Tutorial CRUD em Node.js com driver nativo do MongoDB

Workshop Node+Mongo no TDC SP

Atualizado em 18/07/2020! Este post possui aula em vídeo no meu curso de Node.js e MongoDB.

Tem algum tempo que escrevi um post ensinando como usar Node.js e MongoDB pra fazer listagem e cadastro de usuários (apenas o CR do CRUD), mas na ocasião usei Mongoose, um ORM, para abstrair o acesso ao MongoDB. No entanto, essa abstração pod ter causado confusão na cabeça de alguns leitores e resolvi escrever este outro post aqui desta vez fazendo um CRUD completo usando Node.js (com Express + EJS) e MongoDB,

mas com o driver nativo do mesmo, que é quase idêntico a como usamos MongoDB via terminal.

O conteúdo deste post também vai servir de apoio para os alunos do workshop de Node.js + MongoDB que ministrei recentemente no TDC SP e cujos slides você confere no final do post.

Neste artigo você vai ver:

Querendo algo mais “pesado” de MongoDB, sugiro este post aqui, focado nesta tecnologia de banco de dados.

Prefere assistir a um vídeo ao invés de ler um post grande? O vídeo abaixo é uma aula gratuita do meu curso de Node.js e MongoDB. Aproveite!

#1 – Configurando o Node.js

Nesta parte vamos instalar e configurar o ambiente necessário para o restante do tutorial.

Passo 1: Instalar Node.js

Bem fácil: apenas clique no link do site oficial e depois no grande botão verde para baixar o executável certo para o seu sistema operacional. Esse executável irá instalar o Node.js e o NPM, que é o gerenciador de pacotes do Node.

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
```

Passo 2: Crie um projeto Express

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.

Você pode rapidamente criar a estrutura básica de um projeto Express via linha de comando, da seguinte maneira (aqui considero que você salva seus projetos na pasta C:node):

```
1 express -e --git workshoptdc
```

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 workshoptdc
2 npm install
```

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

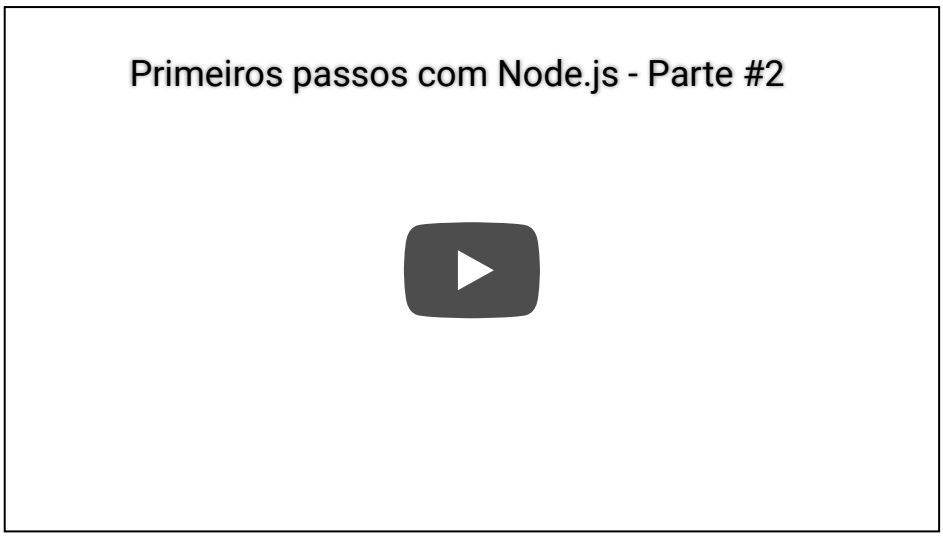
```
npm start
```

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

Happy Hello World!

#2 – Entendendo o projeto Express

O conteúdo desta parte 2 você pode conferir também no vídeo abaixo, se preferir:



Vamos abrir agora o arquivo app.js, que fica dentro do diretório da sua aplicação NodeJS (workshoptdc no meu caso). Este arquivo é o coração da sua aplicação, embora não exista

nada muito surpreendente dentro. Você deve ver algo parecido com isso logo no início:

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./routes/index');
var users = require('./routes/users');
```

Isto define um monte de variáveis JavaScript e referencia elas a alguns pacotes, dependências, funcionalidades do Node e rotas. Rotas são como uma combinação de models e controllers nesta configuração – elas direcionam o tráfego e contém também alguma lógica de programação (embora você consiga, se quiser, fazer um MVC mais puro se desejar). Quando criamos o projeto Express, ele criou estes códigos JS pra gente e vamos ignorar a rota 'users' por enquanto e nos focar no index, controlado pelo arquivo ./routes/index.js.

Na sequência você deve ver:

```
var app = express();
```

Este é bem importante. Ele instancia o Express e associa nossa variável app à ele. A próxima seção usa esta variável para configurar coisas do Express.

