

Baixe o material da Oficina

acesse o github

<https://github.com/aterroso/ESPServidorWEB>

IoT - desvendando os mistérios de um sistema web embarcado num ESP32 - com demonstração prática

Prof. Anderson Terroso

Outubro/2025

Quem sou??

Portoalegrense, nascido em
15/06/74.



- Formação Acadêmica

- Formado em Engenharia Elétrica/Eletrônica (1992-1996)
- Mestre em Engenharia Elétrica – Sistemas Tolerantes a Falha (1997-1999)

- Experiência Profissional

- Prof. da PUCRS – março/2000 até o momento.
- Coordenador da Eng. De Computação – 2008 a 2010.
- Coordenador da Eng. Elétrica – 2010 a 2012 e de 2024 até o momento.
- Coordenador Acadêmica da Faculdade de Engenharia – 2012 a 2017.
- Lider do Núcleo de Articulação Acadêmica da Escola Politécnica – 2018 até o momento.
- Mais de 200 TCC ´s orientados.

28 anos atuando em projetos de P&D e Gerência de Projetos

Sumário

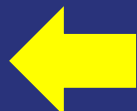
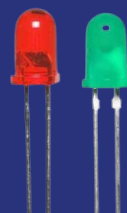
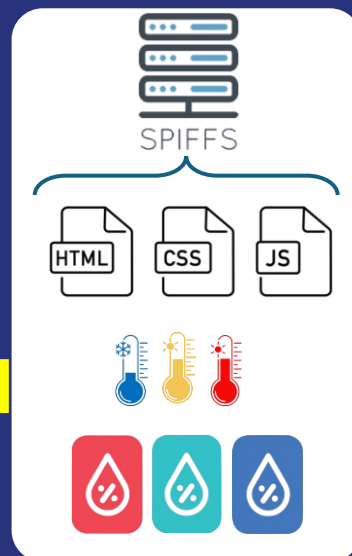
- Introdução
- Topologia do projeto
- Noções sobre programação WEB (Html, CSS, Bootstrap)
- ESP32 – o microcontrolador que transformou o mercado
- Integração do projeto Web + ESP32
- Demonstração prática – ESP32 + DHT11 (Sensor de Temperatura e Umidade)

Introdução....

- IoT = Internet das Coisas...



Requisitos de Projeto....



Servidor Web - ESP32

LED ON

LED OFF

Temperatura: {{TEMP}}



Umidade: {{UMID}}



Desenvolvido por Prof. Anderson Terroso

Outubro/2024

HTML - é uma Linguagem de Marcação de Hiper Texto, ou seja, é a forma utilizada para a construção de páginas web.

Não é uma linguagem de programação!

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

O **<DOCTYPE html>** não é uma tag HTML. Aqui é informado para o navegador que o HTML usado é o 5

É o elemento (HTML) onde toda a página será desenvolvida. Portanto, todas as outras tags devem estar aqui dentro.

É o elemento (HEAD) que compõe o topo da página ou o cabeçalho. O conteúdo não aparece no browser, mas contém instruções sobre seu conteúdo e comportamento.

O elemento body (corpo) compreende toda a área onde irá aparecer o conteúdo visual da página.

Comentários num documento HTML
<!-- aqui vem o comentário -->

Algumas tags usadas em um arquivo HTML

- `<title> </title>` → essa tag é usada dentro do elemento HEAD.
- `<script> </script>` → essa tag compreende um código escrito em javascript.
- `<style> </style>` → essa tag é usada para inserir CSS.
- `<link>` → essa tag é usada para adicionar links.
- `<h1> </h1><h6> </h6>` → essas tags são variações na escrita de títulos (h1 título principal, h2 subtítulo, etc..)
- `<div> </div>` → usado para dividir o conteúdo de uma página em blocos (divisões)
- `<p> </p>` → texto colocado sem formatação em um novo parágrafo (p)
- ` ` → deixa o texto em negrito
- `<i>..... </i>` → deixa o texto em itálico
- `
` → quebra de linha
- `<hr>` → quebra de linha e desenha uma barra horizontal. }

Algumas TAGS não tem fechamento `< ></>`, neste caso pode-se deixar apenas a tag sem a barra `
` ou no HTML5 se permite fazer o seguinte:
`
`

Inserido imagens e criando links

Inserindo imagens, sons e vídeos em página web

- a) `` ou ``
- b) ``
- c) ``

Como clicar numa figura e ser redirecionado para um site?

- a) ``
``
``

Estruturando o conteúdo na página web

Uma página web pode ser estruturada usando alguns elementos, tais como:

1º) Elemento **<div>**: este elemento separa a página web em divisões ou seções. Cada divisão permite agrupar diversos elementos HTML, como por exemplo: texto, imagens, formulários, etc..

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8"/>
```

```
<title>PUC-On-line</title>
```

```
</head>
```

```
<body>
```

```
<div id="banner">
```

```

```

```
</div>
```

```
<div id="menu">
```

```

```

```
</div>
```

```
</body>
```

```
</html>
```

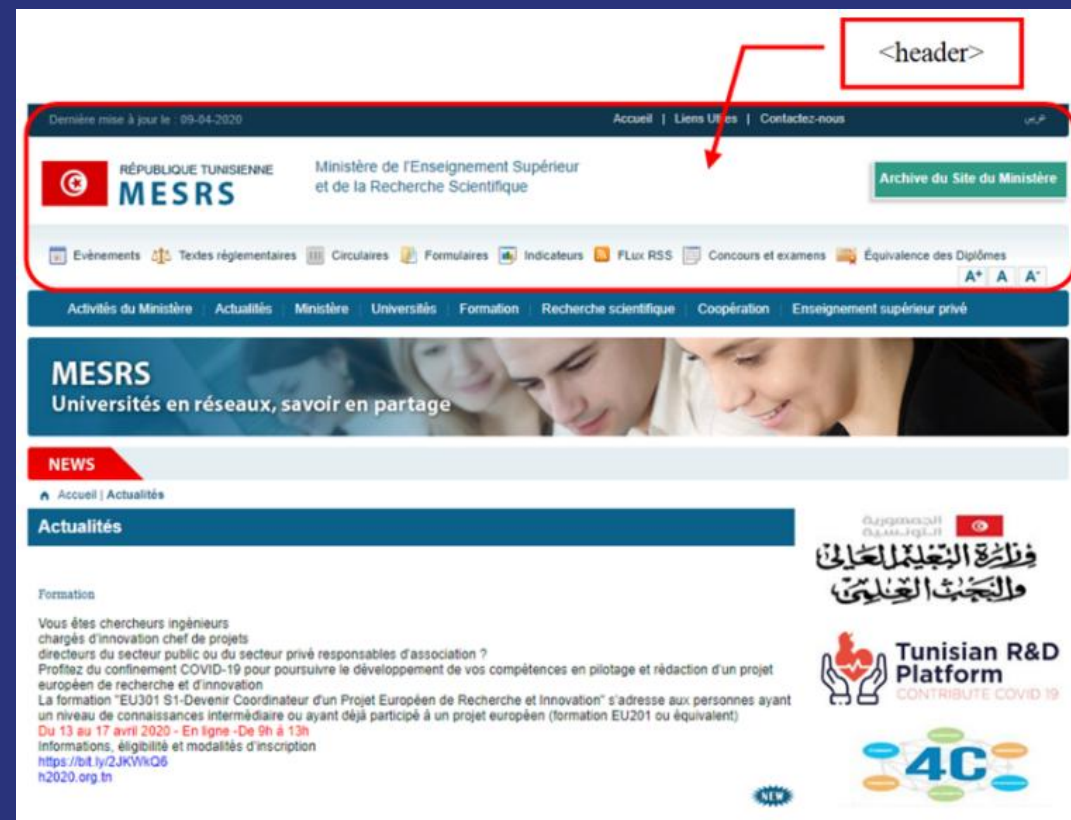
Continuação....

2º) Elemento ``: semelhante ao div, porém é uma forma de atribuir atributos a um pequeno grupo de elementos.

`<p>`Este curso de `` HTML, CSS, Javascript `` vão lhe fazer um programador full-stack.`</p>`

3º) Tag `<header>`: diferentemente do head, o header (cabeçalho) contém normalmente logo, slogan, banner de uma página web.

Esta tag é dupla `<header>` `</header>`



Continuação.....

- 4º) Tag `<footer>`: o footer (rodapé) geralmente está localizado no fim da página web. Contém informações avisos legais e/ou autoria e/ou links. Dupla tag `<footer></footer>`.



Tag's utilizadas na formatação de texto

`<p>` novo
parágrafo

`<blockquote>`

``
negrito

`<i>` itálico

`` itálico

`` negrito

`<small>` texto
menor tamanho

`<abbr>`
significado
abreviatura

`<mark>`
destacar um
texto

`<var>` usado
com variáveis

`<kbd>` tecla de
atalho

`<time>` hora

`<sub>` subscrito
(índice)

`<sup>`
sobrescrito
(exp)

`<pre>`
pre-formatado

`<hr>` linha
divisória

`
` (break)
quebra de linha

Estruturando o texto em formato de listas

Lista ordenada - tag `` - ordered list

`<p> Nesta disciplina iremos aprender:</p>`

``

` HTML5`

` CSS3`

` Javascript`

``

Nesta disciplina iremos aprender:

1. HTML5

2. CSS3

3. Javascript

A tag de lista ordenada `` pode ter um atributo para definir o que será usado para ordenar a lista. O atributo é `type`.

`<ol type="a">` ou `<ol type="A">`: será usado letras.

`<ol type="i">` `<ol type="I">`: será usado números romanos.

Lista não ordenada - tag `` - unordered list

`<p> Nesta disciplina iremos aprender:</p>`

``

` HTML5`

` CSS3`

` Javascript`

``

Nesta disciplina iremos aprender:

• HTML5

• CSS3

• Javascript

Introdução ao CSS (Cascading Style Sheet – Folha de Estilos em Cascata).

- **Objetivo:** o CSS é usado para formatar páginas HTML, modificando a aparência da página.
- **Formas de declarar a estilização:**

1º) deseja-se criar um estilo personalizado para uma figura ou texto, pode simplesmente fazer o seguinte:

```
<style>
```

```
  .css_texto  
  {
```

```
    font-weight : bold;  
    color: blue;
```

```
  }
```

```
</style>
```

```
.....
```

```
<p class="css_texto">Configura texto</p>
```

2º) Pode-se colocar a estilização dentro da própria tag.

```
<p class="color:blue">Semana Acadêmica.</p>
```

**Não esqueça de colocar ponto (.)
antes do nome do estilo**

Obs.: Aqui não vai o ponto (.)

A estilização na própria tag.

2º) pode utilizar o próprio nome da tag, assim não precisa .XXX, mas todos tags serão personalizados com o mesmo estilo. Isso foi feito no HTML da tabela.

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>Document</title>
```

```
  <style>
```

```
    caption
```

```
    {
```

```
      background-color:royalblue;
```

```
      color:white;
```

```
      font-weight: bold;
```

```
    }
```

```
  </style>
```

```
</head>
```

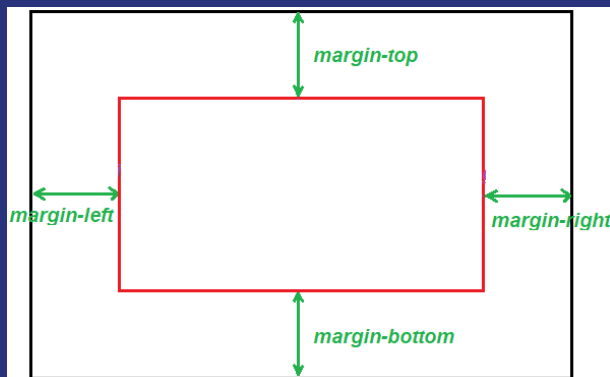
Se usar a própria tag, não vai .

Propriedades de fontes

Propriedade	Descrição	Valores Possíveis
color	Configura a cor de um texto	<i>color</i>
letter-spacing	Aumenta ou diminui o espaço entre os caracteres	normal <i>length</i>
text-align	Alinha o texto num elemento	left right center justify
text-decoration	Adiciona decoração ao texto	none underline overline line-through blink

Propriedade	Descrição	Valores
font-family	Uma lista priorizada de nomes de famílias de fontes e/ou nomes de famílias genéricos para um elemento	<i>family-name</i> <i>generic-family</i>
font-size	Especifica o tamanho de uma fonte	xx-small x-small small <i>length</i> % valor em pixel (ex.: 16px)
font-style	Especifica o estilo da fonte	normal italic oblique
font-weight	Especifica o peso de uma fonte	normal bold bolder (mais negrito) lighter (mais claro) 100 a 900

Configurando um elemento em relação as bordas



Propriedade	Descrição	Valores
margin	Uma propriedade estenográfica para especificar as propriedades das margens em uma declaração	<i>margin-top</i> <i>margin-right</i> <i>margin-bottom</i> <i>margin-left</i>
margin-bottom	Especifica a margem inferior de um elemento	Auto / <i>length</i> / %
margin-left	Especifica a margem esquerda de um elemento	Auto / <i>length</i> / %
margin-right	Especifica a margem direita de um elemento	Auto / <i>length</i> / %
margin-top	Especifica a margem superior de um elemento	Auto / <i>length</i> / %

ESP32

Microcontrolador:

- Processador: Xtensa LX6 dual-core de 32 bits
- Clock: Até 240 MHz
- Memória: 32 KB de cache de nível 1
- Memória: 16 KB de cache de nível 2

Conectividade sem fio:

- Wi-Fi: IEEE 802.11 b/g/n
- Bluetooth: Bluetooth 4.2

O que dá para fazer com ESP32?

O que você pode fazer com a ESP32 depende apenas da sua imaginação e criatividade. Você pode criar, por exemplo:

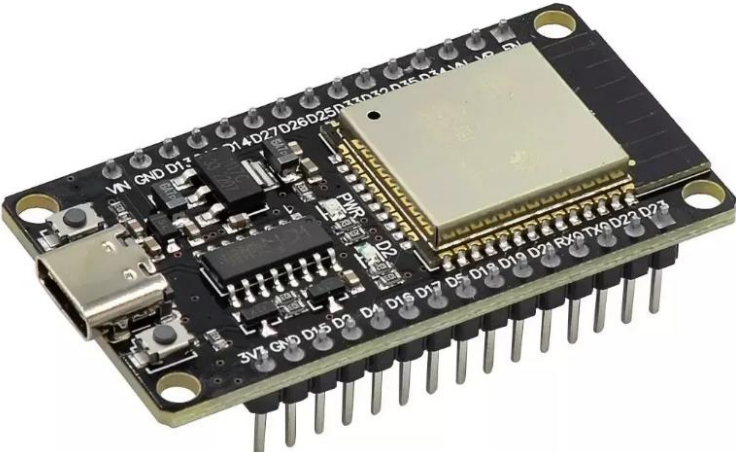
- despertador inteligente que liga a luz da sua sala de estar quando tocar;
 - sensor de fumaça que envia um alerta para o seu telefone se detectar fumaça;
 - dispositivo de rastreamento que pode ser usado para rastrear seus bens;
 - câmera de segurança que pode ser usada para monitorar sua casa;
 - jogo de arcade para jogar na sua TV.
-
- **C:** pode ser usado para criar projetos de alto desempenho, como robótica e automação industrial;
 - **C++:** pode ser usado para criar projetos complexos, como sistemas de controle e processamento de dados;
 - **MicroPython:** pode ser usado para criar projetos rápidos e fáceis, como dispositivos IoT e projetos de prototipagem;
 - **Lua:** pode ser usada para criar projetos criativos, como jogos e aplicações de entretenimento;
 - **JavaScript:** pode ser usado para criar projetos que se conectam à Internet, como dispositivos IoT e aplicações de monitoramento.

