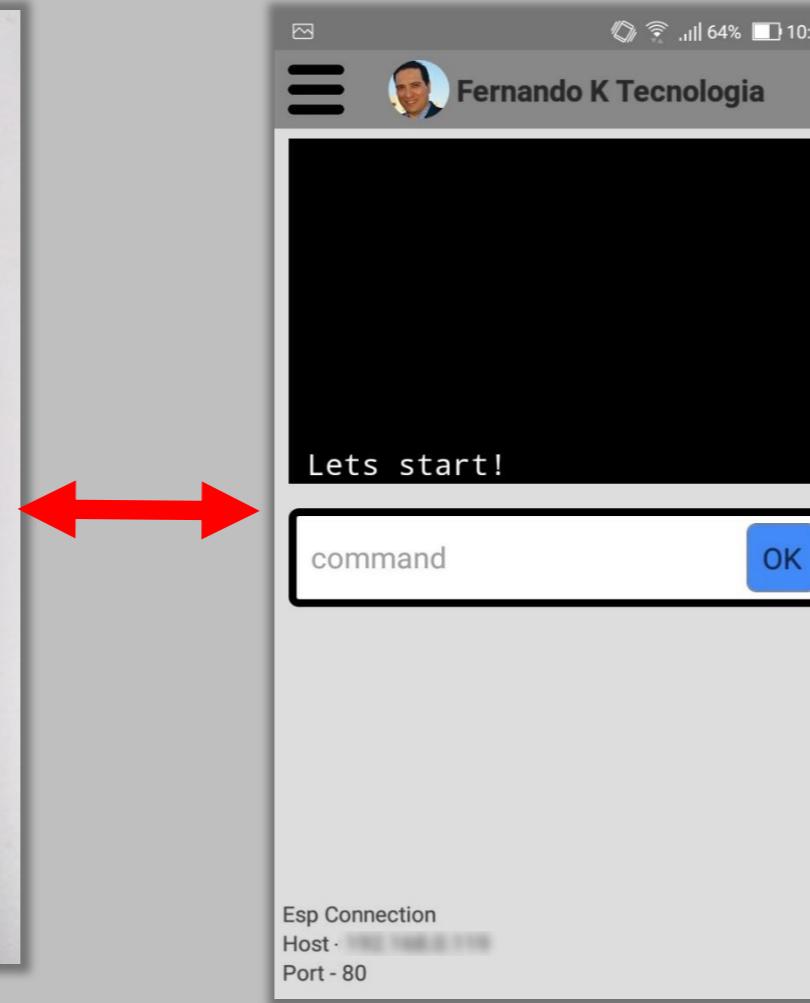
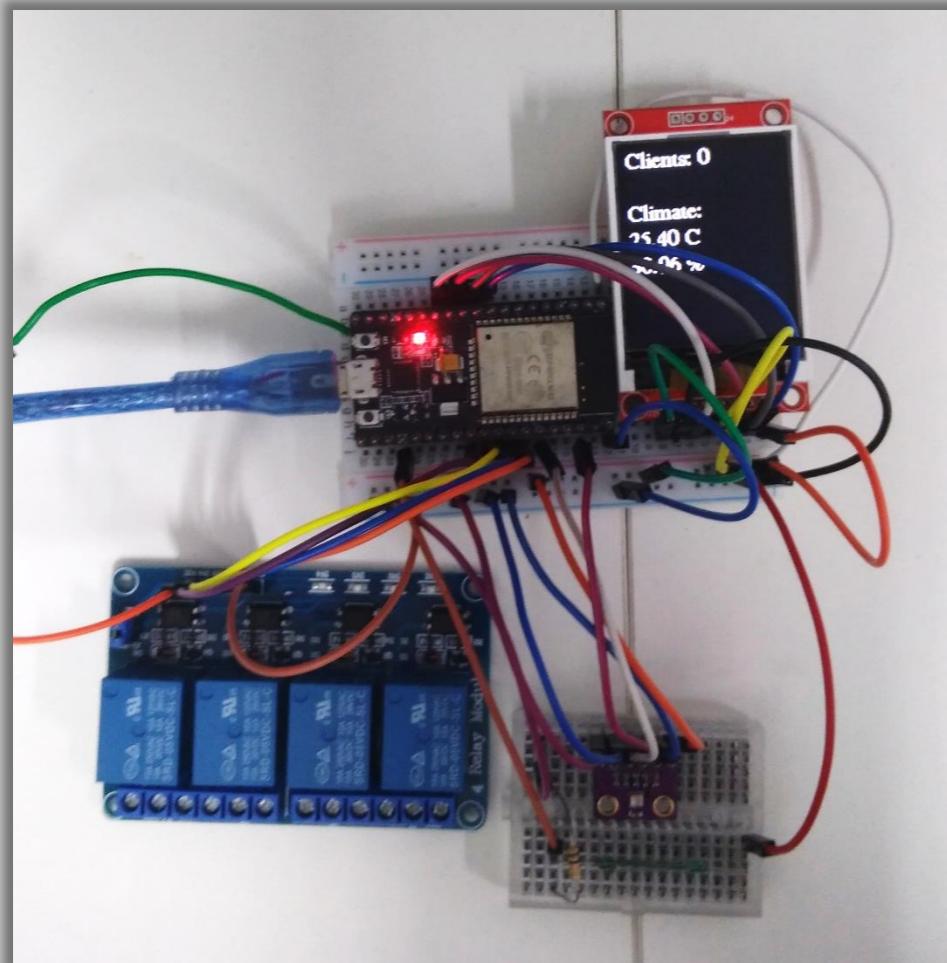


