



Quem se prepara, não para.

# Arquitetura de Aplicações Web

5º período

Professora: Michelle Hanne

# Sumário

- Propriedades dos Bancos Relacionais
- Banco de Dados NoSQL
- MongoDB
- Biblioteca Mongoose
- Como criar uma Web API com Node.js (Express + MongoDB + Mongoose)

# Propriedades dos Bancos Relacionais

- **Atomicidade:** uma transação é uma unidade atômica de processamento; ou ela será executada em sua totalidade ou não será de modo nenhum.
- **Consistência:** uma transação deve ser preservadora de consistência se sua execução passa de um estado consistente para outro também consistente.
- **Isolamento:** uma transação deve ser executada como se estivesse isolada das demais. Não deve sofrer interferência de quaisquer outras transações concorrentes
- **Durabilidade:** as mudanças aplicadas ao banco de dados por uma transação efetivada devem persistir no banco de dados. Essas mudanças não devem ser perdidas em razão de uma falha.

# Propriedades dos Bancos Relacionais

- Transação: programa em execução que forma uma unidade lógica de processamento no BD que deve ser completo e integral. Transação: inclui uma ou mais operações de acesso ao BD – inserção, exclusão, alterações ou consultas



NoSQL (Not Only SQL) utilizado pela primeira vez em 1989. Evoluiu a partir de 2004 com o lançamento do Bigtable do Google. Atualmente, há quatro categorias de banco de dados do tipo NoSQL:

- Banco de Dados orientada a documentos, ex. MongoDB
- Banco de Dados de família de colunas, ex: Cassandra
- Banco de Dados chave valor, ex: Redis
- Banco de Dados de grafos, ex: Neo4j

# Comparação entre Modelo Relacional e NoSQL

	Modelo relacional	NoSQL
<b>Escalabilidade</b>	Possível, mas complexa. Devido à natureza estruturada do modelo, a adição de forma dinâmica e transparente de novos <b>nós</b> no grid não é realizada de modo natural.	Por não possuir nenhum esquema predefinido, este tipo de banco de dados tem maior flexibilidade, o que favorece a inclusão transparente de outros elementos. Dessa forma, a escalabilidade é uma das principais vantagens do modelo.
<b>Consistência</b>	Ponto mais forte do modelo relacional. As regras de consistência presentes propiciam um maior grau de rigor quanto à consistência das informações.	Realizada de modo eventual no modelo, só garante que se nenhuma atualização for realizada sobre o item de dados, todos os acessos a ele devolverão o último valor atualizado.
<b>Disponibilidade</b>	Dada a dificuldade de se conseguir trabalhar de forma eficiente com a distribuição dos dados, esse modelo pode não suportar a demanda muito grande de informações do banco.	O alto grau de distribuição dos dados propicia que um maior número de solicitações aos dados seja atendido por parte do sistema, o qual fica menos tempo não disponível.

- MongoDB é um programa de banco de dados NoSQL, **orientado a documentos livre**, de código aberto e multiplataforma, escrito na linguagem C++. O MongoDB usa documentos semelhantes a JSON com esquemas.
- Suas características permitem com que as aplicações modelem informações de modo muito mais natural, pois os dados podem ser aninhados em hierarquias complexas e continuar a ser indexáveis e fáceis de buscar



