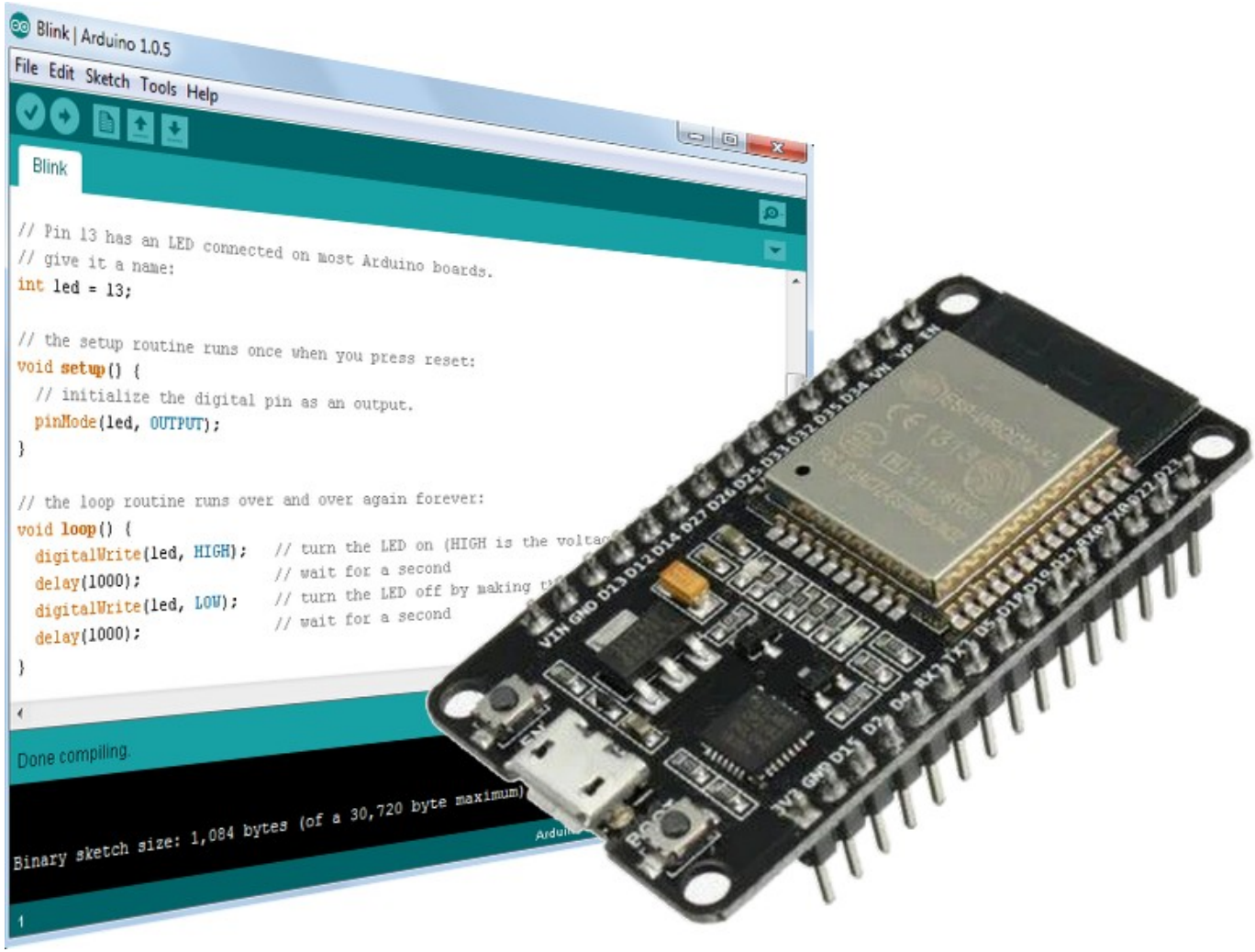
ESP32 mikrovezérlők programozása Arduino környezetben

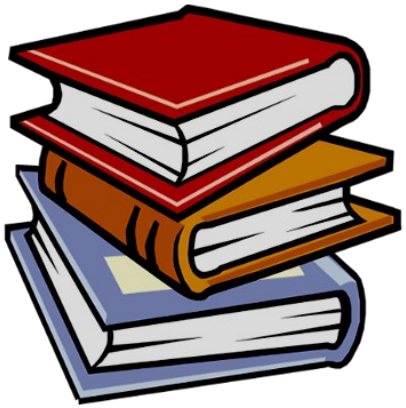


ARDUINO™

10. ESP32 webszerverek – 1. rész

Felhasznált és ajánlott irodalom

- **Leírások, dokumentáció**
  - W3Schools: HTML Tutorial
  - W3Schools: CSS Tutorial
  - W3Schools: AJAX Tutorial
  - ESPRESSIF: ESP32 Arduino Core Documentation



- **Mintaprojektek**
  - ESP8266 Community: ESP8266 WiFiServer example
  - Last Minute Engineers: Simple ESP32 Web Server
  - Circuits4you: ESP8266 (ajax) update part of web page without refreshing
- **Arduino programkönyvtárak**
  - AsyncTCP
  - ESPAsyncWebServer
  - Adafruit: BME280 sensor library
  - Adafruit: DHT sensor library

Webszerver

- **Webszerver:** olyan Internet kapcsolattal rendelkező eszköz, amely **HTTP kéréseket** fogad és szolgál ki *(a **GET** és **POST** kérések formátumát és összehasonlításukat lásd a 2021. ápr. 15-i előadásban)*
- **HTTP status kódok:** A HTTP kérésre küldött válasz egy állapotjelző kódot tartalmaz, amely az alábbiak egyike lehet

Status Code	Meaning	
200	OK: the request was successful	Sikeres lekérés
303	See Other: used to redirect to a different URI, after a POST request, for instance	Átirányítás
400	Bad Request: the server couldn't understand the request, because the syntax was incorrect	
401	Unauthorized: user authentication is required	Jogosulatlan kérés, azonosítás kell
403	Forbidden: the server refuses to execute the request, authorization won't help	Letiltva
404	Not Found: the requested URI was not found	A kért erőforrás nem található
500	Internal Server Error: The server encountered an unexpected condition and couldn't fulfill the request	

ESP32 webszerver STA módban

- Az ESP32 webszerver STA (station) módban kapcsolódik a helyi hálózatra, s a korábbi előadásokban bemutatott módin fix IP címet rendelünk hozzá
- A helyi hálózat eszözeivel (laptop, tablet, mobil) a fix IP cím alapján könnyen elérhető

A csatlakozások titkos adatai

- Az **ESP32**-t a helyi hálózatra kliensként csatlakoztatjuk, az ehhez szükséges személyes adatokat kiszerveztük egy **secrets.h** nevű fejléc állományba, amelyet a **Vázlatfüzet** (Sketchbook) mappa **libraries/secrets** almappájában helyeztünk el

```
#include <WiFi.h>
#include "secrets.h"

void setup_wifi() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected! IP address: ");
  Serial.println(WiFi.localIP());
}
```

```
#define WIFI_SSID  "MY_SSID"
#define WIFI_PASS  "MY_PASSWORD"
```

ESP32 webszerver megvalósítások

- Három lehetőség közül választhatunk, amelyek különböző megközelítést kívánnak és különböző előnyöket kínálnak:
  - A **WiFi** programkönyvtár és a **WiFiServer** osztály használata
    Előny: maximális szabadság az üzenetek és a fejlécek kezelésében
    Hátrány: a kommunikáció a klienssel és az adatok értelmezése ránk marad
    Példaprogram: **ESP32_WiFiServer_demo**
  - A **WebServer** programkönyvtár használata
    Előny: kényelmes használat
    Hátrány: kötött fejléc generálás, egyidejűleg csak egy kliens kiszolgálása
    Példaprogramok: **ESP32_WebServer_simple**, **ESP32_WebServer_twoled**, **ESP32_ledswitch_ajax**
  - Az ESPAsyncWebServer programkönyvtár (AsyncTCP is kell hozzá)
    Előny: kényelmes használat, gyors, több kliens, gazdag szolgáltatáskínálat
    Hátrány: külön kell telepíteni a könyvtárakat (verzió inkompatibilitás?)
    Példák: **ESP32_AsyncWebserver_simple**, **ESP32_AsyncWebServer_twoled**

WiFiserver – a rögösebb út

ESP32_WiFiServer_demo.ino - az alábbi minta alapján
arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/server-examples.html

```
#include <WiFi.h>
#include "secrets.h"
WiFiServer server(80);

void setup(void){
  setup_wifi();                          // Csatlakozás a helyi hálózatra
  server.begin();                        // Indítjuk a webszervert
  analogReadResolution(12);              // 12-bit felbontás
  analogSetAttenuation(ADC_11db);        // Méréshatár 0-3,3
}

void loop() {
  WiFiClient client = server.available();
  if (client) {                          // wait for a client to connect
    while (client.connected()) {
      if (client.available()) {
        String line = client.readStringUntil('\r');  // read request line by line
        if (line.length() == 1 && line[0] == '\n') { // wait for end of client's request
          client.println(prepareHtmlPage());
          break;
    } } }

    while (client.available()) {
      client.read();                     // let client finish its request
    }
    client.stop();                       // close the connection:
} }
```

Itt küldjük a kliensnek a választ

ESP32_WiFiServer_demo.ino

```
String prepareHtmlPage(){
  String htmlPage;
  htmlPage.reserve(1024);               // prevent ram fragmentation
  htmlPage = F("HTTP/1.1 200 OK\r\n"
               "Content-Type: text/html\r\n"
               "Connection: close\r\n"  // closed after
               "Refresh: 5\r\n"         // refresh every 5s
               "\r\n"
               "<!DOCTYPE HTML>"
               "<html><head><title>ESP32 WiFiServer</title>"
               "</head><body>Analog input:  ");
  float v = (analogRead(34)+80)/1239.0f;
  htmlPage += String(v,3) +" V";
  htmlPage += F("</body></html>\r\n");
  return htmlPage;
}
```

Ez a rész ne a RAM-ban fogja a helyet!

