

PUT & PATCH



pela ultima vez, Analu

Sou desenvolvedora Full-Stack.

Fui da primeira turma de Back-End da Reprograma

Hoje trabalho como Engenheira de Software no Banco

Itaú. E sou professora de desenvolvimento web.

Email: sampaioaanaluiza@gmail.com



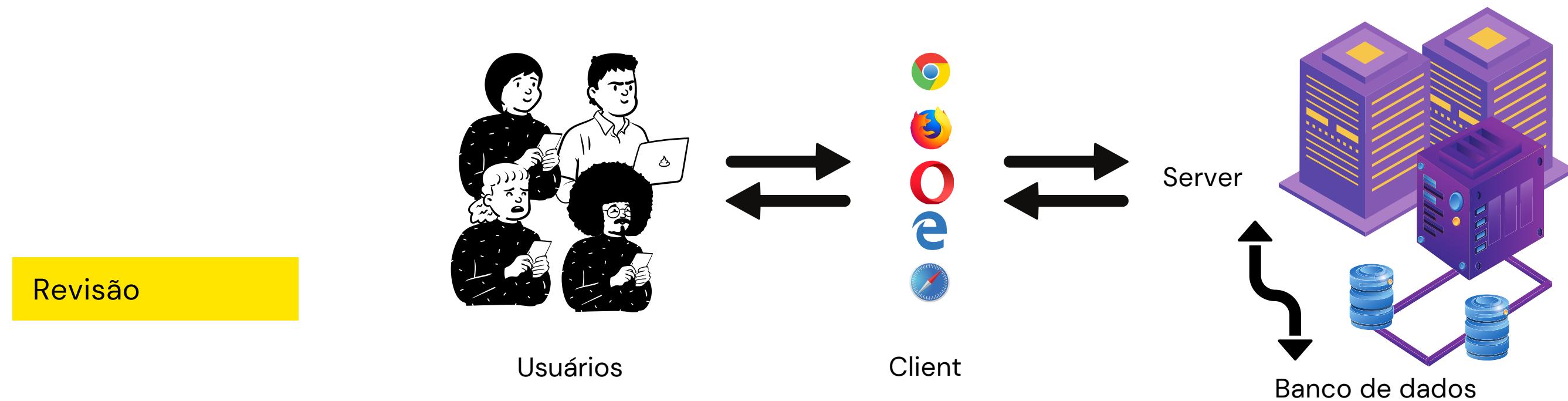
Modelo Server-Client

CLIENTE

Entenda cliente como a interface que o usuário interage. É o Cliente que **solicita** serviços e informações de um ou mais servidores.

SERVIDOR

E o Servidor é o responsável pelo processo, organização e gerenciamento das informações. É ele que **responde** às solicitações feitas pelo usuário. Ele é um processo reativo, disparado pela chegada de pedidos de seus clientes.

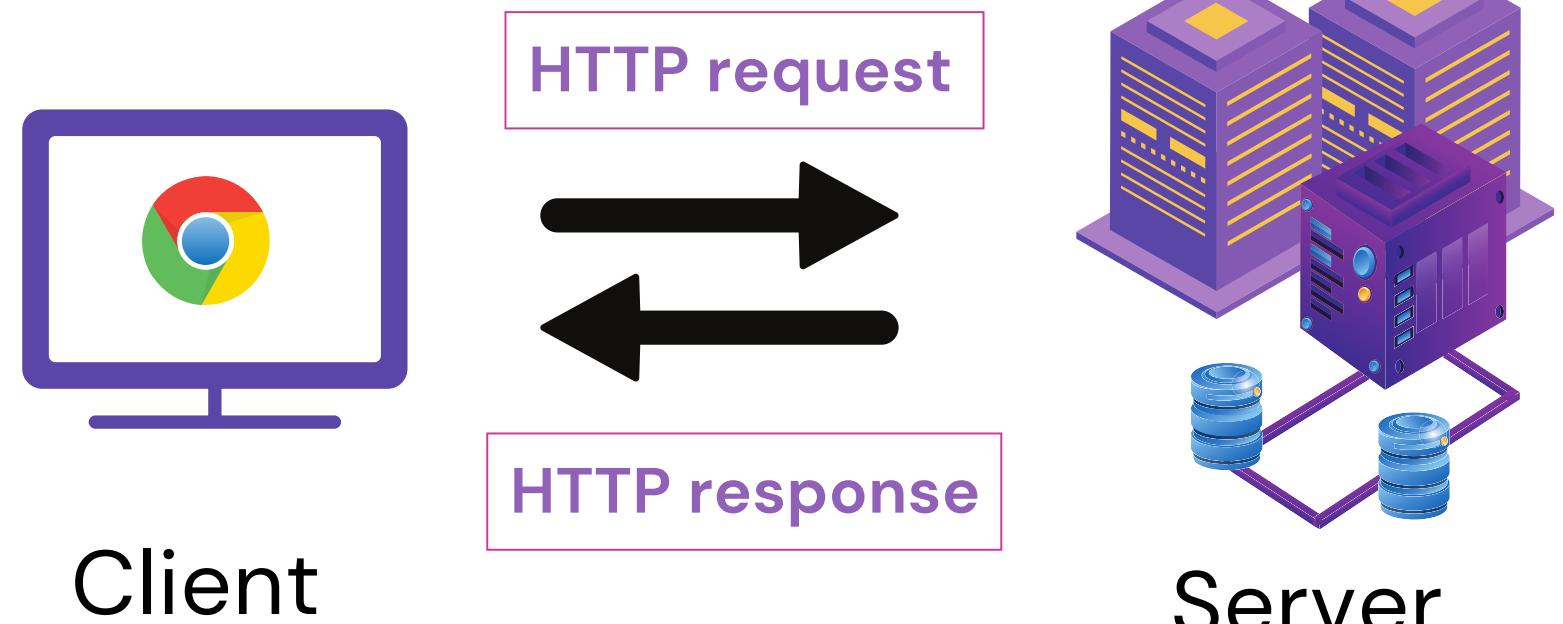


HTTP

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

Ele é a forma em que o Cliente e o Servidor se comunicam.

