

PROGRAMAÇÃO \_

FRONT-END \_

DATA SCIENCE \_

INTELIGÊNCIA ARTIFICIAL \_

DEVOPS \_

UX &amp; DESIGN \_

MOBILE \_

INOVAÇÃO &amp; GESTÃO \_

Artigos > **Programação**

# Como criar um servidor HTTP com NodeJS

**marco.bruno**

07/03/2017

COMPARTILHE



Nesse primeiro post criaremos um **servidor HTTP** conhecido também por **servidor web** em [JavaScript](#) puro que utilizará a API do [NodeJS](#) chamada



alura

MATRICULE-SE

http. Teremos um pequeno projeto para deixar o post mais feliz e um pouco mais divertido de como abordo esse assunto no [curso de NodeJS](#) [preço que todo Caelum](#), a única diferença desse post para o curso é que durante as aulas utilizamos um framework chamado ExpressJS para criar o servidor HTTP e responder as rotas.

## Qual será o nosso projeto?

Vamos criar um servidor HTTP para responder a 4 rotas de um site de um evento focado em desenvolvedores FrontEnd, com uma estrutura de links parecida com o site do [FrontInSampa](#) que vai acontecer agora no dia 1 de Julho 2017. Essa são as 4 rotas que vamos criar:

1. <http://localhost:3000/> (GET)
2. <http://localhost:3000/inscreva-se> (GET)
3. <http://localhost:3000/local> (GET)
4. <http://localhost:3000/contato> (GET)

Agora que já sabemos qual é o nosso projeto, precisamos trabalhar para preparar o nosso ambiente de desenvolvimento com o NodeJS de plataforma.

Para instalar o NodeJS é tudo bem simples, basta você entrar nesse link: <https://nodejs.org/en/download>; em seguida fazer o download para o seu respectivo sistema operacional e seguir o processo de instalação. Se tiver qualquer dúvida durante a instalação, só fazer um comentário ou me adicionar em qualquer rede social da vida que eu te ajudo. Você vai conseguir me achar nas redes sociais por MarcoBrunoBR.

**Matricule-se na escola de PROGRAMAÇÃO**

Junte-se a uma comunidade de **+500 mil** estudantes

→ Acesso a **TODOS** os cursos em uma única assinatura

→ Novos lançamentos a cada semana

→ Desafios práticos

**SAIBA MAIS**



Show! Agora podemos começar a criar o nosso **servidor HTTP** utilizando o NodeJS como plataforma. Começaremos criando um arquivo chamado *server.js* dentro de uma pasta com o nome de *app*, por favor crie essa pasta no seu Desktop (Área de trabalho).

Inicialmente nosso arquivo **server.js** ficará assim:

```
const http = require('http') const port = 3000 const ip = 'localhost'

const server = http.createServer((req, res) => { console.log('Recebendo uma request!') res.end('
```

## Aqui fica o que vamos enviar para o navegador como resposta!

```
`) })
```

```
server.listen(port, ip, () => { console.log(`Servidor rodando em http://\${ip}:\${port}`) console.log('Para derrubar o servidor: ctrl + c'); })
```

Link desse código no GitHub: <https://github.com/MarcoBrunoBR/server-http-with-nodejs/tree/24700745eea7f3722b9335e58916f8d7682d103d>.

Show! Agora temos um arquivo JS com os comandos necessários para criar o nosso servidor HTTP, no nosso próximo passo precisamos fazer com que a nossa plataforma NodeJS execute nosso código (*server.js*). Para conseguirmos fazer isso, abra o seu terminal e dentro dele execute os seguintes comandos:

```
~$ cd ~/Desktop/app ~/Desktop/app$ node server.js
```

Na primeira linha, nós estamos acessando a pasta do nosso projeto com o comando **cd** seguido do caminho de onde criamos a pasta **app**. O **~** é um atalho para ir até a pasta do usuário que você está logado no seu sistema operacional. Já na segunda linha nós estamos falando para o NodeJS executar o nosso arquivo **server.js**, após ter executado esse comando você deverá receber a seguinte saída no seu terminal:

**alura****MATRICULE-SE**

Servidor rodando em <http://localhost:3000> Para derrubar o servidor: ctrl + c

Pronto! Agora nós temos o nosso servidor HTTP de pé e já podemos testar.  
Para ver se tudo está certo, abra o seu navegador e acesse o