```
// view engine setup
app.engine('html', require('ejs').renderFile);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);
```

Isto diz ao app onde ele encontra suas views, qual engine usar para renderizar as views (EJS) e chama algumas funções para fazer com que as coisas funcionem. Note também que esta linha final diz ao Express para acessar os objetos estáticos a partir de uma pasta /public/, mas no navegador elas aparecerão como se estivessem na raiz do projeto. Por exemplo, a pasta images fica em ./public/images mas é acessada em http://localhost:3000/images

```
// catch 404 and forward to error handler
```

```
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handlers

// development error handler
// will print stacktrace
if (app.get('env') === 'development') {
  app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.render('error', {
      message: err.message,
      error: err
    });
  });
}

// production error handler
// no stacktraces leaked to user
app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.render('error', {
    message: err.message,
    error: {}
  });
});
```

Estes são manipuladores de erros para desenvolvimento e produção (além dos 404). Não vamos nos preocupar com eles agora, mas resumidamente você tem mais detalhes dos erros quando está operando em desenvolvimento.

```
module.exports = app;
```

Uma parte importantíssima do Node é que basicamente todos os módulos exportam um objeto que pode ser facilmente chamado em qualquer lugar no código. Nosso app master exporta seu objeto app.

OK! Vamos em frente, agora mexendo com persistência de dados.

#3 – Configurando o MongoDB

OK, agora que temos, entendemos e aprendemos a alterar a estrutura básica vamos fazer mais alguns ajustes em um arquivo que fica na raiz do seu projeto chamado package.json. Ele é o arquivo de configuração do seu projeto e determina, por exemplo, quais as bibliotecas que você possui dependência no seu projeto.

Passo 1: Instalar o MongoDB

Agora vamos deixar nosso editor de texto um pouco de lado e voltar ao prompt de comando. Na verdade vamos primeiro usar nosso navegador para acessar o site oficial do MongoDB e baixar o Mongo. Clique no link de download e busque a versão do Community Server 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, seguido de uma pasta server/version, o que é está ok para a maioria dos casos, mas que eu prefiro colocar em C:Mongo.

Passo 2: Executar mongod e mongo

Dentro da pasta do seu projeto Node, que aqui chamei de workshoptdc, deve existir uma subpasta data. Você pode criar manualmente ou via terminal:

```
1 mkdir data
```

Nesta pasta vamos armazenar nossos dados do MongoDB. Se este diretório não for criado, teremos problemas mais tarde.

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:nodeworkshopdata
```

Isso irá iniciar o servidor do Mongo. Uma vez que apareça no prompt “[initandlisten] waiting for connections on port 27017”, está pronto, o servidor está executando corretamente. Em versões mais recentes, ele dá uma mensagem diferente, mas e procurar algumas acima, irá ver esta mensagem.

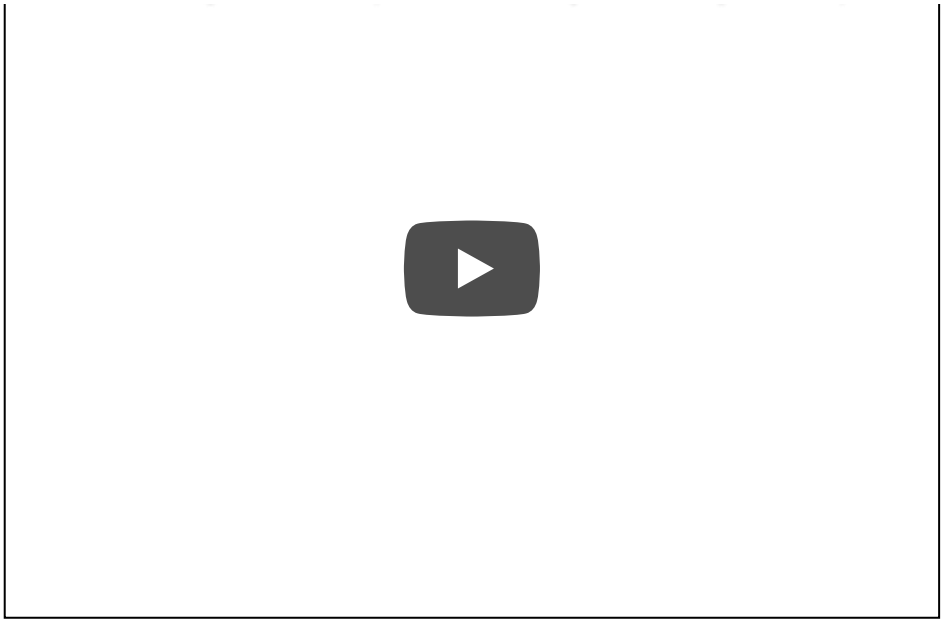
Agora abra outro prompt de comando (o outro ficará executando o servidor) e novamente dentro da pasta bin do Mongo, digite:

```
1 mongo
```

Após a conexão funcionar, se você olhar no prompt onde o servidor do Mongo está rodando, verá que uma conexão foi estabelecida. mongod é o executável do servidor, e mongo é o executável de cliente, que você acabou de conectar.

