

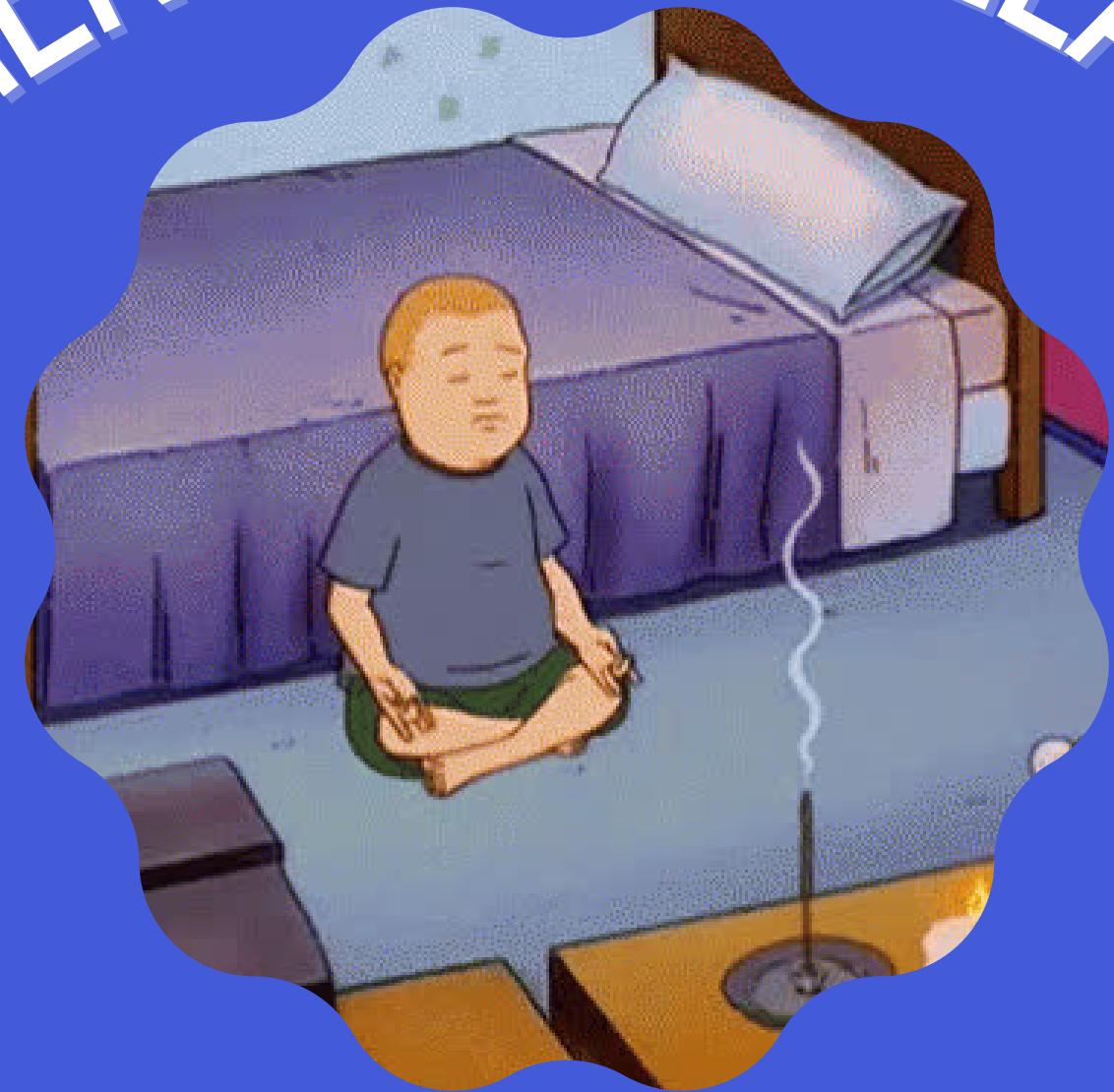
# PRIMEIRO SERVIDOR

## PUT - PATCH - DELETE

INICIAR

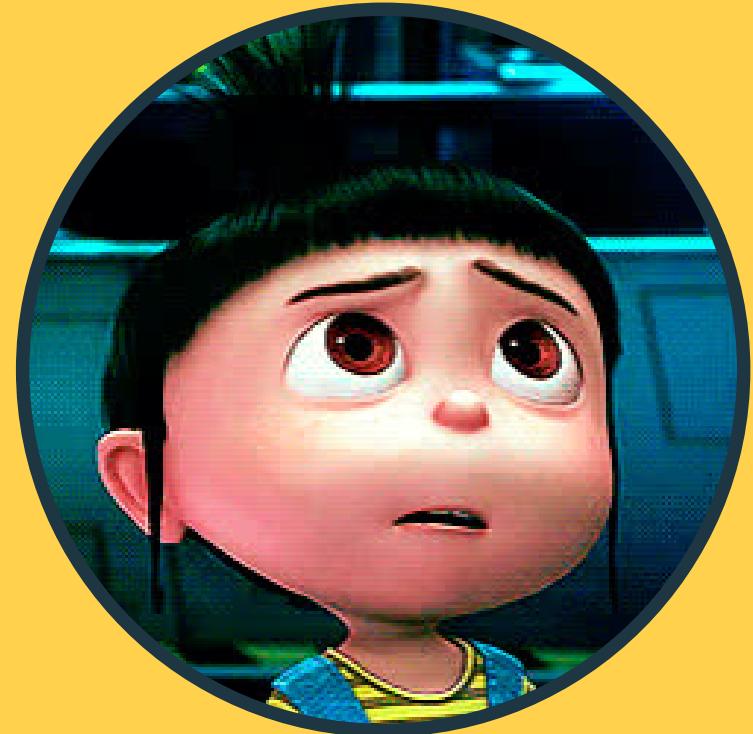


MOMENTO PARA RELAXAR



SURTO PARTE 2...

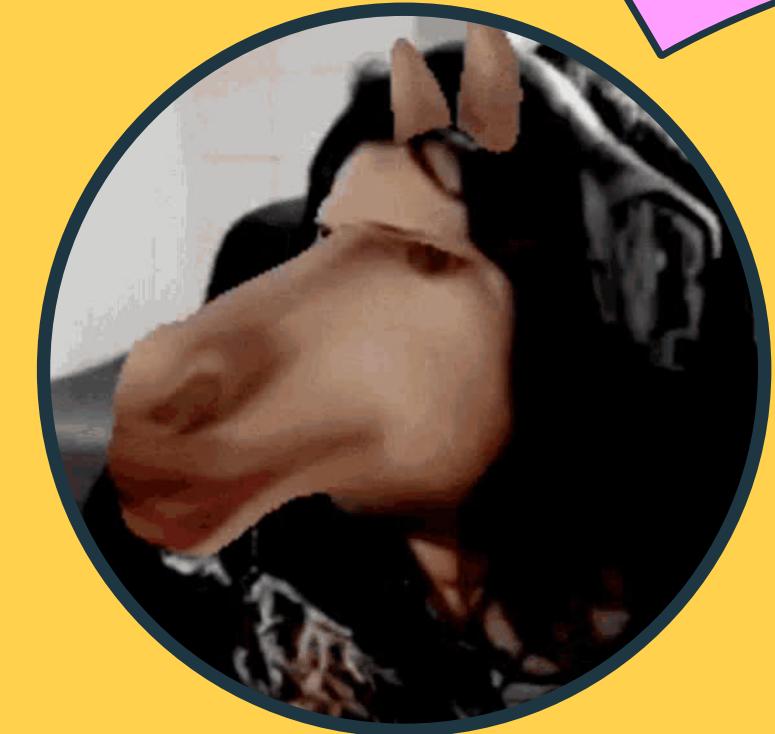
# ANTES DE COMEÇAR



Respire



Expire



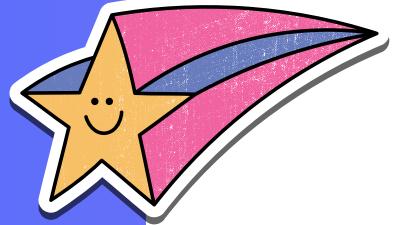
Não pira!!!!!!



HTTP - CRUD



RELEMBRANDO



CRUD é a composição da primeira letra de quatro operações básicas de um banco de dados.

Elas são o que a maioria das aplicações fazem.

- 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

CRUD	HTTP	REST
Create	<b>POST</b>	/api/movie
Read	<b>GET</b>	/api/movie/{id}
Update	<b>PUT</b>	/api/movie
Delete	<b>DELETE</b>	/api/movie/{id}

