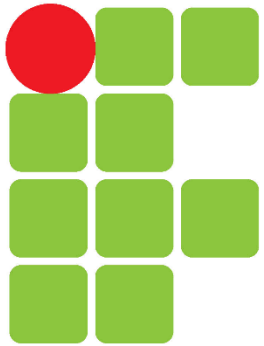
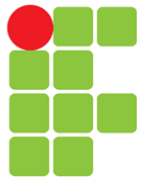


INSTITUTO FEDERAL
SÃO PAULO



Protocolo HTTP

Desenvolvimento de Sistemas Web (DSWI6)

Prof. Luiz Gustavo Diniz de Oliveira Véras

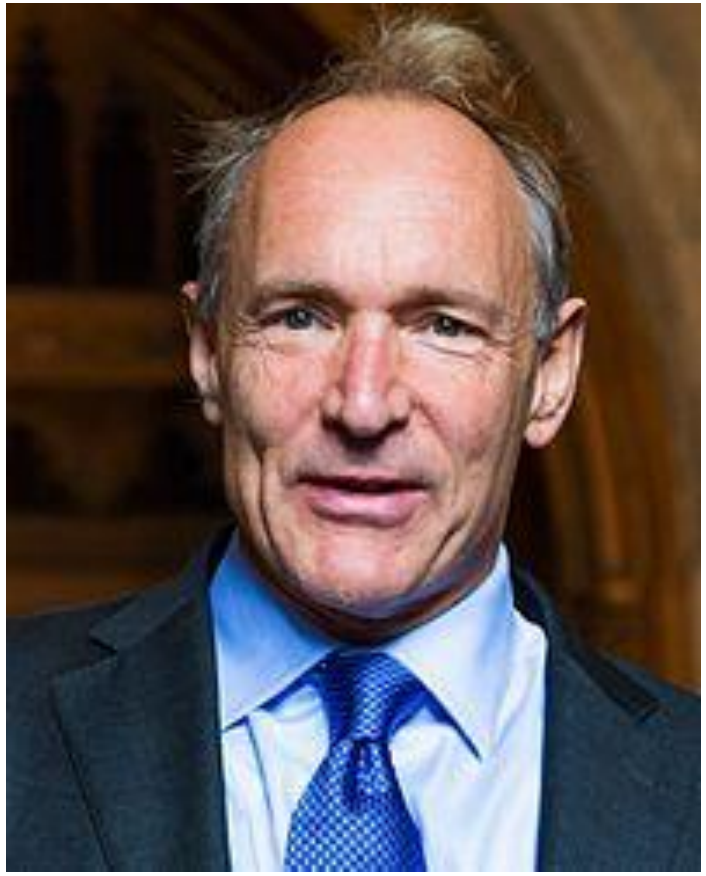
E-mail: gustavo_veras@ifsp.edu.br

Objetivos

- ✓Protocolo HTTP da camada de aplicação
  - ✓Recursos na web
  - ✓Mime Types
  - ✓Verbos
  - ✓Status
  - ✓Cabeçalhos (Headers)

Protocolo HTTP

Em 1989, enquanto trabalhava no **CERN**, Tim Berners-Lee escreveu uma proposta para construir um sistema de hipertexto pela internet. Inicialmente chamado de **Mesh**, mais tarde foi renomeado para **World Wide Web** durante sua implementação em 1990.



Tim Berners-Lee, o criador da WWW

Protocolo HTTP

Construído sobre os protocolos **TCP e IP** existentes, a WWW consistia em 4 blocos de construção:

- Um formato textual para representar documentos de hipertexto, o **HyperText Markup Language (HTML).**
- Um protocolo simples para trocar esses documentos, o H**yperText Transfer Protocol (HTTP)**.
- Um **cliente** para exibir (e editar) esses documentos;
- Um **servidor** para dar acesso ao documento.

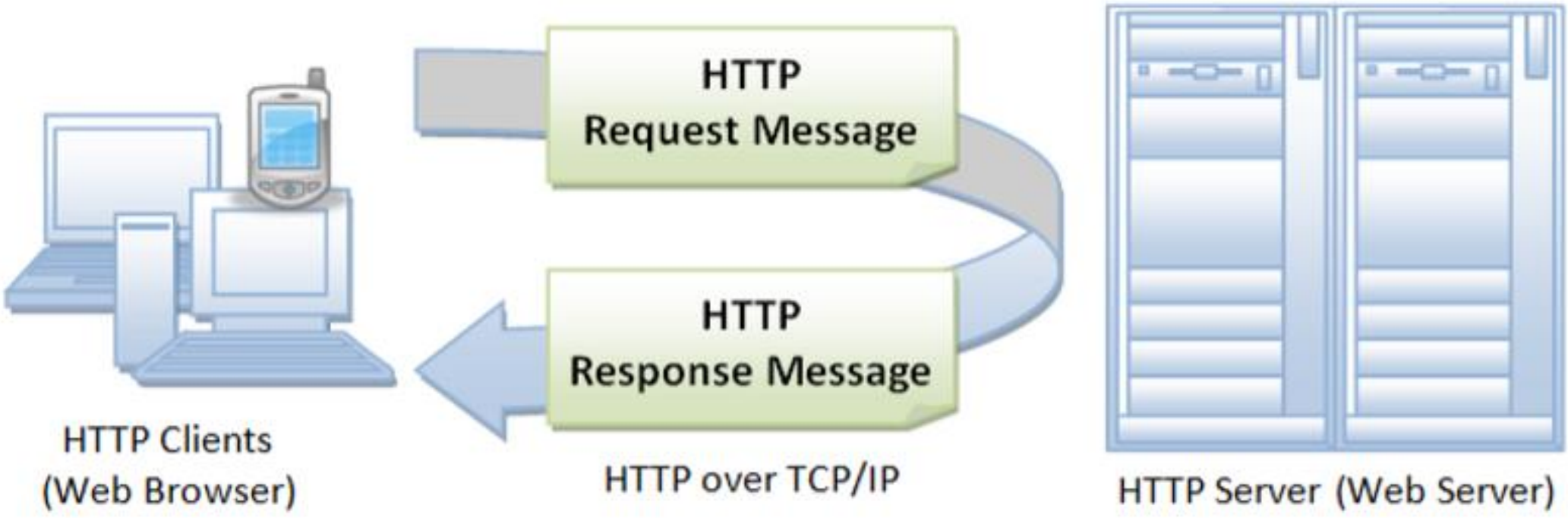
Protocolo HTTP

O HTTP (*Hypertext Transfer Protocol*) é o principal protocolo utilizado na WWW (*World Wide Web*).

