HTML basics
HTTP

SCOMRED, October 2017

URL (*Uniform Resource Locator*)

O URL é uma notação que permite identificar um recurso disponível em ambiente de rede:

PROTOCOL://SERVER[:PORT]/[FOLDER][NAME]

PROTOCOL identifica o protocolo de aplicação usado para aceder ao recurso, pode ser por exemplo **ftp**, **http** ou **https**. O valor **file** permite identificar um recurso (ficheiro) local.

SERVER identifica o nó de rede que disponibiliza o recurso, normalmente é um nome DNS, mas pode ser um endereço IP. O atributo **:PORT** pode ser usado caso o servidor não esteja a usar o número de porto standard para protocolo identificado.

[FOLDER][NAME] identifica o recurso através da sua localização dentro do servidor (pasta) e nome do recurso.

A omissão de atributos no URL é tratada pelas aplicações (clientes e servidores) segundo critérios próprios. Por exemplo num browser (cliente) quando **PROTOCOL** é omitido normalmente assume-se **http**.

Num servidor se **NAME** é omitido normalmente assume **índex.html** como nome do recurso, mas podem ser definidos outros.

Hipertexto – HTML (*Hypertext Mark-up Language*)

Documentos em hipertexto são documentos que possuem ligações a outros documentos disponíveis através da rede. Essas referências externas (*hyperlinks*) podem servir para o utilizador navegar entre diversos documentos ou podem fazer parte da apresentação do próprio documento como no caso de imagens conteúdos de vídeo/som.

Cada referência externa é representada sob a forma de um **URL**, no contexto de um documento (conteúdo), um URL pode ser especificado de forma relativa à origem do documento (URL de onde o documento foi carregado).

Por exemplo se for carregado um conteúdo (documento) do URL **http://server1.ipp.pt/users/teste.html**, então nesse conteúdo, os seguintes URL são equivalentes:

adduser.html ⇔ http://server1.ipp.pt/users/adduser.html

/deluser.html ⇔ http://server1.ipp.pt/deluser.html

/images/user.gif ⇔ http://server1.ipp.pt/images/user.gif

Ou seja, para os atributos omitidos são assumidos os valores correspondentes à origem do presente conteúdo. **Vantagens?**

TAGs HTML

Os conteúdos em formato HTML estão em formato de texto, mas utilizam uma sintaxe própria baseada em TAGS que permitem apresentar diversos tipos de formatação.

A totalidade do conteúdo HTML deve encontrar-se dentro de um TAG correspondente, ou seja deve começar por <html> e terminar em </html>. O documento deve conter uma secção de cabeçalho <head>…</head> e um corpo com o conteúdo a ser apresentado <body>…</body>.

Alguns TAGs básicos:

<h1>…</h1> - Define um cabeçalho (título de uma secção)

… - Define um parágrafo.

… - Define um conteúdo que deve ser centrado.

… - Define uma ligação a outro conteúdo.

<img src=**URL**> - Define uma imagem a ser incluída no documento.

… - Permite apresentar um conteúdo de texto usando uma fonte de carateres alternativa e com cor diferente.

TAGs HTML – atividade prática de teste

- (Windows) Crie uma pasta C:\scomred
- Use o **notepad** para criar o ficheiro C:\scomred\index.html
- Coloque o seguinte conteúdo no ficheiro:

```
<html><head><title>Teste de SCOMRED</title></head>
<body bgcolor=#00FFFF>
<h1>SCOMRED (testing)</h1>
<img src=https://moodle.isep.ipp.pt/acesso/logo.png></img>
<p>Isto é um teste HTML de SCOMRED</p>
<font size=-1 color=#FF0000>2017-2018</font>
<center><a href=http://moodle.isep.ipp.pt>GOTO MOODLE</a></center>
<hr><img src=https://moodle.isep.ipp.pt/acesso/MoodleLogoCertificacao.jpg></img>
</body><html>
```

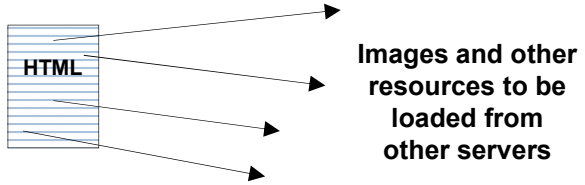
Num browser abra o URL **file:///C:/scomred/index.html**

O HTML permite de uma forma muito simples apresentar ao utilizador uma GUI (*Graphical User Interface*) num browser.

HTTP (*Hypertext Transfer Protocol*)

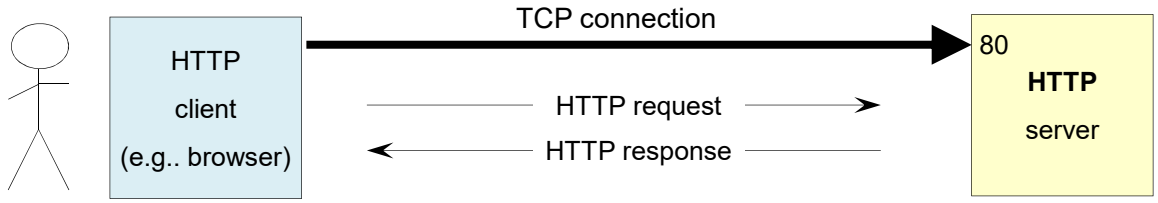
A utilização de conteúdos em hipertexto criou novos desafios para os quais os protocolos tradicionais de transferência de ficheiros como o FTP (*File Transfer Protocol*) não estão adaptados.

