaixo:

```
1 async function update(id, customer) {
2   const db = await connect();
3   return db.collection(COLLECTION).updateOne({ _id: new ObjectId(id) }, { $set: customer });
4 }
5 module.exports = { findAll, insert, findOne, update }
6
```

O processo não é muito diferente do insert, apenas temos de passar o filtro do update para saber qual documento será afetado (neste caso somente aquele que possui o id específico). Também já incluí o module.exports atualizado na última linha para que você não esqueça.

Para encerrar, apenas precisamos configurar uma rota para receber o POST em /edit com o id do cliente que está sendo editado, chamando a função que acabamos de criar:

```
1 router.post('/edit/:id', async (req, res) => {
2   const id = req.params.id;
3   const name = req.body.name;
4   const age = parseInt(req.body.age);
5   try {
6     const result = await db.update(id, { name, age });
7     console.log(result);
8     res.redirect('/');
9   } catch (err) {
10    next(err);
11  }
12 })
13
```

Aqui carreguei o id que veio como parâmetro na URL, e os dados de nome e idade no body da requisição (pois foram postados via formulário). Apenas passei esses dados nas posições corretas da função de update e passei um callback bem simples parecido com os anteriores, mas que redireciona o usuário para a index do projeto em caso de sucesso, voltando à listagem.

E com isso finalizamos o update!

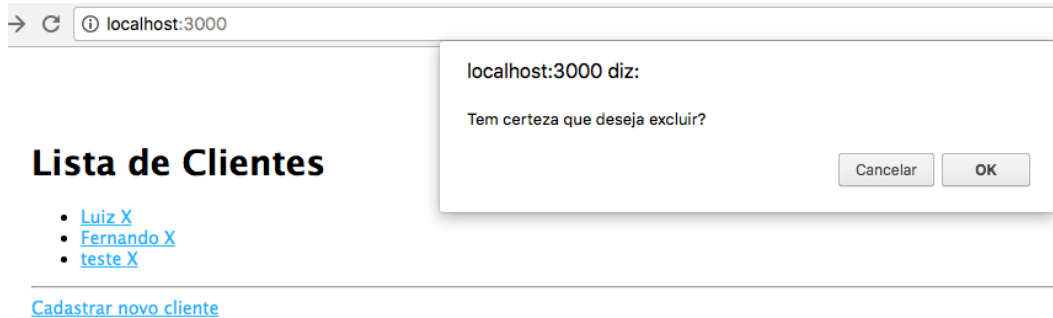
## #6 – Excluindo clientes

Agora em nossa última parte do tutorial, faremos o D do CRUD, D de Delete!

Vamos voltar à listagem e adicionar um link específico para exclusão, logo ao lado do nome de cada cliente, incluindo uma confirmação de exclusão nele via JavaScript, como abaixo:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title><%= title %></title>
5     <link rel='stylesheet' href='/stylesheets/style.css' />
6   </head>
7   <body>
8     <h1><%= title %></h1>
9     <ul>
10      <% docs.forEach(function(customer){ %>
11        <li>
12          <a href="/edit/<%= customer._id %>">
13            <%= customer.name %>
14          </a>
15          <a href="/delete/<%= customer._id %>"
16            onclick="return confirm('Tem certeza que deseja excluir?');">
17            X
18          </a>
19        </li>
20      <% }) %>
21    </ul>
22    <hr />
23    <a href="/new">Cadastrar novo cliente</a>
24  </body>
25 </html>
```

Note que não sou muito bom de front-end, então sinta-se à vontade para melhorar os layouts sugeridos. Aqui, o link de cada cliente aponta para uma rota /delete passando \_id do mesmo. Essa rota será acessada via GET, afinal é um link, e devemos configurar isso mais tarde.



Vamos no db.js adicionar nossa última função, de delete:

```
1 async function deleteOne(id) {
2   const db = await connect();
3   return db.collection(COLLECTION).deleteOne({ _id: new ObjectId(id) });
4 }
5 module.exports = { findAll, insert, findOne, update, deleteOne }
6
```

Essa função é bem simples de entender se você fez todas as operações anteriores. Na sequência, vamos criar a rota GET /delete no routes/index.js:

