Universidade de Aveiro
Departamento de Engenharia Mecânica
Sistemas Flexíveis de Produção 2022/2023

# CAPÍTULO

# HTML

JPSantos
2022

## 1. HTML – HiperText Markup Language

https://www.w3.org/wiki/Html/Specifications

### 1.1. Conceitos sobre HTML

HTML significa "Hypertext Markup Language". Esta linguagem foi proposta por Tim-Berners e tinha por objectivo permitir escrever e formatar documentos com texto, tabelas e hiperligações para outros documentos, imagens, filmes e ficheiros em geral.
De uma forma simples e incompleta podemos dizer que esta linguagem define um conjunto de palavras chave, etiquetas, <tags> que um browser Web sabe interpretar e em função delas apresenta ao utilizador uma página WEB formatada.
Com base nestas etiquetas é possível escrever documentos de texto com uma estrutura e uma formatação conhecida, não proprietária, podendo por isso ser utilizados por todos.
As páginas WEB, escritas em HTML, são documentos de texto que têm de ser previamente gravados, por exemplo em disco, antes de poderem ser lidas localmente pelos browsers WEB.

### 1.1.1. Estrutura de um documento HTML

Um documento HTML está organizado em duas partes principais: cabeçalho "header" e o corpo principal "body". O documento HTML, como um todo, está contido pelas etiquetas <HTML> … </HTML>.

```
<HTML>
        <HEAD>
        ….
        </HEAD>
        <BODY>
        …..
        </BODY>
</HTML>
```

### 1.1.2. Etiquetas HTML

As etiquetas HTML estão definidas como um conjunto de palavras reservadas que aparecem no documento de texto precedidas pelo sinal menor e seguidas pelo sinal maior, envolvendo a palavra chave <etiqueta>. As palavras reservadas <HTML>, <HEAD> e <BODY> são alguns exemplos de etiquetas. A maior parte destas etiquetas são apresentadas na secção seguinte e são explicadas exemplo a exemplo.

## 1.2. Exemplos de páginas HTML

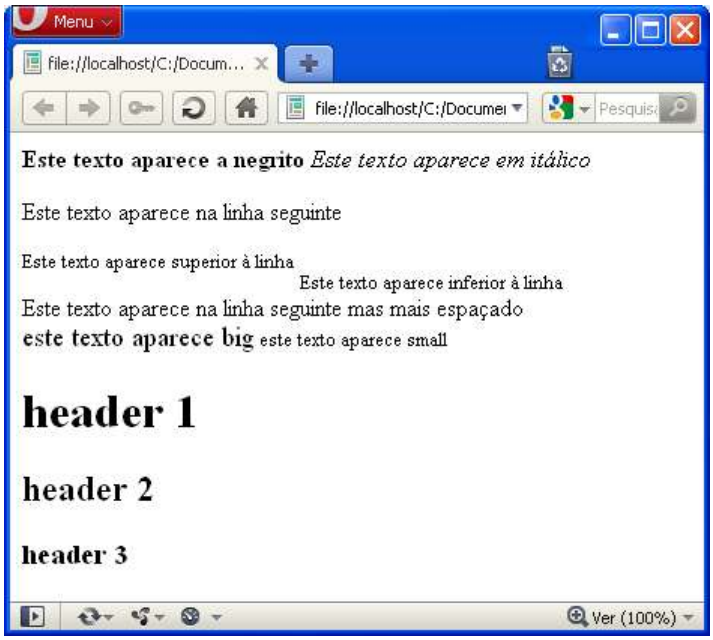
Guarde num ficheiro de texto diferente, cada um dos exemplos seguintes. O primeiro exemplo deve ficar gravado com o nome "exemplo1.html" e os exemplos seguintes deverão ter um nome à sua escolha. Depois de editar e guardar em disco cada um dos exemplos seguintes visualize-os com um browser WEB.
Em cada exemplo é apresentado o "texto" HTML e o aspecto da página WEB correspondente, tal como o Browser a interpreta e a visualiza. Estes exemplos são autoexplicativos.

