03

Parâmetros na requisição com métodos GET e POST

Transcrição

Vamos ver um outro exemplo de uma URL e acessar http://www.youtube.com (http://www.youtube.com). Como tem muito conteúdo no YouTube, vamos pesquisar, por exemplo por *Ayrton Senna*. Repare que, ao pesquisar no YouTube, a URL mudou um pouco. O recurso acessado pela busca se chama `/results` (os resultados da pesquisa) mas agora temos um **parâmetro da requisição**, indicado pela `?`: `https://www.youtube.com/results?search_query=Ayrton+Senna`

O parâmetro se chama `search_query` com o valor `Ayrton+Senna`. Esses parâmetros da URL normalmente são chamados de *Query Params*. O HTTP permite enviar mais de um parâmetro, basta concatenar o próximo parâmetro através do caractere `&`.

Por exemplo, a busca avançada do Google usa vários parâmetros para refinar a pesquisa como o idioma, o país ou data. Veja como o Google concatena os *Query Params*: `https://www.google.com.br/?gws_rd=ssl#lr=lang_pt&tbs=lr:lang_1pt&q=neymar`

Parâmetros com GET

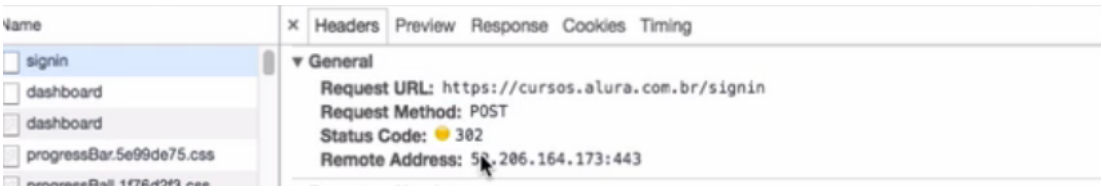
Reparem que no console sempre aparece o tipo (ou método) da requisição, sendo `GET`. Uma característica da requisição `GET` é enviar os parâmetros pela URL! Isso é útil quando queremos deixar os parâmetros visíveis. Assim podemos facilmente guardar a URL com os parâmetros para repetir a requisição algum momento depois. Mas será que isso também é uma boa opção na hora de enviar credenciais como `login` e `senha`? Queremos que apareça a senha de um usuário na URL?

Imagina que você efetue o login no seu banco e na URL apareça: `https://www.bb.com.br/login?login=nico&password=supersecreto`

Não parece ser uma boa solução, não é mesmo?

O método HTTP POST

Vamos efetuar um login na Alura para ver como esse sistema envia dados do usuário para o servidor.



Repare que a URL para enviar o *login* e *senha* se chama `https://www.alura.com.br/signin`. Repare também que o método HTTP utilizado mudou. Estamos usando o **HTTP POST**! Usando o `POST`, o navegador envia os dados do formulário no corpo da requisição e não na URL! Se fosse um `GET`, todos os dados seriam enviados através da URL. Como a Alura não deseja que os dados do *login* e *senha* apareçam na URL do navegador, foi utilizado um HTTP `POST`. Faz sentido?

GET para receber, POST para criar?

Um outro exemplo de um método `POST` na Alura é quando criamos uma pergunta no forum. Nesse momento o usuário submete um formulário com dados para criar um novo recurso na Alura (a pergunta do aluno se torna um recurso). O método `POST` foi inicialmente pensado para criar algo novo no servidor como acabamos de fazer. Ou seja, ao enviar uma requisição `POST` para o servidor, a nossa intenção é criar algo novo. No entanto, nem sempre isso é realmente utilizado dessa maneira. Por exemplo, acabamos de usar um `POST` para verificar o login, ou seja, não alteramos ou adicionamos nada na Alura. Nossa motivação para o `POST` era esconder os parâmetros apenas.

Como o servidor realmente reage quando recebe uma requisição `POST` depende da implementação, depende da lógica atrás. Os métodos como `GET` e `POST` definem uma intenção mas o que realmente será executado depende do servidor.

No dia a dia, vocês vão ver códigos usando `GET` para fazer pesquisas mas também para alterar ou remover algo no servidor. A mesma coisa podemos dizer sobre `POST` . Vocês vão usar o `POST` para inserir e alterar dados, mas também para pesquisar. As aplicações podem adaptar o significado dos métodos HTTP quando for necessário.