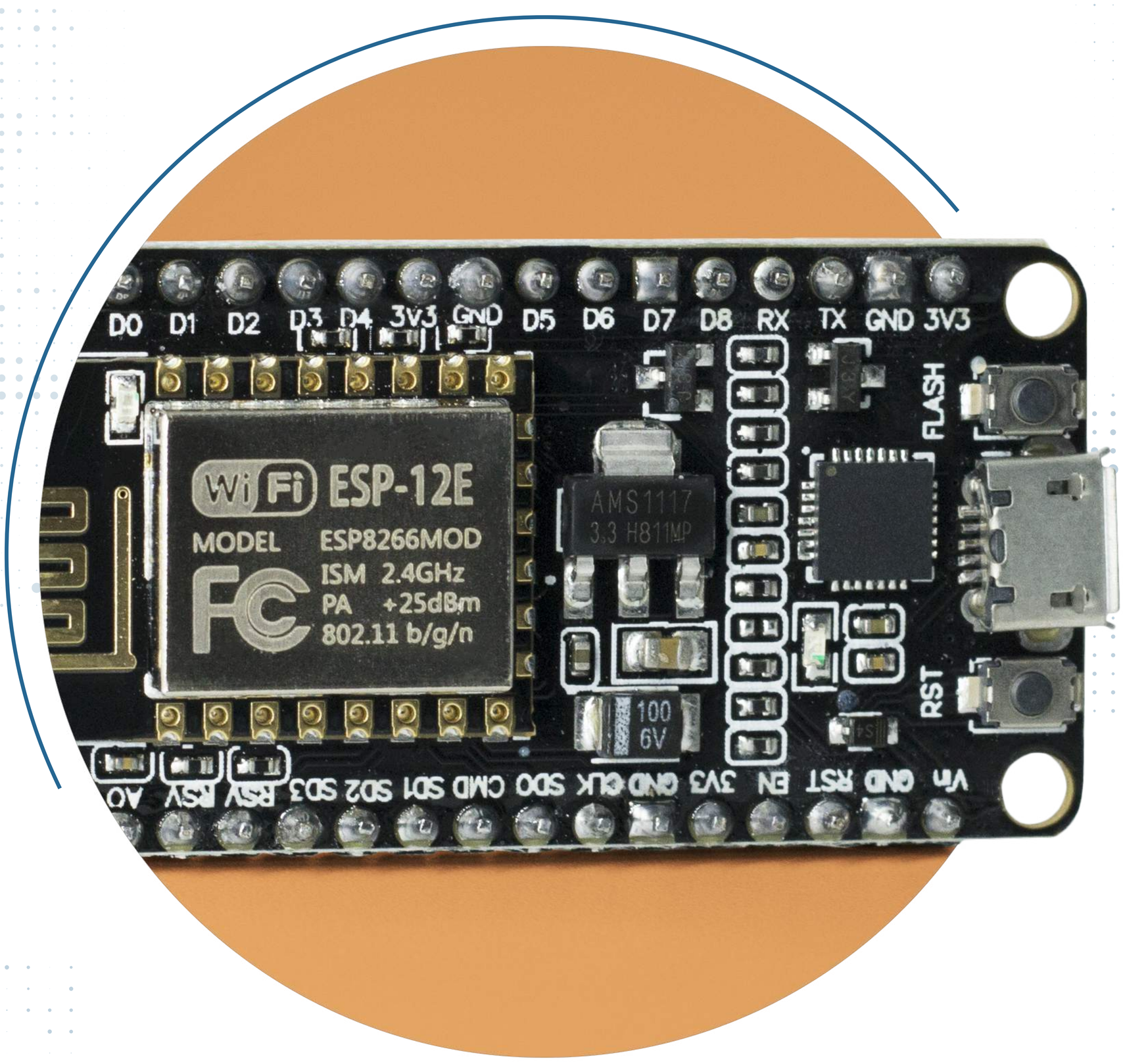
GUIA IOT PARA INICIANTES EM ELETRÔNICA:

Tudo que você precisa saber para começar



FILIPEFLOP

INTRODUÇÃO

Um assunto que anda bastante na moda é a “internet das coisas” ou “IoT”, vinda do inglês *Internet of Things*. O termo foi utilizado pela primeira vez em 1999 para descrever um sistema onde os objetos poderiam ser conectados à internet. Mas o que é exatamente a internet das coisas? Como “funciona”? Como aplicá-la ao nosso dia a dia?

O objetivo deste guia é justamente elucidar essas questões do ponto de vista da área de eletrônica e programação. Vamos explorar melhor o conceito de IoT, mostrar alguns princípios básicos e também mostrar alguns projetos que podem ser feitos utilizando componentes como Raspberry Pi Zero W, ESP8266 NodeMCU, entre outros.

_BOA LEITURA.

O QUE É INTERNET DAS COISAS?

A verdade é que não existe uma unanimidade na definição formal do que é exatamente a internet das coisas, mas ela é bastante utilizada para definir a revolução na utilização da internet que vivemos hoje. Para se ter uma noção, quando criado o sistema IPV4 de IPs (Internet Protocol), em 1983, existiam 4.294.967.296 de endereços possíveis, que na época parecia um numero absurdo. Em 1998, com a entrada da internet em ambientes comerciais, já foi percebida a necessidade de expandir esse número. O novo protocolo, o IPV6, tem disponível aproximadamente 3,4×1038 de endereços possíveis (esse número não vou escrever inteiro aqui não). Isso é quase 5×1028 de endereços por pessoa no planeta!

ONDE SE ENCONTRA A IOT?

Antigamente a internet ligava apenas computadores a computadores de uso exclusivo militar e de algumas poucas faculdades enquanto hoje em dia até um tênis pode ter acesso à internet. Além da necessidade de muito mais endereços de IP, essa revolução na internet traz muitas outras características para o meio. O fato de ter tantos dispositivos conectados à rede literalmente revolucionou o modo em que vivemos.

Na indústria, especialistas chegam a falar que a internet das coisas está trazendo a quarta revolução industrial, também chamada de Indústria 4.0. Nas 'fábricas inteligentes' vários sistemas passaram a ser distribuídos, com processamento no local. Os sensores e atuadores apenas enviam, através da internet, os dados para um software remoto, ou para a nuvem. Isso fez várias empresas se modernizarem, tendo que mudar até o modo como é feita a administração.

No nosso dia-a-dia, geladeiras e carros conectados à rede já são realidade. Além deles, temos um dispositivo IoT que não sai de perto, tão essencial, mas que embarca muita tecnologia e nem nos damos conta, o telefone celular. Os aparelhos de hoje têm milhões de vezes mais poder computacional que a missão que foi para a lua, em 1969.

A tendência é que os dispositivos estejam mais conectados conosco e façam parte do nosso cotidiano de uma forma tão natural que nem percebemos mais. Existe até uma categoria de dispositivos que 'vestimos', como o Google Glass, por exemplo.

Somada à conexão com a internet, outro aspecto interessante da internet das coisas é a quantidade de dados que se consegue a partir dos dispositivos. Atualmente são produzidos algo na ordem de 10^{18} bytes de dados todos os dias. Para essa quantidade de dados, a estatística convencional já não funciona mais e é necessária a utilização de outros termos da moda como "Big Data" e a "Inteligência Artificial". Grande parte disso se dá pelo barateamento dos sensores de diversos tipos. Hoje, qualquer dispositivo embarca um sensor e com ele vêm os dados que compõem esses valores astronômicos.

O QUE VOCÊ PODE FAZER COM A INTERNET DAS COISAS

Felizmente, o barateamento dos dispositivos os torna mais acessível ao público geral. Em nosso blog, falamos das diversas ideias promissoras da internet das coisas só aqui no Brasil. Particularmente, é uma das minhas motivações do meu trabalho: Se até pouco tempo só engenheiros tinham acesso a esse tipo de tecnologia e mesmo assim já chegamos tão longe e tão rápido, onde podemos chegar com pessoas comuns desenvolvendo produtos baseados nessas tecnologias?

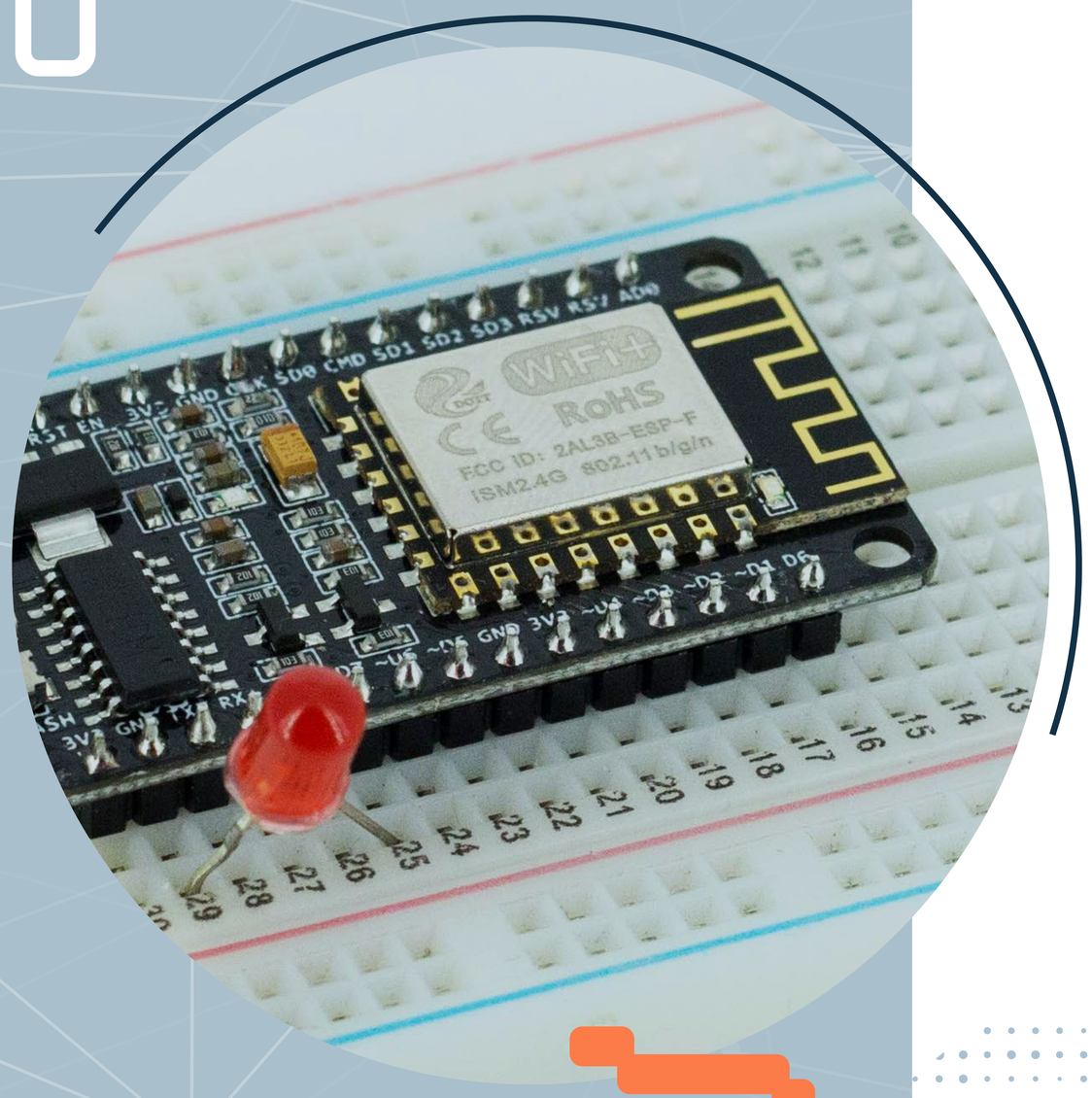
Existem diversas placas de desenvolvimento de projetos e prototipagem bastante acessíveis. Da Raspberry Pi ao ESP8266, qualquer um pode tirar sua ideia do papel e desenvolver uma aplicação IoT baseada na sua experiência de vida para solucionar problemas encontra pelo caminho.

Claro que existem problemas envolvidos com essa revolução, questões de segurança e privacidade dessa chuva de dados são discutidas frequentemente. Por exemplo, seria ético que seu plano de saúde recebesse os dados do seu batimento cardíaco e te cobrasse a mais por isso? No meio dessa discussão, alguns especialistas chegaram até a criar um **manifesto IoT**, para guiar desenvolvedores e tentar direcionar a tecnologia para trazer mais benefícios à sociedade.

De qualquer forma, a internet das coisas é uma realidade, tanto para o presente quanto para o futuro. Até o governo, que geralmente é mais atrasado, já criou um **fundo de incentivo** especialmente para a internet das coisas.

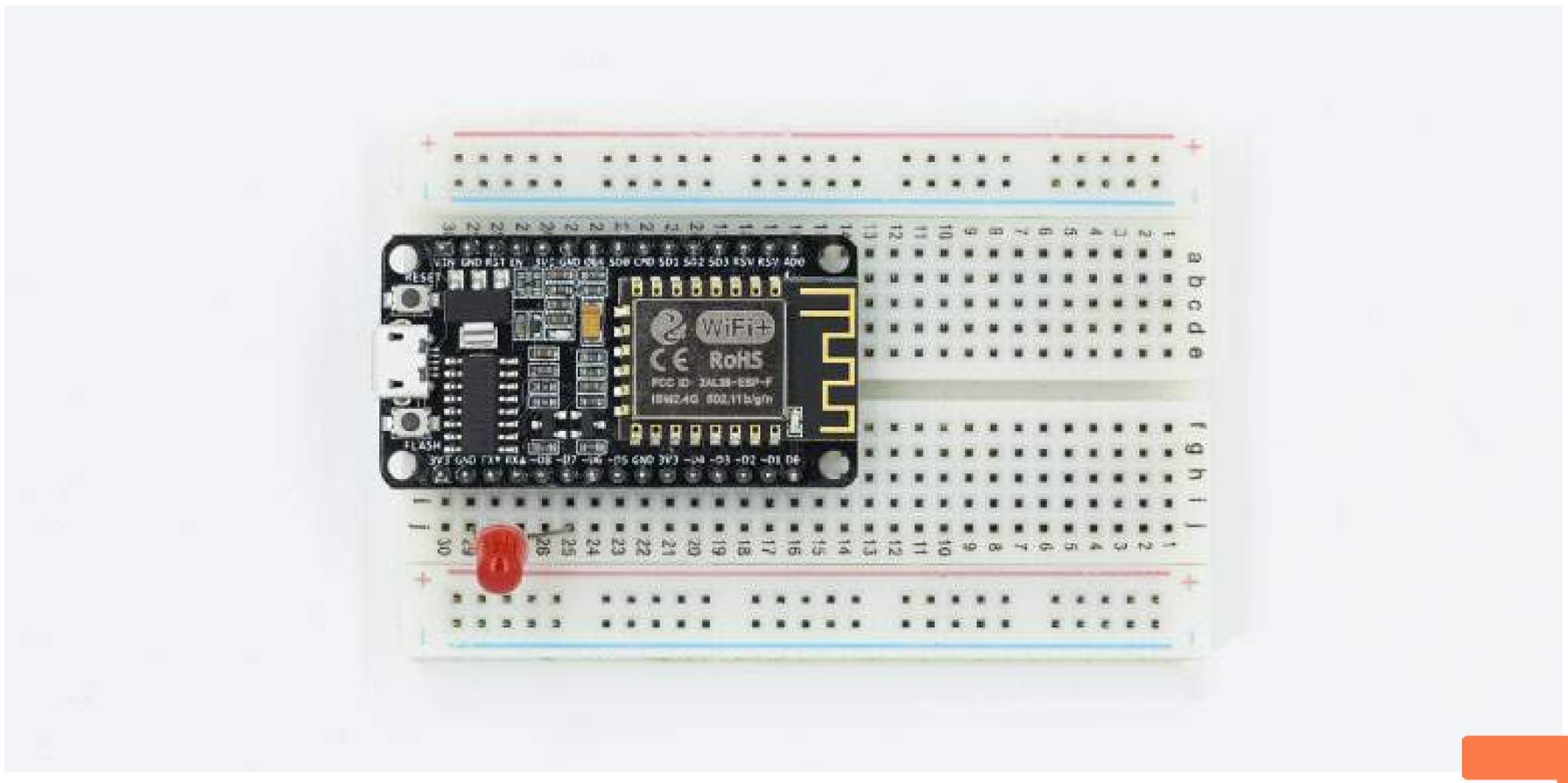
Se você tem alguma ideia que gostaria de pôr em prática não existe mais desculpas para não fazer, a própria internet está repleta de conteúdos (como este guia e o próprio Blog da FilipeFlop) que podem te auxiliar na hora de tirar o projeto do papel.

COMO PROGRAMAR O NODEMCU COM IDE ARDUINO



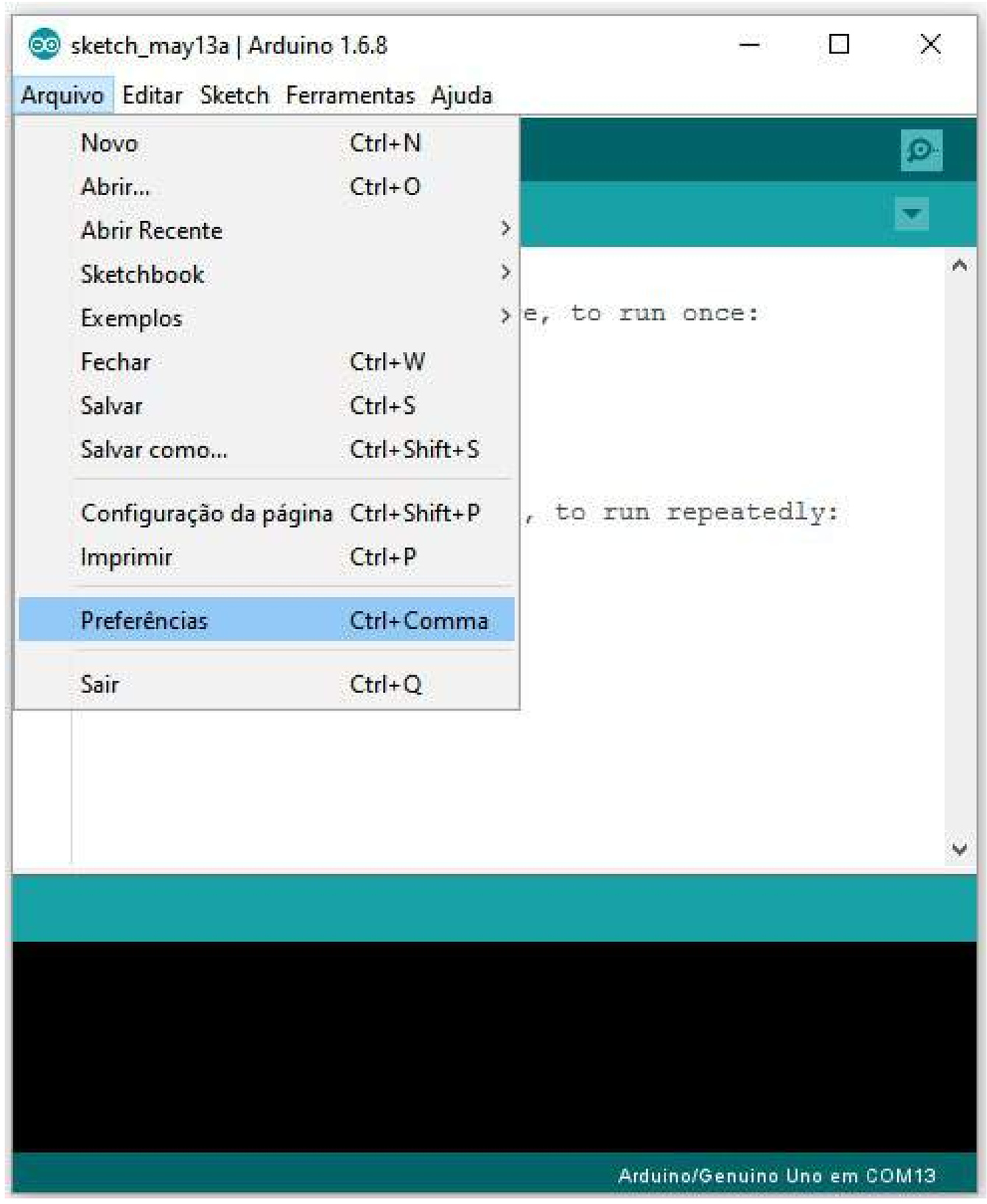
O NodeMCU pode ser programado usando Lua, como vimos no post Como Programar o Módulo ESP8266 NodeMCU. Apesar da facilidade de uso do Lua, você também tem a opção de programar na linguagem padrão do Arduino, utilizando inclusive a mesma IDE. Então aperte os cintos e veja como é fácil programar o NodeMCU com IDE Arduino.

