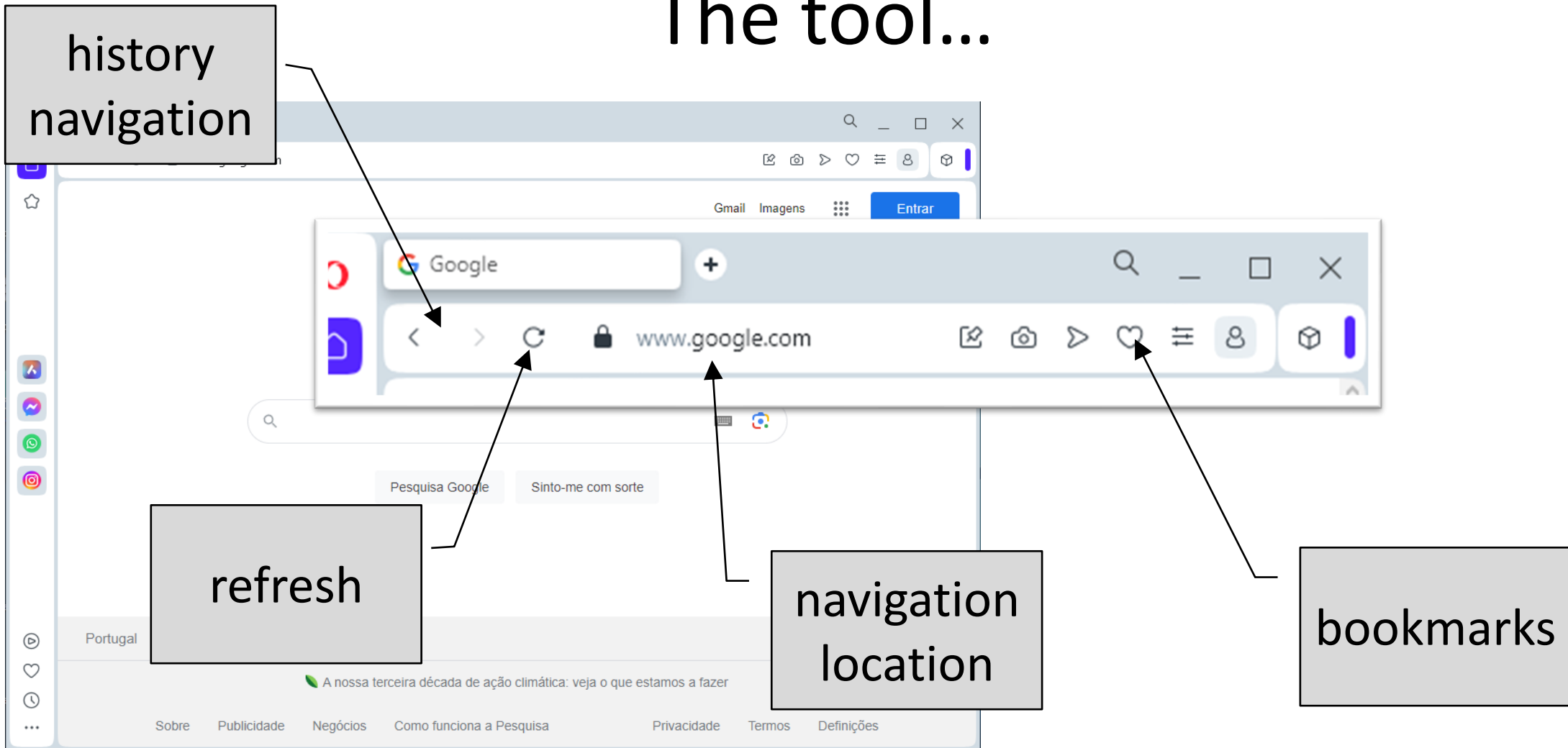


Request-response sequence of a web page

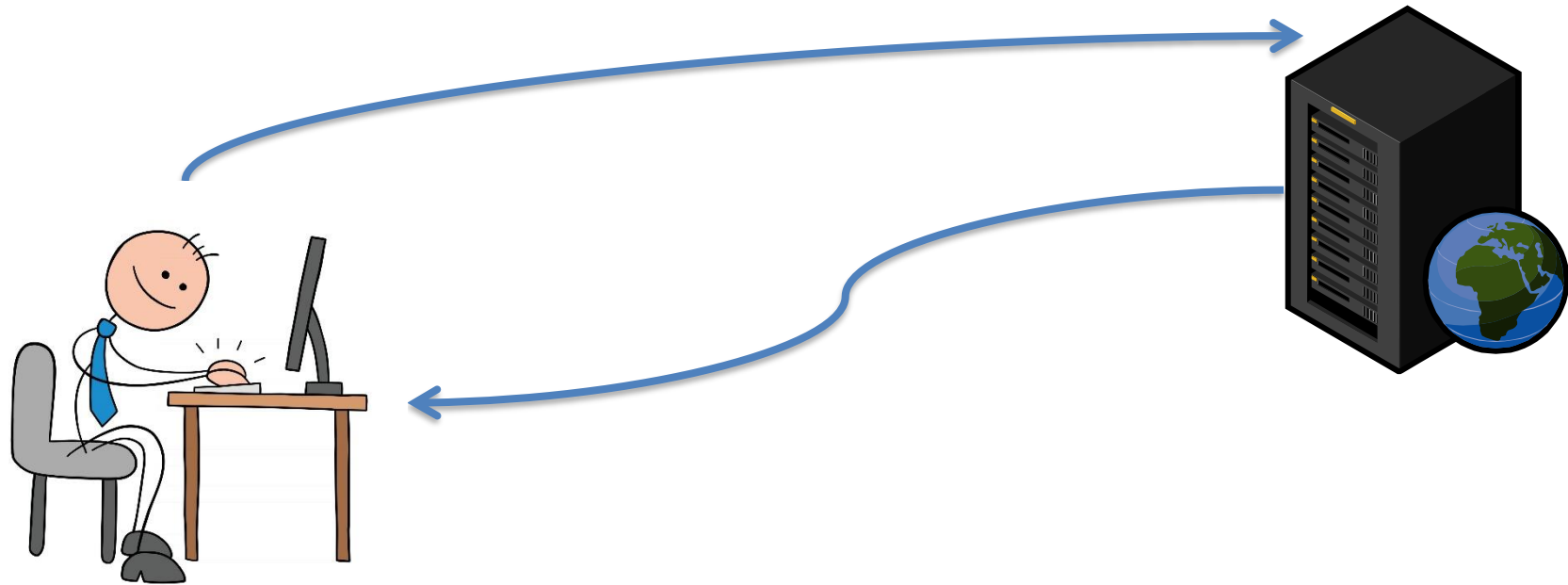
Web Engineering

The tool...



Browse a website...

...how it works?

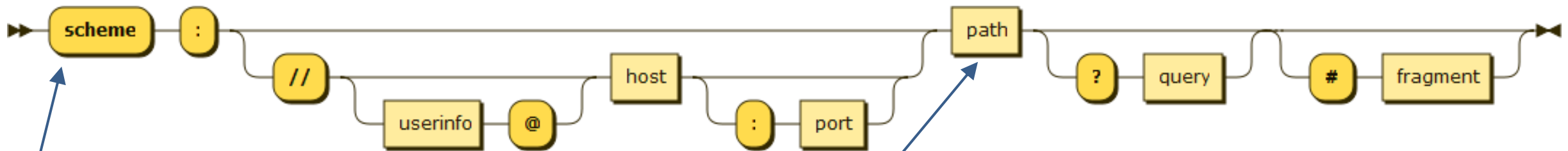


<http://www.diocese-vilareal.pt/a-diocese.html>

What is a URL?

Universal Resource Locator

`scheme : [// [user:password@] host [:port]] [path] [?query] [#anchor]`



http,
https,
ftp,
mailto,
file,
data,
and irc

- URLs can only be sent over the Internet using the ASCII character-set.
- Since URLs often contain characters outside the ASCII set, the URL has to be converted into a valid ASCII format.
- URL encoding replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits.
- URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign or with %20.

Universal Resource Locator

Examples:

<https://www.sapo.pt/>


<ftp://home.utad.pt/~lfb>

<https://www.continente.pt/pt-pt/public/Pages/searchresults.aspx?k=arroz>

<https://www.fnac.pt/livro/h5#bl=MMlivros>

<https://realvitur.pt/pesquisa/Mal%C3%A1sia>

url encoded
From UTF-8
(default character-set in HTML5)

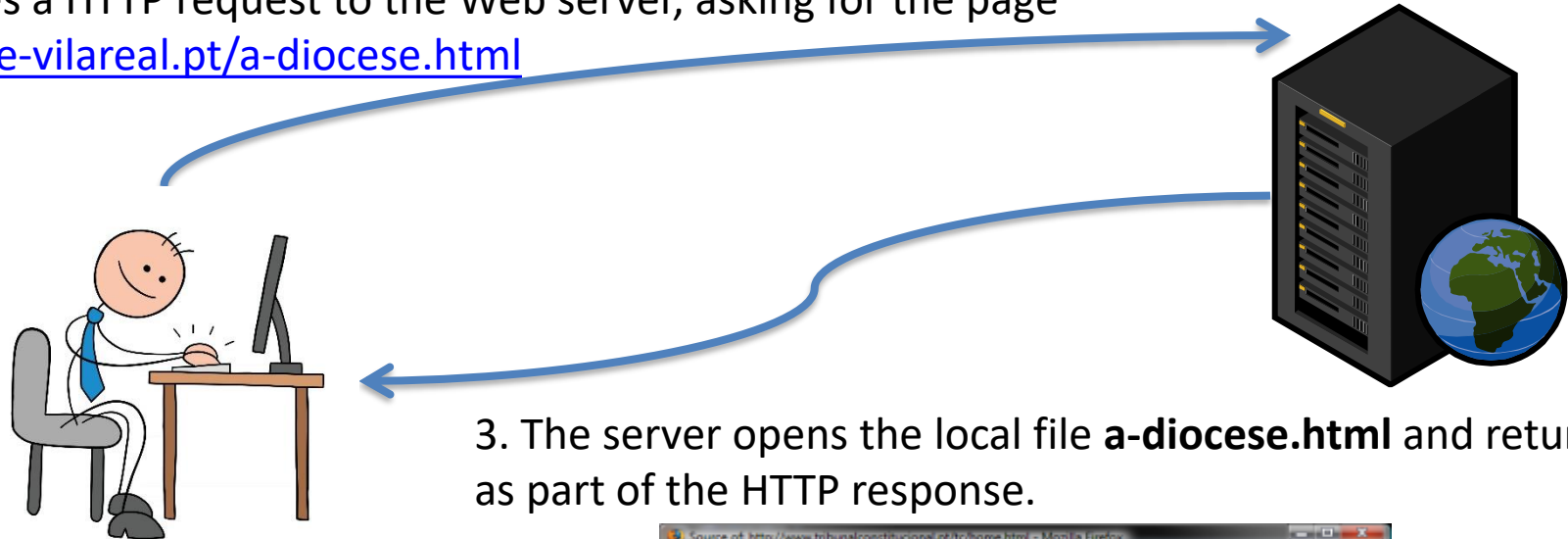


<https://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html#sec2.2>

Browsing a website...

