

Desenvolvimento NodeJS

FATEC PROF. JOSÉ CAMARGO – FATEC JALES
TÓPICOS ESPECIAIS EM SISTEMAS PARA INTERNET III
PROF. Me. TIAGO RIBEIRO CARNEIRO

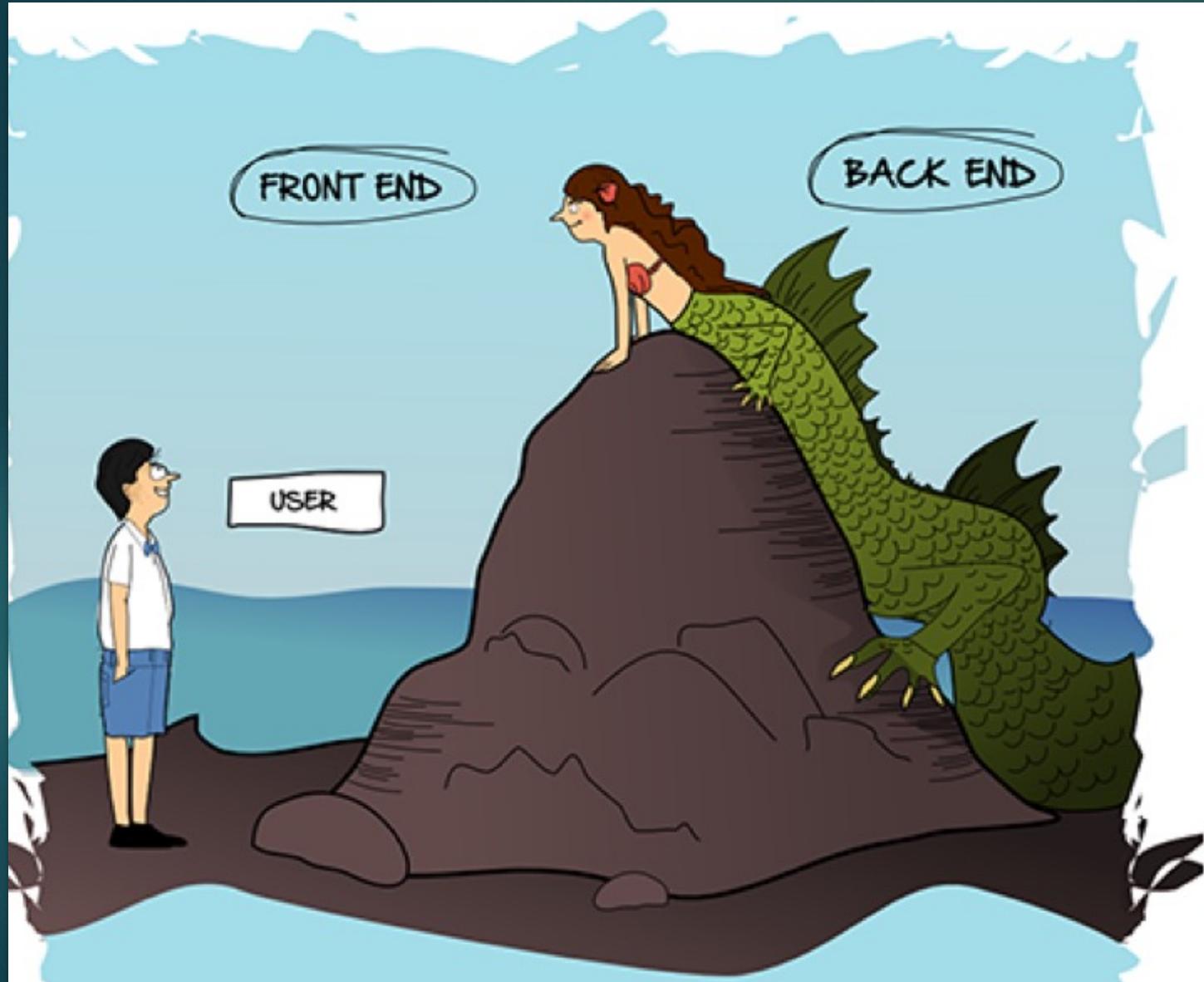
Ementa

- ▶ Arquitetura de uma aplicação web.
- ▶ Fundamentos da plataforma Node.js.
- ▶ Gerenciamento de pacotes com NPM.
- ▶ Sistema de módulos.
- ▶ Programação assíncrona.
- ▶ Framework Express. Acesso a banco de dados.