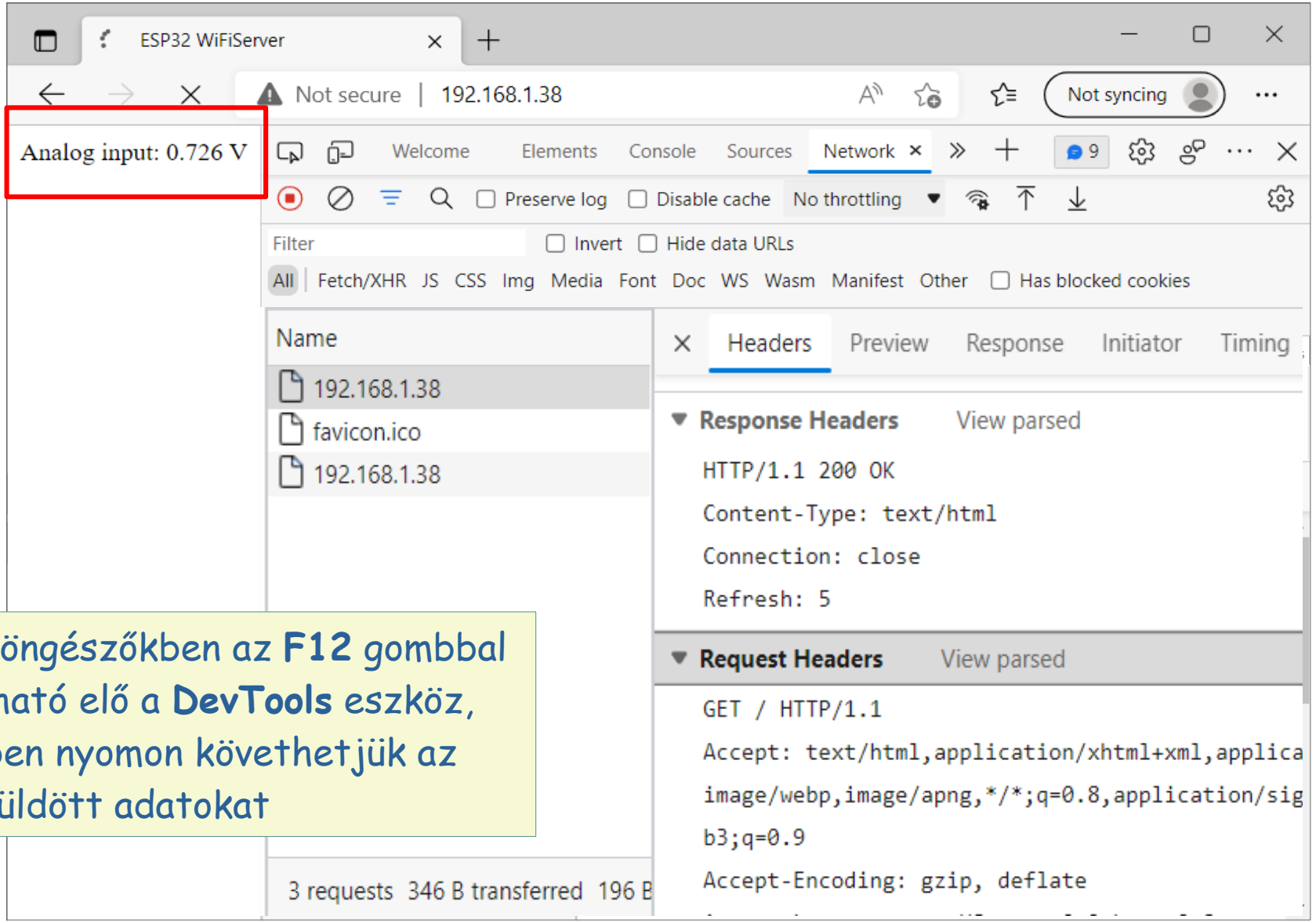
Üzenet fejrésze

Üzenet törzse

- A weblap összeállítását a fenti függvény végzi
- Az ADC kiolvasásával minden lekéréskor aktualizáljuk a lapot
- A kliens kapcsolatot a kiszolgálás végén lezárjuk, s a kliens 5 s elteltével új lekéréssel frissíti a böngészőben a lapot

ESP32_WiFiServer_demo futási eredmény

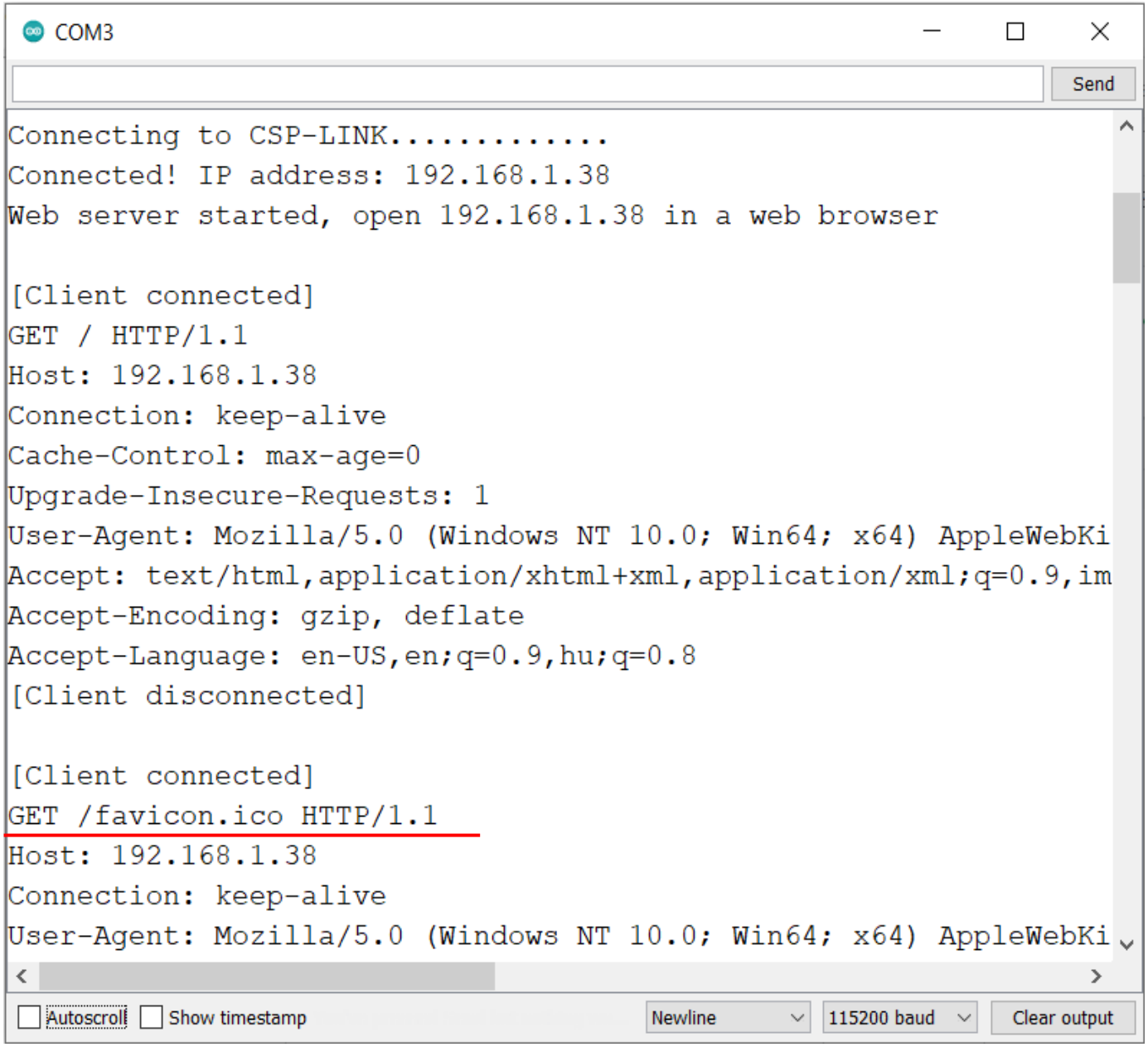
- Az alábbi ábrán a kijelzett eredmény és az üzenetváltás látható



A böngészőkben az **F12** gombbal hívható elő a **DevTools** eszköz, ebben nyomon követhetjük az átküldött adatokat

favico.ico lekérés – egy kis probléma

- A naplózás jól mutatja, hogy a HTML lekérés után a böngésző a (nemlétező) **favico.ico** ikont is lekéri automatikusan
- A következő oldalon megmutatjuk, hogyan lehet ezt megakadályozni



favico.ico lekérés – a probléma megoldva

```
String prepareHtmlPage() {
  String htmlPage;
  htmlPage.reserve(1024);               // prevent ram fragmentation
  htmlPage = F("HTTP/1.1 200 OK\r\n"
               "Content-Type: text/html\r\n"
               "Connection: close\r\n"  // closed after completion of the response
               "Refresh: 5\r\n"         // refresh the page automatically every 5 sec
               "\r\n"
               "<!DOCTYPE HTML>"
               "<html><head><title>ESP32 WiFiServer</title>"
               "<link rel=\"icon\" href=\"data:,\">"
               "</head><body>Analog input:  ");
  float v = (analogRead(34)+80)/1239.0f;
  htmlPage += String(v,3) +" V";
  htmlPage += F("</body></html>\r\n");
  return htmlPage;
}
```

- Bővítsük a HTML lap fejrészét az alábbi sorral:

  `<link rel="icon" href="data:,">`

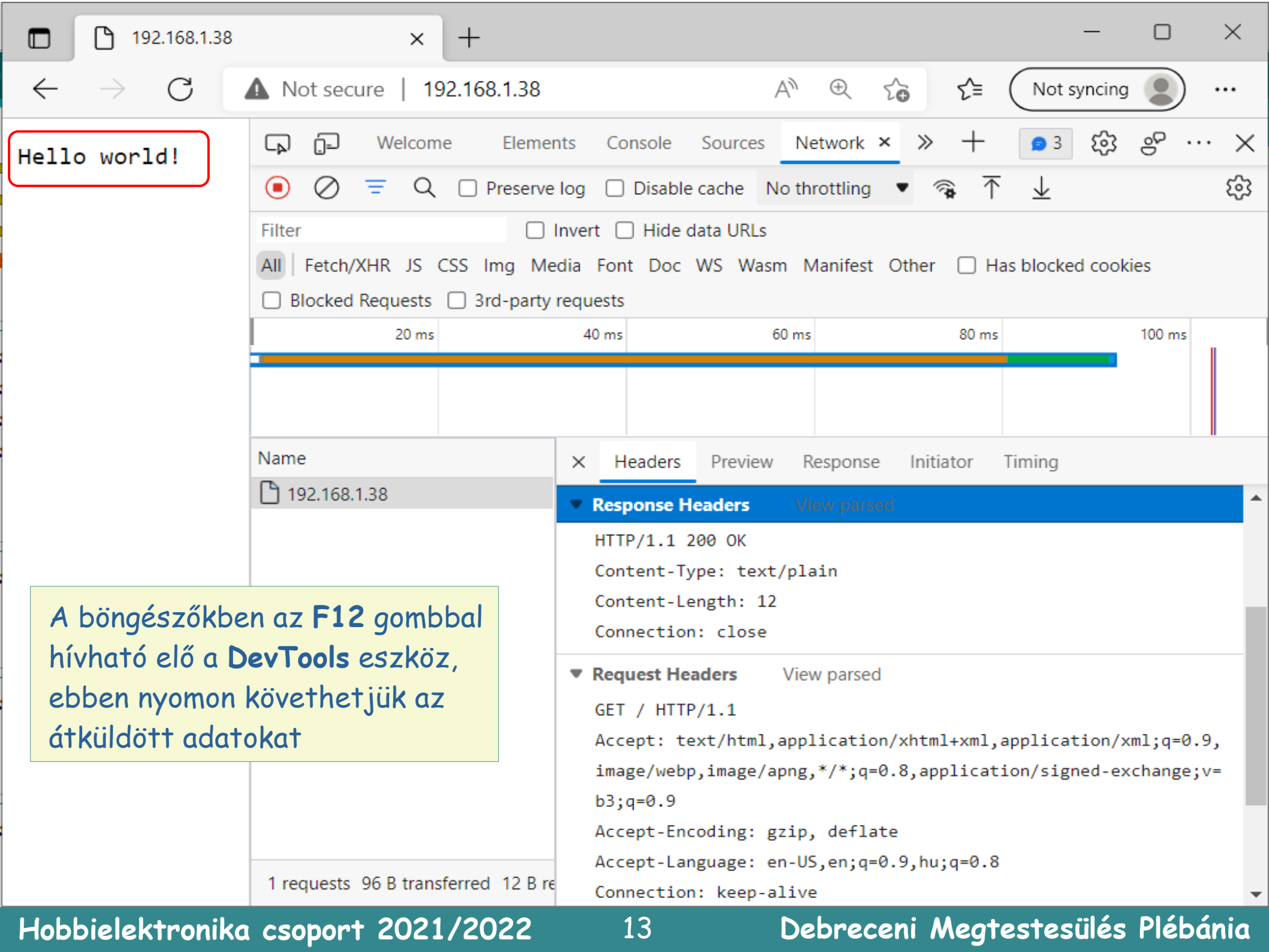
  bővebben lásd itt: How to Prevent Automatic Favicon Requests

A WebServer programkönyvtár

- Az **ESP32 Arduino Core** részeként a **WebServer** könyvtárat is megkaptuk, ezzel kényelmesebb dolgozni

ESP32_WebServer_simple.ino - a kényelmes út

```
#include <WiFi.h>
#include <WebServer.h>
#include "secrets.h"
WebServer server(80);                          // Webszerver objektumpéldány létrehozása

void setup(void){
  setup_wifi();                                // Csatlakozás a helyi hálózatra
  server.on("/", handleRoot);                  // Visszahívási függvény hozzárendelés
  server.onNotFound(handleNotFound);           // Visszahívási függvény hozzárendelés
  server.begin();                              // Indítjuk a webszervert
}

void loop(void){
  server.handleClient();                       // A kliens kapcsolatok kezelése
}

void handleRoot() {                            // A "/" URI lekérés kiszolgálása
  server.send(200, "text/plain", "Hello world!");
}

void handleNotFound(){                         // A sikertelen lekérések kiszolgálása
  server.send(404, "text/plain", "404: Not found");
}
```


192.168.1.38
Not secure | 192.168.1.38
Not syncing
Hello world!
Welcome
Elements
Console
Sources
Network
3
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
20 ms
40 ms
60 ms
80 ms
100 ms
Name
192.168.1.38
Headers
Preview
Response
Initiator
Timing
Response Headers
View parsed
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 12
Connection: close
Request Headers
View parsed
GET / HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=
b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,hu;q=0.8
Connection: keep-alive
1 requests 96 B transferred 12 B re
A böngészőkben az F12 gombbal hívható elő a DevTools eszköz, ebben nyomon követhetjük az átküldött adatokat

ESP32_AsyncWebserver_simple.ino

- Néhány eltéréstől eltekintve az **ESPAsyncWebserver** programkönyvtárnak is hasonló a használata, mint a **WebServer** könyvtáré

```
#include <WiFi.h>
#include <AsyncTCP.h>          // https://github.com/me-no-dev/AsyncTCP
#include <ESPAsyncWebServer.h> // https://github.com/me-no-dev/ESPAsyncWebServer
#include "secrets.h"

AsyncWebServer server(80);               // Webszerver objektumpéldány létrehozása

void setup(void) {
  setup_wifi();                          // Csatlakozás a helyi hálózatra
  server.on("/", HTTP_GET, handleRoot);// Visszahívási függvény hozzárendelés
  server.onNotFound(handleNotFound);     // Visszahívási függvény hozzárendelés
  server.begin();                        // Indítjuk a webszervert
}

void loop(void) { }

void handleRoot(AsyncWebServerRequest *request) {
  request->send(200, "text/plain", "Hello world!");
}

void handleNotFound(AsyncWebServerRequest *request) { // sikertelen lekérések
  request->send(404, "text/plain", "404: Not found");
}
```

A módszer megadása opcionális

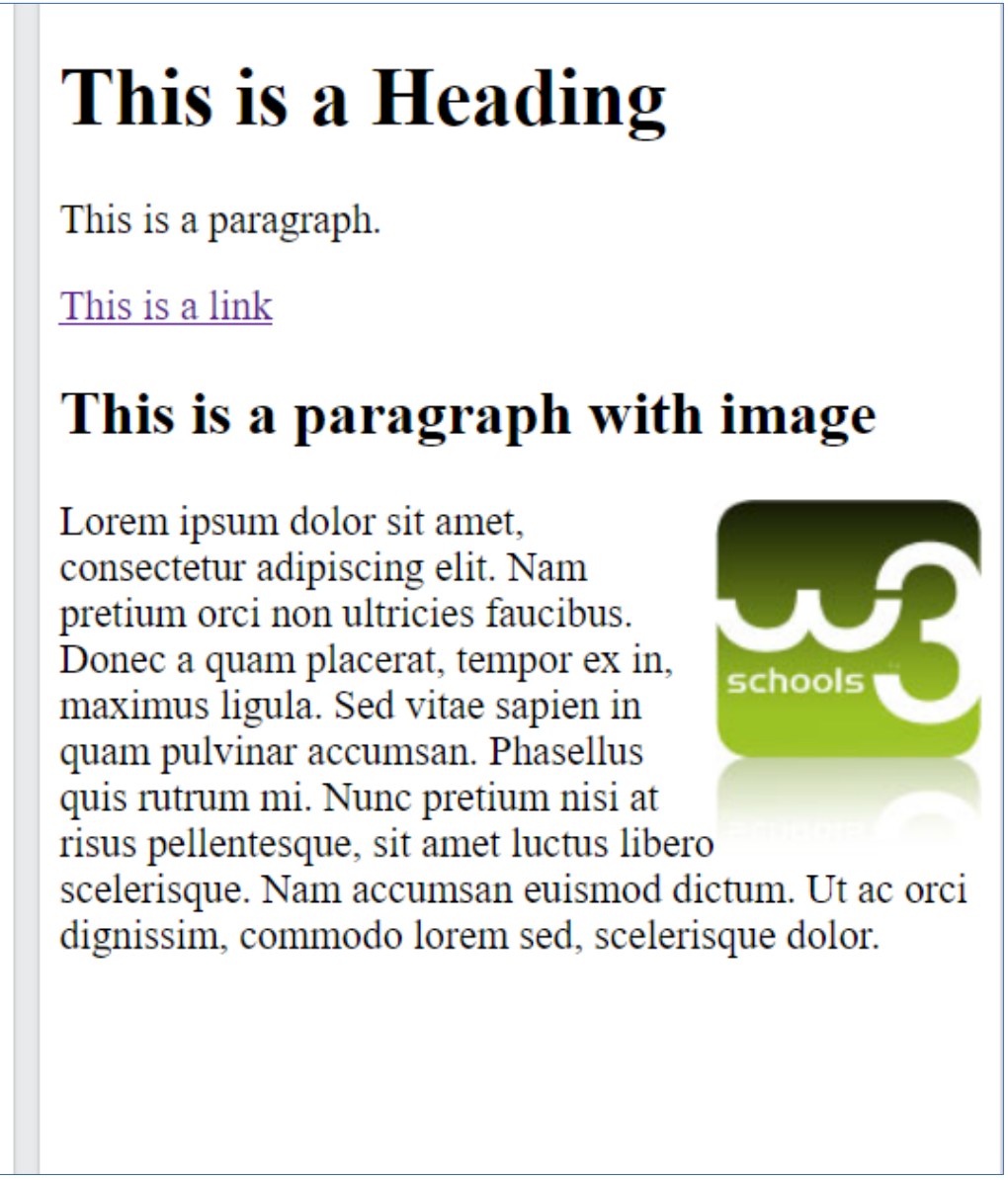
Nincs tennivaló!