Tutorial de como utilizar o aplicativo Android FernandoK - ESP32 Reed Switch

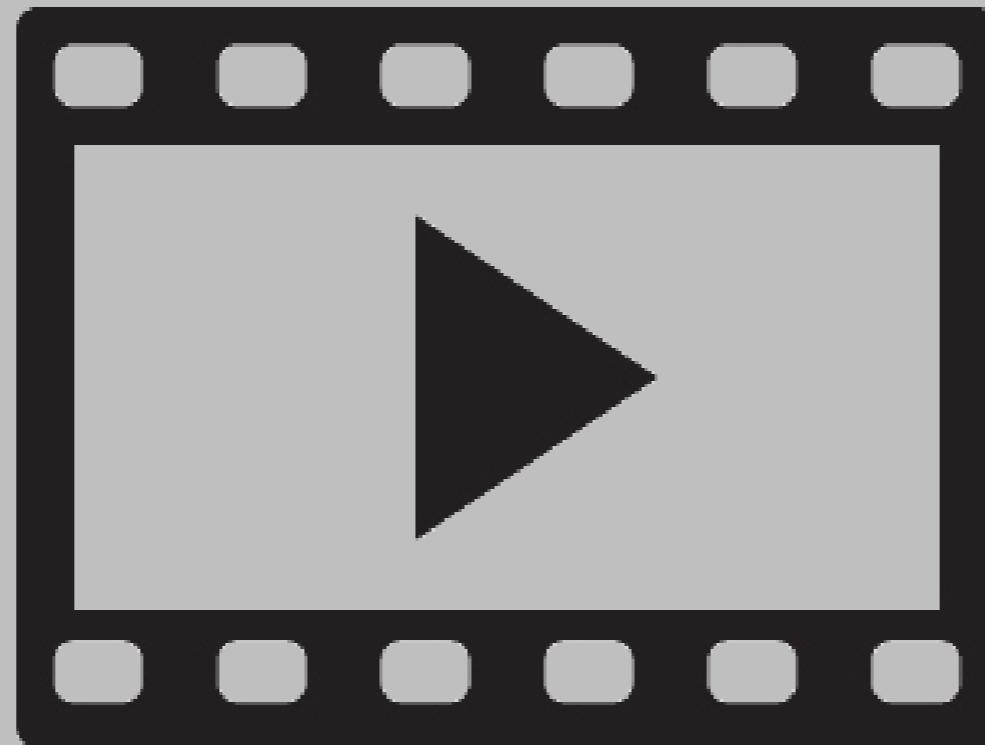


Por Fernando Koyanagi

Intenção dessa aula

- 1. Apresentar o aplicativo Android de automação**
- 2. Apresentar um exemplo multiclient com ESP32 e enviar/receber mensagens para o aplicativo**

Demonstração



Domingo, Janeiro 14 2018



Em www.fernandok.com

PRINCIPAL SOBRE FERNANDO K ARDUINO ESP8266 ESP32 LORAWAN MOTOR DISPLAY MATERIAIS DOWNLOAD

Receba o meu conteúdo
GRATUITAMENTE

Insira aqui seu melhor email...

QUERO RECEBER GRÁTIS



Seu e-mail



Motor de Passo Nema 23 com Driver TB6600 e Arduino Due

by Fernando K Tecnologia - 2:44 PM

Hoje vamos voltar a falar de Motor de Passo. Vamos utilizar um Nema 23 que será controlado por um Driver TB6600 e um Arduino Due. É p...

[Leia mais](#)

QUAL ASSUNTO VOCÊ TEM?

- Arduino
- ESP8266
- ESP32
- Motor
- Display
- Sensor

You may select multiple answers.

[Votar](#) [Exibir resultados](#)

Votos até o momento: 32
Dias restantes para votar: 49

FACEBOOK



ESP32 Longa Distância - LoRaWan

by Fernando K Tecnologia - 9:46 AM

Neste artigo vamos tratar da LoRaWAN, uma rede que vai longe gastando pouca energia. Mas, o quanto "longe"? Com o chip que uso no vídeo...

[Leia mais](#)



Motor de HD com Arduino

by Fernando K Tecnologia - 2:00 PM

forum.fernandok.com

Fórum Fernando K Tecnologia
Fórum sobre dúvidas com relação ao conteúdo disponibilizado pelo Fernando Koyanagi

Nosquisar...

www.fernandok.com /fernandokoyanagi /fernandokoyanagi /fernandok_oficial /fernandok_oficial

[Links rápidos](#) [L...](#) [fernandokoyanagi](#)

Bem-vindo: 05/Oct/2018, 11:16 A sua última visita foi em 10/Set/2018, 15:47

Assinalar todos os fóruns como lidos

SUporte Fórum Fernandok

	TÓPICOS	MENSAGENS	ÚLTIMA MENSAGEM
Feedback Dúvidas, críticas ou sugestões sobre o Fórum FernandoK. Para demais questões utilize o fórum correto.	6	11	Re: O russo voltou por Ipmehi 01/Oct/2018, 08:25

FERNANDO K

	TÓPICOS	MENSAGENS	ÚLTIMA MENSAGEM
Arduino Projetos de arduino	31	79	skardy bogii por Sorororcem 05/Oct/2018, 10:55
ESP32 Projetos de ESP32	29	62	Dúvidas sobre como instalar a... por Marcos Sarge 04/Oct/2018, 15:52
ESP8266 O ESP8266 é um microcontrolador do fabricante chinês Espressif que inclui capacidade de comunicação por Wi-Fi.	24	51	Re: NodeMCU não conecta em qu... por ivanribeira 04/Oct/2018, 14:39
LoRa Projetos com LoRa	11	31	Projeto de irrigação de jardim por marlendo 04/Oct/2018, 21:30
STM32 Projetos com STM32	3	8	Re: Imprecisão de tempo de de... por biazoto 12/Sep/2018, 09:15
Motor Projetos com motor	5	11	Re: impressora 3d com motor dc por Magneton 24/Sep/2018, 19:05
Display Projetos com Display	4	11	Re: Alguém conhece o VIRTUINO... por Jod Luz 21/Sep/2018, 11:39

QUEM ESTÁ ONLINE
No total, há 4 usuários online :: 2 usuários registrados, 0 invitado e 2 visitantes (baseado em usuários ativos nos últimos 5 minutos)
O recorde de usuários online foi de 19 em 11/Sep/2018, 05:37

Usuários registrados: alberto, fernandokoyanagi
Legenda: Administradores, Moderadores globais

ANIVERSÁRIOS
Não há aniversários hoje

ESTATÍSTICAS
Total de mensagens 703 • Total de tópicos 114 • Total de membros 469 • Novo usuário: Sorororcem

[L...](#) [☰](#)

Powered by phpBB® Forum Software © phpBB Limited
Traduzido por: Suporte phpBB
Painel de Controle da Administração