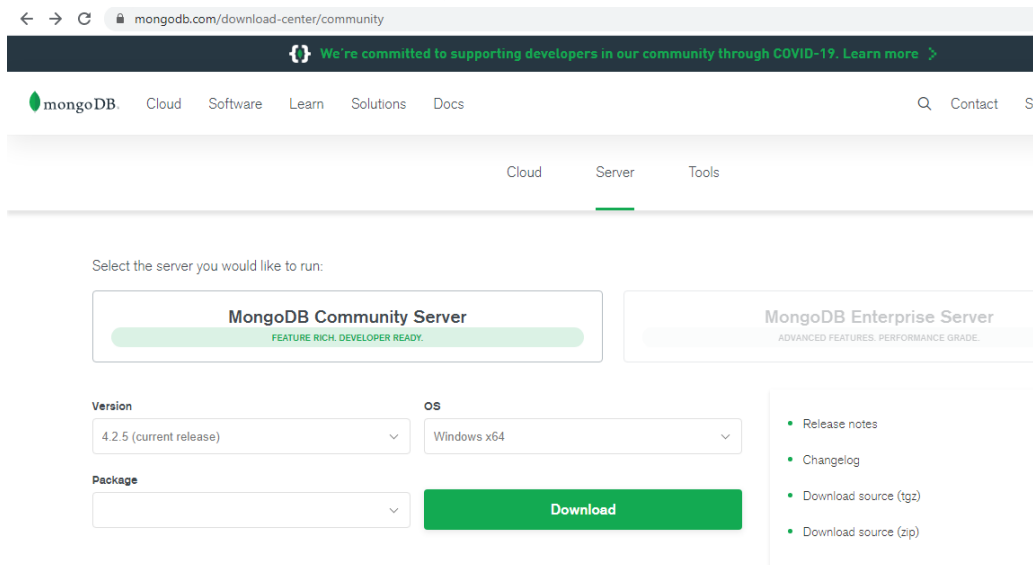
- Mongoose é uma biblioteca do Nodejs que proporciona uma solução baseada em **esquemas para modelar os dados da sua aplicação**. Ele possui sistema de conversão de tipos, validação, criação de consultas e lógica de negócios.

- Mongoose é uma biblioteca do Nodejs que proporciona uma solução baseada em **esquemas para modelar os dados da sua aplicação**. Ele possui sistema de conversão de tipos, validação, criação de consultas e lógica de negócios.

# Instalação

Acessar o [site oficial do MongoDB](https://www.mongodb.com/download-center/community) e baixar o **Server** e faça o download da versão de produção mais recente para o seu sistema operacional.

<https://www.mongodb.com/download-center/community>



The screenshot shows the MongoDB download center community page. At the top, there's a navigation bar with links for Cloud, Software, Learn, Solutions, and Docs. Below this, there's a section titled "Select the server you would like to run:" with two options: "MongoDB Community Server" (highlighted with a green bar) and "MongoDB Enterprise Server". Under the "MongoDB Community Server" option, there are dropdown menus for "Version" (set to "4.2.5 (current release)") and "OS" (set to "Windows x64"). There's also a "Package" dropdown menu. A green "Download" button is visible. To the right of the download options, there's a list of links: "Release notes", "Changelog", "Download source (tgz)", and "Download source (zip)".

# Como criar uma Web API com Node.js (Express + MongoDB + Mongoose)

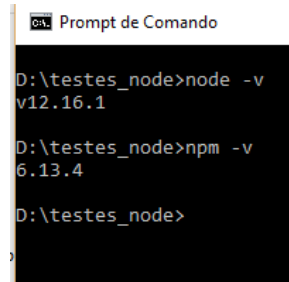
- <https://www.luiztools.com.br/post/como-criar-uma-web-api-com-nodejs/>

# CRUD em Node.js com driver nativo do MongoDB

- CRUD completo usando Node.js (com Express + EJS) e MongoDB (driver nativo). Os passos:
  1. Configurando o Node.js
  2. Entendendo o projeto Express
  3. Configurando o MongoDB
  4. Conectando no MongoDB com Node
  5. Cadastrando no banco
  6. Atualizando clientes
  7. Excluindo clientes

# 1 – Configurando o Node.js

- Instale o Node.JS do site oficial (<https://nodejs.org/en/>), também será instalado o NPM, que é o gerenciador de pacotes do Node.
- **Verifique a versão:**



```
014 Prompt de Comando

D:\testes_node>node -v
v12.16.1

D:\testes_node>npm -v
6.13.4

D:\testes_node>
```