Opcionalmente você pode usar ferramentas visuais como o MongoDB Compass, que é gratuita.

MongoDB Compass - Node.js & MongoDB Tips ...



Passo 3: Criando uma base de dados

No console do cliente mongo, digite:

```
use workshoptdc
```

Agora estamos usando a base “workshoptdc.” No entanto, ela somente será criada de verdade quando adicionarmos registros nela, o que faremos a partir do próprio cliente para exemplificar.

Passo 4: Inserindo alguns dados

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:

```
{
"_id" : ObjectId("1234"),
"nome" : "luiztools",
```

```
4 "idade" : 29
5 }
```

O atributo _id pode ser omitido, neste caso o próprio Mongo gera um id pra você. No seu cliente mongo, digite:

```
1 db.customers.insert({ "nome" : "Luiz", "idade" : 32 })
```

Uma coisa importante aqui: “db” é a base de dados na qual estamos conectados no momento, que um pouco antes havíamos definido como sendo “workshoptdc”. A parte “customers” é o nome da nossa coleção, que passará a existir assim que adicionarmos um objeto JSON nela. Tecle Enter para que o comando seja enviado ao servidor. Se tudo deu certo, uma respostas com “nInserted: 1” (number of inserted) deve aparecer.

Para ver se o registro foi parar no banco realmente, digite:

```
1 db.customers.find().pretty()
```

O pretty() no final do comando find() é para identar o resultado, que retornará:

```
1 {
2 "_id" : ObjectId("5202b481d2184d390cbf6eca"),
3 "nome" : "Luiz",
4 "idade" : 29
5 }
```

Claro, o seu _id pode ser diferente desse, uma vez que o Mongo irá gerá-lo automaticamente. Isto é tudo que precisamos saber de MongoDB no momento, o que me parece bem fácil, aliás! Para saber mais sobre o MongoDB, Google!

Agora vamos adicionar mais alguns registros no seu console mongo:

```
1 custArray = [{ "nome" : "Fernando", "idade" : 29 }, { "nome" : "Teste", "idade" : 20 }
2 db.customers.insert(custArray);
```

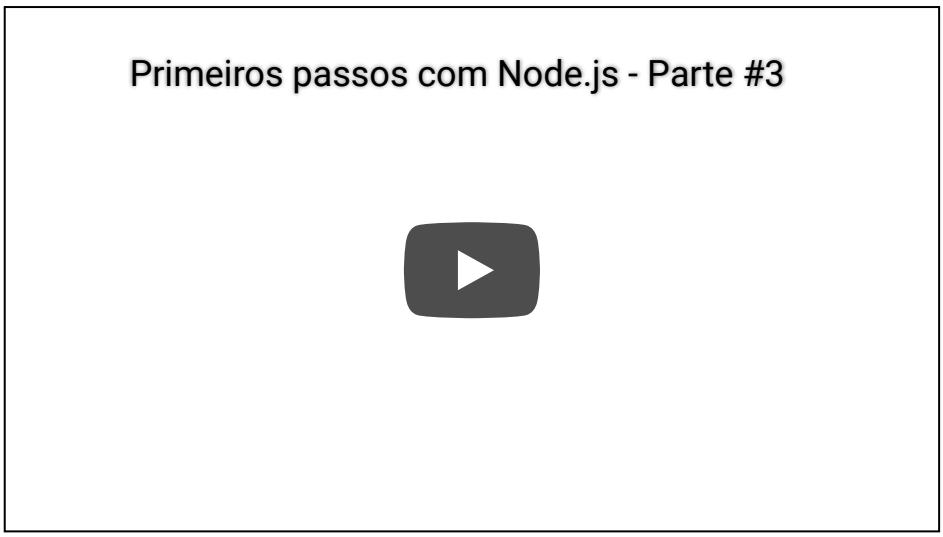
Nesse exemplo passei um array com vários objetos para nossa coleção. Usando novamente o comando db.customers.find().pretty() irá mostrar que todos foram salvos no banco. Agora sim, vamos interagir de verdade com o web server + MongoDB.

Curso Node.js e MongoDB

#4 – Conectando no MongoDB com Node

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

O conteúdo desta parte 4, de maneira resumida, pode ser visto no vídeo abaixo também:



Passo 1: Instalando a dependência do Mongo

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:

```
npm install mongodb
```

Com isso, uma dependência nova será baixada para sua pasta node_modules e uma novas linha de dependência será adicionada no package.json para dar suporte a MongoDB.

