



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Fundamentos de NodeJS

QXD0020 - Desenvolvimento de Software para Web

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Introdução
- Criando um projeto Node
- Criando um servidor web com Node

Introdução



Introdução

Node

- É uma cross-platform runtime de código aberto que permite que desenvolvedores criem aplicações server-side em JS
- Executada “diretamente” no sistema operacional, fora do contexto do navegador
- Prover suporte a API mais tradicionais dos sistemas operacionais
 - Ex: HTTP, FileSystem

Introdução

História

- Enquanto a web tem 30 anos, JavaScript 26, Node tem apenas 12 anos
- Antes do sucesso do Node, a Netscape havia investido no LiveWire, o ambiente capaz de criar páginas web dinâmicas usando JavaScript no server-side, no entanto não obteve sucesso
- Aplicações server-side com JavaScript se popularizam a partir da introdução do Node.js
 - Fator decisivo: Timing
 - JavaScript passou ser utilizado em aplicações de maior porte graças a Web 2.0. Ex: Flickr, Gmail, etc.
 - Engine JavaScript melhoraram consideravelmente devido a competição entre navegadores
- Node usa a V8 ou Chrome V8, uma engine open-source JavaScript do Projeto Chromium que evoluiu bastante devido a essa competição

Introdução

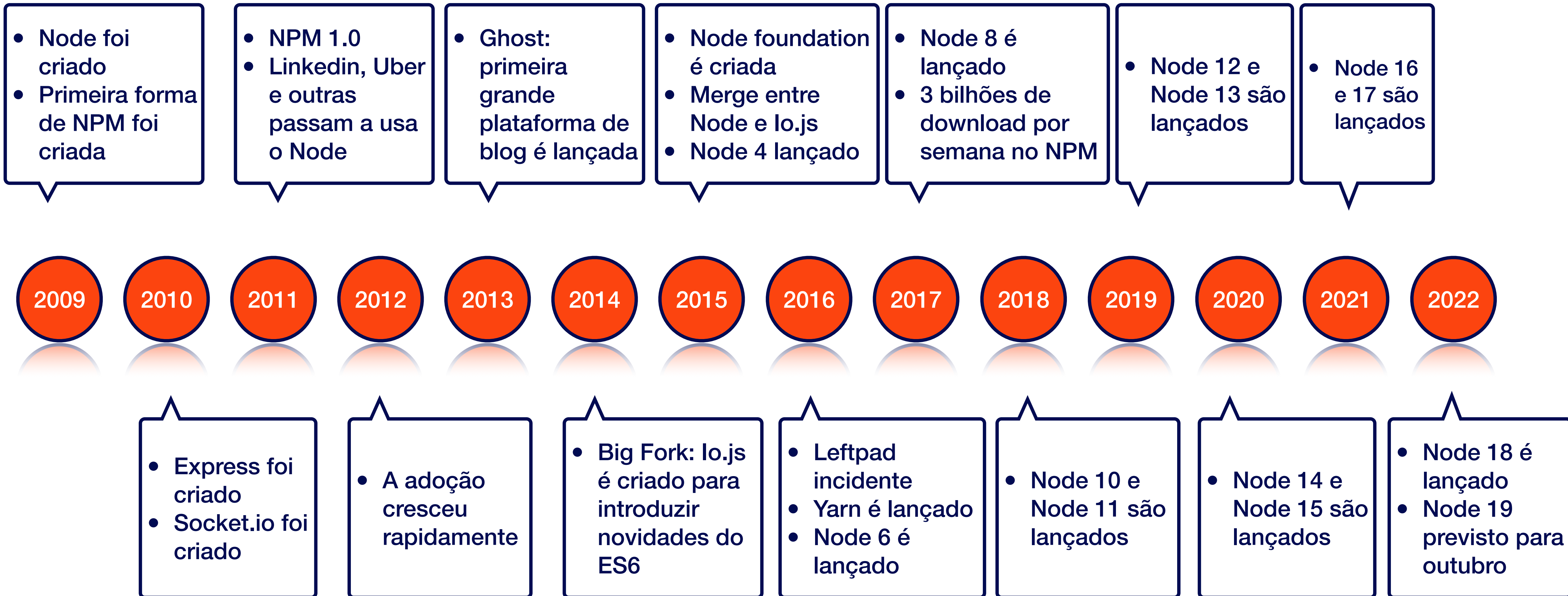
História - v8

- Engine de alto desempenho JavaScript e WebAssembly
- Escrita em C++
- Usada no Chrome e no Node entre outros projetos
- Compila e executa código JS, gerencia a alocação de memória e realiza a desalocação de objetos não necessário (*garbage collector*)