1. The user inserts <http://www.diocese-vilareal.pt/a-diocese.html> in browser

2. The browser does a HTTP request to the Web server, asking for the page <http://www.diocese-vilareal.pt/a-diocese.html>



3. The server opens the local file **a-diocese.html** and returns it as part of the HTTP response.

```
Source of http://www.tribunalconstitucional.pt/co/home.html - Mozilla Firefox
File Edit View Help

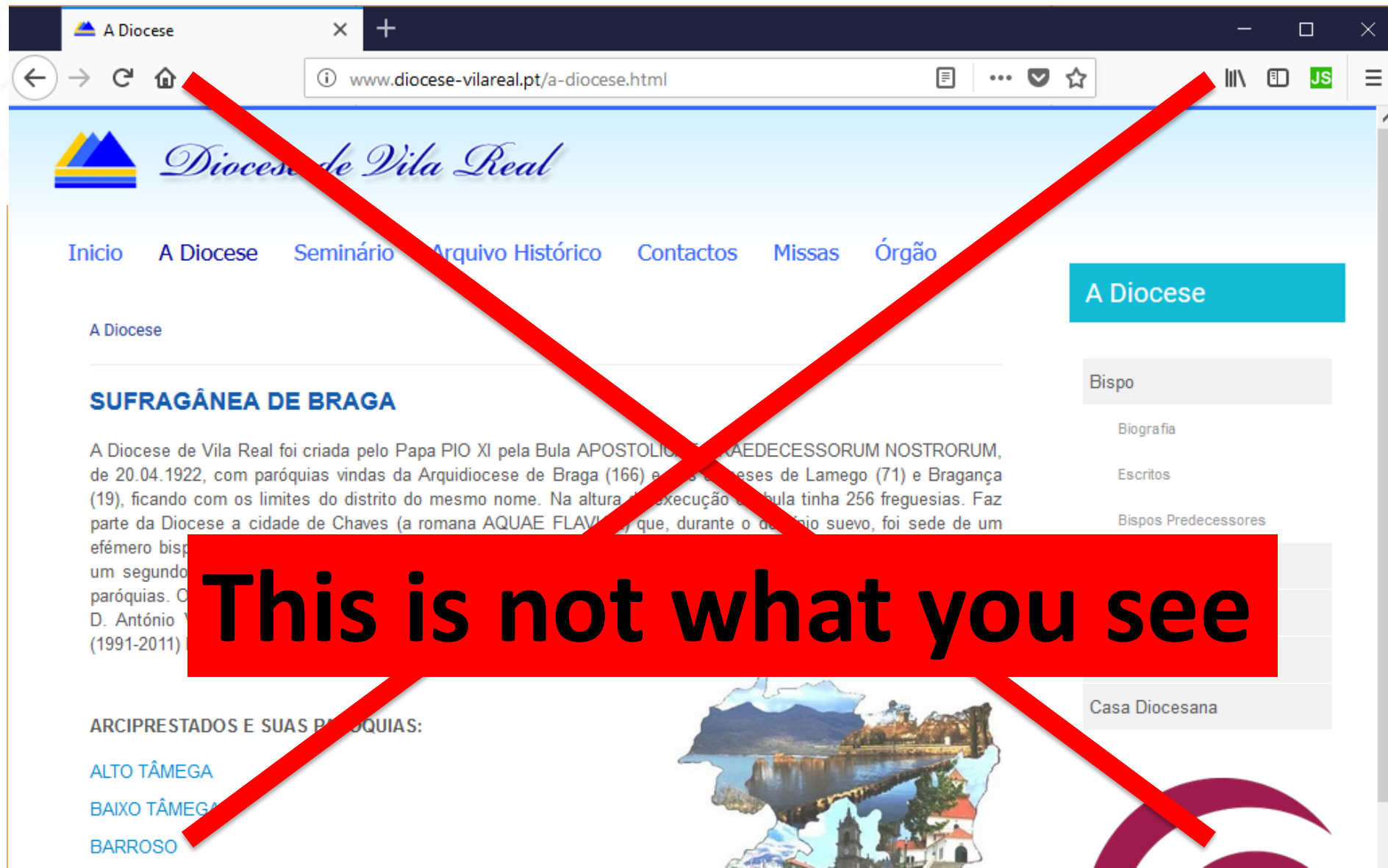
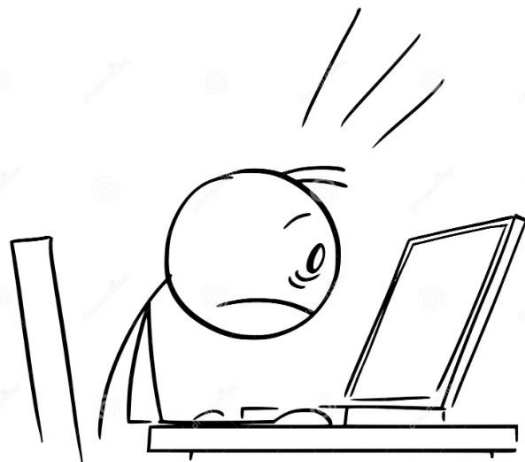
<html>
<head>
<title>TC > Bem-vindos</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="http://www.tribunalconstitucional.pt/co/estilo.css" rel="stylesheet" type="text/css">
<style type="text/css">
<!--
/* CSS Document */

margin-left: auto;
margin-right: auto;
float: center;
text-align: justify;

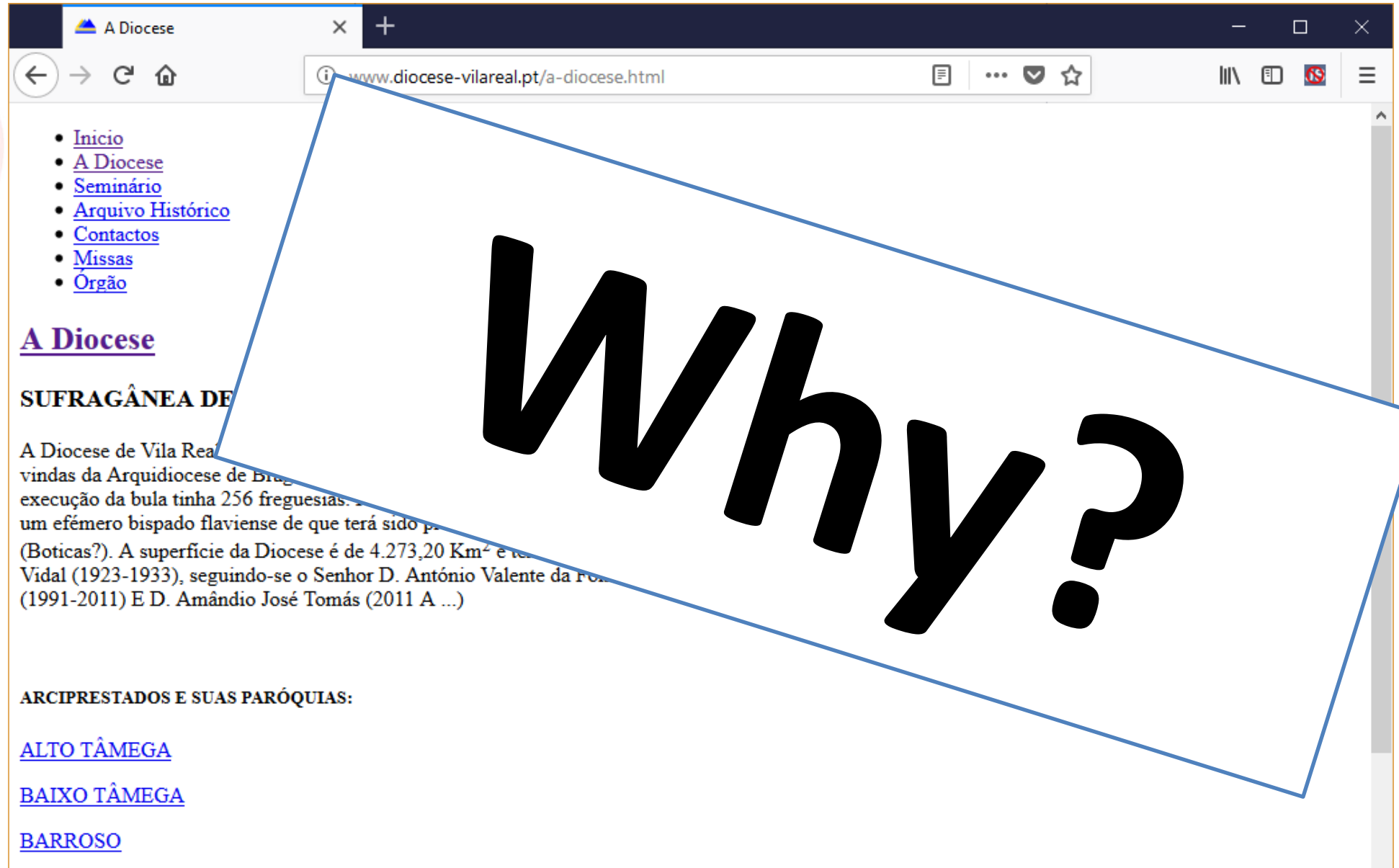
line 12, Col 1
```

(Request for static content)

Expected result...



This is what you see



A screenshot of a web browser displaying the website www.diocese-vilareal.pt/a-diocese.html. The browser's developer tools are open, showing the file system structure of the website. The file system tree on the left includes:

- www.diocese-vilareal.pt
 - images
 - media
 - com_attachments
 - css
 - attachments_hide.css
 - attachments_list.css
 - js
 - attachments_refresh.js
 - jui/js
 - bootstrap.min.js?ee731f2c5f74876e9a...
 - jquery-migrate.min.js?ee731f2c5f7487...
 - jquery-noconflict.js?ee731f2c5f74876e...
 - jquery.min.js?ee731f2c5f74876e9a2a4...
 - system
 - css
 - modal.css?ee731f2c5f74876e9a2a4e...
 - js
 - a-diocese.html
 - templates/classyhome
 - css
 - images
 - bulletBlue.png
 - facebook.png
 - footerBg.png
 - headerBg.png
 - logo.png

The main content area of the website shows the header with the logo and navigation links: Início, A Diocese, Seminário, Arquivo Histórico, Contactos, Missas, Órgão. The main heading is "A Diocese". Below it, there is a section titled "SUFAGÂNEA DE BR" (partially visible) and a paragraph of text. To the right, there is a sidebar with the title "A Diocese" and a list of links: Biografia, Escritos, Bispos Predecessores, Serviços Centrais, Clero, Pastoral, Casa Diocesana. At the bottom right, there is a logo for "CENTRO CATÓLICO CULTURA VILA REAL".

Three green callout boxes are overlaid on the image, pointing to specific files in the file system tree:

- Images**: Points to the `images` folder.
- Scripts**: Points to the `js` folder under `system`.
- Style sheets**: Points to the `css` folder under `system`.

A large orange banner at the bottom of the image contains the text: **analyzing the contents associated with this page...**

Browsing a website...

1. The user inserts <http://www.diocese-vilareal.pt/a-diocese.html> in browser
2. The browser does a HTTP request to the Web server, asking for the page <http://www.diocese-vilareal.pt/a-diocese.html>
3. The server opens the local file **a-diocese.html** and returns it as part of the HTTP response.

