**SENAI – I2HNO**

Pedro Muniz

Beatriz Pupo

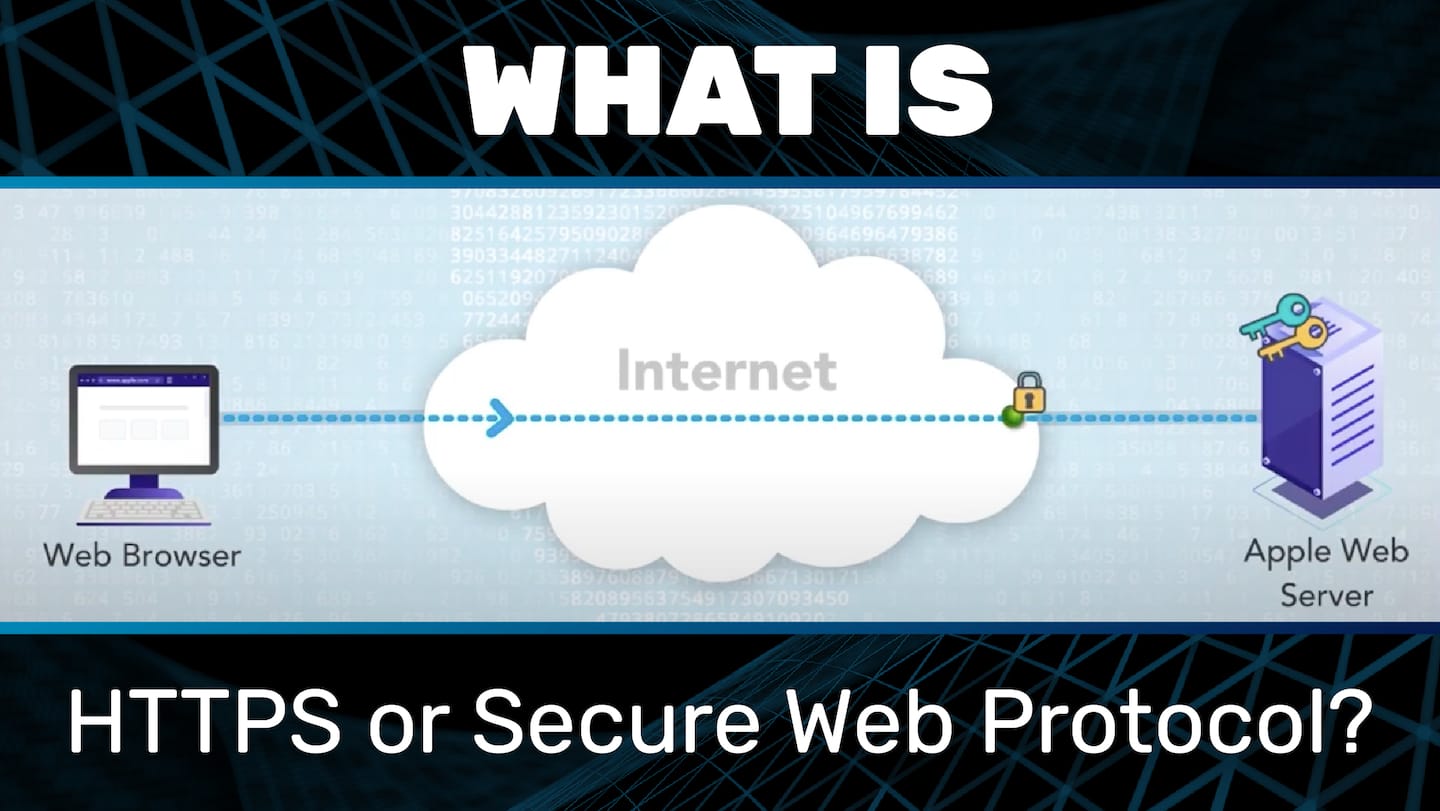
Samuel Baek

Rafaela Barrionuevo

**Relatório sobre protocolo HTTP**

Nosso objetivo com a apresentação desse relatório é identificar e dominar os conhecimentos sobre diferentes protocolos HTTP.

