

Introdução à Web e APIs



Prazer, Analu.

Eu sou desenvolvedora Back-End, professora e Bootcamp Lead na Reprograma e organizadora na comunidade Java Girls JUG
Fui da primeira turma de Back-End da Reprograma
Hoje trabalho como Engenheira de Software no Banco Itaú.

Email: sampaioanaluiza@gmail.com



Objetivos



Modelo Server/Client



API e Web APIs



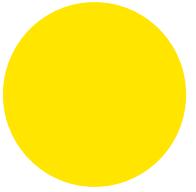
URI, URL, URN, Dominio, IP e DNS



Iniciando com Nest.js



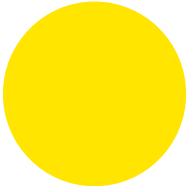
Protocolo HTTP



to-do API



Request e Response



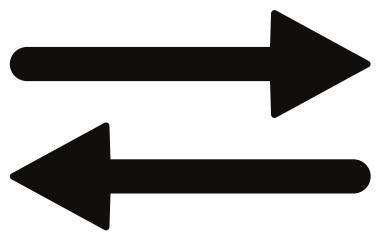
Para o lar

Como funciona a Internet?

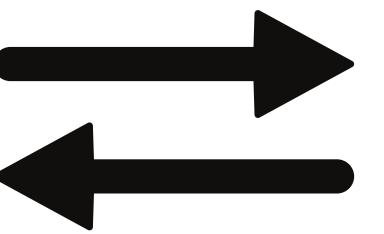
Servidor/Cliente



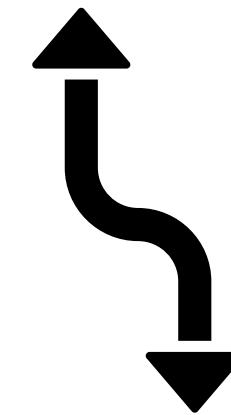
Usuários



Client



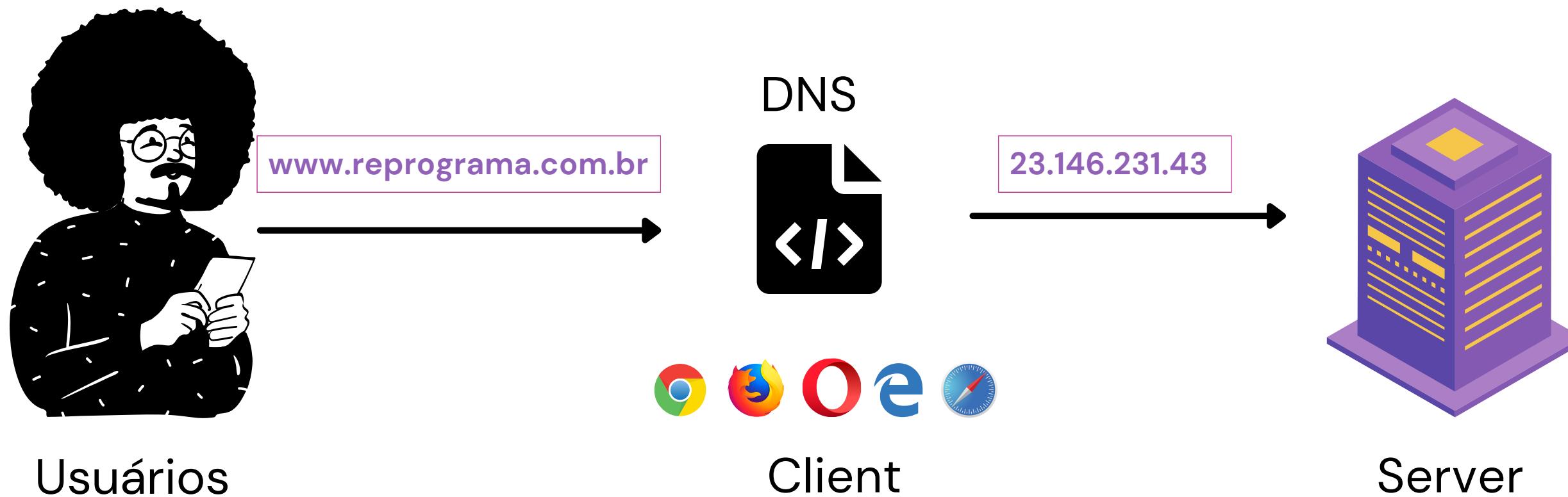
Server



Banco de dados



Servidor/Cliente

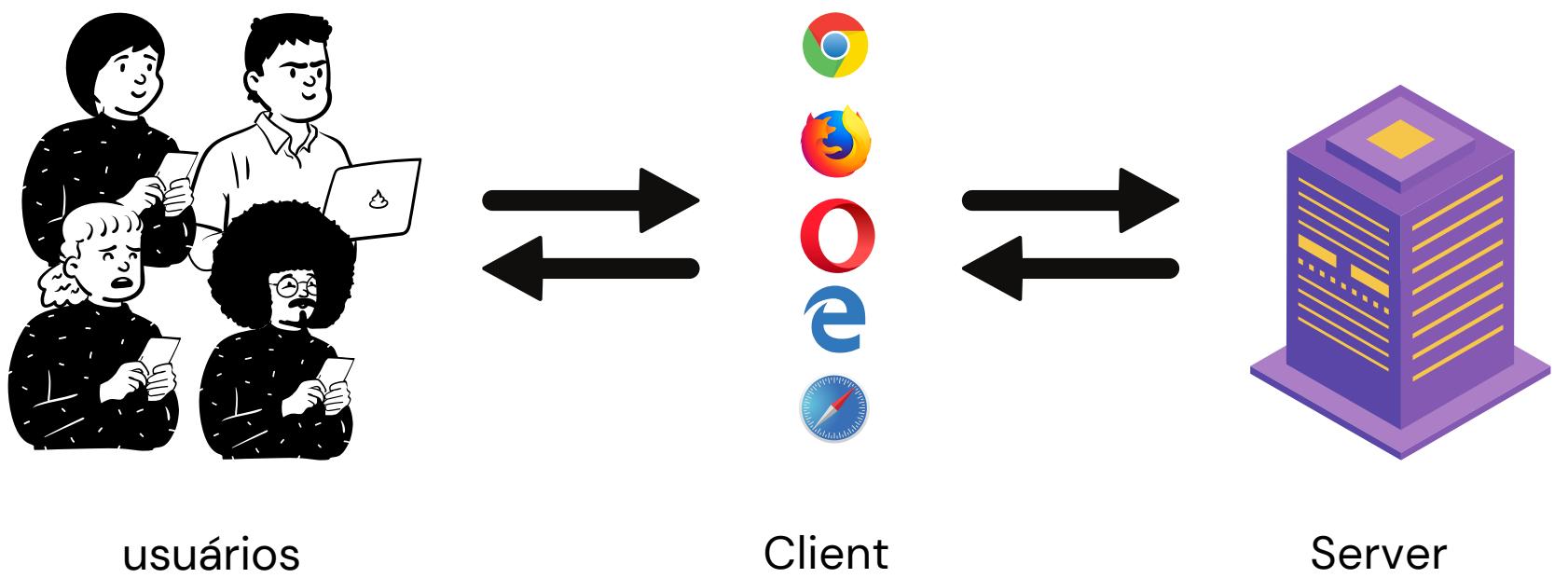


Client

Client é a interface que os usuários interagem, é essa camada que é responsável de **solicitar** serviços e informações de um ou mais servidores.

Algumas tarefas a serem realizadas pelo Cliente:

- Manipulação de tela
- Interpretação de menus ou comandos
- Entrada e validação dos dados
- Recuperação de erro
- Manipulação de janelas
- Gerenciamento de som e vídeo (em aplicações multimídia)



Server

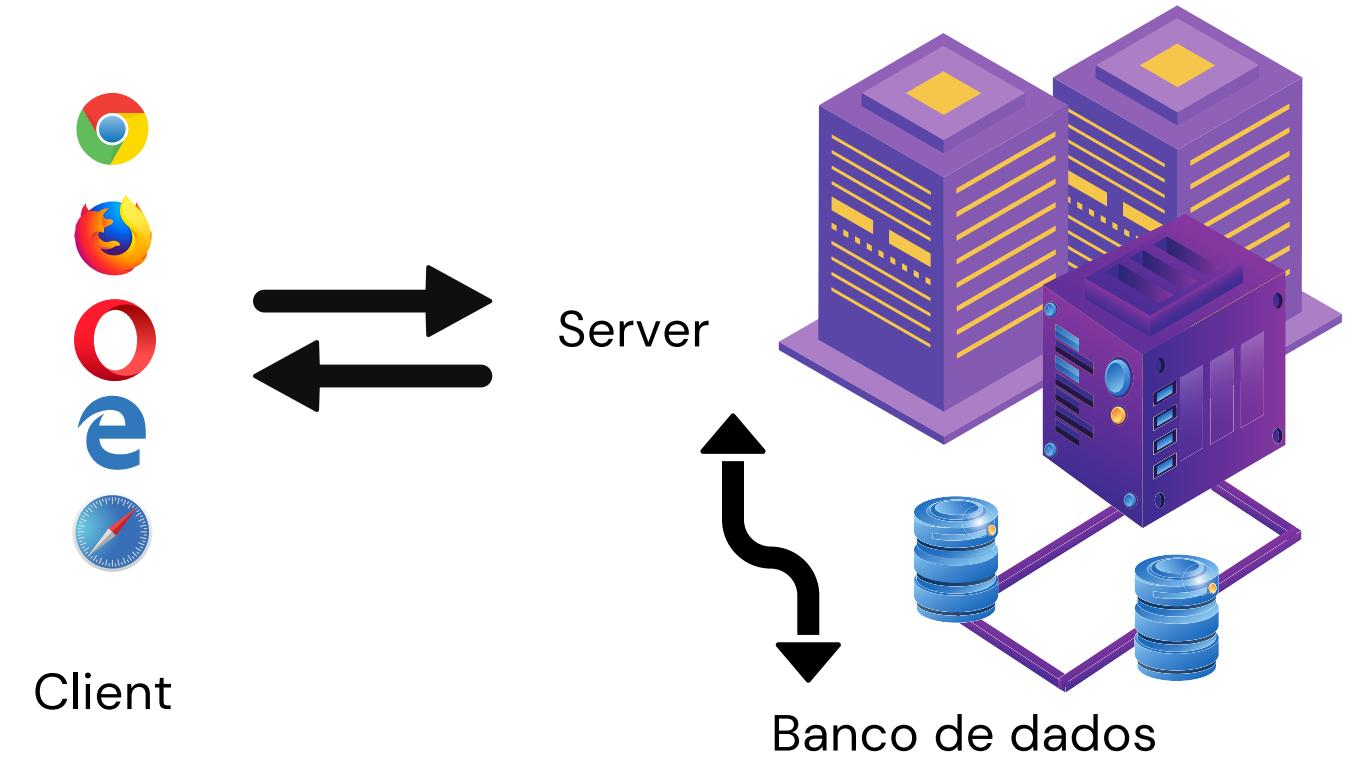
O Servidor é o responsável pelo processo, organização e gerenciamento das informações.

É ele que **responde às solicitações** feitas pelo Client.

Ele é um processo reativo, disparado pela chegada de pedidos de seus clientes

O processamento do servidor geralmente inclui:

- Acessar
- Organizar os dados compartilhados
- Fazer a comunicação com o Banco de Dados
- Atualizar dados previamente armazenados
- Gerenciamento dos recursos compartilhados.



Objetivos



Modelo Server/Client



API e Web APIs



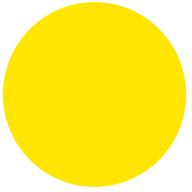
URI, URL, URN, Dominio, IP e DNS



Iniciando com Nest.js



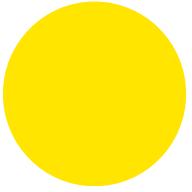
Protocolo HTTP



to-do API



Request e Response



Para o lar

www.reprograma.com.br

O que acontece quando acessamos um site?



- 1 A URI é processada
- 2 É feita uma requisição
- 3 É dada uma Resposta
- 4 A pagina é renderizadas e aparece na tela

1

A URI é processada – Domínio



Todo site tem um **domínio**, normalmente é por ele que acessamos e conhecemos o Site.

Porém, no Server esse site não está registrado pelo nome de domínio, e sim pelo **endereço de IP**

Internet Protocol Address é o endereço exato de onde o site está dentro do servidor.

