**Como ler json no ESP32**

Para quem não conhece, json é um tipo de estrutura que se assemelha a um dicionário em Python:

{chave: {valor1:1, valor2:2}}

Pode-se ter montes de chaves e valores, sem problemas. Esses valores podem vir do header HTTP ou do corpo de uma requisição GET. Assim sendo, utilizaremos 2 bibliotecas importantes para essa tarefa; a **HTTPClient** e a **ArduinoJson**.

**Pegando o valor do bitcoin através de API**

Já que falamos de bitcoin, vou mostrar a dica com parte do que será mostrado no mini-curso (mas não será só isso, tem o processo de organização e composição do código e todo o desenvolvimento da ideia até chegar no resultado pretendido. Não deixe de fazer o mini-curso para ver o desenrolar de um projeto).

Existem mais formas de fazer esse tratamento. Uma vez que fazemos a requisição HTTP, obtemos um resultado como o exemplo mais acima. Daí o que precisamos fazer é tratar cada valor de cada chave individualmente. Esse processo é chamado de “parsing”, e pode ser a parte mais trabalhosa de um código, principalmente se estivermos fazendo esse parsing manualmente. Utilizando a biblioteca **ArduinoJson**para ler json no ESP32, a captura dos valores será transparente.

Para pegar o valor do bitcoin, utilizaremos a API do Mercado Bitcoin, [cuja documentação é essa](https://www.mercadobitcoin.com.br/api-doc/).

Para ter um parâmetro, vamos ver o resultado de um GET na API:

{"ticker": {"high":"33470.00000000","low":"32515.16000000","vol":"49.23741181","last":"33054.00730000","buy":"33054.00730000","sell":"33191.90847000","open":"33091.31237000","date":1578863707}}

Agora que sabemos o que esperar, podemos iniciar o código.

**Código para ler json no ESP32**

Já há algum tempo que tenho usado o Visual Studio Code para programar embarcados. Como ele tem uma versão para Linux e as qualidades são incomparáveis, [recomendo que você baixe para a sua plataforma operacional, seja Windows, Linux ou Mac](https://code.visualstudio.com/). Depois, basta instalar o PlatformIO pela própria IDE.

Independente da IDE de programação selecionada, não se esqueça de instalar as bibliotecas HTTPClient e ArduinoJson.

**Headers**

Fora das funções, crie o objeto HTTP que será utilizado para fazer a requisição e um array de char com 400 posições. Aproveite para definir também as credenciais da rede WiFi. No exemplo, exponho as credenciais da minha rede de laboratório:

#include <Arduino.h>

#include <WiFi.h>

#include <HTTPClient.h>

#include "ArduinoJson.h"

#define SSID "SuhankoFamily"

#define PASSWD "fsjmr112"

//0 - Criamos o objeto de comunicação HTTP

HTTPClient http;

char json[400] = {0}

No setup, fazemos o processo padrão para conexão com a rede WiFi e inicialização da comunicação serial. Depois, criamos uma variável com o tamanho adequado para comportar a manipulação do json. Para calcular o tamanho correto dessa variável, acesse essa URL do ArduinoJson e passe uma amostra do que será manipulado. Os valores ideais serão exibidos na coluna da direita:

[](https://i0.wp.com/www.dobitaobyte.com.br/wp-content/uploads/2020/01/arduinojson.jpg?ssl=1)

**Setup**

Feito isso, crie o objeto dinâmico do tamanho requerido para a transação. O setup ficará desse jeito:

void setup()

{

    //1 - Para testar, vamos usar a serial

    Serial.begin(9600);

    //2 - iniciamos a conexão WiFi...

    WiFi.begin(SSID,PASSWD);

    //...e aguardamos até que esteja concluída.

    while (WiFi.status() != WL\_CONNECTED)

{

        delay(1000);

        Serial.println(".");

    }

    Serial.println("Conectado!");

  //3 - acesse arduinojson.org/v6/assistant e passe uma amostra pra calcular a capacidade

    constsize\_t capacity = JSON\_OBJECT\_SIZE(1) + JSON\_ARRAY\_SIZE(8) + 146;

    DynamicJsonDocument doc(capacity);

}

**Parsing**

Daí vamos finalmente para o loop e fazemos a requisição HTTP e a manipulação dos valores. Dá pra  ler json no ESP32 de outras formas? Sim, dá, mas esse jeito me pareceu o mais simples. Mas como não gosto de esparramar código pelo programa, criei uma função para fazer a manipulação do json. A função ficou assim:

void resultOfGet(String msg)

{

memset(json,0,sizeof(json));

msg.toCharArray(json, 400);

deserializeJson(doc, json);

JsonObject ticker = doc["ticker"];

const char\* ticker\_high = ticker["high"]; // "33395.00000000"

const char\* ticker\_low = ticker["low"]; // "32911.01001000"

const char\* ticker\_vol = ticker["vol"]; // "29.80139592"

const char\* ticker\_last = ticker["last"]; // "33146.89715000"

const char\* ticker\_buy = ticker["buy"]; // "33005.10011000"

const char\* ticker\_sell = ticker["sell"]; // "33146.89715000"

const char\* ticker\_open = ticker["open"]; // "33094.94851000"

long ticker\_date = ticker["date"]; // 1578889119

Serial.println(ticker\_last);

}

Repare que essa função tem uma rotina bastante simples, mas essencial. Primeiramente, limpamos a variável criada para armazenar o json que eventualmente já foi utilizada e não deve conter valores anteriores. Isso é feito com **memset**, passando o array de char, o valor a preencher e o tamanho do array. Depois, o alimentamos com o resultado da requisição GET, feita no loop. Como o retorno da requisição gera um payload em um objeto String, devemos convertê-lo para array de char. Para isso, utilizamos o método **.toCharArray(\*array,tamanho\_max)**. Esse método pertence ao objeto String passado como parâmetro de função. Em nossa função, esse parâmetro é um ponteiro com o nome **msg**.

