

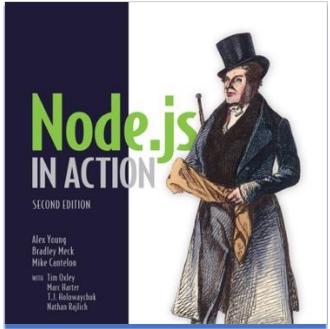
Plataforma Node.js

Tópicos

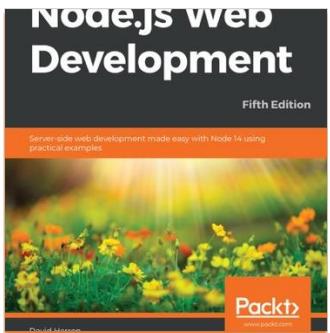
- A plataforma Node.js
 - História e visão geral da plataforma Node.js
 - Linguagem JavaScript
 - Arquitetura da Web (Visão Geral, URI e Protocolo HTTP)
 - Arquitetura da plataforma Node.js
- Módulos e Frameworks
 - Módulos em JavaScript e Node.js
 - Módulos internos do Node.js (*built-in*)
 - Gestão de pacotes e módulos com NPM e yarn
 - Framework Express
- APIs e Acesso a Banco de Dados com Node.js
 - Fundamentos de APIs e RESTful APIs
 - Frameworks de acesso a bancos de dados
 - Database migrations
- Segurança e Tópicos avançados
 - Segurança na Web com Protocolo HTTP
 - JSON Web Tokens (JWT)
 - Open Authorization (OAuth)
 - Tópicos avançados (WebScrap, WebSockets, Shebang, nvm, Docker, https)



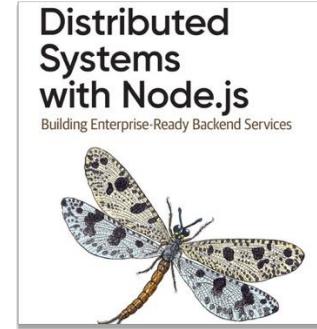
Bibliografia da Disciplina



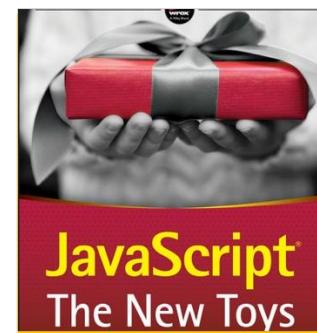
[Node.js in Action, Second Edition \(oreilly.com\)](#) |
[Node.js in Action, Second Edition \(manning.com\)](#)



[Node.js Web Development](#)



[Distributed Systems with Node.js - Thomas Hunter \(2020\)](#)



[JavaScript - The New Toys](#)

Plataforma Node.js

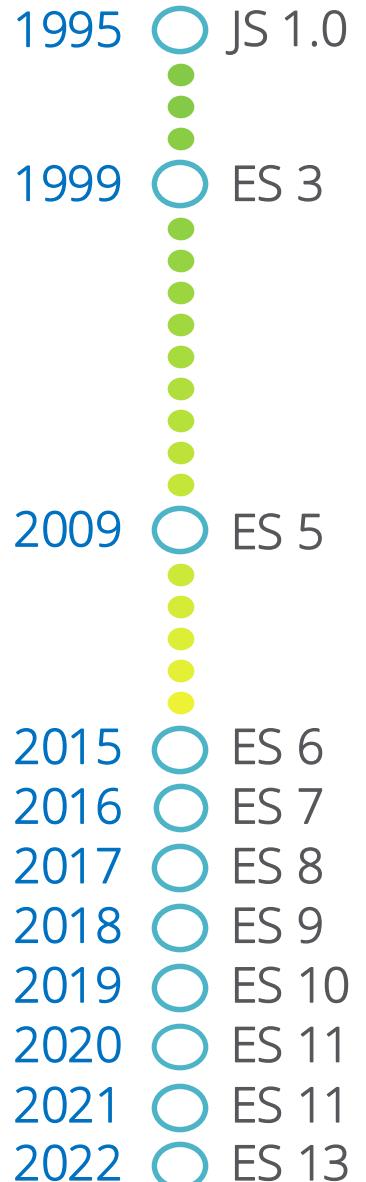
História e visão geral da
plataforma Node.js

História do JavaScript e Node.js

- 1995 - O **JavaScript (JS)** é criado por **Brendan Eich** na Netscape
- 1996 - JS é transferido para o [**ECMA International**](#) para padronização
- 1997 - É lançada a primeira versão do **ECMAScript (ES)** batizada [**ECMA-262**](#)
- 1998 - O ES é aprovado como o padrão [**ISO/IEC 16262**](#)
- 2009 - É lançado o **ES 5**, versão mais compatível em browsers atuais
- 2009 - Ryan Dahl cria o Node.js para tratar o [**problema C10K**](#)
- 2009 - É criado o **CommonJS** para levar o JavaScript para além do Browser
- 2011 - Microsoft implementa uma versão nativa do Node.js no Windows
- 2015 - É lançada o **ECMA Script 6 (ES2015)**
- 2015 - É criada a Fundação Node.js para fomentar a adoção da plataforma

Fontes:

- [ECMA International](#)
- [International Organization for Standardization \(ISO\)](#)



Visão geral da plataforma Node.js

Node.js é um ambiente de execução de programas JavaScript baseado no Google V8 e na biblioteca de processamento assíncrono libuv.

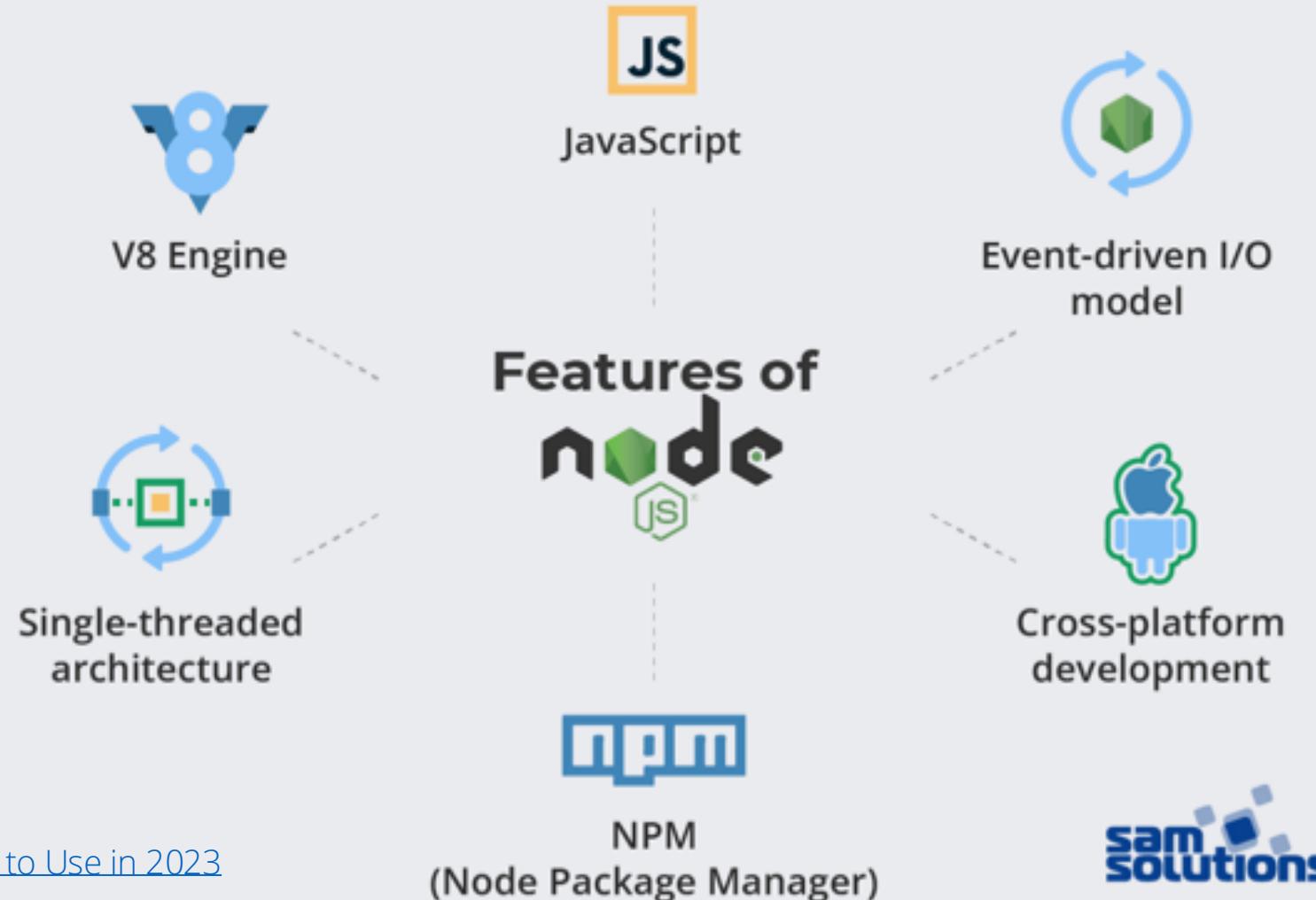


libuv

Node.js tem arquitetura orientada a eventos com modelo não bloqueante de entrada e saída (I/O).

Leve e eficiente, a plataforma é perfeita para a criação de aplicações distribuídas com intensa troca de dados em tempo real.

Visão geral da plataforma Node.js



Fonte: [Node vs Python: What to Use in 2023](#)

Visão geral da plataforma Node.js

Instalação – LTS vs Current

Download for macOS

18.16.0 LTS

Recommended For Most Users

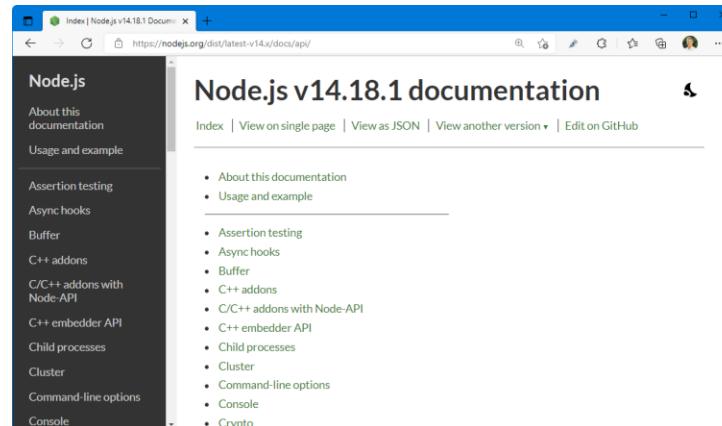
20.3.0 Current

Latest Features

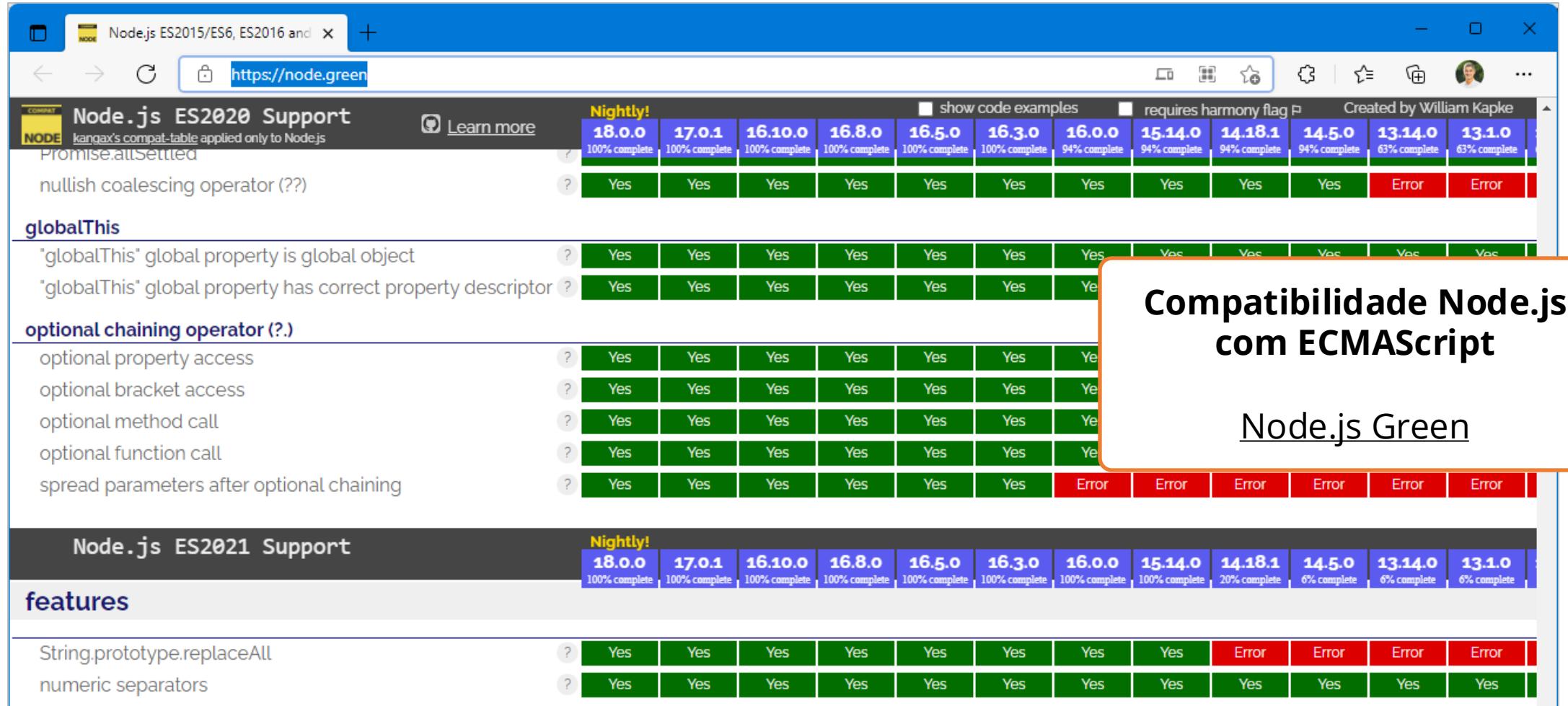
[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Documentação

- [Documentation Node.js \(nodejs.org\)](#)
- [Guides | Node.js \(nodejs.org\)](#)
- [Introduction to Node.js \(nodejs.dev\)](#)



Visão geral da plataforma Node.js



The screenshot shows a compatibility matrix for Node.js ES2020 Support. It includes sections for `Promise.allSettled`, `nullish coalescing operator (??)`, `globalThis`, and `optional chaining operator (?)`. A legend at the top indicates that green means "100% complete" or "Yes", yellow means "94% complete", and red means "Error". A tooltip box highlights the `optional chaining operator (?)` section.

Node.js ES2020 Support

Nightly! 18.0.0, 17.0.1, 16.10.0, 16.8.0, 16.5.0, 16.3.0, 16.0.0, 15.14.0, 14.18.1, 14.5.0, 13.14.0, 13.1.0

Compatibility: kangax's compat-table applied only to Node.js

Promise.allSettled

nullish coalescing operator (??)

globalThis

- "`globalThis`" global property is global object
- "`globalThis`" global property has correct property descriptor

optional chaining operator (?)

- optional property access
- optional bracket access
- optional method call
- optional function call
- spread parameters after optional chaining

Node.js ES2021 Support

Nightly! 18.0.0, 17.0.1, 16.10.0, 16.8.0, 16.5.0, 16.3.0, 16.0.0, 15.14.0, 14.18.1, 14.5.0, 13.14.0, 13.1.0

features

- `String.prototype.replaceAll`
- numeric separators

Compatibilidade Node.js
com ECMAScript

Node.js Green

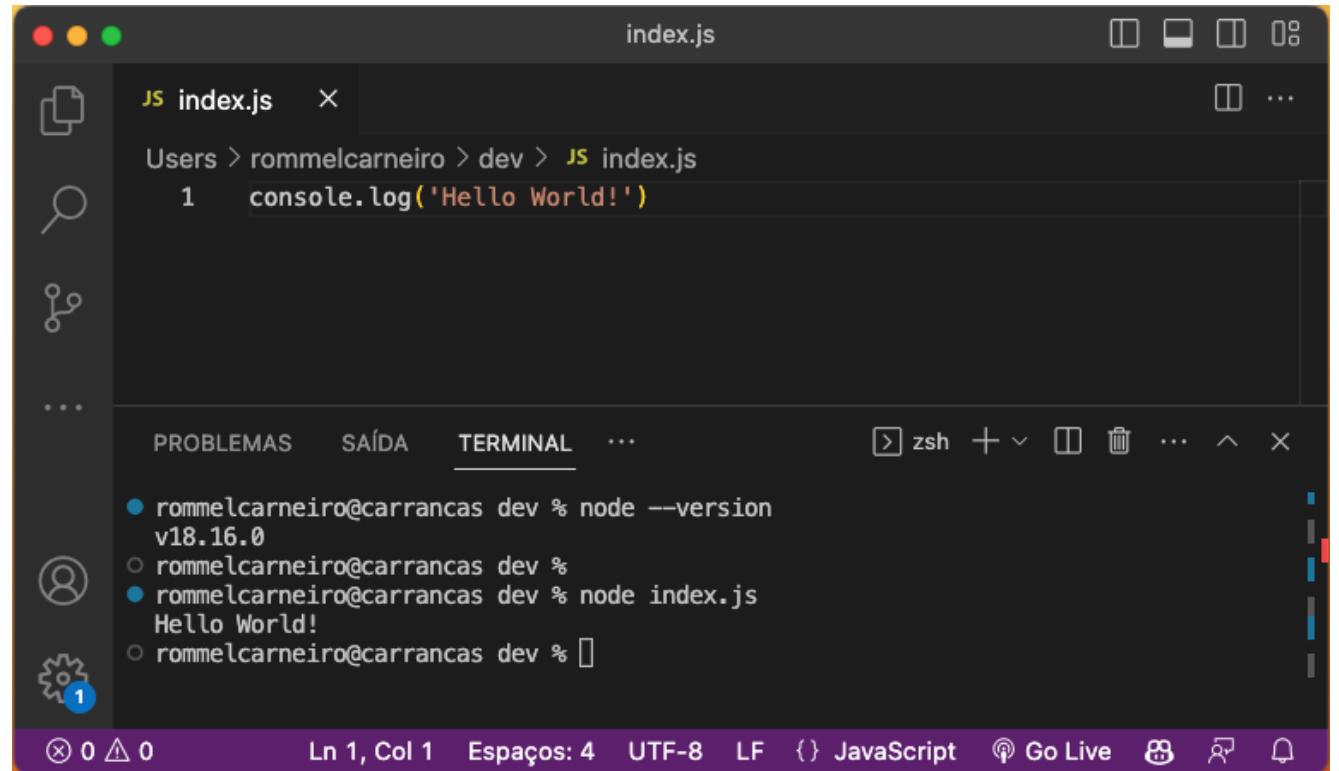
Primeiros passos com Node.js

Instalação e Teste do Ambiente

- Instale o [Node.js](#)
- Instale o [VS Code](#)
- Crie um arquivo index.js
- Execute o arquivo com Node
`node index.js`

Extensões VS Code

- [Live Server](#) (Five Server)
- Postman
- [Docker](#)
- [SQLite Viewer](#)



The screenshot shows a dark-themed instance of VS Code. In the center, there's a terminal window titled 'index.js' with the command 'node index.js' entered. The output shows the console.log('Hello World!') message. Below the terminal, the 'PROBLEMAS', 'SAÍDA', and 'TERMINAL' tabs are visible, with 'TERMINAL' being the active tab. On the left, the sidebar shows a file tree with 'index.js' selected. At the bottom, the status bar displays file information like 'Ln 1, Col 1' and encoding 'UTF-8'.

Ambiente REPL

REPL (READ, EVAL, PRINT, LOOP) é um ambiente interno de uma plataforma que permite executar comandos em uma tela de terminal sem a necessidade de criar um arquivo para isso.

O Node.js permite que sejam executados comandos via REPL, bastando executar **node** no terminal do sistema operacional.

```
$ node
Welcome to Node.js v18.16.0.
Type ".help" for more information.

> .help
.break  Sometimes you get stuck, this gets you out
.clear  Alias for .break
.editor  Enter editor mode
.exit   Exit the REPL
.help   Print this help message
.load   Load JS from a file into the REPL session
.save   Save all evaluated commands in this REPL session to a file
```

Press Ctrl+C to abort current expression, Ctrl+D to exit the REPL

```
> console.log ("Hello World.")
Hello World.
Undefined
```

```
> .exit
$
```

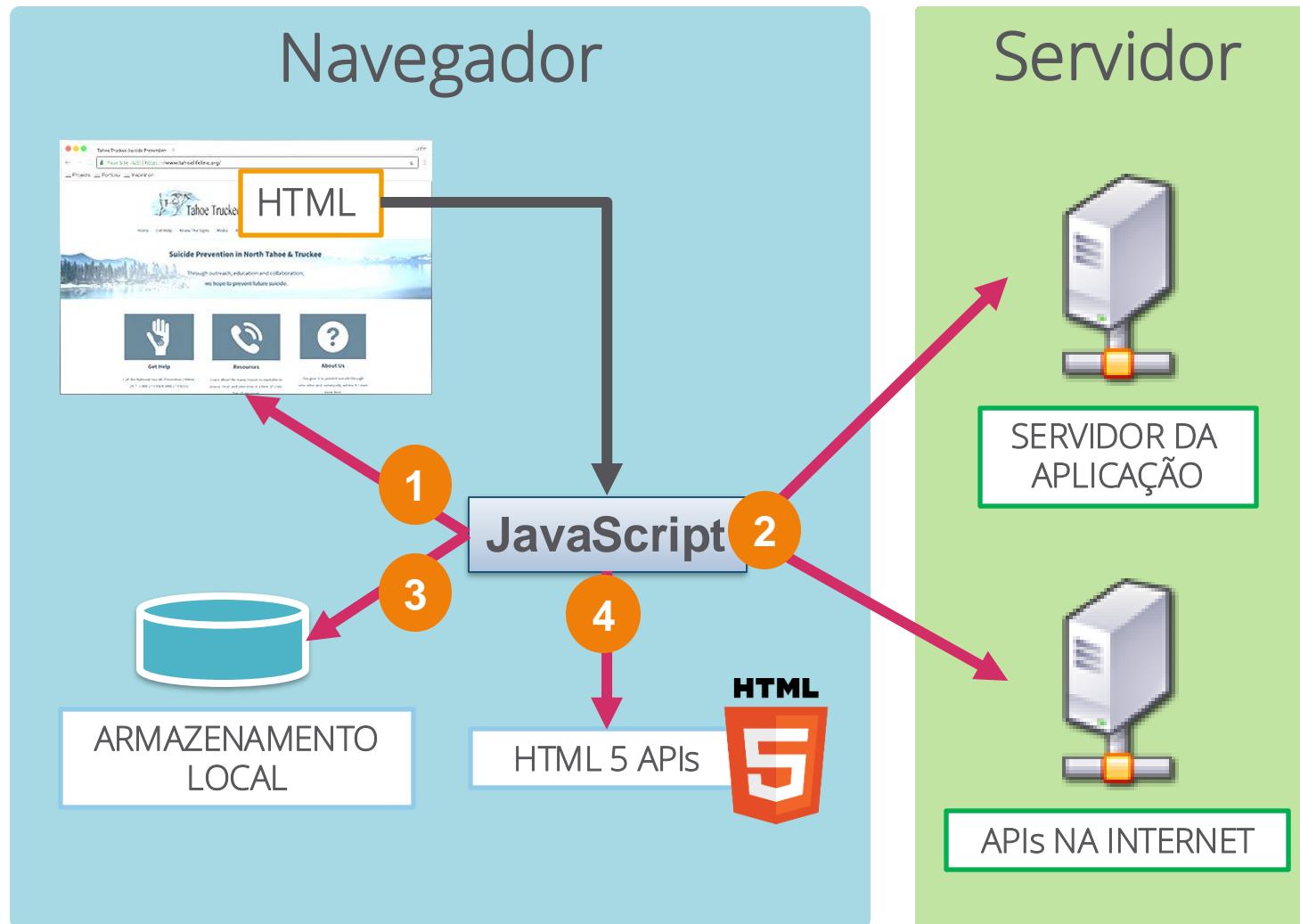

Plataforma Node.js

Linguagem JavaScript

Browsers Engines e ECMAScript engines

Browser / Ambiente	Web Engine	ECMAScript Engine
Mozilla Firefox	Gecko	<u>Spider Monkey</u>
Google Chrome	Blink	<u>Google V8</u>
Apple Safari	WebKit	<u>JavaScriptCore</u>
Internet Explorer	Trident	<u>Chakra Core</u>
Edge	EDGE	<u>Chakra Core</u> <u>Google V8</u>

Capacidades do JavaScript no Navegador



- 1 Manipulação de objetos da página HTML e tratamento de eventos (**DOM Document Object Model**)
- 2 Comunicação com o servidor e uso de Application Programming Interface (API) via AJAX (**XMLHttpRequest | Fetch**)
- 3 Persistência de dados em estruturas providas pelo Browser (**Indexed DB e LocalStorage**)
- 4 Interação com recursos das novas APIs do HTML 5 (**Canvas, Media, File, Drag and Drop, Geolocation, Web Workers, History**)

Navegador vs Node.js

Característica	Navegador	Node.js
Contexto de Execução	Voltado para aplicações web, gerenciando interações de usuário, manipulação de DOM e renderização de páginas.	Voltado para aplicações server-side e ferramentas. Não tem DOM por padrão.
APIs Nativas	DOM (Document, Window), Web APIs (XMLHttpRequest, Fetch, WebSockets), Gráficos (Canvas, WebGL).	APIs para sistemas de arquivos (fs), redes (http, https), módulos, etc.
Manipulação de Arquivos	Acesso limitado ao sistema de arquivos por segurança.	Tem acesso direto ao sistema de arquivos, permitindo leitura e escrita.
Gestão de Módulos	Suporte a ES6 modules com import e export	Usa o sistema CommonJS por padrão, mas também suporta ES6 modules.
Multi-threading	Web Workers permitem multi-threading, mas a manipulação de DOM é single-threaded.	single-threaded com o loop de eventos, mas pode usar threads com o módulo worker_threads ou outros pacotes.
Gerenciamento de Pacotes	Geralmente se usa CDNs ou bundles gerados por ferramentas como Webpack.	Usa o npm (Node Package Manager) para gerenciar e instalar pacotes.
Acesso a Bancos de Dados	IndexedDB, WebStorage para armazenamento local.	Diversos módulos oferecem conexão a bancos de dados, como MongoDB, MySQL, etc.
Variáveis Globais	Objeto window	Objeto global
Tempo de Vida	Enquanto a página estiver aberta ou até o usuário fechar/reabrir ou navegar.	Pode ser contínuo, como um servidor web que fica rodando indefinidamente.



16



Let e Const

- **var** define elementos de escopo global ou escopo de função, se for dentro de funções
- **let** e **const** definem elementos de escopo local.

```
var z = 'original Z';
{
  let x = 'original X';
  var y = 'original Y';
  console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
  {
    // novo escopo ==> novo x
    const x = "novo---- X";
    console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
  }
  // retorno ao escopo anterior ==> original x
  console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
}
console.log(`z: ${z}`); // z é global
console.log(`y: ${y}`); // y é global
console.log(`x: ${x}`); // x é local e linha gera erro
```

Hoisting em declarações com var

Hoisting (içamento) refere-se ao fato de que as declarações de variáveis e funções são antecipadas para o início do bloco onde elas acontecem.

Nos exemplos, podemos usar a variável x e a função antes da sua declaração.

```
// Hoisting com declarações var
{
  console.log ("[ANTES]", x)
  var x = 2
  console.log ("[DEPOIS]", x)
}

// Resultado
// [ANTES] undefined
// [DEPOIS] 2
```

```
// Hoisting com funções
imprime ()

function imprime() {
  console.log ("Hello World")
}

// Resultado
// Hello World
```

Atribuição por valor e por referência

Tipos Primitivos → Atribuição por valor

number, string, boolean, undefined, null, symbol

```
// Atribuição por valor
```

```
var x = 5
var y = x
y = 2

console.log("Valor de x:", x)
console.log("Valor de y:", y)
```

```
// Resultado
// Valor de x: 5
// Valor de y: 2
```

Tipos Complexos → Atribuição por referência

objetos, arrays, date

```
// Atribuição por referência
```

```
var o1 = { valor: 5 }
var o2 = o1
o1.valor = 2
```

```
console.log ("Valor de o1:", o1)
console.log ("Valor de o2:", o2)
```

```
// Resultado
// Valor de o1: { valor: 2 }
// Valor de o2: { valor: 2 }
```

Template Literals

- Similar a strings, com crases (` .. `) no lugar de apostrofes (' .. ') ou aspas (" .. ")
- Permite textos em múltiplas linhas
- Permite a inserção de expressões por meio da construção \${ expr }

```
// Basic literal string creation
let s1 = `In JavaScript '\n' is a line-feed.`

// Multiline strings
let s2 = `In JavaScript
this is
not legal.`

// String interpolation
var name = "Bob", time = "today";
let s3 = `Hello ${name}, how are you ${time}?` 

console.log(` ${s1} \n${s2} \n${s3}`);
```

Tagged Template

Tagged Templates permite processar uma template literal com uma função

```
const person = "Mike"; const age = 28;

function myTag(strings, personExp, ageExp) {
  const str0 = strings[0]; // "That "
  const str1 = strings[1]; // " is a "
  const str2 = strings[2]; // "."

  const ageStr = ageExp > 99 ? "centenarian" : "youngster";

  // We can even return a string built using a template literal
  return `${str0}${personExp}${str1}${ageStr}${str2}`;
}

const output = myTag`That ${person} is a ${age}.`;

console.log(output); // That Mike is a youngster.
```

Arrow Functions

Arrow Functions são uma sintaxe simplificada para a definição de funções.

```
// forma tradicional de declaração
soma = function (a, b) { return a + b }
```

```
// Declaração com arrow functions
soma = (a, b) => { return a + b }
```

```
// ou sem as chaves
soma = (a, b) => a + b
```

Arrow Functions – Exemplos

```
// Expression bodies
evens = [0, 2, 4, 6, 8, 10, 12, 14, 16];
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);
var pairs = evens.map(v => ({ even: v, odd: v + 1 }));
```



```
// Statement bodies
fives = [];
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});
```

Arrow Functions

É importante ressaltar que o operador **this** em uma Arrow Function faz referência ao bloco que contém a função, diferentemente de funções normais.

```
// Lexical this
var bob = {
  _name: "Bob",
  _friends: [],

  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " knows " + f));
  }
}
```

Parâmetros default

Funções agora suportam definição de valores padrão para parâmetros não enviados

```
function foo(index = 0, testing = true) {  
    console.log(`index: ${index} | testing: ${testing}`);  
}  
  
foo(); // Imprime ==> index: 0 | testing: true  
foo(5); // Imprime ==> index: 5 | testing: true  
foo(8, false); // Imprime ==> index: 8 | testing: false
```

Destructuring

Permite a associação de elementos utilizando casamento de padrões

```
// list matching
var [a, b, c] = [1, 'orange', true];
console.log(`a: ${a} | b: ${b} | c: ${c}`)
// Imprime ==> a: 1 | b: orange | c: true

// Can be used in parameter position
function g({ name }) {
  console.log(`name: ${name}`); // Imprime ==> name: 5
}
g({ name: 5 })

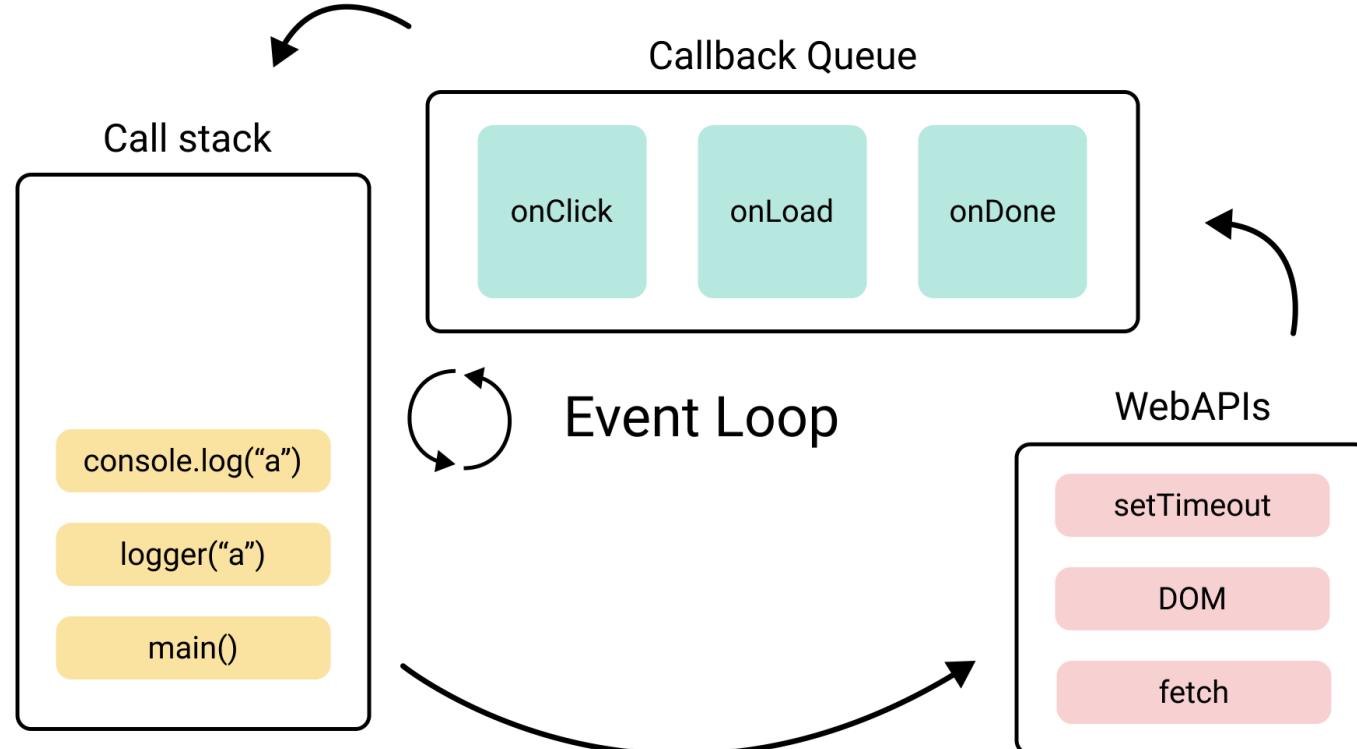
// Fail-soft destructuring
var [a] = [];
console.log(`a: ${a}`); // Imprime ==> a: undefined

// Fail-soft destructuring with defaults
var [a, b = 1] = [2];
console.log(`a: ${a} | b: ${b}`); // Imprime ==> a: 2 | b: 1
```

Arquitetura JavaScript

Componentes

- Call Stack
- Event Loop
- Callback Queue
- Web APIs



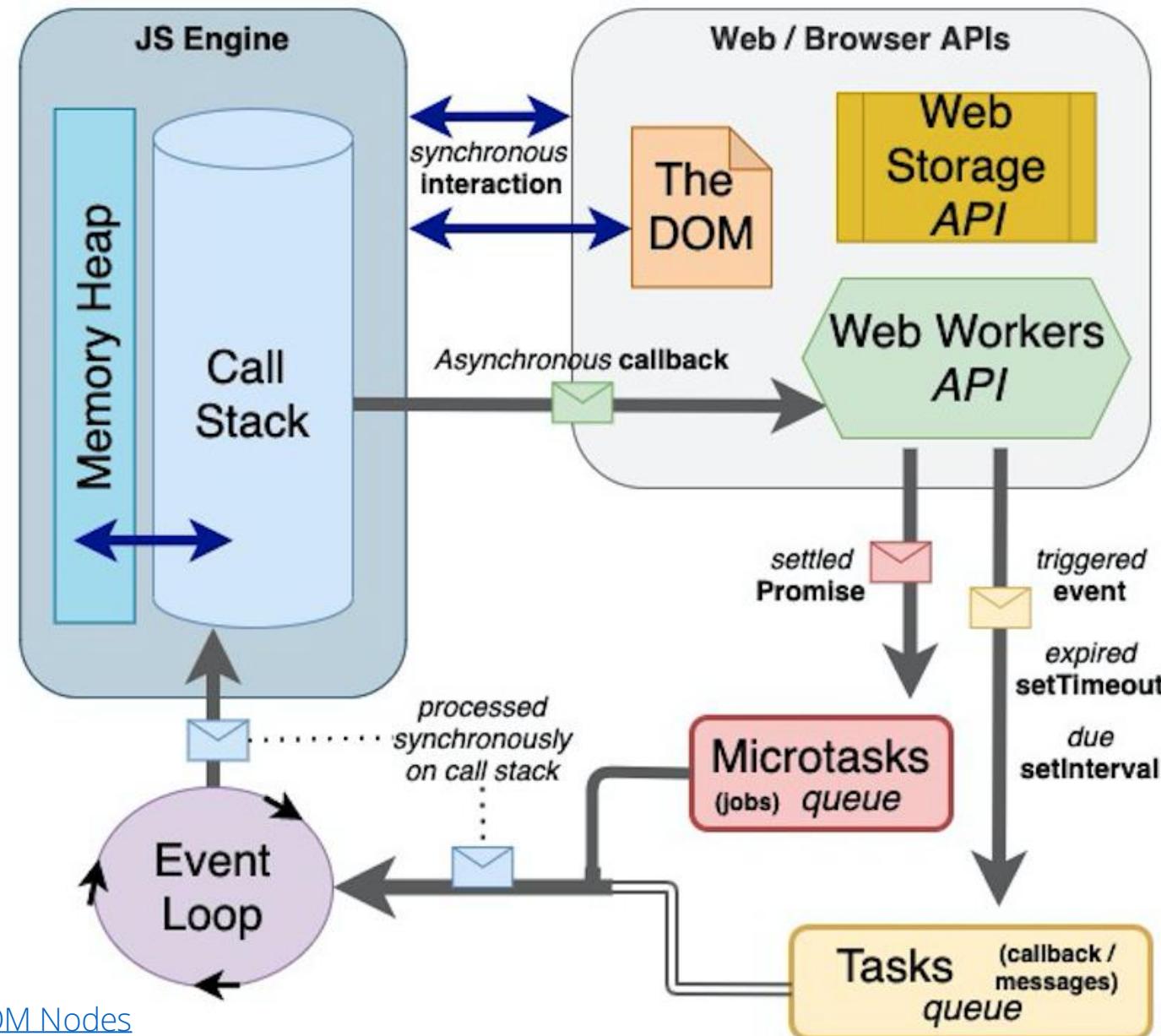
Fontes:

- [Javascrip Event Loop Explained](#)
- [Mas que diabos é o loop de eventos?](#)
- [Loupe](#)
- [JavaScript Concurrency Model and Event Loop](#)
- [Modelo de Concorrência e Event Loop](#)

Arquitetura JavaScript

Componentes da Arquitetura JS

- JavaScript Engine
 - Memory Heap
 - Call Stack
- Event Loop
- Callback Queue
- Microtasks Queue
- Web APIs



Fonte: [Understanding the JavaScript Runtime Environment and DOM Nodes](#)

Arquitetura - Single Thread JavaScript

Desafio → Qual é a ordem de saída do texto no console e em que tempo?

```
setTimeout(() => console.log('A'), 0);
console.log('B');
setTimeout(() => console.log('C'), 100);
setTimeout(() => console.log('D'), 0);

let i = 0;
while (i < 1000000000) { // Assume this takes ~500ms
  let ignore = Math.sqrt(i);
  i++;
}
console.log('E');
```

Log	B	E	A	D	C
Time	1ms	501ms	502ms	502ms	502ms

Fonte: [Distributed Systems with Node.js](#) - Thomas Hunter (2020)