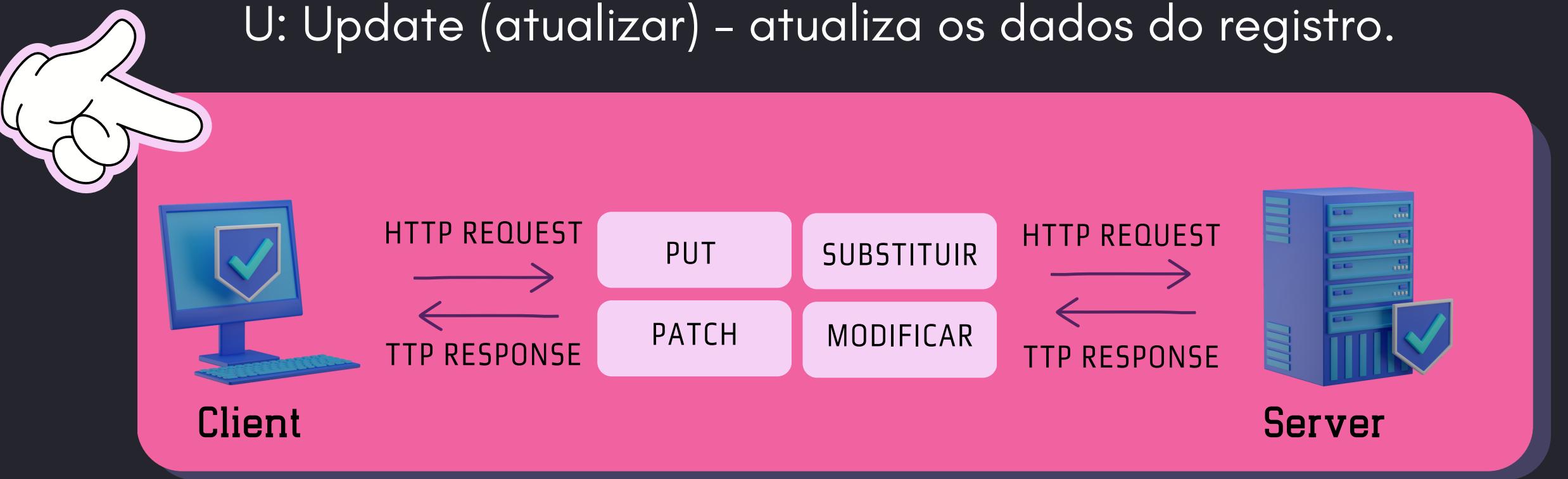
HTTP - PUT & PATCH



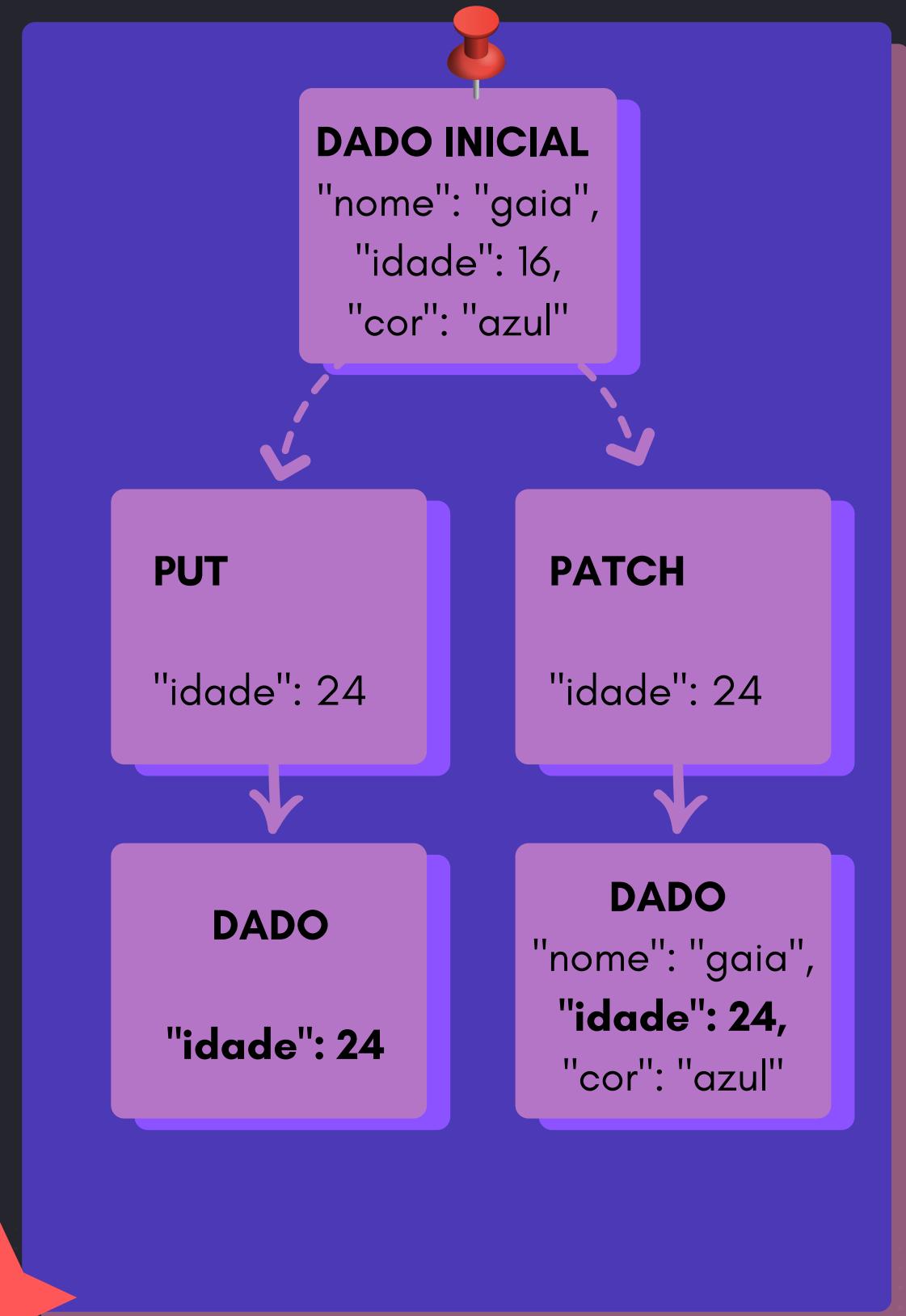
PARA RELEMBRAR...

# HTTP - PUT & PATCH

U: Update (atualizar) - atualiza os dados do registro.



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



# HTTP - PUT & PATCH

- Mas então por que ainda usamos o PUT?