Passo 2: Organizando o acesso ao banco

Primeiramente, 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:

```
const mongoClient = require("mongodb").MongoClient;
mongoClient.connect("mongodb://localhost", { useUnifiedTopology: true })
            .then(conn => global.conn = conn.db("workshoptdc"))
            .catch(err => console.log(err))

module.exports = { }
```

Estas linhas carregam o objeto mongoClient a partir do módulo 'mongodb' e depois fazem uma conexão em nosso banco de dados localhost, sendo 27017 a porta padrão do MongoDB. Essa conexão é armazenada globalmente, para uso posterior e em caso de erro, o mesmo é logado no console.

A última linha ignore por enquanto, usaremos ela mais tarde. Agora abra o arquivo www que fica na pasta bin do seu projeto Node e adicione a seguinte linha no início dele:

```
global.db = require('../db');
```

Nesta linha nós estamos carregando o módulo db que acabamos de criar e guardamos o resultado dele em uma variável global. Ao carregarmos o módulo db, acabamos fazendo a conexão com o Mongo e retornamos aquele objeto vazio do module.exports, lembra? Usaremos ele mais tarde, quando possuir mais valor.

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.

Passo 3: Criando a função de consulta

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:

```
function findAll(callback){
    global.conn.collection("customers").find({}).toArray(callback);
}
```

Nesta função 'findAll', esperamos uma função de callback por parâmetro que será executada quando a consulta no Mongo terminar. Isso porque as consultas no Mongo são assíncronas e o único jeito de conseguir saber quando ela terminou é executando um callback.

A consulta aqui é bem direta: usamos a conexão global conn para navegar até a collection de customers e fazer um find sem filtro algum. O resultado desse find é um cursor, então usamos o toArray para convertê-lo para um array e quando terminar, chamamos o callback para receber o retorno.

Agora no final do mesmo db.js, modifique o module.exports para retornar a função findAll. Isso é necessário para que ela possa ser chamada fora deste arquivo:

```
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). Dentro dele temos apenas a rota default, que é um get no path raiz. Vamos editar essa rota da seguinte maneira:

```
/* GET home page. */
router.get('/', function(req, res) {
  global.db.findAll((e, docs) => {
      if(e) { return console.log(e); }
      res.render('index', { title: 'Lista de Clientes', docs: docs });
  })
})
```

router.get define a rota que trata essas requisições com o verbo GET. Quando recebemos um GET /, a função de callback dessa rota é disparada e com isso usamos o findAll que acabamos de programar. Por parâmetro passamos a função callback que será executada quando a consulta terminar, exibindo um erro ou renderizando a view index com os docs como model.

Passo 4: Editando a view de listagem

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:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <ul>
      <% docs.forEach(function(customer){ %>
        <li>
            <%= customer.nome %>
        </li>
      <% }) %>
    </ul>
  </body>
</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. Salve o arquivo e reinicie o servidor Node.js. Ainda se lembra de como fazer isso? Abra o prompt de comando, derrube o processo atual (se houver) com Ctrl+C e depois:

```
npm start
```

Agora abra seu navegador, acesse http://localhost:3000/userlist e maravilhe-se com o resultado.

Lista de Clientes

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 5 – 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.

Passo 1: Criando sua view de cadastro

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:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <form action="/new" method="POST">
      <p>Nome:<input type="text" name="nome"/></p>
      <p>Idade:<input type="number" name="idade" /></p>
      <input type="submit" value="Salvar" />
    </form>
  </body>
</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:

```
router.get('/new', function(req, res, next) {
  res.render('new', { title: 'Novo Cadastro' });
});
```

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

Novo Cadastro

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!.

Passo 2: Codificando para cadastrar clientes

Primeiro, vamos alterar nosso db.js para incluir uma nova função, desta vez para inserir clientes usando a conexão global e, novamente, executando um callback ao seu término:

```
function insert(customer, callback){
    global.conn.collection("customers").insert(customer, callback);
}
```

Não esqueça de adicionar essa nova função no module.exports no final do arquivo:

```
module.exports = { findAll, insert }
```

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:

```
router.post('/new', function(req, res) {
  var nome = req.body.nome;
  var idade = parseInt(req.body.idade);
  global.db.insertOne({nome, idade}, (err, result) => {
          if(err) { return console.log(err); }
          res.redirect('/');
      })
})
```

Obviamente no mundo real você irá querer colocar validações, tratamento de erros e tudo mais. 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.

Na função de callback exigida pelo insert colocamos um código que imprime o erro se for o caso ou 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:

```
<hr />
<a href="/new">Cadastrar novo cliente</a>
```

Resultado:

Cadastro funcionando

#6 – 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.

Passo 1: Arrumando a tela de listagem

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:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <ul>
      <% docs.forEach(function(customer){ %>
        <li>
          <a href="/edit/<%= customer._id %>">
            <%= customer.nome %>
          </a>
        </li>
      <% }) %>
    </ul>
    <hr />
    <a href="/new">Cadastrar novo cliente</a>
  </body>
</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:

Listagem com Edição

Passo 2: Carregando os dados antigos

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

```
var ObjectId = require("mongodb").ObjectId;
function findOne(id, callback){
    global.conn.collection("customers").find(new ObjectId(id)).toArray(callback);
```