Isso pode ser feito por meio do gerenciador de placas da IDE, onde vamos incluir não só o NodeMCU, mas também outras placas da família ESP8266.



CONFIGURAÇÃO DA IDE DO ARDUINO PARA O NODEMCU

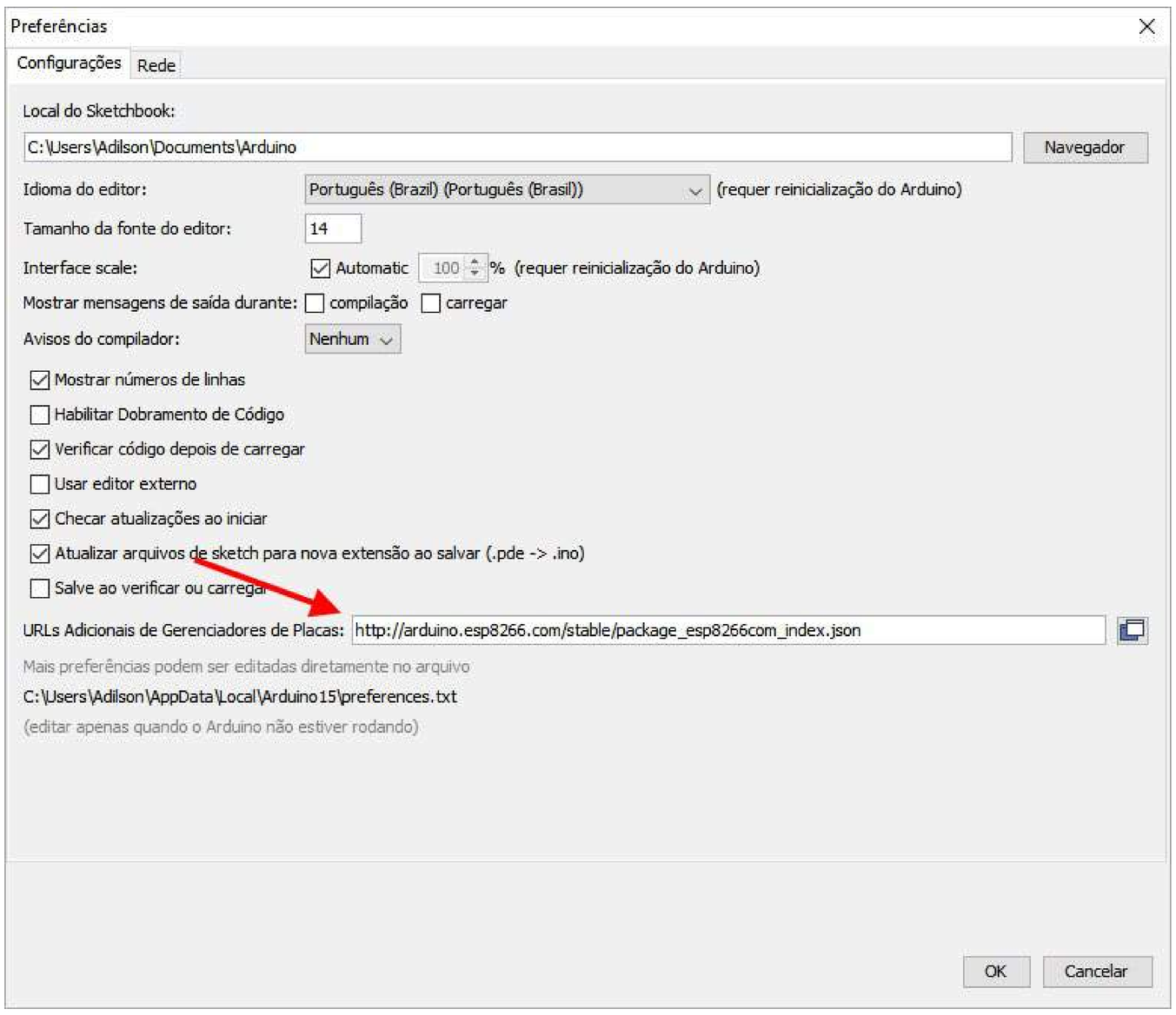
Entre na IDE do Arduino e clique em **Arquivo > Preferências:**



Na tela seguinte, digite este link no campo URLs adicionais de Gerenciadores de Placas:

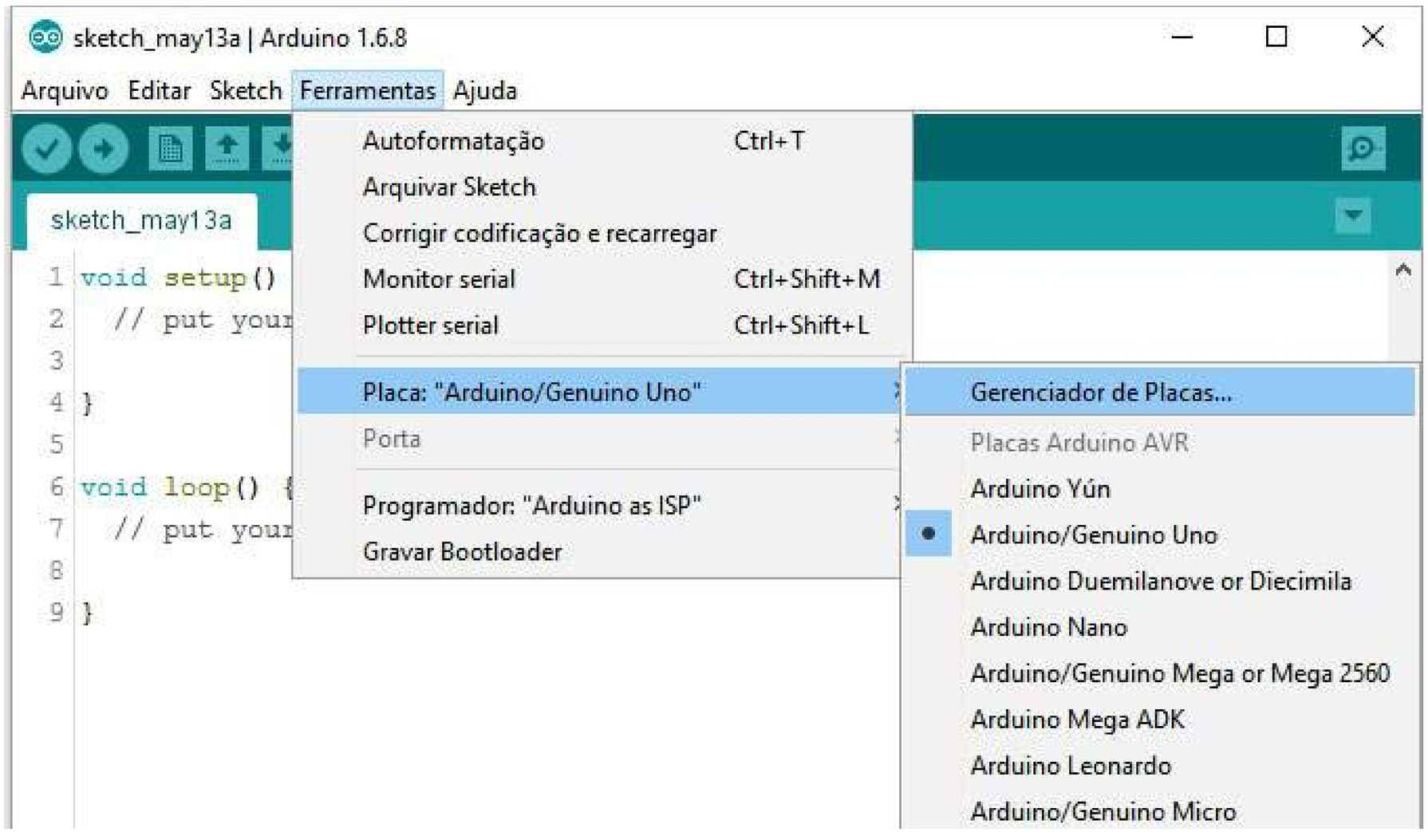
http://arduino.esp8266.com/stable/package_esp8266com_index.json

A sua tela ficará assim:

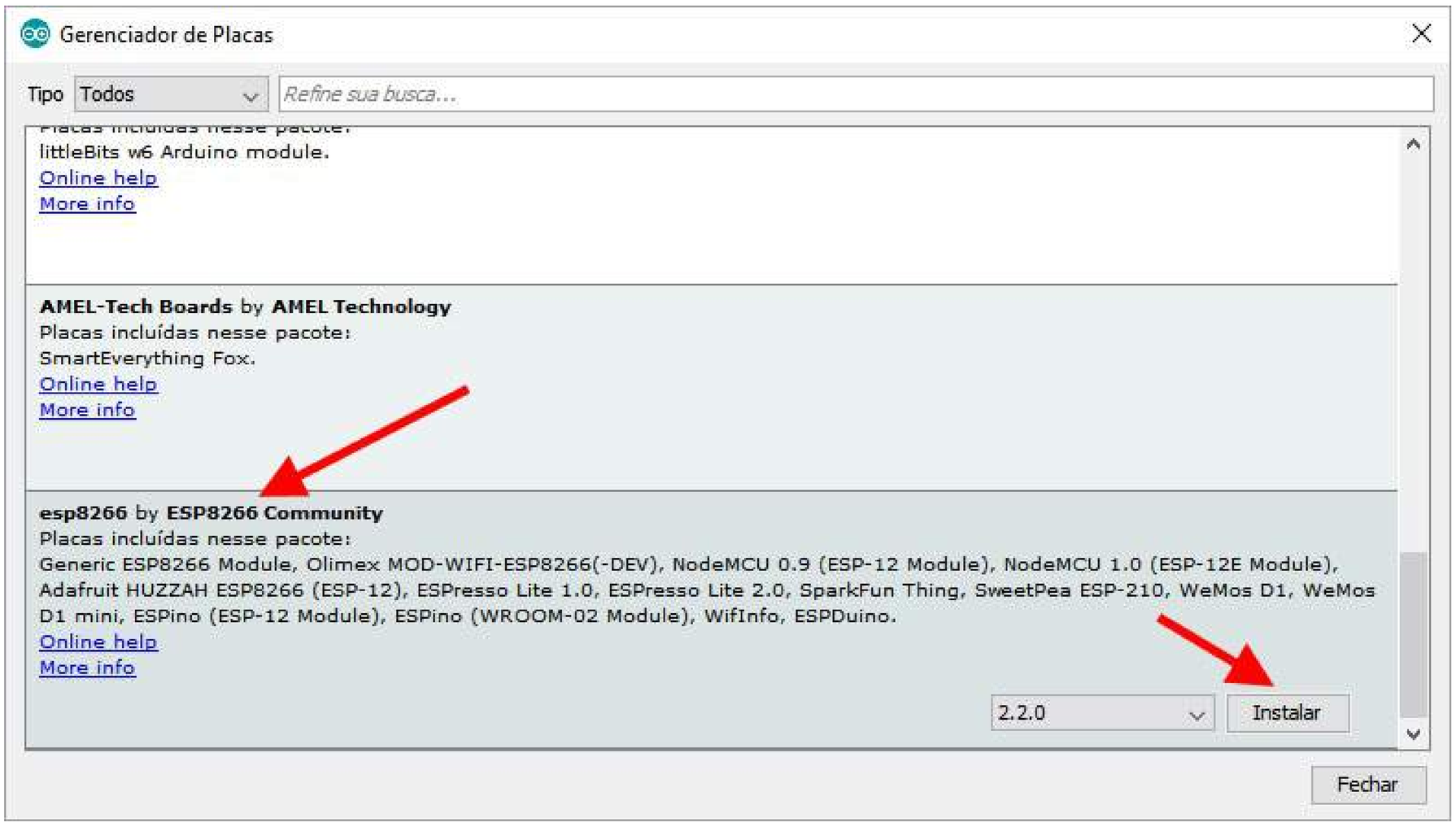


Clique em OK para retornar à tela principal da IDE.

Agora clique em **Ferramentas > Placa > Gerenciador de Placas**



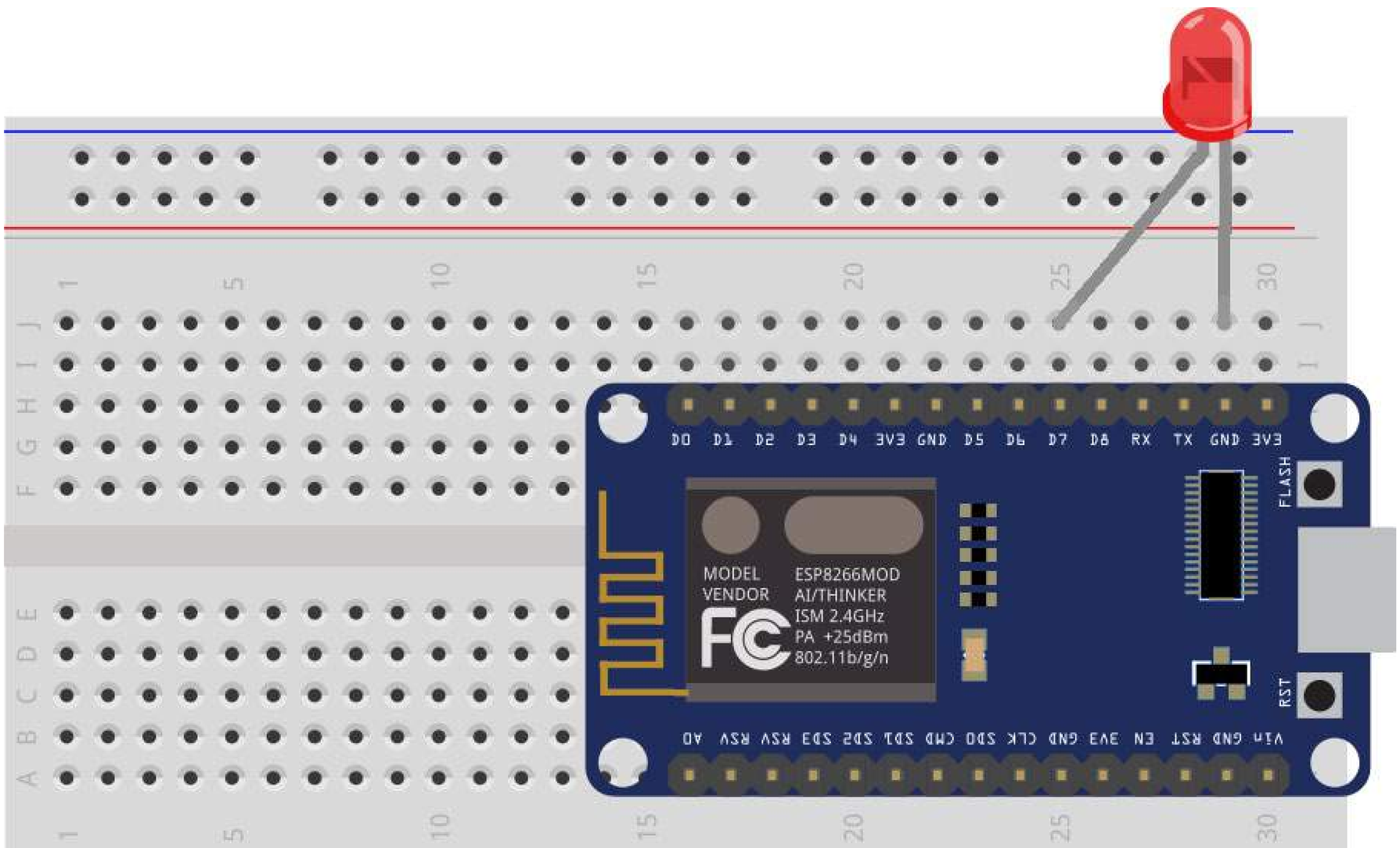
Utilize a barra de rolagem para encontrar o ***esp8266 by ESP8266 Community*** e clique em **Instalar**.



Após alguns minutos as placas da linha ESP8266 já estarão disponíveis na lista de placas da IDE do Arduino.

