# Web Server no ESP32

*Gabriel Alves de Faria*

## Visão geral do projeto

Antes de seguir para o projeto é preciso descrever o que o servidor web fará.

- O servidor irá realizar uma requisição em uma API para obter um JSON com informações sobre o covid-19;
- Criar uma página na web com as informações obtidas;
- Para o desenvolvimento desse projeto será necessário apenas a placa ESP32;

Este é apenas um exemplo simples para ilustrar como criar um servidor Web que realiza requisições.

## Instalando a placa ESP32 no Arduino IDE

Há um complemento para o IDE do Arduino que permite programar o ESP32 direto na IDE. Para a instalação desse complemento acesse Arduino ESP32 e siga as instruções fornecidas para a instalação dependendo do sistema operacional.

## Código do servidor Web ESP32

O código completo está disponível no github.

### Bibliotecas utilizadas

```
#include <WiFi.h>
#include <WebServer.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
```

- **WiFi.h**: Necessária para conectar ao WiFi;
- **WebServer.h**: Ajuda na configuração do servidor;
- **HTTPClient.h**: Para realizar a requisição na API;
- **ArduinoJson.h**: Utilizada para trabalhar com estruturas no formato de JSON.

### Definindo suas credenciais de rede

Você precisa modificar as seguintes linhas com suas credenciais de rede: *SSID* e *senha*.

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

## Estrutura para guardar informações

Uma estrutura com as informações esperadas da requisição na API.

```
struct infos
{
    String country;
    String cases;
    String confirmed;
    String deaths;
    String recovered;
    String updated_at;
};
```

## Declaração dos objetos

No *WebServer*, o número que informamos será a porta que o servidor irá ficar no local host.

```
WebServer server(80);
infos infos;
HTTPClient http;
StaticJsonDocument<500> doc;
```

## Função Setup ()

Como parâmetro no *http.begin()* espera-se a url em que irá realizar as requisições http.

```
http.begin("https://covid19-brazil-api.now.sh/api/report/v1/brazil");
```

Conectando ao Wifi.

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

Enquanto o ESP32 tenta se conectar à rede, podemos verificar o status da conectividade com o método *WiFi.status()*. As possíveis respostas estão presentes na documentação do mesmo.

```
while (WiFi.status() != WL_CONNECTED)
{
    delay(1000);
}
```

Para acessar o servidor, basta copiar o IP e acessar a partir de um navegador.

```
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
```

```
Connecting to Pericles
.
WiFi connected
IP address:
192.168.0.105
```

Para lidar com solicitações HTTP recebidas no servidor, precisamos especificar qual código executar quando um URL específico for solicitada. Para fazer isso, usamos o

método *on*. Este método necessita de dois parâmetros, o primeiro é um caminho da URL e o segundo é o nome da função que queremos executar quando essa URL for solicitada.

Por exemplo, a primeira linha do trecho de código abaixo indica que quando um servidor recebe uma solicitação HTTP no caminho raiz ( / ), ele dispara a função *handle_ OnConnect()*.

Quando o servidor receber uma solicitação que não foi declarada no método *on*, ele deve responder com um status *HTTP 404* (Lista de códigos HTTP) e uma mensagem para o usuário. Para isso é utilizado o método *onNotFound()* que irá dizer oque o servidor deve fazer quando receber uma solicitação de URL não declarada.

```
server.on("/", handle_OnConnect);
server.onNotFound(handle_NotFound);
```

Inicia o servidor.

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

## Função Loop ()

Para lidar com as solicitações HTTP, precisamos chamar o método *handleClient()* no objeto do servidor.

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

## Função handleOnConnect ()

Primeiramente realiza-se a requisição com o método *GET(Métodos de requisição)* no objeto http. Esse método não recebe argumentos e retorna o código HTTP da solicitação que é armazenado em uma variável para o tratamento de erros.

```
int httpCode = http.GET();
```

Em seguida é verificado o código HTTP de retorno, por padrão quando o retorno é igual a 200 sabe-se que a requisição foi realizada com sucesso. Para obter a resposta da solicitação é usado o método *getString*, que não recebe argumentos e retorna os dados da requisição.

A resposta dessa requisição será um (JSON), para poder trabalhar com essa estrutura é preciso converter o payload(uma string com o JSON) para uma estrutura melhor definida. Para isso é utilizado o método *deserializeJson* que retorna um erro caso não seja possível converter. Se não tinha nenhum problema com a conversão é atribuído os valores recebidos ao objeto criado anteriormente para salvar as informações e em seguida enviado para o web server com o código HTTP 200.

```
if (httpCode == 200)
    {
        Serial.println("Successful request");
        String payload = http.getString();
        DeserializationError error = deserializeJson(doc, payload);
        if (error)
        {
            Serial.print("Fail! Error: ");
            Serial.println(error.c_str());
        }
        else
        {
```

```
            infos = {doc["data"]["country"],

            doc["data"]["cases"],

            doc["data"]["confirmed"],

            doc["data"]["deaths"],

            doc["data"]["recovered"],

            doc["data"]["updated_at"]};
        }
    }
    else
    {
        Serial.println("Request failed");
    }
    Serial.println("On Connect");
    server.send(200, "text/html", SendHTML());
```

## Função handleNotFound ()

Caso seja solicitado uma requisição que não existe no web server, o retorno dessa requisição será a função *handleNotFound* que envia um código 404 que é conhecido como *Not Found*.

```
server.send(404, "text/plain", "Not found");
```

## Função SendHTML ()

A função é responsável por gerar uma página da Web sempre que o web server receber uma solicitação. Apenas concatena o código HTML em uma grande string e retorna à server.send().