# 1 – Configurando o Node.js

- Criando um projeto com Express

- a) Crie uma pasta para o seu projeto

```
C:\> Selecionar Prompt de Comando  
  
D:\testes_node>mkdir CRUD_Node  
  
D:\testes_node>cd CRUD_Node
```

- b) Instale o módulo do Express Generation

```
C:\> Prompt de Comando  
  
D:\testes_node\CRUD_Node>npm install -g express-generator  
npm WARN deprecated mkdirp@0.5.1: Legacy versions of mkdirp are no  
longer supported. Please use the latest version of mkdirp from the  
npm registry.  
C:\Users\michelle_pc\AppData\Roaming\npm\express -> C:\Users\mich  
elle_pc\AppData\Roaming\npm\express-cli.js  
+ express-generator@4.16.1  
added 10 packages from 13 contributors in 4.267s  
  
D:\testes_node\CRUD_Node>_
```

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.

# 1 – Configurando o Node.js

- Crie rapidamente uma estrutura básica de um projeto Express via linha de comando

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

```
Prompt de Comando

added 10 packages from 13 contributors in 4.267s

D:\testes_node\CRUD_Node>express -e --git workshoptdc

warning: option `--ejs' has been renamed to `--view=ejs'

create : workshoptdc\
create : workshoptdc\public\
create : workshoptdc\public\javascripts\
create : workshoptdc\public\images\
create : workshoptdc\public\stylesheets\
create : workshoptdc\public\stylesheets\style.css
create : workshoptdc\routes\
create : workshoptdc\routes\index.js
create : workshoptdc\routes\users.js
create : workshoptdc\views\
create : workshoptdc\views\error.ejs
create : workshoptdc\views\index.ejs
create : workshoptdc\.gitignore
create : workshoptdc\app.js
create : workshoptdc\package.json
create : workshoptdc\bin\
create : workshoptdc\bin\www

change directory:
> cd workshoptdc

install dependencies:
> npm install
```



# 1 – Configurando o Node.js

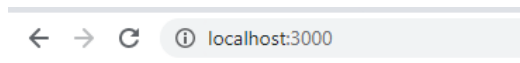
- Entrar na pasta e **instalar as dependências com npm install**, em seguida dentro da pasta do projeto vamos **inicializar o npm**

```
ca. npm
D:\testes_node\CRUD_Node>cd workshoptdc
D:\testes_node\CRUD_Node\workshoptdc>npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
added 54 packages from 38 contributors and audited 141 packages in 6.202s
found 0 vulnerabilities

D:\testes_node\CRUD_Node\workshoptdc>npm start
> workshoptdc@0.0.0 start D:\testes_node\CRUD_Node\workshoptdc
> node ./bin/www
```

