

PROGRAMAÇÃO WEBII

1

API REST

História

O que é uma API...

- Você pode até não saber o que é API, mas elas certamente já fazem parte de sua rotina:
 - ▷ efetuar um pagamento online,
 - ▷ reservar um hotel para as férias,
 - ▷ usar o Google Maps para descobrir como chegar a um endereço,
- São apenas alguns exemplos de atividades comuns de nosso dia a dia e que funcionam por meio de APIs.



O que é uma API...

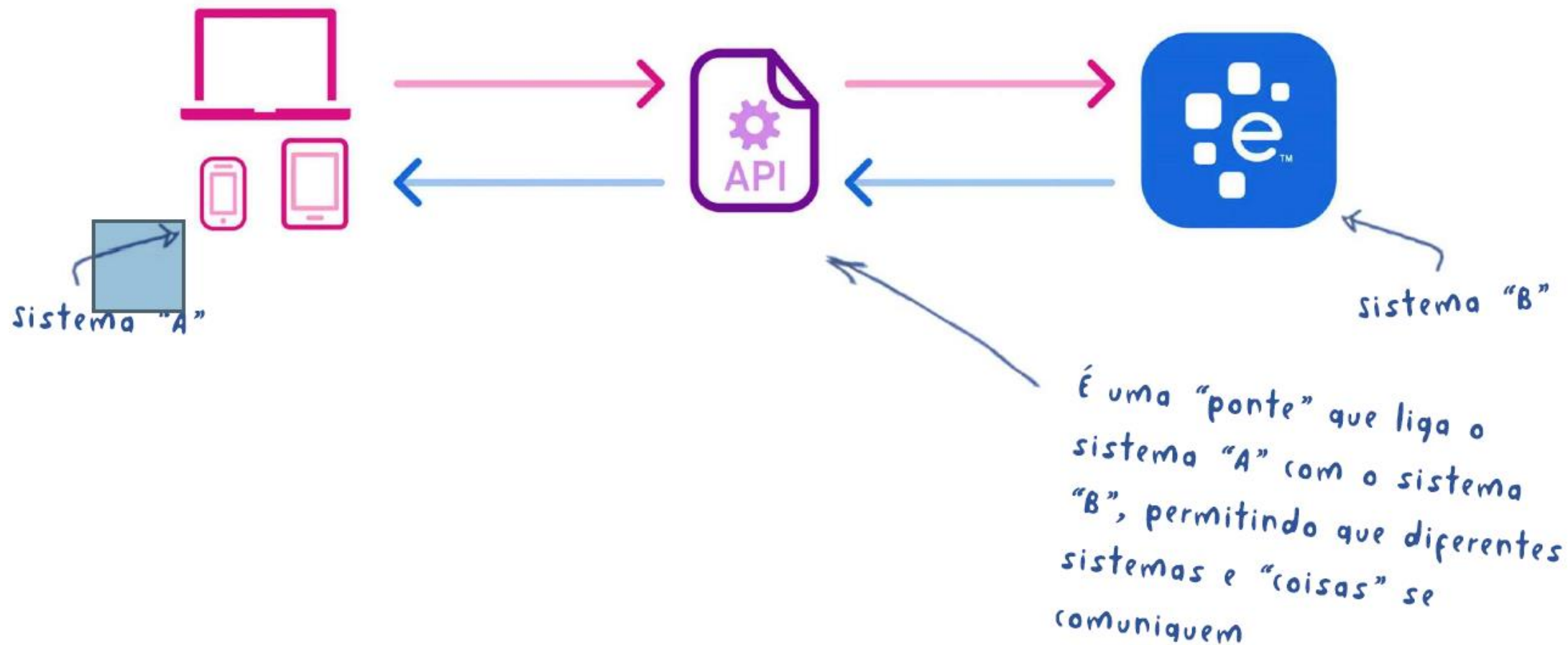
- API é a sigla utilizada para Application Programming Interface ou, em português, Interface de Programação de Aplicativos.
- Através das APIs, os aplicativos podem se comunicar uns com os outros sem conhecimento ou intervenção dos usuários.

O que é uma API...



Servem para integrar sistemas independente da linguagem de programação, proporcionando uma troca de informações de forma muito segura.

O que é uma API...



Como a API funciona...

- Em um app que utiliza GPS, por exemplo, não programamos linhas de código para conectar-se a um satélite para obter as coordenadas do local onde o usuário se encontra. O Google já possui tudo isso desenvolvido e, assim, a 99 Táxi, Uber, Cabify, Correios e etc podem, simplesmente, chamar a API do Google Maps.
- Outro exemplo é um site que utiliza informações do Facebook/Gmail para cadastrar novos usuários: através das APIs, o site envia uma requisição solicitando as informações necessárias para que esse novo usuário possa criar uma conta, utilizando tudo o que precisa e que já está presente nos servidores do Facebook/Gmail.

As APIs já são realidade para diversos usos e seu crescimento ainda vai longe

Como a API funciona...

**Qualquer dispositivo
(celular, PC, Carro, Totem,
Site, Aplicativo) com
conectividade com a
Internet pode usar uma
API, independentemente do
sistema operacional ou da
linguagem de programação.**

Por que usar APIs?

- Com as APIs, uma solução ou serviço pode se comunicar com outros produtos e serviços sem a necessidade de saber como foram implementados, simplificando o desenvolvimento das aplicações e gerando economia de tempo e dinheiro.
- A interface dos novos hardwares interconectados, wearables e, em breve, os carros sem motorista mostram como as APIs se tornaram importantes em nossas vidas.



Gigantes da tecnologia têm orientado seus sistemas para um futuro centrado em APIs



Google

Representações

Vimos que uma API permite a interoperabilidade entre usuários e aplicações. Com isso, é muito importante pensarmos em algo padronizado e, de preferência, de fácil representação e compreensão por humanos e máquinas.

Qual dos exemplos abaixo você escolheria para informar o endereço em uma carta?

Derivado do JavaScript, JSON é um acrônimo de JavaScript Object Notation.

Representações

Derivado do JavaScript, JSON é um acrônimo de JavaScript Object Notation.

```
<?xml version="1.0" encoding="UTF-8"?>
<endereco>
  <rua>Rua Alcântara,113</rua>
  <cidade>São Paulo</cidade>
</endereco>
```

↑ A representação em XML é mais verbosa e exige um esforço extra de quem está escrevendo

```
{
  "endereco": {
    "rua": "Rua Alcântara,113",
    "cidade": "São Paulo"
  }
}
```

↑ O JSON além de ser um formato leve para troca de dados é também muito simples de ler

Representações

Derivado do JavaScript, JSON é um acrônimo de JavaScript Object Notation.