Então, antes de uma requisição ser feita o domínio deve virar o IP, e pra isso, usamos o **DNS**, o Domain Name System (Sistema de Nome de Domínio) que é como um grande dicionário de domínio para IP que já vem "de fábrica" no browser

1

A URI é processada – URL

Para entendermos uma URI precisamos entender a URL e a URN.



URL – Uniform Resource Locator (localizador de recurso uniforme). Ela representa o local/Host que estão localizados os recursos

URL

www.reprograma.com.br

«endereço do servidor»

1

A URI é processada – URN



URN – Uniform Resource Name (Nome de Recursos Universal). Ela representa um recurso específico na web que está sendo acessado

/equipe.html

<recurso>

/alunas.html

<recurso>

URN

/parcerias.html

<recurso>

1

A URI é processada – URI



URI – Uniform Resource Identifier (Identificador de Recursos Universais). É o identificador contendo que une o protocolo http + o localizador do recurso (URL) + nome do recurso (URN)

Pode ser uma página html, imagem, video ou qualquer outro arquivo web tem um endereço dentro da internet, esse endereço é a **URI**

protocolo

URL

URN

https://

www.reprograma.com.br

/equipe.html

<protocolo>

<endereço do servidor>

<recurso>

Objetivos



Modelo Server/Client



API e Web APIs



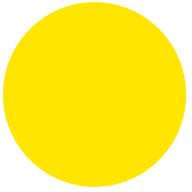
URI, URL, URN, Dominio, IP e DNS



Iniciando com Nest.js



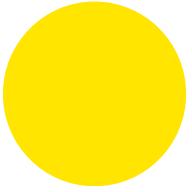
Protocolo HTTP



to-do API



Request e Response



Para o lar

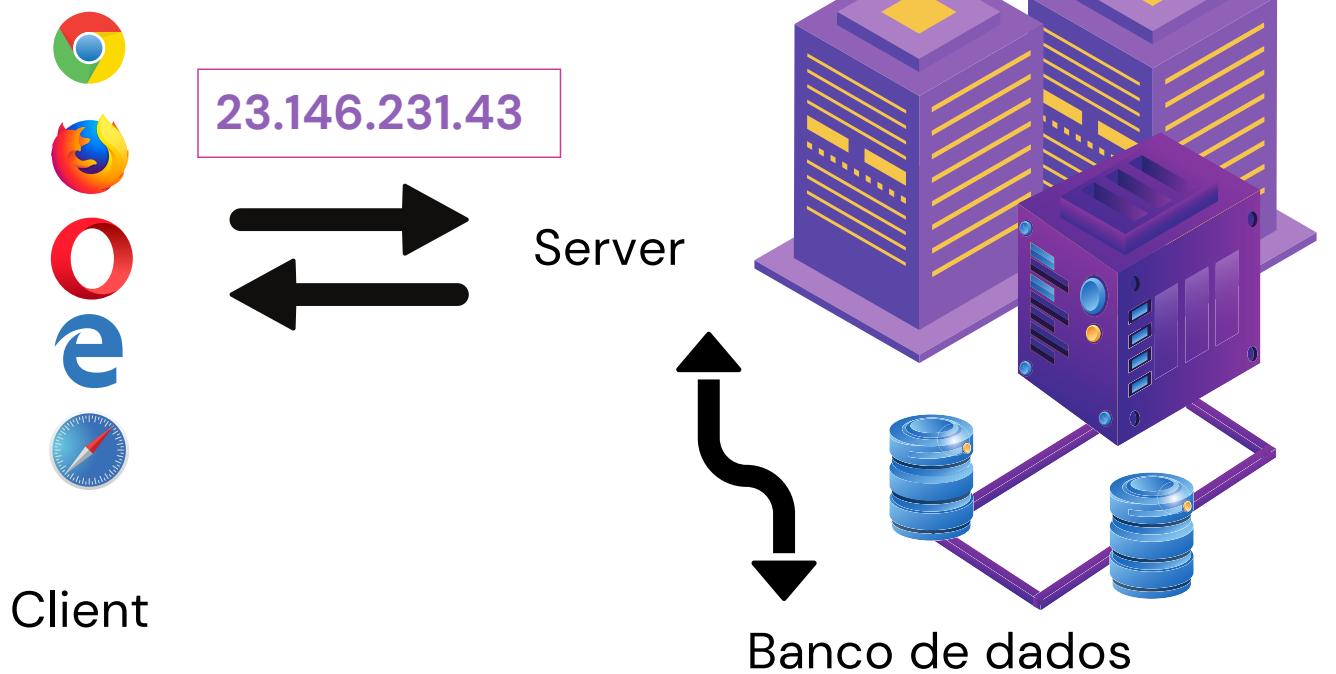
2

O Request é enviado

Agora com endereço certo, o Client faz uma requisição, ou Request, cheio de informações desejadas.

Pra que isso aconteça, tanto o Server quanto o Client devem "falar a mesma língua".

Na maioria dos casos, essa comunicação entre Server e Client é feita a partir do **Protocolo HTTP**



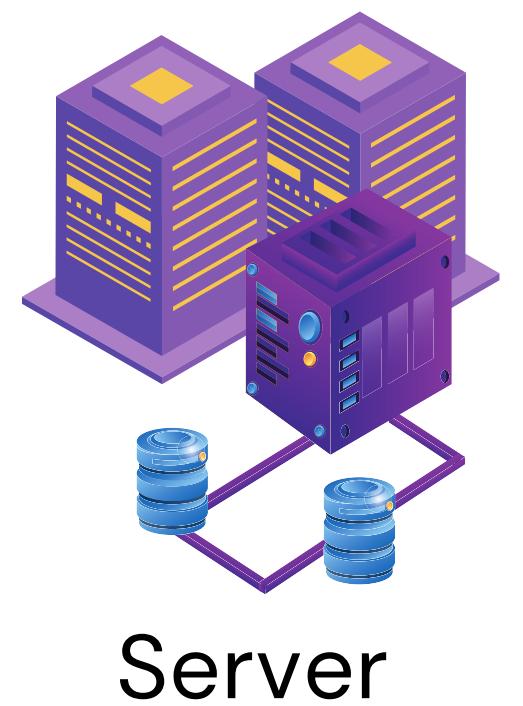
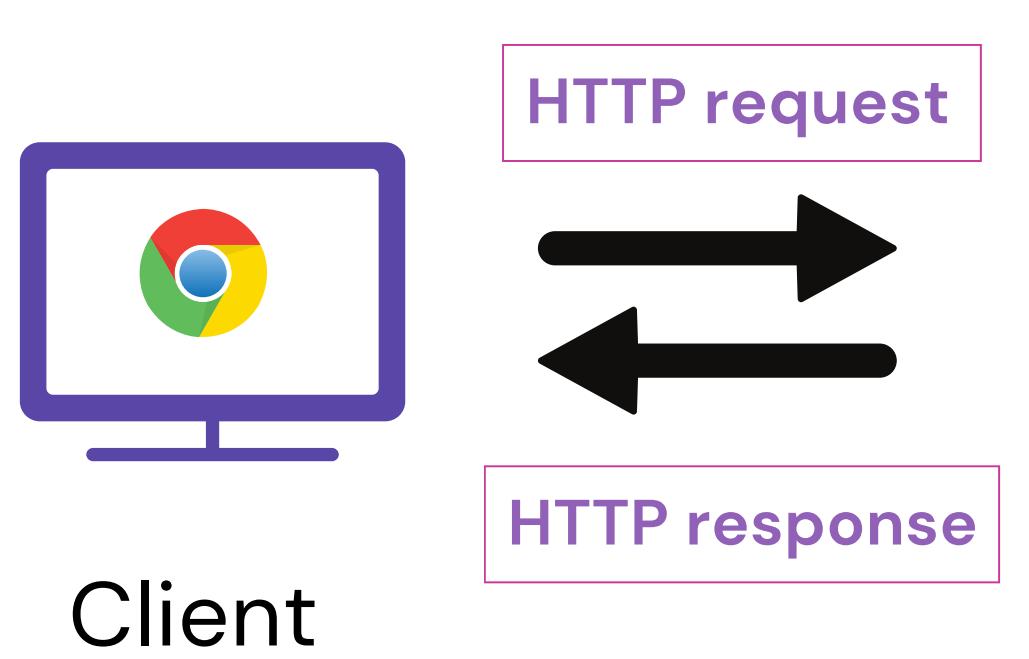
2

O Request é enviado – HTTP

Protocolo de Transferência de Hipertexto, o HTTP, é um protocolo usado dentro do modelo Client/Server é baseado em pedidos (requests) e respostas (responses).

O protocolo HTTP define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada.

Eles são chamados de **Verbos HTTP** ou **Métodos HTTP**.



2

O Request é enviado – CRUD

CRUD é a composição da primeira letra de 4 operações básicas de um banco de dados, e são o que a maioria das aplicações faz

- ✓ C: Create (criar) - criar um novo registro
- 👁 R: Read (ler) - exibir as informações de um registro
- ♻️ U: Update (atualizar) - atualizar os dados do registro
- ✗ D: Delete (apagar) - apagar um registro

Os verbos HTTP mais utilizados são:

- GET
- POST
- PUT
- PATCH
- DELETE

Cada um deles corresponde a uma ação real no banco de dados.

GET

POST

PUT

PATCH

DELETE

ler

criar

substituir

modificar

excluir

3

O Server responde

Quando o Client faz um Request o Server envia um **Response**.

E na resposta tem, além do resultado do que foi pedido, um código de status numérico padronizado

código

100-199

200-299

300-399

400-499

500-599

tipo de resposta

informação

sucesso

redirecionamento

erro do cliente

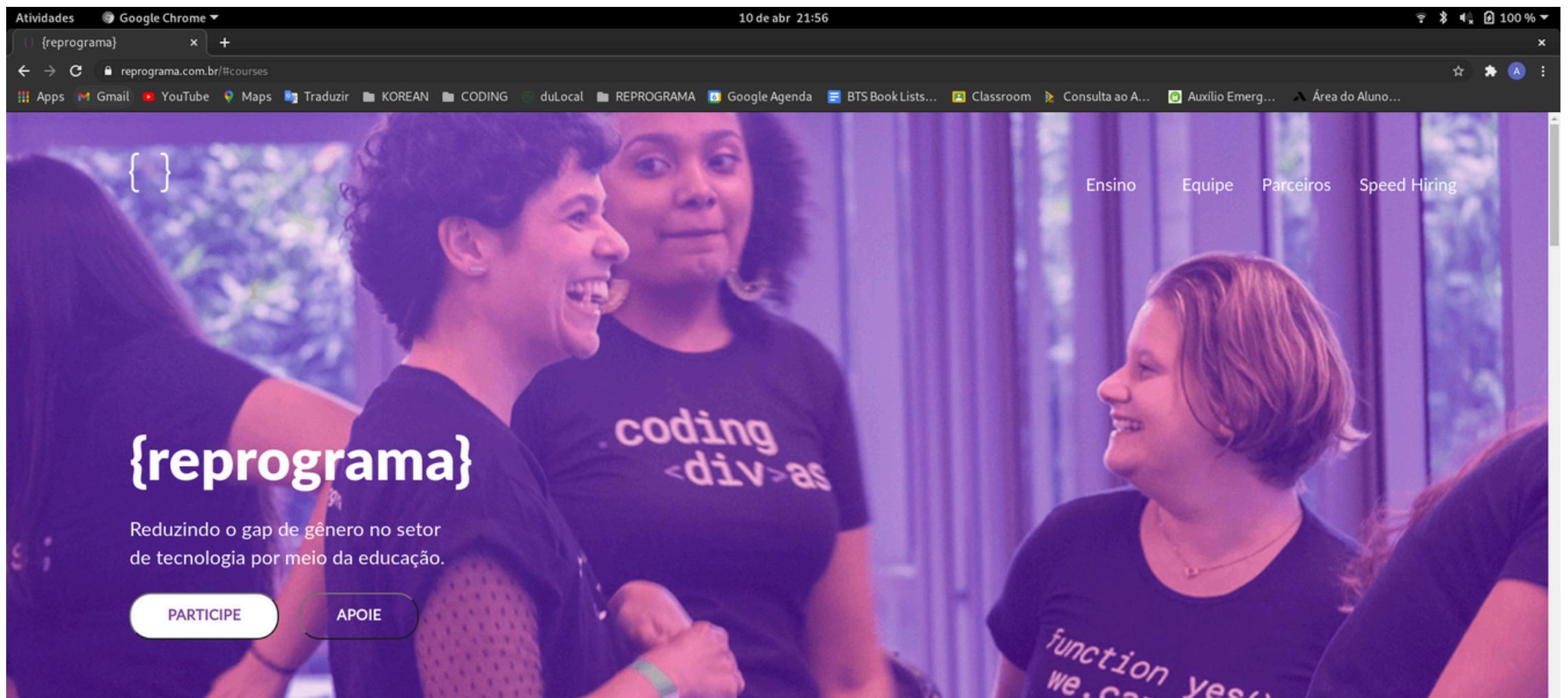
erro de servidor

4

O site aparece na tela



Finalmente!



