
FIAP

PROTOCOLO HTTP

INTRODUÇÃO

Quando falamos de internet atualmente, muito do tráfego dos dados é via protocolo HTTP. HTTP é a sigla para **Hypertext Transfer Protocol**, que traduzindo fica Protocolo de Transferência de Hipertexto.

Ele é uma comunicação baseado no protocolo TCP/IP, usado para entregar dados (HTML, arquivos de imagens, buscas, etc.) na rede mundial, ou formalmente conhecida como WWW (World Wide Web).

A porta padrão de comunicação utilizada para utilizar o HTTP é a porta 80, mas sua aplicação pode rodar em outra porta também.

Sua especificação determina como as requisições feitas pelos clientes são estruturadas e enviadas ao servidor e como o servidor responde a essas requisições.

Cliente sempre são aqueles que estão fazendo a requisição enquanto servidor é aquele que Resposta da requisição feita pelo cliente.

Recursos Básicos

- **Conexão assíncrona**: Uma vez que o cliente fez a requisição e o servidor devolveu a resposta, o cliente desconecta do servidor. Ou seja, o cliente e o servidor só sabem da existência um do outro durante a requisição. Caso seja necessário fazer novas requisições, novas conexões serão abertas
- **Dados independentes**: Ou seja, via protocolo HTTP é possível enviar qualquer tipo de dados, desde que cliente e servidor saibam como lidar com eles.
- **Sem estado**: Como já citado anteriormente, a conexão é feita de forma assíncrona e cliente e servidor só sabem da existência um do outro durante a requisição, uma vez que ela é finalizada nenhum dos dois lados retem informações sobre ambos

Representação Gráfica

Métodos

Os métodos HTTP, também conhecidos como verbos, servem para o cliente informar qual a sua intenção na requisição. São eles:

- GET -> Usado para obter informações e somente para isso. Exemplo: Baixar fotos, exibir uma página
- POST -> Usado para enviar dados ao servidor. Exemplo: Um formulário de cadastro de usuário
- PUT -> Informa que vai haver uma substituição de dados. Exemplo: Atualização de dados do usuário
- DELETE -> Utilizado para informar ao servidor que algum recurso deve ser deletado.
- PATCH -> Atualização parcial de dados
- HEAD -> Funciona como o GET, mas ele retorna apenas o status e o cabeçalho da resposta.
- OPTIONS -> Solicita ao servidor quais métodos e opções estão disponíveis.

Códigos de Status

Os códigos de status do HTTP são números de 3 dígitos que o servidor retorna para o cliente informado qual o status da requisição feita. O primeiro dígito informa qual a classe do status e os outros dois dígitos são informativos. Existem 5 classes de status:

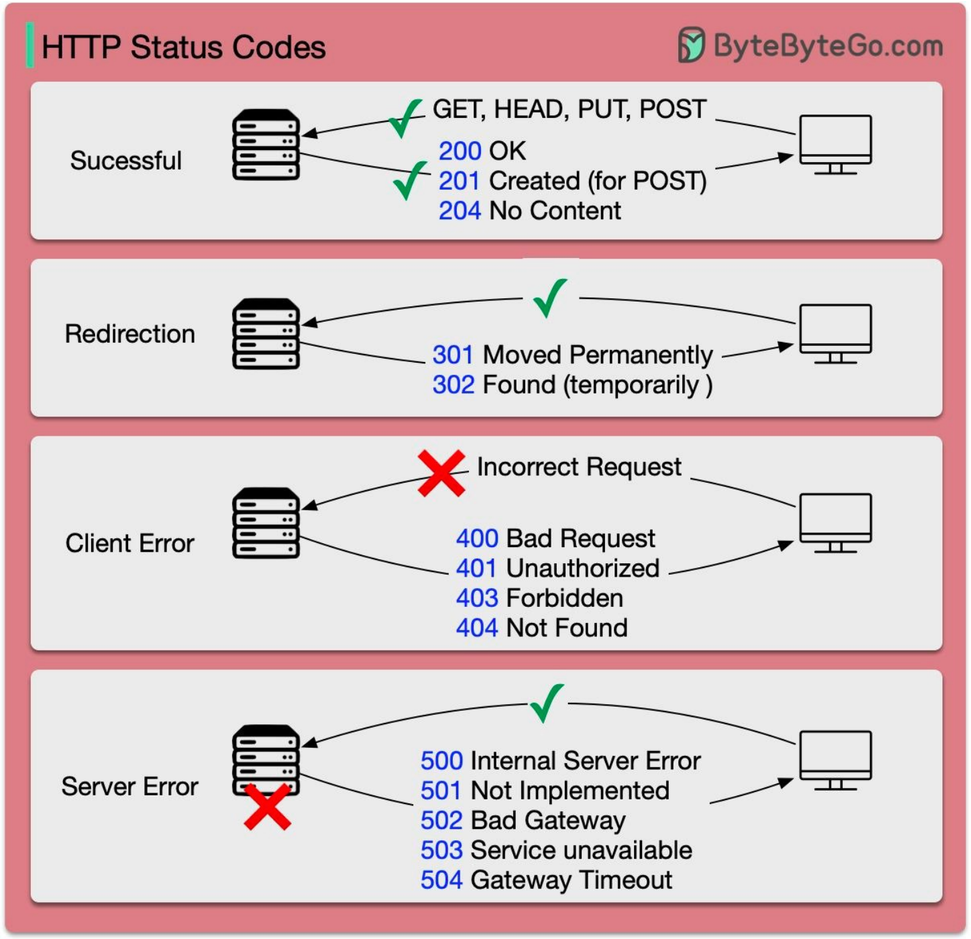
- 1xx -> A requisição foi recebida e continua sendo processada
- 2xx -> A requisição foi processada com sucesso
- 3xx -> A requisição foi recebida e algumas etapas adicionais foram executadas
- 4xx-> A requisição contém erros
- 5xx -> O servidor teve algum problema no processamento da requisição