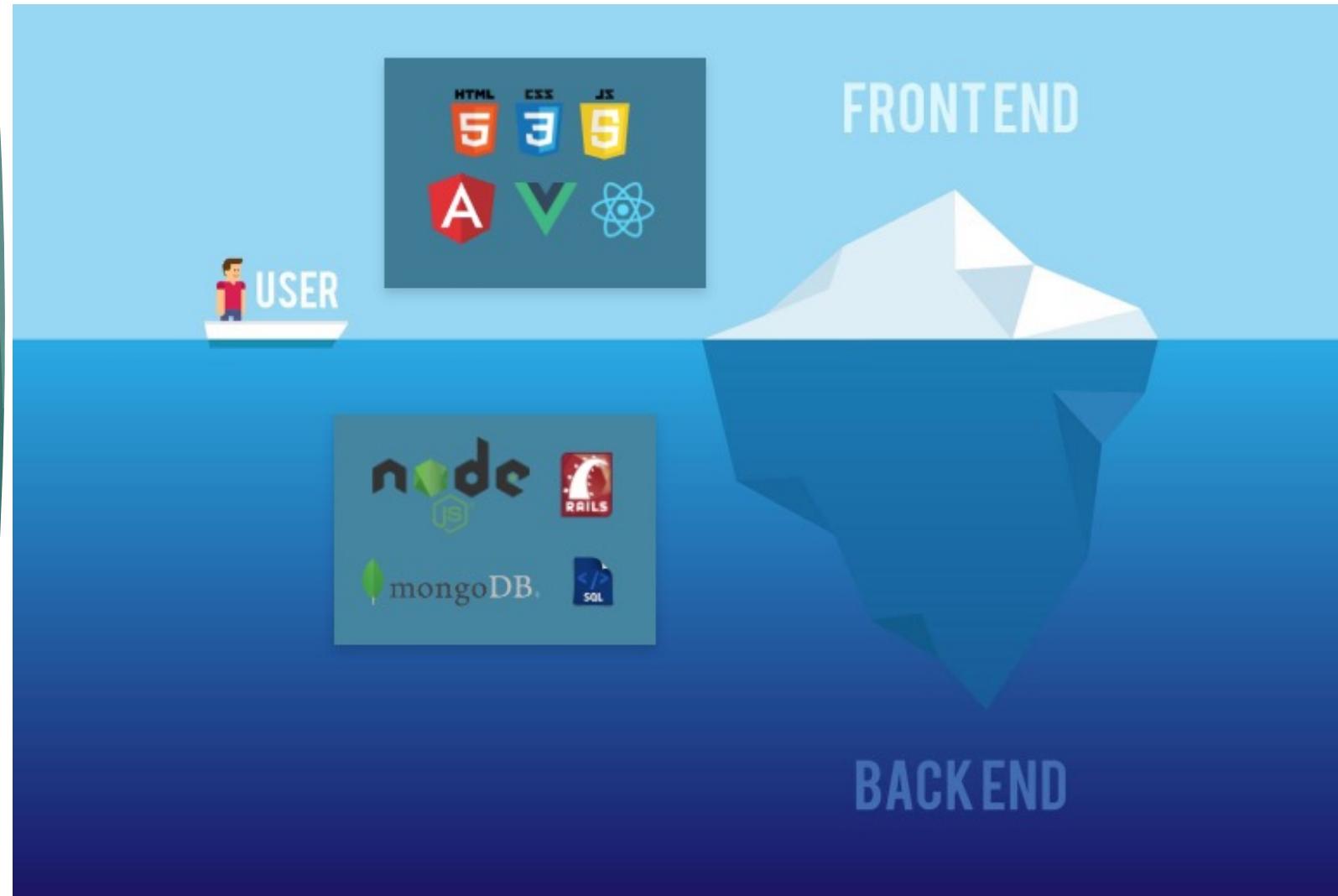
Carga horária: 24h/a

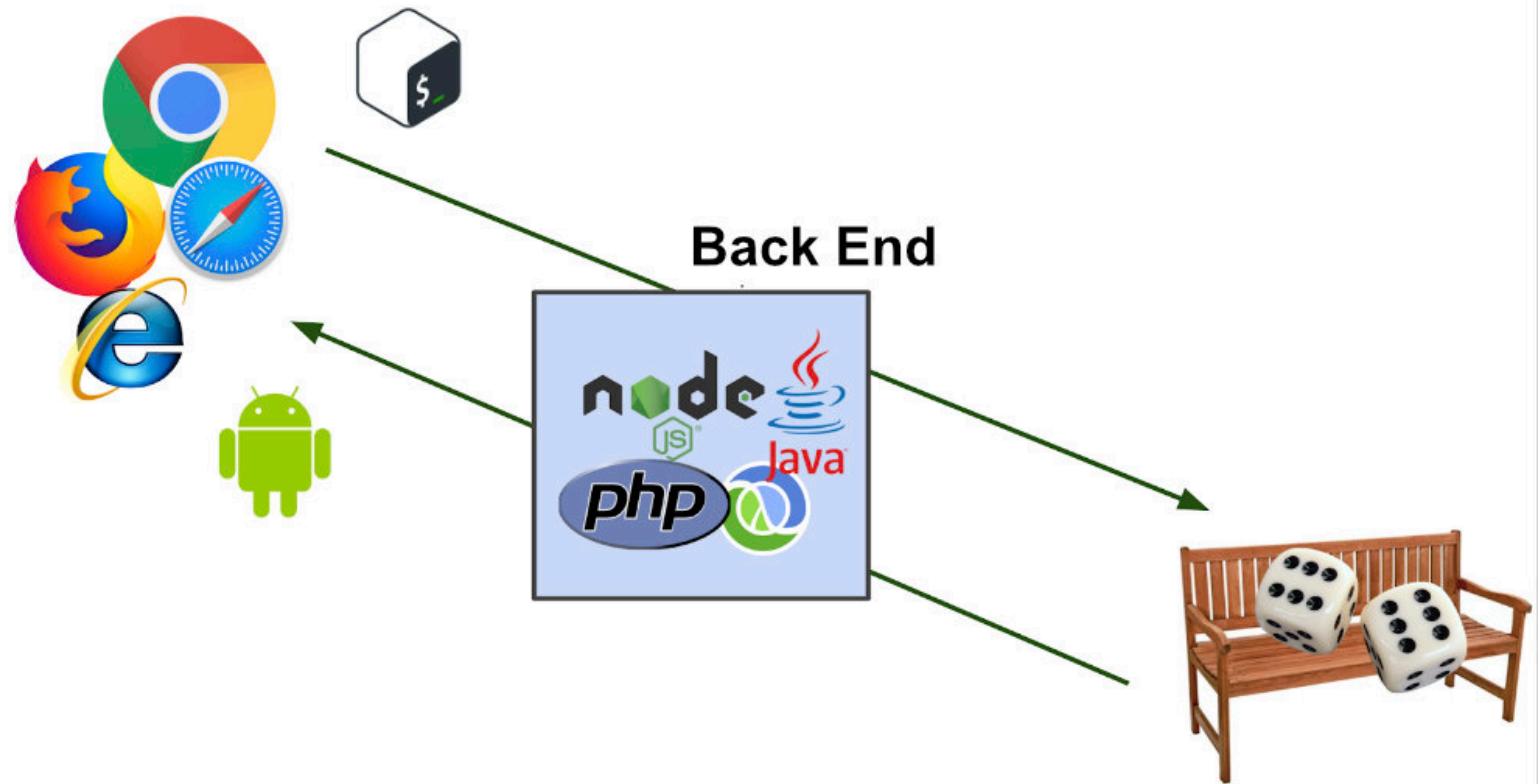
Introdução

Front-End vs Back-End



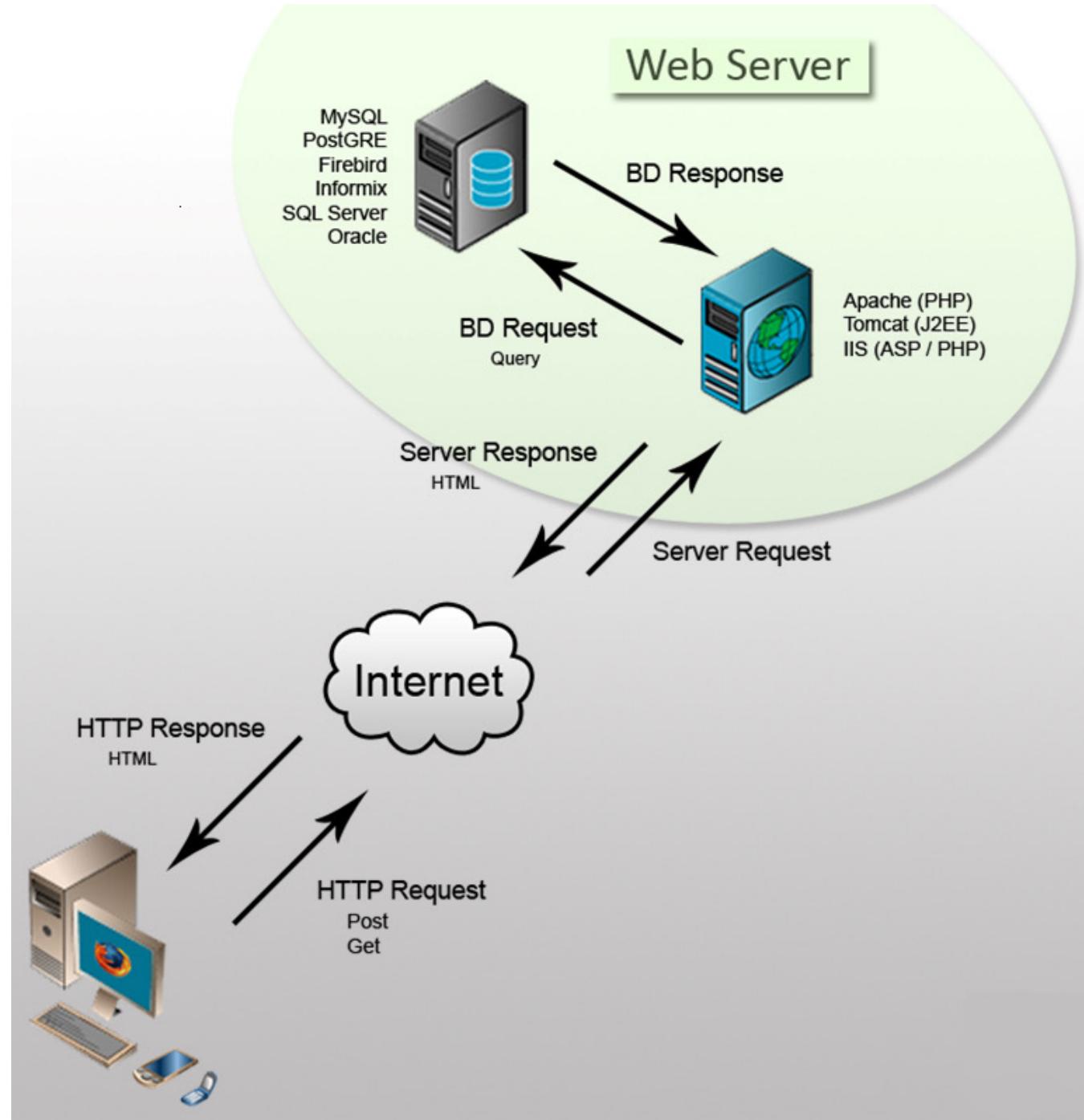
Front-End + Back-End





Front-End + Back- End + BD

Arquitetura de aplicações web





Problemas das
arquiteturas
bloqueantes...

Vejamos no detalhe:

- ▶ Imagine um sistema com um 1 milhão de acessos...
- ▶ Para cada acesso, é criada uma thread, para cuidar dessa requisição...
- ▶ E... Essa thread permanece travada, até que a requisição termine...
- ▶ Agora, imagine isso, em um sistema com milhões de acesso, utilizados em larga escala!!! É ai que está a **ZICA!!!**
- ▶ **E quando o número de threads que o SO possui chega ao limite?**
- ▶ As requisições são enfileiradas, gerando um gargalo nas respostas ao cliente pois, é necessário que uma requisição termine para que a próxima seja iniciada.