Objetivos



Modelo Server/Client



API e Web APIs



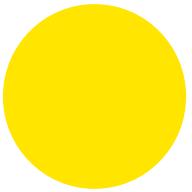
URI, URL, URN, Dominio, IP e DNS



Iniciando com Nest.js



Protocolo HTTP



to-do API

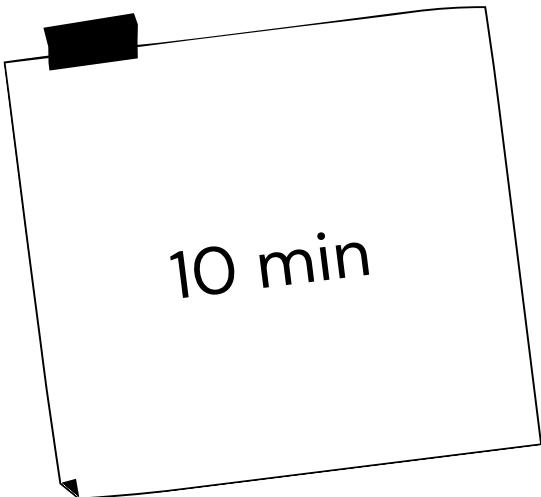


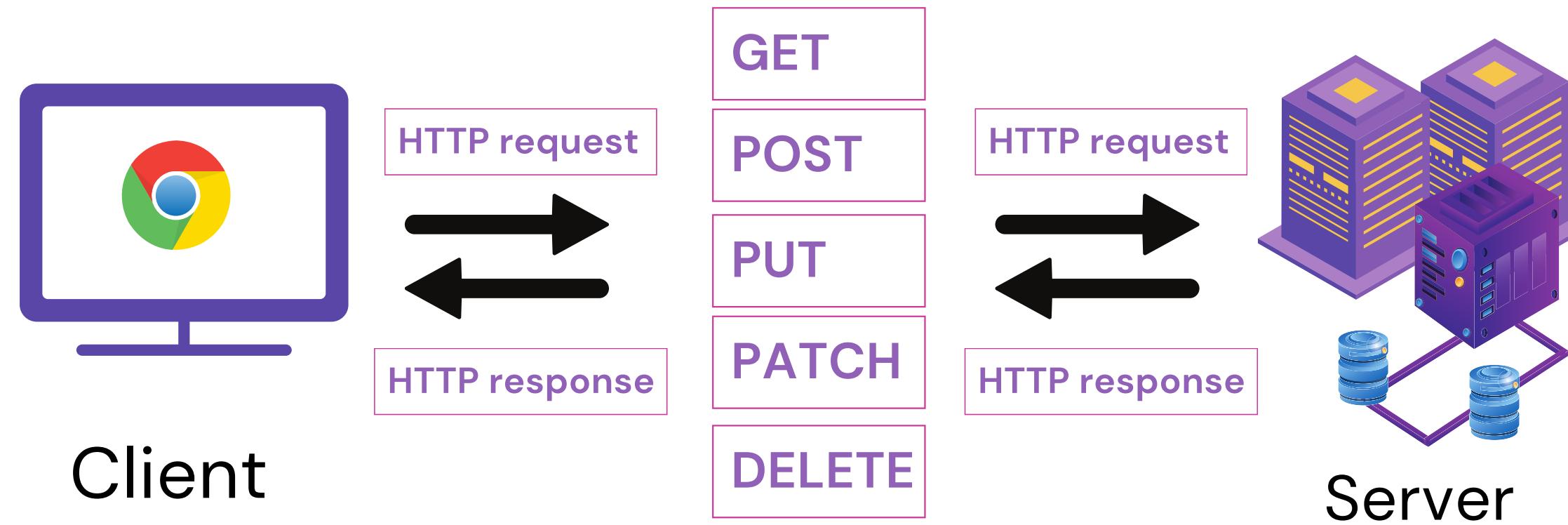
Request e Response



Para o lar

Ai, que canseira





verbo	ação
GET	ler
POST	criar
PUT	substituir
PATCH	modificar
DELETE	excluir

código	tipo de resposta
100-199	informação
200-299	sucesso
300-399	redirecionamento
400-499	erro do cliente
500-599	erro de servidor

Nós, as Devs Back-end



Dev Front

São as pessoas que são responsáveis muito mais do que construir as telas bonitas e funcionais.

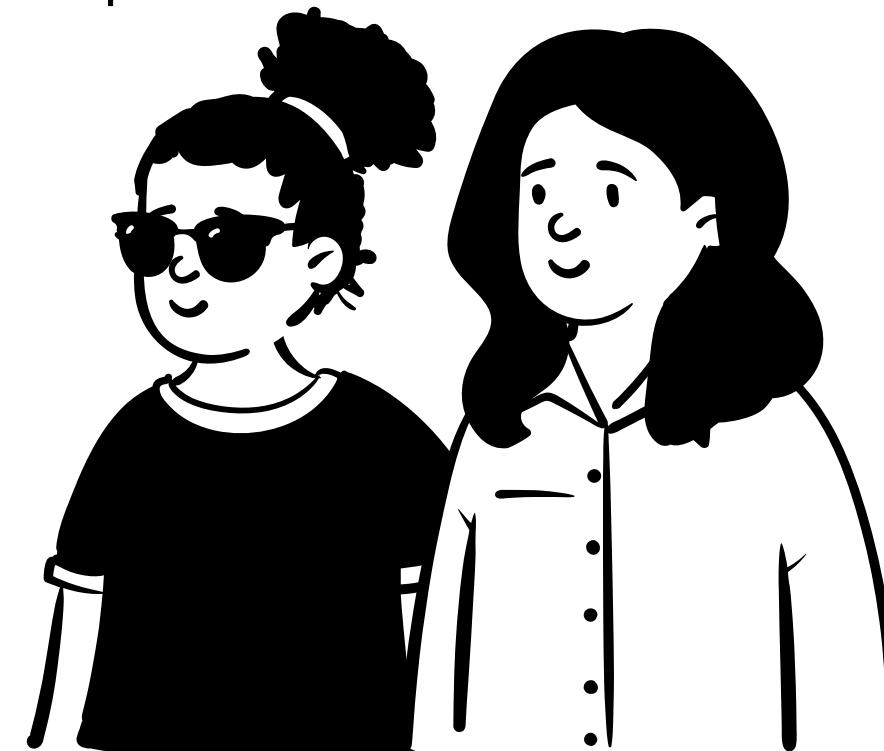
Elas tem que criar aplicações preparadas para enviar Requests corretamente e receber as Responses, também disponibilizar elas para os usuários



Dev Back

Dentro desse fluxo, são as pessoas que constroem toda a dinâmica do recebimento de Requests, o envio das Responses corretas, o tratamento das Responses, as execuções de ação no Banco de Dados e a disponibilização para a Dev Front.

Quando trabalhando construindo APIs somos as construtoras da ponte entre o front-end e o banco de dados



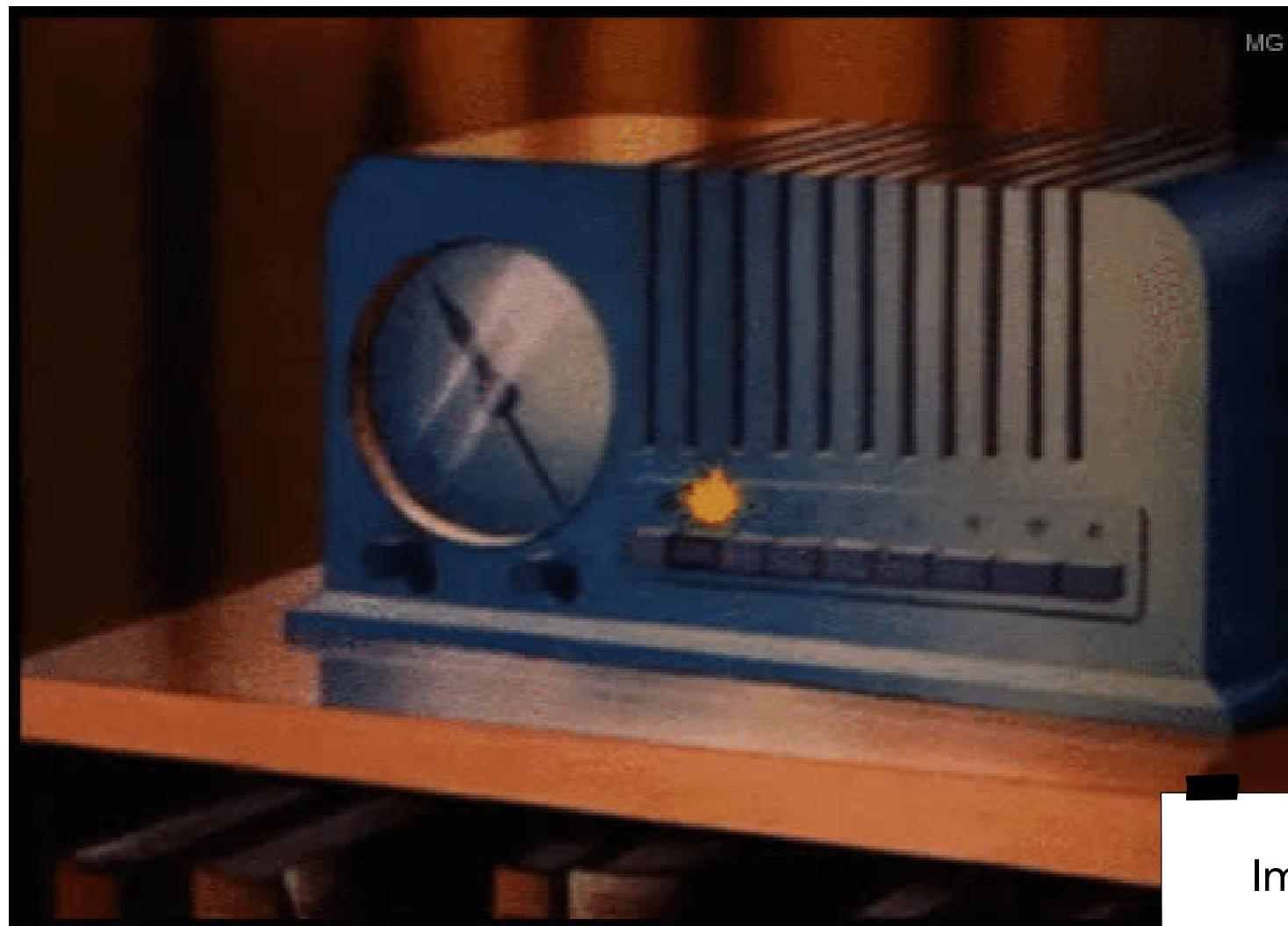
API

tá, mas o que realmente é isso?