Instagram
fernandok_oficial

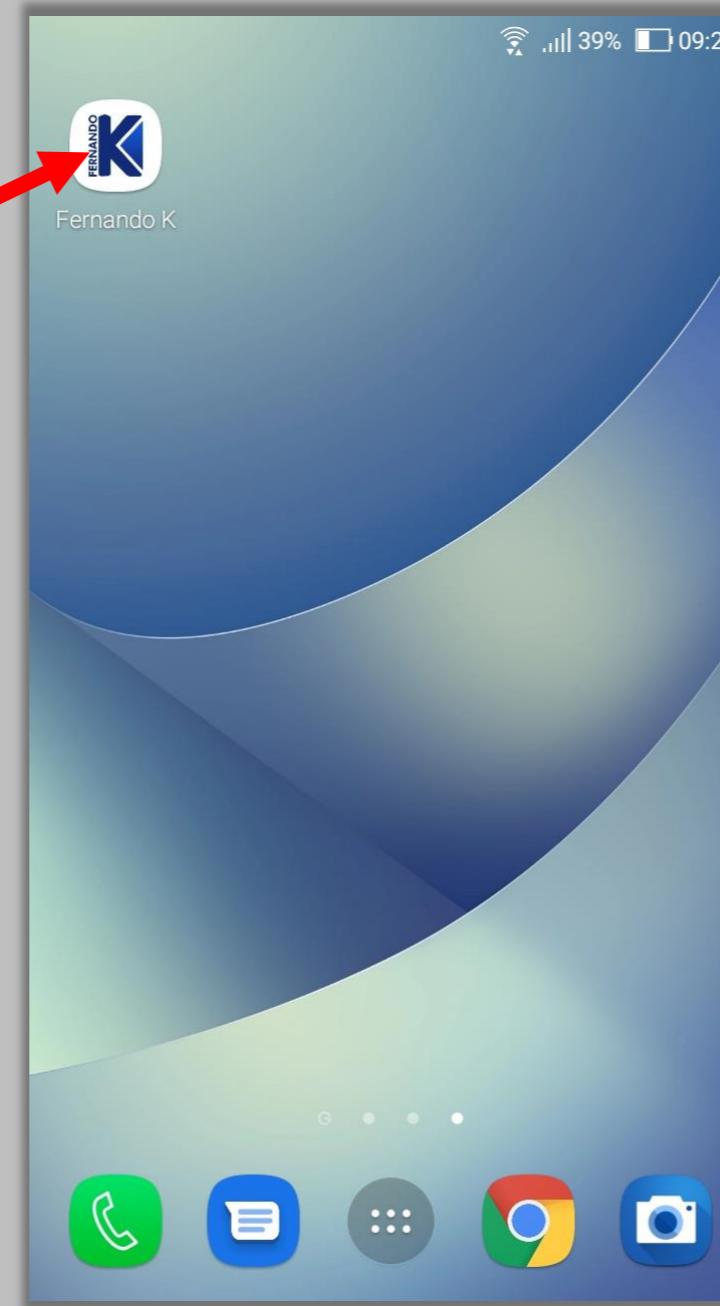


Telegram
fernandok_oficial



Utilizando o aplicativo Ícone

Execute o aplicativo



Utilizando o aplicativo Home

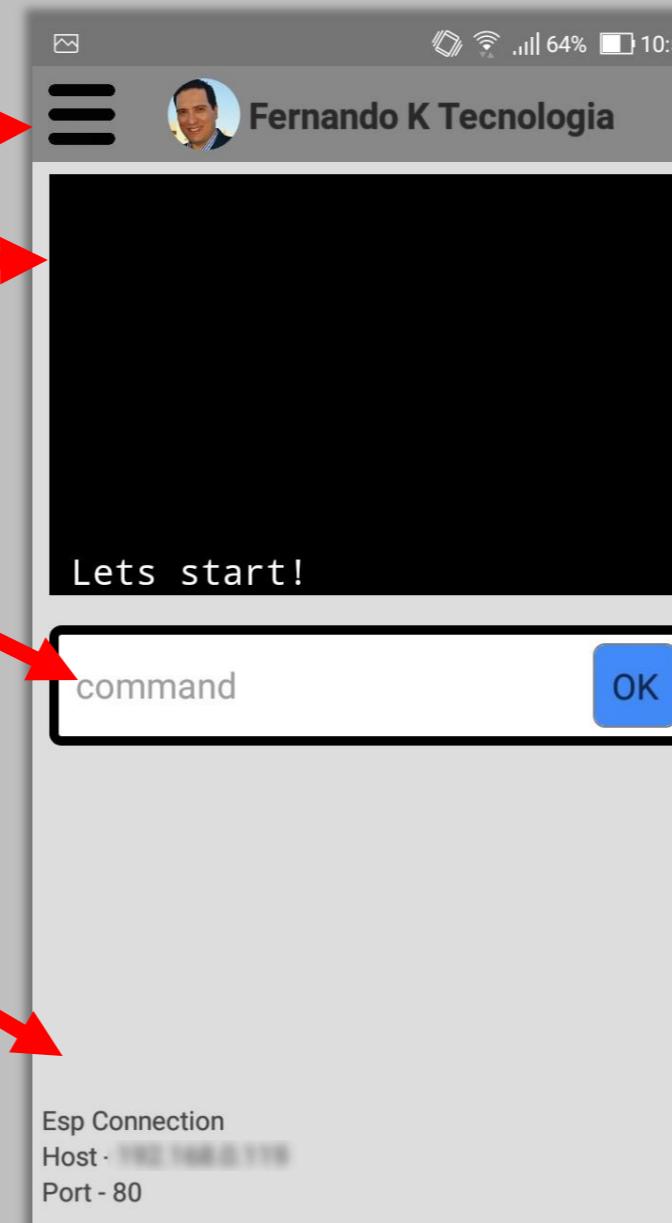
A tela inicial é composta pelos seguintes componentes

Configurações

Terminal (saída)

Linha de comando

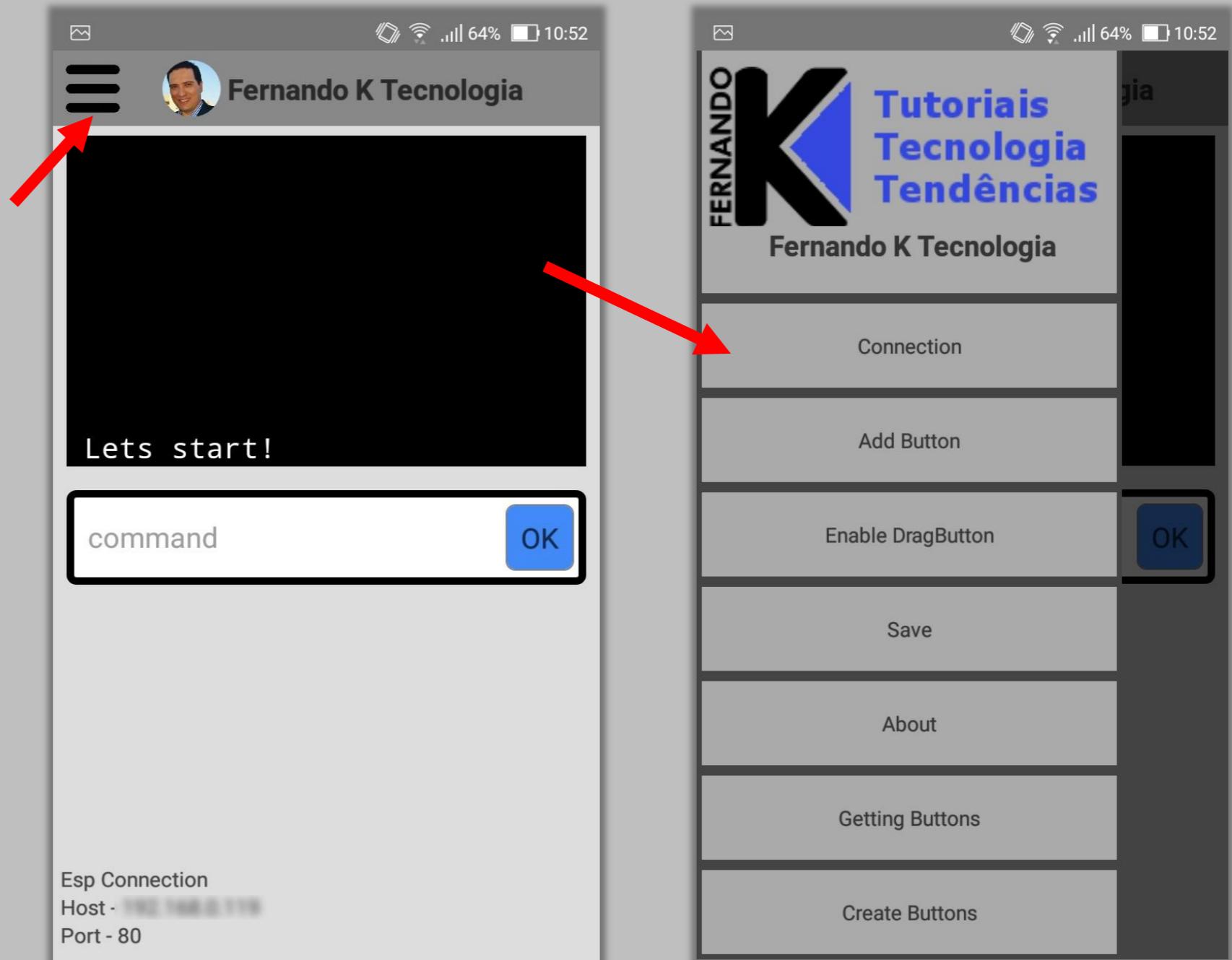
Informações de conexão:
descrição, ip e porta