**Introdução:** O HTTP (Hyertext Transfer Protocol) foi desenvolvido para a comunicação entre cliente (navegador web) e servidor (servidores web). Segue um modelo cliente-servidor, onde o cliente executa uma requisição e espera até receber a resposta do servidor. Geralmente utilizado na camada de transporte TCP/IP



<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.akamai.com%2Fpt%2Fglossary%2Fwhat-is-http&psig=AOvVaw3A9VlefzF9FLg82ufMF0IB&ust=1739033795365000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCPjQib6FsosDFQAAAAAdAAAAABAE>

**Métodos HTTP:**

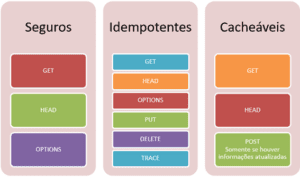
* Get: O método GET é utilizado para obter recursos do servidor. Por exemplo, quando você acessa a rota “/planets” em uma API de planetas, está fazendo uma requisição GET para obter a lista completa de planetas. Como o GET só busca dados, ele pode ser executado diretamente no navegador. Basta digitar a URL da rota desejada que os dados serão retornados. Portanto, o método GET é indicado quando desejamos buscar ou consultar dados de um servidor web.
* Post: O método POST é utilizado quando desejamos enviar dados para serem processados pelo servidor, como em um cadastro ou envio de formulário. Como o POST altera dados, ele não pode ser executado pelo navegador. É necessário utilizar outras ferramentas como Postman ou Insomnia para conseguir enviar requisições POST corretamente. Nessas ferramentas, além de definir a rota, também conseguimos preencher o corpo (*body*) com os dados a serem enviados em formato JSON. Portanto, utiliza-se POST quando o objetivo é enviar ou criar dados no servidor.

* Put: O método PUT é muito parecido com o POST, porém com uma diferença importante. Enquanto o POST cria um novo recurso, o PUT atualiza um recurso existente. Além disso, o PUT sempre atualiza o recurso por completo, substituindo todos os dados anteriores pelos dados enviados na requisição atual.
* Por exemplo, se quisermos atualizar completamente os dados de um usuário, podemos fazer uma requisição PUT enviando no corpo todos os novos dados, como nome, email, telefone etc. Portanto, o método PUT deve ser utilizado quando o objetivo é atualizar por completo os dados de um determinado recurso.

* Delete: Conforme o próprio nome já diz, o método DELETE serve para excluir recursos do servidor.
* Por exemplo, se quisermos excluir permanentemente um usuário, podemos enviar uma requisição DELETE informando o ID desse usuário na URL. Dessa forma, o método DELETE deve ser utilizado quando o objetivo é remover ou excluir de forma permanente um determinado recurso.

* Patch: O método PATCH é similar ao PUT, pois também serve para atualizar um recurso. Porém a diferença é que o PATCH faz uma atualização parcial, modificando apenas os dados enviados, sem afetar o restante.
* Por exemplo, se quisermos atualizar apenas o nome de um usuário, podemos enviar uma requisição PATCH passando apenas o novo nome, sem a necessidade de enviar também o email, telefones etc. Dessa forma, o método PATCH é indicado quando o objetivo é fazer uma atualização parcial de dados de um recurso.

* Options: O método HTTP OPTIONS é utilizado para que um cliente possa descobrir quais as opções de requisição permitidas para um determinado recurso em um servidor. O cliente pode especificar uma URL específica no método OPTIONS ou um asterisco indicando que se refere ao servidor como um todo para sanar suas dúvidas em relação as opções de requisição permitidas.



<https://blog-static.infra.grancursosonline.com.br/wp-content/uploads/2023/02/22170807/Imagem4.artigo.22.02-300x177.png>

* Postman:

[Postman](https://www.linkedin.com/redir/redirect?url=https%3A%2F%2Fwww%2Epostman%2Ecom%2F&urlhash=7ee9&trk=article-ssr-frontend-pulse_little-text-block) é uma plataforma de desenvolvimento e teste de APIs que oferece uma variedade de recursos. Permite que os desenvolvedores criem e compartilhem facilmente solicitações e coleções de APIs, automatizem testes, simulem APIs e monitorem o desempenho.

O Postman também fornece ferramentas de colaboração e documentação, permitindo que as equipes trabalhem juntas de forma mais eficiente e comuniquem o comportamento da API de maneira eficaz.

Com o Postman o usuário pode interagir com a API utilizando o sistema CRUD (Create, Read, Update e Delete) ou os Métodos HTTP para gerenciar a API a fim de desenvolver novas aplicações e variáveis, otimizar, deletar ou até mesmo solicitar informações do cliente para o servidor.



