



# Tutorial - Enviando um JSON com ESP32

04/06/2020

PET - Tecnologia em Eletrônica e Computação

Itajubá, Minas Gerais

## Objetivos

Descrever um algoritmo que enviará informações processadas na placa ESP32 em uma requisição a um servidor em forma de JSON.

## Visão geral

Dentre as várias funcionalidades presentes na placa ESP32 está a capacidade de se conectar com uma rede Wi-Fi e transmitir e receber informações através dela. Dessa forma, é cabível que a placa consiga trocar informações com um banco de dados, seja enviando ou recebendo. Nesse tutorial ensinaremos como estabelecer a conexão entre a placa e o banco NoSQL e como enviar uma requisição POST a esse banco para gravar informações recebidas em forma de JSON (JavaScript Object Notation). **Não abordaremos como essa requisição será tratada após ser recebida pelo servidor.**

Para esse projeto são necessárias apenas uma placa ESP32-DevKitC e uma conexão wi-fi com acesso a internet.

## O que é um JSON?

O formato JSON, descrito por Douglas Crockford em 2000, é uma estrutura para enviar informações de forma leve e rápida. Ele descreve uma forma simples tanto para o computador interpretar quanto para seres humanos entenderem e é independente da linguagem utilizada, pois utiliza convenções comuns a linguagem C e seus similares (C++, C#, Java, Python, etc.).

A estrutura do JSON é formada por uma coleção de duplas [nome, valor], sendo "valor" único ou uma lista de valores. A estrutura de um JSON é mostrada a seguir.

```
{
  "json": {
    "lista": {
      "valor1": "1",
      "valor2": "2"
    }
  }
}
```

## O que são requisições?

Uma requisição HTTP é definida como uma solicitação para que certa ação seja executada por um determinado recurso. Elas podem possuir vários tipos e formatos, porém os mais importantes, e mais utilizados estão listados a seguir.

- GET: Solicita apenas um conjunto de dados do recurso;
- POST: Usada para submeter uma certa informação ao recurso, geralmente causando uma modificação interior a ele, adicionando dados ao recurso.
- PUT: Utilizado para modificar dados já existentes.
- DELETE: Utiliza-se o DELETE para remover um dado existente.

## O que será feito

O intuito principal do tutorial é demonstrar como estruturar e enviar um JSON por uma requisição HTTP, logo, as informações contidas neste JSON não são de interesse. Sendo assim, utilizaremos como informação os nomes das redes Wi-Fi cuja placa capta sinal. Além disso, será usado um servidor de teste para o envio dessas informações. Para este tutorial usaremos o servidor *httpbin.org*, que aceita requisições de todo tipo, retornando uma resposta com dados da requisição feita.

## Ambiente de Programação

Para este projeto utilizaremos a Arduino IDE, uma IDE de programação gratuita distribuída pela Arduino Software, podendo ser baixada no site oficial da Arduino ([www.arduino.cc/en/main/software](http://www.arduino.cc/en/main/software)). Além da IDE, utilizaremos as bibliotecas de arduino para ESP32, que podem ser encontradas no repositório da Espressif no GitHub (<https://github.com/espressif/arduino-esp32>) .

Um tutorial didático de como utilizar a IDE do Arduino para programação da ESP32 pode ser encontrado no artigo “Programar ESP32 com a IDE do Arduino - Tutorial completo”, do autor Gustavo Teixeira, Bacharel em Ciência da Computação pela URI. O artigo encontra-se no link (<http://www.usinainfo.com.br/blog/programar-esp32-com-a-ide-arduino-tutorial-completo/>) .