Utilizando o aplicativo

Configurando nova conexão

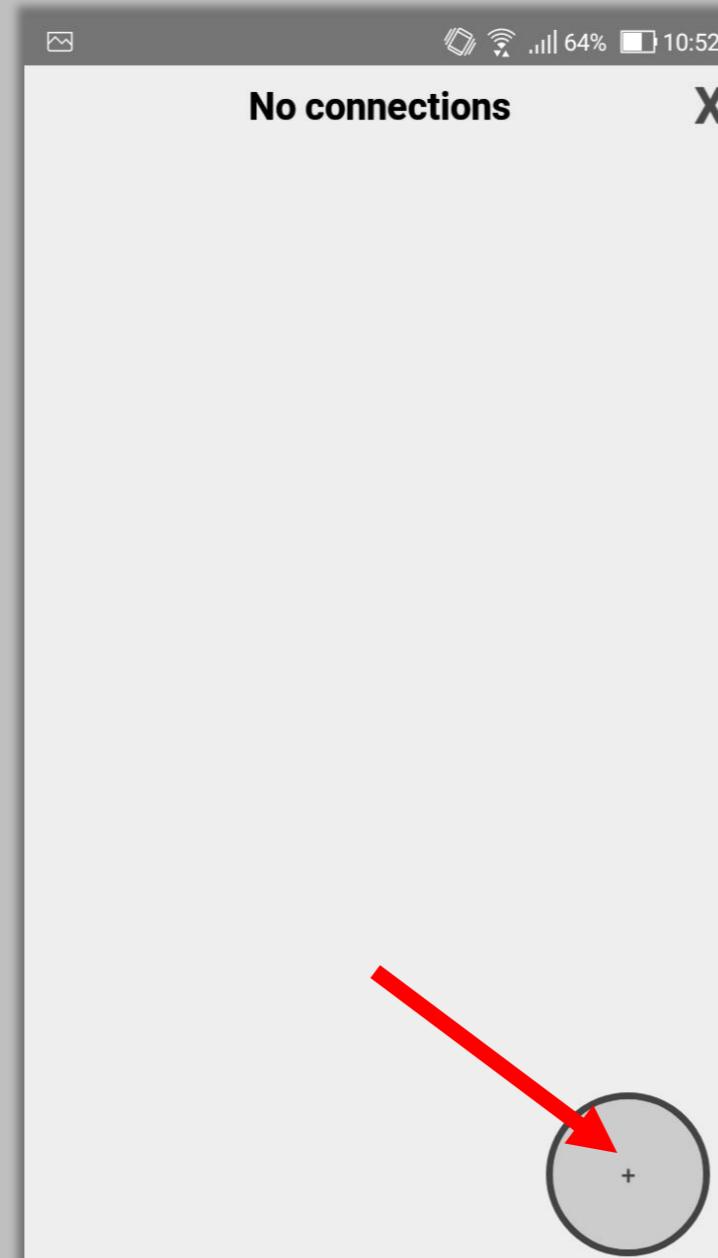
1. Selecione configurações
2. Selecione Connection