ESP32 x Kit Arduino UNO

Característica	ESP32	Arduino Uno
Processador	Dual-core de 32 bits	8 bits
Clock	Até 240 MHz	Até 16 MHz
Conectividade sem fio	Wi-Fi e Bluetooth	Não
Memória	520 KB flash, 80 KB RAM	32 KB flash, 2 KB RAM
Periféricos	Portas GPIO, UART, I2C, SPI	Portas GPIO, UART, I2C



Valores de mercado....(base ML)



Novo | +50 vendidos

Microcontrolador Esp32 Dev-kit Usb-c - Compatível Arduino Ide

5.0 ★★★★★ (2)

R\$ 42³⁹
em 12x R\$ 4¹¹

[Ver os meios de pagamento](#)

Chegará entre sexta-feira e sábado
[Mais formas de entrega](#)

Retire entre sexta-feira e sábado em uma agência Mercado Livre
[Ver no mapa](#)

Último disponível!



Novo | +25 vendidos

Placa Uno R3 Dip Com Cabo Usb - Compatível Com Arduino Uno

5.0 ★★★★★ (4)

R\$ 69
em 12x R\$ 6⁷¹

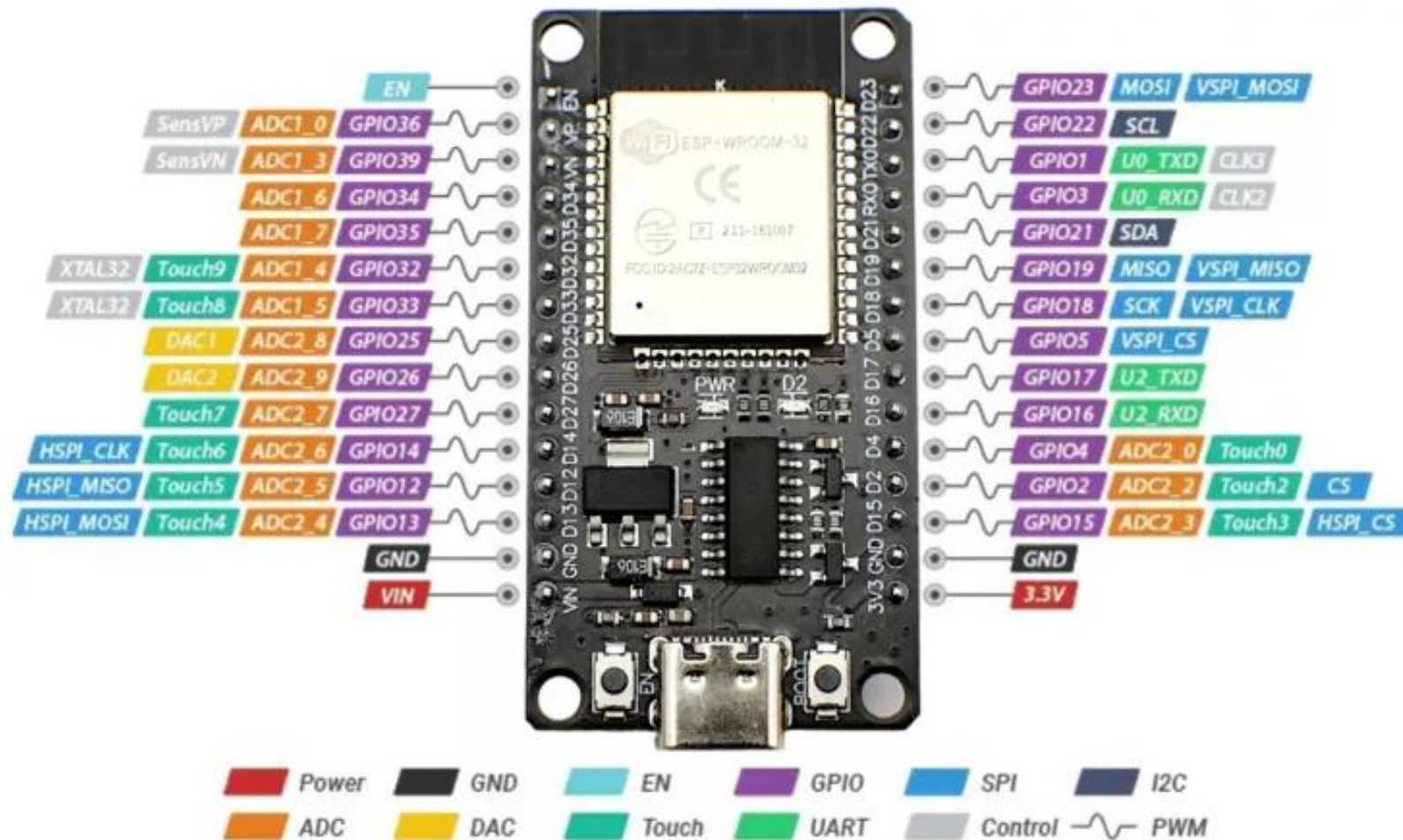
[Ver os meios de pagamento](#)

Chegará sexta-feira
[Mais formas de entrega](#)

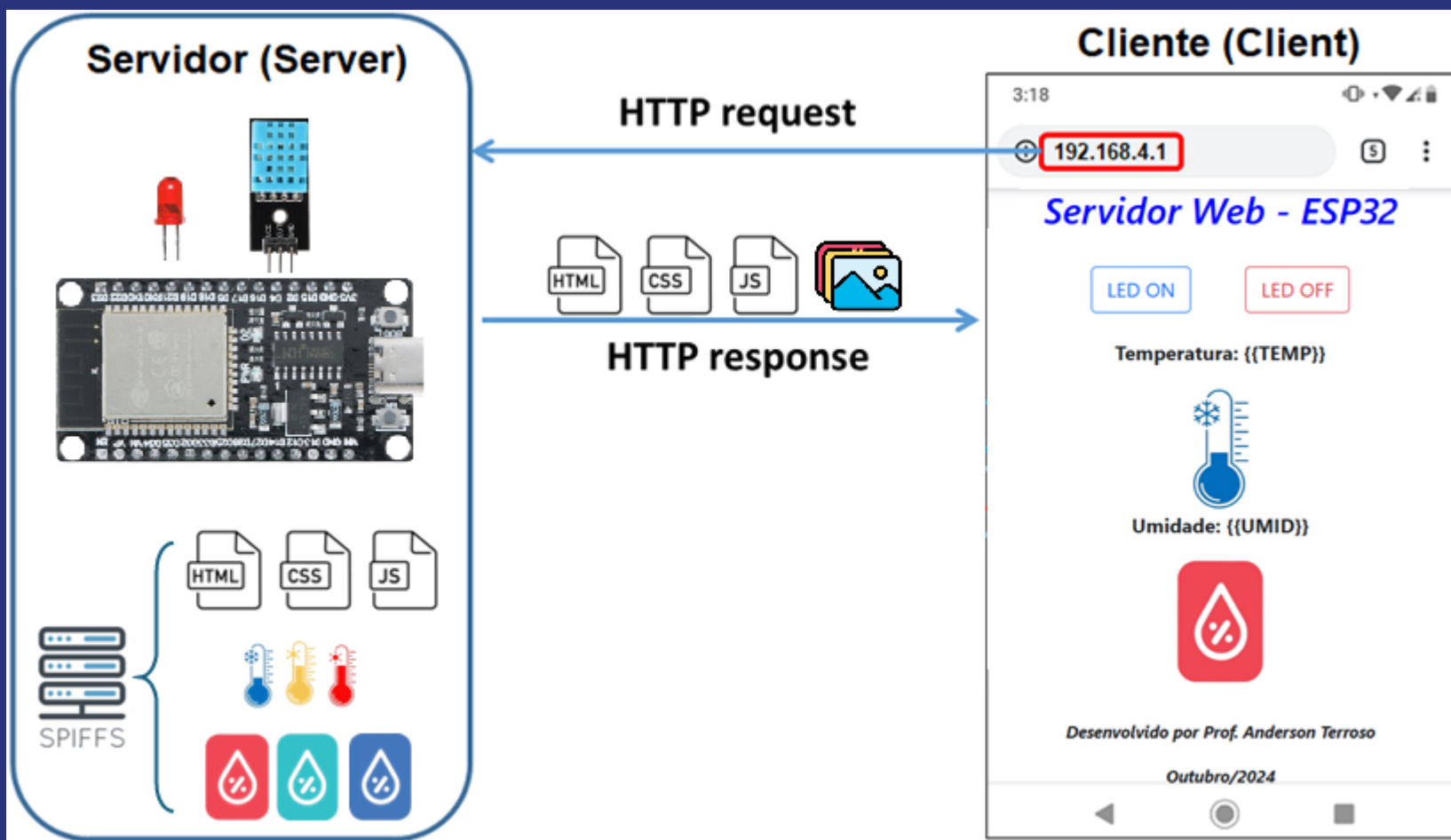
Retire entre sexta-feira e sábado em uma agência Mercado Livre
[Ver no mapa](#)

Estoque disponível

Pinagem do ESP32

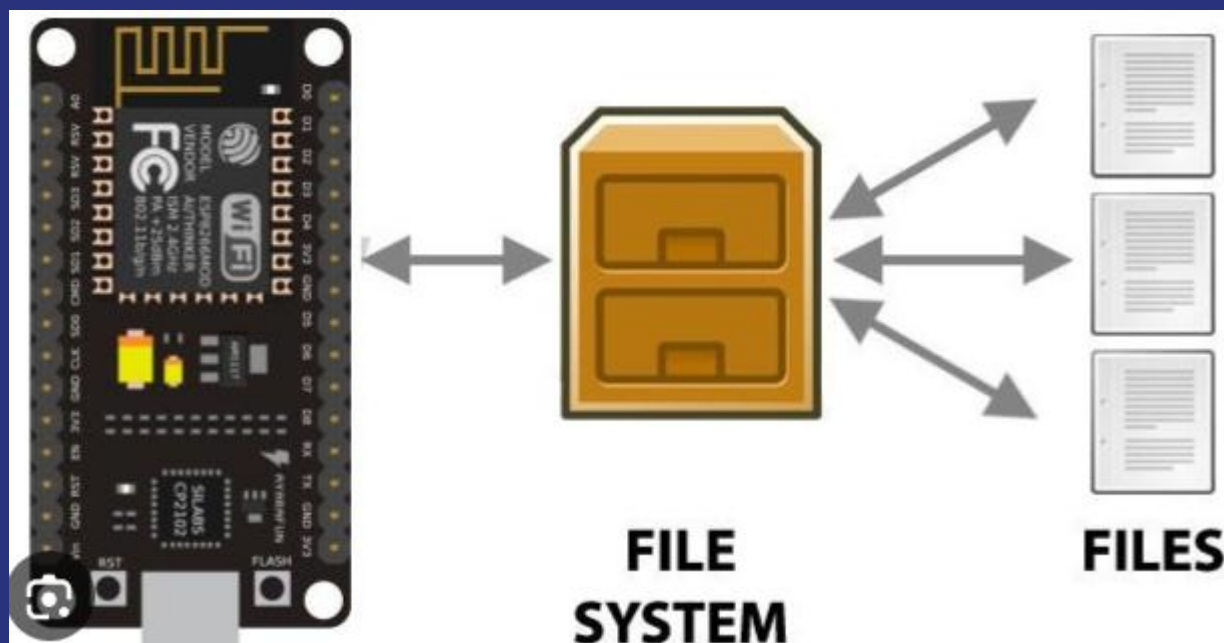


Desenvolvimento do webserver no ESP32



O QUE É SPIFFS (*SPI FLASH FILE SYSTEM*)?

- SPIFFS é um sistema de arquivos que permite acessar a memória flash de dispositivos como o ESP8266/32. Ele funciona de forma semelhante a um sistema de arquivos de computador, mas com algumas limitações e de forma mais simples.



Exemplo 1: Modelo simples de uma página web no próprio código em C

```
/****** MODO STATION (CONECTA A REDE WIFI)******/
```

```
#include <Arduino.h>
```

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

```
#include <WebServer.h>
```

```
const char* ssid = "motog8";
```

```
const char* password = "0123456789";
```

```
WebServer server(80);
```

```
#include <DHT.h>
```

```
#include <Adafruit_Sensor.h>
```

```
#define pino_DHT 4
```

```
#define tipo_DHT DHT11
```

```
DHT dht(pino_DHT, tipo_DHT);
```

```
void setup()
```

```
{
```

```
  Serial.begin(115200);
```

```
  dht.begin();
```

```
  Serial.println("Conectando.....");
```

```
  WiFi.begin(ssid, password);
```

```
while (WiFi.status() != WL_CONNECTED)
```

```
{
```

```
  delay(1000);
```

```
  Serial.print(".");
```

```
}
```

```
Serial.println("");
```

```
Serial.println("WiFi conectado com sucesso");
```

```
Serial.print("IP do ESP32 eh: ");
```

```
Serial.println(WiFi.localIP());
```

```
server.on("/", funcao_chama_paginaHtml);
```

```
server.begin();
```

```
delay(100);
```

```
}
```

```
void loop()
```

```
{
```

```
  server.handleClient();
```

```
}
```

Continuação.....

```
void funcao_chama_paginaHtml()
{
    float umid = dht.readHumidity();
    float temp = dht.readTemperature();

    String HTML = "<!DOCTYPE html><html>";
    HTML += "<head>";
    HTML += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
    HTML += "<meta http-equiv=\"refresh\" content=\"5\">";
    HTML += "<title>Monitoramento de Temperatura e Umidade</title>";
    HTML += "</head>";
    HTML += "<body><h1><center>Monitoramento de Temperatura e Umidade</center></h1>";
    HTML += "<p><h3>A temperatura atual é de: ";
    HTML += (float)temp;
    HTML += " graus</h3></p>";
    HTML += "<p><h3>A umidade relativa do ar é de: ";
    HTML += (float)umid;
    HTML += "%.</h3></p>";
    HTML += "</body></html>";

    server.send(200, "text/html", HTML);
}
```

Exemplo 2: Colocando imagens.....

```
/****** MODO STATION (CONECTA A REDE WIFI) *****/  
#include <Arduino.h>  
  
#include <DHT.h>  
#include <Adafruit_Sensor.h>  
  
#define pino_DHT 4  
#define tipo_DHT DHT11  
  
DHT dht(pino_DHT, tipo_DHT);  
  
#include <WiFi.h>  
#include <WebServer.h>  
  
const char* ssid = "motog8"; // Enter your SSID here  
const char* password = "0123456789"; //Enter your Password here  
  
WebServer server(80); // Object of WebServer(HTTP port, 80 is default)  
  
#include <SPIFFS.h>  
  
void funcao_chama_paginaHtml();  
void func_imgTemp();  
void func_imgUmid();
```

```
void setup()  
{  
  dht.begin();  
  Serial.begin(115200);  
  
  if (!SPIFFS.begin(true)) {  
    Serial.println("Falha ao montar o SPIFFS");  
    return;  
  }  
  Serial.println("Conectando.....");  
  WiFi.begin(ssid, password);  
  
  while (WiFi.status() != WL_CONNECTED)  
  {  
    delay(1000);  
    Serial.print(".");  
  }  
  Serial.println("");  
  Serial.println("WiFi conectado com sucesso");  
  Serial.print("IP do ESP32 eh: ");  
  Serial.println(WiFi.localIP());  
  
  server.on("/", funcao_chama_paginaHtml);  
  server.on("/normalTemp.png", func_imgTemp);  
  server.on("/normalUmid.png", func_imgUmid);  
  
  server.begin();  
  delay(100);  
}
```

Continuação....

```
void loop()
{
  server.handleClient();
}
```

```
void funcao_chama_paginaHtml()
{
  float umid = dht.readHumidity();
  float temp = dht.readTemperature();
```

```
  String HTML = "<!DOCTYPE html><html>";
  HTML += "<head>";
  HTML += "<meta charset=\"UTF-8\">";
  HTML += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
  HTML += "<meta http-equiv=\"refresh\" content=\"5\">"; // Reload a cada 5 segundos
  HTML += "<title>Monitoramento de Temperatura e Umidade</title>";
  HTML += "</head>";
  HTML += "<body><h1><center>Monitoramento de Temperatura e Umidade</center></h1>";
  HTML += "<p><h3>A temperatura atual é de: ";
  HTML += (float)temp;
  HTML += " graus</h3></p>";
  HTML += "<center><img src=\"/normalTemp.png\" alt=\"Imagem\" style=\"width:40px;height:80px;\"></center>";
  HTML += "<p><h3>A umidade relativa do ar é de: ";
  HTML += (float)umid;
  HTML += "%.</h3></p>";
  HTML += "<center><img src=\"/normalUmid.png\" alt=\"Imagem\" style=\"width:40px;height:80px;\"></center>";
  HTML += "</body></html>";
  server.send(200, "text/html", HTML);
}
```