Muitas vezes ainda usamos o PUT pela performance que ele tem quando relacionado a 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 de idade no dado de id=4

procure id= 4

The diagram illustrates a search interface for a database. At the top, a search bar contains the text "procure id= 4" and a magnifying glass icon. Below the search bar, a film strip icon indicates a search operation. Six data cards are displayed, each representing a record with an ID and three properties: nome, idade, and cor. An arrow points from the search bar to the fourth card, which corresponds to ID 4. The cards are arranged in two columns of three.

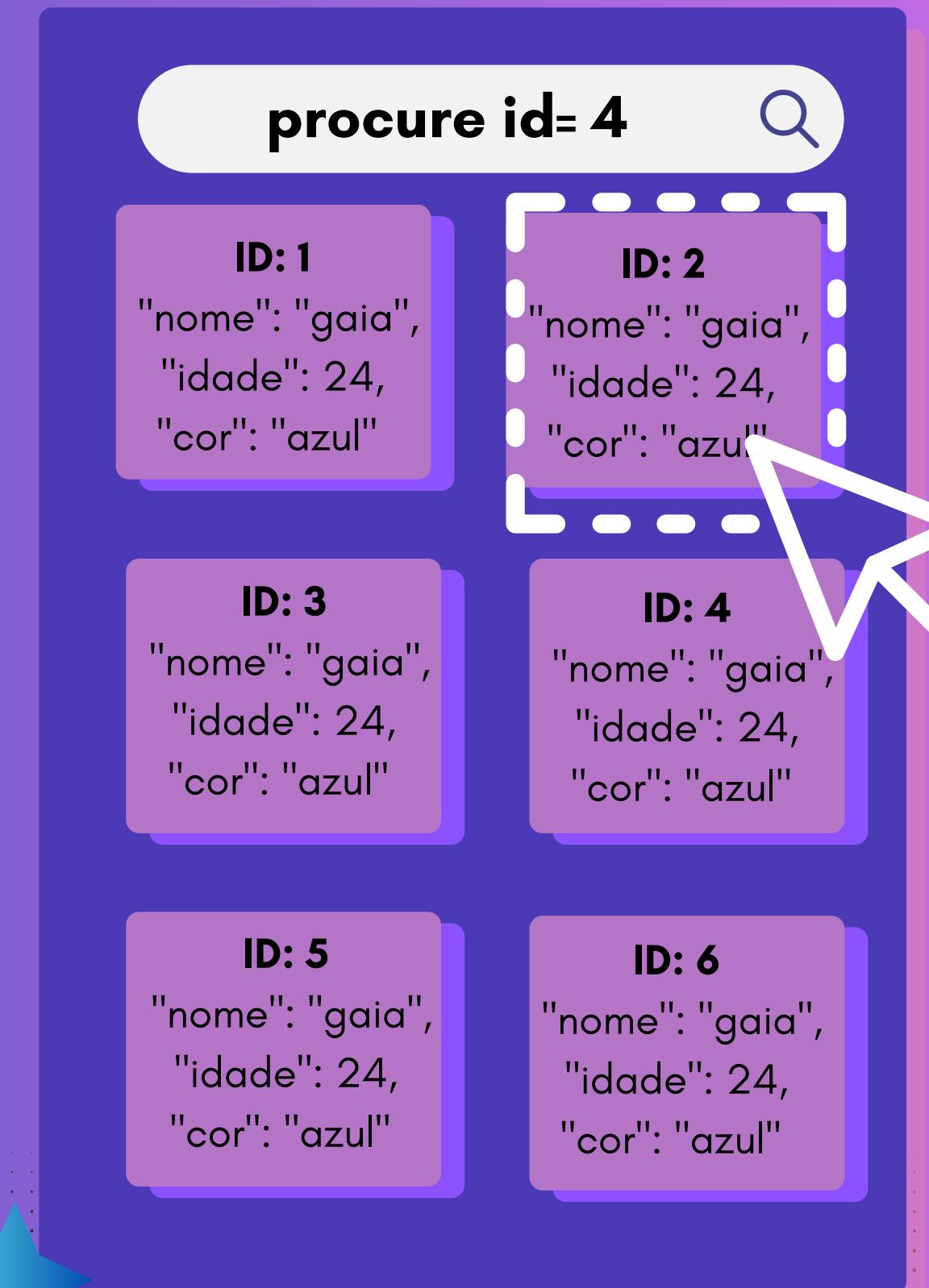
ID	nome	idade	cor
1	gaia	24	azul
2	gaia	24	azul
3	gaia	24	azul
4	gaia	24	azul
5	gaia	24	azul
6	gaia	24	azul

# HTTP - PUT & PATCH

- Mas então por que ainda usamos o PUT?

Vamos simular a uma edição do campo de idade no dado de id=4

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



# HTTP - PUT & PATCH

- Mas então por que ainda usamos o PUT?

Vamos simular a uma edição do campo de 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

procure id= 4

ID: 1  
"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 2  
"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 3  
"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 4  
"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 5  
"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 6  
"nome": "gaia",  
"idade": 24,  
"cor": "azul"

# HTTP - PUT & PATCH

- Mas então por que ainda usamos o PUT?