CHECK LIST

Vantagens do JSON

- Leitura mais simples
- É tipado
- Arquivo com tamanho reduzido
- Velocidade maior na execução e transporte de dados
- Quem utiliza? Google, Facebook, Yahoo!, Twitter...

No passado...

- Antigamente, as aplicações eram grandes e pesadas, conversavam com banco de dados e até mainframe, faziam transações distribuídas e tornavam qualquer plataforma confiável para execução.
- Porém, muitas vezes surgiam o velho e conhecido problema de reaproveitamento de código: depois de vários sistemas feitos por diferentes equipes, descobria-se que todos eles tinham um mesmo cadastro de usuários.
- Que tal isolar essa parte em um sistema único?



SOAP

Assim, na virada do século, surgiu o SOAP que fornecia serviços (internos e externos) usando o padrão de comunicação XML. Assim, os sistemas de diferentes linguagens e sistemas operacionais passaram a trocar informações, iniciando a era da arquitetura orientada a serviços - Service-Oriented Architecture (SOA).



SOAP significa simple object
Access Protocol



O problema do SOAP era a sua complexidade em fazer qualquer coisa: um servidor de um lado e obrigatoriamente um cliente do serviço de outro, trafegando XML para ambos os lados, em cima do protocolo usado na internet (HTTP).

Origem do REST

- Uma alternativa bem simples ao SOAP foi proposta por Roy Thomas Fielding, um dos fundadores do projeto Apache HTTP, em sua tese de doutorado intitulada "Architectural Styles and the Design of Network-based Software Architectures": o famoso Representational State Transfer (REST).
- Essa simples alternativa ao SOAP aproveita os métodos existentes no protocolo HTTP para fazer as operações mais comuns existentes nos serviços, tais como: busca, inserção e atualização.



Roy Fielding

Pela simplicidade e rapidez, o REST foi rapidamente adotado pelo mercado como o estilo arquitetônico.

Então, REST é...

REST define um conjunto de regras e boas práticas para o desenvolvimento de APIs que possibilitam a execução de solicitações e o recebimento de respostas via protocolo HTTP, como GET e POST, permitindo a comunicação entre aplicações.

E RESTful... qual a diferença?

Existe uma certa confusão quanto aos termos REST e RESTful. Entretanto, a diferença é apenas gramatical: sistemas que utilizam os princípios REST são chamados de RESTful.

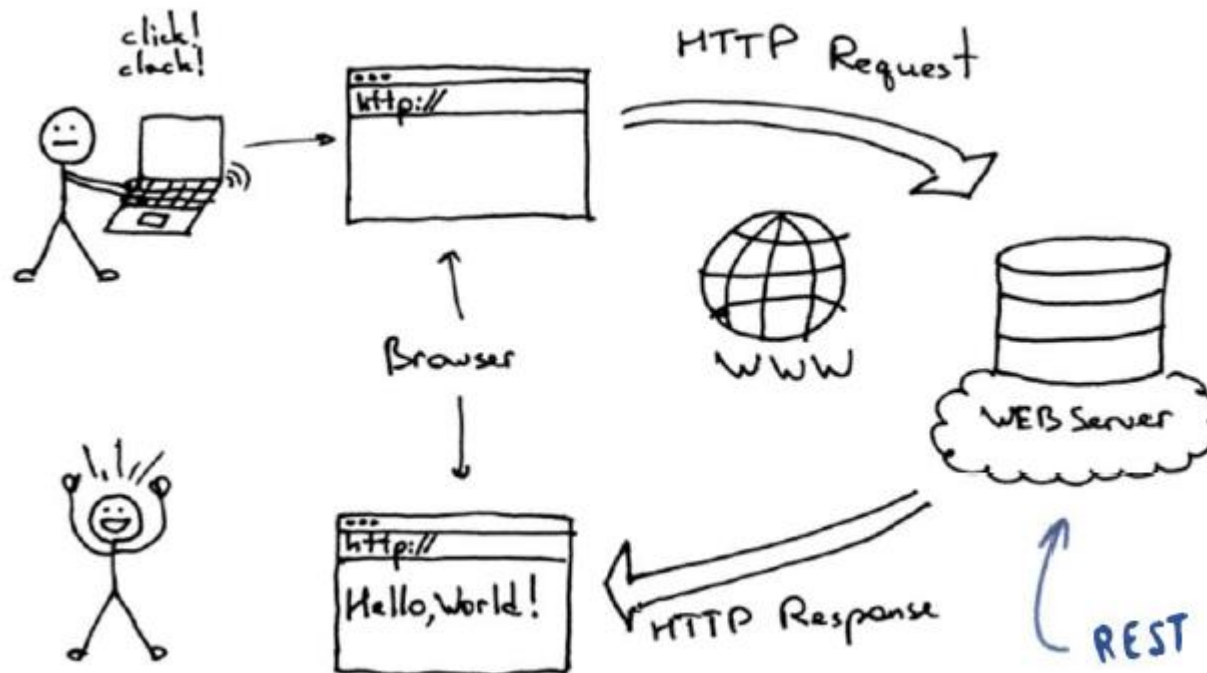


X

RESTful

Como o REST funciona?

Toda a comunicação da interface REST é feita via web, ou seja, através de uma requisição (pedido feito pelo cliente) a uma URI (Uniform Resource Identifier), que referencia um recurso, utilizando um dos quatro métodos HTTP (GET, PUT, POST ou DELETE) que, por sua vez, traz uma resposta.



Método: informa o tipo de ação

1 GET http://www.etechoracio.com.br/usuarios

URI

Pela simples leitura (mesmo o método GET sendo em inglês) é possível entender que estamos pegando (GET) uma lista de todos os usuários que estão cadastrados no sistema.

Recurso: tipo de informação buscada

2 DELETE http://www.etechoracio.com.br/usuarios/valter

No exemplo acima, estamos apagando (DELETE) o usuário valter.

Item desejado

3 POST http://www.etechoracio.com.br/usuarios

Agora, estamos usando o método POST, em que recebemos o envio de dados extras para cadastrar um novo usuário.

Os dados enviados via POST podem ser do tipo JSON

2

Como testar API REST com Insomnia

Praticando...

Vamos lá!