Continuação....

```
void func_imgTemp() {  
    File file = SPIFFS.open("/normalTemp.png", "r");  
    if (!file) {  
        Serial.println("Falha ao abrir a imagem");  
        return;  
    }  
    server.streamFile(file, "image/png");  
    file.close();  
}
```

```
void func_imgUmid() {  
    File file = SPIFFS.open("/normalUmid.png", "r");  
    if (!file) {  
        Serial.println("Falha ao abrir a imagem");  
        return;  
    }  
    server.streamFile(file, "image/png");  
    file.close();  
}
```


Exemplo 3: Código do ESP32 + HTML + CSS

```
#include <Arduino.h>
```

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

```
#include <AsyncTCP.h>
```

```
#include <ESPAsyncWebServer.h>
```

```
#include <FS.h>
```

```
#include <SPIFFS.h>
```

```
#include <DHT.h>
```

```
#include <Adafruit_Sensor.h>
```

```
#define pino_DHT 4
```

```
#define tipo_DHT DHT11
```

```
DHT dht(pino_DHT, tipo_DHT);
```

```
const char* ssid = "motog8";
```

```
const char* password = "0123456789";
```

```
#define led 2
```

```
float temp, umid;
```

```
AsyncWebServer server(80);
```

**server é uma instancição da
classe AsyncWebServer**

**ESP8266WebServer => para
ESP8266
WebServer => para ESP32
AsyncWebServer => ESP8266 ou
ESP32**

**A porta 80 é usada para HTTP e a
443 é usada para HTTPS. A porta
8080 é uma variação da porta 80.**

```
String processor(const String &aux)
```

```
{  
  if(aux == "TEMP"){  
    temp = dht.readTemperature();  
    return String(temp);  
  }  
  else if(aux == "UMID"){  
    umid = dht.readHumidity();  
    return String(umid);  
  }  
  return String();  
}
```

```
void setup(){  
  if(!SPIFFS.begin()){  
    Serial.println("Erro inicialização SPIFFS");  
    return;  
  }  
  dht.begin();  
  
  Serial.begin(115200);  
  
  pinMode(led, OUTPUT);  
  Serial.println(dht.readTemperature());  
  Serial.println(dht.readHumidity());
```

Continuação....

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando ao WiFi..");
}
Serial.print("Conectado a rede wifi e o IP=");
Serial.println(WiFi.localIP());

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", "text/html", false, processor);
});

server.on("/estilos.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/estilos.css", "text/css");
});

server.on("/bootstrap.min.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/bootstrap.min.css", "text/css");
});

server.on("/on", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(led, HIGH);
    flag_led = true;
    request->send(SPIFFS, "/index.html", "text/html", false, processor);
});
```

A sintaxe `[](AsyncWebServerRequest *request)` é uma **lambda function** em C++.

As funções lambda são uma maneira de definir funções anônimas (ou seja, funções que não têm um nome) diretamente dentro do código.

O que significa cada item:

`[]`: Este é a **captura** da lambda. Aqui, são capturadas variáveis do contexto externo.

(AsyncWebServerRequest *request): Este é o **parâmetro** da lambda. Neste caso, a lambda recebe um ponteiro para um objeto do tipo AsyncWebServerRequest. Esse objeto contém informações sobre a requisição HTTP que foi feita, como parâmetros da URL, cabeçalhos, etc.

Continuação.....

```
server.on("/off", HTTP_GET, [](AsyncWebServerRequest *request){  
    digitalWrite(led, LOW);  
    flag_led = false;  
    request->send(SPIFFS, "/index.html", "text/html", false, processor);  
});
```

```
server.on("/normalTemp.png", HTTP_GET, [](AsyncWebServerRequest *request){  
    request->send(SPIFFS, "/normalTemp.png", "image/png");  
});
```

```
server.on("/normalUmid.png", HTTP_GET, [](AsyncWebServerRequest *request){  
    request->send(SPIFFS, "/normalUmid.png", "image/png");  
});
```

```
server.begin();  
}  
void loop()  
{
```

**AsyncWebServer,
ESP8266WebServer ou tem
vários métodos, são eles:
begin(), on(), send(),
handleClient() e
onNotFound().**

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" type="text/css" href="estilos.css">
  <link rel="stylesheet" type="text/css" href="bootstrap.min.css">
  <meta http-equiv="refresh" content="5">
  <title>Semana Acadêmica</title>
</head>
<body>
  <div>
    <h1>Servidor Web - ESP32</h1>
  </div>
  <div>
    <a href="/on"><button class="btn btn-outline-primary cssbtn">LED ON</button></a>
    <a href="/off"><button class="btn btn-outline-danger cssbtn">LED OFF</button></a>
  </div>
  <div>
    <p>Temperatura: <b> %TEMP% </b><sup>&deg;C</sup></p>
    
  </div>
  <div>
    <p>Umididade: <b> %UMID% </b><sup>&#37;</sup></p>
    
  </div>
  <div class="cssrodape">
    <p>Desenvolvido por Prof. Anderson Terroso</p>
    <p>Outubro/2024</p>
  </div>
</body>
</html>
```

CSS

```
html {  
  font-family: Arial;  
  text-align: center;  
  font-size: 1.5rem;  
}
```

```
h1{  
  color:blue;  
  font-weight: bolder;  
  font-style: italic;  
  font-size:60px;  
}
```

```
p{  
  font-weight: bolder;  
  font-size: 45px;  
}
```

```
.imgtemp{  
  width: 120px;  
  height: 150px;  
}
```

```
.imgumid{  
  width: 120px;  
  height: 150px;  
}
```

```
.cssbtn{  
  margin: 20px;/* distância entre os  
botões */  
  width: 20%;  
  height: 100px;  
  font-size: 30px;  
  font-weight: bold;  
}
```

```
.cssrodape{  
  margin-top: 30px;  
  font-style: italic;  
  font-size: smaller;  
  font-size: 1.5rem;  
}
```


Detalhamento da Função request->send()

request->send(SPIFFS, "/index.html", "text/html", false, processor);

1.request:

•Este é um ponteiro para o objeto AsyncWebServerRequest, que representa a requisição HTTP recebida. A partir desse objeto, você pode acessar informações sobre a requisição, como os parâmetros, cabeçalhos e o cliente que fez a requisição.

2.send():

•Este é o método chamado no objeto request. Ele é utilizado para enviar uma resposta de volta ao cliente que fez a requisição.

3.SPIFFS:

•Este é o sistema de arquivos SPIFFS (SPI Flash File System) que você configurou no ESP32. Ele permite que você armazene e acesse arquivos no flash do dispositivo, como HTML, CSS, JavaScript, imagens, etc.

4."/index.html":

•Este é o caminho do arquivo que você deseja enviar. Neste caso, é o arquivo HTML que você quer que o cliente receba. O caminho é relativo ao sistema de arquivos SPIFFS.

5."text/html":

•Este é o tipo de conteúdo (MIME type) que você está enviando. Ele informa ao navegador que o conteúdo retornado é um documento HTML. Isso é importante para que o navegador possa interpretar corretamente o conteúdo que está recebendo.

6.false:

•Este parâmetro especifica se a resposta deve ser mantida viva (keep-alive) ou não. Passar false significa que a conexão será fechada após a resposta. Isso é útil em muitos casos onde não é necessário manter a conexão aberta.

7.processor:

•Este é um ponteiro para uma função que você definiu para processar variáveis no HTML. Quando você usa placeholders no HTML (como {{NOME}}), essa função é chamada para substituir esses placeholders pelos valores reais.

Os métodos de ESP8266WebServer, WebServer ou AsyncWebServer

ESP8266 → ESP8266WebServer **server**(80);

ESP32 → WebServer **server**(80);

Ambos → AsyncWebServer **server**(80);

server



***.begin()** → método que inicializa o servidor, esse método normalmente é chamado após todas as configurações tenham sido realizadas.

***.send()** → método que envia a resposta ao cliente, neste caso quem acessa o server recebe a resposta enviada por esse método.

***.on()** → método para acessar a URL quando é realizado as requisições HTTP.

Respostas básicas de um servidor web assíncrono.

```
request->send( a, b, c);
```

```
request->send(200, "text/plain", "Ola mundo!");
```

➤ **a → é o código HTTP de status de resposta. A seguir alguns códigos:**

Respostas Informativas (100 – 199)

Respostas bem-sucedidas (200 – 299) – normalmente usamos 200 para indicar OK

Mensagens de redirecionamento (300 – 399)

Respostas de erro do cliente (400 – 499) – normalmente 404 para NotFound

Respostas de erro do servidor (500 – 599)

➤ **b → tipo de mídia da internet. A seguir as mais comuns:**

text/plain

image/png

image/gif

image/jpeg

audio/mpeg

video/mpeg

➤ **c → o conteúdo da resposta.**

Diferenças entre libs: WebServer x AsyncWebServer

Característica	WebServer	AsyncWebServer
Tipo	Síncrono	Assíncrono
Requer <code>server.handleClient()</code>	✓ Sim	✗ Não
Reatividade	Menor, uma conexão por vez	Alta, múltiplas conexões paralelas
Complexidade	Mais simples	Mais robusto, mas requer mais libs
Ideal para	Projetos simples e diretos	Dashboards, WebSockets, alta carga

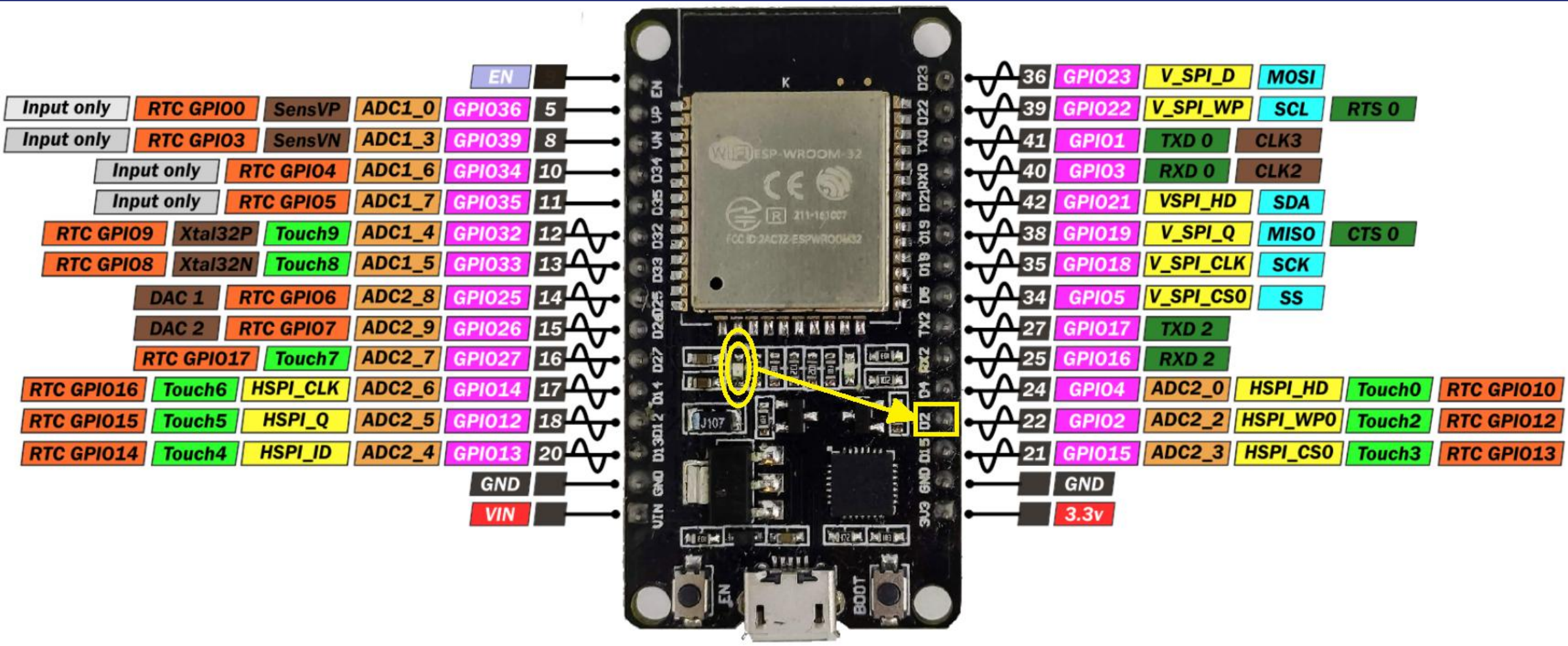
WebServer → Isso significa que a cada iteração do `loop()`, o ESP32 verifica se chegou uma nova requisição HTTP e a processa. **Se você não colocar o `server.handleClient()`, ele não responde às requisições.**

AsyncWebServer → Este é o **servidor assíncrono**, fornecido pela biblioteca **ESPAsyncWebServer**, que usa recursos internos do ESP32 para tratar conexões **em paralelo e em segundo plano** (eventos), sem depender do `loop()`.

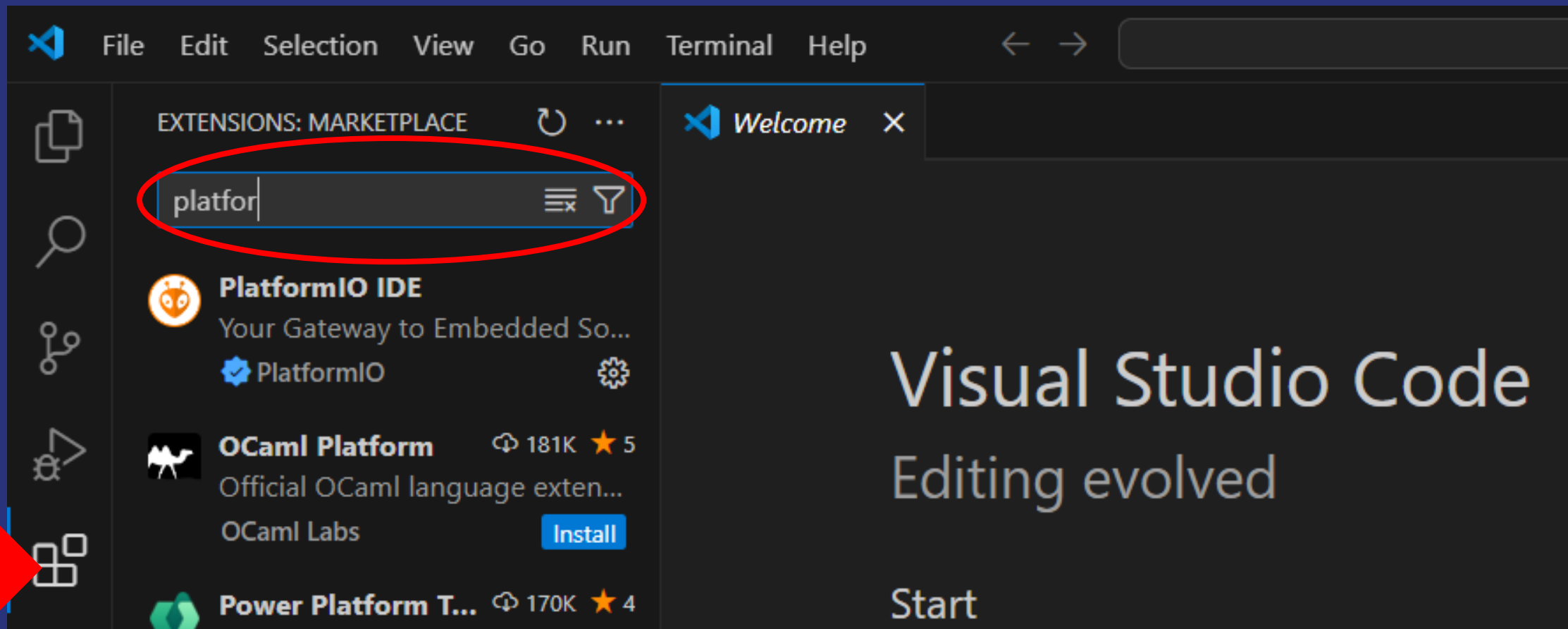
Uso do vscode com o esp32

compilar, gravar, spiffs, etc...

