**Curso básico Node.JS**

**Prof. Luiz Duarte (Luiz Tools)**

# Aula 01 - O que é Nodejs

JavaScript:

* Linguagem web mais popular do mundo;
* baixa curva de aprendizado;
* forte comunidade;



Leve e Multiplataforma:

* baixo custo de hardware;
* baixo custo de SO;
* sem licença;

- É utilizada por várias instituições corporativas:



# Aula 02 - Ambiente Nodejs e MongoDB

## Ambiente



- O Node, em si, não é uma linguagem de programação. Ele é uma plataforma para aplicações composta por:

* Interpretador V8 (o mesmo do Google Chrome): Motor de JavaScript;
* Biblioteca IO (libuv): biblioteca C++ para leitura e escrita;
* package manager (NPM): gerenciador de pacotes;
* ferramenta REPL: linha de comando onde se pode usar o Node diretamente do console;
* version manager (NVM): gerenciador de versões que permite usar mais de uma versão do Node na máquina

- Acesse nodejs.org, baixe e instale o Node.

## Ferramenta

- Acesse code.visualstudio.com, baixe e instale o Visual Studio Code.



- Acesse mongodb.org, baixe e instale o MongoDB (Versão Community Server)



# Aula 03 - O que é JavaScript

Características:

* Criado por Brendan Eich;
* Usado inicialmente no Netscape (1995);
* Especificação formal (norma): ECMAScript (1996);
* Linguagem imperativa e estruturada;
* Tipagem dinâmica (não se declara tipos nos dados);
* Baseada em objetos (em JavaScript tudo é um objeto);
* Interpretada (É executado linha a linha. Não retorna erros de compilação);

- JavaScript não é orientado a classes. É orientado a protótipos.

**- Typescript**: é um subset criado encima do JavaScript para que ele forneça recursos de tipagem, de objetos principalmente, e alguns outros recursos ligados a orientação a objetos. Adiciona uma nova camada.

- Neste curso não será usado typescript.

# Aula 04 - Revisão de JavaScript - Parte 1

Sintaxe

* const, let, var
* logical, relational e arithmetic operators
* functions e anonymous/arrow functions

## Variáveis em JavaScript

* var cria uma variável que pode ser alterada.
* const cria uma variável que não pode ser alterada.
* let cria uma variável que só existe dentro do escopo em que foi criado (bloco entre chaves).

- Crie uma pasta chamada aula1 e dentro dela adicione um arquivo chamado index.js:



cd aula1

**index.js**

var nome = 'Roberto';

console.log("Nome: " + nome);

let numero1 = 3;

console.log("Número 1: " + numero1);

let numero2 = 6.9;

console.log("Número 2: " + numero2);

var isAccepted = true; // ou false

console.log("É aceito: " + isAccepted);

// objeto

var user = {};

user.nome = 'Gilberto';

user.idade = 17;

console.log("Nome do usuário: " + user.nome);

console.log("Idade do usuário: " + user.idade);

// array

var colecao = [];

colecao[0] = 1;

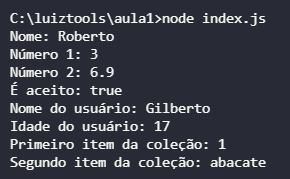
colecao[1] = 'abacate';

console.log("Primeiro item da coleção: " + colecao[0]);

console.log("Segundo item da coleção: " + colecao[1]);

- O ponto e vírgula no final de um comando é opcional na maioria das versões do JavaScript (exceto nas versões mais antigas)

node index.js



# Aula 05 - Revisão de JavaScript - Parte 2

## Operadores

### Operadores de comparação

**operadores\_comparacao.js**

if(3 == '3'){

console.log("3 == '3': true");

}

if(3 === '3'){

console.log("3 === '3': true");

} else {

console.log("3 === '3': false");

}

if(3 != '3'){

console.log("3 != '3': true");

} else {

console.log("3 != '3': false");

}

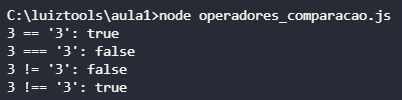
if(3 !== '3'){

console.log("3 !== '3': true");

} else {

console.log("3 !== '3': false");

}



### Operadores lógicos

|  |  |
| --- | --- |
| && | and |
| || | or |
| ! | not |

### Operadores relacionais

|  |  |
| --- | --- |
| > | maior |
| < | menor |
| >= | maior ou igual |
| <= | menor ou igual |

### Operadores aritméticos

|  |  |
| --- | --- |
| + | soma |
| - | subtração |
| \* | multiplicação |
| / | divisão |
| % | resto |

## Funções

**somar.js**

function somar(num1, num2){

return num1 + num2;

}

var result = somar(3, 4.5);

console.log(result);

node somar.js



## Anonymous Function

**anonymous\_function.js**

const somar = function (num1, num2) {

return num1 + num2;

}

var result = somar(3, 61);

console.log(result);

node anonymous\_function.js



## Arrow Function

**arrow\_function.js**

const somar = (num1, num2) => {

return num1 + num2;

}

var result = somar(6, 8);

console.log(result);

node arrow\_function.js



# Aula 06 - Revisão de JavaScript - Parte 3

## Estruturas de controle e repetição

**estruturas.js**

// Estruturas de controle e repetição

alunos = [{nome: "Carlos", media: 7.5},{nome: "Renato", media: 4.75},{nome:"Francisco",media: 3.5}];

for (let i = 0; i < alunos.length; i++) {

if(alunos[i].media > 5){

console.log(alunos[i].nome + " - Média: " + alunos[i].media + " - Aprovado!" );

} else if(alunos[i].media > 4 && alunos[i].media < 5) {

console.log(alunos[i].nome + " - Média: " + alunos[i].media + " - Em recuperação!" );

} else {

console.log(alunos[i].nome + " - Média: " + alunos[i].media + " - Reprovado!" );

}

}

var j = 0;

while(j < 5){

console.log(j);

j++;

}

var option = 3;

switch(option){

case 1:

console.log("Escolheu a opção 1");

break;

case 2:

console.log("Escolheu a opção 2");

break;

case 3:

console.log("Escolheu a opção 3");

break;

case 4:

console.log("Escolheu a opção 4");

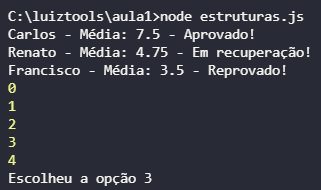
break;

default:

console.log("Não escolheu nenhuma das quatro opções!");

}

node estruturas.js



## Tipos de dados

* Number;
* String;
* Boolean;
* Array;
* Object;
* Function;

## Funções de String

* length, indexOf e toString
* toUpperCase e toLowerCase
* endsWith e startsWith
* replace e trim
* parseFloat e parseInt
* split: transforma uma string em array;
* charAt: informa qual é a letra passada por parâmetro;
* slice: para pegar um pedaço de uma string (uma substring);

**strings.js**

var nome1 = "Roberto";

var nome2 = "Lucas";

var nome3 = " Felipe ";

var nome4 = "Roberto Pinheiro";

var num1 = "5";

var num2 = "57.31";

if(nome1.indexOf("o") != -1){

console.log(nome1.toUpperCase());

} else {

console.log(nome1.toLocaleLowerCase());

}

if(nome2.indexOf("o") != -1){

console.log(nome2.toUpperCase());

} else {

console.log(nome2.toLocaleLowerCase());

}

if(nome1.indexOf("o") != -1){

console.log(nome1.replace("o", "3"));

}

console.log(nome3.trim());

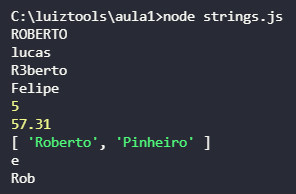
console.log(parseInt(num1));

console.log(parseFloat(num2));

console.log(nome4.split(" "));

console.log(nome1.charAt(3));

console.log(nome1.slice(0,3));



## Arrays

* length
* push
* forEach
* splice: para remover pedaços de um array
* indexOf: para saber em qual posição do array está um determinado texto

**arrays.js**

var nomes = ["Geraldo", "Cássio", "Ademir", "José", "Valdir"];

nomes.push("Valter");

var total\_nomes = nomes.length;

console.log("Total de nomes: " + total\_nomes);

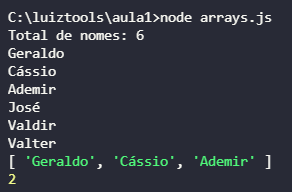
nomes.forEach(function(item){

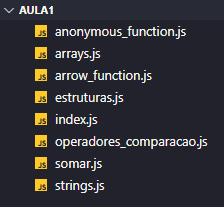
console.log(item);

});

console.log(nomes.splice(0, 3));

console.log(nomes.indexOf("Valter"));





# Aula 07 - O que é Nodejs - Parte 2

Características:

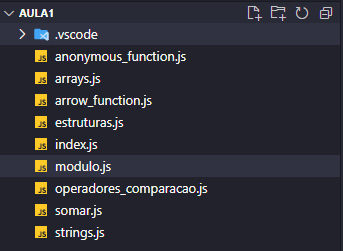
* Interpretador (V8) + libuv
* Single Thread
* Assíncrono
* Plataforma != linguagem != framework
* Bom para API's e scripts
* backend de games
* IoT
* mensageria

O Node é uma plataforma. É um hospedeiro de aplicações.