É através dele que um Client (web browser, outro servidor, etc) faz comunicação com um Server (um computador com um programa que “escuta” solicitações de **recursos**) através de um modelo de **requisição/resposta (request/response)**.

Protocolo HTTP

Modelo Request/Response do HTTP



Protocolo HTTP

EVOLUÇÃO DO HTTP

HTTP/0.9 – The one-line protocol (1991):

O HTTP/0.9 era extremamente simples: os pedidos consistiam em uma única linha e começavam com o único método possível GET seguido do caminho para o recurso.

```
GET /mypage.html
```

A resposta também foi extremamente simples: consistia apenas no próprio arquivo.

```
<html>
A very simple HTML page
</html>
```

Protocolo HTTP

EVOLUÇÃO DO HTTP

HTTP/1.0 – Construindo extensibilidade (1996):

Principais mudanças:

- A versão do HTTP passou a ser enviada em cada solicitação (o HTTP/1.0 foi anexado à linha GET).
- Código de status foi adicionado no início de uma resposta. Isso permitiu que o próprio navegador reconhecesse o sucesso ou falha de uma solicitação e adaptasse seu comportamento de acordo (200 OK; 404 Page Not Found).
- O conceito de cabeçalhos (headers) HTTP foi introduzido para solicitações e respostas. Os metadados puderam ser transmitidos e o protocolo tornou-se extremamente flexível e extensível.
- Documentos que não sejam arquivos HTML simples podem ser transmitidos graças ao cabeçalho (headers) Content-Type.

Protocolo HTTP

Exemplo de uma **request** e **response** típicas do **HTTP/1.0** :

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
<HTML>
A page with an image
  <IMG SRC="/myimage.gif">
</HTML>
```

request

response

Protocolo HTTP

Exemplo de uma **request** e **respo**

```
GET /mypage.html HTT
User-Agent: NCSA_Mos

200 OK
Date: Tue, 15 Nov 19
Server: CERN/3.0 lib
Content-Type: text/h
<HTML>
A page with an image
  <IMG SRC="/myimage.gif">
</HTML>
```

```
GET /myimage.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

200 OK
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif
(image content)
```

response

Como há uma imagem no documento, uma segunda requisição é feita.

Protocolo HTTP

EVOLUÇÃO DO HTTP

HTTP/1.1 – O protocolo padronizado (1997):

A primeira versão padronizada do HTTP, HTTP/1.1, foi publicada no início de 1997, apenas alguns meses após o HTTP/1.0.

- Uma conexão pode ser reutilizada, o que economiza tempo. Ele não precisava mais ser aberto várias vezes para exibir os recursos incorporados no único documento original.
- Pipeline foi adicionado. Isso permitiu que uma segunda solicitação fosse enviada antes que a resposta à primeira fosse totalmente transmitida. Isso diminuiu a latência da comunicação.
- Respostas fragmentadas também foram suportadas.
- Mecanismos de controle de cache adicionais foram introduzidos.
- A negociação de conteúdo, incluindo idioma, codificação e tipo, foi introduzida. Um cliente e um servidor agora podem concordar sobre qual conteúdo trocar.
- Graças ao header **Host**, a capacidade de hospedar diferentes domínios do mesmo endereço IP permitiu a colocação do servidor.

Protocolo HTTP

EVOLUÇÃO DO HTTP

Exemplo de uma **request** e **response** típicas do **HTTP/1.1:**

request

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101
Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header
```

Protocolo HTTP

EVOLUÇÃO DO HTTP

Exemplo de uma **request** e **response** típicas do **HTTP/1.1:**

response

```
200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding

(content)
```

Protocolo HTTP

EVOLUÇÃO DO HTTP

HTTP/2.0 – Um protocolo para grande perfomance (2015):

Usado por 46,4% dos sites da web (https://w3techs.com/technologies/details/ce-http2):

- É um protocolo binário em vez de um protocolo de texto. Ele permite a implementação de técnicas de otimização aprimoradas.
- É um protocolo **multiplexado**. Requisições paralelas podem ser feitas na mesma conexão, removendo as restrições do protocolo HTTP/1.x.
- Ele comprime cabeçalhos (headers). Como muitas vezes são semelhantes entre um conjunto de solicitações, isso elimina a duplicação e a sobrecarga dos dados transmitidos.
- Ele permite que um servidor preencha dados em um cache de cliente por meio de um mecanismo chamado **push server**.

