

Como criar uma web-app usando **Node.js + MongoDB**



Sumário

1. Introdução	3
2. Preparando o ambiente	7
3. Conhecendo o NODE	11
4. Conhecendo o banco de dados	25
5. Conectando no banco de dados	32
6. Persistindo os dados	41
7. Conclusão	51



Introdução



*Walking on water and developing software
from a specification are easy if both are frozen.*

- Edward Berard

Hoje eu vou contar uma história de como um coala muito esperto aprendeu a criar aplicações usando Node.js e MongoDB. Essa história se passa em meados de 2016, antes do Umblerito ajudar a criar nossas plataformas de Node.js e MongoDB na Umbler e estava apenas aprendendo a trabalhar com estas tecnologias.

Se você é um programador iniciante nestas duas tecnologias, tenho certeza que você vai se identificar. Além disso, ensinar um coala de dois anos a programar é algo deveras divertido também.

Se você nunca programou antes, nem na faculdade ou no técnico, talvez este não seja o melhor livro para você aprender. Considero aqui que você já programa para web, em qualquer tecnologia. Ter assistido os vídeos de Node.js e MongoDB que gravei para a Umbler ajuda também (estão disponíveis em nosso canal no YouTube, assiste lá!).

Espero que você goste da leitura e que o conteúdo seja útil.

Um abraço e sucesso.



Luiz Fernando
Dev Evangelist
Umbler

Eu estava passeando pela Umbler outro dia quando encontrei o Umblerto cabisbaixo. Sim, nosso querido coala estava atirado em um puff na sala de jogos sem muito ânimo, com uma cara de partir o coração. Ele coçava sua barriga que, confesso, está um pouco mais avantajada do que deveria, e olhava para o horizonte, com um marasmo que dava sono só de olhar. Obviamente, fiquei intrigado, uma vez que nosso mascote geralmente é muito inquieto e está sempre fazendo mil e uma coisas pelos corredores da empresa. Tratei de pegar um biscoito Passatempo (o favorito do Umblerto) e sentei ao lado dele para conversar.

Conversa vai, conversa vem, ele me explicou que estava chateado. Ao que parece todo mundo sabia criar aplicações Node.js com MongoDB, exceto ele. Claro que isso era um exagero. Ele se sentia atrasado em relação aos outros programadores pois ainda criava todas suas aplicações com C# ou Java e usando bancos SQL tradicionais. Não sei de onde ele tirou que isso era um problema, mas reconheci nesse desabafo uma preocupação genuína e que corriqueiramente percebo em meus alunos da faculdade também: a preocupação de não estar acompanhando a velocidade das mudanças tecnológicas.

Como bom amigo que sou, me ergui do puff com um pulo e estendi a mão pra ele.

- Bora criarmos algo em Node.js com MongoDB?

Ele me olhou incrédulo com seus pequenos olhos negros. Afinal, era um fim de tarde de sexta-feira e todos já estavam se preparando para ir embora após mais uma sessão de ShareIT, uma espécie de TEDx que fazemos internamente na Umbler semanalmente, evento no qual um de nós ensina algo novo aos demais.

- Não preciso mais do que algumas poucas horas para eu te dar um overview e, juntos, criarmos uma aplicação completa em Node e Mongo. Topa o desafio?

Desafiar o Umblérito é muito engraçado. Ele é pior que o Marty McFly quando chamado de franguinho. Como eu esperava, ele se ergueu meio desajeitado, tirou um notebook do seu marsúpio (aquela bolsa na barriga, sabe?) e caminhou em direção a uma pequena sala de reunião que estava desocupada. Ao notar que eu não o segui, ele se virou e questionou:

- Você vem *homo sapiens*? Ou está com medo que um coala programe melhor que você?

Eu balancei a cabeça com um sorriso e pensando que empresa maluca na qual eu fui me meter.

Vamos, seu abusado!





Preparando o ambiente



*A language that doesn't affect the way you
think about programming is not worth knowing.*

- Alan Perlis

Vai rodar na minha máquina?

Esta foi a primeira dúvida do Umblérito. Ele tinha um computador bem antigo, 2008 ou 2009 eu acho.

- Claro que vai!

Eu falava isso com conhecimento de causa. Meu próprio notebook à época era um modelo 2009, um Core 2 Duo que ao longo dos anos eu fui fazendo alguns upgrades como 8GB RAM e 256GB SSD. Mas já vi Node rodar liso em configurações bem inferiores a minha. É realmente uma tecnologia muito democrática considerando que é gratuita, multi plataforma, open-source e muito leve.

- E como se instala?

Disse o rechonchudo mamífero estralando seus dedos e pescoço como se estivesse se alongando para algo esportivo.

- Acessa aí: nodejs.org e procura por um grande e verde botão de download. Para nossa aplicação de teste tanto faz se você baixar a versão mais recente ou a LTS, que em teoria é a mais estável.

- Hm. Já baixou e instalei usando NNF. É só isso mesmo?

O Umblérito começou sua carreira em TI no suporte técnico, como a maioria dos programadores. Ele ainda carrega algumas gírias dessa época como NNF para Next-Next-Finish, bem coisa de sysadmin mesmo.

- Sim, mas se quiser, você pode baixar o Visual Studio Code também, ele é um editor de texto com alguns recursos bem legais para programadores Node.js. Você sabia que ele foi criado usando Node.js?

- Sério? Mas isso não é da Microsoft? Eles não apóiam apenas .NET?

Enquanto falava comigo, ele colocava o nome da ferramenta no Google seguido da palavra 'download' para buscar pelo link de download. Como todo bom programador faz.

- Sim é da Microsoft, mas mesmo a Microsoft tem usado Node.js em vários projetos, além de investir pesado na Node.js Foundation, organização sem fins-lucrativos que mantém a tecnologia.
- Será que já está funcionando?
- É fácil de saber, abre o seu terminal de linha de comando e digita aí “node -v”, deve aparecer a versão do Node.js instalado na sua máquina.
- Realmente, apareceu. Mas e se eu quiser testar se o código em Node está funcionando?
- Você não testa código em Node, ele é uma plataforma para aplicações, não uma linguagem de programação!
- Ah é, verdade. Já sabia disso.
- Sei...De qualquer forma, digite apenas “node” no terminal que o interpretador REPL vai se abrir e você pode escrever JavaScript diretamente no console
- Aí sim hein, de JavaScript eu entendo!

O Umblerto tem larga experiência em desenvolvimento web, desde o início dos anos 2000. Isso o habilita a navegar facilmente entre as diferentes tecnologias web que surgiram desde então e principalmente a linguagem mais popular neste ambiente que é a JavaScript de Brendan Eich.

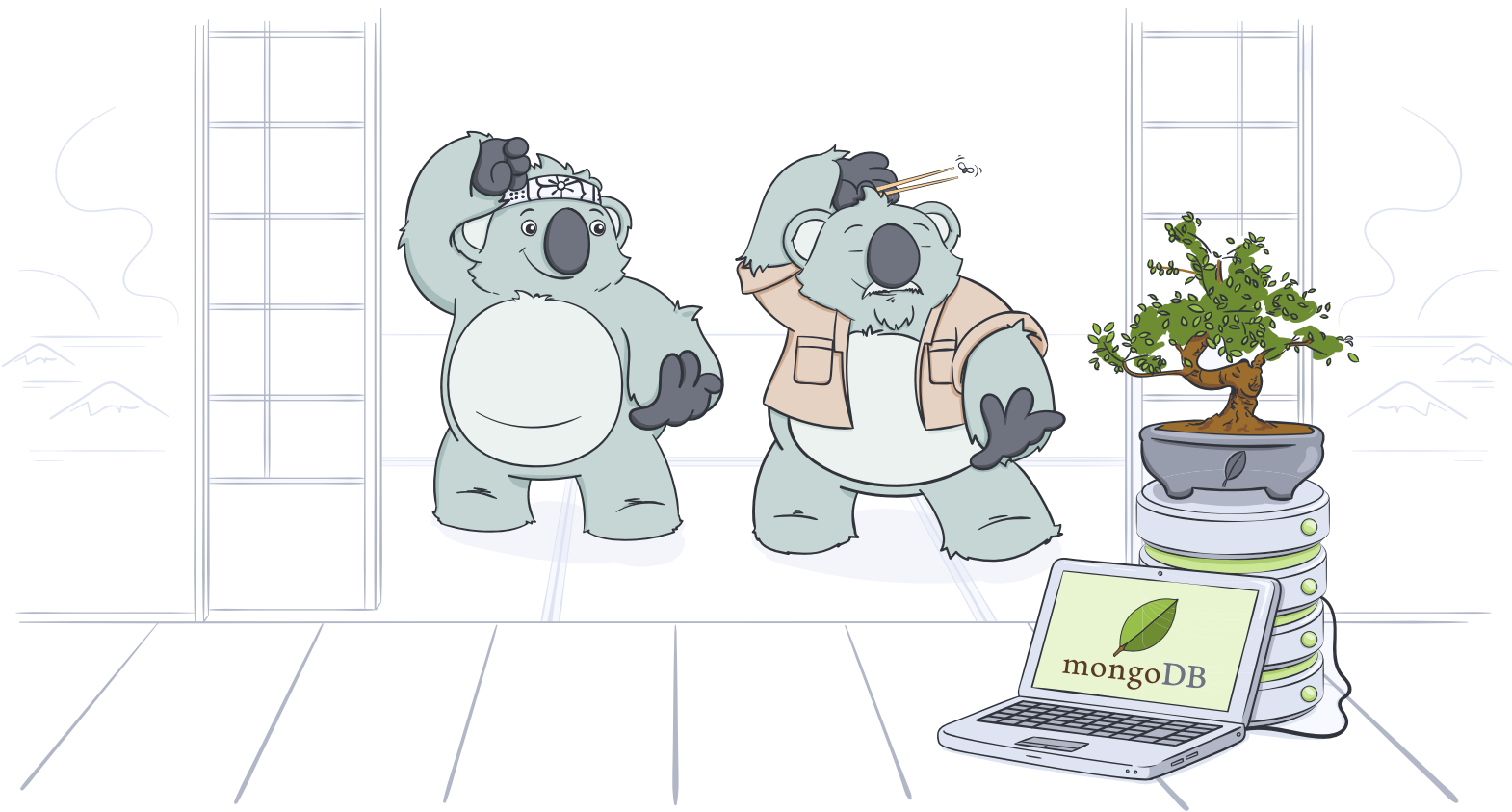
- Bacana - ele disse - é só escrever código JavaScript e dar ENTER que o console vai interpretando os comandos linha por linha. E o Mongo?