```
}
```

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:

```
module.exports = { findAll, insert, findOne }
```

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:

```
router.get('/edit/:id', function(req, res, next) {
  var id = req.params.id;
  global.db.findOne(id, (e, docs) => {
      if(e) { return console.log(e); }
      res.render('new', { title: 'Edição de Cliente', doc: docs[0], action: '/edit/' +
    });
})
```

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'.

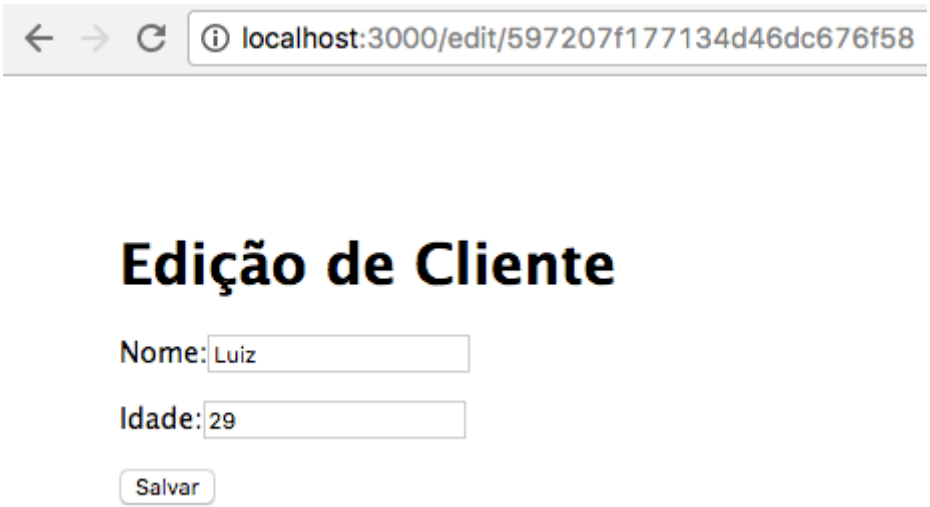
Mas antes de editar a view new.ejs, vamos editar a rota GET em /new para incluir no model dela o cliente e a action, mantendo a uniformidade entre as respostas:

```
router.get('/new', function(req, res, next) {
  res.render('new', { title: 'Novo Cadastro', doc: {"nome":"","idade":""}, action: '/r
});
```

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:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <form action="<%= action %>" method="POST">
      <p>Nome:<input type="text" name="nome" value="<%= doc.nome %>" /></p>
      <p>Idade:<input type="number" name="idade" value="<%= doc.idade %>" /></p>
      <input type="submit" value="Salvar" />
    </form>
  </body>
</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 quase funcionando

Passo 3: Codificando o update

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

```
1 function update(id, customer, callback){
2     global.conn.collection("customers").updateOne({_id: new ObjectId(id)}, {$set: cust
3 }
4
5 module.exports = { findAll, insert, findOne, update }
```

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 inclui 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', function(req, res) {
2   var id = req.params.id;
3   var nome = req.body.nome;
4   var idade = parseInt(req.body.idade);
5   global.db.update(id, {nome, idade}, (e, result) => {
6         if(e) { return console.log(e); }
7         res.redirect('/');
8     });
9 });
```

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!

#7 – Excluindo clientes

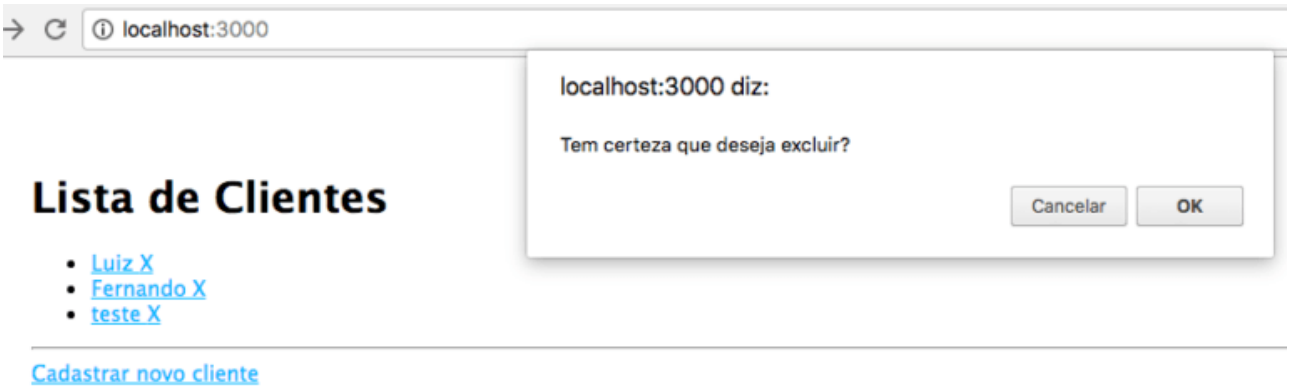
Agora em nossa última parte do tutorial, faremos o D do CRUD, D de Delete!