Interface de Programação de Aplicativos

Interface

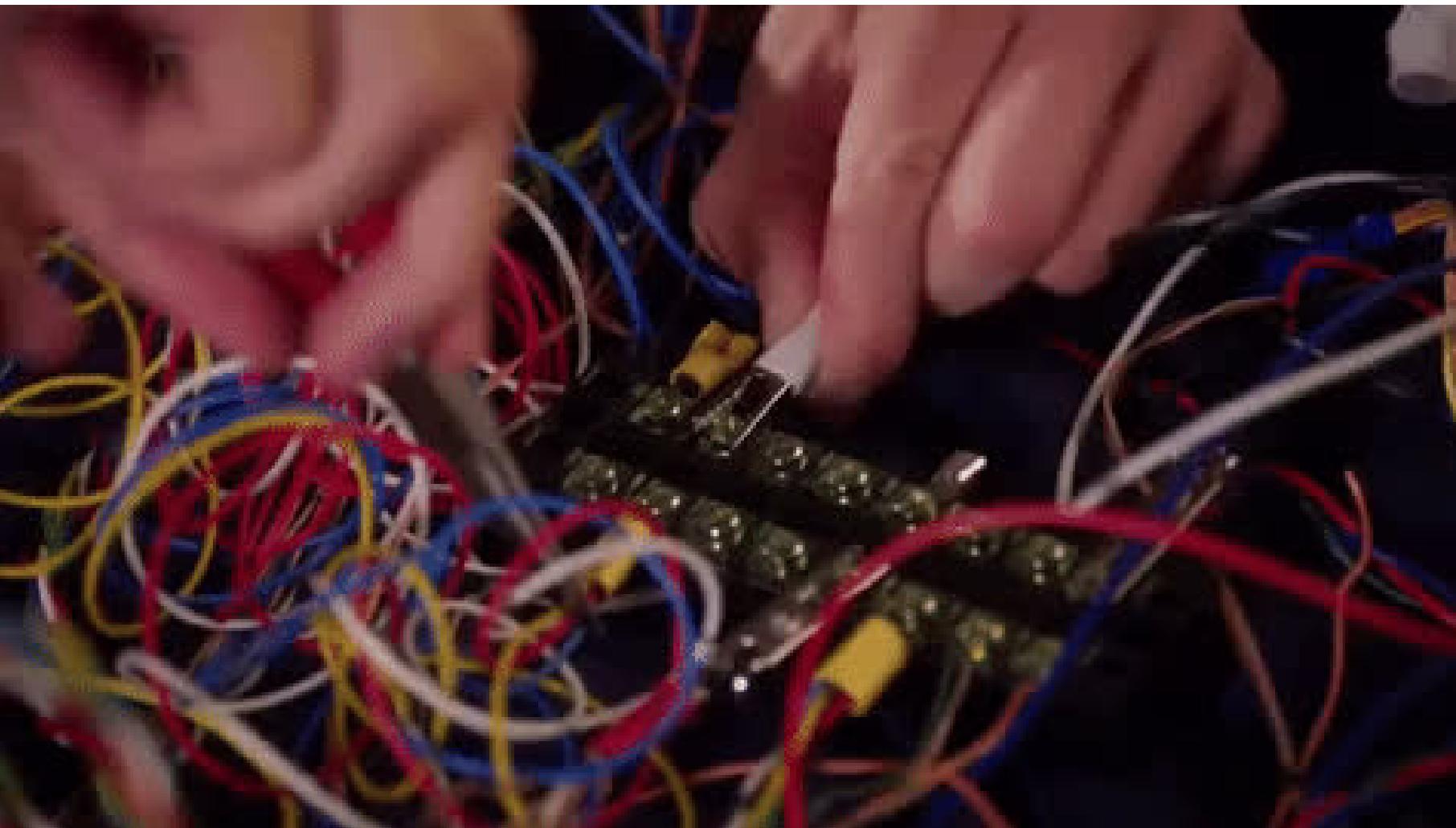
o I da API



Imagine um
rádio

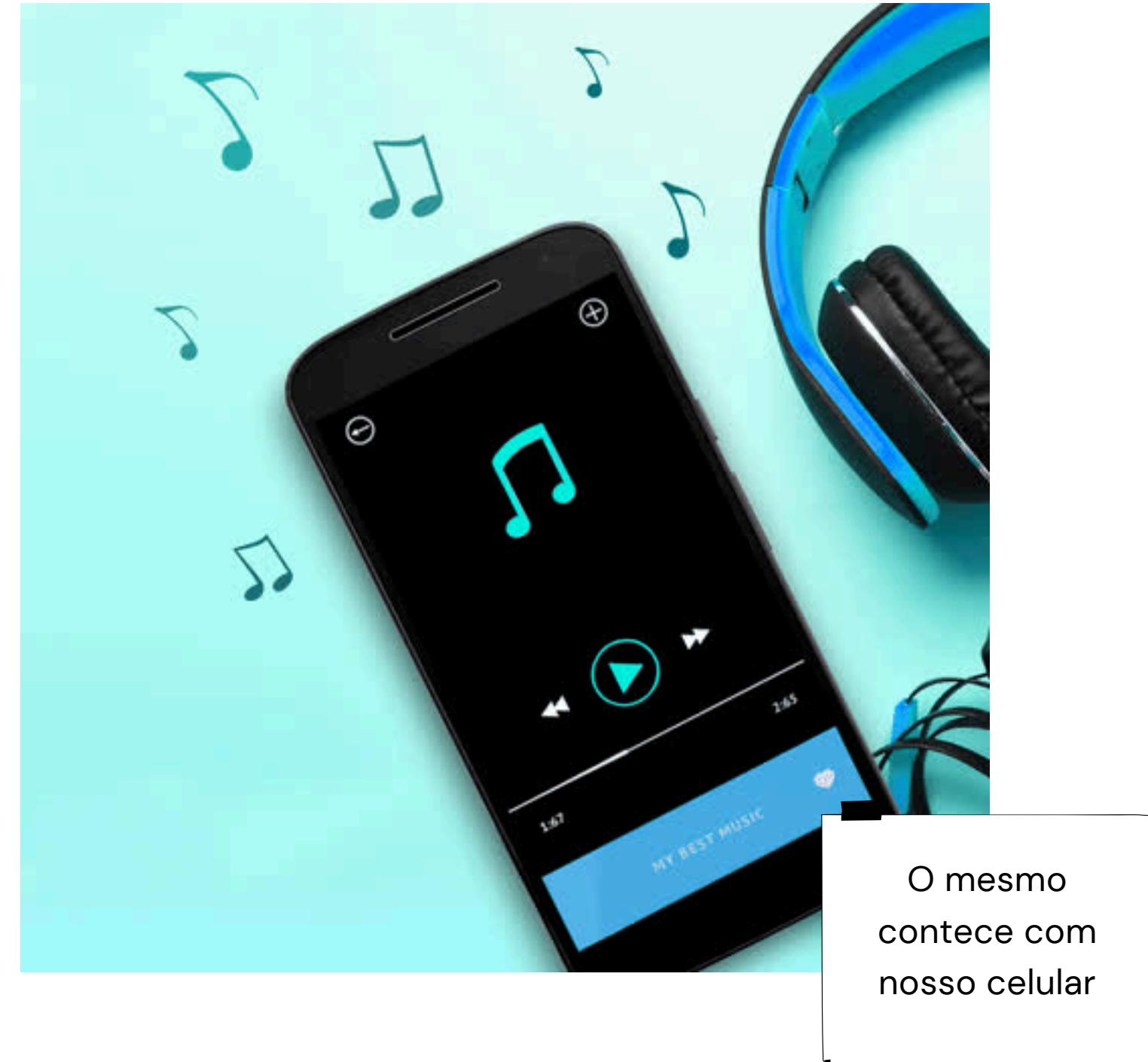
Interface

o I da API



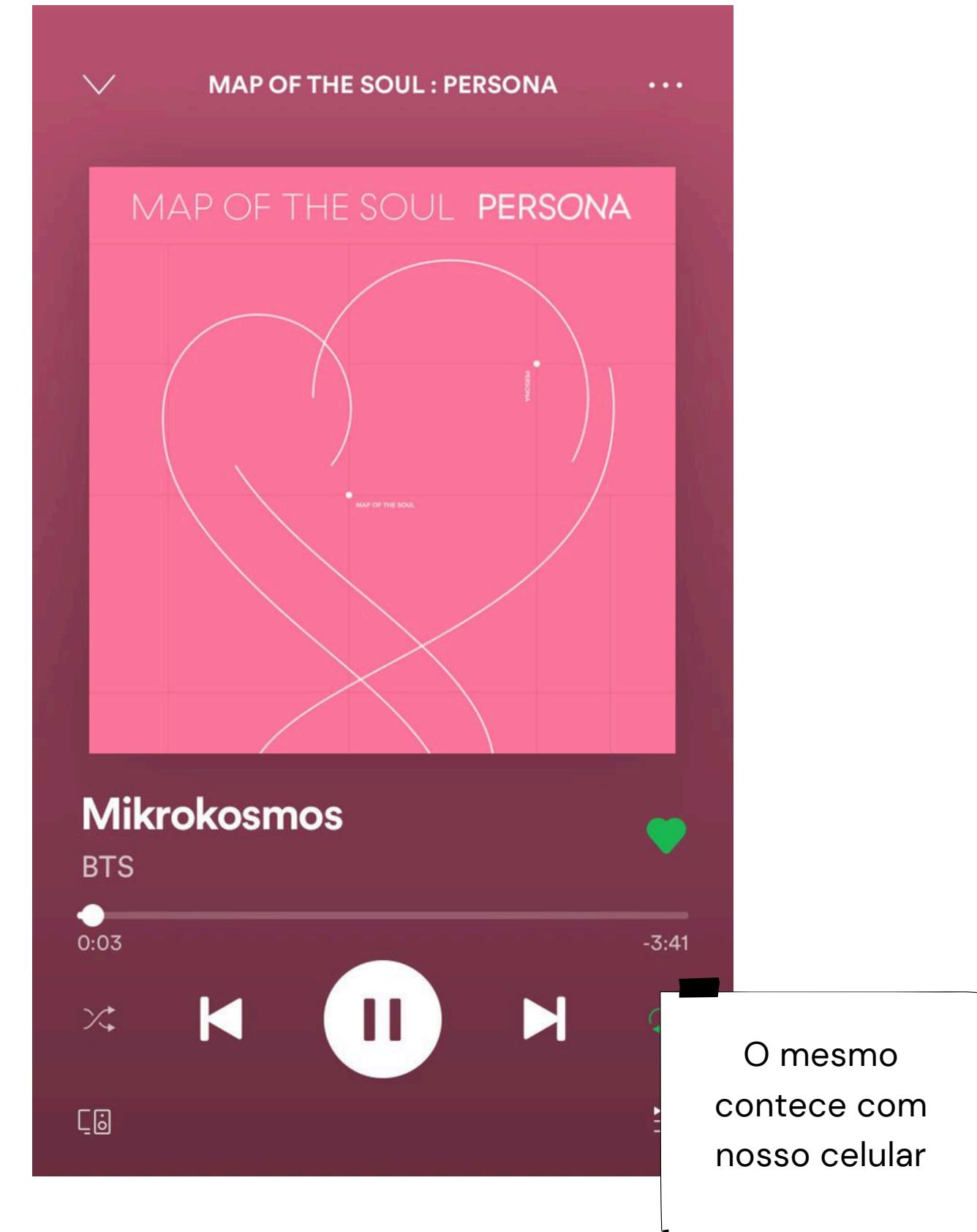
Interface

o I da API



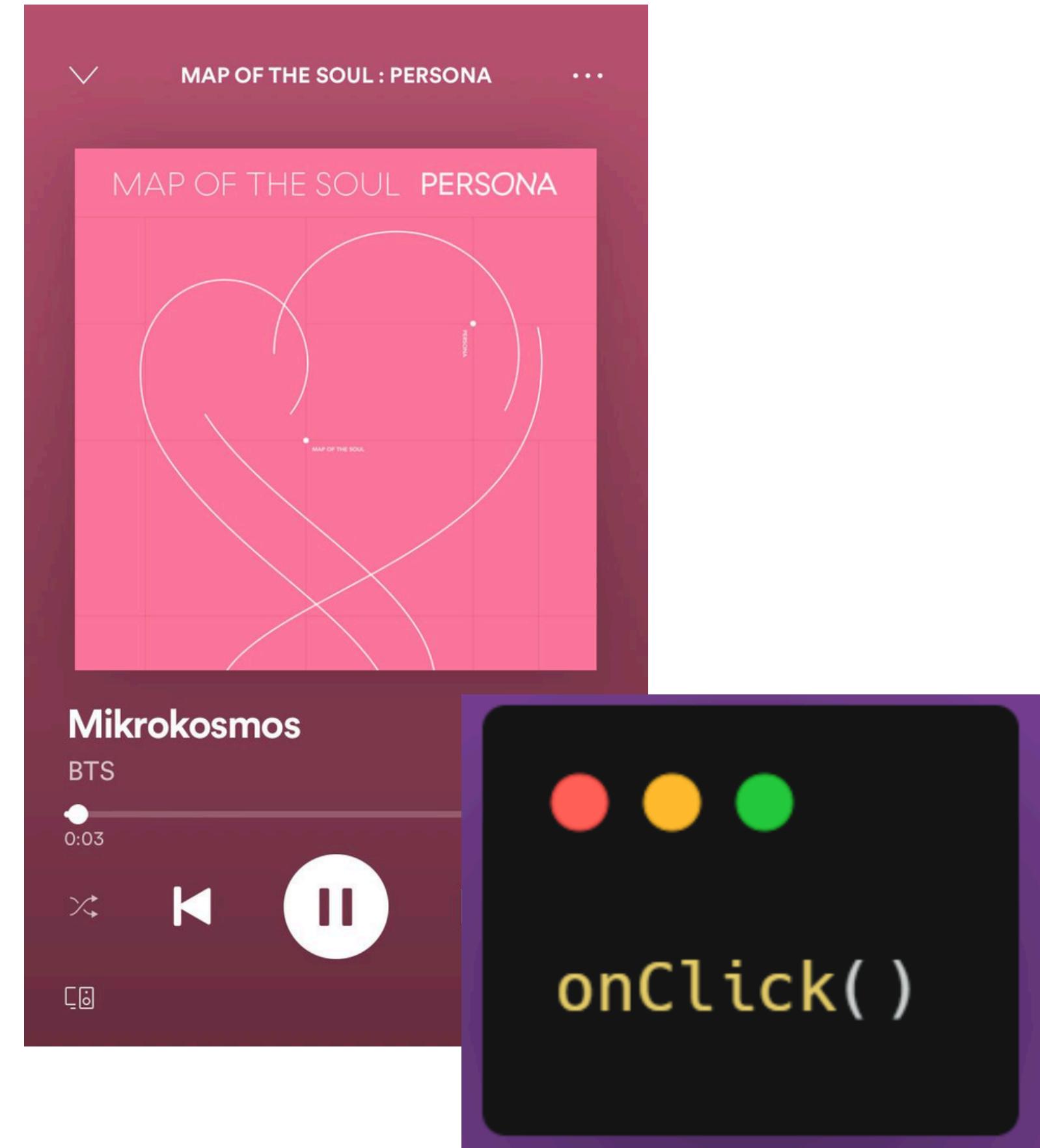
Interface

o I da API



Interface

o I da API



Interface de Programação de Aplicativos

Assim como a interface do rádio, a API busca criar formas e ferramentas de se usar uma funcionalidade ou uma informação sem realmente ter que "reinventar a tal função."

Ela não necessariamente está num link na Web, ela pode ser uma lib ou um framework, uma função já pronta em uma linguagem específica, etc.

Web APIs

São um conjunto de instruções e padrões de programação para acesso a um aplicativo de software. Uma empresa de software lança sua API para o público de modo que outros criadores de software possam desenvolver produtos acionados por esse serviço.



APIs Publicas

São aquelas que são disponibilizadas gratuitamente para desenvolvedoras e usuarios com restrição minima. Podem precisar de cadastro, o uso de API Key ou ser completamente abertas.

Elas estão relacionadas com uso externo de dados ou serviços.



APIs Privadas

São oposto das APIs públicas. Elas estão ligadas à serviços sigilosos, dados sensíveis, transações de empresas privadas, comunicação e ferramentas interna da empresa, etc



APIs REST e RESTfull

Trata-se de um conjunto de princípios e definições necessários para a criação de um projeto com interfaces bem definidas, Rest, que é a abreviatura de **Representational State Transfer**,

é um conjunto de restrições utilizadas para que as requisições HTTP atendam as diretrizes definidas na arquitetura.

Objetivos



Modelo Server/Client



API e Web APIs



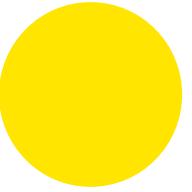
URI, URL, URN, Dominio, IP e DNS



Iniciando com Nest.js



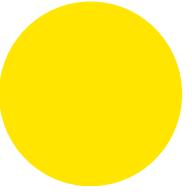
Protocolo HTTP



to-do API



Request e Response

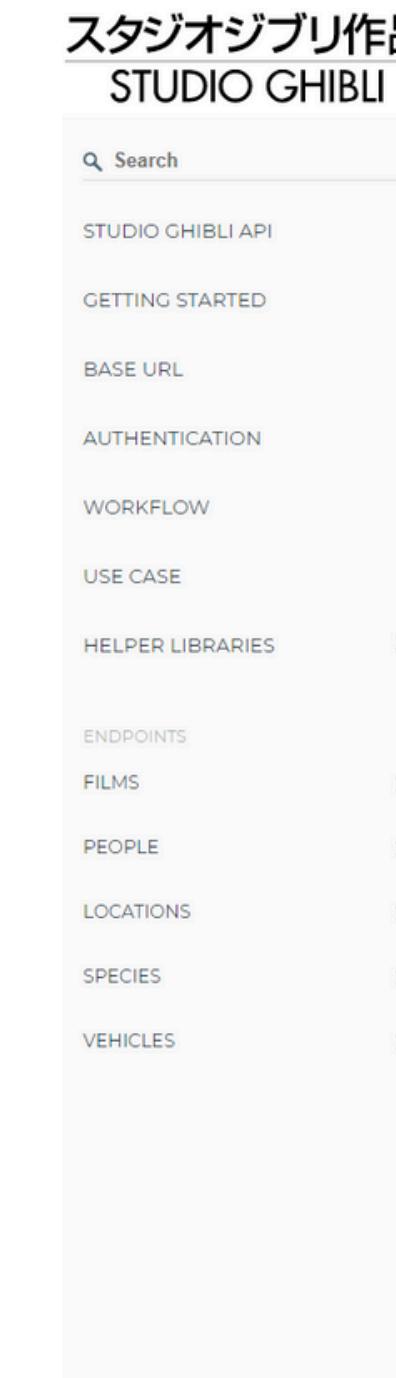


Para o lar

Consumindo API

Documentação

<https://ghibliapi.herokuapp.com/#>



Studio Ghibli API (v1.0.1)

Download OpenAPI specification: [Download](#)

Studio Ghibli API

The Studio Ghibli API catalogs the people, places, and things found in the worlds of Ghibli. It was created to help users discover resources, consume them via HTTP requests, and interact with them in whatever way makes sense. Navigation can be found on the left sidebar, and the right sidebar shows examples of returned objects for successful calls.

Users can raise an issue, ask for help, or find a contribution guide at the main repo: <https://github.com/janaipakos/ghibliapi>

Getting Started

Requests can be made with `curl` or other helper libraries by following regular REST calls. For example, here is how to GET the resource for the film *My Neighbor Totoro*:

```
curl https://ghibliapi.herokuapp.com/films/58611129-2dbc-4a81-a72f-77ddfclb1b49
```

Calling this resource will respond with the following object:

```
{
  "id": "58611129-2dbc-4a81-a72f-77ddfclb1b49",
  "title": "My Neighbor Totoro",
  "description": "Two sisters move to the country with their father in order to be closer to their hospitali",
  "director": "Hayao Miyazaki",
  "producer": "Hayao Miyazaki",
  "release_date": "1988",
  "rt_score": "93",
  ...
}
```

Consumindo API

curl

Docs Overview Project Protocols Releases Tool Who and Why

curl / Docs / Tool Documentation / Man Page

curl.1 the man page

NAME	Related:
curl - transfer a URL	File a bug about this man page
SYNOPSIS	Manual
curl [options / URLs]	FAQ
DESCRIPTION	HTTP Scripting

curl is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, MQTT, POP3, POP3S, RTMP, RTMPS, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP). The command is designed to work without user interaction.

curl offers a busload of useful tricks like proxy support, user authentication, FTP upload, HTTP post, SSL connections, cookies, file transfer resume, Metalink, and more. As you will see below, the number of features will make your head spin!

curl is powered by libcurl for all transfer-related features. See *libcurl(3)* for details.

URL

The URL syntax is protocol-dependent. You'll find a detailed description in [RFC 3986](#).

You can specify multiple URLs or parts of URLs by writing part sets within braces and quoting the URL as in:

`"http://site.{one,two,three}.com"`

Consumindo API

Consumindo com curl



Requests podem ser feitas com `curl` ou outras libs que seguem o protocolo REST.
Por exemplo, aqui tem como fazer um GET para o filme Me Visinho Totoro:

```
curl https://ghibliapi.herokuapp.com/films/58611129-2dbc-4a81-a72f-77ddfc1b1b49
```

```
$ curl GET https://ghibliapi.herokuapp.com/films/58611129-2dbc-4a81-a72f-77ddfc1b1b49
