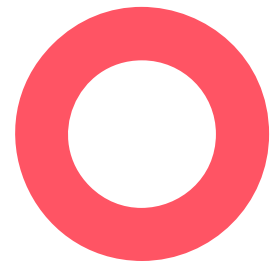


DESENVOLVIMENTO DE SISTEMAS

UC13

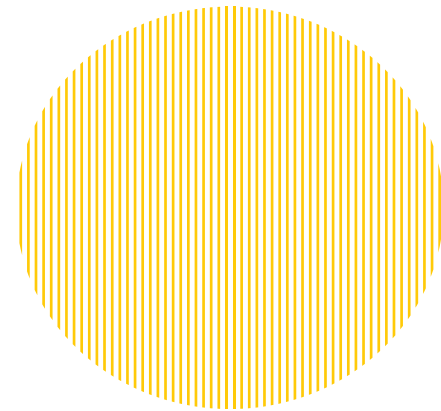
Prof. Viviane de Lima

viviane.lfrancelino@sp.senac.br

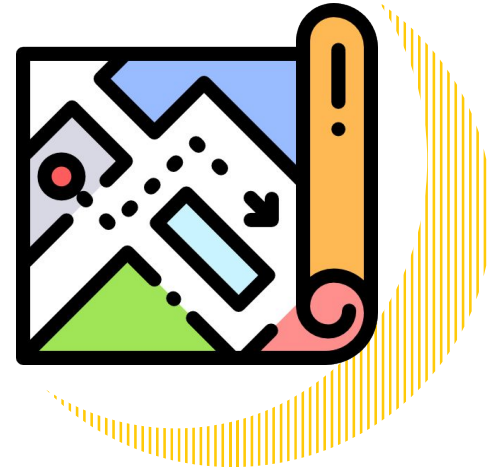


AULA 03

TEORIA DAS ROTAS



O QUE SÃO ROTAS?



O QUE SÃO ROTAS?



g1.globo.com

g1.globo.com/ciencia/nasa/titulo-da-noticia

g1.globo.com/saude/covid/titulo-da-noticia



O QUE SÃO ROTAS?



Vamos supor que você gostaria de ver as notícias do dia e acessa o site do G1:

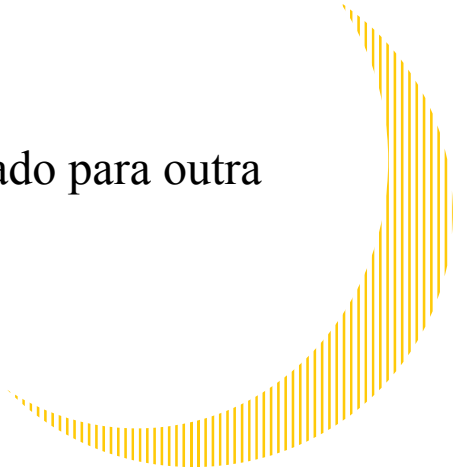
g1.globo.com

Em seguida você viu uma notícia interessante e resolveu clicar nessa notícia:

g1.globo.com/titulo-da-noticia



Observe que quando você clicar nessa notícia, você será redirecionado para outra página, isso é o que chamamos de **rota**



O QUE SÃO ROTAS?

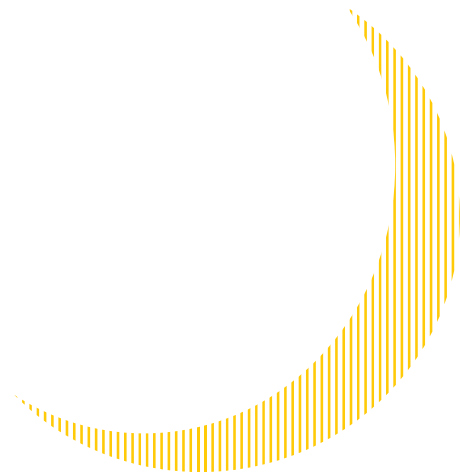


Quando colocamos apenas o endereço do site, estamos indo para a rota /
Ou seja, essa é a rota da home

g1.globo.com/

Quando temos alguma informação após a / , estamos sendo direcionados para outra
página específica

g1.globo.com/titulo-da-noticia



O QUE SÃO ROTAS?



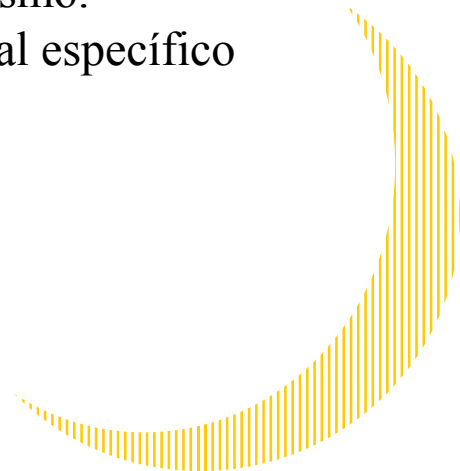
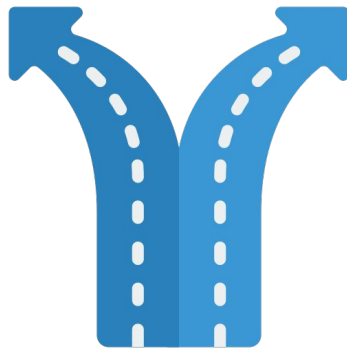
[Sp.senac.br/](https://sp.senac.br/)

[Sp.senac.br /senac-largo-treze](https://sp.senac.br/senac-largo-treze)

[Sp.senac.br /senac-largo-treze/cursos-tecnicos](https://sp.senac.br/senac-largo-treze/cursos-tecnicos)

Cada item desse é uma rota que vai nos mandar para uma página diferente, é como se fosse um caminho, e no servidor o funcionamento é o mesmo.

Tudo o que eu colocar após a / ela vai me direcionar para um local específico



EXISTEM DOIS TIPOS DE ROTAS



Rotas estáticas:

`g1.globo.com/contato`


É apenas um formulário para você entrar em contato com o G1, essa rota não vai sofrer alteração, sempre terá esse nome

Rotas dinâmicas:

`g1.globo.com/noticia/fulano-foi-agredido-no-rj`

`g1.globo.com/noticia/ciclano-foi-eleito-em-sc`

As páginas de notícias são adicionadas dinamicamente no site, a estrutura (tamanho da fonte, espaçamento) é padrão, o que muda é o título da notícia.

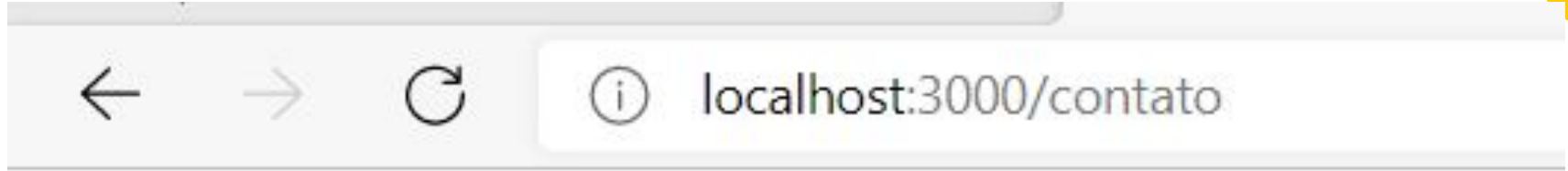


TESTANDO ROTAS NA PRÁTICA:

```
server.ts  X  package.json

src > server.ts > ...
1  import express,{Request, Response } from 'express'
2  import { Server } from 'http'
3  import { send } from 'process'
4
5  const server = express()
6
7  server.get('/',(req,res)=>{
8    |   res.send("Hello World!")
9  })
10
11 //criando uma nova rota
12 server.get('/contato', (req:Request,res:Response)=>{
13   |   res.send("Está é a página de contato")
14 })
15
16 server.listen(3000)
```

