

PUT & PATCH

DELETE



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 back-end web.

Email: sampaioaanaluiza@gmail.com



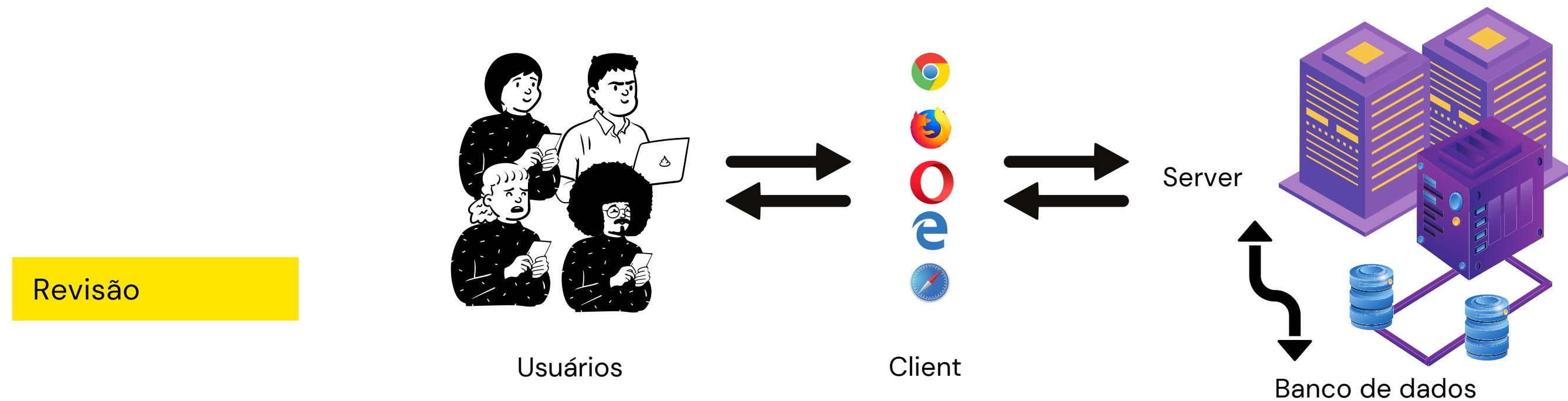
Modelo Server-Client

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.

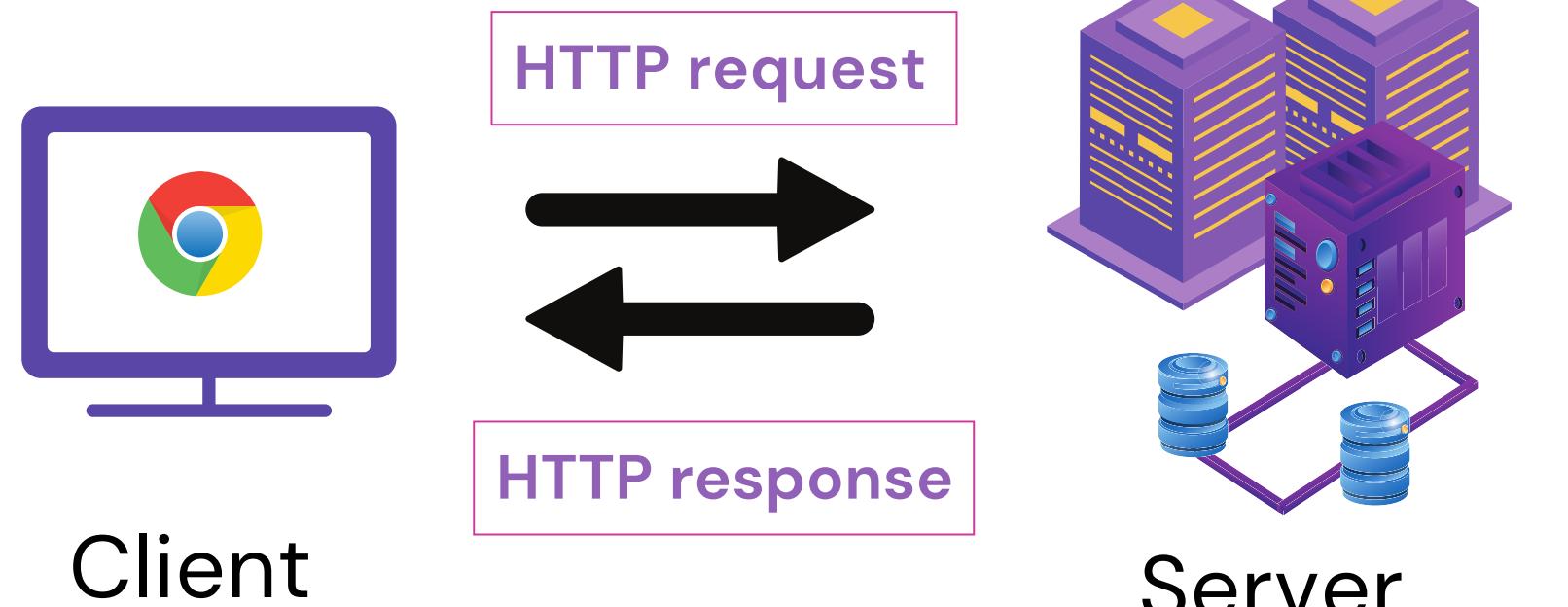


HTTP

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:

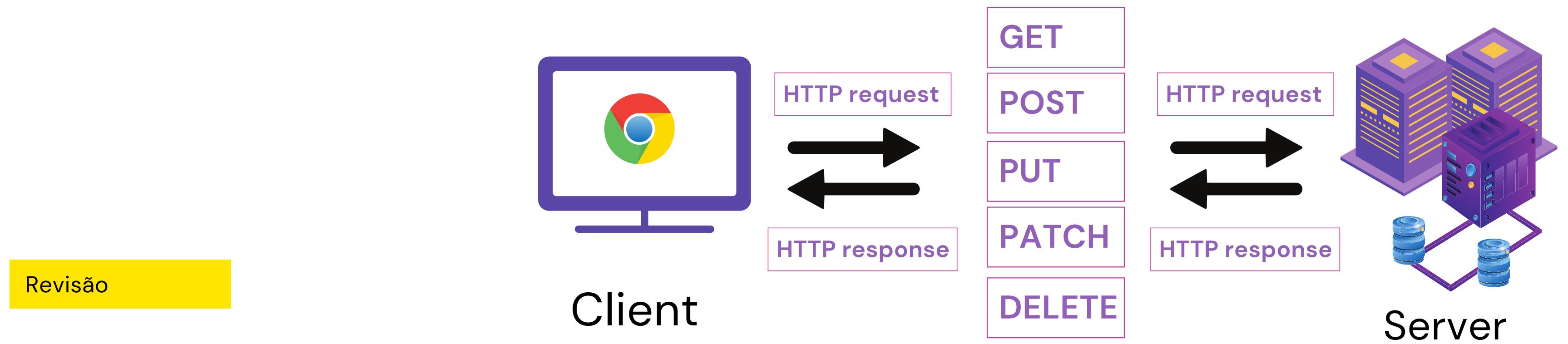
É a desenvolvedora Back end que coloca na construção do servidor quais serão as situações referentes a cada resposta.