Passo 1: Editando a listagem

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:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <ul>
      <% docs.forEach(function(customer){ %>
        <li>
          <a href="/edit/<%= customer._id %>">
            <%= customer.nome %>
          </a>
          <a href="/delete/<%= customer._id %>"
            onclick="return confirm('Tem certeza que deseja excluir?');">
            X
          </a>
        </li>
      <% }) %>
    </ul>
    <hr />
    <a href="/new">Cadastrar novo cliente</a>
  </body>
</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.



Exclusão funcionando

Passo 2: Codificando a exclusão

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

```
function deleteOne(id, callback){
    global.conn.collection("customers").deleteOne({_id: new ObjectId(id)}, callback);
}

module.exports = { findAll, insert, findOne, update, deleteOne }
```

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:

```
router.get('/delete/:id', function(req, res) {
  var id = req.params.id;
  global.db.deleteOne(id, (e, r) => {
        if(e) { return console.log(e); }
        res.redirect('/');
      });
});
```

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 e as videoaulas estão neste curso.

Curtiu o post? Então clica no banner abaixo e dá uma conferida no meu livro sobre programação web com Node.js!

18/07/2020 Luiz Duarte Desenvolvimento, Node.js mongodb, nodejs

192.168.0.54.
consigo acessar normalmente da minha máquina windows pelor robomando ou pelo console do mandodb.
Mas quando tento em node, ele dá erro.
O meu outro problema, é que me dá o seguinte erro:
C:\xampp\htdocs\mongo\luiztools\workshoptdc\node_modules\express\lib\router\i
throw new TypeError('Router.use() requires a middleware function but got a ' +
gettype(fn))
^

TypeError: Router.use() requires a middleware function but got a Object
at Function.use
(C:\xampp\htdocs\mongo\luiztools\workshoptdc\node_modules\express\lib\router\
at Function.<anonymous>
(C:\xampp\htdocs\mongo\luiztools\workshoptdc\node_modules\express\lib\applic

Então, se voce pudesse me passar um modelo de todo o diretório compactado, poderia testar. Muito obrigado

^ | v • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Kornel ricardo • 6 meses atrás

No final do tutorial tem um formulário que você deixa o e-mail e recebe o zip.

^ | v • Responder • Compartilhar ›

Marcelo Macrino dos Santos • 6 meses atrás

Professor, Bom dia. Primeiramente Gostaria de agradecer todas as explicações sobre esta poderosa ferramenta.

Muito Grato a Deus por colocar no meu caminho pessoas compartilhando grandes ensinamentos.

Sou professor de colégio técnico e recentemente adquiri o seu livro no Kindle Programação Web com Node.js: Completo do Front-End ao Back_end pela Amazon para seguir melhor suas explicações.

Na posição de leitura 5568 (Assunto: Exclusão de registros) existe um erro :

global.db.deleteOne(id, (e, r) => {
if(e) { return console.log(e) }
res.redirect('/?delete=true')
})

o id da função global.db.delete deve ser _id.

Obrigado mais uma vez. Sucesso.

^ | v • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Marcelo Macrino dos Santos
• 6 meses atrás

Bom dia Marcelo, tudo bem? Fico feliz que esteja gostando do livro. O ponto que você citou ali, não é um erro, pois o deleteOne espera um _id como primeiro parâmetro (chave primária do documento no MongoDB), mas a variável passada não precisa se chamar literalmente _id, desde que contenha um ObjectId dentro.

^ | v • Responder • Compartilhar ›

Roberto Chaves • 7 meses atrás

Muito bom !!! eu comprei seu livro na Amazon, conteúdo de qualidade, pra ser perfeito faltou apenas falar de webSockets, socket.io :) mas parabens;

^ | v • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Roberto Chaves • 7 meses atrás

Fico feliz que tenha gostado do livro. Realmente, websockets é uma coisa que eu quero aprender mais para poder compartilhar com meus leitores. Aqui no blog mesmo, tem apenas um tutorial.

^ | v • Responder • Compartilhar ›

JH13 • 8 meses atrás

Sempre tem que fazer - -dbpath na pasta data, ou apenas uma vez?

^ | v • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ JH13 • 8 meses atrás

Sempre que subir o seu servidor de banco você tem de informar onde estão os dados dele. Se você instalar ele como um windows service, não precisa, pois vai ficar sempre rodando com a mesma configuração.

^ | v • Responder • Compartilhar ›

Wescley Serrão - Sundae • 2 anos atrás

Luiz, Meus Parabéns pelo seu post! Muito e bem claro!

Achei um erro no update: o nome da função db... esta com o nome do método desenvolvido, estão invertidos:
Errado:
function --> update(id, customer, callback){
global.conn.collection("customers").--> updateOne({_id:new ObjectId(id)}, customer, callback);
}

Correto:
function --> updateOne (id, customer, callback){
global.conn.collection("customers").--> update({_id:new ObjectId(id)}, customer, callback);
}

• Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Wescley Serrão - Sundae • 2 anos atrás

updateOne é uma function do MongoDB:
https://docs.mongodb.com/ma...