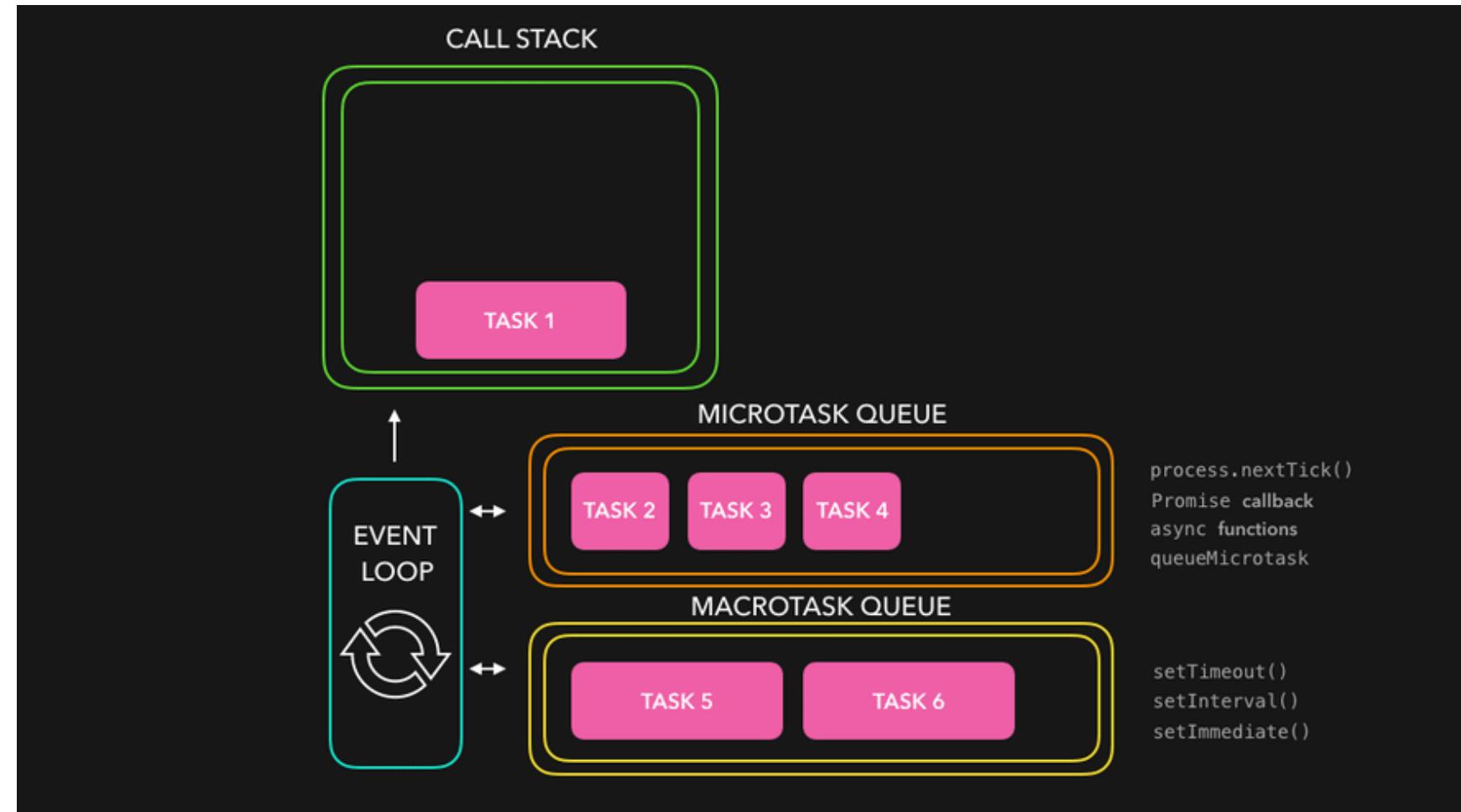
Arquitetura JavaScript – Microtask vs Macrotask

Macrotasks

- script tags in the DOM
- keyboard/mouse events
- setTimeout/setInterval

Microtasks

- Settled promises
- MutationObserver
- queueMicrotask



Fontes: [Microtasks and \(Macro\)tasks in Event Loop](#) | [JavaScript Microtask Vs. Macrotask - Differences in Event Loop](#)

Promises – Programação Assíncrona

Modelo Assíncrono Tradicional

```
function successCallback(result) {  
  console.log("It succeeded with " + result);  
}  
  
function failureCallback(error) {  
  console.log("It failed with " + error);  
}  
  
doSomething(successCallback, failureCallback);
```

Modelo Assíncrono com Promises

```
function successCallback(result) {  
  console.log("It succeeded with " + result);  
}  
  
function failureCallback(error) {  
  console.log("It failed with " + error);  
}  
  
const promise = doSomething();  
promise.then(successCallback, failureCallback);
```

Fontes

- [Usando promises - JavaScript | MDN \(mozilla.org\)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises)

Promises – Programação Assíncrona

Modelo Assíncrono Tradicional

```
doSomething(function(result) {  
  doSomethingElse(result, function(newResult) {  
    doThirdThing(newResult, function(finalResult) {  
      console.log('Final: ' + finalResult);  
    }, failureCallback);  
  }, failureCallback);  
}, failureCallback);
```

Modelo Assíncrono com Promises

```
doSomething()  
.then(result => doSomethingElse(result))  
.then(newResult => doThirdThing(newResult))  
.then(finalResult => {  
  console.log(`Final: ${finalResult}`);  
})  
.catch(failureCallback);
```

Fontes

- [Usando promises - JavaScript | MDN \(mozilla.org\)](#)

Promises

Uma **Promise** é um mecanismo que recebe uma função que por sua vez apresenta dois callbacks:

1. Uma operação **resolve** a ser executada caso a função seja bem sucedida
2. Uma operação **reject** a ser executada caso a função falhe

As Promises buscam simplificar a **programação assíncrona**. Recursos similares realizados, anteriormente, via bibliotecas como: jQuery ou deferred.js.

```
let p = new Promise(function (resolve, reject) {  
    // Executa operação demorada  
  
    if /* operação bem sucedida */ {  
        resolve("Valor a ser procesado");  
    }  
    else { /* operação falhou */  
        reject(Error("A operação falhou"));  
    }  
});
```

Fontes

- Promises in JavaScript explained whimsically - <https://medium.com/@kevinyckim33/what-are-promises-in-javascript-f1a5fc5b34bf>
- Working With Promises (Google) - <https://developers.google.com/web/ilt/pwa/working-with-promises>
- Promises Explained - <http://www.thedevnotebook.com/2017/02/javascript-promises.html>

Promises – Exemplo

```
function loadImage(url) {  
    // Gera uma Promise para carregar uma imagem  
    return new Promise(function (resolve, reject) {  
        // Cria uma imagem e atribui sua fonte (src)  
        var image = new Image();  
        image.src = url;  
  
        image.onload = function () { // Se der erro, executa callback resolve (then)  
            resolve(image);  
        };  
        image.onerror = function () { // Se der erro, executa callback reject (catch)  
            reject(new Error('Could not load image at ' + url));  
        };  
    });  
}
```

Função que retorna uma Promise para tratar eventos após o carregamento de uma imagem

Exemplo de Uso

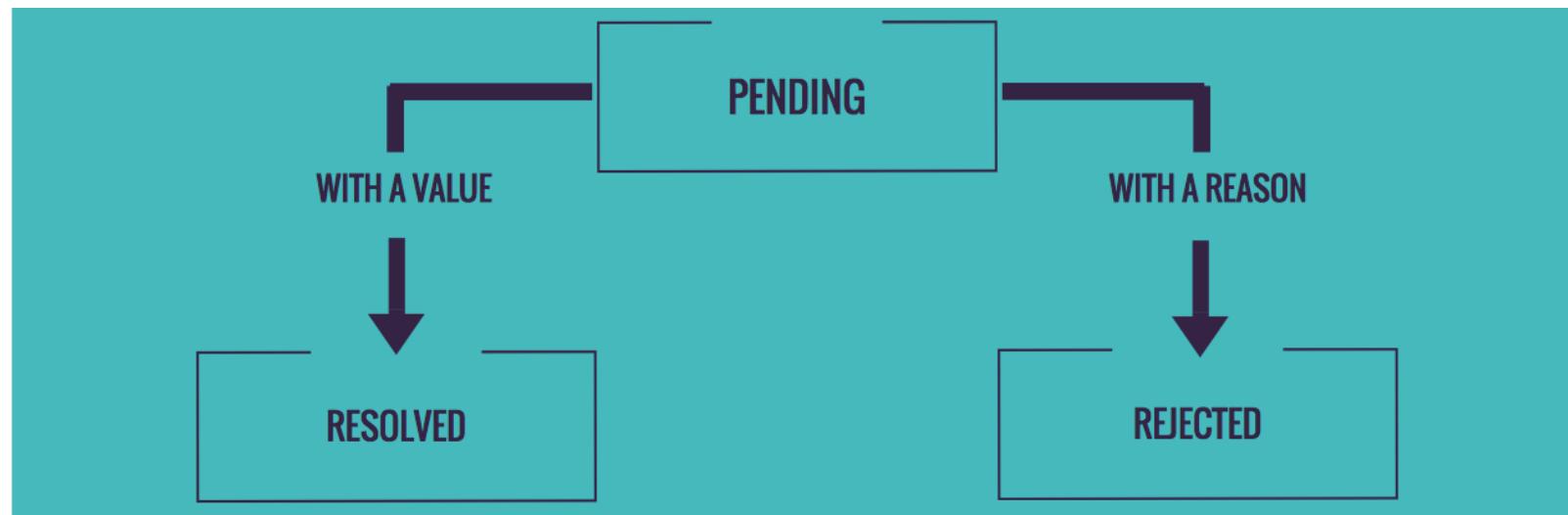
```
loadImage ('http://lorempixel.com/200/200/people')  
.then (imgElem => document.body.appendChild (imgElem))
```

Fonte: Working With Promises (Google) - <https://developers.google.com/web/ilt/pwa/working-with-promises>

Promises

Estados de uma promise – Possíveis estágios de uma promise no tempo

1. Pendente (pending) – A operação disparada ainda está sendo executada
2. Resolvida (resolved) – A operação teve êxito e executa-se a função **resolve**
3. Rejeitada (rejected) – A operação falhou e executa-se a função **reject**



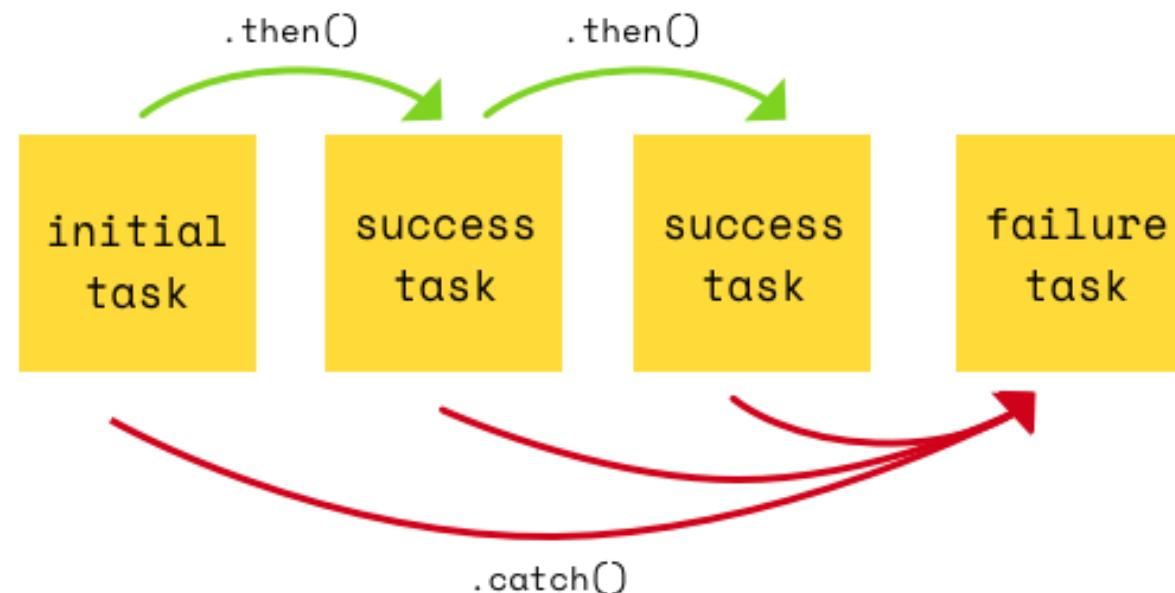
Promises

Encadeamento de Promises

Ao receber uma Promise, pode-se disparar

o método **then** para executar uma operação no caso de sucesso ou
o método **catch** no caso de uma falha.

Os dois métodos retornam outra Promise, permitindo o encadeamento de novas operações.



Promises – Exemplo – Fetch API

Encadeamento de Promises – Passo a passo da requisição AJAX com Fetch

1. Fetch executa requisição AJAX, e retorna uma Promise objeto de **resposta (res)**
2. Via **then**, encadeamos função para tratar a **resposta (res)**, retornamos um **JSON (obj)**
3. Via **then**, encadeamos função que trata **obj**, um array, e converte em outro array **usernames**
4. Via **then**, encadeamos função que imprime array **usernames** de nomes no console

```
1  fetch('https://jsonplaceholder.typicode.com/users')  
2    .then(res => res.json())  
3    .then(obj => obj.map(user => user.username))  
4    .then(userNames => console.log(userNames));
```

Promises – All

Promise.all permite criar tratamentos que são executados quando várias promises forem concluídas.

O resultado passado como parâmetro para a função de callback possui um array com o retorno de cada uma das promises processadas.

```
Promise.all ([  
    fetch('http://httpbin.org/delay/5'),  
    fetch('http://httpbin.org/delay/3'),  
]).then(result => console.log('Ok', result))
```

Fonte: [Programação Assíncrona em JavaScript - do básico ao avançado - Speaker Deck](#)

Referências

- **ECMAScript 2015 Language Specification**
<http://www.ecma-international.org/ecma-262/6.0/>
- **The ES6 Guide**
<https://flaviocopes.com/es6/>
- **ECMAScript 6 New Features: Overview & Comparison**
<http://es6-features.org>
- **Exploring ES6**
<http://exploringjs.com/es6/>
- **ECMAScript 6 - Luke Hoban** → **Fonte de parte dos códigos de exemplo deste material**
<https://github.com/lukehoban/es6features>
- **Top 10 ES6 features by example**
<https://blog.pragmatists.com/top-10-es6-features-by-example-80ac878794bb>
- **Top 10 ES6 Features Every Busy JavaScript Developer Must Know**
<https://webapplog.com/es6/>

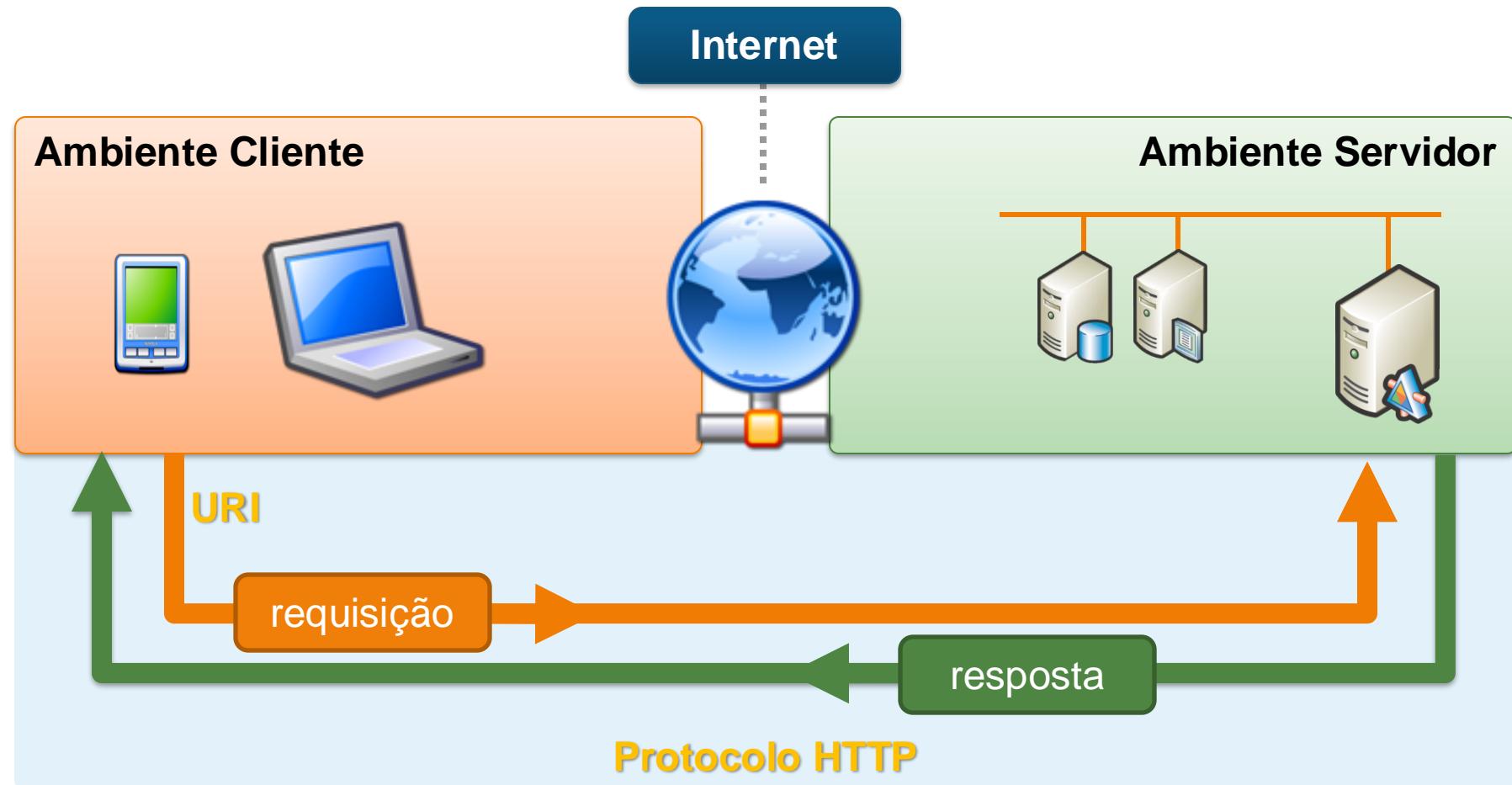
Plataforma Node.js

Arquitetura da Web

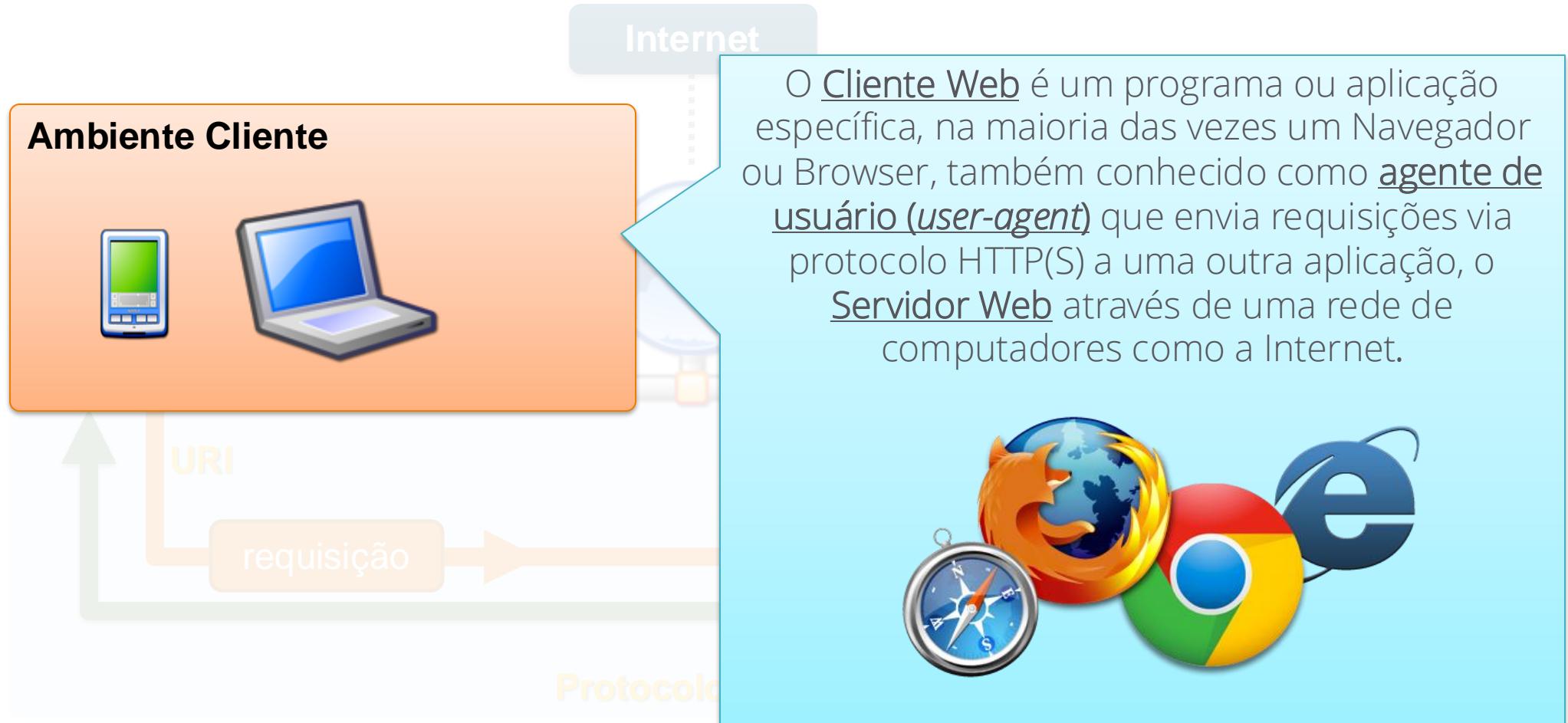
Plataforma Node.js

Arquitetura da Web
Visão Geral

Arquitetura da Web



Arquitetura da Web – Cliente



Arquitetura da Web – Servidor Web

O Servidor Web é um programa que recebe requisições HTTP(S), interpreta a URL e em seguida envia resposta ao Cliente Web com o recurso solicitado (arquivo HTML, CSS, JavaScript, imagens, vídeos, folhas de estilo) por meio da rede.

NGINX



Internet

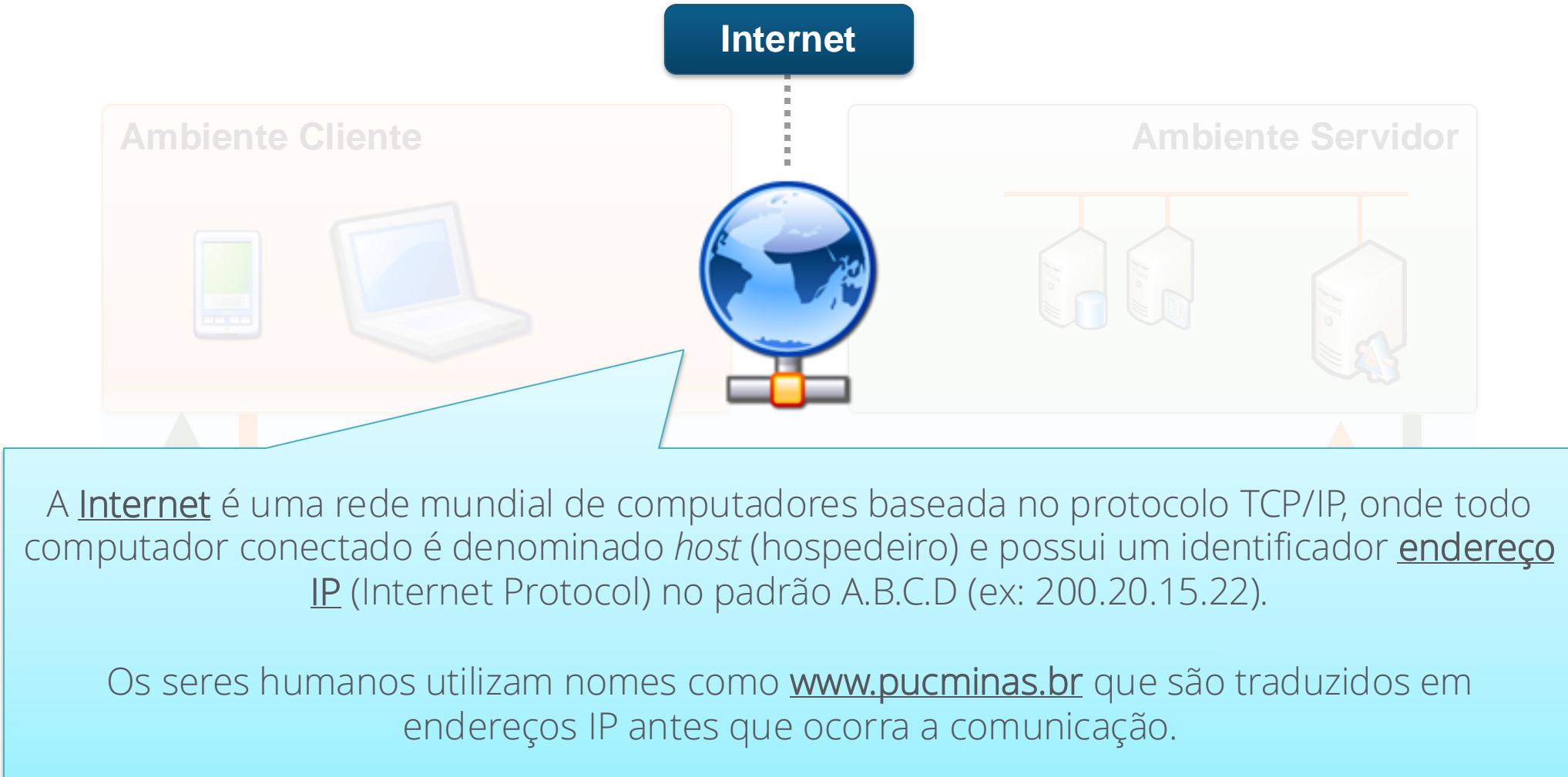
Ambiente Servidor



resposta

requisição HTTP

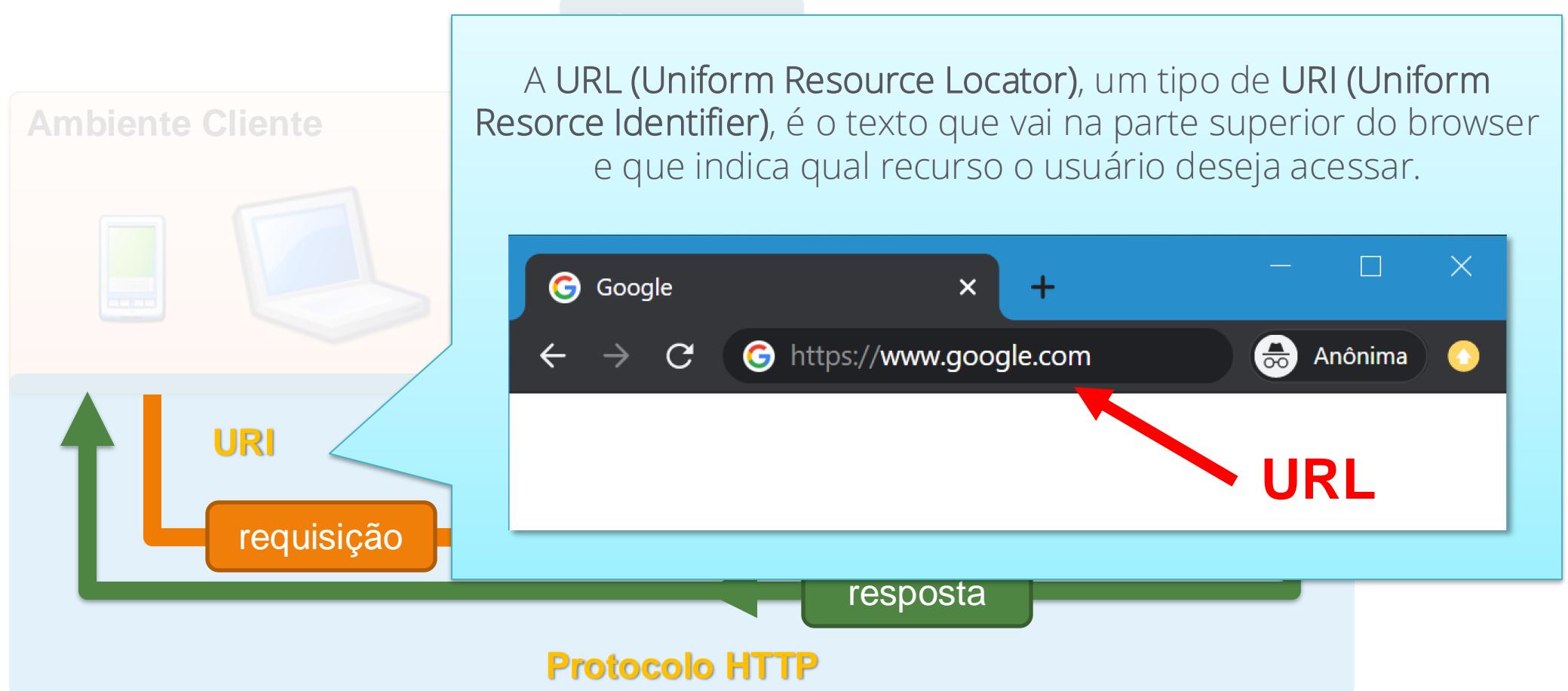
Arquitetura da Web – Internet



46



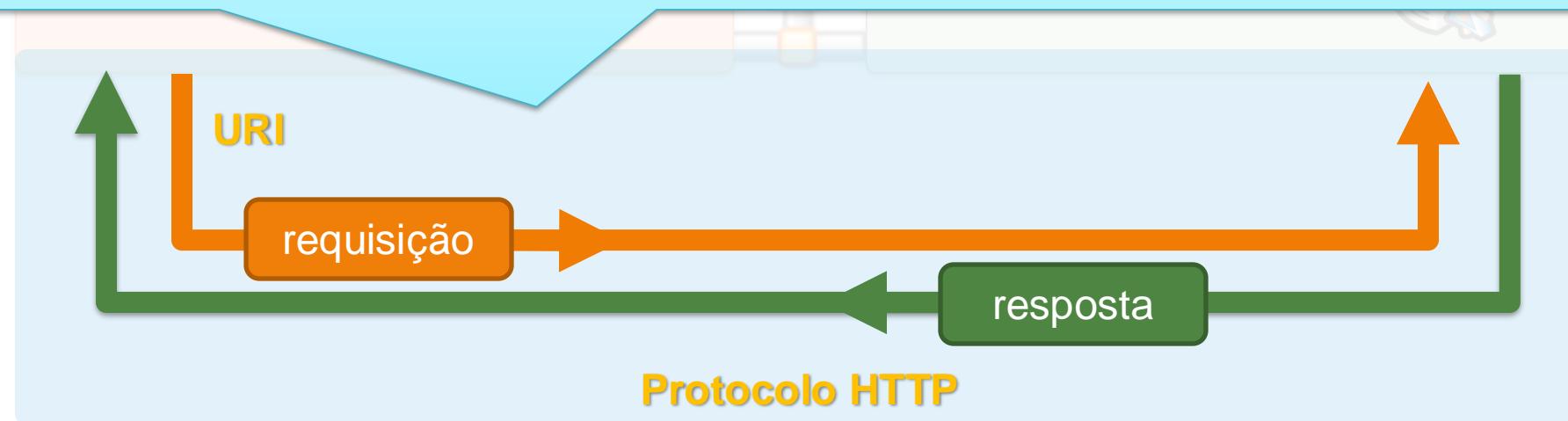
Arquitetura da Web – URI, URL e URN



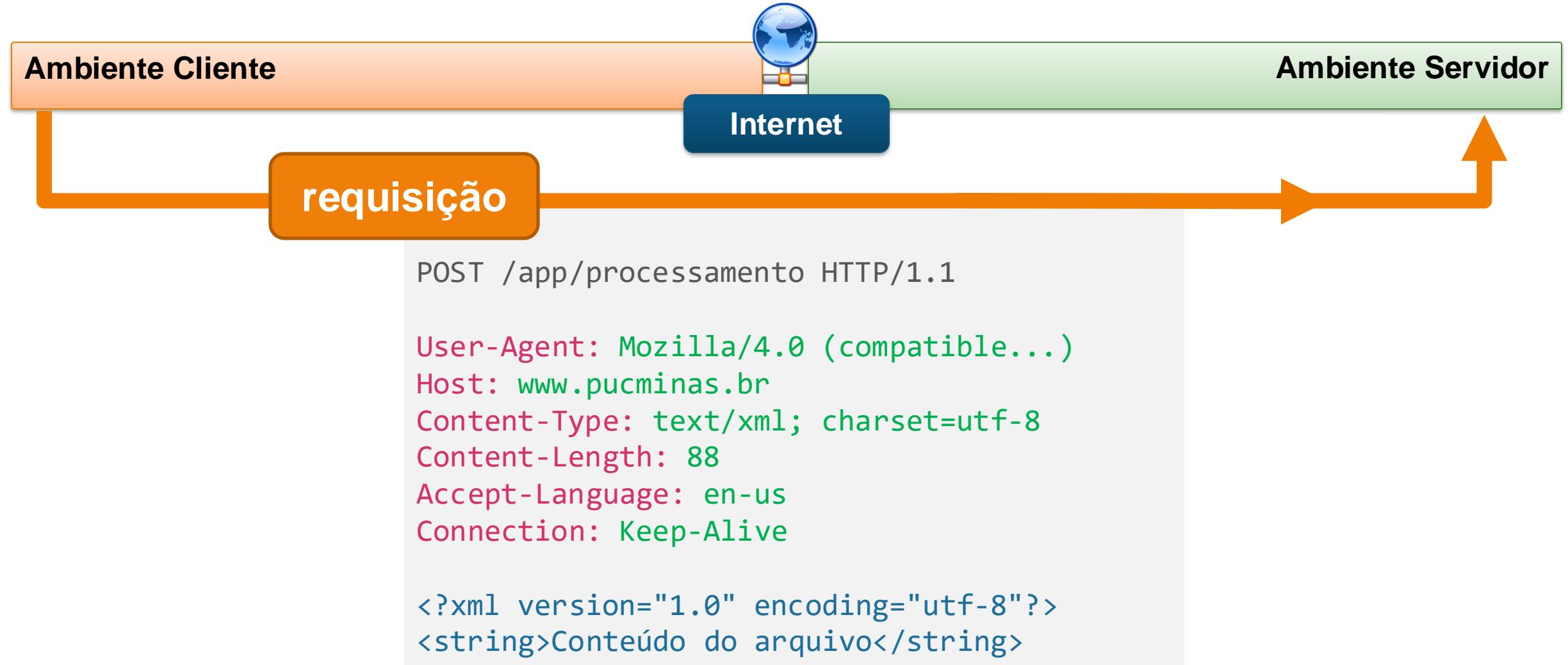
Arquitetura da Web – Protocolo HTTP

O protocolo HTTP é a forma como clientes e servidores se comunicam na rede. As requisições e as respostas obedecem aos padrões estabelecidos pelo protocolo HTTP.

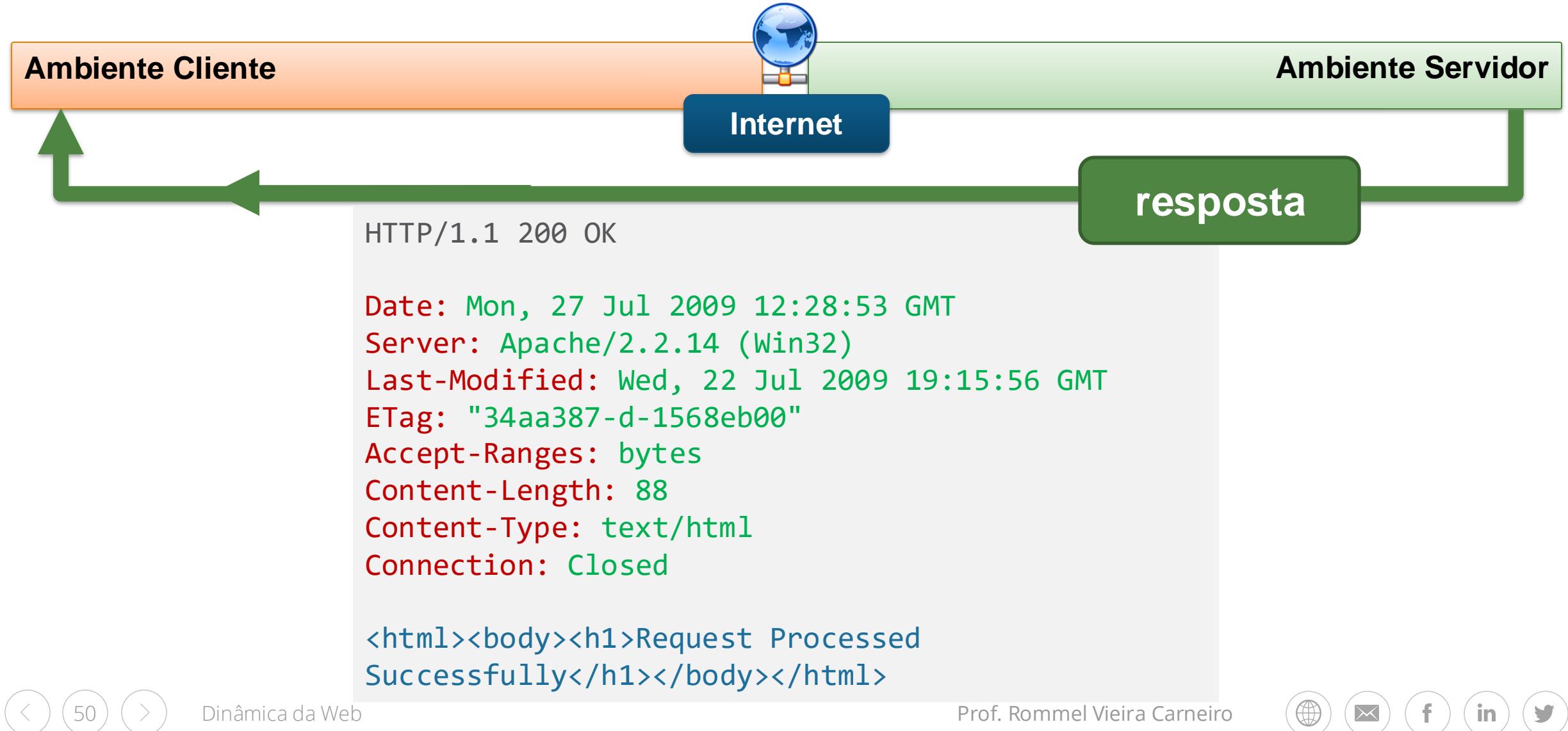
A requisição HTTP é um pacote de dados enviado pela rede pelo Cliente Web para o Servidor Web e identifica o recurso solicitado. A resposta HTTP é formada por pacotes de dados enviados pelo Servidor Web para o Cliente Web com os recursos solicitados.