Pensando em uniformizar a comunicação entre servidores e clientes foram criados **códigos e verbos** que são usados por ambas as partes, e essas requisições são feitas em **URLs** que possuem uma estrutura específica.



HTTP - Status Code

Quando o Client faz uma requisição o Server responde com um código de status numérico também padronizado.

Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi concluída. As respostas são agrupadas em cinco classes:

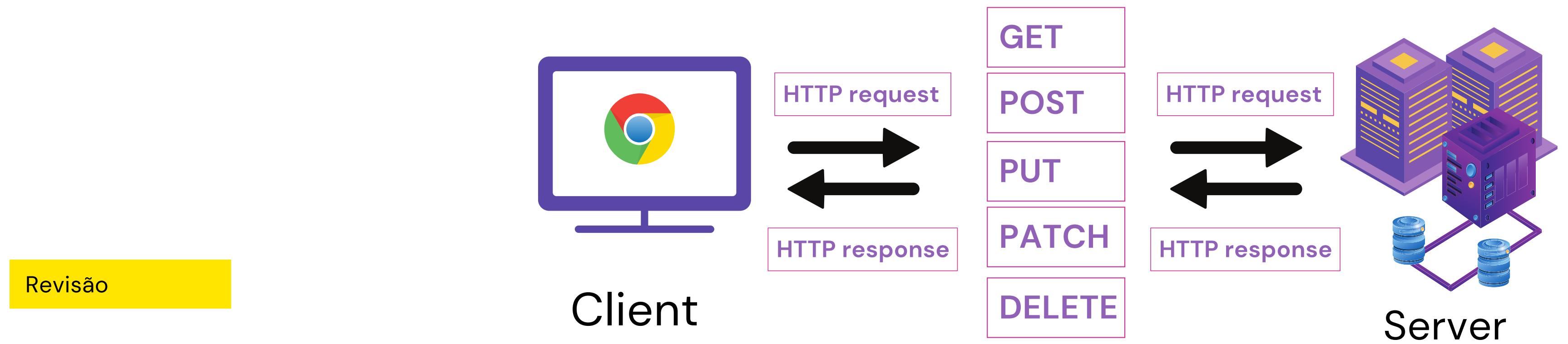
É a desenvolvedora Back end que coloca na construção do servidor quais serão as situações referentes a cada resposta.

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

HTTP - Verbos

Os verbos HTTP são um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada.

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



HTTP - 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

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

GET

POST

PUT

PATCH

DELETE

ler

criar

substituir

modificar

excluir

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.

Revisão

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.

Node.js

O JavaScript do lado do servidor

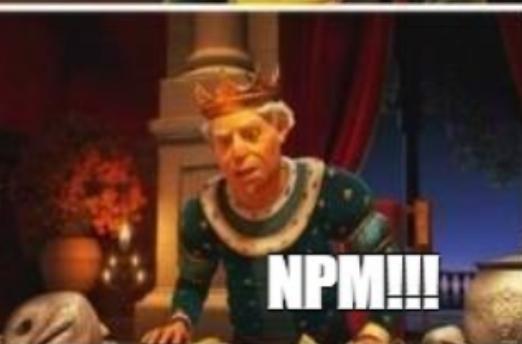
Interpretador JavaScript que não precisa de navegador.

Ele pode:

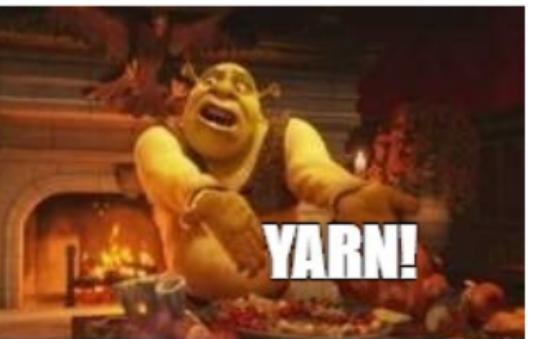
Ler e escrever arquivos no seu computador
Conectar com um banco de dados
Se comportar como um servidor



npm install



yarn



Revisão

fs

acessando arquivos no sistema pelo node.js

The Node.js fs module

The `fs` module provides a lot of very useful functionality to access and interact with the file system.

There is no need to install it. Being part of the Node.js core, it can be used by simply requiring it:

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

iniciando o projeto

npm init

instalando as dependências

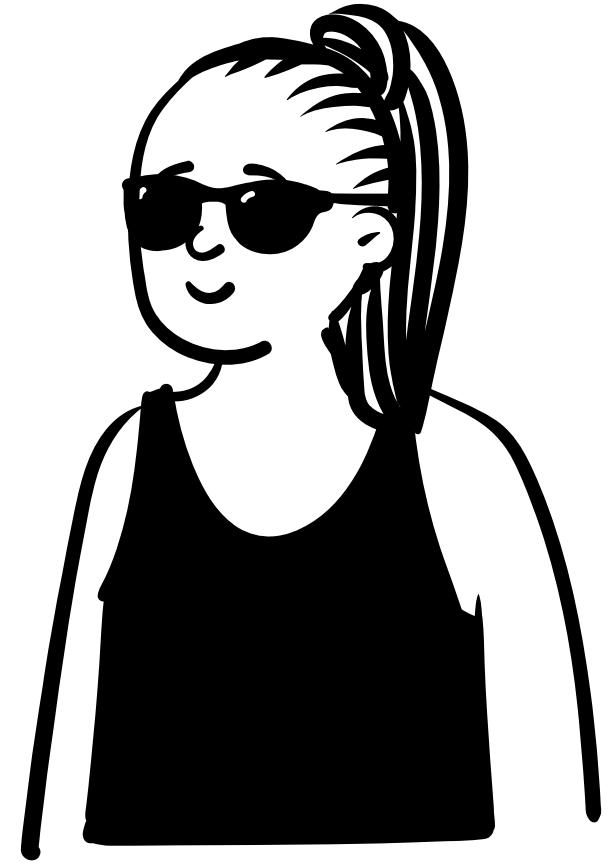
npm install NOME-DA-DEPENDÊNCIA

estartando o servidor

npm start

instalando dependencias de um servidor já iniciado

npm install

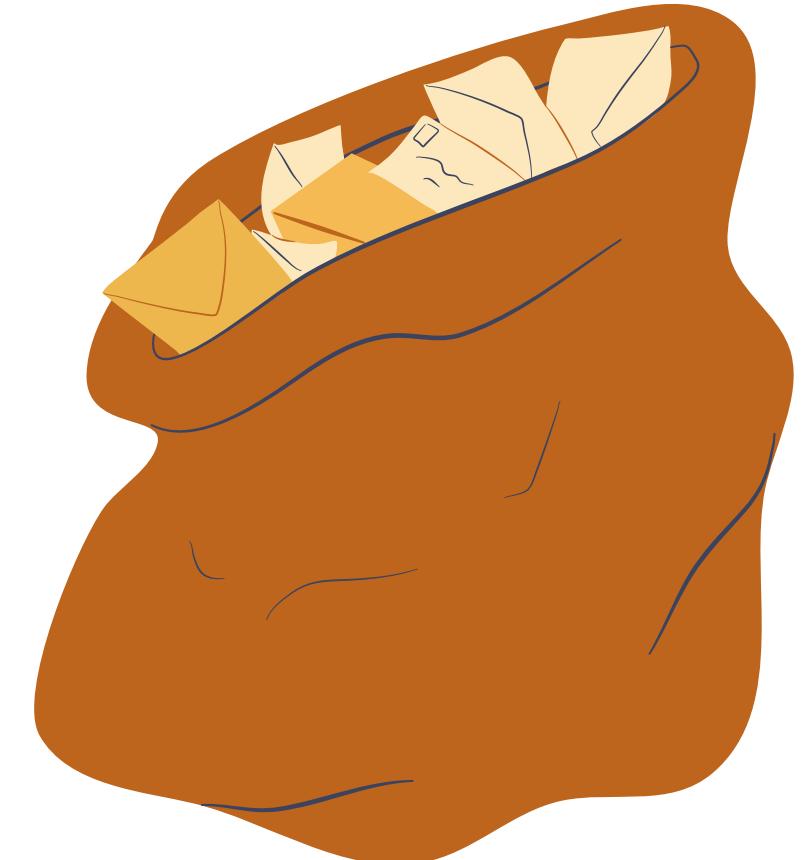


Pacotes

```
✓ servidor-em-aula
  > node_modules
  {} package-lock.json
  {} package.json
  JS server.js
```

Sempre que iniciamos um servidor com Node.js o resumo do seu projeto ficara no **package.json**

Quando instalamos uma dependência pelo npm, ele será referenciado no **package-lock.json** e será instalado na pasta **node_modules**.



Package.json

O arquivo package.json é o ponto de partida de qualquer projeto NodeJS. Ele é responsável pela descrição do projeto, indicação das dependências de desenvolvimento, etc

Package-lock.json

O package-lock especifica a versão e suas dependências próprias, assim, a instalação criada será sempre a mesma, toda vez.

node_modules

Na node_modules estarão baixadas as dependências que o seus pacotes precisarão pra funcionar

.gitignore

Devemos ignorar a node_modules, pois ela é muito pesada e desnecessária no versionamento, como git.

Se apagarmos a node_modules ou clonarmos um projeto que já possui package.json e package-lock.json, ou seja, já foi inicializado, só precisamos dar o comando **npm install** que as dependências serão baixadas de novo e pasta node_modules reaparecerá.

Express

npm install express

express
4.17.1 • Public • Published a year ago

Readme Explore (BETA) 30 Dependencies 46.033 Dependents 264 Versions

express

Fast, unopinionated, minimalist web framework for node.

npm v4.17.1 downloads 58M/month linux passing windows passing coverage 100%

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

Install > npm i express

Weekly Downloads 13.961.907

Version 4.17.1 License MIT

Unpacked Size 16 Total Files 16

nodemon

npm install nodemon

nodemon
2.0.4 • Public • Published 4 months ago

Readme Explore (BETA) 10 Dependencies 2.477 Dependents 215 Versions



nodemon

nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

Install > npm i nodemon

Fund this package

Weekly Downloads 2.893.116

Version 2.0.4 License MIT

Unpacked Size 107 kB Total Files 43

Revisão

cors

npm install cors

CORS (Cross-Origin Resource Sharing) é uma especificação que permite que um site acesse recursos de outro site mesmo estando em domínios diferentes.

Os navegadores fazem uso de uma funcionalidade de segurança chamada **Same-Origin Policy**: um recurso de um site só pode ser chamado por outro site se os 2 sites estiverem sob o mesmo domínio.

Isso porque o navegador considera recursos do mesmo domínio somente aqueles que usam o **mesmo protocolo** (http ou https), a mesma **porta** e o **mesmo endereço**

Revisão

Pensando assim, o front-end e o back-end deveriam estar no mesmo Servidor e na mesma camada, o que não acontece!

Para resolver esse problema usamos o CORS

Arquitetura

Vamos começar a organizar isso aqui

Nas primeiras semanas nós
estávamos fazendo tudo dentro
de um arquivo só, o server.js