% Total    % Received % Xferd  Average Speed   Time   Time   Current
          Dload  Upload   Total Spent  Left  Speed
0       0     0      0      0      0      0      0 --:--:-- 0:00:01 --:--:-- 0curl: (6) Could not
resolve host: GET
% Total    % Received % Xferd  Average Speed   Time   Time   Current
          Dload  Upload   Total Spent  Left  Speed
100  1868  100  1868      0      0  2587      0 --:--:-- --:--:-- 2587{
  "id": "58611129-2dbc-4a81-a72f-77ddfc1b1b49",
  "title": "My Neighbor Totoro",
  "description": "Two sisters move to the country with their father in order to be closer to their hospitalized mother, and discover the surrounding trees are inhabited by Totoros, magical spirits of the forest. When the youngest runs away from home, the older sister seeks help from the spirits to find her.",
  "director": "Hayao Miyazaki",
  "producer": "Hayao Miyazaki",
  "release_date": "1988",
  "rt_score": "93",
  "people": [
    "https://ghibliapi.herokuapp.com/people/986faac6-67e3-4fb8-a9ee-bad077c2e7fe",
    "https://ghibliapi.herokuapp.com/people/d5df3c04-f355-4038-833c-83bd3502b6b9",
    "https://ghibliapi.herokuapp.com/people/3031caa8-eb1a-41c6-ab93-dd091b541e11",
    "https://ghibliapi.herokuapp.com/people/87b68b97-3774-495b-bf80-495a5f3e672d",
    "https://ghibliapi.herokuapp.com/people/d39deecb-2bd0-4770-8b45-485f26e1381f",
    "https://ghibliapi.herokuapp.com/people/591524bc-04fe-4e60-8d61-2425e42ffb2a",
    "https://ghibliapi.herokuapp.com/people/c491755a-407d-4d6e-b58a-240ec78b5061",
    "https://ghibliapi.herokuapp.com/people/f467e18e-3694-409f-bdb3-be891ade1106",
    "https://ghibliapi.herokuapp.com/people/08fffbce4-7f94-476a-95bc-76d3c3969c19",
    "https://ghibliapi.herokuapp.com/people/0f8ef701-b4c7-4f15-bd15-368c7fe38d0a"
  ],
  "species": [
    "https://ghibliapi.herokuapp.com/species/af3910a6-429f-4c74-9ad5-dfe1c4aa04f2",
    "https://ghibliapi.herokuapp.com/species/603428ba-8a86-4b0b-a9f1-65df6abef3d3",
    "https://ghibliapi.herokuapp.com/species/74b7f547-1577-4430-806c-c358c8b6bcf5"
  ],
  "locations": [
    "https://ghibliapi.herokuapp.com/locations/"
  ],
  "vehicles": [
    "https://ghibliapi.herokuapp.com/vehicles/"
  ],
  "url": "https://ghibliapi.herokuapp.com/films/58611129-2dbc-4a81-a72f-77ddfc1b1b49",
  "length": null
}
```

Consumindo API

Consumindo como um Front

The screenshot shows a user interface for consuming a movie API. At the top is the Studio Ghibli logo. Below it are three movie cards:

- Castle in the Sky**: A blue card with white text. Description: "The orphan Sheeta inherited a mysterious crystal that links her to the mythical sky-kingdom of Laputa. With the help of resourceful Pazu and a rollicking band of sky pirates, she makes her way to the ruins of the once-great civilization. Sheeta and Pazu must outwit the evil Muska, who plans to use Laputa's science to make himself ruler of the world."
- Grave of the Fireflies**: A green card with white text. Description: "In the latter part of World War II, a boy and his sister, orphaned when their mother is killed in the firebombing of Tokyo, are left to survive on their own in what remains of civilian life in Japan. The plot follows this boy and his sister as they do their best to survive in the Japanese countryside, battling hunger, prejudice, and pride in their own quiet, personal battle."
- My Neighbor Totoro**: A pink card with white text. Description: "Two sisters move to the country with their father in order to be closer to their hospitalized mother, and discover the surrounding trees are inhabited by Totoros, magical spirits of the forest. When the youngest runs away from home, the older sister seeks help from the spirits to find her."

```
const app = document.getElementById('root');
const container = document.createElement('div');
container.setAttribute('class', 'container');

app.appendChild(container);

const request = new XMLHttpRequest();

request.open('GET', 'https://ghibliapi.herokuapp.com/films', true);

request.onload = function(){
  const data = JSON.parse(this.response);

  console.log(data)
  console.log(data[0].title)

  if(request.status >= 200 && request.status < 400){
    console.log("SUCESSO!!!")
    data.forEach(movie => {
      const card = document.createElement('div');
      card.setAttribute('class', 'card');

      const h1 = document.createElement('h1');
      h1.textContent = movie.title;

      const p = document.createElement('p');
      movie.description = movie.description;
      p.textContent = movie.description;

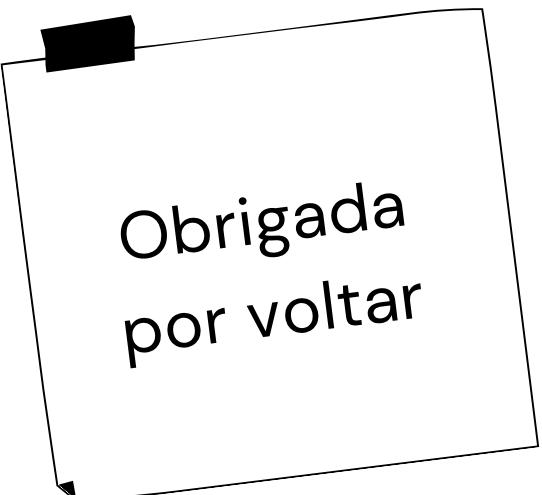
      container.appendChild(card);
      card.appendChild(h1);
      card.appendChild(p);
    });
  }else{
    console.log("erro :c")
  }
}

request.send()
```

Ai, que fome...