## O Código

### 1. Pacotes e Definições

```
1 #include <WiFi.h>
2 #include <HTTPClient.h>
3
4 #define DELAY 7000
5
6
7 String serverName = "http://httpbin.org/post";
8 unsigned long lastUpdate = 0;
```

Primeiramente, é preciso importar os pacotes necessários para se estabelecer uma conexão Wi-Fi (WiFi.h) e para se enviar uma requisição HTTP, cujas funções são fornecidas pela biblioteca HTTPClient.h. Além disso, é preciso definir o caminho url do servidor para o qual será feita a requisição: "<http://httpbin.org/post>". Ainda, define-se uma contante, chamada "DELAY", e uma variável, "lastUpdate", inicializada com zero. A constante DELAY diz respeito ao intervalo entre cada execução (coleta e envio de dados), já a variável servirá para o controle desse tempo entre cada execução e será atualizada com o instante de cada execução. Ambas serão utilizadas mais a frente.

### 2. Coletando Informações

```
1 String getWifiList() {
2     int n = WiFi.scanNetworks();
3     String results = "{";
4     for (int i = 0; i < n; ++i) {
5         if (i == n-1){
6             results = results+"\"wifi\"+i+\"\": \""+WiFi.SSID
7                 (i)+"\"";
8         } else {
9             results = results+"\"wifi\"+i+\"\": \""+WiFi.SSID
10                (i)+"\"";
11        }
12    }
13    results = results+"}";
14    return results;
15 }
16
17 String getInformations(){
18     String result = "{\"signals\":{"\"wi-fi\": "+getWifiList()+"
19     }}";
20     return result;
21 }
```

A função "getWifiList()" descreve a execução, de fato, da ação de coletar as informações e estruturar o JSON. Para isso, primeiramente é preciso coletar as informações das redes Wi-Fi existentes na região. Isso é feito através da função "scanNetworks()", fornecida pela biblioteca WiFi.h. Essa função irá escanear as redes Wi-Fi presentes, retornando como resposta um inteiro correspondente ao número de redes encontradas. Além disso, ao realizar o *scan*, automaticamente serão armazenadas as informações dessas redes, dentro de variáveis presentes na biblioteca. Para o projeto, será de interesse o nome de cada rede, que é armazenado dentro do vetor WiFi.SSID.

Dessa forma, já temos quais as informações que serão enviadas na requisição em formato JSON. Assim, precisamos estruturar essas informações em uma String, com o formato que será enviado. Suponhamos que foram encontradas duas redes, cujos nomes são Wi-Fi-1 e Wi-Fi-2, a seguir é descrito qual o formato JSON que contém essas informações.

```
{  
    "wifi0" : "Wi-Fi-1",  
    "wifi1" : "Wi-Fi-1"  
}
```

Para enquadrar esse texto em uma string basta colocá-lo entre aspas duplas e trocarmos a quebra de linha por um simples 'espaço', uma vez que o formato JSON também será reconhecido se escrito em uma única linha. Entretanto, os nomes e atributos dentro do JSON também devem estar entre aspas duplas, ou seja, haverá um conflito, uma vez que o compilador interpreta essas aspas interiores como o fechamento das aspas externas. Assim, precisa-se indicar cada aspas interior para que o compilador as ignore como "fecha aspas". Para isso, basta colocar uma barra à esquerda ( \ ) antes de cada aspas interior. A seguir são listados os passos para reescrever o JSON em formato string.

- i. Escreva a informação em formato JSON:

```
{  
    "wifi0" : "Wi-Fi-1",  
    "wifi1" : "Wi-Fi-1"  
}
```

- ii. Indique cada aspas duplas internas com uma barra à esquerda:

```
{
    \"wifi0\" : \"Wi-Fi-1\",
    \"wifi1\" : \"Wi-Fi-1\"
}
```

- iii. Substitua as quebras de linha por 'espaço':

```
{ \"wifi0\" : \"Wi-Fi-1\", \"wifi1\" : \"Wi-Fi-1\" }
```

Assim, ao enviar essa string como corpo de uma requisição, ela será interpretada como um JSON com as informações de Wi-Fi.