PROGRAMAR NODEMCU COM IDE ARDUINO

O último passo é programar o NodeMCU com IDE Arduino, e vamos fazer isso montando o circuito a seguir, com um led ligado nos pinos GND e D7, que é o pino correspondente à porta 13 na programação do Arduino:

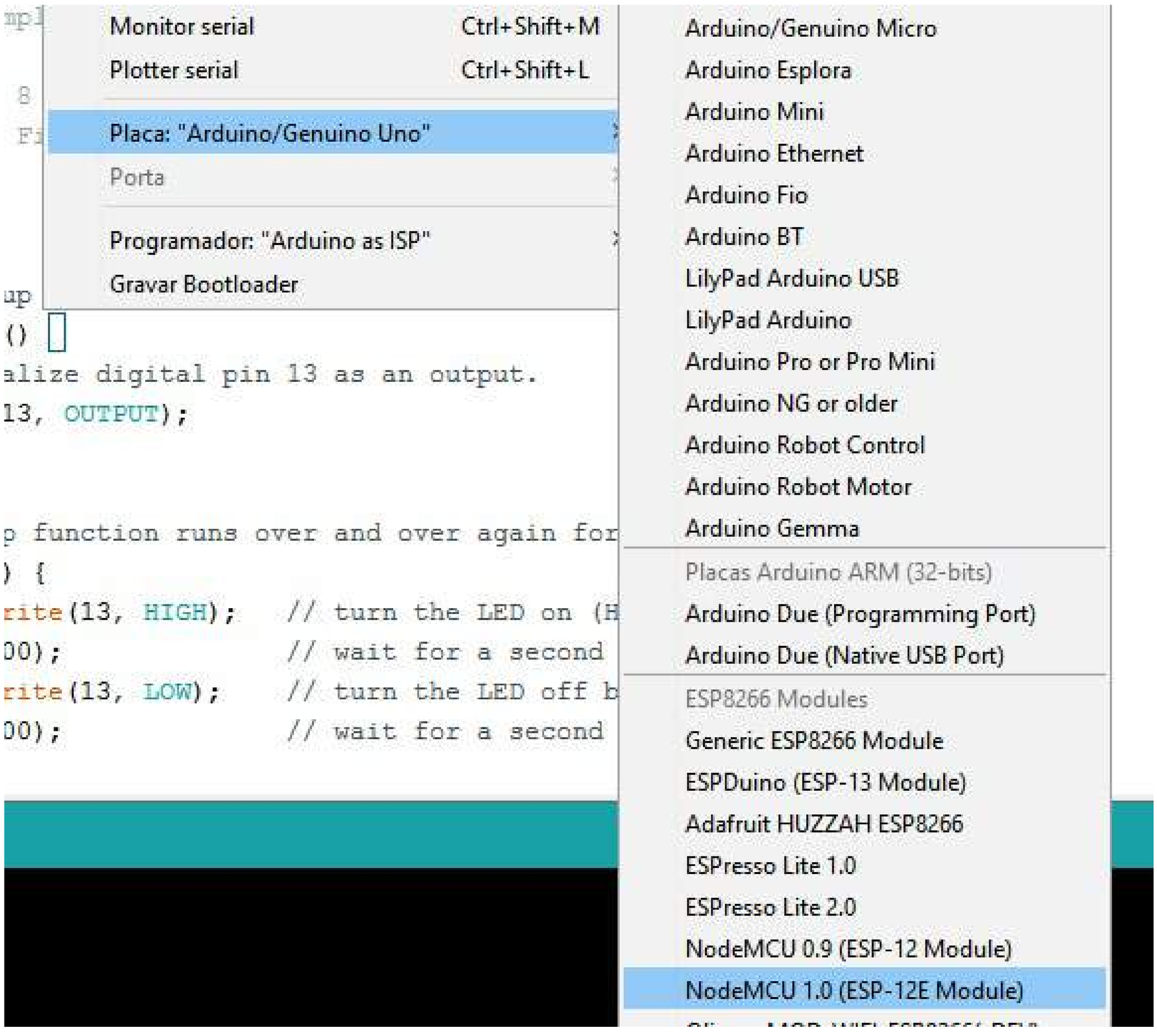


Carregue na IDE o exemplo blink, ou use o programa abaixo:

```
1  void setup() {
2    // Define o pino 13 como saida
3    pinMode(13, OUTPUT);
4  }
5  void loop() {
6  digitalWrite(13, HIGH); // Acende o Led
7  delay(1000); // Aguarda 1 segundo
8  digitalWrite(13, LOW); // Apaga o Led
9  delay(1000); // Aguarda 1 segundo
10 }
```

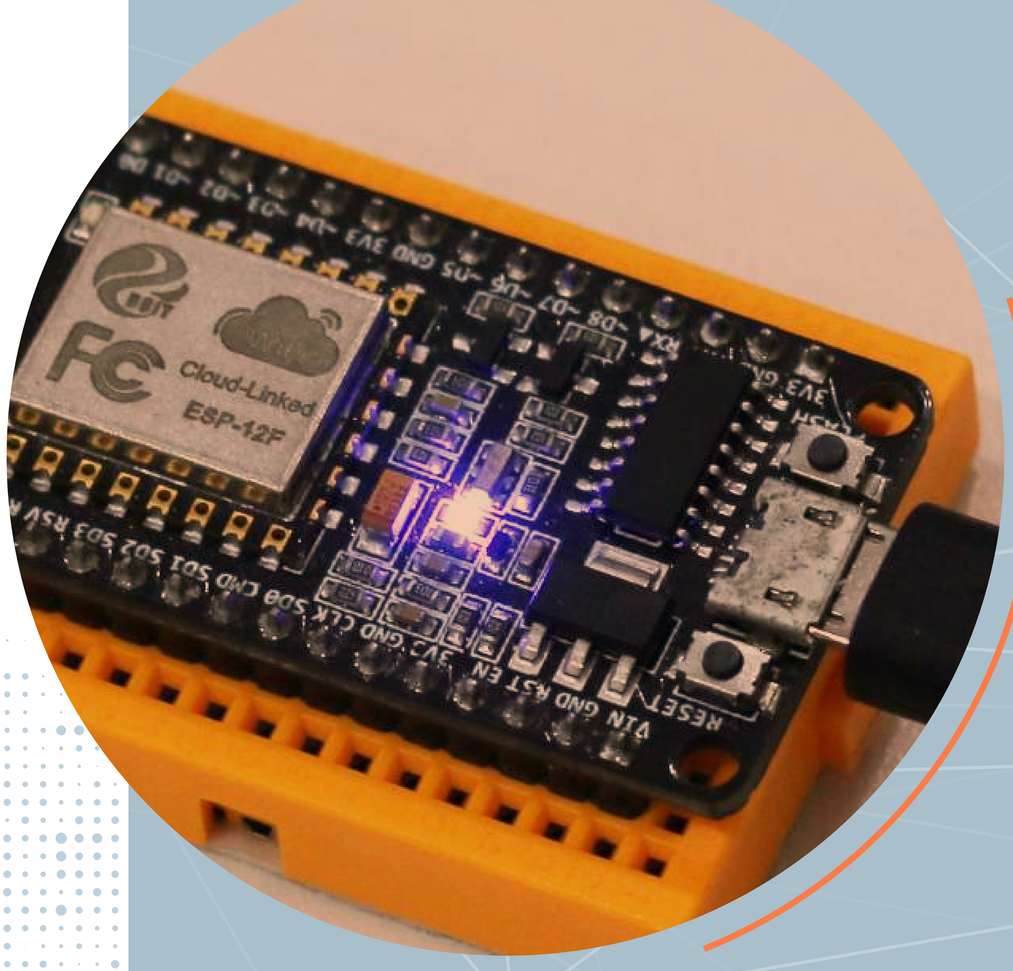
No menu **Ferramentas > Placas**, selecione a placa

NodeMCU 1.0 (ESP 12-E module)



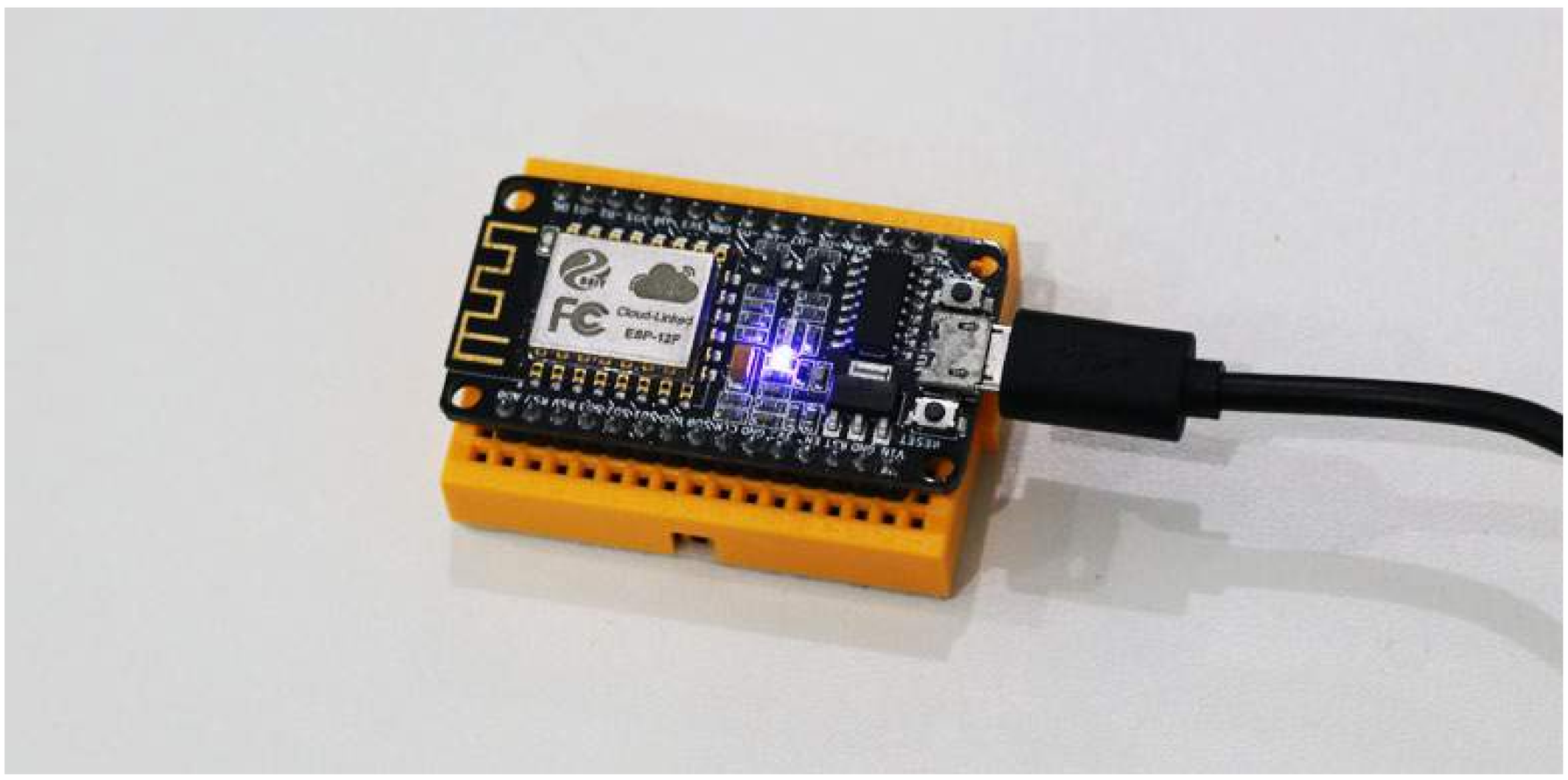
Transfira o programa normalmente para o NodeMCU, do mesmo jeito que você faz com as outras placas Arduino.

No menu da IDE, em **Arquivo > Exemplos**, serão adicionados vários exemplos de uso das placas ESP8266, como webserver, httpclient e DNS, entre outros.



CONTROLE E MONITORAMENTO IOT COM NODEMCU E MQTT

O módulo Wifi ESP8266 NodeMCU é uma das mais interessantes placas / plataformas existentes. A razão disso é simples: em uma pequena placa estão disponíveis I/Os, circuitos de regulação de tensão, conectividade USB para programação (em Lua ou pela Arduino IDE) e conectividade WI-FI (ESP8266 12-E), caracterizando uma placa auto-suficiente para projetos envolvendo IoT, NodeMCU e MQTT.



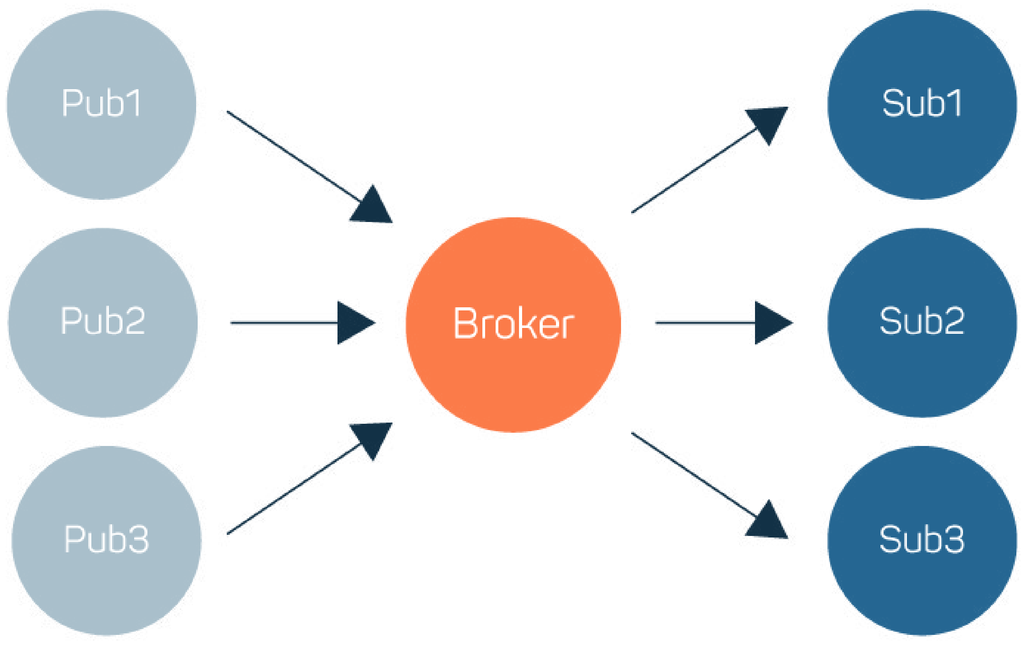
Será mostrada uma das mais interessantes aplicações desta placa: a conectividade com a nuvem através de MQTT para monitoramento e controle de um output, com um diferencial, que é a interface com o usuário usando uma página web.

FUNCIONAMENTO DO MQTT

O **MQTT** (*Message Queue Telemetry Transport*) consiste em um protocolo de mensagens leve, criado para comunicação M2M (*Machine to Machine*). Por exigir muito pouco “poder de fogo” em termos de processamento e banda / consumo de Internet, este é um dos protocolos ideais para dispositivos embarcados. Por esta razão, o MQTT é famoso no conceito IoT (*Internet of Things*).

Uma comunicação MQTT é composta das seguintes partes: há **publishers** (quem irá disponibilizar informações), **subscribers** (quem irá receber as informações) e **Broker** (servidor MQTT, na nuvem / acessível de qualquer lugar do planeta que contenha conexão com a Internet). Teoricamente, não há limite especificado de subscribers e publishers em uma mesma comunicação MQTT, pois o limite nesse aspecto é do servidor em lidar com as conexões.

Em suma: publishers enviam informação para o Broker, subscribers recebem informação do Broker e o Broker gerencia a troca de mensagens. Ou seja, o trabalho pesado fica a cargo do Broker, deixando os sistemas embarcados livre para gerenciar outras coisas.



Sem entrar no mérito da especificação do protocolo MQTT (ou seja, byte a byte do protocolo), grosso modo uma mensagem MQTT publicada / enviada possui duas partes importantes:

- **Tópico** – "chave" / identificação da informação publicada. É usado para direcionar a informação publicada / enviada a quem assina (quem "dá subscribe") no tópico. O tópico consiste de uma string (por exemplo: MQTTTesteTopico);

- **Payload** – informação que deseja enviar (propriamente dita).

Um publisher, conectado ao Broker (servidor MQTT), envia/publica as informações em um dado momento. Os subscribers, assim como os publishers, também estão conectados aos brokers e "escutando" mensagens trafegadas com o tópico-alvo. Quando uma mensagem com o tópico alvo é publicada, automaticamente são direcionadas aos subscribers.

Em outras palavras: uma solução em IoT que usa MQTT possui somente um servidor (Broker), sendo todo o restante composto de clients MQTT.

Outra informação importante é que um mesmo client MQTT pode ser subscriber e publisher de diversos tópicos.

COMO UTILIZAR O NODEMCU E MQTT

O NodeMCU pode ser programado para interagir com um broker MQTT, ou seja, ele pode ser programado para ser um client MQTT. Antes de ser programado para isso, é necessário:

1. Preparar a IDE Arduino para programar o NodeMCU. Veja como fazer isso no post Como programar o NodeMCU com IDE Arduino;

2. Baixar e instalar a biblioteca pubsubclient. Para baixar, visite o GitHub do projeto.

Feito isso, devemos escolher um broker MQTT para utilizar. Há inúmeros brokers disponíveis para uso (tanto públicos / sem custo quanto privados / pagos). Dos públicos, eu recomendo fortemente o iot.eclipse.org (broker oficial da equipe que mantém o MQTT), que irá funcionar perfeitamente com o nosso projeto de NodeMCU e MQTT.

Enfim, agora é possível usarmos o NodeMCU como um client MQTT! Para isso, será feito um projeto que permite controlar e monitorar um output do NodeMCU via MQTT. O output em questão é o próprio LED da placa (que, neste caso, está ligado diretamente ao output D0).

IMPORTANTE

1. O LED possui acionamento em lógica invertida, ou seja, para acendê-lo é preciso enviar o estado LOW para D0 e para

apagá-lo é necessário enviar estado High. Isto ocorre pois o output está ligado ao catodo do LED na placa.

2. O ID MQTT no código serve como identificador para o broker, o permitindo gerenciar as conexões. Se você escolher um ID MQTT que já está sendo utilizado, a conexão do mesmo será interrompida para a sua ser estabelecida. Isto pode gerar grandes transtornos para ambos os dispositivos com mesmo ID MQTT. Desta forma, recomendo que o ID MQTT seja escolhido como algo aleatório (assim garante-se unicidade do mesmo).

3. Como não foi utilizado nenhum componente além da própria NodeMCU, não há circuito esquemático envolvido (ou seja, em termos de hardware é preciso somente do NodeMCU!)

PROGRAMA NODEMCU E MQTT

Vamos à programação!

> **FAÇA O DOWNLOAD DO CÓDIGO DIRETO PELA IDE ARDUINO AQUI**

Desta forma, seu NodeMCU vira um client MQTT acessível de qualquer lugar do planeta!

INTERAÇÃO COM O NODEMCU

Vimos até agora que é possível transformar a incrível placa NodeMCU em um client MQTT acessível por todo o planeta. Mas e para interagir com ele (ligar e desligar o output), como faremos?

Para isso, desenvolvemos uma interface web que pode ser baixada clicando aqui. Você pode hospedar essa interface em QUALQUER servidor web ou mesmo rodar no seu próprio computador / rodar localmente (desde que o computador possua conexão com Internet, claro)!

Esta interface web é basicamente um websocket que se comunica diretamente com o broker, por isso pode estar rodando em qualquer lugar com disponibilidade de Internet que funciona.