**Nossa, você aqui
de novo?**



API

Interface de Programação de Aplicativos

API busca criar formas e ferramentas de se usar uma funcionalidade ou uma informação sem realmente ter que "reinventar a tal função."

Ela não necessariamente está num link na Web, ela pode ser uma lib ou um framework, uma função já pronta em uma linguagem específica, etc.

Web API e API REST

Web API é uma interface que é disponibilizada de forma remota, pela web, que possibilita a programação aplicativos e softwares.

E as APIs RESTfull são aquelas que são capazes de fazer o REST. Que nada mais é uma API que usa os protocolos HTTP para comunicação entre o usuário e o servidor.

Objetivos



Modelo Server/Client



API e Web APIs



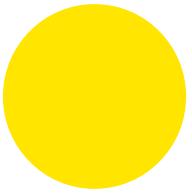
URI, URL, URN, Dominio, IP e DNS



Iniciando com Nest.js



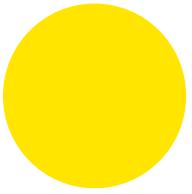
Protocolo HTTP



to-do API



Request e Response

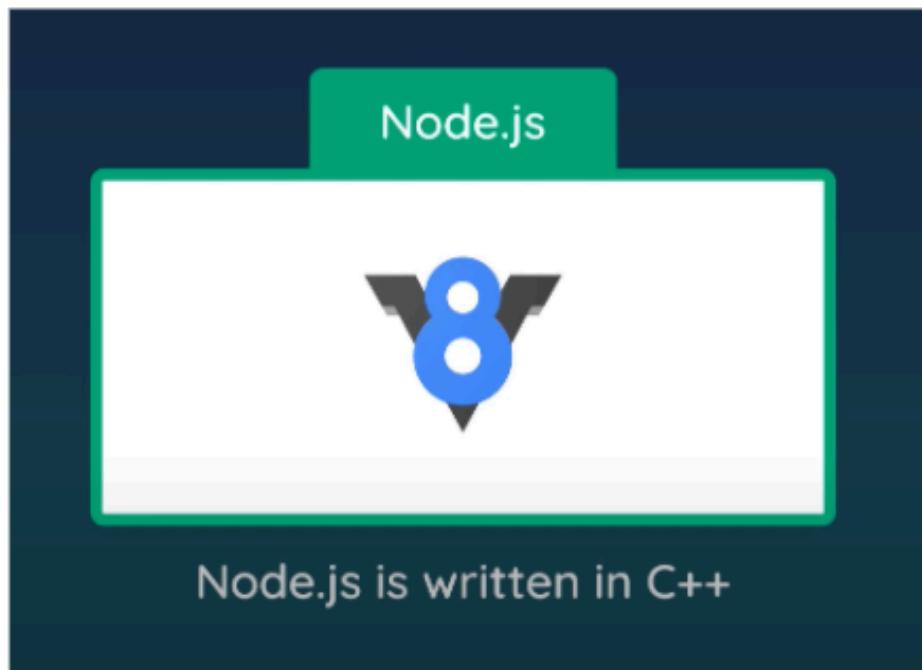


Para o lar

Node.js

Antes de tudo, um pouco de história!

Tudo começou em 2009...



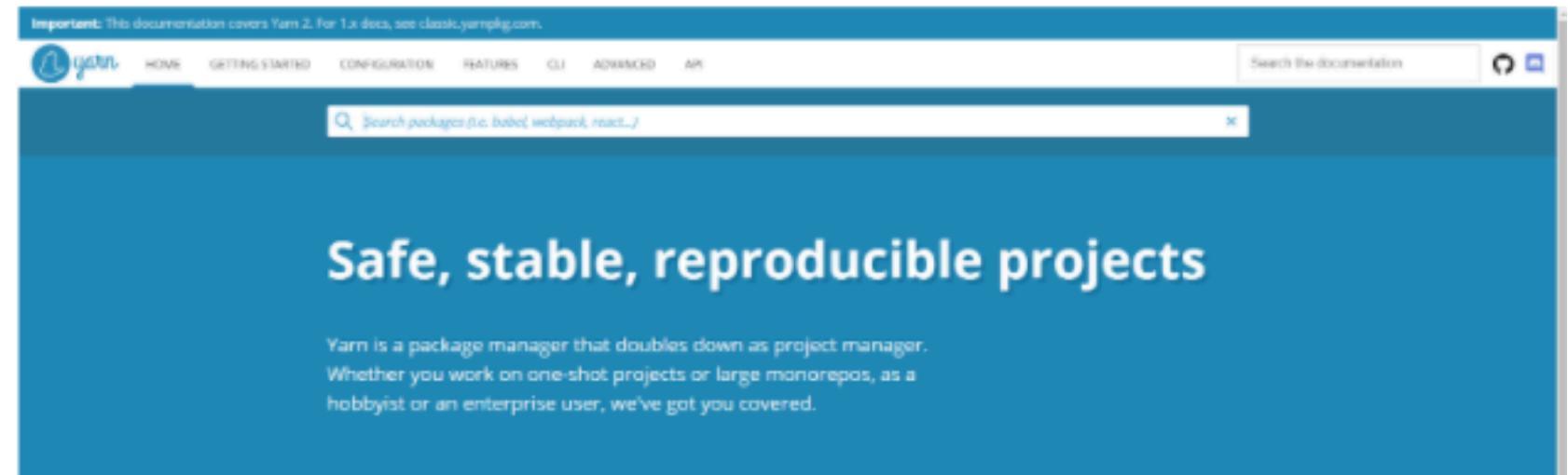
Interpretador JavaScript que não precisa de navegador.

Ele pode:

- Ler e escrever arquivos no seu computador
- Conectar com um banco de dados
- Se comportar como um servidor

Node.js

e os gerenciadores de pacotes



npm X yarn

e a guerra dos 100 anos...



npm install

yarn



NPM



NPM!!!



YARN!



YARN!!!!

NestJS

Framework para construção de aplicações
Node.js construído em TypeScript



Nest JS

- Combina elementos de Programação Orientada a Objetos (POO), Programação Funcional (PF) e Programação Funcional Reativa (PRF)
- Fornece uma arquitetura de aplicação pronta para uso
- A arquitetura é fortemente inspirada no Angular

NestJS

Framework para construção de aplicações
Node.js construído em TypeScript



Nest JS

```
npm install -g @nestjs/cli
```

NestJS

Framework para construção de aplicações
Node.js construído em TypeScript



Nest JS

```
npm install -g @nestjs/cli
```

```
nest new ts-todo-api --package-manager npm
```

GET

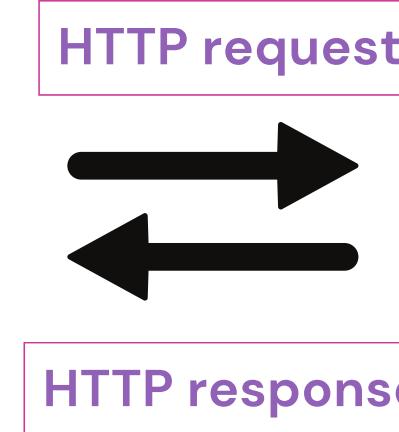
Dentro do CRUD o método GET é representado pela letra R

👁️ R: Read (ler) - exibir as informações de um registro

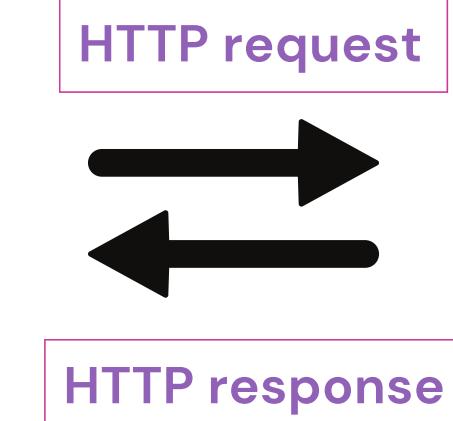
O Client manda um **request** solicitando realizar um GET e o Server deve estar preparado para receber esse GET e responde-lo com um **response**.



Client



- GET todos
- GET por id
- GET por nome



Server

Parâmetros

Path params

- são aqueles que são adicionados diretamente na URL
- "/rota/:id"
- `request.params.id`
- bons, porém limitantes, por exemplo, se quisermos filtrar por uma string

Query params

- são aqueles que são adicionados a chave e o valor desejados
- "/rota?id=1234"
- `request.query.id`
- a forma mais efetiva de fazer requests com strings ou diversos valores

[https://www.canva.com/design/
DAGlrLRu1JM/jOUQcuwiQYBFxj4
emYz9Mw/edit](https://www.canva.com/design/DAGlrLRu1JM/jOUQcuwiQYBFxj4emYz9Mw/edit)

**[https://www.google.com/search?
q=spring+web&q=java](https://www.google.com/search?q=spring+web&q=java)**

POST

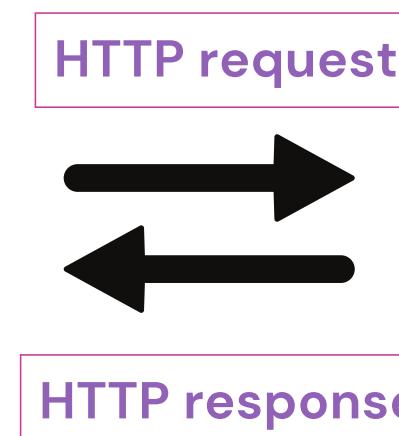
Dentro do CRUD o método POST é representado pela letra C

C: Create (criar) – criar um novo registro

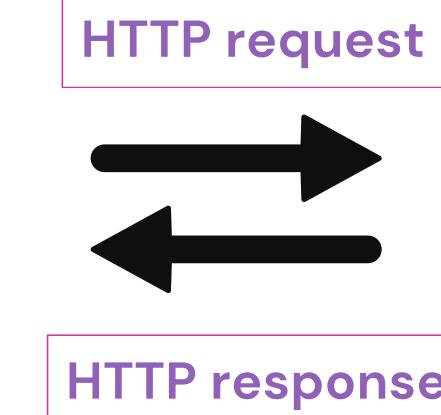
O Client manda um **request** solicitando realizar um POST e envia no **Body** o registro que quer cadastrar, o Server deve estar preparado para receber esse POST, incluir o novo item solicitado e responde-lo com um **response**.