Arquitetura da Web – Requisição e Resposta

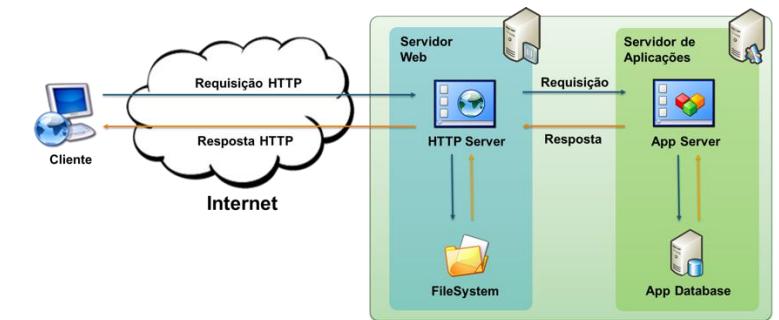


Arquitetura da Web – Requisição e Resposta

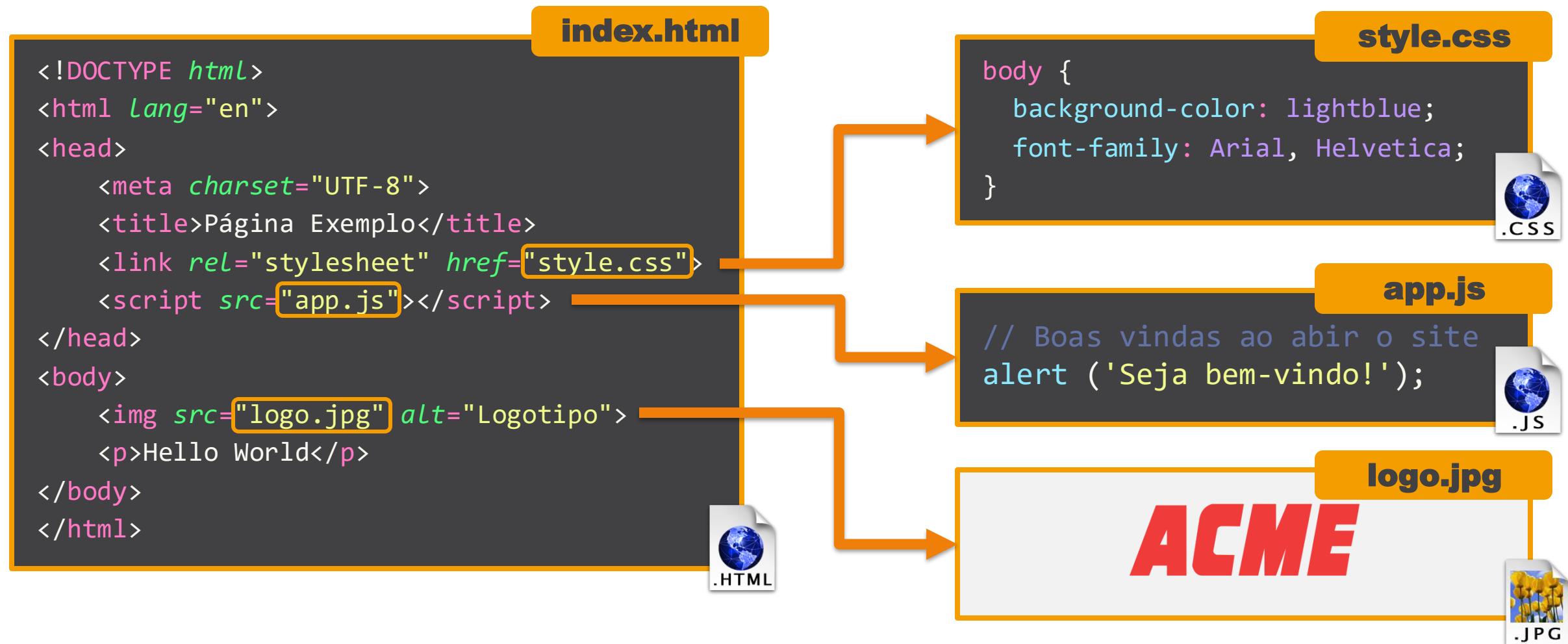


Dinâmica da Web

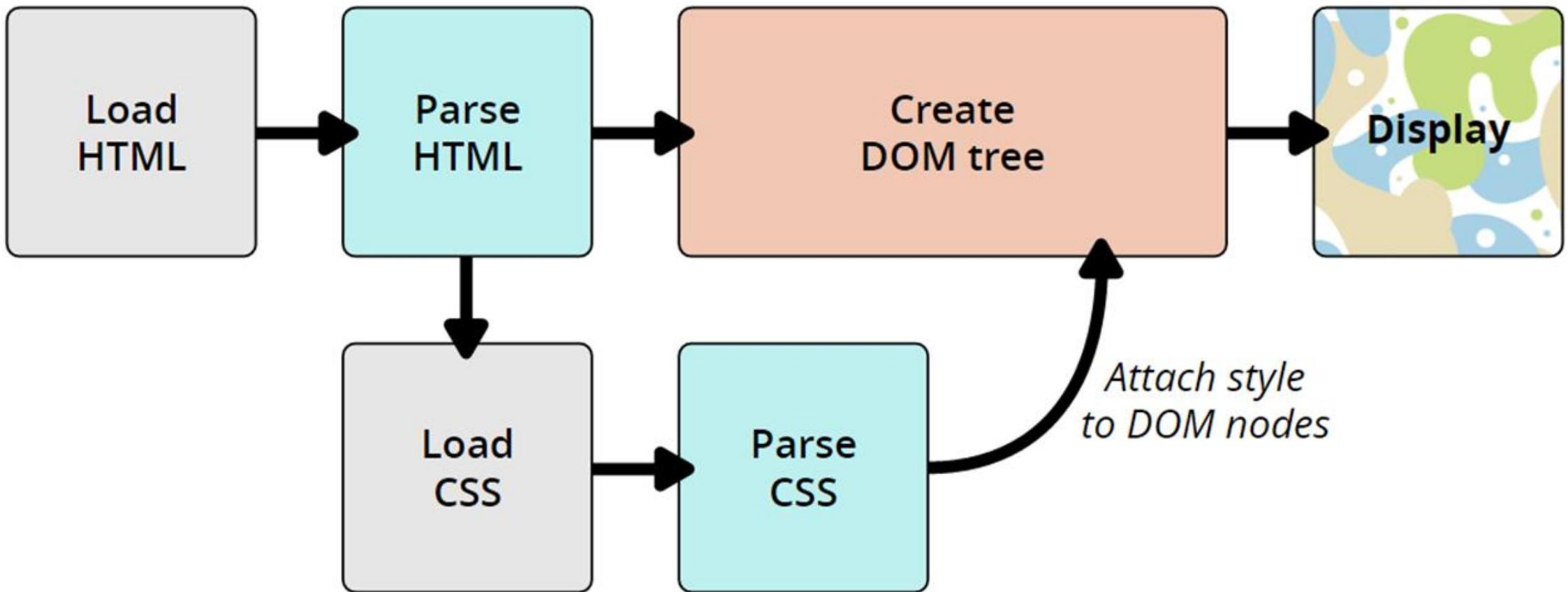
- Estrutura de um Site Web
- Site Estático
 - Processamento da página HTML e CSS
 - Processo de Navegação
- Site Dinâmico
 - Estrutura dos Servidores Web
 - Processo de Navegação



Dinâmica da Web – Estrutura de um Site Web

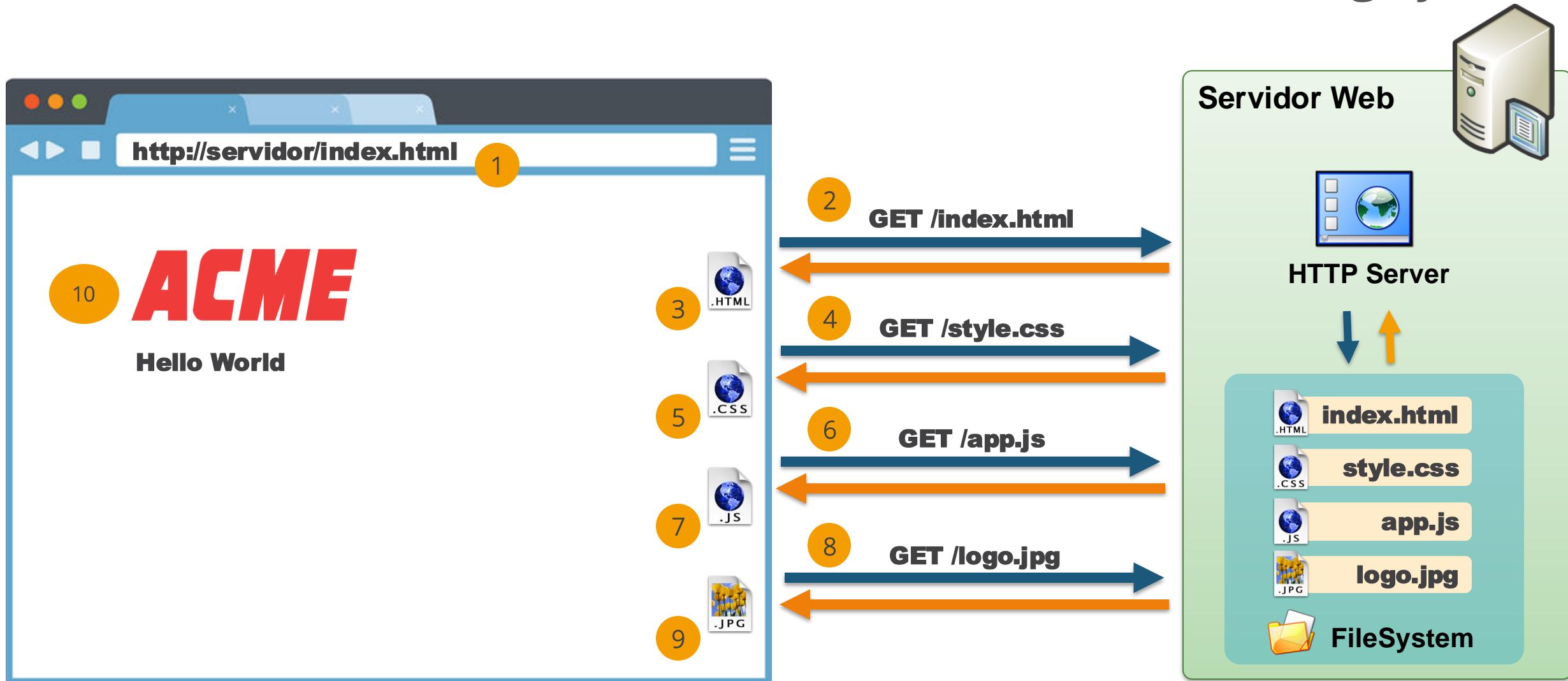


Dinâmica da Web – Estrutura de um Site Web



Fonte: Mozilla Developer Network - https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works

Dinâmica da Web – Site Estático – Processo de Navegação



Dinâmica da Web – Site Estático – Processo de Navegação

Passo 1 – Solicitação e recuperação da página index.html

1. Usuário informa a URL no Navegador
2. Navegador solicita a página inicial (index.html) ao Servidor Web
3. Servidor recupera o arquivo index.html e envia ao Navegador que interpreta em seguida

Passo 2 – Processamento do HTML, solicitação e recuperação do arquivo style.css

1. Navegador processa o HTML, identifica link p/ style.css e solicita o arquivo ao Servidor Web
2. Servidor Web recupera o arquivo style.css e envia para o Navegador

Passo 3 – Solicitação e recuperação do arquivo app.js

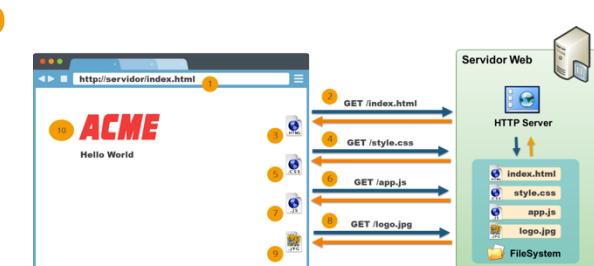
1. Navegador identifica link p/ app.js e solicita este arquivo ao Servidor Web
2. Servidor Web recupera o arquivo app.js e envia para o Navegador

Passo 4 – Solicitação e recuperação do arquivo logo.jpg

1. Navegador identifica link p/ logo.jpg e solicita este arquivo ao Servidor Web
2. Servidor Web recupera o arquivo logo.jpg e envia para o Navegador

Passo 5 – Apresentação da página completa para o usuário

1. Navegador apresenta a página para o Usuário



HANDS ON – Processo de Navegação

1) Abra as Ferramentas do Desenvolvedor

2) Escolha Rede

The screenshot shows the Chrome DevTools Network tab for the website www.pudim.com.br. The tab bar has 'Rede' selected. The main area displays a table of network requests:

Nome	Status	Tipo	Iniciador	Tamanho	Cascata
www.pudim.com.br	200	document	Outros	1.2 kB	[Timeline]
estilo.css	200	stylesheet	(índice)	755 B	[Timeline]
pudim.jpg	200	jpeg	(índice)	27.9 kB	[Timeline]
analytics.js	307	script / Redirecionar	(índice):21	0 B	[Timeline]
analytics.js	200	script	analytics.js	19.8 kB	[Timeline]

Below the table, the status bar shows: 5 / 7 solicitações 49.6 kB/50.1 kB transferidos 78.4 kB /78.6 kB recursos Concluir: 452 ms DOMContentLoaded: 140 ms Load: 410 ms

Annotations in pink text and arrows:

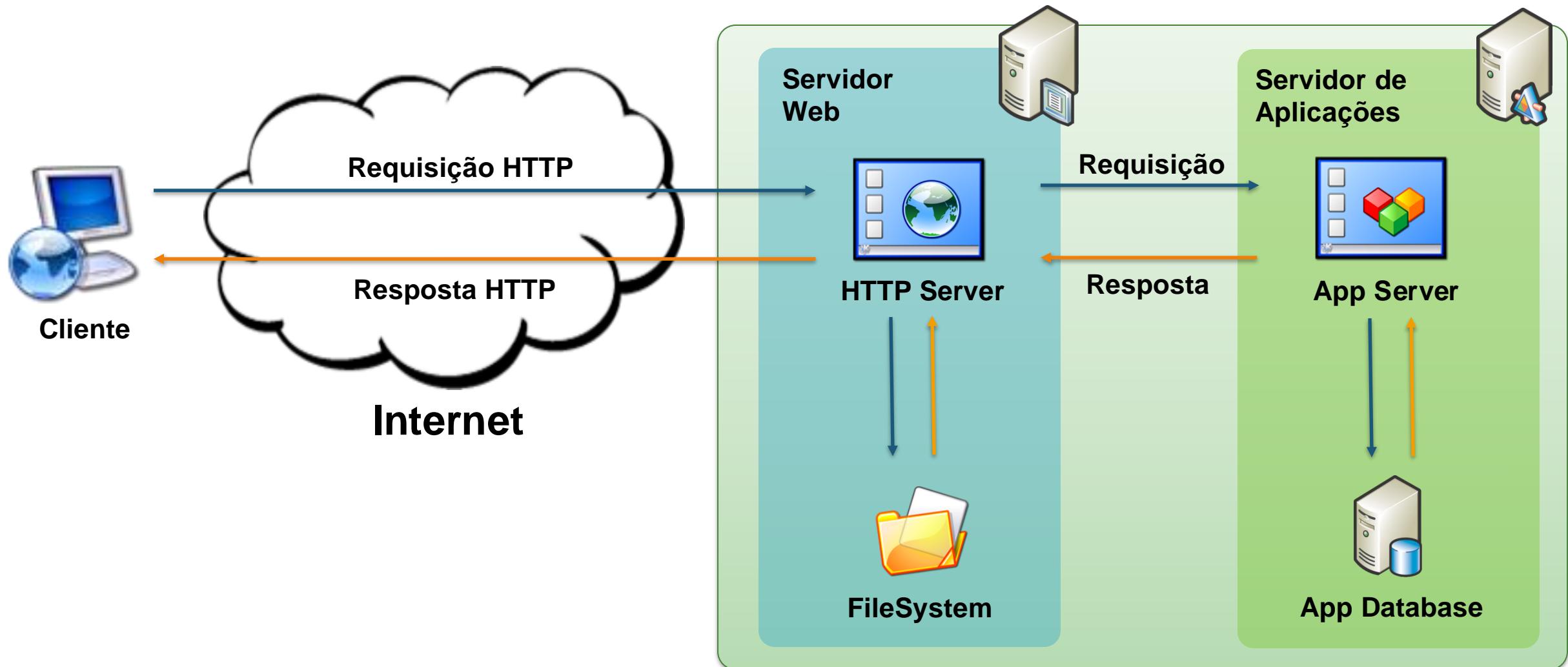
- 3) Filtre as requisições (points to the 'Filtrar' section at the top of the table)
- 4) Observe as requisições realizadas (points to the table of requests)
- 1) Abra as Ferramentas do Desenvolvedor (points to the three dots icon in the top right of the browser toolbar)
- 2) Escolha Rede (points to the 'Rede' tab in the top bar)



56



Dinâmica da Web – Estrutura de Servidores Web



Dinâmica da Web – Site Dinâmico - Processo de Navegação

Passo 1 – Solicitação e recuperação da página index.html

1. Usuário informa uma URL no Navegador ou aplicação solicita um recurso ao Servidor Web
2. Servidor processa a requisição e verifica se trata-se de recurso estático ou aplicação Web correspondente

Encaminhamento 1 – Recurso Estático

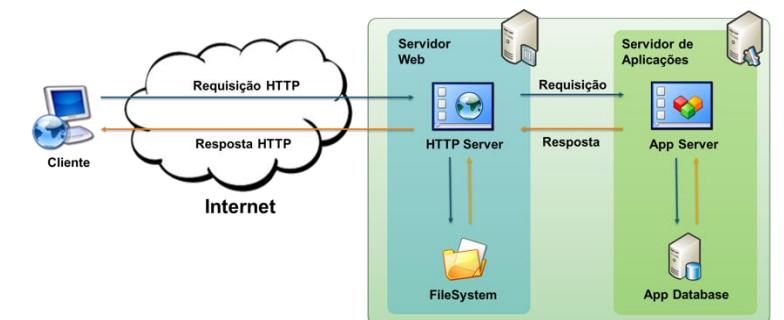
1. O Servidor Web recupera o recurso solicitado a partir do Sistema de Arquivos e envia para o Navegador

Encaminhamento 2 – Recurso de Aplicação (Dinâmico)

1. Servidor encaminha requisição para aplicação Web
2. Aplicação processa requisição, executa o algoritmo, gera o resultado e encaminha ao servidor Web
3. Servidor Web empacota o resultado gerando uma resposta HTTP e envia ao Navegador

Passo 2 – Apresentação do recurso para o usuário

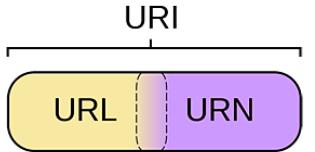
1. Navegador apresenta o recurso para o Usuário ou Aplicação processa recurso e apresenta o resultado para o Usuário



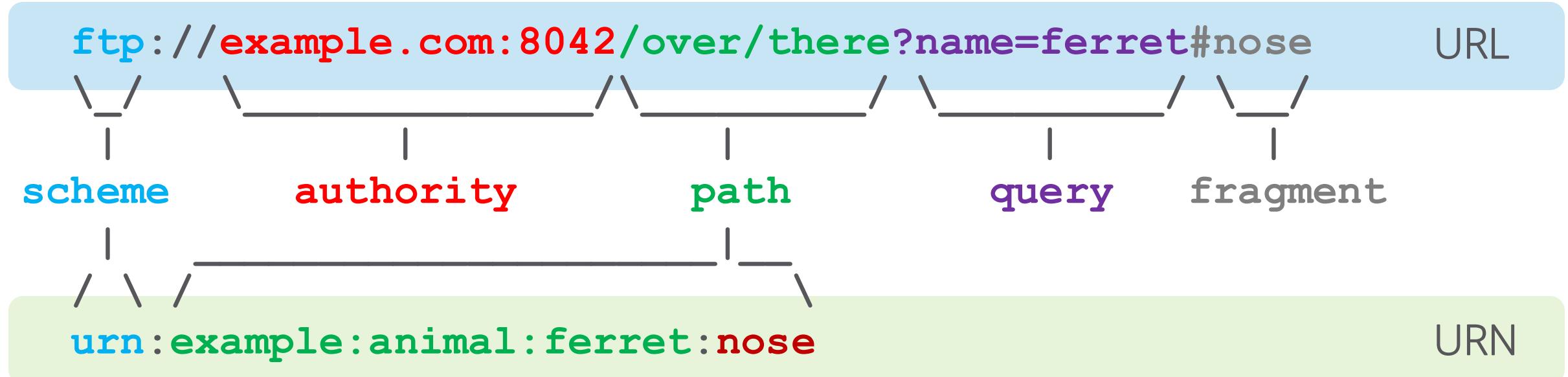
Plataforma Node.js

Arquitetura da Web
URI, URL e URN

URI, URL e URN – URI



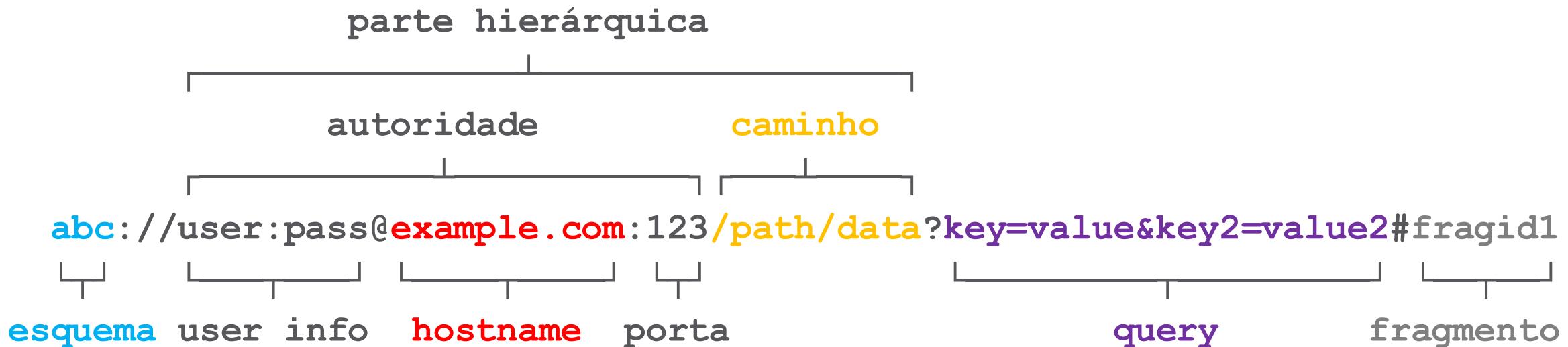
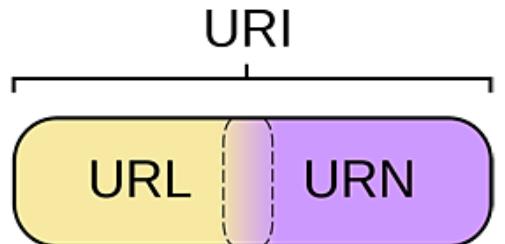
URI (Uniform Resource Identifier) é um padrão para o endereçamento de recursos disponíveis na rede que engloba os conceitos de **URL** (Uniform Resource Locator) e **URN** (Uniform Resource Name).



Fonte: Wikipedia - Uniform Resource Identifier - https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

URI, URL e URN – URL

URL (Uniform Resource Locator) é um padrão de URI que serve para referenciar um recurso e sua localização, normalmente na Internet.

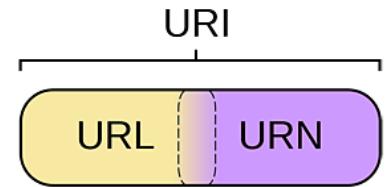


Fonte: Wikipedia - Uniform Resource Identifier - https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

URI, URL e URN – URL

Estrutura de um URL

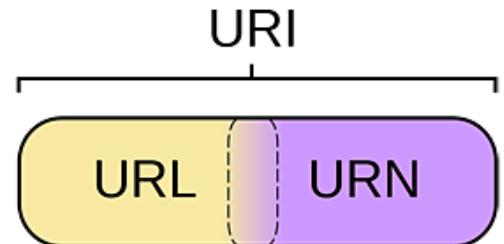
- **Esquema** – identifica a forma de interação entre um cliente e um servidor, como por exemplo http, https, ftp, entre outros
- **User:pass** – informações de usuário
- **Host** – nome ou número IP onde se encontra a aplicação servidor
- **Porta** – identifica a porta TCP/IP associada ao servidor. A porta padrão do HTTP (80) pode ser omitida
- **Caminho** – indica o local exato onde o recurso se encontra
- **Query** – dados não hierárquicos, detalhando a consulta normalmente sob a forma de pares nome e valor
- **Fragmento** – identifica uma seção no recurso



`esquema://user:pass@host:porta/caminho?query#fragmento`

URI, URL e URN – URN

URN (Uniform Resource Name) é um tipo de URI que identifica um recurso específico (NSS) pelo nome em um *namespace* (NID).



urn:example:mammal:monotreme:echidna

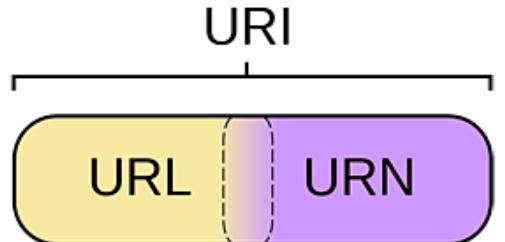
Legenda:

- NID – Namespace Identifier
 - NSS – Namespace Specific String

Fonte: Wikipedia - Uniform Resource Identifier - https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

URI, URL e URN – Exemplos

Exemplos de URI



URL → **esquema://user:pass@host:porta/caminho?query#fragment**

- **http://localhost:80/admin/index.php?z1=w1&z2=w2**
- **ftp://user:pass@server.net:21/documentos/arquivo.zip**
- **news://pl.com.os.linux**
- **telnet://192.168.1.1**

URN → **esquema:namespace_identifier:namespace_specific_string**

- **urn:isbn:978-1-491-91866-1**

Plataforma Node.js

Arquitetura da Web
Protocolo HTTP

Protocolo HTTP

O *Hypertext Transfer Protocol* (HTTP) é um protocolo da camada de aplicação para sistemas distribuídos e colaborativos de informação no formato de hipertextos.

RFC - 2068

Características

- Requer a atuação de dois programas: Cliente e Servidor
- Atua na camada de aplicação da pilha TCP/IP
- A comunicação utiliza conexões TCP (e UDP no caso do HTTP v3.0)
- O servidor HTTP, por padrão, utiliza a porta 80
- Protocolo que não guarda estado do cliente (stateless)

Histórico

- 1991 • **HTTP/0.9**
- 1994 • **HTTPS**
- 1996 • **HTTP/1.0**
- 1999 • **HTTP/1.1**
- 2009 • **SPDY 1.0**
- 2015 • **HTTP/2**
- 2016 • **QUIC**
- 2018 • **HTTP/3**

Protocolo HTTP – Histórico de Versões

- **1991** - O HTTP 0.9 é lançado
- **1994** - O HTTPS foi criado pela Netscape
- **1996** - O HTTP 1.0 foi lançado
 - Conceito de cabeçalhos
 - Códigos de Status
- **1999** - O HTTP 1.1 foi lançado
 - Conexões TCP persistentes
 - Suporte a Virtual Host (Cabeçalho Host)
 - Autenticação Digest
 - Controle de cache
 - Possibilidade de compressão de dados
- **2009** - Google propõe o SPDY
- **2015** - O HTTP 2.0 é lançado
 - Baseado no SPDY
 - Compressão de dados obrigatória
 - Cabeçalhos binários
 - Requisições paralelas
 - Envio apenas de cabeçalhos alterados nas próximas requisições
 - Priorização de requisições
 - Server PUSH – Envio automático de arquivos adicionais.
- **2018** – O HTTP 3 é lançado
 - Protocolo de transporte QUIC baseado em UDP

Protocolo HTTP – Requisição

Linha de Requisição



Linhas de Cabeçalho



Corpo da entidade



POST /app/processamento HTTP/1.1

User-Agent: Mozilla/4.0 (compatible...)
Host: www.pucminas.br
Content-Type: text/xml; charset=utf-8
Content-Length: 88
Accept-Language: en-us
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string>Conteúdo do arquivo</string>

Protocolo HTTP – Resposta

Linha de Resposta

versão HTTP

code status

msg status

Linhas de Cabeçalho

campo cabeçalho:

valor

...

campo cabeçalho:

valor

Corpo da entidade

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 88

Content-Type: text/html

Connection: Closed

<html><body>

<h1>Request Processed Successfully</h1>

</body></html>

Protocolo HTTP – Códigos de Retorno

Código	Propósito	Descrição
1xx	Informacional	Requisição recebida, processo em continuidade
2xx	Sucesso	A ação foi recebida, entendida e aceita
3xx	Redirecionamento	Ações adicionais devem ser executadas para completar o pedido
4xx	Erro no cliente	O pedido contém erro de sintaxe ou não pode ser completado
5xx	Erro no servidor	O servidor falhou em completar um pedido aparentemente válido

Exemplos mais comuns

- 200 – Ok
- 403 – Acesso negado
- 404 – Página não encontrada
- 500 – Erro interno do servidor



Protocolo HTTP – Métodos

Método	Propósito	Safe (readonly)	Idempotente
GET	Requisitar a representação de um recurso específico	Sim	Sim
POST	Enviar dados a serem processados por um recurso. Usado para incluir recursos ou submeter dados de processamento	Não	Não
HEAD	Similar ao GET, porém retorno deve ser somente do conjunto de cabeçalhos associados ao recurso solicitado	Sim	Sim
PUT	Requisitar a criação ou atualização de um recurso no servidor a partir dos dados no corpo da requisição	Não	Sim
DELETE	Excluir um recurso do servidor	Não	Sim
TRACE	Solicita ao servidor uma cópia (eco) da requisição. Usado para testar se a requisição foi alterada no caminho	Sim	Sim
PATCH	Utilizado para realizar alterações parciais de um recurso	Não	Não
OPTIONS	Usado pelo cliente para entender, ou descobrir, os métodos HTTP e outras opções suportadas por um servidor web	Sim	Sim
CONNECT	Usado quando o cliente estabelece uma conexão HTTPS com um servidor via um proxy	Não	Não

Protocolo HTTP – Métodos – GET

Método GET

- Tem por objetivo requisitar a representação de um recurso ao servidor
- **Por definição, não deve alterar o estado do servidor (safe)**
- As requisições podem ser mantidas em cache (favoritos ou bookmarks)
- Envia dados ao servidor via parâmetros na *query string* que ficam visíveis na URL
- Tem restrição quanto ao tamanho e ao formato das informações enviadas ao servidor
 - Formato: limitado a caracteres textuais (ASCII) incluídos na *query string*
 - Tamanho:
 - Apache: 4.000 caracteres
 - MS IIS: 16.384 caracteres
 - Tomcat: padrão 8.192 podendo chegar até 65.536 caracteres

Protocolo HTTP – Métodos – GET

Método GET

- Este é o método mais utilizado em aplicações Web.
- Ao informar uma URL em um navegador, o usuário está disparando uma requisição do tipo GET

Requisição

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
Host: www.pucminas.br
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Response

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
  <body> <h1>Hello, World!</h1> </body>
</html>
```

Protocolo HTTP – Métodos – POST

Método POST

- Envia dados ao servidor para serem processados
- **Por definição tem objetivo de alteram o estado do servidor (*not safe*)**
- Pode enviar dados via query string ou via corpo da requisição
 - Os dados enviados pelo corpo **não** ficam visíveis na URL
 - Muito utilizado para envio de dados sensíveis como senhas de acesso
- Não podem ser ‘favoritados’ (bookmarked)
- Não possuem restrição quanto ao tamanho e ao tipo de dados a serem enviados ao servidor

Protocolo HTTP – Métodos – POST

Método POST

- Normalmente é utilizado em conjunto com formulários HTML
- Observe os dados enviados no corpo da Requisição

Requisição

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
Host: www.pucminas.br
Content-Type: text/xml; charset=utf-8
Content-Length: 88
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string
</string>
```

Resposta

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html><body><h1>Request Processed
Successfully</h1></body></html>
```

Protocolo HTTP – Métodos – HEAD

Método HEAD

Possui estrutura e objetivo similar às requisições de GET, porém o servidor deve enviar apenas o conjunto de cabeçalhos associados ao recurso informado.

Requisição

```
HEAD /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
Host: www.pucminas.br
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Resposta

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

Protocolo HTTP – Métodos – PUT

Método PUT

Requisita a criação ou atualização de um recurso no servidor a partir dos dados no corpo da requisição. Utilizado no upload de arquivos para servidores Web.

Requisição

```
PUT /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
Host: www.pucminas.br
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 182

<html><body>
<h1>Hello, World!</h1>
</body></html>
```

Resposta

```
HTTP/1.1 201 Created
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>The file was created.</h1>
</body>
</html>
```

Protocolo HTTP – Métodos – DELETE

Método DELETE

Solicita ao servidor a exclusão de dados ou representações associados ao recurso informado.

Requisição

```
DELETE /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
Host: www.pucminas.br
Accept-Language: en-us
Connection: Keep-Alive
```

Resposta

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html><body><h1>URL deleted.</h1></body></html>
```

Protocolo HTTP – Métodos – TRACE

Método TRACE

Usado para ecoar o conteúdo de uma requisição HTTP ao servidor. Usado para verificar se a requisição é alterada no caminho por agentes intermediários (servidores de cache ou proxy).

Requisição

```
TRACE / HTTP/1.1  
Host: www.pucminas.br  
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
```

Resposta

```
HTTP/1.1 200 OK  
Date: Mon, 27 Jul 2009 12:28:53 GMT  
Server: Apache/2.2.14 (Win32)  
Connection: close  
Content-Type: message/http  
Content-Length: 39  
  
TRACE / HTTP/1.1  
Host: www.pucminas.br  
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
```

Protocolo HTTP – Métodos – OPTIONS

Método OPTIONS

Usado pelo cliente para descobrir os métodos HTTP e outras opções suportados por um servidor web. O cliente pode especificar uma URL para o método de opções ou um asterisco (*) para se referir a todo o servidor.

Requisição

```
OPTIONS * HTTP/1.1  
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
```

Resposta

```
HTTP/1.1 200 OK  
Date: Mon, 27 Jul 2009 12:28:53 GMT  
Server: Apache/2.2.14 (Win32)  
Allow: GET,HEAD,POST,OPTIONS,TRACE  
Content-Type: httpd/unix-directory
```

Protocolo HTTP – Métodos – CONNECT

Método CONNECT

Usado pelo cliente para estabelecer uma conexão com o servidor web que pode ser via protocolo seguro (TLS). É utilizado no caso de requisições a proxies.

Requisição

```
CONNECT www.pucminas.br HTTP/1.1  
User-Agent: Mozilla/4.0 (compatible; MSIE5.01)
```

Resposta

```
HTTP/1.1 200 Connection established  
Date: Mon, 27 Jul 2009 12:28:53 GMT  
Server: Apache/2.2.14 (Win32)
```

Protocolo HTTP – Dados trafegados

- A transmissão via HTTP pode trafegar dados em formato texto ou binário
- Uma requisição deve especificar via cabeçalho **Content-Type**

MIME (Multipurpose Internet Mail Extensions)

- Os valores expressos no cabeçalho **Content-Type** seguem o padrão denominado MIME
- Abaixo são apresentados alguns exemplos de tipos MIME
 - **Image/jpg**: transmissão de imagens (jpe, jpg, jpeg, ...)
 - **text/html**: transmissão de textos em HTML
 - **x-application/java**: transmissão de classes java (.class)

GET vs POST

http://www.w3schools.com/tags/ref_httpmethods.asp

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL



Protocolo HTTP – Cabeçalhos

Os cabeçalhos utilizados em requisições e respostas do protocolo HTTP carregam informações adicionais sobre a comunicação entre cliente e servidor.

Tipos de Cabeçalhos

- **Request header:** informações sobre a requisição feita ou sobre o cliente Web.
- **Response header:** informações sobre a resposta encaminhada ou sobre o servidor Web.
- **Entity header:** informações sobre o conteúdo da entidade trocada como tamanho e tipo.
- **General header:** Usado tanto em requisições quanto em respostas.

POST /app/processamento HTTP/1.1

User-Agent: Mozilla/4.0 (compatible...)
Host: www.pucminas.br
Content-Type: text/xml; charset=utf-8
Content-Length: 88
Accept-Language: en-us
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string>Conteúdo do arquivo</string>

Protocolo HTTP – Cabeçalhos de Requisição

Cabeçalho	Utilidade e exemplos	requisição
Accept	<p>Lista os tipos de mídia aceitáveis para a resposta. Indica que a solicitação está limitada a um pequeno conjunto de tipos desejados.</p> <p>Accept: application/json</p> <p>Accept: text/html,application/xhtml+xml, application/xml;q=0.9,*/*;q=0.8</p>	
Accept-Charset	<p>Lista os conjuntos de caracteres que são aceitáveis para a resposta.</p> <p>Accept-Charset: utf-8, iso-8859-1;q=0.5</p>	
Accept-Encoding	<p>Lista conjuntos de codificações que são aceitáveis para a resposta.</p> <p>Accept-Encoding: gzip, deflate</p>	
Accept-Language	<p>Lista os conjuntos de idiomas naturais aceitáveis e preferidos pelo usuário para a resposta.</p> <p>Accept-Language: pt-BT, en;q=0.9, *;q=0.8</p>	

Protocolo HTTP – Cabeçalhos de Requisição

Cabeçalho	Utilidade e exemplos	requisição
Authorization	Informa as credenciais de autenticação do User Agent Authorization: Basic SGxsdfRp32hgIKVrw5VzW1	
Host	Indica o host e a porta de onde o recurso está sendo solicitado Host: pucminas.br	
Referer	Informa a URL do recurso de origem, ou visitado antes da requisição atual e que, possivelmente, direcionou o usuário para este recurso referer: https://acesso.gov.br/login?id=acesso.gov.br	
User-Agent	Informa, ao servidor, detalhes sobre o user agent (cliente Web) que está enviando esta requisição user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36	

Protocolo HTTP – Cabeçalhos de Resposta

Cabeçalho	Utilidade e exemplos	resposta
Server	Informa detalhes do software que implementa o servidor Web Server: Apache/2.4.34 OpenSSL/1.0.2k-fips PHP/5.5.38	
Etag	Traz um identificador da versão do recurso que altera toda vez que este for alterado. É utilizado pelo controle de cache. ETag: "353-527867f65e8ad"	
Set-Cookie	Apresenta cookies a serem armazenados pelo cliente e que devem ser enviados ao servidor nas próximas requisições. set-cookie: MLPRICING=1; Domain=magazineluiza.com.br;	

Protocolo HTTP – Cabeçalhos de Resposta

Cabeçalho	Utilidade e exemplos	resposta
Location	Redireciona o cliente Web outra URI Location: https://www.pucminas.br/Paginas/main.aspx	
WWW-Authenticate	indica que o servidor requer a autenticação do usuário para ter acesso ao recurso e de que forma WWW-Authenticate: Basic realm="Site X", charset="UTF-8"	

Protocolo HTTP – Cabeçalhos de Entidade

Cabeçalho	Utilidade e exemplos	entidade
Content-Encoding	Indica uma modificação ao tipo de mídia empregado no conteúdo Content-Encoding: gzip	
Content-Language	Descreve a linguagem na qual o conteúdo foi criado (en, pt, etc) Content-Language: pt-br	
Content-Length	Indica a quantidade em número de bytes na notação decimal Content-Length: 17515	
Content-Location	Local alternativo para o recurso solicitado Content-Location: /index.htm	
Content-Type	Indica o tipo de mídia do conteúdo Content-Type: text/html; charset=utf-8	

Protocolo HTTP – Cabeçalhos de Entidade

Cabeçalho	Utilidade e exemplos	entidade
Expires	Informa a data de expiração do recurso recebido Expires: Sun, 31 Jul 2016 05:00:00 GMT	
Last-Modified	Informa a data e hora de última modificação do recurso no servidor Last-Modified: Tue, 06 Nov 2018 22:45:26 GMT	

Plataforma Node.js

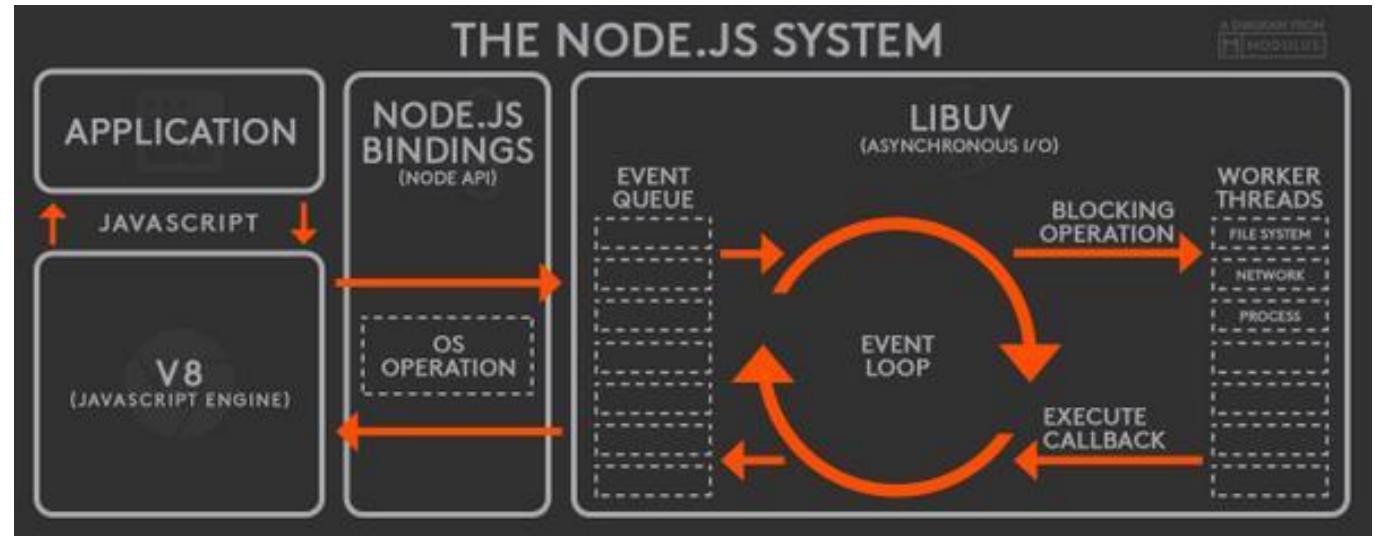
Arquitetura da plataforma Node.js

Arquitetura Node.js

A plataforma Node.js é orientada a eventos com modelo não bloqueante de I/O

Temas relacionados

- [JavaScript single Thread Programming](#)
- [Event-Driven Architecture \(EDA\)](#)
- [Blocking vs Non-Blocking](#)
- [Problema C10K](#)
- [Reverse proxy e Load balancing](#)



Outras Fontes:

- [Node.js Architecture – Idealwebtutor](#)
- [EDA em uma Arquitetura de Microserviços \(Marcelo M. Gonçalves\)](#)
- [What is a Reverse Proxy vs. Load Balancer? \(NGINX\)](#)

Arquitetura - Blocking vs Non-Blocking

Abordagem Blocking