Protocolo HTTP

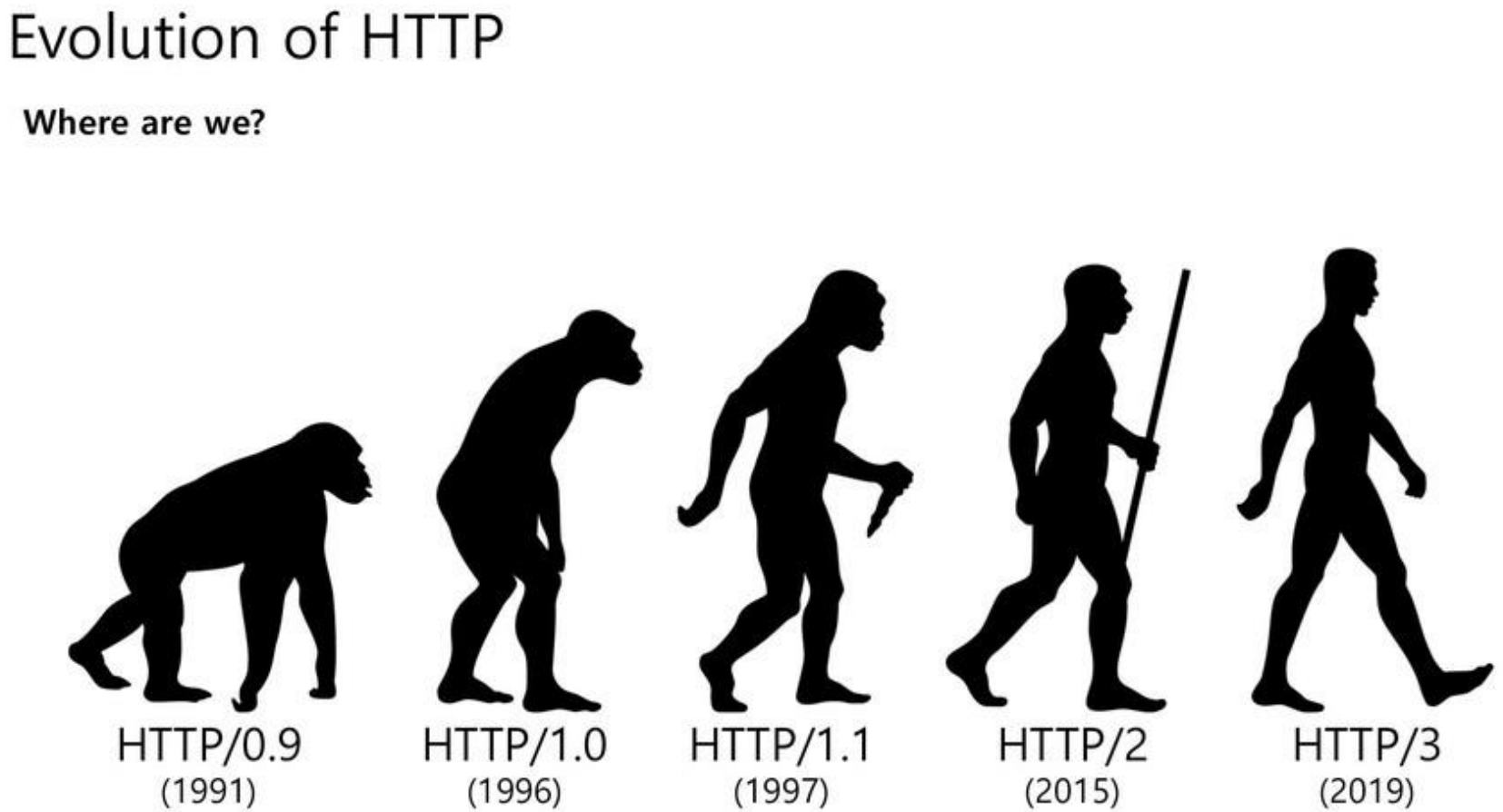
EVOLUÇÃO DO HTTP

HTTP/3.0 – HTTP sobre o Protocolo QUIC (2015):

A próxima versão principal do HTTP, HTTP/3, usará QUIC em vez de TCP para a parte da camada de transporte.

- QUIC é um novo protocolo desenvolvido pela Google e é construído sobre o Protocolo UDP ao invés do TPC.
- Mudar para UDP permitirá conexões mais rápidas e experiência de usuário mais rápida ao navegar on-line.

Protocolo HTTP



Recursos na Web

O alvo de uma requisição HTTP é chamada de "**resource**", ou **recurso** em português, com a natureza ainda não definida. Isso significa que um **recurso** pode ser:

- um documento
- uma foto
- qualquer outra coisa

Cada recurso é identificado por uma **Localização de recursos uniforme**, do inglês **Uniform Resource Locator (URL)** usada pelo HTTP para identificar recursos. **URLs são conhecidos como endereços da Web.**

Recursos na Web

Um recurso é qualquer **porção de informação** identificado por uma URL.

URL - Uniform Resource Location

protocol://hostname:port/path-and-name

Protocolo utilizado tanto por cliente quanto por servidor. Ex.: telnet, mailto, ftp, http, https, etc.

DNS do domínio ou endereço IP;

Porta que está escutando requisições dos clientes;

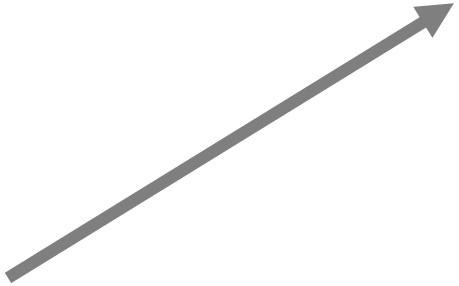
Nome ou caminho (paths) do arquivo no diretório do servidor.

Recursos na Web

Um recurso é qualquer **porção de informação** identificado por uma URL.

URL - Uniform Resource Location

protocol://hostname:port/path-and-name?key1=value1&key2=value2



Querys (ou parâmetros) são uma lista de pares chaves/valores separados com o símbolo &. O servidor web pode usar esses parâmetros para fazer coisas extras depois retornando o recurso para o usuário.

Protocolo HTTP

Uma requisição HTTP tem o seguinte formato:

E uma resposta HTTP:

Recursos na Web

Exemplo:

Em uma requisição HTTP a URL é convertida para um formato próprio.

```
http://www.meudominio.com:8080/docs/index.htm
l
```