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:

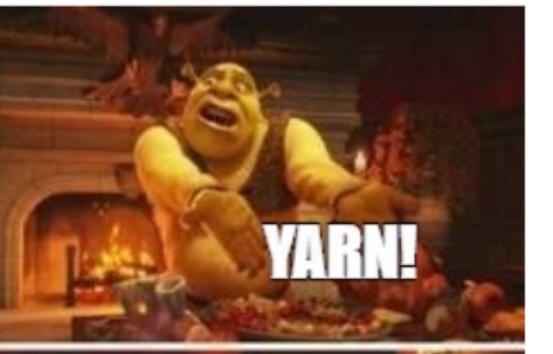
Ler e escrever arquivos no seu computador
Conectar com um banco de dados
Se comportar como um servidor



npm install



yarn



Reprogramaflix



tudum

Revisão

iniciando o projeto

npm init

instalando as dependências

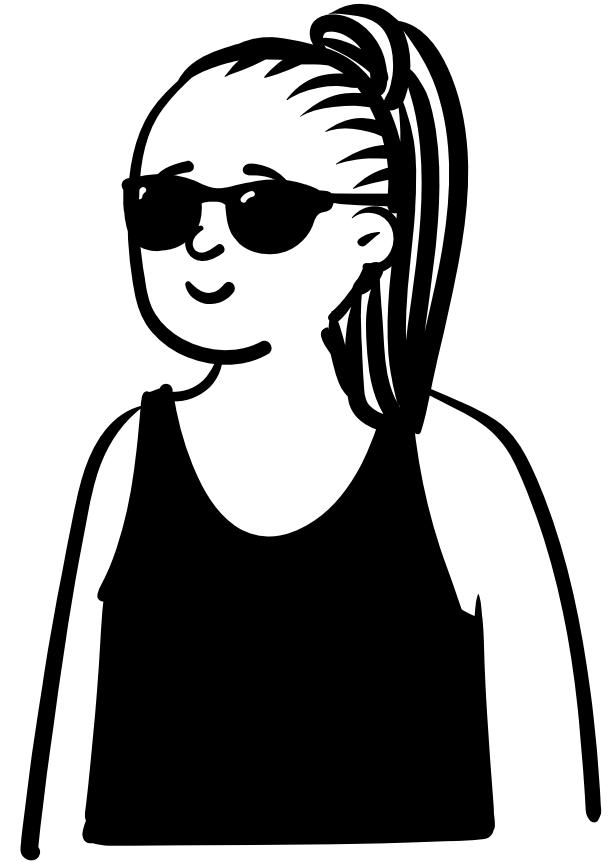
npm install NOME-DA-DEPENDÊNCIA

estartando o servidor

npm start

instalando dependencias de um servidor já iniciado

npm install

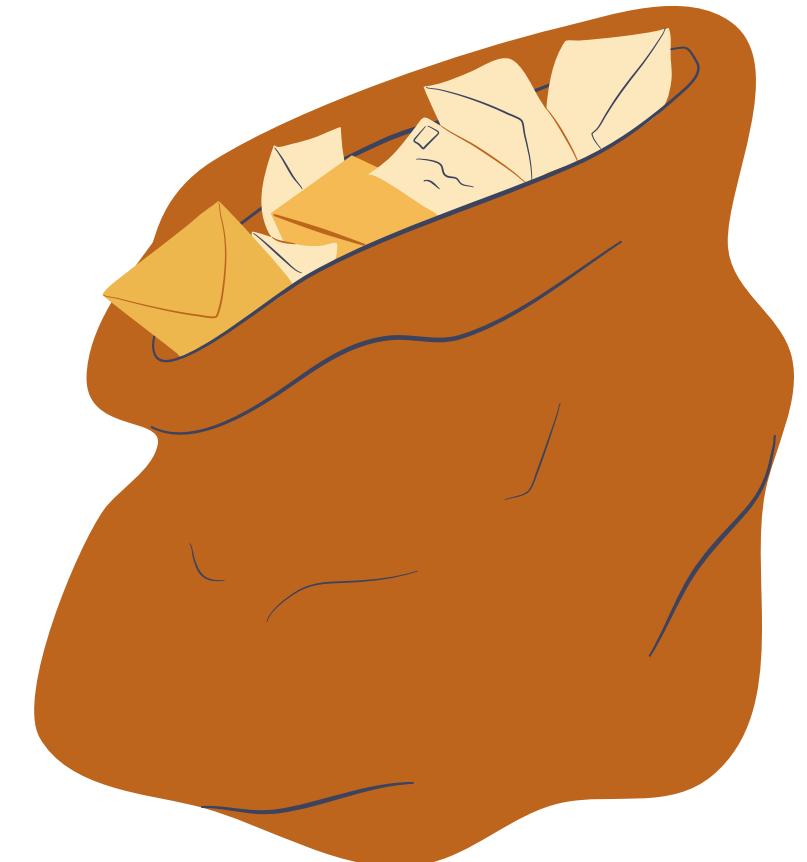


Pacotes

```
✓ 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

Install
> npm i 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()
```

nodemon

npm install nodemon

nodemon
2.0.4 • Public • Published 4 months ago

Readme Explore (BETA) 10 Dependencies 2.477 Dependents 215 Versions

Install
> npm i nodemon

Fund this package

Weekly Downloads 2.893.116

Version License
2.0.4 MIT

Unpacked Size Total Files
107 kB 43



Demandas da API

Lá vem a galera de negócio....



- Quero uma rota que venha todos os filmes e series
- duas rotas a **/filmes** e a **/series**
- **/filmes** deve retornar **todos os filmes**
- **/series** deve retornar **todos as series**
- Devo conseguir **filtrar** por **titulo**, **id** e **genero**
- Devo conseguir **cadastrar** filmes e series
- Devo conseguir **editar o nome** de qualquer um deles
- Devo conseguir **substituir tudo** de um filme ou serie
- Devo conseguir **editar** qualquer campo deles



Demandas da API

Lá vem a galera de negócio....

- [GET] /catalogo
 - retorna todos os filmes
- [GET] /filmes
 - retorna todos os filmes
- [GET] /filmes/{id}
 - retorna um filme pelo id
- [GET] /filmes?{titulo}
 - retorna um filme pelo nome
- [POST]/filmes/criar
 - cria novo filmes
- [GET] /series
 - retorna todos os filmes
- [GET] /series?{id}
 - retorna um filme pelo id

aaaata



HTTP - GET

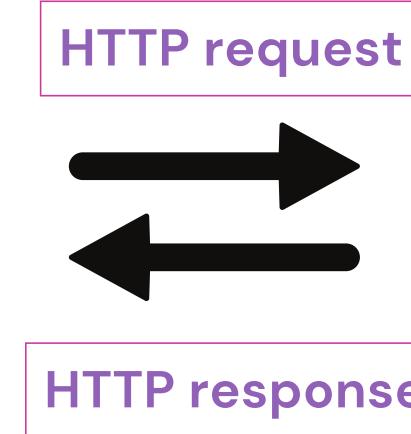
Dentro do CRUD o método GET é representado pela letra R

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

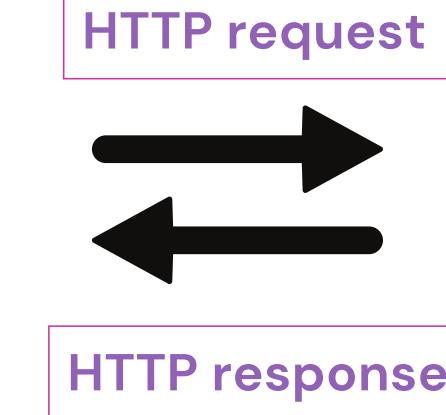
O Client manda um **request** solicitando realizar um GET e o Server deve estar preparado para receber esse GET e responde-lo com um **response**.



Client



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



Server

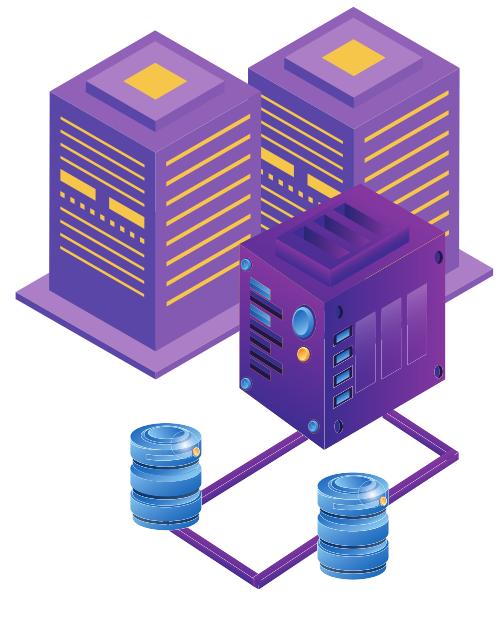
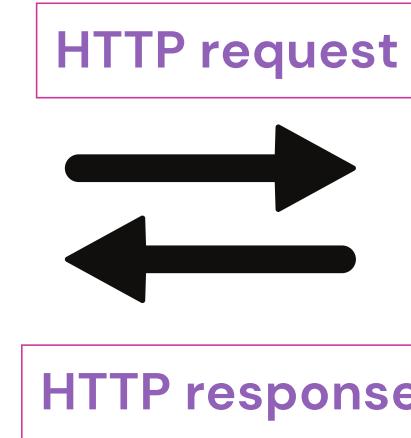
HTTP - POST

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



Client

Revisão



Server

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

The screenshot shows the npm package page for 'cors'. At the top, it displays the package name 'cors' in bold, followed by its version '2.8.5', status 'Public', and publication date 'Published 2 years ago'. Below this is a navigation bar with tabs: 'Readme' (selected), 'Explore (BETA)', '2 Dependencies', '7.113 Dependents', and '34 Versions'. The main content area has a heading 'cors' and a brief description: 'CORS is a node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options.' It includes links for 'Follow me (@troygoode) on Twitter!', 'Installation', and 'Usage'. To the right, there's an 'Install' button with the command 'npm i cors', a 'Weekly Downloads' chart showing 3.985.079, a 'Version' section showing '2.8.5', and a 'License' section showing 'MIT'.