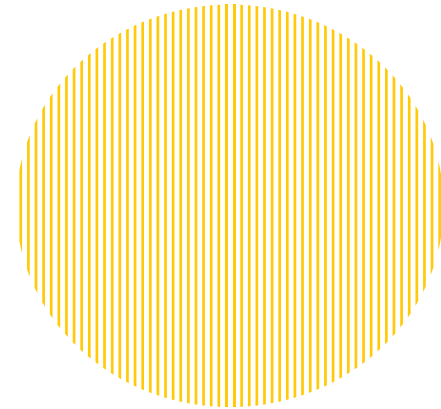
COLOQUE LOCALHOST/CONTATO



Está é a página de contato



TRABALHANDO COM ROTAS



MÉTODOS DAS ROTAS

Agora vamos compreender melhor o código que fizemos:

```
✓ server.get('/', (req, res) => {  
  |   res.send("Hello World!")  
  |  
  | })
```

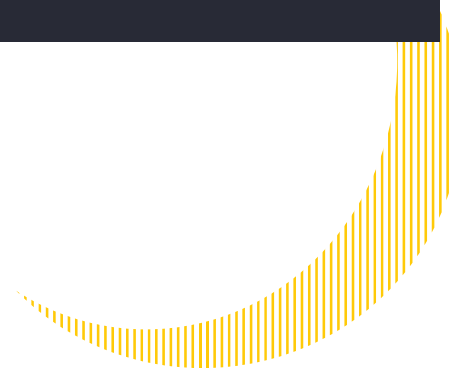
Sempre quando vamos acessar uma página ela vem junto coma requisição que chamamos de método, existem vários métodos, os principais são esses:

GET e POST

ESSA É UMA ROTA ESTÁTICA SIMPLES



```
server.get('/', (req: Request, res: Response) => {  
  res.send("Hello World!")  
})
```



CRIANDO UMA ROTA DINÂMICA

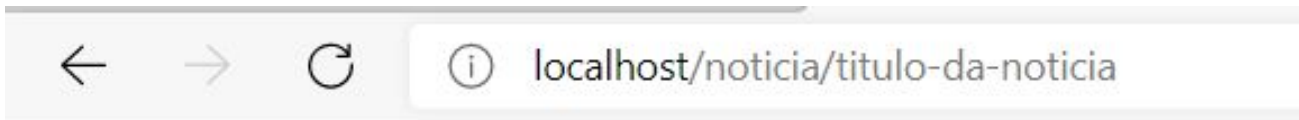


```
server.ts  ×  package.json
src > server.ts > ...
5  const server = express()
6
7  //ROTA ESTÁTICA
8  server.get('/', (req: Request, res: Response) => {
9    res.send("Hello World!")
10 })
11
12 //ROTA DINÂMICA
13 server.get('/noticia/titulo-da-noticia', (req: Request, res: Response) => {
14   res.send("Noticia Aparecendo na Tela")
15 })
16
17 server.get('/noticia/outra-noticia', (req: Request, res: Response) => {
18   res.send("Outra Noticia Aparecendo na Tela")
19 })
20
21 /*caso na porta 3000 esteja dando erro, troque a porta para 80 */
22 server.listen(80)
```

SERÁ QUE NOSSA ROTA ESTÁ DINÂMICA MESMO?



VEJA NO NAVEGADOR:

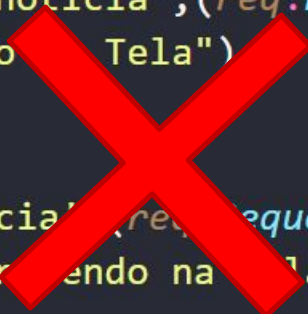


Noticia Aparecendo na Tela

O CERTO PARA UMA ROTA DINÂMICA, SERIA COLCOAR APENAS **/NOTICIA E A PÁGINA ME REDIRECIONAR PARA QUALQUER NOTÍCIA (SEM PRECISAR FICAR CRIANDO VÁRIAS ROTAS DO TIPO **/NOTICIA/ALGUMA-NOTICIA****

```
//ROTA DINÂMICA
server.get('/noticia/titulo-da-noticia',(req:Request,res:Response)=>{
  res.send("Noticia Aparecendo na Tela")
})

server.get('/noticia/outra-noticia',(req:Request,res:Response)=>{
  res.send("Outra Noticia Aparecendo na Tela")
})
```

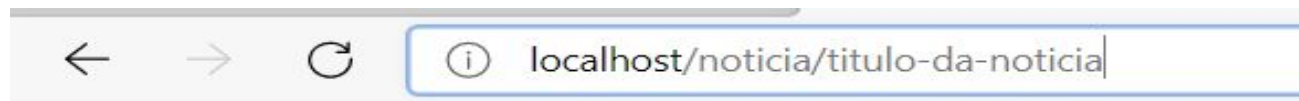


CRIANDO UMA ROTA DIÂMICA DE VERDADE

```
//ROTA ESTÁTICA
server.get('/',(req:Request,res:Response)=>{
  res.send("Hello World!")
})

/*vamos deixar o /noticia dinâmico!
utilizamos a palavra slug para transformar
essa rota em dinâmica
*/
server.get('/noticia/:slug',(req:Request,res:Response)=>{
  /*agora vamos receber na requisição(req)
  os dados que eu preciso para exibir
  a notícia X ou a notícia Y */
  let slug: string = req.params.slug
  res.send(`Notícia: ${slug}`)
})
server.listen(80)
```


COLOQUE O NOME QUE VOCÊ QUISE APÓS **NOTICIA/**



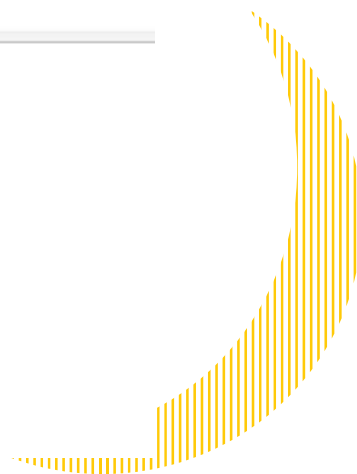
Notícia: titulo-da-noticia



Notícia: viviane



Notícia: fulano-ganha-na-megasena



OUTRO EXEMPLO DE ROTA DINÂMICA EM UM SITE DE VOOS (COM DOIS VALORES DIFERENTES)

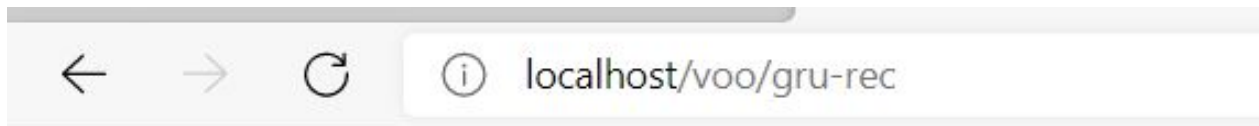
```
import express,{Request, Response } from 'express'
import { Server } from 'http'
import { send } from 'process'
const server = express()

//ROTA ESTÁTICA
server.get('/',(req:Request,res:Response)=>{
  res.send("Hello World!")
})

//ROTA DINÂMICA COM DOIS VALORES
server.get('/voo/:origem-:destino',(req:Request, res:Response) => {
  let {origem, destino } = req.params;

  res.send(`Procurando voos de ${origem} até ${destino}`)
})
server.listen(80)
```

