

# Requisições HTTP

## Exercício

Documentação da API: <https://deividfortuna.github.io/fipe/>

## Básico sobre Requisições em JS

Antes de mais nada precisamos falar sobre AJAX. Em poucas palavras, é o uso do objeto `XMLHttpRequest` para se comunicar com os scripts do lado do **servidor**. Ele pode enviar e receber informações em uma variedade de formatos, incluindo JSON, XML, HTML, e até mesmo arquivos de texto. Sua natureza "assíncrona" é a melhor parte, significa que ele pode fazer tudo isso sem a necessidade de atualizar a página permitindo a você atualizar partes de uma página com base em eventos do usuário.

Uma das maneira de utilizar esse objeto é via a API Fetch que fornece uma interface para acessar e manipular partes do HTTP, tais como os **chamadas e respostas**. Pelo método global `fetch()` usamos de maneira fácil e lógica a busca de recursos.

Este tipo de funcionalidade era obtida anteriormente utilizando `XMLHttpRequest` mas com `fetch` temos uma alternativa melhor que pode ser facilmente utilizada por outras tecnologias como *Service Workers*. Também temos outras bibliotecas como *Axios* para fazer requisições HTTP.

```
fetch('http://localhost:8000/api')
  .then(response => response.json()) // callback que retorna a resposta da API
  .then(data => console.log(data)) // callback para o uso da resposta da API
  .catch(error => { // callback que executa caso a API der algum erro
    console.error(error);
  });
}
```

## Headers de Requisições e Respostas

Tanto rodando por *Fetch* quanto por bibliotecas de clientes HTTP as requisições vão usar *Promises* e logo por ser assíncrona devemos tratar as suas exceções e seus retornos e as requisições HTTP também funcionam da mesma forma, podendo retornar um erro por causa de algum problema com o envio da requisição quanto

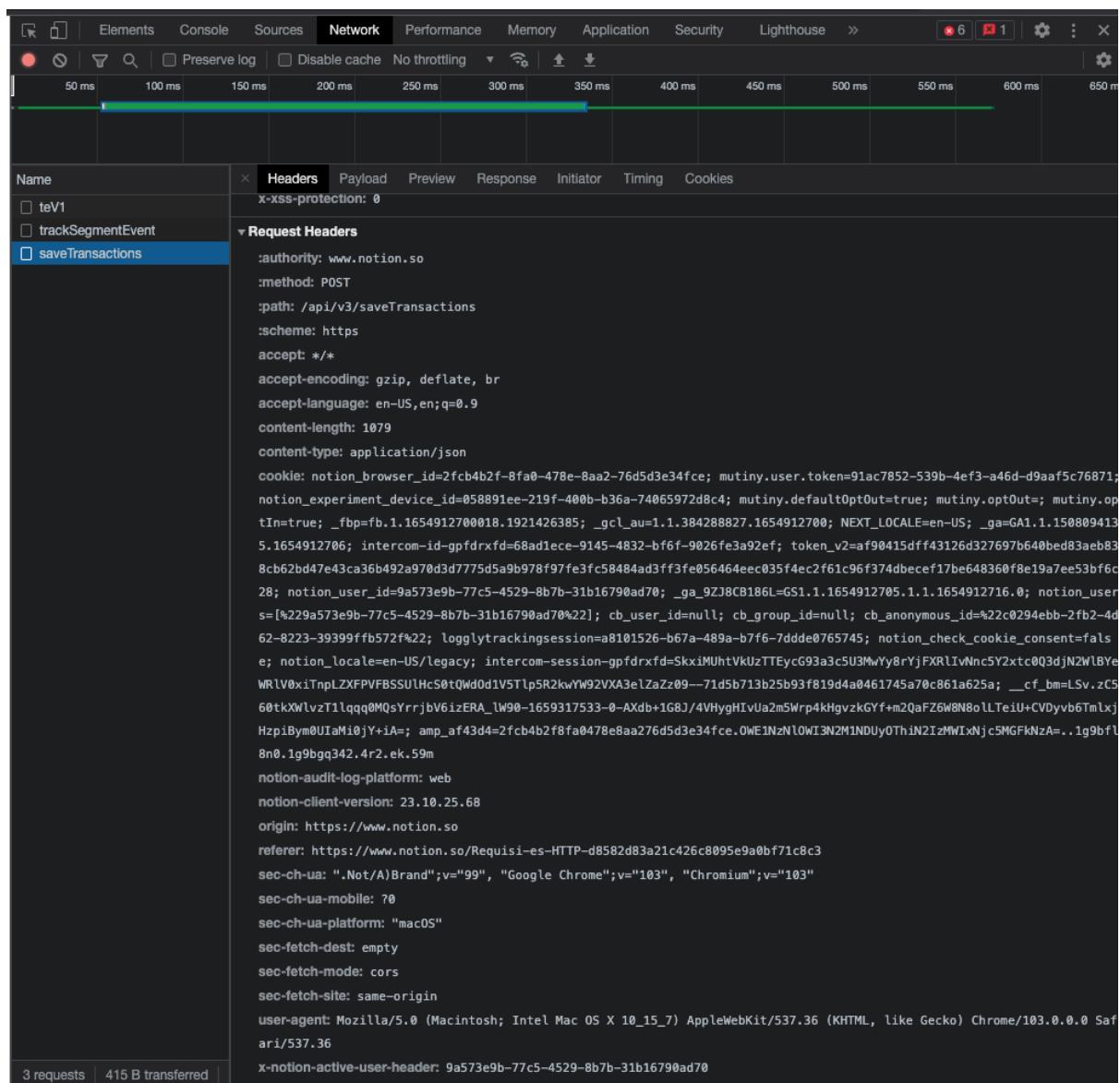
por algum problema com o código do Back-end. Seja ele qual for, temos códigos de status em que cada um possui um significado.