Client



POST



Server

Body

- são usados nos métodos POST, PATCH e PUT
- enviam dados a serem cadastrados no banco de dados
- request.body



```
... "descricao": "exemplo · de · Body",  
... "nomeColaborador": "Ana"  
...
```

DELETE

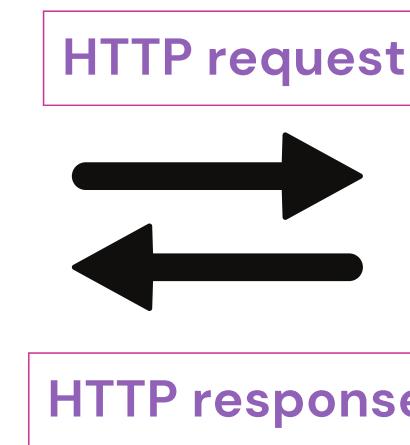
Dentro do CRUD o método DELETE é representado pela letra D

✗ D: Delete (apagar) – apagar um registro

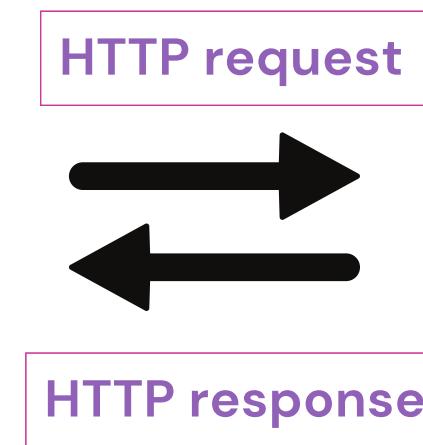
O Client manda um **request** solicitando realizar um DELETE e o Server deve estar preparado para receber esse DELETE e responde-lo com um **response**.



Client

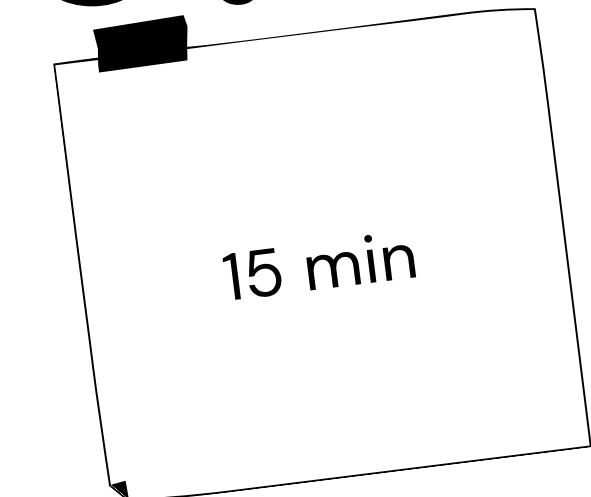


DELETE por id



Server

**Ai... vamo para
um pouquinho?**



Parâmetros

Tanto o body quanto o query e o path são parâmetros enviados na requisição e podem ser acessados pelo servidor afim de definir a requisição e as ações.



request.params

usado para pesquisa simples, enviado diretamente na rota

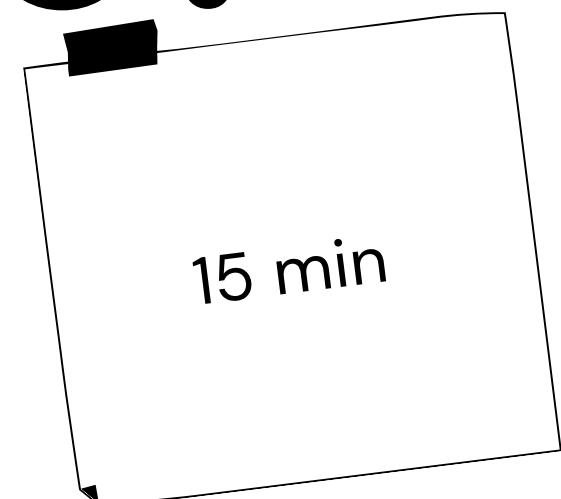
request.query

usado para pesquisa de uma ou multipas strings

request.body

usado para enviar dados que serão cadastrados no banco, podem ser combinados com o query ou o path params

**Ai... vamo para
um pouquinho?**



HTTP - DELETE

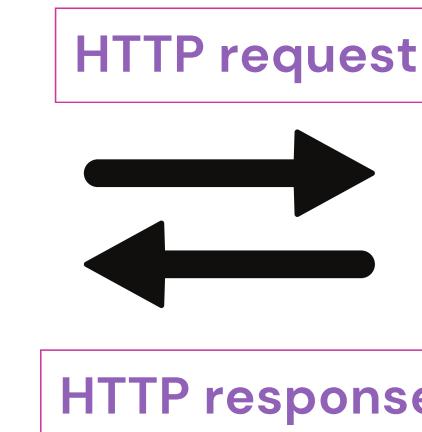
Dentro do CRUD o método DELETE é representado pela letra D

✗ D: Delete (apagar) - apagar um registro

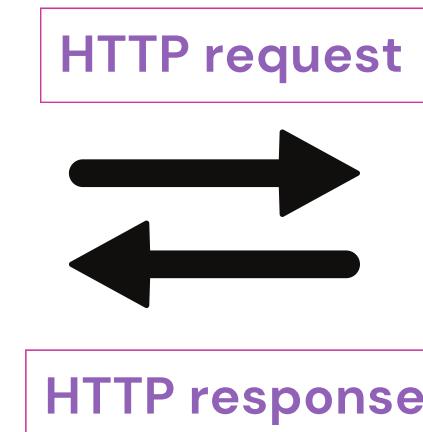
O Client manda um **request** solicitando realizar um DELETE e o Server deve estar preparado para receber esse DELETE e responde-lo com um **response**.



Client



DELETE por id



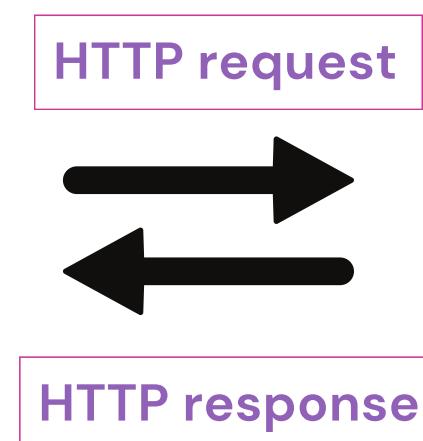
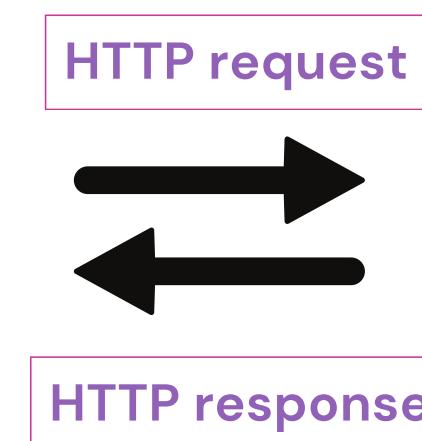
Server

HTTP - PUT & PATCH

♻️ U: Update (atualizar) – atualizar os dados do registro



Client



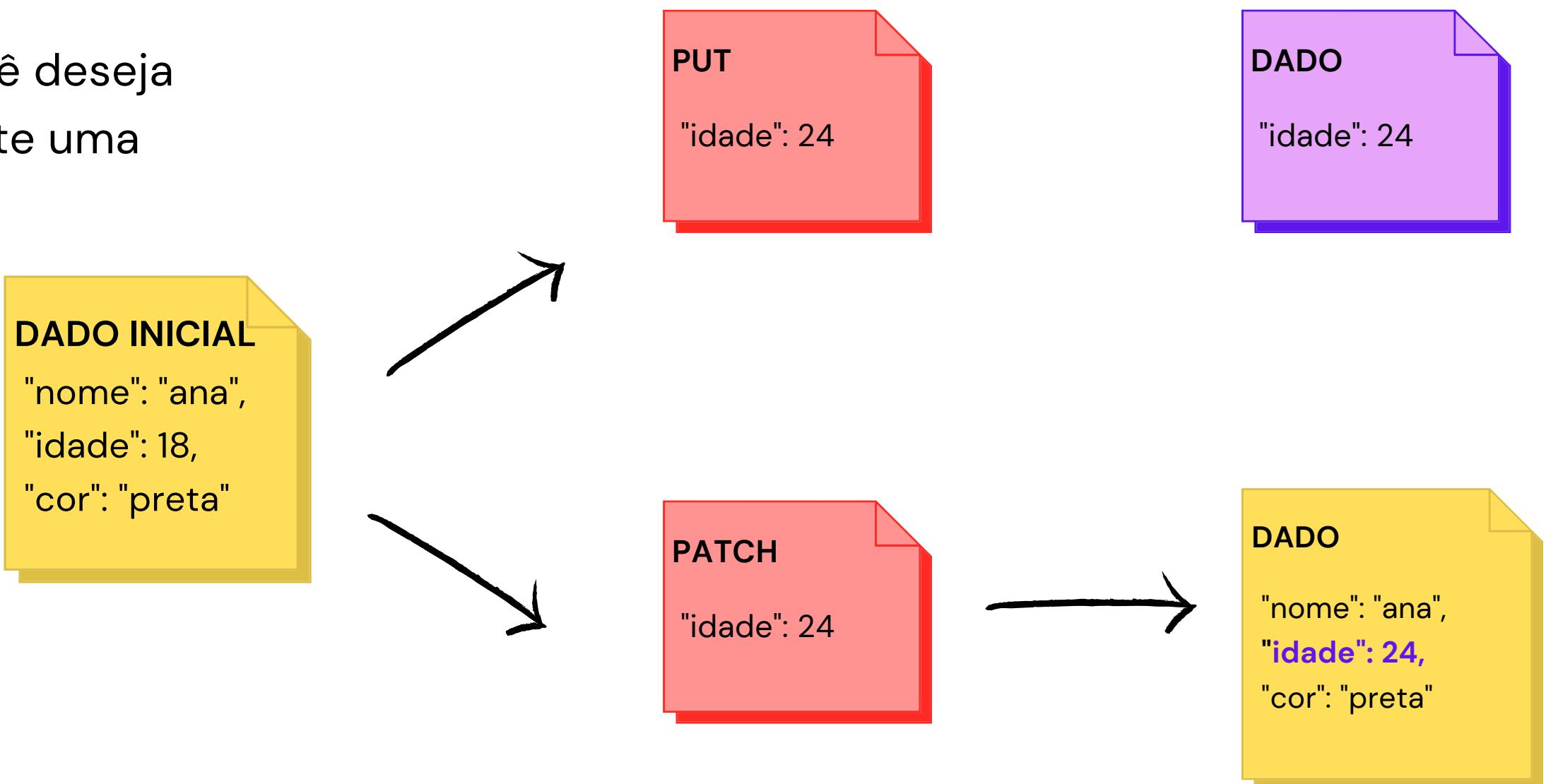
Server

PUT x PATCH

É tudo a mesma coisa?

NÃO!

O PUT substitui todo o objeto que você deseja modificar, já o PATCH modifica somente uma propriedade dentro do seu objeto.