A korábbiakhoz képest eltérés, hogy most a visszahívási függvényekben **request** objektumokat kezelünk

HTML alapok

- Egy tipikus HTML dokumentum szerkezete

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
<a href="https://www.w3schools.com">This is a
link</a>

<h2>This is a paragraph with image</h2>
<p><img src="w3schools.jpg"
alt="W3Schools.com" width="104" height="142"
align="right">Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nam pretium orci
non ultricies faucibus. Donec a quam placerat,
tempor ex in, maximus ligula. Sed vitae sapien
in quam pulvinar accumsan. Phasellus quis
rutrum mi. Nunc pretium nisi at risus
pellentesque, sit amet luctus libero
scelerisque. Nam accumsan euismod dictum. Ut
ac orci dignissim, commodo lorem sed,
scelerisque dolor. </p>
</body>
</html>
```



<head> és <meta> elemek

- A HTML **<head>** elem egy konténer a következő elemek számára: **<title>, <style>, <meta>, <link>** és **<script>**
- A **<meta>** elemek kísérő információkat adnak át a böngészőnek
- A **<link>** elemek külső erőforrások viszonyát/helyét adja meg a dokumentumhoz képest (stíluslapok, favico.ico). Globális Attribútumok megadását is támogatja

```
<!DOCTYPE html>
<html lang="hu">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="demopage">
  <title>Hobbielektronika csoport</title>
  <link rel="stylesheet" href="css/main.css">
  <link rel="icon" href="data:,">
</head>
<body>

</body>
</html>
```

Karakterkódolás

Reszponzivitás

Stíluslap betöltés

favico.ico lekérés tiltás

HTML stílusok

1. A stílust attribútumként használva egy elem megjelenését befolyásolja, formája:

```
<tagname style="property:value;">
```

```
<h2 style="color:blue;">This is a heading</h2>
<p style="color:red;">This is a paragraph.</p>
```

This is a heading

This is a paragraph.

```
<h1 style="font-family:Arial;">This is a heading</h1>

<p style="font-family:courier;">This is a paragraph.</p>

<p style="font-size:160%;">This is a paragraph.</p>

<p style="text-align:center;">Centered paragraph.</p>
```

This is a heading

This is a paragraph.

This is a paragraph.

Centered paragraph.

2. A stílust `<style>` elemként a `<head>` szekcióban is megadhatjuk

```
<!DOCTYPE html><html> <head>
<style>
  h2 {color:red;}
  p {color:blue;}
</style>
</head><body>
<h1>Big Title</h1>
<h2>This is a level 2 heading</h2>
<p>This is a paragraph.</p>
<h2>Another level 2 heading</h2>
<p>This is another paragraph</p>
</body></html>
```

Big Title

This is a level 2 heading

This is a paragraph.

Another level 2 heading

This is another paragraph

HTML stílusok

3. A HTML stílust külső fájlban is megadhatjuk, ekkor a **<head>** szekcióban egy **<link>** elem segítségével kell becsatolni: **<link rel="stylesheet" href="styles.css">**

- A HTML stílus leíró **CSS** (Cascaded Style Sheets) nyelv erősségét az ún. szelektorok adják, ezek segítségével dönti el a böngésző, hogy egy adott elemere melyik stílusdefiníció vonatkozik (vagy melyek vonatkoznak)

```
#para1 { text-align: center; color: red; }

.center { text-align: center; color: red; }

p.center { text-align: center; color: red; }
```

```
<p id="para1">Bekezdés</p>

Minden class=center elemre

<p class="center">Bekezdés</p>
```

```
<p class="center large">Bekezdés</p>   A center és a large osztály is vonatkozik rá
```

- Csoportosítás: az alábbi három stílusdefiníció összevonható

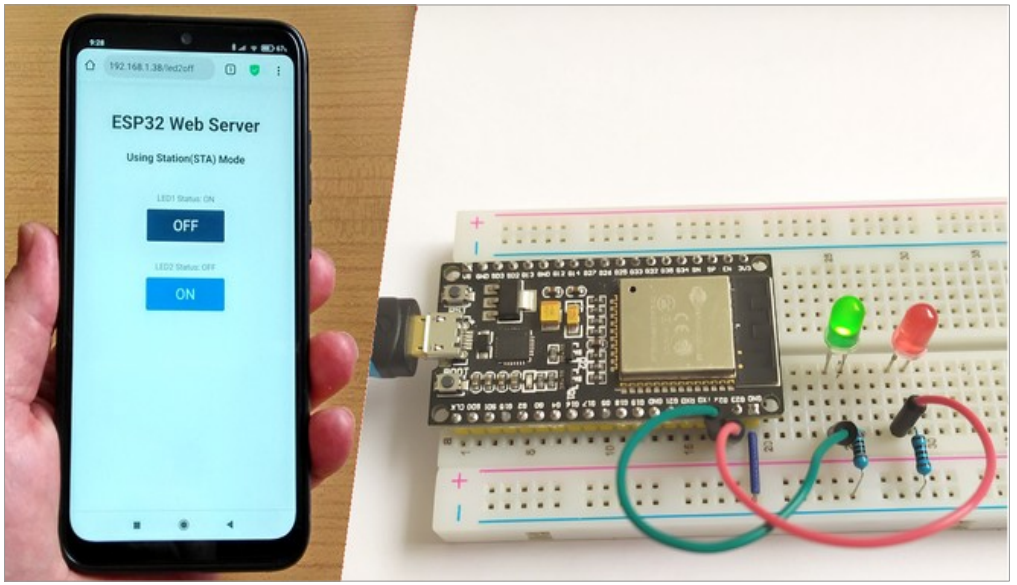
```
h1 { text-align: center; color: red; }

h2 { text-align: center; color: red; }

p { text-align: center; color: red; }
```

```
h1, h2, p {
    text-align: center;
    color: red;
}
```

ESP32_WebServer_twoled.ino 5/1.

- A Last Minute Engineers Simple ESP32 Web Server mintapéldája két LED távvezérlését mutatja be (a programot apró módosításokkal közöljük)

```
#include <WiFi.h>
#include <WebServer.h>
#include "secrets.h"
WebServer server(80);
bool LED1status = LOW;
bool LED2status = LOW;

void setup() {
  Serial.begin(115200);
  delay(100);
  pinMode(22, OUTPUT);
  pinMode(23, OUTPUT);
  setup_wifi();

  server.on("/", handle_OnConnect);
  server.on("/led1on", handle_led1on);
  server.on("/led1off", handle_led1off);
  server.on("/led2on", handle_led2on);
  server.on("/led2off", handle_led2off);
  server.onNotFound(handle_NotFound);
  server.begin();
  Serial.println("HTTP server started");
}
```



ESP32_WebServer_twoled.ino 5/2.

```
void loop() {
  server.handleClient();
  if (LED1status) { digitalWrite(LED1pin, HIGH); }
  else { digitalWrite(LED1pin, LOW);  }
  if (LED2status) { digitalWrite(LED2pin, HIGH); }
  else { digitalWrite(LED2pin, LOW);  }
}

void handle_OnConnect() {
  Serial.println("GPIO22 Status: OFF | GPIO23 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status, LED2status));
}

void handle_led1on() {
  LED1status = HIGH;
  Serial.println("GPIO22 Status: ON");
  server.send(200, "text/html", SendHTML(true, LED2status));
}

void handle_led1off() {
  LED1status = LOW;
  Serial.println("GPIO22 Status: OFF");
  server.send(200, "text/html", SendHTML(false, LED2status));
}
```

ESP32_WebServer_twoled.ino 5/3.

```
void handle_led2on() {
  LED2status = HIGH;
  Serial.println("GPIO23 Status: ON");
  server.send(200, "text/html", SendHTML(LED1status, true));
}

void handle_led2off() {
  LED2status = LOW;
  Serial.println("GPIO23 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status, false));
}