Introdução

História - Timeline



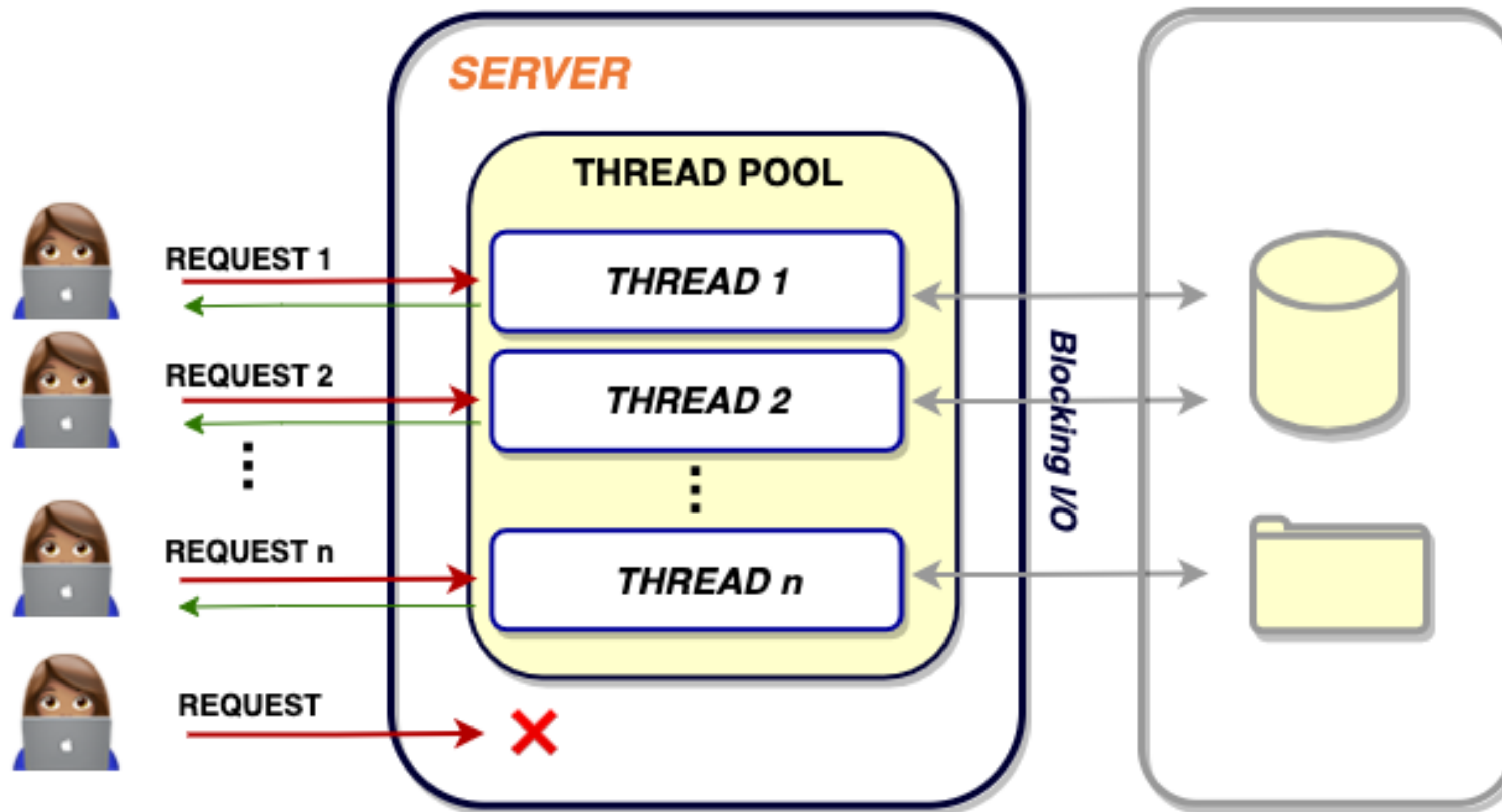
Introdução

Características

- Node.js app são executadas em um único processo
 - Não é necessária a criação de uma thread para cada requisição
- Fornece um conjunto de operações primitivas de I/O assíncronas
 - Evita que códigos de maneira geral sejam “bloqueantes”
- Escalável e mais simples de debugar, não há concorrência entre threads
- Novidades do ESMA Script podem ser usadas sem problemas já que o usuário possui o controle do ambiente de execução
 - No front-end dependemos dos navegadores