- ▶ **Ou seja, trata-se de uma Arquitetura Síncrona!!!**

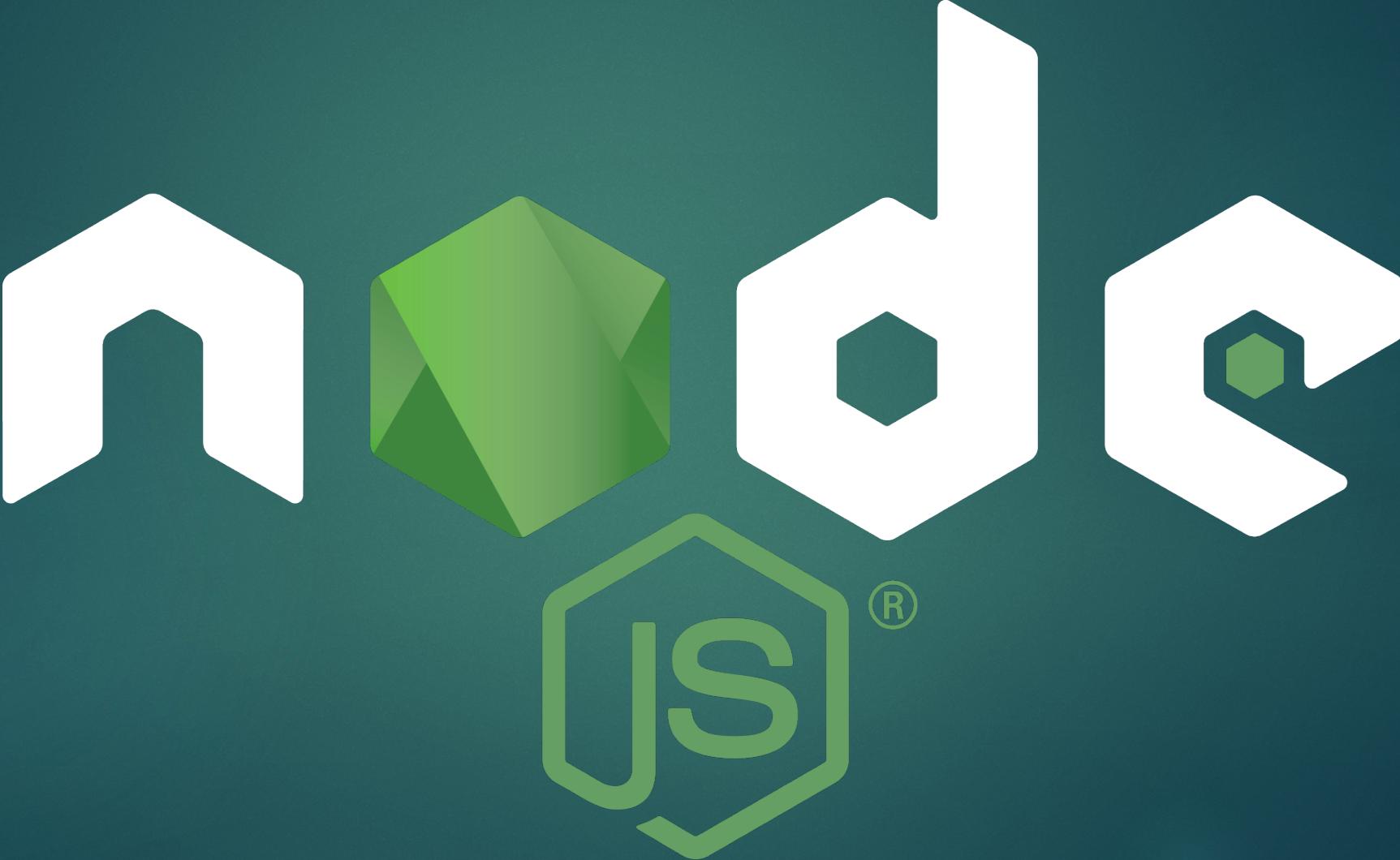
Resumindo...





Acho que
vivemos isso em
outros lugares
também...
Kkkkkkk....

Mas, como resolver isso???



Mas, o que é Node.js?

- ▶ Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
- ▶ Foi criada em 2009 por Ryan Dahl.

Fonte: <https://nodejs.org>

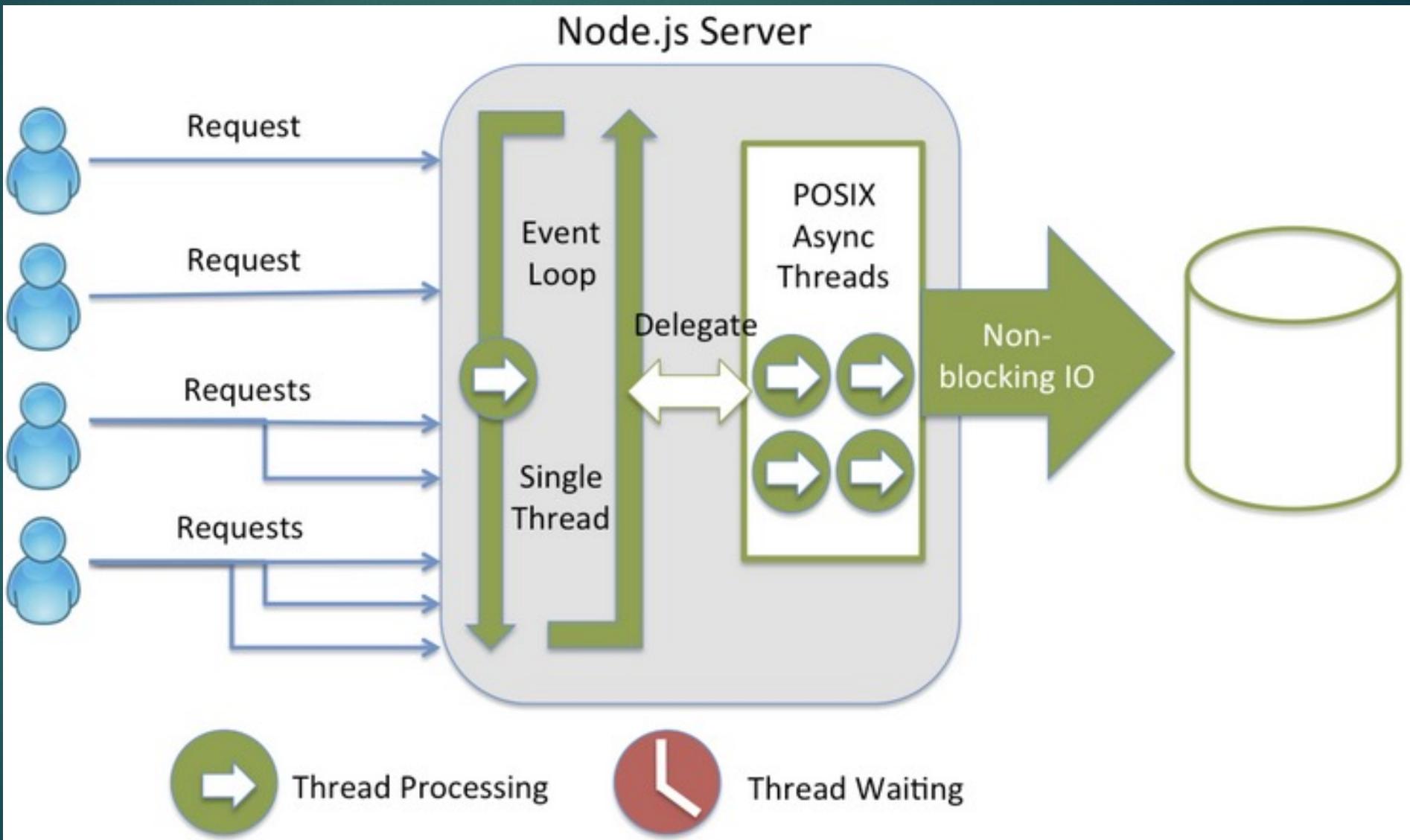
E como ele faz esse milagre?