HTTP Request

```
GET /docs/index.html HTTP/1.1
Host: www.meudominio.com:8080
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

HTTP Server

Recursos na Web

Exemplo:

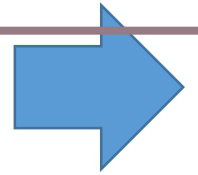
Em uma requisição HTTP a URL é convertida para um formato próprio.

`http://www.meudominio.com:8080/doc/index.html`

GET **/docs/index.html** HTTP/1.1

Host: **www.meudominio.com:8080**
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)

HTTP Request



HTTP Server

Recursos na Web

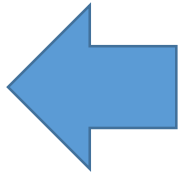
Exemplo:

O servidor interpreta a requisição e devolve o conteúdo solicitado com uma resposta.

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

HTTP Response



HTTP Server

Recursos na Web

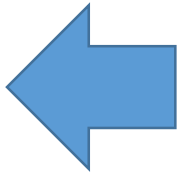
Exemplo:

O servidor interpreta a requisição e devolve o conteúdo solicitado.

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

HTTP Response



HTTP Server

Recursos na Web

Exemplo:

O servidor interpreta a requisição e devolve o conteúdo solicitado.

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

HTTP Response

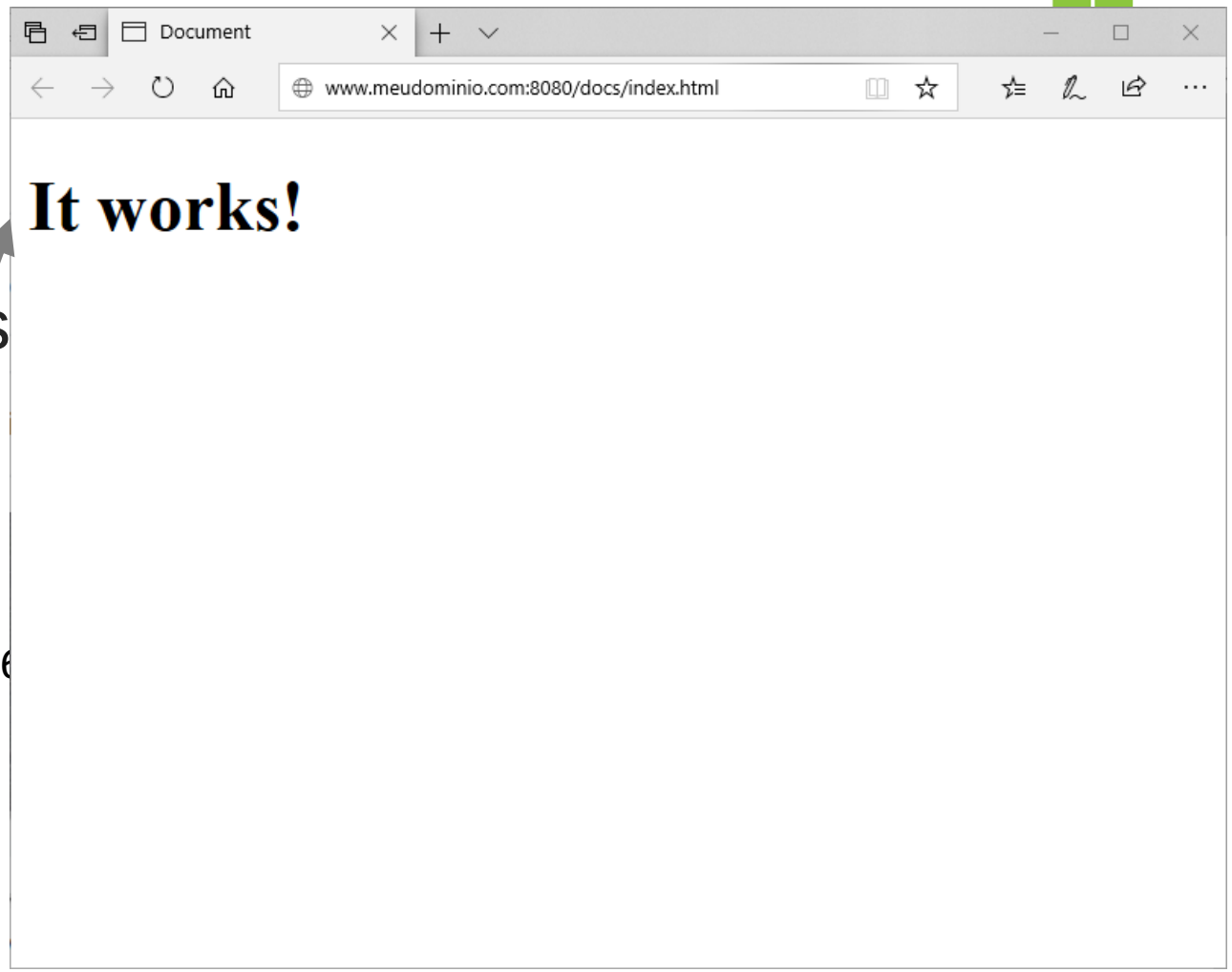
HTTP Server

Existe uma linha em branco que separa o header do body da resposta.

Recursos na Web

Exemplo:

O servidor interpreta a requis
solicitado.



HTTP Server

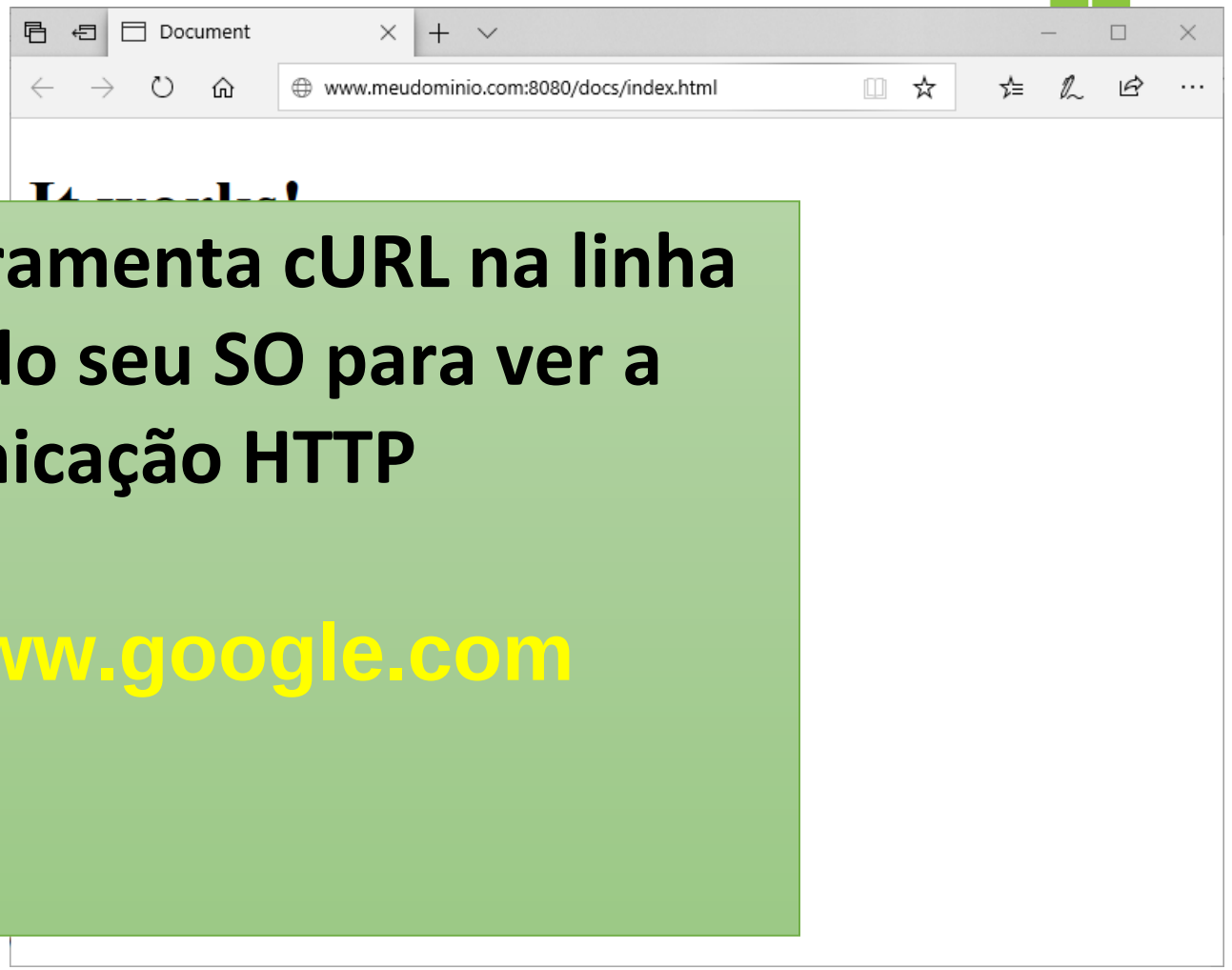
```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