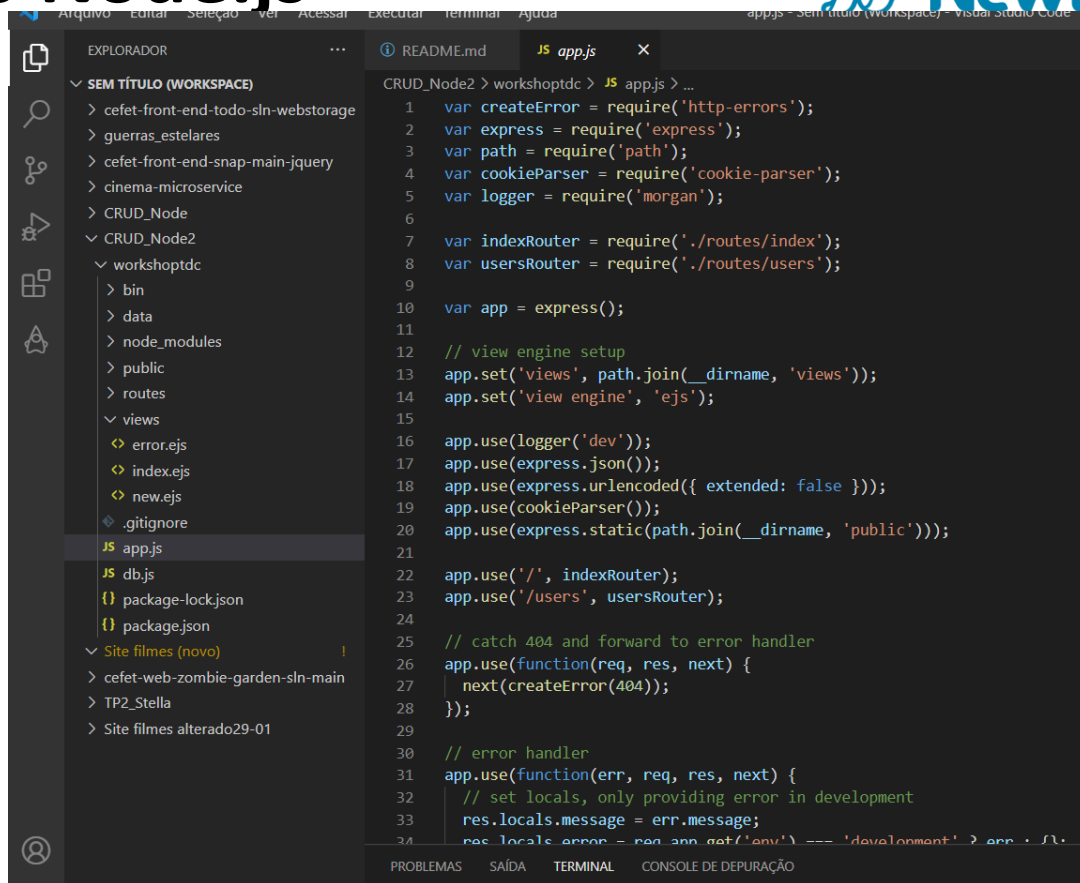
Isso vai fazer com que a aplicação default inicie sua execução em localhost:3000, que você pode acessar pelo seu navegador.

# 1 – Configurando o Node.js



## Express

Welcome to Express



# 2 – Entendendo o projeto Express

## Abrir o arquivo app.js

```
var createError = require('http-errors');  
var express = require('express');  
var path = require('path');  
var cookieParser = require('cookie-parser');  
var logger = require('morgan');  
  
var indexRouter = require('./routes/index');  
var usersRouter = require('./routes/users');  
  
var app = express();
```

Define variáveis JavaScript e referencia 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. 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.

Instancia o Express e associa nossa variável app à ele.

## 2 – Entendendo o projeto Express

```
// view engine setup
app.set('views', path.join(__dirname, 'views
'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false
}));
app.use(cookieParser());
app.use(express.static(path.join(__dirname,
'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
```

Usa a variável `app` para configurar recursos do Express. Diz ao `app` onde ele encontra suas views, qual engine usar para renderizar as views (EJS) e chama outras funções.

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 `c:\node\workshoptdc\public\images` mas é acessada em `http://localhost:3000/images`

## 2 – Entendendo o projeto Express

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') ===
  'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

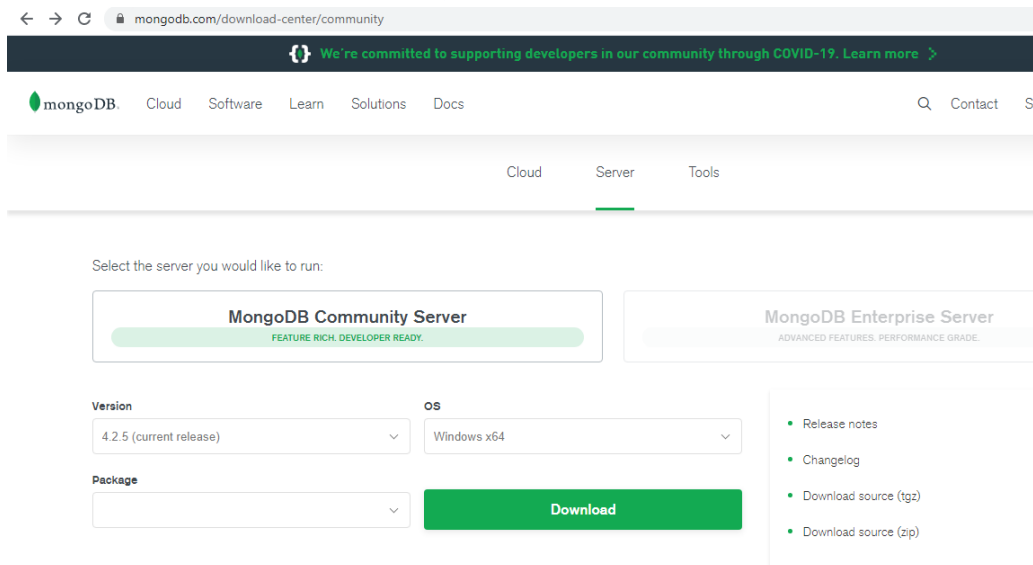
São manipuladores de erros para desenvolvimento e produção (além dos 404).