Esse protocolos foram concebidos para transferir vários conteúdos extensos todos do mesmo servidor.



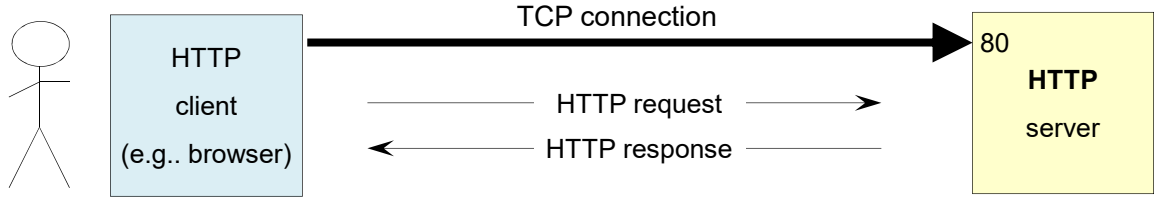
Para carregar e apresentar um documento em HTML é necessário aceder de forma expedita a vários servidores, além disso seria desejável que o servidor pudesse dar mais informações sobre o tipo de conteúdo que está a disponibilizar, destas necessidades surgiu o HTTP.

O HTTP utiliza o modelo cliente-servidor através de uma conexão TCP:



HTTP - *Methods*

O HTTP opera segundo o modelo cliente-servidor em TCP, a iniciativa cabe ao cliente que estabelece um ligação TCP com o número de porto 80 do servidor (número de porto standard para o serviço HTTP).



Após estabelecer a ligação TCP, cabe ao cliente enviar um pedido (***HTTP request message***) e de seguida deve aguardar uma resposta do servidor (***HTTP response message***).

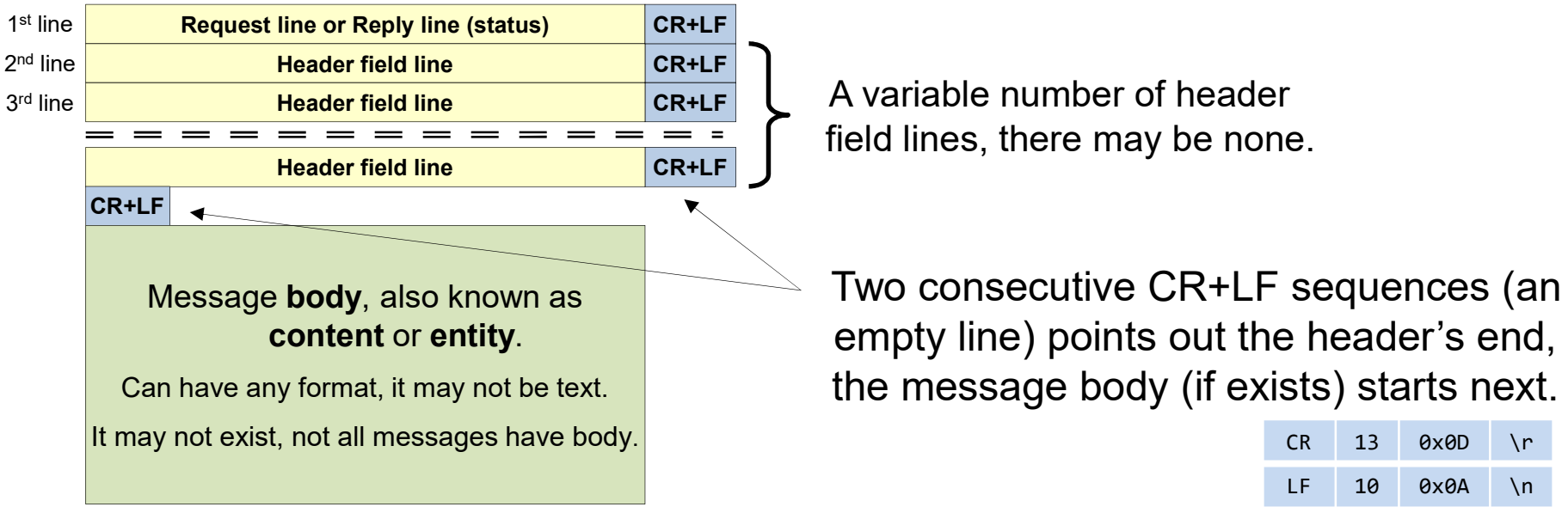
O HTTP define vários tipos de pedidos, designados métodos (methods).

No HTTP 1.0 existem os métodos GET, POST e HEAD. O HTTP 1.1 adiciona a estes os métodos OPTIONS, PUT, DELETE, TRACE e CONNECT.