Utilizando o aplicativo

Configurando nova conexão

Clique no botão para **adicionar**



Utilizando o aplicativo

Configurando nova conexão

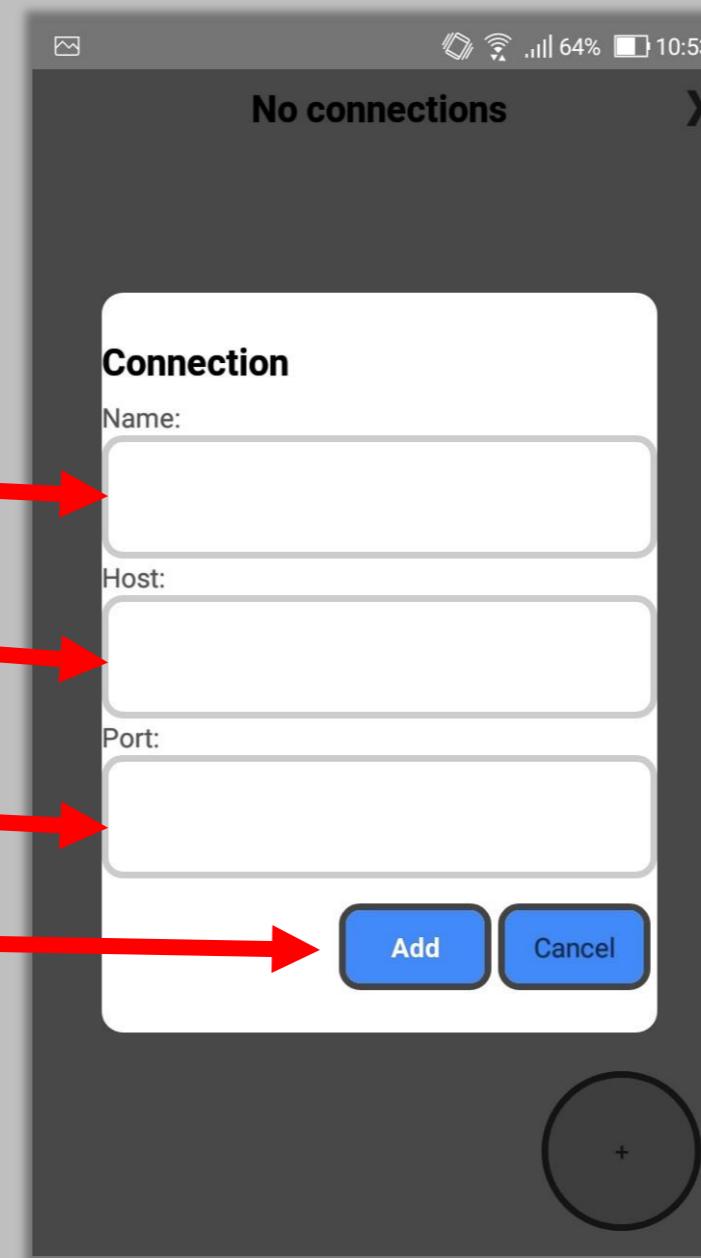
Preencha os campos

1. Descrição

2. IP do Host (ESP32)

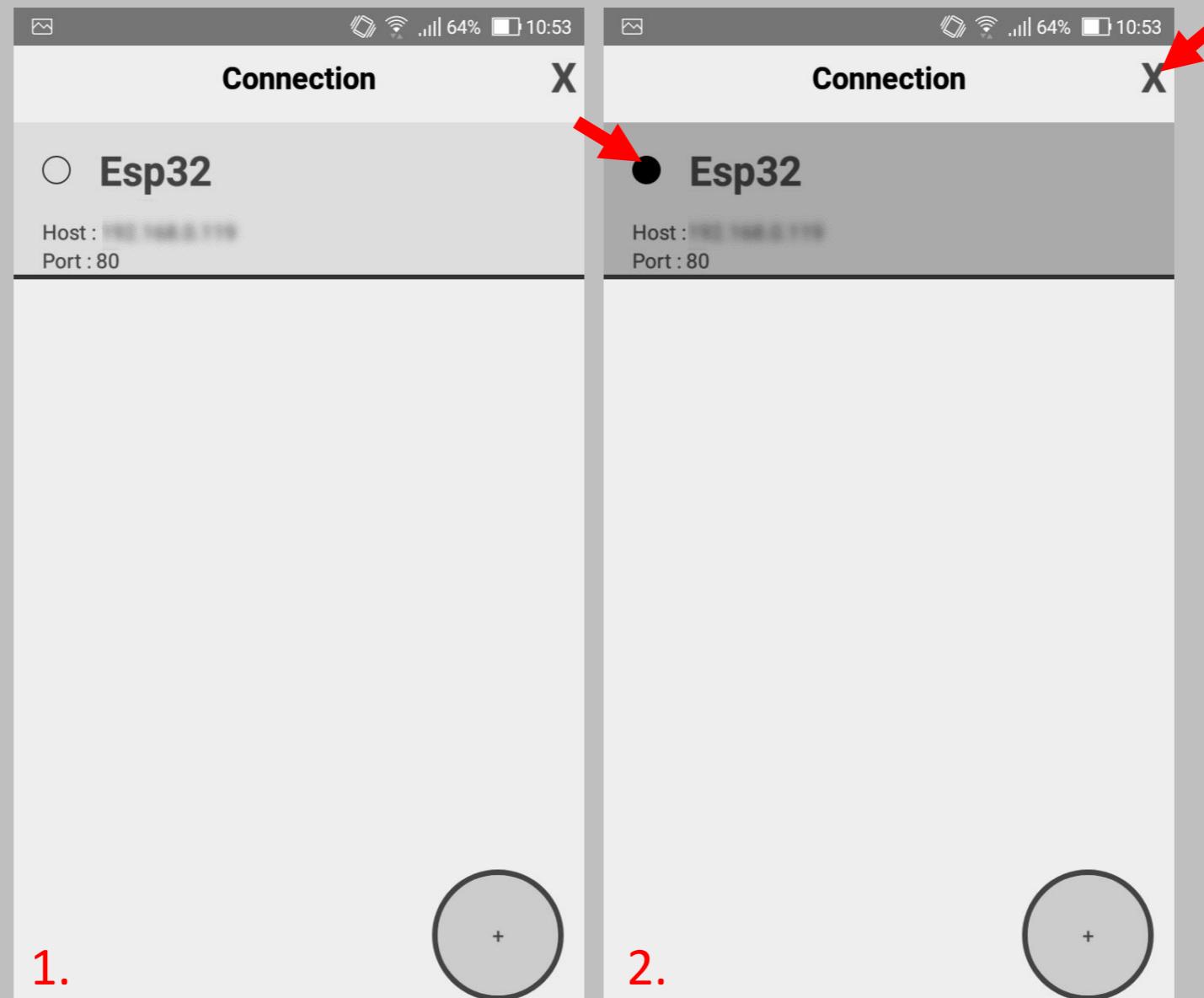
3. Porta

4. Adicionar



Utilizando o aplicativo

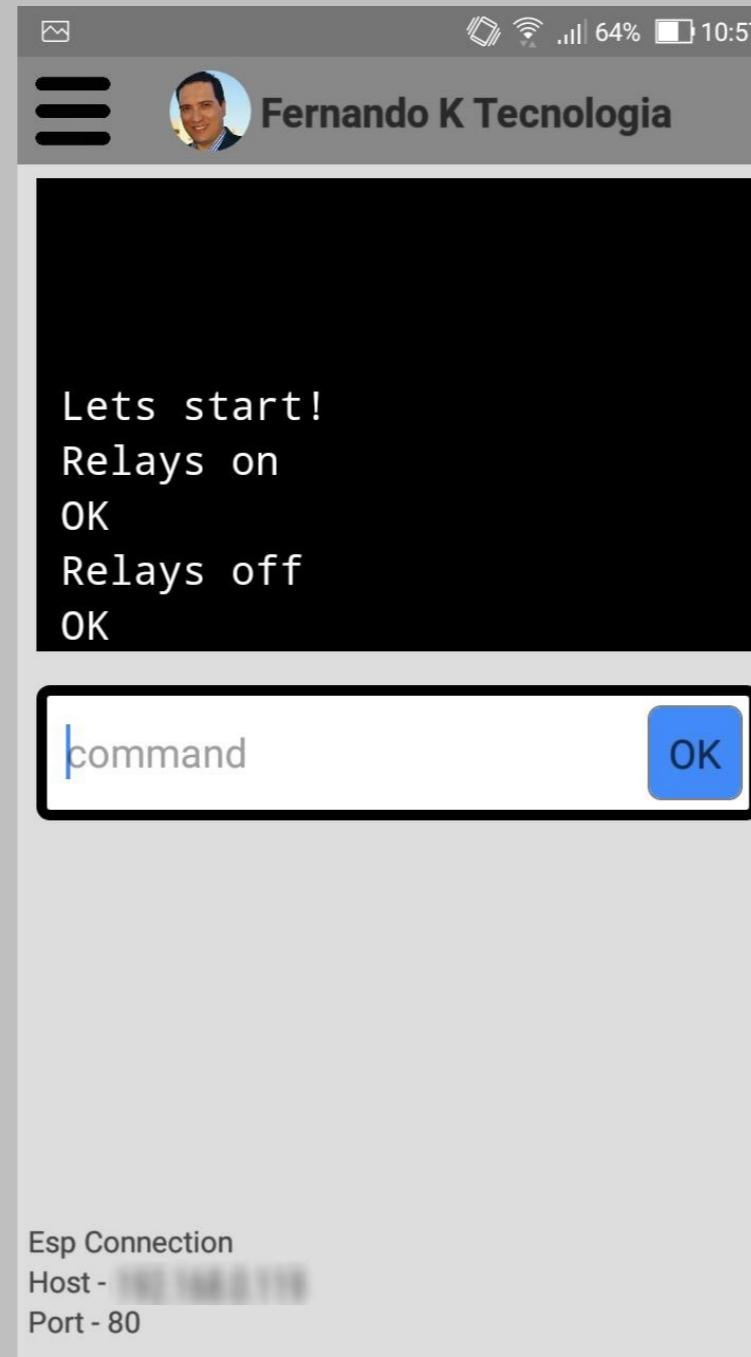
Um novo item da lista de conexões será exibido conforme a **figura 1**, selecione a conexão conforme a **figura 2** e retorne para a tela inicial clicando no X