acessando o JSON

conectando com express

criando as rotas

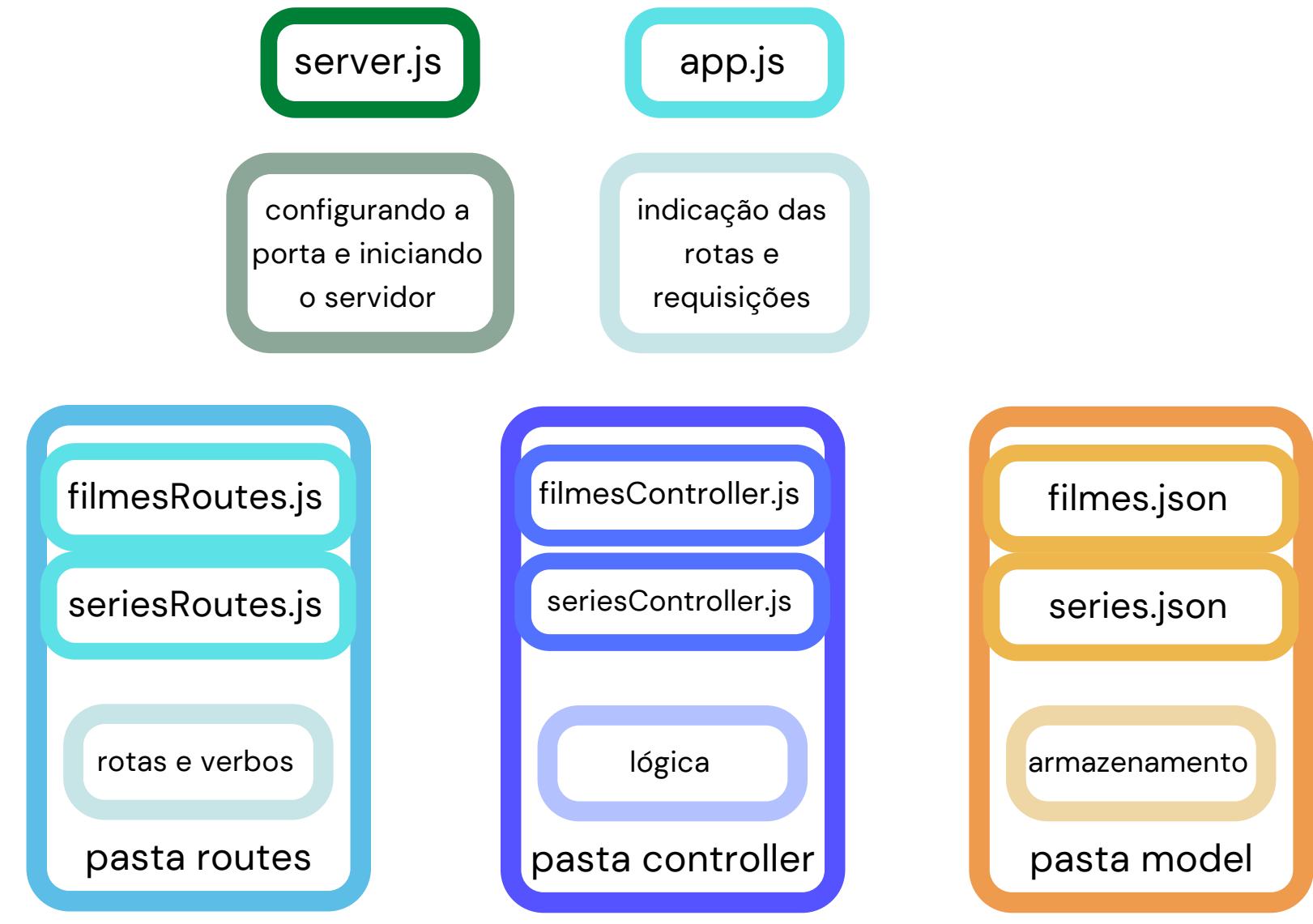
criando as lógicas

criando as rotas

criando as lógicas

configurando a porta e iniciando o servidor

Arquitetura - MVC

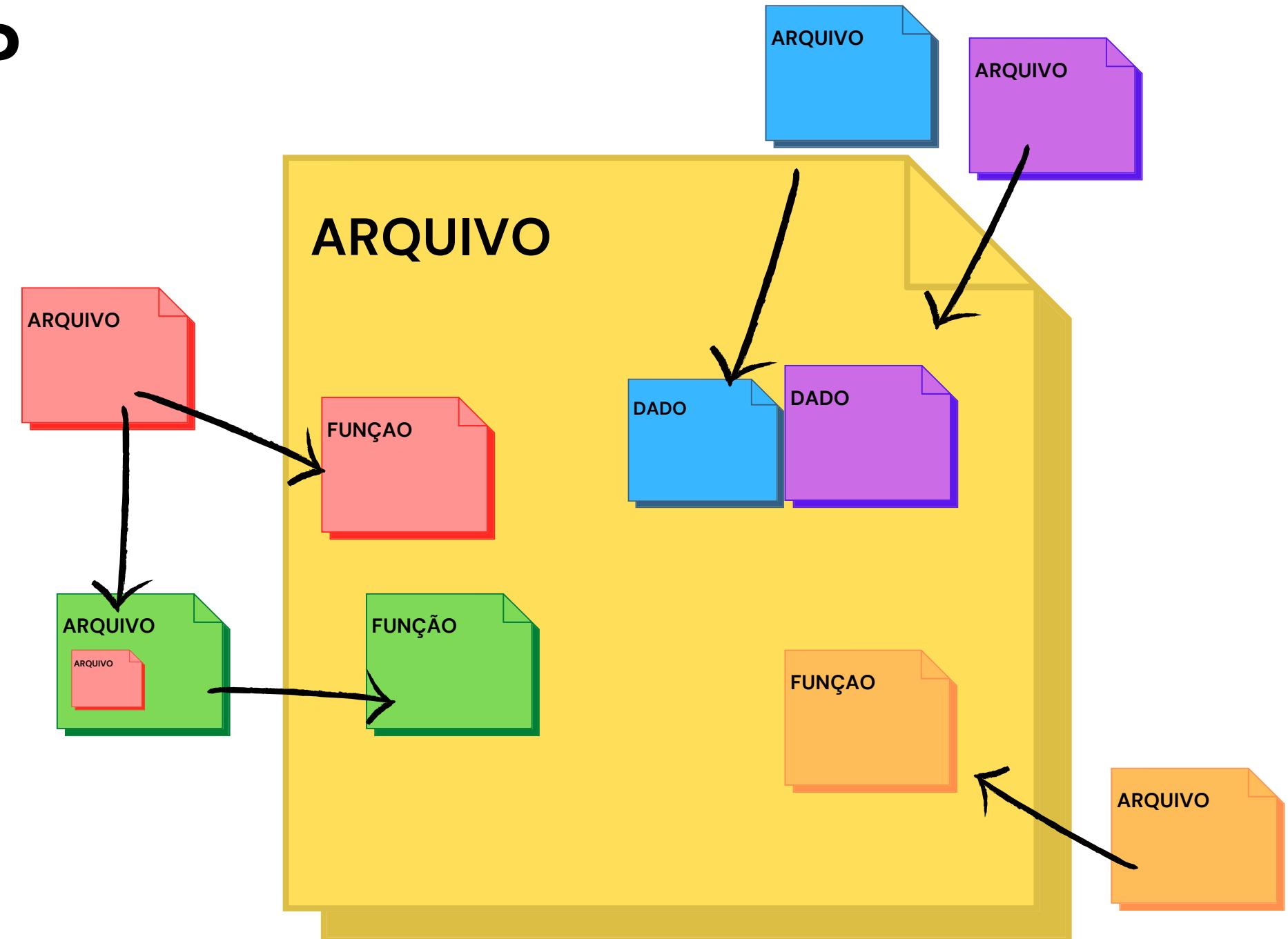


```
\-- NOME-DO-SEU-SERVIDOR
|   .gitignore
|   package-lock.json
|   package.json
|   server.js
\-- node_modules
\-- src
    |   app.js
    |
    \--- controller
        |   NOMEController.js
    |
    \--- model
        |   NOME.json
    |
    \--- routes
        |   NOMERoute.js
```