Sabe quando você tem certeza que alguém nasceu para ser programador? É quando essa pessoa se diverte tanto programando, quanto jogando um bom videogame, eu percebo isso de longe, graças aos meus anos como professor. O Umblerto é um exemplo disso, embora ele mesmo goste de se chamar de ‘mamífero DevOps’ por gostar de infra também.

- O Mongo você baixa em MongoDB.org, na seção Download, opção Community Server.
- Ok, está baixando. O Mongo é gratuito também, certo?
- Sim, a versão community é gratuita e, assim como o Node, é multiplataforma, open-source, etc, etc.
- Certo, já baixou. Como instalo?
- Não precisa.

- S3rio?
- Sim, o MongoDB 3 um conjunto de utilit3rios de linha de comando escritos em C++, n3o precisa de instalador, 3 s3o extrair os arquivos em uma pasta e pronto.
- Legal!
- Sim, e est3 apenas come3ando.

Eu poderia ajudar ele a testar o MongoDB agora, mas vou deixar para depois, quando fizer mais sentido. Temos muito o que explorar no Node antes de precisar de persist3ncia.





Conhecendo o NODE



Some of the best programming is done on paper, really. Putting it into the computer is just a minor detail.

- Max Kanat-Alexander

Não havia mais qualquer sinal de tédio no semblante do Umblérito, e olha que passaram-se apenas alguns minutos desde que eu o convenci a experimentar Node.js, no final da tarde de uma sexta-feira qualquer. Uma das vantagens da plataforma era o seu setup super rápido e prático.

- Qual o próximo passo? Por onde começamos? O que vamos programar? Q...
- Calma!

Curioso como sempre, o Umblérito queria começar logo a criar aplicações escaláveis, performáticas, leves e tudo mais que já tinha ouvido falar das qualidades do Node.js. No entanto, ninguém consegue aprender a correr antes de aprender a caminhar, certo?

- Vamos começar com algo simples e depois vamos incrementando, ok?
- Yes, sir!

Já estou acostumado com as gracinhas do nosso mamífero australiano, mas quem olha de fora um coala batendo continência não tem como segurar o riso.

- Você está com o terminal ainda aberto Umblérito? Crie uma pasta “nodeprojects” no seu computador e dentro dela uma pasta ...
- Lista de Compras.
- O quê?
- Vamos fazer um sistema de Lista de Compras, por favor?
- Alguma razão específica?
- Sim, eu preciso organizar minha lista de compras do mercado. Eu sempre esqueço de alguma coisa, semana passada foi a granola.
- E você come granola? Achei que essa sua pança era de chocolate hehehe
- Engaçadinho. Isso aqui - batendo na barriga - é conhecimento estocado!
- Além de ser útil para apoiar o note quando está trabalhando deitado, hehehe.
- Podemos continuar? - disse ele desconfiado.
- Claro - eu disse enquanto limpava uma lágrima de riso no canto do olho. Bom, então crie sua pasta “listadecompras” dentro de “nodeprojects”, sem espaços ou letras maiúsculas para facilitar. Navegue via terminal até essa pasta usando “cd”, tipo isso:

```
cd c:\users\umblerito\nodeprojects\listadecompras (Windows)
ou
cd /users/umblerito/nodeprojects/listadecompras (Unix)
```

- Uma vez dentro desta pasta, meu pequeno amigo, vamos instalar uma extensão global do Node.js chamada Express Generator. Ela serve para facilitar o nosso trabalho, um scaffold que gera um projeto Node.js funcional e básico usando um framework web chamado Express.
- Que complicado. Não tem um jeito mais fácil?
- Na verdade é bem fácil, você vai ver, apenas execute os seguintes comandos no console:

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

- O NPM é o gerenciador de pacotes do Node, certo?
- Sim, o Node Package Manager. INSTALL é o comando para instalar um novo pacote e “-g” indica que este pacote será instalado globalmente, para todo ambiente Node.js da sua máquina. Se não usar o “-g” você instalaria o express-generator somente para este projeto, entende?
- Sim, sim. E como se usa o Express Generator na prática?
- Usando o comando “express” na pasta do seu projeto e uma série de parâmetros. Experimente assim:

```
express -e --git
```

- Opa, acho que funcionou. O terminal está dizendo que vários arquivos e pastas foram criados.
- Sim, todo um projeto básico Express foi criado. Além disso, a flag “-e” habilitou o suporte a EJS no seu projeto, um view render baseado em HTML + JS e a flag “--git” criou...
- O repositório git local e o .gitignore!
- Exato. Tá vendo ali no console aquela instrução que diz para rodar um “npm install” na pasta do seu projeto? Faz isso agora.

npm install

- O comando NPM INSTALL quando usando sem especificar o nome de um pacote, procura por um arquivo chamado package.json na raiz do seu projeto Node.

- Abri aqui, é um arquivo JSON com as configurações do projeto, certo?

- Sim, dentre as configurações estão todas dependências deste projeto, os pacotes que ele precisa para rodar com sucesso. O NPM INSTALL lê essas dependências e baixa todas elas para sua máquina, em uma pasta node_modules dentro do seu projeto.

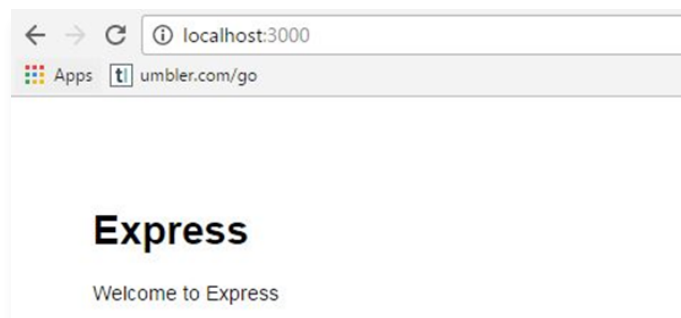
- Aham, criou aqui. E agora?

- Agora rode o comando NPM START na raiz do seu projeto para executá-lo no seu navegador.

npm start

- Funcionou?

- Acho que sim, olha:



- Isso mesmo. Ele subiu um servidor web na porta 3000.

- Então o Node é pra isso, para fazer websites?

- No way! Node é muito mais que isso. Na verdade fazer sites é a última coisa que eu te recomendaria fazer com ele. O foco do Node é I/O não bloqueante, ou seja, chamadas de leitura e escrita assíncronas, como disco, rede, database, etc. Ele é muito bom para fazer APIs, bots, mensageria, IoT e muito mais coisas interessantes. Claro, você pode fazer aplicações web completas com ele também, como vou te mostrar já, já.

- Sim, não esquece que eu quero uma lista de compras, outro dia esqueci de comprar sucrilhos também.
- Mas você não esquece a cabeça porque está colada, né?
- Foco no Node por favor. - Disse Umblerito falsamente emburrado.

Tinha muita coisa que eu deveria explicar ao Umblerito sobre Node. Que Node.js possui uma arquitetura orientada a eventos, ou seja, ele processa requisições e retorna respostas conforme elas são enviadas para o seu runtime. Que ele foi criado com um event loop single thread, mas que delega operações demoradas para um pool de threads em background, consumindo requisições de uma fila e processando em outra.

Isso é diferente do funcionamento tradicional da maioria das linguagens de programação, que trabalham com o conceito de multi-threading, onde, para cada requisição recebida, cria-se uma nova thread para atender à mesma. Isso porque a maioria das linguagens tem comportamento bloqueante na thread em que estão, ou seja, se uma thread faz uma consulta pesada no banco de dados, a thread fica travada até essa consulta terminar.

No entanto, resolvi por uma abordagem mais prática. A teoria ele poderia ler depois em um ebook gratuito que escrevi chamado Node.js para Iniciantes, disponível no Umbler Academy, em nosso site.

- Antes de sair criando nossas próprias aplicações, vamos entender o framework Express. Entre na pasta bin do projeto e depois abra o arquivo www que fica dentro dela. Esse é um arquivo sem extensão que pode ser aberto com qualquer editor de texto.
- Ok, abrindo!
- Dentro do www você deve ver o código JS que inicializa o servidor web do Express e que é chamado quando digitamos o comando 'npm start' no terminal. Ignorando os comentários e blocos de funções, temos:

```

1 var app = require('./app');
2 var debug = require('debug')('workshop:server');
3 var http = require('http');
4
5 var port = normalizePort(process.env.PORT || '3000');
6 app.set('port', port);
7
8 var server = http.createServer(app);
9
10 server.listen(port);
11 server.on('error', onError);
12 server.on('listening', onListening);

```

- Na primeira linha é carregado um módulo local chamado app, que estudaremos na sequência. Depois, um módulo de debug usado para imprimir informações úteis no terminal durante a execução do servidor. Na última linha do primeiro bloco, carregamos o módulo http, elementar para a construção do nosso webserver.