Não se preocupe em aprender e decorar todos os códigos de status, pois são muitos e alguns deles são bem raros de se utilizar.

Para quem gosta de gatos e quer ver uma lista completa, acesse o site:
https://http.cat

Códigos de Status

A seguir, uma imagem que ilustra como funciona os códigos de status



Requisição

A requisição (request) é a solicitação enviada pelo cliente ao servidor. Tem uma estrutura bem definida para informar ao servidor o que está sendo solicitado. Toda requisição tem a seguinte estrutura:

- Linha inicial: informa qual o método será realizado, em qual endereço do servidor e qual a versão do HTTP será utilizada
- Cabeçalho: informa ao servidor dados adicionais sobre a requisição
- Corpo: o conteúdo da requisição

Exemplo de uma requisição:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

Resposta

A resposta (response) como o nome já diz, é o retorno do servidor para a requisição feita pelo cliente. Tem a mesma estrutura da requisição, mas da seguinte forma:

- **Linha inicial**: informa qual a versão do HTTP foi utilizada, o código de status e a mensagem de status
- **Cabeçalho**: informa ao cliente dados adicionais Resposta
- **Corpo**: o conteúdo da resposta

Exemplo de uma resposta:

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=iso-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>404 Not Found</title>
</head>
<body>
   <h1>Not Found</h1>
   <p>The requested URL /t.html was not found on this server.</p>
</body>
</html>
```

Requisição GET

Exemplo de como uma requisição `GET` é feita:

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36
Accept: application/json
Accept-Language: en-US,en;q=0.5
Authorization: Token abc123
Cache-Control: no-cache
Connection: keep-alive
```

Requisição GET

Resposta de uma requisição `GET`

```
HTTP/1.1 200 OK
Date: Sun, 28 Mar 2023 10:15:00 GMT
Content-Type: application/json
Server: Apache/2.4.39 (Unix) OpenSSL/1.1.1c PHP/7.3.6
Content-Length: 1024
{
    "name": "John Doe",
    "email": "johndoe@example.com",
    "age": 30,
    "address": {
        "street": "123 Main St",
        "city": "Anytown",
        "state": "CA",
        "zip": "12345"
    }
}
```

Requisição POST

Exemplo de como uma requisição `POST` é feita:

```
POST /save HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

field1=value1&field2=value2
```

Requisição POST

Resposta de uma requisição `POST`:

```
HTTP/1.1 201 Created
Date: Sun, 28 May 2022 10:19:10 GMT
Content-Type: application/json
Server: Apache/2.4.39 (Unix) OpenSSL/1.1.1c PHP/7.3.6
```

Requisição PUT

Exemplo de como uma requisição `PUT` é feita:

```
PUT /api/users/1 HTTP/1.1
Host: example.com
Content-type: aplication/json
Content-length: 16

{
    "name": "Leonardo"
}
```

Requisição PUT

Resposta de uma requisição `PUT`

```
HTTP/1.1 200 OK
Content-Location: /api/users/1
```

Requisição DELETE

Exemplo de uma requisição `DELETE`

```
DELETE /file.html HTTP/1.1
Host: example.com
```

Requisição DELETE

Resposta de uma requisição `DELETE`

```
HTTP/1.1 200 OK
Date: Wed, 21 Oct 2015 07:28:00 GMT

<html>
  <body>
    <h1>File deleted.</h1>
  </body>
</html>
```

Requisição PATCH

Exemplo de uma requisição `PATCH`

```
PATCH /file.txt HTTP/1.1
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[description of changes]
```

Requisição PATCH

Resposta de uma requisição `PATCH`

```
HTTP/1.1 204 No Content
Content-Location: /file.txt
ETag: "e0023aa4f"
```

Requisição HEAD

Exemplo de uma requisição `HEAD`

```
HEAD /echo/head/json HTTP/1.1
Host: reqbin.com
Accept: application/json
```

Requisição HEAD

Resposta de uma requisição `HEAD`

```
HTTP/1.1 200 OK
Content-Length: 19
Content-Type: application/json
```

Requisição OPTIONS

Exemplo de uma requisição `OPTIONS`

```
OPTIONS /api/v1/requests HTTP/1.1
Host: api.reqbin.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: content-type
Origin: https://reqbin.com
```

Requisição OPTIONS

Reposta de uma requisição `OPTIONS`

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: https://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
Vary: Accept-Encoding, Origin
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
```


FIAP