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.**

# 3 – Configurando o MongoDB

Acessar o [site oficial do MongoDB](https://www.mongodb.com/download-center/community) e baixar o Mongo. Clique em **Server** e faça o download da versão de produção mais recente para o seu sistema operacional.

<https://www.mongodb.com/download-center/community>




The screenshot shows the MongoDB download page for the community server. The page has a dark header with the MongoDB logo and navigation links. Below the header, there's a section titled "Select the server you would like to run:" with two options: "MongoDB Community Server" (highlighted with a green bar) and "MongoDB Enterprise Server". Below these options, there are dropdown menus for "Version" (set to "4.2.5 (current release)") and "OS" (set to "Windows x64"). There is also a "Package" dropdown menu. A green "Download" button is visible. On the right side, there's a list of links: "Release notes", "Changelog", "Download source (tgz)", and "Download source (zip)".

# 3 – Configurando o MongoDB

## Executar mongod e mongo

Dentro da pasta do seu projeto CRUD-Node, acessar a subpasta workshoptdc, verificar se existe uma subpasta data. Você pode criar manualmente ou via terminal:

 Prompt de Comando

```
D:\testes_node\CRUD_Node\workshoptdc>mkdir data  
D:\testes_node\CRUD_Node\workshoptdc>
```

Nesta pasta vamos armazenar nossos dados do MongoDB

# 3 – Configurando o MongoDB

Pelo prompt de comando, entre na **subpasta bin** dentro da pasta de instalação do seu **MongoDB** e digite:

```
C:\>cd Program Files\MongoDB\Server\4.2\bin

C:\Program Files\MongoDB\Server\4.2\bin>mongod --dbpath d:\teste_node\CRUD_Node\workshoptdc\data\
2020-04-04T22:19:08.076-0300 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --s
sabledProtocols 'none'
2020-04-04T22:19:08.830-0300 W ASIO [main] No TransportLayer configured during NetworkInterface startup
2020-04-04T22:19:08.835-0300 I CONTROL [initandlisten] MongoDB starting : pid=9960 port=27017 dbpath=d:\teste_node
D_Node\workshoptdc\data\ 64-bit host=DESKTOP-810DIJK
2020-04-04T22:19:08.839-0300 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2020-04-04T22:19:08.842-0300 I CONTROL [initandlisten] db version v4.2.5
2020-04-04T22:19:08.844-0300 I CONTROL [initandlisten] git version: 2261279b51ea13df08ae708ff278f0679c59dc32
2020-04-04T22:19:08.847-0300 I CONTROL [initandlisten] allocator: tcmalloc
2020-04-04T22:19:08.849-0300 I CONTROL [initandlisten] modules: none
2020-04-04T22:19:08.850-0300 I CONTROL [initandlisten] build environment:
2020-04-04T22:19:08.852-0300 I CONTROL [initandlisten] distmod: 2012plus
2020-04-04T22:19:08.854-0300 I CONTROL [initandlisten] distarch: x86_64
2020-04-04T22:19:08.856-0300 I CONTROL [initandlisten] target_arch: x86_64
2020-04-04T22:19:08.859-0300 I CONTROL [initandlisten] options: { storage: { dbPath: "d:\teste_node\CRUD_Node\work
tdc\data\" } }
2020-04-04T22:19:08.885-0300 I STORAGE [initandlisten] exception in initAndListen: NonExistentPath: Data directory
teste_node\CRUD_Node\workshoptdc\data\ not found., terminating
2020-04-04T22:19:08.890-0300 I NETWORK [initandlisten] shutdown: going to close listening sockets...
2020-04-04T22:19:08.893-0300 I - [initandlisten] Stopping further Flow Control ticket acquisitions.
```