HTTP – Formato das mensagens

Tanto os pedidos como as respostas utilizam o mesmo formato geral de mensagem, começam sempre por uma sequência de linhas de texto, cada uma terminada com o byte CR (Carriage Return) seguido do byte LF (Line Feed). Cada linha tem comprimento variável, a esta sequência de linhas chama-se o cabeçalho da mensagem HTTP, o cabeçalho termina formalmente com uma linha vazia.

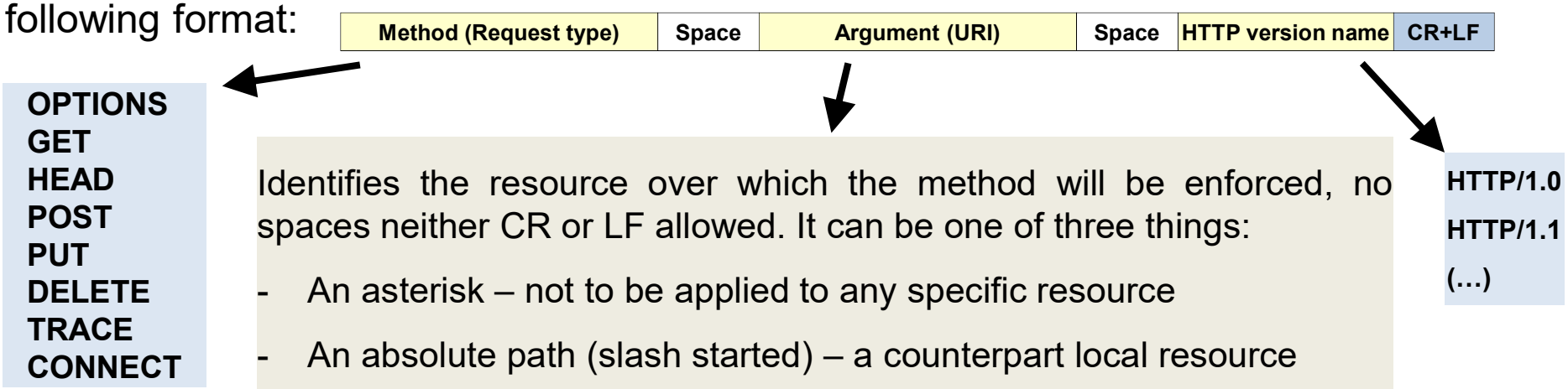
A primeira linha do cabeçalho é especial, é a linha de pedido ou linha de resposta conforme o caso, as restantes linhas do cabeçalho são opcionais:



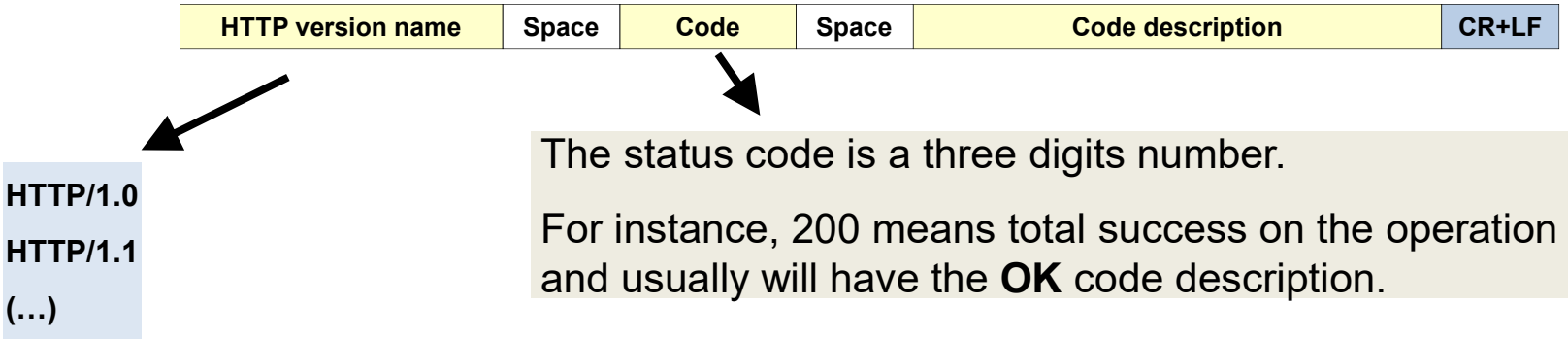
CR	13	0x0D	\r
LF	10	0x0A	\n

HTTP – Request and response messages

The first line in an **HTTP request message** is called the **request line** and has the following format:



The first line in an **HTTP response message** is called the **status line** and has the following format:



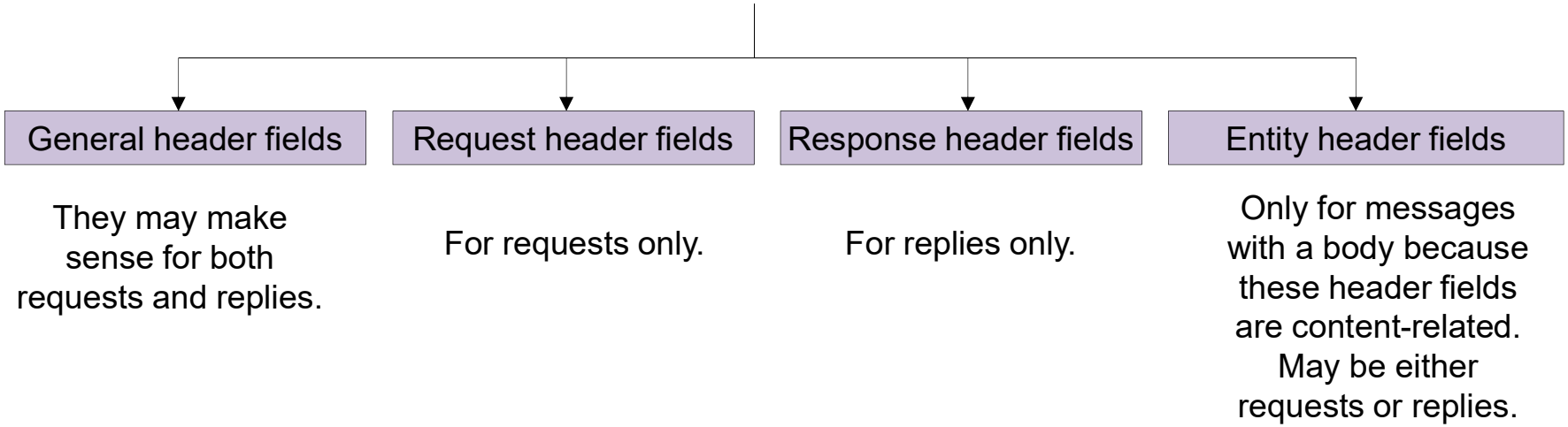
Linhas de cabeçalho (header fields)

As linhas de cabeçalho são usadas para transferir informação de controlo do HTTP e informações sobre o conteúdo, o seu formato geral é:

Field name	:	Field value	CR+LF

Onde ***Field name*** é um nome de atributo não case-sensitive e deve seguir-se sem espaços o sinal de dois pontos, o **Field value** é o valor do atributo e pode ser precedido de espaços.

Tanto pedidos como respostas podem transportar linhas de cabeçalho, mas nem todos os atributos são utilizáveis em todos os tipos de mensagens, assim tradicionalmente as linhas de cabeçalho agrupam-se em:



HTTP/1.1 general-header fields

Podem ser usados em pedidos ou respostas e não se referem ao conteúdo transportado, algumas mais comuns são:

Cache-Control	Settles how data may be cached by clients, servers and proxies, some possible values are **no-cache**, **no-store**, **max-age**, **public** and **private**. Not available for HTTP/1.0, Pragma must be used instead.
Connection	Unlike in HTTP/1.0, in HTTP/1.1 the default behavior is keeping connections open to allowing multiple requests and replies. The header field **Connection: close** informs the counterpart the connection is going to be closed after the current transaction.
Date	Hold the date/time of the message creation. For instance: **Date: Tue, 15 Nov 1994 08:12:31 GMT**
Pragma	Settles operational directives, most used value is **no-cache** to indicate no data caching is allowed.
Upgrade	Clients may include this on requests to inform the server about new protocol versions they support, the server may then inform the client it wants to switch to on of them by sending a **101 Switching Protocols** response with the Upgrade header field indicating to which is switching to. For instance: **Upgrade: HTTP/2**
Transfer-Encoding	Informs the counterpart about a transformation applied to the message body. This is similar to the entity header field **Content-Encoding**.

HTTP/1.1 entity-header fields

Referem-se ao conteúdo transportado, por isso só fazem sentido para mensagens que contêm corpo (body), exemplos:

Allow	Informs about supported methods to access a resource. It will be included in a **405 Method Not Allowed** response.
Content-Encoding	Informs the message receiver about some coding was applied to the content, for instance: **Content-Encoding: gzip**
Content-Language	Describes the natural language of the intended audience for the content.
Content-Length	Defines the size in octets of the content. This is supposed to be used by message receivers to know how many bytes they should read from the body starting point.
Content-MD5	Holds the result of applying the Message Digest 5 algorithm to the content, used for integrity checking.
Content-Range	This is used for a partial content message body. It must specify the body position within the original content and the total original content length. Example: **Content-Range: bytes 21010-47021/47022**
Content-Type	Informs the receiver about the content media, thus, how the content should be interpreted and ultimately displayed.
Expires	Contains a date/time after which cached copies of the content are no longer valid.
Last-Modified	Contains the date/time the content was last modified. If the content comes from a file may be the file last modification time.