void handle_NotFound() {
  server.send(404, "text/html", SendErrorPage());
}

void setup_wifi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    Delay(500); Serial.print(".");
  }
  Serial.print("\r\nConnected! IP address: ");
  Serial.println(WiFi.localIP());
}
```

ESP32_WebServer_twoled.ino 5/4.

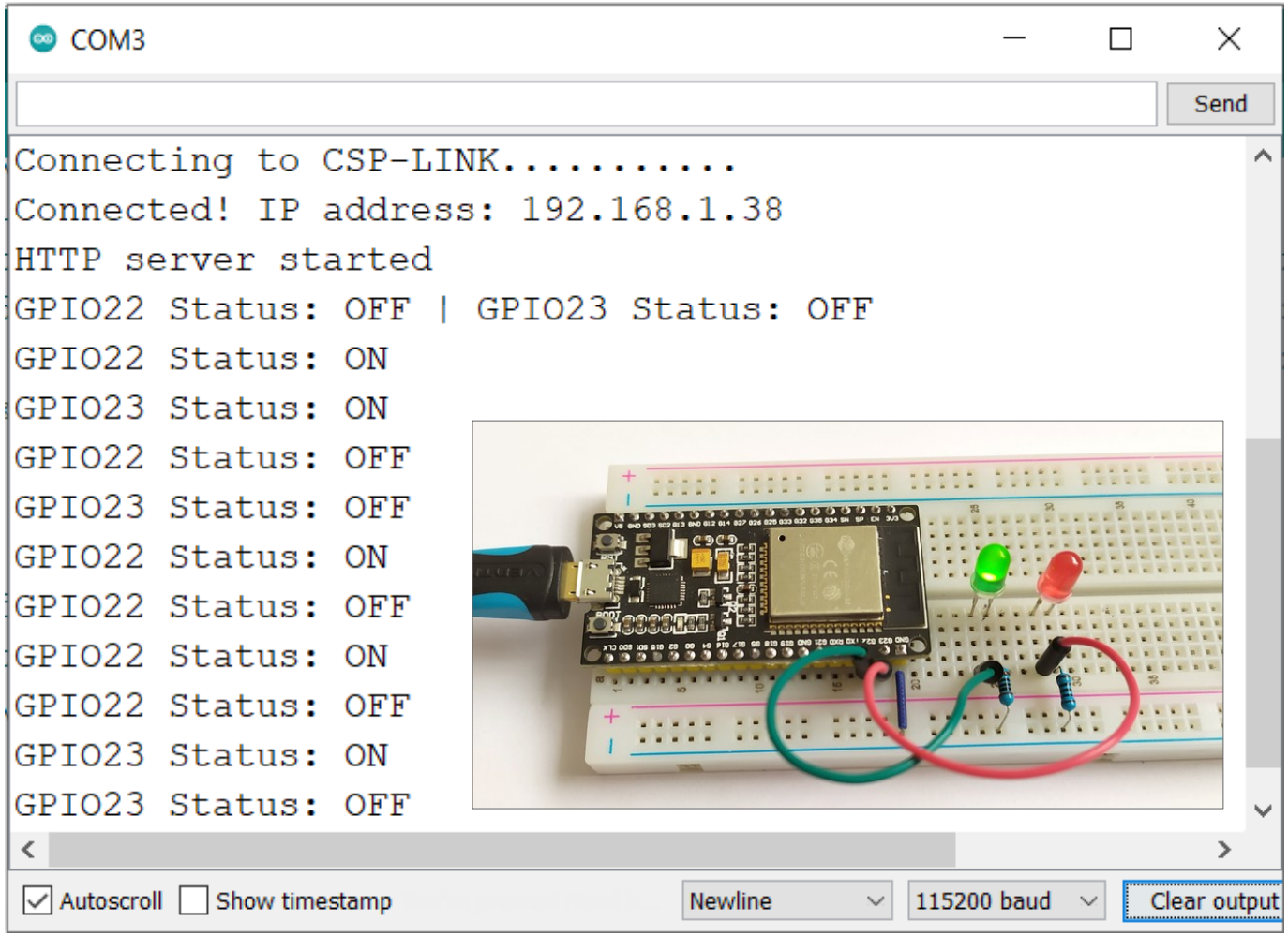
```
String SendErrorPage() {
  String ptr;
  ptr.reserve(1024);
  ptr = F("<!DOCTYPE html> <html>\n"
          "<head><meta name=\"viewport\" content=\"width=device-width,
           initial-scale=1.0, user-scalable=no\">\n"
          "<title>LED Control</title>\n"
          "<style>html { font-family: Helvetica; display: inline-block;
           margin: 0px auto; text-align: center;}\n"
          "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}
           h3 {color: #444444;margin-bottom: 50px;}\n"
          ".button {display: block;width: 80px;background-color: #3498db;
           border: none;color: white;padding: 13px 30px;text-decoration: none;
           font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius:4px;}\n"
          "p {font-size: 14px;color: #888;margin-bottom: 10px;}\n"
          "</style>\n"
          "</head>\n"
          "<body>\n"
          "<h1>404 Page not found</h1>\n"
          "<h3>Press button to open start page</h3>\n"
          "<a class=\"button\" href=\"/\">RESTART</a>\n"
          "</body></html>\n");
  return ptr;
}
```

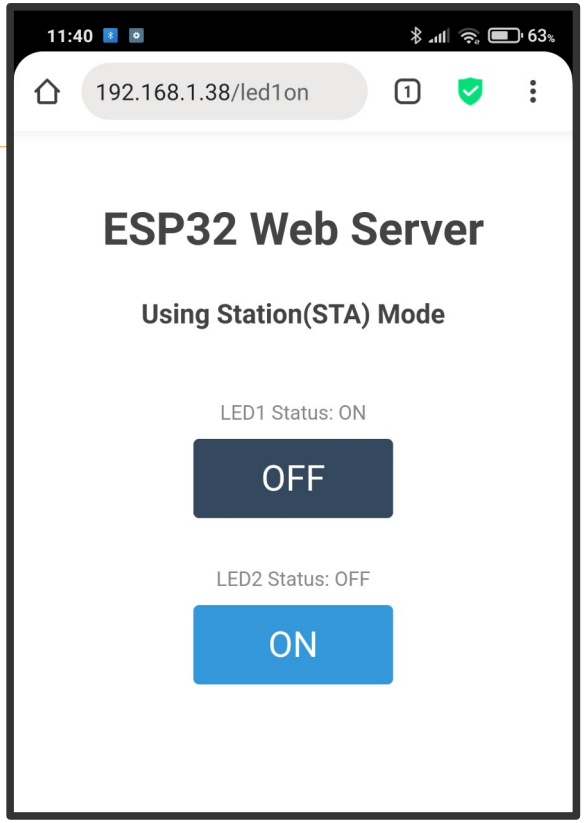
```
String SendHTML(uint8_t led1stat, uint8_t led2stat) {
  String ptr; ptr.reserve(1024);
  ptr = F("<!DOCTYPE html> <html>\n"
          "<head><meta name=\"viewport\" content=\"width=device-width,
           initial-scale=1.0, user-scalable=no\">\n
           <title>LED Control</title>\n"
          "<style>html { font-family: Helvetica; display: inline-block;
           margin: 0px auto; text-align: center;}\n"
          "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}
           h3 {color: #444444;margin-bottom: 50px;}\n"
          ".button {display: block;width: 80px;background-color: #3498db;border: none;
           color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;
           margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n"
          ".button-on {background-color: #3498db;}\n"
          ".button-on:active {background-color: #2980b9;}\n"
          ".button-off {background-color: #34495e;}\n"
          ".button-off:active {background-color: #2c3e50;}\n"
          "p {font-size: 14px;color: #888;margin-bottom: 10px;}\n"
          "</style>\n</head>\n<body>\n<h1>ESP32 Web Server</h1>\n"
          "<h3>Using Station(STA) Mode</h3>\n");
  if (led1stat) { ptr += "<p>LED1 Status: ON</p><a class=\"button button-off\"
                   href=\"/led1off\">OFF</a>\n"; }
  else { ptr += "<p>LED1 Status: OFF</p><a class=\"button button-on\"
                  href=\"/led1on\">ON</a>\n"; }
  if (led2stat) { ptr += "<p>LED2 Status: ON</p><a class=\"button button-off\"
                   href=\"/led2off\">OFF</a>\n"; }
  else { ptr += "<p>LED2 Status: OFF</p><a class=\"button button-on\"
                 href=\"/led2on\">ON</a>\n"; }
  ptr += F("</body></html>\n");
  return ptr;
}
```

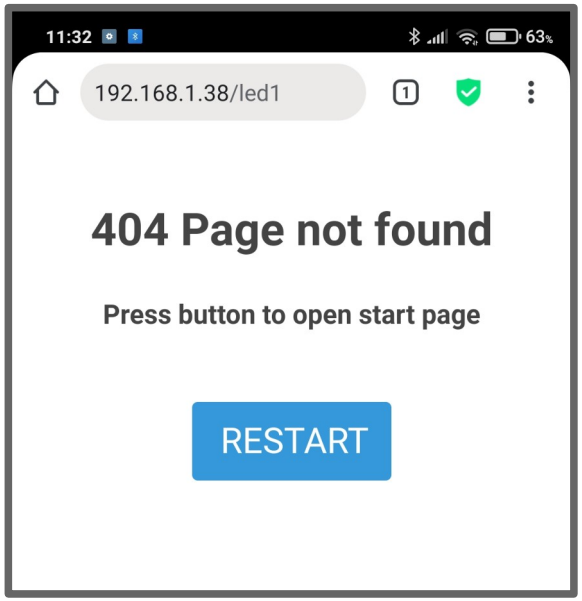
ESP32_WebServer_twoled.ino 5/5.

ESP32_WebServer_twoled

- Az ábrákon a program futási eredménye látható
- A hibajelző oldal mobiltelefonon is jól olvasható







ESP32_AsyncWebServer_twoled

- Az előző programot az **AsyncWebServer** könyvtárhoz is átírtuk
  - A **loop()** függvényben nem kell kezelgetni a szervert
  - A kiküldeni kívánt **HTML** lapot könnyen aktualizálhatjuk egy feldolgozó függvénnyel, amely a szövegben elhelyezett *helyőrzőket* kicseréli az aktuális adatra vagy szövegre, például:

```
String processor(const String& var) {
  if(var == "HELLO_FROM_TEMPLATE")      // A %HELLO_FROM_TEMPLATE% helyőrzőt
    return F("Hello world!");           // "Hello world!"-re cseréljük
  return String();
}
// ...
const char index_html[] PROGMEM = "..."; // large char array, tested with 14k
request->send_P(200, "text/html", index_html, processor);
```

- A HTML kódot külön állományban és kezelhetőbb formában tároljuk

```
const char index_html[] PROGMEM = R"=====(
<!DOCTYPE html>
<html><head> ... </head>
<body><h1>ESP32 Web Server</h1>
<p>%HELLO_FROM_TEMPLATE%</p>
</body></html>)=====";
```

ESP32_AsyncWebServer_twoled.ino 5/1.