# 3 – Configurando o MongoDB

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

```
C:\>cd Program Files\MongoDB\Server\4.2\bin
C:\Program Files\MongoDB\Server\4.2\bin>mongo
MongoDB shell version v4.2.5
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("6adecaa1-a331-42d5-98ca-bfbc8934f209") }
MongoDB server version: 4.2.5
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
Server has startup warnings:
2020-04-04T22:11:36.552-0300 I  CONTROL  [initandlisten]
2020-04-04T22:11:36.552-0300 I  CONTROL  [initandlisten] ** WARNING: Access control is not
2020-04-04T22:11:36.552-0300 I  CONTROL  [initandlisten] **           Read and write access
unrestricted.
2020-04-04T22:11:36.552-0300 I  CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
```

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.

# 3 – Configurando o MongoDB

## Criando uma base de dados

No console do cliente mongo, digite:

```
> use workshoptdc  
switched to db workshoptdc  
>
```

Agora estamos usando a base “workshoptdc.” No entanto, ela somente será criada de verdade quando adicionarmos registros nela

# 3 – Inserindo Alguns Dados

O MongoDB usa JSON como estrutura de dados. 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:

# 3 – Inserindo Alguns Dados

CA Prompt de Comando - mongo

```
> db.customers.insert({ "nome" : "Luiz", "idade" : 29 })
WriteResult({ "nInserted" : 1 })
>
```

CA Prompt de Comando - mongo

```
> db.customers.insert({ "nome" : "Pedro", "idade" : 35 })
WriteResult({ "nInserted" : 1 })
> db.customers.insert({ "nome" : "Maria", "idade" : 18 })
WriteResult({ "nInserted" : 1 })
>
```

Podem inserir mais registros:

```
db.customers.insert({ "nome" : "Pedro", "idade" : 35 })
```

```
db.customers.insert({ "nome" : "Maria", "idade" : 18 })
```

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.

# 3 – Inserindo Alguns Dados

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

```
> db.customers.find().pretty()
{
  "_id" : ObjectId("5e8937658d25446a4c79e40a"),
  "nome" : "Luiz",
  "idade" : 29
}
{
  "_id" : ObjectId("5e8938f88d25446a4c79e40b"),
  "nome" : "Pedro",
  "idade" : 35
}
{
  "_id" : ObjectId("5e89390c8d25446a4c79e40c"),
  "nome" : "Maria",
  "idade" : 18
}
> 1
1
> db.customers.find().pretty()
```

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

*o seu \_id pode ser diferente desse, uma vez que o Mongo irá gerá-lo automaticamente*

## 3 – Inserindo Alguns Dados

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

Exemplo com um array com vários objetos para a 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.**

# 4 – Conectando no MongoDB com Node

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:

Ca Prompt de Comando

```
D:\testes_node\CRUD_Node\workshoptdc>npm install -S mongodb
+ mongodb@3.5.5
added 16 packages from 10 contributors and audited 162 packages in 6.549s
found 0 vulnerabilities

D:\testes_node\CRUD_Node\workshoptdc>_
```