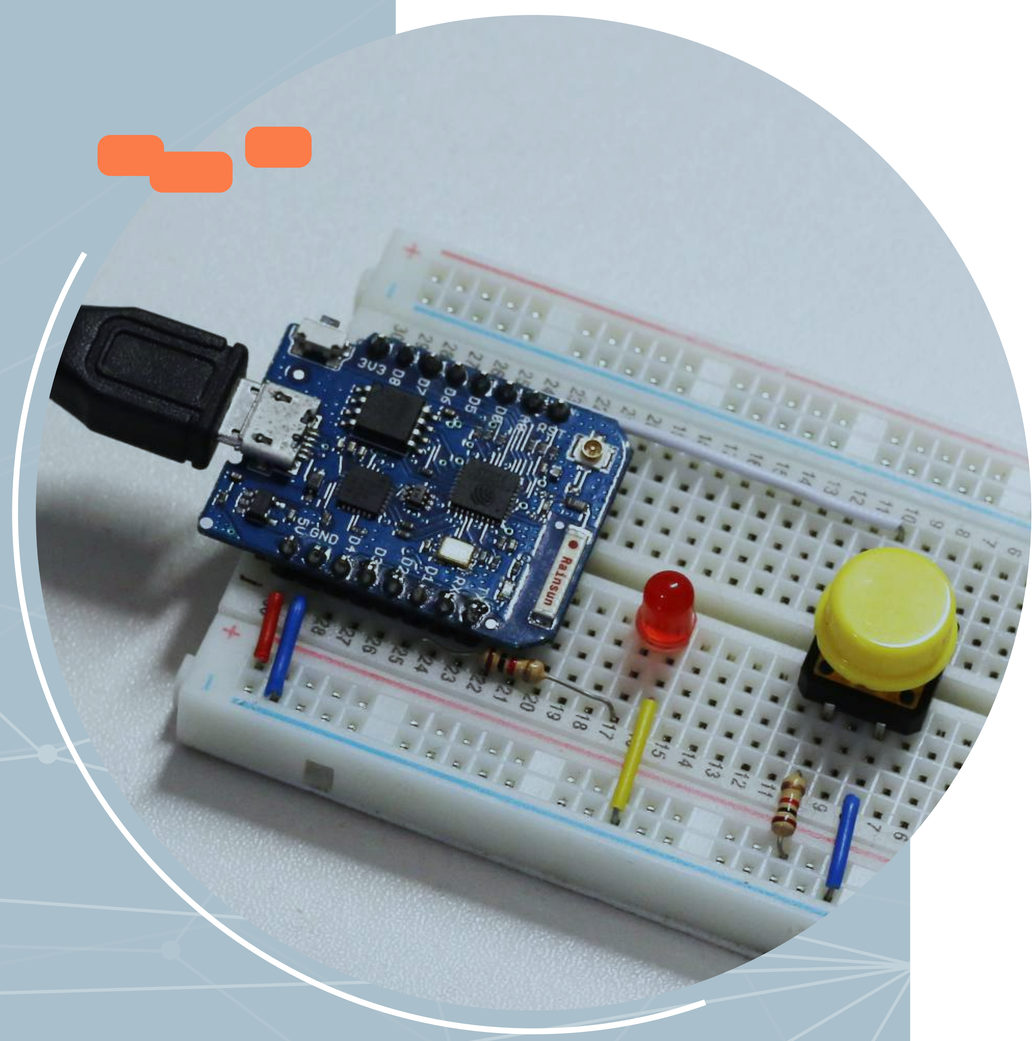
Observe a figura abaixo:

PARA FUNCIONAR

1. Definir as strings dos tópicos de publish e subscribe e clicar em “Conectar”. Aguardar a mensagem Conectado ao Broker! aparecer na sessão “Debug / respostas do servidor”.

2. Clicar em Ligar e Desligar para comandar o LED da placa NodeMCU. Observe que na sessão “Status do output” irá constar o estado atual do LED. Este estado é recebido do próprio NodeMCU, via MQTT (o que torna este controle um controle em tempo real).

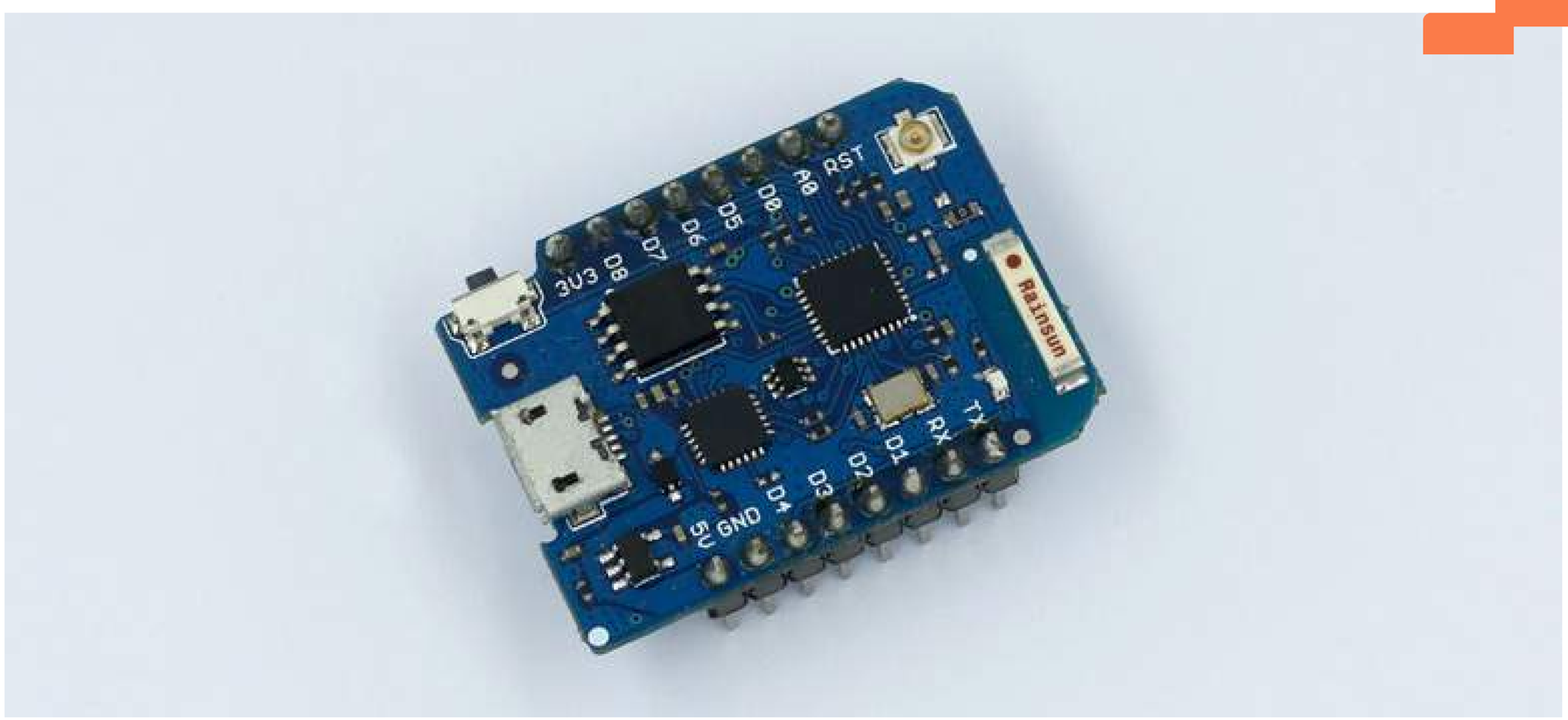
3. Se divertir com o NodeMCU e MQTT!



ESP8266: GRAVANDO DADOS PERMANENTES NA MEMÓRIA FLASH

Vamos mostrar como tomar proveito da memória flash SPI encontrada nos módulos ESP8266 para armazenamento de dados e arquivos, utilizando seu sistema de arquivos SPIFFS. Com os conhecimentos adquiridos aqui, você poderá ir além, criando, por exemplo, um datalogger de sensores e até mesmo hospedando uma página web.

Para melhor exemplificar os conceitos de utilização da memória Flash, utilizaremos a placa Wemos D1 Mini Pro e faremos dois exemplos.



O primeiro exemplo liga/desliga um LED e salva seu estado na memória. O estado ficará salvo mesmo quando o Wemos for desligado. Ao ligar novamente o Wemos, o sistema busca na memória o estado do LED acionando o mesmo de acordo com seu estado.

No segundo exemplo iremos mostrar como carregar um arquivo index.html contendo algumas linhas de HTML na memória flash e visualizar esse arquivo através de um navegador de internet.

BIBLIOTECA SPIFFS

Para utilizar o sistema de arquivos SPIFFS (SPI Flash File System) é necessário incluir nos programas a biblioteca FS.h, que já é nativa da IDE Arduino com suporte a placas ESP8266.

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

A biblioteca nos possibilita algumas manipulações básicas de arquivos como open, close, read e write e as funções básicas da biblioteca são mostradas abaixo.

Inicia o sistema de arquivos.

```
SPIFFS.begin()
```

Apaga todos os arquivos na região de memória reservada.

```
SPIFFS.format()
```

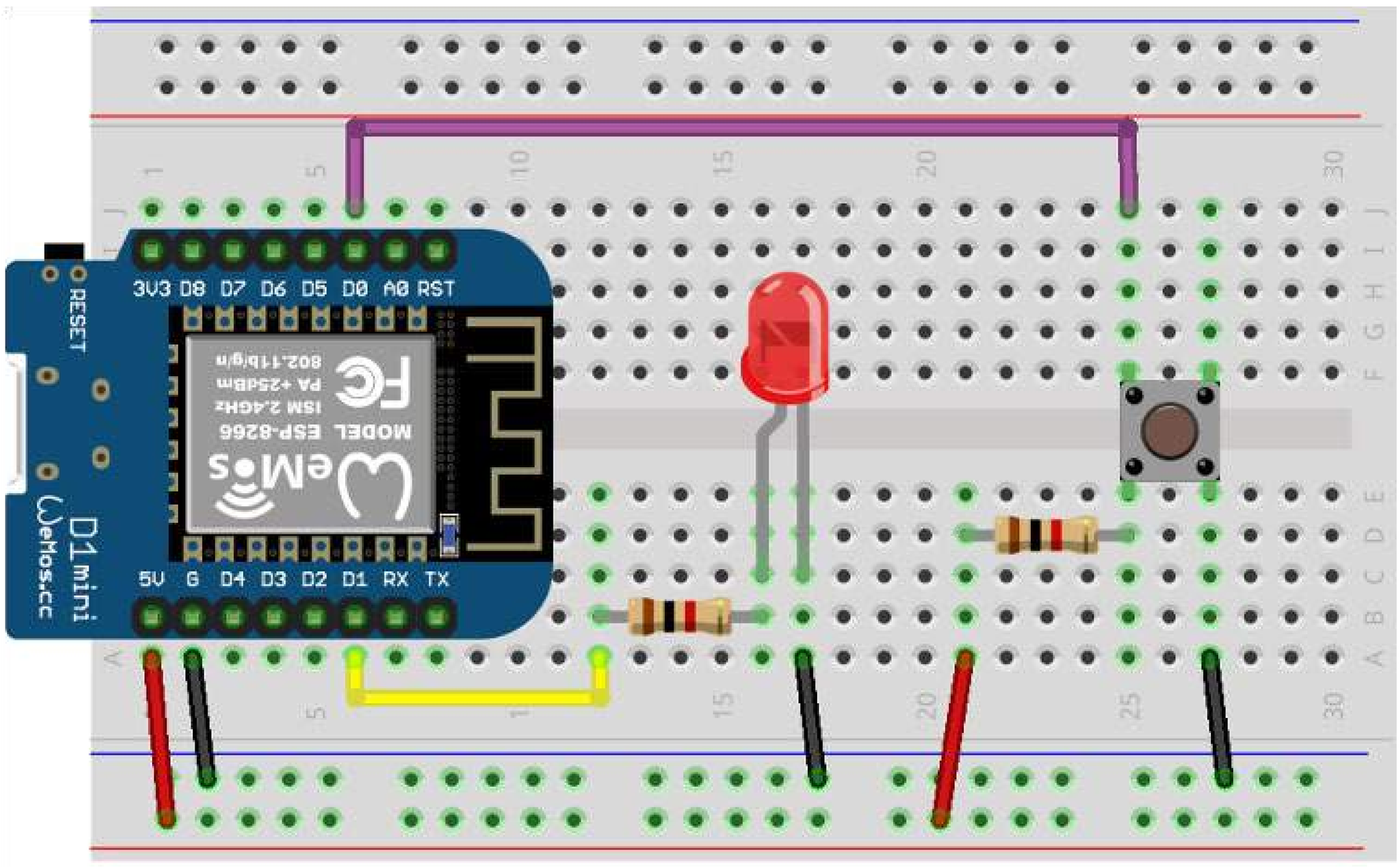
Abre um arquivo no modo selecionado. O nome do arquivo deve ser um diretório iniciado com uma "/", exemplo: "/log.txt". O modo pode ser "r", "w", "a", "r+", "w+", "a+", seguindo o mesmo padrão da função fopen da linguagem C.

```
SPIFFS.open(arquivo, modo)
```

Para saber mais sobre a memória Flash dos módulos ESP8266 e seu sistema de arquivos acesse aqui.

EXEMPLO 1: SALVANDO ESTADO DO LED NA MEMÓRIA

Para realizar o exemplo 1 deste tutorial, utilizamos um circuito bem simples que consiste de um LED, resistores 1K e uma chave push-button. Os componentes são conectados conforme o esquema abaixo:



O primeiro exemplo utiliza-se de um arquivo para salvar o estado on/off de um LED. Apertando o botão podemos mudar o estado. Mesmo se a placa for desligada o estado ficará salvo na memória. Quando ligamos novamente a placa o sistema busca o estado salvo na memória e atualiza o LED de acordo.

Vale notar que o programa deve ser gravado duas vezes. Na primeira vez a linha 62 (***writeFile("off", "/state.txt")***)

deve ser descomentada, criando um arquivo na memória. Na segunda e demais programações, a linha deve ser comentada, pois o arquivo já existirá na memória. Veja o exemplo 1 abaixo:

```
#include <FS.h>

int led = D1;
int chave = D0;

/**
  * @desc escreve conteúdo em um arquivo
  * @param string state - conteúdo a se escrever no arquivo
  * @param string path - arquivo a ser escrito
*/
void writeFile(String state, String path) {
  //Abre o arquivo para escrita (“w” write)
  //Sobreescreve o conteúdo do arquivo
  File rFile = SPIFFS.open(path,”w+”);
  if(!rFile){
    Serial.println(“Erro ao abrir arquivo!”);
  } else {
    rFile.println(state);
    Serial.print(“gravou estado: “);
    Serial.println(state);
  }
  rFile.close();
}

/**
  * @desc lê conteúdo de um arquivo
  * @param string path - arquivo a ser lido
  * @return string - conteúdo lido do arquivo
*/
String readFile(String path) {
  File rFile = SPIFFS.open(path,”r”);
  if (!rFile) {
    Serial.println(“Erro ao abrir arquivo!”);
  }
  String content = rFile.readStringUntil(‘\r’); //desconsidera ‘\r\n’
  Serial.print(“leitura de estado: “);
  Serial.println(content);
  rFile.close();
  return content;
}

/**
  * @desc inicializa o sistema de arquivos
*/
```

.....

```
void openFS(void){
  //Abre o sistema de arquivos
  if(!SPIFFS.begin()){
    Serial.println("\nErro ao abrir o sistema de arquivos");
  } else {
    Serial.println("\nSistema de arquivos aberto com sucesso!");
  }
}

void setup() {
  pinMode(led, OUTPUT);
  pinMode(chave, INPUT);
  Serial.begin(9600);
  openFS();

  // no primeiro upload de programa o arquivo state.txt deve ser criado com o conteúdo "off"
  // no segundo upload a linha deve ser comentada.
  //writeFile("off", "/state.txt");

  // verifica o último estado do LED e ativa de acordo
  String state = readFile("/state.txt");
  if(state == "on")
  {
    digitalWrite(led, HIGH);
  }
  else if(state == "off")
  {
    digitalWrite(led, LOW);
  }
}

void loop() {
  /*
   * verifica o estado anterior salvo no arquivo,
   * ativa o LED de acordo
   * e escreve novo estado no arquivo.
   */
  if(digitalRead(chave) == LOW)
  {
    String state = readFile("/state.txt");
    if(state == "off")
    {
      writeFile("on", "/state.txt");
      digitalWrite(led, HIGH);
    }
    else if(state == "on")
    {
      writeFile("off", "/state.txt");
      digitalWrite(led, LOW);
    }
    while(digitalRead(chave) == LOW);
  }
}
```

EXEMPLO 2: SALVANDO UMA PÁGINA WEB INDEX.HTML NA MEMÓRIA FLASH

O exemplo 2 busca um arquivo index.html salvo na memória e exibe essa página em um navegador. Veja o código abaixo:

```
#include <FS.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = “Nome da rede Wifi”;
const char* password = “Senha da rede Wifi”;

String webPage = “”;

ESP8266WebServer server(80);

void handleRoot() {
  server.send(200, “text/html”, webPage);
}

void readFile(void) {
  //Faz a leitura do arquivo HTML
  File rFile = SPIFFS.open(“/index.html”,”r”);
  Serial.println(“Lendo arquivo HTML...”);
  webPage = rFile.readString();
//  while(rFile.available()) {
//    String line = rFile.readStringUntil(‘\n’);
//    webPage += line;
//  }
  Serial.println(webPage);
  rFile.close();
}

void setup() {
  Serial.begin(9600);
  SPIFFS.begin();

if(SPIFFS.exists(“/index.html”))
{
  Serial.println(“\n\nfile exists!”);
}
else Serial.println(“\n\nNo File :(“);
```