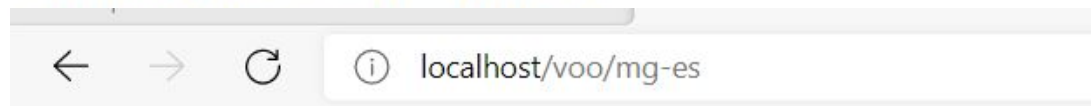
VEJA NO NAVEGADOR



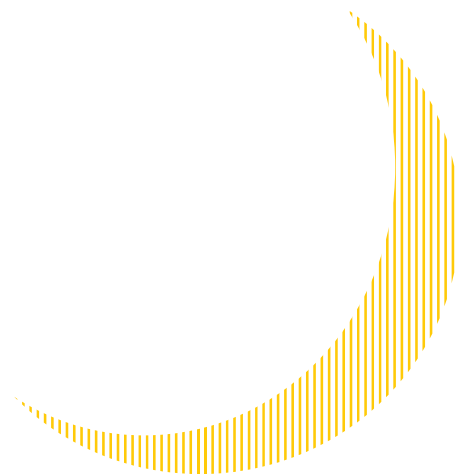
Procurando voos de gru até rec

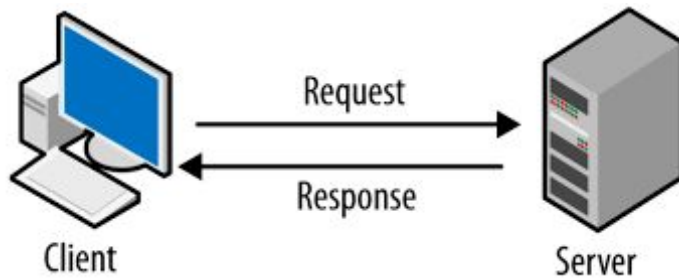


Procurando voos de sp até rj



Procurando voos de mg até es





MÉTODOS HTTP

GET

PUT

POST

DELETE



ACESSAR UMA PÁGINA
GET


RECEBER OS DADOS INTERNAMENTE
POST




MÉTODO GET

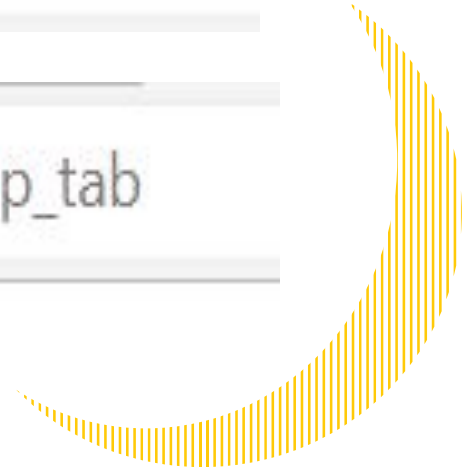


Quando vamos acessar uma página, nós acessamos através do método GET

 <https://www.facebook.com/friends>

 <https://www.facebook.com/friends/list>

https://www.facebook.com/marketplace/?ref=app_tab



HTTP REQUISIÇÃO GET



`www.server.com`



Quando utilizamos o GET os arquivos/recursos **são passados no cabeçalho da requisição**. Por isso, podem ser vistos pela URN

MÉTODO POST

Quando temos uma página de login, e o usuário faz o login nessa página, No momento em que ele entra, não podemos exibir os dados do usuário (como e-mail e senha na barra de endereços do navegador)



Email ou telefone

Senha

Entrar

[Esqueceu a senha?](#)

Criar nova conta

[Criar uma Página](#) para uma celebridade, uma marca ou uma empresa.

O método post faz o encapsulamento da url e esconde essas informações

HTTP REQUISIÇÃO POST



`www.server.com/user/`



O POST encapsula os arquivos/recursos **no corpo da requisição HTTP**.
Escondendo eles da URN

FUNÇÃO CALLBACK

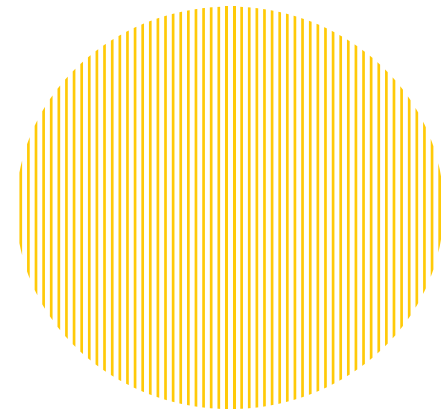
```
server.get('/', (req: Request, res: Response) => {  
    res.send("Hello World!")  
})
```

Req: Quando acessamos um site, junto desse acesso mandamos várias informações, como cookies, identificação do navegador, IP etc.

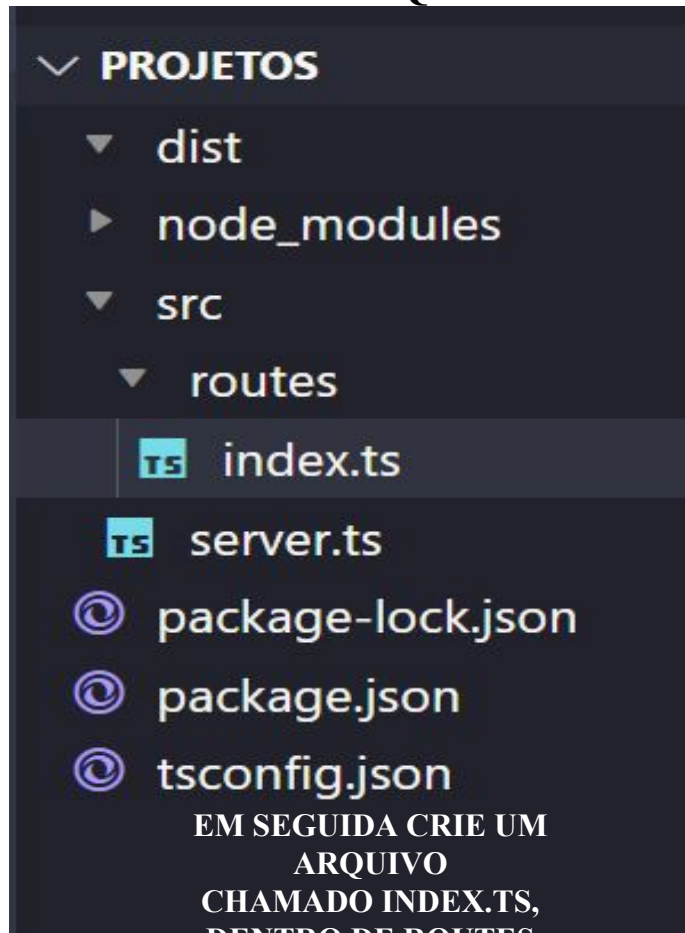
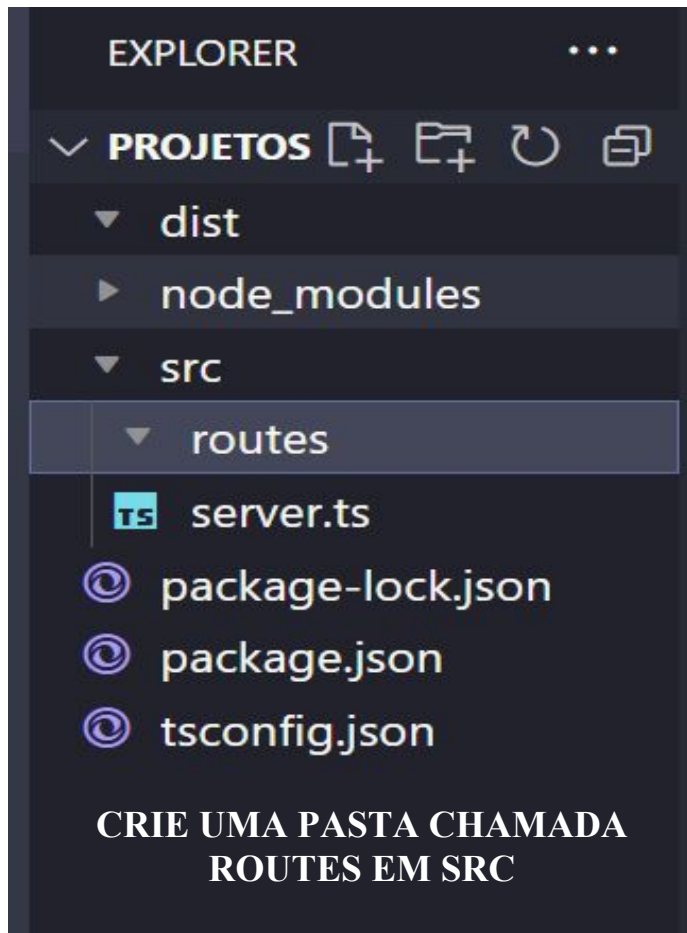
Res: Ele é responsável pela resposta que o servidor vai dar para o usuário que fez a requisição, essa resposta pode ser uma frase, uma página, um dado específico, uma imagem etc.