```
#include <WiFi.h>
#include <AsyncTCP.h>                   // http://github.com/me-no-dev/AsyncTCP
#include <ESPAsyncWebServer.h>          // http://github.com/me-no-dev/ESPAsyncWebServer
#include "secrets.h"                    // WIFI_SSID és WIFI_PASS
#include "index.h"                      // index_html és error_html szövegállományok

AsyncWebServer server(80);              // Webszerver objektumpéldány létrehozása
uint8_t LED1pin = 22;                   // LED1 anódja a GPIO22 kivezetésre van kötve
uint8_t LED2pin = 23;                   // LED2 anódja a GPIO23 kivezetésre van kötve
bool LED1status = LOW;
bool LED2status = LOW;

void setup() {
  Serial.begin(115200);
  pinMode(LED1pin, OUTPUT);
  pinMode(LED1pin, OUTPUT);
  setup_wifi();                         // Csatlakozás a helyi hálózatra
  server.on("/", handle_OnConnect);     // Nyitólap lekérése
  server.on("/led1on", handle_led1on);  // LED1 be
  server.on("/led1off",handle_led1off); // LED1 ki
  server.on("/led2on", handle_led2on);  // LED2 be
  server.on("/led2off",handle_led2off); // LED2 ki
  server.onNotFound(handle_NotFound);   // Ismeretlen kérése
  server.begin();                       // Szerver indítása
  Serial.println("HTTP server started");
}
```

ESP32_AsyncWebServer_twoled.ino 5/2.

```
void handle_OnConnect(AsyncWebServerRequest *request) {
  LED1status = false; LED2status = false;
  Serial.println("GPIO22 Status: OFF | GPIO23 Status: OFF");
  request->send_P(200, "text/html", index_html, processor);
}

void handle_led1on(AsyncWebServerRequest *request) {
  LED1status = true;
  Serial.println("GPIO22 Status: ON");
  request->send_P(200, "text/html", index_html, processor);
}

void handle_led1off(AsyncWebServerRequest *request) {
  LED1status = false;
  Serial.println("GPIO22 Status: OFF");
  request->send_P(200, "text/html", index_html, processor);
}

void handle_led2on(AsyncWebServerRequest *request) {
  LED2status = true;
  Serial.println("GPIO23 Status: ON");
  request->send_P(200, "text/html", index_html, processor);
}

void handle_led2off(AsyncWebServerRequest *request) {
  LED2status = false;
  Serial.println("GPIO23 Status: OFF");
  request->send_P(200, "text/html", index_html, processor);
}
```

ESP32_AsyncWebServer_twoled.ino 5/3.

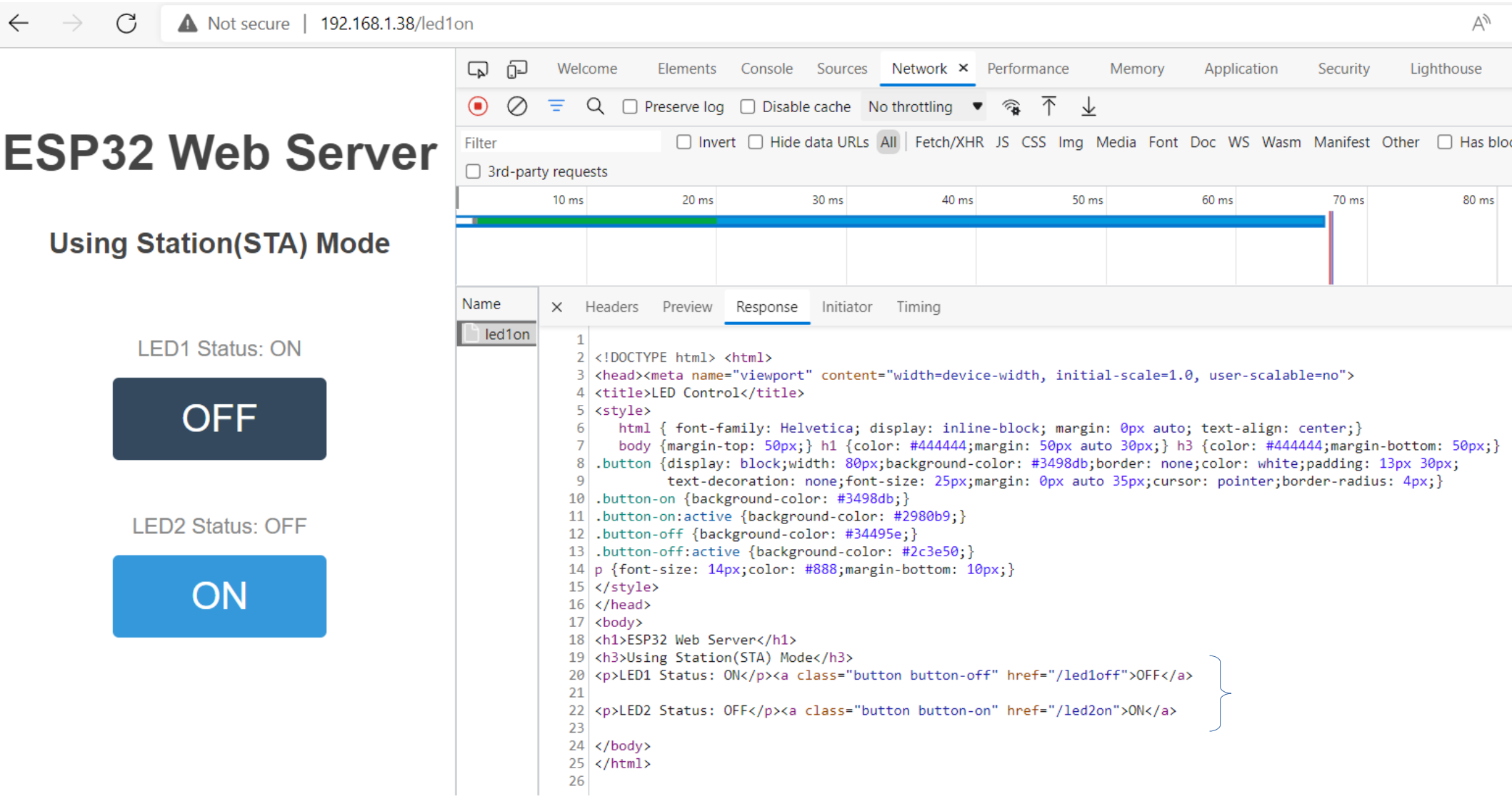
```
void loop() {
  if (LED1status) { digitalWrite(LED1pin, HIGH); }
  else { digitalWrite(LED1pin, LOW);  }
  if (LED2status) { digitalWrite(LED2pin, HIGH); }
  else { digitalWrite(LED2pin, LOW);  }
}

String processor(const String& var) {
  if(var == "LED1BTN") {       // Helyettesítjük a %LED1BTN% sablont
    if(LED1status) return F("<p>LED1 Status: ON</p>
                  <a class=\"button button-off\" href=\"/led1off\">OFF</a>\n");
    else return F("<p>LED1 Status: OFF</p><a class=\"button button-on\"
                 href=\"/led1on\">ON</a>\n");
  }
  if(var == "LED2BTN") {       // Helyettesítjük a %LED2BTN% sablont
    if(LED2status) return F("<p>LED2 Status: ON</p>
                  <a class=\"button button-off\" href=\"/led2off\">OFF</a>\n");
    else return F("<p>LED2 Status: OFF</p><a class=\"button button-on\"
                  href=\"/led2on\">ON</a>\n");
  }
  return String();
}