<https://www.benner.com.br/tecnologia/wp-content/uploads/2023/06/Postman.jpg>

  Fontes:

<https://www.escoladnc.com.br/blog/o-que-e-protocolo-http-e-metodos-de-requisicao-em-apis/>

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods/OPTIONS>

<https://pt.linkedin.com/pulse/o-que-%C3%A9-postman-vivien-ross-mt9hc>

**Tipos de passagem de parâmetros:** Tipos de Passagem de Parâmetros: Recebe os dados da requisição como parâmetro na URL.  Caso de uso: Filtros para fazer consultas na aplicação.  Pode conter um ou mais parâmetros:

Exemplos:

<http://minhaapi.com/banks?name=nubank>  
No exemplo acima acesso o recurso (ou rota) *banks*, filtrando por *name*. Para inserir parâmetros coloco “?” após a rota e adiciono a propriedade e valor: name=nubank.

<http://minhaapi.com/movies?name=transformers&actors=megan,peter>   
No exemplo acima acesso API de filmes, pesquisando por *name* e *actors*. Sempre que for passar mais de um parâmetro podemos colocar & para separar os parâmetros.

* Body Params: Recebe os dados da requisição no corpo da requisição, em um objeto em JSON. Sempre utilizando no método POST da requisição.

*{    
"name": "Thiago", "age": 18, "email": "[thiago@mail.com](mailto:thiago@mail.com" \t "_blank)"   
}*

No *Controller* você pega a requisição para salvar os dados no banco de dados.

*server.post("/users", (req, res) {    
const { name, age, email } = req.body;    
await connection("users").insert({ name, age, email });    
return res.json({ id }); }*

* Query Parameters: Utilizados na URL para filtrar ou modificar o comportamento da requisição. Por exemplo: */usuarios?idade=25.*
* Body Parameters: Enviados no corpo da requisição, geralmente em POST ou PUT, para fornecer dados detalhados.

* Query Parameters: Uso Seguro: Evite expor dados sensíveis nos query parameters, pois eles são visíveis na URL.

*Interface BuscaUsuario {   
    idade: number;   
}   
   
app.get('/usuarios', (req, res) => {   
let filtro: BuscaUsuario = req.query;   
// Implementação da lógica de busca com o filtro   
});*

Utilize bibliotecas para validar e sanitizar os query parameters, reduzindo o risco de ataques como SQL Injection.

* Body Parameters: Estruturação e Validação: Estruture os body parameters usando interfaces e valide-os antes do processamento.

*Interface DadosUsuario { nome: string;   
    email: string;   
}   
app.post('/usuarios', (req, res) => {   
  let dadosUsuario: DadosUsuario = req.body;   
  // Validação e lógica de criação do usuário*

*});*

* Header Parameters: Esses parâmetros são apresentados no cabeçalho da solicitação e geralmente estão relacionados à autorização, como tokens, controle de sessão e dados de cookies. Esse tipo de parâmetro aparece em qualquer método HTTP (GET, POST, PUT, DELETE).
* Path Parameters: Os parâmetros de caminho são partes variáveis de um caminho de URL. Eles geralmente são usados para apontar para um recurso específico dentro de uma coleção, como um usuário identificado por ID. Um URL pode ter vários parâmetros de caminho, cada um denotado por chaves { }.

//http://myapi/users/{id}   
//http://myapi/cars/{carId}/drivers/{driverId}



<https://static.semrush.com/blog/uploads/media/5b/8c/5b8c60306da55359372c923ee2456566/PT-Graphic-3.png>

**Cabeçalhos HTTP:**

* Host: O cabeçalho de solicitação de Host especifica o nome de domínio do servidor (para hospedagem virtual), e (opcionalmente) o número da porta TCP no qual o servidor está escutando. Se nenhuma porta for fornecida a porta padrão para o serviço solicitado (por exemplo, “80”; para um HTTP URL) está implícita. Um campo de cabeçalho de Host deve ser enviado em todas as mensagens desolicitação HTTP / 1. 1. Um código de status 400 (Bad Request) será enviado para qualquer mensagem de solicitação HTTP/1.1 que não contenha um campo de cabeçalho do host ou contenha mais de um.
* Exemplo: Host: developer.mozilla.org

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Host>

* Accept: O cabeçalho de requisição HTTP Accept anuncia quais tipos de conteúdo, expressos como MIME types, o cliente é capaz de entender. Usando negociação de conteúdo, o servidor então seleciona uma das propostas, a usa e então informa o cliente da sua escolha com o cabeçalho de resposta Content-Type. Navegadores colocam valores adequados para este cabeçalho dependendo do contexto onde a requisição é feita: quando buscando um “CSS stylesheet”, um valor diferente é colocado para a requisição, o mesmo acontece com imagens, vídeos ou scripts.
  + Exemplo:

*Accept: text/html*

*Accept: image/\**

// Padrão geral

*Accept: \*/\**

// Padrão para requisições de navegação

Accept: text/html, application/xhtml+xml, application/xml; Q=0.9,

\*/\*; Q=0.8

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Accept>

* User-Agent: O cabeçalho de requisição User-Agent é uma cadeia de caracteres característica que permite servidores e pares de rede identificar a aplicação, sistema operacional, fornecedor, e/ou versão do agente de usuário requisitante.
  + Exemplo: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

Mozilla/5.0 (Macintosh; Intel Mac OS X x.y; rv:42.0) Gecko/20100101 Firefox/42.0

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/UserAgent>

* Request Method: O cabeçalho de requisição Access-Control-Request- Method é usado pelos navegadores quando executando uma  preflight request , para deixar o servidor sabendo quais métodos HTTP vão ser usados quando a requisição verdadeira for feita. Este cabeçalho é necessário como toda requisição pré-vôo é um OPTIONS e não usa o mesmo método na requisição verdadeira.
* Exemplo: Access-Control-Request-Method: POST
* Acept-Language: O cabeçalho Accept-Language permite especificar em quais idiomas você deseja receber o recurso solicitado. Se necessário, você pode incluir o código do país ou o tipo de alfabeto. Por exemplo, se você definir Accept-Language como “en-US”, significa que deseja que o recurso esteja em inglês, na variante falada nos Estados Unidos da América, mesmo se você estiver em outro continente. Você também pode usar o tipo de alfabeto para definir o cabeçalho, como a versão do idioma sérvio no alfabeto latino, especificando “sr-Latn”. Isso garante que você recupere os dados localizados apropriados. Quando há vários idiomas, o cabeçalho Accept-Language se torna uma lista de idiomas separados por vírgula com valores de qualidade que ajudam a definir a ordem de prioridade
  + Exemplo: Um exemplo disso é

“*en – GB; Q=1.0, en-US; Q=0.9, fr; Q=0.8*”,

em que valores maiores de q indicam maior prioridade. Além disso, *Q* varia de 0 a 1.

<https://developer.mozilla.org/ptBR/docs/Web/HTTP/Headers/AcceptLanguage>

* Content-Type: O cabeçalho Content-Type é utilizado para indicar o tipo de arquivo do recurso.Em respostas, o Content-Type diz para o cliente qual é o tipo de conteúdo que a resposta, de fato, tem. Alguns browsers vão realizar o chamado &quot;MIME Sniffing&quot; em alguns casos e não vão, necessariamente, seguir o valor declarado por este cabeçalho. Para prevenir este comportamento é possível definir o cabeçalho X-Content-Type- Options para o valor nosniff.Em requisições, como POST ou PUT, o client diz ao servidor qual o tipo de dado que está, de fato, sendo enviado.
  + Exemplo:

*<form action=”/” method=”post” enctype=”multipart/form-data”>*

*<input type=”text” name=”description” value=”some text” />*

*<input type=”file” name=”myFile” />*

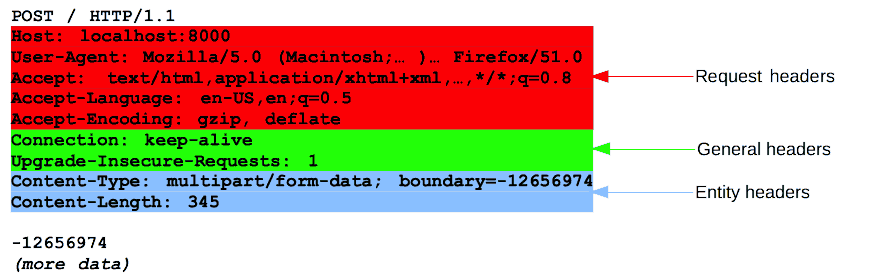
*<button type=”submit”>Submit</button>*

*</form>*

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Content-Type>

* Authorization: O cabeçalho de requisição HTTP Authorization contém as credenciais para autenticar o agente de usuário com o servidor, geralmente o servidor responderá com um status 401 Unauthorized se não for possível fazer a autenticação, e com o cabeçalho WWW-Authenticate.
  + Exemplo: Basic YWxhZGRpbjpvcGVuc2VzYW1l