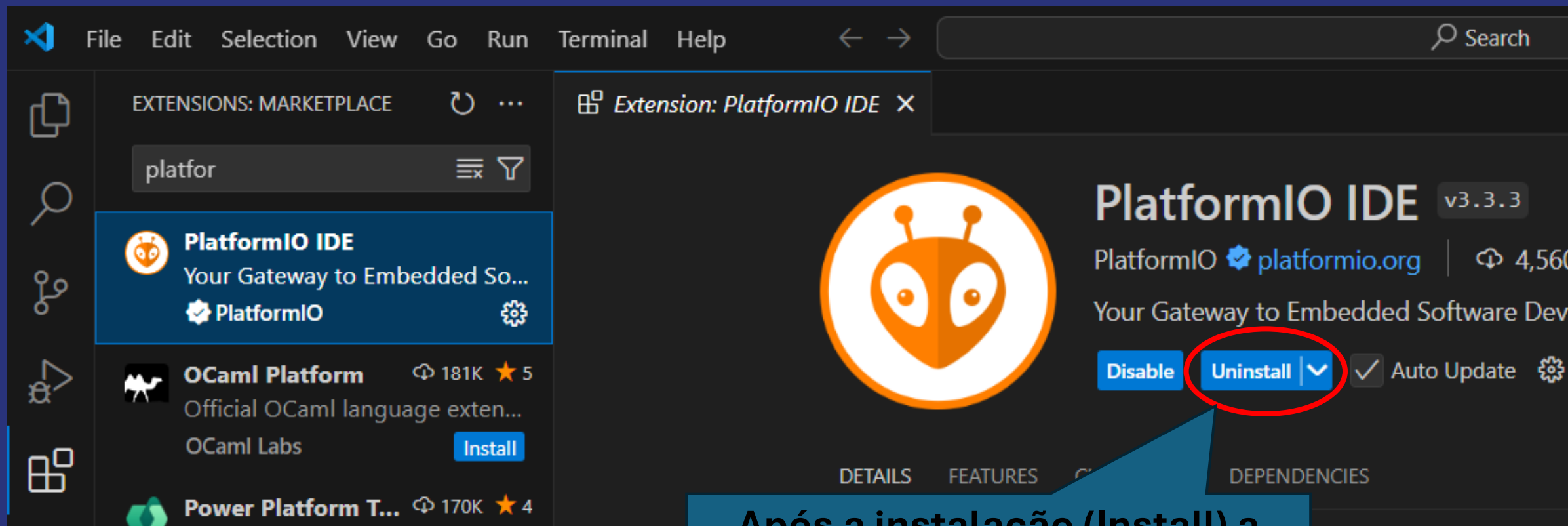
Placa ESP32



Instalar a extensão platformIO IDE



Basta instalar a extensão....



Após a instalação (Install) a
opção mudará para Uninstall

Como habilitar a aba terminal serial no vscode?

Instale a extensão terminal serial

1º

2º

The screenshot shows the Visual Studio Code interface. On the left, the 'EXTENSIONS: MARKETPLACE' sidebar is open. A search bar contains 'serial mo'. The 'Serial Monitor' extension by Microsoft is highlighted with a red dashed box. Below it, other extensions like 'Web and Desktop Ser...', 'Close all opened s...', and 'Json to Dart Model' are visible. On the right, the 'Serial Monitor' extension details are shown, including its version (v0.12.0), publisher (Microsoft), and a description: 'Send and receive text from serial ports.' The extension is currently installed, as indicated by the 'Uninstall' button. The background of the slide is dark blue with a light blue gradient at the top.

File Edit Selection View Go Run Terminal Help

WebServer_HTML_CSS_AGO24

EXTENSIONS: MARKETPLACE

serial mo

Serial Monitor 140ms
Send and receive text from seri...
Microsoft

Web and Desktop Ser... 746
Views serial port output on des...
Eclipse CDT Cloud

Close all opened s... 119 ★ 5
Closes all open serial ports in V...
pietro.cz

Json to Dart Model 188K ★ 5
Extension convert Json to Dart ...
hirantha

Serial Monitor v0.12.0 Preview
Microsoft microsoft.com | 1,109,603
Send and receive text from serial ports.
Disable Uninstall Switch to Pre-Release Version

DETAILS FEATURES CHANGELOG

Serial Monitor

The Serial Monitor extension provides a serial monitor to view output from as we
This is often useful when testing or debugging programs on embedded devices.

Agora basta conectar a placa, selecionar a porta serial (port) , baud rate e clicar em start monitoring. Quando for fazer uma nova gravação do firmware, lembre-se de fechar a conexão (close monitoring)

```
9  #define led 2
10
11 const char* ssid = "motog8"; // Enter your SSID here
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

SERIAL MONITOR

+ Open an additional monitor

Monitor Mode

Serial

View Mode

Text

Port

NA - No port selected

Baud rate

115200

Line ending

None

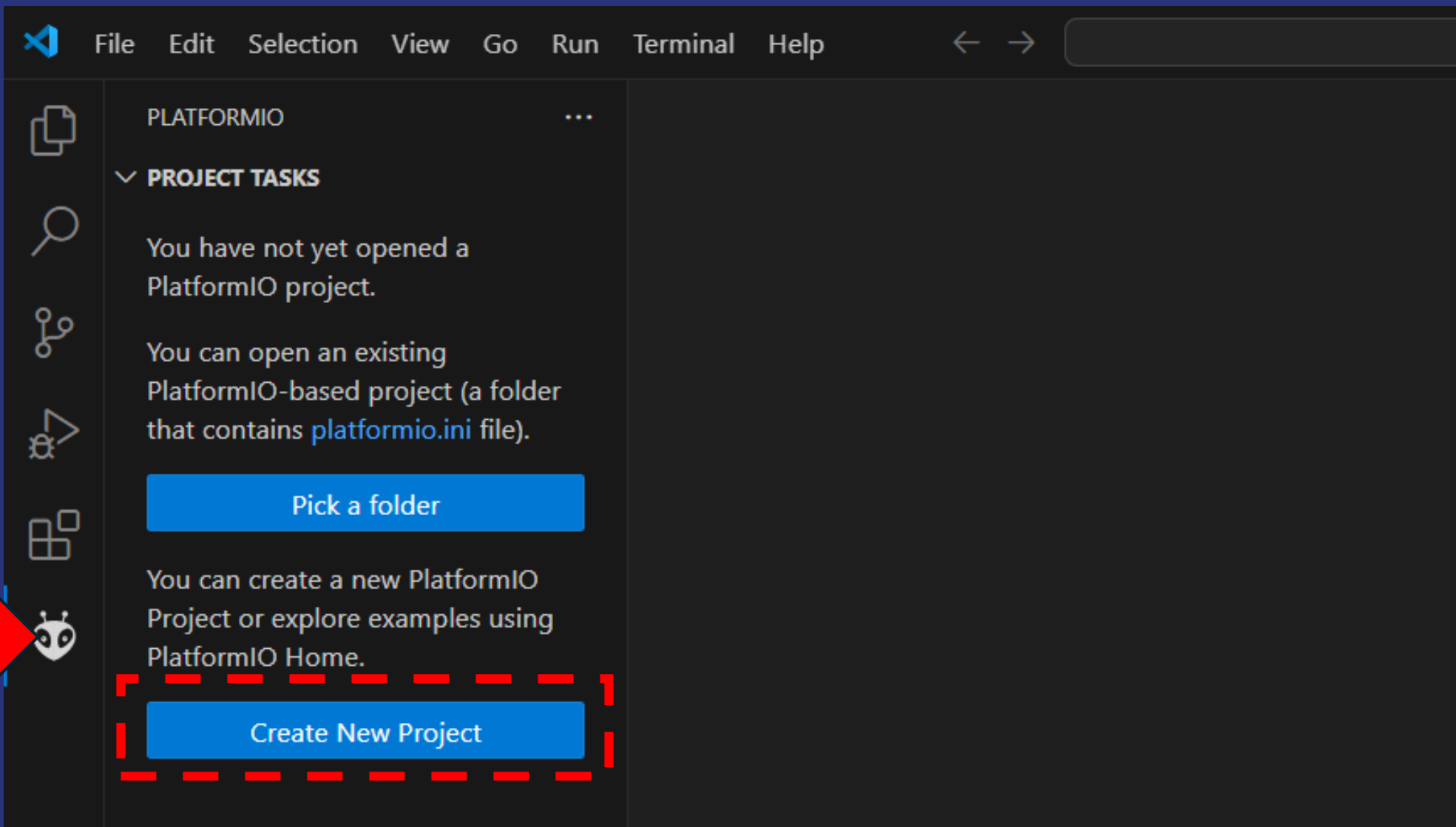
▶ Start Monitoring

Type in a message to send to the serial port.

Send as Text


Send Ctrl + C

Para criar um novo projeto.....





Clique em “new Project”....


PIO Home X




< > ↗

 Follow Us







Home

Projects

Inspect


Libraries

Boards

Platforms

Devices


Welcome to PlatformIO





Core 6.1.15 · Home 3.4.4

Quick Access

+ New Project

 Import Arduino Project


 Open Project

 Project Examples


Recent News

STATE MACHINE

"State Machine" is a set of all states (equivalence class of past histories of a system) plus the rules for changing from one state to another (state transitions)




PlatformIO Labs · 2d



PlatformIO Labs · 5d

Variables & Pointers



PlatformIO Labs · 1w

Defina o nome do projeto, a placa e o framework.

Nome do
projeto.

Selecione a
placa.

Framework
= Arduino.

Project Wizard

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name: Project name

Board: Select a board (1523 available)

Framework:

Location: ☒ Use default location ?

Cancel

Finish

Dados do projeto....

Project Wizard ×

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.


Name: WebServer_HTML_CSS_AGO24

Board: Espressif ESP32 Dev Module ▼

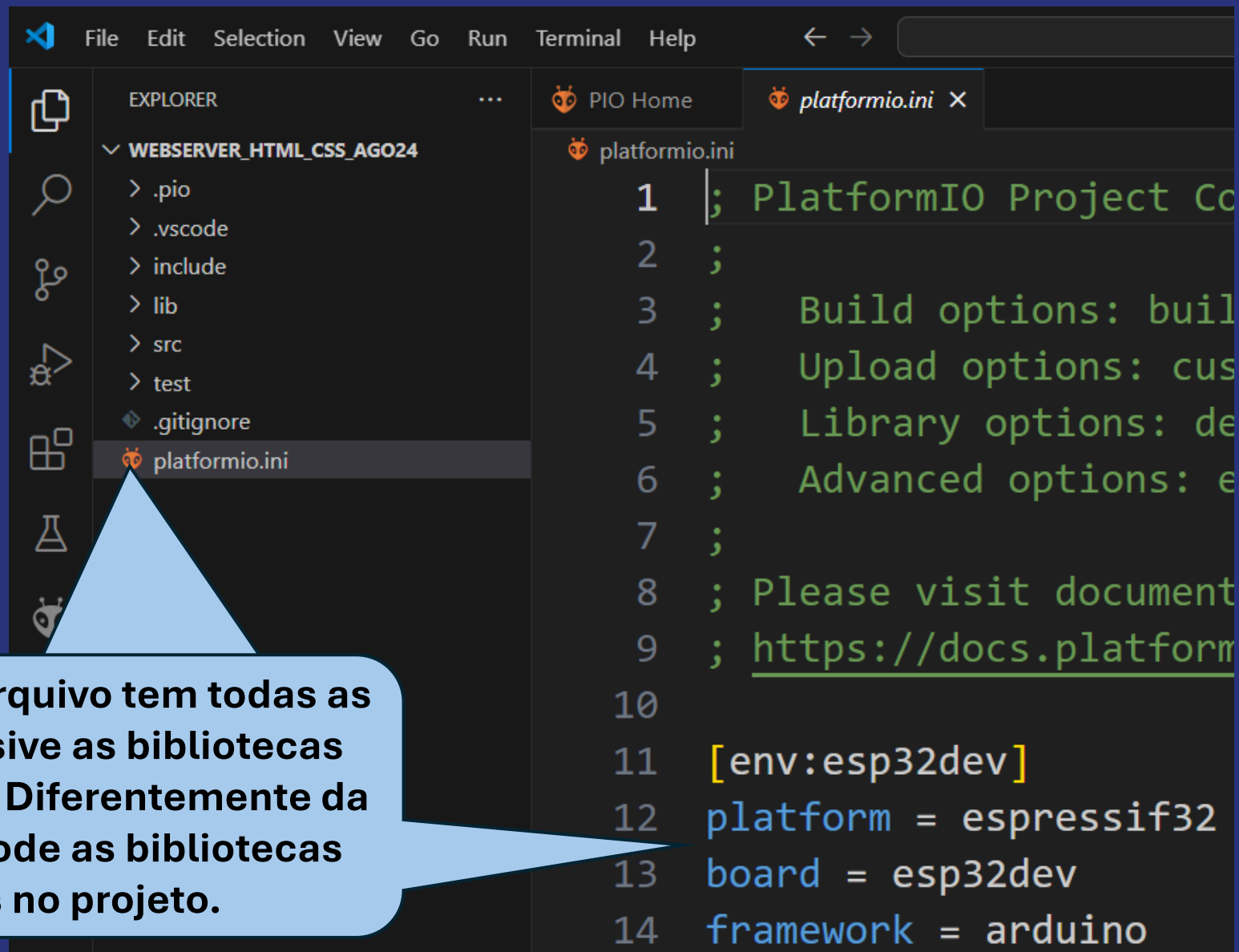
Framework: Arduino ▼

Location: ☒ Use default location ?

Cancel Finish



Foi criado o projeto.....

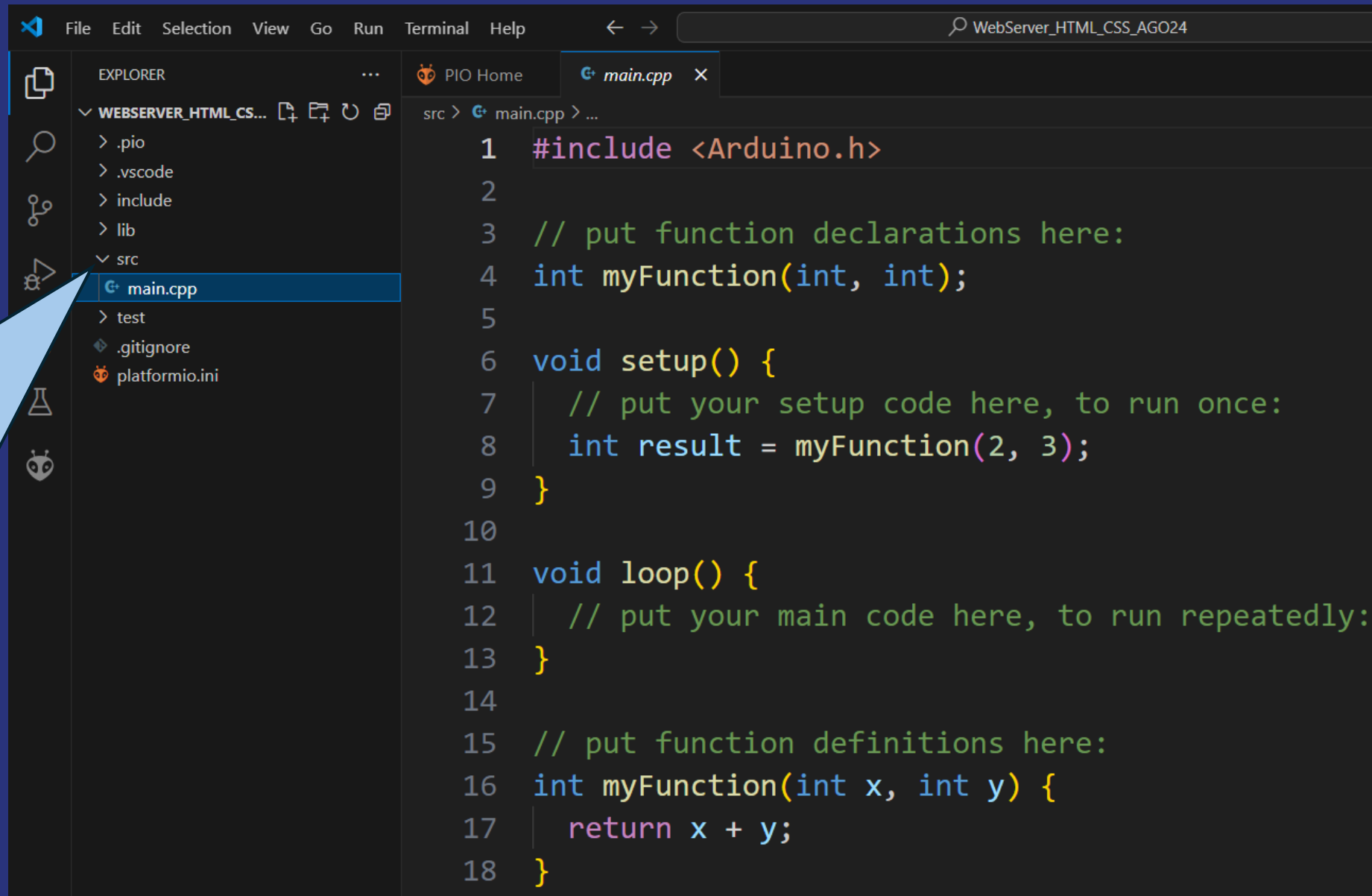