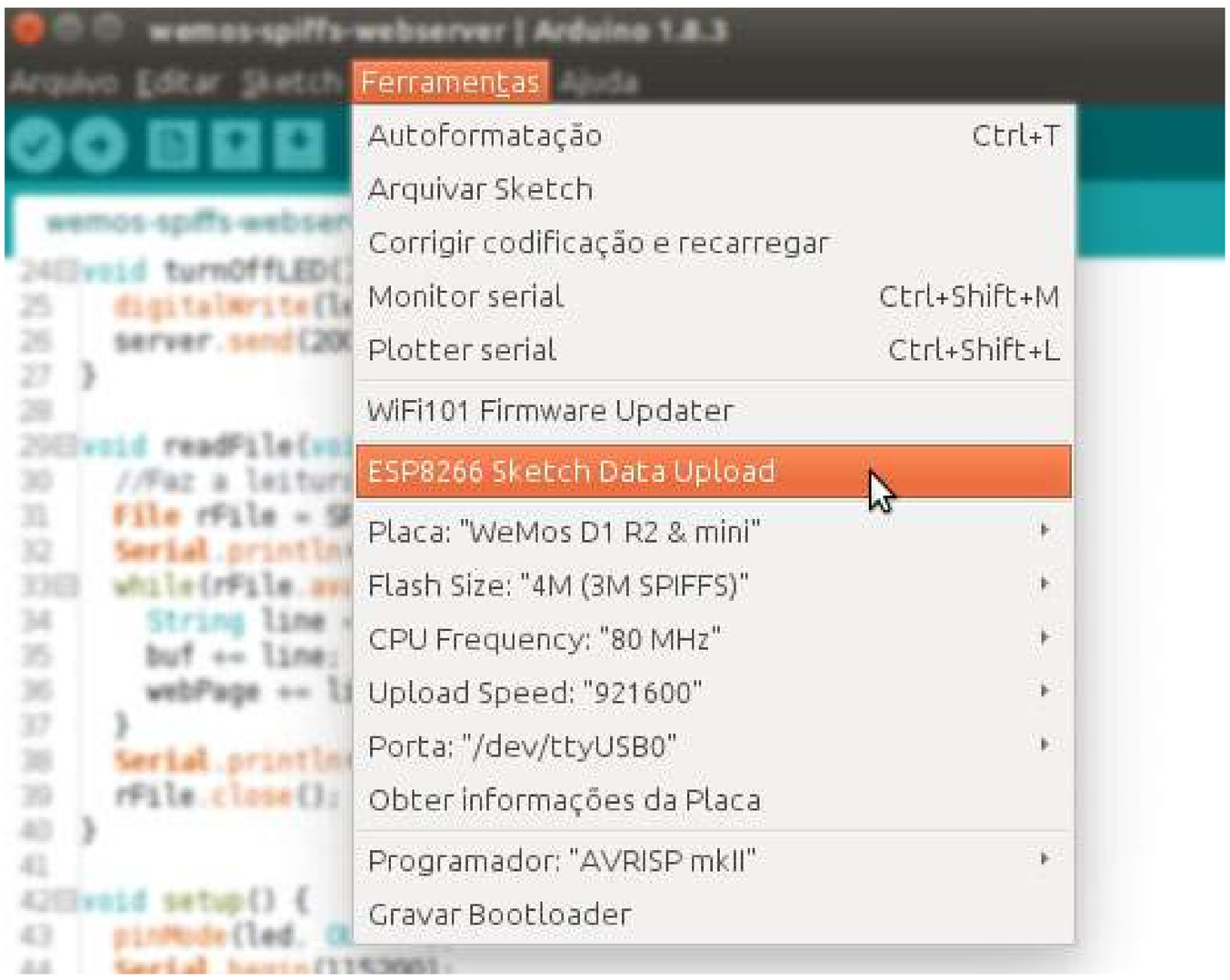
```
39
40    readFile();
41
42    WiFi.begin(ssid, password);
43    // Wait for connection
44    while (WiFi.status() != WL_CONNECTED) {
45      delay(500);
46      Serial.print(“.”);
47    }
48      Serial.println(“”);
49      Serial.print(“Connected to “);
50      Serial.println(ssid);
51      Serial.print(“IP address: “);
52      Serial.println(WiFi.localIP());
53
54      server.on(“/”, handleRoot);
55
56      server.begin();
57    }
58
59    void loop() {
60      server.handleClient();
61    }
```

Para que o código acima funcione é necessário gravar o arquivo **index.html** na memória flash do Wemos. Para isso existe uma ferramenta para IDE Arduino que nos possibilita carregar arquivos inteiros para memória flash.

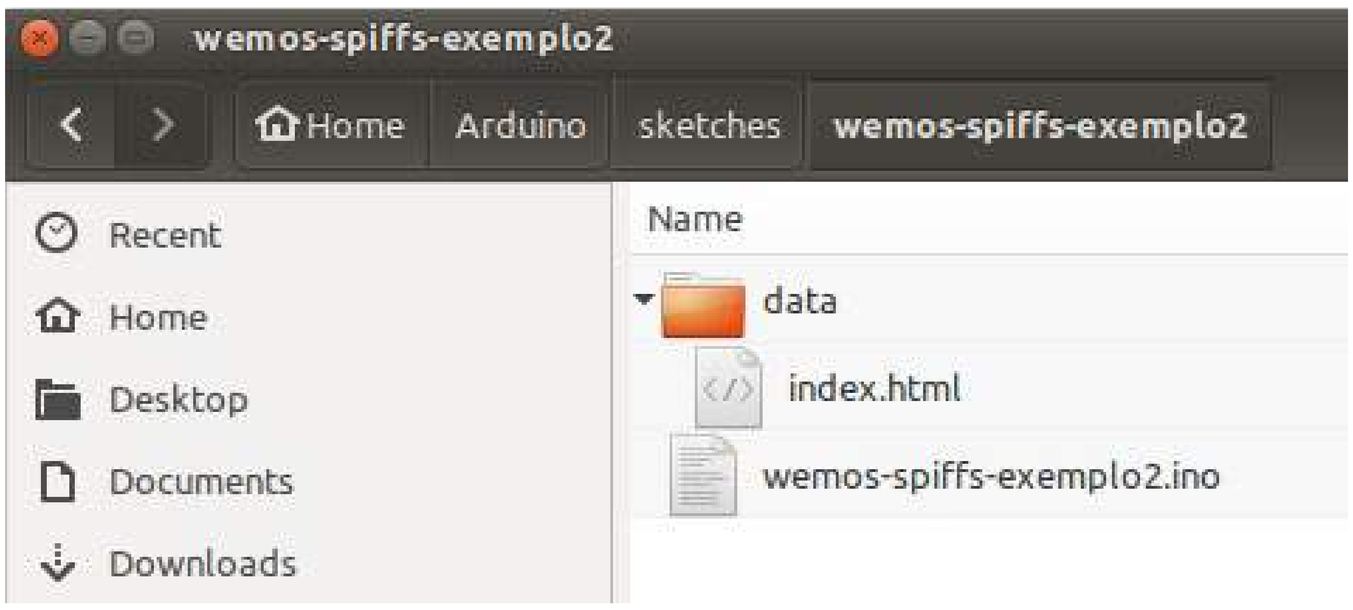
Primeiramente baixe a ferramenta neste link. Extraia o conteúdo para o diretório ***tools*** da pasta de instalação do Arduino. Neste caso, o sistema Linux. Portanto, o caminho do diretório ficou assim:

../Arduino/tools/ESP8266FS/tool/esp8266fs.jar

Reinicie a IDE Arduino e você deverá ver a nova ferramenta no menu Ferramentas.



Essa ferramenta carrega qualquer arquivo que estiver dentro de uma pasta chamada “data”. A pasta data deve estar dentro da pasta do seu sketch.



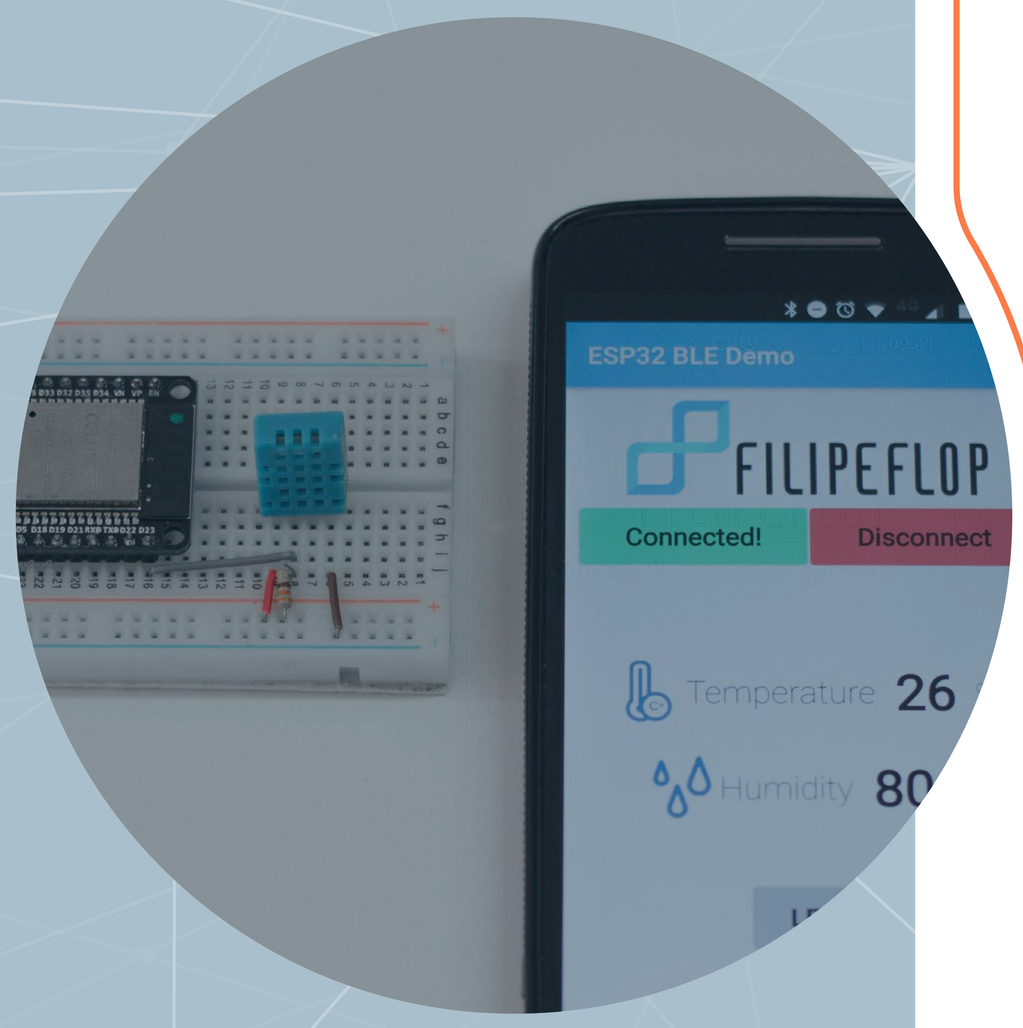
Coloque o arquivo ***index.html*** abaixo dentro da sua pasta data.

```
<!DOCTYPE html>
<html>
  <body>
     <h1>Hello Wemos SPIFFS!</h1>
  </body>
</html>
```

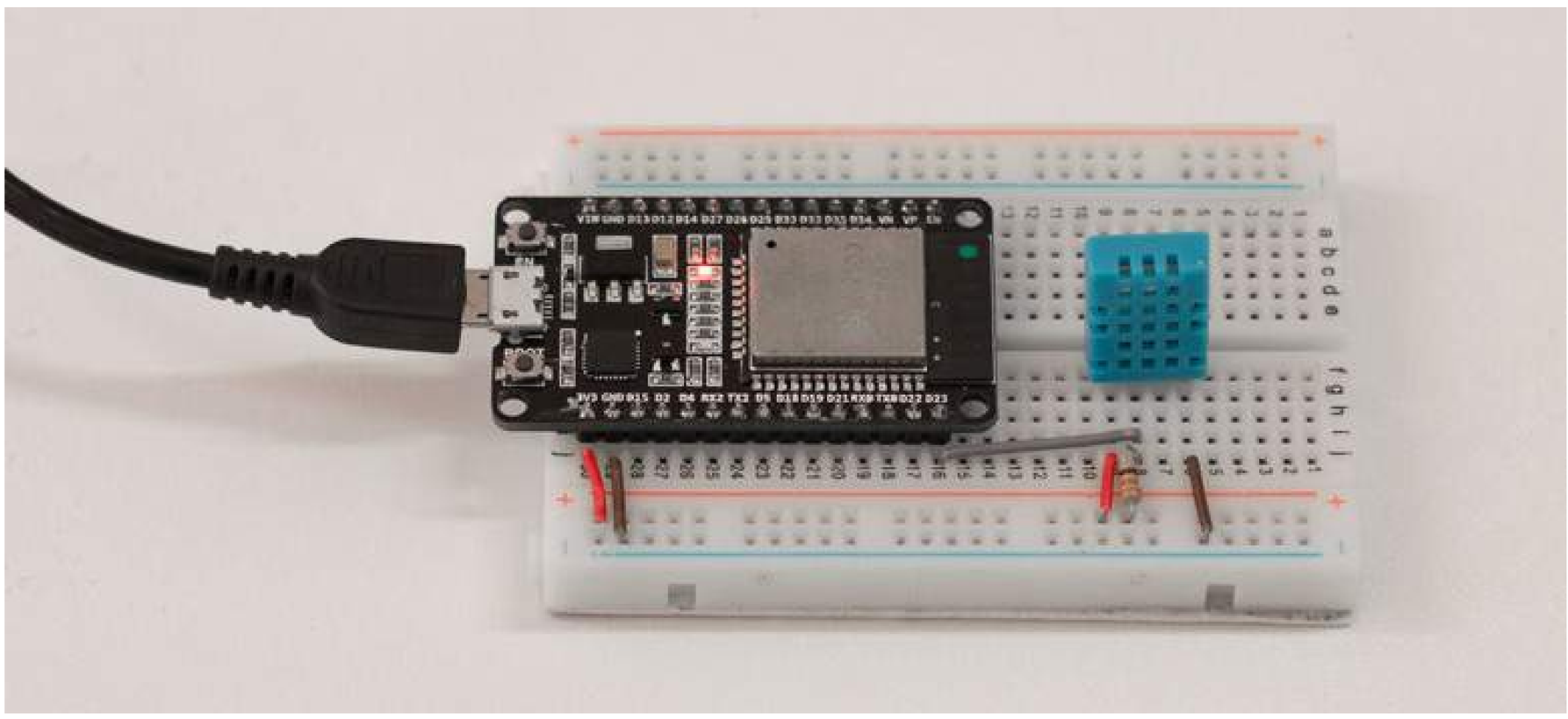
E então grave o arquivo utilizando a ferramenta mostrada acima. Mais informações sobre a ferramenta de upload de arquivos aqui.

Se tudo der certo, acesse o IP do seu Wemos em um navegador e você deverá ver a página carregada como abaixo:

BLUETOOTH LOW ENERGY COM ESP32 E DHT11



Você já deve ter ouvido falar de Bluetooth Low Energy, Bluetooth Smart ou BLE. Tem sido uma tecnologia bastante aplicada a dispositivos como sensores de batimento cardíaco, pedômetros, sensores para bicicleta entre outros. E se você já é cliente da FilipeFlop também viu que temos em nossa loja alguns módulos disponíveis para quem quer começar a trabalhar com BLE, como é o caso do módulo BLE Keyes HM-10 e a placa de desenvolvimento ESP32 WiFi+Bluetooth. Neste tutorial, vamos abordar um projeto de uso do Bluetooth Low Energy com ESP32.



Veremos alguns conceitos básicos sobre BLE, desenvolvendo um programa que utiliza os recursos de BLE do ESP32, para enviar a um aplicativo de celular, dados de temperatura e umidade, coletados de um DHT11. O aplicativo de celular será desenvolvido no Thunkable (uma plataforma parecida com MIT App Inventor).

CONCEITOS BÁSICOS SOBRE BLUETOOTH LOW ENERGY

O Bluetooth Low Energy, ou BLE, é um subconjunto do clássico Bluetooth e foi introduzido juntamente com as especificações do Bluetooth 4.0. Em contraste com o BLuetooth clássico, o BLE tem um baixo consumo de energia mesmo mantendo um alcance similar.

Os dispositivos que trabalham com BLE podem ter duas funções diferentes em uma conexão, **Dispositivo Central** ou **Dispositivo Periférico** (*Central Device or Peripheral Device*). Geralmente os dispositivos centrais são telefones celulares, tablets, computadores, etc. São dispositivos centrais que recebem dados. Já os dispositivos periféricos são sensores e dispositivos *low power* que se conectam ao dispositivo central. Podemos pensar também como uma estrutura cliente/servidor, onde um celular é o cliente e o sensor é o servidor que “serve” seus dados para o cliente.

GATT (*Generic Attribute Profile*), é a forma com que os dados são organizados para comunicação entre os dispositivos. GATT é composto por um ou mais **serviços**

que por sua vez são compostos de **características**. Existem especificações padrão de GATT para os tipos de aplicação mais comuns encontradas no mercado. Várias dessas especificações podem ser encontradas no site oficial do Bluetooth. As características por sua vez são basicamente os valores em si.

Os serviços e características são identificados por um **UUID**. Por exemplo "0x180F" ou "6E400001-B5A3-F393-E0A9-E50E24DCCA9E". O legal é que podemos criar nossas próprias características customizadas ou até mesmo utilizar as características já existentes como por exemplo Sensoriamento de Ambiente.

Para sermos um pouco mais práticos, vamos analisar, por exemplo, o serviço oficial de nível de bateria Battery Service. Esse serviço tem um UUID 0x180F e uma característica chamada **Battery Level** de UUID 0x2A19. Imagine que existe um sensor BLE que mede nível de bateria(serviço) e que envia dados indicando o nível da bateria(característica). Caso um celular se conecte a esse sensor, o celular já irá saber que aquele sensor mede nível de bateria e seus dados enviados são o nível da bateria em si. Pensando em um ESP32, se ele tiver programado essas características, ele "seria um sensor" de nível de bateria e seria reconhecido como tal por qualquer aplicativo BLE de celular (fica a dica para colocar sua criatividade em ação criando um dispositivo BLE, pode ser qualquer um destes).

BLUETOOTH CLÁSSICO VERSUS BLE

Além da diferença do consumo de energia, que no caso do BLE é mais baixo se comparado ao Bluetooth clássico, a forma com que os dispositivos se comunicam e organizam seus dados também é diferente.

Com os módulos tipo HC-05 usávamos o bluetooth clássico. Eles basicamente faziam uma comunicação UART sem fio, bastando apenas parear os dispositivos como visto no Tutorial Módulo Bluetooth com Arduino. Mas se tentarmos parear um dispositivo BLE da mesma forma, veremos que a comunicação não é tão simples assim pois como vimos acima o BLE usa uma organização diferente de dados.