Revisão

Mas vai ser sempre assim?

Não! Quanto maior a aplicação sua arquitetura se torna mais complexa, muitas vezes existem mais camadas de organização e comunicações entre arquivos. E apesar de usarem o MVC acabam ficando bem maiores e mais setorizadas



HTTP - GET

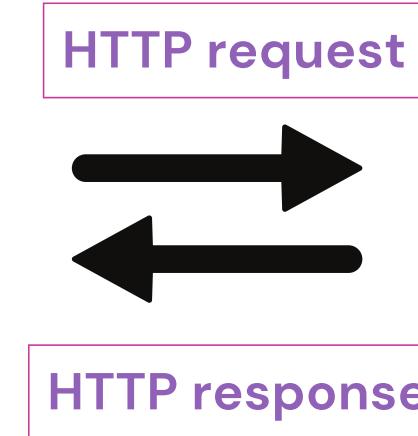
Dentro do CRUD o método GET é representado pela letra R

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

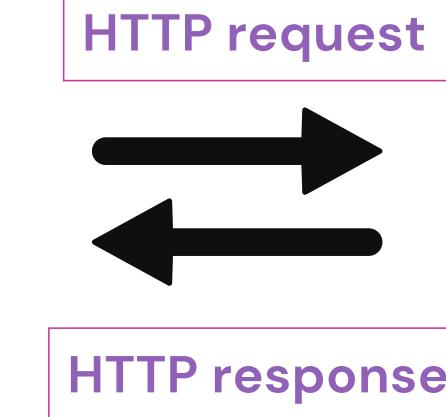
O Client manda um **request** solicitando realizar um GET e o Server deve estar preparado para receber esse GET e responde-lo com um **response**.



Client



- GET por id
- GET todos
- GET por nome
- GET por gênero



Server

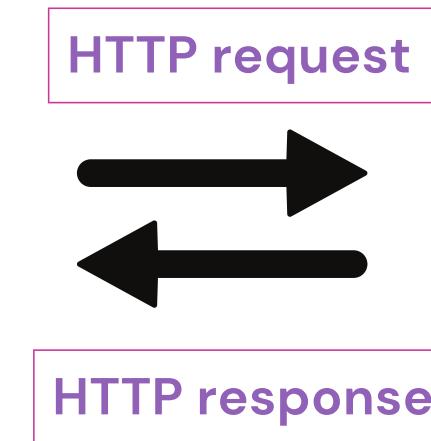
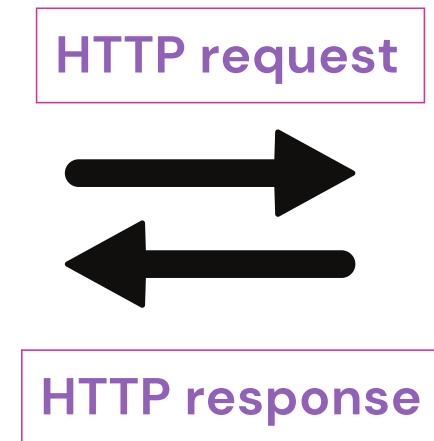
HTTP - POST & DELETE

✓ C: Create (criar) – criar um novo registro

✗ D: Delete (apagar) – apagar um registro



Client



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

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"  
}
```

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 multiplas strings

request.body

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

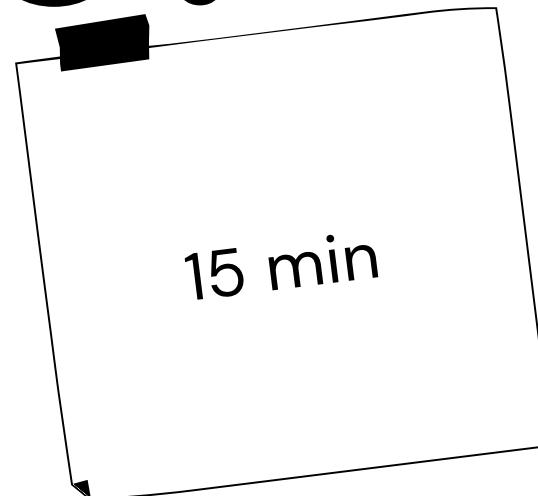
Body Parse