void handle_NotFound(AsyncWebServerRequest *request) {
  request->send_P(404, "text/html", error_html);
}
```

Az index.h állomány tartalma (részletek)

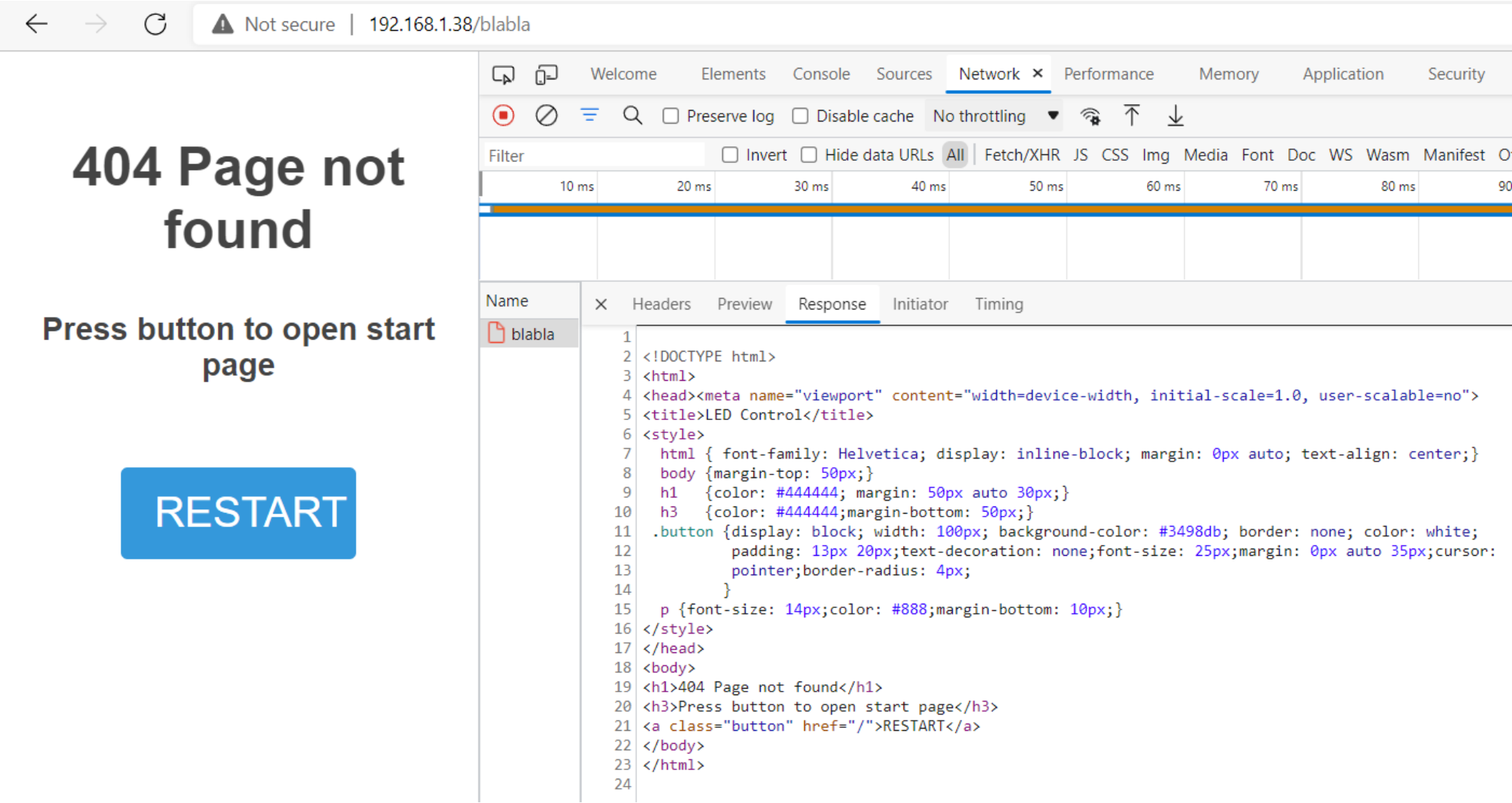
```
const char index_html[] PROGMEM = R"=====(
<!DOCTYPE html> <html>
<head> ... </head>
<body>
  <h1>ESP32 Web Server</h1>
  <h3>Using Station(STA) Mode</h3>
  %LED1BTN%
  %LED2BTN%
</body>
</html>
)=====";
const char error_html[] PROGMEM = R"=====(
<!DOCTYPE html>
<html><head>
 .button {display: block; width: 100px; background-color: #3498db;
          border: none; color: white; padding: 13px 20px;text-decoration: none;
          font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;
         }
 . . .
</head> <body>
<h1>404 Page not found</h1>
<h3>Press button to open start page</h3>
<a class="button" href="/">RESTART</a>
</body>
</html>
)=====";
```

ESP32_AsyncWebServer_twoled futási eredmény

- A processor függvény a LED-ek állapotától függően kicseréli a `%LED1BTN%` és `%LED2BTN%` szöveget a megfelelő HTML kódra



Hibás lekérés eredménye

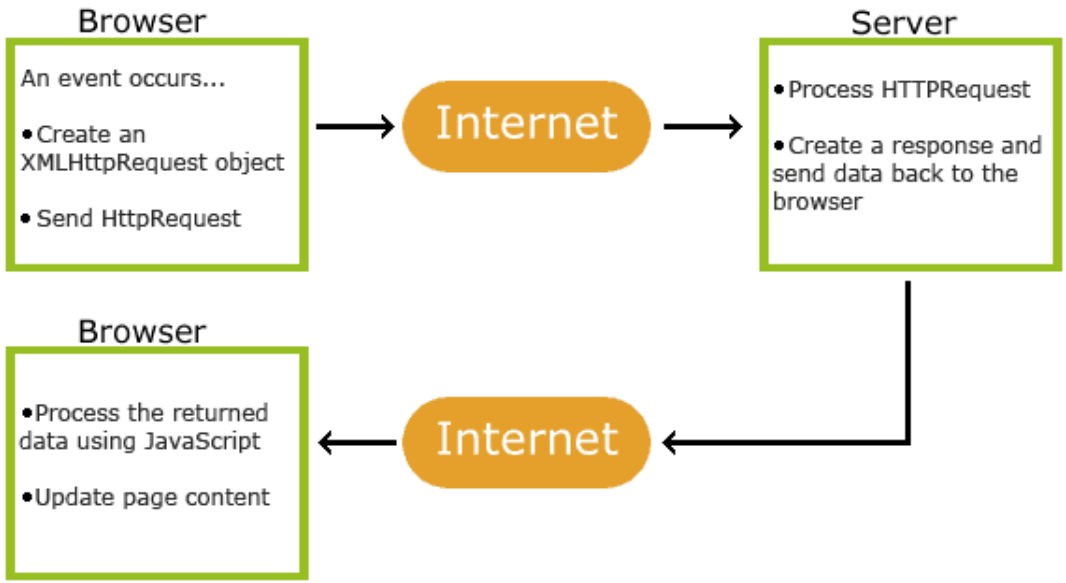


AJAX - Asynchronous JavaScript And XML

- Az **AJAX** nem programnyelv, hanem egy technika, amellyel
  - Frissíthető egy weblap újratöltés nélkül
  - Adatot kérhetünk le a szerverről – a weblap letöltése után
  - Adatot fogadhatunk a szervertől – a weblap letöltése után
  - Adatot küldhetünk a szervernek – a háttérben
- Az **AJAX** technika kombinálja a böngésző beépített **XMLHttpRequest** objektumát (az adatok lekérésére) és a **JavaScriptet** az adatok dinamikus megjelenítésére

- Bővebb információ:

  W3Schools: XML AJAX



Weblapok automatikus frissítése

- Ha a weblap HTML fejrészében szerepel az alábbi sor:
  `<meta http-equiv="refresh" content="5">`
  akkor a weblapot a böngésző 5 s-onként automatikusan frissíti
- Ehelyett az AJAX technikát is használhatjuk, lecserélve az automatikus frissítést előíró sort egy **JavaScript** függvényre:
  (a </head> címke előtt) helyezzük el az az alábbi **JavaScript** kódot:

```
<script>
  setInterval(loadDoc,5000);                    // 5000 ms-onként indíja a loadDoc() fv-t
  function loadDoc() {
    var xhttp = new XMLHttpRequest();                    // Új objektum példányosítás
    xhttp.onreadystatechange = function() {              // Inline visszahívási fv.
      if (this.readyState == 4 && this.status == 200) {
        document.body.innerHTML = this.responseText;     // A weblap frissítése
      }
    };
    xhttp.open("GET", "/", true);                        // Új Request beállítása
    xhttp.send();                                        // A lekérés indítása
  }
</script>
```

Ezzel így nem sokkal jutottunk előbbre, mert így is a teljes lap frissül

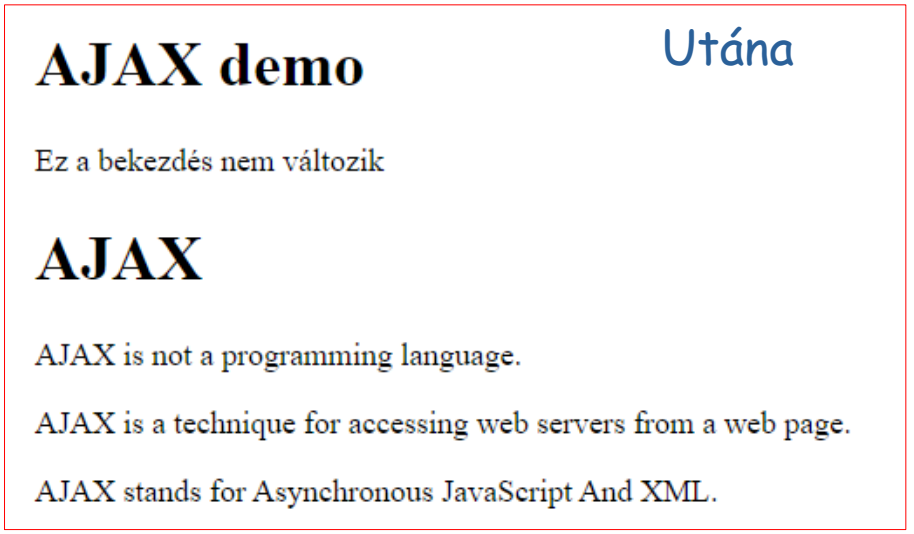
AJAX – finomítjuk a technikát

- A `<div id=”demo”>` megjelöl egy szakaszt, s a `getElementsById()` metódus segítségével csak a megnevezett szakaszt cseréljük le

```
<!DOCTYPE html>
<html>
<body>
<h1>AJAX demo</h1>
<p>Ez a bekezdés nem változik</p>
<div id="demo">
<h1>Ezt fogjuk lecserélni</h1>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>

</body>
</html>
```

Előtte

Utána



ESP32_ledswitch_ajax

- A circuits4you.com portálon található egy ideillő mintapélda: ESP8266 (ajax) update part of web page without refreshing (a programot természetesen átírtuk ESP32-re)
- Ebben a webszerver nyitólapja statikus szövegként a flash memóriában tárolódik, a `GET / HTTP/1.1` lekérésre ezt küldjük ki
- A nyitólapba beágyazott `getData()` JavaScript függvény **XmlHttpRequest** segítségével két másodpercenként lekéri az **ADC** által mért adatot (`GET readADC HTTP/1.1` request) és a weblap megjelölt helyére beszúrja a kapott értéket
- A weblap két nyomógombot is definiál, ezek kattintáskor a beágyazott `sendData()` JavaScript függvényt hívják meg, különböző paraméterrel. Ez a függvény **XmlHttpRequest** segítségével adatot küld a szervernek a beépített LED kapcsolgatására (`GET setLED?LEDstate=0 HTTP/1.1` vagy `GET setLED?LEDstate=1 HTTP/1.1` request)

index.h 2/1.

- A weblapunkat egy fejléc állományban helyezzük el

```
const char MAIN_page[] PROGMEM = R"=====(
<html><head><title>LED Control</title></head>
<body>
<div id="demo">
<h1>ESP32 webserver</h1><h3>Update web page without refresh</h3>
  <button type="button" onclick="sendData(1)">LED ON</button>
  <button type="button" onclick="sendData(0)">LED OFF</button><br>
</div>

<div>
  ADC Value is : <span id="ADCValue">0 </span> V<br>
  LED State is : <span id="LEDState">NA</span>
</div>
<script>
function sendData(led) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("LEDState").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "setLED?LEDstate="+led, true);
  xhttp.send();
}
```

Cserére kijelölt elemek

index.h 2/2.