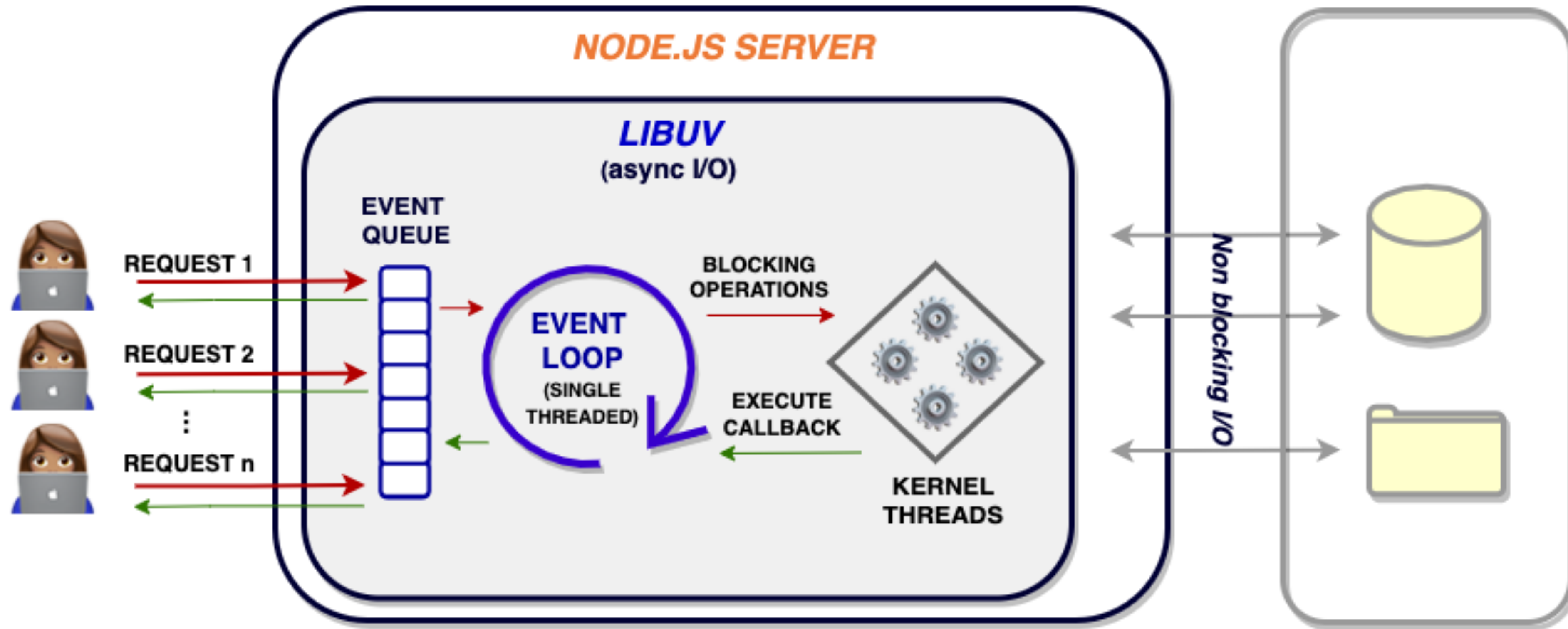
Introdução

Características



Introdução

Características



Introdução

Vantagens

- Excelente desempenho e escalável
- Escrito em JS, familiar para desenvolvedores Web
- Grande comunidade de usuários e desenvolvedores
- O gerenciador de pacote do Node, NPM, prover acesso a diversas bibliotecas reusáveis
 - Gerenciamento de dependências
- Portável, disponível para Windows, macOS, Linux, Solaris, FreeBSD, OpenBSD, WebOS, and NonStop OS

Criando um projeto Node



Criando um projeto Node

Hello World

- O seu primeiro programa em Node

```
console.log("Olá mundo");
```

```
node app.js
```

Criando um projeto Node

Módulos nativos

- [assert](#)
- [buffer](#)
- [child_process](#)
- [console](#)
- [cluster](#)
- [crypto](#)
- [dgram](#)
- [dns](#)
- [events](#)
- [fs](#)
- [http](#)
- [http2](#)
- [https](#)
- [net](#)
- [os](#)
- [path](#)
- [perf_hooks](#)
- [process](#)
- [querystring](#)
- [readline](#)
- [repl](#)
- [stream](#)
- [string_decoder](#)
- [timers](#)
- [tls](#)
- [tty](#)
- [url](#)
- [util](#)
- [v8](#)
- [vm](#)
- [wasi](#)
- [worker](#)
- [zlib](#)

Criando um projeto Node

Módulos nativos

Nome	Descrição
<u>console</u>	Prover um console para debug
<u>events</u>	Prover uma API para o gerenciamento de eventos
<u>fs</u>	Prover uma API para interagir com o sistema de arquivos
<u>http</u>	Prover uma implementação HTTP cliente/servidor
<u>os</u>	Prover propriedades e métodos utilitários relacionados ao sistema operacional
<u>path</u>	Prover utilitários para trabalhar com path e diretórios
<u>querystring</u>	Prover utilitários para " <i>parsear</i> " e formatar URL de string de consulta (querystring)

Criando um projeto Node

Módulos nativos

Nome	Descrição
<u>repl</u>	Prover um implementação Read-Eval-Print-Loop (REPL) disponível como um versão standalone, mas que também pode ser adicionada a outras aplicações
<u>timers</u>	Prover funções para agendar execuções de funções em um período futuro
<u>url</u>	Prover utilitários para resolução e “parseamento” de URL

Criando um projeto Node

NPM

- Node Package Manager - Gerenciador de pacotes do Node
- Inicialmente era uma maneira de fazer download e gerenciar as dependências
- Atualmente é também utilizado em projetos front-end
- Possui mais de 1.3 milhões de pacotes disponíveis
 - Maior repositório de software do mundo



Criando um projeto Node

- Para iniciar um projeto **node**, é necessário criar um arquivo chamado **package.json**
 - Lista todas as dependências do projeto e suas versões
 - Torna o processo de build reproduzível e portanto mais fácil de compartilhar com outros desenvolvedores
 - Deve conter pelo menos o atributo **name** e **version**
- A maneira mais simples de criar esse arquivo é usando o comando:
 - `$ npm init --yes`

Criando um projeto Node

```
{
  "name": "my package",
  "description": "",
  "version": "1.0.0",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/monatheoctocat/my_package.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/monatheoctocat/my_package/issues"
  },
  "homepage": "https://github.com/monatheoctocat/my_package"
}
```

Nome do diretório

Informação contida no README ou string vazia

Sempre 1.0.0

Script de test vazio

No caso de se um repositório git

Sempre vazio

Sempre vazio

Por padrão ISC

Caso hospedado no GitHub

Caso hospedado no GitHub

Criando um projeto Node

NPM e suas funções

- Instalar e atualizar dependências
 - `$ npm install` ou `$ npm install <package-name>`
 - `$ npm update` ou `$ npm update <package-name>`
- Versionamento
- Execução de tarefas
 - Ex: Executar em produção, testar ...