Utilizando o aplicativo *Envio via Terminal*

Digite um *comando no campo de texto e clique em **OK**

*Os comandos devem estar devidamente programados no código fonte do ESP32, tanto para saída dos relés, quanto para envio de resposta para o aplicativo.



Utilizando o aplicativo

Criando um botão

1. Selecione Configurações
2. Selecione Add Button

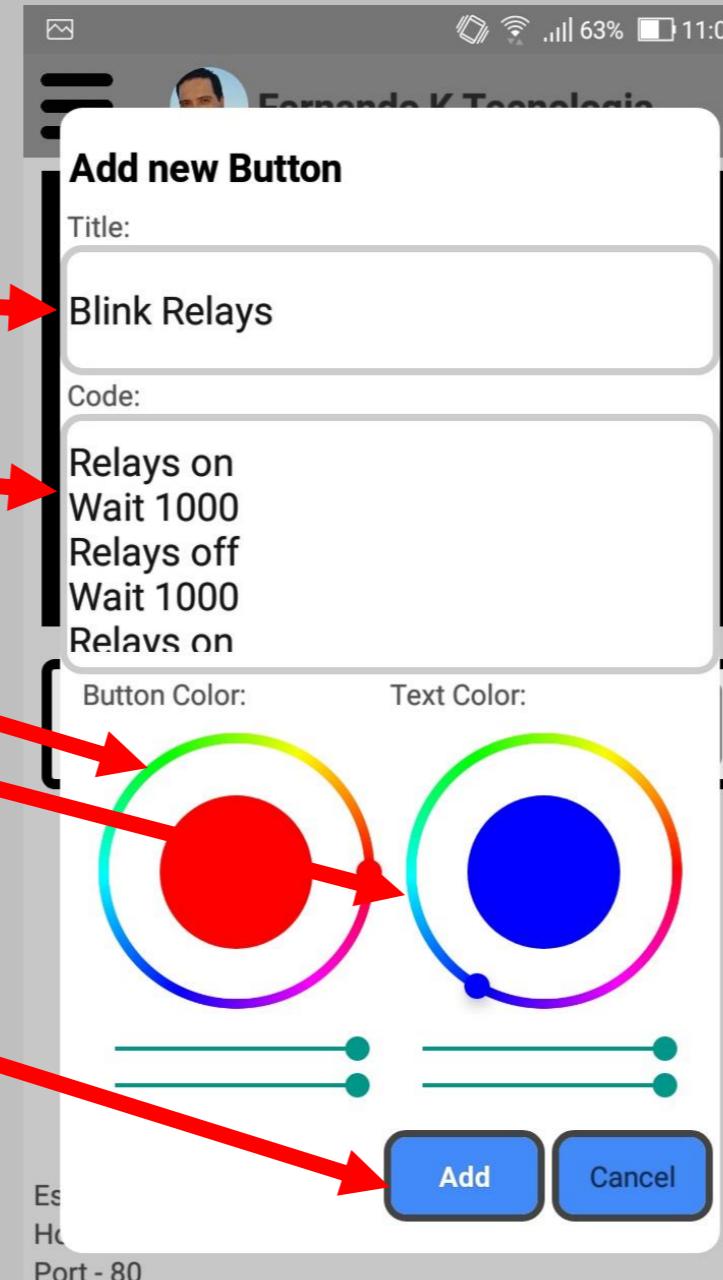


Utilizando o aplicativo

Criando scripts (botões)

Preencha os campos conforme a imagem

1. Título
2. *Script
3. Cor do botão
4. Cor do texto do botão
5. Adicionar



*A lista de comandos **reservados** (como o **Wait**) está disponível no *manual do usuário* do aplicativo

Utilizando o aplicativo

Criando scripts (botões)

Um novo botão será criado
conforme a imagem



Utilizando o aplicativo

Criando scripts (botões)

Você pode habilitar a movimentação de botões clicando em **EnableDragButton**



Utilizando o aplicativo

Criando scripts (botões)

Um ícone de **setas** será exibido indicando a movimentação do objeto.

Para editar/excluir o botão arraste para a **engrenagem** localizada no canto superior direito