The diagram illustrates the process of browsing a website. It shows a user sitting at a desk with a computer, representing the client. A blue arrow points from the user's computer to a server rack, representing the HTTP request. The server rack is labeled with a globe icon, indicating it's a web server. Another blue arrow points from the server back to the user's computer, representing the HTTP response. Below the server, there is a screenshot of a web browser window displaying the HTML code for the requested page, `a-diocese.html`. The code includes a DOCTYPE declaration, a charset declaration, a meta tag for the page title, a link tag for a stylesheet, and a body tag with a CSS document type. The code is shown in a monospaced font, typical of a code editor or browser's developer tools.

the browser
presented the page
corresponding to the
HTML code received in
request.

g a website...

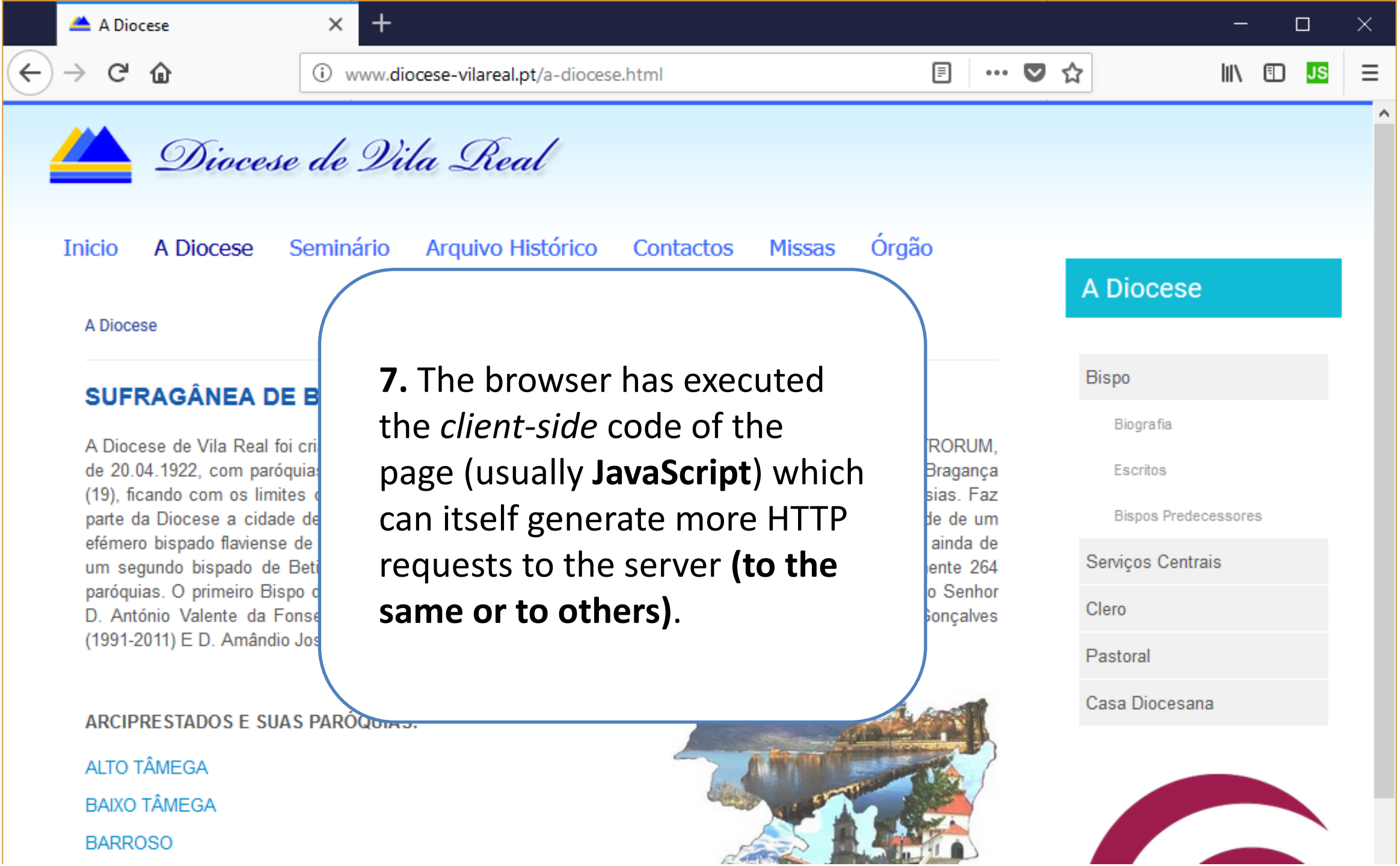
[http://www.a-diocese.html](#) in browser

server, asking for the page

The server opens the local file **a-diocese.html** and returns it as part of the HTTP response.

```
<html>  
  <head>  
    <title><C> > How-random</title>  
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
    <link href="http://www.k2b0n0d0m0l10n0m0l.g170/v0r1110.c0f" rel="stylesheet"  
      type="text/css">  
    <style type="text/css">  
      <!--  
/* CSS Document */  
  
#background {  
  text-align: center;  
}  
  
#padding {  
  margin-left: auto;  
  margin-right: auto;  
  float: center;  
  text-align: justify;  
  <br/>  
}</-->  
</style>  
</head>  
<body>
```

6. As he received the HTTP responses, the browser presented the images and other elements.



7. The browser has executed the *client-side* code of the page (usually **JavaScript**) which can itself generate more HTTP requests to the server (**to the same or to others**).

Why browser gets additional Components to a Web Page?

How to insert an image:

```

```

external sources...

Point to an external JavaScript file:

```
<script src="myscripts.js"></script>
```

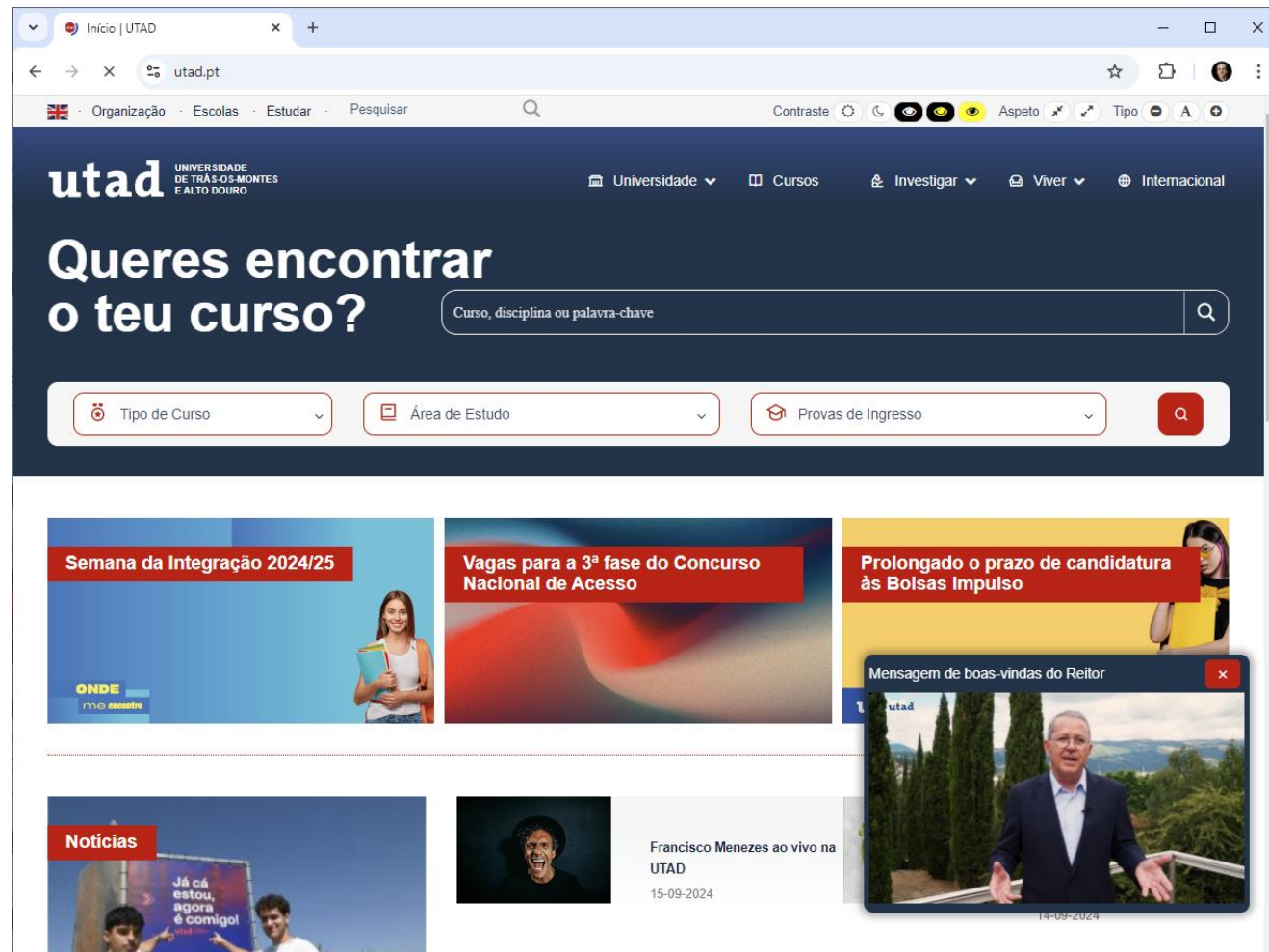
```
<head>
```

```
<link rel="stylesheet" type="text/css" href="mystyle.css">
```

```
</head>
```

running scripts...

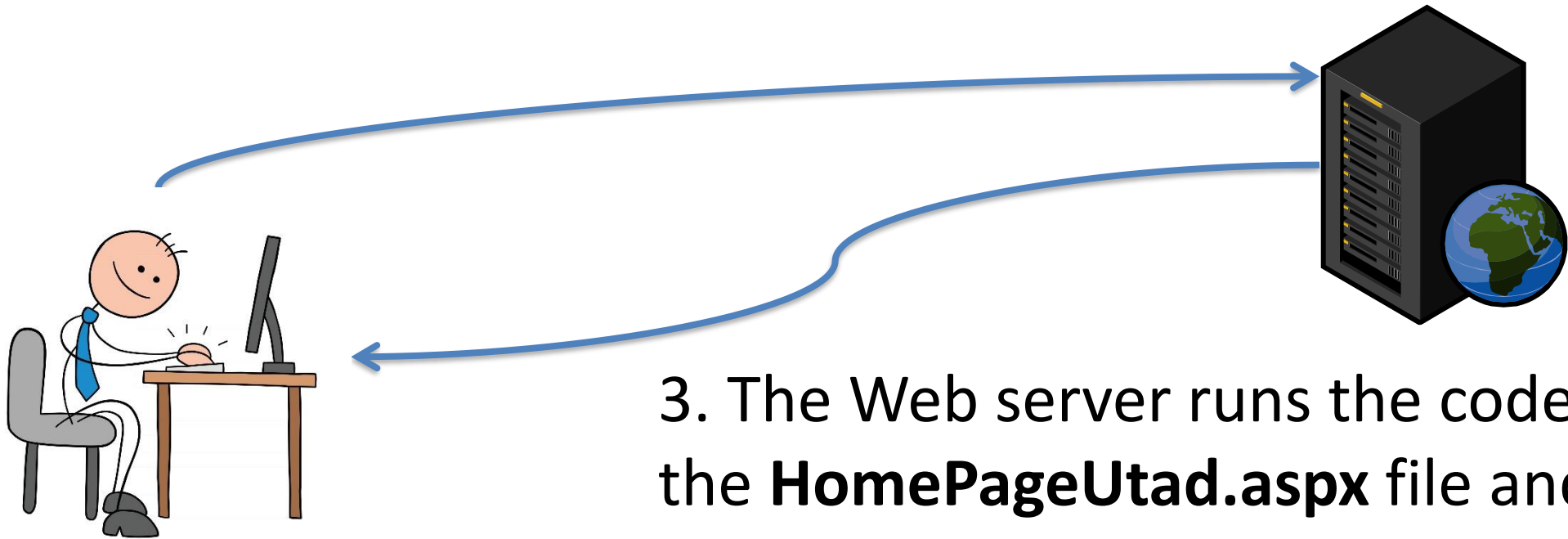
```
$(document).ready(function() {  
    $.ajax({  
        url: "http://uads.ir/l.php?s=125125&w=5307cd5c027373e1773c9869",  
        dataType: "script",  
        cache: true  
    });  
});
```



How does it work with dynamic content?