Criando um projeto Node

Pacotes populares



Express



Socket.io



Fastify



Rxjs



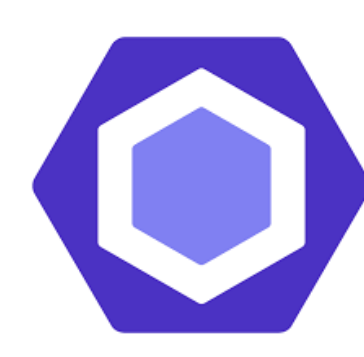
Lodash



Ramda



Jest



ESLint



Nodemon



Dotenv



Yargs



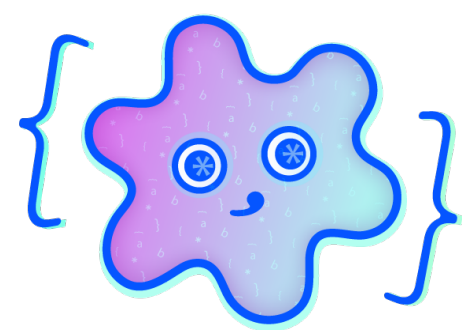
Passport



Nodemailer



Mongoose

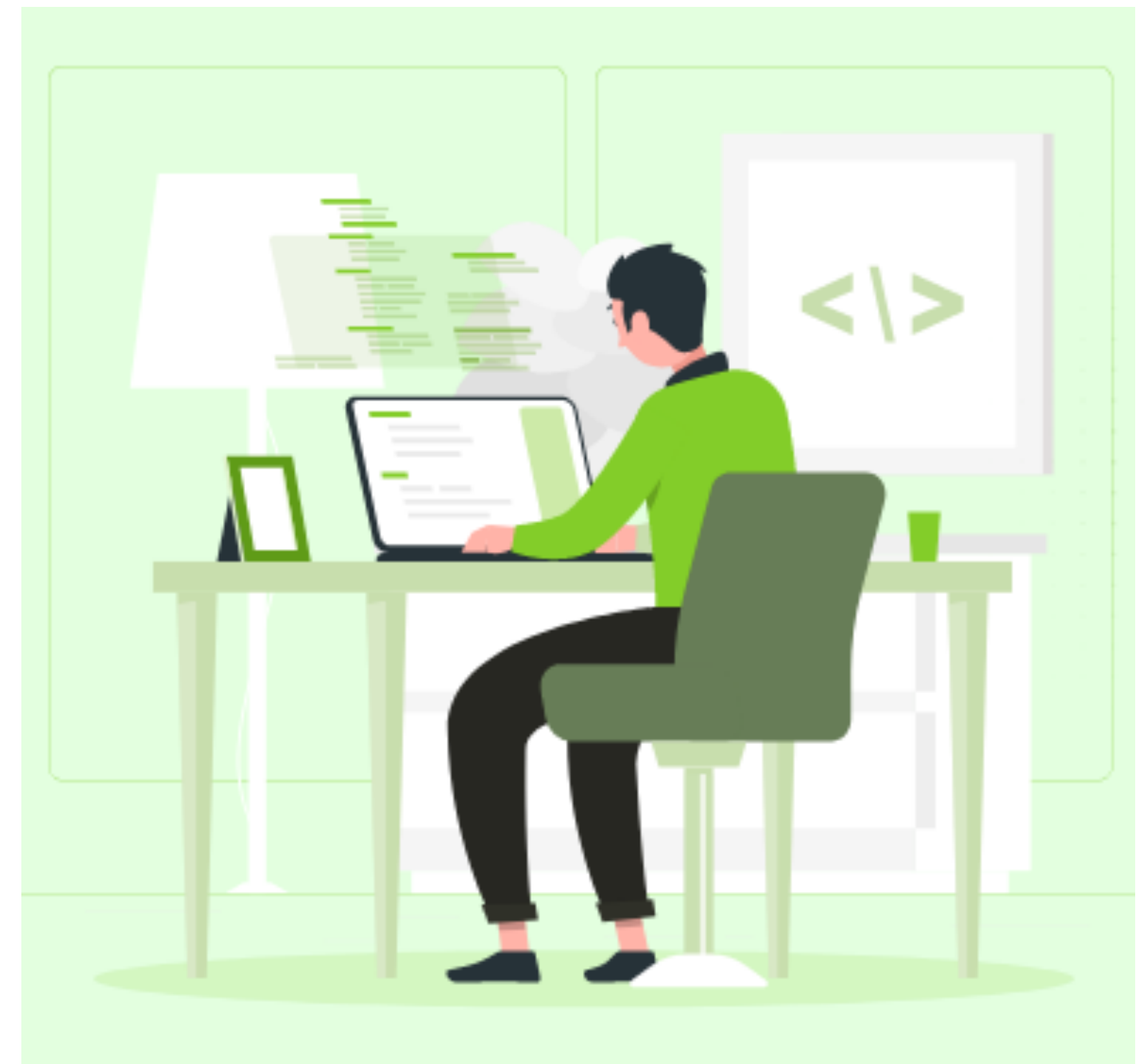


glob

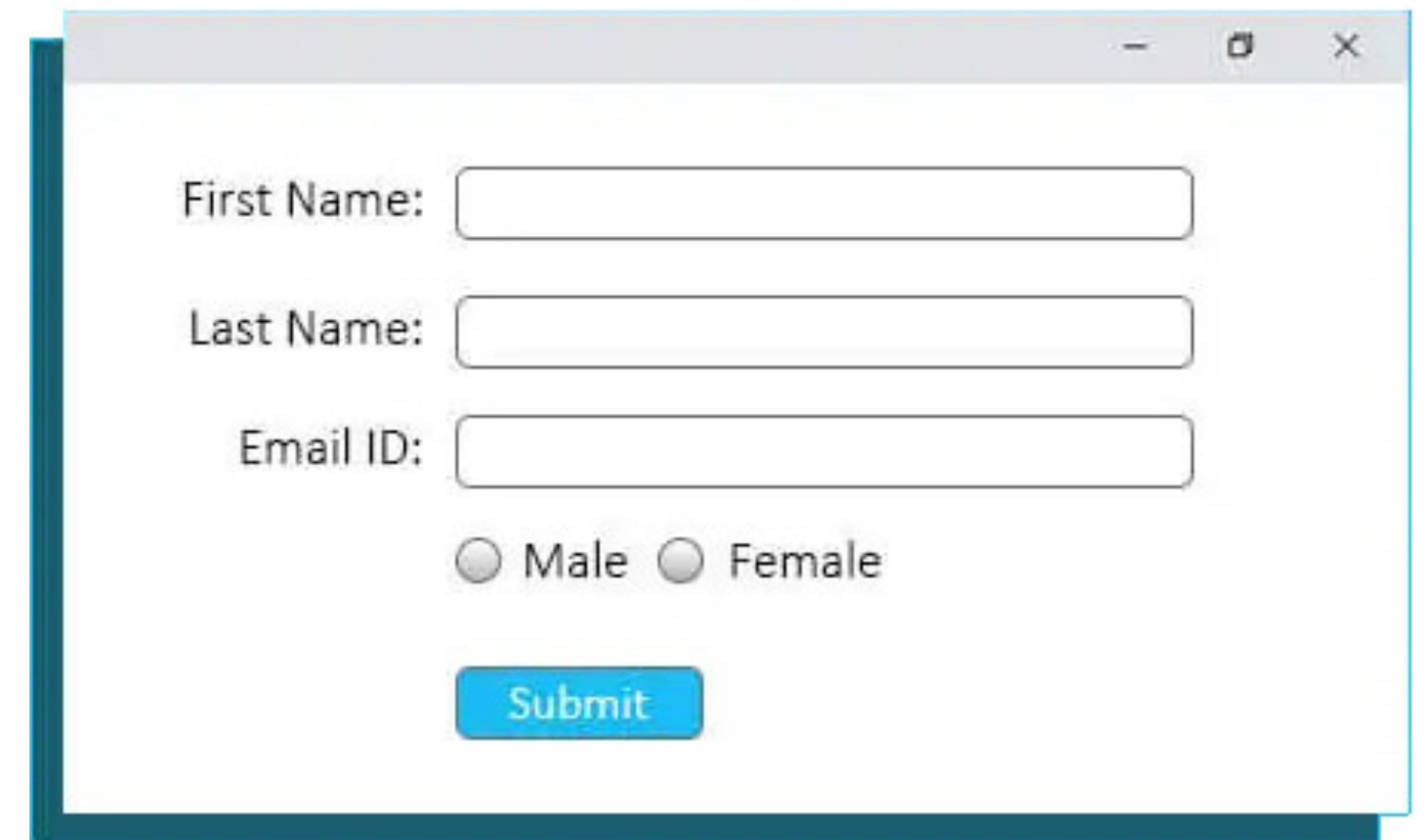


[Lista completa contendo 40 pacotes](#)

Criando um servidor web com Node



Formulários HTML



First Name:

Last Name:

Email ID:

Male Female

Formulário HTML

- É um grupo de componentes de interface do usuário que aceita informações do usuário e envia as informações do usuário para um servidor web
- JavaScript pode ser usado para criar componentes interativos e para realizar validações

Formulário HTML

<form>

- É necessária o atributo `action` com a URL da página que irá processar os dados deste formulário
- Quando o formulário for preenchido e submetido, seus dados serão enviados para URL da ação

```
<form action="URL de destino">  
  <!-- componentes do formulario -->  
</form>
```

Formulário HTML

Exemplo

```
<form action="http://www.google.com/search">  
  <label>  
    Let's search Google:  
    <input name="q" />  
    <input type="submit" />  
  </label>  
</form>
```

Formulário HTML

Atributos

Atributo	Descrição
<code>accept-charset</code>	Determina a codificação de caracteres usada na submissão
<code>action</code>	Determina o local para o qual os dados serão submetidos
<code>autocomplete</code>	Determina se o formulário deve ser ou não autocompletado
<code>enctype</code>	Determina com os dados vão ser codificados (apenas no método POST)
<code>method</code>	Determina o método HTTP que será utilizado no envio dos dados
<code>name</code>	Determina o nome do formulário
<code>novalidate</code>	Determina se o formulário deve ou não ser validado antes do envio

