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

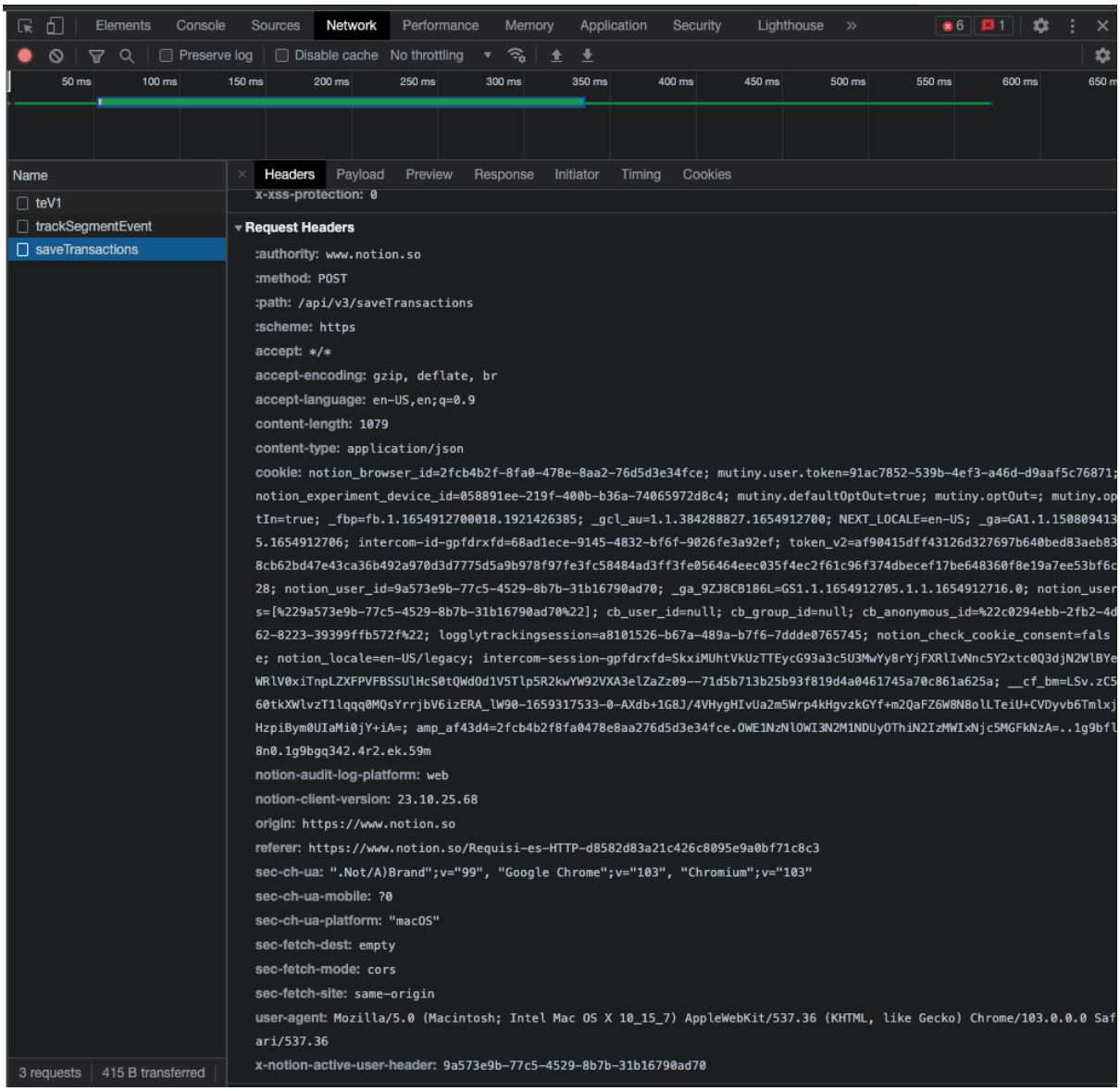
por algum problema com o código do Back-end. Seja ele qual for, temos códigos de status em que cada um possui um significado.

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

Além disso temos um lugar especifico 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

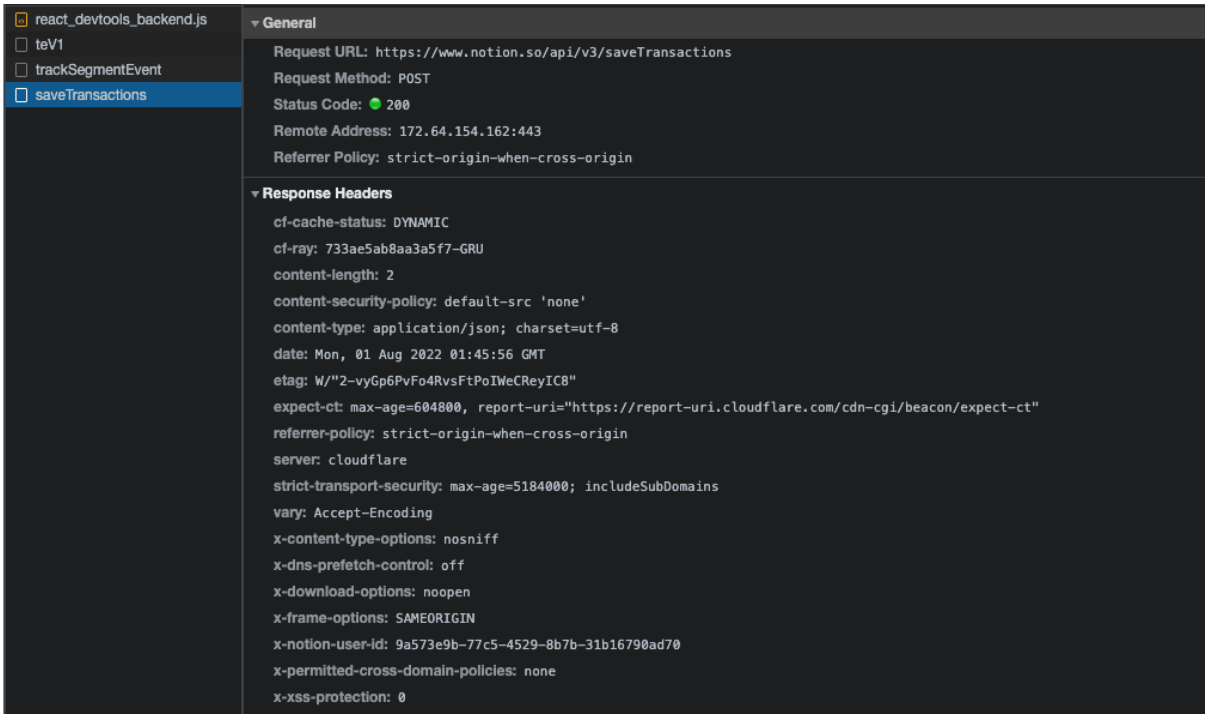
São os *headers* enviados para o back-end que na maioria das vezes são enviados automaticamente pelo navegador que da instruções para o servidor de como ele deve lidar com essa requisição. Esses são os *headers* enviados para uma API do Notion.so:



- 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.



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.