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



procure id= 4

ID: 1

"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 2

"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 3

"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 4

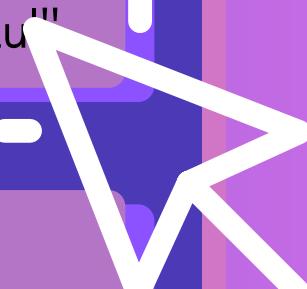
"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 5

"nome": "gaia",  
"idade": 24,  
"cor": "azul"

ID: 6

"nome": "gaia",  
"idade": 24,  
"cor": "azul"



## HTTP - PUT & PATCH

Em contrapartida, se escolhermos um método PATCH, a procura continuaria, só que agora é dentro do dado.



procure id= 4



procure idade



ID: 4

"nome": "gaia",  
"idade": 24,  
"cor": "azul"





## HTTP - PUT & PATCH

Quando encontrado a propriedade aí sim o dado seria modificado.

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

**procure id= 4**



**procure idade**

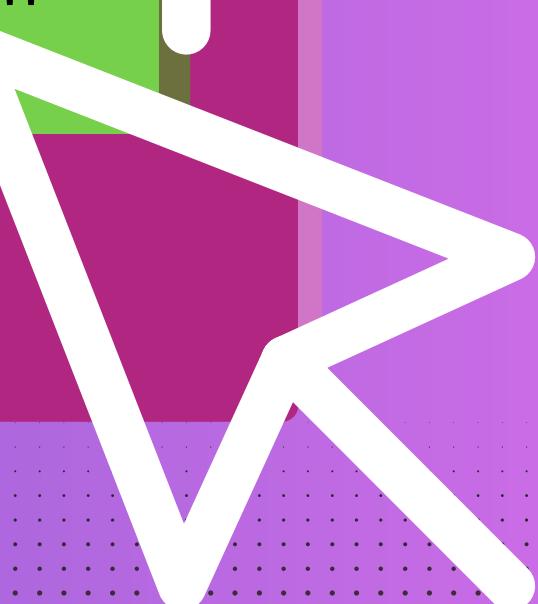


**ID: 4**

"nome": "gaia",

**"idade": 24,**

"cor": "azul"



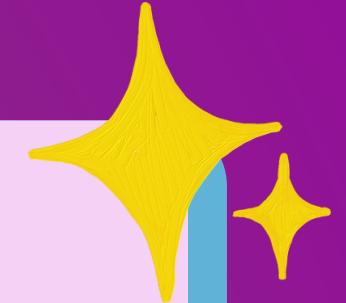
AGORA O MAIS TEMIDO DE TODOS



D E L E T E



## HTTP - DELETE



Usamos o método DELETE para remover um recurso ou uma coleção de recursos. Quando em um formulário você clica no botão de “Excluir”, o evento que está sendo disparado passa pelos recursos do método DELETE.



Fácil né? kkkkkkkkkk desculpa o susto

MOMENTO ALEGORIA



E SE...