Recursos na Web

Exemplo:

O servido
solicitado.



Teste com a ferramenta cURL na linha de comando do seu SO para ver a comunicação HTTP

curl –v www.google.com

HTTP Server

```
HTTP/1.1 200 OK
Date: Sun, 18 O
Server: Apache/
Last-Modified:
ETag: "10000000
Accept-Ranges:
Content-Length:
Connection: clo
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

MIME Types

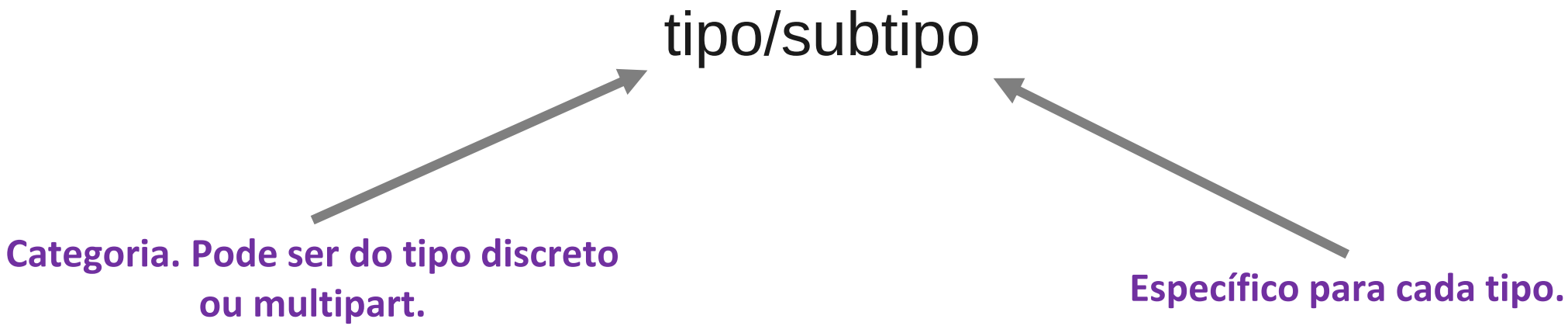
O MIME (Multipurpose Internet Mail Extensions) type, ou media types, é o mecanismo para dizer ao navegador indica a natureza e o formato de um documento, arquivo ou variedade de bytes transmitidos numa **response**.

Portanto, é importante que o servidor esteja configurado corretamente, de modo que o MIME-type correto seja transmitido com cada documento. Os navegadores costumam usar o MIME-type para determinar qual ação usar como padrão para fazer quando um recurso é obtido.

Existem muitos tipos de documentos, por isso há muitos MIME-types.

MIME Types

Estrutura básica de um MIME Type



MIME Types

Tipos discretos

text/plain
text/html
image/jpeg
image/png
audio/mpeg
audio/ogg
audio/*
video/mp4
application/octet-stream
application/json
…

Tipos multipart

multipart/form-data
multipart/byteranges

MIME Types

Tipos discretos

text/plain
text/html
image/jpeg
image/png
audio/mpeg
audio/ogg
audio/*
video/mp4
application/octet-stream
application/json
…

Tipo	Descrição	Exemplos de subtipos típicos
text	Qualquer documento que contenha texto e é teoricamente legivel para o ser humano.	text/plain, text/html, text/css, text/javascript
image	Qualquer tipo de imagens. Os vídeos não estão incluídos, embora imagens animadas (como gif animado) sim..	image/gif, image/png, image/jpeg, image/bmp, image/webp
audio	Qualquer tipo de arquivo de áudio	audio/midi, audio/mpeg, audio/webm, audio/ogg, audio/wav
video	Qualquer tipo de arquivo de vídeo	video/webm, video/ogg
application	Representa qualquer tipo de dados binários.	application/octet-stream, application/json, application/xhtml+xml, application/xml, application/pdf

MIME Types

Tipos discretos

Multipart types indicam uma categoria de documento que são quebrados em partes distintas, muitas vezes com diferentes tipos MIME. É uma forma de representar um documento composto.

multipart/form-data: que são usados em relação de formulários HTML e método POST.Enviado do navegador para o servidor.

multipart/byteranges: que são usados em conjunto com mensagem de status de conteúdo parcial enviados do servidor para o navegador. Geralmente faz o navegador propor uma janela de "Salvar como".

Tipos multipart