```
File Edit Selection View Go Run Terminal Help
EXPLOLERER
WEBSERVER_HTML_CSS_AGO24
> .pio
> .vscode
> include
> lib
> src
> test
.gitignore
platformio.ini
platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: upload speed, binary format
5 ; Library options: search paths, always include
6 ; Advanced options: http proxy, no-https
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/en/latest/reference/project/configuration.html
10
11 [env:esp32dev]
12 platform = espressif32
13 board = esp32dev
14 framework = arduino
```

platformio.ini = este arquivo tem todas as inicializações. Inclusive as bibliotecas que forem instaladas. Diferentemente da IDE Arduino, no VSCode as bibliotecas são instaladas no projeto.

Como acessar o arquivo *.cpp

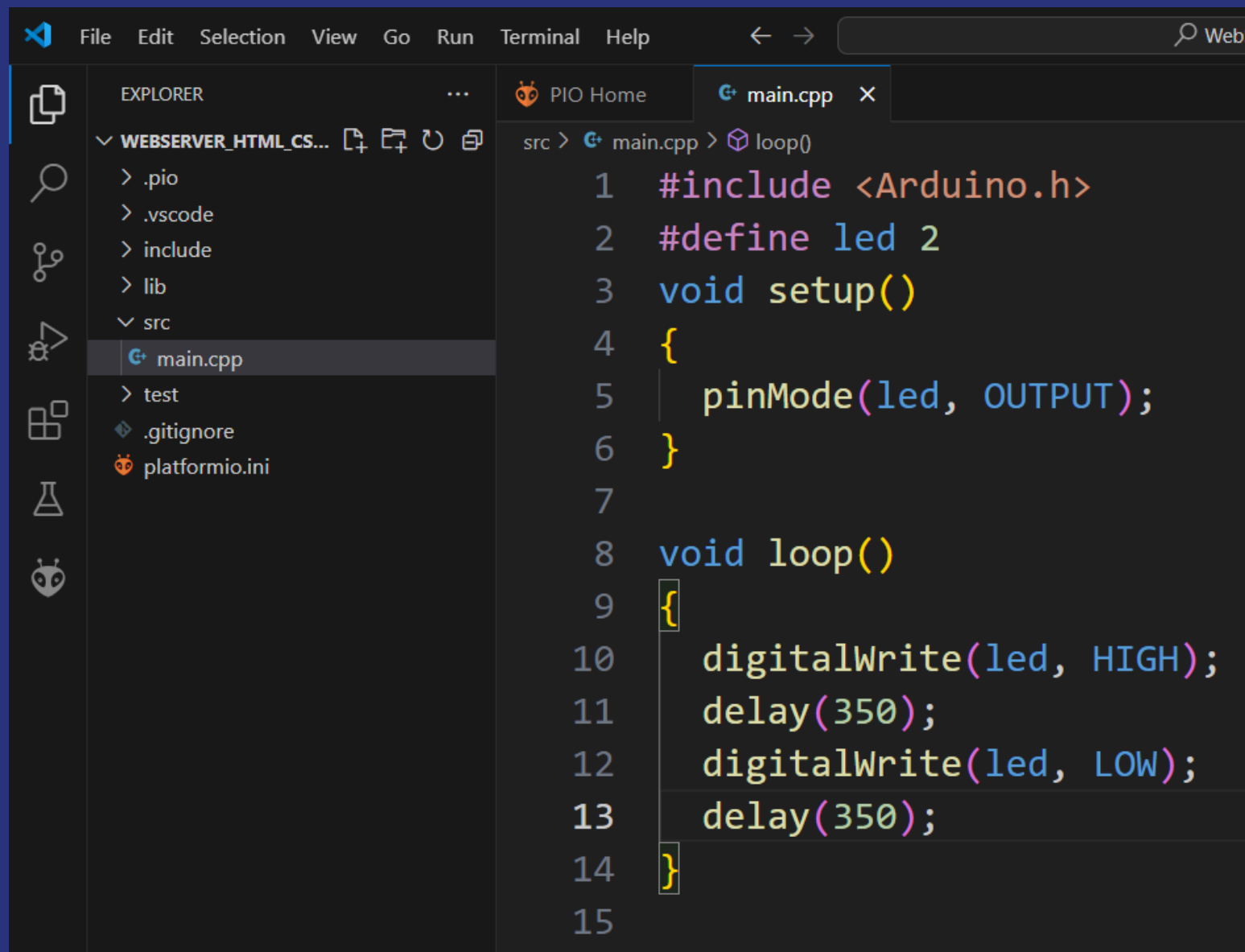
Na pasta src está o arquivo principal. A estrutura do programa main.cpp é igual ao feito na IDE Arduino. Tem duas funções void setup() e void loop(). A única diferença é que deve manter o #include <Arduino.h>



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure for 'WEBSERVER_HTML_CSS_...'. The 'src' folder is expanded, and 'main.cpp' is selected. The main editor window shows the content of 'main.cpp', which includes the Arduino.h header, function declarations for myFunction, and definitions for setup and loop functions.