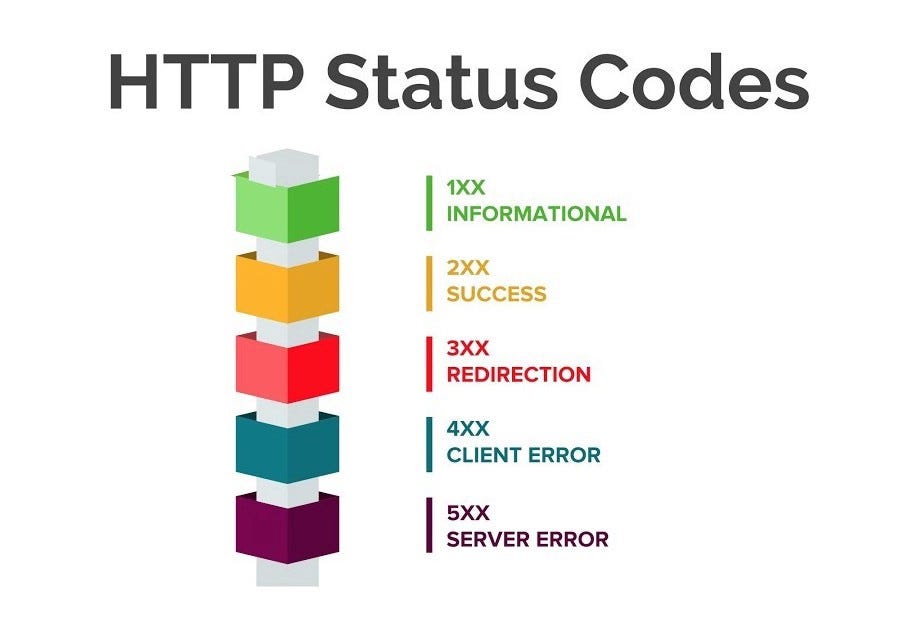
<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Authorization>



<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Messages/http_request_headers2.png>

**Códigos de Status HTTP:** Como o servidor sempre retornará à requisição do cliente ele também retornará erros ou requisições não atendidas. Esses retornos são conhecidos como “Códigos de Status”. Basicamente, existem cinco tipos de códigos de status, aqueles começados com o número 1 (1XX), os começados com o número 2 (2XX), os começados com o número 3 (3XX), os começados em número 4 (4XX) e os começados em número 5 (5XX).

* 1XX – Os códigos de status começados em 1 significam que o requerimento de API do cliente foi recebido e está sendo processado, informando que ele aguarde uma resposta.
  + 100 – Indica que o servidor recebeu a solicitação do cliente e que o cliente deve continuar com a solicitação.
  + 101 – Sinaliza que o servidor está mudando para um protocolo diferente, conforme o cliente pediu.
* 2XX – Já os códigos começados em 2 significam que o requerimento do cliente foi bem-sucedido. O servidor informa esse código juntamente com os recursos solicitados.
  + 200 – Indica que a solicitação foi bem-sucedido.
  + 202 – Indica que o servidor analisou a solicitação, ela foi aceita, mas ainda não iniciou ou concluiu o processamento.
* 3XX – A classe de códigos que se inicia com o número 3 indica que o cliente precisa realizar ações adicionais para que seu requerimento seja atendido. Essa classe geralmente aparece em redirecionamento de *URL’s.*
  + 301 – Aponta que o redirecionamento não pôde ser concluído por conta da URL requerida não ser daquele navegador.
  + 302 – A URL foi encontrada e o endereço atual está redirecionando o cliente para outro endereço.
* 4XX – Os códigos iniciados com o número 4 sinalizam uma mensagem de erro por parte do cliente através da solicitação.
  + 400 – Esse código indica uma solicitação inválida que está impossibilitado de ser processado por conta de um requerimento errado por parte do cliente. Os erros mais comuns que geram esse código são erros de sintaxe ou cookies inválidos.
  + 404 – Informa que o recurso solicitado não pôde ser encontrado no servidor, mas não informa se o recurso está ausente permanentemente ou apenas temporário, algumas das causas mais comuns são URL digitadas incorretamente.
* 5XX – Os códigos iniciados com o número 5 indicam uma solicitação com falha devido a um erro de servidor ao processar um requerimento.
  + 500 – O servidor não pode atender à solicitação devido a uma condição inesperada. Geralmente um erro do navegador.
  + 503 – O serviço não pode lidar com a solicitação porque ficou sem recurso ou está em manutenção. Geralmente enviam esse código devido a um erro temporário.



[https://miro.medium.com/v2/resize:fit:920/1\*yrMWEpUC-hXED7oGD0j2og.jpeg](https://miro.medium.com/v2/resize:fit:920/1*yrMWEpUC-hXED7oGD0j2og.jpeg)