Mais informações em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/form>

Formulário HTML

<input>

- É usado para criar vários tipos de campos para interação com o usuário
- O atributo **name** especifica o nome do parâmetro a ser enviado ao servidor
- O atributo **value** especifica o valor inicial do campo
- O tipo do campo pode ser determinado através do atributo **type**
- Compartilham uma série de atributos HTML pois todos são instância de **HTMLInputElement**

Formulário HTML

Tipos de input

Tipo	Descrição	
text	Um campo de texto com uma só linha; quebras de linha são automaticamente removidas	
password	Um campo de texto com uma só linha cujo valor é obscurecido.	
checkbox	Uma caixa de marcação. Várias opções podem ser marcadas.	
radio	Um botão de escolha. Apenas uma opção pode ser marcada.	
submit	Um botão que envia o formulário.	
button	Um botão sem comportamento padrão.	
file	Um controle que permite ao usuário selecionar um arquivo.	
number	Um controle para inserir um número de ponto flutuante.	HTML5
email	Um campo para editar um endereço de e-mail.	HTML5

Formulário HTML

Tipos de input

Tipo	Descrição	
<code>color</code>	Um controle para especificar cores.	
<code>date</code>	Um controle para inserir uma data (ano, mês e dia, sem horário).	
<code>datetime-local</code>	Um controle para inserir data e horário, sem fuso horário.	HTML5
<code>hidden</code>	Um controle que não é exibido mas cujo valor é enviado ao servidor.	
<code>image</code>	Um botão gráfico para enviar o formulário.	
<code>month</code>	Um controle para inserir mês e ano, sem fuso horário.	HTML5
<code>range</code>	Um controle para inserir um número limitado por um intervalo.	HTML5
<code>reset</code>	Um botão que faz o conteúdo do formulário voltar a ter seus valores padrão.	