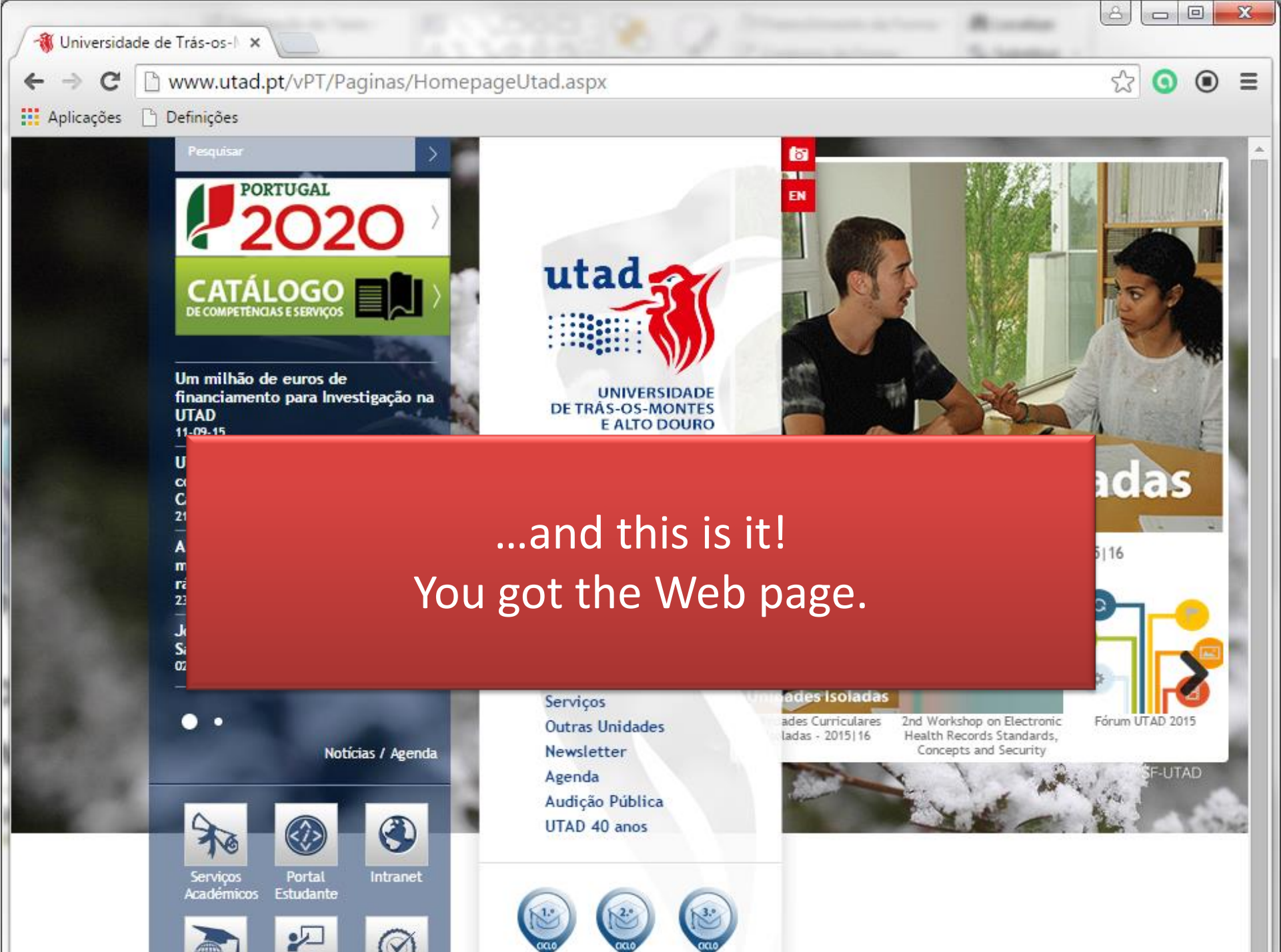
1. User inserts <http://www.utad.pt/vPT/Paginas/HomepageUtad.aspx> in the browser

2. The browser makes an HTTP request to the Web server, requesting the page <http://www.utad.pt/vPT/Paginas/HomepageUtad.aspx>



(Request for dynamic content)

3. The Web server runs the code in the **HomePageUtad.aspx** file and produces the HTTP response, which can contain HTML or other code, including binaries.

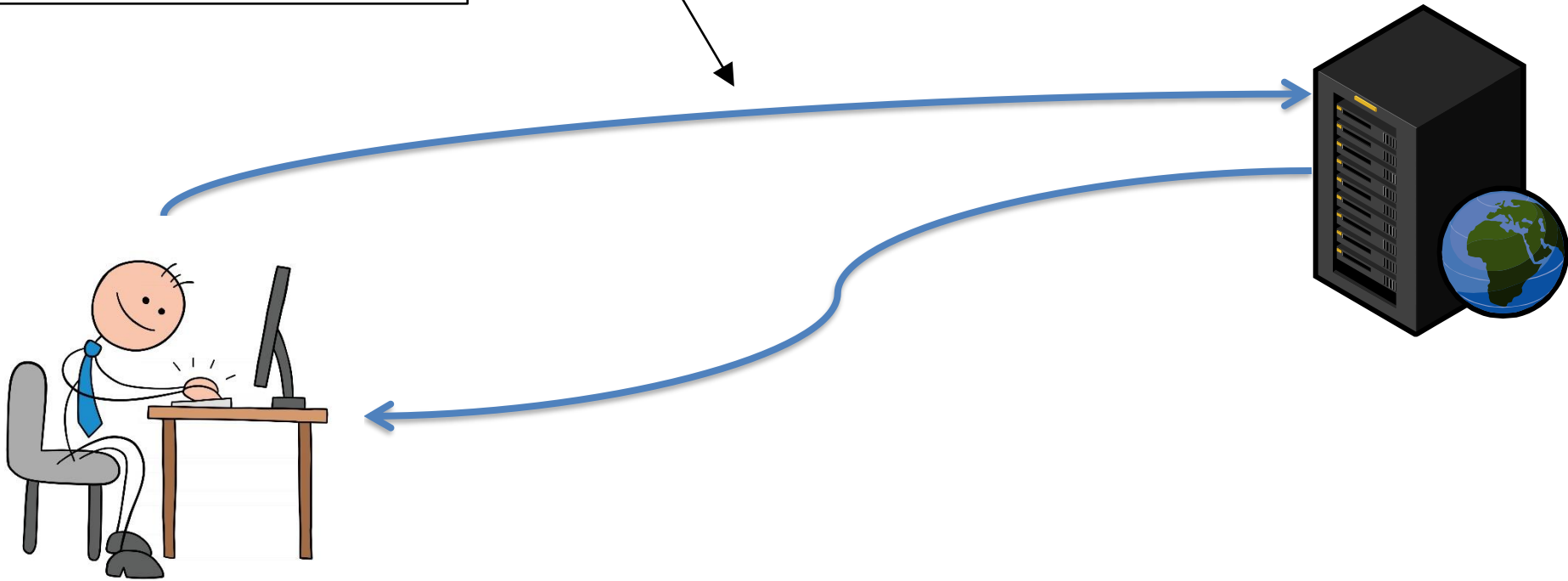


...and this is it!
You got the Web page.

HTTP Protocol Terminology

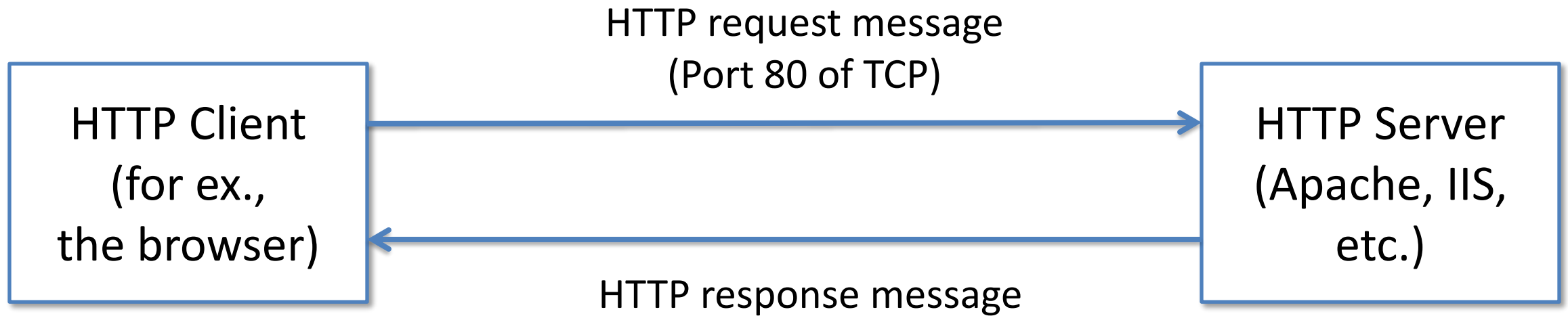
Web Engineering

What is this
information really
about?



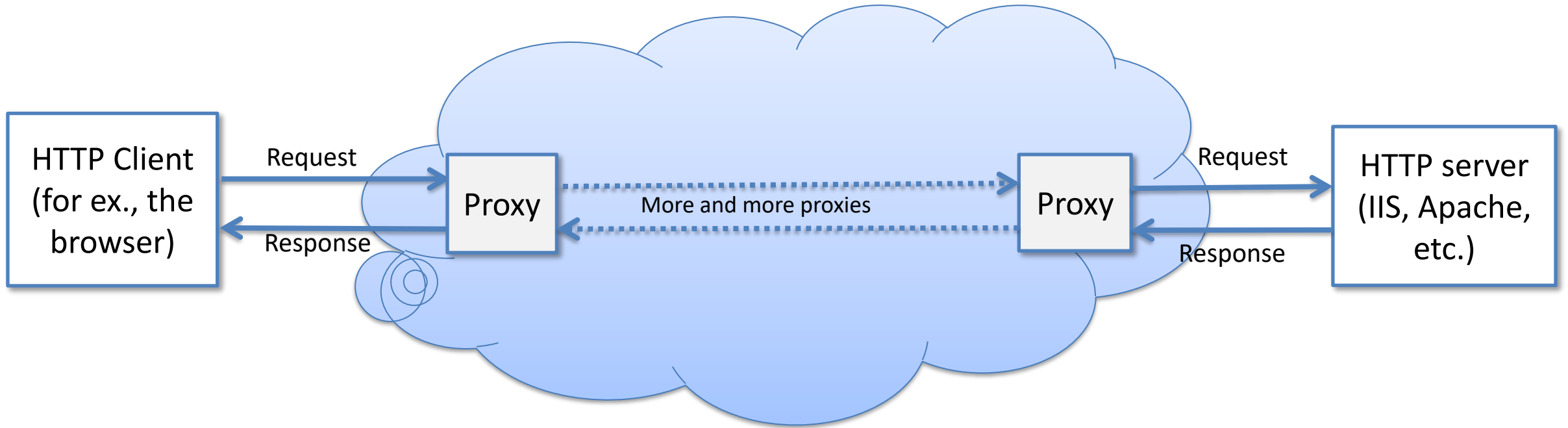
<http://www.diocese-vilareal.pt/a-diocese.html>

basic agents...