```
const fs = require('fs');

// blocks here until file is read
const data = fs.readFileSync('/file.md');

console.log(data);

// will run after console.log
moreWork();
```

Abordagem Non-Blocking

```
const fs = require('fs');

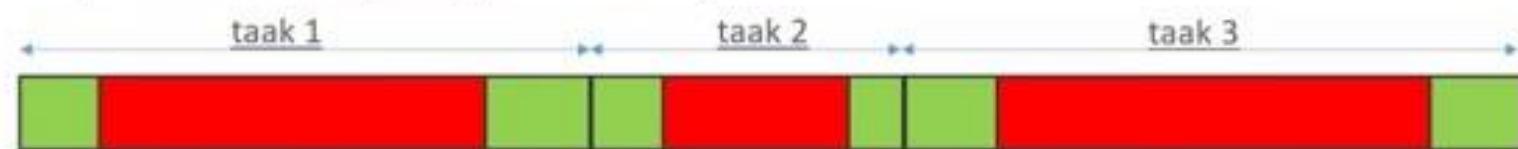
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
});

// will run before console.log
moreWork();
```

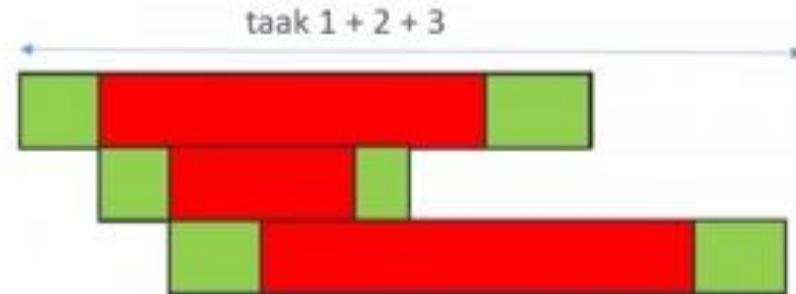
Fonte: [Overview of Blocking vs Non-Blocking | Node.js \(nodejs.org\)](https://nodejs.org/en/docs/guides/using=event-loop-blocking)

Arquitetura - Programação Síncrona vs Assíncrona

- Synchroon (sequentieel)

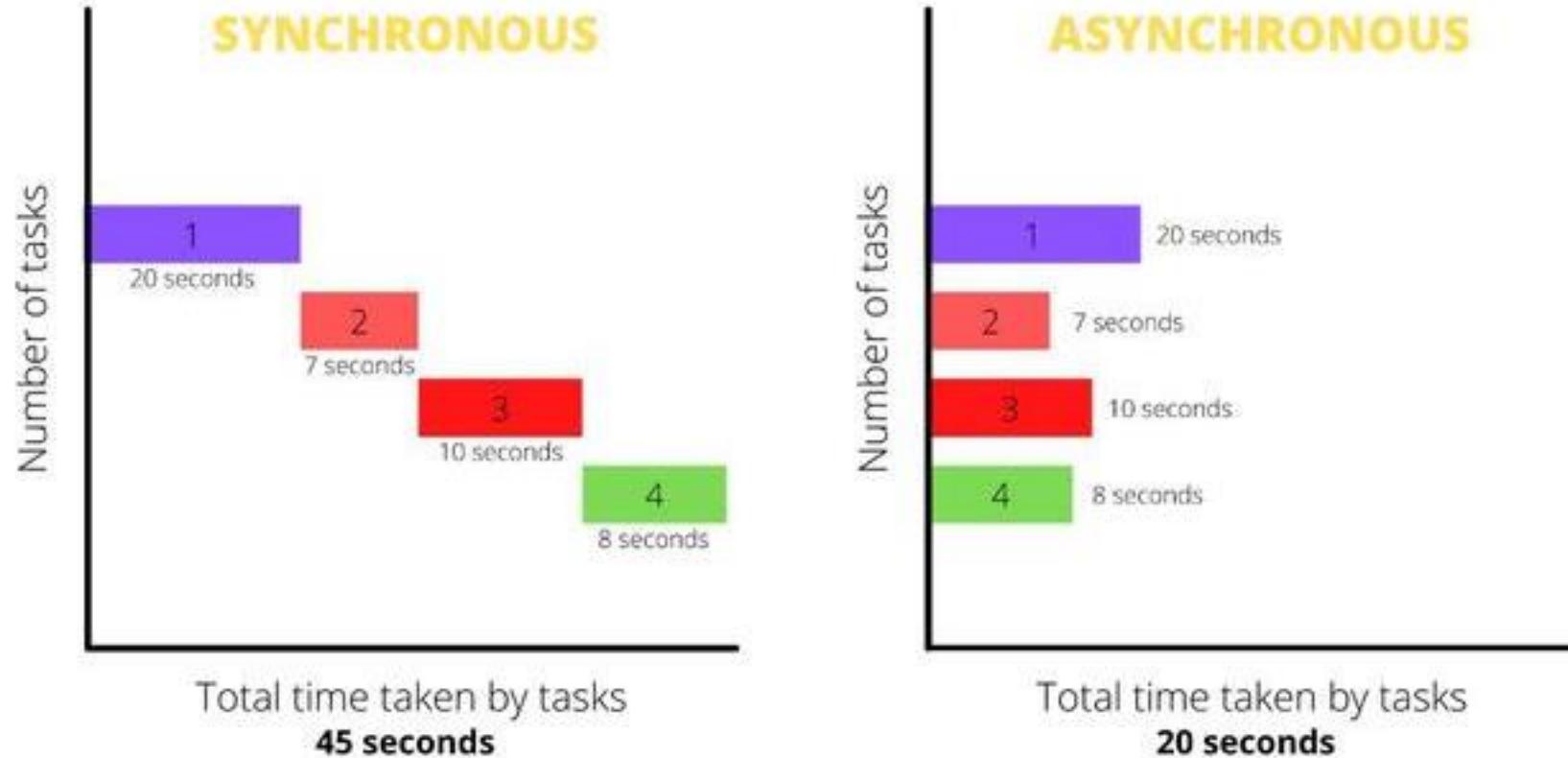


- Asynchroon (callbacks):



Fonte: [Introduction to Node.js \(SynTouch\)](#)

Arquitetura - Programação Síncrona vs Assíncrona



Fonte: [How Node.js overcome the problem of blocking of I/O operations?](#) (GeeksforGeeks)

Plataformas Web Tradicionais



Cliente



Servidor Web

- Autenticação
- Conectividade
 - Protocolo HTTP
 - Criptografia (SSL)
 - Domínios, IPs e Portas
 - Controle de Sites
 - Cache de recursos
- Gestão do FileSystem
- Auditoria (Logs de acesso, sistema e erros)



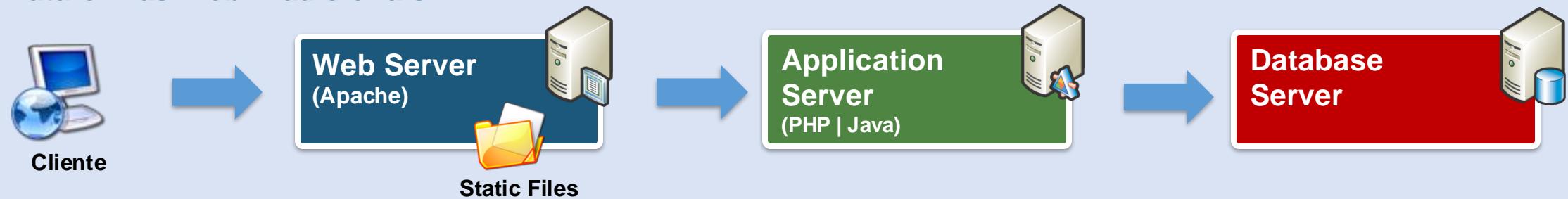
Servidor de Aplicações

- Gestão de Aplicações
- Controle de Sessões
- Controle de Transações
- Pool de recursos
- Integração com bancos de dados

OBS: Alguns Servidores de Aplicações possuem módulos que atuam como Servidor Web.

Plataformas Tradicionais x Plataforma Node.js

Plataformas Web Tradicionais



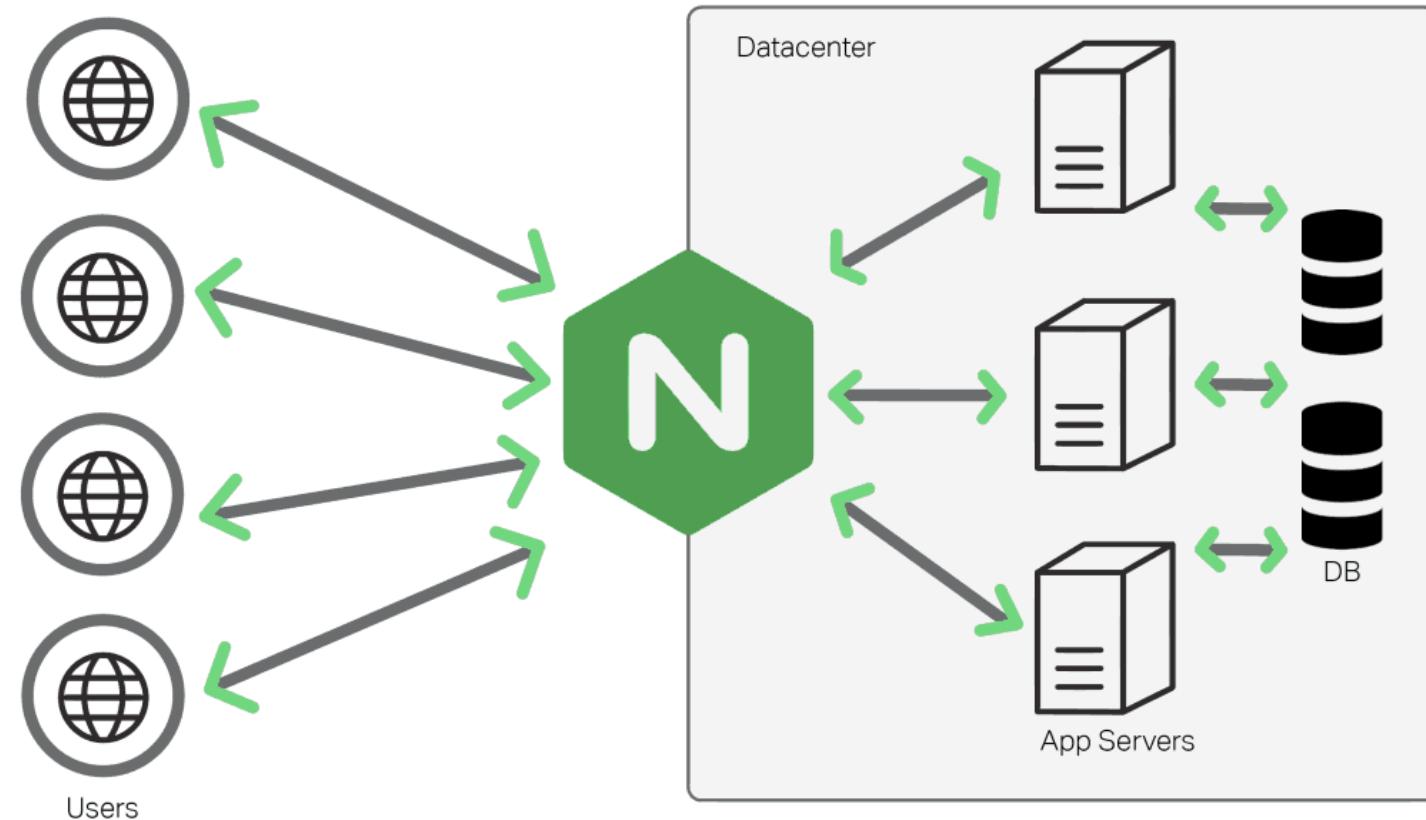
Plataforma Node.js



Plataforma Node.js Otimizada (NGINX + Proxy Reverso + Cluster Servers)



Arquitetura – Load Balancing e Proxy Reverso



Fonte:

- [What is a Reverse Proxy vs. Load Balancer? \(NGINX\)](#)
- [Nginx Reverse Proxy for Scalability](#)

Plataforma Node.js

Gestão de pacotes e módulos

Pacotes e Módulos

Pacote

Pode ser ...

- a) ... uma pasta contendo um programa descrito por um arquivo package.json
- b) ... um arquivo compactado (tar.gz) contendo (a)
- c) ... uma URL que direciona para (b)
- d) ... uma string <name>@<version> publicada no registro com (c)
- e) ... uma string <name>@<tag> que aponta para (d)
- f) ... uma string <name> que tenha uma tag latest que satisfaça (e)
- g) ... uma URL git que, quando clonada, resulta em (a)

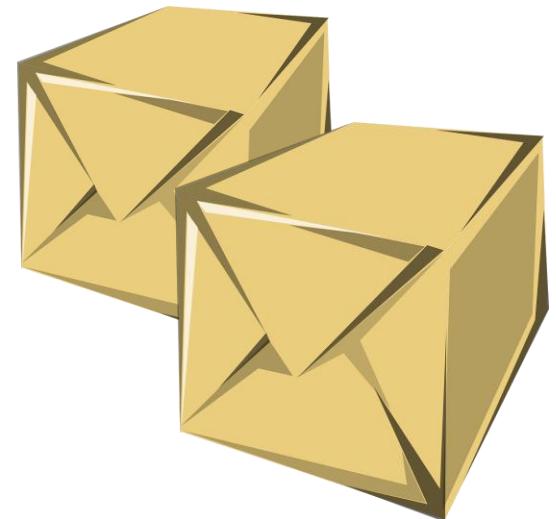
Módulo

Pode ser ...

- ... um arquivo Javascript
- ... uma pasta com um arquivo package.js que tenha um atributo main

Três tipos de módulos

- Módulos internos (core)
ex: process, os, dns
- Módulos locais
ex: lib_a, lib_b
- Módulos de terceiros
ex: express, mongoose



Fonte: [About packages and modules](#) (npm Docs)

Fonte da imagem: [Package clipart](#)

Gerenciadores de Pacotes e Dependências

Os **gerenciadores de pacotes** auxiliam o desenvolvedor nas tarefas de instalação, atualização, configuração e remoção de programas e componentes (**Pacotes**).

Os **gerenciadores de dependências** atuam de forma similar, automatizando tarefas de download e configuração de pacotes, porém, com foco em um projeto específico.



Enquanto um **gerenciador de pacotes** permite configurar todo o ambiente de desenvolvimento, um **gerenciador de dependências** configura um projeto específico.



- O NPM é, ao mesmo tempo, Gerenciador de Pacotes e um Gerenciador de dependências para o ambiente JavaScript e Node.Js
- O NPM é instalado juntamente com o Node.Js
- Possui um repositório de mais de 600.000 pacotes
- Possui versão pública e versão corporativa

Web Site: <https://www.npmjs.com/>

NPM – Contexto Global x Contexto Local



Contexto Global

- Refere-se ao computador como um todo
- Pasta no Linux e MacOs: /usr/local/lib/node_modules
Pasta no Windows: %app_data%\Roaming\npm\node_modules

Contexto Local

- Refere-se ao projeto específico
- Configurações mantidas no arquivo **package.json**
- Módulos mantidos no diretório **node_modules**
- Utilize o npx para executar partes dos módulos localmente

```
$ npx nodemon // Executa o nodemon instalado localmente no projeto
```

NPM – Primeiros passos



Nesta parte vamos verificar a versão do NPM e, caso necessário, atualizá-lo. Em seguida, vamos inicializar um projeto, criando o arquivo de configurações com nome **package.json** na pasta raiz. Serão solicitadas as informações referentes ao módulo que compõem o arquivo de configurações.

```
$ npm -version          // Verifica a versão do NPM  
5.6.0  
  
$ npm install npm@latest -g    // Atualiza a versão do NPM  
  
$ npm init                 // Inicializa o arquivo package.json para o projeto
```

NPM - Instalação de Módulos



Os módulos podem ser instalados localmente ou globalmente.

Os módulos locais são colocados na subpasta **node_modules**, dentro do projeto e podem compor uma lista de dependências do projeto no arquivo **package.json**.

```
$ npm install express          // Instalação do pacote express (EXEMPLO)
// localmente no projeto atual

$ npm install express@3.0.0      // Instalação de versão específica

$ npm install express --save    // Opção --save informa para incluir pacote
// nas dependências do projeto

$ npm install nodemon -g        // Instalação do pacote globalmente

$ npm install nodemon --save-dev // Opção -save-dev informa para incluir pacote
// nas dependências de desenvolvimento do projeto
```

NPM – Estrutura do arquivo package.json



```
{  
  "name": "projeto_teste",  
  "version": "1.0.0",  
  "description": "Projeto de Teste",  
  "main": "app.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "Rommel Carneiro",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.16.3"  
  }  
}
```

NPM – Estrutura do arquivo package.json



Atributo	Descrição
name	Nome do pacote
version	Versão do pacote
description	Descrição do pacote
homepage	Site Web do pacote
author	Autor do pacote
contributors	Nome dos colaboradores do pacote
dependencies	Lista de dependências associadas ao pacote
devDependencies	Lista de dependências do ambiente de desenvolvimento
main	Ponto de entrada do pacote
scripts	Comandos de script para o ambiente de desenv.
keywords	Palavras-chave associadas ao pacote

NPM – Estrutura do arquivo package.json



Dependências de produção

Utilizadas durante a execução do software e imprescindíveis para que o software possa ser executado.

Dependências de desenvolvimento

Pacotes úteis para o processo de desenvolvimento, nas etapas de construção, testes e implantação. Não são necessárias em um ambiente de produção.

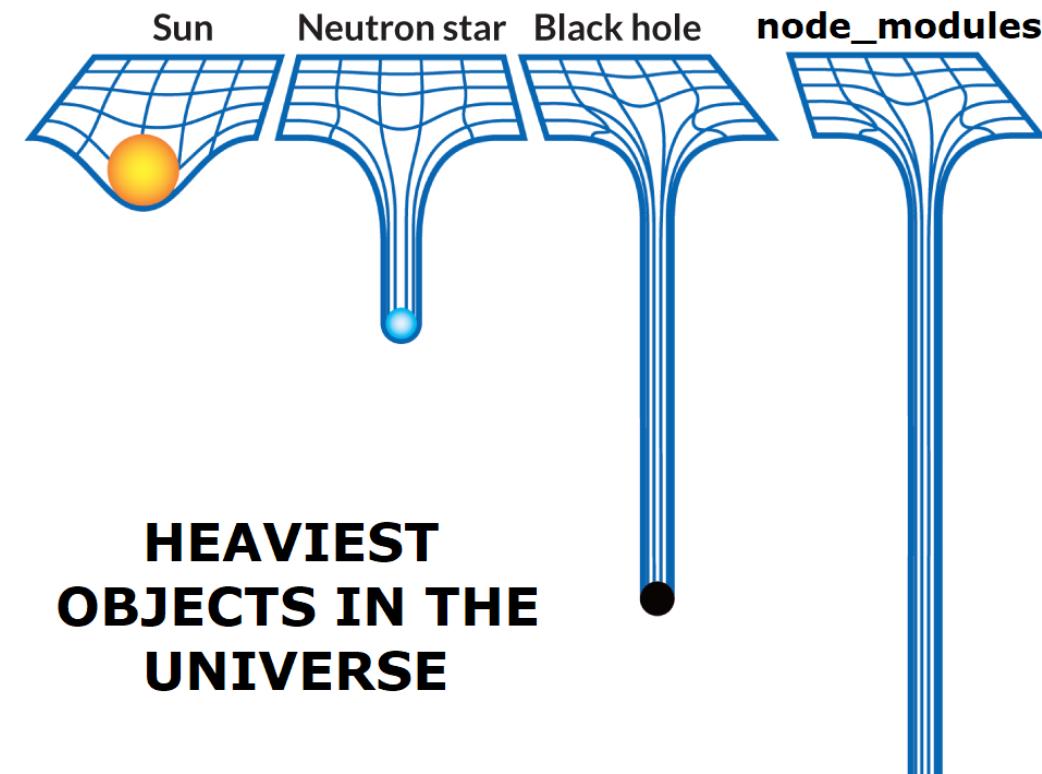
```
$ npm install express --save          // Instalação de dependências de produção
```

```
$ npm install nodemon --save-dev      // Instalação de dependências de desenvolvimento
```

NPM – Pasta node_modules



- Repositório dos módulos do projeto
- Pasta `node_modules/.bin` → Command Line Interfaces (CLI)
Execução via `npx`
- Inclusão no `.gitignore`



NPM - Versionamento Semântico



"express": "4.16.3"

Versão Maior

- Possível quebra de compatibilidade

Versão Menor

- Compatibilidade retroativa
- Funcionalidade depreciada, mas funcional
- Refatoração interna

Patch

- Correção de bugs

Fonte: Semantic Versioning - <https://semver.org>

NPM - Versionamento Semântico



Utilize o versionamento semântico para estabelecer a dinâmica de atualização dos pacotes do projeto.

```
{  
  . . .  
  "dependencies": {  
    "express": "4.16.3",           // Versão exata 4.16.3  
    "express": "^4.16.3",          // Aceita versões 4.*.*  
    "express": "~4.16.3",          // Aceita versões 4.16.*  
    "express": ">4.16.3",          // Aceita versões superiores a 4.16.3  
    "express": ">=4.16.3",         // ||| superiores ou iguais a 4.16.3  
    "express": "<4.16.3",          // ||| inferiores a 4.16.3  
    "express": "<=4.16.3",         // ||| inferiores ou iguais a 4.16.3  
  }  
}
```

NPM – Listagem de Módulos Instalados



Para verificar os módulos instalados localmente, use o comando **npm list**.

É possível verificar ainda os módulos globais com o comando **npm list -g**.

OBS: A lista traz todas as dependências em forma de árvore. Use o parâmetro **--depth=0** para informar que você deseja ver apenas o primeiro nível da árvore.

```
$ npm list // Lista módulos instalados no projeto
```

```
$ npm list -g --depth=0 // Lista módulos instalados globalmente
```

NPM - Atualização de Módulos



Para verificar os módulos desatualizados, use o comando **npm outdated**.

Para atualizar um módulo, utilize o comando **npm update**.

```
$ npm outdated          // Verifica os pacotes desatualizados
Package      Current  Wanted   Latest  Location
@angular/cli    1.1.3   1.7.4    6.0.3
grunt           1.0.1   1.0.2    1.0.2
http-server     0.10.0  0.10.0   0.11.1

$ npm update grunt      // Atualiza o modulo grunt
```

NPM – Desinstalação de Módulos



Para desinstalar módulos locais do projeto ou globais, utilize o comando **npm uninstall**.

```
$ npm uninstall express
```

---> Desinstalação do modulo express
da pasta do projeto projeto atual

```
$ npm uninstall express -g
```

---> Desinstalação do módulo globalmente

NPM – Configurações do ambiente



Para verificar as configurações do ambiente do NPM tais como: caminho do Node.JS e pasta dos módulos globais utilize o comando **npm config list**

```
$ npm config list
; cli configs
metrics-registry = "https://registry.npmjs.org/"
scope = ""
user-agent = "npm/6.0.1 node/v10.0.0 win32 x64"

; globalconfig C:\Users\UserX\AppData\Roaming\npm\etc\npmrc
msvs_version = "2015"
python = "C:\\\\Users\\\\UserX\\\\.windows-build-tools\\\\python27\\\\python.exe"

; builtin config undefined
prefix = "C:\\\\Users\\\\UserX\\\\AppData\\\\Roaming\\\\npm"

; node bin location = C:\\Desenvolvimento\\nodejs\\node.exe
; cwd = C:\\Desenvolvimento\\xampp\\htdocs\\AulasWeb\\Lab-CSS-Sprites
; HOME = C:\\Users\\UserX
; "npm config ls -l" to show all defaults.
```



119



Pacotes e Dependências

Prof. Rommel Vieira Carneiro



Plataforma Node.js

Módulos em JavaScript e Node.js

Sistema de Módulos

Node.js suporta dois sistemas de módulos para a Linguagem JavaScript

CommonJS

```
// Formato da importação
const fs = require( 'fs' )
const { env } = require('process')
```

```
// Formato da exportação
const msg = 'Hello'
module.exports = { msg }
```

ES6 Modules

```
// Formato da importação
import fs from 'fs'
import process, { env } from 'process'
```

```
// Formato da exportação
export const msg = 'Hello'
```

Sistema de Módulos

FEATURES	REQUIRE	IMPORT
Syntax	const x = require()	import x from "./"
Used In	CommonJS Modules	ES Modules
Asynchronous	✗	✓
Conditional Usage	✓	✗
Selective Load	✗	✓
Node Support	✓	V 13+
TypeScript Support	✓	✓

ES Modules (ESM)

A Linguagem JavaScript incorporou o conceito de módulos a partir do ECMAScript 6 (ES2015).

O Node.js, desde a versão 13.2.0, suporta a implementação dos **ES Modules (ESM)**.

Para habilitar ES Modules no lugar do CommonJS, podem ser adotadas as seguintes estratégias:

- Acrescentar extensão **.mjs** aos arquivos
- Incluir o atributo **type** no **package.json**
- Usar a flag **-input-type=module** em comandos via string

```
$ node app.mjs
```

```
// package.json
{
  "name": "my-app",
  "version": "1.0.0",
  "type": "module",
  // ...
}
```

```
$ node -input-type=module
-e "import sep from path;
  console.log(sep)"
```

ES Modules (ESM)

Suporte para módulos de componentes na Linguagem JavaScript

```
// lib/math.js
export function sum(x, y) {
    return x + y;
}
export var pi = 3.141593;
```

```
// app.js
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));
```

```
// otherApp.js
import { sum, pi } from "lib/math";
alert("2π = " + sum(pi, pi));
```

ES Modules (ESM)

Tipos de exportação: **Named Exports | Default Export**

Permite exportação de vários componentes independentes por módulo

```
// lib.js
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}
```

```
// main.js
import { square, diag } from 'lib';

console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

Fonte: ECMAScript 6 modules: the final syntax - <http://2ality.com/2014/09/es6-modules-final.html>

ES Modules (ESM)

Tipos de exportação: Named Exports | Default Export

Permitem um único componente a ser exportado pelo módulo como default

```
// myFunc.js  
export default function () { ... };
```

```
// MyClass.js  
export default class { ... };
```

```
// main1.js  
import myFunc from 'myFunc';  
myFunc();
```

```
// main2.js  
import MyClass from 'MyClass';  
let inst = new MyClass();
```

Fonte: ECMAScript 6 modules: the final syntax - <http://2ality.com/2014/09/es6-modules-final.html>

ES Modules (ESM)

Formas de Importação de Componentes

```
// Default exports and named exports
import theDefault, { named1, named2 } from 'src/mylib';
import theDefault from 'src/mylib';
import { named1, named2 } from 'src/mylib';

// Renaming: import named1 as myNamed1
import { named1 as myNamed1, named2 } from 'src/mylib';

// Importing the module as an object
// (with one property per named export)
import * as mylib from 'src/mylib';

// Only load the module, don't import anything
import 'src/mylib';
```

Fonte: ECMAScript 6 modules: the final syntax - <http://2ality.com/2014/09/es6-modules-final.html>

Plataforma Node.js

Módulos internos do Node.js (built-in)

Principais Módulos Nativos do Node.js

Módulo	Descrição
events	Criar e manipular de eventos (classe EventEmitter)
fs	Trabalhar com arquivos (criar, ler, alterar e excluir)
http	Criar servidores Web e fazer requisições http
dns	Resolver nomes da Internet
path	Manipular caminhos de arquivos (funções como join, basename)
os	Provê informações sobre o Sistema Operacional
process	Provê informação sobre o processo Node.js em execução
crypto	Operações criptográficas como encrypt, decrypt, hash
net	Criação de servidores e clientes
url	Interpreta strings de URLs
stream	Manipula dados em fluxos

Módulo events – Classe eventEmitter

Método	Descrição
addListener (event, listener)	Inclui um listener na fila para o evento passado como parâmetro
on (event, listener)	Inclui um listener na fila para o evento passado como parâmetro
once (event, listener)	Inclui um listener de um único disparo na fila para o evento passado como parâmetro
removeListener (event, listener)	Remove o listener da fila do evento passado como parâmetro
removeAllListeners (event)	Remove todos os listeners para o evento passado como parâmetro
listeners (event)	Retorna um array com os listeners ativos para o evento passado como parâmetro
emit (event)	Executa os listeners ativos para o evento passado como parâmetro

Módulo events

```
// Importa o módulo eventos
import events from 'events'

// Cria um gerenciador de eventos
const eventMgmt = new events.EventEmitter ();
eventMgmt.on('bomdia', (data) => {
    console.log(`Recebi um bom dia de: ${data}`)
})

// Inclui um subscriber ao evento
eventMgmt.addListener ('bomdia', (data) => {
    console.log(`Aconteceu de ${data}`);
})

// Dispara o evento
eventMgmt.emit ('bomdia', 'Rommel')
```

Módulo process

```
// Apresenta variáveis de ambiente  
console.log ('Variáveis de ambiente ----- \n', process.env)
```

```
// Apresenta o diretório de trabalho  
console.log ('Diretório de trabalho ----- \n', process.cwd())
```

```
// Apresenta o diretório de execução  
console.log ('Diretório de execução ----- \n', __dirname)
```

```
// Apresenta o arquivo (entrypoint) de execução  
console.log ('Arquivo de execução ----- \n', __filename)
```

```
// Apresenta o processo  
console.log ('Processo ----- \n', process)
```

Módulo http

```
// Importa o módulo do HTTP
const http = require('http');

// Cria um servidor e atribui uma callback de processamento da requisição
const server = http.createServer((req, res) => {
  res.statusCode = 200; // Retorno OK
  res.setHeader('Content-Type', 'text/html');
  res.end('Hello World');
});

// Define parâmetros (hostname e porta) e inicia o servidor
const hostname = '127.0.0.1';
const port = 3000;
server.listen(port, hostname, () => {
  console.log(` Servidor rodando http://${hostname}:${port}/`);
});
```

Módulo http

```
// Importa o módulo do HTTP
const http = require('http');

// Cria um servidor e atribui uma callback de processamento da requisição
const server = http.createServer((req, res) => {
  res.statusCode = 200; // Retorno OK
  res.setHeader('Content-Type', 'application/json');
  res.end(`{ "user": "Rommel Carneiro", "empresa": "PUC Minas" }`);
});

// Define parâmetros (hostname e porta) e inicia o servidor
const hostname = '127.0.0.1';
const port = 3000;
server.listen(port, hostname, () => {
  console.log(`Servidor rodando: http://${hostname}:${port}/`);
});
```

Módulo http

```
import http from 'http';

const server = http.createServer((req, res) => {
  if (req.method == "GET") {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/html');
    res.end(`<form method="POST">Nome: <input type="text" name="nome"></form>`);
  }
  else {
    let str = '';
    req.on('data', function (chunk) { str += chunk; });
    req.on('end', function () {
      let nome = str.split('=')[1];
      res.statusCode = 200;
      res.setHeader('Content-Type', 'application/json');
      res.end(`{ "username": "${nome}" }`);
    });
  }
});

server.listen(3000, () => { console.log(`Servidor rodando`); });
```

Módulo dns

```
const { Resolver } = require('dns');
const resolver = new Resolver();

// Define o servidor a ser utilizado
resolver.setServers(['8.8.8.8']);

// Realiza a tradução de um nome de domínio para um endereço IP
resolver.resolve4('pucminas.br', (err, addresses) => {
  if (err)
    console.log(`Erro ao traduzir: ${err.message}`);
  else
    console.log('Endereço IP: ' + addresses[0]);
});
```

Obrigado!

Plataforma Node.js

Framework Express

Framework Express

Framework minimalista e popular na comunidade Node.js
voltado a criação de aplicações Web.

Características

- Gerencia de rotas a partir do método HTTP e da URI da requisição
- Simplifica o provimento de arquivos estáticos
- Simplifica o uso de mecanismos de templates (EJS, HBS, PUG)
- Baseado no [Connect Middleware \(Chain of Responsibility Design Pattern\)](#)

Frameworks para Node.js

- Tipos de Frameworks
 - API Framework | REST Framework
 - HTTP server
 - Web MVC
 - Full Stack
- Opções de Frameworks
 - [Express](#)
 - [Hapi](#)
 - [LoopBack](#)
 - [Meteor](#)
 - [Restify](#)
 - [Sails](#)

	LoopBack	Express	Hapi	Sails	Restify	Meteor
Type	API framework	HTTP server library	HTTP server framework	Web MVC framework	REST HTTP library	Full-stack JavaScript app platform
Top Features	Enterprise connectivity, API Explorer, generators, client SDKs, websocket microservices	HTTP routing, middleware	Modularity, security	Rails familiarity, MVC	Simplicity, REST routing	Universal JavaScript, reactive rendering, websocket microservices
Suitable For	Web apps, APIs	Simple web apps	Web apps, APIs	Web apps, APIs	Simple REST APIs	Web apps

Fonte: <https://loopback.io/lb3/resources>

Framework Express - Exemplo básico

```
// Importa o módulo do Express Framework
const express = require ('express')

// Inicializa um objeto de aplicação Express
const app = express ()

// Cria um manipulador da rota padrão
app.get ('/', function (req, res) {
    res.send ('Hello World')
})

// Inicializa o servidor HTTP na porta 3000
app.listen (3000, function () {
    console.log ('Servidor rodando na porta 3000')
})
```

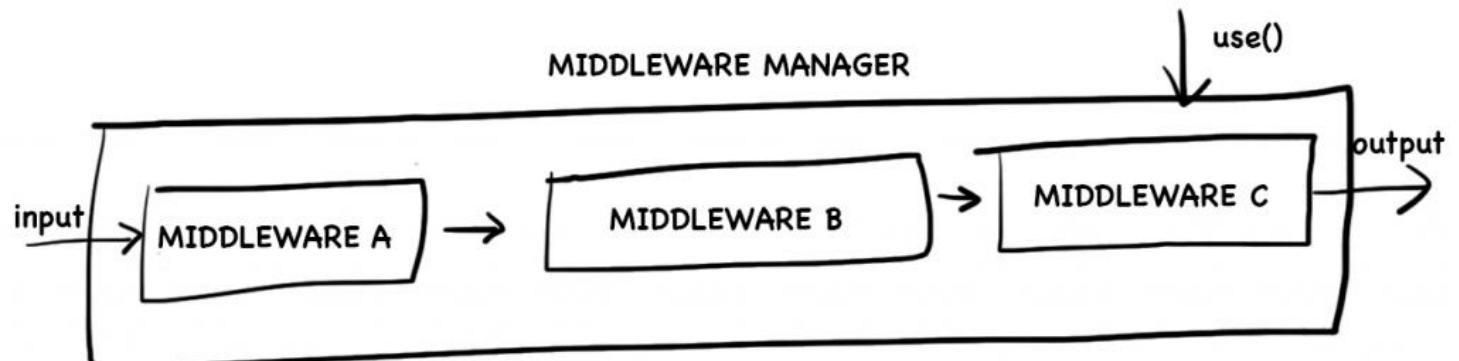
Framework Express – Middleware

O Express utiliza uma estrutura de roteamento baseada no padrão Middleware que implementa um conjunto de funções para o tratamento de requisições Web.

Funções de middleware têm acesso à requisição (req) e resposta (res) HTTP e à próxima função no pipeline do ciclo de requisição e resposta do aplicativo.

Cada função pode:

- Executar alguma lógica
- Mudar a requisição/reposta
- Encerrar o ciclo de requisição/reposta
- Chamar a próxima função do ciclo



Fontes:

- [Usando middlewares do Express \(expressjs.com\)](https://expressjs.com/pt/4x/api.html#middleware)
- [Entendendo o Middleware pattern em Node.js - Waldemar Neto](#)

Framework Express – Middleware

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

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

The diagram illustrates the flow of middleware execution. It shows blue arrows pointing from the code annotations to the corresponding arguments in the middleware function:

- An arrow points from the first annotation ("O método HTTP para o qual a função de middleware é aplicada.") to the path argument in the `app.get('/')` call.
- An arrow points from the second annotation ("Caminho (rota) para o qual a função de middleware é aplicada.") to the path argument in the `app.get('/')` call.
- An arrow points from the third annotation ("A função de middleware.") to the middleware function body (`function(req, res, next) { next(); }`).
- An arrow points from the fourth annotation ("Argumento de retorno de chamada para a função de middleware, chamado de "next" por convenção.") to the `next()` call in the middleware function.
- An arrow points from the fifth annotation ("Argumento de resposta HTTP para a função de middleware, chamado de "res" por convenção.") to the `res` argument in the middleware function.
- An arrow points from the sixth annotation ("Argumento de solicitação HTTP para a função de middleware, chamado de "req" por convenção.") to the `req` argument in the middleware function.

O método HTTP para o qual a função de middleware é aplicada.

Caminho (rota) para o qual a função de middleware é aplicada.

A função de middleware.

Argumento de retorno de chamada para a função de middleware, chamado de "next" por convenção.

Argumento de resposta HTTP para a função de middleware, chamado de "res" por convenção.

Argumento de solicitação HTTP para a função de middleware, chamado de "req" por convenção.