### 1.2.1. Estrutura de um documento HTML

```
<!DOCTYPE html>
  <html>
        <head>
              <title>
              Um titulo para o documento
              </title>
        </head>
  <body>
        Este é o meu primeiro documento em HTML
  </body>
  </html>
```

### *1.2.2.* Formatar texto em HTML

```
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="refresh" content="1" />
    </head>
    <body>
        <b> Este texto aparece a negrito </b>
        <i> Este texto aparece em itálico </i>
        <sup> Este texto aparece superior à linha </sup>
        <sub> Este texto aparece inferior à linha </sub>
        <p> Este texto aparece na linha seguinte </p>
        <br> Este texto aparece na linha seguinte mas mais espaçado
        <br>
        <big>este texto aparece big</big>
        <small>este texto aparece small</small>
        <h1> header 1 </h1>
        <h2> header 2 </h2>
        <h3> header 3 </h3>
    </body>
</html>
```



Universidade de Aveiro
Departamento de Engenharia Mecânica
Sistemas Flexíveis de Produção 2022/2023

# CAPÍTULO

# HTTP

JPSantos
2022

## 2. HTTP – HiperText Transfer Protocol

O protocolo *HTTP* é a pedra angular da WEB. Este protocolo, juntamente com a linguagem *HTML*, foi proposto por Tim Berners-Lee enquanto investigava no *CERN*. De facto Tim Berners propôs um sistema de gestão de informação baseado em Hipertexto. Na altura o termo *World Wide Web -WWW* ainda não existia, mas o sistema proposto por Tim Berners já definia:

1. Uma linguagem para formatar documentos de texto (*HTML – HyperText Markup Language*). Esta linguagem e a sua sintaxe são apresentadas no capítulo seguinte.
2. Um protocolo para transferir esses documentos entre equipamentos (*HTTP – Hyper Text Transfer Prototocol*).
3. Um esquema de endereços para identificar e aceder a esses recursos na rede (*URL- Uniform resource locator*).

Convém relembrar que quando Tim Berner propôs o protocolo HTTP já existia a Internet, já existiam servidores e clientes *FTP – File Transfer Protocol* e por isso já era possível transferir documentos entre equipamentos, mas não era fácil localizá-los, nem existia uma forma fácil de um desses documentos fazer uma referência a outro documento (*Hiperligação*). Além disso a forma como cada documento estava formatado e guardado em disco variava de processador de texto para processador de texto.

## 2.1. Conceitos sobre HTTP

O protocolo *HTTP* define um conjunto de interações entre as aplicações clientes (*Browsers WEB*) e as aplicações servidoras (*Servidores WEB*) e define a sintaxe de um conjunto de mensagens que todos os clientes e servidores *HTTP* reconhecem e sabem processar.

### 2.1.1. Localização dos recursos na Internet

Tim Berners propôs uma forma de identificar e localizar um documento, um recurso, na Internet e chamou-lhe *URL – Uniform Resource Locator*, mais tarde passou a chamar-se *URI - Uniform Resource Identifier*.

De uma forma simples e incompleta, podemos dizer que o URL não é mais que um texto, formado pelo nome do computador de destino e pela localização do documento no disco duro do equipamento de destino.

O URL tem a estrutura seguinte:

Protocolo://computador:[tcpport number]/localização do documento dentro do computador de destino

[? Paramentos do pedido] [# ancora]

Alguns exemplos de URLs

http://www.ua.pt/index.html
ftp://...../.....

### 2.1.2. Interações HTTP

A interação entre programas *HTTP* é do tipo "*cliente – servidor*". Isto significa que um programa (o *browser web*) toma a iniciativa de enviar uma mensagem *HTTP* com um pedido, ao passo que o outro programa "limita-se" a processar e a responder a esse pedido com uma mensagem *HTTP* de resposta (*servidor web*).

#### 1. Pedido de ligação

No entanto, antes de ser possível fazer pedidos HTTP, ou seja, antes de enviar mensagens HTTP, a aplicação cliente tem de estabelecer uma ligação TCP com a aplicação remota.
Na figura seguinte, através de um browser Web, o utilizador pretende aceder a um documento remoto com o seguinte URL:
http://localhost/index.html

Através do URL, o browser sabe que deve usar o protocolo http e tem de estabelecer uma ligação TCP com o computador "localhost"

## 2. Envio de uma mensagem HTTP para o servidor e a mensagem de resposta

Depois da ligação TCP estar estabelecida, o browser pede ao servidor remoto o documento índex.html, enviando para isso uma mensagem HTTP do tipo GET:

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

Quando o *servidor WEB* consegue processar a *mensagem HTTP* descrita na figura, gera uma mensagem de resposta começada por "*HTTP/1.1 200 OK*", acede ao documento pedido, copia o seu conteúdo e acrescenta esse conteúdo à mensagem de resposta.