<http://localhost:3000>, se tudo estiver certo você verá no seu navegador exatamente o texto que passamos para o seguinte linha do nosso **server.js**:

```
res.end('
```

## Aqui fica o que vamos enviar para o navegador como resposta!

```
')
```

Nosso servidor está funcionando mas está com a mesma resposta para qualquer path(caminho), para testar isso, vá até o seu navegador e entre em <http://localhost:3000/contato>, a resposta será a mesma. Então vamos resolver esse problema criando uma resposta única para cada uma das [4 rotas que comentamos no começo do post](#).

## Respondendo diferente para cada rota

Toda vez que o nosso servidor HTTP recebe uma requisição a função de callback que passamos para o método `createServer` é executada, por isso vamos colocar as condições para respostas diferentes conforme o endereço da requisição. Vamos começar criando uma resposta para a nossa home (<http://localhost:3000/>):

### server.js

```
const http = require('http') const port = 3000 const ip = 'localhost'
```

```
const server = http.createServer((req, res) => { if (req.url === '/') { res.end('
```



```
res.end('
```

## URL sem resposta definida!

```
) }
```

```
server.listen(port, ip, () => { console.log(`Servidor rodando em  
http://\${ip}:\${port}`) console.log('Para derrubar o servidor: ctrl + c'); })
```

Link do código no GitHub: <https://github.com/MarcoBrunoBR/server-http-with-nodejs/tree/70636dfd82f9329c9b1c3128d3a48be3226a829d>

Após você alterar o código é necessário reiniciar o servidor. Para fazermos isso você precisa ir até o terminal e teclar **ctrl + c**, dessa forma derrubamos o servidor que estava no ar. Agora precisamos subi-lo novamente com o nosso novo código, para fazer isso digite no terminal:

## Terminal, dentro da pasta app

```
~/Desktop/app$ node server.js
```

Obs. Lembre-se que você precisa estar na pasta do projeto que chamamos de **app** e criamos no nosso **Desktop**.

Entre novamente no browser e veja o que acontece quando você entrar na URL <http://localhost:3000>? Se tudo deu certo até aqui você verá no seu navegador a palavra Home. :-)

O que acontece se você tenta entrar na URL <http://localhost:3000/inscreve-se>? Acabou aparecendo o texto **URL sem resposta definida!** dentro de uma tag h1. Tem algo interessante acontecendo no nosso código, vamos dar uma olhada nessa parte:

```
const server = http.createServer((req, res) => { if (req.url === '/') { res.end('
```

## Home

```
) }
```

# URL sem resposta definida!

```
')} )}
```

Já vimos que toda vez que o nosso servidor recebe uma requisição HTTP ele executa o callback que passamos para o `createServer`, outra coisa que é bem interessante é que quando o NodeJS cai em um **res.end** tudo que está a seguir dele continua a ser executado e a resposta já foi enviada, isso não é bom e nem ruim é apenas algo que precisamos lembrar quando estamos desenvolvendo.

Agora que já sabemos fazer respostas diferentes para as nossas requisições, podemos implementar para uma resposta para 3 rotas restantes:

## server.js

```
const http = require('http') const port = 3000 const ip = 'localhost'

const server = http.createServer((req, res) => { if (req.url === '/') { res.end('
```

## Home

```
')} )}
```

```
if (req.url === '/inscreva-se') { res.end('
```

## Inscreva-se

```
')} )}
```

```
if (req.url === '/local') { res.end('
```

## Local

```
')} )}
```

```
if (req.url === '/contato') { res.end('
```

```
) }
```

```
res.end('
```

## URL sem resposta definida!

```
) })
```

```
server.listen(port, ip, () => { console.log(`Servidor rodando em  
http://\${ip}:\${port}`) console.log('Para derrubar o servidor: ctrl + c'); })
```

Link do código no GitHub: <https://github.com/MarcoBrunoBR/server-http-with-nodejs/tree/7c651957b589071efbe8cb942c26557996b49bf6>

É isso, conseguimos criar uma resposta para as 4 rotas que nos propomos no começo do post. Apesar do código estar bem legível, eu não faria com **if**, nesse caso eu utilizaria um Array:

## server.js

```
const http = require('http') const port = 3000 const ip = 'localhost'
```

```
const server = http.createServer((req, res) => { const responses = []  
responses['/'] = '<h1>Home</h1>' responses['inscreva-se'] = '
```

## Inscreva-se

```
' responses  
'/local' = '<h1>Local</h1>' responses  
'/contato' = '
```

## Contato

```
' responses``'/naoExiste' = '
```

## URL sem resposta definida!

```
,
```





MATRICULE-SE

```
res.end(responses req.url || responses '/naoExiste' ) })  
server.listen(port, ip, () => { console.log(`Servidor rodando em  
http://${ip}:${port}`) console.log('Para derrubar o servidor: ctrl + c'); })
```

Link do código no GitHub: <https://github.com/MarcoBrunoBR/server-http-with-nodejs/tree/48158fadfe2a9993f38f04e04fc8c75461e5e30a>

Nosso código ficou com uma quantidade menor de linhas mas com uma legibilidade questionável. :-) O que você faria diferente na hora de criar o seu servidor HTTP?

Tanto o código com if ou array, são códigos complicados de se manter. Por esse motivo e outros que a comunidade de JavaScript começou a criar frameworks para cuidar das rotas. Nos próximos posts vamos ver como criar rotas com os frameworks:

1. [Restify](#)
2. [ExpressJS](#)
3. [HapiJS](#)
4. [KoaJS](#)

Além do curso presencial da Caelum, temos cursos online de [Node JS](#) na Alura.

### Confira neste artigo:

- [Qual será o nosso projeto?](#)
- [Criando um Servidor HTTP em NodeJS](#)
- [Respondendo diferente para cada rota](#)

← [Artigo Anterior](#)

[Próximo Artigo](#) →

**[Portfólio na área de tecnologia: um guia para construir um ideal!](#)**

**[A arquitetura do novo site da Alura](#)**



**alura**Veja outros artigos sobre  
[Programação](#)**MATRICULE-SE**

## Quer mergulhar em tecnologia e aprendizagem?

Receba a newsletter que o nosso CEO escreve pessoalmente, com insights do mercado de trabalho, ciência e desenvolvimento de software

**ME INSCREVA****alura**

### Nossas redes e apps



#### Institucional

[Sobre nós](#)[Trabalhe conosco](#)[Para Empresas](#)[Para Escolas](#)

#### A Alura

[Formações](#)[Como Funciona](#)[Todos os cursos](#)[Depoimentos](#)

[Política de Privacidade](#)[Instrutores\(as\)](#)**alura**[Compromisso de Integridade](#)[Dev em <T>](#)[MATRICULE-SE](#)[Termos de Uso](#)[Luri by ChatGPT](#)[Status](#)

## Conteúdos

[Alura Cases](#)[Imersões](#)[Artigos](#)[Podcasts](#)[Artigos de educação corporativa](#)

## Fale Conosco

[Email e telefone](#)[Perguntas frequentes](#)

## Novidades e Lançamentos

[ENVIAR](#)

## CURSOS

### Cursos de Programação

[Lógica](#) | [Python](#) | [PHP](#) | [Java](#) | [.NET](#) | [Node JS](#) | [C](#) | [Computação](#) | [Jogos](#) | [IoT](#)

### Cursos de Front-end

[HTML, CSS](#) | [React](#) | [Angular](#) | [JavaScript](#) | [jQuery](#)

### Cursos de Data Science

[Ciência de dados](#) | [BI](#) | [SQL e Banco de Dados](#) | [Excel](#) | [Machine Learning](#) | [NoSQL](#) | [Estatística](#)

**Cursos de Inteligência Artificial**

IA para Programação | IA para Dados

**Cursos de DevOps**

AWS | Azure | Docker | Segurança | IaC | Linux

**MATRICULE-SE****Cursos de UX & Design**

Usabilidade e UX | Vídeo e Motion | 3D

**Cursos de Mobile**

React Native | Flutter | iOS e Swift | Android, Kotlin | Jogos

**Cursos de Inovação & Gestão**

Métodos Ágeis | Softskills | Liderança e Gestão | Startups | Vendas