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

Body

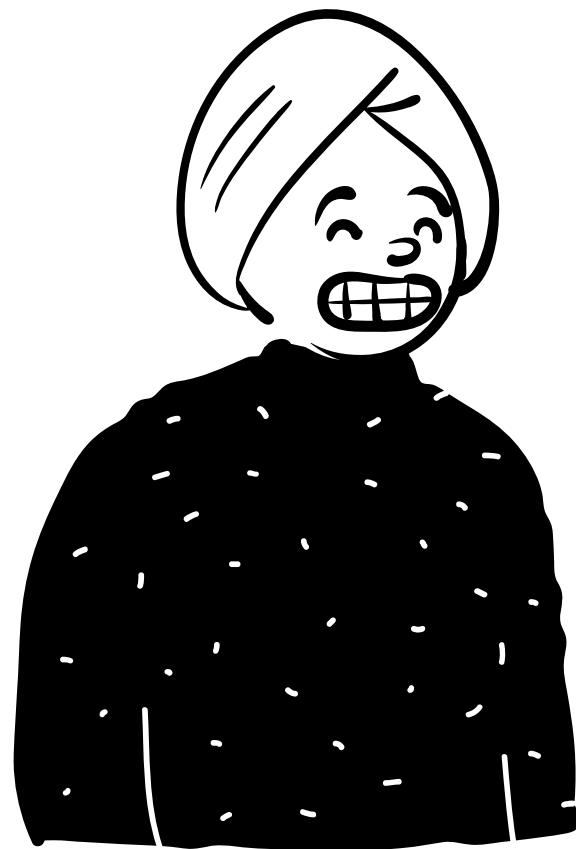
- são usados nos métodos POST, PATCH e PUT
- enviam dados a serem cadastrados no banco de dados
- request.body



```
{  
  "descricao": "exemplo de Body",  
  "nomeColaborador": "Ana"  
}
```

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 manipulavel

Revisão

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

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

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

15 min

Arquitetura

Vamos começar a organizar isso aqui

Até hoje 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

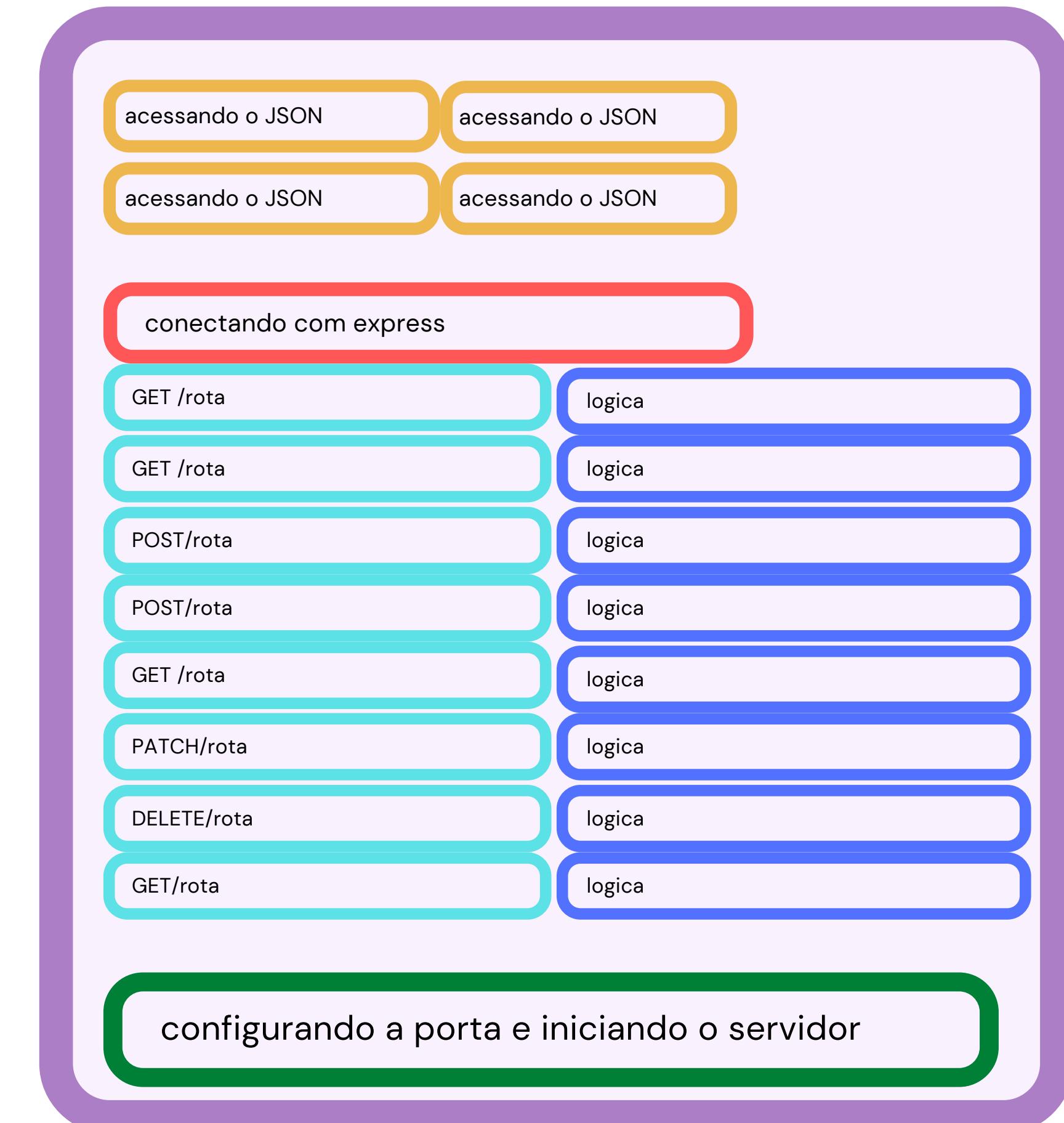
criando as lógicas

configurando a porta e iniciando o servidor

Arquitetura

Vamos começar a organizar isso aqui

E tudo isso faz muito sentido se a gente tiver um projeto simples, mas... como a gente faz quando temos um projeto mais complexo?



Arquitetura - MVC

MVC é um padrão de arquitetura de software, separando sua aplicação em 3 camadas. A camada de interação do usuário(**view**), a camada de manipulação dos **dados(model)** e a camada de **controle(controller)**

Já que estamos lidando com um projeto que tem somente back-end, não lidaremos com as views, porém lidaremos com as **rotas (routes)**.