DAI O PALMEIRAS GANHA O MUNDIAL

Falando em
milagre...
kkkkk

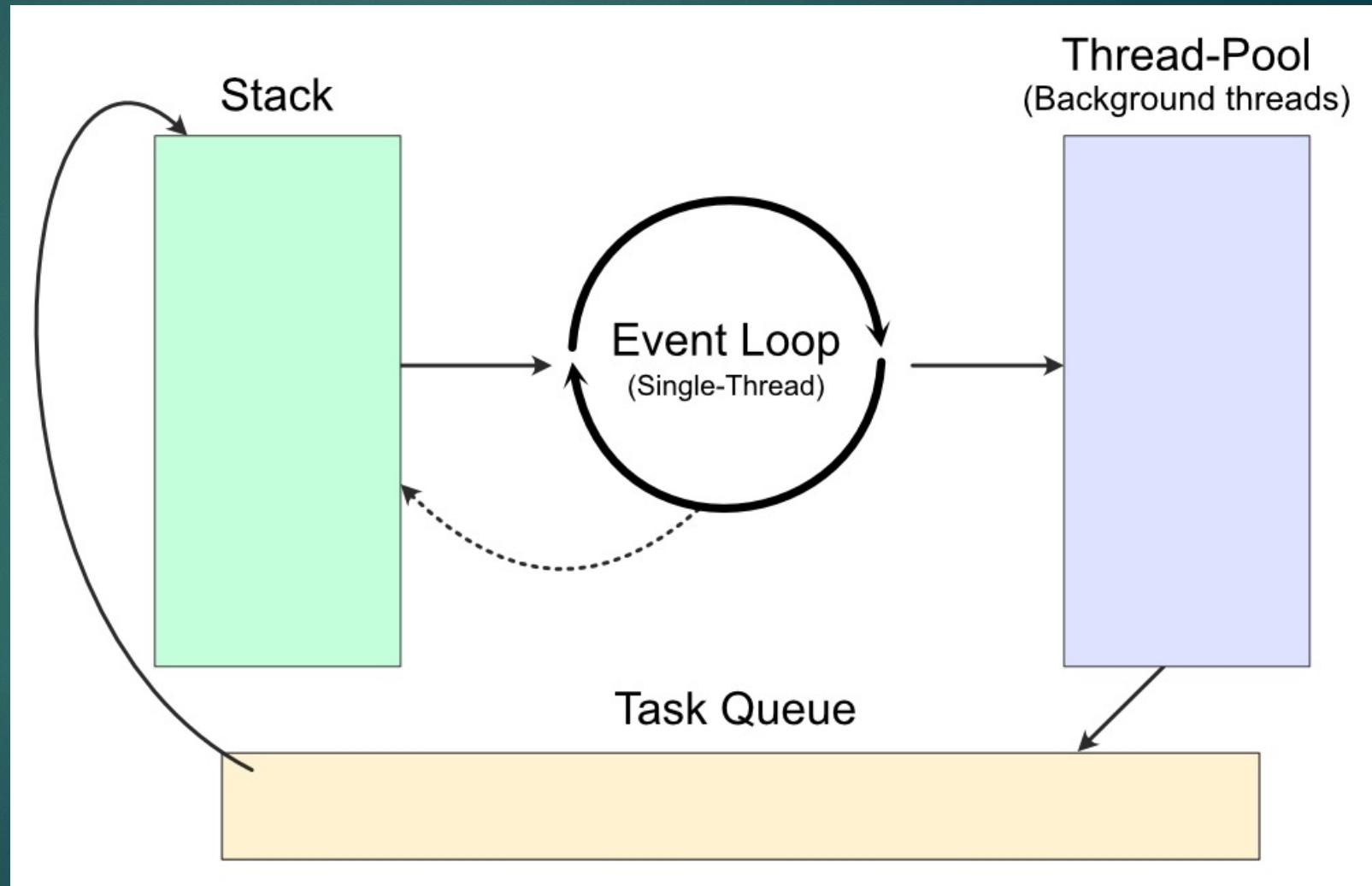


Com uma arquitetura assíncrona!



Vixe! Mas, como funciona isso?

- ▶ Basicamente com:
 - ▶ Single Thread e
 - ▶ Event Loop.

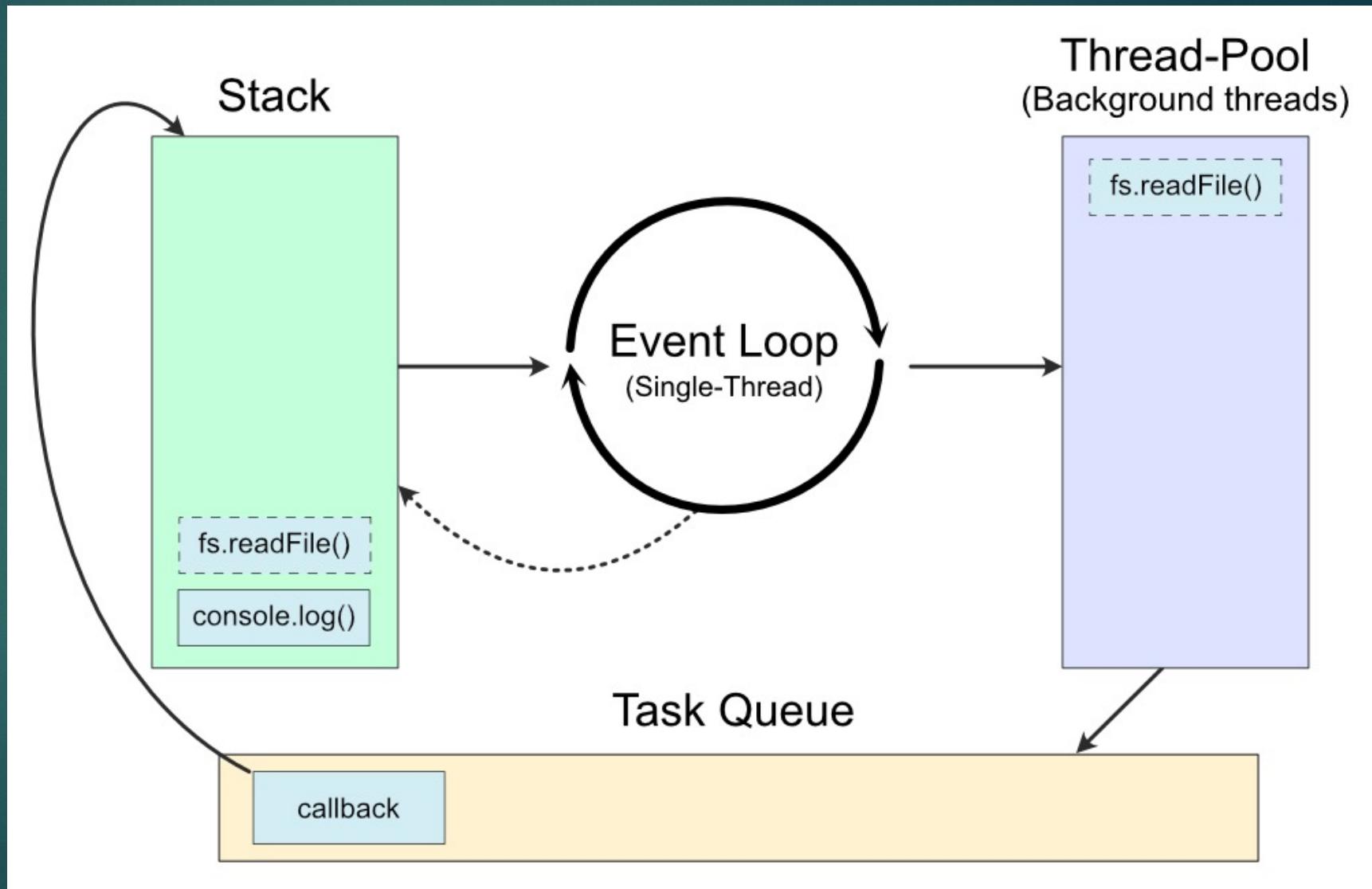


Ou seja...

- ▶ Quando uma requisição chega, ela é empilhada na Stack...
- ▶ O Event Loop fica monitorando a Stack para verificar se a novas requisições...
- ▶ Quando o Event Loop encontra uma requisição de processamento “rápido”, ela é processada e o Event Loop é liberado para prosseguir sua busca por novas requisições....
- ▶ **Mas... E se tivermos uma requisição de processamento “longo”?**
- ▶ O Event Loop envia essa requisição para o Background Thread, juntamente com seu call-back...
- ▶ Ou seja, o Background Thread fica responsável por processar a requisição, liberando o Event Loop para prosseguir sua busca por novas requisições na Stack...
- ▶ Quando essa requisição, que está rodando separada no Background Thread, é concluída, a sua função de call-back é adicionado a Task Queue...
- ▶ Quando o Event-Loop termina suas tarefas, esse call-back entrar na Stack e tem sua instrução executada.

Resumindo...

Single Thread e
Event Loop em ação.



No exemplo anterior, temos duas requisições (eventos) na Stack....

Qual deles será executado primeiro?

Desmistificando algumas coisas sobre Node.js

O Node.js é uma linguagem de
programação?

Desmistificando algumas coisas sobre Node.js

O Node.js é uma linguagem de
programação?

Não!!! Você programa utilizando Java Script!!!

Desmistificando algumas coisas sobre Node.js

O Node.js é um framework Java
Script?

Desmistificando algumas coisas sobre Node.js

O Node.js é um framework Java
Script?

Não!!! Ele está mais para uma plataforma de aplicação, na qual você escreve seus programas usando Java Script que serão compilados, otimizados e interpretados pela máquina virtual V8. Essa VM é a mesma que o Google utiliza para executar Java Script no browser Chrome, e foi a partir dela que o criador do Node.js, Ryan Dahl, criou o projeto.

Desmistificando algumas coisas sobre Node.js

O Node.js serve para fazer APIs?

Desmistificando algumas coisas sobre Node.js

O Node.js serve para fazer APIs?

Sim!!! Esse talvez seja o principal uso da tecnologia, uma vez que, por padrão, ela apenas sabe processar requisições. Além disso, seu modelo não bloqueante de tratar as requisições o torna excelente para essa tarefa, consumindo pouquíssimos recursos de hardware.