# Uma divindade decide criar um planeta, ao decorrer do tempo mudanças vão ocorrendo



**POST**

Dante e Aristóteles  
são criados



**GET**

As sereias encontram estrelas  
e as pegam para usar de sutiã



**PATCH**

A divindade altera  
cavalos por unicórnios



**PUT**

Cobras se transformam  
em drações

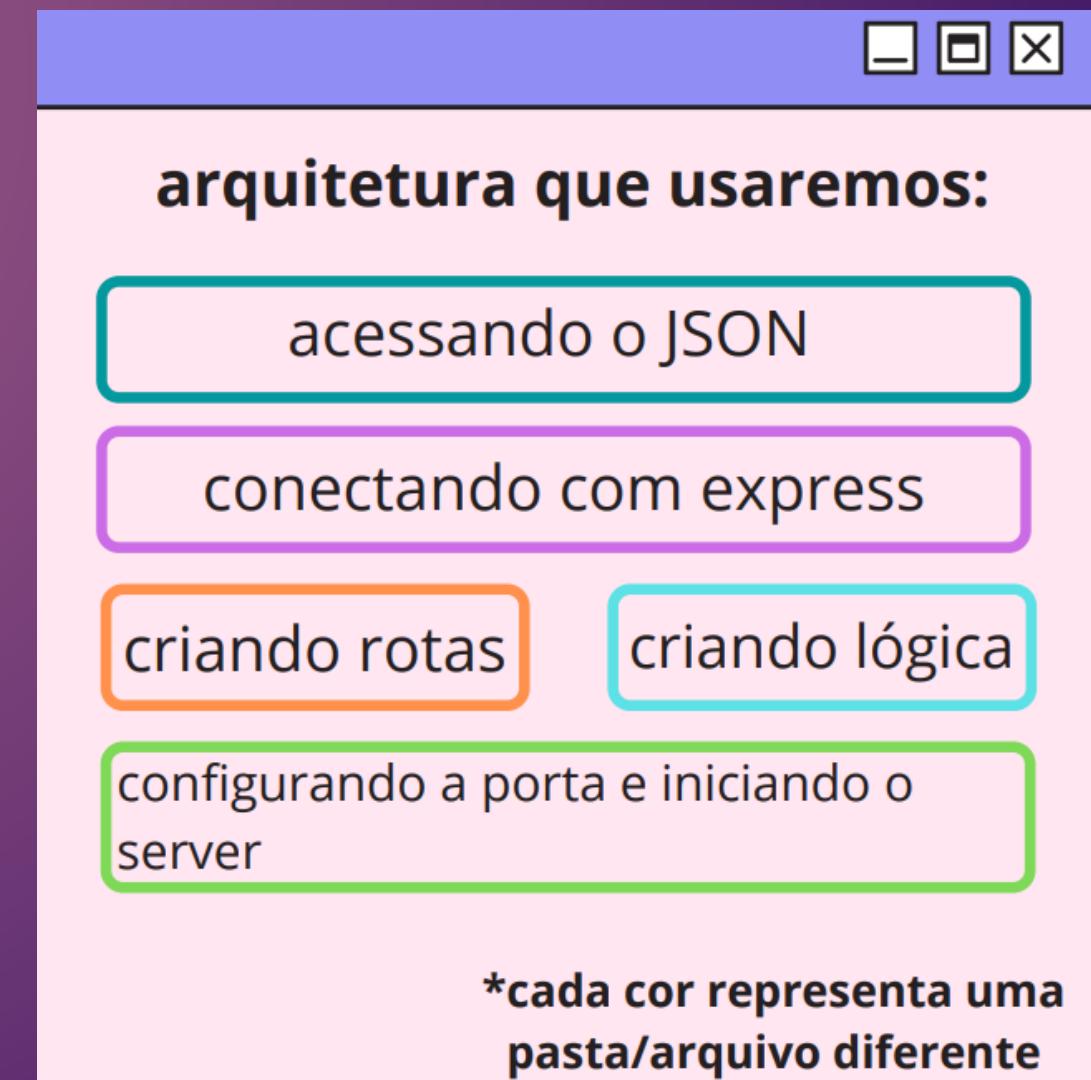
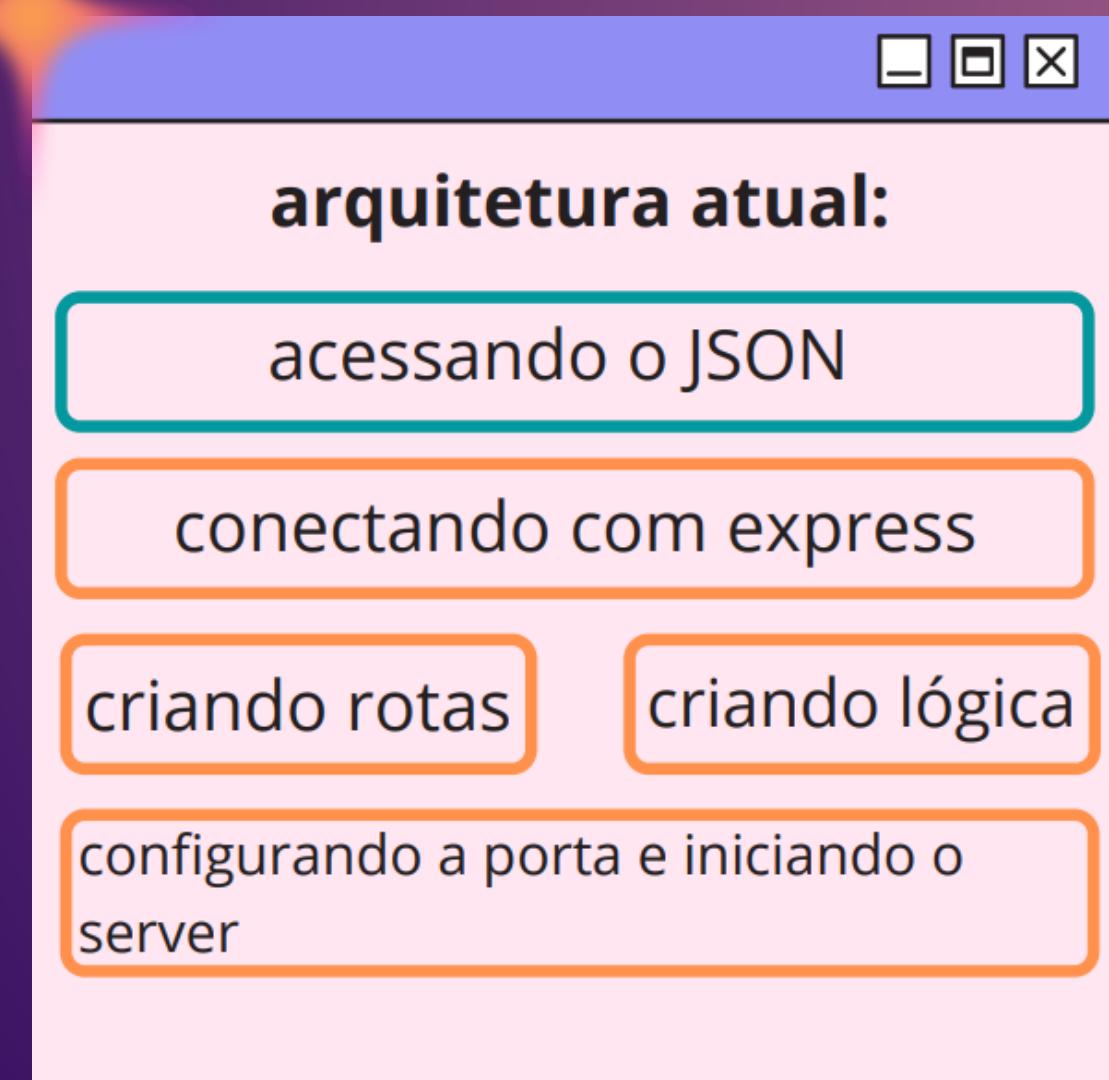