O MVC nada mais é que uma forma de **organizar** o nosso código.

acessando o JSON

conectando com express

criando as rotas

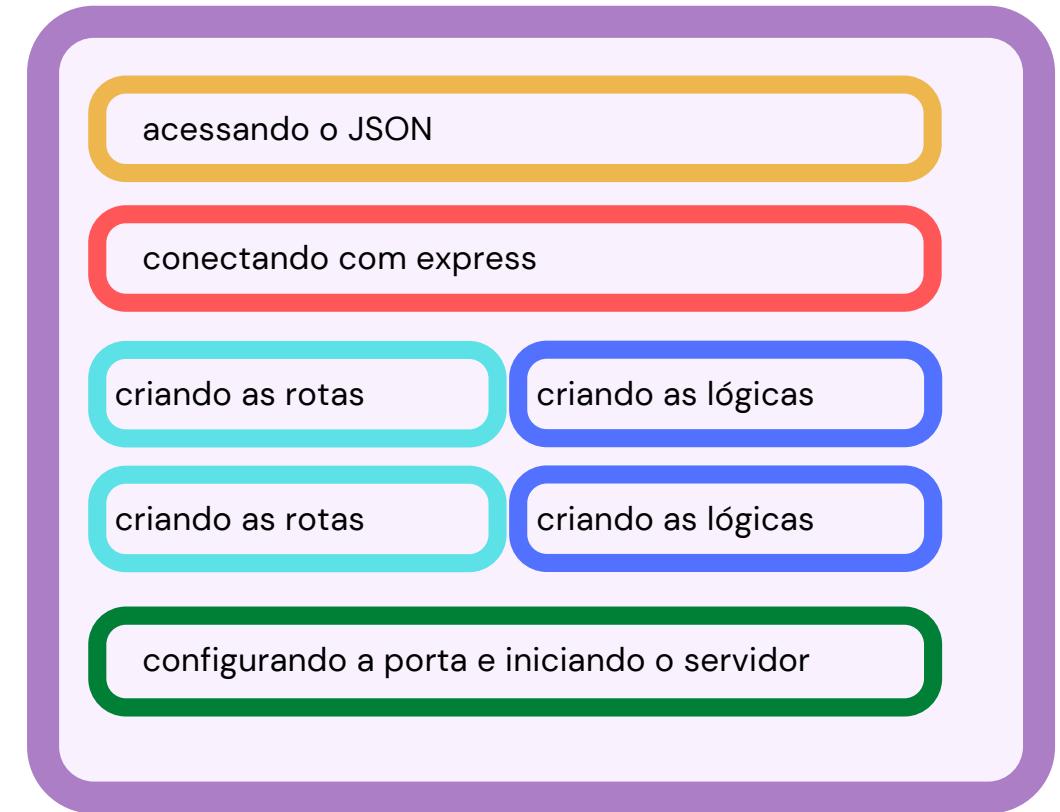
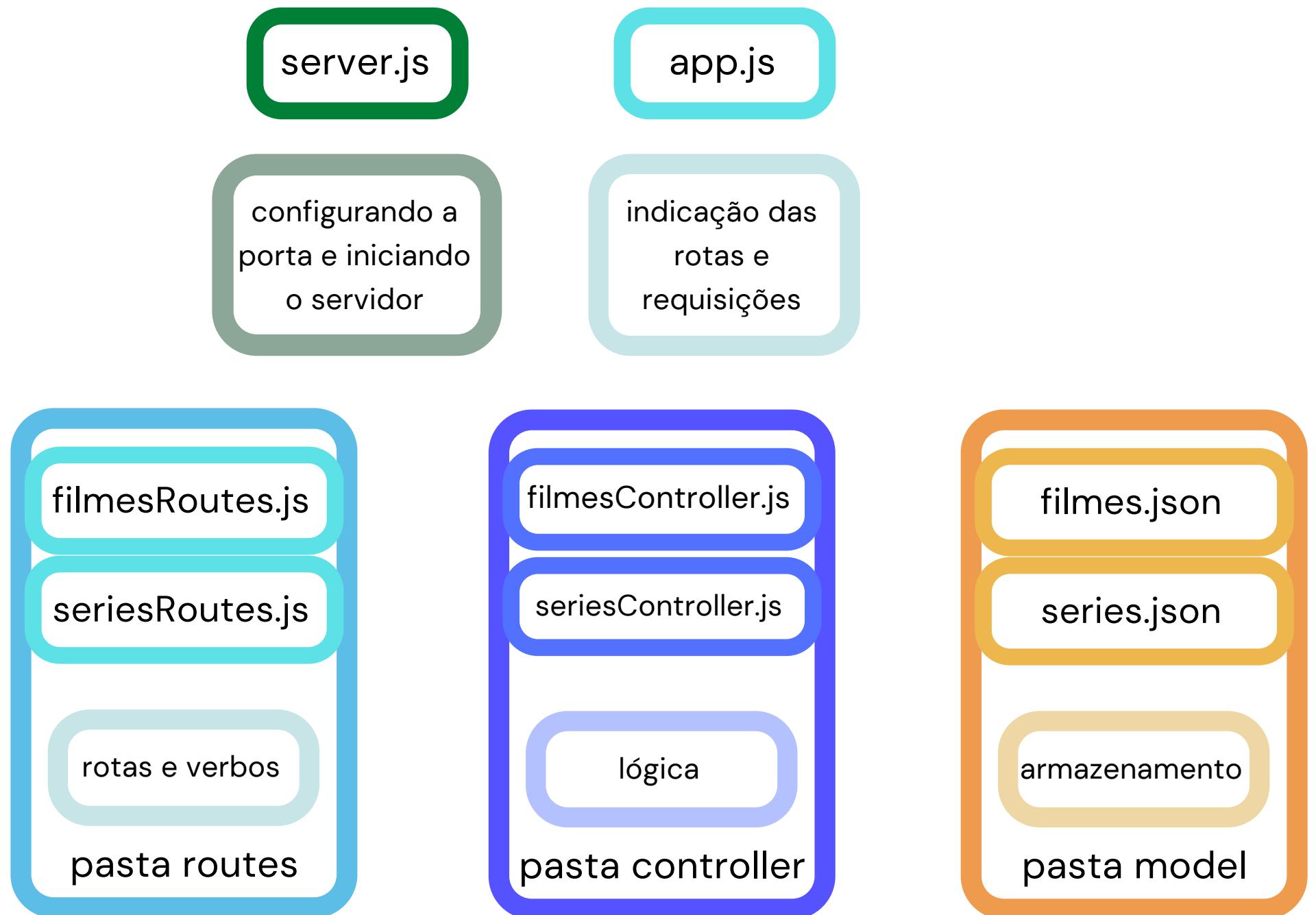
criando as lógicas