Para simular uma comunicação UART usando BLE, o fabricante Nordic Semiconductor criou um serviço proprietário chamado Nordic UART Service. É um serviço que tem duas características sendo elas TX para transmissão de dados e RX para recepção de dados. Esses serviços e suas características tem os seguintes UUIDs:

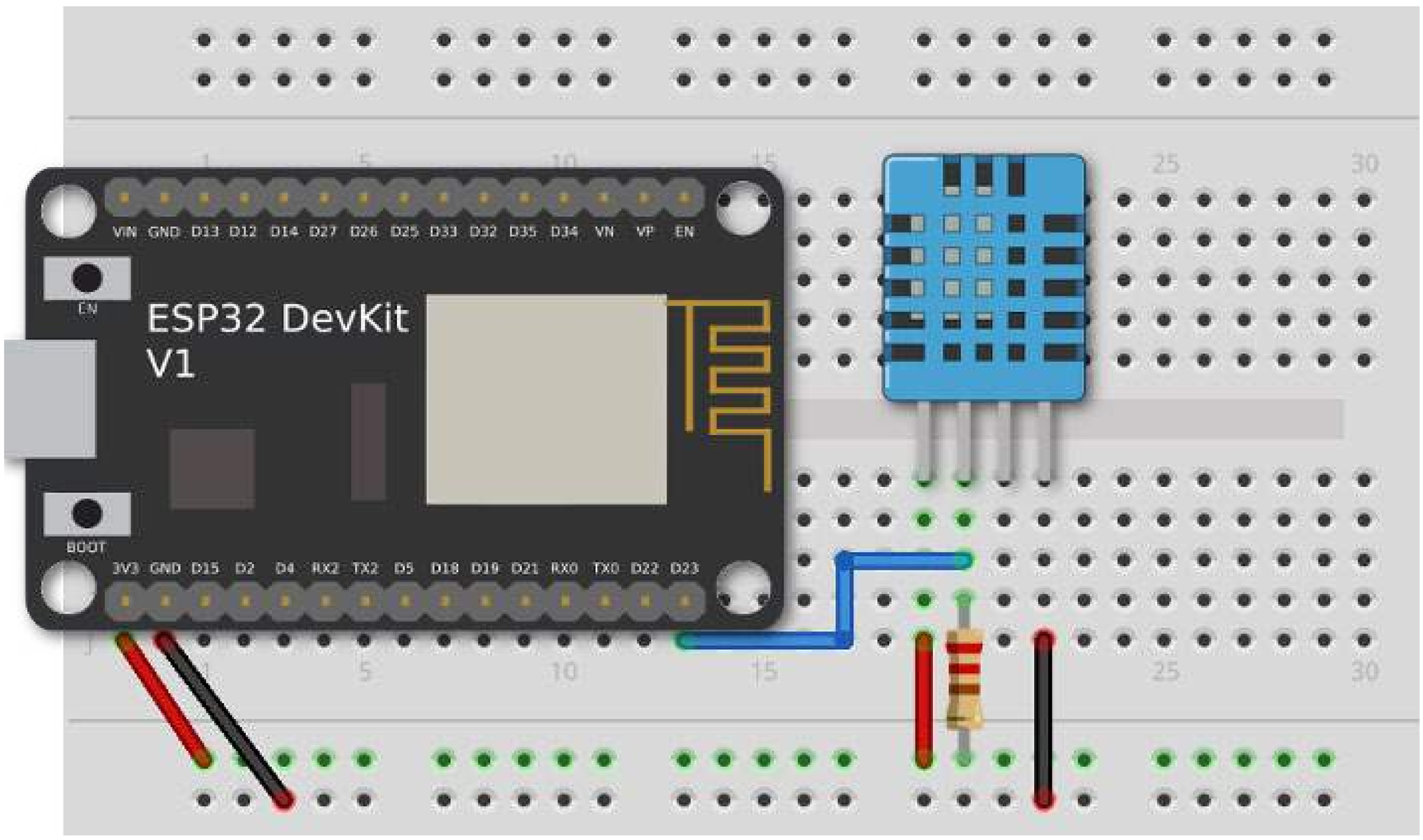
Nordic UART Service: 6E400001-B5A3-F393-E0A9-E50E24DCCA9E

RX Characteristic: 6E400002-B5A3-F393-E0A9-E50E24DCCA9E

TX Characteristic: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E

MONTAGEM DO CIRCUITO

O circuito do nosso teste usando Bluetooth Low Energy com ESP32 é bem simples e utilizaremos um **sensor de umidade e temperatura DHT11**. O pino de dados do DHT11 deve ser ligado ao pino 23 do ESP32.



PROGRAMA BLUETOOTH LOW ENERGY COM ESP32

Para programar o ESP32 utilizando a IDE Arduino, siga as **instruções de instalação** de acordo com seu sistema operacional. Certifique-se de conseguir compilar e carregar qualquer programa para o ESP32, por exemplo um Blink LED.

Carregue então o programa abaixo na IDE Arduino. Certifique-se também de ter a biblioteca para sensor DHT instalada em sua IDE.

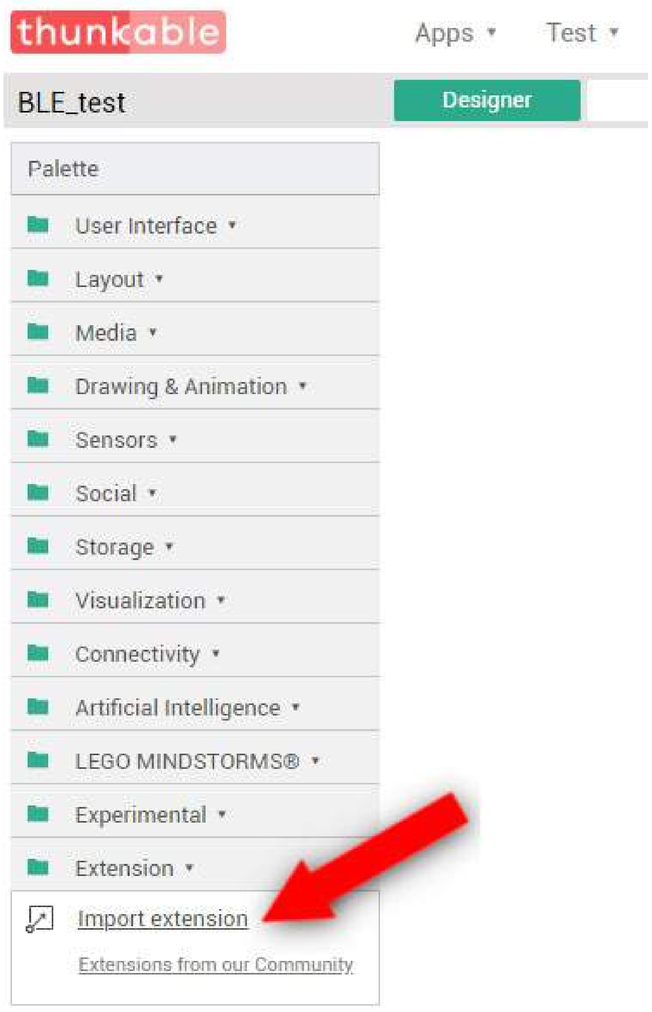
O programa basicamente define os UUID do serviço de comunicação UART, faz a leitura de umidade e temperatura do sensor DHT e transmite esses dados para o aplicativo no celular. Os dados são enviados em uma única variável, mas num formato tipo CSV, com temperatura e umidade separados por vírgula.

FAÇA O DOWNLOAD DO CÓDIGO DIRETO PELA IDE ARDUINO AQUI

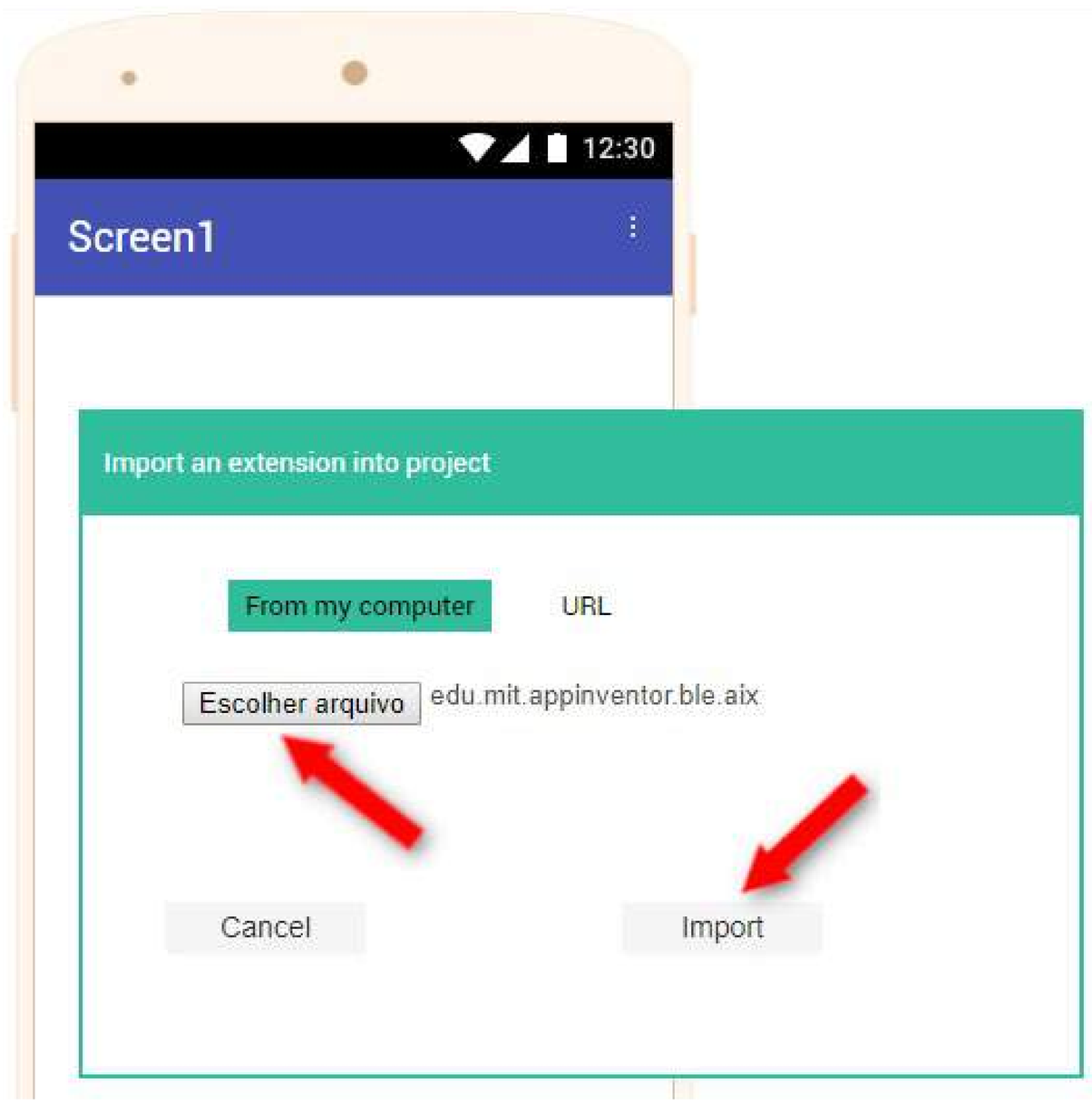
APLICATIVO DE CELULAR COM THUNKABLE

Para que possamos visualizar no celular os dados enviados pelo ESP32, desenvolvemos um aplicativo utilizando a plataforma Thunkable. Pra quem já utilizou o MIT App Inventor já estará familiarizado com o Thunkable e irá achar super legal o Material Design.

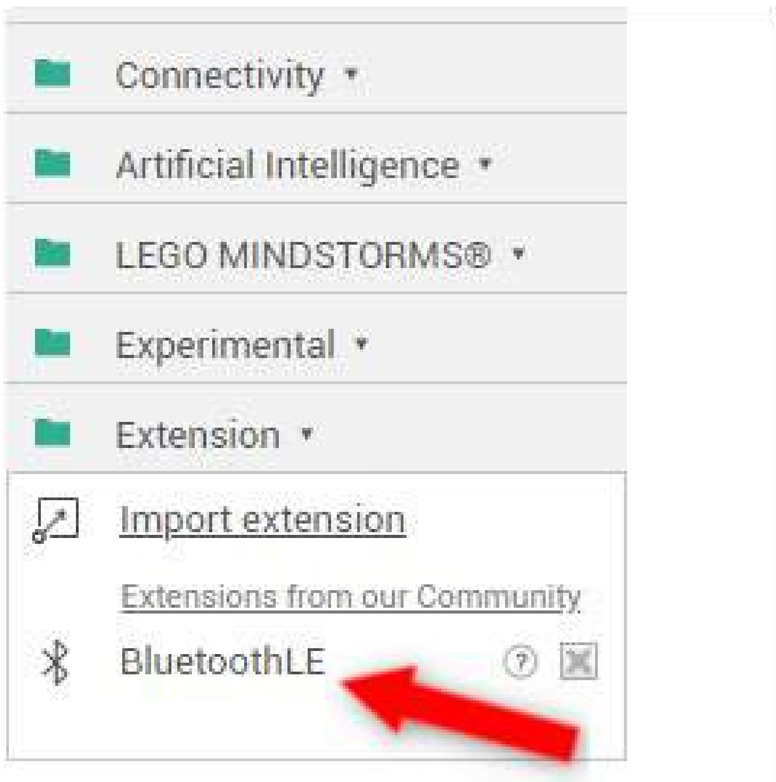
Existe uma extensão feita especialmente para comunicação de dispositivos usando BLE. Essa extensão pode ser baixada aqui e importada no Thunkable. Veja como fazer essa importação como nas figuras a seguir:



Escolha o arquivo baixado no link da extensão.

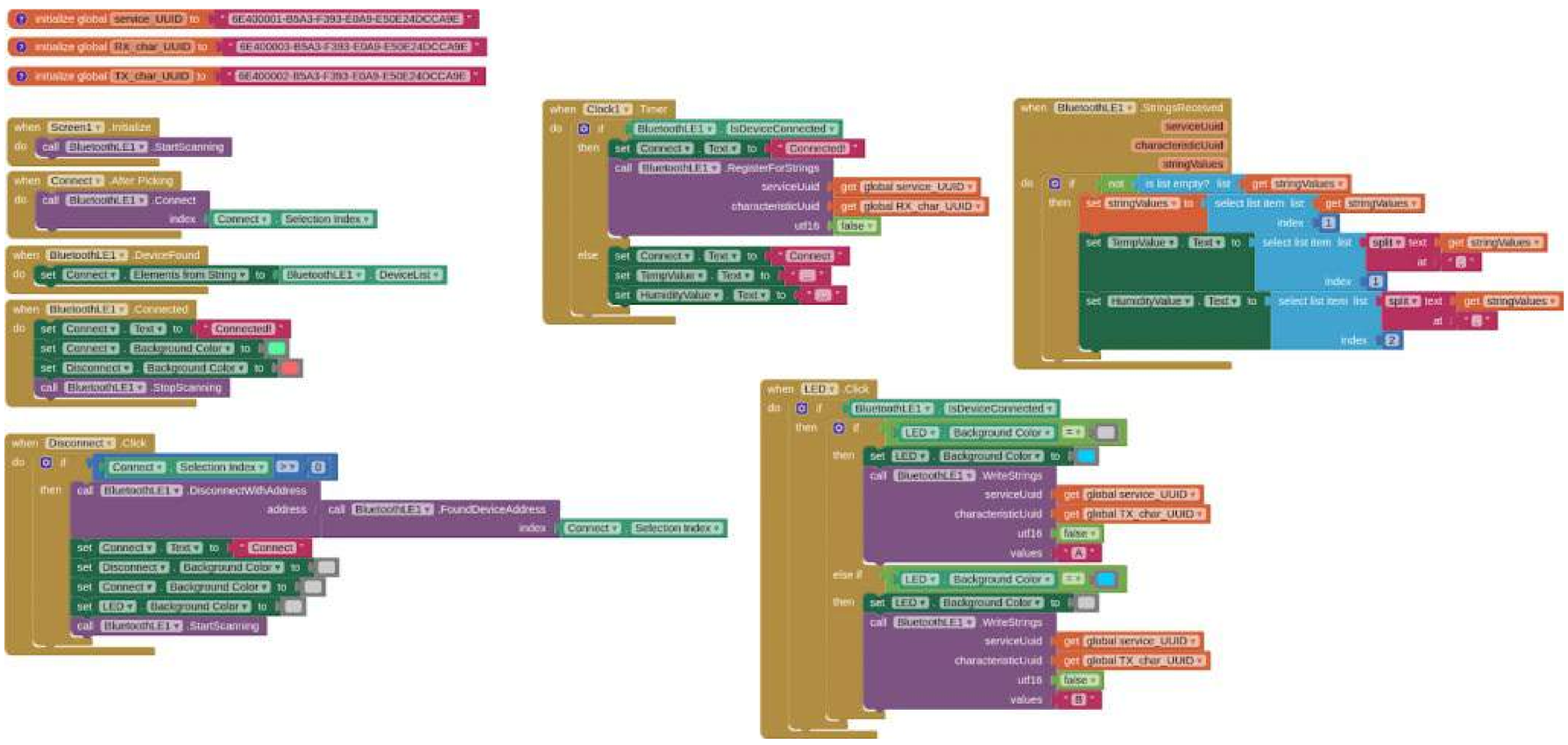


Agora a extensão estará disponível para ser usada na aplicação.

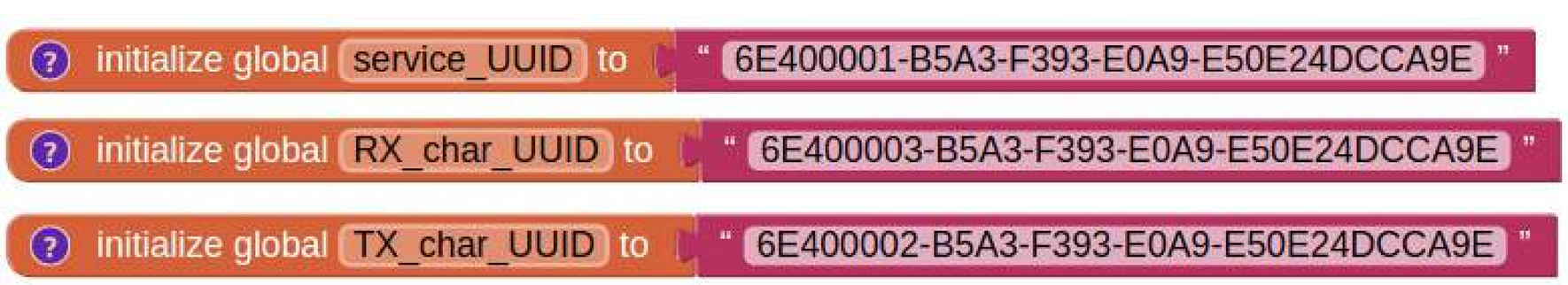


O design da aplicação ficou como a seguir, mas você pode usar sua criatividade e mudar a “cara” da aplicação. Existem dois botões para gerenciar a conexão com o ESP32 e um botão mais abaixo para ligar e desligar o led built-in do ESP32. Em dois labels mostramos a temperatura e umidade do DHT11.