multipart/form-data
multipart/byteranges

MIME Types

Tipos discr

text/plain
text/html
image/jpeg
image/png
audio/mpeg
audio/ogg
audio/*
video/mp4
application/octet-stream
application/json
…

n-data
eranges

IANA é o registrador oficial de MIME Types. Acesse em

https://www.iana.org/assignments/media-types/media-types.xhtml

MIME Types

Quais os mais importantes para desenvolvedores web?

application/octet-stream

- Este é o valor padrão para um arquivo binário.

text/plain

- Este é o valor padrão para arquivos de texto.

text/css

- Quaisquer arquivos CSS que têm de ser interpretados como tal em uma página da Web

text/html

- Todo o conteúdo HTML deve ser exibido com este tipo.

multipart/form-data

- O tipo multipart/form-data pode ser usado ao enviar o conteúdo de um formul´zrio HTML preenchido do navegador para o servidor.

MIME Types

Porque é importante definir o MIME type correto?

Por razões de segurança, a maioria dos navegadores não permite definir uma ação padrão para MIME types não bem especificados, forçando o usuário a armazená-lo no disco para usá-lo.

Exemplos:

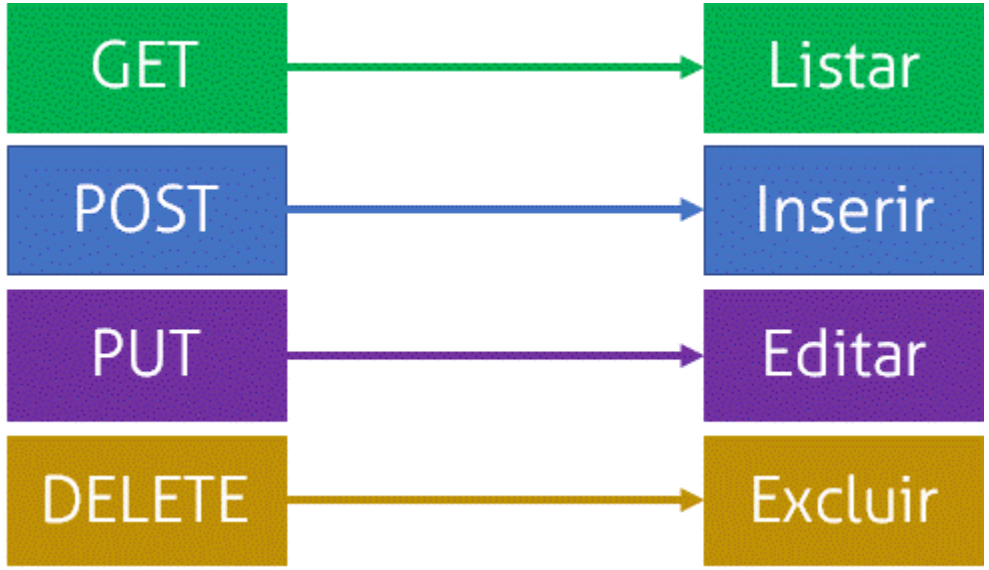
- **Arquivos de áudio e vídeo:** Somente recursos com o Tipo MIME correto serão reconhecidos e reproduzidos em elementos <video> ou <áudio>
- **Arquivos css:** Se o MIME type text/css não for especificado para arquivos .css, eles não serão reconhecidos como tal pela maioria dos navegadores e serão silenciosamente ignorados.

Protocolo HTTP

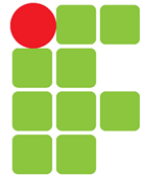
Verbos e Status HTTP

Os verbos permitem indicar a um servidor que tipo de operação um Navegador (ou outro software) gostaria de solicitar na comunicação pela internet.

É importante conhecê-los para indicar operações ao servidor na requisição HTTP e interpretar o resultado da solicitação.



Verbos HTTP



Principais

GET

O método `GET` solicita a representação de um recurso específico. Requisições utilizando o método `GET` devem retornar apenas dados.

POST

O método `POST` é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.

PUT

O método `PUT` substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.

DELETE

O método `DELETE` remove um recurso específico.

Verbos HTTP

Principais

GET

O método GET solicit Requisições utilizando o método GET devem reto

POST

O método POST é m recurso específico, frequentemente cau u efeitos colaterais no servidor.

PUT

O método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.

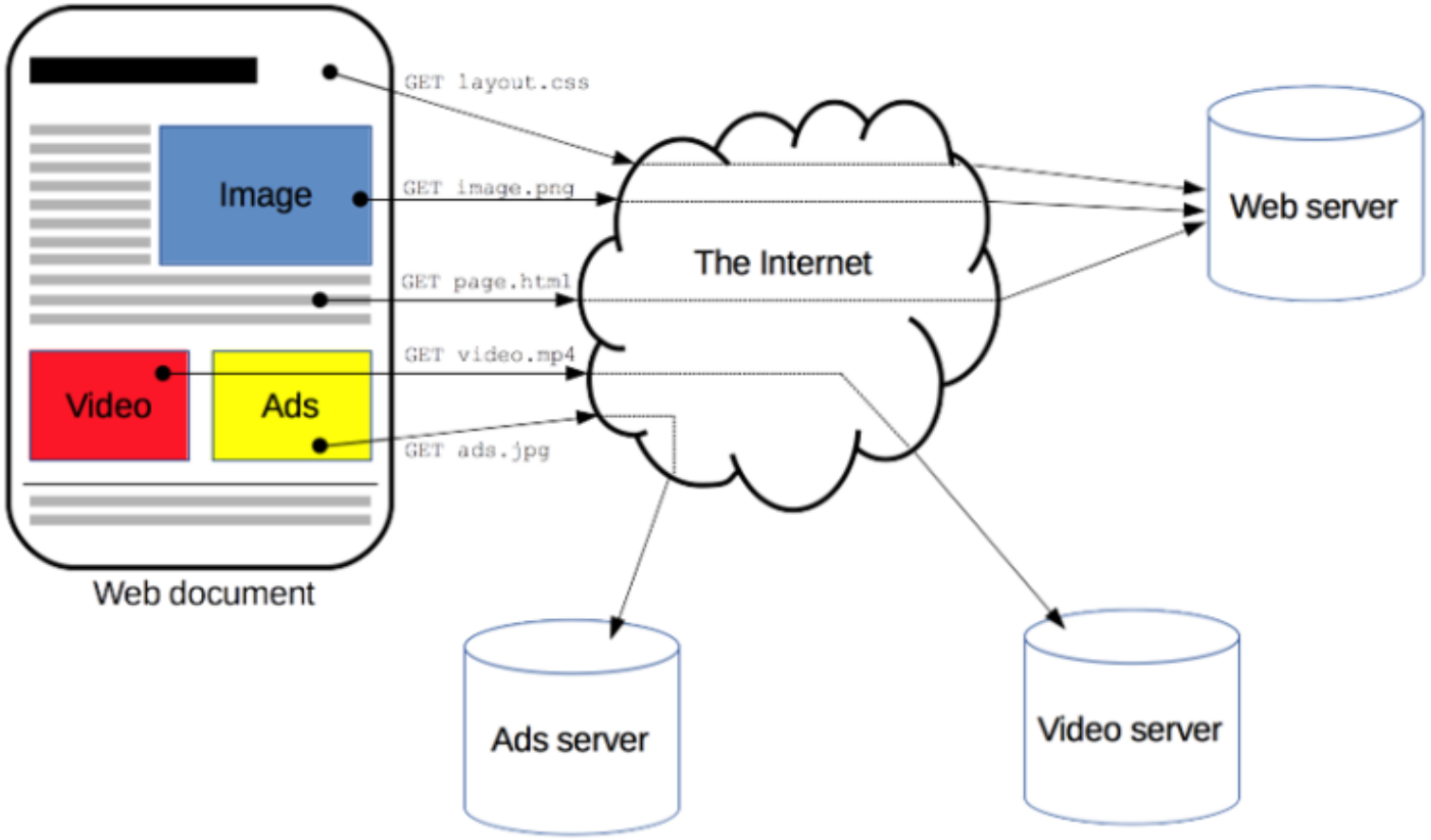
DELETE

O método DELETE remove um recurso específico.

Para mais métodos de requisição