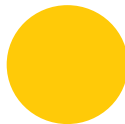


ORGANIZANDO ROTAS

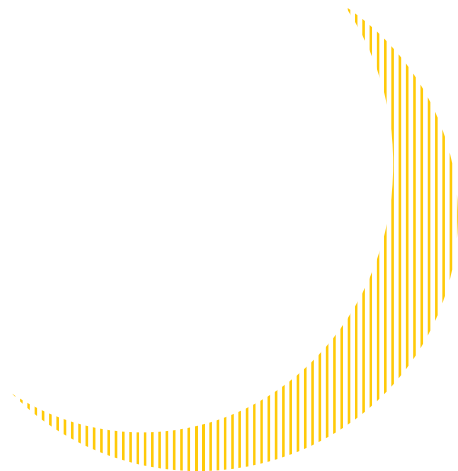


AGORA PRECISAMOS ORGANIZAR AS NOSSAS ROTAS, NÃO VAMOS CRIAR O NOSSO SITE INTEIRO EM UM ARQUIVO SÓ





**VOCÊ JÁ APRENDEU COMO CRIAR ROTAS, ENTÃO DENTRO DE
INDEX.TS CRIE TRÊS ROTAS ESTÁTICAS PARA HOME, CONTATO E
SOBRE**



CRIANDO E EXPORTANDO A ROTA



server.ts

index.ts

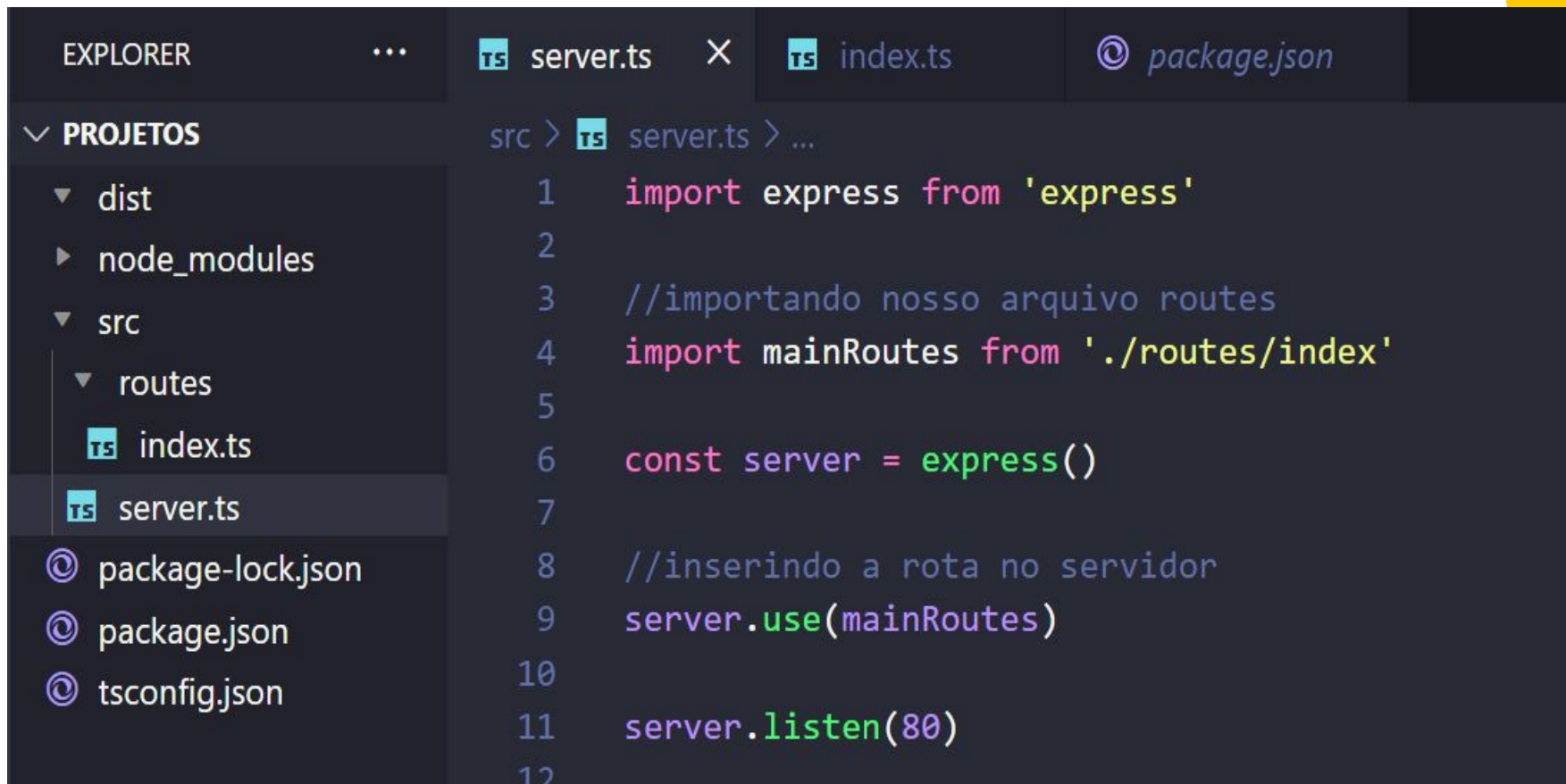


package.json

src > routes > index.ts > ...

```
1  import { Router, Request, Response } from "express";
2
3  const router = Router()
4
5  router.get('/', (req: Request, res: Response) =>{
6    |   res.send("Olá mundo!")
7  })
8
9  router.get('/contato', (req: Request, res: Response) =>{
10   |   res.send("Formulário de contato")
11 })
12
13 router.get('/sobre', (req: Request, res: Response) =>{
14   |   res.send("sobre nós")
15 })
16
17 //exportando o arquivo router
18 export default router
```

USANDO A ROTA NO SERVIDOR



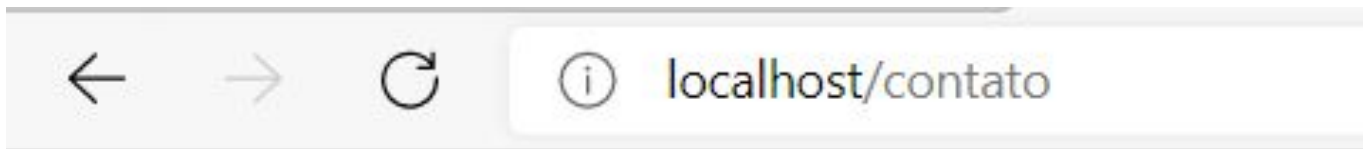
The image shows a VS Code editor window with a dark theme. On the left is the Explorer sidebar with a tree view of a project named 'PROJETOS'. The tree structure is as follows:

- PROJETOS
 - dist
 - node_modules
 - src
 - routes
 - index.ts
 - server.ts (selected)
 - package-lock.json
 - package.json
 - tsconfig.json

On the right, the editor displays the contents of 'server.ts'. The code is as follows:

```
src > TS server.ts > ...  
1  import express from 'express'  
2  
3  //importando nosso arquivo routes  
4  import mainRoutes from './routes/index'  
5  
6  const server = express()  
7  
8  //inserindo a rota no servidor  
9  server.use(mainRoutes)  
10  
11 server.listen(80)  
12
```

VERIFIQUE SE SUAS ROTAS ESTÃO FUNCIONANDO



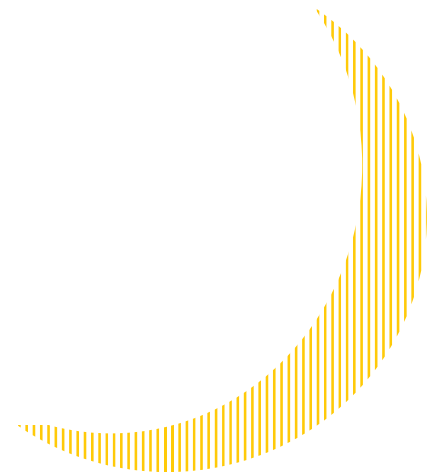
Formulário de contato



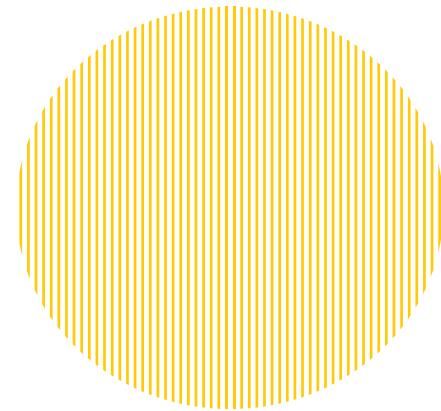
sobre nós



Olá mundo!



**CRIANDO 404
PÁGINA NÃO
ENCONTRADA**

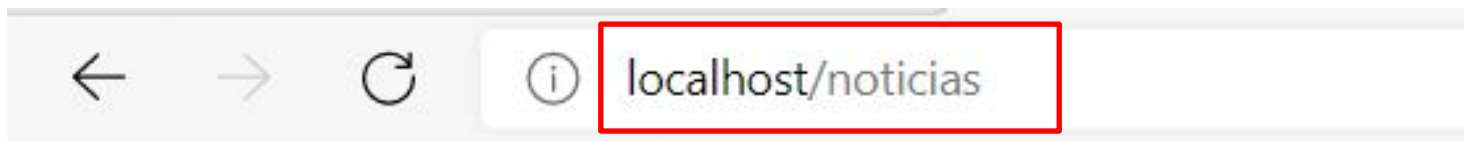


VAMOS CRIAR UMA ROTA CASO O USUÁRIO DIGITE ALGUM ENDEREÇO QUE NÃO EXISTE

```
server.ts  ×  index.ts  package.json

src > server.ts > ...
 1  //puxe o Request e o Responde
 2  import express,{Request, Response} from 'express'
 3
 4  import mainRoutes from './routes/index'
 5
 6  const server = express()
 7
 8  server.use(mainRoutes)
 9
10  //criando a rota 404
11  server.use((req: Request, res:Response) =>{
12      /* quando a página não encontrar nenhuma das rotas,
13      ele entrará aqui */
14      res.status(404).send('Página não encontrada!')
15  })
16
17  server.listen(80)
```

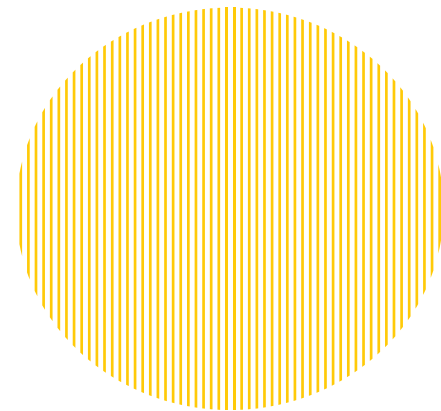
COLOQUE UMA PÁGINA QUE NÃO EXISTE NO NAVEGADOR



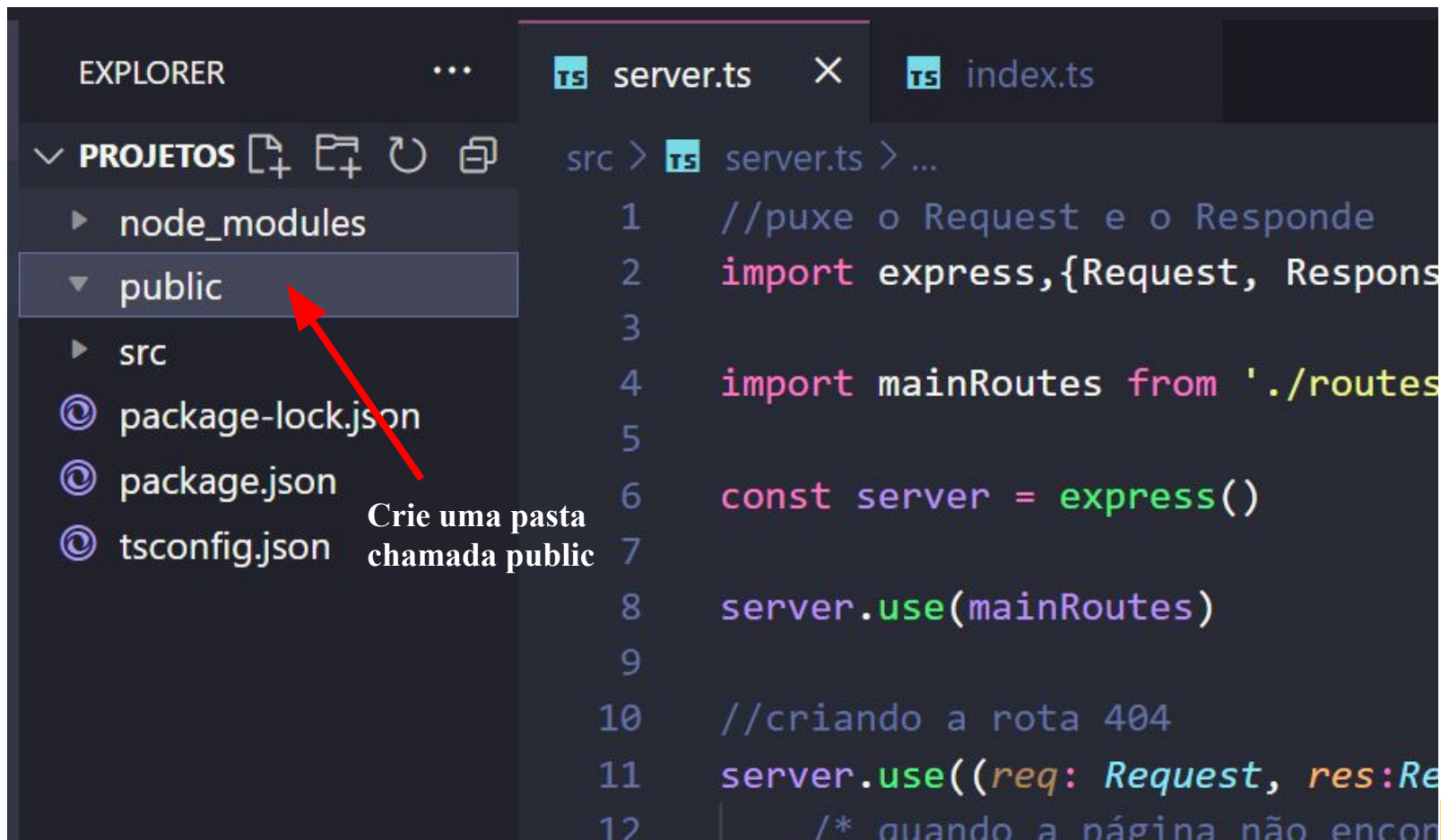
Página não encontrada!