```
setInterval(function() {
  // Call a function repetatively with 2 Second interval
  getData();
}, 2000); //2000mSeconds update rate

function getData() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("ADCValue").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "readADC", true);
  xhttp.send();
}

</script>
<br><br><a href="https://circuits4you.com">Circuits4you.com</a>
</body>
</html>
)=====";
```

ESP32_ledswitch_ajax.ino 2/1.

```
#include <WiFi.h>
#include <WebServer.h>
#include "secrets.h"
#include "index.h"                          // Our HTML webpage contents with javascripts
#define LED 2                               // On board LED (anode driven)
WebServer server(80);                       // Server on port 80

void setup(void) {
  Serial.begin(115200);
  analogReadResolution(12);                 // 12-bit felbontás
  analogSetAttenuation(ADC_11db);           // Méréshatár 0-3,3 V
  setup_wifi();
  //Onboard LED port Direction output
  pinMode(LED, OUTPUT);
  server.on("/", handleRoot);               // This is display page
  server.on("/setLED", handleLED);          // Serve LED requests
  server.on("/readADC", handleADC);         // Serve ADC requests
  server.begin();                           //Start server
  Serial.println("HTTP server started");
}

void loop(void) {
  server.handleClient();                    // Handle client requests
}
```

Amint látjuk, a webszervert működtető program egyszerű, mint a faék...

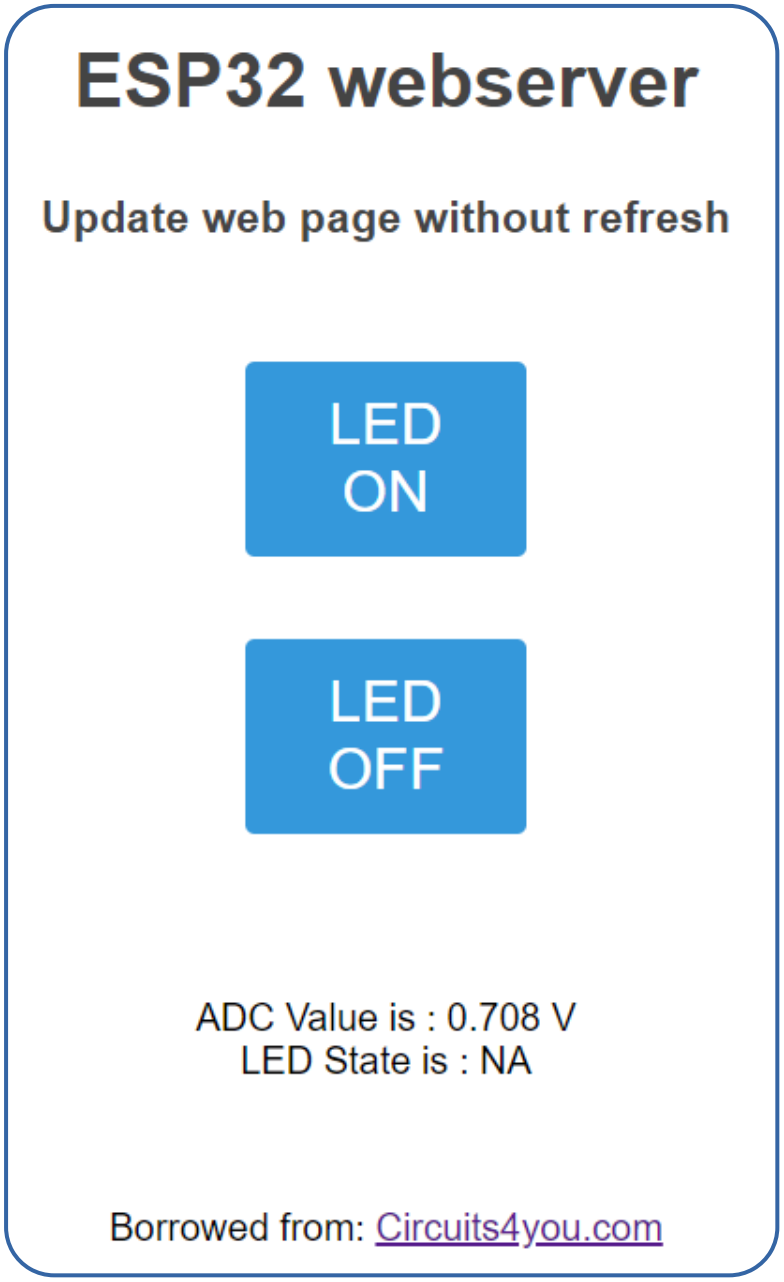
ESP8266_ledswitch_ajax.ino 2/2.

```
void handleRoot() {
  String s = FPSTR(MAIN_page);      //Read HTML contents from flash
  server.send(200, "text/html", s); //Send web page
}

void handleADC() {
  float v = (analogRead(34)+80)/1239.0f;    // Measure voltage az GPIO34
  String adcValue = String(v,3);
  server.send(200, "text/plain", adcValue); //Send ADC value to client ajax request
}

void handleLED() {
  String ledState = "OFF";
  String t_state = server.arg("LEDstate"); //xhttp.open("GET","setLED?LEDstate="+led,true);
  Serial.println(t_state);
  if (t_state == "1") {
    digitalWrite(LED, HIGH);          //LED ON
    ledState = "ON";                  //Feedback parameter
  }
  else  {
    digitalWrite(LED, LOW);           //LED OFF
    ledState = "OFF";                 //Feedback parameter
  }
  server.send(200, "text/plain", ledState); //Send web page
}
```

A kiszolgáló függvények sem túl bonyolultak...

ESP32_ledswitch_ajax futási eredménye

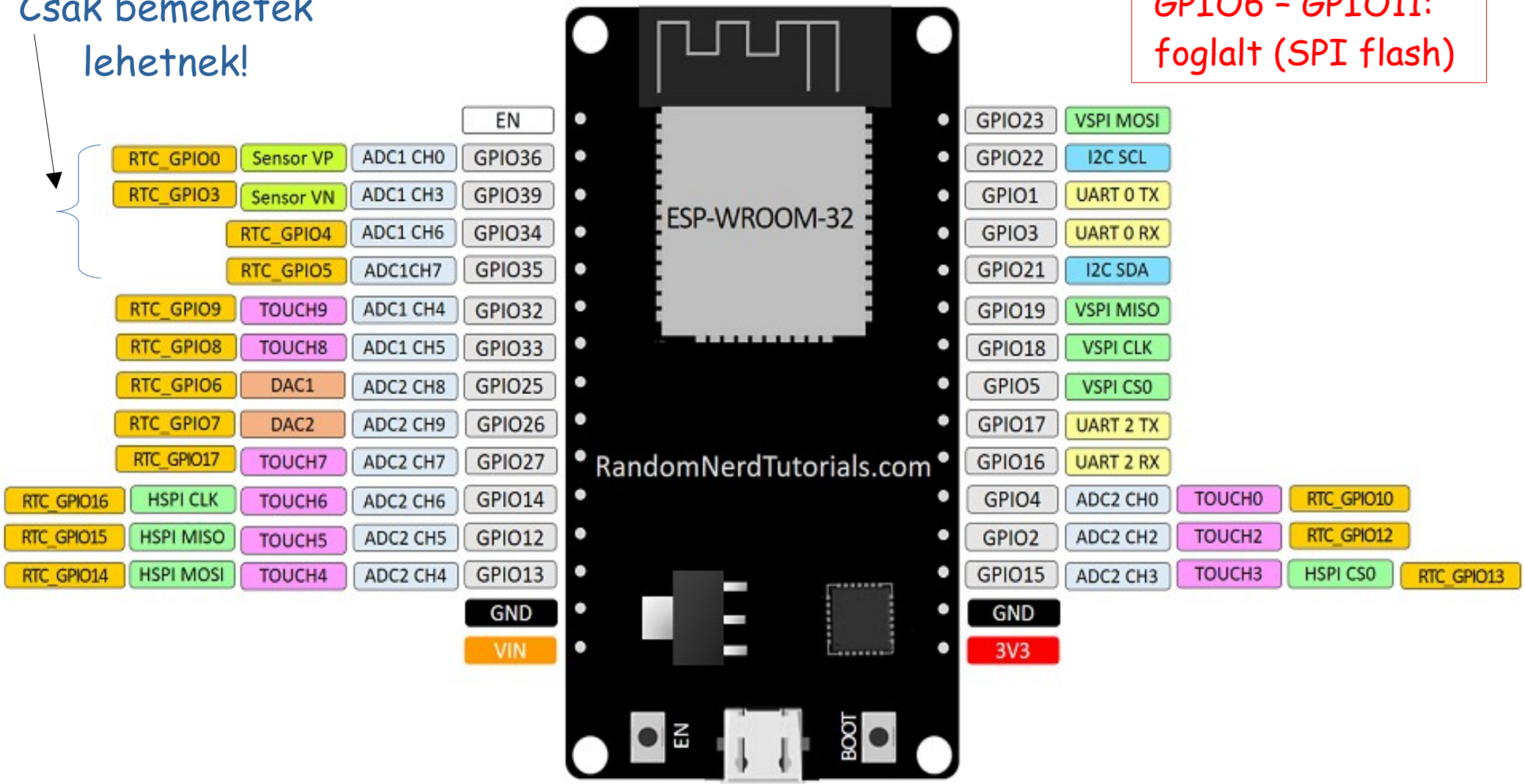
- Az alábbi ábrán a program futási eredménye látható
- A program az előző oldalakon nem részletezett stílus formázásokat is tartalmaz
- Az ADC lekérdezése automatikusan lefut két másodpercenként
- A LED állapota csak valamelyik nyomógomb lenyomásakor frissül



A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



- **Forrás:** randomnerdtutorials.com/getting-started-with-esp32/

Ellenállás színkódok