**DELETE**

Divindade cansada de  
problema, manda o  
apocalipse



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 lidamos com as rotas(routes). O MVC nada mais é que uma forma de organizar o nosso código.



\*cada cor representa uma pasta/arquivo diferente

# ARQUITETURA MVC



**arquitetura que usaremos:**

acessando o JSON

conectando com express

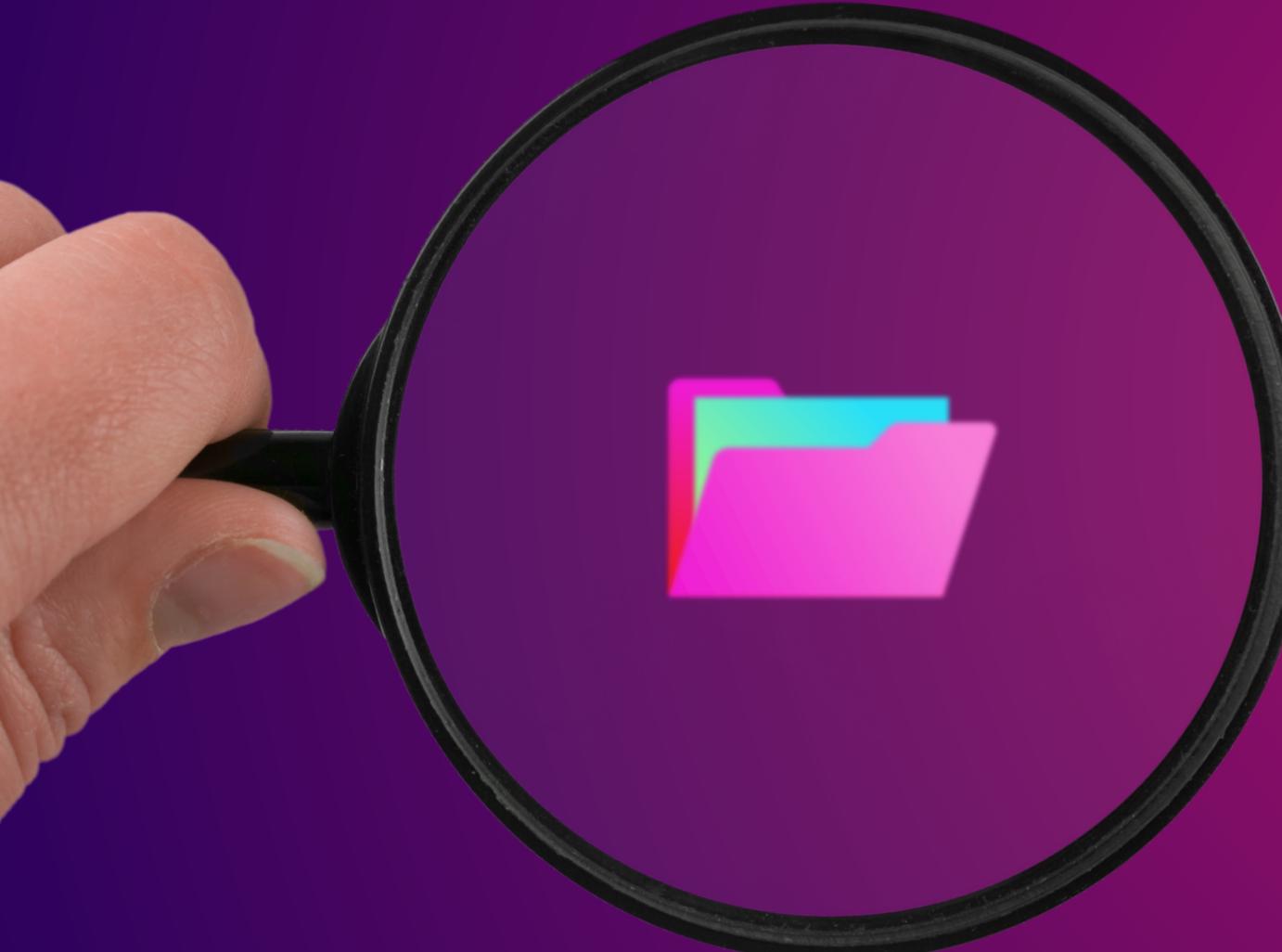
criando rotas

criando lógica

configurando a porta e iniciando o server

\*cada cor representa uma pasta/arquivo diferente

# ARQUITETURA MVC



## ARQUITETURA MVC - EXEMPLO

### PROJETO (ESSA É A PASTA RAIZ)

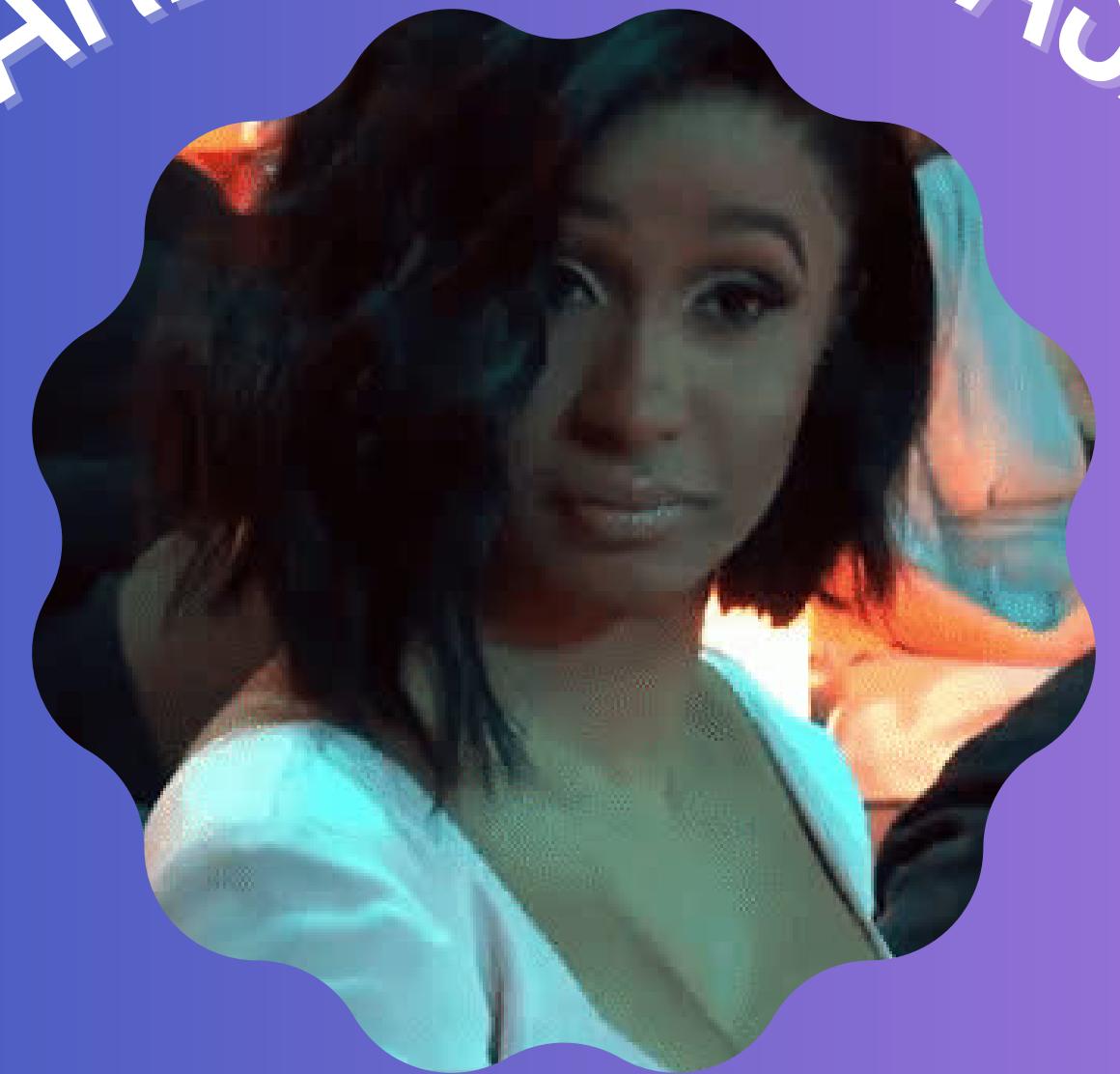
```
|  
|- NODE_MODULES ( DENTRO DA PASTA RAIZ)  
|  
|- SRC (DENTRO DA PASTA RAIZ)  
|  
|   |- CONTROLLERS (DENTRO SRC)  
|   |   |- XCONTROLLER.JS  
|  
|   |- MODELS (DENTRO SRC)  
|   |   |- XMODEL.JS  
|  
|   |- ROUTES (DENTRO SRC)  
|   |   |- XROUTES.JS  
|  
|   |- APP.JS (DENTRO SRC)  
|  
|- .GITIGNORE (PASTA RAIZ)  
|- PACKAGE-LOCK.JSON (PASTA RAIZ)  
|- PAKAGE.JSON (PASTA RAIZ)  
|- README.MD (PASTA RAIZ)  
|- SERUER.JS (PASTA RAIZ)
```

EVAMOS DE VS CODE



BORA CODAR?

TAREFA PARA CASA



CUIDADO PARA NÃO ACUMULAR