■ As APIs REST trabalham com os métodos http:

- ▷ GET
- ▷ POST
- ▷ PUT
- ▷ PATCH
- ▷ DELETE

■ Como segue no próximo slide

Métodos HTTP

ENDPOINT	MÉTODO	AÇÃO
/produtos	GET	Retorna uma lista de Produtos
/produtos/{id}	GET	Retorna somente o Produto com id = {id}
/produtos	POST	Insere um novo Produto
/produtos/{id}	PUT	Altera os dados do Produto com id = {id}
/produtos/{id}	PATCH	Altera alguns dados do Produto com id = {id}
/produtos/{id}	DELETE	Deleta o Produto com id = {id}

Instalando o Insomnia

■ Vamos acessar o site do Insomnia:

▷ <https://insomnia.rest/>

■ Selecione: GET STARTED FOR FREE

Build APIs that work.

Deliver high quality APIs through standards and collaboration with the Insomnia API design platform.

[Get Started for Free](#)

[Explore Plugins](#)

Instalando o Insomnia

Na opção Free, vamos clicar no download APP

Pricing

Free

\$0

user / year

[Download App](#)

Free updates forever



Unlimited Devices

API Client



Design APIs



Test APIs



E2EE Sync

E2EE Team Sync [↗](#)

User Management

Community Support [?](#)

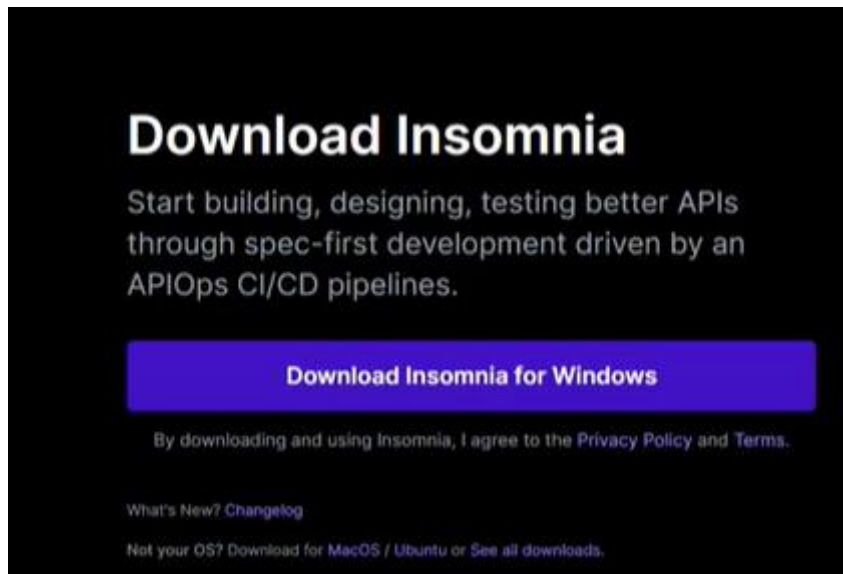


Priority Support

[Download App](#)

Instalando o Insomnia

- Ele já direciona para o download de acordo com o SO, mas tem versões para Ubuntu, Mac..



- Vamos iniciar o download

Instalando o Insomnia

Alguns anos depois...



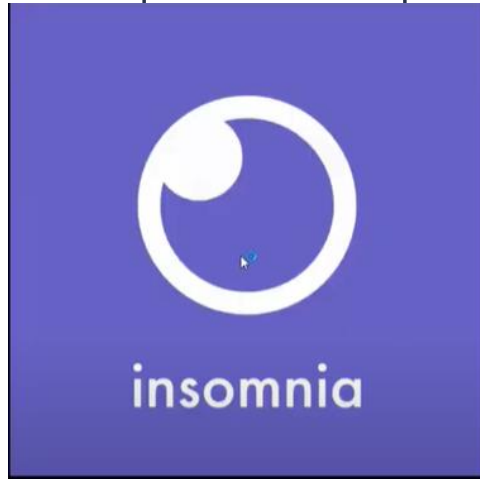
Instalando o Insomnia

Finalizou o download...



Instalando o Insomnia

- 2 cliques no arquivo para instalar:



- Pronto... Está instalado... É muito rápido e fácil.