```
1 #include <Arduino.h>
2
3 // put function declarations here:
4 int myFunction(int, int);
5
6 void setup() {
7     // put your setup code here, to run once:
8     int result = myFunction(2, 3);
9 }
10
11 void loop() {
12     // put your main code here, to run repeatedly:
13 }
14
15 // put function definitions here:
16 int myFunction(int x, int y) {
17     return x + y;
18 }
```

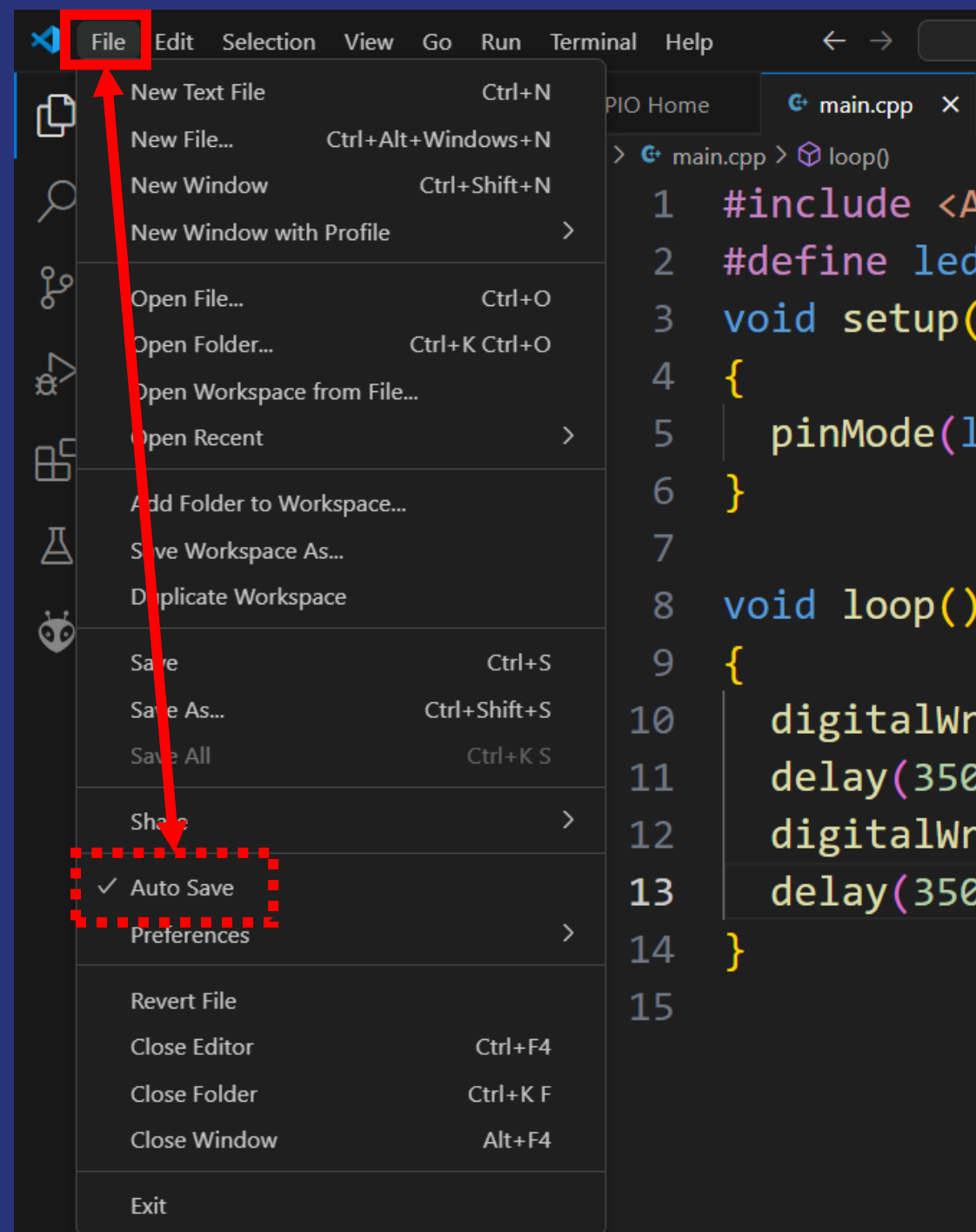
Exemplo simples, apenas para demonstrar o uso do vscode



The image shows a screenshot of the Visual Studio Code (VS Code) editor interface. The left sidebar displays the Explorer view, showing a project structure with folders like .pio, .vscode, include, lib, and src. The file main.cpp is selected under the src folder. The main editor area shows the code for main.cpp, which is a simple C++ program for an Arduino. The code includes the Arduino.h header, defines a pin number 2, and implements setup() and loop() functions. The loop() function toggles the LED state (HIGH and LOW) with a 350ms delay.

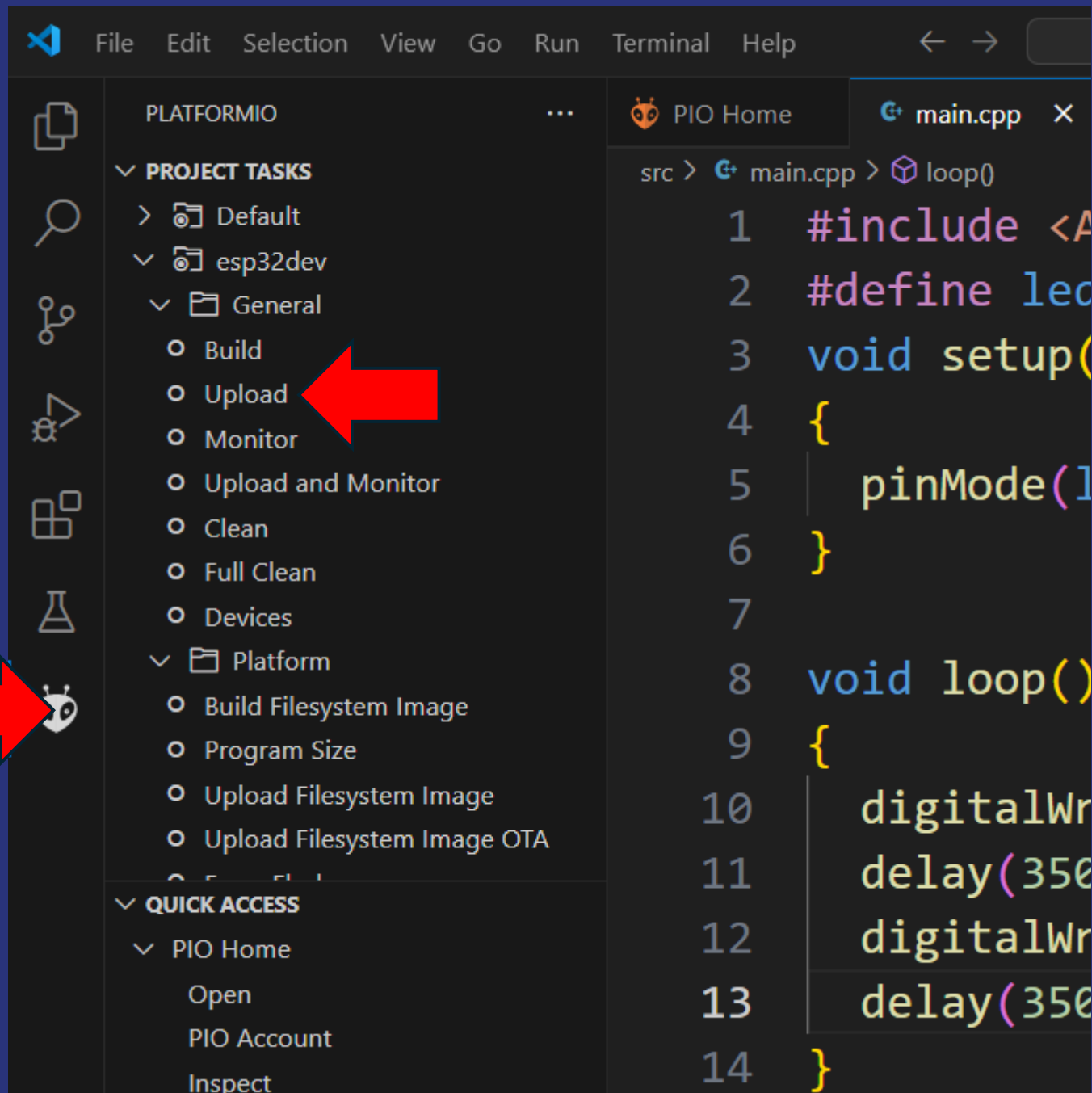
```
src > main.cpp > loop()
1  #include <Arduino.h>
2  #define led 2
3  void setup()
4  {
5      pinMode(led, OUTPUT);
6  }
7
8  void loop()
9  {
10     digitalWrite(led, HIGH);
11     delay(350);
12     digitalWrite(led, LOW);
13     delay(350);
14 }
15
```


Uma dica de usar
o vscode é
habilitar o auto
save, assim não
esquece de
“salvar” a cada
alteração.

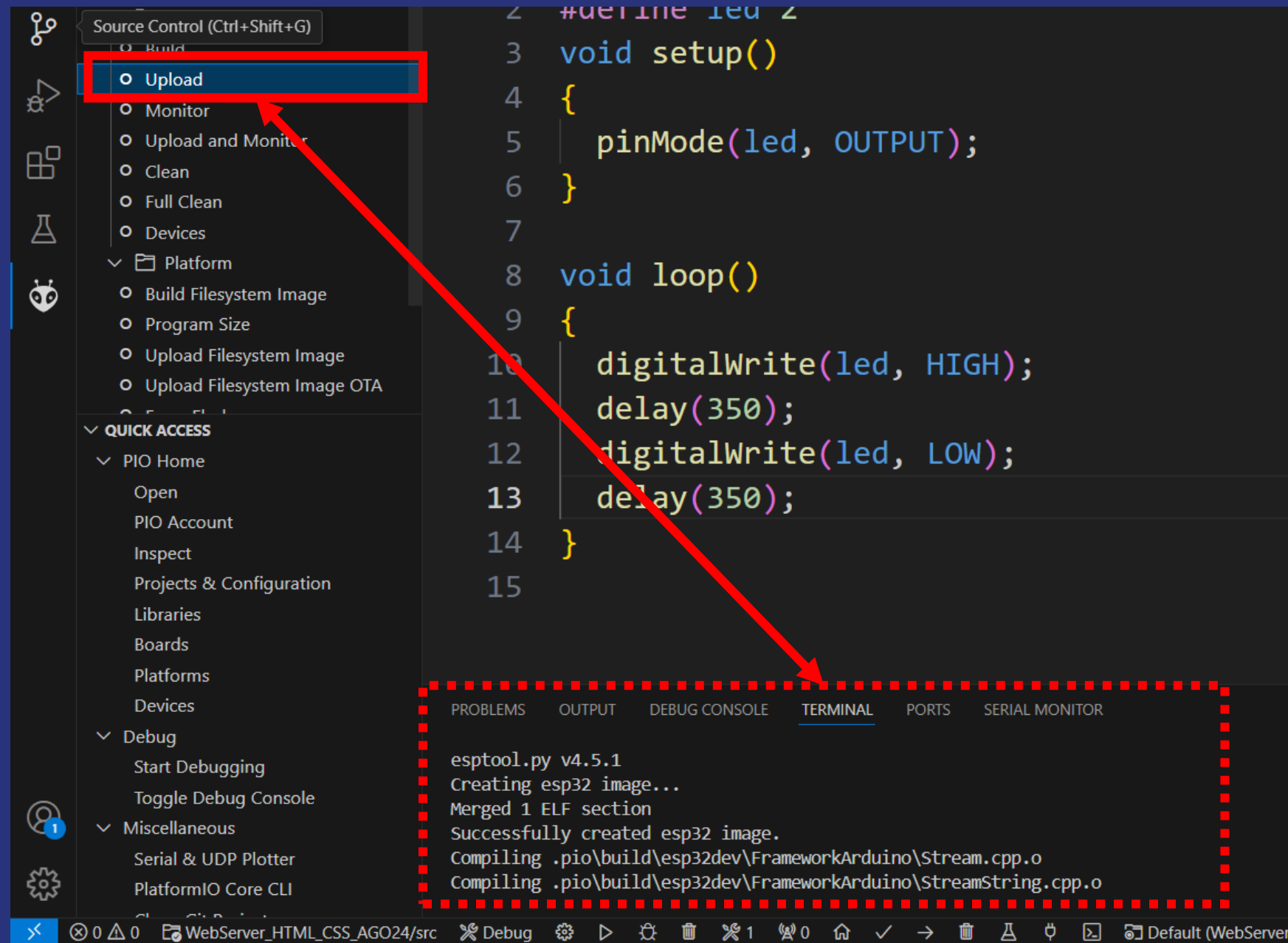


Para compilar, temos que
selecionar o ícone da
platformio.

Abrirá todas estas
opções...para compilar e
gravar o dispositivo clique
em upload....



Depois de clicar em
upload, começa o
processo de compilar
e fazer a programação
do esp32



Se tudo der certo, o console vai indicar “success” e fazer o reset da placa...

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR

Hash of data verified.

Leaving...

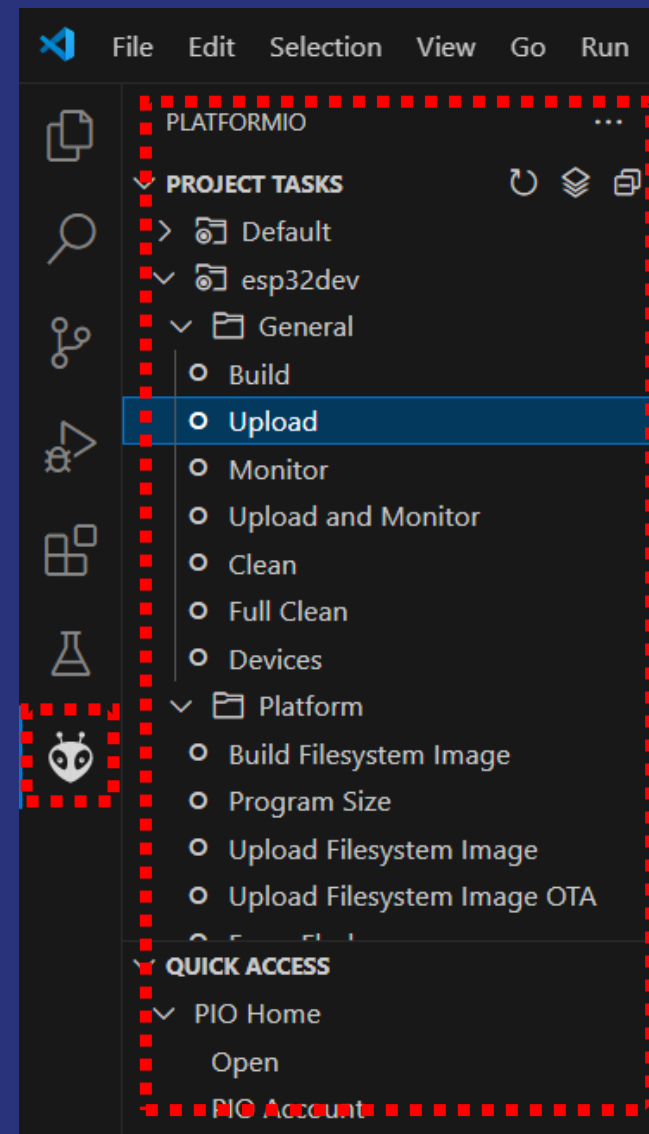
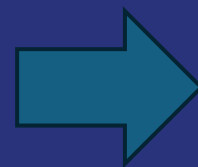
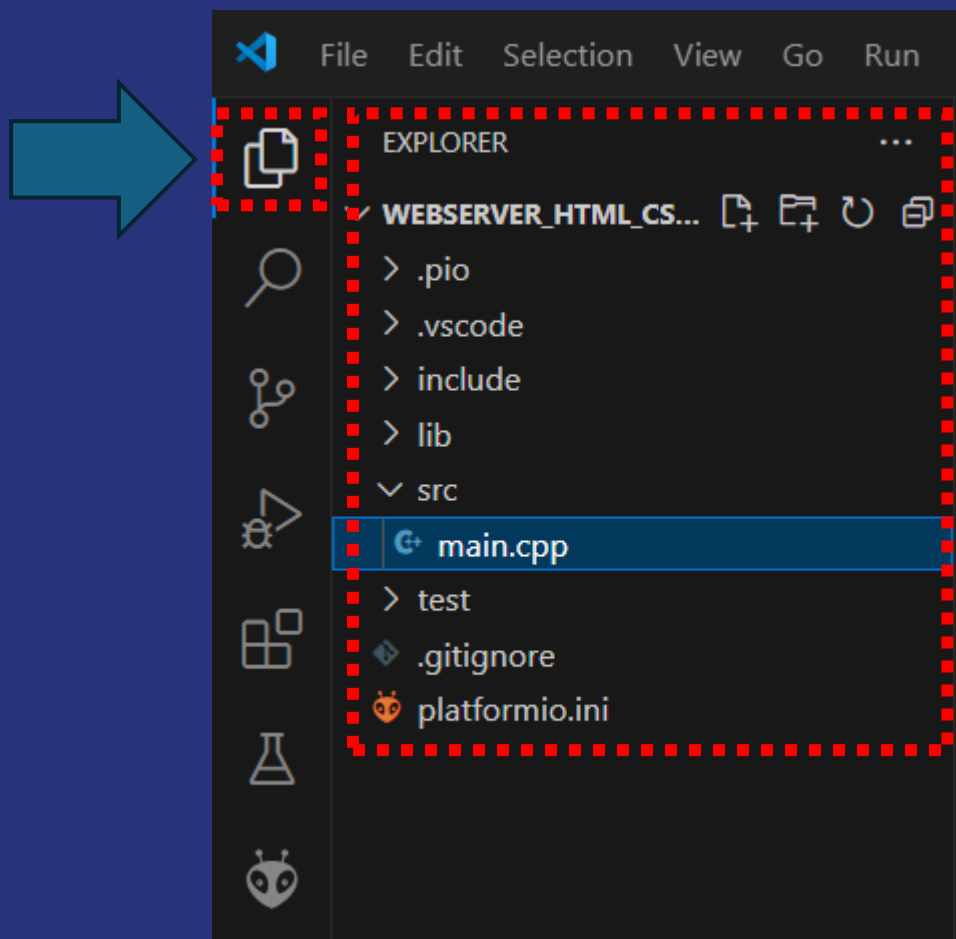
Hard resetting via RTS pin...

===== [SUCCESS] Took 254.46 seconds =====

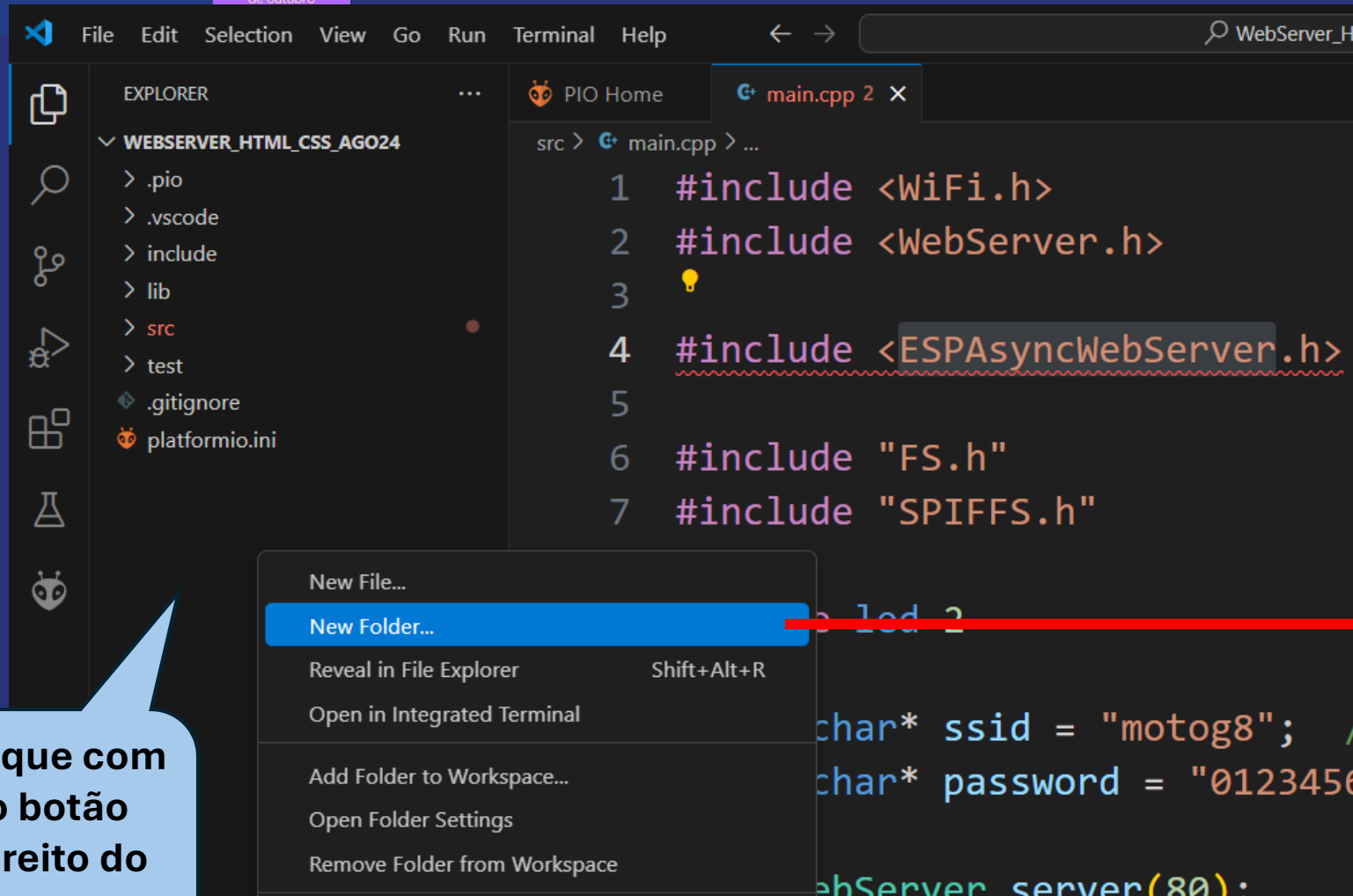
* Terminal will be reused by tasks, press any key to close it.

Resumindo:

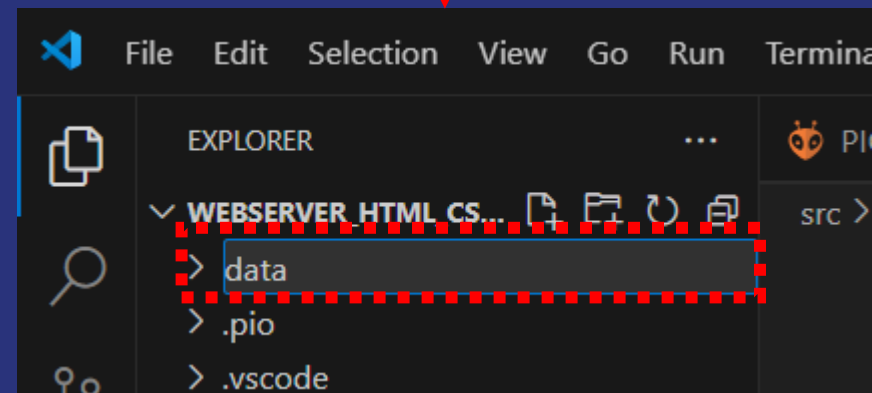
- platformio → comandos (Upload, etc..)
- explorer → pastas e arquivos do projeto.



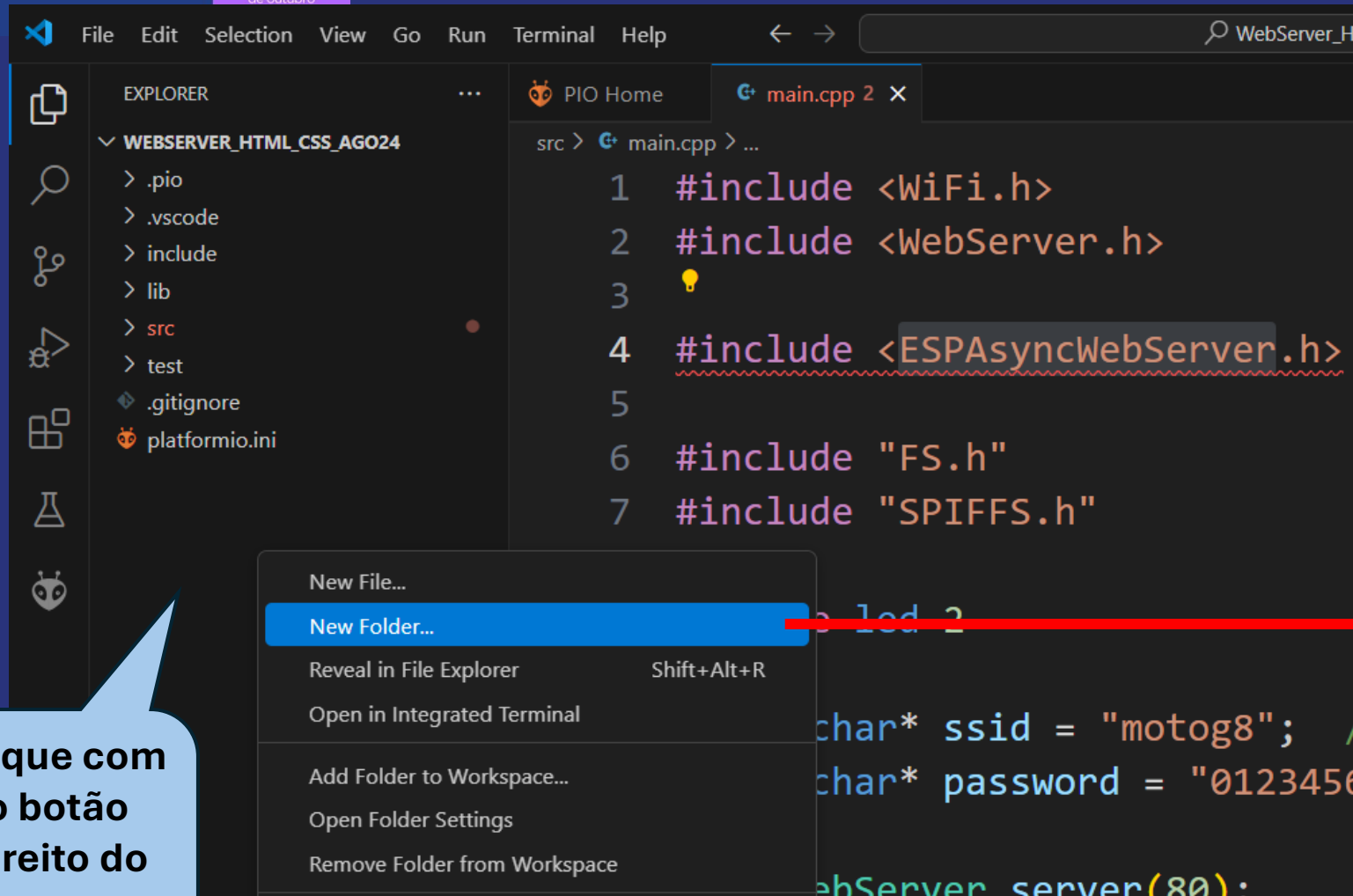
Criando pasta data para gravar na memória flash do esp32



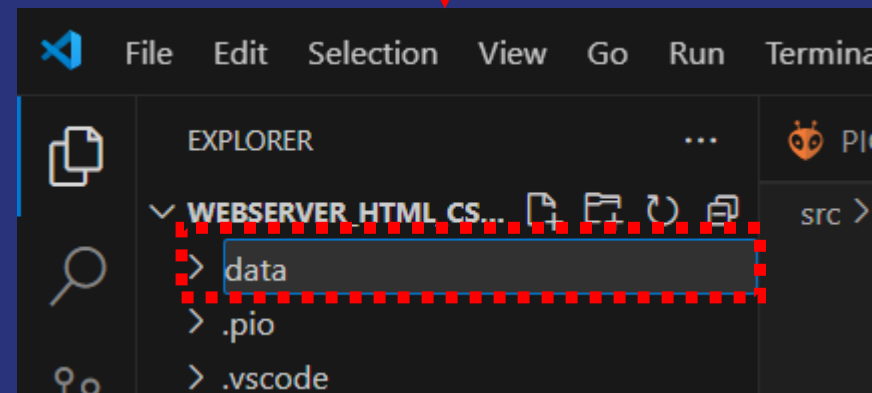
Clique com
o botão
direito do
mouse
nesta área



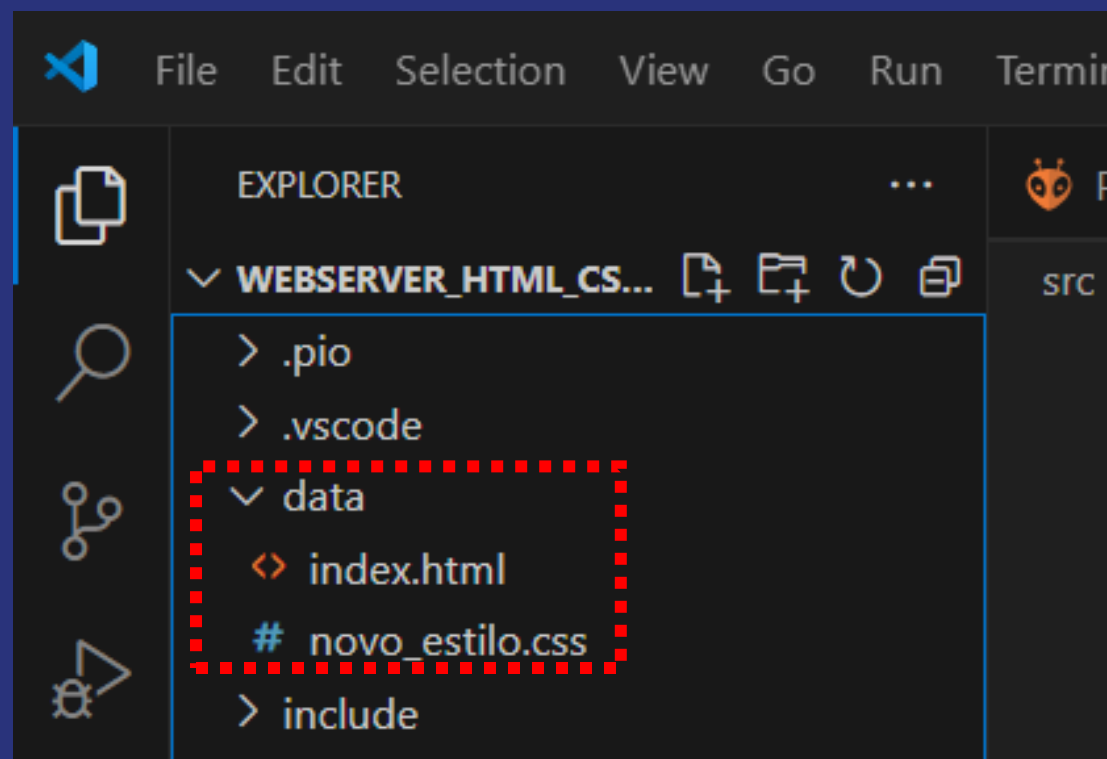
Criando pasta data para gravar na memória flash do esp32