• Responder • Compartilhar ›

Kaue Jorge • 2 anos atrás

Boa tarde, primeiramente, parabéns pelo trabalho, eu estou com um erro na hora de recuperar os dados para alteração:

Argument passed in must be a single String of 12 bytes or a string of 24 hex characters
Error: Argument passed in must be a single String of 12 bytes or a string of 24 hex characters
at new ObjectID
(C:\Projeto\project\node_modules\bson\lib\bson\objectid.js:57:11)
at Object.findOne (C:\Projeto\project\db.js:17:50)
at C:\Projeto\project\routes\index.js:27:13
at Layer.handle [as handle_request]
(C:\Projeto\project\node_modules\express\lib\router\layer.js:95:5)
at next (C:\Projeto\project\node_modules\express\lib\router\route.js:137:13)
at Route.dispatch
(C:\Projeto\project\node_modules\express\lib\router\route.js:112:3)
at Layer.handle [as handle_request]
(C:\Projeto\project\node_modules\express\lib\router\layer.js:95:5)
at C:\Projeto\project\node_modules\express\lib\router\index.js:281:22
at param (C:\Projeto\project\node_modules\express\lib\router\index.js:354:14)
at param (C:\Projeto\project\node_modules\express\lib\router\index.js:365:14)

pode me dar uma luz??

• Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Kaue Jorge • 2 anos atrás

Boa noite Kaue. Esse erro é na hora de converter uma String para ObjectId. Você não está passando uma String com o ID correto no findOne.

• Responder • Compartilhar ›

Kaue Jorge ➚ Luiz Fernando Jr • 2 anos atrás

Era exatamente isso, acabei achando o erro, tinha uma "aspas simples", no lugar errado no meu findOne.

Obrigado pelo retorno.

1 • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Kaue Jorge

• 2 anos atrás

Fico feliz em ter ajudado!

• Responder • Compartilhar ›

Molinex • 2 anos atrás

Ótimo tutorial, estou começando com node, e mongo, e essas dicas vão ajudar...
Depois desse só me restou uma duvida. Se eu precisar criar mais do que uma 'tabela' (Sei que não são tabelas, já que é uma base não relacional, mas deve ser mais facil de me explicar), por exemplo, pessoas e cahorros, em uma mesma aplicação, como eu faria?
Tem algo que você possa indicar pra me ajudar?
Desde já agradeço Luiz, e parabéns pelo trabalho

• Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Molinex • 2 anos atrás

Bom dia, fico feliz de ter ajudado. Para criar outra coleção basta salvar um documento em outra coleção, na mesma base, da mesma forma que fiz no início do tutorial. Para se aprofundar recomendo o meu livro de MongoDB que está em promoção na Amazon hoje (MongoDB para Iniciantes) e meu curso de Node.js e MongoDB que você encontra à venda aqui no blog.

1 • Responder • Compartilhar ›

Rayelle Vera Cruz • 2 anos atrás

Parabéns e muito obrigada, Luiz!!!
Tava tentando diversos outros tutoriais, mas consegui entender muito bem o seu!!!

• Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Rayelle Vera Cruz • 2 anos atrás

Fico feliz de ter ajudado!

• Responder • Compartilhar ›

Ricardo Reis • 3 anos atrás

Luiz,

No db.js, porque no tutorial vc usa: .then(conn => global.conn = conn.db("workshoptdc"))
Mas no arquivo fonte você usou: .then(conn => global.conn = conn)

E também no tutorial você usou:
global.conn.collection("customers").updateOne({_id:new ObjectId(id)}, customer, callback);
Mas no seu arquivo fonte está:
global.conn.collection("customers").updateOne(new ObjectId(id), customer, callback);

callback);

Ou eu estou louco e fazendo confusão?

abraço

• Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Ricardo Reis • 3 anos atrás • edited

Diferença de versão do driver do MongoDB. Nas versões atuais algumas coisas mudaram, logo, dê preferência ao que está no post, pois consigo atualizar ele com mais facilidade.

Se tu baixar os fontes e rodar um NPM INSTALL, vai instalar a versão antiga do driver e deve rodar tudo normalmente. Agora se criar um projeto do zero, tem de usar os fontes do post.

1 • Responder • Compartilhar ›

Ricardo Reis ➚ Luiz Fernando Jr • 3 anos atrás

Certo Luiz, obrigado pela resposta.
Mas estou com um erro quando tento fazer o Update, poderia me ajudar?

ERRO:

ver mais

• Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Ricardo Reis • 3 anos atrás

O updateOne exige que você use operações atômicas ao invés de substituir o documento inteiro, algo nesse modelo {$set: { nome: novoNome, idade: novaIdade }} ao invés de passar o customer inteiro.