criando as rotas

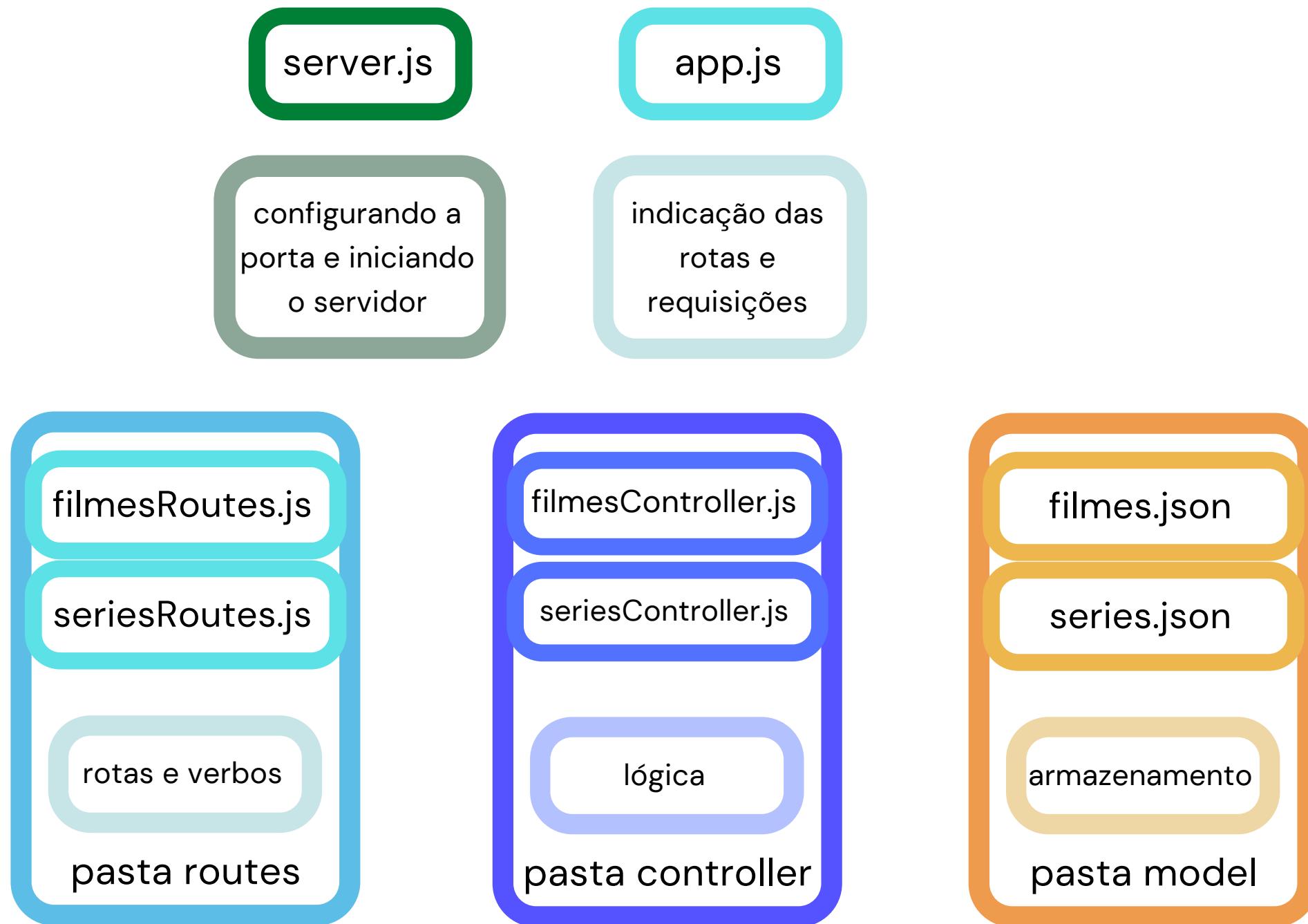
criando as lógicas

configurando a porta e iniciando o servidor

Arquitetura - MVC

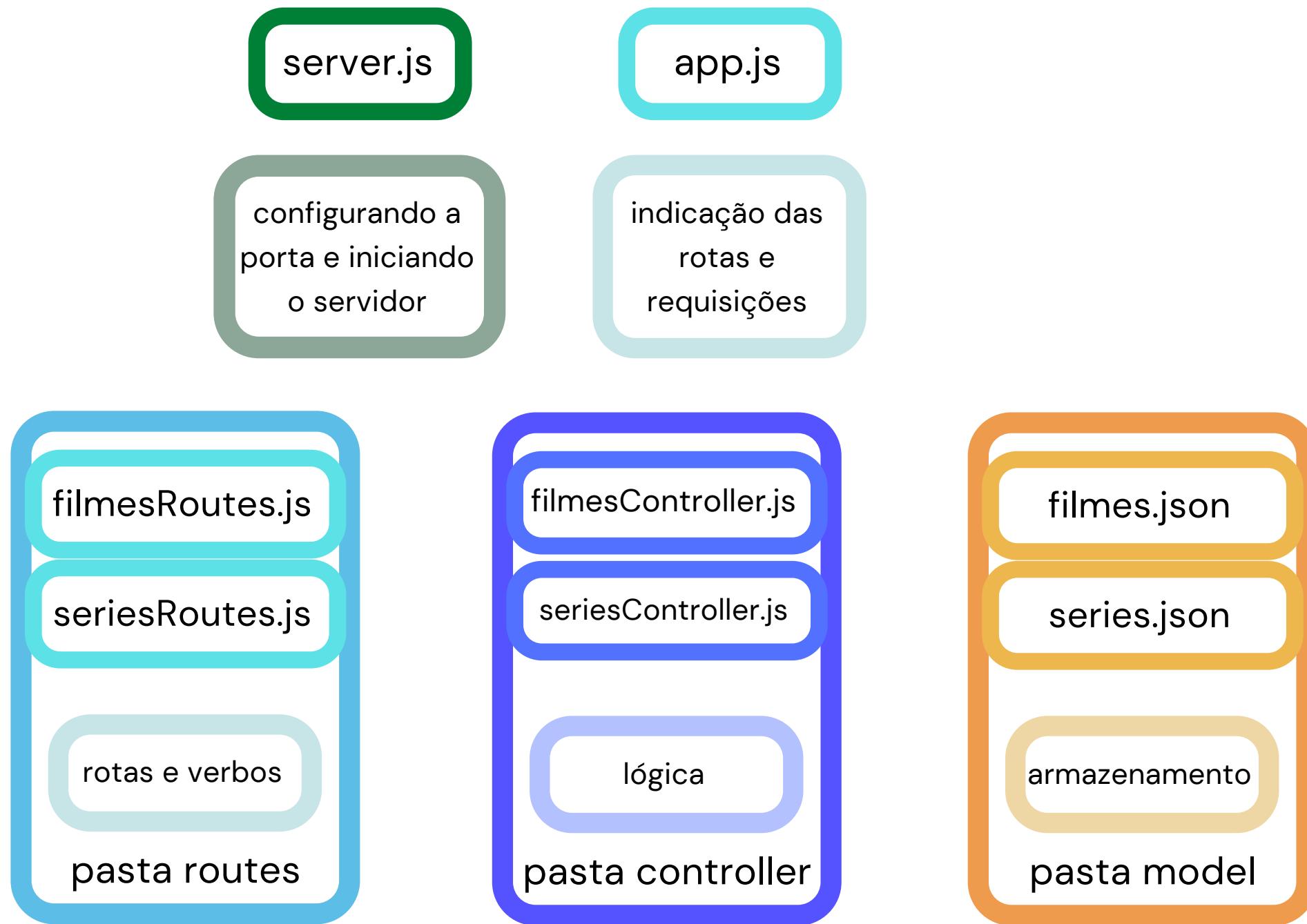


Arquitetura - MVC



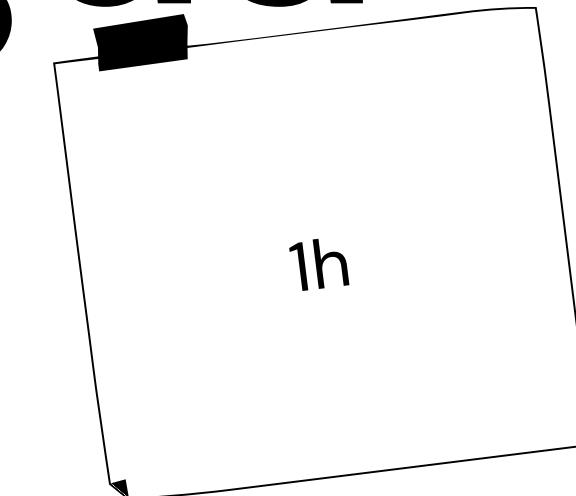
```
\-- NOME-DO-SEU-SERVIDOR
|   server.js
|
\-- src
|   app.js
|
|   ---controller
|       NOMEController.js
|
|   ---model
|       NOME.json
|
|   ---routes
|       NOMERoute.js
```

Arquitetura - MVC

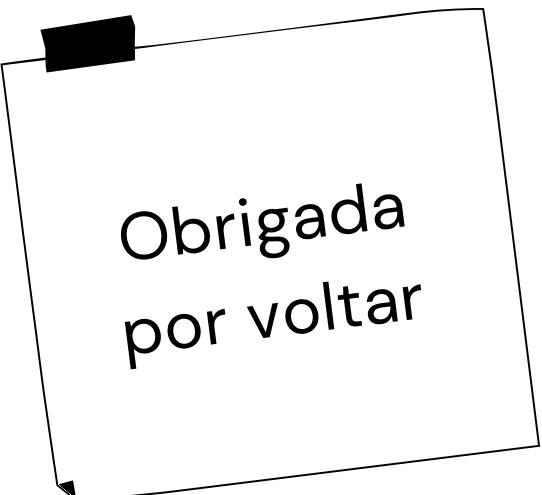


```
\-- NOME-DO-SEU-SERVIDOR
| .gitignore
| package-lock.json
| package.json
| server.js
\-- node_modules
\-- src
| app.js
|
| ---controller
|   NOMEController.js
|
| ---model
|   NOME.json
|
| ---routes
|   NOMERoute.js
```

**Calma, deixa eu
beber uma água**



Nossa, você aqui
de novo?