HTTP is a stateless protocol.
What does that mean?

real world agents...

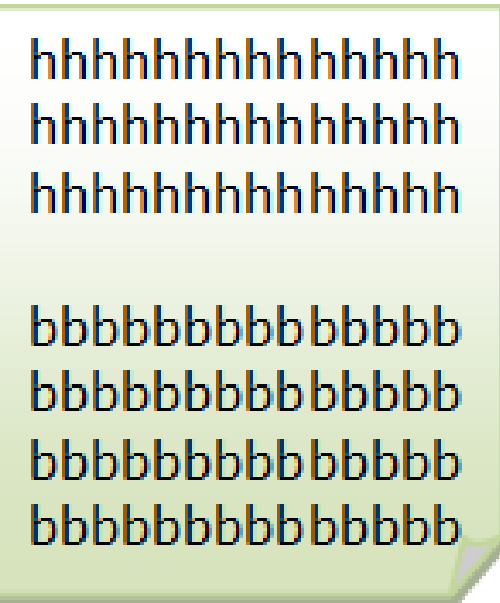


Proxies?

Proxies act as intermediaries between clients (such as web browsers) and servers, and they can serve various purposes in a web application environment.

The use of proxies in web application communication can enhance security, performance, and scalability while providing various services like caching, load balancing, and content filtering.

HTTP protocol Messages

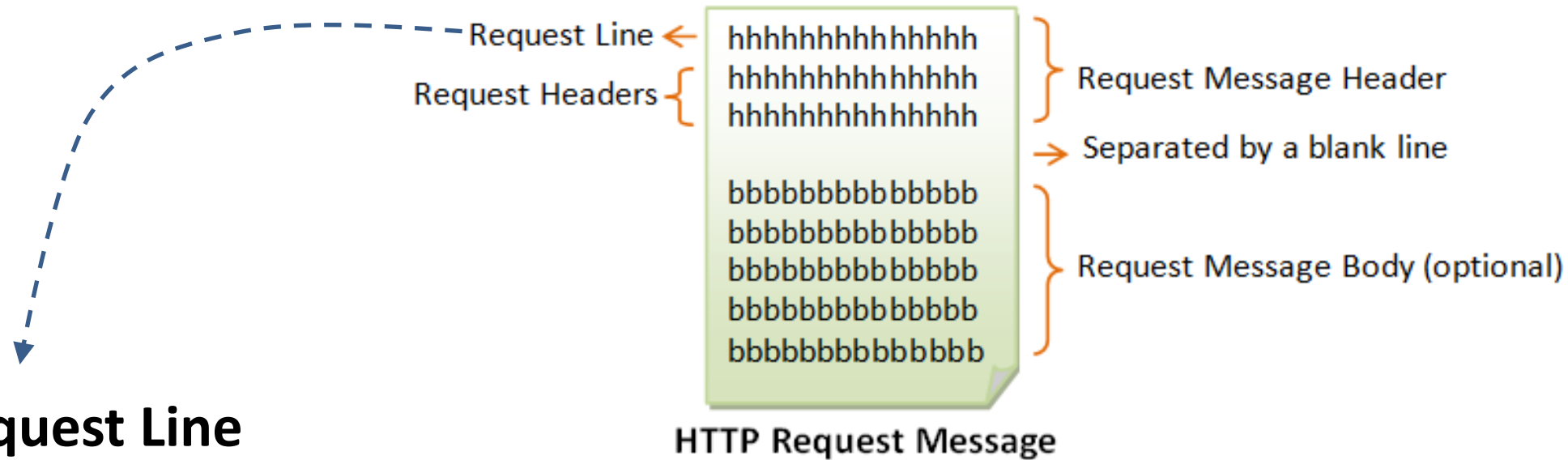


Message Body (optional)

→ A blank line separates the header and body

HTTP Messages

HTTP Request Messages



Request Line

The first line of the header is called the request line, followed by optional request headers.

The request line has the following syntax:

request-method-name request-URI HTTP-version

- *request-method-name*: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, OPTIONS, etc.
- *request-URI*: specifies the resource requested.
- *HTTP-version*: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.

HTTP Request Message

example

```
GET /doc/test.html HTTP/1.1
```

```
Host: www.test101.com
```

```
Accept: image/gif, image/jpeg, */*
```

```
Accept-Language: en-us
```

```
Accept-Encoding: gzip, deflate
```

```
User-Agent: Mozilla/4.0
```

```
Content-Length: 35
```

```
bookId=12345&author=Tan+Ah+Teck
```

Request Line

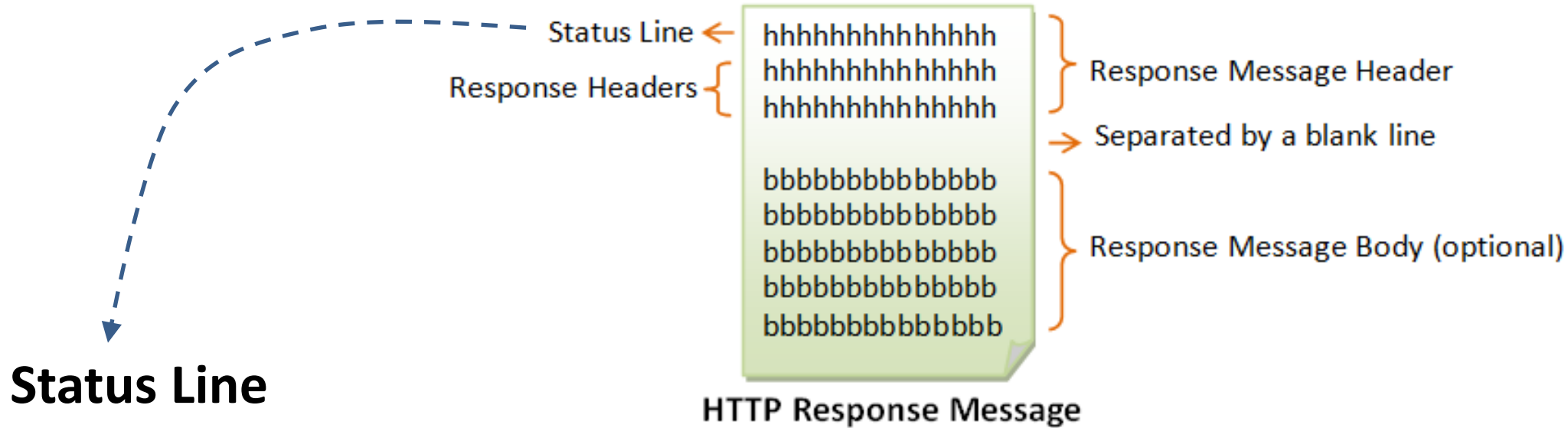
Request Headers

Request
Message
Header

A blank line separates header & body

Request Message Body

HTTP Response Messages



Status Line

The first line is called the status line, followed by optional response header(s).

The status line has the following syntax:

HTTP-version status-code reason-phrase

- *HTTP-version:* The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- *status-code:* a 3-digit number generated by the server to reflect the outcome of the request.
- *reason-phrase:* gives a short explanation to the status code.

Common status code and reason phrase are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error"

HTTP Response Message

example

HTTP/1.1 200 OK

Date: Sun, 08 Feb xxxx 01:11:12 GMT

Server: Apache/1.3.29 (Win32)

Last-Modified: Sat, 07 Feb xxxx

ETag: "0-23-4024c3a5"