Formulário HTML

Tipos de input

Tipo	Descrição	
<code>search</code>	Um campo de texto com uma só linha para digitar termos de busca	HTML5
<code>tel</code>	Um controle para inserir um número de telefone	HTML5
<code>time</code>	Um controle para inserir um horário sem fuso horário.	HTML5
<code>url</code>	Um campo para editar uma URL.	HTML5
<code>week</code>	Um controle para inserir uma data consistindo de ano da semana e número da semana sem fuso horário.	HTML5

Formulário HTML

text e password

```
<input type="text" size="10" maxlength="8" required="true">  
<input type="password" size="10" minlength="4" required="true">
```

- A única diferença entre **text** e **password** é que ao escolher password ao digitar os caracteres não aparecem

Formulário HTML

text e password

Atributo	Descrição	Aplicável
<code>disabled</code>	Indica se o campo está desabilitado	*
<code>readonly</code>	Indica se o campo pode ser alterado	*
<code>minlength</code>	Determina a qualidade mínima de caracteres aceitos	Campos com texto
<code>min</code>	Determina o valor mínimo	number, range, date, datetime-local, month, time e week
<code>maxlength</code>	Determina a qualidade máxima de caracteres aceitos	Campos com texto
<code>max</code>	Determina o valor máximo	number, range, date, datetime-local, month, time e week

Mais informações em: <https://html.spec.whatwg.org/multipage/input.html#the-input-element>

Formulário HTML

text e password

Atributo	Descrição	Aplicável
<code>pattern</code>	Determina o padrão a ser correspondido	text, date, search, url, tel, email, e password.
<code>placeholder</code>	Determina o texto contido no campo quando nenhum valor é informado	text, search, url, tel, email, e password.
<code>required</code>	Determina se um campo é obrigatório ou não	text, search, url, tel, email, password, date pickers, number, checkbox, radio, e file.
<code>size</code>	Determina a largura do formulário	text, search, tel, url, email, e password.
<code>step</code>	Determina o tamanho do incremento em input de intervalo	number, range, date, datetime-local, month, time e week
<code>value</code>	Valor inicial do campo	*

Mais informações em: <https://html.spec.whatwg.org/multipage/input.html#the-input-element>

Formulário HTML

<textarea>

- Define um campo de texto que permite múltiplas linhas
 - A quantidade de texto pode ou não ser limitada
- É uma boa prática usar os atributos **cols** e **rows** para determinar o tamanho do campo e garantir a consistência entre navegadores
- O texto inicial deve ser colocado dentro da tag

```
<textarea rows="4" cols="20">  
  Type your comments here.  
</textarea>
```

Formulário HTML

Checkbox

- Fornece um formulário de múltiplas escolhas para os usuários
- O atributo **checked** determina se uma opção está selecionada
- Todas as caixas marcadas serão enviado com valor **on** ao servidor
 - Se nenhum opção for marcada, a variável que representa o campo não será enviada na requisição
 - Quando especificado, o atributo value, o seu conteúdo é enviado ao servidor se a opção for selecionada

```
<input type="checkbox" name="lettuce" > Lettuce  
<input type="checkbox" name="tomato" checked> Tomato  
<input type="checkbox" name="pickles" checked> Pickles
```

Formulário HTML

Checkbox

```
<input name="opcao[]" type="checkbox" value="1"/> Escolha 1  
<input name="opcao[]" type="checkbox" value="2"/> Escolha 2  
<input name="opcao[]" type="checkbox" value="3" checked/> Escolha 3
```

- Escolha 1
- Escolha 2
- Escolha 3

Formulário HTML

Radio Button

- São agrupados através do atributo **name**
 - Apenas um pode ser selecionado
- Através do atributo **checked** é possível determinar se um radio está selecionado
- Deve-se especificar o atributo **value** para cada opção, caso contrário o valor **on** será enviado

```
<input type="radio" name="cc" value="visa" checked> Visa  
<input type="radio" name="cc" value="mastercard"> MasterCard  
<input type="radio" name="cc" value="amex"> American Express
```

Formulário HTML

`<label>`

- Representa uma legenda para um item em uma interface de usuário
- Pode estar associado a um elemento de controle
- O elemento label pode ser alvo de regras de estilo CSS
 - Particularmente útil se usado com checkbox ou radiobutton
 - Aumenta a área de clique o elemento alvo
- Importante para leitores de tela

Formulário HTML

<label>

```
<label>  
  <input type="radio" name="cc" value="visa" checked> Visa  
</label>  
<label>  
  <input type="radio" name="cc" value="mastercard"> MasterCard  
</label>  
<label>  
  <input type="radio" name="cc" value="amex"> American Express\  
</label>
```

```
<label for="username">Enter your username:</label>  
<input id="username">  
<label for="username">Forgot your username?</label>
```


Formulário HTML

<select>

- É utilizada para definir uma lista de seleção
- `<option>` dentro do elemento `<select>` define uma das opções de escolha
- `<optgroup>` é utilizada para agrupar opções
- O atributo `selected` determina que a opção selecionada

```
<select name="favoritecharacter">
  <option>Jerry</option>
  <option>George</option>
  <option selected>Kramer</option>
  <option>Elaine</option>
</select>
```

Formulário HTML

<select>

- O atributo multiple permite seleção múltipla

```
<select name="favoritecharacter" size="3" multiple>  
  <option>Jerry</option>  
  <option>George</option>  
  <option>Kramer</option>  
  <option>Elaine</option>  
  <option selected>Newman</option>  
</select>
```

Formulário HTML

Reset

- É representado por um botão
- Quando apertado, restaura todos os elementos de entrada do formulário aos seus valores defaults

```
Name: <input type="text" name="name"> <br>  
Food: <input type="text" name="meal" value="pizza"> <br>  
<label>Meat? <input type="checkbox" name="meat"></label> <br>  
<input type="reset" />
```

Formulário HTML

Hidden

- Define uma campo invisível que não será renderizado pelo navegador
 - Continua visível no código-fonte
- Geralmente é utilizado para enviar valores já conhecidos
- Também é utilizado para evitar spam em formulários
- Útil para passar uma informação adicional que não deve ser modificada pelo usuário