```
app.use(express.json())
```

Quando recebemos um request os dados do body são enviados de uma forma que não conseguimos facilmente acessar e manipular.

Por isso, devemos "parsear" o body: essa função analisa a transforma num json manipulável

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



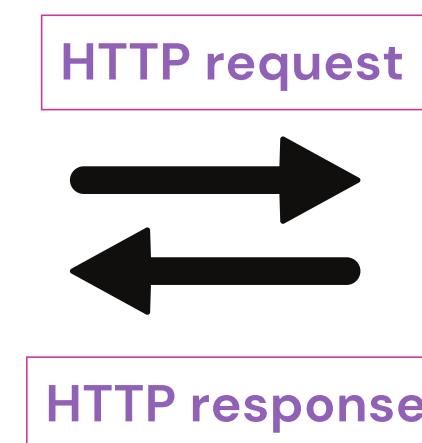
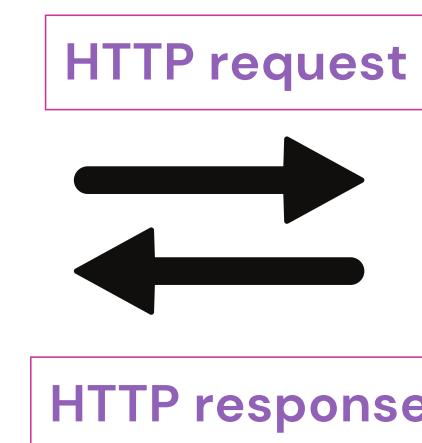
15 min

HTTP - PUT & PATCH

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



Client