Accept-Ranges: bytes

Content-Length: 35

Connection: close

Content-Type: text/html

<h1>My Home page</h1>

Status Line

Response Headers

Response
Message
Header

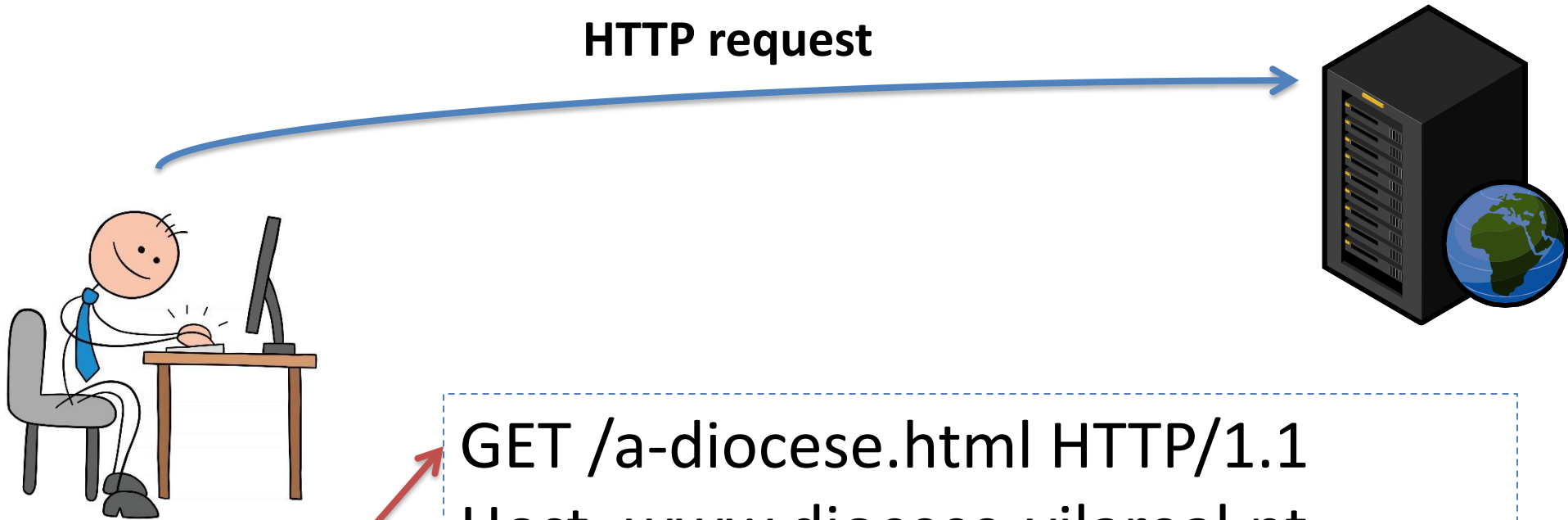
A blank line separates header & body

Response Message Body

e.g. requesting a HTML page

`http://www.diocese-vilareal.pt/a-diocese.html`

HTTP request

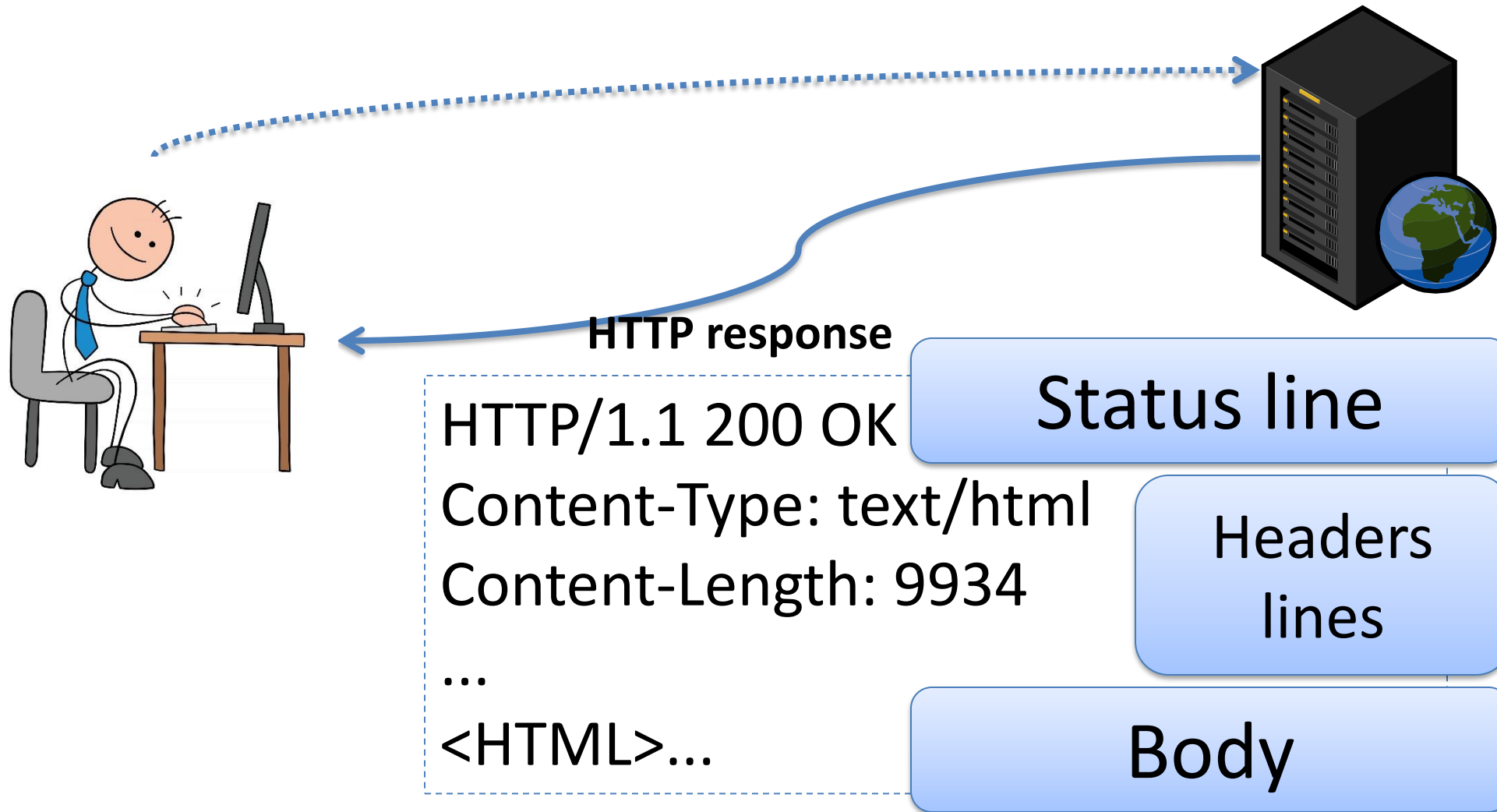


```
GET /a-diocese.html HTTP/1.1
Host: www.diocese-vilareal.pt
...
```

Request line

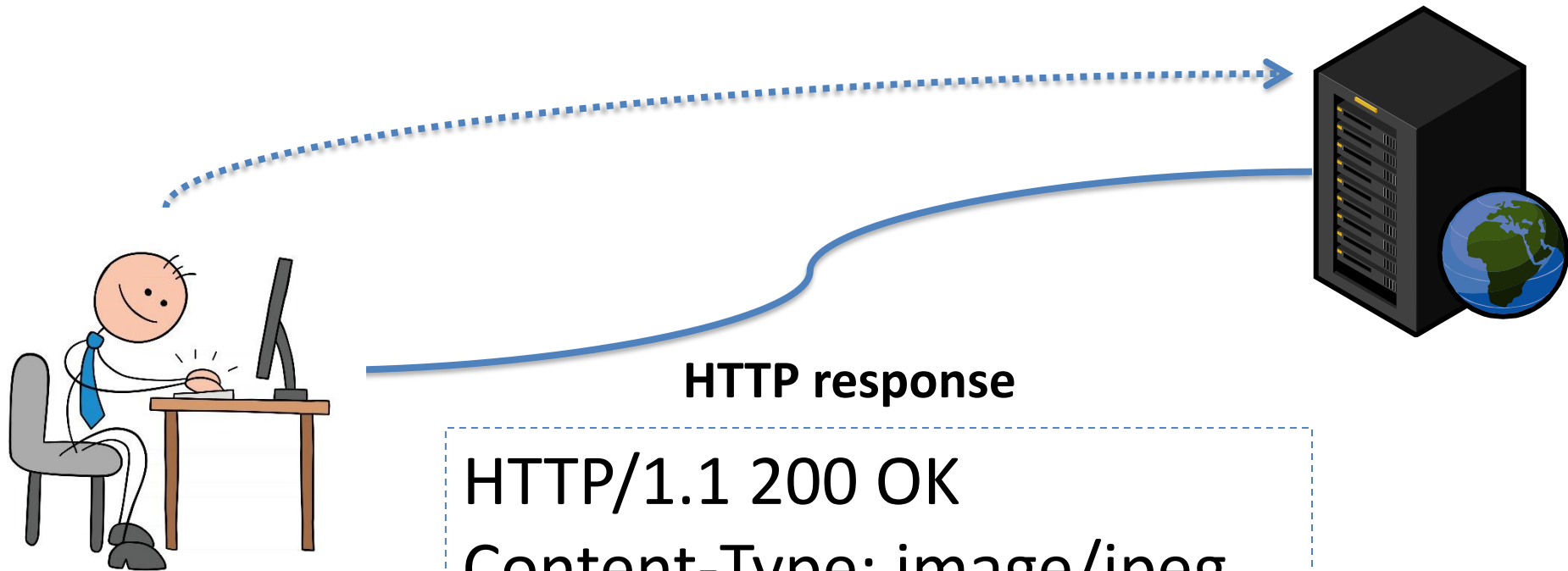
Header line

<http://www.diocese-vilareal.pt/a-diocese.html>



e.g. requesting a image

<http://www.diocese-vilareal.pt/images/stories/diocese.jpg>



HTTP response

HTTP/1.1 200 OK

Content-Type: image/jpeg

Content-Length: 36075

...

ÿØÿà JFIF ڤ

ڤ ڤÛ C

HTTP requests methods

- HTTP 0.9

- GET

Request a resource (Web page, image, etc.)

- HTTP 1.0

- HEAD

Ask only the headers of a resource (headers),
not the body.

To compare versions, eg.

- POST

Ask to send information in the request body (for
submitting forms, for example)

- HTTP 1.1

- PUT

Ask to save a resource on the server (upload)

- DELETE

- OPTIONS

....

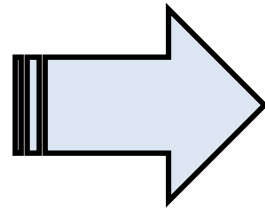
- TRACE

- CONNECT

- PATCH

Standard requests from web applications or websites are based only on GET and POST methods

- HTTP 0.9
 - **GET**
- HTTP 1.0
 - HEAD
 - **POST**
- HTTP 1.1
 - PUT
 - DELETE
 - OPTIONS
 - TRACE
 - CONNECT
 - PATCH



RESTful APIs use the following methods

HTTP METHOD	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Partial Update/Modify
DELETE	Delete



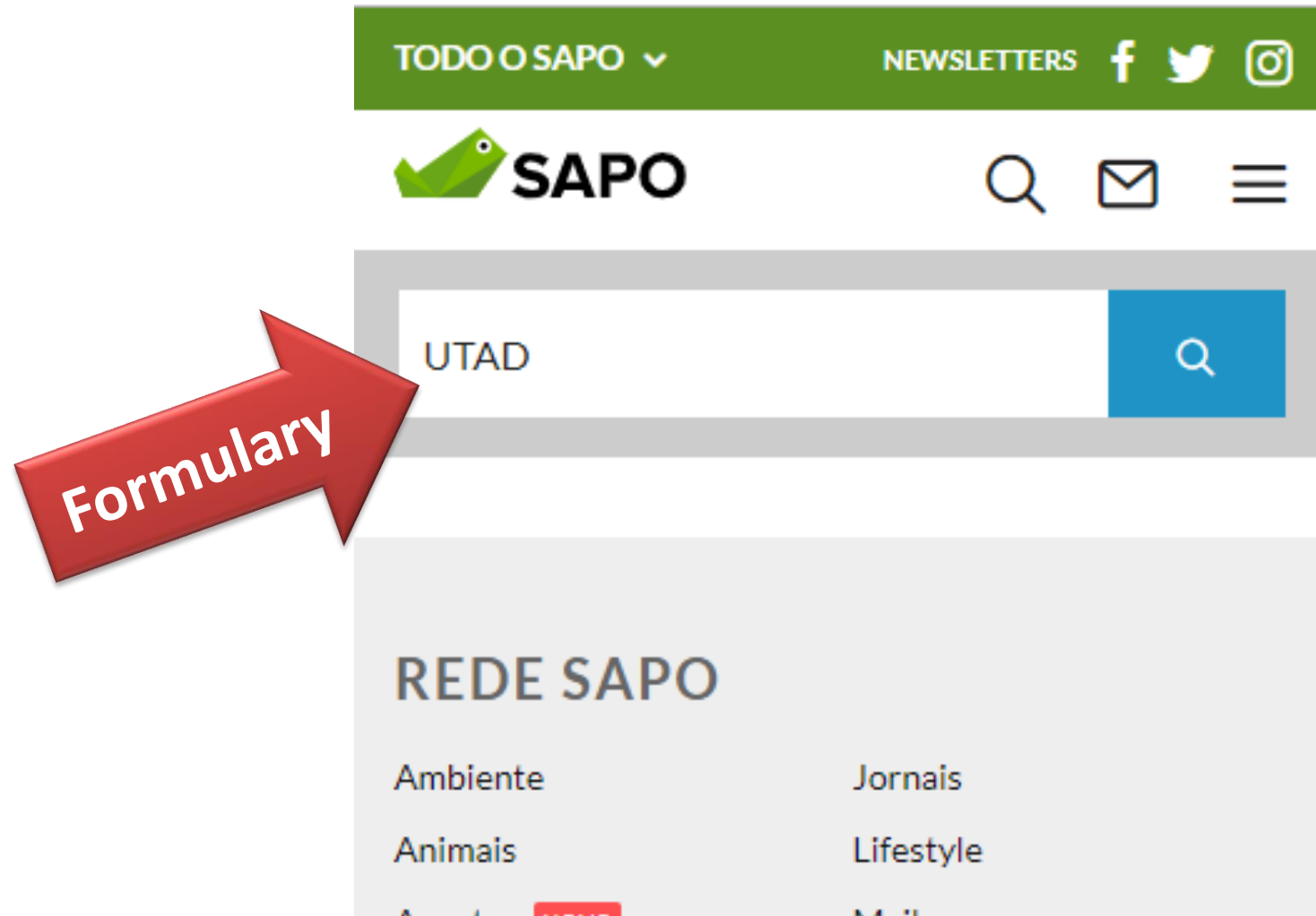
PESQUISAR



- | | | | |
|-------------------------|--------------|----------------------|-------------|
| Jornais | Celebridades | Lifestyle | Viagens |
| Apostas ^{NOVO} | Cinema e TV | Mail | Vídeos |
| Astrologia | Desporto | Notícias | Voucher |
| Beleza ^{NOVO} | Economia | Oficinas | Angola |
| Bilheteira | Emprego | Receitas | Cabo Verde |
| Blogs | Farmácias | Saúde | Moçambique |
| Carros | Folhetos | Tecnologia | Timor-Leste |
| Casas | Hotéis | Tempo | |
| | Jornais | Transferir ficheiros | |

```
<form action="/pesquisa" method="get">
    ...
    <legend>Pesquisa SAPO</legend>
    <input name="q" type="search" placeholder="Insira o texto a pesquisar">
    <button type="submit" value="Pesquisar">Pesquisar</button>
    ...
</form>
```


What is the difference between using "GET" or "POST" in this form?