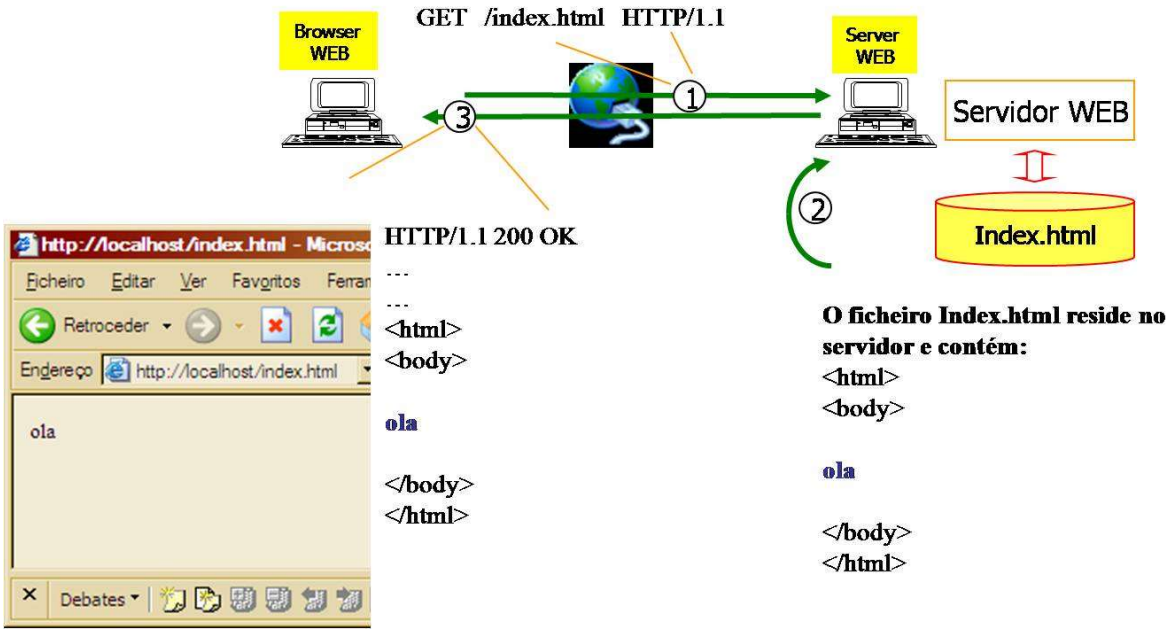



Figura 2.1: Troca de mensagens HTTP

## 3. Fim de ligação

Depois da troca de mensagens http tanto o browser como o servidor Web podem terminar a ligação TCP.

### 2.1.3. Mensagens HTTP

Depois da ligação TCP ter sido previamente estabelecida o browser Web pode enviar um de vários tipos de pedidos para o servidor, por exemplo: GET, HEAD e POST.

#### GET

As mensagens do tipo GET levam o servidor a responder com informações sobre o documento pedido, incluindo o seu conteúdo.

*Quando no Browser o utilizador insere o URL e os parâmetros:*

> *http://192.18.43.197/index.html?a=1&b=2*

*O Servidor recebe a mensagem:*

> *GET /index.html?a=1&b=2 HTTP/1.1*
> *Host: 192.168.43.197*
> *Connection: keep-alive*
> *Upgrade-Insecure-Requests: 1*
> *User-Agent: Mozilla/5.0 (Linux; Android 9; Redmi Note 8T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Mobile Safari/537.36*
> *Accept:*
> *text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9*
> *Accept-Encoding: gzip, deflate*
> *Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7*

#### HEAD

O browser pode pedir ao servidor para lhe enviar apenas informação sobre as características de um documento, por exemplo, pedir-lhe que confirme se esse documento existe no servidor, mas sem pedir o seu conteúdo.

#### POST

O browser pode enviar um pedido de atualização de um documento no servidor. Neste caso a mensagem enviada pelo browser é do tipo POST e contém também no final do cabeçalho os novos parâmetros/dados que o servidor irá utilizar para atualizar o documento ou executar algumas ações.