# Aula 09 - Nodejs na Prática - Parte 2

- No Node cada arquivo js é um módulo.

## Módulos



**modulo.js**

function somar(valor1, valor2){

return valor1 + valor2;

}

module.exports = {somar}

**index.js**

const modulo = require("./modulo");

var nome = 'Roberto';

console.log("Nome: " + nome);

let numero1 = 3;

console.log("Número 1: " + numero1);

let numero2 = 6.9;

console.log("Número 2: " + numero2);

var isAccepted = true; // ou false

console.log("É aceito: " + isAccepted);

// objeto

var user = {};

user.nome = 'Gilberto';

user.idade = 17;

console.log("Nome do usuário: " + user.nome);

console.log("Idade do usuário: " + user.idade);

// array

var colecao = [];

colecao[0] = 1;

colecao[1] = 'abacate';

console.log("Primeiro item da coleção: " + colecao[0]);

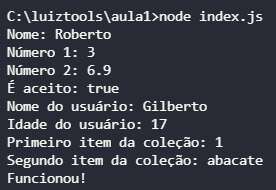
console.log("Segundo item da coleção: " + colecao[1]);

if(modulo.somar(1,2) === 3 ){

console.log("Funcionou!");

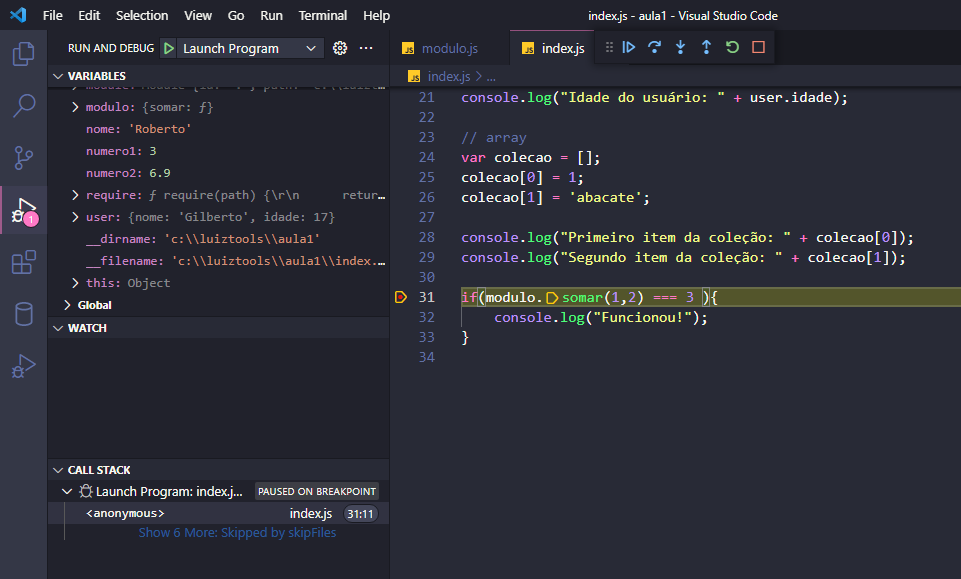
}

node index.js

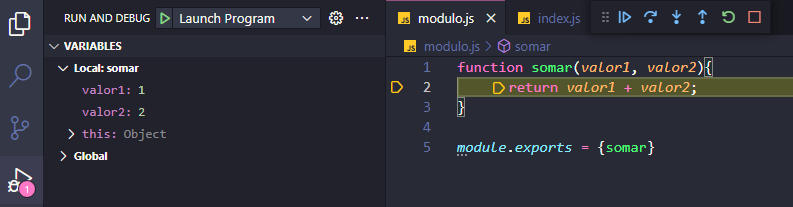


## Debug no Visual Studio Code

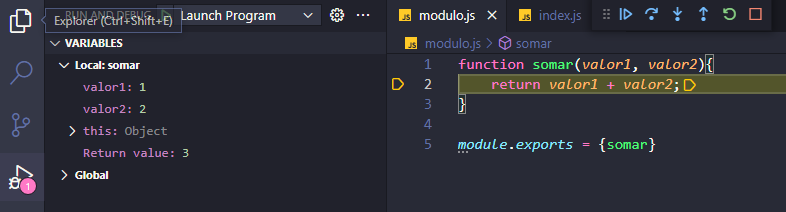
- Faça um breakpoint na linha 31.



- Clique no botão Step into:



- Clique novamente no botão Step into:

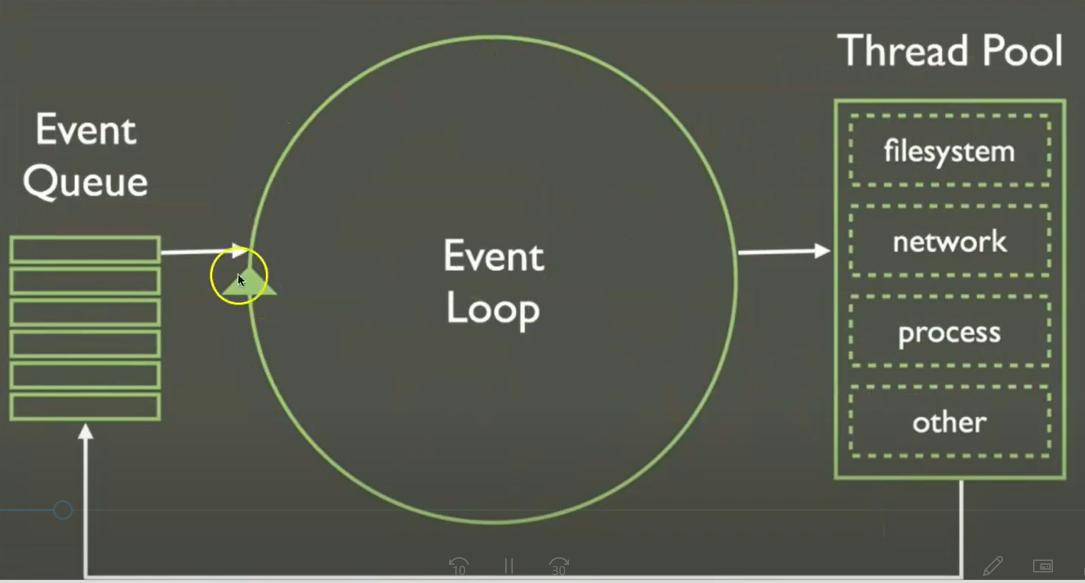


# Aula 10 - Event Loop do Nodejs

- O comportamento assíncrono no JS se deve em parte a sua arquitetura centrada em event loop.

- Event loop é uma threat única, uma single threat principal, onde todas as requisições chegam inicialmente nele, são processadas imediatamente e depois retornam a sua execução (resultado da sua computação) no fluxo principal e consequentemente para o usuário.

## Event loop



- Ao chegar uma requisição, ela vai para uma fila de eventos.

- Se ela for a primeira da fila, ela vai entrar no Event Loop. Se é rápido de processar, ele já processa e devolve para o usuário. Depois vai para o próximo evento da fila. Se não for rápido de processar, vai para um pool de threads. O Node tem uma thread principal (single thread), mas mantém um pool de threads (quatro). Por ele não abrir uma thread por requisição, mas sim compartilhar um pool de threads, ele se torna mais eficiente em consumo de recursos.

- O Node envia para o pool de threads tudo o que é de execução demorada e retorna para atender a próxima requisição da fila.

Características:

* V8 Single Thread
* libuv (C++) Multi Thread
* 1 ping = 88M instruções de CPU
* O que fazer enquanto espera?

# Aula 11 - O que é ExpressJS

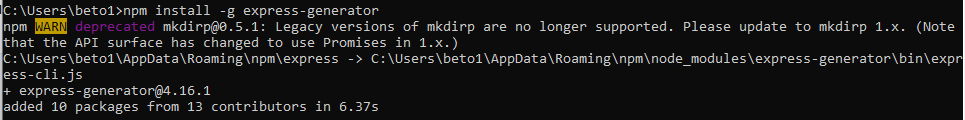
- O Express é um web framework.

- Express Generator é o scaffold de Express.