The image shows a screenshot of the SAPO website's search interface. At the top, there is a green navigation bar with the text "TODO O SAPO" and a dropdown arrow, followed by "NEWSLETTERS" and social media icons for Facebook, Twitter, and Instagram. Below this is a white header with the SAPO logo (a green frog) and search, email, and menu icons. The main search area features a light gray bar containing a white search input field with the text "UTAD" and a blue search button with a magnifying glass icon. A red arrow points from the left towards the search input field, with the word "Formulary" written in white text inside the arrow. Below the search bar, the section "REDE SAPO" is visible, listing various categories like "Ambiente", "Jornais", "Animais", and "Lifestyle".

```
<form action="/pesquisa" method="get">
```

```
...
```

```
<legend>Pesquisa SAPO</legend>
```

```
<input name="q" type="search" placeholder="Insira o texto a pesquisar">
```

```
<button type="submit" value="Pesquisar">Pesquisar</button>
```

```
...
```

```
</form>
```

HTTP Request



versão simplificada
da *query string*



```
GET /pesquisa?q=UTAD HTTP/1.1  
Host: www.sapo.pt  
...
```



Pesquisa — SAPO



<https://www.sapo.pt/pesquisa?q=UTAD#gsc.tab=0&gsc.q=UTAD&gsc.page=1>

```
<form action="/pesquisa" method="post">
```

```
...
```

```
<legend>Pesquisa SAPO</legend>
```

```
<input name="q" type="search" placeholder="Insira o texto a pesquisar">
```

```
<button type="submit" value="Pesquisar">Pesquisar</button>
```

```
...
```

```
</form>
```

HTTP Request



versão simplificada
do recurso (sem
parâmetros)



POST /pesquisa HTTP/1.1
Host: www.sapo.pt
...
q=UTAD



Pesquisa — SAPO



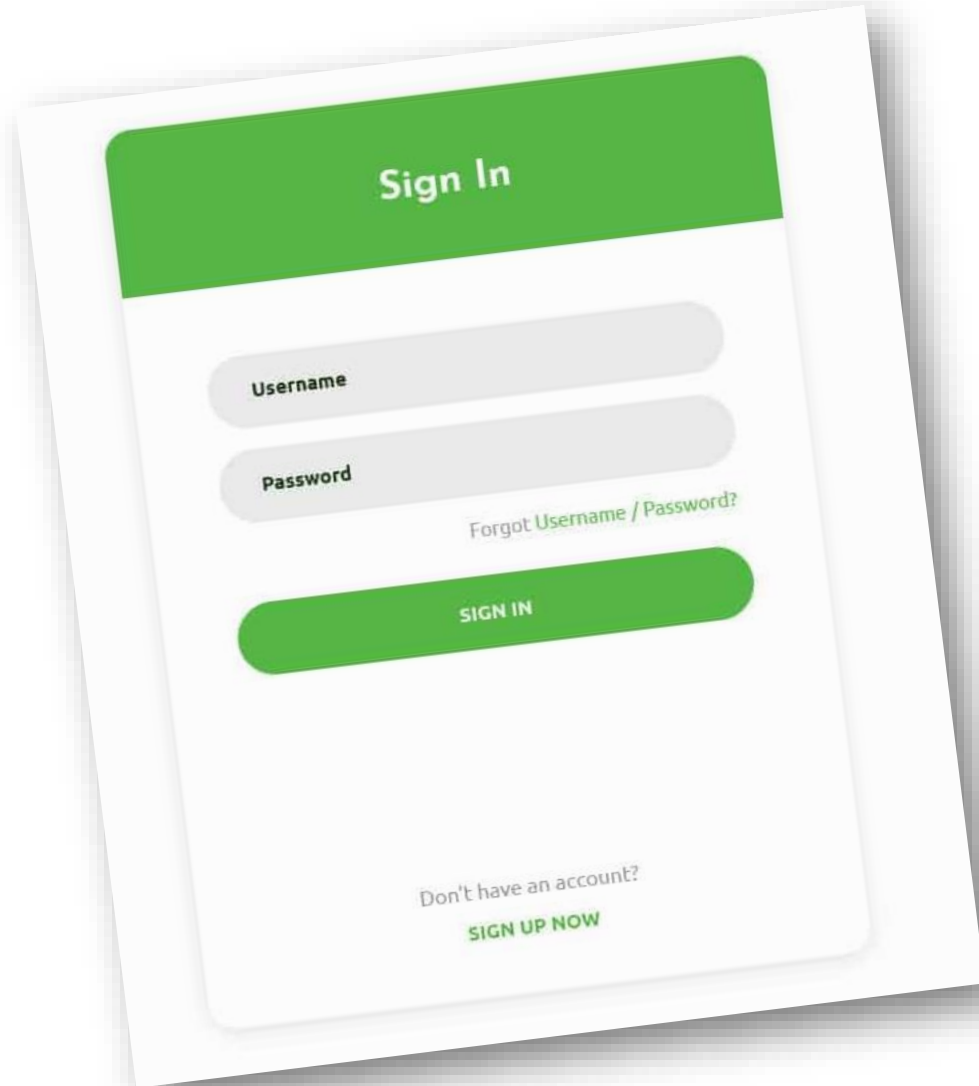
<https://www.sapo.pt/pesquisa#gsc.tab=0&gsc.page=1>

GET vs POST

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history

	GET	POST
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

GET or POST ???



A 'Sign In' form with a green header. It contains two input fields for 'Username' and 'Password', a link for 'Forgot Username / Password?', a green 'SIGN IN' button, and a 'Don't have an account?' link with a green 'SIGN UP NOW' button.

Sign In

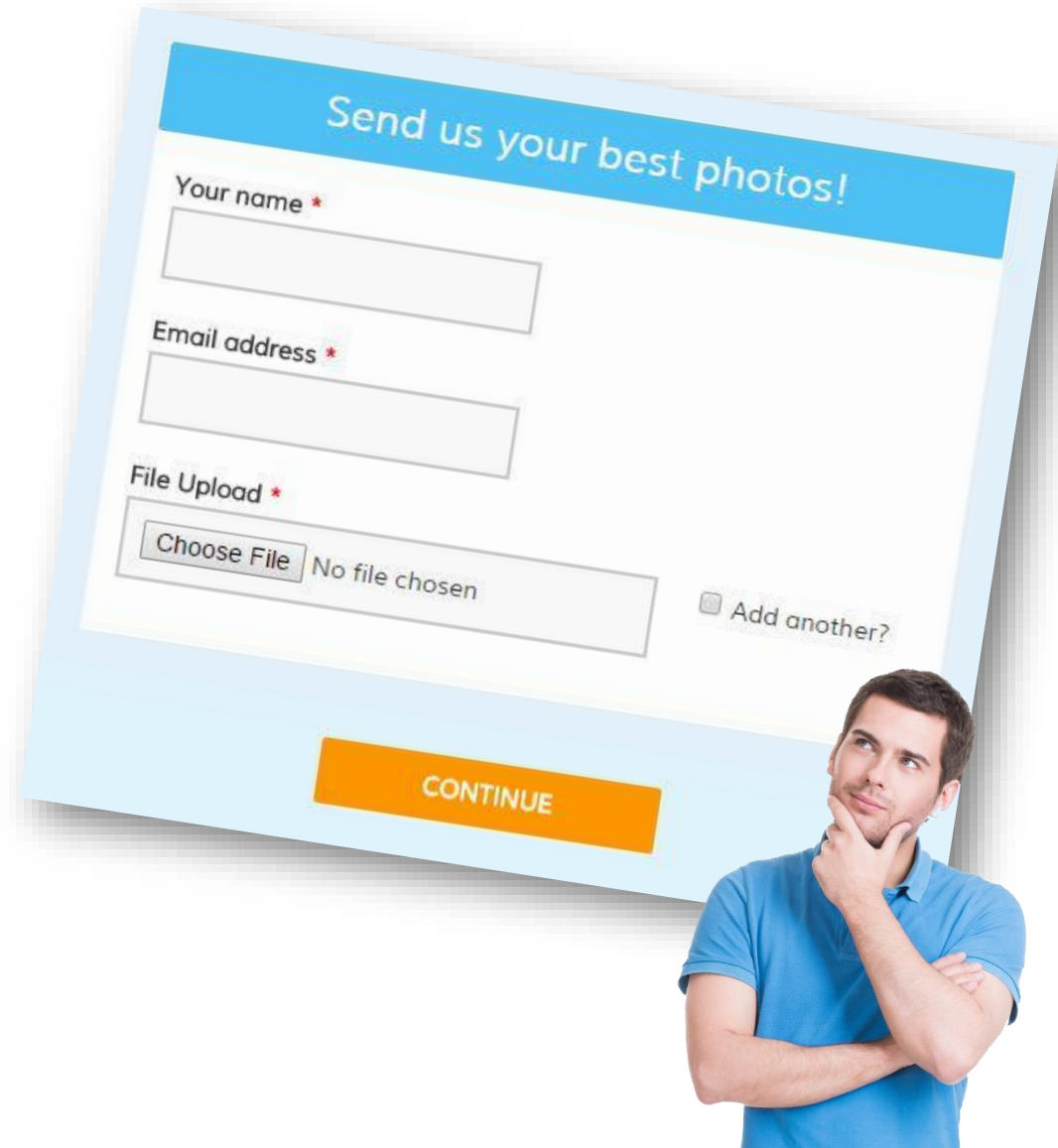
Username

Password

[Forgot Username / Password?](#)

SIGN IN

Don't have an account?
[SIGN UP NOW](#)



A photo upload form with a blue header. It contains two input fields for 'Your name' and 'Email address', a 'File Upload' section with a 'Choose File' button and 'No file chosen' text, a checkbox for 'Add another?', and an orange 'CONTINUE' button.

Send us your best photos!