```
1 router.get('/delete/:id', async (req, res) => {
2   const id = req.params.id;
3   try {
4     const result = await db.deleteOne(id);
5     console.log(result);
6     res.redirect('/');
7   } catch (err) {
8     next(err);
9   }
10 })
11
```

Nessa rota, após excluirmos o cliente usando a função da variável global.db, redirecionamos o usuário de volta à tela de listagem para que a mesma se mostre atualizada.

E com isso finalizamos nosso CRUD!


A continuação deste tutorial você confere [neste post \(https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb-2/\)](https://www.luiztools.com.br/post/tutorial-crud-em-node-js-com-driver-nativo-do-mongodb-2/).



De 0 a 10, o quanto você recomendaria este artigo para um amigo?



#### ARTIGOS PUBLICADOS POR ESTE AUTOR

 LUIZ FERNANDO DUARTE JUNIOR ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))  
1 NOV, 2023

JavaScript: Análise e Tratamento de Erros (<https://imasters.com.br/javascript/javascript-analise-e-tratamento-de-erros>)

 LUIZ FERNANDO DUARTE JUNIOR ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))  
29 SET, 2023

O que é IPFS – InterPlanetary File System? (<https://imasters.com.br/tecnologia/o-que-e-ipfs-interplanetary-file-system>)



LUIZ FERNANDO DUARTE JUNIOR ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))  
5 SET, 2023



Deploy de aplicação Node.js + PostgreSQL na Amazon AWS (LightSail) (<https://imasters.com.br/banco-de-dados/deploy-de-aplicacao-node-js-postgresql-na-amazon-aws-lightsail>)



LUIZ FERNANDO DUARTE JUNIOR ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))  
24 AGO, 2023



Integração de Smart Contract com Uniswap (<https://imasters.com.br/arquitetura-da-informacao/integracao-de-smart-contract-com-uniswap>)



LUIZ FERNANDO DUARTE JUNIOR ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))  
18 AGO, 2023



Tutorial de Smart Contract para Staking em Solidity (<https://imasters.com.br/tecnologia/159823-2>)



LUIZ FERNANDO DUARTE JUNIOR ([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))  
8 AGO, 2023



Erros comuns ao se integrar com a PancakeSwap (<https://imasters.com.br/tecnologia/erros-comuns-ao-se-integrar-com-a-pancakeswap>)



SAIBA MAIS  
([HTTPS://IMASTERS.COM.BR/PERFIL/LUIZFERNANDO](https://imasters.com.br/perfil/luizfernando))

Luiz Fernando Duarte Junior



(<http://fb.com/luiztools>)



(<https://twitter.com/luiztools>)



(<http://www.l>)

88 Artigo(s)

Pós-graduado em computação, trabalha com software desde 2006 e é professor, quando não está ocupado programando, está escrevendo blog LuizTools.

1 comentário

Classificar por **Mais antigos**

Adicione um comentário...



**Sapi Sopi**

Mainkan slot online yang lagi tergacor dikala ini di web Dewapoker. Sebagai Web Slot Gacor, ( <https://199.192.24.140> ) pula tetap membagikan berbagai jenis keuntungan yang bisa kalian miliki dalam melakukan taruhan judi online semacam promo bonus yang bisa kalian miliki masing- masing harinya. Dewapoker sajikan bonus new member, bonus deposit tiap hari, bonus rebate hingga pula bonus cashback yang bisa kalian miliki masing- masing minggunya.

Curtir · Responder · 45 sem

[Plugin de comentários do Facebook](#)

ASSINE NOSSA  
Newsletter

Fique em dia com as novidades do iMasters! Assine nossa newsletter e receba conteúdos especiais curados por nossa equipe



Qual é o seu e-mail?

ASSINAR



[SOBRE O IMASTERS \(HTTPS://IMASTERS.COM.BR/P/SOBRE-O-IMASTERS\)](https://imasters.com.br/p/sobre-o-imasters)

[POLÍTICA DE PRIVACIDADE \(HTTPS://IMASTERS.COM.BR/P/POLITICA-DE-PRIVACIDADE\)](https://imasters.com.br/p/politica-de-privacidade)

[FALE CONOSCO \(HTTPS://IMASTERS.COM.BR/FALE-CONOSCO/\)](https://imasters.com.br/faq)

[QUERO SER AUTOR \(HTTPS://IMASTERS.COM.BR/P/QUERO-SER-AUTOR\)](https://imasters.com.br/p/quero-ser-autor)

[FÓRUM \(HTTPS://FORUM.IMASTERS.COM.BR/\)](https://forum.imasters.com.br/)

[IMASTERS BUSINESS \(HTTP://BUSINESS.IMASTERS.COM.BR\)](http://business.imasters.com.br)