## Instalando o pacote express-generator

No terminal do DOS, entre com o seguinte comando:

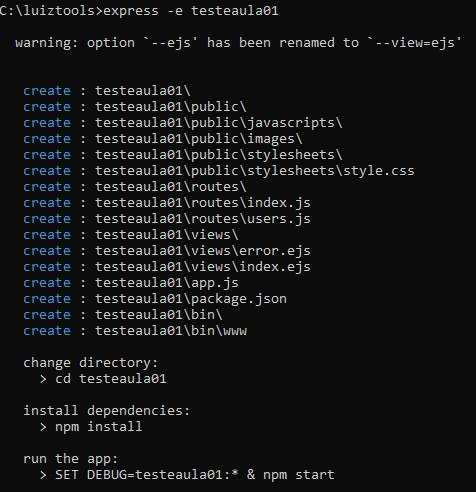
npm install -g express-generator



- Instalando o express-generator de maneira global é habilitado o comando express no console.

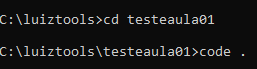
## Criando o scaffold

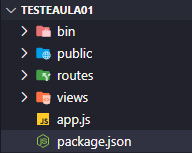
express -e testeaula01



cd testeaula01

code .





**package.json**

{

"name": "testeaula01",

"version": "0.0.0",

"private": true,

"scripts": {

"start": "node ./bin/www"

},

"dependencies": {

"cookie-parser": "~1.4.4",

"debug": "~2.6.9",

"ejs": "~2.6.1",

"express": "~4.16.1",

"http-errors": "~1.6.3",

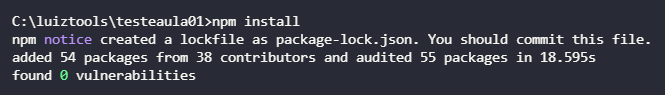
"morgan": "~1.9.1"

}

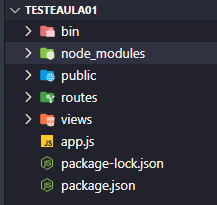
}

- No terminal do Visual Studio Code, entre com o comando:

npm install

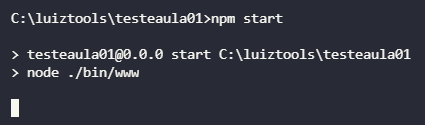


- É criada uma pasta chamada "node\_modules" com todas as dependências e não deve ser subida ao servidor.



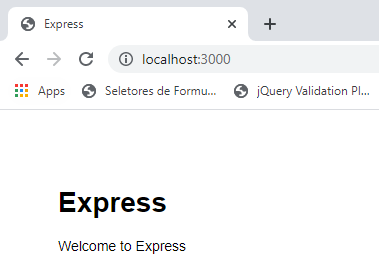
## Rodando o projeto

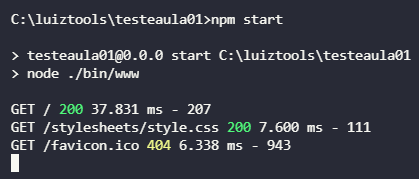
npm start



- No browser:

localhost:3000





# Aula 12 - ExpressJS na Prática - Parte 1

**bin\www**

#!/usr/bin/env node

/\*\*

\* Module dependencies.

\*/

var app = require('../app');

var debug = require('debug')('testeaula01:server');

var http = require('http');

/\*\*

\* Get port from environment and store in Express.

\*/

var port = normalizePort(process.env.PORT || '3000');

app.set('port', port);

/\*\*

\* Create HTTP server.

\*/

var server = http.createServer(app);

/\*\*

\* Listen on provided port, on all network interfaces.

\*/

server.listen(port);

server.on('error', onError);

server.on('listening', onListening);

/\*\*

\* Normalize a port into a number, string, or false.

\*/

function normalizePort(val) {

var port = parseInt(val, 10);

if (isNaN(port)) {

// named pipe

return val;

}

if (port >= 0) {

// port number

return port;

}

return false;

}

/\*\*

\* Event listener for HTTP server "error" event.

\*/

function onError(error) {

if (error.syscall !== 'listen') {

throw error;

}

var bind = typeof port === 'string'

? 'Pipe ' + port

: 'Port ' + port;

// handle specific listen errors with friendly messages

switch (error.code) {

case 'EACCES':

console.error(bind + ' requires elevated privileges');

process.exit(1);

break;

case 'EADDRINUSE':

console.error(bind + ' is already in use');

process.exit(1);

break;

default:

throw error;

}

}

/\*\*

\* Event listener for HTTP server "listening" event.

\*/

function onListening() {

var addr = server.address();

var bind = typeof addr === 'string'

? 'pipe ' + addr

: 'port ' + addr.port;

debug('Listening on ' + bind);

}

**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();

// 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);

// 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;

# Aula 13 - ExpressJS na Prática - Parte 2

**routes\index.js**

var express = require('express');

var router = express.Router();

/\* GET home page. \*/

router.get('/', function(req, res, next) {

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

});

module.exports = router;

**views\index.ejs**

<!DOCTYPE html>

<html>

<head>

<title><%= title %></title>

<link rel='stylesheet' href='/stylesheets/style.css' />

</head>

<body>

<h1><%= title %></h1>

<p>Welcome to <%= title %></p>

</body>

</html>

**public\stylesheets\style.css**

body {

padding: 50px;

font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;

}

a {

color: #00B7FF;

}

# Aula 14 - ExpressJS na Prática - Parte 3

**views\teste.ejs**

<!DOCTYPE html>

<html>

<head>

<title>

<%= title %>

</title>

<link rel='stylesheet' href='/stylesheets/style.css' />

</head>

<body>

<h1><%= title %></h1>

<p>Welcome to <%= title %></p>

<p>Teste de Node.js! O número do model é <%= numero %></p>

</body>

</html>

**routes\teste.js**

var express = require('express');

var router = express.Router();

/\* GET home page. \*/

router.get('/', function(req, res, next) {

res.render('teste', { title: 'Node is Super Cool', numero: 10 });

});

module.exports = router;

**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 testeRouter = require('./routes/teste');

var app = 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);

app.use('/teste', testeRouter);

// 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

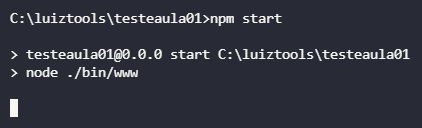
res.status(err.status || 500);

res.render('error');

});

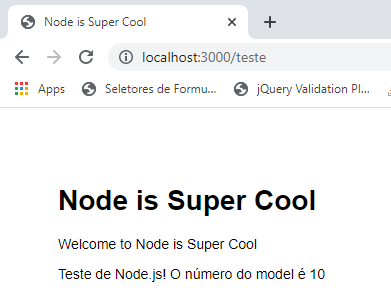
module.exports = app;

npm start



- No browser:

localhost:3000/teste



# Aula 15 - O que é MongoDB

- MongoDB é orientado a documentos.

- É um banco de dados não-relacional.

Características:

* BSON
* sem JOIN's e FK's
* documentos independentes
* schemaless
* usa JavaScript para consulta

ACID:

* Atomicidade
* Consistência
* Isolamento
* Durabilidade

- Em vez de tabelas, no MongoDB temos coleções de documentos.

# Aula 16 - Quando usar MongoDB

Cenários de uso do MongoDB:

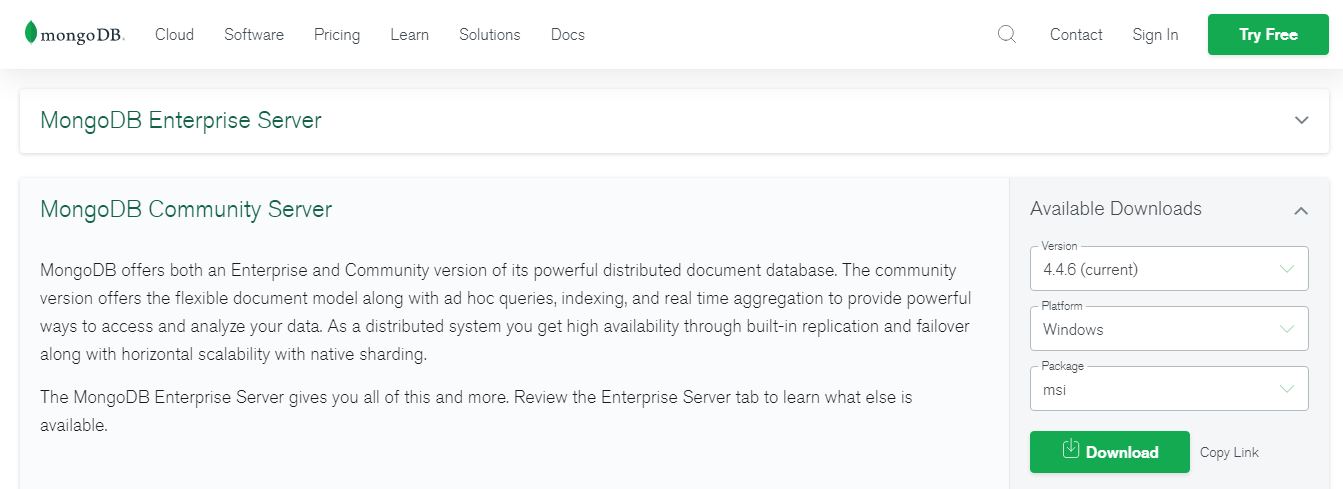
* Big Data
* dados desestruturados ou com estrutura dinâmica
* JavaScript Full-Stack
* Cache

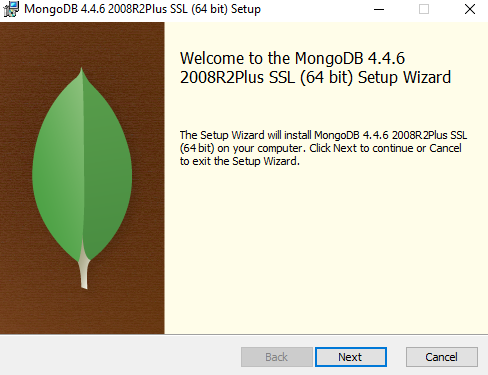
- É muito usado em ecommerce (em catálogo de produtos).