Your name *

Email address *

File Upload *

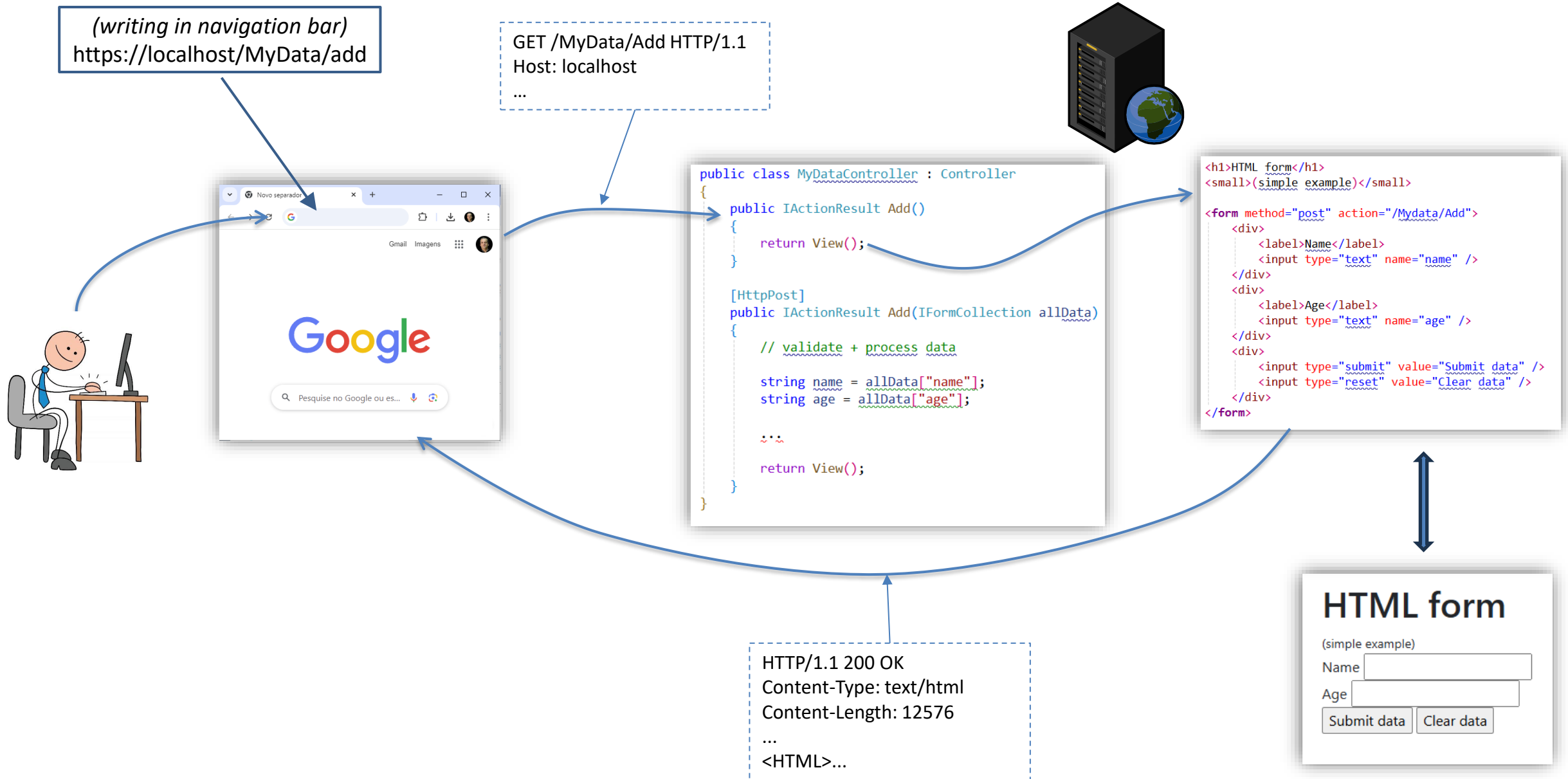
Choose File No file chosen

☐ Add another?

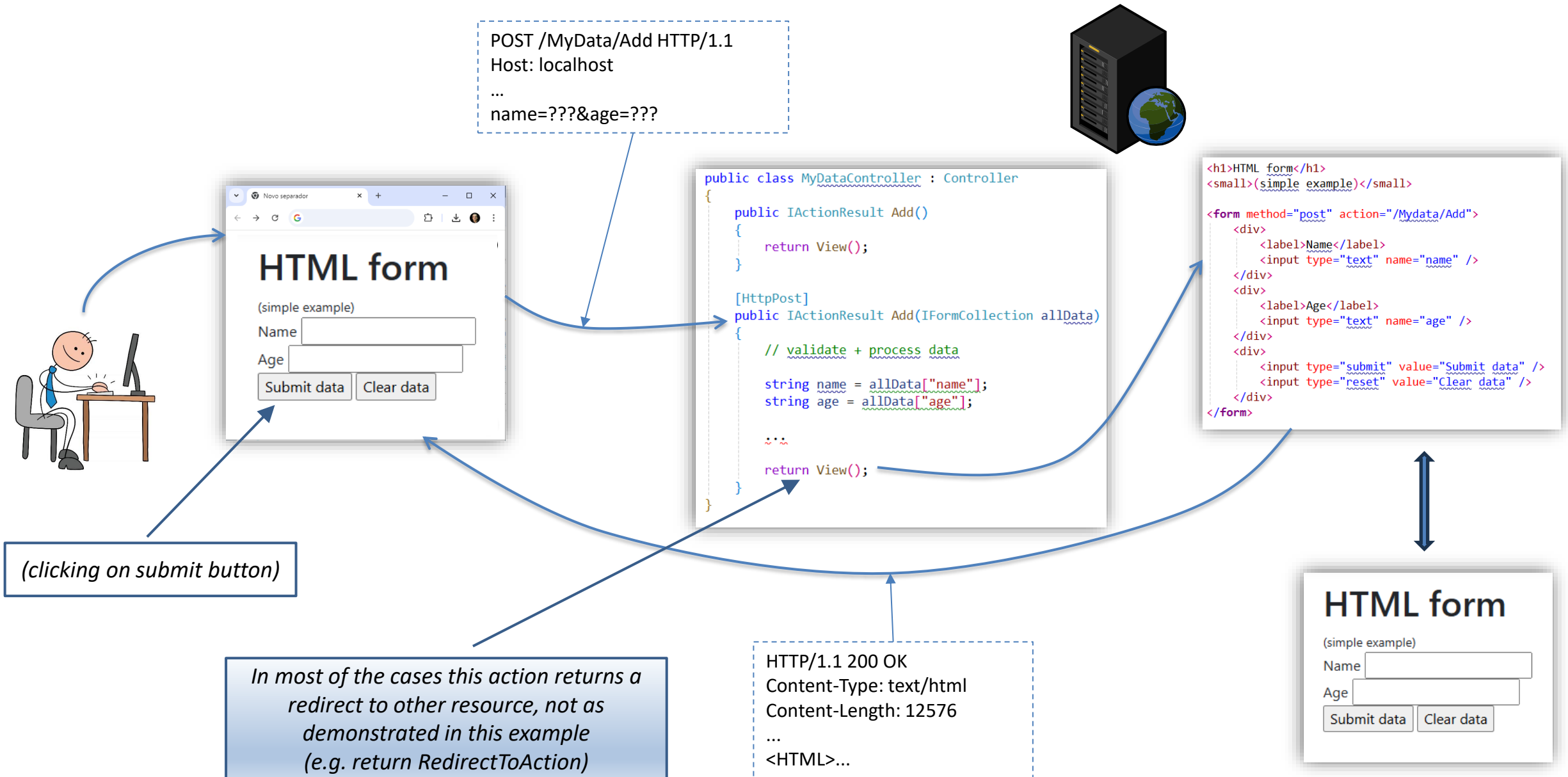
CONTINUE



Web forms in ASP.NET Core MVC apps (1/2)

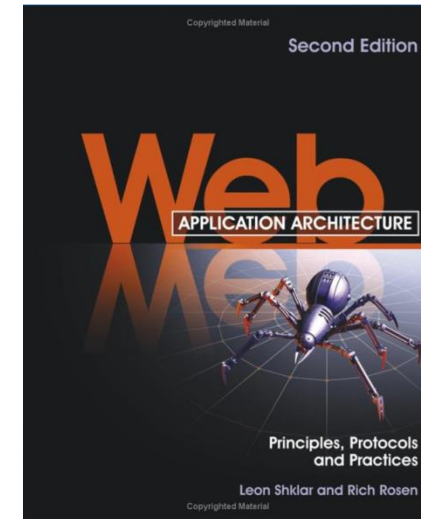


Web forms in ASP.NET Core MVC apps (2/2)



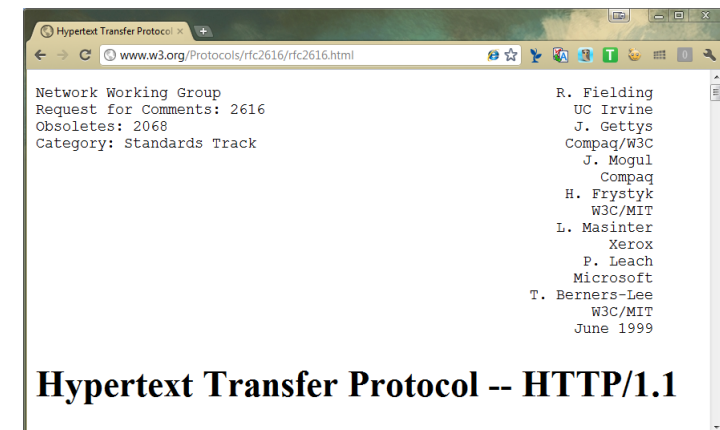
Associated Readings

Web Application Architecture, Second Edition
Cap. 3: “Birth of the Web: HTTP”, pages 29 to 34.



Hypertext Transfer Protocol -- HTTP/1.1

<http://www.w3.org/Protocols/rfc2616/rfc2616.html>



Resume

WEB SITE / APPLICATION NAVIGATION

HOW DOES WEBSITE NAVIGATION WORK?

WHAT IS A URL AND WHAT ARE ITS COMPONENTS?

WHAT IS A QUERY STRING FOR?

WHY DOES A BROWSER AUTOMATICALLY MAKE SEVERAL REQUESTS TO RENDER A WEB PAGE?

WHAT TYPES OF CONTENT CAN A WEB PAGE CONTAIN?

ARE THERE DIFFERENCES FOR THE BROWSER IF THE WEB APPLICATION HAS STATIC CONTENT OR DYNAMICALLY GENERATED CONTENT?

HTTP TERMINOLOGY

WHAT DOES IT MEAN FOR THE HTTP PROTOCOL TO BE STATELESS?

WHAT DOES AN HTTP MESSAGE CONSIST OF? WHAT IS ITS STRUCTURE?

WHAT ARE THE DIFFERENCES BETWEEN REQUEST AND RESPONSE MESSAGES?

WHAT HTTP METHODS ARE USED WHEN BROWSING WEB APPLICATIONS OR WEB SITES?

WHAT ARE THE DIFFERENCES BETWEEN THE GET METHOD AND THE POST METHOD?

Next section

HTTP ELEMENTS

(STATUS CODES, HEADERS, AND MIME TYPES)

Readings until September 30's class

Cap. 3: “Birth of the World Wide Web: HTTP”, pp. 44 to 60.

3.4	Better Information Through Headers
3.4.1	Support for content types
3.4.2	Caching control
3.4.3	Security
3.4.4	Session support
3.5	Evolution of the HTTP Protocol
3.5.1	Virtual hosting
3.5.2	Caching support
3.5.3	Persistent connections