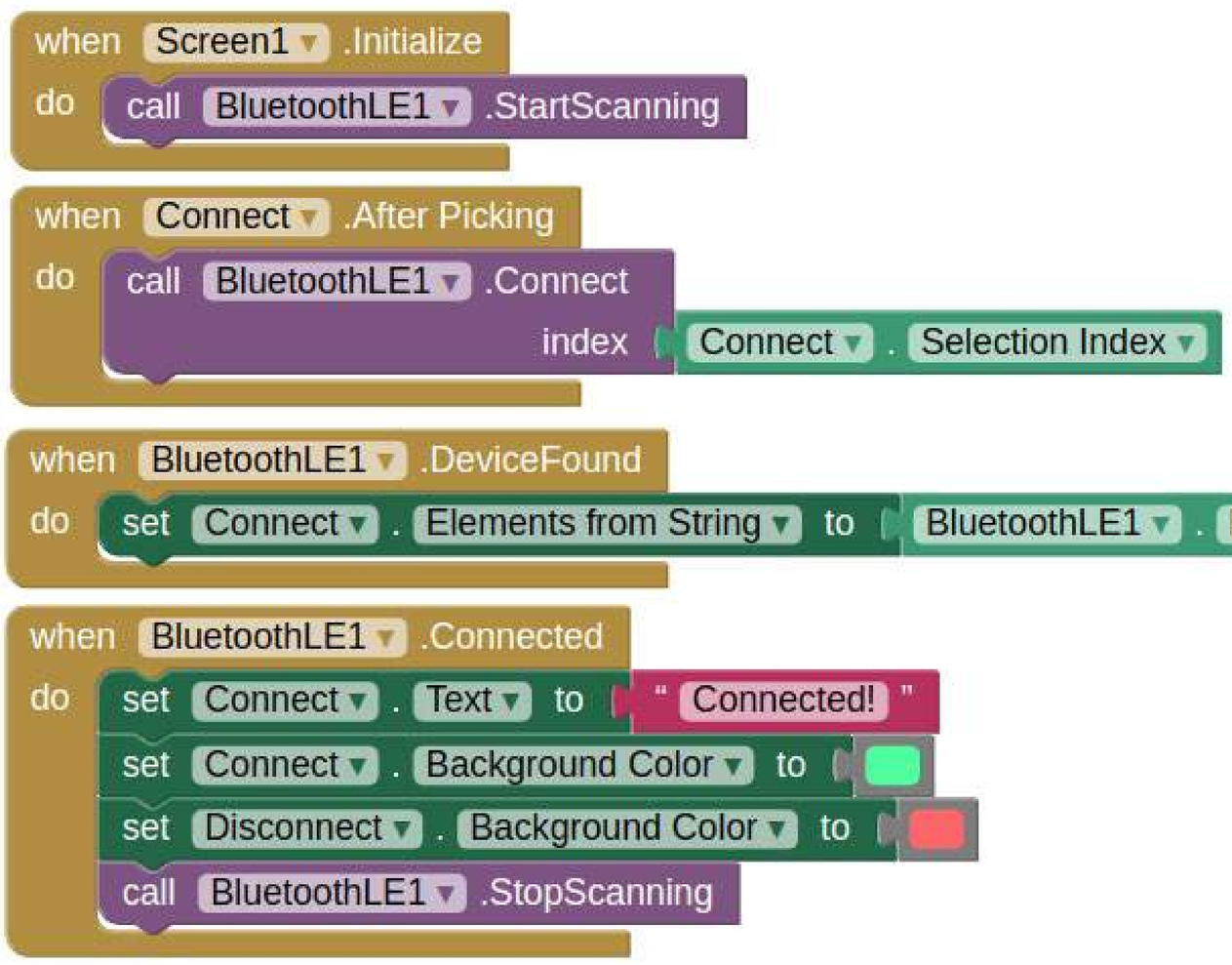
É na seção de blocos onde é feita a programação do aplicativo.
Veja na foto a seguir a estrutura geral do aplicativo:



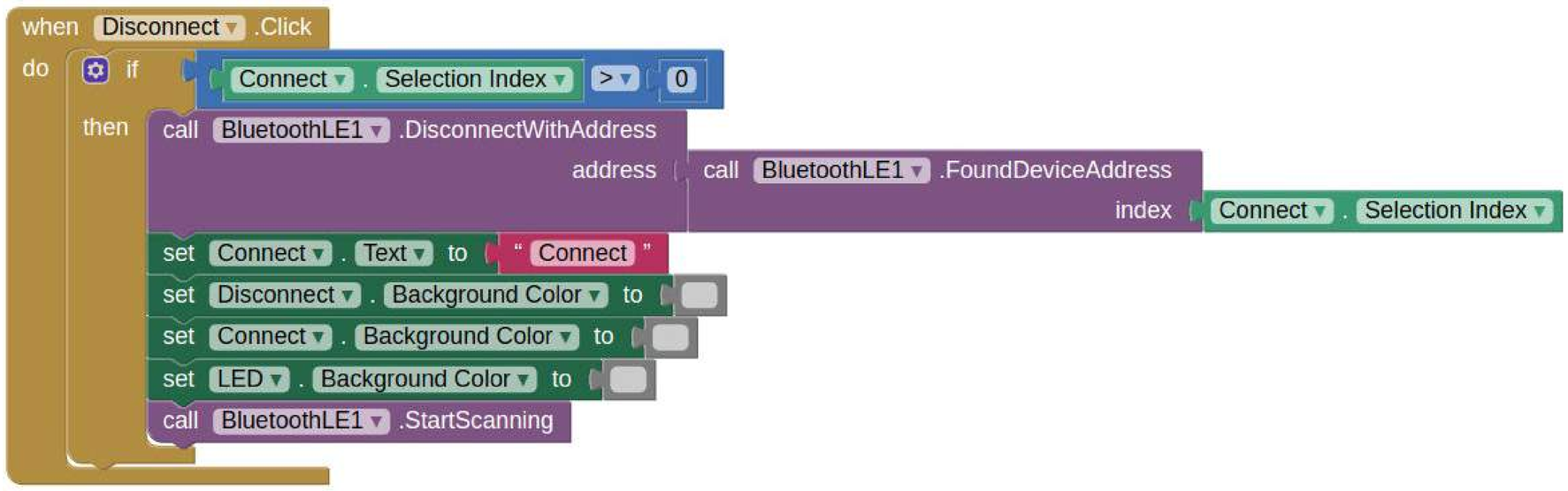
Primeiramente declaramos os UUIDs utilizados pelo ESP32.



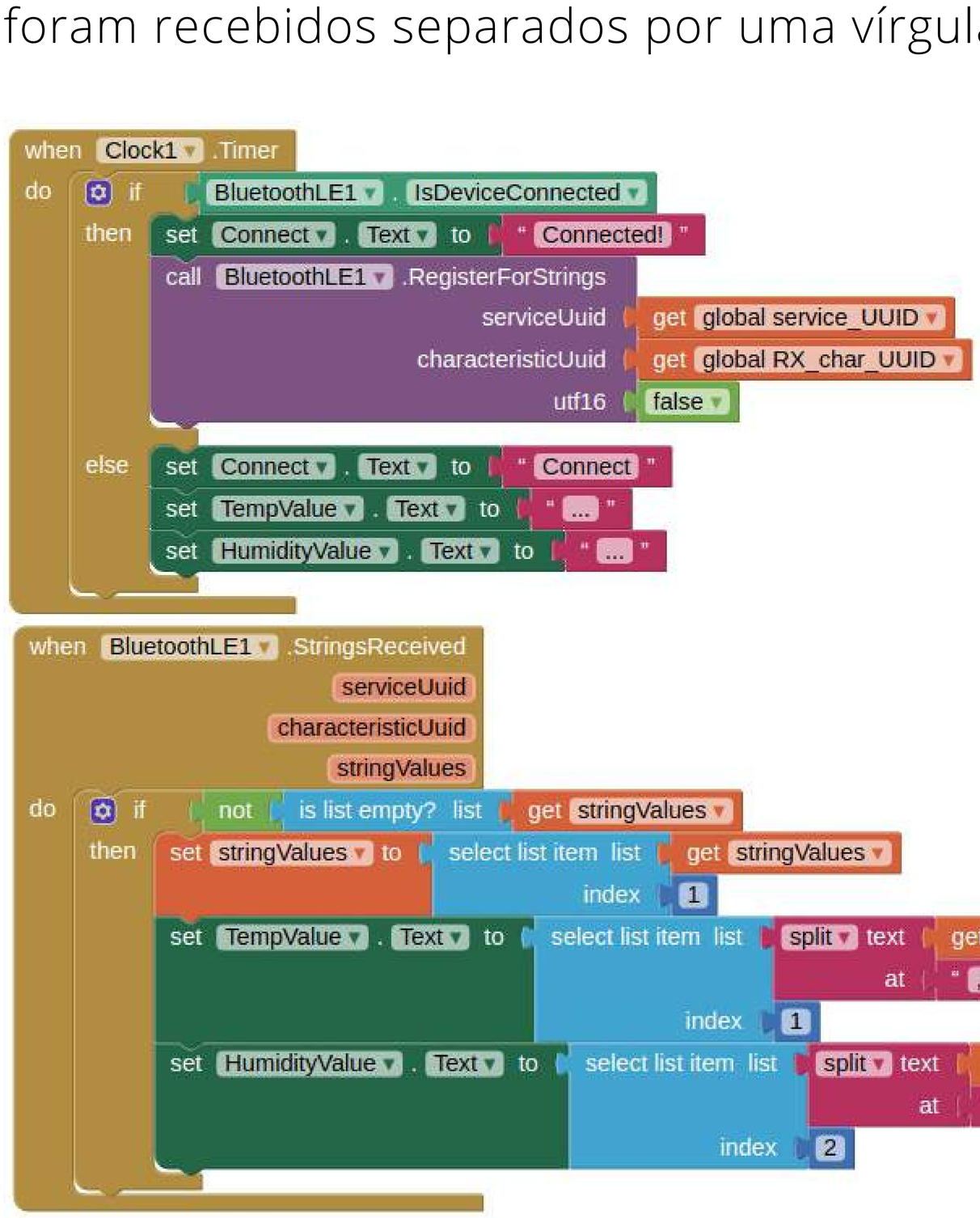
Então gerenciamos o escaneamento e conexão de dispositivos:



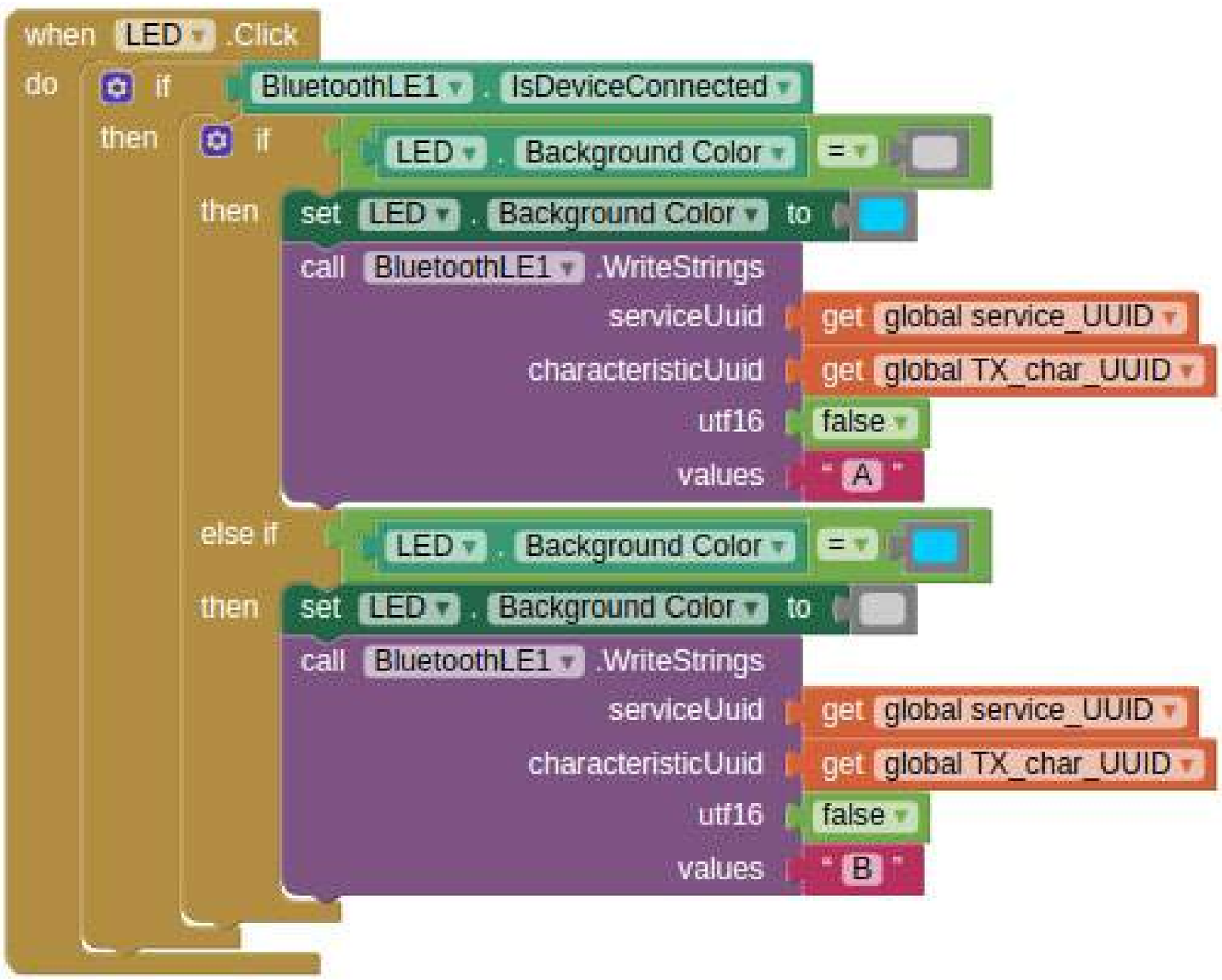
O botão disconnect fica responsável por desconectar o ESP32.



Um Timer verifica se o dispositivo ainda está conectado e se foi recebido algum dado do ESP32. Assim que um dado é recebido, o evento StringsReceived é disparado, exibindo os dados de temperatura e umidade nos respectivos labels. Nesse bloco fazemos a separação dos dados de temperatura e umidade que foram recebidos separados por uma vírgula.



Quando o botão LED é clicado, o bloco a seguir envia um caractere A ou B para o ESP32, que por sua vez liga o LED se receber A ou desliga se receber B.

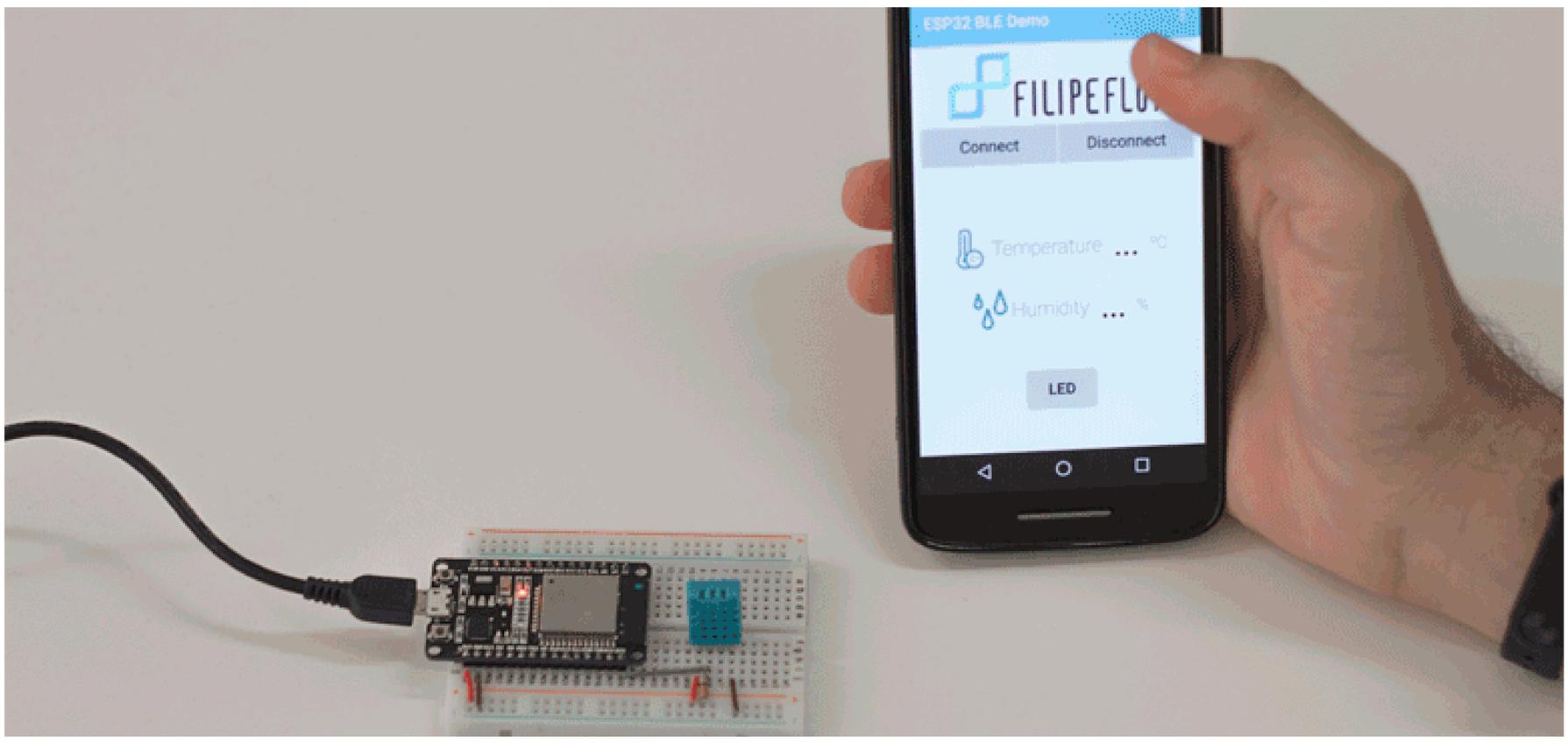


Caso queira modificar a aplicação você pode baixar o arquivo do projeto (.aia) **aqui**. Ou se preferir, pode instalar o **.apk** direto no seu celular.