<http://httpstat.us/> → Aqui você pode ver mais sobre os *Status Codes* existentes.

Além disso temos um lugar específico para receber algumas instruções dessas requisições HTTP, são os Headers e temos eles tanto para o **envio (Request Headers)** quanto para a **resposta (Response Headers)**.

## Request Headers

São os *headers* enviados para o back-end que na maioria das vezes são enviados automaticamente pelo navegador que da instruções para o servidor de como ele deve lidar com essa requisição. Esses são os *headers* enviados para uma API do [Notion.so](#):



The screenshot shows the Network tab in the Chrome DevTools developer console. A single request is listed under the 'Name' column, which is 'saveTransactions'. The 'Headers' section is expanded, showing the following details:

- :authority: www.notion.so
- :method: POST
- :path: /api/v3/saveTransactions
- :scheme: https
- accept: \*/\*
- accept-encoding: gzip, deflate, br
- accept-language: en-US,en;q=0.9
- content-length: 1079
- content-type: application/json
- cookie: notion\_browser\_id=2fcb4b2f-8fa0-478e-8aa2-76d5d3e34fce; mutiny.user.token=91ac7852-539b-4ef3-a46d-d9aaef5c76871; notion\_experiment\_device\_id=058891ee-219f-400b-b36a-74065972d8c4; mutiny.defaultOptOut=true; mutiny.optOut=true; \_fbp=fb.1.1654912700018.1921426385; \_gcl\_au=1.1.384288827.1654912700; NEXT\_LOCALE=en-US; \_ga=GA1.1.1508094135.1.1654912706; intercom-id=gpfdrxid=68ad1ece-9145-4832-bf6f-9026fe3a92ef; token\_v2=af90415dff43126d327697b640bed83aeb838cb62bd47e43ca36b492a970d3d7775d5a9b978197fe3fc58484ad3ff3fe056464eec035f4ec2ff61c96f7374dbecef17be648360f8e19a7ee53bf6c28; notion\_user\_id=9a573e9b-77c5-4529-8b7b-31b16790ad70%22; cb\_user\_id=null; cb\_group\_id=null; cb\_anonymous\_id=%22c0294ebb-2fb2-4d62-8223-39399ffbb572f%22; logglytrackingsession=a8101526-b67a-489a-b7f6-7ddde0765745; notion\_check\_cookie\_consent=false; notion\_locale=en-US/legacy; intercom-session-gpfdrxid=SkxiMUhtVkJUzTTEycG93a3c5U3MwYy8rYjFXRlIvNnc5Y2xtc0Q3djN2WlBYeWRlV0xiTnplZXFpVFBSSUlHcs0tQWd0d1V5Tlp5R2kwYW92VXA3elzaZz09--71d5b713b25b93f819d4a0461745a70c861a625a; \_\_cf\_bm=LSv.zC560tkXWlvzT1lqqq0MQsYrrjbV6izERA\_lW90-1659317533-0-AXdb+1G8J/4VHygHIVua2m5Wrp4kHgvzkgYf+n2QaFZ6W8N8oLLTeiU+CVdyvb6TmlxjHzpiBym0UIaMi0jY+ia=; amp\_af43d4=2fcb4b2f8fa0478e8aa276d5d3e34fce.0WE1NzN10WI3N2M1NDUy0Th1N2IzMWIXNjc5MGFKNzA..1g9bfL8n0.1g9bgq342.4r2.eK.59m
- notion-audit-log-platform: web
- notion-client-version: 23.10.25.68
- origin: https://www.notion.so
- referer: https://www.notion.so/Requisições-HTTP-d8582d83a21c426c8095e9a0bf71c8c3
- sec-ch-ua: ".Not/A)Brand";v="99", "Google Chrome";v="103", "Chromium";v="103"
- sec-ch-ua-mobile: ?0
- sec-ch-ua-platform: "macOS"
- sec-fetch-dest: empty
- sec-fetch-mode: cors
- sec-fetch-site: same-origin
- user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36
- x-notion-active-user-header: 9a573e9b-77c5-4529-8b7b-31b16790ad70