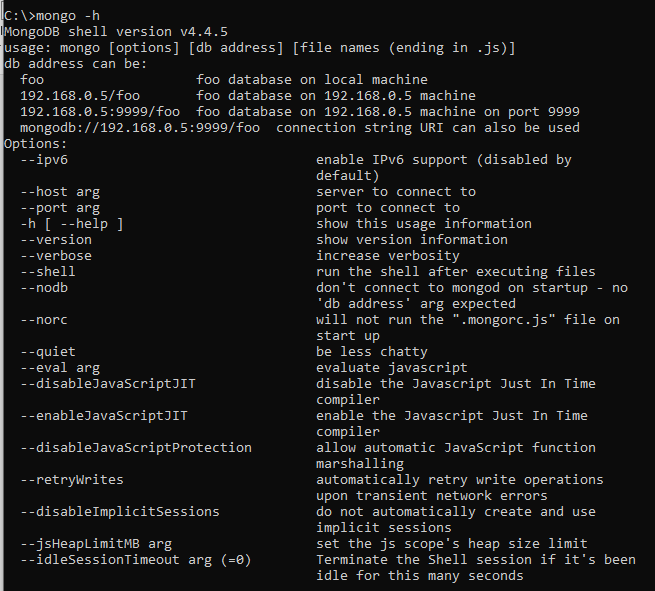
# Aula 17 - Como rodar servidor MongoDB

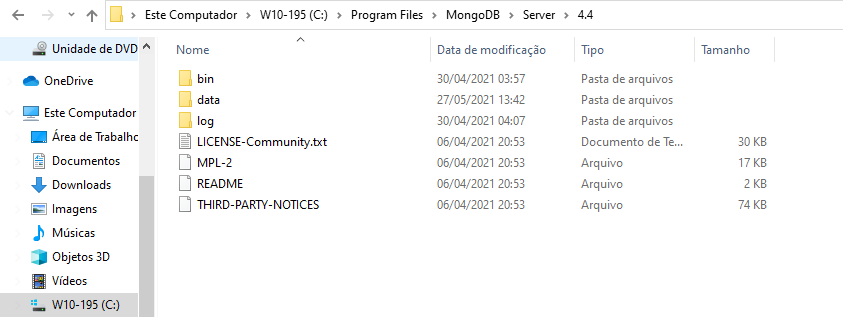
## Instalação

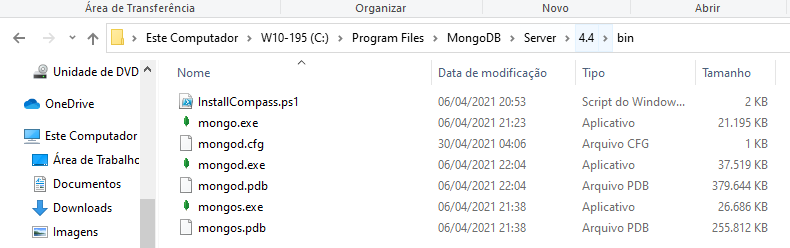
<https://www.mongodb.com/try/download/community>

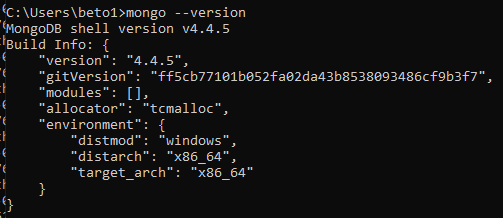










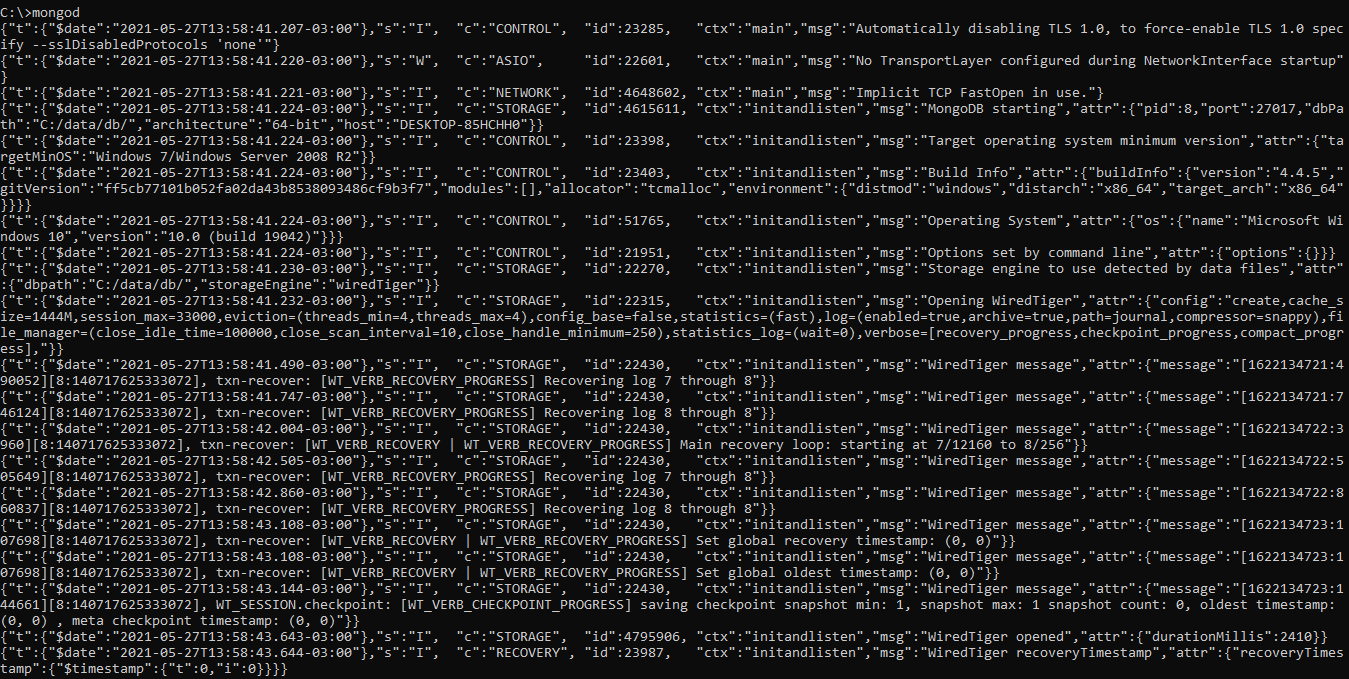


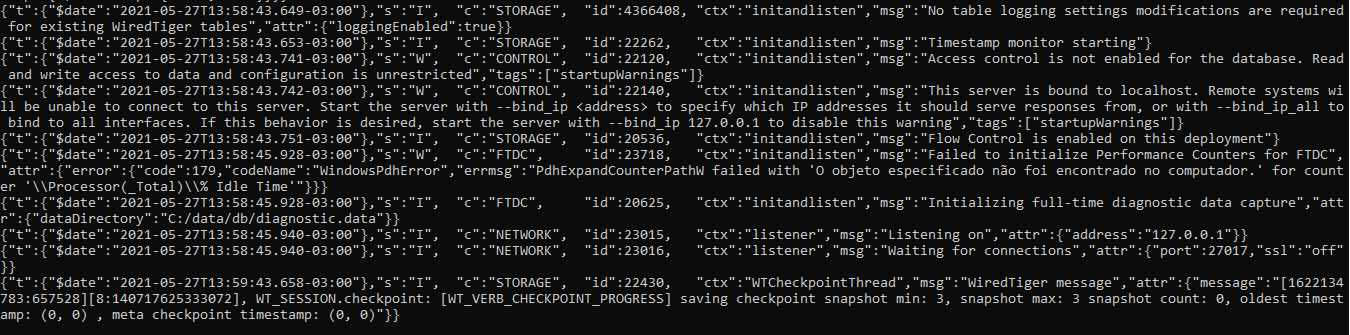
- Em C:/ crie uma pasta chamada "data":