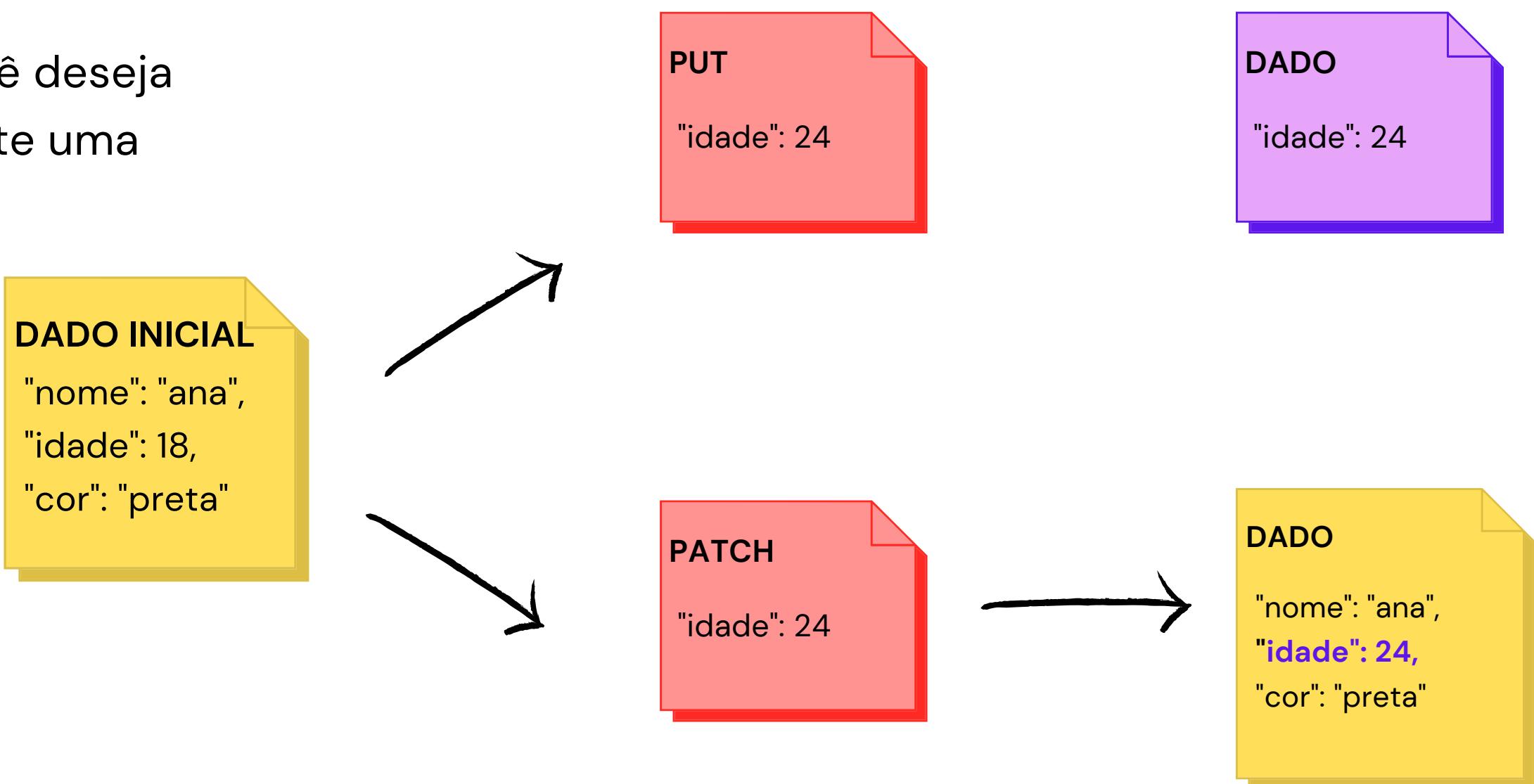
Server

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.

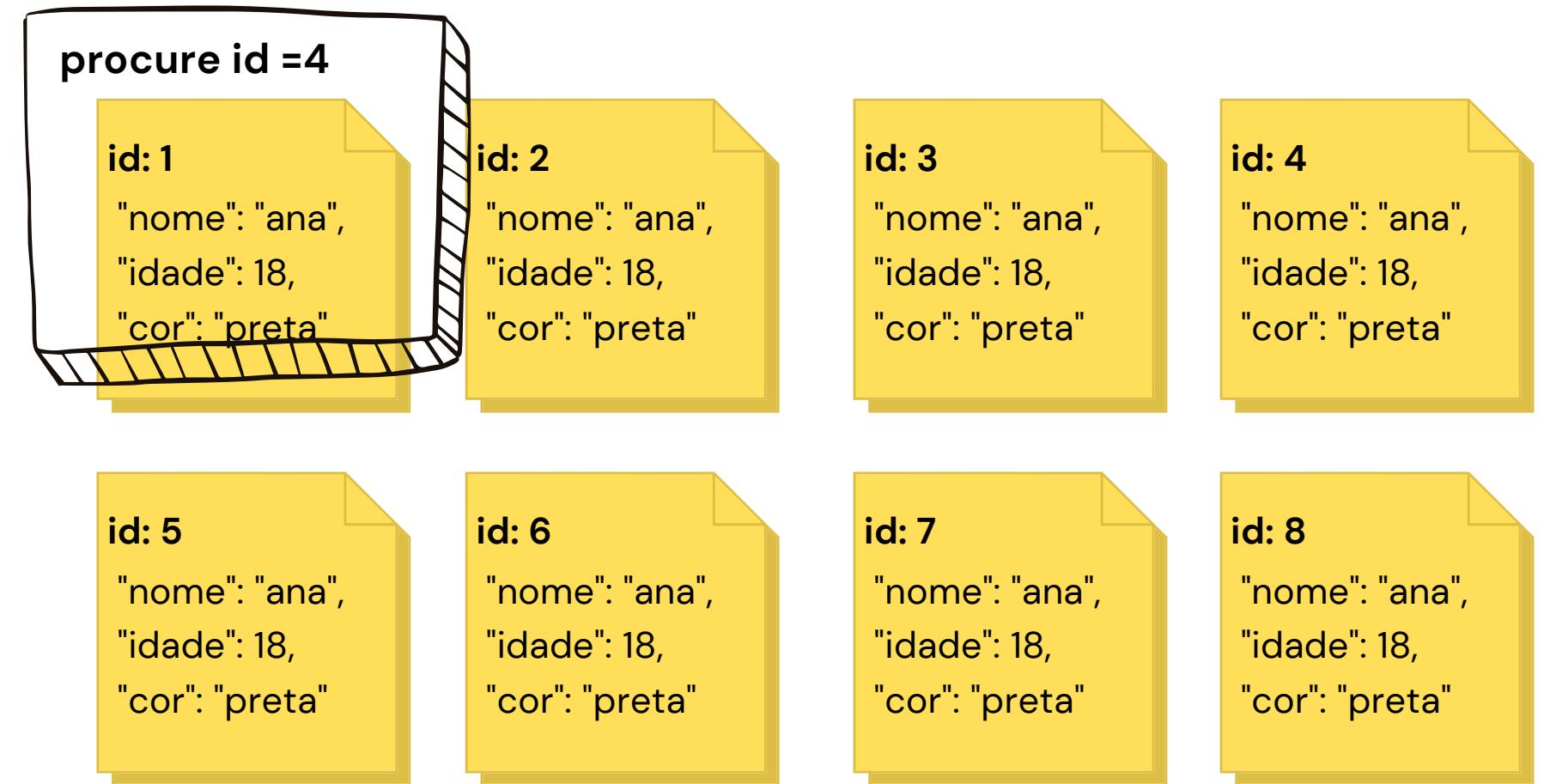


PUT x PATCH

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`

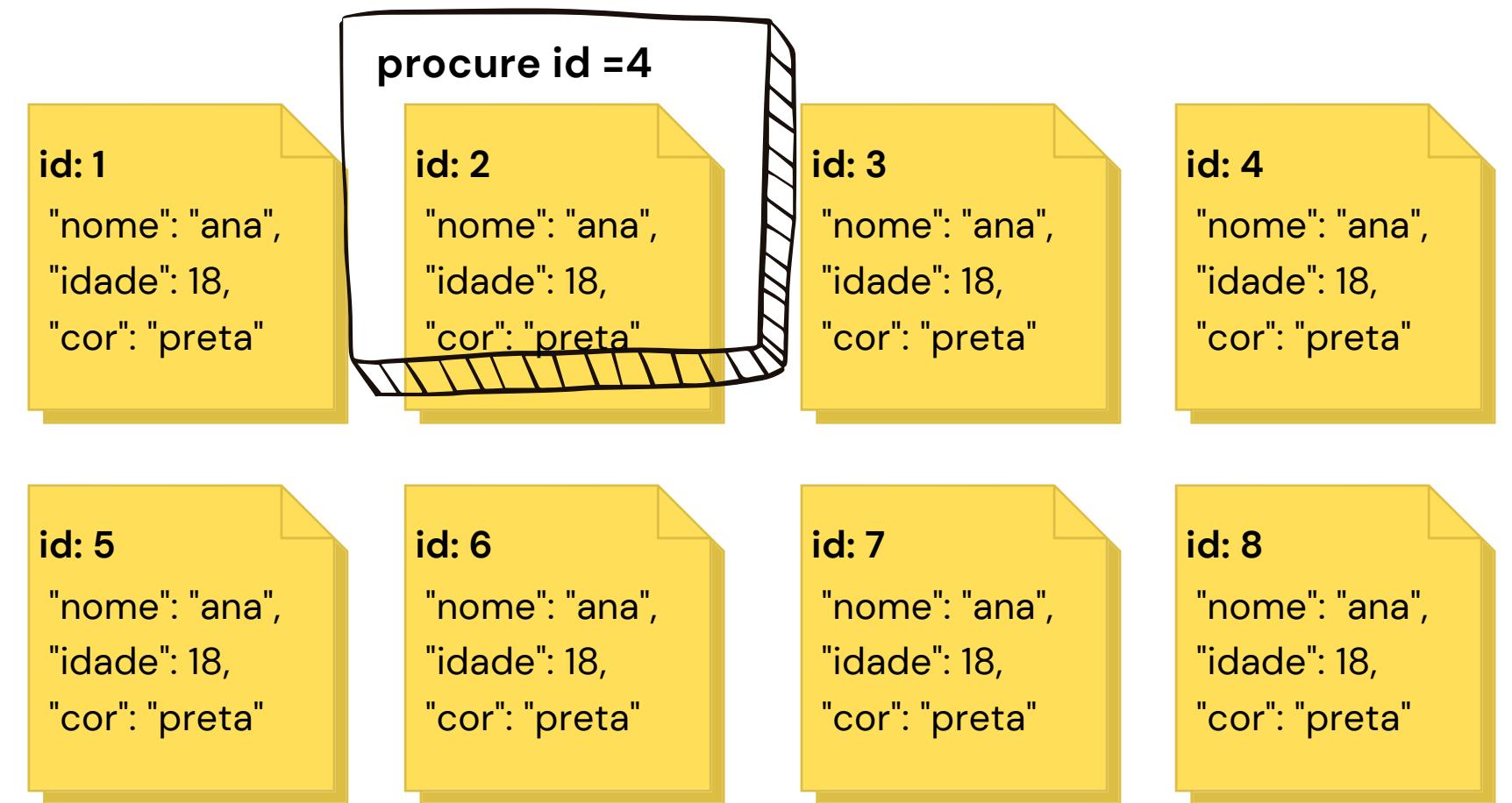


PUT x PATCH

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



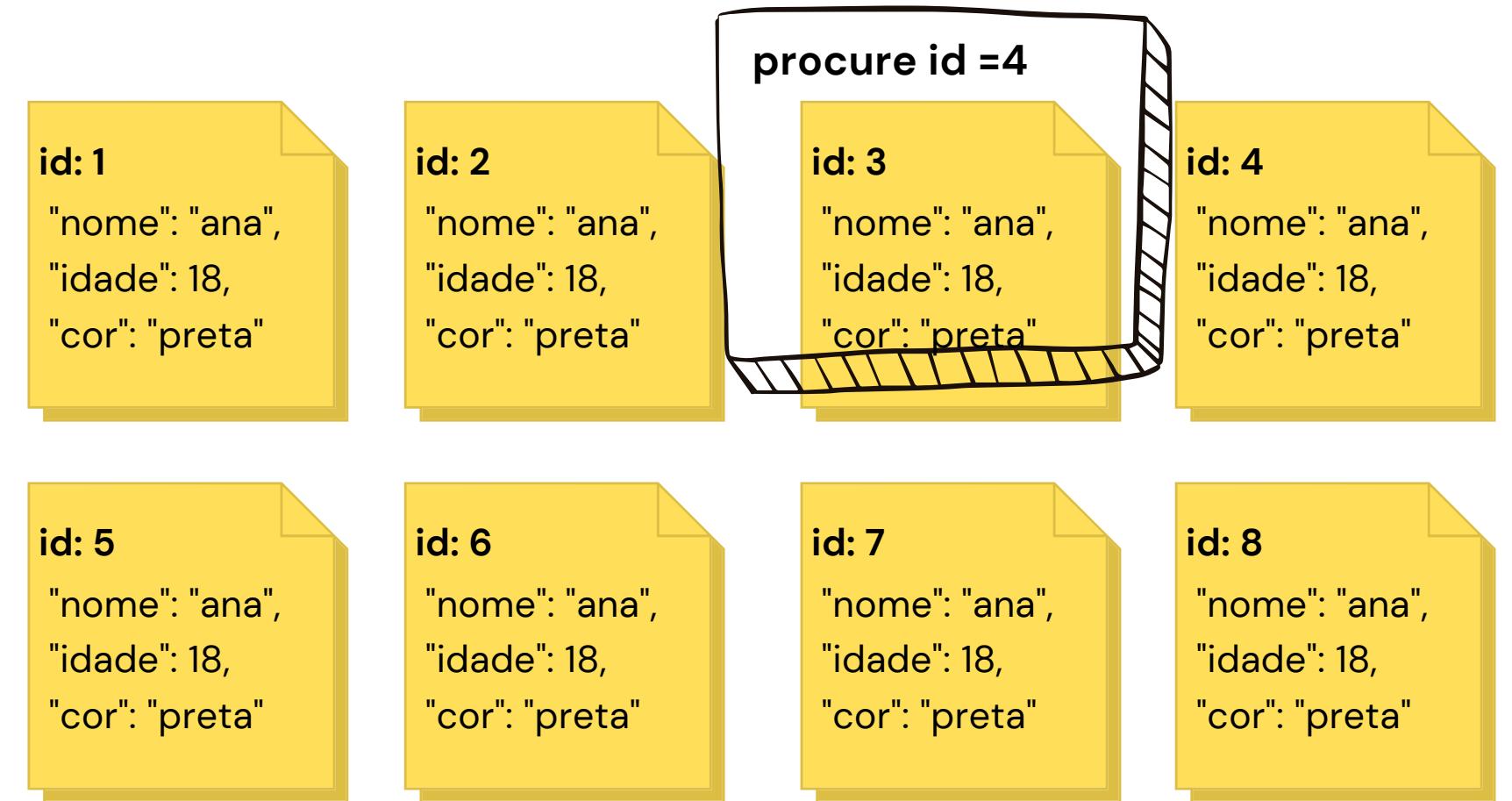
PUT x PATCH

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
  "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"

blog