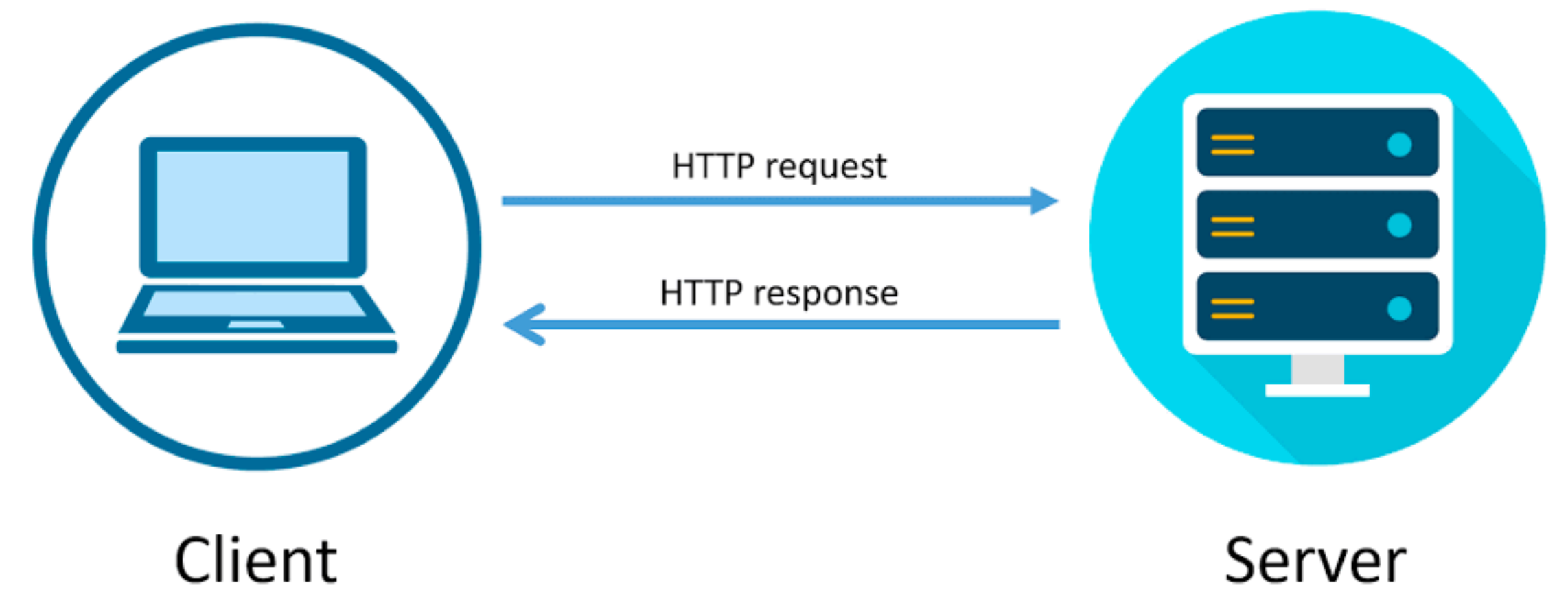
Formulário HTML

Submit

- Representado por um botão
- Quando clicado envia as informações para o servidor
 - A URL do servidor é especificada na tag form

```
<input type="submit" value="Envia Mensagem"/>
```

Enviando parâmetros na requisição



Enviando parâmetros na requisição

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Minha primeira página</title>
  </head>
  <body>
    <form action="oi" method="get">
      <input type="submit" value="Gravar" />
    </form>
  </body>
</html>
```

Método utilizado para o envio dos dados

Para onde (URL) os dados do formulário serão enviados

Enviando parâmetros na requisição

Parâmetros

- O que são?
 - São dados submetidos ao servidor por uma requisição **HTTP**
 - **GET**
 - **POST**
- Qual o tipo do dado retornado?
 - São sempre String

Enviando parâmetros na requisição

GET vs POST

- **GET**

- Solicita ao servidor uma página de dados
- Se o pedido tem parâmetros, eles são enviados na URL como uma string de consulta

- **POST**

- Envia dados para o servidor e recupera a resposta do servidor

Enviando parâmetros na requisição

O método GET

- Os dados do formulário preenchido é visível na URL
 - São restritos aos códigos ASCII
 - Não suporta dados binários, imagens e outros arquivos
 - Cuidados especiais devem ser tomados para codificar e decodificar outros tipos de caracteres
 - Há uma limitação na quantidade de dados que pode ser enviado via URL
- Também é armazenado no usuário histórico de navegação web / logs

Enviando parâmetros na requisição

```
<html>
  <head>
    <title>Servlet</title>
  </head>
  <body>
    <form action="oi" method="get">
      <label> Nome: <input type="text" name="nomePessoa" /></label>
      <label><input type="submit" value="Gravar" /></label>
    </form>
  </body>
</html>
```

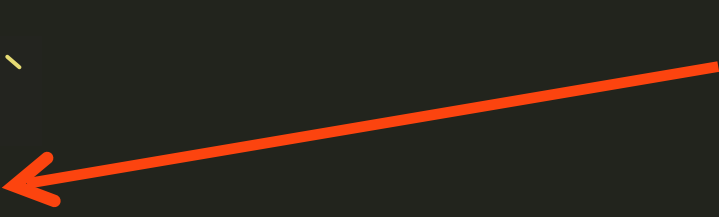

Nome do parâmetro



Enviando parâmetros na requisição

Obtendo parâmetros de uma requisição GET

- Precisamos utilizar acessar o objeto a URL dentro do objeto que encapsula a requisição HTTP