**PASTA PÚBLICA
E ARQUIVOS
ESTÁTICOS**



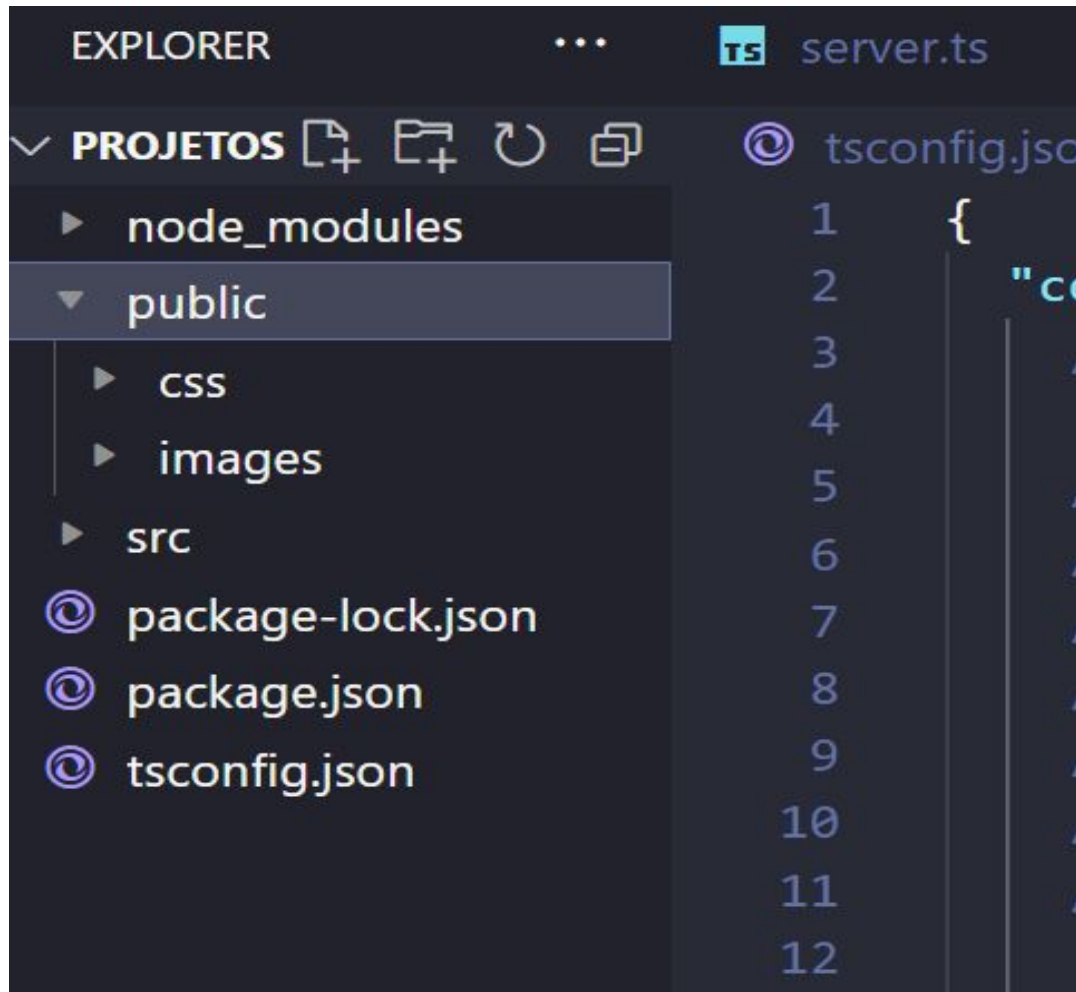
CHEGOU A HORA DE COLOCARMOS ARQUIVOS NO NOSSO NODEJS E MOSTRAR ALGO VISUAL NA TELA



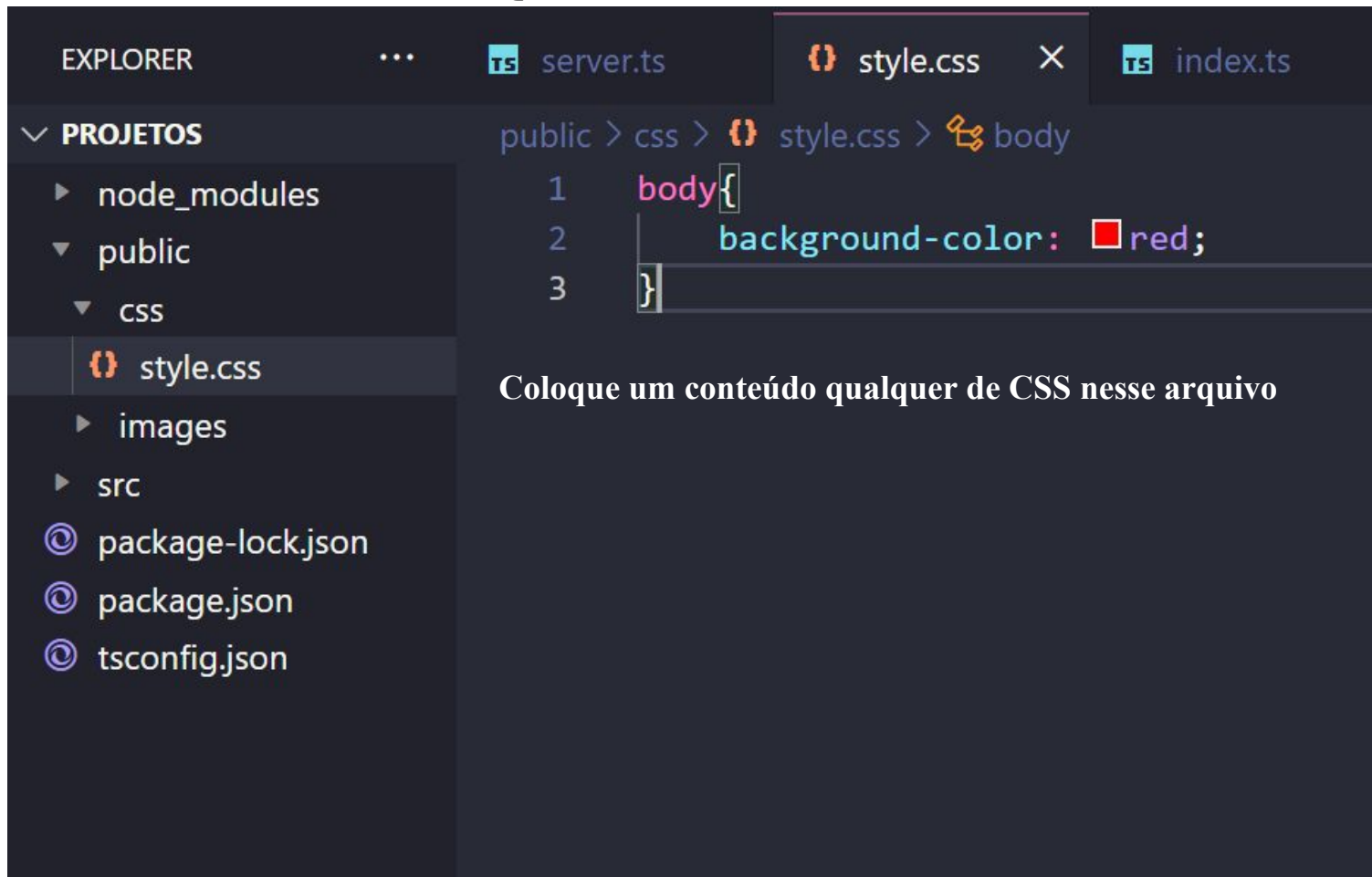
The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under 'PROJETOS'. The 'public' folder is highlighted, and a red arrow points to it with the text 'Crie uma pasta chamada public'. The main editor area shows the 'server.ts' file with the following TypeScript code:

```
src > TS server.ts > ...
1 //puxe o Request e o Responde
2 import express,{Request, Respons
3
4 import mainRoutes from './routes
5
6 const server = express()
7
8 server.use(mainRoutes)
9
10 //criando a rota 404
11 server.use((req: Request, res:Re
12 /* quando a página não encon
```