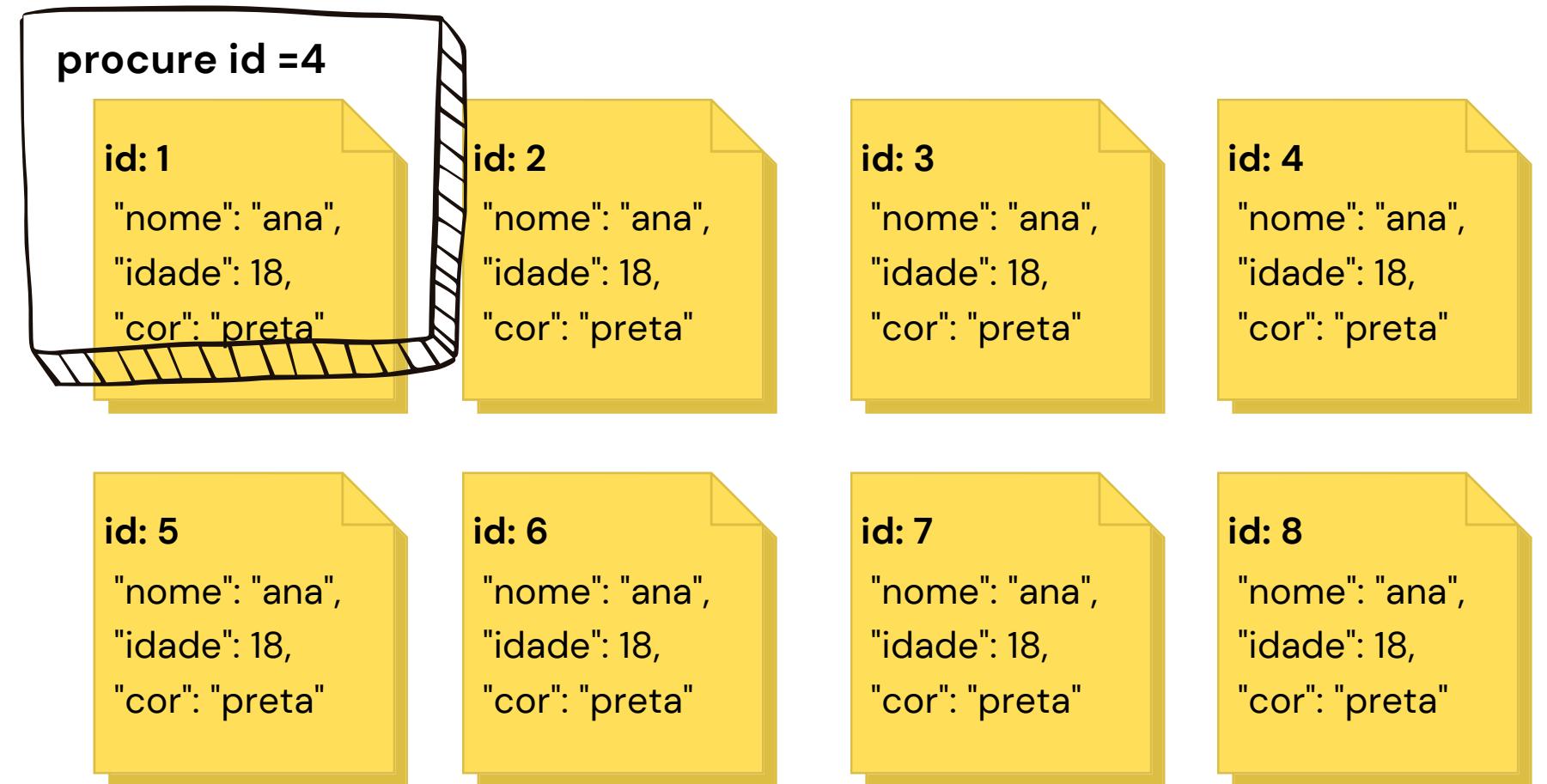


PUT x PATCH

Mas então por que ainda usamos o PUT?

Muitas vezes ainda usamos o PUT pela performance que ele tem quando relacionado o banco de dados. Substituir um dado inteiro é mais rápido do que somente uma propriedade dele.

Por exemplo, vamos simular a uma edição do campo `idade` no dado de `id=4`

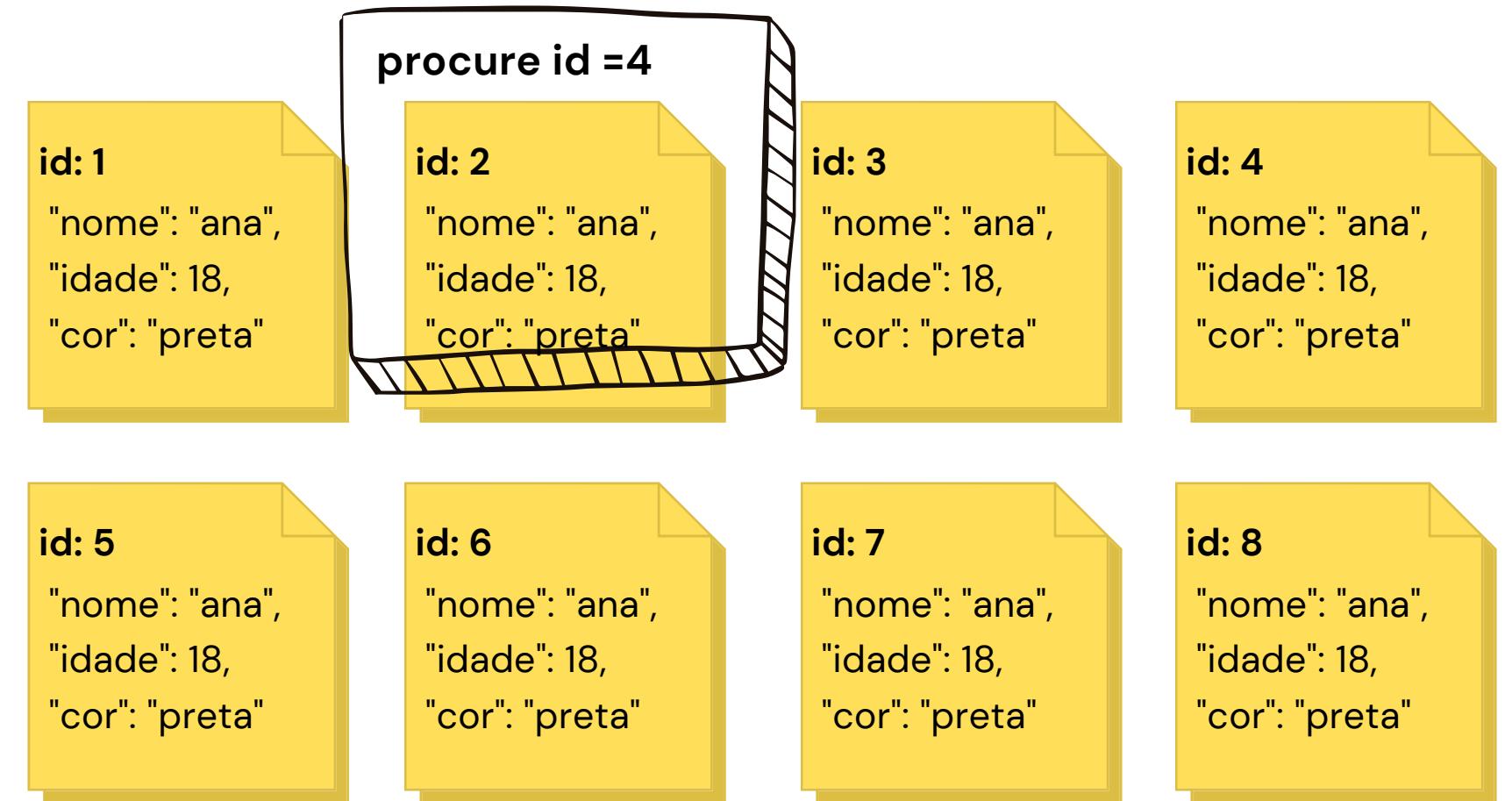


PUT x PATCH

Mas então por que ainda usamos o PUT?

Por exemplo, vamos simular a uma edição do campo `idade` no dado de `id=4`.

No banco de dados nosso programa tem que percorrer pela memória procurando pelo id que queremos



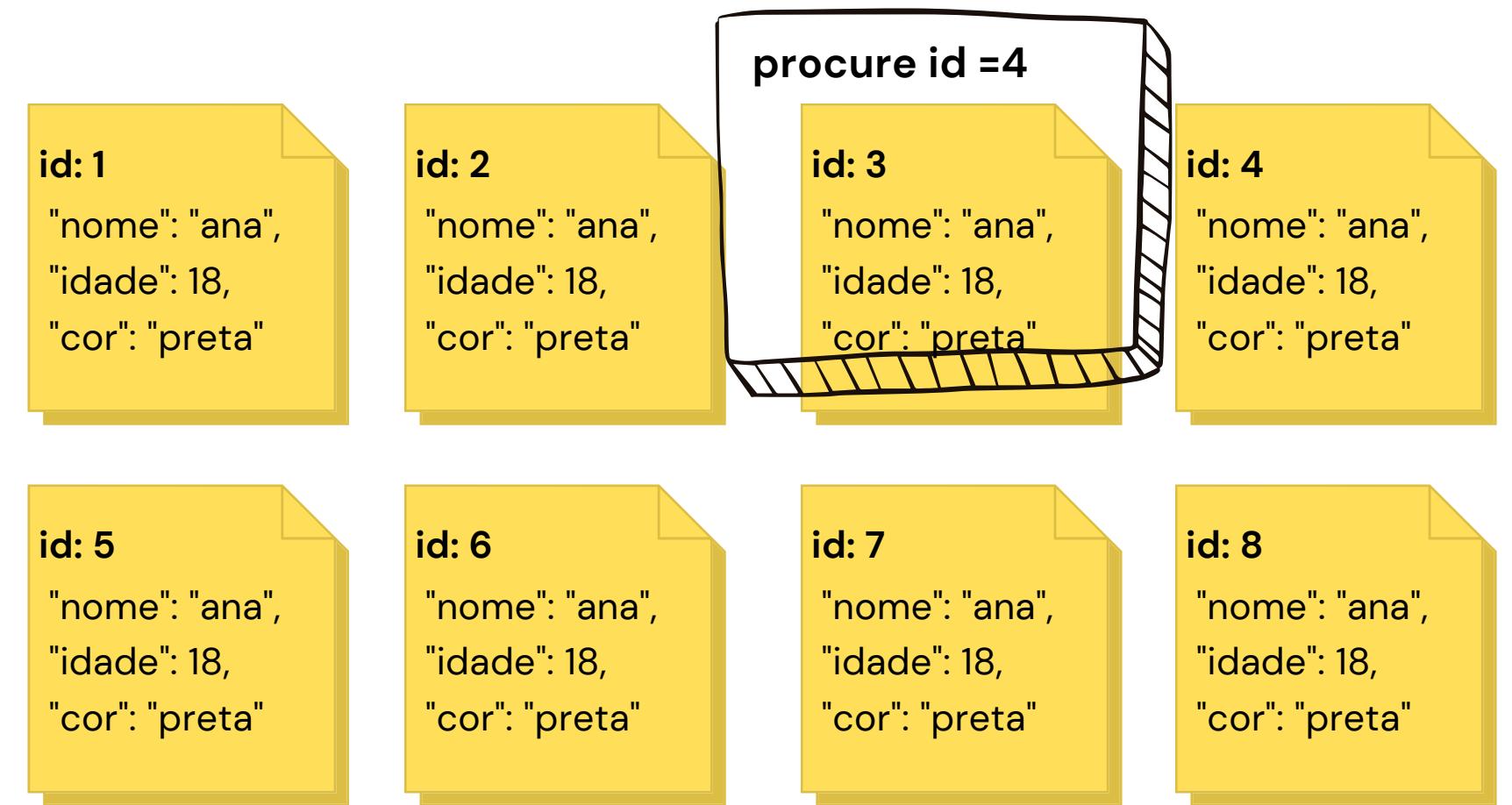
PUT x PATCH

Mas então por que ainda usamos o PUT?

Por exemplo, vamos simular a uma edição do campo `idade` no dado de `id=4`.

No banco de dados nosso programa tem que percorrer pela memória procurando pelo id que queremos.

Ele procura somente pelo índice que indicamos.



PUT x PATCH

Mas então por que ainda usamos o PUT?

Se estivéssemos escolhido um método PUT, a procura pararia aqui e o dado seria substituído por inteiro!

id: 1
"nome": "ana",
"idade": 18,
"cor": "preta"

id: 2
"nome": "ana",
"idade": 18,
"cor": "preta"

id: 3
"nome": "ana",
"idade": 18,
"cor": "preta"

procure id =4
id: 4
"nome": "ana",
"idade": 24,
"cor": "preta"

id: 5
"nome": "ana",
"idade": 18,
"cor": "preta"

id: 6
"nome": "ana",
"idade": 18,
"cor": "preta"

id: 7
"nome": "ana",
"idade": 18,
"cor": "preta"

id: 8
"nome": "ana",
"idade": 18,
"cor": "preta"

PUT x PATCH

Mas então por que ainda usamos o PUT?

Porém estivéssemos escolhido um método **PATCH**, a procura continuaria, agora dentro do dado

procure id =4

id: 1
"nome": "ana"
"idade": 18,
"cor": "preta"

PUT x PATCH

Mas então por que ainda usamos o PUT?

Porém estivéssemos escolhido um método **PATCH**, a procura continuaria, agora dentro do dado. Quando encontrado a propriedade ai sim o dado seria modificado.

Tudo isso seriam frações de segundos para um computador, mas se tivéssemos dezenas de milhares de dados sendo modificados o tempo todo, como uma rede social, por exemplo, isso poderia causar certa lentidão no banco de dados

procure id =4

id: 1

"nome": "ana",
idade
"idade": 24,
"cor": "preta"