Desmistificando algumas coisas sobre Node.js

O Node.js serve para fazer
backend de jogos, IoT e apps de
mensagens?

Desmistificando algumas coisas sobre Node.js

O Node.js serve para fazer
backend de jogos, IoT e apps de
mensagens?

Sim!!! Idem ao slide anterior, principalmente porque
esses tipos de aplicações efetuam altos volumes de
requisições.

Desmistificando algumas coisas sobre Node.js

O Node.js serve para fazer
aplicações em tempo real?

Desmistificando algumas coisas sobre Node.js

O Node.js serve para fazer
aplicações em tempo real?

Sim!!! Usando algumas extensões de web socket, tais como: Socket.io, Comet.io, etc. Dessa forma, é possível criar aplicações de tempo real sem onerar excessivamente o servidor da mesma.

Então, blz...

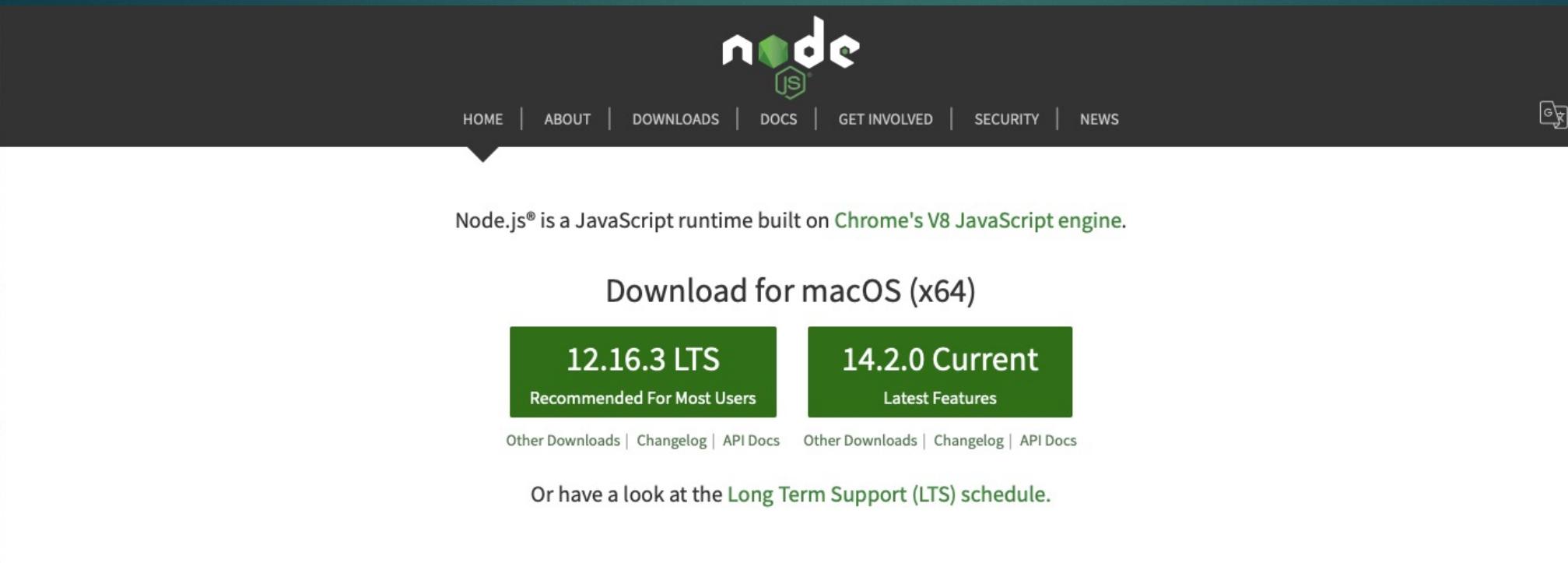
HANDS ON

{ trust_ful } >>>

Ou seja >>>>



Instalação e Configuração



The screenshot shows the official Node.js website. At the top, there's a dark header with the Node.js logo and navigation links: HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, and NEWS. To the right of the header is a small GitHub icon. Below the header, a large green banner states: "Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine." Underneath this, there are two prominent download buttons: one for "12.16.3 LTS" (Recommended For Most Users) and another for "14.2.0 Current" (Latest Features). Below each button are links to "Other Downloads", "Changelog", and "API Docs". A link to the "Long Term Support (LTS) schedule" is also present. On the far right of the banner, the URL "https://nodejs.org" is displayed.

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for macOS (x64)

12.16.3 LTS
Recommended For Most Users

14.2.0 Current
Latest Features

Other Downloads | Changelog | API Docs Other Downloads | Changelog | API Docs

Or have a look at the [Long Term Support \(LTS\) schedule](#).

<https://nodejs.org>



© OpenJS Foundation. All Rights Reserved. Portions of this site originally © Joyent.

Node.js is a trademark of Joyent, Inc. and is used with its permission. Please review the Trademark Guidelines of the OpenJS Foundation.

Node.js Project Licensing Information.

[Report Node.js issue](#) | [Report website issue](#) | [Get Help](#)



Thank you mikeal for being a Node.js
contributor 29 contributions

Instalação e Configuração

- ▶ Após efetuar o download, instale o Node.js em seu SO.
- ▶ Não tem segredo... É só dar um Next... Next... Next...
- ▶ Após instalar, abra um Terminal ou Prompt de Comando e digite:
node -v && npm -v
- ▶ Isso lhe mostrará as versões instaladas do Node.js e do npm (Node Package Manager)

Testando se a instalação rolou...

- ▶ No Terminal, digite:
node
- ▶ Assim, você acessará o REPL (Read-Eval-Print-Loop), que permite executar código Java Script diretamente no Terminal. Digite:
console.log("Hello World");
- ▶ Como resultado deverá aparecer:
Hello World
undefined

Se você viu esse resultado: EUREKA!!! Sua instalação está Ok!

Caso dê problema...

- ▶ Você precisará configurar a variável de ambiente **NODE_ENV** no seu Sistema Operacional.

Gerenciando módulos com NPM...

- ▶ O NPM (Node Package Manager) é o gerenciador de pacotes do Node.js.
- ▶ Ele já vem integrado junto com o instalador do Node.js.
- ▶ O NPM permite que sejam instalados módulos de terceiros em seus projetos.
- ▶ Por meio dele é possível instalar, listar, remover, atualizar módulos no projeto, entre outras funcionalidades diversas, até mesmo publicar um módulo no site do npm.

Módulos e o package.json

- ▶ Todo projeto Node.js é um módulo.
- ▶ O termo módulo vem do conceito de que a arquitetura do Node.js é modular.
- ▶ E todo módulo tem seu descritor, o arquivo package.json.

```
2   "name": "workshop",
3   "version": "0.0.0",
4   "private": true,
5   "scripts": {
6     "start": "node ./bin/www"
7   },
8   "dependencies": {
9     "cookie-parser": "~1.4.4",
10    "debug": "~2.6.9",
11    "ejs": "~2.6.1",
12    "express": "~4.16.1",
13    "http-errors": "~1.6.3",
14    "mongodb": "^3.5.7",
15    "morgan": "~1.9.1"
16  }
17 }
```

Exemplo de
package.json...

Escopo das variáveis globais...

- ▶ Volte ao seu terminal, onde o Node.js está sendo executado e digite:

```
global.hoje = new Date();  
console.log(global.hoje);
```

- ▶ Dessa forma, criamos uma variável global no projeto, sem a necessidade de chamá-la via require ou passá-la por parâmetro em uma função.

E o CommonJS, o que é?



O Node.js utiliza o padrão CommonJS para organização e carregamento dos módulos.



Vejamos na prática:



Crie uma pasta para armazenar seus projetos Node.js. Eu, por exemplo, tenho uma pasta **NodeProjects**.



Nesta pasta, crie uma pasta chamada **exemplocommonjs**

Exemplo do padrão CommonJS...

- ▶ Agora, dentro da pasta **exemplocommonjs**, crie os arquivos a seguir:

hello.js

Que terá como conteúdo:

```
module.exports = function(msg){  
    console.log(msg);  
};
```

Exemplo do padrão CommonJS...

- ▶ Agora, dentro da pasta **exemplocommonjs**, crie os arquivos a seguir:

human.js

Que terá como conteúdo:

```
exports.hello = function(msg){
    console.log(msg);
};
```

Exemplo do padrão CommonJS...

- ▶ Agora, dentro da pasta **exemplocommonjs**, crie os arquivos a seguir:

app.js

Que terá como conteúdo:

```
var hello = require('./hello');  
var human = require('./human');
```

```
hello('Vamos que vamos!!!');  
human.hello('É nós que tá!!!');
```

E aí???

► O que você notou de interessante nisso???