HTTP/1.1 request-header fields

Aplicam-se apenas a pedidos HTTP (feitos por clientes), exemplos:

From	Contains the e-mail address of the human user controlling the client application.
Host	Contains the hostname and port number being accessed, either typed by the user at the browser or from the clicked URL in a document. Default port number is 80. Example: **Host: www.dei.isep.ipp.pt.pt:8080**
Referer	Contains the URL of the document from where the present request was followed (referred by). This field should not exist for directly user typed requests. Example: **Referer: http://www.dei.isep.ipp.pt/index.html**
User-Agent	A string identification for the client application, usually a browser. Example: **User-Agent: Mozilla/5.0 (Linux; Android 4.0.4; Galaxy Nexus Build/IMM76B)**
Authorization	User authentication data, usually username/password. Must be included following an **401 Unauthorized** response.
Cookie	Contains a pair name and value provided by the counterpart in a previous response. Example: **Cookie: sessionToken=ts12325**

HTTP/1.1 request-header fields – conditional requests

Permite aos clientes que fazem os pedidos introduzir exigências relativamente ao conteúdo que pretendem receber. Exemplos:

Accept	Requires the response content to be in one of the specified media types (content types). Example: **Accept: text/*, text/html, text/html;level=1, */***
Accept-Charset	Requires the response content to be in one of the specified charsets. Example: **Accept-Charset: iso-8859-5, unicode-1-1;q=0.8**
Accept-Encoding	Restricts possible content-coding values for the response content. Example: **Accept-Encoding: compress, gzip**
Accept-Language	Restricts possible content languages to a given set. Example: **Accept-Language: da, en-gb;q=0.8, en;q=0.7**
If-Modified-Since **If-Unmodified-Since**	Causes the response to depend on the requested resource last modification time/data.
If-Match **If-None-Match**	Causes the response to depend on the value for the ETag entity-header field. Messages with a body may define a **ETag** entity-header for the content (body) they carry.

HTTP/1.1 response-header fields

Aplicam-se apenas a respostas HTTP (efetuadas por servidores). Exemplos:

Age	For cached replies, this is estimated elapsed time in seconds since the original response was obtained.
Location	This is used to redirect the requester to a different document from the one requested. Contains a document URL. This will be used for **3xx** responses, for instance **307 Temporary Redirect**.
Public	Inform the client about server supported methods in general, not specifically on the requested URI. Example: **Public: OPTIONS, MGET, MHEAD, GET, HEAD**
Retry-After	Used in **503 Service Unavailable** response to inform about when the service is expected to be available, may be a date/time or a time period is seconds.
Server	A string identification for the server application. Example: **Server: CERN/3.0 libwww/2.17**
WWW-Authenticate	Used with **401 Unauthorized** response informing access to the resource requires user authentication. This field informs about the expected authentication mechanism to be used.
Set-Cookie	Contains a pair name and value for the client to use in the **Cookie** header field on subsequent requests. The purpose is the server being able to identify this particular client in next requests, and thus, maintain with it a stateful session. Example: **Set-Cookie: sessionToken=ts12325**

HTTP/1.1 – pedidos (methods) OPTIONS e GET

OPTIONS	Space	Argument (URI)	Space	HTTP/1.1	CR+LF

The response to this request provides the client with a list of available methods to access the URI, if the URI is an asterisk, then a list of methods supported by the server is provided.
The response will be usually **200 OK** and the response-header field **Allow** will l have a list of supported methods and eventually other header fields defining the server capabilities.

GET	Space	Argument (URI)	Space	HTTP/1.1	CR+LF

Used to obtain (download) the content pointed by URI. Typically, URI refers to a static content stored in a file, however, that may not be the case, it may also be dynamically generated by the server.
The technique known as CGI (Common Gateway Interface), allows the server to execute external programs or scripts and return their output as response message content. In these cases, URI refers to something executable and not a static file.
In most cases, CGI applications require input data to be provided by the client (usually collected in an HTML form). However, GET method requests can´t have a body, this is somewhat overcome by embedding form data in the URI, appending the query string.
The query-string starts by a question mark and it's made of an ampersand separated list of pairs form field name and form field value. For instance:

http://www.server1.net/login?username=teste&password=pppttee&dep=5

A URI is obviously not the best-suited local to place forms data, only plain text data is supported and there are length issues. Beyond that, data will be visible in the URL. The POST method request is more suitable because it can have a body to carry data.

HTTP/1.1 – pedidos (methods) HEAD, POST, PUT e DELETE

HEAD	Space	Argument (URI)	Space	HTTP/1.1	CR+LF

The response to the HEAD request is exactly the same that would be achieved with a GET request for the URI, except that it will have no body, nevertheless, all header fields will be the same.

POST	Space	Argument (URI)	Space	HTTP/1.1	CR+LF

The POST request purpose is sending data to an URI, this will normally be some kind of executable application. Unlike with the GET method, data is placed on the message body, therefore, there are no restrictions whatsoever on data content and length.

PUT	Space	Argument (URI)	Space	HTTP/1.1	CR+LF

The PUT request can be interpreted as the reverse of GET method. It allows the upload of a content to a URI. It is mostly intended to upload a content to a file named by the URI, however, it may also be used with the same purpose of POST if URI refers to an application.

DELETE	Space	Argument (URI)	Space	HTTP/1.1	CR+LF

Used to request the removal of a resource on the counterpart. The URI is supposed to represent the name of the file to be removed.