### Subindo o servidor do MongoDB

- No terminal do DOS, entre com o comando:

mongod





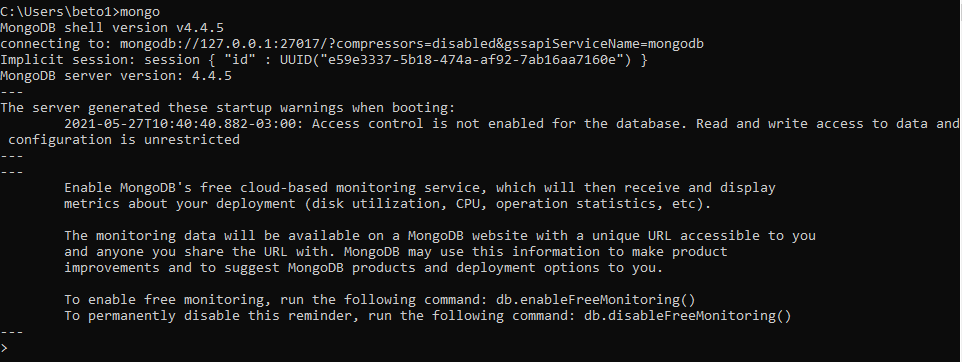
- Indica que foi feita a conexão:

"Waiting for connections","attr":{"port": 27017, "ssl":"off"}

# Aula 18 - CRUD MongoDB no Terminal - Parte 1

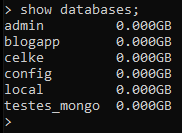
- Em um outro terminal do DOS, entre com o client do MongoDB com o comando:

mongo



## Comandos básicos

show databases;



use aula02;



- Como o database aula02 não existe, ele será criado, porém só será exibido quando conter dados.

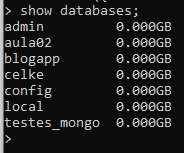
- No MongoDB ao invés de tabelas, existem as collections.

- Inserindo um documento no database aula02:

db.clientes.insert({nome: "Luiz", idade: 29, cidade: "Gravataí"});



show databases;



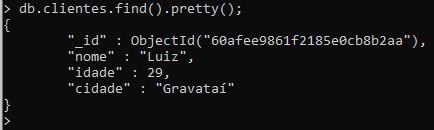
db.clientes.count();



db.clientes.find();



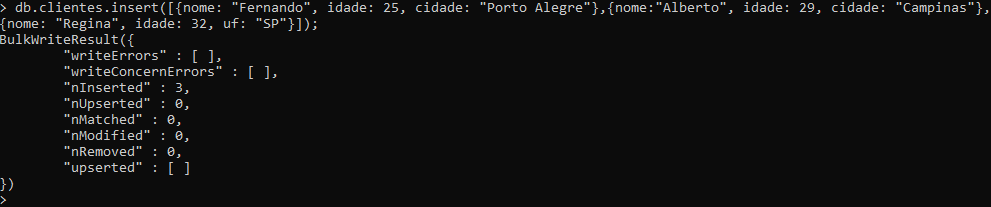
db.clientes.find().pretty();

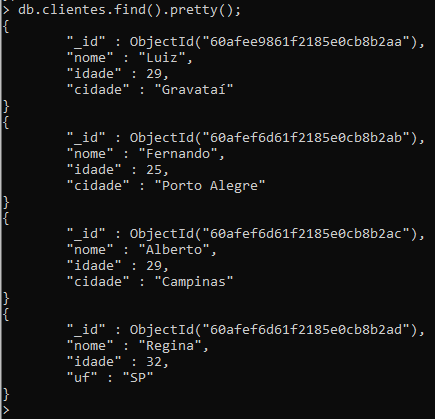


# Aula 19 - CRUD MongoDB no Terminal - Parte 2

## Adicionando clientes

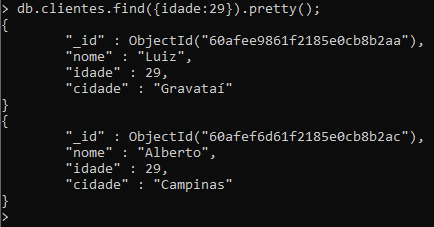
db.clientes.insert([{nome: "Fernando", idade: 25, cidade: "Porto Alegre"},{nome:"Alberto", idade: 29, cidade: "Campinas"},{nome: "Regina", idade: 32, uf: "SP"}]);



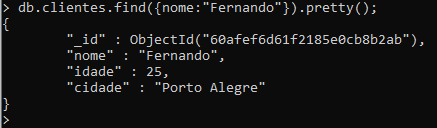


## Consultas

db.clientes.find({idade:29}).pretty();



db.clientes.find({nome:"Fernando"}).pretty();



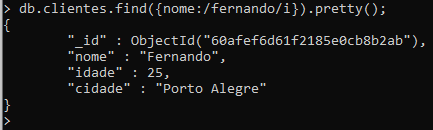
- As strings do MongoDB são case-sensitive.

db.clientes.find({nome:"fernando"}).pretty();



### Busca por string com expressão regular

db.clientes.find({nome:/fernando/i}).pretty();



## Update

db.clientes.update({cidade:"Campinas"}, {nome:"Ricardo"});



db.clientes.find().pretty();



- Faz alterações no documento inteiro.

db.clientes.update({nome:"Ricardo"}, {nome:"Ricardo", idade: 29, cidade:"Campinas",uf:"SP"});

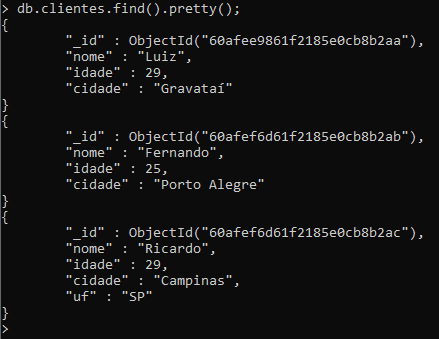




## Removendo um documento

db.clientes.remove({\_id: ObjectId("60afef6d61f2185e0cb8b2ad")});

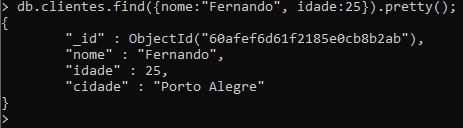




# Aula 20 - Consultas com filtros em MongoDB

### Busca por mais de um parâmetro

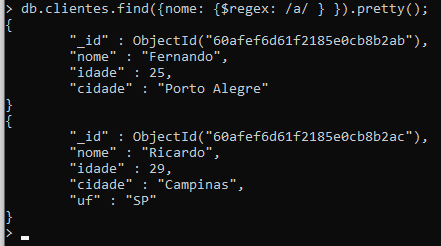
db.clientes.find({nome:"Fernando", idade:25}).pretty();



## Busca usando expressões regulares

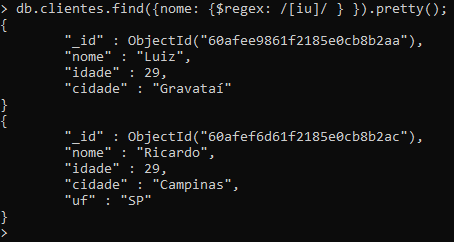
### Busca por nomes que contenham a letra "a"

db.clientes.find({nome: {$regex: /a/ } }).pretty();

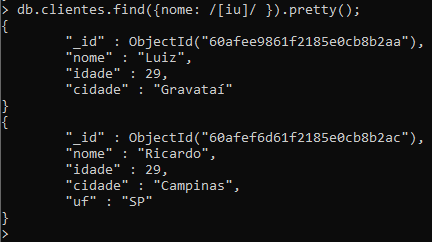


### Busca por nomes que contenham a letra "i" ou "u"

db.clientes.find({nome: {$regex: /[iu]/ } }).pretty();

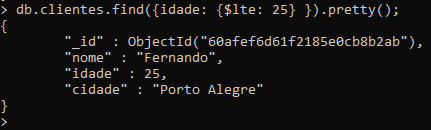


db.clientes.find({nome: /[iu]/ }).pretty();



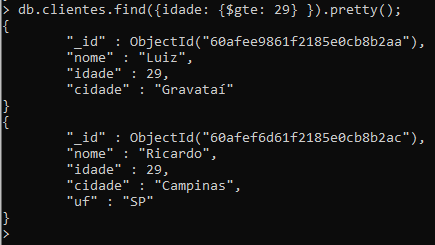
### Busca por idades menores ou iguais a 25

db.clientes.find({idade: {$lte: 25} }).pretty();



### Busca por idades maiores ou iguais a 29

db.clientes.find({idade: {$gte: 29} }).pretty();