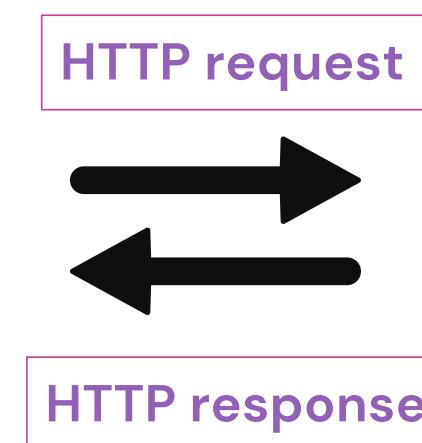
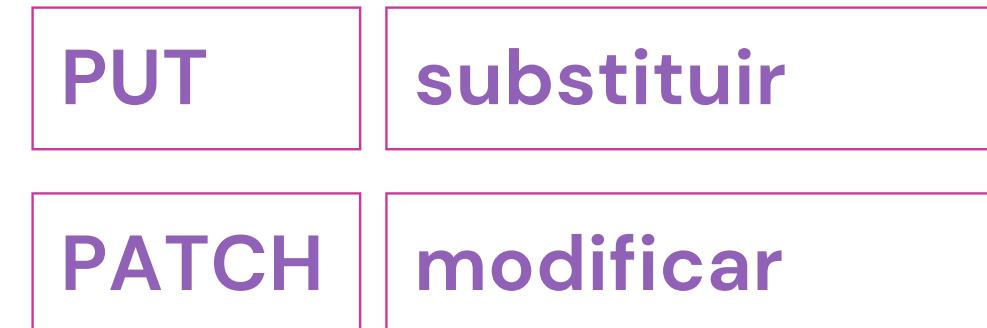
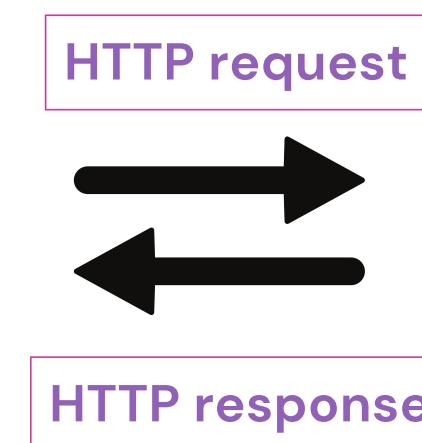


HTTP - PUT & PATCH

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



Client



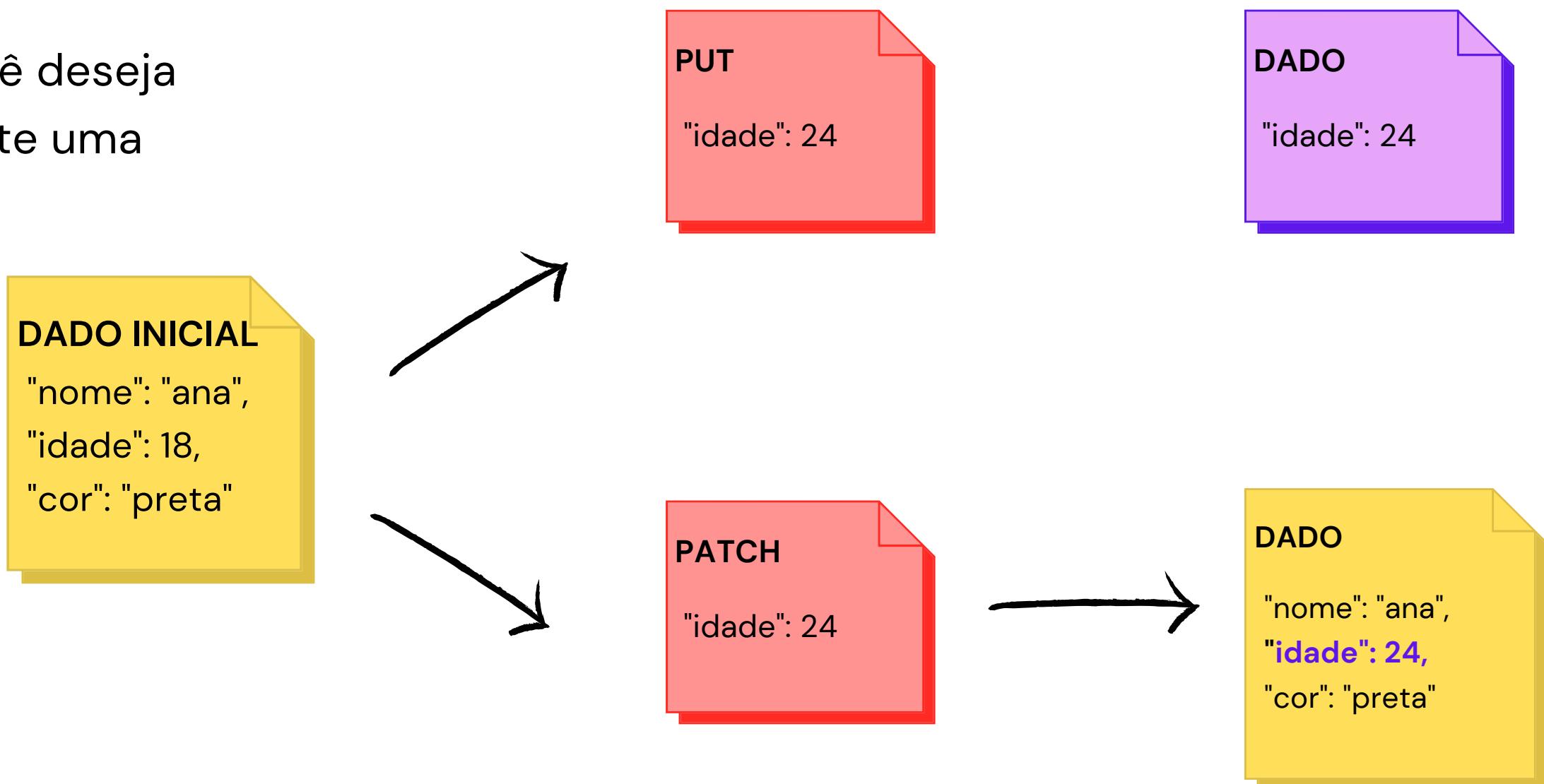
Server

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.