HTTP/1.1 – Responses status-codes (1xx, 2xx, and 3xx)

Os códigos de resposta HTTP podem agrupar-se em cinco categorias de acordo com o dígito da esquerda:

HTTP/1.1	Space	1XX	Space	Textual code description	CR+LF

Codes 1XX didn't exist in HTTP/1.0, they indicate some additional messages are expected over the same connection. For instance **100 Continue** indicates the server has accepted the request first part and is expecting something else. The **101 Switching Protocols** is used when the server wants to upgrade to a higher HTTP version (**Upgrade** general-header field).

HTTP/1.1	Space	2XX	Space	Textual code description	CR+LF

Notify about a success on the requested operation. Examples:

200 OK – indicates total success on a GET, HEAD or POST.
201 Created – as result of the request a new resource has been created.
202 Accepted – the request was accepted, but may not have been yet executed, there may be a delay.
206 Partial Content – the content on the response body is only partial.

HTTP/1.1	Space	3XX	Space	Textual code description	CR+LF

Alert about a failure and the need for the client to reformulated the request. Examples:

300 Multiple Choices – there are several option to execute the request. A list is provided.
301 Moved Permanently – the resource was dislocated, the new location is provided by the **Location** field.
303 Moved Temporarily – temporary dislocation, the new location is provided by the **Location** field.
304 Not Modified – response to a conditional GET request when conditions are not meet.

HTTP/1.1 – Responses status-codes (4xx and 5xx)

HTTP/1.1	Space	4XX	Space	Textual code description	CR+LF

Codes 4XX alert about what the server thinks it's a client side request error. Examples:

400 Bad Request – the server simply did not understand the request made.
401 Unauthorized – the server demands user authentication for the request made.
403 Forbidden – the resource exists but is not accessible due to the lack of permission.
404 Not Found – the requested resource was not found in the server.
405 Method Not Allowed – the used method is not possible for the requested resource.
406 Not Acceptable – an Accept field restriction on the request could not be satisfied by the server.
411 Length Required – the server refuses to accept the request with no Content-Length specified.
412 Precondition Failed – an If field precondition on the request could not be satisfied by the server.

HTTP/1.1	Space	5XX	Space	Textual code description	CR+LF

These codes are about server side issues, they mean the server is aware there is a problem and was unable to fulfill the request. Examples:

500 Internal Server Error – the server has a severe problem and was unable to process the request.
501 Not Implemented – the request method is not supported by the server.
503 Service Unavailable – the server was unable to process the request due to a temporary overload.
505 HTTP Version Not Supported – the request HTTP version is not supported by the server.

HTTPS (*Hyper Text Transfer Protocol Secure*)

HTTP e HTTPS são o mesmo protocolo, a diferença é que enquanto o HTTP funciona diretamente sobre uma ligação TCP, o HTTPS funciona sobre uma ligação TCP com TLS (*Transport Layer Security*).

O TLS começa por comprovar a autenticidade dos nós e depois garante a privacidade dos dados transferidos.

A comprovação da autenticidade do servidor é fundamental para o cliente, caso contrário poderá acabar a dialogar com um servidor falso.

No contexto do TLS, a autenticidade do servidor fica comprovada quando o cliente recebe do servidor um certificado de chave pública válido e fidedigno contendo o nome DNS do servidor.

Embora durante o desenvolvimento de aplicações se possa utilizar o protocolo HTTP, num ambiente de produção deve ser sempre usado o HTTPS.

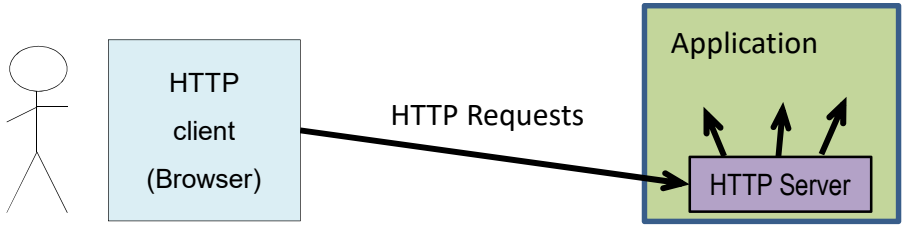
O número de porto padrão do servidor em HTTP é o 80, no caso do HTTPS é o 443.

HTTP/HTTPS + HTML como GUI (WebUI)

Implementar uma GUI (*Graphical User Interface*) para uma aplicação é sempre um esforço significativo que exige uma familiarização com a API usada para o efeito, diferentes linguagens disponibilizam diferentes APIs para esse efeito. Adicionalmente leva a um consumo significativo de recursos pela aplicação.

Um opção é incluir na aplicação um pequeno servidor HTTP e desta forma disponibilizar uma GUI sem grande esforço, usando por exemplo HTML.

O utilizador usa um browser perfeitamente standard e acede via HTTP ao servidor embebido na aplicação.