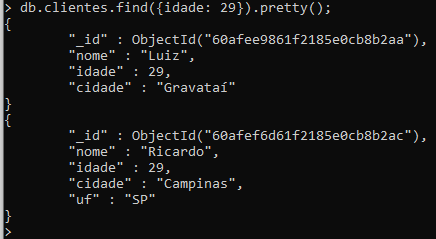
### Busca por idades maiores que 29

db.clientes.find({idade: {$gt: 29} }).pretty();



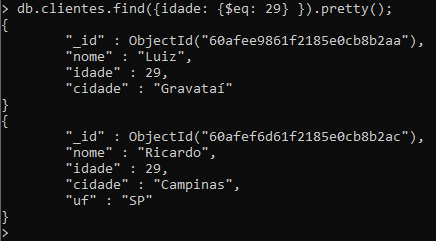
### Busca por idades iguais a 29

db.clientes.find({idade: 29}).pretty();



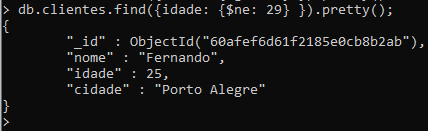
ou:

db.clientes.find({idade: {$eq: 29} }).pretty();



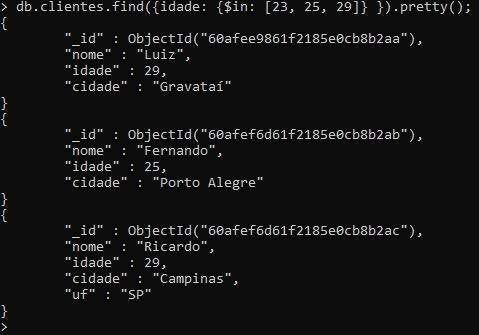
### Busca por idades diferentes de 29

db.clientes.find({idade: {$ne: 29} }).pretty();



### Busca por idades com valores especificados em um array

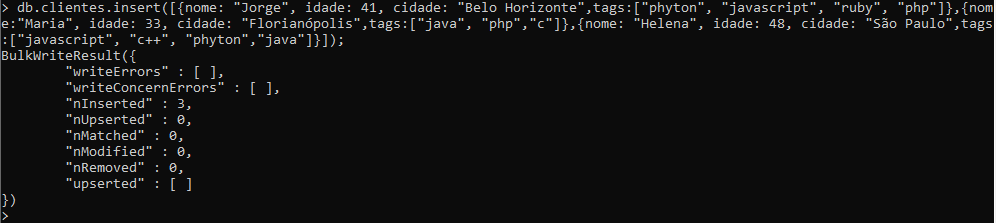
db.clientes.find({idade: {$in: [23, 25, 29]} }).pretty();

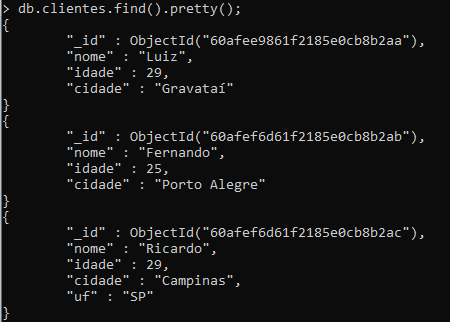


## tags

- Insira os seguintes documentos:

db.clientes.insert([{nome: "Jorge", idade: 41, cidade: "Belo Horizonte",tags:["phyton", "javascript", "ruby", "php"]},{nome:"Maria", idade: 33, cidade: "Florianópolis",tags:["java", "php","c"]},{nome: "Helena", idade: 48, cidade: "São Paulo",tags:["javascript", "c++", "phyton","java"]}]);







### Busca por clientes que contenham pelo menos uma das tags especificadas

db.clientes.find({tags: {$in: ["java", "php"]} }).pretty();



### Busca por clientes que contenham todas as tags especificadas

db.clientes.find({tags: {$all: ["javascript", "phyton"]} }).pretty();



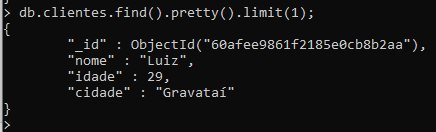
### Consulta de clientes ignorando os quatro primeiros documentos

db.clientes.find().pretty().skip(4);



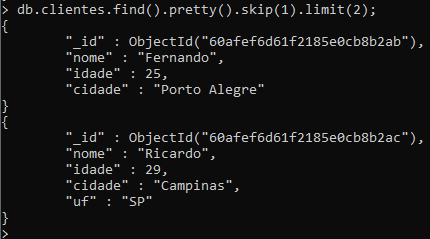
### Consulta de apenas um documento (o primeiro)

db.clientes.find().pretty().limit(1);



### Exibindo apenas o segundo e terceiro documentos

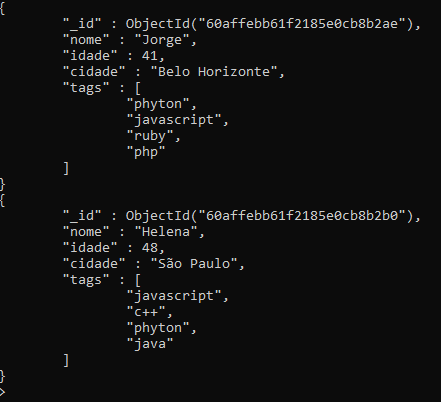
db.clientes.find().pretty().skip(1).limit(2);



### Exibindo clientes por ordem crescente de idade

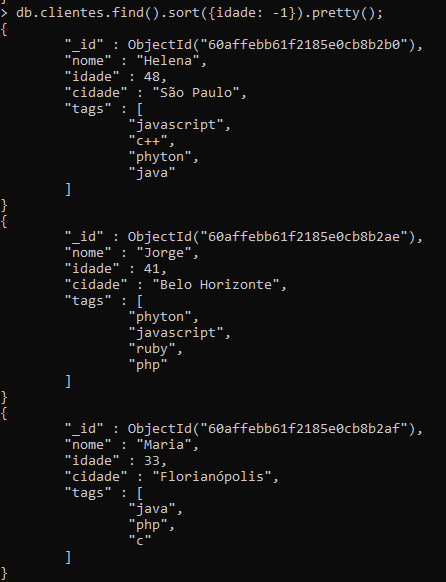
db.clientes.find().sort({idade: 1}).pretty();

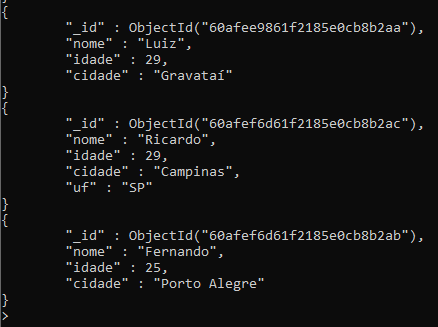




### Exibindo clientes por ordem decrescente de idade

db.clientes.find().sort({idade: -1}).pretty();





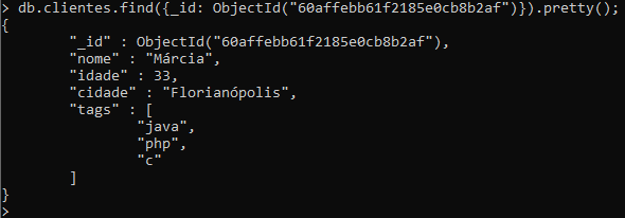
# Aula 21 - Update com operadores no MongoDB

## Atualizando um campo ($set)

db.clientes.updateOne( {\_id: ObjectId("60affebb61f2185e0cb8b2af")}, {$set:{nome:"Márcia"}} );



db.clientes.find({\_id: ObjectId("60affebb61f2185e0cb8b2af")}).pretty();

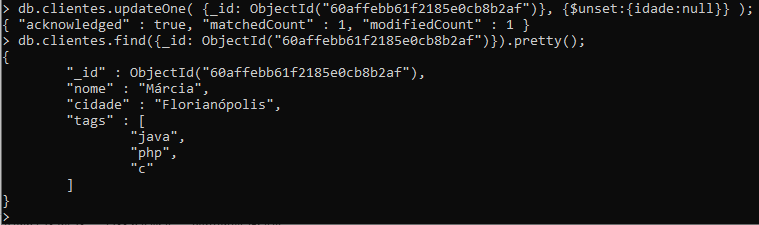


## Eliminando um campo de um documento ($unset)

db.clientes.updateOne( {\_id: ObjectId("60affebb61f2185e0cb8b2af")}, {$unset:{idade:null}} );



db.clientes.find({\_id: ObjectId("60affebb61f2185e0cb8b2af")}).pretty();

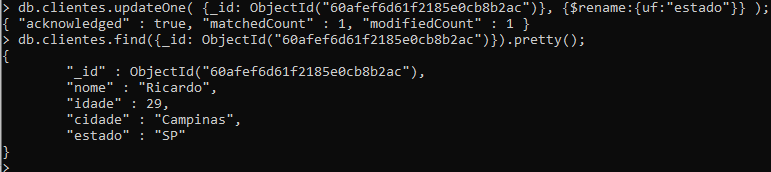


## Renomeando um campo ($rename)

db.clientes.updateOne( {\_id: ObjectId("60afef6d61f2185e0cb8b2ac")}, {$rename:{uf:"estado"}} );



db.clientes.find({\_id: ObjectId("60afef6d61f2185e0cb8b2ac")}).pretty();