Umblerito olhava atentamente a tudo que eu falava, intercalando olhares para a tela do notebook onde eu apontava alguns elementos. Se ele tinha uma boa qualidade de aluno era a escuta ativa. Se você não sabe o que é, sugiro dar uma pesquisada e desenvolver esta habilidade, é muito útil.

- No bloco seguinte, apenas definimos a porta que vai ser utilizada para escutar requisições. Essa porta pode ser definida em uma variável de ambiente (process.env.PORT) ou caso essa variável seja omitida, será usada a porta 3000.

- E o servidor http é criado usando a função createServer?

- Sim, passando o app por parâmetro e depois definindo que o server escute (listen) a porta predefinida. Os dois últimos comandos definem handlers para os eventos de error e listening, que apenas ajudam na depuração dos comportamentos do servidor.

- E só com isso, com pouquíssimas linhas, é possível criar um webserver em Node.js?

- Sim, esse arquivo www é a estrutura mínima para iniciar uma aplicação web com Node.js e toda a complexidade da aplicação em si cabe ao

módulo app.js gerenciar. Ao ser carregado com o comando require, toda a configuração da aplicação é executada. Abre o arquivo app.js na raiz do seu projeto, para eu poder te explicar ele:

```
1 var express = require('express');
2 var path = require('path');
3 var favicon = require('serve-favicon');
4 var logger = require('morgan');
5 var cookieParser = require('cookie-parser');
6 var bodyParser = require('body-parser');
7
8 var index = require('./routes/index');
9 var users = require('./routes/users');
```

- Tá vendo estas declarações de variáveis no topo? Isto referencia elas a alguns pacotes, dependências, funcionalidades do Node e rotas. Rotas direcionam o tráfego e contém também alguma lógica de programação. Quando criamos o projeto Express, ele criou estes códigos JS pra gente. Na sequência você deve ver:

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

- Este é bem importante. Ele cria uma instância do Express e associa nossa variável app a ele. A próxima seção usa esta variável para configurar coisas do Express.

```

1 // view engine setup
2 app.engine('html', require('ejs').renderFile);
3 app.set('views', __dirname + '/views');
4 app.set('view engine', 'ejs');
5
6 // uncomment after placing your favicon in /public
7 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
8 app.use(logger('dev'));
9 app.use(bodyParser.json());
10 app.use(bodyParser.urlencoded({ extended: false }));
11 app.use(cookieParser());
12 app.use(express.static(path.join(__dirname, 'public')));
13
14 app.use('/', index);
15 app.use('/users', users);

```

- Isto diz ao app onde ele encontra suas views, qual engine usar para renderizar as views (EJS) e chama alguns métodos 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>. Os próximos três blocos são manipuladores de erros para desenvolvimento e produção (além dos 404). Não vamos nos preocupar com eles agora.

- E aquele module.exports no final do arquivo, o que é?

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

- Uma característica importantíssima do Node é que basicamente todos os arquivos .js são módulos e basicamente todos os módulos exportam um objeto que pode ser facilmente chamado em qualquer lugar no código. Nosso objeto app é exportado no módulo acima para que possa ser usado no arquivo www, como vimos anteriormente.

- Show, entendi o www e o app.js, podemos começar a nos divertir?

- Calma, existem duas partes básicas e essenciais que temos de entender do Express para que consigamos programar minimamente usando ele: routes e views ou “rotas e visões”.

- Deixa eu adivinhar, é estilo o padrão MVC?
- Sim, isso aí. Routes são regras para manipulação de requisições HTTP. Você diz que, por exemplo, quando chegar uma requisição no caminho '/teste', o fluxo dessa requisição deve passar pela função 'X'. No app.js, registramos duas rotas:

```
1 // códigos...
2 var index = require('./routes/index');
3 var users = require('./routes/users');
4
5 // mais códigos...
6
7 app.use('/', index);
8 app.use('/users', users);
```

- Carregamos primeiro os módulos que vão lidar com as rotas da nossa aplicação. Cada módulo é um arquivo .js na pasta especificada (routes). Depois, dizemos ao app que para requisições no caminho raiz da aplicação ('/'), o módulo index.js irá tratar. Já para as requisições no caminho '/users', o módulo users.js irá lidar. Ou seja, o app.js apenas repassa as requisições conforme regras básicas, como um...

- Middleware.
- Exato. Agora abre o arquivo routes/index.js para eu te mostrar o que acontece após redirecionarmos requisições na raiz da aplicação para ele.

```
1 var express = require('express');
2 var router = express.Router();
3
4 router.get('/', function(req, res, next) {
5   res.render('index', { title: 'Express' });
6 });
7
8 module.exports = router;
```

- Primeiro está sendo carregado o módulo express e, com ele, o objeto router, que serve para manipular as requisições recebidas por esse módulo. O bloco central de código é o que mais nos interessa.

- Deixa eu adivinhar: o `router.get` especifica que quando o router receber uma requisição GET na raiz do domínio ela será tratada pela função passada como segundo parâmetro.
- Sim, é aqui que a magia acontece.
- E se eu quisesse usar outros verbos HTTP usaria `router.post`, `router.delete`, etc?
- Exato. A função anônima passada como segundo parâmetro do `router.get` será disparada toda vez que chegar um GET na raiz da aplicação.
- Esses são os famosos callbacks do Node...
- Sim, usaremos bastante eles hoje, mas mais para frente sugiro você dar uma olhada em Promises e Async/Await, recursos mais recentes do Node que ajudam a evitar o famoso callback hell.
- Sim, já ouvi falar. E esses três parâmetros: `req`, `res` e `next`?
- Vamos focar nos parâmetros `req` e `res` hoje. O `'req'` é a requisição em si, já o `'res'` é a resposta. Dentro da função de callback do `router.get`, temos o seguinte código:

```
1 res.render('index', { title: 'Express' });
```

- Aqui dizemos que deve ser renderizado na resposta (`res.render`) a view `'index'` com o model, os dados para a view, entre chaves (`{}`).
- Parecido com o MVC, onde o controller liga o model com a view?
- Exato! As views são referenciadas no `res.render` sem a extensão, e todas encontram-se na pasta `views`. Vamos ver elas mais tarde. Já o model é um objeto JSON com informações que você queira enviar para a view utilizar. Nesse exemplo, estamos enviando um título (`title`) para view usar.
- Interessante. Se eu mudar a string `'Express'` para outra coisa e reiniciar minha aplicação vai aparecer outro texto na tela da index?
- Sim. Derruba a sua aplicação no terminal (`Ctrl+C`), execute-a com `'npm start'` e acesse novamente `localhost:3000` para ver o texto alterado.
- Legal, funcionou! Aqui na pasta `views` têm arquivos `.ejs`, que extensão é essa?
- Lembra, lá no início da criação da nossa aplicação Express, usando o `express-generator`, que eu disse para usar a opção `'-e'` no terminal?

"express -e --git"

- Pois é, ela influencia em como as nossas views serão interpretadas e renderizadas nos navegadores. Neste caso, usando -e, nossa aplicação foi configurada com a view-engine EJS (Embedded JavaScript) que permite misturar HTML com JavaScript server-side para criar os layouts.

- Tá, mas porque não usamos o padrão?

- Porque o padrão não usa HTML, usa uma marcação diferente, mais resumida, chamada Pug. Só por isso. Também podemos usar outras opções, como Handlebars, mas vamos simplificar neste primeiro contato, ok?

- Roger that!

- Voltando ao app.js, esse código aqui configura como que o nosso 'view engine' funcionará:

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

- Aqui dissemos que vamos renderizar HTML usando o objeto renderFile do módulo 'ejs'. Depois, dizemos que todas as views ficarão na pasta 'views' da raiz do projeto e por fim dizemos que o motor de renderização (view engine) será o 'ejs'.

- Cada view conterà a sua própria lógica de renderização e será armazenada na pasta views, em arquivos com a extensão .ejs. - Umblerto me cortou rapidamente.

- Exatamente meu amigo coala!

- Já li sobre isso em algum lugar, acho que foi no seu blog.

- Isso! Abre o arquivo /views/index.ejs para entender melhor como essa lógica funciona:

```

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      <p>Welcome to <%= title %></p>
10   </body>
11 </html>

```

- Aquelas tags <% %> são parecidas com as do ASP, JSP e PHP? - perguntou o coala interessado.
- Sim, são server-tags. Elas são processadas pelo Node.js e contém códigos JavaScript dentro. Esses códigos serão acionados quando o backend estiver construindo este arquivo para entregar ao navegador.
- Aquele <%= title %> ali está imprimindo no HTML a variável title do model, certo?
- Sim, definimos o valor desta variável lá no routes/index.js, dentro do router.get:

```

1  res.render('index', { title: 'Express' });

```

- Tudo que você passar como model no res.render, Umblerto, pode ser usado pela view que está sendo renderizada. Para finalizar nosso estudo de Express e para ver se você entendeu direitinho como as routes e views funcionam, proponho um desafio: crie uma nova view que deve ser exibida quando o usuário acessar '/new' no navegador.
- Essa é fácil, primeiro vou criar um arquivo new.ejs dentro de views, já vou fazer com a cara do cadastro de um item na minha lista de compras:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Lista de Compras</title>
5   <link rel='stylesheet' href='/stylesheets/style.css' />
6 </head>
7 <body>
8   <h1><%= title %></h1>
9   <p>Preencha os dados abaixo para salvar um novo item.</p>
10  <form action="/new" method="POST">
11    <p>
12      <label>Nome: <input type="text" name="nome" /></label>
13    </p>
14    <p>
15      <label>Quantidade: <input type="number" name="quantidade" /></
16 label>
17    </p>
18    <p>
19      <a href="/">Cancelar</a> | <input type="submit" value="Salvar" />
20    </p>
21  </form>
22 </body>
23 </html>

```

- Muito bem, e agora? - perguntei a ele, desafiando-o.
- Agora que eu tenho a view criada, vou fazer a rota. Vou criar um arquivo routes/new.js e dentro dele vou criar o código que trata requisições GET:

```

1 var express = require('express');
2 var router = express.Router();
3
4 router.get('/', function(req, res, next) {
5   res.render('new', { title: 'Novo cadastro' });
6 });
7
8 module exports = router;

```

- Muito bem, e agora?
- Agora mando rodar no terminal com 'npm start', certo?
- Errado! Você tem que registrar a sua nova rota no app.js, assim como estão registradas as rotas index e users, que eu expliquei antes, lembra?