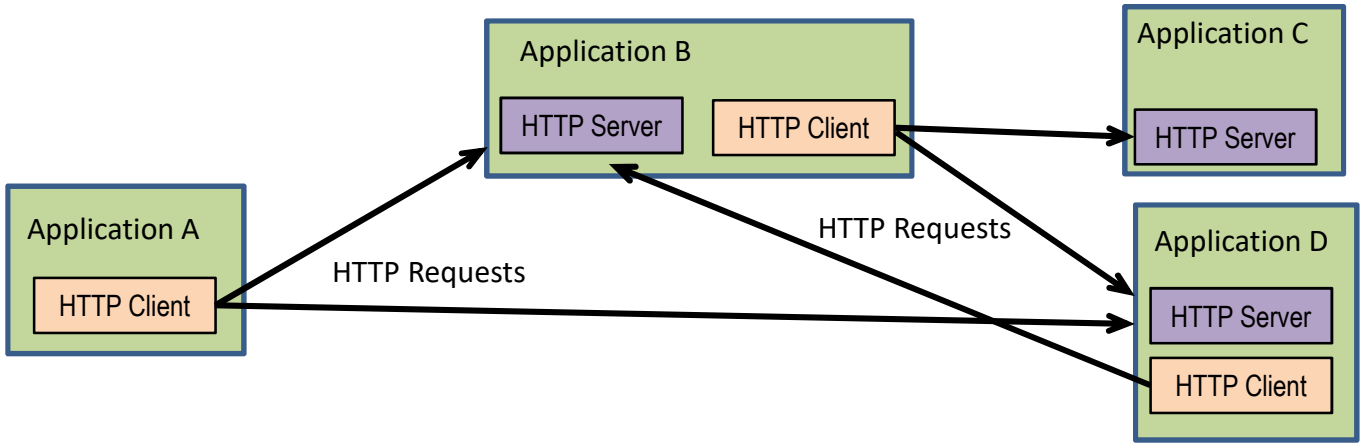
O servidor interage via HTTP com o a interface do utilizador (browser) e funciona como intermediário no acesso às funcionalidades da aplicação.

Devido ao facto de o HTTP usar o modelo cliente-servidor, a implementação desta estratégia é mais simples quando todas as atividades da aplicação são desencadeadas pelo utilizador.

HTTP/HTTPS como protocolo genérico

Desenvolver um protocolo de aplicação para asseguram as comunicações entre um conjunto de aplicações em ambiente de rede é um esforço significativo.

Deve ser sempre ponderada a possibilidade de ser usado um protocolo já existente, o HTTP/HTTPS assume-se como um forte candidato para esse propósito e é largamente utilizado nesse contexto.

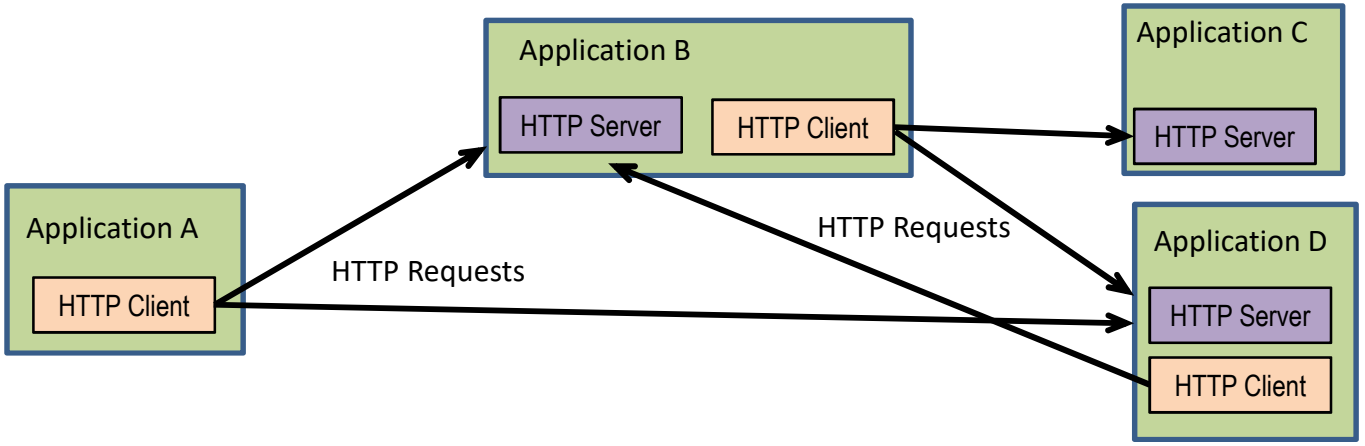


As várias aplicações usam o HTTP/HTTPS para comunicarem entre si de acordo com as necessidades próprias. Não se trata de clientes e servidores HTTP tradicionais, apenas implementam as funcionalidades de que as aplicações necessitam, mas ainda assim têm de obedecer às especificações do protocolo HTTP.

Web Services

O conceito de *Web Services* baseia-se comunicações diretas via HTTP entre aplicações sem intervenção de utilizadores.

Na implementação de um *Web Service*, uma aplicação assume o papel de cliente HTTP designando-se o ***service requestor*** ou ***consumer***, a outra aplicação assume o papel de servidor HTTP designando-se o ***service provider*** ou ***publisher***.