CRIE A PASTA CSS E A PASTA IMAGES DENTRO DE PUBLIC



CRIE UM ARQUIVO CHAMADO STYLE.CSS



AGORA PRECISAMOS DEIXAR A PASTA PUBLIC DE FORMA PÚBLICA



```
server.ts × style.css
src > server.ts > ...
1
2 import express,{Request, Response} from 'express'
3
4 import mainRoutes from './routes/index'
5
6 const server = express()
7
8 //criando a rota para nossa pasta PUBLIC
9 server.use(express.static('public'))
10
11 1 passo - criar a rota para public
12
13 server.use(mainRoutes)
14
15 server.use((req: Request, res:Response) =>{
16
17     res.status(404).send('Página não encontrada!')
18 })
19
20 server.listen(80)
```



VERIFIQUE SE SUA ROTA ESTÁ FUNCIONANDO, ACESSE A PASTA DO SUE NAVEGADOR




```
body{  
  background-color: red;  
}
```

VEJA QUE O ARQUIVO ESTÁ ACESSÍVEL ATRAVÉS DA URL



O PRÓXIMO PASSO É DEIXAR ESSA PASTA PUBLIC NA RAÍZ DO HD, PARA QUANDO FORMOS PROCURAR POR ELA NO CMD, ELA NÃO APAREÇA 'PÁGINA NÃO ENCONTRADA'.

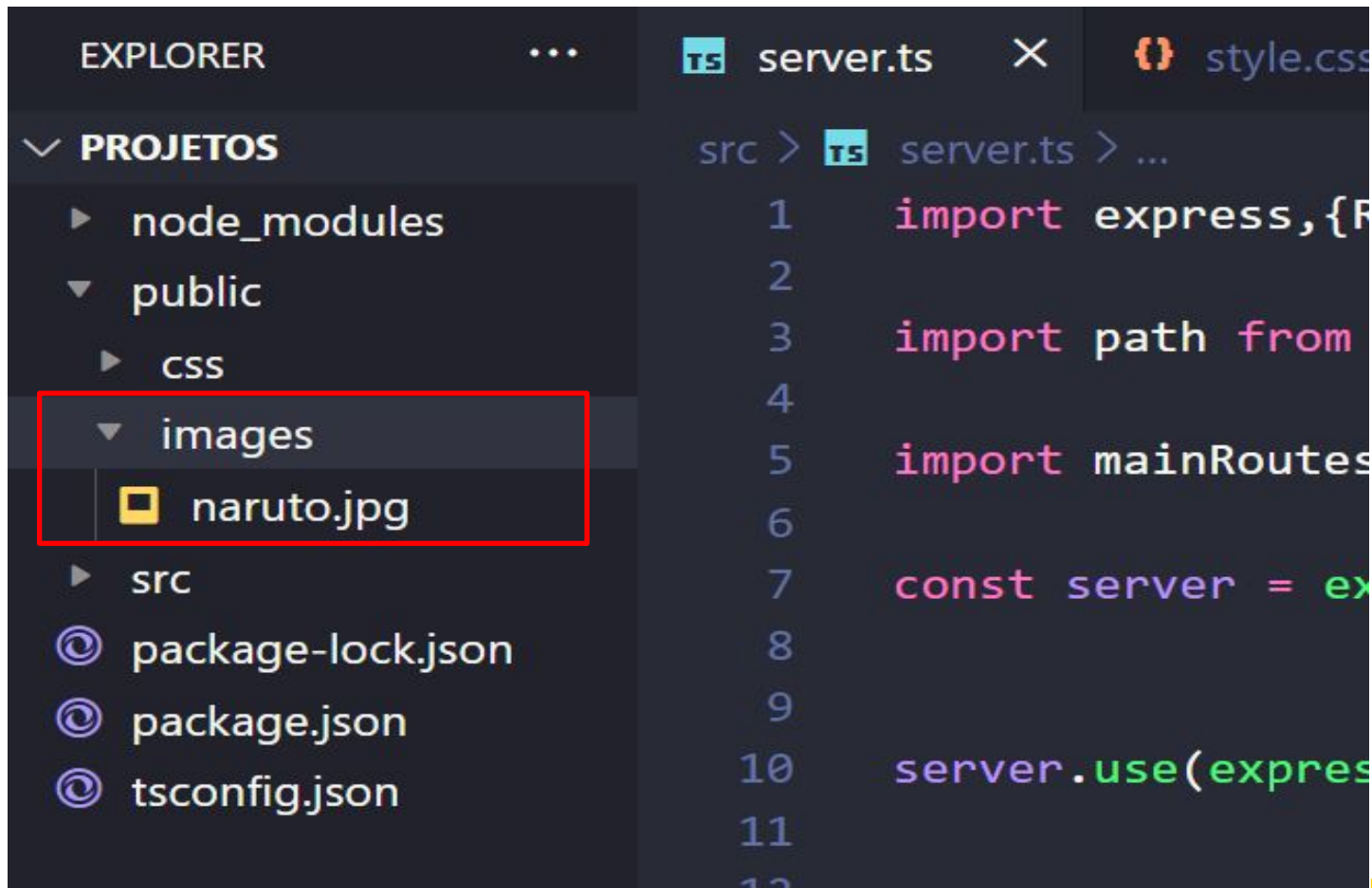


```
server.ts × style.css
src > server.ts > ...
1 import express,{Request, Response} from 'express'
2
3 import path from 'path'
4
5 import mainRoutes from './routes/index'
6
7 const server = express()
8
9
10 server.use(express.static(path.join(__dirname, '../public')))
11
12 server.use(mainRoutes)
13
14 server.use((req: Request, res:Response) =>{
15
16     res.status(404).send('Página não encontrada!')
17
18 })
19
20 server.listen(80)
21
```