Fontes:

- [Usando middlewares do Express \(expressjs.com\)](#)
- [Entendendo o Middleware pattern em Node.js - Waldemar Neto](#)

Framework Express – Middleware – Rotas

A rota de um middleware pode ser definida por:

1. uma string comum representando um caminho da URL
Ex: `app.use ('/api', callback)`
2. um padrão para um caminho da URL
Ex: `app.use ('/ab*c', callback)`
3. uma expressão regular para o caminho da URL
Ex: `app.use (/\\abc|\\xyz/, callback)`
4. um array com a combinação das opções anteriores
Ex: `app.use(['/api', 'client'], callback)`

Framework Express – Middleware – Tipos

O Express permite o uso de vários tipos de funções de middleware. São eles:

Tipo de Middleware	Exemplos
Nível do aplicativo	app.use app.get app.post ...
Nível de roteador	app.use ([rota], router)
Manipulação de erros	app.use (function (err, req, res, next) {})
Integrados	app.use ([rota], express.static (root, [options]))
Terceiros	app.use(cookieParser())

Fontes:

- [Usando middlewares do Express \(expressjs.com\)](https://expressjs.com/pt/docs/using-middleware.html)

Framework Express – Middleware static

Para servir arquivos estáticos por meio do Express, utilize o middleware static, indicando o diretório onde os arquivos estão localizados:

```
const express = require ('express')
const app = express ()

app.use (express.static ('public'))

app.listen (3000)
```

Caso queira associar um prefixo, defina a rota antes

```
...
app.use ('/static', express.static ('public'))
...
```

Framework Express – Middleware static

Importante para o middleware static:

- A referência para o diretório é relativa ao local em que o node foi executado
- Para evitar problemas de path, utilize `_dirname` para indicar o local da pasta e o pacote `path` para incluir separadores de pastas (/ ou \)

```
const express = require ('express')
const path   = require ('path')
const app = express ()

app.use (express.static (path.join (_dirname, '/public')))

app.listen (3000)
```

Framework Express – Middleware static

É possível controlar diversas opções da resposta HTTP gerada para os arquivos estáticos:

- Cache (ETag, maxAge, lastModified)
- Arquivos ocultos (dotfiles)
- Arquivo default (index)
- Cabeçalhos (setHeaders)

```
const express = require ('express')
const app = express ()

var options = {
  dotfiles: 'ignore',
  etag: false,
  extensions: [ 'htm', 'html' ],
  index: false,
  maxAge: '1d',
  redirect: false,
  setHeaders: function (res, path, stat) {
    res.set('x-timestamp', Date.now())
  }
}

app.use(express.static('public', options))

app.listen (3000)
```

Fonte: [Express Static](#)

Framework Express – Middlewares de Parsers

Para tratar dados do corpo da requisição o Express fornece middlewares que processam o conteúdo com base no seu tipo: JSON, urlencoded, raw ou texto.

Os middlewares de parsers populam o objeto req.body com os dados processados.

São eles:

- express.json ()
- express.urlencoded ()
- express.raw ()
- express.text ()

Fonte: [Express API](#)

```
const express = require ('express')
const app = express ()

app.use (express.json())
app.use (express.urlencoded())

app.use ('/', (req, res, next) => {
    const nome = req.body.nome
    res.send ('Hello ' + nome)
})

app.listen (3000)
```

Framework Express – Middlewares de Terceiros

morgan

Para gerenciar logs de acesso ao aplicativo baseado em Express, utilize o morgan.

Funcionalidades:

- Logs no formato Apache access.log
- Geração do log em arquivo
- Rotação de arquivos por data

```
const express = require ('express')
const morgan = require ('morgan')
const app = express ()

app.use (morgan ("common"))
app.use ('/', (req, res, next) => {
    res.send ('Hello World')
})

app.listen (3000)
```

Fonte: [Morgan GitHub](#)

Framework Express – Middlewares de Terceiros

cors

Para tratar a negociação HTTP para chamadas cross-origin, utilize o cors middleware.

Funcionalidades

- Habilita CORS no servidor todo ou em rotas específicas
- Altamente configurável

Instalação

```
npm install cors
```

```
const express = require ('express')
const cors = require ('cors')
const app = express ()

app.use (cors())
app.options('/produtos/:id', cors())
app.del('/produtos/:id', cors(), (req, res, next) => {
  res.json({msg: 'CORS habilitado para todas as origens!'})
})

app.listen (3000)
```

Fonte: [CORS NPM Package](#)

Framework Express – Middlewares de Terceiros

helmet

Para configurar uma série de cabeçalhos HTTP que limitam a vulnerabilidade de aplicações criadas com o Express, utilize o Helmet middleware.

Funcionalidades

- Oculta informações do servidor Express
- Configura cabeçalhos de proteção

Instalação

```
npm install helmet
```

```
const express = require ('express')
const cors = require ('helmet')
const app = express ()

app.use (helmet())

...
app.listen (3000)
```

Fonte: [Helmet Home Page](#)

Framework Express – Middlewares de Terceiros

apicache

Habilita o cache de respostas.

Funcionalidades

- Cria um ambiente de cache
- Armazena resultados de forma simplificada
- Verifica e retorna cache automaticamente

Instalação

`npm install apicache`

```
const express = require('express');
const app = express();

const apicache = require('apicache');
let cache = apicache.middleware;
app.use(cache('5 minutes'));

const employees = [ // employees data in a database
  { firstName: 'Jane', lastName: 'Smith', age: 20 },
  { firstName: 'John', lastName: 'Smith', age: 30 },
  { firstName: 'Mary', lastName: 'Green', age: 50 },
]

app.get('/employees', (req, res) => {
  res.json(employees);
});

app.listen(3000, () => console.log('server started'));
```

Fonte: [Best practices for REST API design - Stack Overflow](#)

Framework Express – Middlewares de Terceiros

apicache (uso com Redis)

Habilita o cache de respostas.

Funcionalidades

- Cria um ambiente de cache
- Armazena resultados de forma simplificada
- Verifica e retorna cache automaticamente

Instalação

`npm install apicache`

```
import express from 'express'
import apicache from 'apicache'
import redis from 'redis'

let app = express()

// if redisClient option is defined,
// apicache will use redis client
// instead of built-in memory store
let redisCache = apicache.options({
  redisClient: redis.createClient() }).middleware

app.get('/will-be-cached', redisCache('5 minutes'),
  (req, res) => {
    res.json({ success: true })
})
```

Fonte: [apicache - npm](#)

Framework Express – Templates

O Express oferece uma estrutura para aplicação de mecanismos de templates que permitem a geração de páginas HTML dinâmicas.

O Express é extensível e permite o uso de diferentes mecanismos de templates. Os mais utilizados são:

- [handlebars](#)
- [EJS](#)
- [pug](#)
- [Jade](#)



Framework Express – Templates

app.js

```
const express = require("express");
const app = express();

app.set("view engine", "ejs");
app.set("views", "views");

app.get("/", function(req, res){
  res.render("index",
    { message: "Boas vindas!!" });
});

app.listen(3000);
```

/views/index.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Express Templates</title>
</head>
<body>
  Mensagem: <%= message %>
</body>
</html>
```

Framework Express – Router

Os roteadores permitem segmentar partes de uma aplicação Express definindo conjuntos separados de rotas e middlewares.

No exemplo ao lado

- API (rota: /api)
- Cliente (rota: /client)

```
const express = require ('express')
const cors = require ('cors')
const app = express ()

const routerAPI = express.Router()
routerAPI.use (cors())
routerAPI.get ('/produtos/:id', (req, res) => {
    res.send ('API produtos' ) })
app.use ('/api', routerAPI)

const routerClient = express.Router ()
routerClient.use ('/', express.static ('public'))
app.use ('/client', routerClient)

app.listen (3000, function () {
    console.log ('Servidor na porta 3000')
})
```

Framework Express – Objeto de Requisição

Propriedades	Descrição
req.body	Contém pares chave-valor com os dados submetidos no corpo da requisição. IMPORTANTE: É preenchida por meio de um bodyParser.
req.cookies	Contém os cookies enviados na requisição. IMPORTANTE: É preenchida por meio de um cookieParser.
req.host, req.hostname, req.ip, req.protocol	Contém as informações de host, hostname, ip e protocolo associados à requisição Ex: <code>console.log (`\${req.protocol} - \${req.method} \${req.url} - [\${req.ip}]`);</code>
req.params	Contém os parâmetros de path na URL da requisição Ex: <code>http://servidor.com/produtos/:id</code> → req.params.id
req.query	Contém os dados passados pela query string na URL da requisição. Ex: <code>http://servidor.com/produtos?sort=asc</code> → req.query.sort
req.path	Contém a parte do path da URL
req.url	Contem a URL completa (path + query string)

Framework Express – Objeto de Requisição

Métodos	Descrição
req.accepts()	Verifica se um determinado tipo de conteúdo é aceito pelo cliente Ex: req.accepts ('application/json')
req.get()	Obtém um cabeçalho específico da requisição Ex: req.get ('Referrer') req.get ('Content-Type')

Framework Express – Objeto de Requisição

O exemplo mostra a definição de um middleware para geração de logs de requisições a partir das propriedades do objeto de requisição:

```
// Define um middleware para geração de log
app.use ((req, res, next) => {
  console.log (`${req.protocol} - ${req.method} ${req.url} - [${req.ip}]`);
  next()
})
```

Framework Express – Objeto de Requisição

O exemplo mostra o uso do objeto de requisição para obter informações recebidas por meio do path e da query string da URL:

```
const express = require ('express')
const app = express ()

app.use ('/produtos/:id', (req, res, next) => {
    res.send ('Hello ' + req.query.nome + '\nProduto: ' + req.params.id)
})

app.listen (3000)
```

Framework Express – Objeto de Resposta

Método	Descrição
res.download()	Envia um arquivo para ser salvo pelo cliente
res.end()	Finaliza o processo de resposta
res.json()	Envia um objeto JSON como resposta
res.jsonp()	Envia um objeto JSON como resposta no formato JSONP
res.redirect()	Redireciona uma requisição para outro recurso
res.render()	Processa um arquivo de template
res.send()	Envia resposta de diversos tipos
res.sendFile()	Envia um arquivo como stream
res.sendStatus() res.status()	Informa o código de status da resposta. O método status permite o encadeamento de outras funções
res.type()	Informa o tipo de conteúdo da resposta (definido no header Content-Type)
res.format()	Verifica formato solicitado e envia resposta apropriada (header Accept)

Framework Express – Objeto de Resposta

O exemplo mostra como criar um middleware simples para exibir a mensagem Hello World, utilizando dois métodos do objeto de resposta:

- status → que define o status code da resposta
- send → envia um texto de resposta

```
app.get ('/', function (req, res) {
    res.status (200).send ('Hello World')
})
```

Framework Express – Objeto de Resposta

O exemplo mostra como criar um middleware que trata requisições inválidas (404) utilizando dois métodos do objeto de resposta:

- status → que define o status code da resposta
- sendFile → envia um arquivo do sistema de arquivos do servidor

```
// Tratamento de solicitações para arquivos não encontrados
app.use ((req, res) => {
    res.status(404).sendFile(__dirname + '/public/erro404.html')
})
```

OBSERVAÇÃO: este middleware deve ser colocado no final do arquivo para ser utilizado caso nenhum outro middleware seja acionado antes.

Framework Express – Objeto de Resposta (format)

Verifica o cabeçalho **Accept** da requisição para definir a resposta a ser enviada.

```
const express = require('express');
const app = express();
app.get('/', function (req, res) {
  res.format({
    html: function () {
      res.send('<h1>Olá mundo</h1>');
    },
    json: function () {
      res.send({ message: 'Olá mundo' });
    }
  });
});

app.listen(3000, function (err) {
  if (err) console.log(err);
  console.log("Server listening on PORT", PORT);
});
```

Framework Express – Express Generator

O aplicativo de linha de comando (CLI) gera o esqueleto de aplicação padrão (**boilerplate**)

```
// Instale o Express Generator globalmente via NPM
$ npm install -g express-generator

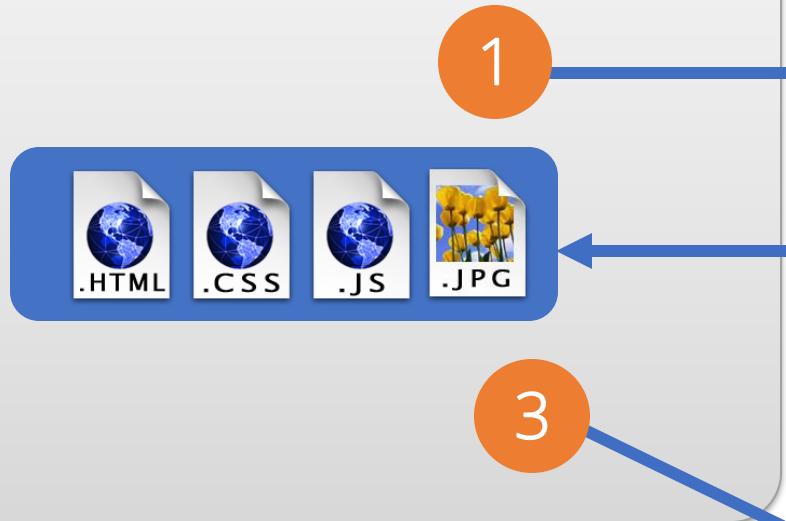
// Execute o express generator
$ express

// Execute o express generator com opções definidas
//   --git  inclui arquivo .gitignore
//   --view define o motor de templates a ser usado
$ express --git --view=hbs
```

```
myapp
  /package.json
  /app.js
  /routes
    /index.js
    /users.js
  /views
    /index.pug
    /layout.pug
    /error.pug
  /public
    /javascripts
    /images
    /stylesheets
      /style.css
  /bin
  /www
```

Hands on sessions ... montar aplicação completa

Ambiente Cliente



Aplicação Server Node.js

CLIENT

2



STATIC FILES



API

Ação	Mapeamento da URL
Incluir um produto	POST / produtos /
Obter a lista de produtos	GET / produtos
Obter um produto específico	GET / produtos /:id
Alterar um produto	PUT / produtos /:id
Excluir um produto	DELETE /produtos/:id

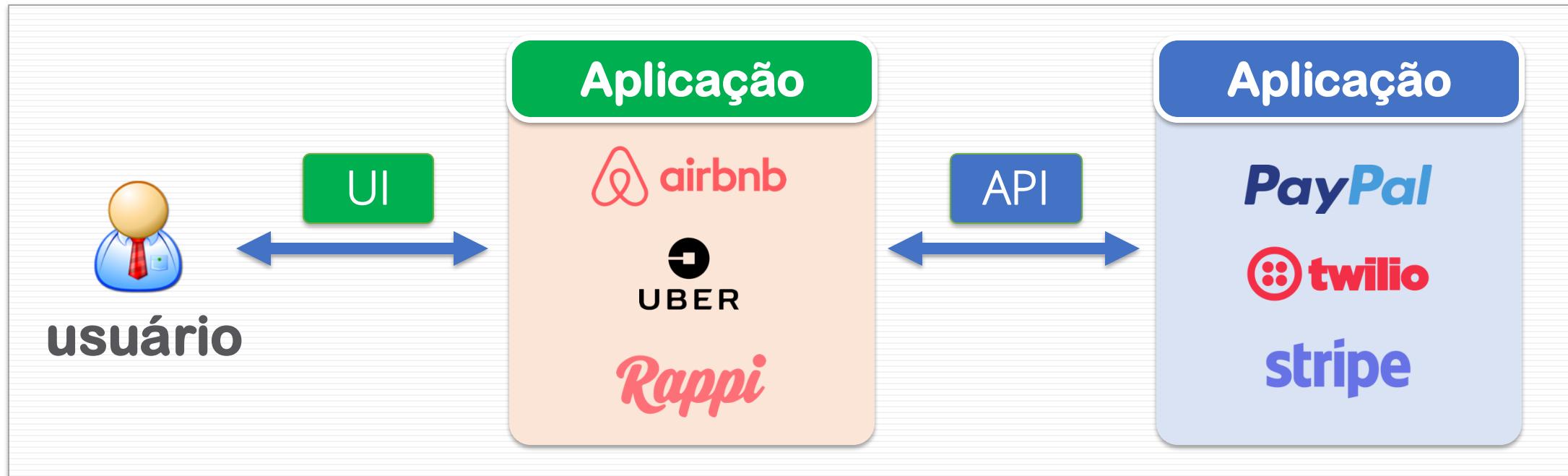
Obrigado!

Plataforma Node.js

Fundamentos de APIs e RESTful APIs

Introdução ao mundo das APIs

Application Programming Interface (API) é uma interface de uma aplicação que é voltada para outras aplicações, proporcionando integração entre sistemas.



Tipos de APIs

Web APIs ou Web Services

- SOAP
- REST e RESTful
- GraphQL
- WebSockets
- WebHooks

APIs Legadas

- Remote Procedure Call (RPC)
- Java Remote Method Invocation (RMI)
- CORBA
- DCOM

Fontes:

- API Is Not Just REST - <https://apievangelist.com/2018/02/03/api-is-not-just-rest/>
- Service-oriented vs Data-centric applications - <https://medium.com/@fidelvi/service-oriented-vs-data-centric-applications-5c3c509e3084>
- 4 estilos de APIs mais usados e comentados do mercado - <https://www.ca.com/pt/blog-latam/estilos-de-apis.html>
- A guide to REST and API Design - <http://docs.huihoo.com/api/A-Guide-to-REST-and-API-Design.pdf>
- Undisturbed REST – A guide to designing the perfect API - <https://www.mulesoft.com/lp/ebook/api/restbook>
- Web API Design – apigee - <https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf>
- Pragmatic REST: APIs without hypermedia and HATEOAS - <http://www.ben-morris.com/pragmatic-rest-apis-without-hypermedia-and-hateoas/>
- API Design: Harnessing Hypermedia Types - <https://apigee.com/about/blog/api-technology/api-design-harnessing-hypermedia-types>



174



Prof. Rommel Vieira Carneiro



RESTful API – Introdução

REST é um estilo arquitetural para construção de serviços Web que significa a Transferência de Estado Representacional (Representational State Transfer).

O termo **RESTful** caracteriza serviços Web que seguem integralmente as recomendações REST, diferentemente daqueles (**RESTlike**) que implementam parcialmente suas recomendações.



RESTful API – Princípios de design

Princípios importantes de design de uma API RESTful:

- Definições do protocolo HTTP
- Estrutura uniforme de Endpoint
- Abordagem stateless (sem estado)
- Abordagem HATEOAS
- Controle do versionamento da API
- Controle adequado do cache
- Documentação clara e adequada da API

Fontes:

- Best Practices for Designing a Pragmatic RESTful API - <https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>
- RESTful Web Services: The basics, Alex Rodriguez - <http://www.ibm.com/developerworks/webservices/library/ws-restful/>
- RESTful API Designing guidelines—The best practices - <https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9>
- RESTful API Design. Best Practices in a Nutshell - <https://blog.philippauer.de/restful-api-design-best-practices/>
- API design - <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design>

RESTful API – Princípios de design

Definições do protocolo HTTP – Métodos HTTP

Utilizar os métodos HTTP de maneira clara em função do seu propósito semântico:

GET	Recuperar um recurso (SELECT)
POST	Criar um recurso no servidor (INSERT)
PUT PATCH	Alterar um estado de um recurso ou atualizá-lo (UPDATE)
DELETE	Remover um recurso (DELETE)
OPTIONS	Lista as operações de um recurso

IMPORTANTE: Métodos GET e parâmetros de query não devem alterar estado de recursos.

RESTful API - Princípios de design

Definições do protocolo HTTP - Métodos HTTP

Antes - Criação com GET

```
GET /adduser?name=Robert HTTP/1.1
```

Depois - Criação com POST

```
POST /users HTTP/1.1
```

Host: myserver

Content-Type: application/xml

```
<?xml version="1.0"?>
```

```
<user>
```

```
    <name>Robert</name>
```

```
</user>
```

Exemplo 1

- 1) Evite utilizar verbos (adduser) na URL, os métodos devem ser suficientes para descrever a operação
- 2) Utilize o método correto para a operação (Criar → POST)

RESTful API – Princípios de design

Definições do protocolo HTTP – Métodos HTTP

Não RESTful		
Verbo	HREF	Ação
POST	/bookmarks/create	Criar (Create)
GET	/bookmarks/show/1	Visualizar (Read)
POST	/bookmarks/update/1	Atualizar (Update)
POST/GET	/bookmarks/delete/1	Apagar (Delete)
RESTful		
Verbo	URI	Ação
POST	/bookmarks	Criar (Create)
GET	/bookmarks/1	Visualizar (Read)
PUT	/bookmarks/1	Atualizar (Update)
DELETE	/bookmarks/1	Apagar (Delete)

RESTful API – Princípios de design

Definições do protocolo HTTP – Códigos de status da resposta HTTP

Code	Propósito	Descrição	Exemplo
1xx	Informacional	Requisição recebida,	Processo em continuidade
		200 – Sucesso na requisição	Requisição de informações (GET) com sucesso
2xx	Sucesso	201 – Recurso Criado	Inclusão de recurso (POST) com sucesso
		202 – Requisição aceita para processamento	Processo assíncrono sem retorno imediato
		204 – Requisição processada com sucesso	Exclusão de recurso (DELETE) com sucesso
3xx	Redirecionamento	301 – Movido permanentemente	Site transferido de servidor
		302 – Movido temporariamente	Recurso temporário no lugar
		400 – Requisição incorreta	Problemas de sintaxe, rotas inexistentes
		401 – Autenticação Requerida	Primeira requisição sem dados de autenticação
4xx	Erro no cliente	403 – Acesso negado	Recurso privado não acessível pelo requisitante
		404 – Recurso não encontrado	Recurso solicitado não existe
		405 – Método não permitido	PUT ou DELETE em endpoints de GET ou POST
5xx	Erro no servidor	O servidor falhou em completar um pedido aparentemente válido	

RESTful API – Princípios de design

Definições do protocolo HTTP – Uso de MIME Types no Content Type

MIME-Type	Content-Type
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml

```
GET /cliente/2 HTTP/1.1  
Host: http://servidor  
Accept: application/json
```

Requisição

```
HTTP/1.1 200 OK  
Content-Type: application/json; charset=utf-8  
Content-Length: 109  
{  
    id: 2  
    name: 'Rommel Vieira Carneiro',  
    email: 'rommel@email.com',  
    alcohol: '5.21'  
}
```

Resposta

RESTful API – Princípios de design

Estrutura uniforme de Endpoint

- Uma URI intuitiva direta, auto-documentada, e comprehensível
- Ter uma estrutura hierárquica semelhantes a diretórios

Exemplo

```
http://myservice.org/discussion/{year}/{day}/{month}/{topic}  
http://myservice.org/discussion/2008/12/10/{topic}
```

Orientações adicionais

- Ocultar a extensão da tecnologia empregada (.asp, .jsp, .php, etc)
- Manter tudo em caixa baixa (minúsculo)
- Evitar espaços nos caminhos da URI
- Evite parâmetros de QueryString, o máximo possível

RESTful API – Princípios de design

Estrutura uniforme de Endpoint – CRUD de Recursos

Ação	Operação	Mapeamento da URL
Incluir um curso	C REATE	POST /cursos/
Obter lista de cursos	R ETRIEVE	GET /cursos
Obter um curso específico	R ETRIEVE	GET /cursos/:id
Pesquisar um curso	R ETRIEVE	GET /cursos?search=param
Alterar um curso	U UPDATE	PUT /cursos/:id PATCH /cursos/:id
Excluir um curso	D ELETE	DELETE /cursos/:id

RESTful API – Princípios de design

Estrutura uniforme de Endpoint – Relacionamentos

Ação	Mapeamento da URL
Listar alunos do curso	GET /cursos/:id/alunos ou GET /alunos/?curso_id=:id
Obter um aluno do curso	GET /cursos/:id/alunos/:id
Incluir aluno no curso	POST /cursos/:id/alunos
Remover aluno do curso	DELETE /cursos/:id/alunos/:id
Listar cursos do aluno	GET /alunos/:id/cursos ou GET /cursos/?aluno_id=:id
Obter um curso do aluno	GET /alunos/:id/cursos/:id

RESTful API - Princípios de design

Estrutura uniforme de Endpoint

Exemplo: The Movie DB - Endpoint: <http://api.themoviedb.org/3>

Ação	Mapeamento da URL
Busca por empresas	GET /search/company?query=xxx&page=x
Busca por filmes	GET /search/movie?query=xxx&page=x GET /search/movie?year=XXXX&language=xx
Dados de filme específico	GET /movie/{movie_id}
Artistas e equipe técnica do filme	GET /movie/{movie_id}/credits
Dados de uma empresa específica	GET /company/{company_id}
Dados de uma pessoa específica	GET /person/{person_id}

RESTful API – Princípios de design

Abordagem stateless (sem estado)

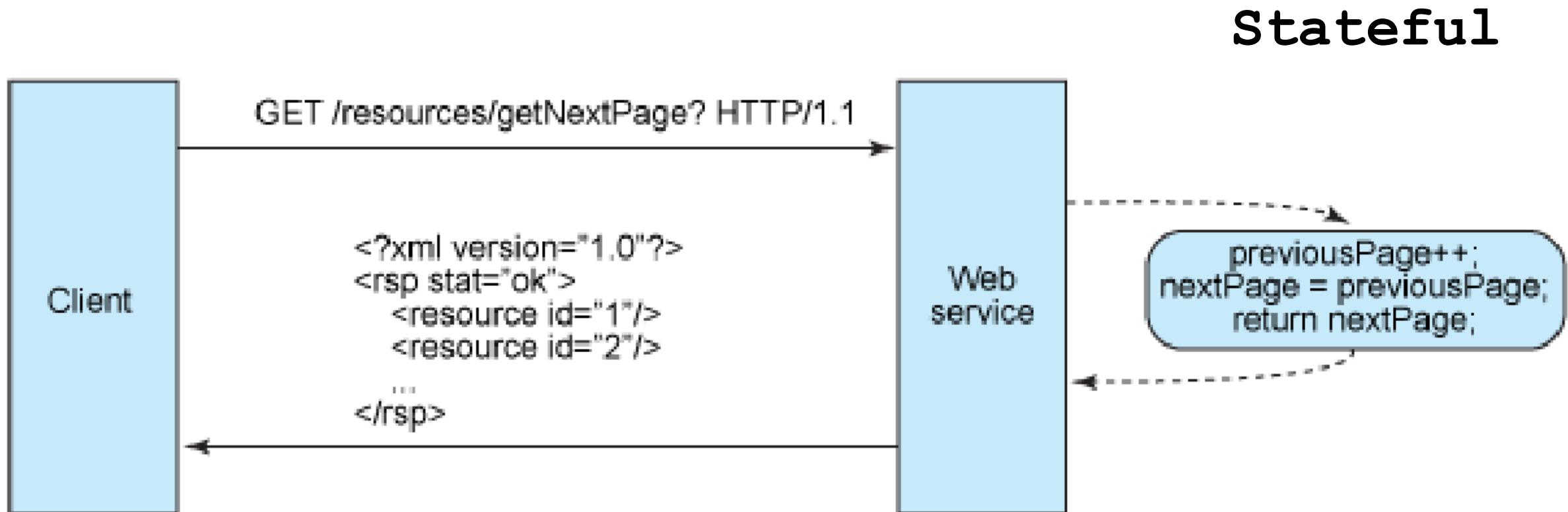
- O servidor não mantém estado sobre a sessão do usuário/aplicação
- Toda requisição deve conter todas as informações requeridas (como parte da URI, na query string, no corpo ou no cabeçalho)

Benefícios

- Simplifica o servidor
- Maior escalabilidade uma vez que o servidor não mantém informações sobre sessão
- Servidores de平衡amento de carga não precisam se preocupar com dados de sessão
- Maior confiabilidade (recuperação de falhas)

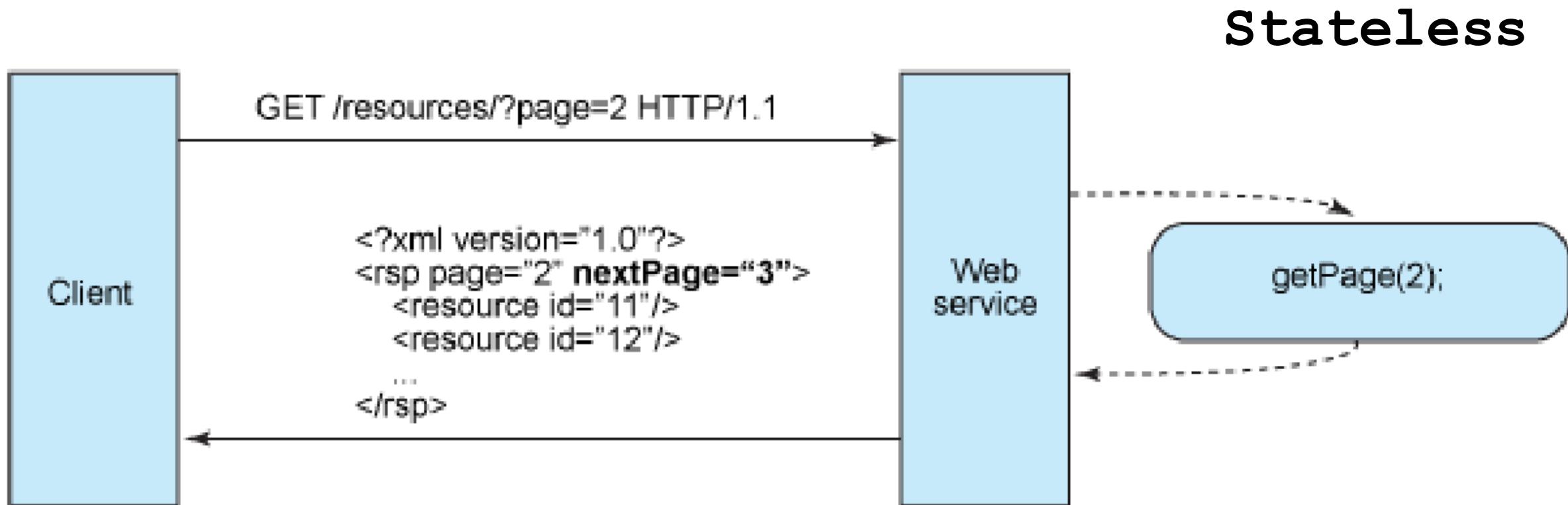
RESTful API – Princípios de design

Abordagem stateless (sem estado)



RESTful API – Princípios de design

Abordagem stateless (sem estado)



RESTful API - Princípios de design

Abordagem HATEOAS - Hipertexto como mecanismo de estado da aplicação

```
GET /account/12345 HTTP/1.1
```

Resposta p/ saldo de 100,00

```
HTTP/1.1 200 OK
```

```
<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
  <link rel="withdraw" href="/account/12345/withdraw" />
  <link rel="transfer" href="/account/12345/transfer" />
  <link rel="close" href="/account/12345/close" />
</account>
```

RESTful API - Princípios de design

Abordagem HATEOAS - Hipertexto como mecanismo de estado da aplicação

```
GET /account/12345 HTTP/1.1
```

```
-----
```

```
HTTP/1.1 200 OK
```

```
<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">-25.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
</account>
```

Resposta p/ saldo de -25,00

RESTful API – Princípios de design

Versionamento da API

- Incorporar a versão do serviço na URI
`http://myservice.org/v1/discussion/{year}/{day}/{month}/{topic}`
- Incluir informações da versão na representação de estado do recurso

```
<beer version="1.0">
  <name>Asahi Draft Beer</name>
  <brewer>
    <name>Asahi</name>
    <country>Japan</country>
  </brewer>
  <calories>41</calories>
  <alcohol>5.21</alcohol>
</beer>
```

RESTful API – Princípios de design

Controle adequado do cache – Cabeçalhos HTTP

Cabeçalho	Aplicação
Date	Data e hora de geração da representação de estado.
Last Modified	Data e hora de última alteração da representação do estado.
Cache-Control	O cabeçalho utilizado pelo HTTP 1.1 para controle de cache.
Expires	Data e hora de expiração a representação do estado. Para compatibilizar com clientes HTTP 1.0.
Age	Tempo passado em segundos desde que o resultado foi extraído do servidor. Pode ser inserido por um componente intermediário.

RESTful API – Princípios de design

Controle adequado do cache – Valores para o Cache-Control

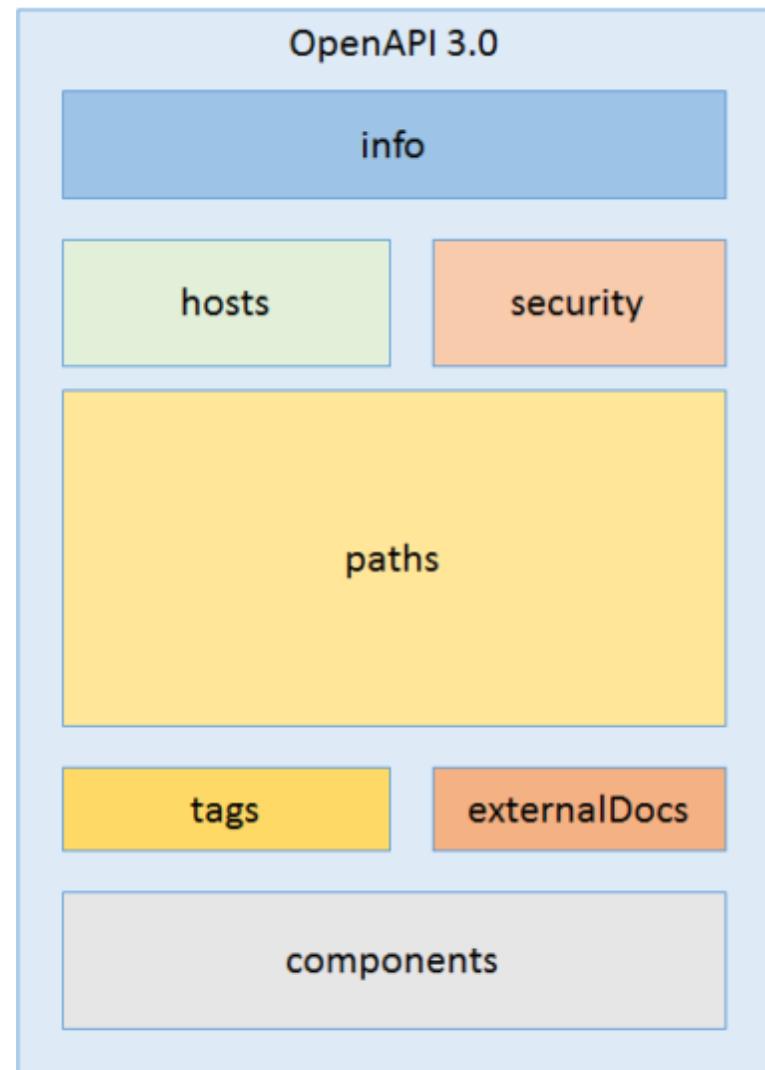
Directive	Application
Public	The default. Indicates any component can cache this representation.
Private	Intermediary components cannot cache this representation, only client or server can do so.
no-cache/no-store	Caching turned off.
max-age	Duration in seconds after the date-time marked in the Date header for which this representation is valid.
s-maxage	Similar to max-age but only meant for the intermediary caching.
must-revalidate	Indicates that the representation must be revalidated by the server ifmax-age has passed.
proxy-validate	Similar to max-validate but only meant for the intermediary caching.

RESTful API – Princípios de design

Documentação clara e adequada da API

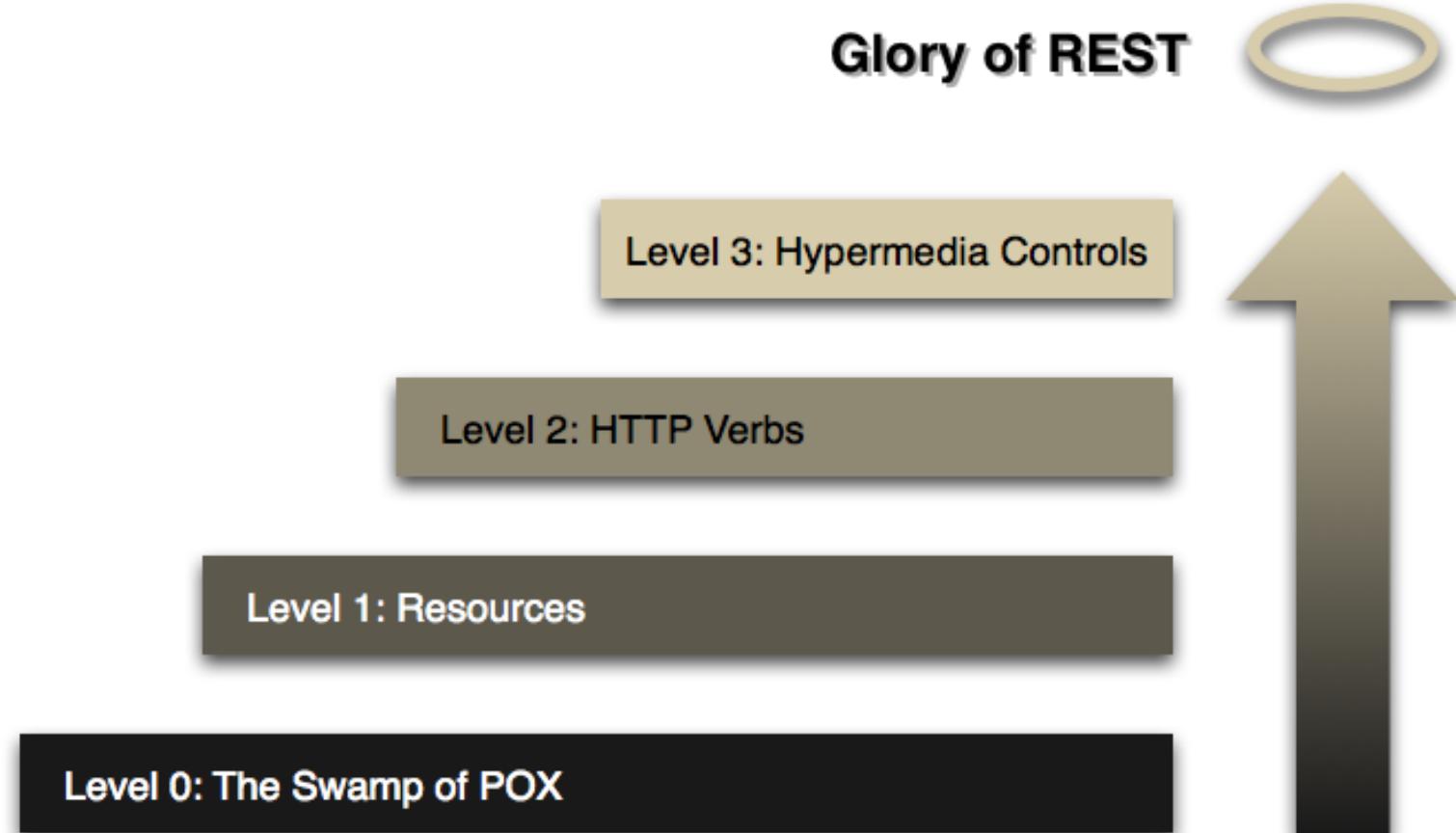
OpenAPI Specification

- Padrão para definição de APIs RESTful
- Independente de linguagem de programação
- Permite a geração automática de código



RESTful API – Princípios de design

Richardson Maturity Model



Fonte: [Richardson Maturity Model](#)

Plataforma Node.js

Acesso a Bancos de Dados

Acesso a Bancos de Dados com Node.js

Alternativas para acesso a Bancos de Dados

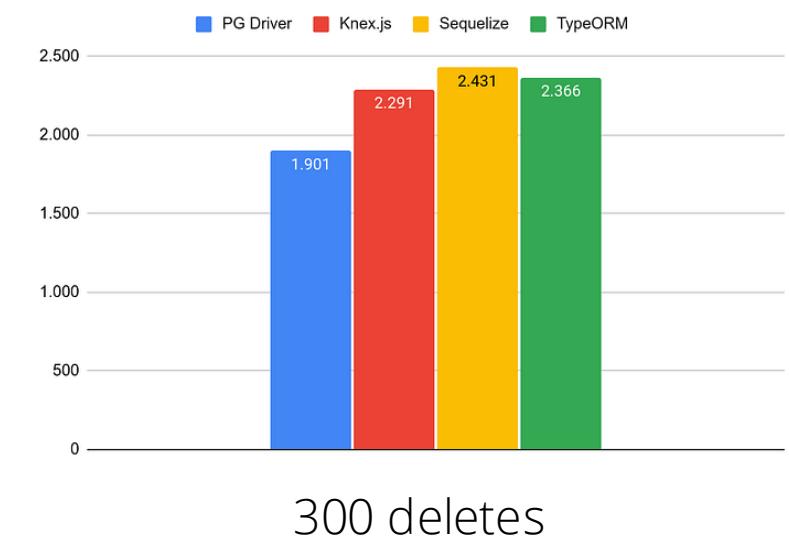
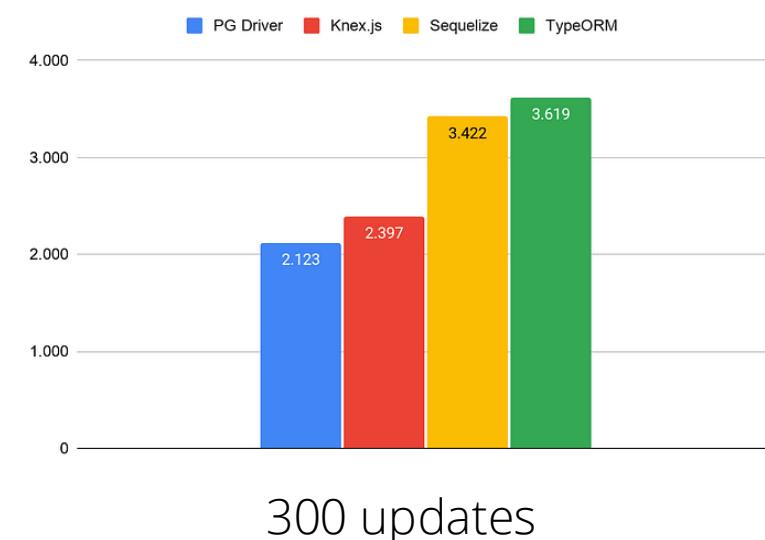
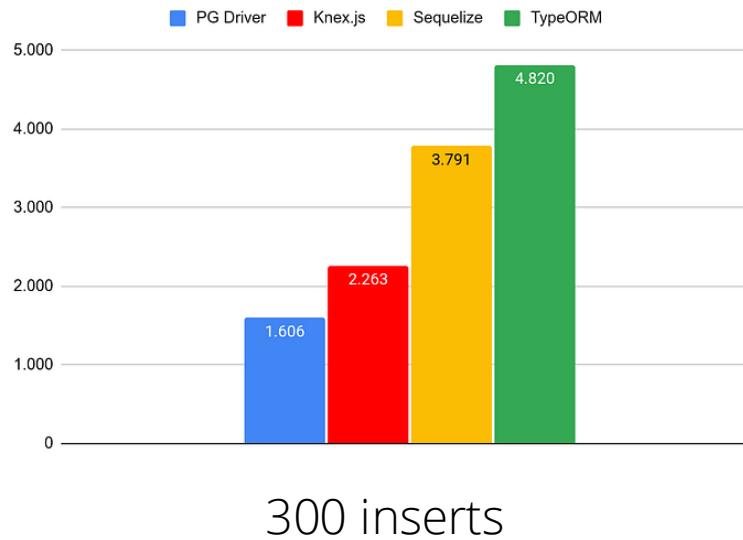
- Drivers Nativos (PG, MySql, etc)
- Object-Relational Mapping (ORM) frameworks
- Object Data Modeling (ODM) frameworks
- SQL Query Builder

Frameworks & Bibliotecas para Node.js



Comparativo de Alternativas para DBs Relacionais

Driver Nativo (pg) x Knex.js x Sequelize x TypeORM



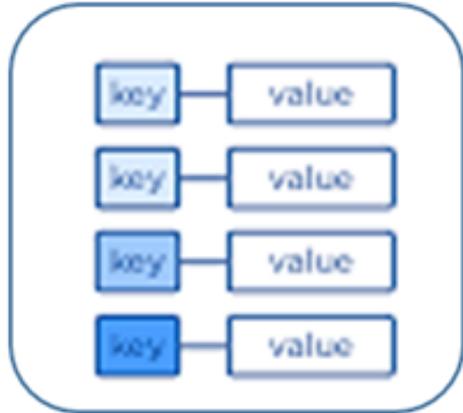
Fonte: [PG Driver vs Knex.js vs Sequelize vs TypeORM](#) (Welington Fidelis)

Tipos de NoSQL Databases



DOCUMENT STORE

Dados e metadados armazenados hierarquicamente em documentos baseados em JSON



KEY VALUE STORE

Dados representados por uma coleção de pares chave-valor (o mais simples)



WIDE-COLUMN STORE

Dados relacionados são armazenados com um conjunto de pares de chave-valor aninhados em uma única coluna.



GRAPH STORE

Dados são armazenados em uma estrutura de grafo (nodos, arestas e propriedades)

Fonte: [Relational vs. NoSQL data](#) (Microsoft Learn)

Bancos Relacionais vs Bancos NoSQL

Bancos de Dados Relacionais	X	Bancos de Dados NoSQL
Modelo estruturado, organizado em tabelas	Estrutura de Dados	Flexíveis na estrutura de dados com modelos de dados diferentes
Escalam verticalmente, adicionando mais poder de processamento ao servidor	Escalabilidade	Escalam horizontalmente, usando mais servidores para lidar com o aumento da carga
Propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade), garantem confiabilidade em transações	Transações ACID	Nem todos suportam propriedades ACID. Alguns oferecem consistência eventual
SQL (Structured Query Language) para consultas	Consultas	Variedade de linguagens de consulta, dependendo do sistema específico
Representar relações entre dados.	Relacionamentos	Relacionamentos são difíceis de representar em alguns bancos
A consistência dos dados é crucial (sistemas bancários, reservas)	Uso	Escalabilidade e a flexibilidade são mais importantes do que a consistência estrita (redes sociais ou big data)

Bancos Relacionais vs Bancos NoSQL

Casos de uso típicos

Bancos de Dados Relacionais	Bancos de Dados NoSQL
<ul style="list-style-type: none">• Content management systems (CMS)• Customer relationship management (CRM)• Enterprise resource planning (ERP)• Data warehouses• Análises Financeiras• Identity management• Dados Geoespaciais• Catálogos de Product	<ul style="list-style-type: none">• Gestão de Logística e Ativos• Gerenciamento de inventário e catálogo• Gerenciamento digital e de mídia• Personalização, recomendações e experiência do cliente• Internet das coisas (IoT) e dados de sensores• Serviços financeiros e pagamentos• Detecção de fraude e autenticação de identidade• Sistemas de mensagens

Fonte: [NoSQL Vs Relational Databases](#) (Boltic)

Obrigado!

Knex Query Builder



- SQL Query Builder para Node.js
- Compatível com diversos bancos de dados
 - PostgreSQL, SQL Server, MySQL, SQLite3, Oracle
- Abordagem via Promises e Callbacks
- Oferece os seguintes recursos
 - Controle de transações
 - Pool de Conexões
 - Streaming Queries
 - Pacote de testes
- Referências
 - [Knexjs Org](#)
 - [Knex CheatSheet](#)

Knex - Instalação e inicialização



Utilize o npm para instalação dos pacotes para o Knex e o npx para executar a inicialização do Knex que cria o arquivo **knexfile.js**