> *POST /path/script.cgi HTTP/1.0*
> *From: frog@jmarshall.com*
> *User-Agent: HTTPTool/1.0*
> *Content-Type: application/x-www-form-urlencoded*
> *Content-Length: 32*
> *home=Cosby&favorite+flavor=flies*

## Respostas do servidor

Consoante o tipo de pedido efetuado pelo browser e dependendo da forma como o servidor foi capaz de reconhecer e executar esse pedido, o servidor Web responde com um de vários códigos possíveis

HTTP/1.1 código de resposta
…..
…..

Código de resposta = "200" ; OK
"201" ; Created
"202" ; Accepted
"204" ; No Content
"301" ; Moved Permanently
"302" ; Moved Temporarily
"304" ; Not Modified
"400" ; Bad Request
"401" ; Unauthorized
"403" ; Forbidden
"404" ; Not Found
"500" ; Internal Server Error
"501" ; Not Implemented
"502" ; Bad Gateway
"503" ; Service Unavailable

https://pt.wikipedia.org/wiki/Lista_de_campos_de_cabe%C3%A7alho_HTTP

De facto, a mensagem HTTP de resposta é mais complexa. Além do documento HTML, começado por <html> e terminado por </html> , a mensagem HTTP começa pelo seu cabeçalho, por exemplo:

```
HTTP/1.1 200 OK
Date: Mon, 20 Nov 2017 00:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 20 Jul 2017 19:15:56 GMT
Content-Length: 38
Content-Type: text/html
Connection: Closed

<!DOCTYPE html>
  <body>

  Esta e' uma pagina de teste

  </body>
</html>
```

## 2.2. ESP Servidor WEB – HTTP

Um *ServidorWEB* troca *Mensagens HTTP* com um, ou vários Browsers, através da Internet. De facto a Internet é usada por várias aplicações para transmitirem as suas mensagens através de todo o mundo. Alguns exemplos são os *ClienteFTP/ServidorFTP*, *Telnet/ServidorTelnet*, e o *Email/Servidor Email*.
Usa-se habitualmente a expressão Internet mas de facto além do *Internet Protocol IP* é também usado, simultaneamente, o *Transmission Control Protocol TCP*.
O computador de destino, onde reside o servidor, ao receber uma dessas mensagens pela Internet, com base no seu "*TCP port number*", entrega-a à aplicação receptora mais adequada, pois só esta a saberá processar corretamente.

De uma forma simplista podemos afirmar que um servidor WEB não é mais que um programa que aceita ligações TCPIP, habitualmente na porta 80, e que responde com o conteúdo dos documentos HTML pedidos pelo Browser. Os pedidos do Browser consistem em mensagens de texto, enviadas por TCPIP para o servidor (ex. GET \index.html ). A sintaxe dessas mensagens/pedidos/respostas é definida pelo protocolo HTTP.

Entre outros, os programas *Word*, *Power Point*, e *Excel* permitem gravar documentos num formato *HTML* criando dessa forma páginas WEB. Estes programas permitem também incluir num documento referências a outros documentos ("Links - hiperligações").

Nos dois exemplos seguintes, o ESP devolve uma página HTML ao Browser,
mas:
- no exemplo a) a página HTML é estática.
- no exemplo b) a página altera-se em função do estado da saída digita nº2 (Led azul pequeno) do ESP.