Nada impede que uma mesma aplicação seja ***consumer***, de alguns *Web Services* e ***publisher*** de outros *Web Services*.

Implementação do protocolo HTTP

Desenvolver um cliente HTTP ou um servidor HTTP que implemente todas as funcionalidades do protocolo é uma tarefa de grande dimensão. A maioria das linguagens de programação possui bibliotecas (*libraries*), *packages* ou *frameworks* que implementam o HTTP e podem ser usadas no desenvolvimento das aplicações.

A utilização de bibliotecas que permitem a utilização de HTTP, quer sob o ponto de vista de cliente como de servidor, obrigam sempre a um estudo aprofundado da respetiva API.

No contexto do desenvolvimento de uma aplicação não é muito complicado implementar um servidor HTTP com um conjunto restrito de funcionalidades do HTTP de acordo com as necessidades específicas de aplicação.

No projeto Java seguinte vamos desenvolver uma aplicação simples de cálculo de múltiplos num intervalo, será dotada de uma interface Web (WebUI). Para esse efeito a aplicação terá de incluir um pequeno servidor HTTP capaz de responder a um conjunto limitado de pedidos previstos, neste caso terá claramente objetivos pedagógicos.

WebUI – Exercício prático

Desenvolver uma aplicação que, dados dois números inteiros positivos correspondentes a um intervalo fechado, calcule e apresente ao utilizador, em listas separadas, os múltiplos de 2, 3, 5 e 7 pertencentes ao intervalo.

A interação com o utilizador será efetuada através de um browser (WebUI), para o efeito a aplicação deverá incluir um pequeno servidor HTTP disponível no número de porto 8080. O servidor HTTP apenas necessita de suportar o método GET.

A aplicação deve validar os dados fornecidos, verificando se são números inteiros positivos, caso contrário deve apresentar uma mensagem de erro elucidativa.

Se os limites do intervalo estiverem invertidos a aplicação deve colocar os mesmos nas posições corretas de forma automática.

Formulários HTML

HTML permite a definição de formulários (form) cujos dados preenchidos pelo utilizador podem ser enviados pelo browser ao servidor.

O TAG <form>…</form> inclui vários atributos relevantes:

<form action=URL method=MÉTODO>…</form>

URL – define qual o URL onde vão ser submetidos os dados do formulário.

MÉTODO – define o tipo de pedido através do qual os dados vão ser submetidos, pode ser POST ou GET (default).

Se o método é POST, os dados serão transportados no corpo da mensagem HTTP, se o método é GET os dados serão adicionados a URL na forma de um ***query-string***.

A utilização do GET para este efeito limita fortemente os tipos de dados que podem constar do formulário, terão de ser texto e não muito extensos.

Elementos de input num form HTML

Dentro do TAG form podem ser definidos elementos de input do formulário como por exemplo campos de texto a serem preenchidos pelo utilizador:

<input type=text name=FIELD-NAME value=DEFAULT-VALUE>

Onde **FIELD-NAME** identifica o campo e será incluído no ***query-string*** e **DEFAULT-VALUE** é o valor inicial para o campo que pode ser alterado pelo utilizador

A submissão do formulário é desencadeada por um input do tipo submit que vai corresponder a um botão:

<input type=submit value=TEXTO>

Onde **TEXTO** será o texto apresentado no botão de submissão do formulário.

Existem vários outros tipos de input disponíveis nos formulários HTML, tais como buttons, checkboxes, radio buttons, drop-down lists e textareas.

Elementos de input num form HTML

Dentro do TAG form podem ser definidos elementos de input do formulário como por exemplo campos de texto a serem preenchidos pelo utilizador:

<input type=text name=FIELD-NAME value=DEFAULT-VALUE>

Onde **FIELD-NAME** identifica o campo e será incluído no ***query-string*** e **DEFAULT-VALUE** é o valor inicial para o campo que pode ser alterado pelo utilizador

A submissão do formulário é desencadeada por um input do tipo submit que vai corresponder a um botão:

<input type=submit value=TEXTO>

Onde **TEXTO** será o texto apresentado no botão de submissão do formulário.

Existem vários outros tipos de input disponíveis nos formulários HTML, tais como buttons, checkboxes, radio buttons, drop-down lists e textareas.

Formulários HTML – atividade prática de teste

- Use o **notepad** para criar o ficheiro C:\scomred\formtest.html
- Coloque o seguinte conteúdo no ficheiro:

```
<html><head><title>FormTeste de SCOMRED</title></head>
<body bgcolor=#00FFFF>
<h1>SCOMRED (from testing)</h1>
<form action=/testform method=GET>
<p>Valor A <input type="text" name="valora" value="0">
<p>Valor B <input type="text" name="valorb" value="100">
<p><input type="submit" value="Calcular">
</form></body><html>
```

Num browser abra o URL **file:///C:/scomred/formtest.html**

Pressione o botão Calcular, a operação vai falhar porque o URL que o browser vai tentar abrir não existe.

Verifique com atenção o URL que o browser tentou abrir, contém um ***query-string*** com os dados do formulário na forma:

/testform?valora=0&valorb=100