Criando um CRUD com NodeJs e MongoDB

- ▶ Para construir essa aplicação, utilizaremos também o framework ExpressJS.
- ▶ O ExpressJS nos permitirá criar aplicações e web API de forma rápida e fácil.
- ▶ Para utilizá-lo temos que instalar o módulo express-generator, por meio NPM:
- ▶ No terminal, digite: **npm install –g express-generator**

Criando o projeto

- ▶ Com o express-generator instalado, acesse, via terminal, a pasta onde seu projeto ficará salvo.
- ▶ Então digite: **express -e --git workshop**
- ▶ Acima, estou considerando que seu projeto se chamará workshop
- ▶ O -e serve para usar a view-engine EJS e o --git serve para preparar o projeto para o versionamento com Git.

Criando e executando o projeto

► Vá digitando y para confirmar as solicitações e o projeto será criado.

► Depois entre na pasta do projeto e instale as dependências:

cd workshop

npm install

► Agora, execute o projeto:

npm start

► Abra um navegador e confira se o projeto está ok acessando:

localhost:3000

Routes and Views

- ▶ Antes de codificarmos pra valer, precisamos entender o que são **Rotas e Visões**:
- ▶ **Routes (Rotas) são regras para manipulação das requisições HTTP.**
- ▶ Para entender melhor as rotas, abra os arquivos:
app.js e routes/index.js

VAMOS ENTENDER O QUE O render, req, res, função anônima (callback) e onde está o MVC nisso tudo.

Routes and Views

- ▶ E para completar toda essa “bruxaria”, como as **views** funcionam?
- ▶ Como estamos utilizando a view-engine EJS (Embedded Java Script), podemos misturar HTML e códigos Java Script server-side para criar nossos layouts.
- ▶ Vide: **app.js**

Codificando nossa projeto

- ▶ Crie, dentro da pasta **views** do seu projeto o arquivo: **novo.ejs**

```
2  <html>
3  <head>
4      <title>Cadastro de Clientes</title>
5      <link rel="stylesheet" href="/stylesheets/style.css" />
6  </head>
7
8  <body>
9      <h1><%= title %></h1>
10     <p>Preencha os dados abaixo para salvar o cliente.</p>
11     <form action="<%= action %>" method="POST">
12         <p>
13             <label>Nome: <input type="text" name="nome" value="<%= doc.nome %>" required /></label>
14         </p>
15         <p>
16             <label>Idade: <input type="number" name="idade" value="<%= doc.idade %>" /></label> </p>
17         <p>
18             <label>UF:
19                 <select name="uf">
20                     <option <% if (doc.uf == 'PR') { %> selected <% } %> >PR</option>
21                     <option <% if (doc.uf == 'RS') { %> selected <% } %> >RS</option>
22                     <option <% if (doc.uf == 'SC') { %> selected <% } %> >SC</option>
23                     <option <% if (doc.uf == 'SP') { %> selected <% } %> >SP</option>
24                     <!-- coloque os estados que quiser -->
25                 </select>
26             </label>
27         </p>
28         <p>
29             <a href="/">Cancelar</a> | <input type="submit" value="Salvar" /> </p>
30     </form>
31 </body>
32 </html>
```

Codificando novo projeto

- ▶ Vá no arquivo:
index.js e adicione
uma nova rota
tratando GET /novo:

```
11
12  /* GET new page. */
13  router.get('/novo', function(req, res, next) {
14    res.render('novo', { title: 'Novo Cadastro de Cliente', doc: {"nome":""}, "idade":""}, "uf":""}, action: '/novo' });
15  });
16
```

Testando:

- ▶ Vá no terminal, encerre o node e execute novamente a aplicação:

npm start

Depois, vá a um navegador e teste.

The screenshot shows a web browser window with the URL `localhost:3000/novo` in the address bar. The main content is a form titled **Novo Cadastro de Cliente**. The instructions say "Preencha os dados abaixo para salvar o cliente." There are three input fields: "Nome" with a text input box, "Idade" with a text input box, and "UF" with a dropdown menu set to "PR". At the bottom are two buttons: "[Cancelar](#)" and a standard "Salvar" button.

Persistindo Dados:

- ▶ Vamos organizar o MongoDB:
 - ▶ Crie uma pasta chamada **data** dentro da pasta do seu projeto.
 - ▶ Nela armazenaremos os dados do MongoDB referente ao projeto.
- ▶ Agora, abra um terminal e entre na pasta **bin** da instalação do MongoDB e digite:
mongod --dbpath caminho_do_seu_projeto/data
- ▶ Obs: Se estiver utilizando Mac ou Linux, acrescente um ./ antes do comando mongod.

Persistindo Dados:

- ▶ Abra outro terminal, entre na pasta **bin** da instalação do MongoDB e digite:
mongo
- ▶ Obs: Se estiver utilizando Mac ou Linux, acrescente um ./ antes do comando mongo.
- ▶ **Para criar uma base de dados:**
 - ▶ Agora, digite: **use workshop**
- ▶ **Para inserir dados:**
 - ▶ Digite: **db.cliente.insert({ "nome" : "Tiago", "idade" : 36, "uf" : "SP" })**
- ▶ **Para listar os dados:**
 - ▶ Digite: **db.cliente.find().pretty()**

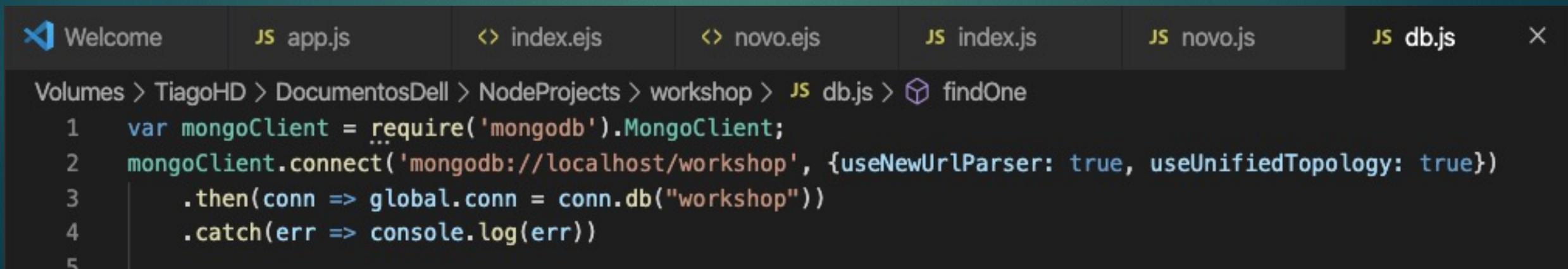
Conectando o MongoDB com o NodeJS

- ▶ Instalando o MongoDBDriver na nossa aplicação:
- ▶ Vá ao terminal do node, já apontando para a pasta do seu projeto e digite:

```
npm install mongodb
```

Conectando o MongoDB com o NodeJS

- ▶ Agora, crie um arquivo chamado **db.js** na raiz do seu projeto. Neste arquivo faremos a conexão e manipulação dos dados entre aplicação e banco. Digite nele:



The screenshot shows a code editor interface with several tabs at the top: Welcome, app.js, index.ejs, novo.ejs, index.js, novo.js, db.js, and a close button. The db.js tab is active. Below the tabs, the file content is displayed:

```
Volumes > TiagoHD > DocumentosDell > NodeProjects > workshop > db.js > findOne  
1 var mongoClient = require('mongodb').MongoClient;  
2 mongoClient.connect('mongodb://localhost/workshop', {useNewUrlParser: true, useUnifiedTopology: true})  
3   .then(conn => global.conn = conn.db("workshop"))  
4   .catch(err => console.log(err))  
5
```

Carregando o módulo db

- ▶ Agora, abra o arquivo **www** , que fica na pasta **bin** do seu projeto e adicione a seguinte linha no início dele:

```
Volumes > TiagoHD > DocumentosDell > NodeProjects > workshop > bin > JS www > ...
1  #!/usr/bin/env node
2
3  global.db = require('../db');
```

CRUD – Create / Retrieve / Update/ Delete - Implementando o Create

- ▶ Para cadastrar no banco de dados, precisamos, primeiramente, incluir uma função insert no nosso arquivo **db.js**. Sendo, abra-o e digite:

```
Volumes > TiagoHD > DocumentosDell > NodeProjects > workshop > JS db.js >  findOne  
9  
10  function insert(customer, callback){  
11    global.conn.collection('cliente').insertOne(customer, callback);  
12  }  
13
```

- ▶ E, ao final do arquivo **db.js** digite:

```
Volumes > TiagoHD > DocumentosDell > NodeProjects > workshop > JS db.js >  <unknown>  
26  
27  module.exports = { insert }
```