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

Verbos HTTP



Um documento completo é reconstruído a partir de diferentes sub-documentos obtidos, como por exemplo texto, descrição do layout, imagens, vídeos, scripts e muito mais.

Status HTTP

Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi corretamente concluída. As respostas são agrupadas em cinco classes:

- Respostas de informação (100-199),
- Respostas de sucesso (200-299),
- Redirecionamentos (300-399)
- Erros do cliente (400-499)
- Erros do servidor (500-599).

Status HTTP

Principais

200 OK: Estas requisição foi bem sucedida. O significado do sucesso varia de acordo com o método HTTP

201 Created: A requisição foi bem sucedida e um novo recurso foi criado como resultado. Esta é uma tipica resposta enviada após uma requisição POST.

400 Bad Request: Essa resposta significa que o servidor não entendeu a requisição pois está com uma sintaxe inválida.

401 Unauthorized: Embora o padrão HTTP especifique "unauthorized", semanticamente, essa resposta significa "unauthenticated". Ou seja, o cliente deve se autenticar para obter a resposta solicitada.

404 Not Found: O servidor não pode encontrar o recurso solicitado. Este código de resposta talvez seja o mais famoso devido à frequência com que acontece na web.

500 Internal Server Error: O servidor encontrou uma situação com a qual não sabe lidar.

Status HTTP

Principais

200 OK: Estas requisição foi bem sucedida. O significado do sucesso varia de acordo com o método H

201 Created: A requisiç rso foi criado como resultado. Esta é uma tip ção POST.

400 Bad Request: Essa r tendeu a requisição pois está com uma sinta

401 Unauthorized: E ue "unauthorized", semanticamente, essa re seja, o cliente deve se autenticar para obter

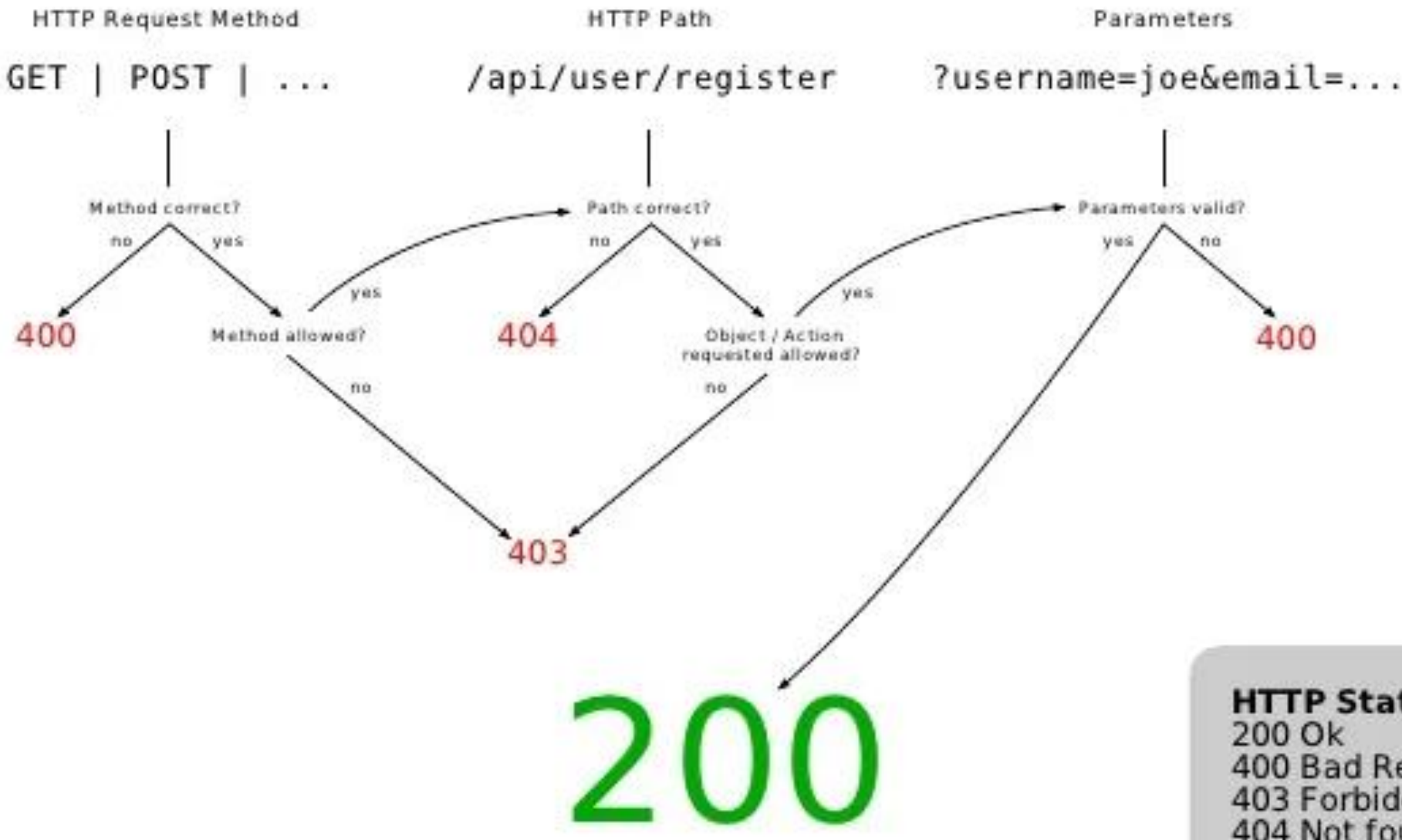
404 Not Found: O servidor não pode encontrar o recurso solicitado. Este código de resposta talvez seja o mais famoso devido à frequência com que acontece na web.

500 Internal Server Error: O servidor encontrou uma situação com a qual não sabe lidar.

Para mais status

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

REST HTTP Response Code Cheat Sheet



HTTP Status Codes
200 Ok
400 Bad Request
403 Forbidden
404 Not found

Cabeçalhos (Headers) HTTP

Os **cabeçalhos HTTP** permitem que o cliente e o servidor passem informações adicionais com uma solicitação ou resposta HTTP.

Um cabeçalho HTTP consiste em :

- seu nome que não diferencia maiúsculas de minúsculas seguido por dois pontos (:)
- e, em seguida, por seu valor. Espaço em branco antes do valor ser ignorado.