1 • Responder • Compartilhar ›

Ricardo Reis ➚ Luiz Fernando Jr • 3 anos atrás

Luiz, muito obrigado.

Eu usei o código abaixo e deu certo.

function update(id, customer, callback){
global.conn.collection("customers").updateOne({_id:new ObjectId(id)}, {$set:{nome:customer.nome, idade:customer.idade}}, callback);
}

10 ^ | v • Responder • Compartilhar ›

daniel ➚ Ricardo Reis • 2 anos atrás

aeee!!!! comentario salvador! da uma tristeza quando chega no finzinho do tutorial e da um baita erro como esse! hhahaha

2 ^ | v • Responder • Compartilhar ›

Ricardo Reis • 3 anos atrás

No tutorial está escrito para acessar " http://localhost:3000/userlist " mas na verdade não precisa do "userlist", basta acessar "http://localhost:3000/"

^ | v • Responder • Compartilhar ›

Ricardo Reis • 3 anos atrás

No #4, passo 2, onde editamos o arquivo "www" é importante dizer que o código " global.db = require('../db'); " deve ser inserido uma linha acima do código " var app = require('../app'); ".
Eu havia inserido logo no início do arquivo, ou seja, na primeira linha, e então deu erro.

^ | v • Responder • Compartilhar ›

Leandro Missel • 3 anos atrás

Grande Luiz, valeu por ajudar a comunidade.

Cara, tá rolando um erro com a linha "var id = req.params.id;"

Sabe se algo mudou?

Valeu, abraço!

Cannot read property 'id' of undefined

TypeError: Cannot read property 'id' of undefined
at /var/html/www/testeMongoDB/wekshoptdc/routes/index.js:27:22
at Layer.handle [as handle_request]
...

^ | v • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Leandro Missel • 3 anos atrás

No router.get a rota deve ser edit/:id. Pelo visto deve ter alguma diferença no seu código em relação ao meu. Já baixou o zip dos fontes para dar uma olhada?

^ | v • Responder • Compartilhar ›

Leandro Missel ➚ Luiz Fernando Jr • 3 anos atrás • edited

Vou baixar meu velho, mas no router.get está certo...

UPDATE: Baixei o teu zip e copiei teu trecho de código. Mesmo erro... me perdi heheh

^ | v • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Leandro Missel • 3 anos atrás

Vamos lá: o erro está reclamando que o req.params está undefined, ou seja, não estão vindo parâmetros na sua requisição. Provavelmente o seu GET não está trazendo o id na URL (pode ser culpa do seu HTML que gera este GET, que pode estar errado). Experimenta colocar um console.log(req.url) na primeira linha dentro da sua rota para ver o que vai aparecer no console ou revisa a parte que está chamando essa rota.

^ | v • Responder • Compartilhar ›

Manoel Braz Maciel • 3 anos atrás

Boa tarde, Luiz ... Sou muito grato pelo seu trabalho. Tem me ajudado muito. Estou fazendo o meu TCC sobre a stack MEAN. E os seus artigos tem me sanado muitas dúvidas. Muito obrigado. Felicidade!

^ | v • Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Manoel Braz Maciel • 3 anos atrás

Boa tarde Manoel,

M, E e N você encontra aqui no blog bastante material. Já o 'A' vou ficar te devendo, hehehehe
Fico feliz em estar ajudando.

^ | v • Responder • Compartilhar ›

Manoel Braz Maciel ➚ Luiz Fernando Jr • 3 anos atrás

Realmente, ainda não cheguei no A ... Mas, já ajudou bastante. Muito obrigado. Felicidade!

1 ^ | v • Responder • Compartilhar ›

erick couto • 3 anos atrás

Sou leigo em node, uma duvida, li o tutorial de criação de API tb, queria saber se os dados que foram cadastrado nesse tutorial aqui pode ser chamado de API tb e posso consumir esses dados?

Ou esse tuto n tem nada a ver com o tutorial de API ? Obrigado.

Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ erick couto • 3 anos atrás

Esse tutorial é uma aplicação web CRUD, com interface gráfica para pessoas poderem usar. Além disso esse tutorial usa o driver nativo do MongoDB.

APIs não possuem interface gráfica e o outro tutorial usa Mongoose ao invés do driver nativo. Você cria uma API quando quer expor o seu banco de dados (ou parte dele) para outros sistemas usarem.

Resumindo: use esse tutorial para criar aplicações web. Use o outro tutorial para criar serviços web.

1 Responder • Compartilhar ›

Luiz Fernando Jr LuizTools ➚ Luiz Fernando Jr • 3 anos atrás

Embora você consiga adaptar uma aplicação para virar um serviço e vice-versa.

1 Responder • Compartilhar ›

Joao Felipe Barros Neto • 3 anos atrás

Estou com esse erro, vc pode me ajudar?
{ [MongoError: wrong type for 'q' field, expected object, found q:

Orgulhosamente mantido com WordPress