```
1 var item = require('./routes/new');  
2 app.use('/new', item);
```

- Agora sim, basta mandar executar sua aplicação com 'npm start' e depois acessar localhost:3000/new no seu navegador.



localhost:3000/new

Novo Cadastro

Preencha os dados abaixo para salvar um novo item.

Nome:

Quantidade:

[Cancelar](#) |

- Demais, Luiz!

- Legal, né? Mas o melhor ainda está por vir, vou te mostrar como fazer esse formulário funcionar de fato, salvando os itens da sua lista de compras no MongoDB.



Conhecendo o banco de dados



*Not all roots are buried down in the ground
some are at the top of the tree.*

- Jinvirle

Nessa altura do campeonato, apenas 15 ou 20 minutos após termos começado a brincar com Node.js no notebook do Umblerto, um universo de possibilidades estava se abrindo à frente do nosso coala felpudo. E a noite estava apenas começando!

- Como faço para colocar o MongoDB para funcionar?
- Isso é bem simples, mas antes eu queria te explicar porque vou te mostrar os dois juntos. O MongoDB é um banco orientado a documentos, ele não é um banco SQL como você está acostumado.
- Tô ligado!
- Enquanto nos bancos SQL você usa tabelas com linhas e colunas, aqui temos documentos JSON com campos e valores.
- Assim como no JavaScript?
- Sim, exatamente como no JavaScript, mas armazenado de maneira binária, o que chamamos de BSON.
- Entendi onde você quer chegar, deve ser muito prático salvar dados no MongoDB pois ele lida com o mesmo formato do JavaScript e consequentemente do Node.js, certo?
- Exatamente. Mas você não pegou toda a vantagem ainda!
- Não?
- Não mesmo. Os bancos tradicionais operam de maneira síncrona para garantir o ACID, com um comportamento bloqueante. Isso funciona bem com sistemas síncronos fazendo paralelismo via multithreading mas...
- Não funciona bem com o event loop single thread do Node.js?
- Exato. Não que não dê para usar, nada disso, mas o comportamento mais dinâmico in-memory do MongoDB com ACID a nível de documento, casa muito bem com a assincronicidade do Node.
- Essa palavra existe?
- Qual palavra?
- Assincronicidade?
- Não faço ideia.
- Entendi. É para parecer mais inteligente... Coisa de professor...
- Vamos focar no Mongo, Umblerto?

Como eu não queria voltar tarde para casa nesse dia e meu peculiar aluno estava muito engraçadinho pro meu gosto, eu peguei o comando do note dele fazendo um revezamento no melhor estilo Pair Programming do XP.

- Para subir o seu servidor de MongoDB é bem simples, vamos abrir uma nova janela do console e navegar até a pasta com os utilitários que extraímos antes, lembra?
- Lembro sim. Para quê servem eles?
- Resumidamente na bin da sua 'instalação' de MongoDB você vai encontrar:
 - mongod: inicializa o servidor de banco de dados;
 - mongo: inicializa o cliente de banco de dados;
 - mongodump: realiza dump do banco (backup binário);
 - mongorestore: restaura dumps do banco (restore binário);
 - Ok, ok, acho que faz mais sentido eu saber dos detalhes depois.
 - Exato. Agora vamos começar usando o mongod. Para subir um servidor de Mongo, você deve navegar até a pasta bin via terminal e executar o mongod, passando o parâmetro dbpath, que é o caminho no qual serão salvos os dados da aplicação. Obviamente, esta pasta já deve ter sido criada previamente:

```
/mongo/bin> mongod --dbpath /mongo/data
```

- Ao executar este comando, nosso servidor está rodando sem autenticação na porta default, que é a 27017, aguardando conexões.
- Legal. Se eu derrubar este terminal o servidor para?
- Exato. Para soluções profissionais de MongoDB, recomenda-se instalar como um serviço, ou já contratar alguma empresa que forneça MongoDB na nuvem.
- Sim, sim. E como eu crio minhas tabelas?
- Não, você não cria tabelas. Aqui você cria coleções de documentos.
- Ok. Mas e se eu não quiser salvar documentos? Afinal, estou fazendo uma lista de compras, eu quero salvar itens do mercado.
- Documentos é modo de falar, Umblerto. Um documento é um arquivo JSON com propriedades e valores, você salva o que quiser dentro dele, mas cada documento é independente dos demais, contendo, no seu caso, todas as informações de um item da sua lista de compras.
- Ah bom. E como eu crio?
- Calma, vamos precisar de outra janela do terminal para isso. Nessa janela eu vou navegar até a pasta bin do MongoDB novamente e mandar apenas um comando bem simples: mongo.

- O utilitário 'mongo' é o client do MongoDB, enquanto 'mongod' é o server.
- O 'd' é de daemon?
- Sim, daemon, um processo que fica executando em background.
- Vi que o 'mongo' se conectou em alguma coisa. Como ele achou o servidor correto?
 - Existem diversos parâmetros no utilitário mongo para especificarmos usuário, senha, host, porta, etc. Se você apenas executar ele 'cru', sem parâmetro algum, ele procura um servidor localhost:27017 sem autenticação, como o que subimos há pouco.
 - Entendi. Agora vamos modelar nossas coleções de documentos?
 - Não precisa, isso aqui não é um banco relacional.
 - Sim, isso eu entendi, mas não precisamos modelar nossos dados antes de sair usando o banco.
 - Não mesmo. Nem mesmo precisamos criar nossa base antes de usarmos ela.
 - Uau. Que disruptivo!
 - É verdade. Não que em um projeto real não devamos pensar a respeito de como vamos armazenar nossos dados, apenas isso não é rígido como nos bancos tradicionais. O MongoDB é o que chamamos de persistência schemaless.
 - Tá, mas e se o meu modelo mudar ao longo do ciclo de vida da minha aplicação? Eu tenho de recriar meu banco?
 - Não. Basta salvar os novos documentos com o novo 'schema' e está tudo resolvido. Talvez você queira rodar algum script para 'remodelar' documentos antigos, caso tenha trocado nomes de propriedades, por exemplo, mas não é mandatório.
 - Que maluquice.
 - Sim, mas exige muita disciplina por parte do desenvolvedor, é muito fácil de fazer besteira. O servidor não tem um schema como base para garantir a consistência dos dados entre os diferentes documentos.
 - E os relacionamentos? Como ficam?
 - Não ficam. Na verdade, até tem um esquema de lookups, mas é uma técnica que deixamos para outro dia. Foque em criar documentos autossuficientes, que não precisem se relacionar com outros documentos. Isso vai gerar repetição de

dados, mas ok, o MongoDB foi projetado assim mesmo.

- Estranho.
- Sim, no início é bem estranho. Eu estranhava muito. Mas depois a gente acostuma. Mas acho que é uma boa trocarmos de lugar agora. Eu volto a ser o navegador e você é o piloto novamente.

Na técnica de Pair Programming da metodologia ágil XP, dois programadores, geralmente de senioridade diferente, se revezam em uma mesma máquina, construindo uma aplicação. O que está digitando no momento é o piloto, e o outro é o navegador, na mesma ideia dos rallys, em que um cuida da rota e do mapa, e o outro do volante.

- No terminal do client mongo, executa o comando 'use listadecompras'

```
use listadecompras
```

- Mas Luiz, eu ainda não criei uma base chamada 'listadecompras'.
- Não tem problema. Sua base será criada automaticamente quando salvar o primeiro item nela.
- Ok. Pronto!
- Agora você tem acesso a uma variável 'db', que é um cursor para a base listadecompras, permitindo que acesse as coleções da mesma. Experimente o comando `db.itens.find()` e dê um ENTER.
- Acho que esse 'itens' seria o nome da coleção de documentos, certo? Mas eu ainda não tenho essa coleção...
- Não tem problema, o MongoDB vai perceber isso e não vai trazer nada.

```
db.itens.find()
```

- A função `find()` é para retornar os documentos de uma coleção.
- Assim como o `SELECT` do SQL?
- Exato, mas com uma sintaxe JavaScript.
- Legal, eu gosto muito de JavaScript.
- Você gosta mais de JavaScript ou mais de batata?
- Eu gosto mais de JavaScript... e mais de batata.

Achei que ele não entenderia a referência, mas fui surpreendido novamente. Da onde esse coala conhece Hermes e Renato?

- Beleza. E se eu quiser cadastrar um item via linha de comando #comofaz?
- Aí você tem de usar o comando INSERT na coleção específica, passando o seu documento JSON por parâmetro.