```
const server = http.createServer(req: IncomingMessage, res: ServerResponse) => {  
  
  const baseUrl = `http://${req.headers.host}/`  
  const parsedUrl = new URL(req.url, baseUrl)  Parseando a URL  
  
  res.statusCode = 200  
  res.setHeader('Content-Type', 'text/html; charset=utf-8')  
  
  const name = parsedUrl.searchParams.get("name")  Nome do parâmetro  
  res.end(`<html><head></head><body> Contéudo </body></html>`)  
});
```

Enviando parâmetros na requisição

Obtendo parâmetros de uma requisição POST

- Precisamos utilizar acessar o objeto a URL dentro do objeto que encapsula a requisição HTTP

```
const server = http.createServer(async (req: IncomingMessage, res: ServerResponse) => {  
  
  const baseUrl = `http://${req.headers.host}/`  
  const parsedUrl = new URL(req.url, baseUrl)  
  
  const chunks = [];  
  for await (const chunk of req) {  
    chunks.push(chunk);  
  }  
  const data = Buffer.concat(chunks);  
  res.end(data.toString())  
});
```

Navegando na
stream de dados



Enviando parâmetros na requisição

GET vs POST

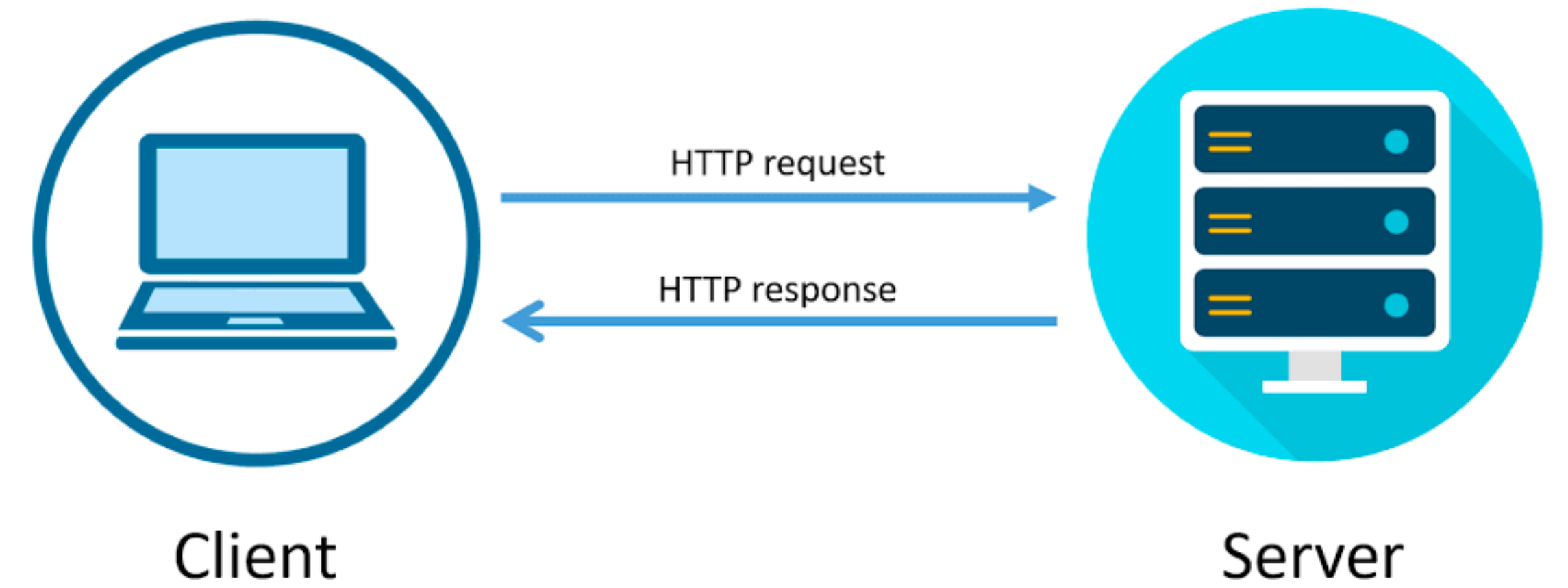
	GET	POST
Restrições de tipo	Apenas caracteres ASCII	Não há restrições, até dados binários podem ser enviados
Visibilidade	Os dados permanecem visíveis	Os dados não ficam visíveis
Limite de tamanho	2000 caracteres	Nenhuma limitação
Botão de voltar	Requisições GET são re-executadas	O navegador avisa que é necessário reenviar dados
Favoritar	Sim	Não
Hackear	Sim	Não*
Cached	Sim	Não

Enviando parâmetros na requisição

GET vs POST - Recomendações gerais

- GET é recomendado para formulários que envolvem consultas ao banco de dados
- Se os dados contém caracteres não ASCII deve-se usar POST
- Se a quantidade de dados do formulário for muito grande, deve-se usar POST
- Para envio de dados confidenciais, deve-se usar POST

Prática



Enviando parâmetros na requisição

Prática

- Web Server simples imprimindo HTML
- Envio de formulário GET & POST -> Server imprimindo as informações
 - Validação do form via Js
- Roteamento
 - Mostrar Form, Tratar Form, Validação / Redirecionamento
 - Teste de validação via Postman
 - Enviar dados via URL
- Retorna JSON
- API REST

Referências

- <https://nodejs.dev/learn>
- <https://leanylabs.com/blog/npm-packages-for-nodejs/>
- <https://flaviocopes.com/node-core-modules/>
- <https://docs.npmjs.com/>
- <https://scoutapm.com/blog/nodejs-architecture-and-12-best-practices-for-nodejs-development>
- <https://leanylabs.com/blog/npm-packages-for-nodejs/>
- <https://ctrly.blog/nodejs-layered-architecture/>

Por hoje é só