Instalando o Insomnia

- Enquanto isso, vamos para a pasta e dar start no nosso servidor:
 - ▶ Prompt de comando
 - ▶ Acessar a pasta do seu projeto
 - ▶ Npm start
- Irá começar a subir o servidor com o arquivo db.json e temos aqui os recursos: posts, comments e profile.
- Minimize a tela e vamos retornar para o Insomnia

```
> backend@1.0.0 start C:\angular\cursoCPS\backend
> json-server --watch db.json

\(^_^)/ hi!

Loading db.json
Done

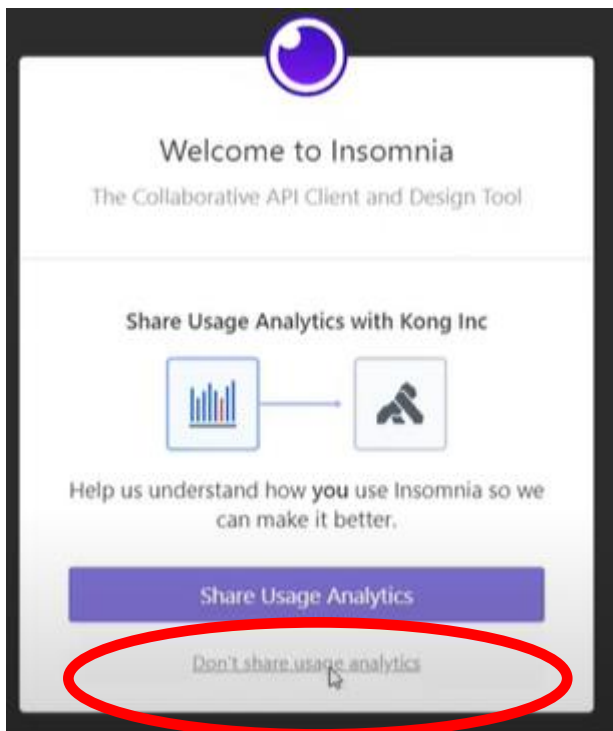
Resources
http://localhost:3000/posts
http://localhost:3000/comments
http://localhost:3000/profile

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

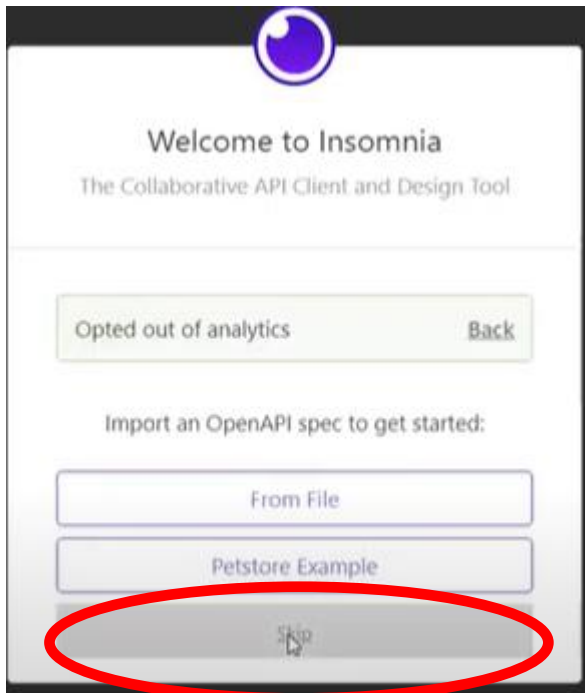
Instalando o Insomnia

- Retornando para o Insomnia...
- Pergunta se desejo compartilhar as estatísticas, respondo que não:



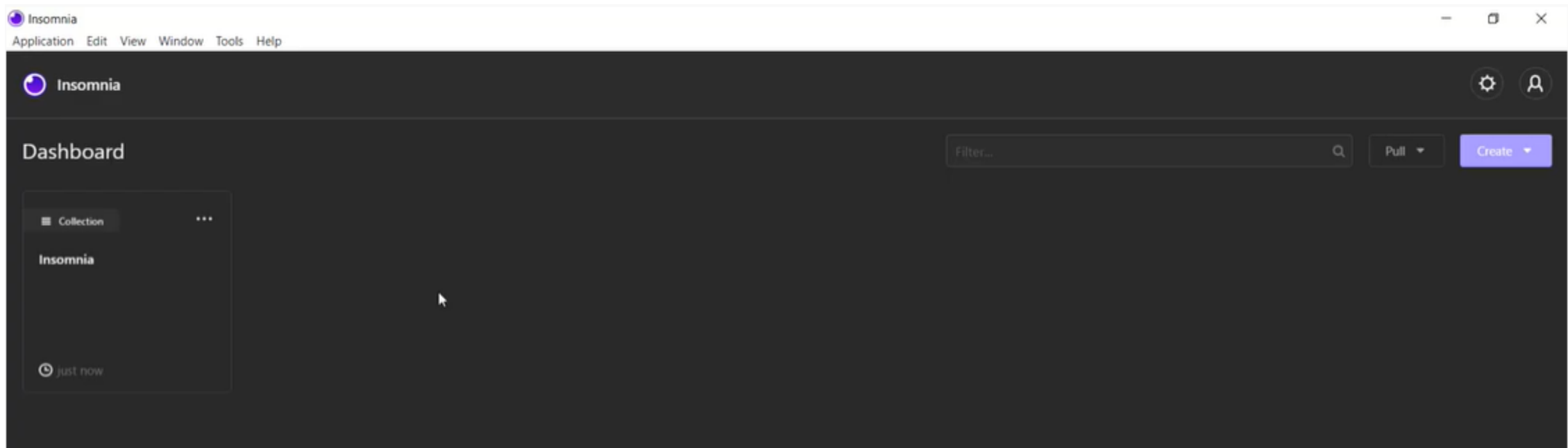
Instalando o Insomnia

■ Posso pular aqui...



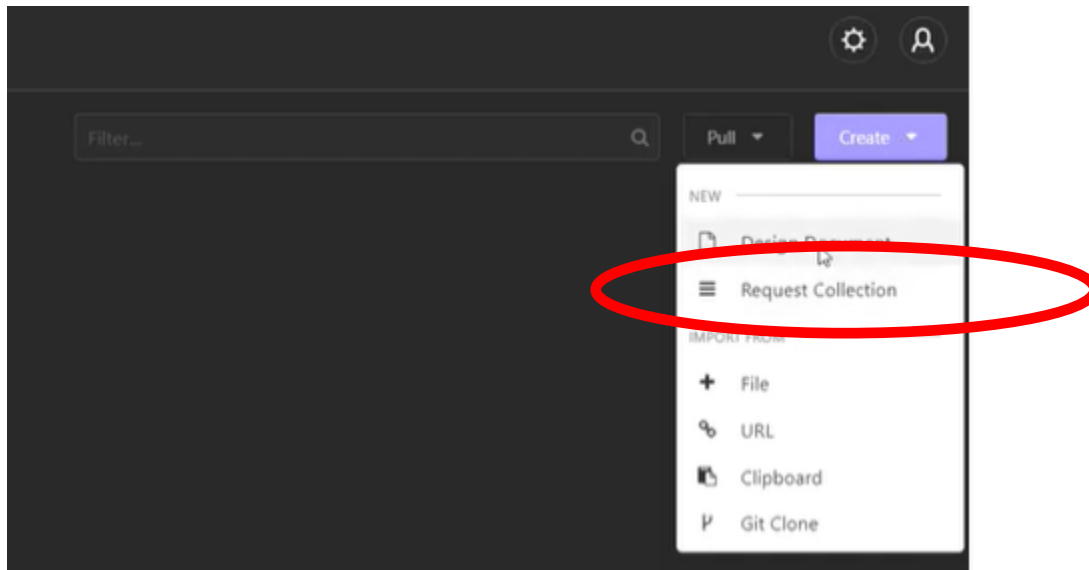
Instalando o Insomnia

Pronto, estamos com o Insomnia pronto.



Testando...

- Vamos clicar no create / Request Collection

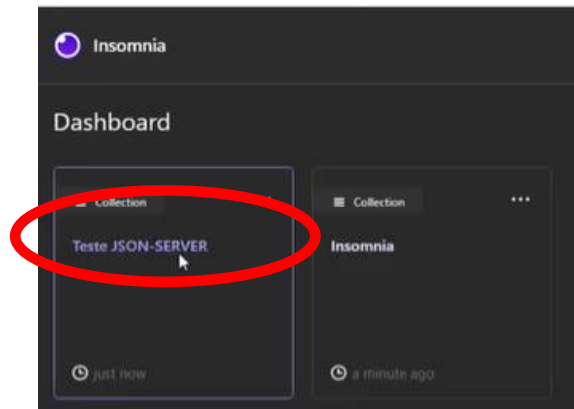


Testando...

- Vamos dar o nome de Teste JSON-SERVER

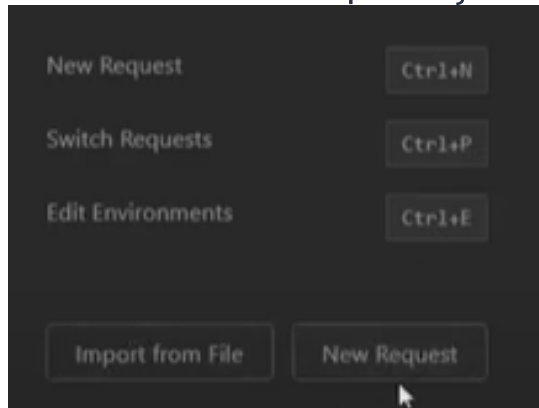


- Ele criou dentro da dashboard uma coleção para realizarmos os testes. Só clicar no Dashboard, depois clicar no Teste JSON-SERVER



Testando...

- Aqui dentro podemos criar importar de arquivos ou criar uma nova requisição. Vamos criar uma nova requisição