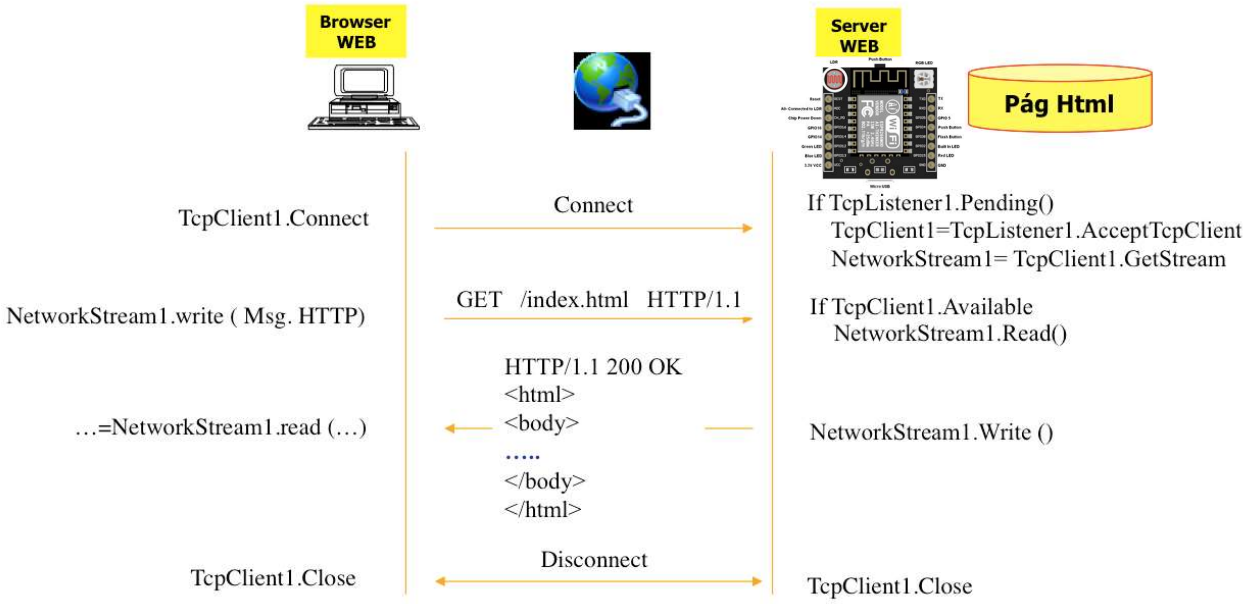



Figura 2.2: Troca de mensagens HTTP usando os objetos do tipo Wifi, WiFiServer, WifiCliente

Neste exemplo são usados três objetos, um de cada tipo: WiFi, WiFiServer e WiFiClient
https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/client-class.html

Objeto do tipo WiFi

| | |
|---|---|
| .begin | (pede ligação wifi ao router) |
| .status | (devolve o estado da ligação Wifi) |
| .localIP | (retorna o endereço IP do ESP, atribuído pelo Router) |

Objeto do tipo WiFiServer

| | |
|---|---|
| .begin | (fica à escuta de um pedido de ligação TCP/IP) |
| .available | (retorna a referencia da nova ligação) |

Objeto do tipo WiFiClient

| | |
|---|---|
| .available() | (devolve nº bytes recebidos pelo ESP) |
| .connected() | (devolve o estado da ligação TCP/IP, exemplo: ligado) |
| .readString() | (lê vários bytes) |
| .flush() | (limpa a msg recebida) |
| .print() | (envia uma string para o cliente, via TCP/IP) |

https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/client-class.html

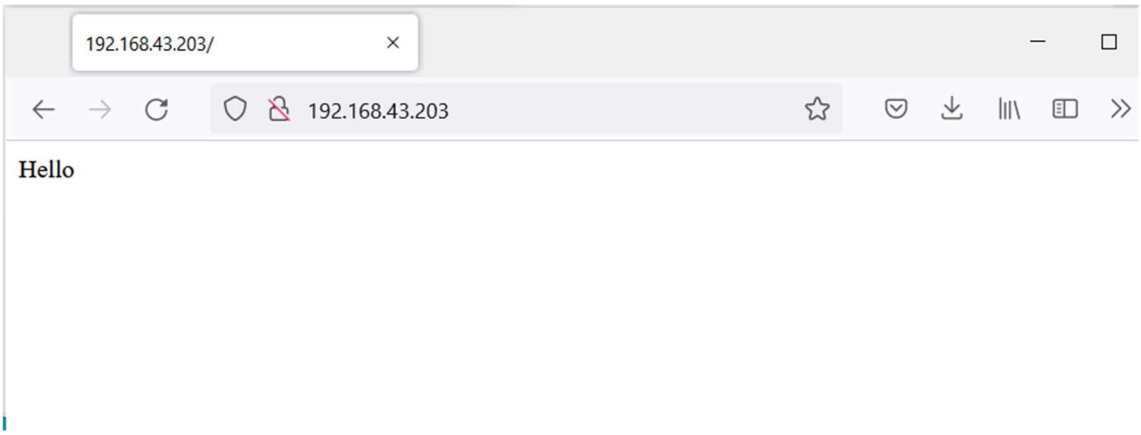
## Exemplo 2a) O ESP como servidor WEB, envia uma página HTML estática para o Browser