At the bottom of the screenshot, it says '3 requests | 415 B transferred'.

- method → Método HTTP utilizado na requisição
- content-type → Tipo do conteúdo que foi enviado, existem muitas e basicamente vamos sempre aprendendo conforme usamos diferentes APIs mas `application/json`, `application/x-www-form-urlencoded` e `multipart/form-data` são as mais utilizadas.
- cookie → Envio de todos os cookies do navegador. Por padrão uma requisição não envia eles é necessária a configuração de segurança desse envio pois nos cookies geralmente ficam os dados pertinentes à acessos e *tokens*.
- origin e referer → URL da página origem da chamada HTTP
- user-agent → Dispositivo origem da chamada HTTP

## Response Headers

Esses servem para ~~quase nada~~ que dados adicionais sejam enviados para o cliente (navegador). Em algumas requisições podemos obter a quantidade total de uma listagem nos *headers* para que a resposta não fique poluída. Eles também servem para salvar novos cookies e configurações no navegador.

```

Request URL: https://www.notion.so/api/v3/saveTransactions
Request Method: POST
Status Code: 200
Remote Address: 172.64.154.162:443
Referrer Policy: strict-origin-when-cross-origin

cf-cache-status: DYNAMIC
cf-ray: 733ae5ab8aa3a5f7-GRU
content-length: 2
content-security-policy: default-src 'none'
content-type: application/json; charset=utf-8
date: Mon, 01 Aug 2022 01:45:56 GMT
etag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
referrer-policy: strict-origin-when-cross-origin
server: cloudflare
strict-transport-security: max-age=5184000; includeSubDomains
vary: Accept-Encoding
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-download-options: noopener
x-frame-options: SAMEORIGIN
x-notion-user-id: 9a573e9b-77c5-4529-8b7b-31b16790ad70
x-permitted-cross-domain-policies: none
x-xss-protection: 0

```

## Usando Requisições no React

Dando alguns passos para trás, lembremos do **Ciclo de Vida do Componente** para pensar: Onde é o melhor lugar para chamar uma API apenas uma vez? No método

`componentDidMount` ! Por isso em quase 99% das vezes vemos chamadas GET (em que obtemos dados de APIs) sendo chamadas nessa hora e depois salvando os dados em um estado. Podemos usar a força das *Promises* e renderizar diversos estados de carregamento ou de erros enquanto esperamos ou recebemos algo do servidor.

Mas claro que não é excessão buscar algo **fora do `componentDidMount`** existem fluxos e dados que só queremos buscar a partir de um *input* do usuário. Uma busca do Google por exemplo, não pode ser feita sem que o usuário digite algo no campo de texto. Então em casos como esse precisamos de uma **função** e um **disparador**.

De resto, nada muda, *Fetch*, *Axios* ou até mesmo o *XMLHttpRequest*, pode utilizar o que preferir mas se atente nos ciclos de vida, eles serão na maioria das vezes seus maiores aliados e mal-feitores ao utilizar *Promises* em React.

© Curso Online de React do Zero ao Pro  
Desenvolvido por Gustavo Vasconcellos e EBAC Online