Na lista posterior fazemos a desserialização (essa palavra nem existe, mas enfim) do array de char contendo o json. Essa função **deserializeJson** se encarrega de fazer todo o parsing necessário e nos livra de uma tarefa bastante chata. Depois, basta pegar os valores resultantes, criando uma chave com o nome desejado utilizando o **JsonObject**.

Finaliza-se a função com um print na serial, mas a partir desse ponto pode-se fazer o que quiser com os valores.

**Loop**

Mais acima vimos a função **setup()** completa, agora vimos a função que trata o resultado da requisição HTTP. Vamos ver essa requisição na função **loop()**:

void loop()

{

//4 - iniciamos a URL alvo, pega a resposta e finaliza a conexão

http.begin("https://www.mercadobitcoin.net/api/BTC/ticker");

int httpCode = http.GET();

if (httpCode > 0)

{

//Maior que 0, tem resposta a ser lida

String payload = http.getString();

//Serial.println(httpCode);

//Serial.println(payload);

resultOfGet(payload);

}

else {

Serial.println("Falha na requisição");

}

http.end();

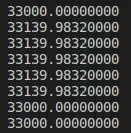
vTaskDelay(pdMS\_TO\_TICKS(60000));

}

Como pode-se notar, ficou bastante limpo e curto o código todo.

Na função **loop()** fazemos inicialmente a configuração do objeto HTTP, passando a URL a utilizar. Em seguida, fazemos a requisição com o método **GET**. Tratamos o retorno e em caso de **status code 200**, chamamos a função criada anteriormente para manipular o resultado. Em qualquer circunstância, não podemos nos esquecer de encerrar a comunicação com **http.end()**. Também não quereremos deixar o loop rodando na velocidade de cada requisição HTTP, portanto, crie um delay. No caso, utilizei o modo ESP32, fazendo uma requisição por minuto. Pode ser menos, pode ser mais, dependendo da real necessidade. Essas coisas todas falarei a respeito no mini-curso do **Arduino Day UNIVAG**.

O resultado na serial será algo como:

[](https://i0.wp.com/www.dobitaobyte.com.br/wp-content/uploads/2020/01/bitcoin_last.jpg?ssl=1)

**Código completo**

Se quiser aproveitar o código, copie daqui invés de pedaço por pedaço das explicações anteriores:

#include <Arduino.h>

#include <WiFi.h>

#include <HTTPClient.h>

#include "ArduinoJson.h"

#define SSID "SuhankoFamily"

#define PASSWD "fsjmr112"

//0 - Criamos o objeto de comunicação HTTP

HTTPClient http;

char json[400] = {0};

StaticJsonDocument<256> doc;

void resultOfGet(String msg){

memset(json,0,sizeof(json));

msg.toCharArray(json, 400);

deserializeJson(doc, json);

JsonObject ticker = doc["ticker"];

const char\* ticker\_high = ticker["high"]; // "33395.00000000"

const char\* ticker\_low = ticker["low"]; // "32911.01001000"

const char\* ticker\_vol = ticker["vol"]; // "29.80139592"

const char\* ticker\_last = ticker["last"]; // "33146.89715000"

const char\* ticker\_buy = ticker["buy"]; // "33005.10011000"

const char\* ticker\_sell = ticker["sell"]; // "33146.89715000"

const char\* ticker\_open = ticker["open"]; // "33094.94851000"

long ticker\_date = ticker["date"]; // 1578889119

Serial.println(ticker\_last);

}

void setup(){

//1 - Para testar, vamos usar a serial

Serial.begin(9600);

//2 - iniciamos a conexão WiFi...

WiFi.begin(SSID,PASSWD);

//...e aguardamos até que esteja concluída.

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println(".");

}

Serial.println("Conectado!");

//3 - acesse arduinojson.org/v6/assistant e passe uma amostra pra calcular a capacidade

const size\_t capacity = JSON\_OBJECT\_SIZE(1) + JSON\_ARRAY\_SIZE(8) + 146;

DynamicJsonDocument doc(capacity);

}

void loop() {

//3 - iniciamos a URL alvo, pega a resposta e finaliza a conexão

http.begin("https://www.mercadobitcoin.net/api/BTC/ticker");

int httpCode = http.GET();

if (httpCode > 0) { //Maior que 0, tem resposta a ser lida

String payload = http.getString();

//Serial.println(httpCode);

//Serial.println(payload);

resultOfGet(payload);

}

else {

Serial.println("Falha na requisição");

}

http.end();

vTaskDelay(pdMS\_TO\_TICKS(60000));

}

Fonte: https://www.dobitaobyte.com.br/como-ler-json-no-esp32/