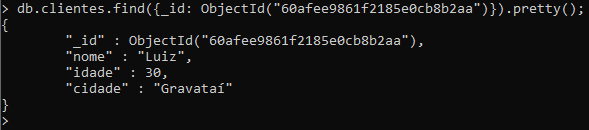


## Incrementando a idade

db.clientes.updateOne( {\_id: ObjectId("60afee9861f2185e0cb8b2aa")}, {$inc:{idade:1}} );



db.clientes.find({\_id: ObjectId("60afee9861f2185e0cb8b2aa")}).pretty();

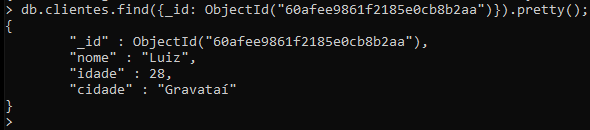


## Decrementando a idade

db.clientes.updateOne( {\_id: ObjectId("60afee9861f2185e0cb8b2aa")}, {$inc:{idade:-2}} );



db.clientes.find({\_id: ObjectId("60afee9861f2185e0cb8b2aa")}).pretty();

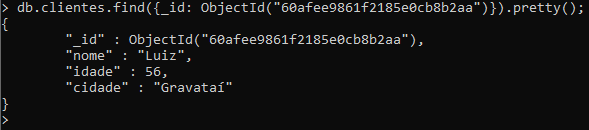


## Multiplicando a idade por 2

db.clientes.updateOne( {\_id: ObjectId("60afee9861f2185e0cb8b2aa")}, {$mul:{idade:2}} );



db.clientes.find({\_id: ObjectId("60afee9861f2185e0cb8b2aa")}).pretty();

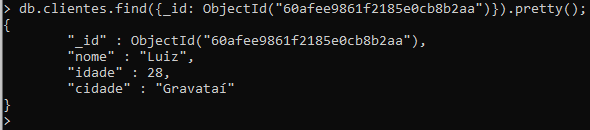


## Dividindo a idade por 2

db.clientes.updateOne( {\_id: ObjectId("60afee9861f2185e0cb8b2aa")}, {$mul:{idade:1/2}} );



db.clientes.find({\_id: ObjectId("60afee9861f2185e0cb8b2aa")}).pretty();

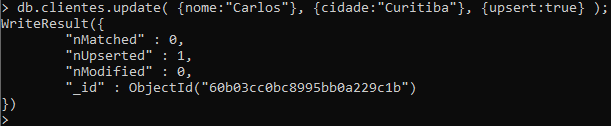


## Update com Insert (upsert)

Atualiza um campo se o mesmo existir. Caso o campo não exista, ele será criado.

db.clientes.update( {nome:"Carlos"}, {cidade:"Curitiba"}, {upsert:true} );



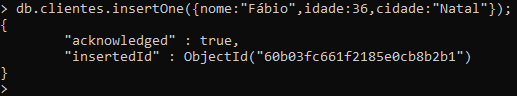


db.clientes.find({cidade:"Curitiba"}).pretty();

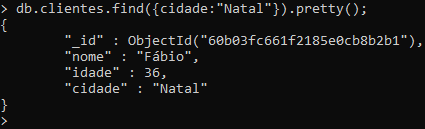


## Inserindo um documento e retornando o id (insertOne)

db.clientes.insertOne({nome:"Fábio",idade:36,cidade:"Natal"});



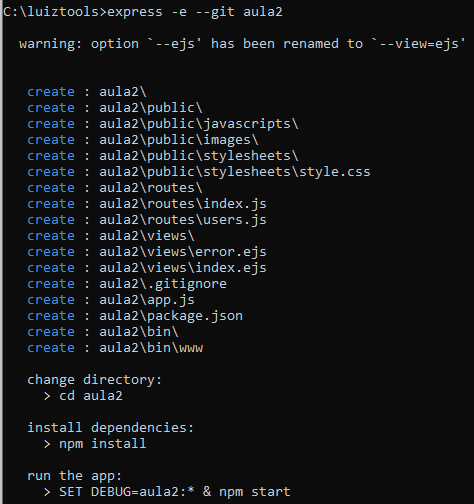
db.clientes.find({cidade:"Natal"}).pretty();



# Aula 22 - Nodejs conectando MongoDB

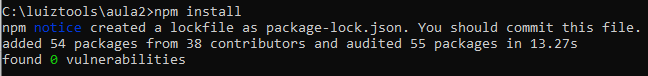
- Inicialmente, crie um novo projeto chamado "aula2". No terminal do DOS, entre com o comando:

express -e --git aula2



cd aula2

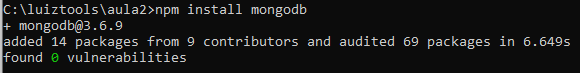
npm install



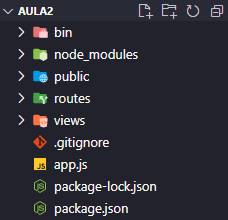
## Instalando o driver mongodb

- Entre com o comando:

npm install mongodb



code .



**package.json**

{

"name": "aula2",

"version": "0.0.0",

"private": true,

"scripts": {

"start": "node ./bin/www"

},

"dependencies": {

"cookie-parser": "~1.4.4",

"debug": "~2.6.9",

"ejs": "~2.6.1",

"express": "~4.16.1",

"http-errors": "~1.6.3",

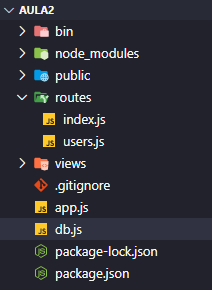
"mongodb": "^3.6.9",

"morgan": "~1.9.1"

}

}

- No diretório raiz do projeto crie um arquivo chamado "db.js":



**db.js**

const mongoClient = require("mongodb").MongoClient;

mongoClient.connect("mongodb://localhost:27017",

{useUnifiedTopology: true},

(error, connection) => {

if(error){

return console.log(error);

} else {

global.connection = connection.db("aula2");

console.log("Connected!");

}

} );

module.exports = {};

**routes\index.js**

const express = require('express');

const router = express.Router();

const db = require("../db");

/\* GET home page. \*/

router.get('/', function(req, res, next) {

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

});

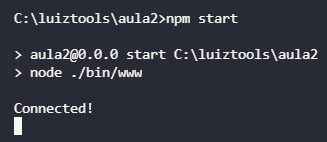
module.exports = router;

- Suba o servidor:

mongod

- No terminal do Visual Studio Code, entre com o comando:

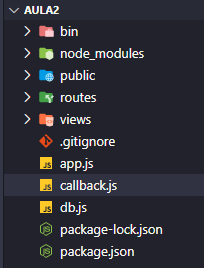
npm start



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

- O Node trabalha de forma assíncrona.

## Callbacks



- fs (filesystem) é o módulo para sistema de arquivos.

- O sistema de arquivo executa leituras e escritas em disco. Trata-se de uma operação lenta (especialmente escrever em disco).

- Enviar pacotes pela rede é outra operação lenta.

**callback.js**

const fs = require('fs');

var texto = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin vel mauris diam. In hac habitasse platea dictumst. Nam at lacinia eros. Proin ut facilisis tellus. Phasellus eget scelerisque ipsum. Donec hendrerit neque id consequat pretium. Aenean vel enim dignissim, venenatis risus auctor, semper lectus. Aenean massa elit, congue quis enim nec, fermentum aliquet risus. Mauris at enim et felis lobortis sollicitudin.';

// call back

console.log("\nPasso 1\n");

function funcAssinc(){

fs.writeFile('teste.txt', `${texto}`, () => {

console.log(texto);

});

}

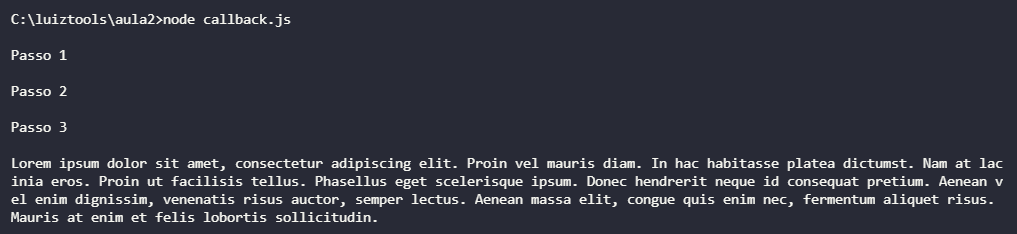
funcAssinc();

console.log("Passo 2\n");

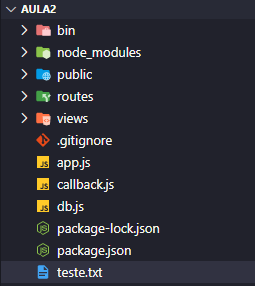
console.log("Passo 3\n");

- O Node envia esse fluxo de execução para ser executado em background e continua a executar as demais linhas de programação do código.

node index.js



- O conteúdo da variável texto é gravado em um arquivo chamado teste.txt:



### Callback hell

- O problema (em termos de estruturação de código) é quando dentro do callback existem outras funções lentas. Colocando um callback dentro de outro e assim por diante, o código fica longo e confuso.