Porém, “Wi-Fi-1” e “Wi-Fi-2” são apenas valores hipotéticos. Os valores de nome de cada rede Wi-Fi encontrada estão armazenados no vetor WiFi.SSID, logo é preciso percorrer esse vetor colocando os nomes de cada rede dentro do JSON. Para isso, basta um loop de tamanho N, onde N é o número de redes Wi-Fi encontradas, ou seja, o valor de retorno da função “scanNetworks()”. Assim, a estruturação do JSON se dá da forma:

```
String results = "{";
for (int i = 0; i < n; ++i) {
    if (i == n-1){
        results = results+"\"wifi"+i+"\": \" "+WiFi.SSID
            (i)+"\"";
    } else {
        results = results+"\"wifi"+i+"\": \" "+WiFi.SSID
            (i)+"\", ";
    }
}
results = results+"}";
```

Primeiramente, inicia-se a string com o abre-chave, indicando o início de um atributo em forma de lista do JSON. Em seguida, percorre-se cada posição do vetor WiFi.SSID, adicionando cada informação à string JSON. Lembre-se de se atentar ao último valor da lista, da posição n-1 do vetor, pois esta posição do JSON não pode se encerrar com vírgula. Por fim, então, basta encerrar a lista adicionando um fecha-chave ao fim. Perceba que no código acima cada trecho de texto inserido à string deve estar entre aspas duplas, porém aquelas que fazem parte da estrutura JSON são indicadas pela barra “\”. Essa string será retornada como resposta da função “getWifiList()” e irá compor parte do JSON final a ser enviado.

A função "getInformations()" irá, então incrementar o JSON, colocando essa lista como um atributo que representa os sinais "wi-fi" encontrados e esse atributo é parte de um nível superior chamado "signals". Assim, temos que o formato final do JSON é:

```
{
  "signals": {
    "wi-fi": {
      "wifi0": "Wi-Fi-1",
      "wifi1": "Wi-Fi-2"
    }
  }
}
```

Que se transforma em uma string no formato:

```
"{"signals":{"wi-fi":{"wifi0":"Wi-Fi-1","wifi1":"Wi-Fi-2"}}}"
```

É importante ressaltar que o retorno função "getWifiList()" já se apresenta em formato JSON, logo o incremento feito pela função "getInformations()" não é necessário.

### 3. setup()

```
1 void setup() {
2   Serial.begin(115200);
3   char* ssid = "wifiSsid";
4   char* password = "wifiPassword";
5   WiFi.begin(ssid, password);
6   Serial.print("Conectando.");
7   while(WiFi.status() != WL_CONNECTED) {
8     Serial.print(".");
9     delay(1000);
10  }
11  Serial.println("Conectado ao Wi-Fi!");
12  ssid = "null";
13  password = "null";
14 }
```

A função "setup()" se refere às configurações da placa e sempre será executada por padrão uma única vez quando a placa é iniciada. Nela serão feitas as configurações de

conexão Wi-Fi e a inicialização da porta serial de comunicação entre a placa e o computador, que será usada para debug.

Para configurar a conexão Wi-Fi precisa-se de duas informações: o nome da rede que deseja-se conectar e a senha de acesso. Basta então passar essas informações como parâmetro da função “WiFi.begin(nome, senha)” e a conexão será iniciada. A conexão é dada como completa quando o retorno da função “WiFi.status()” é igual a “WL\_CONNECTED”, logo, precisa-se de um loop que espera até que a conexão esteja completa, que é descrito no código acima como o loop “while (WiFi.status() != WL\_CONNECTED)”. Para o início da porta serial é simples, basta chamar a função “Serial.begin(freq)”, passando como parâmetro a frequência de transmissão serial que deve ser usada pela placa. No caso, utilizamos 115200.


## 4. loop()

```

1 void loop() {
2   String toSend;
3   long unsigned int now = millis();
4   if ((now - lastUpdate) > DELAY) {
5     if(WiFi.status() == WL_CONNECTED){
6       HTTPClient http;
7       http.begin(serverName.c_str());
8       http.addHeader("Content-Type", "application/json");
9
10      toSend = getInformations();
11
12      int httpResponse = http.POST(toSend.c_str());
13
14      if (httpResponse > 0) {
15        Serial.print("Resposta HTTP: ");
16        Serial.println(httpResponse);
17        Serial.println(http.getString());
18      }
19      else {
20        Serial.print("Erro: ");
21        Serial.println(httpResponse);
22      }
23      http.end();
24    }
25    else {
26      Serial.println("WiFi Desconectado");
27    }
28    lastUpdate = millis();
29  }
30 }
31 }
```

A função loop é executada, por padrão, infinitas vezes, consecutivamente, após o término da setup(). Nessa função, então, será solicitada a coleta dos dados e será feito o envio da requisição.