```
# Instalação do pacote do Knex no projeto
```

```
$ npm install knex --save
```

```
# Instalação da biblioteca de Banco de Dados
```

```
$ npm install pg
```

```
$ npm install mysql
```

```
$ npm install sqlite3
```

```
$ npm install oracledb
```

```
# Inicializa knexfile.js (arquivo de configuração)
```

```
$ npx knex init
```

Knex – Instalação e inicialização



Execute o comando de inicialização do Knex: **npx knex init**

```
module.exports = {
  development: {
    client: 'sqlite3',
    connection: { filename: './db.sqlite3' }
  },
  production: {
    client: 'pg',
    connection: {
      database: 'db',
      user: 'user',
      password: 'pass'
    },
    pool: { min: 2, max: 10 }
}
};
```

knexfile.js

Knex – Conexão com o BD



Configuração de conexão via objeto de parâmetros

```
const knex = require('knex')({  
  client: 'mysql',  
  connection: {  
    host : 'servidor.com',  
    port : 3306,  
    user : 'user',  
    password : 'senha',  
    database : 'database'  
  },  
  pool: { min: 0, max: 7 }  
});
```

Knex – Conexão com o BD



Configuração de conexão via string de conexão (literal e variável de ambiente)

```
const knex = require ('knex') ({  
  client: 'pg',  
  connection: 'postgres://user:senha@servidor:5432/database'  
});
```

```
const knex = require ('knex') ({  
  client: 'pg',  
  connection: process.env.DATABASE_URL  
});
```

Knex – Comandos básicos – SELECT



Selecionar dados a partir de uma tabela

...

knex

```
.select('*')
.from('produto')
.where( { marca: 'Helmanns' } )
.limit(10)
```

...

Knex – Comandos básicos – SELECT



Selecionar dados com filtro (where) a partir de uma tabela e devolver resultado em uma API com middleware Express.

```
...  
  
app.get (endpoint + 'produtos', function (req, res) {  
  knex  
    .select ('*')  
    .from ('produto')  
    .where ( { marca: 'Helmanns' } )  
    .then ( produtos => res.status(200).json (produtos) )  
})  
  
...
```

Knex – Comandos básicos – INSERT



Inserção de múltiplos registros com retorno dos IDs gerados automaticamente

```
...  
  
knex('users')  
.insert([  
  { name: 'Starsky' },  
  { name: 'Hutch' }  
], ['id'])  
  
...
```

Knex – Comandos básicos – UPDATE



Alteração de registro

```
...  
knex('users')  
.where({ id: 2 })  
.update({ name: 'Homer' })  
...
```

Knex – Comandos básicos – DELETE



Exclusão de registros

```
...  
knex('users')  
.where({ id: 2 })  
.del()  
...
```

Obrigado!

Knex – Migrations



Processo de evolução do banco de dados

- [Evolutionary Database Design \(Martin Fowler\)](#)
- [Migração de banco de dados \(Google\)](#)
- [What Are Database Migrations? | Database Migrations in Node](#)
- [Knex Migrations](#)

Knex – Migrations



Cria uma migração

npx knex migrate:make migration_name

npx knex migrate:make migration_name --env production

Executa scripts de migração

npx knex migrate:latest [--env production] # Executa até mais novo

npx knex migrate:up # Executa próximo script

Desfaz scripts de migração

npx knex migrate:rollback [--env production] # Desfaz tudo

npx knex migrate:down # Desfaz último script

Lista migrações já executadas

npx knex migrate:list

Knex - Migrations - Configuração



Configurações para especificar o diretório das migrações e a tabela de controle de migrações no banco de dados.

```
module.exports = {
  development: {
    client: 'sqlite3',
    connection: { filename: './db.sqlite3' },
    migrations: {
      directory: './db/migrations', // Diretório para migrações
      tableName: 'knex_migrations' // Tabela de controle de migrações
    },
    seeds: {
      directory: './db/seeds', // Diretório para arquivos de carga
    }
  },
};
```

knexfile.js

Knex - Migrations



Exemplo de arquivo de migração com rotina up e down

```
exports.up = function(knex) {
  return knex.schema.createTable("produtos", tbl => {
    tbl.increments ('id');
    tbl.text ("descricao", 255).unique ().notNullable();
    tbl.text ("marca", 128).notNullable();
    tbl.decimal ("valor").notNullable();
  });
};

exports.down = function(knex) {
  return knex.schema.dropTableIfExists ("produtos")
};
```

Knex - Migrations



```
exports.up = function(knex) {
  return knex.schema.createTable("usuarios", tbl => {
    tbl.increments ('id');
    tbl.text ("nome", 255).unique().notNullable();
    tbl.text ("login", 100).unique().notNullable();
    tbl.text ("email", 100).notNullable();
    tbl.text ("senha", 100).notNullable();
    tbl.text ("roles", 200).notNullable();
  });
};

exports.down = function(knex) {
  return knex.schema.dropTableIfExists ("usuarios")
};
```

Knex - Migrations (Seed CLI)



Seed é uma API do Knex para realizar carga no banco de dados.

```
# Gera arquivo para carga de dados → Cria pasta seeds
```

```
npx knex seed:make seed_name
```

```
# Executa carga de dados (todos ou arquivo específico)
```

```
npx knex seed:run
```

```
npx knex seed:run --specific=filename.js
```

Knex - Migrations (Seed CLI)



Exemplo de arquivo de carga (seed)

```
exports.seed = async function(knex) {
  // Deletes ALL existing entries
  await knex('produtos').del()
  await knex('produtos').insert([
    {id: 1, descricao: 'Cream Cracker', marca: 'Aymmoré', valor: 3.5},
    {id: 2, descricao: 'Cerveja', marca: 'Spaten', valor: 10.99},
    {id: 3, descricao: 'Filé Mignon', marca: 'Friboi', valor: 78.99},
    {id: 4, descricao: 'Refrigerante', marca: 'Coca-Cola', valor: 15.50}
  ]);
};
```

Obrigado!

Plataforma Node.js

Segurança em Aplicações Web

Plataforma Node.js

Fundamentos de Segurança

Autenticação vs Autorização



Autenticação

Verifica se o usuário é quem diz ser

Autorização

Verifica as permissões de acesso do usuário

Fontes:

- [Authentication vs. Authorization \(okta\)](#)

Autenticação Multi-Fator (MFA)

A Autenticação multi-fator é um mecanismo de segurança que utiliza várias estratégias para identificar um usuário em sistemas eletrônicos.

Fatores de Identificação

Conhecimento

Algo que você sabe

Exemplos

- Senhas
- PIN
- Padrões

Posse

Algo que você possui

Exemplos

- Smartcard
- Telefone celular
- Tokens

Herança

Algo que você é

Exemplos

- Íris do olho
- Digitais
- Voz
- Face

Localização

Onde você está

Exemplos

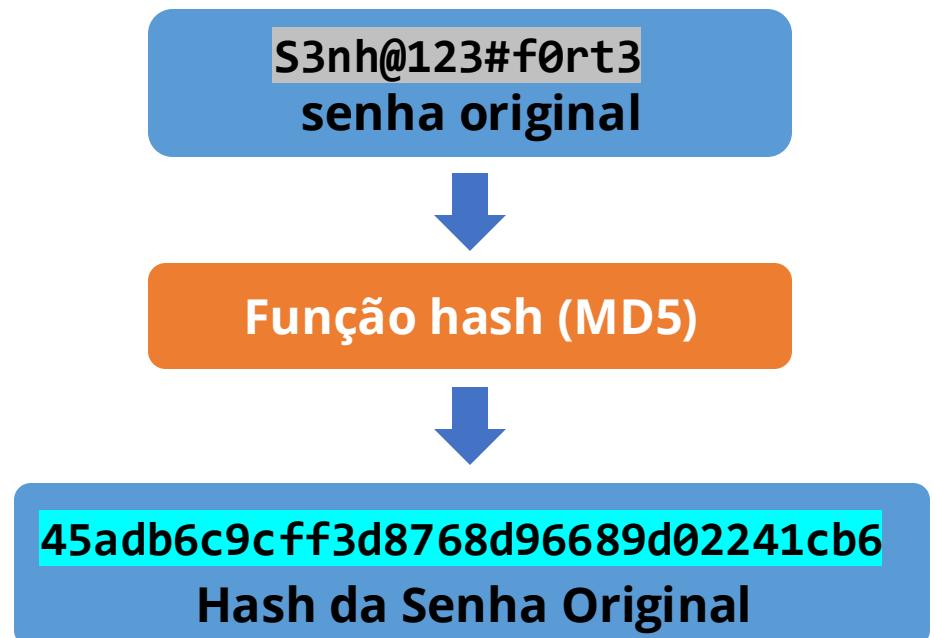
- Coordenadas GPS
- Endereço IP

Message Digest – Função Hash

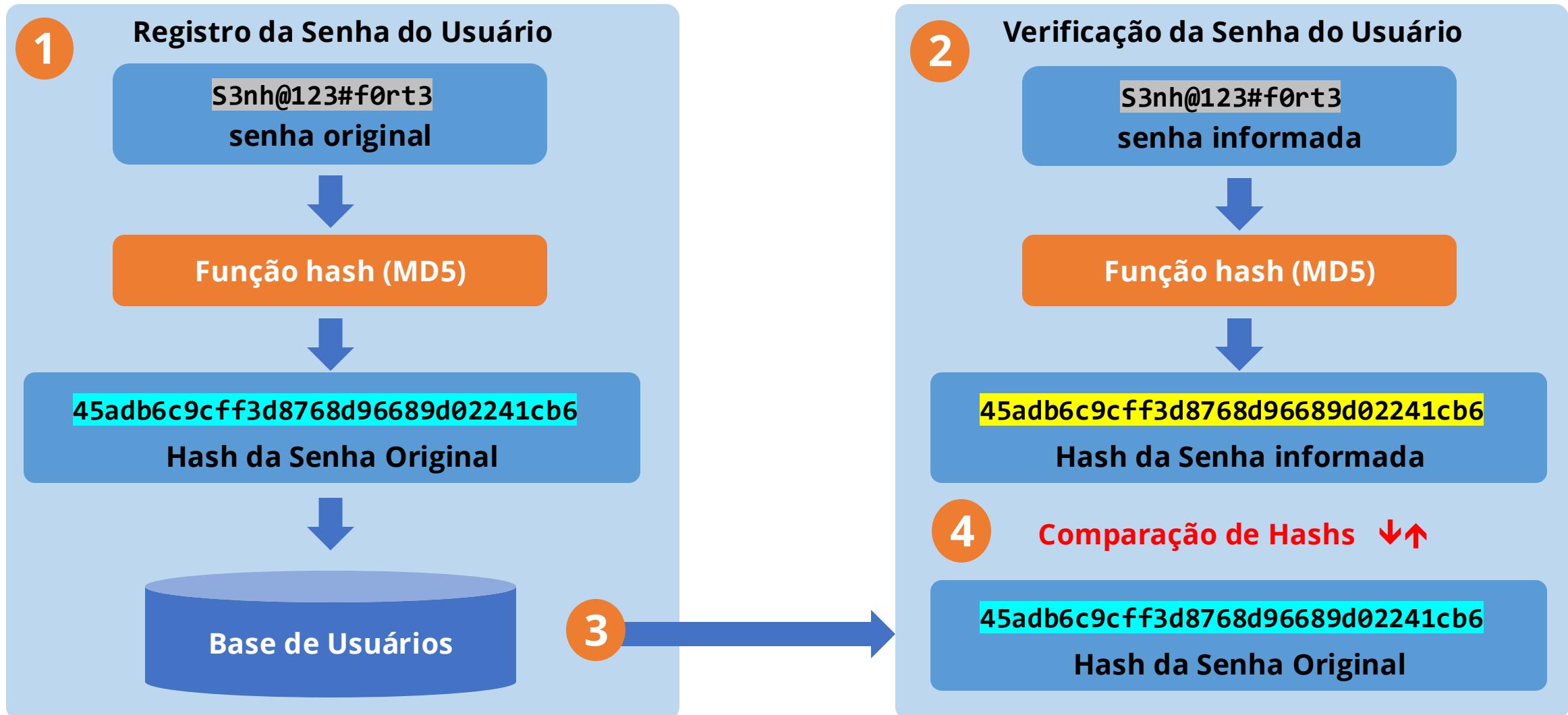
Função hash: algoritmo que produz uma sequência diferentes para cada entrada e de tamanho fixo não reversíveis (não restauram a mensagem original).

Algoritmos

- **Message Digest Algorithm (MD5)**
 - Resumo de mensagem de 128bits
 - Documentação RFC-1321
- **SHA-1**
 - Resumo de mensagem de 160bits
 - Padrão no EUA



Senhas seguras com Hash



Codificação Base64

Método utilizado para converter dados binários em texto e vice-versa.