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

# 4 – Conectando no MongoDB com Node

## 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:



# 4 – Conectando no MongoDB com Node

cmd Prompt de Comando

```
D:\testes_node\CRUD_Node\workshoptdc>type nul >db.js
```

```
D:\testes_node\CRUD_Node\workshoptdc>_
```

```
var MongoClient = require("mongodb").MongoClient;
MongoClient.connect("mongodb://localhost/workshoptdc")
    .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.

# 4 – Conectando no MongoDB com Node

cmd Prompt de Comando

```
D:\testes_node\CRUD_Node\workshoptdc>type nul >db.js
```

```
D:\testes_node\CRUD_Node\workshoptdc>_
```

```
var MongoClient = require("mongodb").MongoClient;
MongoClient.connect("mongodb://localhost/workshoptdc")
  .then(conn => global.conn = conn.db("workshoptdc"))
  .catch(err => console.log(err))
```

```
module.exports = { }
```

Será configurada  
posteriormente

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.

## 4 – Conectando no MongoDB com Node

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.

# 4 – Conectando no MongoDB com Node

## 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**, 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 usa 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.

## 4 – Conectando no MongoDB com Node

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 }
```

# 4 – Conectando no MongoDB com Node

## Rota da função de Consulta

Abra o arquivo `C:\node\workshoptdc\routes\index.js`, edite a rota default, que é um get no path raiz. Vamos editar essa rota da seguinte maneira:

```
/* GET home page. */
router.get('/', async (req, res, next) => {
  try {
    const docs = await global.db.findAll();
    res.render('index', { title: 'Lista de Clientes', docs });
  } catch (err) {
    next(err);
  }
})
```

**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**. 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.*

# 4 – Conectando no MongoDB com Node



## Editando a view de listagem

Agora vamos organizar a nossa view para listar os clientes. Entre na pasta **C:\node\workshoptdc\views\** e edite o arquivo **index.ejs** para que fique desse jeito:

O objeto **docs**, que será retornado pela rota que criamos no passo anterior, será iterado com um **forEach**, cujos objetos irão compor uma lista não-ordenada com nomes.

```
<html>
  <head>
    <title><%= Pagina de teste CRUD %></title>
    <link rel='stylesheet' href='/stylesheets/s
type.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <ul>
      <% docs.forEach(function(customer){ %>
        <li>
          <%= customer.nome %>
        </li>
      <% }) %>
    </ul>
  </body>
</html>
```

# Testando Até o Momento

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 ative novamente.

```
Pasta de D:\testes_node\CRUD_Node\workshoptdc\bin
04/04/2020 21:18 <DIR> .
04/04/2020 21:18 <DIR> ..
04/04/2020 21:18 1.597 www
1 arquivo(s) 1.597 bytes
2 pasta(s) 675.582.717.952 bytes disponíveis

D:\testes_node\CRUD_Node\workshoptdc\bin>cd ..
D:\testes_node\CRUD_Node\workshoptdc>npm start
> workshoptdc@0.0.0 start D:\testes_node\CRUD_Node\workshoptdc
> node ./bin/www
```

Reiniciar seu servidor Node e  
acessar <http://localhost:3000/>



# 5 – Cadastrando no banco

## Fazendo o CRUD Funcionar (Create, Read, Update e Delete)

- Cria a função no db.js
- Altera a rota correspondente
- Atualiza a View se necessário.

# 5 – Cadastrando no banco

## Criando sua view de cadastro

Primeiro vamos criar a nossa tela de cadastro de usuário com dois clássicos. Dentro da pasta **views**, crie um **new.ejs** com o seguinte HTML dentro:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= Cadastro Exemplo %></title>
    <link rel='stylesheet' href='/stylesheets/style.
css' />
  </head>
  <body>
    <h1><%= Cadastro %></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>
```

# 5- Cadastrando no banco

## Adicionando Rota

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, a segunda será a rota POST:

Se você reiniciar seu servidor Node e acessar <http://localhost:3000/newuser> Porém o botão SALVAR não vai funcionar!

# 5- Cadastrando no banco

## 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").insertOne(customer);  
}  
module.exports = { findAll, insert }
```

# 5- Cadastrando no banco

## Adicionando Rota

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**.