**Neste exemplo, tanto o ESP como o BrowserWEB devem estar ligados ao mesmo router ou telemóvel.**

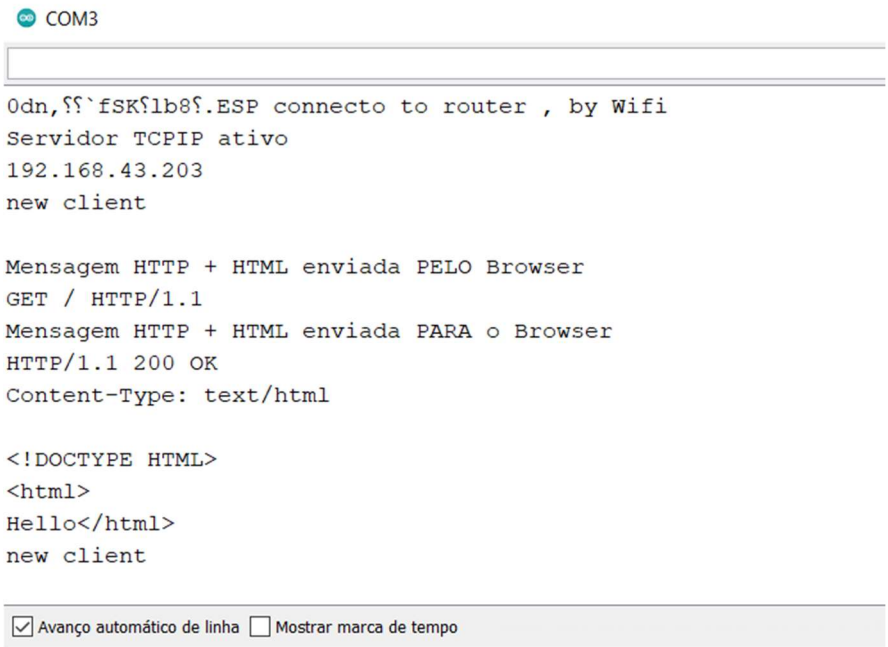
```
#include <ESP8266WiFi.h>
WiFiServer server(80);    // Servidor TCP/IP, espera pedidos vindos do Browser

void setup() {
        // Inicia o objecto Rs232
        Serial.begin(9600);

        // Pede ligação WiFi, ao Router
        WiFi.begin("NOS-88C0","password");

        // espera que o Router aceite a ligação WiFi
        while(WiFi.status() != WL_CONNECTED ) {
                delay(500);  Serial.print(".");
                }

        Serial.println("ESP connect to router , by Wifi");

        // ativa o servidor TCP/IP
        server.begin();
        Serial.println("Servidor TCPIP ativo");

        //  IP address
        Serial.println(WiFi.localIP());
} // setup

void loop() {
  // Espera que o Browser peça ligação TCP/IP
  WiFiClient client = server.available();

  if (client) {
      // Espera que o Browser envie dados, ex. GET index.html
      Serial.println("new client\r\n");
      while(!client.available()){ delay(1); }
      Serial.println("Mensagem HTTP + HTML enviada PELO Browser");
      String ss = client.readStringUntil('\r'); //String ss = client.readString();
      Serial.println(ss);
      client.flush();

      // Send the response to the client/Browser
client.print("HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html> \r\n");
client.print("Hello");
client.print("</html>\n");

      // Debug Send the response to Rs232
Serial.println("Mensagem HTTP + HTML enviada PARA o Browser");
Serial.print("HTTP/1.1 200 OK\r\nContent-Type:text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html> \r\n");
Serial.print("Hello");
Serial.print("</html>\n");
```

```
} // if(client)
delay(50);
} // loop
```

**Neste exemplo o ESP tem o IP: 192.168.43.203**
**Por isso no Browser deve aceder a http://192.168.43.203**
**O ESP responde com "Hello"**