```
iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAf8/9hAAAABGdBTUEAAK/INwWK6QAAABI0RVh0U29mdHdcmU
AQWRvYmUgSW1hZ2VSZWFeXHJZTwAAAndSURBVHjaYvz//z8DJQAggBh9fecyfPr8i4GPI5vh+/cfDL9//WVgYeV
k+P37H8O/v/8zvv34K8TMzJKiqa5YdOfO0e/Hjjc7MDD8fMjAwAw2ACCAWHCZ/PPnLx4mJtYKJSXJJG8fTUI
TY2WGbb
sVGS5fXVH1+dOVdJg6gADCMODvv39Mf3/8rINSls60d9Au
s3OQZZASY2M4d+UPg42zPIPZ1tCQvXuu1DEw/HvJwMDEAB
AYAP+/fvP8Of3X67//xlrIBTI0mzstYStrIEajVkJ+HkZGC5e+M7w5z8zg7QkA4Ovf4jQoYOLin//vlsGtI4BII
BABohxcbHnSUqJpFvZaoIYWMozSlozM7CxMTBwsTMwvHv7h+H1278M8kocDMx/GBicnbQYDAwDok+f6m0B6vOEEEAs
goI8J+3t1RRs7LQY5GTZGNhYgQ5jQnjp7q3vDKLCQJfwMDlwACNMUQHkijDJs2eWZ//796wdIICYjYx8W/TMN
NmevXgHVPye4c2r7wzfvv8FxgAjw7Onvxg+fvzLoKLKzcDCwsjADAx4Lk4GBgEBccb9B+4ovH1zdhFAADH//cuu5OHjb6Sg
Ls7AzsHNAAwHhk8ffjM8evid4d27nwzqmnwMfHwsYFexAD3MDKRIZJgYXr8XEzx0YMdLgABifvfu7W1RQc14Oztddh4O
VgYxYU4GWVluBkVFHgZFJW4GTk5IRDEDKS4uiCHPnjEwnDrzkvH0qV2MAEEEdBYvg65u3Kqjxz79f/3m///nz/7/fwOkP378//r1//v3///v3/9g8BYoPnvWnf9mZg1/2NnVzzMyMvsDBBDQVDEG
Dg4I47rabd8/f/7//2H//8/APGXL////kD0QgycNbMG/9tbet+s7GpngM6KB6IOUAuAwggBi5uaWDSFWIwMkzf
c+Xyj/+fyM0vn717//c2Tf/W1tV/2ZmVjwFVB+LnvgAAoiBg1OCgY1dmIGH18i+r+f4b5DGly9BNl7+b21d8YuFR
ekOUFOUQgdIPyOcBxBALhx80sD4/c/w7z/bwc1b1p9mZWO13LBh7e8DB5ed/fv3YR9QzXog/oMrzwAEEMnpxDYRGBQgajQ6+8/jsl/v58fAgoux64R5IK/DOBUBQQAAQYAP5FRv1nW25oAAAAASUVORK5CYII=
```

Base64 é muito utilizado na Internet pois vários protocolos de aplicação aceitam apenas dados em texto (email, http). É usado em conjunto com o padrão MIME.

Codificação Base64

Senhas são codificadas via Base64 para garantir que os caracteres a serem enviados na comunicação são strings ASCII válidas



IMPORTANTE

Embora seja uma codificação, Base64 não é uma forma de criptografia e deve ser considerado como dados sendo passados de forma aberta.

Internet X.509 Public Key Infrastructure (PKI)

Conceitos

- **ICP - Infraestrutura de Chaves Públicas**

Estrutura que abrange um conjunto de entidades AC Raiz, ACs, ARs, certificados para prover requisitos de segurança em comunicação de sistemas.

- **AC Raiz - Autoridade Certificadora Raiz**

Estabelece a cadeia hierárquica de certificados. Estabelece a política de certificados, controla certificados das ACs e mantém a lista de certificados revogados

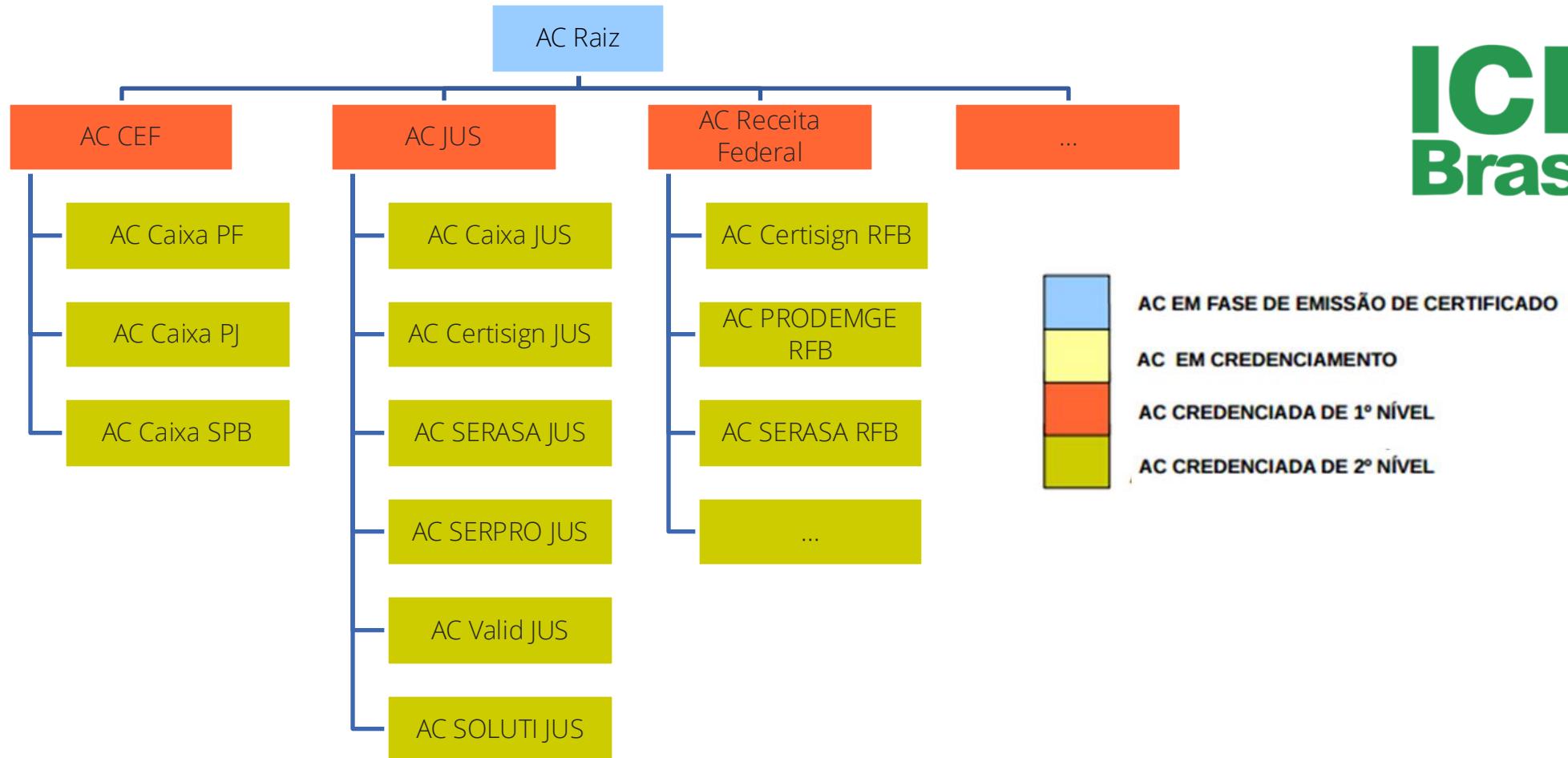
- **AC - Autoridade Certificadora**

Responsável por emitir, distribuir, renovar e gerenciar certificados digitais.

- **Certificado Digital**

Arquivo eletrônico contendo a identidade digital de uma entidade reconhecida pela ICP. Contem: nome da entidade, período de validade, chave pública, nome e assinatura da entidade que assinou o certificado, número de série.

Internet X.509 Public Key Infrastructure (PKI)

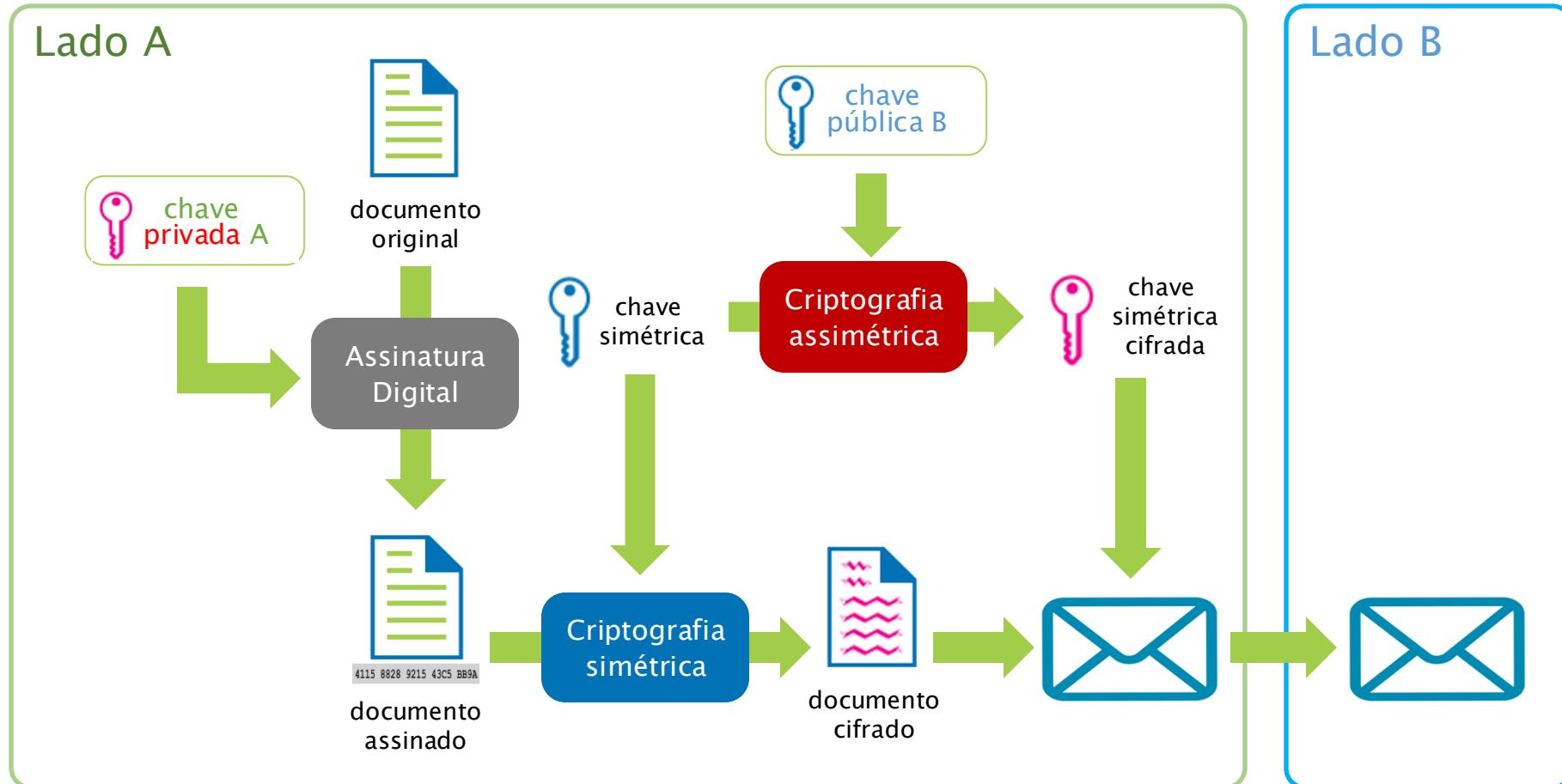


AC EM FASE DE EMISSÃO DE CERTIFICADO
AC EM CREDENCIAMENTO
AC CREDENCIADA DE 1º NÍVEL
AC CREDENCIADA DE 2º NÍVEL

Fonte: [Estrutura da ICP – Brasil](#)

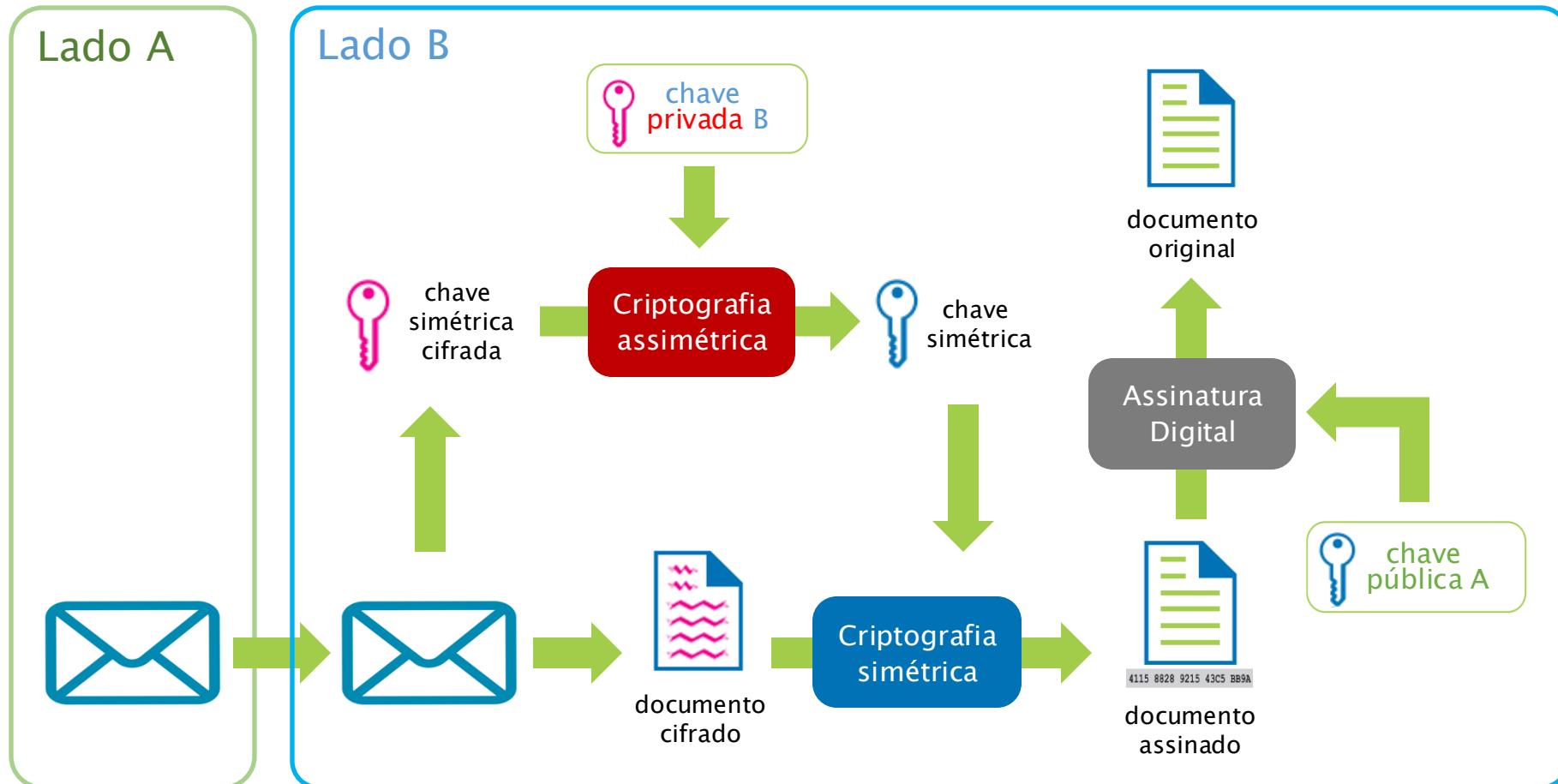
Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital - Envio de Mensagem



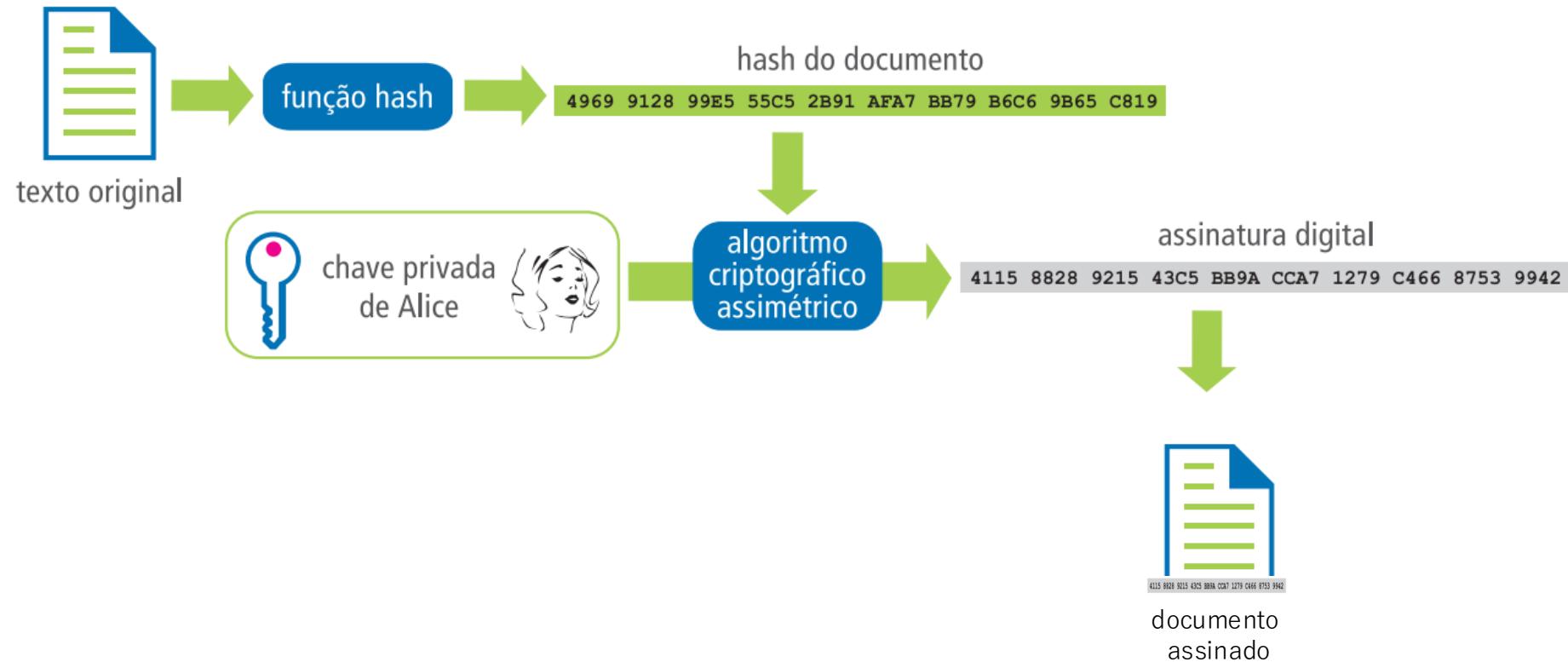
Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital - Recebimento de Mensagem



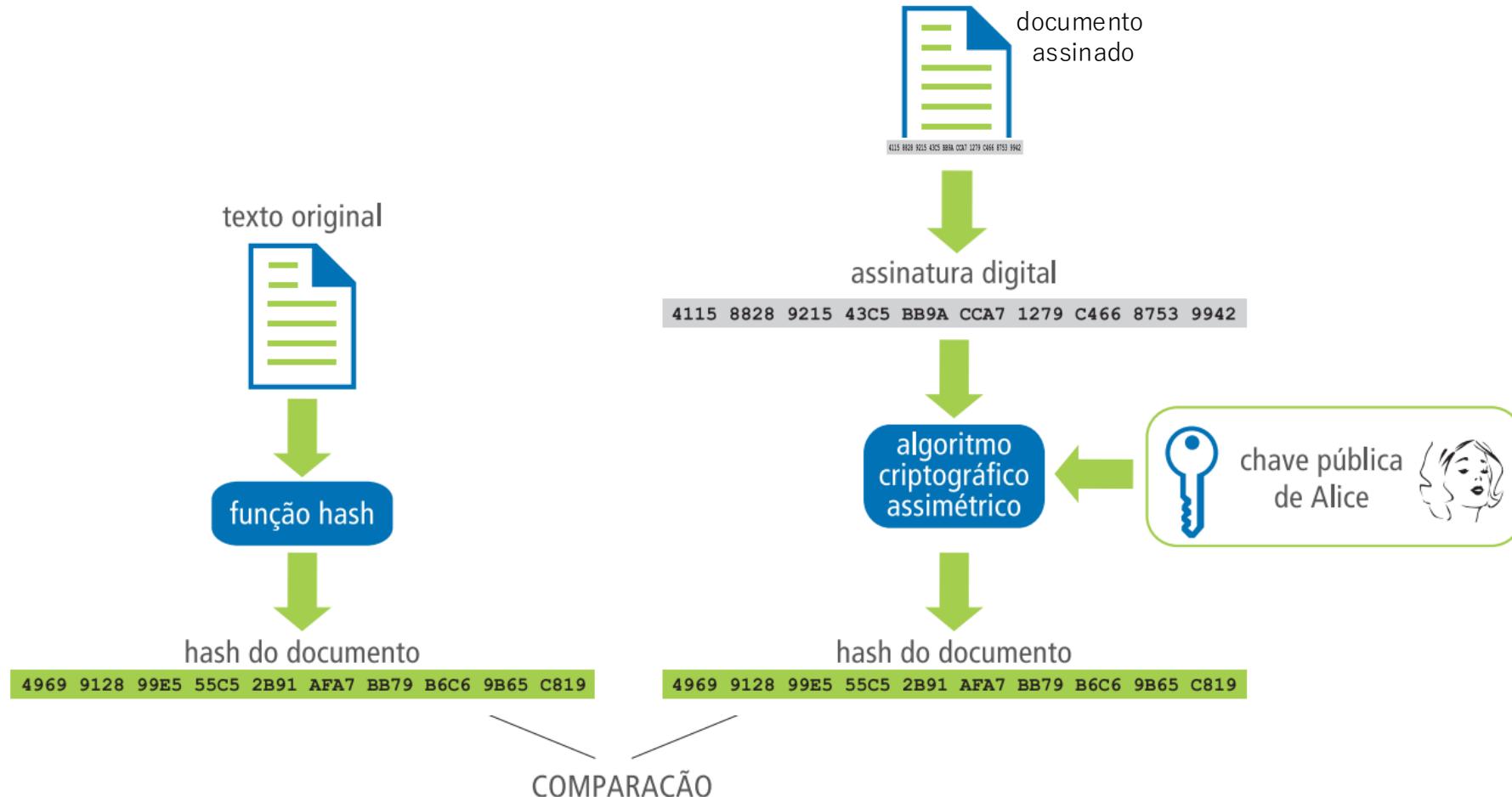
Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital - Assinatura Digital



Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital - Verificação de Assinatura Digital



Obrigado!

Plataforma Node.js

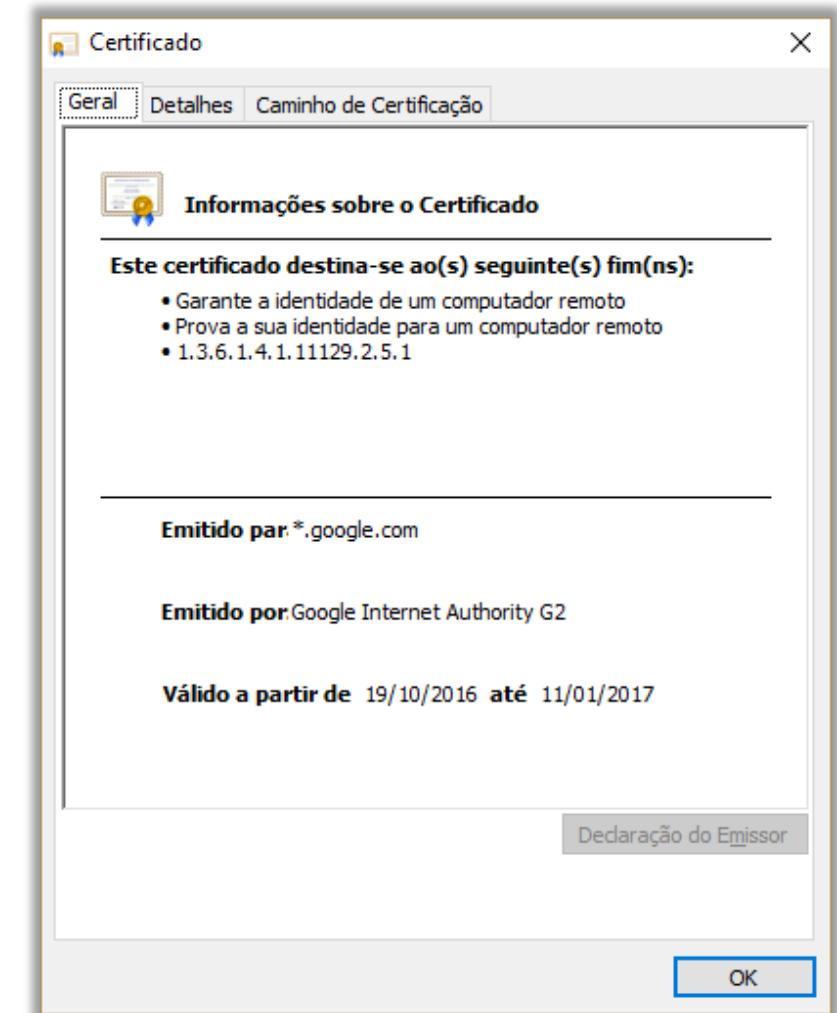
Segurança na Web

Comunicação HTTPS

HTTPS identifica a comunicação segura por meio do protocolo HTTP, na porta 443 (por padrão), utilizando os protocolos TLS ou SSL.

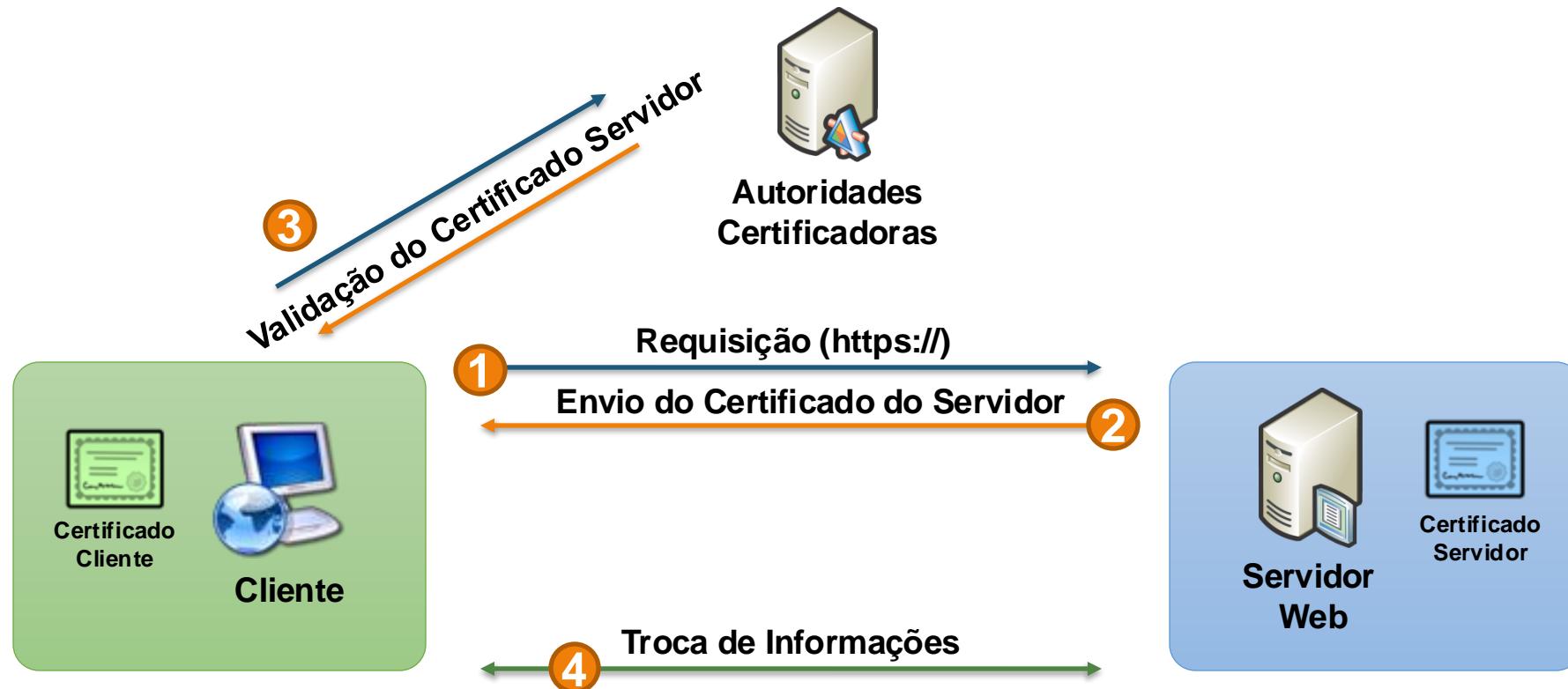
Características

- Fornece uma conexão criptografada com identificação de cliente e servidor
- Baseado em certificados digitais emitidos por autoridades certificadoras
- Requer que servidores Web sejam configurados com certificados digitais
- Requer que os navegadores reconheçam as autoridades certificadoras emissoras dos certificados do servidor



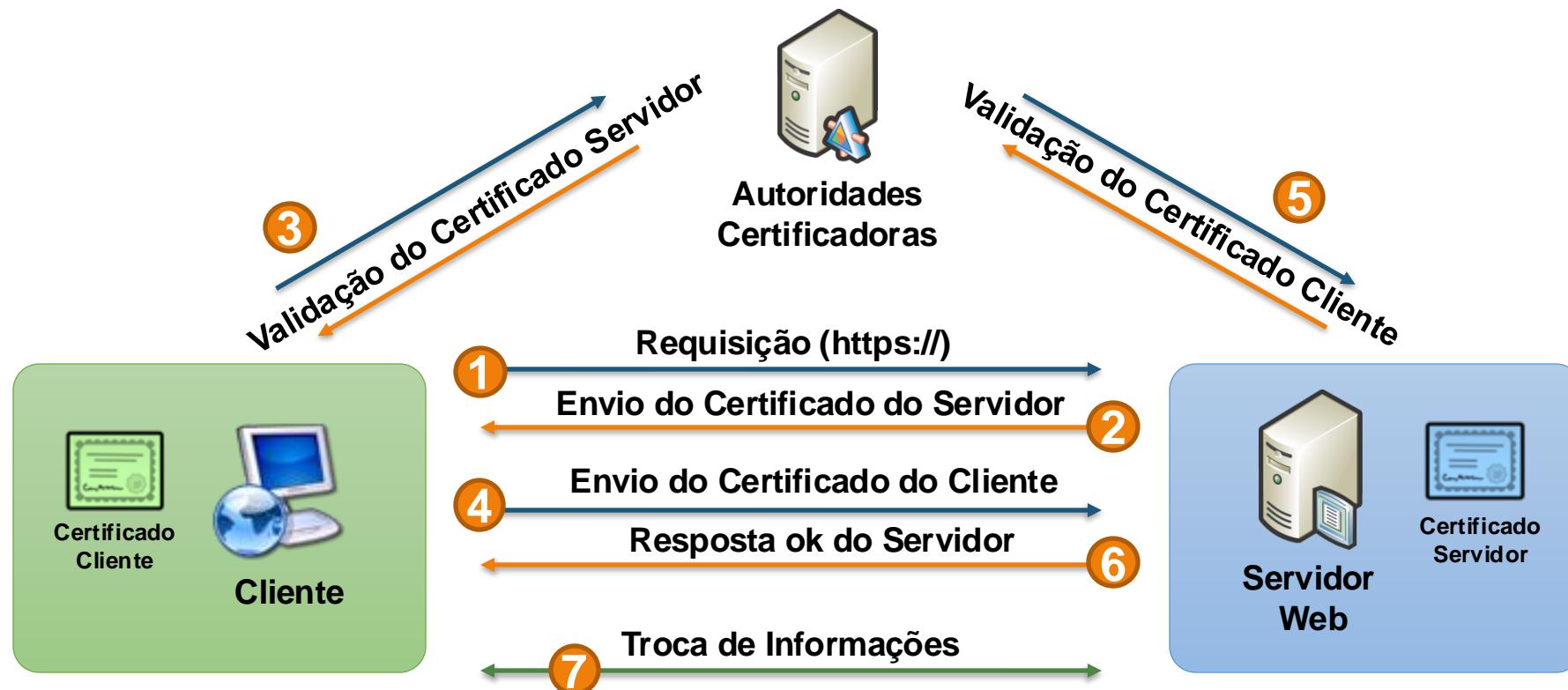
Comunicação HTTPS

HTTPS – Fluxo de Comunicação



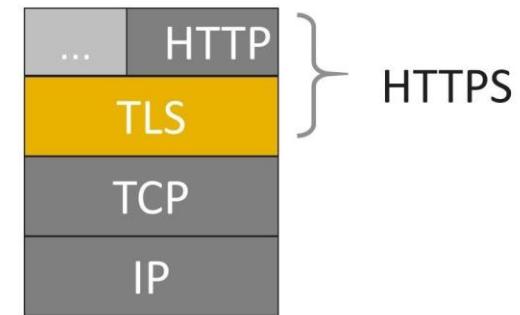
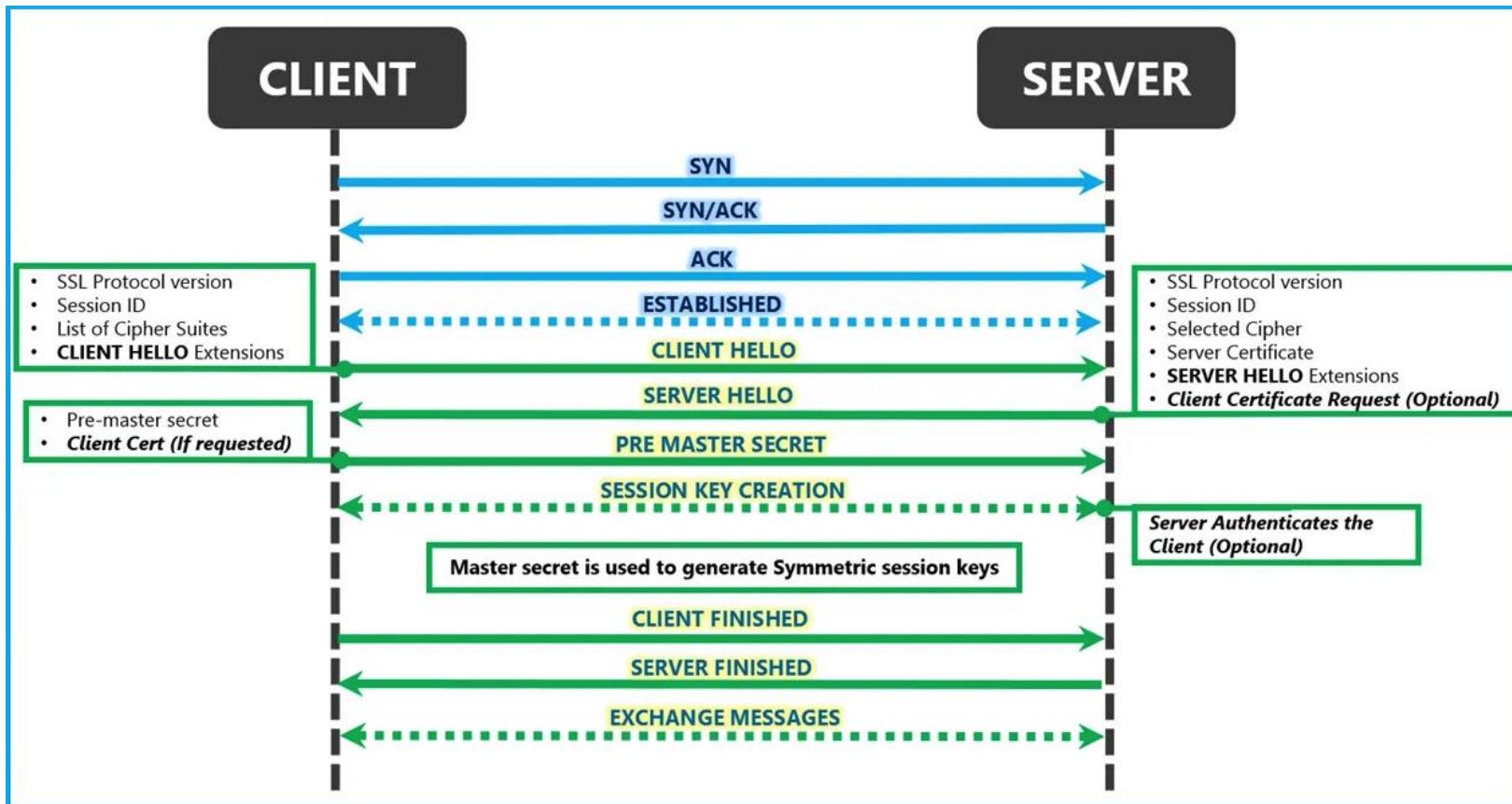
Comunicação HTTPS

HTTPS – Fluxo de Comunicação



Comunicação HTTPS

HTTPS – Fluxo de Comunicação



Fontes: [The HTTPS protocol explained! — Under the Hood | An overview of the SSL Handshake](#)

Comunicação HTTPS

HTTPS – Problemas com Certificados

Um certificado pode apresentar diversos problemas tais como:

- Certificado Expirado
- Certificado de um site diferente do acessado
- Certificado revogado pela autoridade certificadora
- Certificados de autoridades certificadoras desconhecidas
- Certificados assinados pelo próprio site

Testes de Certificados

Utilize o site de testes BADSSL que apresenta um conjunto de sites com certificados com problemas:

<https://badssl.com/>



Autenticação HTTP

Processo para verificar a identidade do usuário de uma aplicação Web.

Esquemas de Autenticação

- Usuário Anônimo + Forms da aplicação (Login)
- Autenticação Basic
- Autenticação Digest
- Autenticação Bearer (Token Authentication)

Provedores / Integrações

- LDAP (Active Directory, Novell NDS)
- Kerberos
- Formulários de login providos pelas aplicações Web

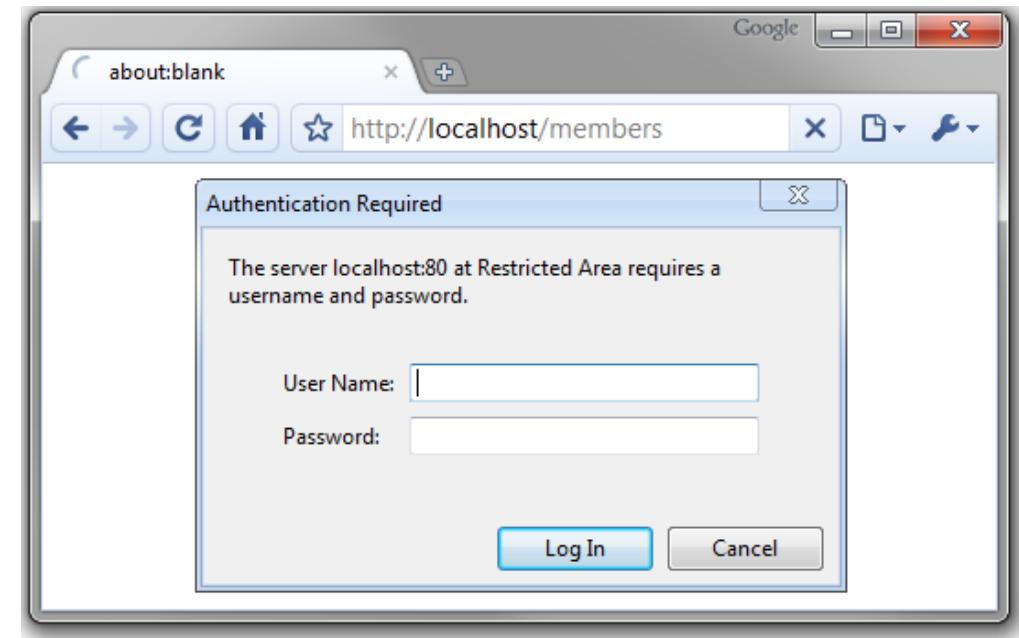
Fonte: RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication (<https://tools.ietf.org/html/rfc2617>)

Autenticação HTTP - Basic



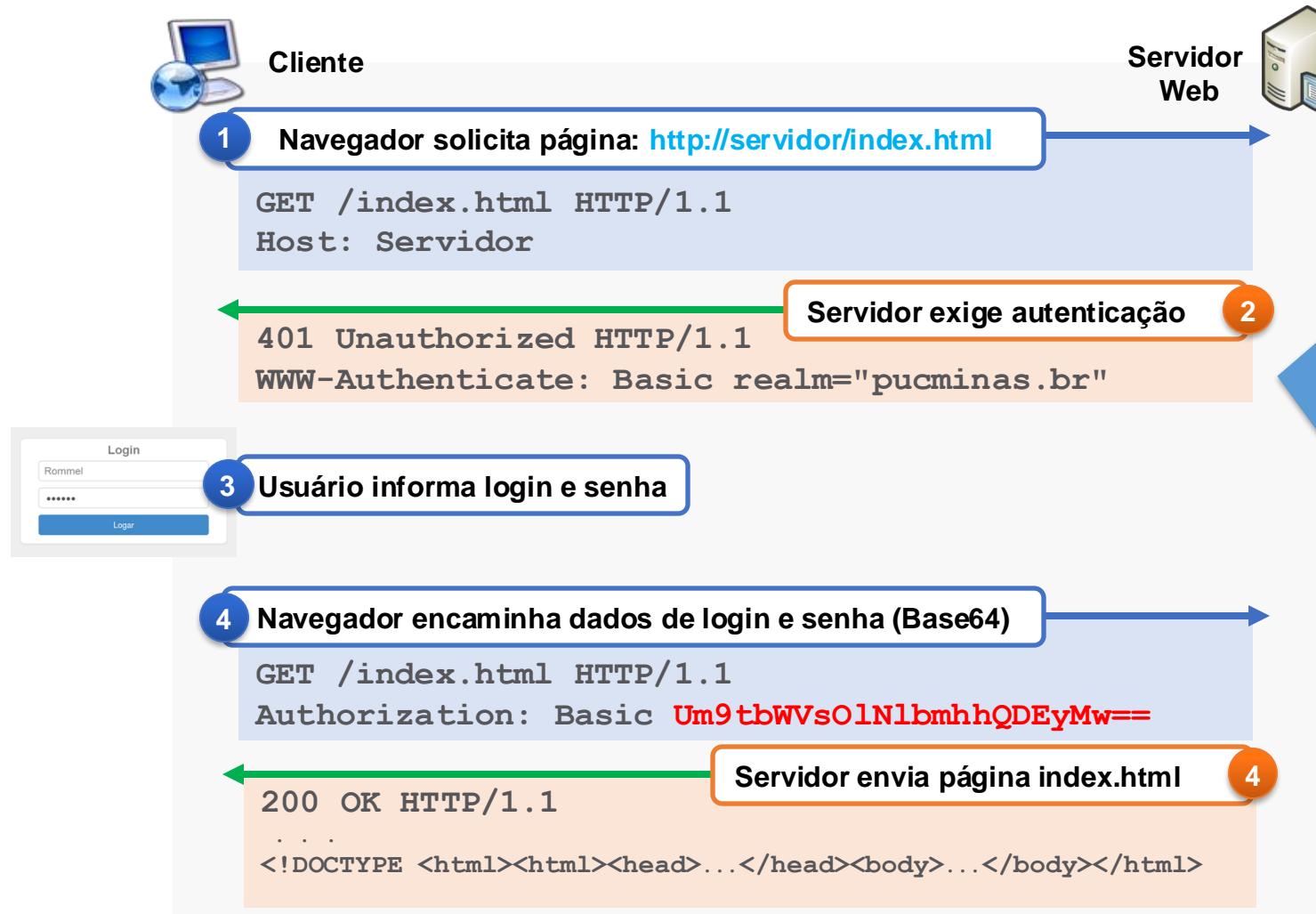
Tela de login exibida pelo próprio browser e envio de *string* codificada em Base64 com informação de usuário e senha.

IMPORTANTE: Recomenda-se utilizar apenas com conexões HTTPS.



Fonte: RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication (<https://tools.ietf.org/html/rfc2617>)

Autenticação HTTP - Basic



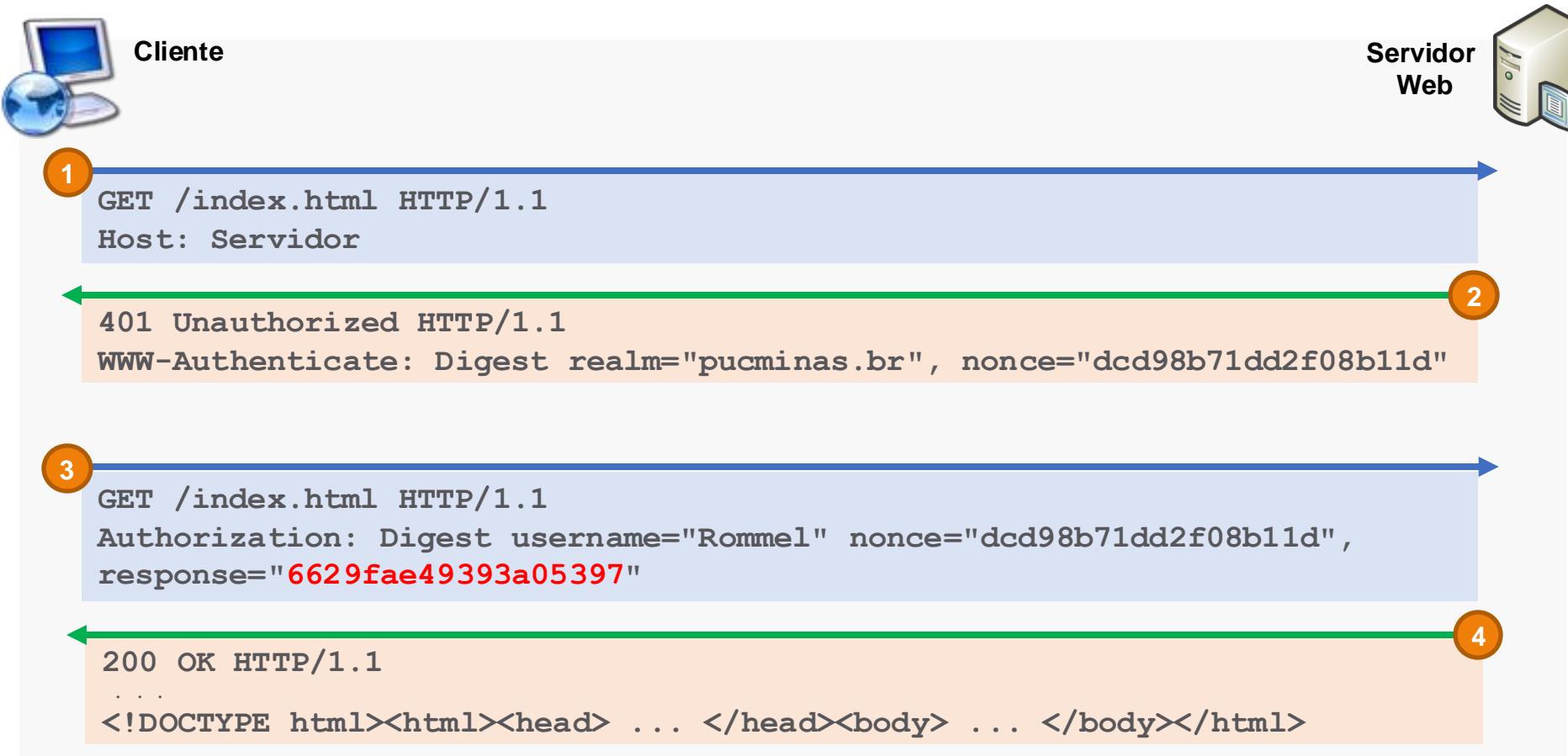
Informações

- **WWW-Authenticate**: Cabeçalho da resposta que exige o envio de dados de autenticação
- **Realm**: Definição do espaço protegido pela autenticação
- **Authorization**: cabeçalho da requisição que leva os dados de autenticação do cliente.
- **Base64**: algoritmo de codificação de dados para a Internet.

Autenticação HTTP – Digest



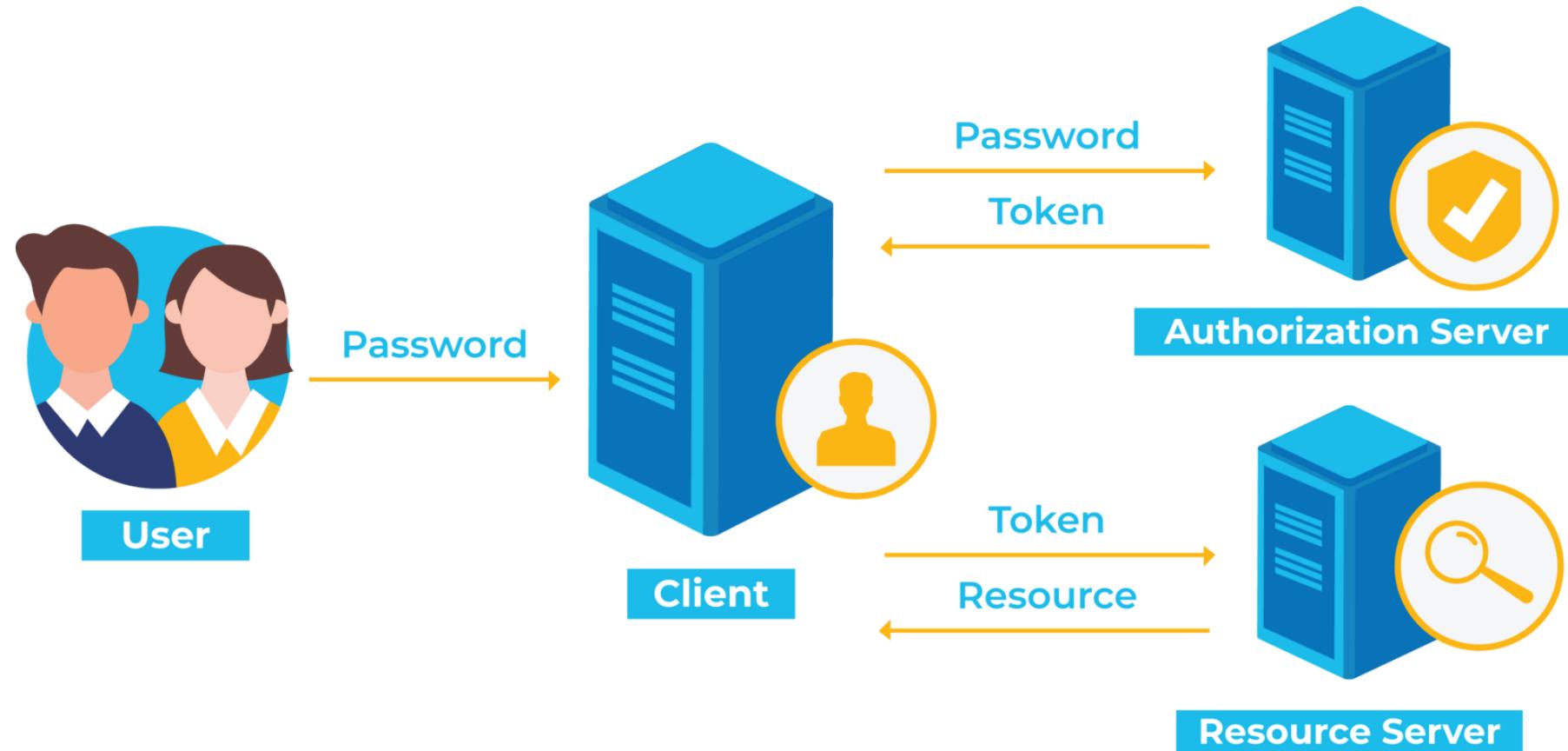
Cliente e servidor não trocam informações de senha, apenas o **hash**.



Autenticação HTTP – Bearer



Token Authentication



Fonte: [What Is Token-Based Authentication? \(Okta\)](#)

okta

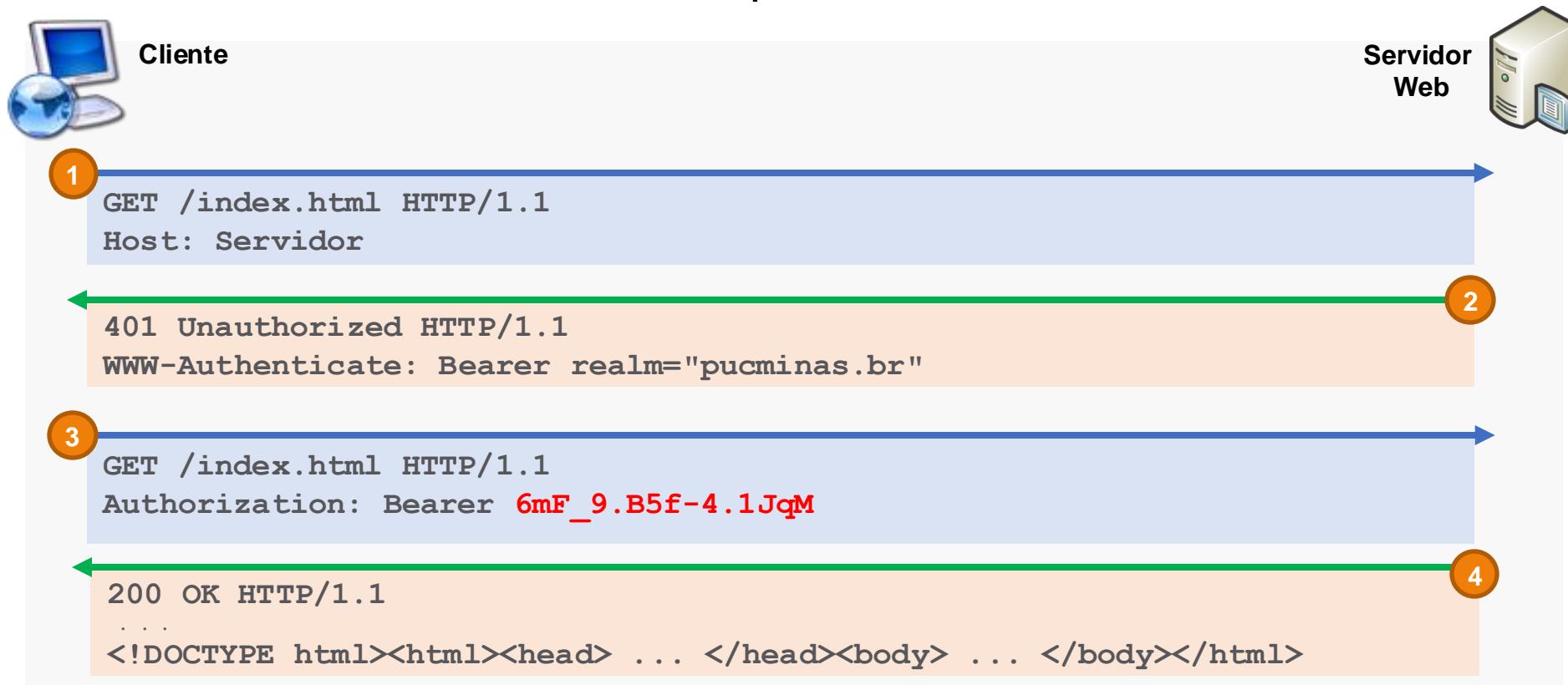
Autenticação HTTP – Bearer



Token Authentication

Cliente e servidor trocam uma token previamente acordada.

IMPORTANTE: Recomenda-se utilizar apenas com conexões HTTPS.



Obrigado!

Plataforma Node.js

JSON Web Token (JWT)

JSON Web Token (JWT)

Padrão que define uma forma compacta e segura de transferência de informações baseada em tokens no formato JSON ([RFC-7519](#)) utilizadas em mecanismos de autenticação e autorização.

Estrutura básica da token JWT

[header]

Informações sobre o tipo da token e o algoritmo de criptografia utilizado

```
base64enc ({  
  "alg": "HS256",  
  "typ": "JWT"  
})
```

[payload]

- Dados da token, denominados Claims que podem ser reservados, públicos ou privados

```
base64enc ({  
  "id": "HS256",  
  "roles": "ADMIN"  
})
```

[signature]

- assinatura da token que garante a confiabilidade da token evitando alterações

```
HMACSHA256 (  
  base64enc(header) + "." +  
  base64enc(payload),  
  secretKey  
)
```

Fonte: [JWT - Auth0](#)

JSON Web Token (JWT)

Campos reservados (claims)

- Iss (Issuer) – Emissor da token JWT
- Sub (Subject) – Assunto da token
- Aud (Audience) – Identifica o público para o qual a token foi emitida
- Exp (Expiration Time) – Horário de expiração da token, a partir do qual não é mais aceita
- Iat (Issued At) – Horário em que a token foi emitida

Fonte: [Wikipedia JSON Web Token](#)

JSON Web Token (JWT)

Exemplo da criação de uma token para informações do usuário via Node.js

Instalação JSONWebToken
npm install jsonwebtoken

```
require ('dotenv').config()
const JWT = require('jsonwebtoken')
const express = require("express")
const app = express()

app.get("/", function(req, res){
    JWT.sign({ userID: 'abc123' },
        process.env.SECRET_KEY,
        { algorithm: 'HS256' },
        (err, token) => {
            if (err) console.error('Erro: ' + err);
            else console.log (token)
        })

        res.send ('Hello JWT')
});

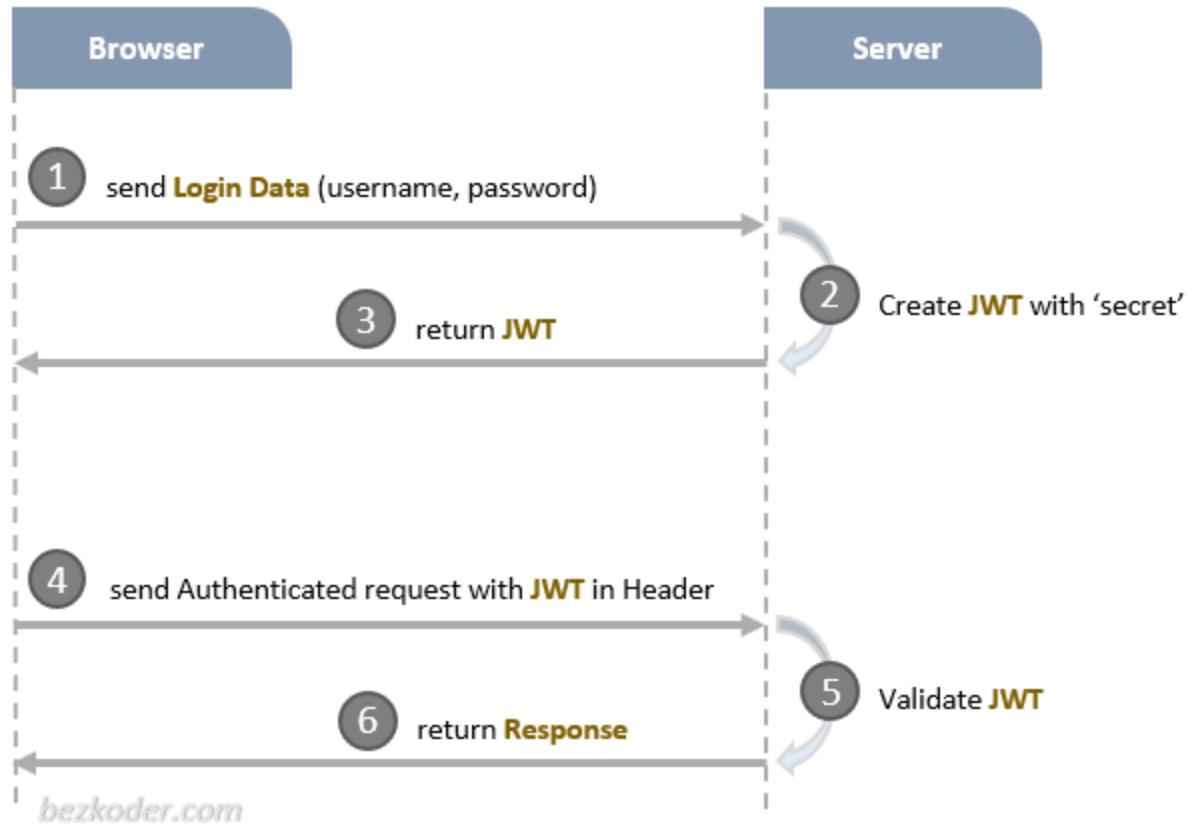
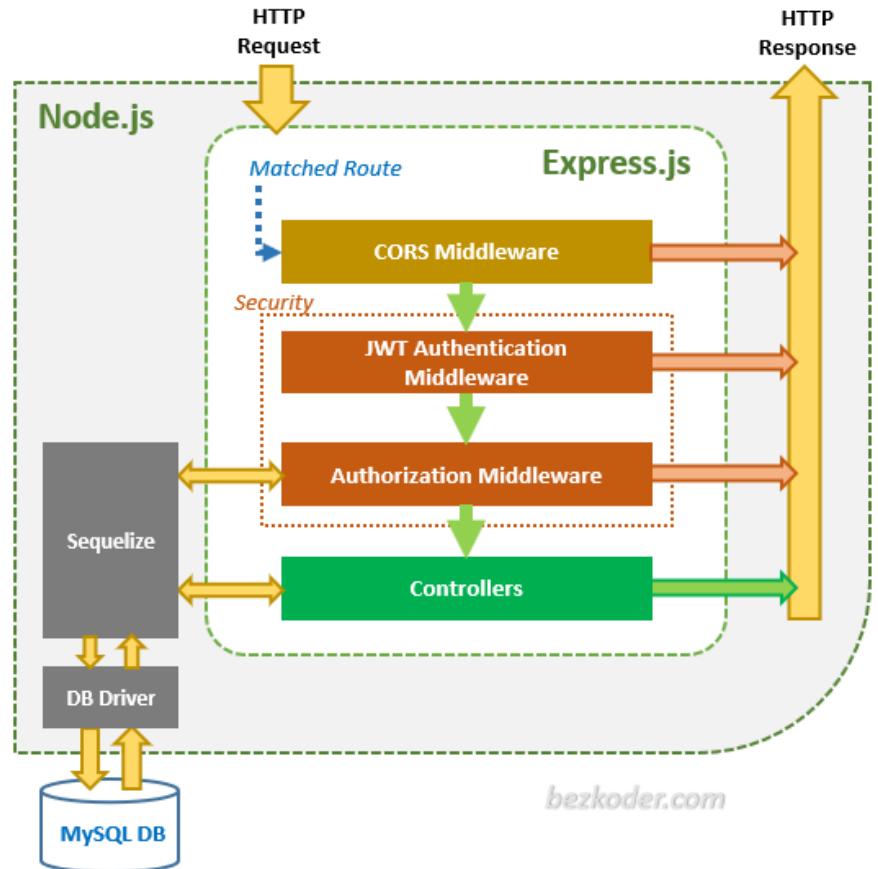
app.listen(3000);
```

JSON Web Token (JWT)

Gerando uma chave forte em JavaScript com Node.js