Clique com
o botão
direito do
mouse
nesta área



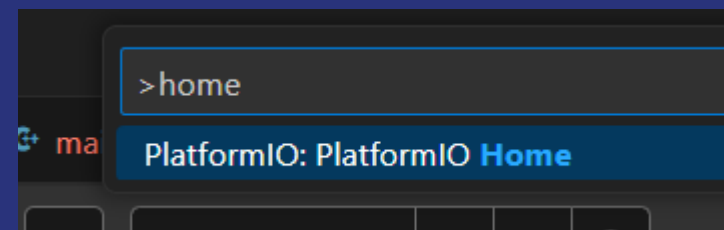
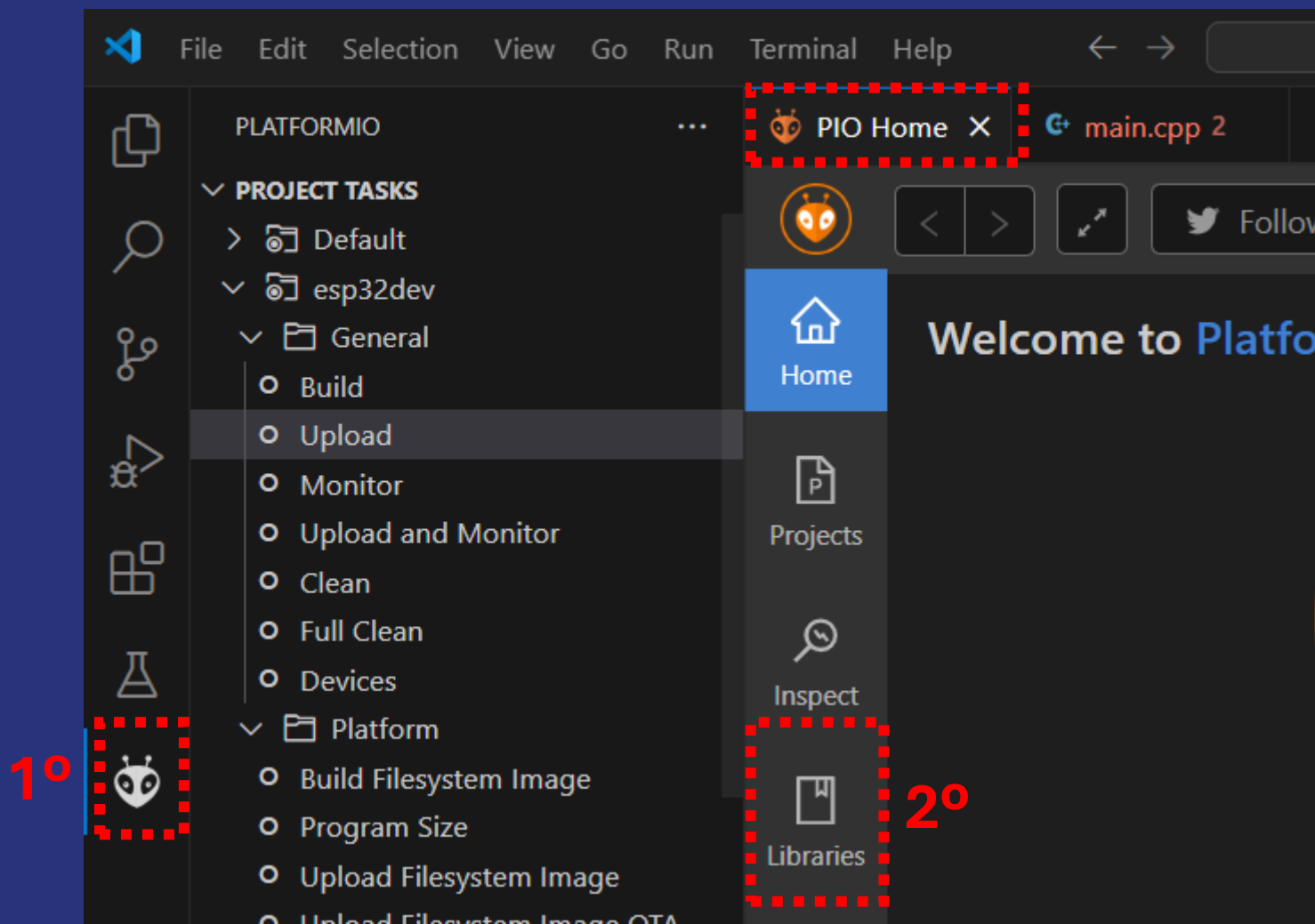
Copia os arquivos para esta pasta data.



Adicionar novas bibliotecas - *ESPAsyncWebServer.h*

OBSERVAÇÃO:

Caso a aba PIO Home tenha sido fechada, para abri-la novamente, basta clicar F1 e digitar home e selecione PlatformIO: PlatformIO Home



Escreva no nome da biblioteca e procure....

The screenshot shows the PlatformIO IDE interface. At the top, there's a tab for 'main.cpp 2'. Below the tab bar, there's a navigation bar with icons for Home, Projects, and Inspect. The main area is divided into sections: 'Registry', 'Installed', 'Built-in', and 'Updates'. The 'Registry' section is active, and a search bar is visible with the text 'ESPAsyncWebServer' entered. A red dashed box highlights the search bar, and a red arrow points from the text to the search button. Below the search bar, there are several filter buttons: 'tft display', 'dht*', 'header:RH_ASK.h', 'keyword:mqtt', 'framework:mbed', 'platform:espressif8266', and 'more...'. The bottom of the interface shows a 'Loading...' status and a footer with links: 'Web', 'Open Source', 'Get Started', 'Docs', 'News', 'Community', and 'Contact Us'. A note at the bottom says 'If you enjoy using PlatformIO, please star our projects on GitHub!'.

PIO Home X main.cpp 2

Follow Us

Registry Installed Built-in Updates

ESPAsyncWebServer

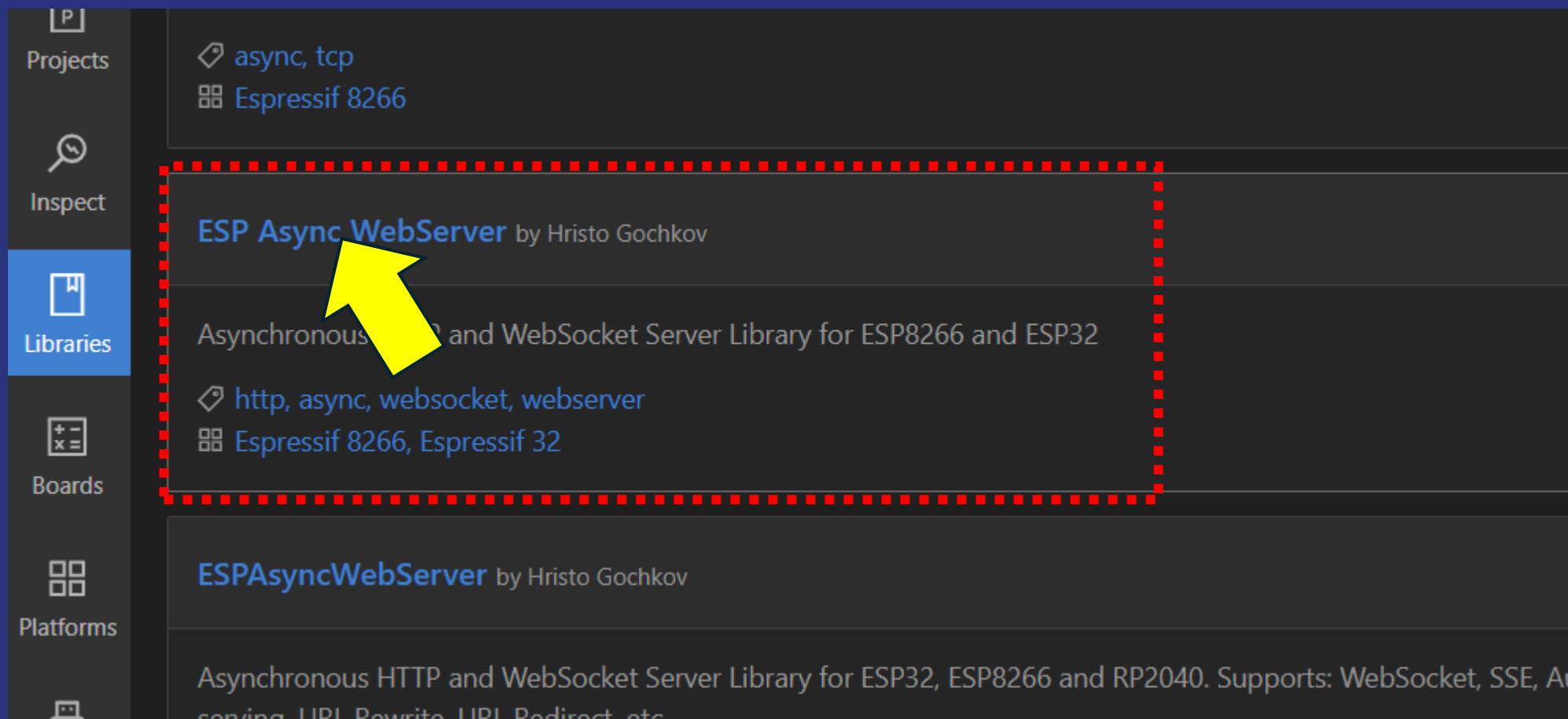
tft display dht* header:RH_ASK.h keyword:mqtt framework:mbed platform:espressif8266 more...

Loading...

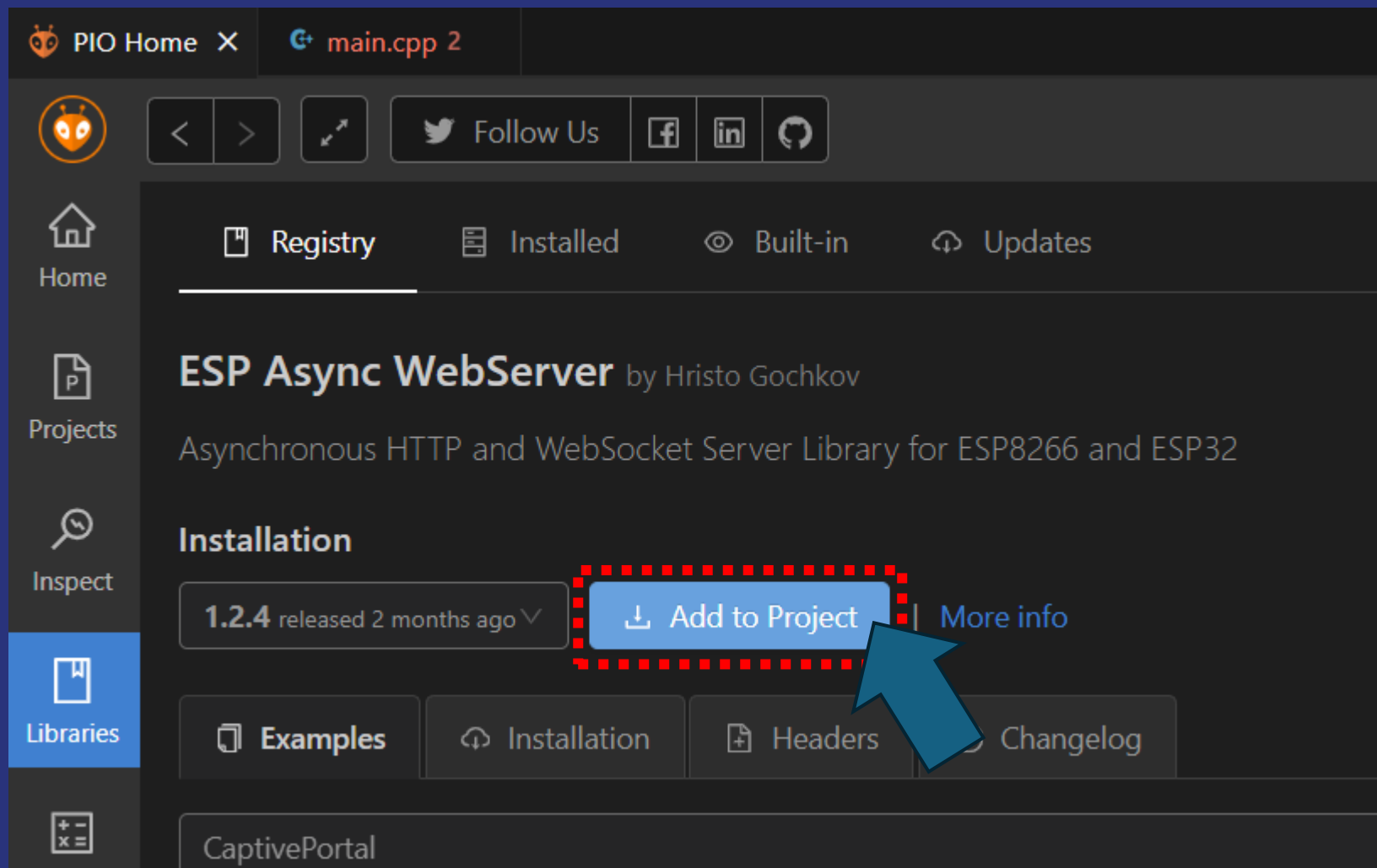
Web · Open Source · Get Started · Docs · News · Community · Contact Us

If you enjoy using PlatformIO, please star our projects on GitHub!

Neste caso, a biblioteca que funciona é a esp async webserver (palavras separadas)



Adicione a biblioteca ao projeto



A biblioteca se aplica ao projeto e não a plataforma....

Add project dependency

me-no-dev/ESP Async WebServer@^1.2.4

Select a project

You can manage your projects in the "Projects" section: create a new or add existing.

Information

> Registry and Specification

> External resources

Cancel

Add

Add project dependency

me-no-dev/ESP Async WebServer@^1.2.4

Select a project

Projects

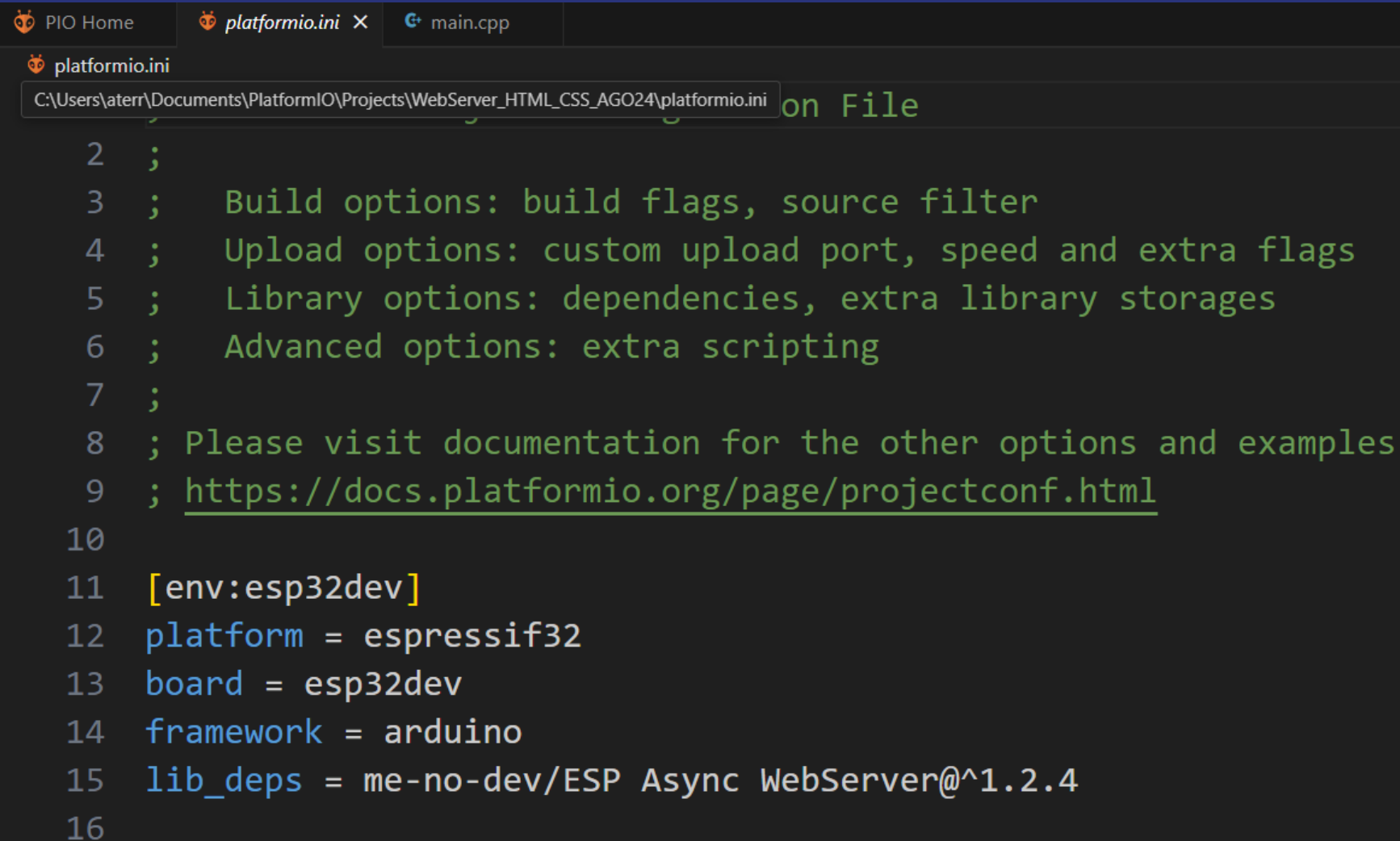
Projects\WebServer_HTML_CSS_AGO24

Projects\NovaPlacaESP32C3

Projects\BLE_ESP32C3_versao1

Selecione o projeto o qual quer
adicionar a dependência da biblioteca
→ depois clique em ADD

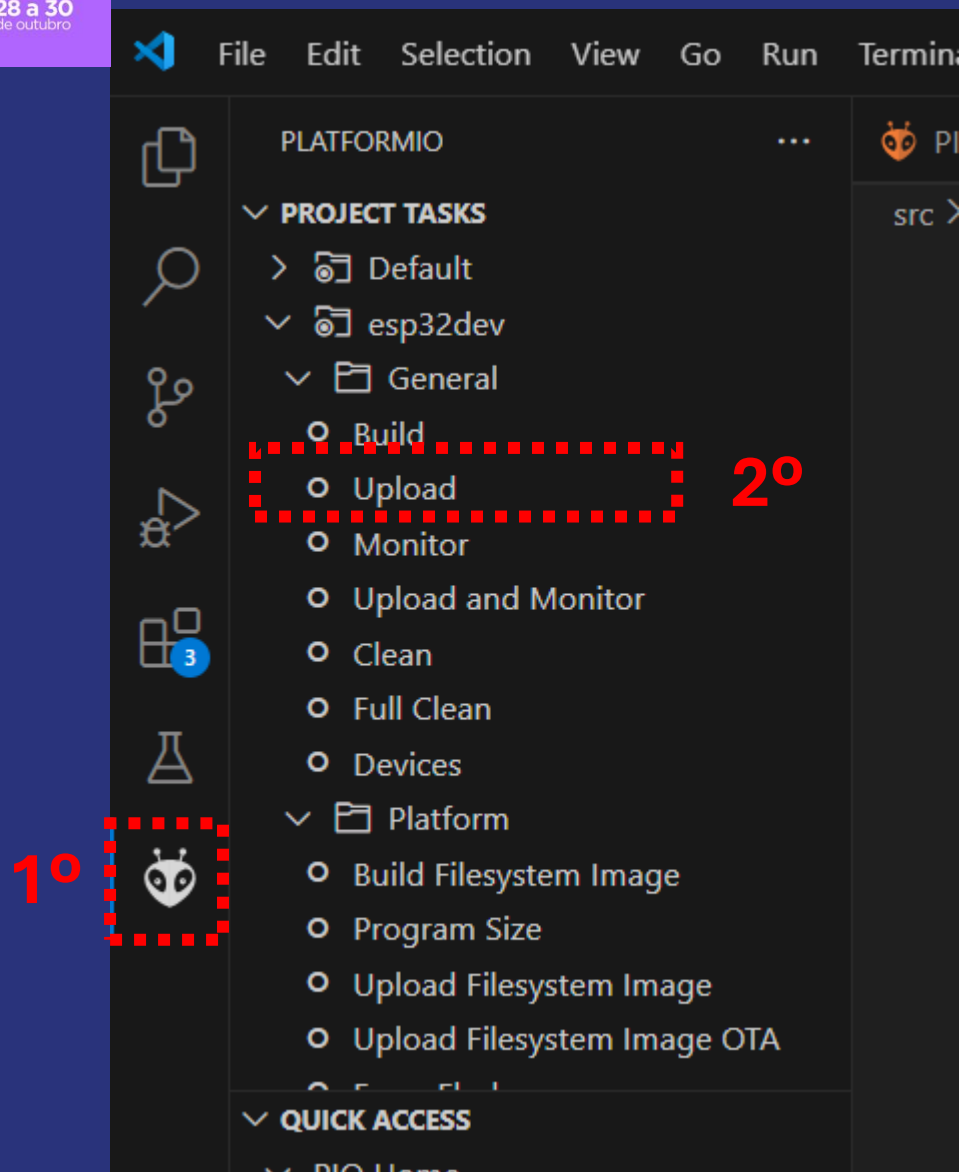
Note que no arquivo platformio.ini consta tudo que é acrescentado no projeto.