**Na porta série, para efeitos de DEBUG, o ESP envia por Rs232 (COM3) a seguinte informação:**



## Exemplo 2b) O ESP como servidor WEB, o Browser controla saída digital do ESP

Assumindo que o IP atribuído ao ESP é o **192.168.43.203**
Através do browser deverá aceder a http://192.168.43.203/GPIO=1
ou a http://192.168.43.203/GPIO=0

```
#include <ESP8266WiFi.h>
WiFiServer server(80);
int val;

void setup() {
Serial.begin(9600);    // Inicia o objecto Rs232

WiFi.begin("NOS-88C0","");   // Pede ligação WiFi, ao Router
// espera que o Router aceite a ligação WiFi
while(WiFi.status() != WL_CONNECTED ) { delay(500);  Serial.print("."); }
Serial.println("ESP connecto to router , by Wifi");

// ativa o servidor TCP/IP interno ao ESP
server.begin();

Serial.println("Servidor TCPIP ativo");
Serial.println(WiFi.localIP());    //  Devolve IP address
pinMode(2,OUTPUT);
}

void loop() {
  WiFiClient client = server.available();     // Espera que o Browser peça ligação TCP/IP
  if (client) {  // Se chegou uma ligação TCPIP
       Serial.println("nova ligação TCPIP \r\n");
       while(!client.available()){ delay(1); }       // Espera que o Browser envie dados , ex. GET index.html
       Serial.println("Mensagem HTTP + HTML enviada PELO Browser");
       // Lê a mensagem recebida pelo ESP, enviada pelo Browser
       String ss = client.readStringUntil('\r'); //String ss = client.readString();
       Serial.println(ss);
       client.flush();
       // Processa mesg recebida pelo ESP
        if (ss.indexOf("GPIO=1") >0 ){val = 1;}
        if (ss.indexOf("GPIO=0") >0 ){val = 0;}
        digitalWrite(2,val);

 // Envia pág HTML para o Browser
 client.print("HTTP/1.1  200  OK\r\nContent-Type:  text/html\r\n\r\n<!DOCTYPE  HTML>\r\n<html>
\r\n");
            if(val==1){
                        client.print("GPIO=1");
                        }else{
                        client.print("GPIO=0");
                        }
       client.print("</html>\n");

 // Debug Send the response to Rs232
       Serial.println("Mensagem HTTP + HTML enviada PARA o Browser");
       Serial.print("HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>
\r\n");
       if(val==1){
          Serial.print("GPIO is 1");
          }else{
          Serial.print("GPIO is 0");
```

```
        }
      Serial.print("</html>\n");

   } // if(client)
delay(50);
} // Loop
```

## Exemplo 2c) O ESP como servidor WEB, envia uma página HTML com dois botões para o Browser

**Neste exemplo, tanto o ESP como o BrowserWEB devem estar ligados ao mesmo router ou telemóvel. A página WEB é definida num array usando o "rawliteral"**

Para criar uma string/array com "o cabeçalho HTTP e a página HTML", com várias linhas em "C", no ESP podem usar a função:

**R"rawliteral(**
**…..**
**Colocar aqui**
**As diversas linhas da resposta HTTP (cabeçalho + HTML)**
**……)rawliteral"**

Por exemplo, o array seguinte tem o nome **index_html[ ]** e contém o cabeçalho HTTP e a página HTML separados por uma linha em branco (a linha em branco é necessária):

```
const char index_html[] = R"rawliteral(
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE HTML>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
<form action="GPIO=0" method="get">
<button type="submit">Ligar Led</button>
</form>
<form action="GPIO=1" method="get">
<button type="submit">Desligar Led</button>
</form>
</body>
</html>)rawliteral";
```

***Programa exemplo:***

```
#include <ESP8266WiFi.h>
WiFiServer server(80);
```

```
WiFiClient client;

int val;

const char index_html[] = R"rawliteral(
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE HTML>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>

<form action="GPIO=0" method="get">
 <button type="submit">Ligar Led</button>
</form>
<form action="GPIO=1" method="get">
 <button type="submit">Desligar Led</button>
</form>
</body>
</html>)rawliteral";


void setup() {

// Inicia o objecto Rs232
Serial.begin(9600);

// Pede ligação WiFi, ao Router
WiFi.begin("Redmi2","");

// espera que o Router aceite a ligação WiFi
while(WiFi.status() != WL_CONNECTED ) { delay(500);  Serial.print("."); }
Serial.println("ESP connecto to router , by Wifi");

// ativa o servidor TCP/IP
server.begin();
Serial.println("Servidor TCPIP ativo");

//  IP address
Serial.println(WiFi.localIP());
pinMode(2,OUTPUT);

}  // Setup()


void loop() {

  // Espera que o Browser peça ligação TCP/IP
  client = server.available();
// Se chegou uma ligação TCPIP
```

```
if (client.connected()){
     Serial.println("nova ligação TCPIP \r\n");
   }

// Se o Browser enviou dados , ex. GET index.html
  if(client.available()){
       // Lê a primeira linha da msg enviada pelo Browser
       String ss = client.readStringUntil('\r');       //String ss = client.readString();
       // Processa mesg recebida pelo ESP
       if (ss.indexOf("GPIO=1") >0 ){val = 1;}
       if (ss.indexOf("GPIO=0") >0 ){val = 0;}
       digitalWrite(2,val);
       client.print(index_html);        // Envia pág HTML para o Browser
       Serial.println(ss);              // Debug – mostra msg enviada pelo Browser, ex GET ...
       Serial.print(index_html);        // Debug - Envia pág HTML por Serie (COM3)
       }

   delay(50);
} // Loop
```