```
>> node -e "console.log(require('crypto').randomBytes(256).toString('base64'));"  
  
//Exemplo de chave secreta gerada  
GdLOU8owaDRw3kF23gicjzWHLsICP5pP365G/8Jd1Vm90qaphvQFRJAtZOFU2BuAj+y0bat3ImenitbJp  
ZF+9toj9+4L7A9XZW98kxLo+oGJhXJwlwc78bvJaHH7zzx1VFBZ9ipGoIz57xqlhqZE7buH1nPjRGOSH  
QuqZ7XxZ1/0dk7Veib182SboX3G8gCI38FIBmiJM4quWDv3MZOIP1ElQlWgVBJH/y6AxZkxSMAn8DyZ9x  
a94pKxwXFMQPZrbD0+rhnEC0rfi5kvCxVLM/w3rB3EHwBHNNaQ941QKpIMjqAg8iIlLUegfZ4tJs49tGV  
fYgwtBKAKXwI6YuCIw==
```

Autenticação e autorização com JWT



Fonte: [Node.js Express: JWT example | Token Based Authentication & Authorization](#)

Obrigado!

Plataforma Node.js

Open Authorization (OAuth)

OAuth - Open Authorization



Descrição

- Framework aberto definido pelo IETF (RFC 6749)
- Foco na autenticação e autorização de recursos na Web

Características

- Evita exposição de senhas
- Facilita a interoperabilidade (Web, Mobile, Server)
- Controla a validade e o escopo do acesso concedido

OAuth - Papéis



Dono do
Recurso



Aplicação
Cliente

Papéis do OAuth

O mecanismo de autorização OAuth define quatro papéis.



Servidor de
Autorização



Servidor de
Recursos



OAuth - Papéis



Dono do
Recurso



Aplicação
Cliente

Dono do Recurso

Entidade que possui recursos na rede e pode ser solicitado a autorizar o acesso a estes recursos protegidos.

Ex: Usuário final



Servidor de
Autorização



Servidor de
Recursos



OAuth - Papéis



Dono do
Recurso



Aplicação
Cliente

Aplicação Cliente

Sistemas envolvidos que são utilizados para acessar recursos disponíveis na rede. Podem ser confidenciais ou públicos.

Ex: aplicações móveis e sites na Web



Servidor de
Autorização



Servidor de
Recursos



OAuth - Papéis



Dono do
Recurso



Aplicação
Cliente

Servidor de Autorização

Sistema que controla a geração de tokens de acesso para as aplicações cliente.

Ex: Google Accounts



Servidor de
Autorização



Servidor de
Recursos



OAuth - Papéis



Dono do
Recurso



Aplicação
Cliente

Servidor de Recursos

Ambiente que hospeda
recursos protegidos
na rede.

Ex: Google Fotos



Servidor de
Autorização



Servidor de
Recursos



OAuth - Access Token

- O Access Token é uma **credencial para acesso** a um recurso protegido.
- Trata-se de uma **string em formato específico** de acordo com a aplicação em questão
- Uma Access Token é obtida de acordo com o **tipo de autorização**
- A Access Token substitui a necessidade de **usuário e senha**



OAuth – Tipos de Autorização

O protocolo OAuth 2 oferece 4 tipos de autorização:

- Código de Autorização (*Authorization Code*)
- Autorização Implícita (*Implicit Grant*)
- Credenciais do Usuário (*Resource Owner Password Credentials*)
- Credenciais do Cliente (*Client Credentials*)

OAuth – Tipos de Autorização

O protocolo OAuth 2 oferece 4 tipos de autorização:

- **Código de Autorização**

Ocorre quando a *Aplicação Cliente* é uma aplicação Web ou nativa e mantém uma chave secreta.

Ex: Site X quer acessar seus dados no **facebook**

- Autorização Implícita

- Credenciais do Usuário

- Credenciais do Cliente

OAuth – Tipos de Autorização

O protocolo OAuth 2 oferece 4 tipos de autorização:

- Código de Autorização
- **Autorização Implícita**

Ocorre quando a *Aplicação Cliente* é baseada no browser, em linguagem de script e não pode manter uma chave secreta.

Ex: [Aplicações SPA \(Single Page Web\)](#)

- Credenciais do Usuário
- Credenciais do Cliente

OAuth – Tipos de Autorização

O protocolo OAuth 2 oferece 4 tipos de autorização:

- Código de Autorização
- Autorização Implícita
- **Credenciais do Usuário**

Ocorre quando a *Aplicação Cliente* é próxima do *Servidor de Autorização* e requer usuário e senha, normalmente, ambos feitos pela mesma empresa.

Ex: Aplicativo "Gerenciador de Negócios" do **facebook**

- Credenciais do Cliente

OAuth – Tipos de Autorização

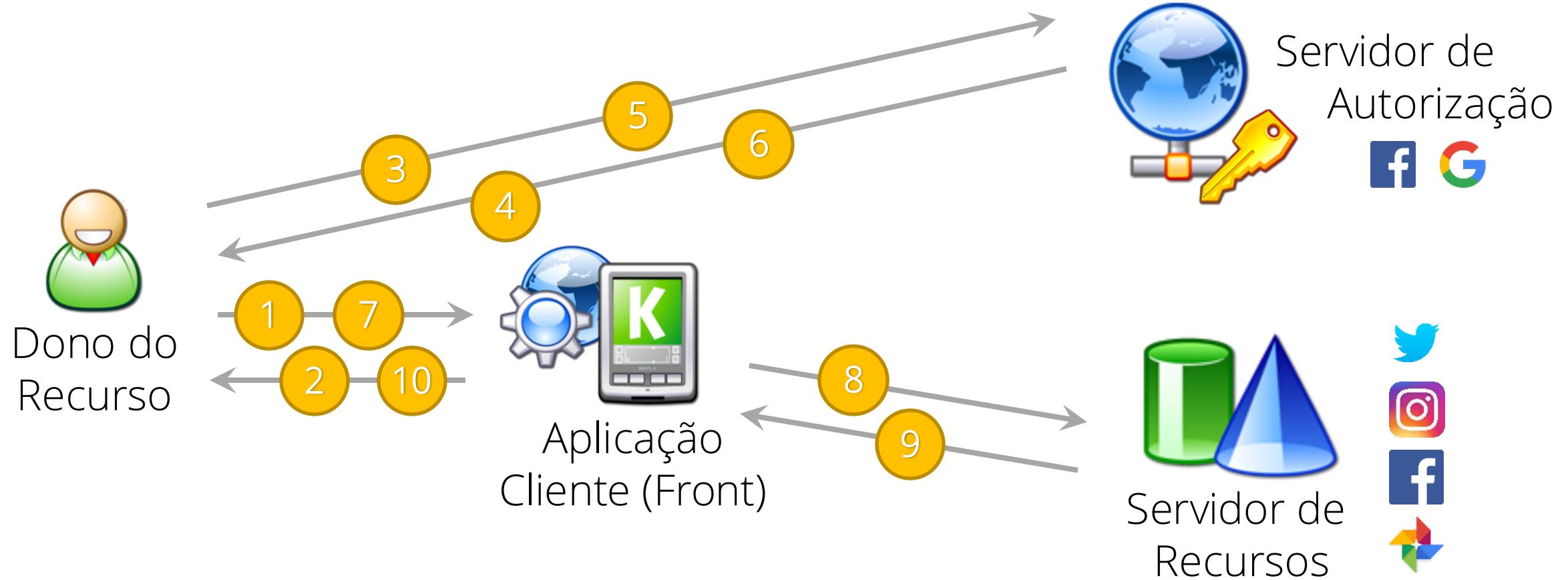
O protocolo OAuth 2 oferece 4 tipos de autorização:

- Código de Autorização
- Autorização Implícita
- Credenciais do Usuário
- **Credenciais do Cliente**

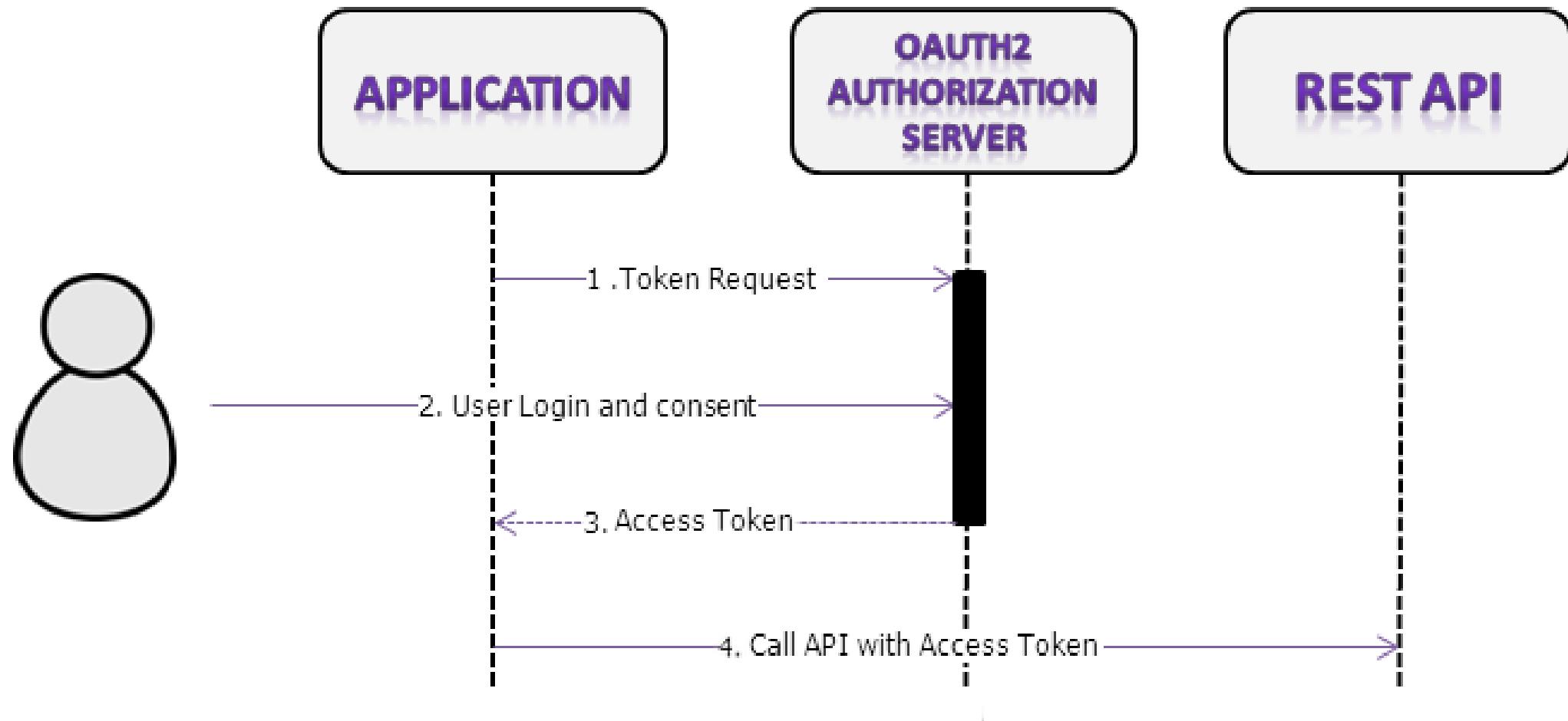
Ocorre quando a *Aplicação Cliente* é a proprietária dos recursos e não o usuário final.

Ex: Cloud Azure acessando dados em storage interno

OAuth - Fluxos - Autorização Implícita

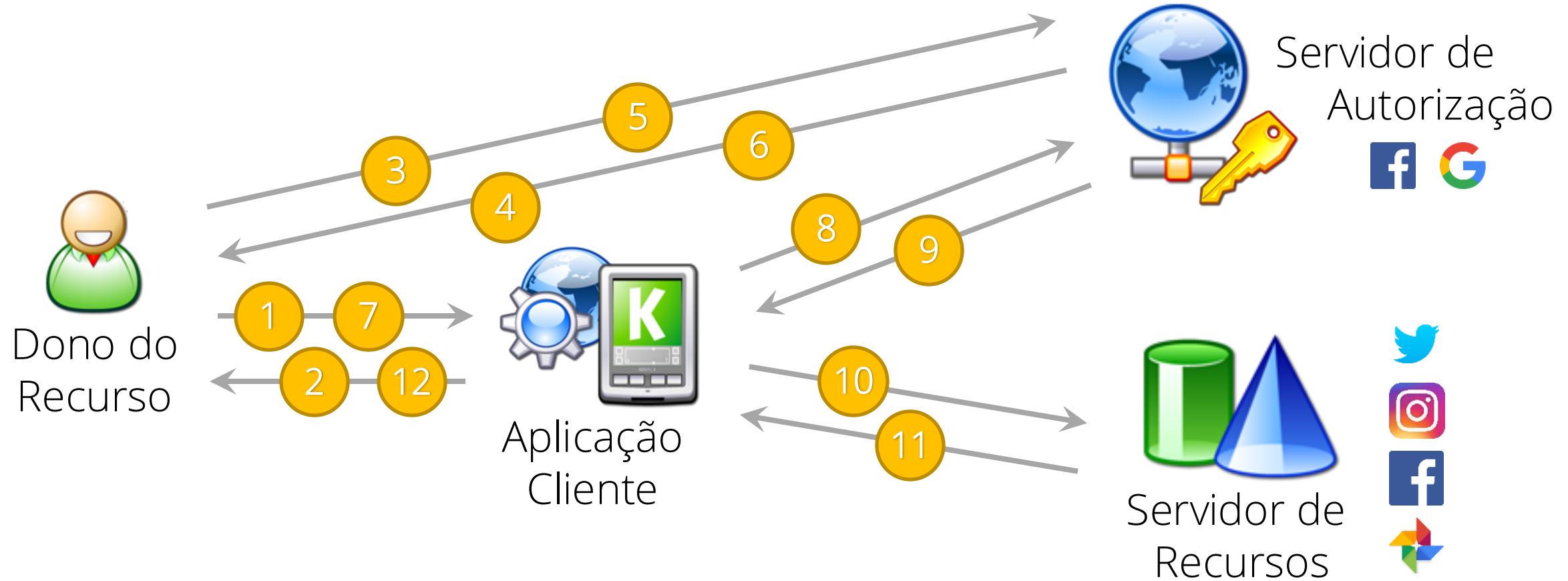


OAuth - Fluxos - Autorização Implícita

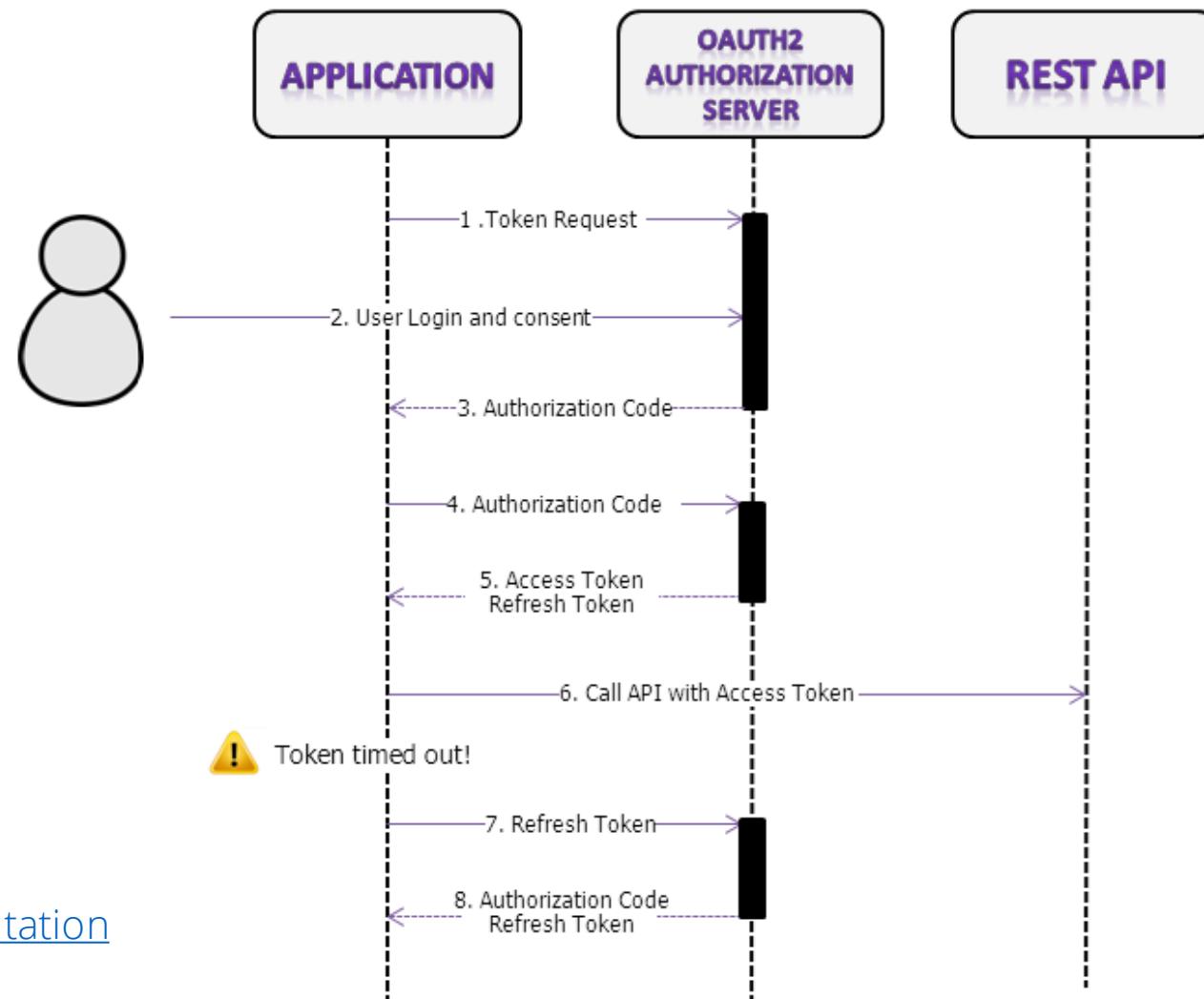


Fonte: [Qeo Native Documentation](#)

OAuth - Fluxos - Código de Autorização



OAuth - Fluxos - Código de Autorização



Fonte: [Qeo Native Documentation](#)

Plataforma Node.js

Porque Node.js

Porque Node se chama Node? ☺

Inicialmente criado para substituir servidores Web bloqueantes como o Apache, o Node foi além e acabou sendo uma plataforma para criar aplicações distribuídas de diversas naturezas, um nó em uma estrutura maior
→ daí ... node.

Originally, Dahl called his project web.js. It was merely a webserver, an alternative to Apache and other "blocking" servers. But the project soon grew beyond his initial webserver library, expanding into a framework that could be used to build, well, almost anything. So he rechristened it node.js.

Fonte: [The Node Ahead: JavaScript leaps from browser into future • The Register](#)



286



Node Package Manager (NPM)

Prof. Rommel Vieira Carneiro



Plataforma Node.js

Node.js Shebang

Node.js shebang (unix-like OSs)

Ao invés de executar seu script por meio da linha de comando node script.js, é possível o arquivo do script executável (chmod a+x script.js) e permitir a execução diretamente pela linha de comando.

Shebang (#!) → linha colocada no início do arquivo que informa ao SO qual é o interpretador do script que está no arquivo. Veja o exemplo:

```
#!/usr/bin/env node
console.log ('Arquivo auto executável')
```

Fonte: [Node.js shebang](#)

Node.js shebang (unix-like OSs)

Começar o shebang com /usr/bin/env permite que o SO inicie um novo shell com o PATH atual para executar o comando que se segue.

Torna o script mais portável. No exemplo, o SO vai procurar o Node.js em qualquer lugar do PATH.

```
#!/usr/bin/env node
console.log ('Arquivo auto executável')
```

Fonte: [Node.js shebang](#)

Plataforma Node.js

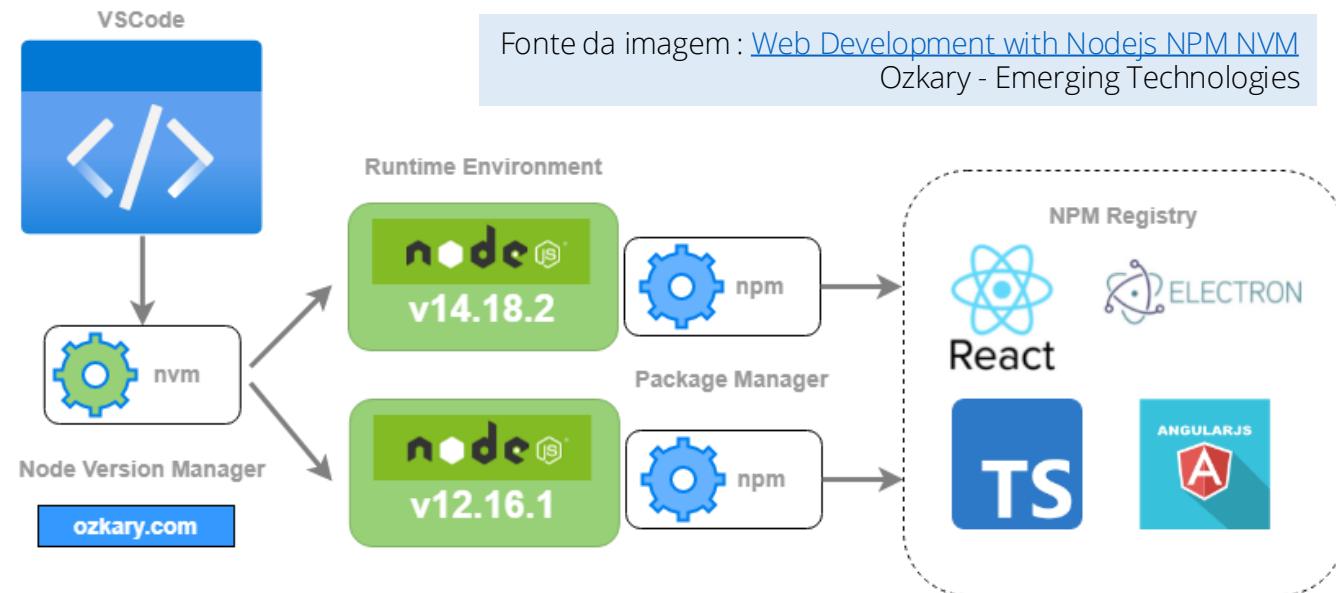
Node Version Manager (NVM)

Node.js Versions (nvm)



Em ambientes profissionais, muitas vezes se torna necessário utilizar diversas versões da plataforma Node.js para projetos diferentes.

Com o **Node Version Manager (NVM)**, conseguimos gerenciar diversas versões do Node.js no mesmo computador.



Node.js Versions (nvm)



1) Instalação a partir do repositório do GitHub (Linux e MacOS)

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
```

or

```
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
```

2) Configuração do PATH no shell do OS (Linux e MacOS)

→ Acrescente o script de configuração para o NVM no `~/.zshrc` ou `~/.bashrc`

```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

Node.js Versions (nvm)



3) Resumo dos principais comandos do NVM

Exemplos de Comandos	Descrição
nvm ls	// Lista as versões instaladas
nvm install latest	// Instala a última versão
nvm install --lts	// Instala a versão LTS
nvm install 14.17.6	// Instala um versão específica
nvm install 14.17.6	// Remove uma versão específica
nvm alias default 14.17.6	// Cria um alias para uma versão instalada
nvm use 14.17.6	// Seleciona uma versão para ser utilizada
nvm use default	
nvm run default script.js	// Executa o script.js em versão específica

Plataforma Node.js

Módulos Populares

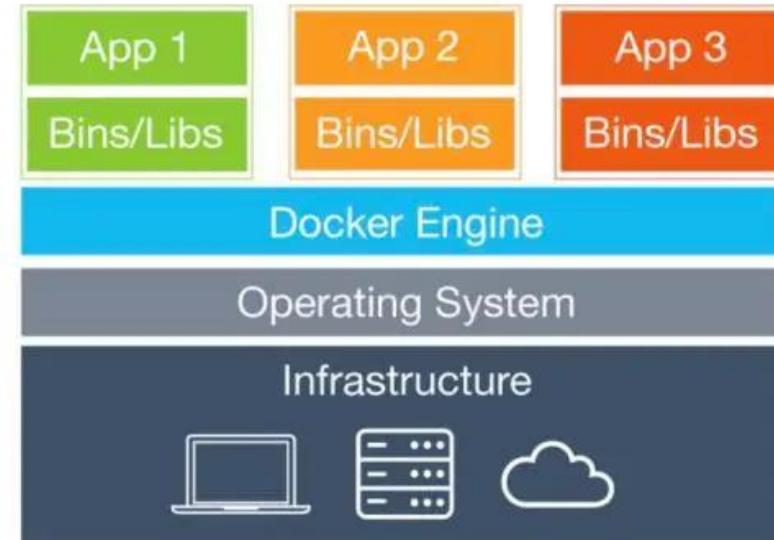
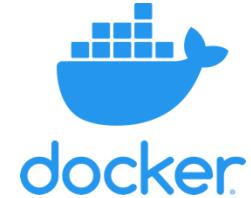
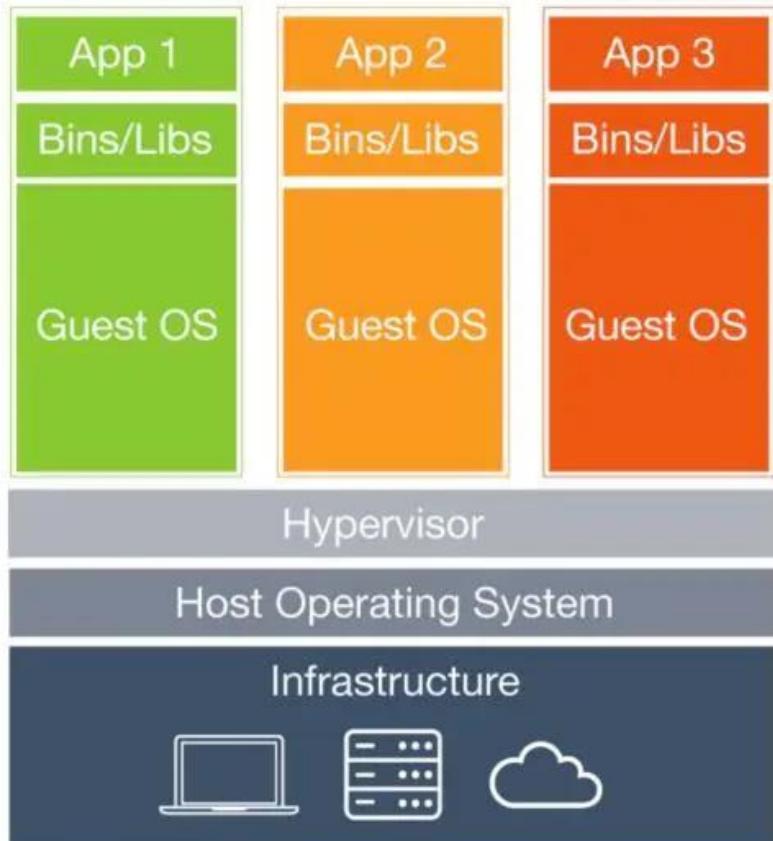
Módulos JavaScript Populares

Módulo	Descrição
Lodash	
PM2	
Axios	
Cherio	
Mocha	
Moment	
Babel	
Socket.io	
Mongoose	
Nodemailer	
Dotenv	
Passport	
Puppeteer	

Plataforma Node.js

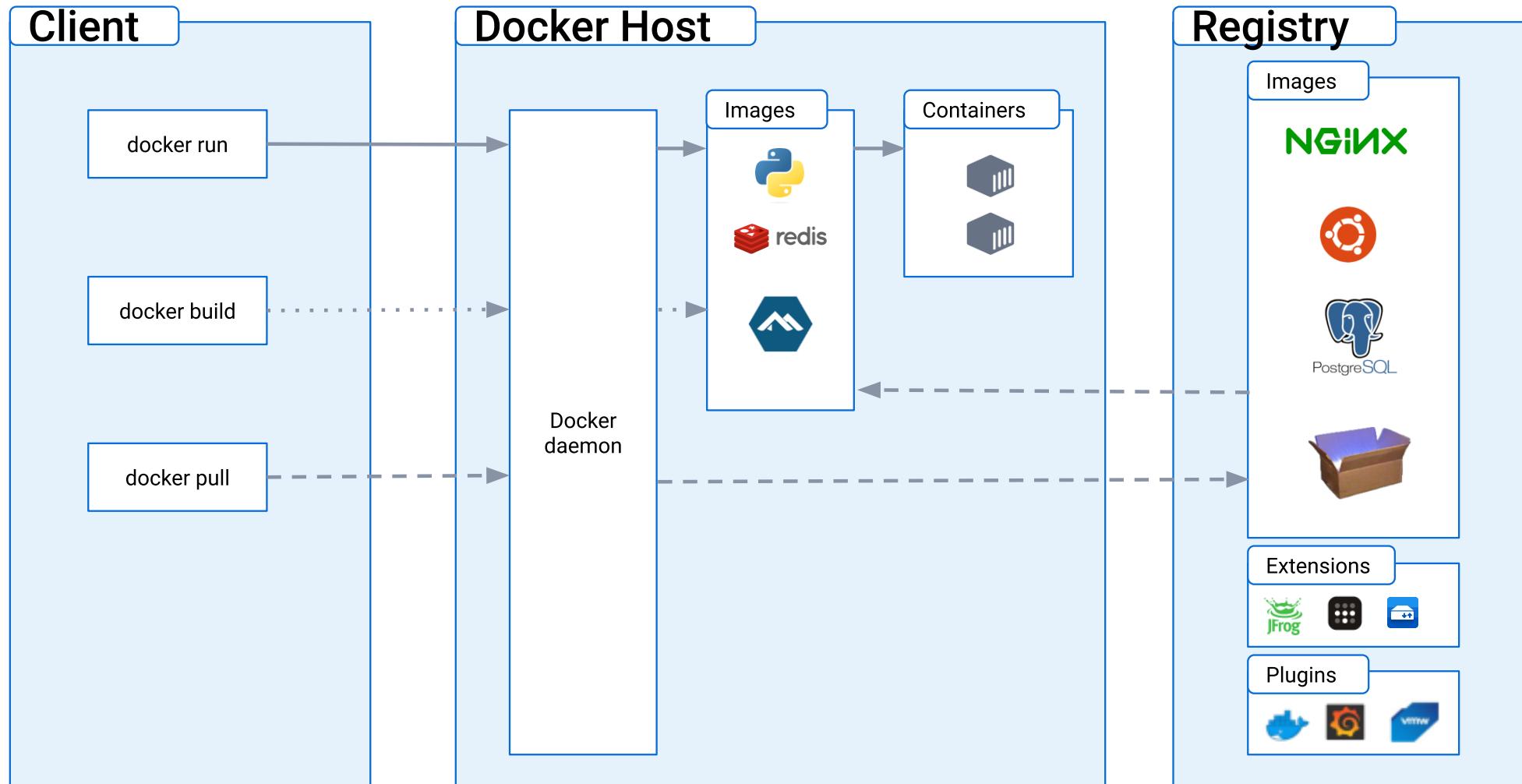
Containers Docker

Máquinas virtuais vs Containers



Fonte: [What is Docker? The spark for the container revolution](#) (InfoWorld)

Containers e Docker



Fonte: [Docker overview | Docker Documentation](#)



docker

```
# Informa a imagem a ser utilizada como base
FROM node:lts-alpine

# Cria o diretório para a aplicação
WORKDIR /app

# Copia o package.json para o diretório da aplicação e instala as dependências
COPY package*.json .
RUN npm install

# Copia o código fonte para o diretório da aplicação
COPY ..

# Define porta de execução da aplicação
EXPOSE 80

# Executa a aplicação
CMD [ "npm", "start" ]
```

Docker - Principais Comandos



docker

Exemplos de Comandos

Descrição

`docker pull node:lts-alpine`

Faz download de uma imagem do Docker Hub.
No exemplo a imagem node com tag lts-alpine

`docker build . -t node-app`

Cria uma imagem a partir de um Dockerfile

. Especifica o caminho do Dockerfile (pasta corrente)
-t Nomeia a imagem (tag) com node-app

`docker images`

Lista todas as imagens disponíveis localmente.

`docker run -d -p 80:3000 --name app node-app`

Cria um container a partir de uma imagem.

-d Executa em segundo plano
-p 80:3000 Associa portas 80 (host):3000 (container)
-name app Nomeia o container como app
node-app Informa a imagem a ser usada (node-app)

`docker logs app`

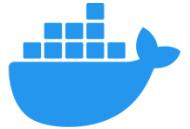
Exibe os logs de um container.

`docker exec -it app /bin/bash`

Executa um terminal interativo no container

-i Executa em modo interativo
-t Aloca um pseudo-terminal (tty)

Docker - Principais Comandos



docker

Exemplos de Comandos

Descrição

`docker run -it alpine`

Cria um container de nome aleatório a partir da imagem do Alpine Linux (alpine) e abre um terminal (shell) no container para interação
-i Executa em modo interativo
-t Aloca um pseudo-terminal (tty)

`docker ps -a`

Lista os containers
-a inclui todos os containers, inclusive os parados

`docker start <container>`

Inicia um container que está parado

`docker pause <container>`

Pausa um container em execução

`docker stop <container>`

Finaliza um container em execução

`docker rm <container>`

Remove um container

`docker rmi <image>`

Remove uma imagem

Docker - Exemplos



Docker Compose

Aplicações Multi Container



Fonte:

- https://docs.docker.com/get-started/07_multi_container/
- <https://docs.docker.com/compose/migrate/>



303



Node Package Manager (NPM)

Prof. Rommel Vieira Carneiro



Obrigado!

Plataforma Node.js

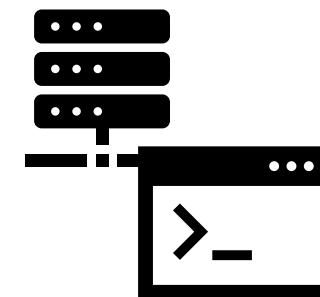
Web Scraping com Node.js

Web Scraping com Node.js - Tópicos

- Conceitos de Web Scraping
- Ferramental utilizado
 - Cliente HTTP – Axios
 - Manipulador DOM – Chero
- Exemplos



**Sites e
Conteúdo Web**



**Ferramentas
de Scraping**



**Dados
Estruturados**

Conceitos de Web Scraping

O Web Scraping ou “raspagem de dados” é o processo de coleta de dados em sites Web que não oferecem uma API pronta para o fornecimento de dados.

Existem bibliotecas específicas que facilitam o processo de Web Scraping:

- [Axios](#) – HTTP Client
- [Cherio](#) – jQuery for the Server
- [Puppeteer](#) – API para navegador Chrome



HTTP Client para Node.js – Axios

Axios é uma biblioteca que oferece um cliente HTTP baseado em promises compatível com navegadores e com o Node.js

A X I O S

Instalação – Node.js

```
npm install axios
```

Instalação – Navegador

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

Fonte: [Axios Getting Started](#)

Manipulador do DOM - Cherio

Cherio é uma implementação do núcleo do jQuery projetado para um ambiente servidor como o Node.js. O Cherio processa um hipertexto HTML e fornece uma API para manipular os elementos de forma similar ao DOM.

Instalação – Node.js

```
npm install cherio
```

Fonte: [Cherio Web Site](#)

Web Scrap - Exemplo Books To Scrap

```
const axios = require ('axios')
const cheerio = require ('cheerio')

app.get ('/books', (req, res) => {
  axios ('http://books.toscrape.com/index.html')
    .then (response => response.data)
    .then (data => {
      const $ = cheerio.load (data)
      const books = []
      let i = 0
      $('.product_pod h3 a').each(function () {
        books.push ( { id: i++, titulo: $(this).attr('title') } )
      })
      res.json (books)
    })
})
})
```

Web Scrap - Exemplo Notícias UOL

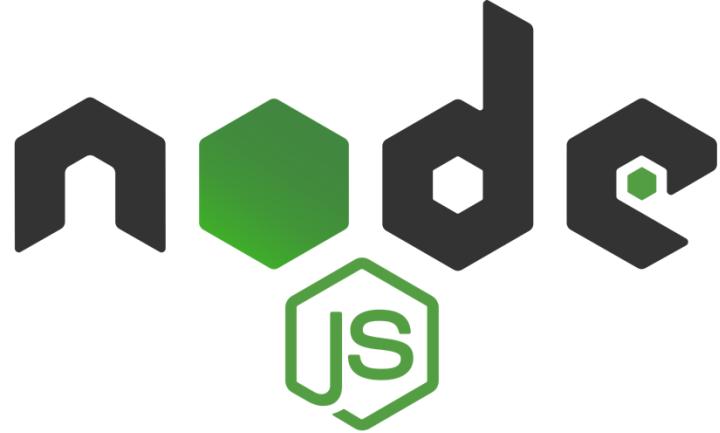
```
const axios = require ('axios')
const cheerio = require ('cheerio')

app.get('/news', function (req, res) {    axios('https://noticias.uol.com.br/')
    .then(response => response.data)
    .then (data => {
        const $ = cheerio.load(data)
        const noticias = []

        $('.thumbnails-wrapper', html).each(function () {
            const titulo = $(this).find('a').attr('href')
            const imagem_link = $(this).find('a img').attr('src')
            const link = $(this).find('a h3').text()
            noticias.push({ titulo, link, imagem_link })
        })
        res.json(noticias)
    })
})
```

Obrigado!

Servidor HTTPS com Node.js



Obrigado!