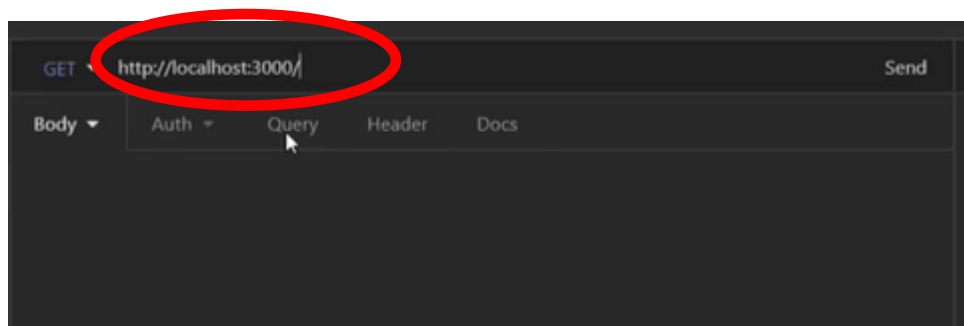
- Temos que dar um nome, vamos chamar de Buscar todos os posts e será do tipo GET. Clicar em CREATE



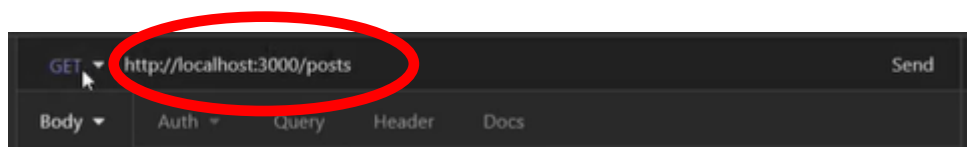
Testando...

Perguntará qual o caminho da API. Vamos testar no browser se está ok e copiamos para aqui:

▶ Localhost:3000

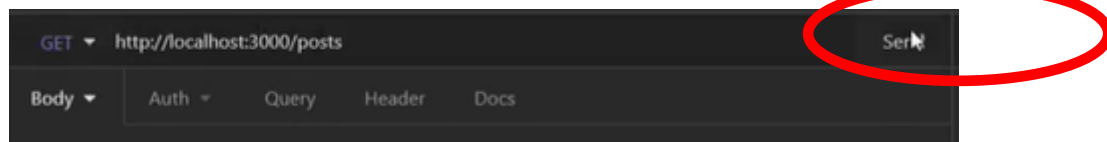


Vamos testar a API, mas somente os POSTS, então precisamos acrescentar post após o endereço da API

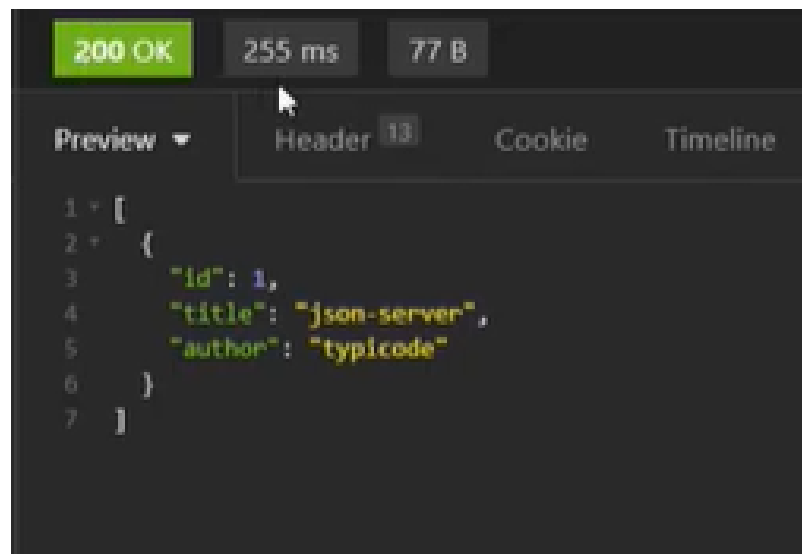


Testando...

- Como essa requisição é do tipo GET e não precisamos passar nenhum parâmetro, vamos clicar no SEND:

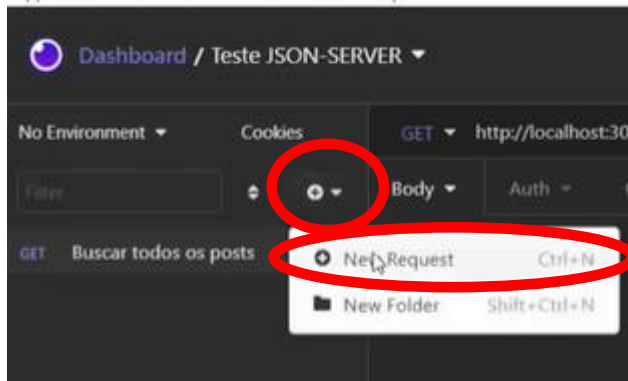


- Irá exibir o preview, informando que o status é 200



Testando...

Agora vamos criar uma nova requisição



O nome dessa requisição será: Adicionar Post, esse tipo será POST e iremos enviar um arquivo JSON. Clicar no botão CREATE:

A screenshot of the 'New Request' dialog box in Postman. The dialog has a title bar 'New Request' with a close button. Below the title bar, there's a 'Name' field with the placeholder text '(defaults to your request URL if left empty)'. The 'Name' field contains the text 'Adicionar Post'. To the right of the 'Name' field, there are two dropdown menus: the first is set to 'POST' and the second is set to 'JSON'. At the bottom right of the dialog, there is a 'Create' button. A small tip at the bottom left says '* Tip: paste Curl command into URL afterwards to import it'.

Testando...

- Vamos escrever o JSON:

```
POST https://api.myproduct.com/v1/users

JSON Auth Query Header 1 Docs

1 {
2   "title": "Novo Post",
3   "author": "Ronan"
4 }
```

Testando...

- Vamos pegar a URL que vamos enviar. Vejam que quando vamos fazer um POST, passamos a URL mãe aqui, sem parâmetros.

Métodos HTTP

ENDPOINT	MÉTODO	AÇÃO
/produtos	GET	Retorna uma lista de Produtos
/produtos/{id}	GET	Retorna somente o Produto com id = {id}
/produtos	POST	Insere um novo Produto
/produtos/{id}	PUT	Altera os dados do Produto com id = {id}
/produtos/{id}	PATCH	Altera alguns dados do Produto com id = {id}
/produtos/{id}	DELETE	Deleta o Produto com id = {id}

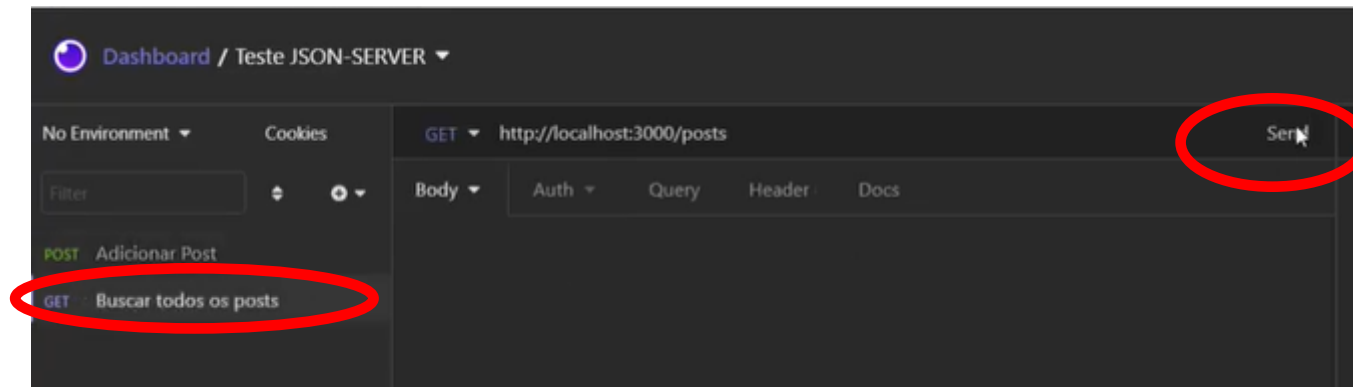
Testando...

- Então está pronta a URL, vamos clicar em SEND e foi criado:

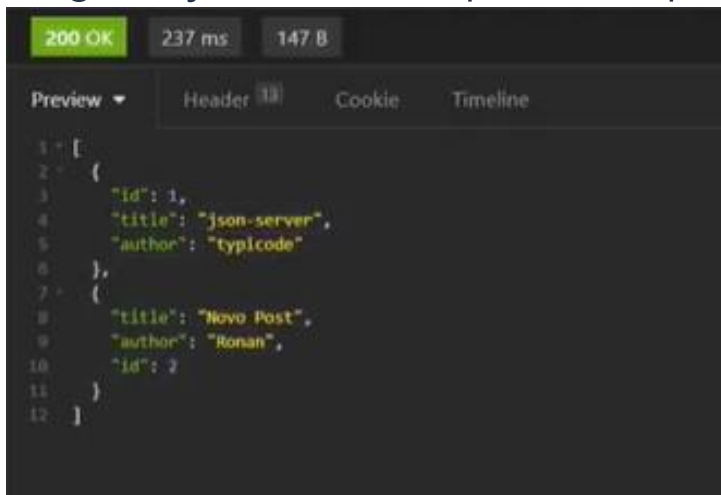


Testando...

Vamos testar e ver se deu certo. Clicar em Buscar todos os posts, SEND:

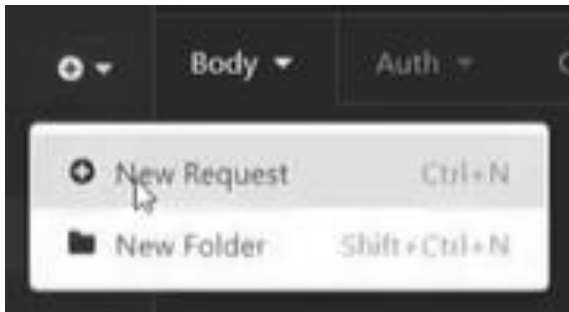


Agora já temos 2 posts aqui:



Testando...

- Vamos testar agora o GET por um campo só, criando uma nova requisição:

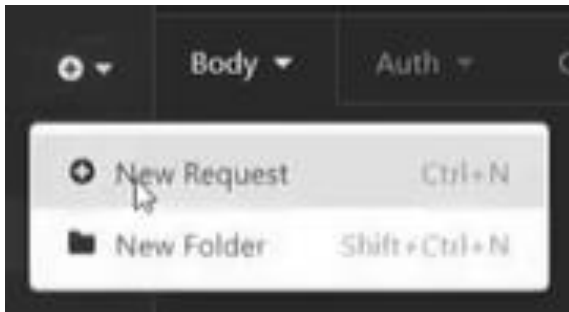


- Nome: Buscar por ID:

A screenshot of a 'New Request' dialog box. It has a title bar with 'New Request' and a close button. Below the title bar, there is a label 'Name (defaults to your request URL if left empty)' followed by a text input field containing 'Buscar por ID'. To the right of the input field is a dropdown menu showing 'GET'. At the bottom left, there is a tip: '* Tip: paste Curl command into URL afterwards to import it'. At the bottom right, there is a 'Create' button.

Testando...

- Vamos testar agora o GET por um campo só, criando uma nova requisição:

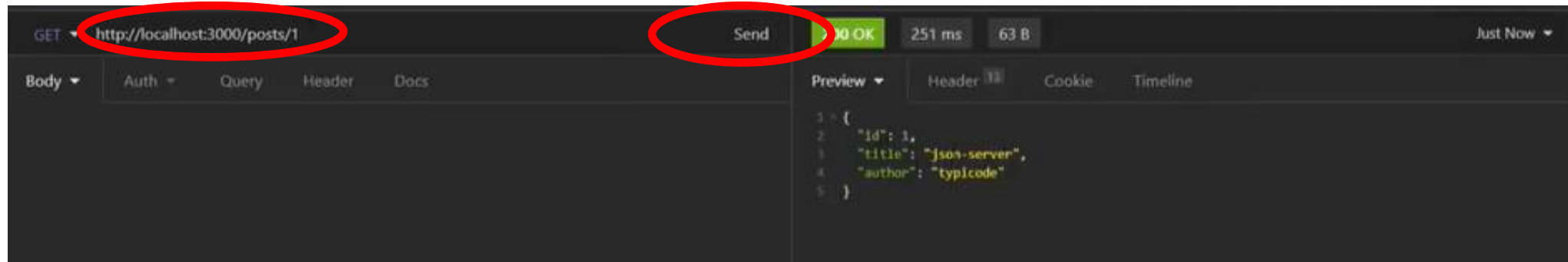


- Nome: Buscar por ID, tipo GET e clicar no botão Create:



Testando...

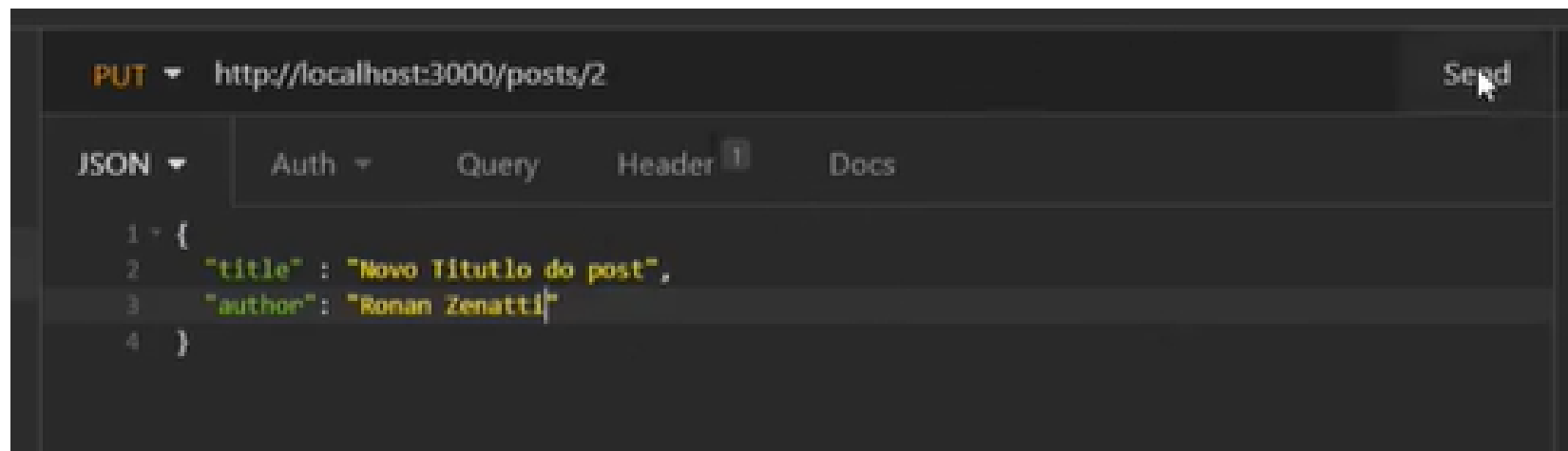
- Vamos copiar a URL, mas precisamos informar um ID. Após o posts, digitar /1, por exemplo. Se digitar o /2 irá trazer o registro 2:



Testando...

- Criar uma nova requisição Atualizar Post
- Método: PUT
- No corpo: JSON
- Altere o titulo e o author para o seu nome

Testando...

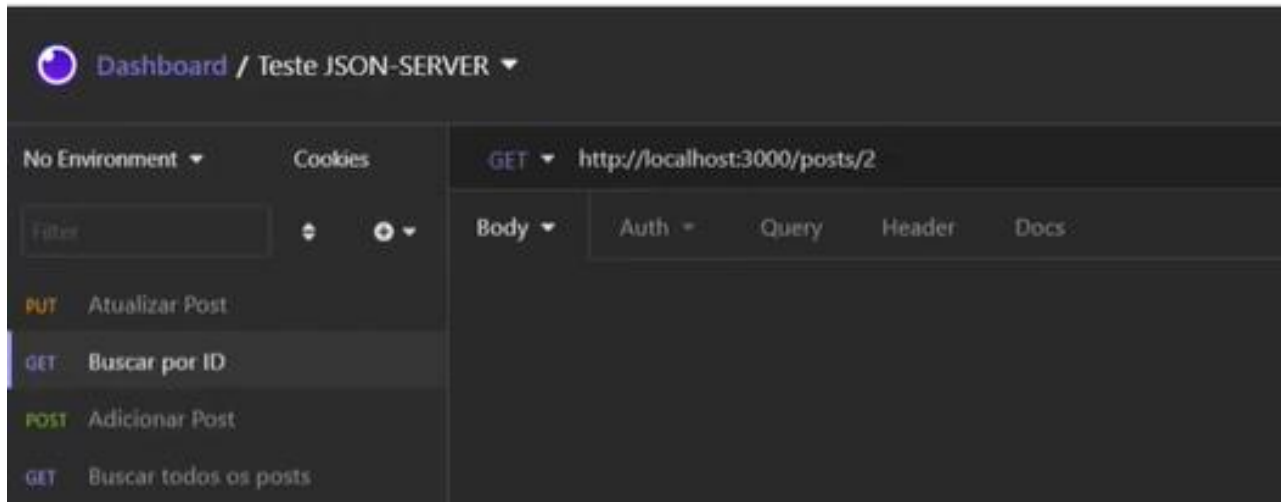


A screenshot of a REST client interface. The top bar shows a PUT request to `http://localhost:3000/posts/2` with a 'Send' button on the right. Below this, a tab bar contains 'JSON', 'Auth', 'Query', 'Header', and 'Docs', with 'JSON' selected. The main area displays a JSON body with the following content:

```
1 {  
2   "title" : "Novo Titulo do post",  
3   "author": "Ronan Zenatti"  
4 }
```

Testando...

Vamos testar Buscando por ID para conferir se realmente foi alterado



Vamos testar Buscar todos os posts

Testando...

- O próximo da lista é o Patch, utilizamos quando queremos alterar somente uma informação
- Criar uma nova requisição Atualizar Apenas 1 informação
- Método: PATCH
- No corpo: JSON
- Altere somente o Author e teste

Testando...

The screenshot displays a REST client interface with a dark theme. The top bar shows a **PATCH** request to `http://localhost:3000/posts/2` with a **Send** button. Below this, the **JSON** tab is active, showing the request body:

```
1 {  
2   "author": "Ronan Adriel Zenatti"  
3 }
```

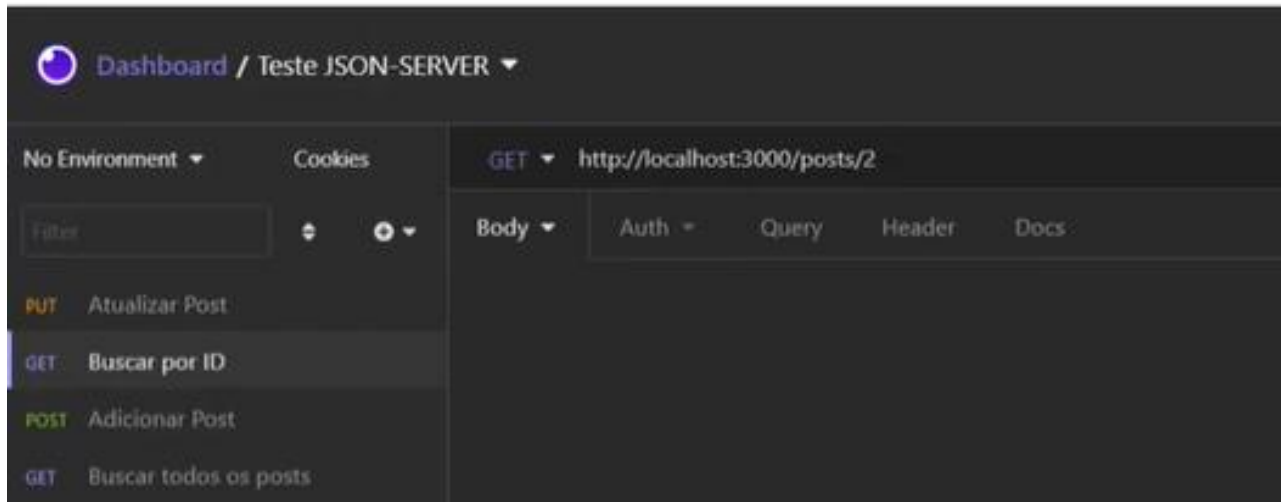
. The right panel shows the response status as **200 OK** with a response time of **266 ms** and a size of **84 B**. The **Preview** tab is active, displaying the response body:

```
1 {  
2   "title": "Novo Titulo do post",  
3   "author": "Ronan Adriel Zenatti",  
4   "id": 2  
5 }
```

. The interface also includes tabs for **Auth**, **Query**, **Header**, **Docs**, **Header**, **Cookie**, and **Timeline**.

Testando...

Vamos testar Buscando por ID para conferir se realmente foi alterado



Vamos testar Buscar todos os posts

Testando...

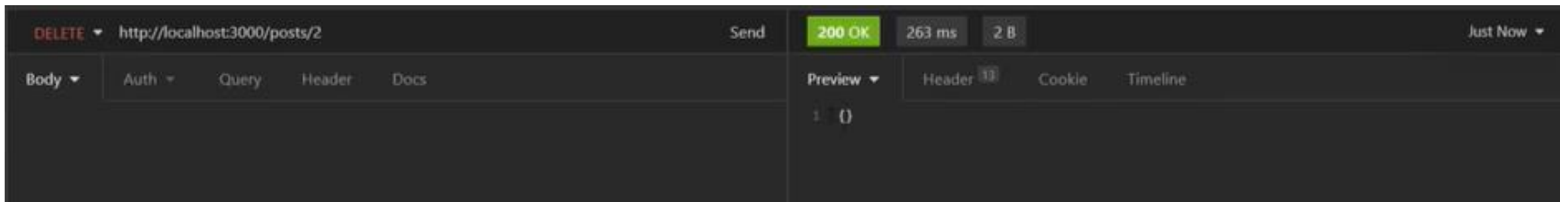
Como utilizamos o DELETE.. Também é muito fácil...

Métodos HTTP

ENDPOINT	MÉTODO	AÇÃO
/produtos	GET	Retorna uma lista de Produtos
/produtos/{id}	GET	Retorna somente o Produto com id = {id}
/produtos	POST	Insere um novo Produto
/produtos/{id}	PUT	Altera os dados do Produto com id = {id}
/produtos/{id}	PATCH	Altera alguns dados do Produto com id = {id}
/produtos/{id}	DELETE	Deleta o Produto com id = {id}

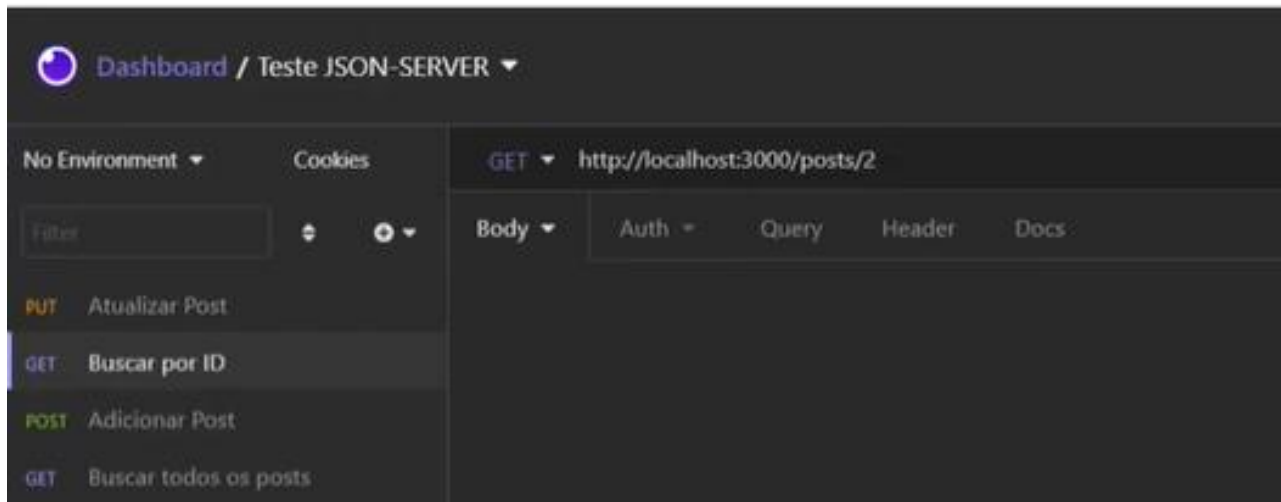
Testando...

- Criar uma nova requisição Deletar Post
- Método: DELETE
- No corpo: JSON
- Digite após o post o numero 2, por exemplo e será deletado. Retornou vazio e significa que conseguiu ser deletado.



Testando...

Vamos testar Buscando por todos os posts



Concluindo...

- Dentro do Dashboard podemos criar várias coleções,
 - ▷ Uma coleção para cada endpoint, para cada API que vamos testar.
- Dentro de cada coleção criamos os métodos e testamos os endpoints.