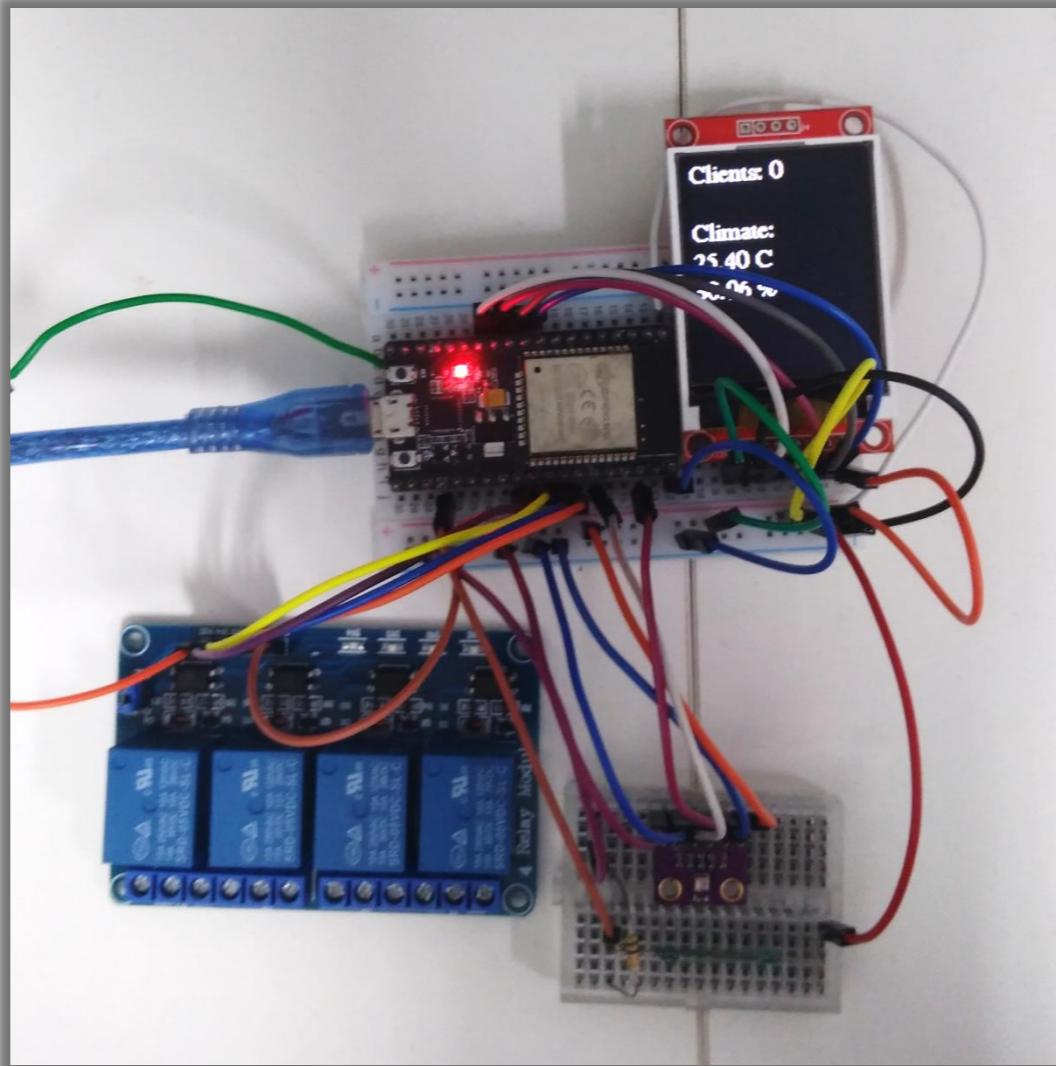


Botões utilizados no nosso exemplo

Os **botões** usados no nosso exemplo são os destacados na imagem

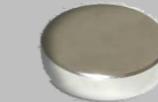
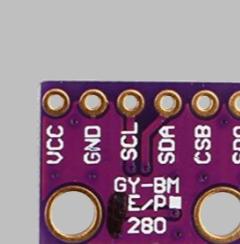


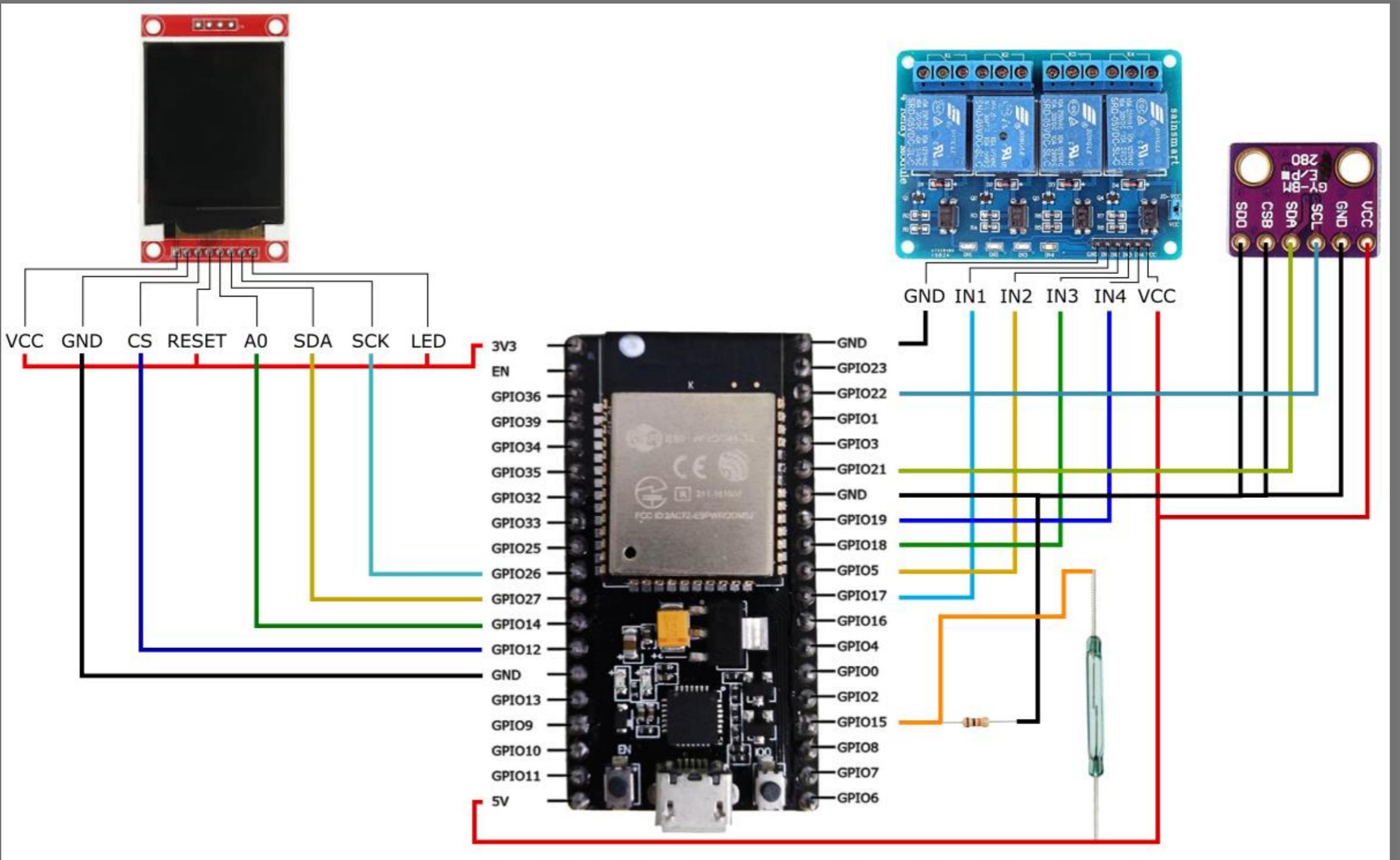
Montagem



Recursos usados

- ESP32 WROOM Dev Module
- Display TFT 1.8"
- Módulo de 4 relés
- BME280
- Reed Switch
- Imã de neodímio
- Resistor de 10k ohm
- Smartphone
- Aplicativo *AppFernandoK*
- Jumpers