CRUD – Create / Retrieve / Update/ Delete - Implementando o Create

- ▶ Agora, vamos implementar a rota **POST /novo** que vai receber a request vinda da view novo.ejs. Abra o arquivo **index.js** e digite:

```
17  /* POST new page. */
18  router.post('/novo', function (req, res, next) {
19    const nome = req.body.nome;
20    const idade = parseInt(req.body.idade);
21    const uf = req.body.uf;
22    global.db.insert({ nome, idade, uf }, (err, result) => {
23      if (err) { return console.log(err); }
24      res.redirect('/');
25    });
26  });
```

CRUD – Create / **Retrieve** / Update/ Delete - Implementando o Retrieve

- ▶ Note que, no rota POST /novo, ao final, redirecionamos para a raiz do projeto (/). Vamos organizar nosso projeto para que, quando sempre que redirecionarmos para uma rota GET / , possamos listar todos os clientes cadastrados.
- ▶ Sendo assim, abra o arquivo **db.js** e implemente o seguinte método:

```
5
6  function findAll(callback){
7    global.conn.collection('cliente').find({}).toArray(callback);
8  }
9
```

CRUD – Create / **Retrieve** / Update/ Delete - Implementando o Retrieve

- ▶ Note que, na rota POST /novo, ao final, redirecionamos para a raiz do projeto (/). Vamos organizar nosso projeto para que, quando sempre que redirecionarmos para uma rota GET / , possamos listar todos os clientes cadastrados.
- ▶ Sendo assim, abra o arquivo **db.js** e implemente o seguinte método:

```
3  
4  
5  
6   function findAll(callback){  
7     global.conn.collection('cliente').find({}).toArray(callback);  
8   }  
9
```

- ▶ E no final do arquivo **db.js**, digite:

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

CRUD – Create / **Retrieve** / Update/ Delete - Implementando o Retrieve

- ▶ Como visto anteriormente, quando recebermos um **GET** / , iremos chamar o método findAll() e mostrar a lista de clientes no arquivo **index.ejs**. Sendo assim, abra o arquivo index.js e vamos implementar a rota **GET** / , como apresentado abaixo:

```
4  /* GET home page. */
5  router.get('/', function (req, res) {
6    global.db.findAll((err, docs) => {
7      if (err) { return console.log(err); }
8      res.render('index', { docs });
9    });
10 });


```

CRUD – Create / Retrieve / Update/ Delete - Implementando o Retrieve

- E, por último, precisamos preparar nossa **index.ejs**, para receber para receber os docs retornados do banco e apresentá-los para o usuário.
- Abra o arquivo **index.ejs** e digite:

(Parte 1)

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Cadastro de Clientes</title>
6      <link rel="stylesheet" href="/stylesheets/style.css" />
7  </head>
8
9  <body>
10     <h1>Listagem de Clientes</h1>
11     <p>Clientes já cadastrados no sistema.</p>
12     <table style="width:50%">
13         <thead>
14             <tr style="background-color: #CCC">
15                 <td style="width:50%">Nome</td>
16                 <td style="width:15%">Idade</td>
17                 <td style="width:15%">UF</td>
18                 <td colspan="2" style="width:20%">Ações</td>
19             </tr>
20         </thead>
21         <tbody>
22             <% if(!docs || docs.length == 0) { %>
23             <tr>
24                 <td colspan='4'>Nenhum cliente cadastrado.</td>
25             </tr>
26             <% } else {
27                 docs.forEach(function(customer){ %>
28                 <tr>
29                     <td style='width:50%'><%= customer.nome %></td>
30                     <td style='width:15%'><%= customer.idade %></td>
31                     <td style='width:15%'><%= customer.uf %></td>
```

CRUD – Create / Retrieve / Update/ Delete - Implementando o Retrieve

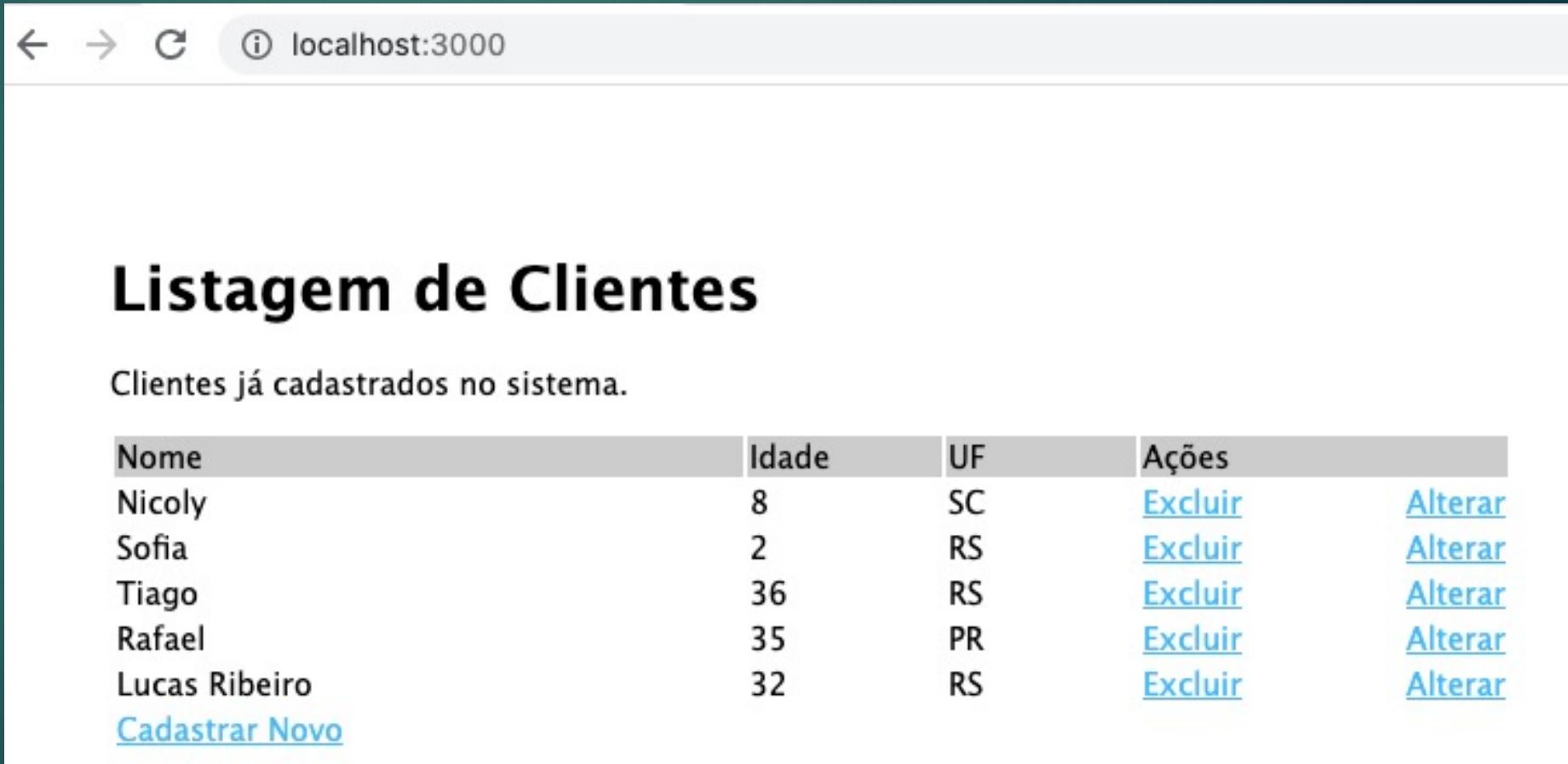
- E, por último, precisamos preparar nossa **index.ejs**, para receber para receber os docs retornados do banco e apresentá-los para o usuário.
- Abra o arquivo **index.ejs** e digite:

(Parte 2)

```
20 </thead>
21 <tbody>
22   <% if(!docs || docs.length == 0) { %>
23     <tr>
24       <td colspan='4'>Nenhum cliente cadastrado.</td>
25     </tr>
26   <% } else {
27     docs.forEach(function(customer){ %>
28       <tr>
29         <td style='width:50%'><%= customer.nome %></td>
30         <td style='width:15%'><%= customer.idade %></td>
31         <td style='width:15%'><%= customer.uf %></td>
32         <td style='width:20%'>
33           <a href="/delete/<%= customer._id %>" onclick="return confirm('Tem certeza?');" >Excluir</a>
34         </td>
35         <td style='width:20%'>
36           <a href="/edit/<%= customer._id %>" title="Alterar Cliente" >Alterar</a>
37         </td>
38       </tr>
39     <% }) 
40   >%>
41 </tbody>
42 <tfoot>
43   <tr>
44     <td colspan="4">
45       <a href="/novo">Cadastrar Novo</a>
46     </td>
47   </tr>
48 </tfoot>
49 </table>
50 </body>
51 </html>
```

CRUD – Create / **Retrieve** / Update/ Delete - Implementando o Retrieve

- O resultado deverá ser algo, mais ou menos, assim:



The screenshot shows a web browser window with the URL `localhost:3000`. The page title is **Listagem de Clientes**, which translates to "Client List". Below the title, a message says "Clientes já cadastrados no sistema." (Clients already registered in the system). A table displays the following data:

Nome	Idade	UF	Ações
Nicoly	8	SC	Excluir Alterar
Sofia	2	RS	Excluir Alterar
Tiago	36	RS	Excluir Alterar
Rafael	35	PR	Excluir Alterar
Lucas Ribeiro	32	RS	Excluir Alterar

[Cadastrar Novo](#)

CRUD – Create / Retrieve / Update/ Delete - Implementando o Delete

- ▶ Como pode ser visto, já criamos, no arquivo **index.ejs**, um link que faz uma requisição tipo **GET /delete**. Porém, esta rota não foi configurada e nem o método para exclusão foi implementado. Então, vamos lá!!!
- ▶ Abra o arquivo **db.js** e implemente os seguintes códigos:
 - ▶ Primeiro, convertemos o id do usuário para **ObjectId**, pois ele virá na URL como uma String e o MongoDB não entende Strings como ids.