**callback.js**

const fs = require('fs');

var texto = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin vel mauris diam. In hac habitasse platea dictumst. Nam at lacinia eros. Proin ut facilisis tellus. Phasellus eget scelerisque ipsum. Donec hendrerit neque id consequat pretium. Aenean vel enim dignissim, venenatis risus auctor, semper lectus. Aenean massa elit, congue quis enim nec, fermentum aliquet risus. Mauris at enim et felis lobortis sollicitudin.';

var texto2 = 'Aenean non est arcu. Curabitur volutpat pellentesque dolor eu euismod. Etiam quis vulputate nunc. Quisque ac commodo tortor, vel fermentum metus. Aliquam auctor, quam sit amet rutrum feugiat, massa elit rhoncus felis, quis dictum ante mi in velit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam erat volutpat. Ut consectetur augue ullamcorper felis consequat vulputate. Phasellus vehicula, massa et hendrerit congue, ligula lacus consequat augue, bibendum maximus ipsum felis vel ex. Aliquam erat volutpat. Curabitur pulvinar sagittis ligula, a finibus tellus tempor et. Maecenas leo mi, iaculis vitae porttitor vel, tincidunt et erat. Cras non scelerisque dui.';

var texto3 = 'In et consequat justo. Nulla vel porta eros, non consectetur ligula. Pellentesque arcu massa, interdum a elementum et, ultricies in felis. Duis laoreet mi turpis, eu luctus urna viverra sed. Donec quis cursus nibh. Suspendisse potenti. Vivamus laoreet nisl sed augue mattis, efficitur semper diam tincidunt. Nullam iaculis, metus quis varius aliquam, velit eros rutrum erat, et finibus nunc quam at ex. Suspendisse cursus ullamcorper erat eget rhoncus. Ut eu sollicitudin dui. Donec auctor eleifend dolor pharetra tempus. Phasellus ac commodo ipsum.';

// call back

console.log("\nPasso 1\n");

function funcaoAssinc(){

fs.writeFile('teste.txt', `${texto}`, () => {

console.log(texto);

console.log("");

fs.writeFile('teste2.txt', `${texto2}`, () => {

console.log(texto2);

console.log("");

fs.writeFile('teste3.txt', `${texto3}`, () => {

console.log(texto3);

console.log("");

})

});

});

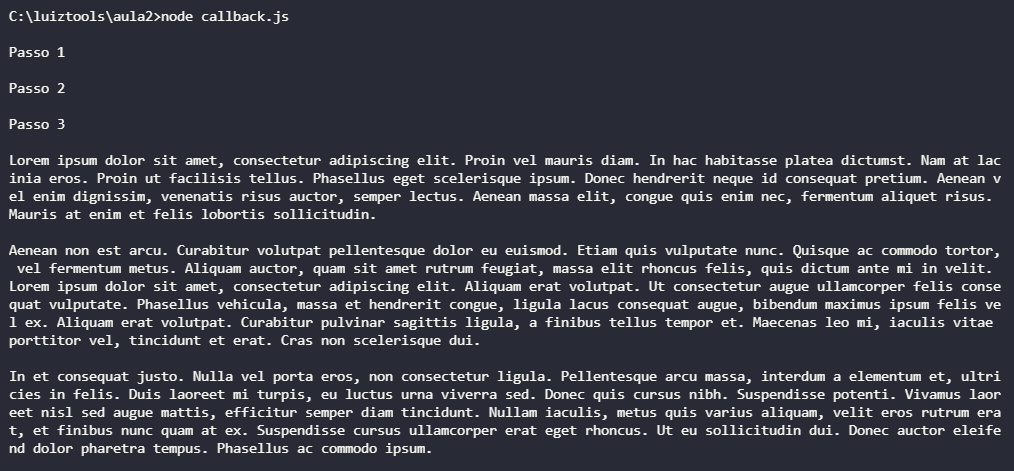
}

funcaoAssinc();

console.log("Passo 2\n");

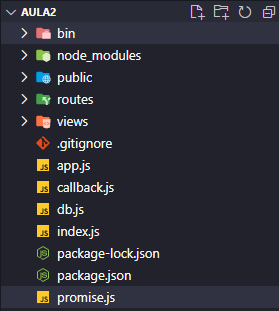
console.log("Passo 3\n");

node callback.js





## Promise



- É uma forma mais elegante de se escrever um callback;

**promise.js**

// promise

console.log("\nPasso 1\n");

function fazRequisicao(resolver = true) {

return new Promise((resolve, reject) => {

setTimeout(() => {

if (!resolver) {

reject("Deu erro!");

}

resolve("Promise resolvida!");

}, 5000);

});

}

fazRequisicao(false)

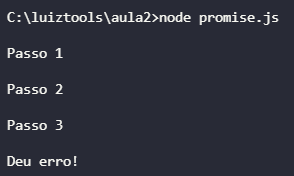
.then(console.log)

.catch(console.error);

console.log("Passo 2\n");

console.log("Passo 3\n");

node promise.js



**promise.js**

// promise

console.log("\nPasso 1\n");

function fazRequisicao(resolver = true) {

return new Promise((resolve, reject) => {

setTimeout(() => {

if (!resolver) {

reject("Deu erro!");

}

resolve("Promise resolvida!");

}, 5000);

});

}

fazRequisicao()

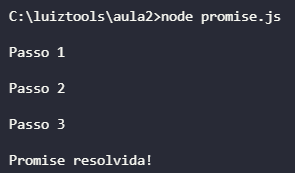
.then(console.log)

.catch(console.error);

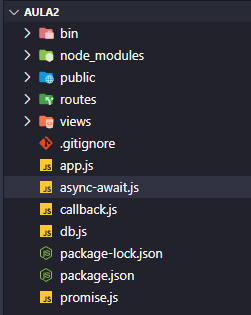
console.log("Passo 2\n");

console.log("Passo 3\n");

node promise.js



## Async/Await



**async-await.js**

const fs = require('fs');

var texto = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin vel mauris diam. In hac habitasse platea dictumst. Nam at lacinia eros. Proin ut facilisis tellus. Phasellus eget scelerisque ipsum. Donec hendrerit neque id consequat pretium. Aenean vel enim dignissim, venenatis risus auctor, semper lectus. Aenean massa elit, congue quis enim nec, fermentum aliquet risus. Mauris at enim et felis lobortis sollicitudin.';

var texto2 = 'Aenean non est arcu. Curabitur volutpat pellentesque dolor eu euismod. Etiam quis vulputate nunc. Quisque ac commodo tortor, vel fermentum metus. Aliquam auctor, quam sit amet rutrum feugiat, massa elit rhoncus felis, quis dictum ante mi in velit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam erat volutpat. Ut consectetur augue ullamcorper felis consequat vulputate. Phasellus vehicula, massa et hendrerit congue, ligula lacus consequat augue, bibendum maximus ipsum felis vel ex. Aliquam erat volutpat. Curabitur pulvinar sagittis ligula, a finibus tellus tempor et. Maecenas leo mi, iaculis vitae porttitor vel, tincidunt et erat. Cras non scelerisque dui.';

var texto3 = 'In et consequat justo. Nulla vel porta eros, non consectetur ligula. Pellentesque arcu massa, interdum a elementum et, ultricies in felis. Duis laoreet mi turpis, eu luctus urna viverra sed. Donec quis cursus nibh. Suspendisse potenti. Vivamus laoreet nisl sed augue mattis, efficitur semper diam tincidunt. Nullam iaculis, metus quis varius aliquam, velit eros rutrum erat, et finibus nunc quam at ex. Suspendisse cursus ullamcorper erat eget rhoncus. Ut eu sollicitudin dui. Donec auctor eleifend dolor pharetra tempus. Phasellus ac commodo ipsum.';

console.log("\nPasso 1\n");

async function funcaoAssinc(){

try {

await fs.writeFile('teste.txt', `${texto}`, () => {

console.log(texto);

console.log("");

});

await fs.writeFile('teste2.txt', `${texto2}`, () => {

console.log(texto2);

console.log("");

});

await fs.writeFile('teste3.txt', `${texto3}`, () => {

console.log(texto3);

console.log("");

});

} catch (error) {

console.log(error);

}

}

funcaoAssinc();

console.log("Passo 2\n");

console.log("Passo 3\n");

node async-await.js