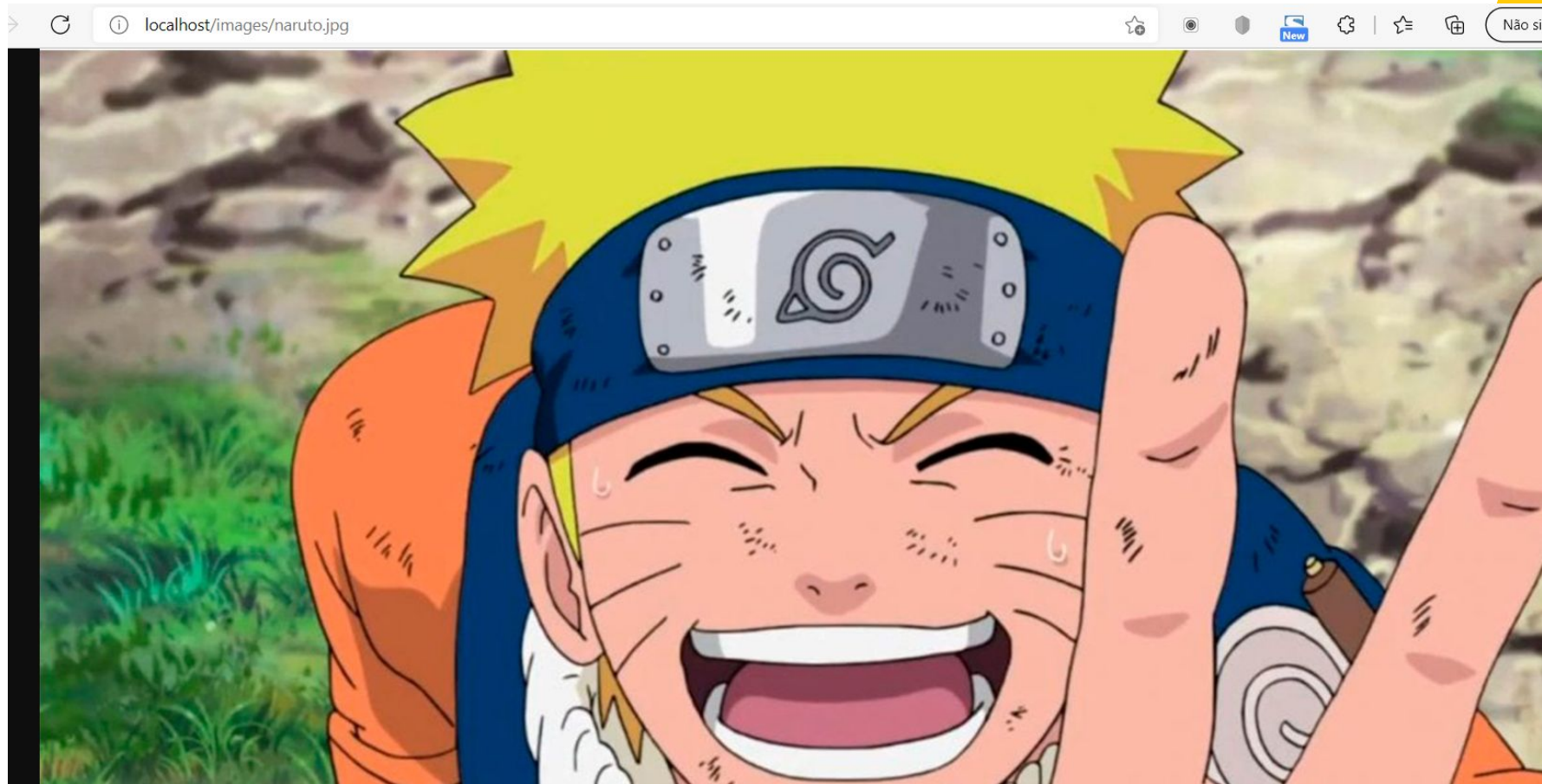
Public é onde está a pasta dos meus arquivos que quero executar

Dirname vai especificar o diretório do arquivo que eu estou executando essa rota, que é em server.ts

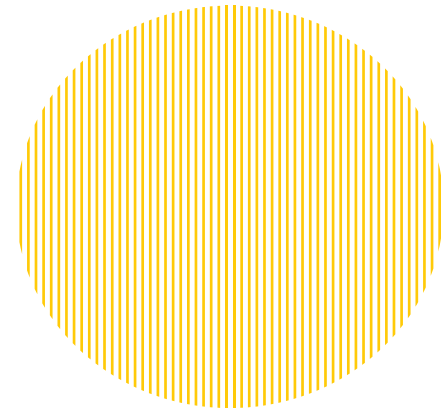
ESCOLHA UMA IMAGEM DE SUA PREFERÊNCIA E COLOQUE NA NOSSA PASTA IMAGES



ACESSE A IMAGEM PELO SEU SERVIDOR



TEMPLATES ENGINES



O QUE VIMOS ATÉ AGORA COM NODEJS?



- Configuração do projeto com NodeJS com Express
- Criamos rotas
- Criamos nossa pasta public

O PRÓXIMO PASSO AGORA É APRENDER A COLOCAR HTML NO NODEJS

**FAZEMOS ISSO ATRAVÉS DO USO DE BIBLIOTECAS QUE SÃO
TEMPLATES ENGINES**



TEMPLATES ENGINES MAIS USADOS NO MERCADO

github.com/pugjs/pug

Pug

Full documentation is at pugjs.org

Pug is a high-performance template engine heavily influenced by [Haml](#) and implemented with JavaScript for [Node.js](#) and browsers. For bug reports, feature requests and questions, [open an issue](#). For discussion join the [chat room](#).

You can test drive Pug online [here](#).

Professionally supported pug is now available



Packages

Package Name	Version
pug	NPM V3.0.2
pug-attrs	NPM V3.0.0
pug-code-gen	NPM V2.0.3
pug-error	NPM V2.0.0
pug-filters	NPM V4.0.0

**O PRÓPRIO EXPRESS
RECOMENDA O USO DO PUG**

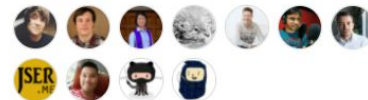
Packages

No packages published

Used by 390k

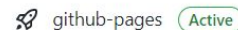


Contributors 252

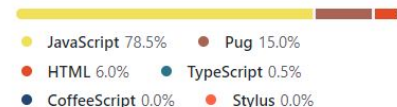


+ 241 contributors

Environments 1



Languages



TEMPLATES ENGINES MAIS USADOS NO MERCADO

github.com/janl/mustache.js



☰ README.md

🔗 mustache.js - Logic-less {{mustache}} templates with JavaScript

What could be more logical awesome than no logic at all?

build passing

**MUSTACHE É UM DOS MELHORES TEMPLATES
POIS FUNCIONA PARA PHP, RUBY, PYTHON ETC**

[mustache.js](#) is a zero-dependency implementation of the [mustache](#) template system in JavaScript.

[Mustache](#) is a logic-less template syntax. It can be used for HTML, config files, source code - anything. It works by expanding tags in a template using values provided in a hash or object.

We call it "logic-less" because there are no if statements, else clauses, or for loops. Instead there are only tags. Some tags are replaced with a value, some nothing, and others a series of values.

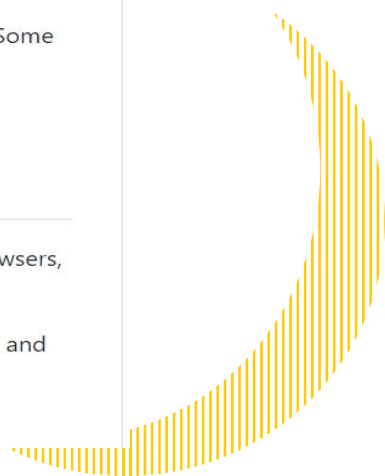
For a language-agnostic overview of mustache's template syntax, see the [mustache\(5\)](#) [manpage](#).

Where to use mustache.js?

You can use mustache.js to render mustache templates anywhere you can use JavaScript. This includes web browsers, server-side environments such as [Node.js](#), and [CouchDB](#) views.

mustache.js ships with support for the [CommonJS](#) module API, the [Asynchronous Module Definition](#) API (AMD) and [ECMAScript modules](#).

In addition to being a package to be used programmatically, you can use it as a [command line tool](#).



TEMPLATES ENGINES MAIS USADOS NO MERCADO

github.com/mde/ejs



☰ README.md

Embedded JavaScript templates

build passing

david no longer available

vulnerabilities 0

🔗 Installation

```
$ npm install ejs
```

Features

- Control flow with `<% %>`
- Escaped output with `<%= %>` (escape function configurable)
- Unescaped raw output with `<%- %>`
- Newline-trim mode ('newline slurping') with `-%>` ending tag
- Whitespace-trim mode (slurp all whitespace) for control flow with `<%_ _%>`
- Custom delimiters (e.g. `[? ?]` instead of `<% %>`)
- Includes
- Client-side support
- Static caching of intermediate JavaScript