Content-Type: text/html

Cabeçalhos (Headers) HTTP

Há numerosos cabeçalhos de requisição disponíveis. Eles podem ser divididos em vários grupos:

- **Cabeçalhos de requisição**, como ***User-Agent***, ***Accept-Type***, modificam a requisição, especificando-a mais (como ***Accept-Language***), dando-a contexto (como ***Referer***), ou restringindo-a condicionalmente (como ***If-None***).
- **Cabeçalho de resposta**, retém informações adicionais sobre a resposta como a localização de um outro servidor (**Location**) ou sobre o servidor (**Server**) que o fornece.

Cabeçalhos (Headers) HTTP

Há numerosos cabeçalhos de requisição disponíveis. Eles podem ser divididos em vários grupos:

- **Cabeçalhos de representação,** contêm informações sobre o corpo do recurso, como seu tipo MIME (***Content-Type***) ou codificação/compactação aplicada (***Content-Encoding***).
- **Cabeçalhos de payload**, como ***Content-Length*** que se aplicam ao corpo da mensagem. Obviamente este cabeçalho não será transmitido se não houver corpo na requisição.

Cabeçalhos (Headers) HTTP

Cabeçalhos também podem ser organizados por categoria de uso:

- Authentication
- Caching
- Client hints
- Conditionals
- Connection management
- Content negotiation
- Controls
- Cookies
- CORS

Lista em: https://developer.mozilla.org/en-US/docs/Web/HTTP

Cabeçalhos (Headers) HTTP

Veja a seguir alguns cabeçalhos de resposta e representação após uma solicitação GET.

```
200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Mon, 18 Jul 2016 16:06:00 GMT
Etag: "c561c68d0ba92bbeb8b0f612a9199f722e3a621a"
Keep-Alive: timeout=5, max=997
Last-Modified: Mon, 18 Jul 2016 02:36:04 GMT
Server: Apache
Set-Cookie: mykey=myvalue; expires=Mon, 17-Jul-2017 16:06:00 GMT; Max-
Age=31449600; Path=/; secure
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Backend-Server: developer2.webapp.scl3.mozilla.com
X-Cache-Info: not cacheable; meta data too large
X-kuma-revision: 1085259
x-frame-options: DENY
```

Na dúvida sobre um Cabeçalho específico? Busque em

https://developer.mozilla.org/

TP

solicitação GET

```
trol-Allow-Origin: *
: Keep-Alive
coding: gzip
pe: text/html; charset=utf-8
Date: Mon, 18 Jul 2016 16:06:00 GMT
```


```
X-Cache-Info: not cacheable; meta data too large
X-kuma-revision: 1085259
x-frame-options: DENY
```

Referências

- https://pt.wikipedia.org/wiki/Tim_Berners-Lee
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP
- https://www.websiterating.com/resources/http-status-codes-cheat-sheet/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
- https://riptutorial.com/http