FUNCIONAMENTO DO PROJETO

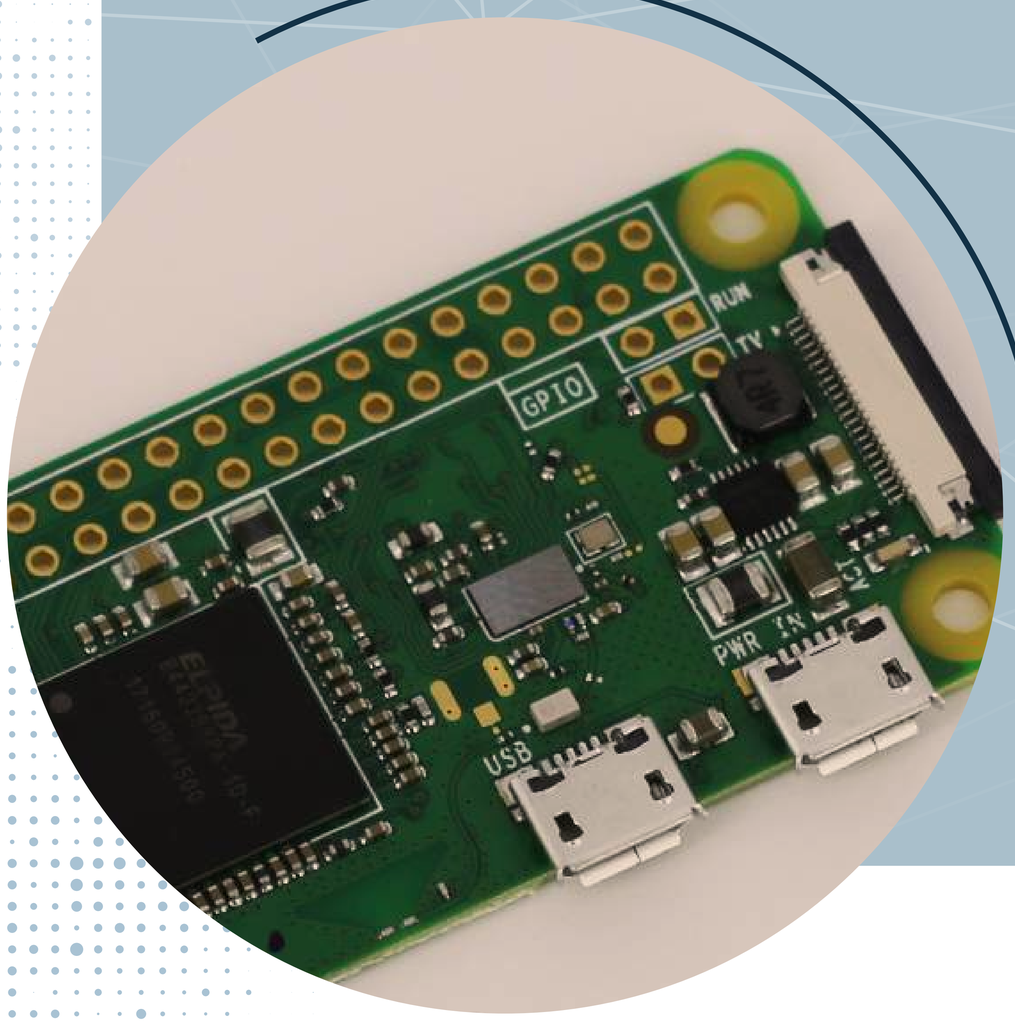
Veja no vídeo abaixo o funcionamento do projeto Bluetooth Low Energy com ESP32. Basta clicar no botão connect, escolher

o ESP32 na lista de dispositivos e os dados já começam a aparecer. Acenda e apague o LED clicando no botão LED.



Esse projeto de Bluetooth Low Energy com ESP32 foi baseado no projeto original de Timothy Woo e pode ser encontrado no Hackster.io. Agradecemos também ao Timothy por ter nos ajudado com algumas questões técnicas, sobre como enviar mais de um dado ao aplicativo usando apenas uma característica.

BROKER MQTT COM RASPBERRY PI ZERO W



A Raspberry Pi Zero W é uma placa e tanto! Mas ela pode ir além, podendo ser aplicada tranquilamente como servidores caseiros de pequeno porte. Isto leva a vantagens muito interessantes, como ter uma solução simples (de uma só placa), compacta (devido ao pequeno tamanho físico da placa) e móvel na sua casa (já que possui conectividade Wi-Fi à rede). É justamente nesse ponto que este tutorial irá tocar: como transformar uma Raspberry Pi Zero W em um pequeno servidor caseiro, no caso um broker MQTT (Mosquitto). Além disso, conectaremos nesse broker MQTT com Raspberry Pi o projeto da Planta IoT, outro projeto que falamos em nosso Blog.

Para seguir este projeto, você precisará de uma **placa Raspberry Pi Zero W** e uma **fonte para a Raspberry**.

IMPORTANTE

Antes de seguir adiante, tenha certeza que a sua Raspberry Pi Zero W está totalmente operacional (com cartão SD, com Raspbian instalado e conectividade à Internet operante). Lembramos também que apesar deste projeto usar a Raspberry Pi Zero W, para montar o seu projeto de Broker MQTT, você também pode usar qualquer uma das outras placas da linha Raspberry, ok?

INSTALAÇÃO DO BROKER MQTT MOSQUITTO NA RASPBERRY PI ZERO W

Já explicamos anteriormente o que é Broker MQTT, mas, caso tenha restado alguma dúvida, volte ao capítulo onde falamos sobre “Controle e Monitoramento IoT com NodeMCU e Broker MQTT”.

Neste caso, será utilizado um broker MQTT chamado **Mosquitto**, um broker MQTT open-source da **comunidade open-source Eclipse**. Antes de tudo, é uma boa ideia fazer update dos repositórios do seu gerenciador de pacotes e upgrade dos seus pacotes já instalados. Para isso, execute os comandos abaixo:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Feito isso, o broker MQTT Mosquitto pode ser instalado na sua Raspberry Pi Zero W com o comando abaixo:

```
sudo apt-get install mosquitto
```

Ao término da instalação, ele já está instalado e rodando como um serviço Linux!

CONFIGURAÇÃO DE UM HOSTNAME PARA A RASPBERRY PI ZERO W

Agora será feito algo muito útil para o uso e testes com o broker MQTT que roda na Raspberry: a configuração de

um hostname para a Raspberry Pi Zero W. Grosso modo, o hostname serve para podermos nos conectar de forma mais intuitiva à Raspberry PI Zero W, utilizando na conexão o hostname ao invés do IP. Como o IP pode ser dinâmico (DHCP), configurar um hostname não é só cômodo como também é produtivo. Para isso, siga o passo-a-passo abaixo:

1. No terminal Linux / console, acesse o programa de configuração da Raspberry PI Zero W com o seguinte comando:

```
sudo raspi-config
```

2. No programa de configuração, busque pela configuração **hostname** e acesse-a.

3. Nela, configure um nome amigável como hostname. Não pode haver espaços nem caracteres especiais (ou seja, somente letras e números são permitidos). Como sugestão, coloque o nome *rasppizerow*.

4. Vá até a opção “Ok” e depois em “Finish”.

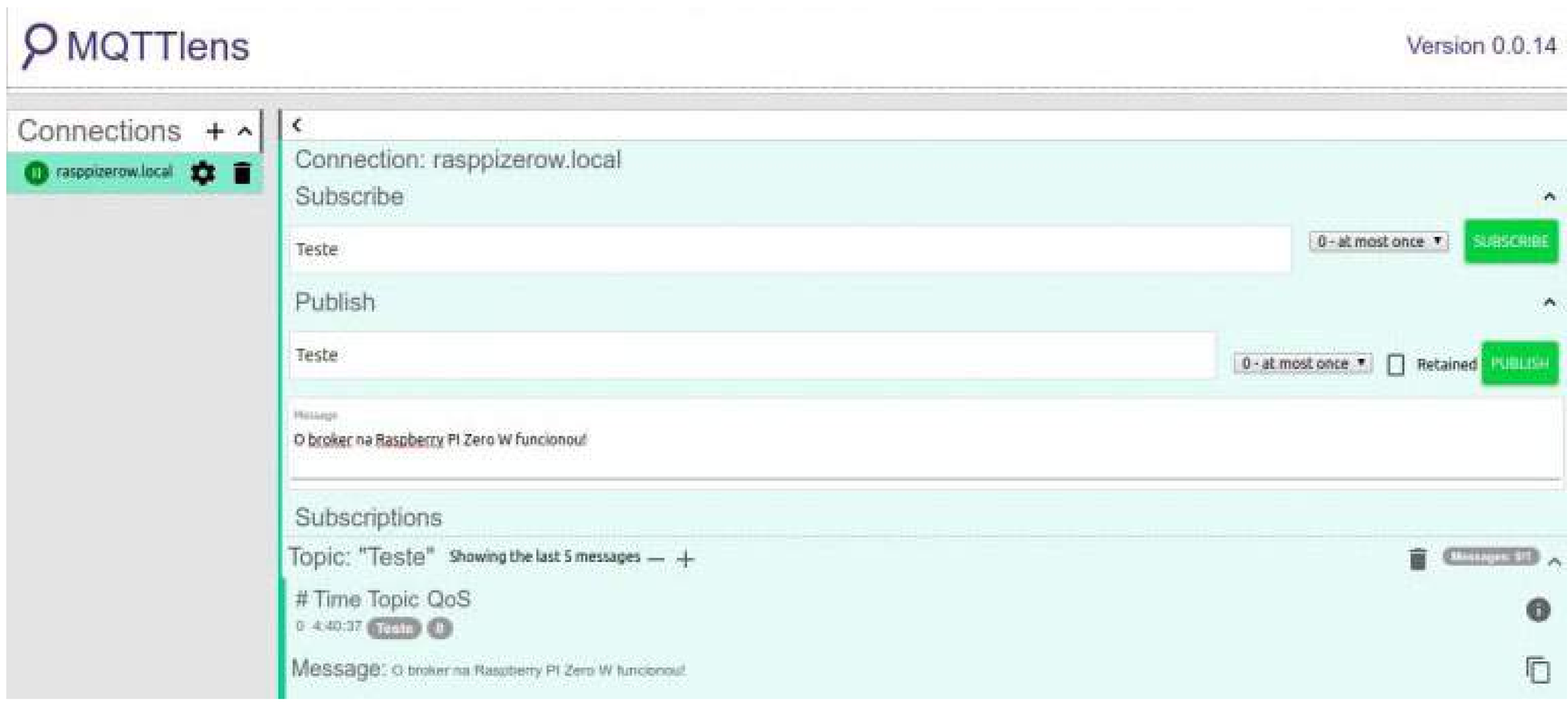
5. Será solicitado que a Raspberry Pi Zero W seja reiniciada. Clique em Ok e aguarde que a Raspberry Pi Zero W complete o processo de reboot.

Após o reboot da Raspberry Pi Zero W, do seu computador (que deve estar na mesma rede que a Raspberry Pi Zero W) faça um ping em NOME_QUE_VOCE_CONFIGUROU_NO_HOSTNAME.local (onde NOME_QUE_VOCE_CONFIGUROU_NO_HOSTNAME será, se você seguiu minha sugestão de nome, *rasppizerow*). Você notará que o ping terá resposta, significando que o hostname foi configurado com sucesso.

TESTE DO BROKER MQTT COM RASPBERRY PI

Uma vez configurado um hostname para a Raspberry Pi Zero W, podemos prosseguir para o teste do broker MQTT com Raspberry PI Zero W. Para testa-lo, utilize um computador ou smartphone **que esteja na mesma rede que sua Raspberry Pi** e instale um client MQTT (para computadores, sugiro o MQTTLens e, para smartphones Android, sugiro o aplicativo MyMQTT).

No cliente MQTT de sua escolha, coloque como endereço do broker ***rasppizerow.local*** e como porta o valor ***1883***. Feito isso, você notará que o client MQTT se conectou ao broker que está rodando na sua Raspberry Pi Zero W, comprovando portanto seu funcionamento. Como teste adicional, ainda usando o cliente MQTT, publique e faça subscribe em um mesmo tópico e verifique que o publish e subscribe ocorrem sem problemas. Observe a figura abaixo:



Com isso, seu broker MQTT local está pronto para uso!

UTILIZANDO O PROJETO DA PLANTA IOT NESTE BROKER

Visando mostrar uma das várias possibilidades de uso deste broker MQTT local, o projeto da Planta IoT será aqui integrado. Tal projeto é dividido em quatro partes / posts, sendo o terceiro post o qual se faz uso de MQTT (para acessar este post, clique aqui). As mensagens MQTT neste projeto sevem para transmitir, de forma periódica (com tempo definido no código-fonte), a umidade do solo medida.

Portanto, ao invés da informação da umidade de solo ser publicada a nível global (utilizando um broker MQTT publico na nuvem, como é feito no projeto original), tal informação será publicada no seu broker MQTT local. Desse modo, somente quem estiver na mesma rede do broker MQTT e se subescrever ao tópico correto conseguirá tal informação. Embora possa parecer retrógrado utilizar um broker MQTT local, esta é uma estratégia muito interessante que pode ser aplicada em sensoriamento em uma empresa, garantindo assim maior proteção de dados importantes / medições importantes e também aumento na velocidade de comunicação, já que tudo estará em rede local nesta arquitetura proposta.

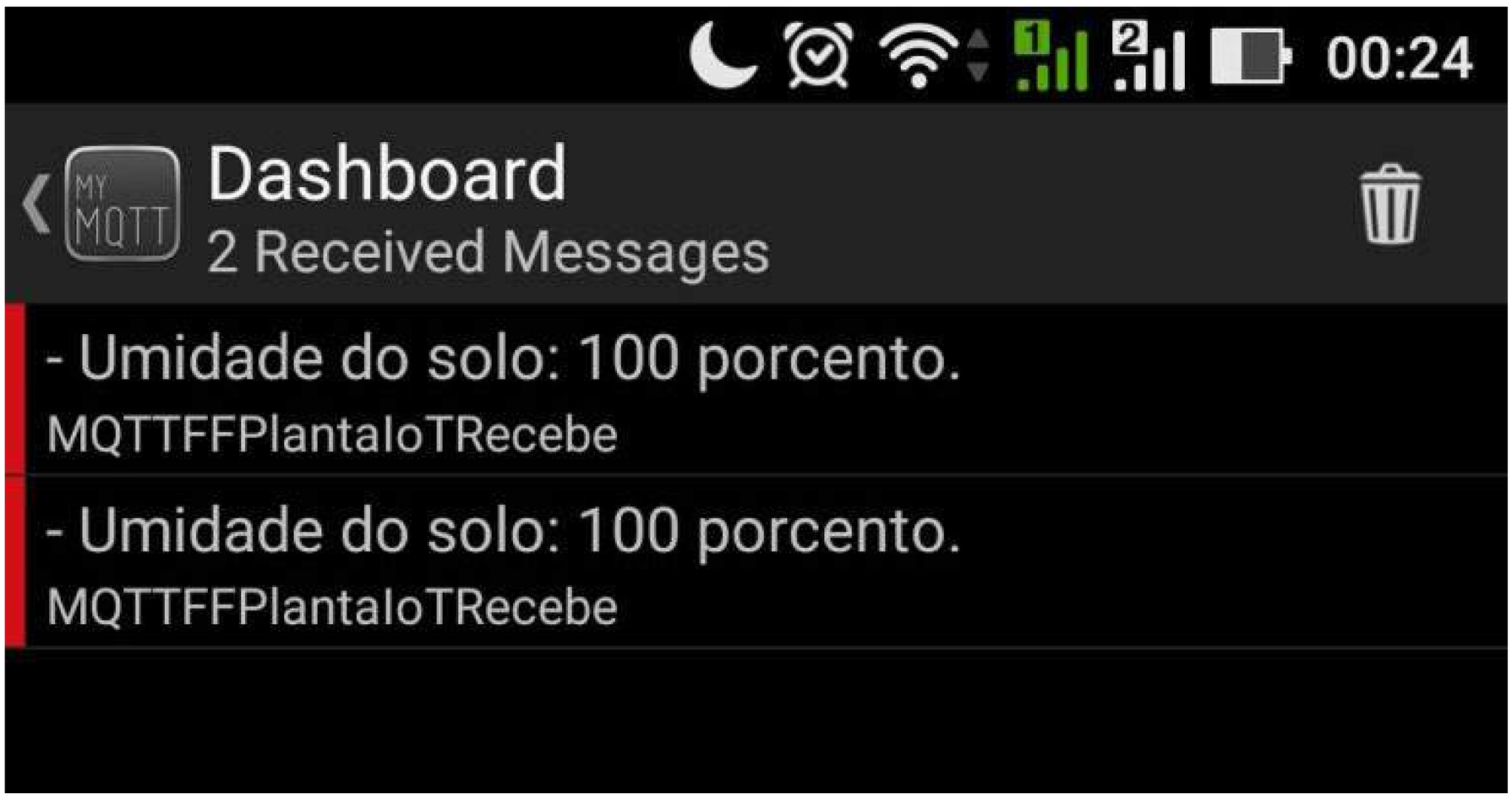
Para fazer com que o projeto da Planta IoT envie/publique a informação de umidade do solo medida no seu broker MQTT local, basta trocar seguinte linha do código (linha 41):

```
const char* BROKER_MQTT = “iot.eclipse.org”; //URL do broker MQTT que se deseja utilizar
```

Pela linha mostrada abaixo:

```
const char* BROKER_MQTT = “rasppizerow.local”; //URL do broker MQTT que se deseja utilizar
```

Agora, se seu computador e/ou smartphone estiver na mesma rede que seu broker MQTT local, você poderá utilizar um cliente MQTT qualquer para ver as informações de umidade de solo publicadas localmente. Na figura 3 é mostrado o resultado utilizando o cliente MQTT MyMQTT (em um smartphone Android).



USOS ADICIONAIS DE UM BROKER MQTT LOCAL

Conforme comentado aqui, o uso de um broker MQTT com Raspberry Pi na sua rede local pode trazer vantagens interessantes, sobretudo a uma empresa, já que assim garante-se maior proteção de dados importantes e aumento

na velocidade de comunicação. Logo, o uso deste tipo de broker pode ser justificado, principalmente, por aumento na segurança. Segue abaixo algumas sugestões de uso de um broker MQTT local:

- Central de comunicação em um sistema de automação residencial
- Central de comunicação de sensoriamento em uma empresa
- Central de comunicação em sistemas de automação comercial

Além disso, você não contará com limitações de mensagens publicadas por segundo que podem estar presentes em brokers MQTT públicos / na nuvem. Isso pode ser uma grande vantagem se seu projeto exigir grande quantidade de mensagens trocadas em pouco tempo (como sensoriamento de eventos muito rápidos, por exemplo).

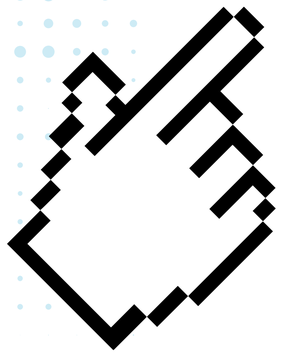
CONCLUSÃO

A internet das coisas já faz parte da nossa realidade e é muito mais acessível do que se imagina. Para os makers de carteirinha que queiram inventar projetos de IoT dos mais variados tipos, é possível encontrar uma série de conteúdos na internet, como o próprio Blog da FilipeFlop. E se você quiser compartilhar o seu próprio projeto, tirar dúvidas e interagir com outros makers, acesse nosso Fórum!

VISITAR BLOG

ACESSAR FÓRUM

WORK HARD HAVE FUN & MAKE THINGS.



FILIPEFLOP

www.filipeflop.com

Acompanhe
nossas redes sociais