```
db.itens.insert({nome:"bolo de pacote", quantidade: 1})
```

- Mas MongoDB não tem chave primária também? Aí já é demais, né! Como ficam as consultas depois?

- Calma, seu narigudo ansioso. Quando não especificado, o MongoDB cria automaticamente uma propriedade `_id` com um objeto único e auto-incremental chamado ObjectId. Dá uma olhada executando o FIND novamente:

```
db.itens.find()  
{_id: ObjectId("5aa9c02f377f6d0b464a68cc"), nome: "bolo de pacote",  
  quantidade: 1}
```

- Interessante. E isso não repete mesmo? Não era melhor usar um INT ali?
- O MongoDB foi criado com Big Data em mente. Um INT é muito limitado, estamos falando aí de apenas 2^{32} possibilidades, isso dá o quê, uns 4 bilhões de possibilidades aproximadamente? É muito pouco para Big Data. Mas claro, você pode definir que o seu `_id` vai trabalhar com o estilo de dados que quiser, desde que você controle na sua aplicação a entrada desse dado. O MongoDB por padrão sempre usa ObjectId, que permite zilhões de possibilidades.
- Zilhões? - ele perguntou incrédulo.
- Sim, chutando por baixo. - eu disse confiante.
- E você inventa palavras e números pros seus alunos da faculdade também?
- Sim, o tempo todo.
- E não tem vergonha disso?
- Nem um pouco.
- Aff...
- :)



Conectando no banco de dados



Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

-John Woods

Uma coisa fantástica na dupla dinâmica Node.js e MongoDB é a baixa curva de aprendizado. Em questão de 30 minutos eu e o Umblerto não apenas subimos o ambiente completo para estas duas plataformas, como também já estávamos prestes a programar a conexão entre as duas.

- Vamos focar agora em juntar as duas pontas, Umblerto. Neste momento temos duas telas: uma index e outra de cadastro (a tela new). Essa tela inicial (views/index.ejs) deve ser modificada para se tornar uma tela de listagem. Você já programou ASP, certo?

- Sim, e só Thor pode me julgar.

- Não te bobeia, é igual no PHP, quando você quer imprimir uma lista de elementos vindos do banco na tela, você faz um FOR, certo? Aqui a ideia é a mesma, mas usando JavaScript. Vamos começar com a listagem estática mesmo, depois deixamos ela dinâmica. Primeiro muda o HTML da views/index.ejs, enquanto eu vou ali no frigobar pegar o suco que você deixou lá “dando sopa”.

- Ei!

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Lista de Compras</title>
5     <link rel='stylesheet' href='/stylesheets/style.css' />
6   </head>
7   <body>
8     <h1>Lista de Compras</h1>
9     <p>Não esquecer de comprar os seguintes itens:</p>
10    <table style="width:50%">
11      <thead>
12        <tr style="background-color: #CCC">
13          <td style="width:85%">Nome</td>
14          <td style="width:15%">Qtd.</td>
15        </tr>
16      </thead>
17      <tbody>
18        <tr>
19          <td colspan="2">Nenhum item cadastrado.</td>
20        </tr>
21      </tbody>
```



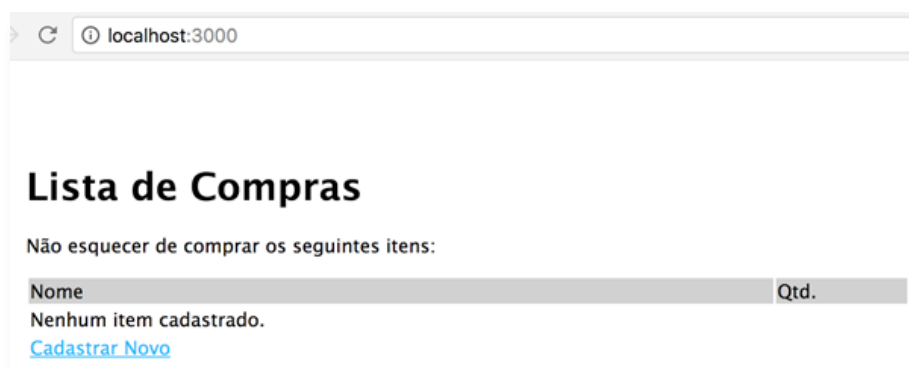
```

22     <tfoot>
23     <tr>
24         <td colspan="3">
25             <a href="/new">Cadastrar Novo</a>
26         </td>
27     </tr>
28 </tfoot>
29 </table>
30 </body>
31 </html>

```

Enquanto ele escrevia eu fui até um dos frigobares que temos próximo dos times.

- Ô Umblérito, não tem suco nenhum aqui! - reclamei alto.
- Sim, eu te falei que preciso de uma lista de compras, eu esqueci de comprar as frutas para espremer.
- Mas tem cerveja no frigobar da empresa. Isso você não esqueceu?
- Eu sou um coala, não um burro. E é para levar para casa, não é para tomar aqui, antes que você pergunte!
- Sei... O que você fez aí? Deixa eu ver, salva tudo e executa com 'npm start', por favor:



- Tá feio pra caramba, hein! Como designer você morreria de fome, hehehe.
- Sinta-se à vontade para fazer melhor, senhor.
- Eu passo, meu foco é backend e mobile.
- Sei... Mas vamos voltar ao Node aqui, que me interessa mais. Como que eu faço o Node conectar no MongoDB?
- Para isso você vai precisar do pacote mongodb do NPM.

- NPM INSTALL MONGODB, certo?
- Exatamente jovem padawan!

npm install mongodb

- Existem diferentes formas de se conectar a bancos de dados usando Node.js, nós vamos usar o driver oficial dos criadores do MongoDB, que se chama apenas mongodb. Para deixar nossa aplicação minimamente organizada não vamos sair por aí escrevendo lógica de acesso à dados em qualquer lugar. Vamos centralizar tudo o que for responsabilidade do MongoDB dentro de um novo módulo chamado db, que nada mais é do que um arquivo db.js na raiz do nosso projeto.

- Saquei.

- Esse arquivo será o responsável pela conexão e manipulação do nosso banco de dados, usando o driver nativo do MongoDB. Me passa o note aqui que eu escrevo este db.js para você:

```
1 var MongoClient = require("mongodb").MongoClient;
2 MongoClient.connect("mongodb://localhost:27017/listadecompras")
3   .then(conn => global.conn = conn.db("listadecompras"))
4   .catch(err => console.log(err))
5
6 module.exports = { }
```

- Explica essa parada aí, que não consegui pegar tudo?

- Na primeira linha, carreguei o módulo mongodb usando o comando require e de tudo que este módulo exporta vamos usar apenas o objeto MongoClient, que armazenamos em uma variável local.

- Explica essa parada aí, que não consegui pegar tudo?

- Na primeira linha, carreguei o módulo mongodb usando o comando require e de tudo que este módulo exporta vamos usar apenas o objeto MongoClient, que armazenamos em uma variável local.

- Sim, essa foi a parte que entendi. Você fez a conexão a partir do MongoClient também, mas e o then/catch ali?

- Com essa variável MongoClient carregada, usamos a função connect passando a connection string. A terceira e a quarta linha podem parecer

muito estranhas, mas são um recurso mais recente de programação JavaScript, chamada Promises. Promises é um jeito mais elegante do que callbacks para lidar com funções assíncronas. A conexão com o banco de dados pode demorar um pouco dependendo de onde ele esteja hospedado, e sabemos que o Node.js não permite bloqueio da thread principal. Por isso, usamos a função 'then' para dizer qual função será executada (callback) após a conexão ser estabelecida com sucesso.

- Hum! Então é quase um “callback Nutella”?
- Sim, basicamente ele reduz o aninhamento de código, facilitando a leitura.
- E aqueles ponteiros ali, estilo o do C?
- Nesse caso eu usei outro conceito mais recente, que são as arrow functions, uma notação especial para funções anônimas nas quais declaramos os parâmetros (entre parênteses se houver mais de um) seguido de um '=>' e depois os comandos da função (entre chaves se houver mais de um). No caso da function connection do MongoClient, ela retorna um objeto de conexão em caso de sucesso, que armazenei globalmente usando essa função aqui, que seleciona o banco:

```
1 | conn => global.conn = conn.db("listadecompras")
```

- E não é gambiarra salvar a conexão em uma variável global?
- Não se usarmos o global com parcimônia. Até onde sei, no caso do MongoDB especificamente, é uma boa prática salvar a conexão para reutilizar nas múltiplas requisições que esta nossa aplicação Node pode receber, ganhando em performance e consumo de memória reduzido.
- Mas, e se a conexão não for estabelecida com sucesso, o que acontece, cai no catch?
- Exato! Nesse caso, usamos a função 'catch' passando outra função de callback, mas essa para o caso de dar erro no connect. Em nosso exemplo apenas mandei imprimir no console a mensagem de erro.
- E o module.exports no final é estilo o que tinha no app.js?
- Sim, a última linha do nosso db.js contém o module.exports, que é a instrução que permite compartilhar objetos com o restante da aplicação. Como já compartilhei globalmente a nossa conexão com o MongoDB

(conn), não há nada para colocar aqui, por enquanto.

- Tá, entendi agora, mas onde chamamos este módulo db.js na nossa aplicação?

- Vai na pasta bin do seu projeto e abre o arquivo www. Adiciona a seguinte linha no início dele:

```
1 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.

- Você não tinha que colocar db.js ao invés de apenas db?

- Não é necessário. Além disso, caso eu estivesse carregando um módulo que estivesse dentro da node_modules, nem mesmo precisaria usar o '../'.

- Certo, e como eu listo na index aquele “bolo de pacote” que a gente cadastrou antes via terminal?

- Para nos organizarmos, vamos criar uma function findAll dentro do nosso db.js, olha só, antes do module.exports do arquivo.