Primeiramente, para o controle, será verificado se o intervalo entre a última execução (`lastUpdate`) e o tempo atual já é maior que o delay desejado entre os envios (`DELAY`). Se for menor, o dispositivo não precisa realizar nenhuma ação, apenas passar para o próximo ciclo de execução, no qual irá testar o intervalo novamente. Quando atingir o tempo de delay, então, será executada, de fato, a coleta e envio dos dados.

Para isso, precisa-se inicializar uma requisição nova, ou seja, criar uma variável do tipo `HTTPClient` (linha 6) e definir a url do servidor que irá receber a requisição (linha 7). Repare que ao se passar o endereço da URL usa-se também a extensão `“.c_str()"`. Isso é preciso pois a função espera como parâmetro um vetor de caracteres, e não um objeto do tipo `string`, como foi definida nossa variável. Assim, a função `c_str()` faz essa conversão.

A seguir, precisa-se configurar o cabeçalho da requisição. Para isso, basta usar o método `“addHeader()"` da classe `HTTPClient`, passando como primeiro parâmetro o nome do atributo de cabeçalho e como segundo seu conteúdo. Assim sendo, para este projeto precisamos definir que o corpo da nossa requisição trata-se de uma informação em formato JSON, logo, basta configurar o `Content-Type` da requisição como JSON (linha 8).

Com o cabeçalho configurado, precisamos apenas então preencher o corpo da requisição, que será o JSON contendo as informações. Esse JSON vai ser, então, o retorno da função `“getInformations()"`, explicada na seção 2. Logo, chama-se a função correspondente a requisição desejada, da classe `HTTPClient`, no caso, chamasse a função `“POST()"`, que recebe como parâmetro o corpo da requisição, e a realiza, enviando-a para o servidor de destino (linha 12).

Por fim, basta apenas esperar pelo retorno do servidor e testar se a requisição ocorreu foi executada. Se a requisição chegou ao servidor, o método `“POST()"` irá retornar o status da resposta do servidor, podendo ser 200 para uma resposta de sucesso, 400 para erros de cliente, 500 para erro de servidor, entre outros. Caso a requisição não chegue ao servidor, o valor de retorno do método será 0.

## 5. Conclusão

Ao fim do tutorial, com o código pronto e executado na placa, temos a seguinte resposta, retirada do terminal serial da Arduino IDE:

```
Resposta HTTP: 200
{
  "args": {},
  "data": "{\\\"sinals\\\":{\\\"wi-fi\\\":{\\\"wifi0\\\":\\\"cabosat.2g-familia\\\",\\\"wifi1\\\":\\\"extender\\\"}}}",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "identity;q=1,chunked;q=0.1,*;q=0",
    "Content-Length": "70",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "ESP32HTTPClient",
    "X-Amzn-Trace-Id": "Root=1-5fda73eb-5b3e255d0483ac8f546df993"
  },
  "json": {
    "sinals": {
      "wi-fi": {
        "wifi0": "cabosat.2g-familia",
        "wifi1": "extender"
      }
    }
  },
  "origin": "177.188.39.73",
  "url": "http://httpbin.org/post"
}
```

Os principais dados a se verificar nesta resposta acima são:

- A url do servidor que recebeu a requisição;
- O atributo “data”, que indica o corpo da requisição, e se configura da forma como foi proposta na seção 2;
- E também é possível verificar se a informação foi de fato interpretada como um JSON. No caso isso se mostra pelo atributo “json”, com os valores enviados. Caso o texto não fosse interpretado como JSON, este atributo estaria sem valor.