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', async (req, res, next) => {  
  const nome = req.body.nome;  
  const idade = parseInt(req.body.idade);  
  
  try {  
    const result = await global.db.insert({ nome, idade });  
    console.log(result);  
    res.redirect('/');  
  } catch (err) {  
    next(err);  
  }  
})
```

# Cadastrando no banco

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.

# 5 – Cadastrando no banco

## Link para a Página

Edite o views/index.ejs para incluir um link para a página /new:

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

# Parte 5 – Cadastrando no banco

## Link para a Página

Edite o views/index.ejs para incluir um link para a página /new:

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



# 6 – Atualizando clientes

Para atualizar clientes (update) 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.

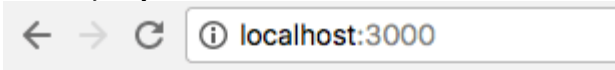
## Passo 1: Arrumando a tela de listagem

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:

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

# 6 – Atualizando clientes

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



## Lista de Clientes

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

---

[Cadastrar novo cliente](#)

# 6 – Atualizando clientes

## Carregando os dados existentes

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

```
function findOne(id) {  
    return global.conn.collection("customers").findOne(new ObjectId(id));  
}  
  
module.exports = { findAll, insert, findOne }
```

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`.**

# 6 – Atualizando clientes

## Criando a Rota GET

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/' + docs[0]._id });  
  });  
})
```

Pedimos ao **db** que encontre o cliente cujo **id** veio como parâmetro da requisição (**req.params.id**). Em seguida, mandamos renderizar a mesma **view** de cadastro, porém com um **model** inteiramente novo contendo apenas o cliente a ser editado) e a action do **form** da **view** 'new.ejs'.

# 6 – Atualizando clientes

## Alterando a Rota GET em /new

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: '/new' });  
});
```

# 6 – Atualizando clientes

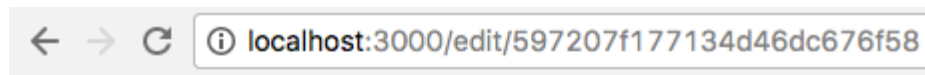
## Alterando o arquivo New.js

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**:

```
<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>
```

# 6 – Atualizando clientes

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



## Edição de Cliente

Nome:

Idade:

# 6 – Atualizando clientes

## Codificando o update

Primeiro precisamos criar um nova função no **db.js** para fazer update, como abaixo:

```
function update(id, customer, callback){  
    global.conn.collection("customers").updateOne({_id:new ObjectId(id)}, customer, callback);  
}  
  
module.exports = { findAll, insert, findOne, update }
```

Temos de passar o filtro do update para saber qual documento será afetado (neste caso somente aquele que possui o id específico).



# 6 – Atualizando clientes

## Configurando Rota para o update

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:

```
router.post('/edit/:id', function(req, res) {  
  var id = req.params.id;  
  var nome = req.body.nome;  
  var idade = parseInt(req.body.idade);  
  global.db.update(id, {nome, idade}, (e, result) => {  
    if(e) { return console.log(e); }  
    res.redirect('/');  
  });  
});
```

O **id**, que veio como parâmetro na URL foi carregado, e os dados de **nome** e **idade** no **body** da requisição.

# 7- Excluindo clientes

## Editando a listagem

Vamos voltar à listagem (**views\index.jes**) 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

```
<html>
  <head>
    <title><%= Pagina de teste CRUD %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= Listagem de Clientes %></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>
```

# 7- Excluindo clientes

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 }
```

# 7- Excluindo clientes

## Configurando Rota para o Delete

Criar a rota GET /delete no routes/index.js:

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.

```
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('/');
  });
});
```

# DICAS

[https://youtu.be/uQTtFuxq\\_Qs](https://youtu.be/uQTtFuxq_Qs)



# DICAS

<https://www.youtube.com/watch?v=qT5y17gJug>



# DICAS

<https://www.youtube.com/watch?v=Khr8bMODung>