da Reprograma

Demandas da API

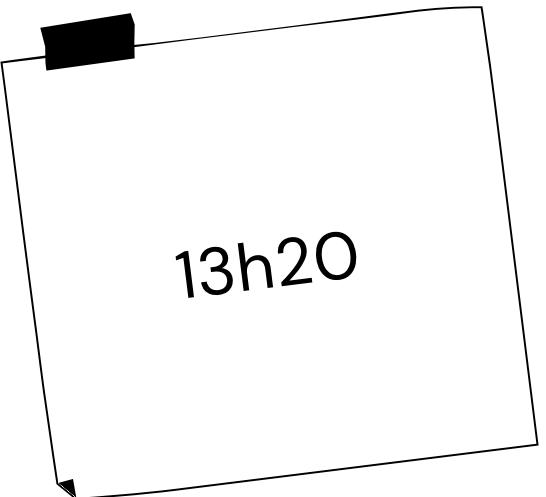
Lá vem a galera de negócio....



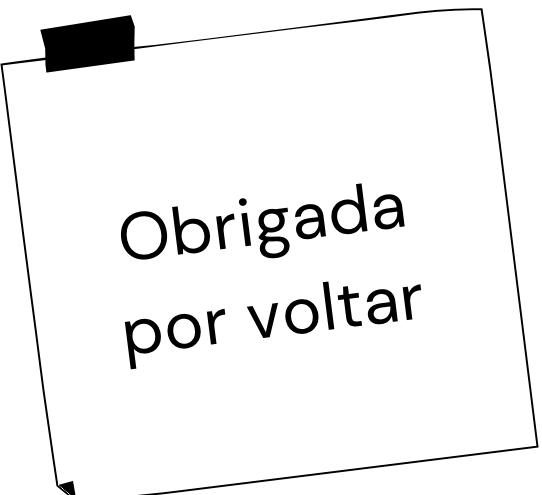
- devo conseguir ver todos os post
- devo conseguir um post específico
- devo conseguir deletar post
- devo conseguir atualizar post
- devo conseguir atualizar título do post
- devo conseguir atualizar qualquer parte do post separadamente



Ai, que fome...



Nossa, você aqui
de novo?



Continue a codar



a codar... a codar

Para olar

UHUUUL