```
platformio.ini
C:\Users\aterr\Documents\PlatformIO\Projects\WebServer_HTML_CSS_AGO24\platformio.ini on File

2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:esp32dev]
12 platform = espressif32
13 board = esp32dev
14 framework = arduino
15 lib_deps = me-no-dev/ESP Async WebServer@^1.2.4
16
```

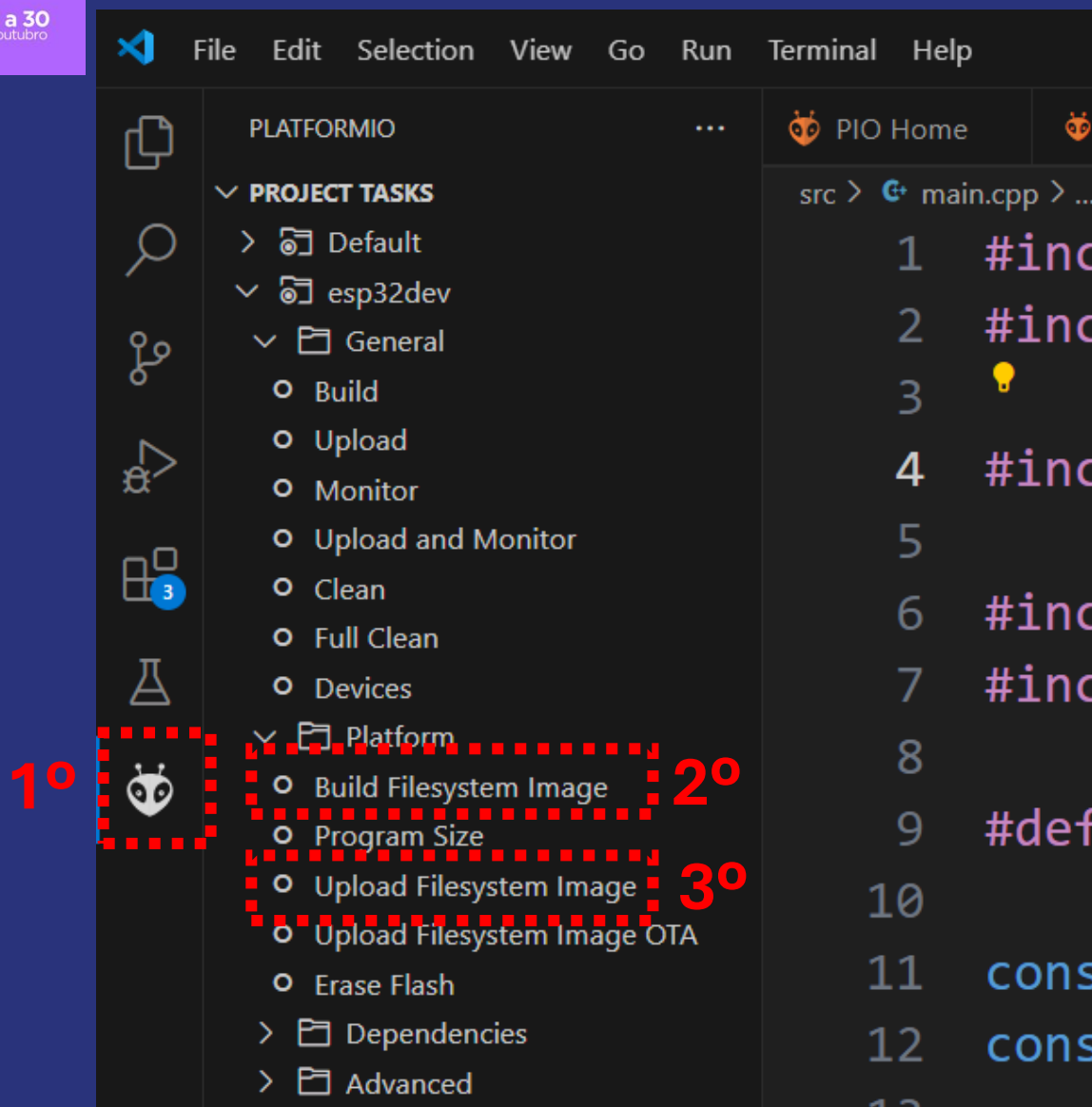
Como compilar o projeto



Se você deseja apenas compilar e programar o microcontrolador, clique em (1) e depois em (2).

Lembre-se de ter a placa conectada no computador.

Baixar arquivos da pasta data para o esp32



Se você deseja baixar os arquivos da pasta data para a memória Flash do ESP32, clique em (1), depois em (2) e depois em (3). Lembre-se de ter a placa conectada no computador.

DEMONSTRAÇÃO PRÁTICA

Baixe o material da palestra

acesse o github

<https://github.com/aterroso/ESPServidorWEB>