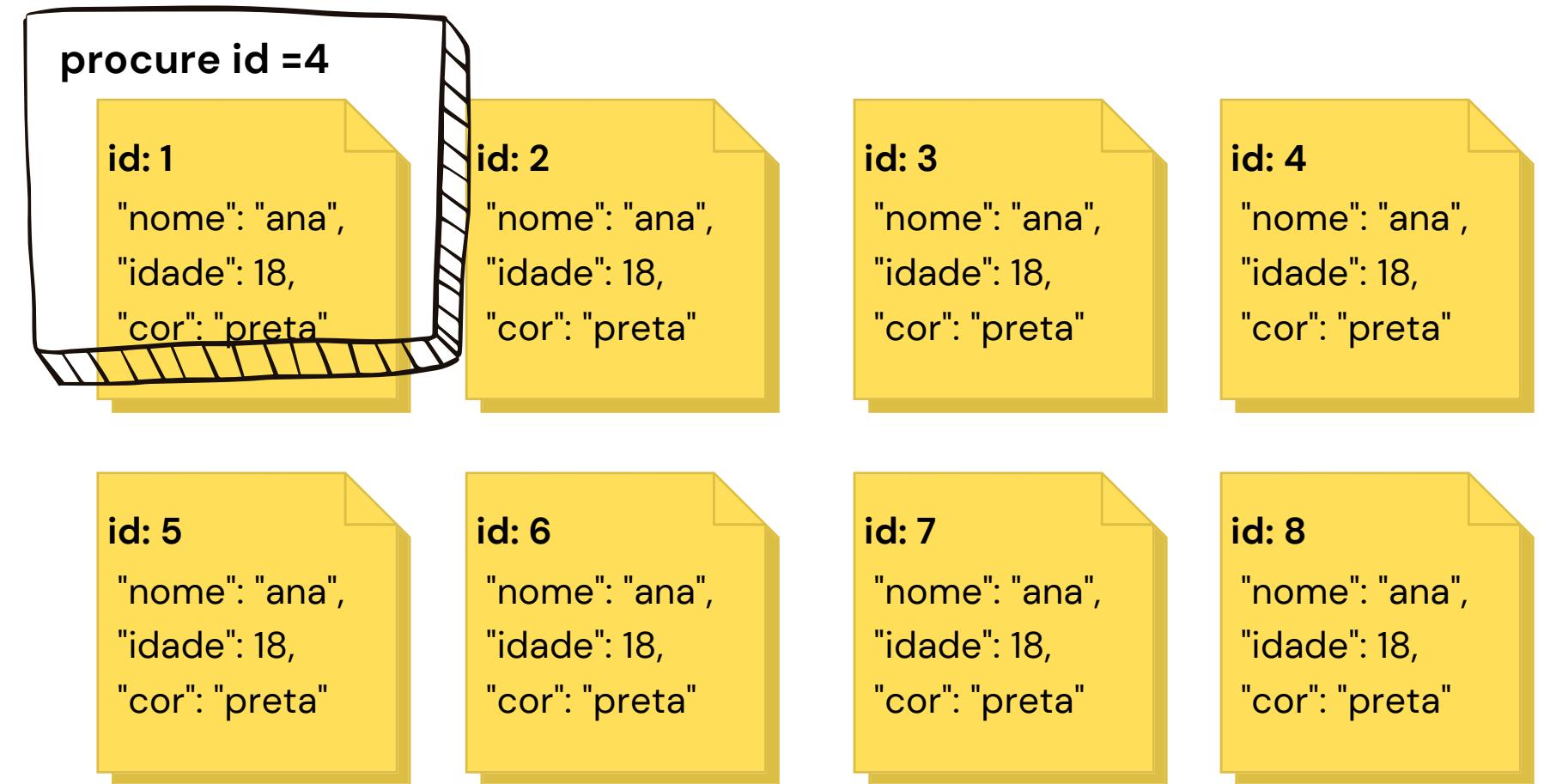


PUT x PATCH

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`

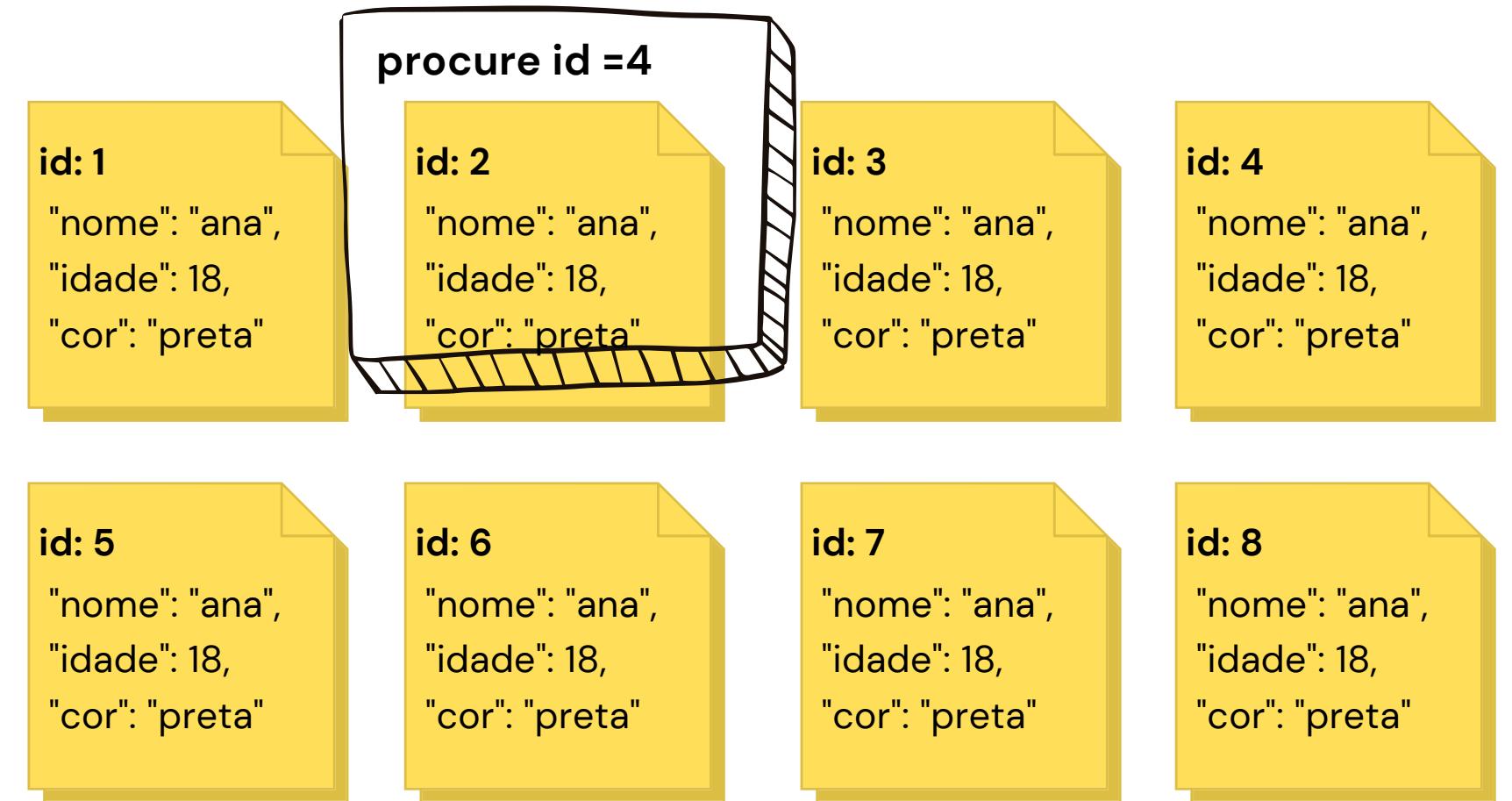


PUT x PATCH

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



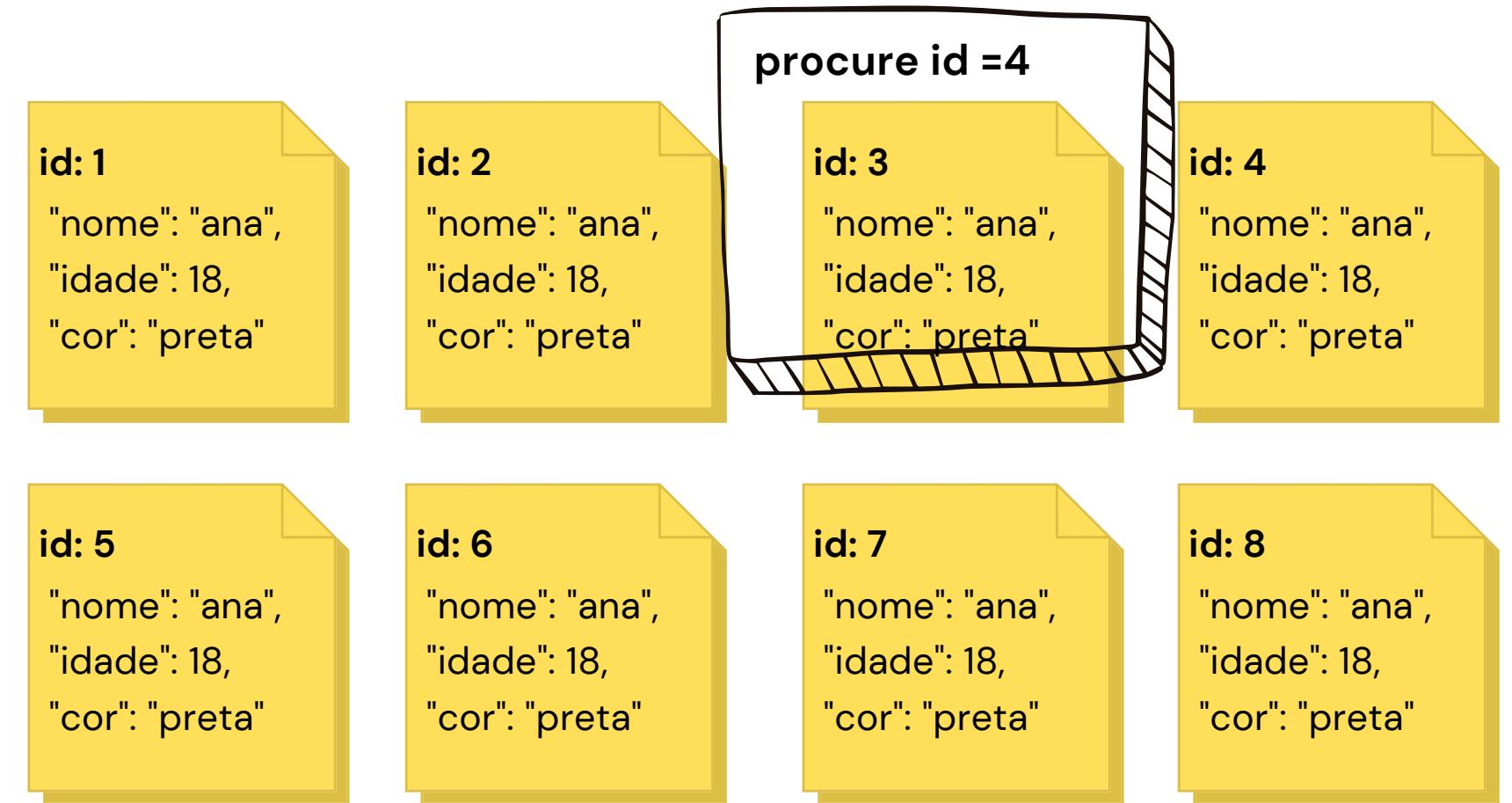
PUT x PATCH

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": 24,
"cor": "preta"

Continue a codar



a codar... a codar

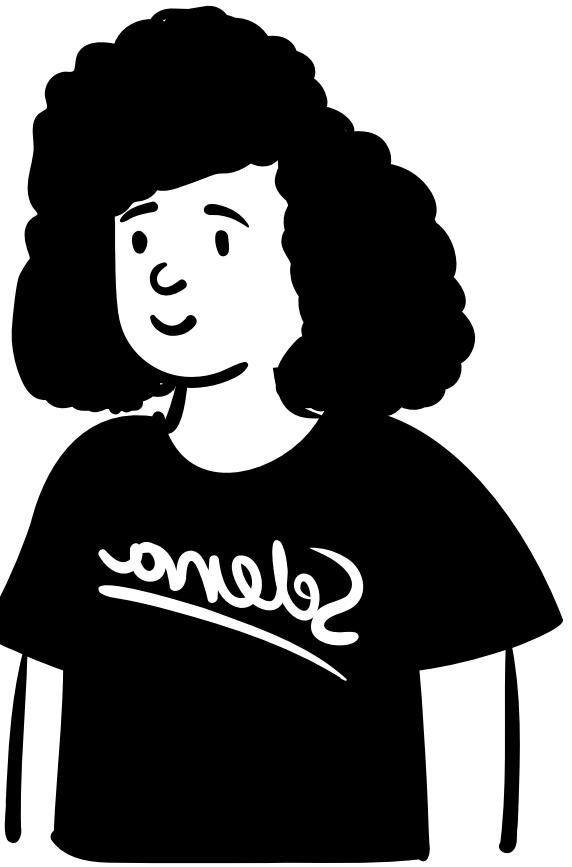
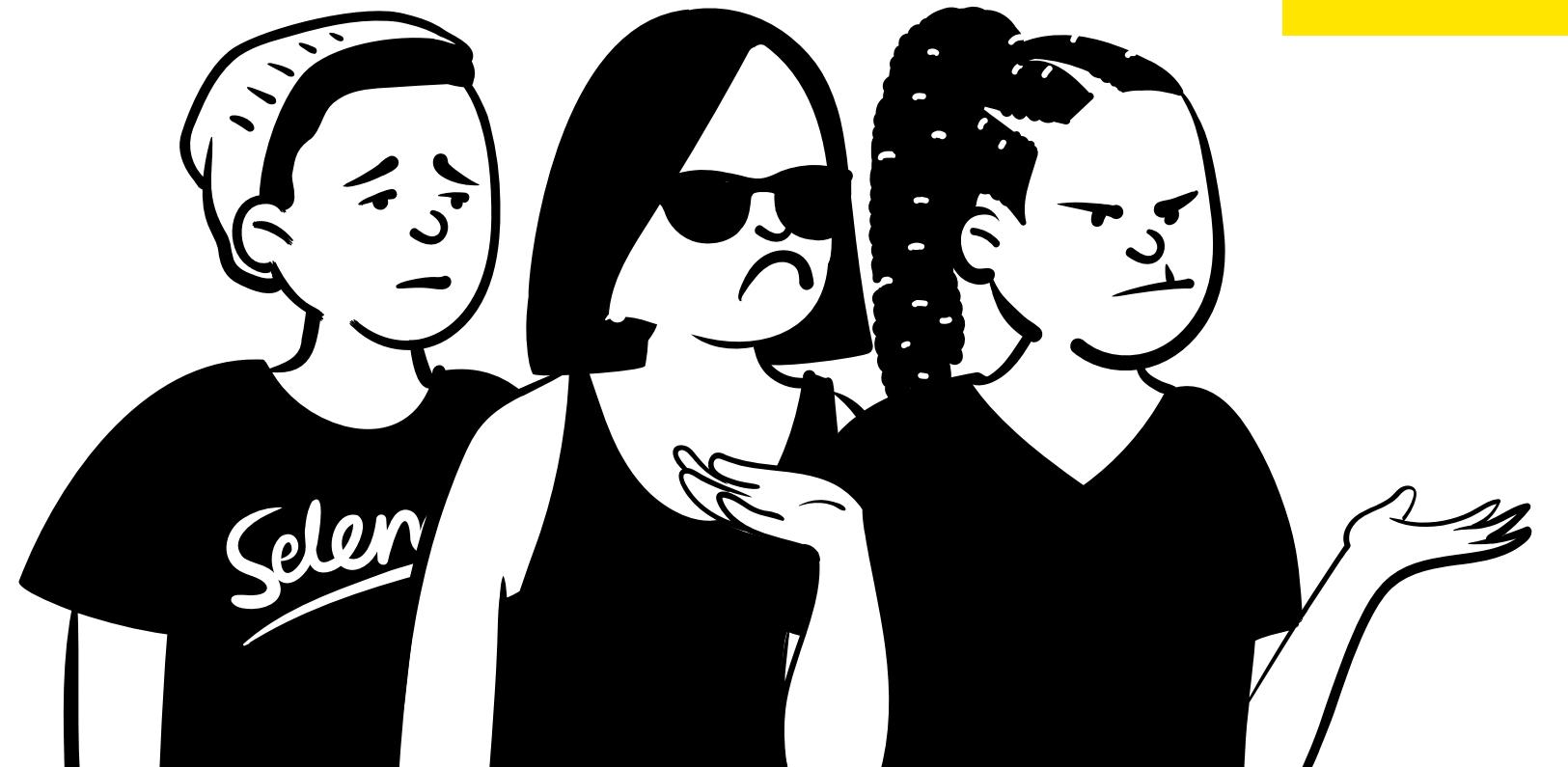
Demandas da API

Lá vem a galera de negócio....

- **[DELETE]/filmes/deletar/{id}**
 - deleta filme
- **[PUT]/filmes/update/{id}**
 - atualiza um filme por inteiro
- **[PATCH]/filmes/updateTitle?{id}**
 - atualiza titulo
- **[PATCH]/filmes/update/{id}**
 - atualiza o que vier no body

Para o lar

UHUUUL



Demandas da API

- [GET] /assistir
- [GET] /filmes
- [GET] /filmes/buscar/{id}
- [GET] /filmes/buscar?{titulo}
- [GET] /filmes/filtrar?{genero}
- [POST]/filmes/criar
- [PUT]/filmes/update/{id}
- [PATCH]/filmes/updateTitle?{id}
- [PATCH]/filmes/update/{id}
- [DELETE]/filmes/deletar/{id}
- [GET] /series
- [GET] /series{id}
- [GET] /series{titulo}
- [GET] /series{genero}
- [POST]/series/criar
- [PUT]/series/update/{id}
- [PATCH]/series/updateTitle?{id}
- [PATCH]/series/update/{id}
- [DELETE]/series/deletar/{id}



é...

a gente faz
tudo de novo,
o que sobro
e as series??