```
1 function findAll(callback){  
2   global.conn.collection("itens").find({}).toArray(callback);  
3 }
```

- 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.

- Isso é meio estranho. É realmente necessário?

- É uma das muitas formas de fazer e a que considero mais simples para começarmos. Se não fizemos assim, não teremos o resultado da consulta quando precisarmos. A consulta aqui é bem direta: usamos a conexão global conn para navegar até a collection de itens 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.

- Não é tão difícil...

- Não mesmo. 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:

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

- Agora, vamos programar a lógica que vai usar esta função. Abra o arquivo routes/index.js e edite a rota padrão GET / para que quando essa página for acessada, ela faça a consulta por todos os itens no banco, da seguinte maneira:

```
1 /* GET home page. */
2 router.get('/', function(req, res) {
3   global.db.findAll((e, docs) => {
4     if(e) { return console.log(e); }
5     res.render('index', { docs });
6   })
7 })
```

- Ok, deixa eu ver se entendi: router.get define a rota que trata essas requisições com o verbo GET, como vimos isso antes. Quando recebemos um GET /, a função de callback dessa rota é disparada e, com isso, usamos o findAll que acabamos de programar no db.js?

- Exato! Não esqueça que, por parâmetro, passamos a função callback que será executada quando a consulta terminar, exibindo um erro (e) ou renderizando a view index com o array de documentos (docs) como model.

- Mas ainda não funcionou, executei aqui com o 'npm start' e ainda não lista nada na minha index...

- Isso ainda não lista os clientes pois não preparamos a view para tal, mas já enviamos os dados para ela. Agora vamos arrumar a nossa view para listar os clientes. Abra a view views/index.ejs e altere a linha da tabela em

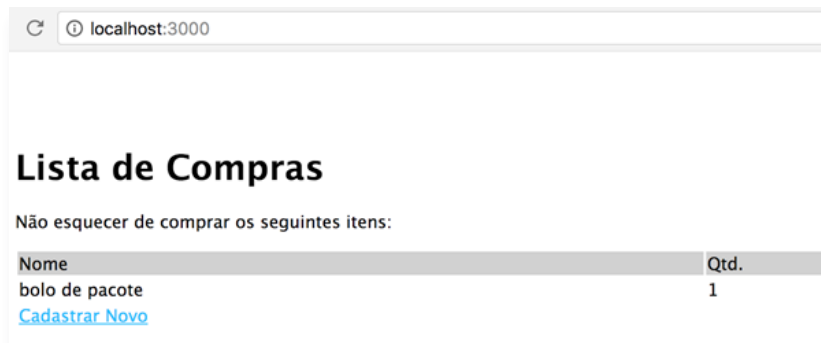
que é exibida uma mensagem de que ‘não há itens cadastrados’ para que essa mensagem somente seja exibida no caso de não haver, de fato, itens no model:

```
1 <% if(!docs || docs.length == 0) { %>
2     <tr>
3         <td colspan="2">Nenhum item cadastrado.</td>
4     </tr>
5 <% } %>
```

E agora, modificando a última linha deste código, junto à chave que fecha o if no bloco anterior, adicione um else para que execute um laço sobre o array de documentos do model e, dessa maneira, construa a nossa tabela dinamicamente:

```
1 <% } else {
2     docs.forEach(function(item){ %>
3         <tr>
4             <td style="width:80%"><%= item.nome %></td>
5             <td style="width:20%"><%= item.quantidade %></td>
6         </tr>
7         <% })
8     }%>
```

- Nossa, bem parecido com ASP clássico mesmo...
- 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 linhas na tabela com seus dados. Isto é o bastante para a listagem funcionar. Vai lá!
- Certo. Salvando o arquivo e reiniciando o servidor Node.js...
- Ainda se lembra de como fazer isso?
- Sim, sim, abro o prompt de comando, derrubo o processo atual com Ctrl+C e depois ‘npm start’ na veia...Wow, Santa Tartaruga!

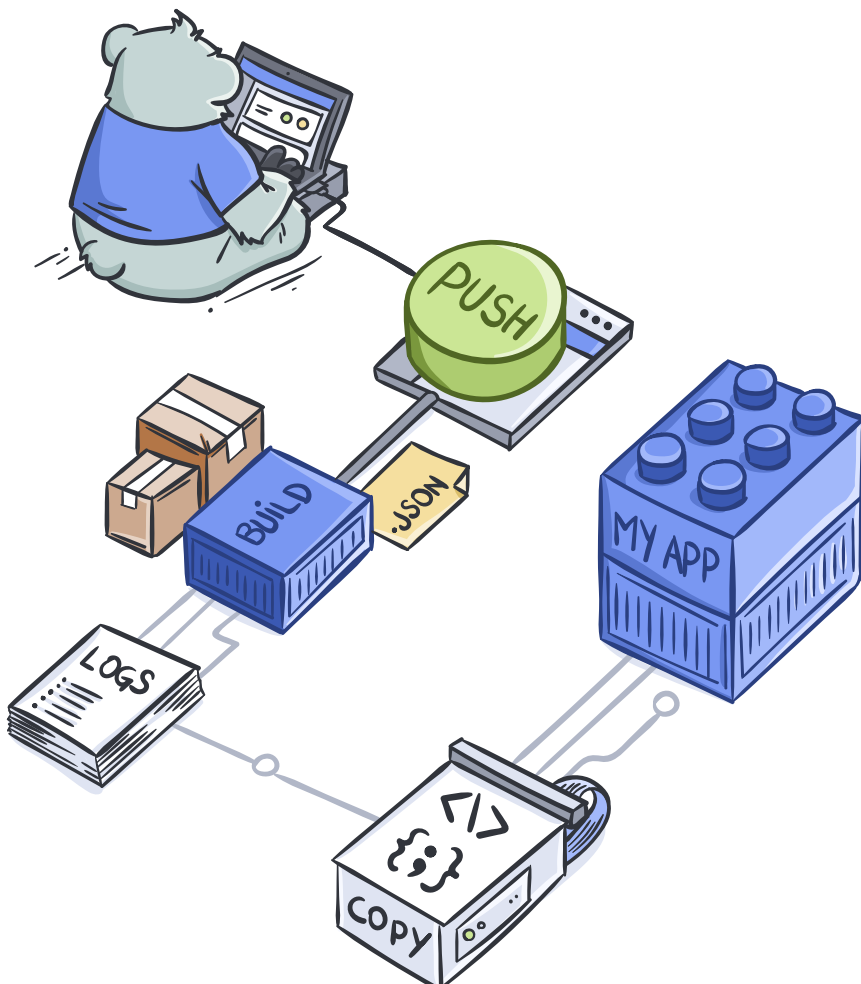


- Legal, né Umblérito? Isso nos mostra que sua conexão com o banco de dados está funcionando como deveria.

- Aham, interessante mesmo. Foi mais simples do que eu imaginava que seria. Posso adicionar mais itens na lista de compras através do console mongo?

- Não, vamos fazer isso através do seu formulário de cadastro. Já vai pensando nos itens que quer cadastrar...

- Pode deixar, minha barriga está me lembrando de alguns que estão em falta lá em casa.





Persistindo os dados



Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.

- Rick Cook

- Sucrilhos.
- Hein?
- Vamos começar cadastrando sucrilhos, Luiz.
- Certo. - eu disse revirando os olhos - Listar dados é moleza e salvar dados no MongoDB não é algo particularmente difícil. Lembra que deixamos uma rota GET /new que renderiza a view new.ejs?
 - Sim, se acessarmos localhost:3000/new ela já funciona.
 - Mas antes de mexer nas rotas novamente, 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:

```
1 function insert(item, callback){
2   global.conn.collection("itens").insert(item, callback);
3 }
```

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

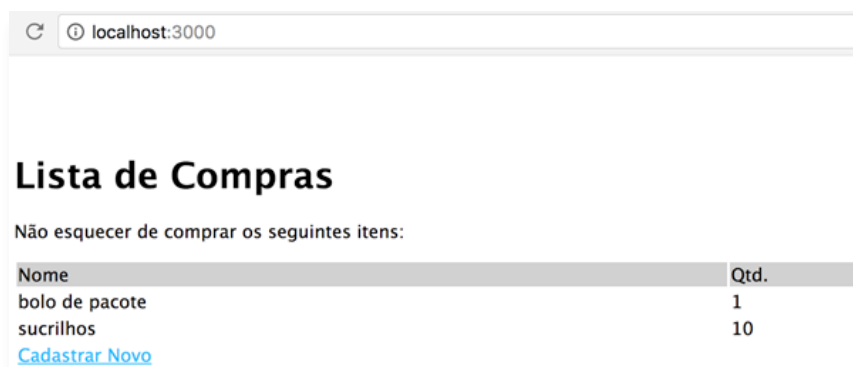
```
1 module.exports = { findAll, insert }
```

- Agora abra o seu routes/new.js e vamos criar uma rota POST / que vai receber o POST do HTML FORM da view new.ejs. Nós chamaremos o objeto global db para salvar os dados no Mongo adicionando o seguinte bloco de código:

```
1 /* POST new page. */
2 router.post('/', function(req, res, next) {
3   const nome = req.body.nome;
4   const quantidade = parseInt(req.body.quantidade);
5   global.db.insert({nome, quantidade}, (err, result) => {
6     if(err) { return console.log(err); }
7     res.redirect('/');
8   })
9 });
```

Nesse router.post eu poderia colocar validações, tratamento de erros e tudo mais que eu quiser fazer antes de salvar no banco, certo?

- Exato. Apenas peguei os dados que foram postados no body da requisição HTTP usando o objeto req (request/requisição). Criei um JSON com essas variáveis e enviei para função insert que criamos agora a pouco.
- E no callback exigido pelo insert você colocou um código que imprime o erro, se for o caso, ou redireciona para a index novamente, para que vejamos a lista atualizada.
- Exatamente, pequeno gafanhoto. E você sabe me dizer porque que usei o parseInt no req.body.quantidade?
- Por que, você não gosta de strings?
- Aff! Não, né! O MongoDB infere os tipos das propriedades dos documentos a partir dos valores fornecidos nas mesmas. Ou seja, se você passar uma propriedade com uma string dentro ele vai entender que 'quantidade' é do tipo string, o que pode interferir mais tarde nas buscas.
- Aaaaaah, entendi! Semelhante a tipagem dinâmica do próprio JavaScript!
- Exatamente! Agora reinicie sua aplicação e tente cadastrar o sucrilhos que já deve funcionar, redirecionando automaticamente para tela inicial depois de salvo.
- Ok. Nome: sucrilhos. Quantidade: 10.



localhost:3000

Lista de Compras

Não esquecer de comprar os seguintes itens:

Nome	Qtd.
bolo de pacote	1
sucrilhos	10

[Cadastrar Novo](#)

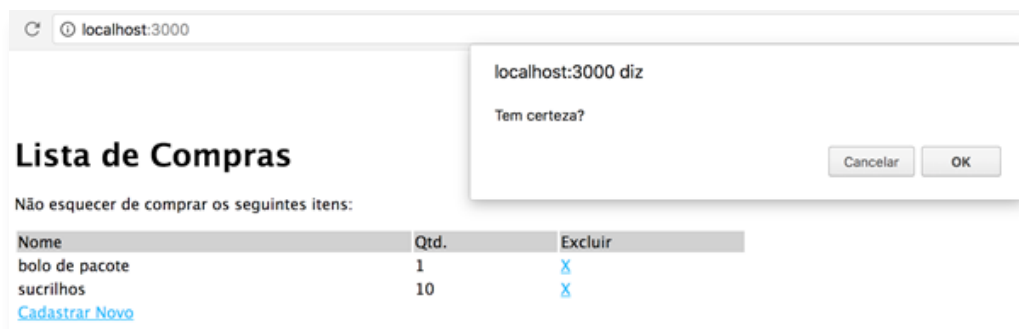
- Peraí, você vai comprar 10 pacotes de sucrilhos, Umblerito?
- Claro que não, é só para fins de teste do sistema.
- Sei...
- Aproveitando o gancho, como que eu faço para programar a exclusão deste item?
- Vamos voltar à index.ejs (tela de listagem) e adicionar um link específico para exclusão, em uma terceira coluna, ficando com uma table assim:

```

1 <table style="width:50%">
2   <thead>
3     <tr style="background-color: #CCC">
4       <td>Nome</td>
5       <td style="width:15%">Qtd.</td>
6       <td>Excluir</td>
7     </tr>
8   </thead>
9   <tbody>
10    <% if(!docs || docs.length == 0) { %>
11      <tr>
12        <td colspan="3">Nenhum item cadastrado.</td>
13      </tr>
14    <% } else {
15      docs.forEach(function(item){ %>
16        <tr>
17          <td><%= item.nome %></td>
18          <td style="width:20%"><%= item.quantidade %></td>
19          <td><a href="/delete/<%= item._id %>"
20 onclick="return confirm('Tem certeza?');">X</a></td>
21        </tr>
22      <% })
23    }%>
24  </tbody>
25  <tfoot>
26    <tr>
27      <td colspan="3">
28        <a href="/new">Cadastrar Novo</a>
29      </td>
30    </tr>
31  </tfoot>
32 </table>

```

- Nossa, ficou uma tabela linda!



- Sarcasmo é algo típico entre os coalas, ou você pegou isso com a galera aqui do escritório?

- Você jamais saberá!

- Certo! Você viu que eu adicionei um confirm JavaScript para evitar exclusões acidentais? Assim, somente se o usuário confirmar a exclusão é que a requisição delete/_id vai acontecer. Essa rota será acessada via GET, afinal é um link e devemos configurar isso na sequência.

- Primeiro temos que criar uma nova function no db.js, correto?

- Exato, faremos isso agora!

```
1 var ObjectId = require("mongodb").ObjectId;
2 function deleteOne(id, callback){
3   global.conn.collection("itens").deleteOne({_id:
4     new ObjectId(id)}, callback);
5 }
6
7 module.exports = { findAll, insert, deleteOne }
```

- Essa é uma função bem simples de entender, depois que passamos pelas anteriores. A única diferença, porém, é que eu tive que carregar aquele objeto ObjectId ali para poder transformar o id string em um id, objeto necessário para o MongoDB achar o item certo na coleção de itens.

- Isso era realmente necessário?

- Sim, uma vez que o estilo de dado do _id padrão do MongoDB é ObjectId e não String. Na hora dele percorrer o índice, procurando pelo id a ser excluído, ele primeiro verificaria a tipagem dos dados e não encontraria nenhuma equivalência, não excluindo nada.

- I got it!

- Agora sim vamos criar a rota GET /delete no routes/index.js:

```
1 /* GET delete page. */
2 router.get('/delete/:id', function(req, res) {
3   var id = req.params.id;
4   global.db.deleteOne(id, (e, r) => {
5     if(e) { return console.log(e); }
6     res.redirect('/');
7   });
8 });
```

- Deixa eu ver se entendi: nessa rota, após excluirmos o cliente usando a função da variável global.db, redirecionaremos o usuário de volta à tela de listagem, para que a mesma se mostre atualizada?
- Exatamente, Umblerito. E como poderíamos fazer uma edição?
- Acho que agora consigo fazer sozinho. Vou começar mexendo na interface da tabela da index.js, adicionando um link de edição ao redor do nome do item naquele forEach lá da index.ejs:

```
1 <tr>
2   <td><a href="/new/<%= item._id %>"><%= item.nome %></a></td>
3   <td style="width:20%"><%= item.quantidade %></td>
4   <td><a href="/delete/<%= item._id %>" onclick="return
5   confirm('Tem certeza?');">X</a></td>
6 </tr>
```

- Sim, e você já pode testar acessando localhost:3000.



- Mas, por que você colocou o link apontando para /new/_id, Umblerito?
- Porque vou aproveitar a mesma tela de cadastro para fazer a edição de itens da lista de compras. Quando a página /new carregar eu verifico se veio um id na request, e se vier, eu preencho os dados do mesmo para edição.
- Saquei! Mas vai dar trabalho. Você vai ter que editar diversos locais do seu projeto para acomodar cadastro e edição na mesma tela.
- Sim! Mas assim eu consigo finalizar esse CRUD em grande estilo. Vou começar criando uma function no db.js que retorne apenas um item do banco a partir do seu id:

```

1 function findOne(id, callback){
2   global.conn.collection("itens").findOne({_id: new
3   ObjectId(id)}), callback);
4 }
5
6 module.exports = { findAll, insert, deleteOne, findOne }

```

- Agora vou alterar o new.js para adicionar uma rota GET que espere um id, parecido com o que você fez antes para o delete:

```

1 router.get('/', function(req, res, next) {
2   res.render('new', { title: 'Novo Cadastro', item: {_id: '',
3   nome: '', quantidade:0} });
4 });
5
6 router.get('/:id', function(req, res, next) {
7   var id = req.params.id;
8   global.db.findOne(id, (err, item) => {
9     if(err) { return console.log(err); }
10    res.render('new', { title: 'Editar Cadastro', item });
11  })
12 });

```

- Legal, Umblérito! Você se preocupou em incluir a propriedade 'item' no model do GET / também, e com valores default. Isso evita alguns possíveis problemas na hora de carregar o EJS.

- Sim, imaginei que poderia dar algum problema. Minha mãe sempre dizia: "um coala prevenido vale por dois!"

- Agora entendi porque você sempre come o dobro do que precisa, hehehe...

- Não entendi seu comentário, mas vou continuar programando como se você não tivesse falado nada. Agora, vou editar o new.ejs para exibir os dados do item caso tenha vindo algum no model, preenchendo o value dos campos e adicionando um hidden field para guardar o _id:

```

1 <form action="/new" method="POST">
2   <p>
3     <label>Nome: <input type="text" name="nome" value="<%=
4 item.nome %>" /></label>
5   </p>
6   <p>
7     <label>Quantidade: <input type="number" name="quantidade"
8 value="<%= item.quantidade %>" /></label>
9   </p>
10  <p>
11    <input type="hidden" name="id" value="<%= item._id %>" />
12    <a href="/">Cancelar</a> | <input type="submit"
13 value="Salvar" />
14  </p>
15 </form>

```

- E para finalizar?
- Para finalizar eu tenho que ajustar o POST no new.js, para que quando ele receba um id ele faça um update ao invés de um insert.
- Mas não temos uma function de update no db.js ainda.
- Verdade, Luiz, você deveria ser professor, hein!
- Engraçadinho! Eu não devia, mas vou lhe ensinar como fazer um update em MongoDB funcionar. Digita aí:

```

1 function updateOne(id, item, callback){
2   global.conn.collection("itens").updateOne(
3     { _id: new ObjectId(id) },
4     { $set: {nome: item.nome, quantidade: item.quantidade } },
5     callback);
6 }
7
8 module.exports = { findAll, insert, deleteOne, findOne, updateOne }

```

- O primeiro parâmetro _id eu entendi que é o filtro da atualização, mas qual é a moral do parâmetro com o '\$set' ali?
- Diversas functions complexas do MongoDB possuem operadores iniciados com '\$', como os update operators das functions de atualização. O operador '\$set' em questão diz quais propriedades do documento serão atualizadas e com quais valores.

- Então o documento com o id, passado no primeiro parâmetro, terá o seu nome e quantidade atualizados conforme os valores do segundo parâmetro? Aquele com o \$set?
- Gotcha!
- Agora eu acho que consigo terminar sozinho, só tenho que ajustar o new.js para decidir entre a inserção e a atualização, conforme o id ter vindo no body do POST ou não.

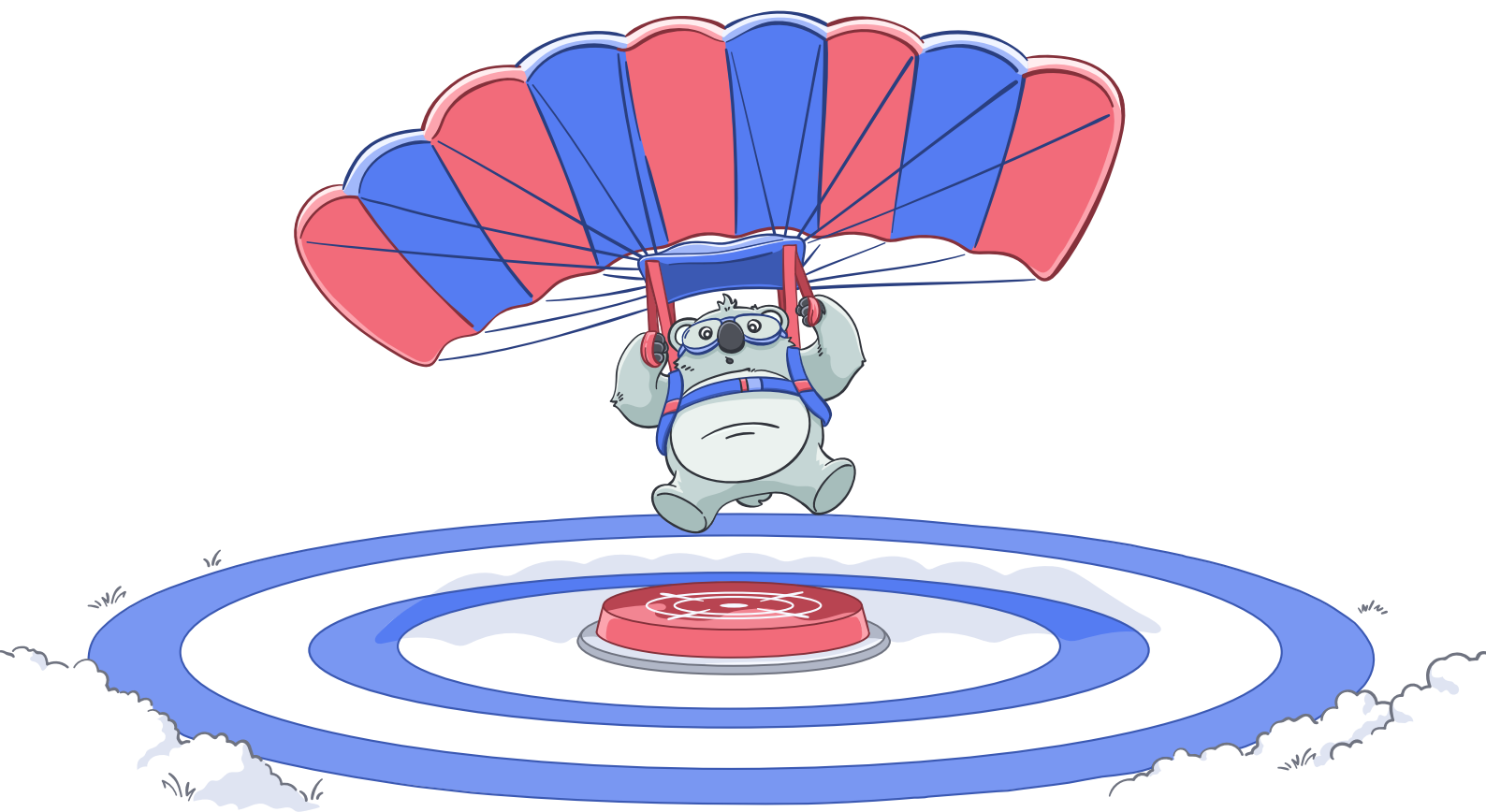
```
1  /* POST new page. */
2  router.post('/', function(req, res, next) {
3    const id = req.body.id;
4    const nome = req.body.nome;
5    const quantidade = parseInt(req.body.quantidade);
6
7    if(id){
8      global.db.updateOne(id, {nome, quantidade}, (err, result) => {
9        if(err) { return console.log(err); }
10       res.redirect('/');
11     })
12   }else {
13     global.db.insert({nome, quantidade}, (err, result) => {
14       if(err) { return console.log(err); }
15       res.redirect('/');
16     })
17   }
18 });
```

- Será que vai funcionar de primeira?
- Não, nunca funciona de primeira.
- Se funcionar de primeira você me paga uma barra de chocolate.
- Uhul! Você me deve uma barra de chocolate. Eu alterei a quantidade do



bolo de pacote e atualizou de primeira!

- Ei, eu nem havia concordado com a aposta!





Conclusão



Think twice, code once.

- Waseem Latif

Já era noite fechada lá fora, quando finalmente terminamos de fazer funcionar o CRUD usando Node.js e MongoDB. Não que tenhamos levado muito tempo, coisa de pouco mais de uma hora, mas porque já havíamos começado tarde mesmo. Hoje em dia eu teria incluído na lição como fazer deploy da aplicação web, na estrutura da Umbler. Algo tão fácil quanto fazer um push na master do nosso Git local. No entanto, foi nesse dia que o Umblerito realmente se motivou a trabalhar na nossa plataforma de Node.js e em nosso produto de MongoDB, que surgiram bem depois. E, que modéstia à parte, ficaram muito bons! :)

E agora, Luiz?

Agora eu vou para casa.

Como assim? Ainda não virei 'craque' em Node.js, você não pode ir ainda!

Certamente você ainda tem muito para aprender. Na sequência eu vou escrever um post para o blog e, de repente, um ebook aqui para o Umbler Academy mesmo, daí te passo os links. Estou pensando até em uns vídeos para o nosso canal do YouTube, mas não sei se o pessoal iria curtir.

Vídeos seriam legais. Você poderia fazer vídeos bem curtos, de 10 minutos, com esse ritmo prático e corrido das suas aulas.

Pois é, Sr. Umblerito, mas isso terá que ficar para outro dia. Pois, minha esposa e meu filho estão me esperando para o jantar. Tenho que ir agora. - Disse enquanto me levantava e pegava minha mochila. Enquanto me afastava, percebi que ele me fitava com os olhos, digno daquela famosa cena do Gato de Botas do Shrek.

...

Ok, Umblerito, você pode vir junto e continuamos a lição de Node lá em casa, no meu home office.

Ah sim, a lição... Na verdade, estava pensando mesmo era no jantar!

Hahaha, você não existe mesmo!

Nem preciso dizer o quão engraçado foi, naquela noite, os meus vizinhos de condomínio me vendo chegar com um coala gigante na garupa da moto.

Eu já falei o quanto essa empresa é maluca?

Este ebook termina aqui, mas a sua missão de se tornar um programador Node.js está longe de acabar. Boa sorte nos estudos!

Se eu fiz bem o meu trabalho, agora você está com uma vontade louca de aprender mais e de criar aplicações incríveis com Node.js, que o tornem um profissional diferenciado no mercado. A demanda por programadores Node.js está aumentando rapidamente. Meus parabéns, você está saindo na frente!

Então, agora que terminou de ler este guia, e que já conhece o básico de como criar aplicações com esta plataforma, inicie hoje mesmo (não importa se for tarde) um projeto de aplicação que use estes conhecimentos.

Buscando outros materiais?

Dê uma olhada na série de vídeos que gravei para a Umblar sobre Node.js e sobre MongoDB no canal da Umblar no YouTube e no [Umblar Academy](#), lá você encontrará muitos materiais educativos. Inclusive, muito conteúdo que eu produzi, incluindo um e-book de Boas Práticas com Node.js - complementar a este aqui.

Aproveito para fazer a indicação dos materiais que escrevo para meu blog pessoal, o [Luiz Tools](#), onde você encontra bastante conteúdo sobre Node e Mongo, além de outros livros que escrevi.

Caso tenha gostado do material, indique esse e-book para um amigo que também deseje começar a aprender Node. Não tenha medo da concorrência e abrace a ideia de ter um sócio que possa lhe ajudar em seus projetos.

Caso não tenha gostado tanto assim, envie suas dúvidas, críticas e sugestões para amigos@umblar.com. Estamos sempre dispostos a melhorar!

Ah, e é claro, não deixe de conhecer a nossa plataforma de Node.js e MongoDB no [umblar.com](#). Te garanto que o Umblarito fez um bom trabalho lá! ;)

Um abraço e até a próxima!

Luiz Duarte

Isso é a Umbler

A Umbler é uma startup de Cloud Hosting focada em agências e desenvolvedores, que oferece serviços como hospedagem de sites, registro de domínios, criação de contas de e-mail e banco de dados com pagamento pay as you go, ou por demanda. Com um painel de controle moderno e usual, o usuário consegue criar estruturas complexas de hospedagem com apenas alguns cliques. Além disso, o cliente tem total controle sobre suas aplicações, escalando, pausando ou cancelando os serviços em tempo real. No ar desde abril de 2015, conta com uma equipe com mais de 10 anos de experiência na área de hosting.