```
14 var ObjectId = require("mongodb").ObjectId;
```

CRUD – Create / Retrieve / Update/ Delete - Implementando o Delete

- ▶ Já, com o id convertido para ObjectId, implementamos o método:

```
24  function deleteOne(id, callback){  
25    |   global.conn.collection('cliente').deleteOne({_id: new ObjectId(id)}, callback);  
26  }
```

- ▶ Não se esqueça de exportar o referido método:

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

CRUD – Create / Retrieve / Update/ Delete - Implementando o Delete

- ▶ Agora, vamos implementar a rota **GET /delete**. Abra o arquivo **index.js** e digite:

```
49  /* GET delete page. */
50  router.get('/delete/:id', function (req, res) {
51    var id = req.params.id;
52    global.db.deleteOne(id, (err, r) => {
53      if (err) { return console.log(err); }
54      res.redirect('/');
55    });
56  });
```

- ▶ Pronto! A exclusão já estará acontecendo!!!

CRUD – Create / Retrieve / **Update**/ Delete - Implementando o Update

- ▶ Alterar um elemento, exige que antes se carregue os dados a serem alterados no formulário **novo.ejs**. Ele já está pronto para receber tais dados.
- ▶ Porém, ainda precisamos implementar o método para carregar um objeto do banco de dados tendo o seu ID como filtro. Sendo assim, abra o arquivo **db.js** e digite:

```
16  function findOne(id, callback){  
17    global.conn.collection("cliente").find(new ObjectId(id)).toArray(callback);  
18  }
```

- ▶ E não se esqueça de torná-lo acessível:

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

CRUD – Create / Retrieve / **Update**/ Delete - Implementando o Update

- ▶ Agora, precisamos implementar a rota **GET /edit**. Abra o arquivo **index.js** e digite:

```
28  /* GET edit page. */
29  router.get('/edit/:id', function(req, res, next) {
30    var id = req.params.id;
31    global.db.findOne(id, (err, docs) => {
32      if(err) { return console.log(err); }
33      res.render('novo', { title: 'Alterar Cliente', doc: docs[0], action: '/edit/' + docs[0]._id });
34    });
35  });
```

- ▶ Com isso, renderizaremos a página novo.ejs e enviaremos para ela a model contendo título, os dados do cliente e a action para o form, sendo uma /edit, contendo o id do cliente como parâmetro.

CRUD – Create / Retrieve / **Update**/ Delete - Implementando o Update

- ▶ Agora, que os dados já estão carregados no form, podemos fazer a atualização dos dados no banco. Para tanto, abra o arquivo **db.js** e digite:

```
24  function update(id, customer, callback){  
25    global.conn.collection("cliente").replaceOne({_id: new ObjectId(id)}, customer, callback);  
26  }  
27  
28 module.exports = { findAll, insert, deleteOne, findOne, update }
```

- ▶ **Mais uma vez, não esqueça de atualizar o module.exports...**

CRUD – Create / Retrieve / **Update**/ Delete - Implementando o Update

- ▶ E, por fim, precisamos implementar a rota POST /edit, para invocarmos a função update e renderizarmos a lista de clientes, já com os dados atualizados. Abra o arquivo **index.js** e digite:

```
37  /* POST edit page. */
38  router.post('/edit/:id', function(req, res, next) {
39    const id = req.params.id;
40    const nome = req.body.nome;
41    const idade = parseInt(req.body.idade);
42    const uf = req.body.uf;
43    global.db.update(id, {nome, idade, uf}, (err, result) => {
44      if(err) { return console.log(err); }
45      res.redirect('/');
46    });
47 });
```

CRUD PRONTO!

AGORA É COM VOCÊS, PESSOAL!!!

COMO ATIVIDADE, NESTE MESMO PROJETO, IMPLEMENTE UM CRUD DE UM **PRODUTO**, QUE TEM COMO CARACTERÍSTICAS: **id, descricao, valor de compra e valor de venda.**

O projeto compactado, deverá ser enviado na respectiva atividade no Teams.

Vamos fazer um chat???

- Utilizando:

NodeJS e Socket.io



Configurando e instalando

- ▶ Bem, já temos o Framework Express instalado...
- ▶ Então, abra um terminal e entre na pasta onde será criado o seu projeto e digite:

express -e --git chatpos

- ▶ Depois:
- ▶ Entre na pasta chatpos e digite:

npm install

Configurando e instalando

- ▶ Instalando a dependência do módulo Socket.io:
npm install -S socket.io
- ▶ O Socket.io é dividido em duas partes: **cliente e servidor**.
- ▶ A dependência instalada acima é necessária para criar o servidor que rodará em nossa aplicação Node.js.

Configurando e instalando

- ▶ Agora, precisamos inicializar o servidor Socket.io. Para tanto, abra o arquivo **www** presente na pasta bin do seu projeto e digite o seguinte código após o comando `createServer`:

```
var io = require('socket.io')(server);
io.on('connection', function(socket){
  console.log('Há um usuário conectado!');
  socket.on('disconnect', function(){
    console.log('Usuário desconectado');
  });
});
```

Depois, vá ao terminal e rode seu projeto com um: **npm start**

Quando um navegador abre ou fecha o chat, tal informação é descrita no log.

Programando o chat

- ▶ Crie um arquivo **chat.ejs** e salve-o na pasta views do seu projeto.

```
<!doctype html>
<html>

<head>
<title>Socket.IO Chat - Pós</title>
<style>
* {
margin: 0;
padding: 0;
box-sizing: border-box;
}

body {
font: 13px Helvetica, Arial;
}

form {
background: #000;
padding: 3px;
position: fixed;
bottom: 0;
width: 100%;
}
```

```
form input {
border: 0;
padding: 10px;
width: 90%;
margin-right: .5%;
}

form button {
width: 9%;
background: rgb(130, 224, 255);
border: none;
padding: 10px;
}

#messages {
list-style-type: none;
margin: 0;
padding: 0;
}

#messages li {
padding: 5px 10px;
}

#messages li:nth-child(odd) {
background: #eee;
}
</style>
</head>
```

```
<body>
<ul id="messages"></ul>
<form action="">
<input id="m" autocomplete="off" /><button>Send</button>
</form>

<script src="/socket.io/socket.io.js"></script>
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<script>
$(function () {
var socket = io();
$('form').submit(function(){
socket.emit('chat message', $('#m').val());
$('#m').val("");
return false;
});
socket.on('chat message', function(msg){
$('#messages').append($('- ').text(msg));
});
});
</script>
</body>
</html>

```

Programando a rota do chat

- ▶ Abra seu arquivo index.js e acrescente o seguinte código:

```
router.get('/chat', function(req, res){  
  res.render('chat', {});  
});
```

Finalizando a codificação do servidor

- ▶ Volte ao arquivo **www**, presente no pasta bin do seu projeto o código anterior para:

```
var io = require('socket.io')(server);
io.on('connection', function (socket) {
  console.log('Há um usuário conectado!');
  socket.on('chat message', function(msg){
    console.log('message: ' + msg);
    io.emit('chat message', msg);
  });
  socket.on('disconnect', function(){
    console.log('Usuário desconectado!');
  });
});
```

Hora de testar

- ▶ Agora, pessoal!!! É só testar!!!
- ▶ Boa sorte!!!
- ▶ Muito Obrigado!!!
- ▶ tiago.carneiro01@fatec.sp.gov.br

THE
END

Desenvolvimento NodeJS

FATEC PROF. JOSÉ CAMARGO – FATEC JALES
TÓPICOS ESPECIAIS EM SISTEMAS PARA INTERNET III
PROF. Me. TIAGO RIBEIRO CARNEIRO