# Trab2 Módulo IOT como servidor TCP/IP c/ pág HTML para controlo de saída digital

Pretende-se ter um só botão na página WEB que permita ligar e desligar o led azul pequeno (como fizemos na aula) mas agora o botão formatado com CSS. (Neste trabalho pretende-se criar uma página HTML + CSS "dentro" do ESP.*)*

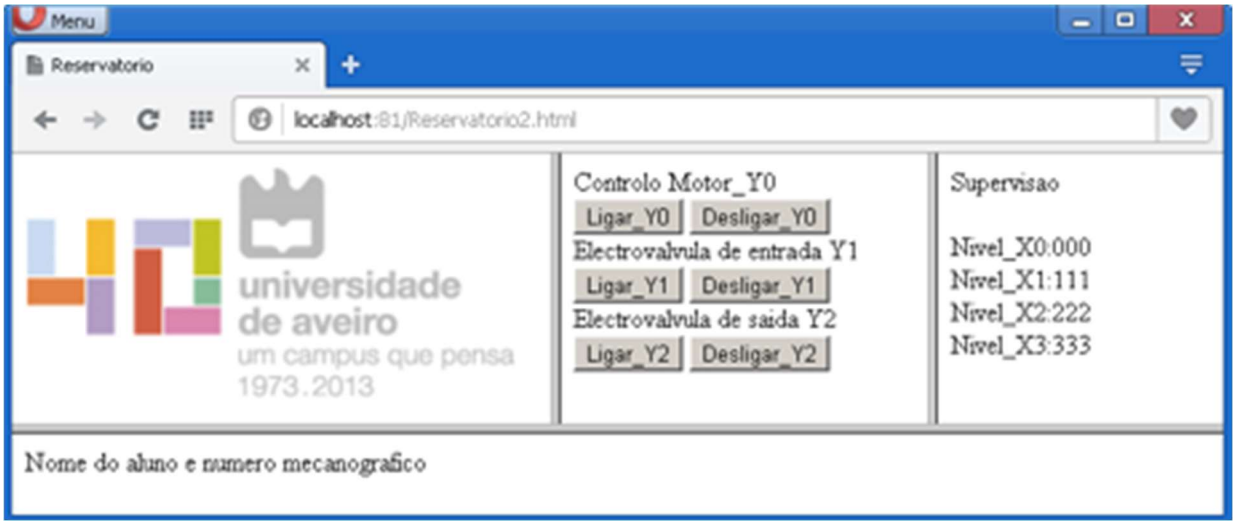
**– Módulo IOT Wifi Servidor**
Com o Módulo deve criar um servidor HTTP, enviar páginas HTML e CSS, por HTTP, para o telemóvel, permitindo monitorizar no Browser do telemóvel as entradas e saídas digitais e analógicas do módulo para detectar o nível e água. *Utilize um sensor de nível analógico para adquirir também o nível de água..*

*Bibliografia:*
*Além dos apontamentos, vejam este youtube:* Como Fazer um WebServer com ESP8266 sendo Controlado pela Web - Video #4 - ESP8266 Primeiros Passos
https://www.youtube.com/watch?v=woPFuwvHPoA&list=PL7CjOZ3q8fMe6DxojEFuDx4BP0qbbpKtP&index=4

**– Web Front end - HTML e CSS**
desenvolva a páginas Web de supervisão e controlo do reservatório (com as várias frames e CSS)

Recordam-se da interface que desenvolvemos em Informática Industrial para controlar o reservatório ?!

Não necessita de ser tão elaborada:




1º objetivo: ter um só botão na página WEB que permita ligar e desligar o led azul pequeno (como fizemos na aula) mas agora o botão formatado com CSS. (Neste trabalho pretende-se criar uma página HTML + CSS "dentro" do ESP.)

2º objetivo: além do botão que controla o led azul, receber no browser o estado de outros pinos do ESP (n1, n2 e do próprio led azul)