Pinout

ESP32-WROOM-32 PINOUT (aka ESP32-DevKitC)

JTAG



	Power
	GND
	Serial Pin
	Analog Pin
	Control
	Physical Pin
	Port Pin
	Touch Pin
	DAC Pin
	PWM Pin

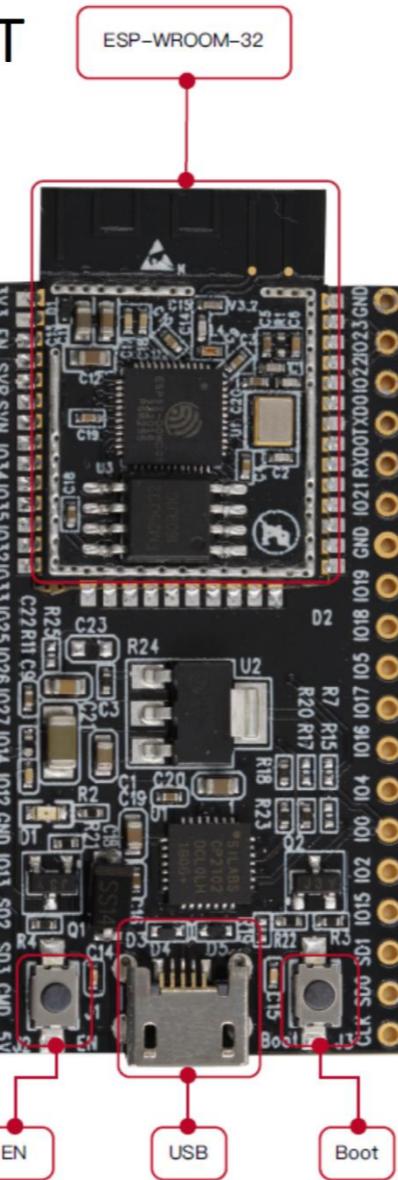
JTAG RESET	
	ChipPU
3.3V	9
EN	
SVP	
SVN	
IO34	
IO35	
IO32	
IO33	
IO25	
IO26	
IO27	
IO14	
IO12	
GND	
IO13	
SD2	
SD3	
CMD	
5V	

TMS	
HS2CLK	SDCLK
SDCLK	HSPICLK
HSPICLK	MTMS
Touch6	RTCIO16
RTCIO16	ADC2_6
ADC2_6	EMACTXD2
EMACTXD2	GPIO14
GPIO14	17
TDI	
HS2DATA2	SDDATA2
SDDATA2	HSPIQ
HSPIQ	MTDI
MTDI	Touch5
Touch5	RTCIO15
RTCIO15	ADC2_5
ADC2_5	EMACRXD3
EMACRXD3	GPIO12
GPIO12	18
GND	
TCK	
HS2DATA3	SDDATA3
SDDATA3	HSPIID
HSPIID	MTCK
MTCK	Touch4
Touch4	RTCIO14
RTCIO14	ADC2_4
ADC2_4	EMACRXER
EMACRXER	GPIO13
GPIO13	20
FLASH D2	
SDDATA2	HS1DATA2
HS1DATA2	U1RXD
U1RXD	SPIHD
SPIHD	GPIO9
GPIO9	28
FLASH D3	
SDDATA3	HS1DATA3
HS1DATA3	U1TXD
U1TXD	SPIWP
SPIWP	GPIO10
GPIO10	29
FLASH CMD	
SDCMD	HS1CMD
HS1CMD	U1RTS
U1RTS	SPICS0
SPICS0	GPIO11
GPIO11	30

Adapted from data at:

http://www.espressif.com/sites/default/files/documentation/esp_wroom_32_datasheet_en.pdf

https://cdn-shop.adafruit.com/product-files/3269/pinout_wroom_pinout.png



JTAG 20-pin	
Red 3V3	VTrst
Gray	RSTnTRST
Orange IO12	TDI
Yellow IO14	TMS
Blue IO13	TCK (to GND) RTCK
Purple IO15	TDO
	RESET
	DBGREQ
	5V-Supply
1 ● ● 2	---
3 ● ● 4	GND GND
5 ● ● 6	GND Black
7 ● ● 8	GND
9 ● ● 10	GND Green
11 ● ● 12	GND White
13 ● ● 14	GND
15 ● ● 16	Pins 14, 16, 18, 20: On some models like the high-end model J-Link PRO, these pins may not be connected to GND but are reserved for future use/extension. In case of doubt, leave open on target hardware.
17 ● ● 18	GND
19 ● ● 20	GND

GND	GND
IO23	36 GPIO23 VSPID ID HS1STROBE SPI MOSI
IO22	39 GPIO22 EMACTXD0 U0RTS VSPIW P Wire SCL
TXD0	41 GPIO1 EMACRXD2 U0TXD CLKOUT3 PROGRAM Port
RXD0	40 GPIO3 U0RXD CLKOUT2
IO21	42 GPIO21 EMACTXEN VSPID HD Wire SDA
GND	NC
IO19	38 GPIO19 EMACTXD0 U0CTS VSPIQ SPI MISO
IO18	35 GPIO18 VSPICLK HS1DATA7 SPI SCK
IO5	34 GPIO5 EMACRXD1 VSPICSE HS1DATA6 SPI SS
IO17	27 GPIO17 EMACCOL0 U2TXD HS1DATA5
IO16	25 GPIO16 EMACRXDOUT U2RXD HS1DATA4
IO4	24 GPIO4 EMACTXER ADC2_0 RTCIO10 Touch0 HSPID HD SD1DATA1 HS2DATA1
IO0	23 GPIO0 EMACTXCLK ADC2_1 RTCIO11 Touch1 CLKOUT1
IO2	22 GPIO2 HSPID WP ADC2_2 RTCIO12 Touch2
IO15	21 GPIO15 EMACRXD3 ADC2_3 RTCIO13 Touch3 MTD0 HSPICSO SDCMD HS2CMD TDO
SD1	33 GPIO8 SPIID U2CTS HS1DATA1 SD1DATA1 FLASH D1
SD0	22 GPIO7 SPIQ U2RTS HS1DATA0 SD1DATA0 FLASH D0
CLK	31 GPIO6 SPICLK U1CTS HS1CLK SDCLK FLASH SCK



Código

Instalação de Bibliotecas

Adafruit_GFX

<https://github.com/adafruit/Adafruit-GFX-Library>

Adafruit_ST7735

<https://github.com/adafruit/Adafruit-ST7735-Library>

Adafruit_BME280

https://github.com/adafruit/Adafruit_BME280_Library

Código *Diagrama*

taskReedSwitch

Task 1: Core 0

Leitura reed
switch

Envio de alerta
para os clients

Task responsável por **ler** o sensor reed switch e **enviar avisos** aos clientes caso o sensor seja ativado.

Código *Diagrama*

taskNewClients

Task 2: Core 0

Verificação de novos clientes

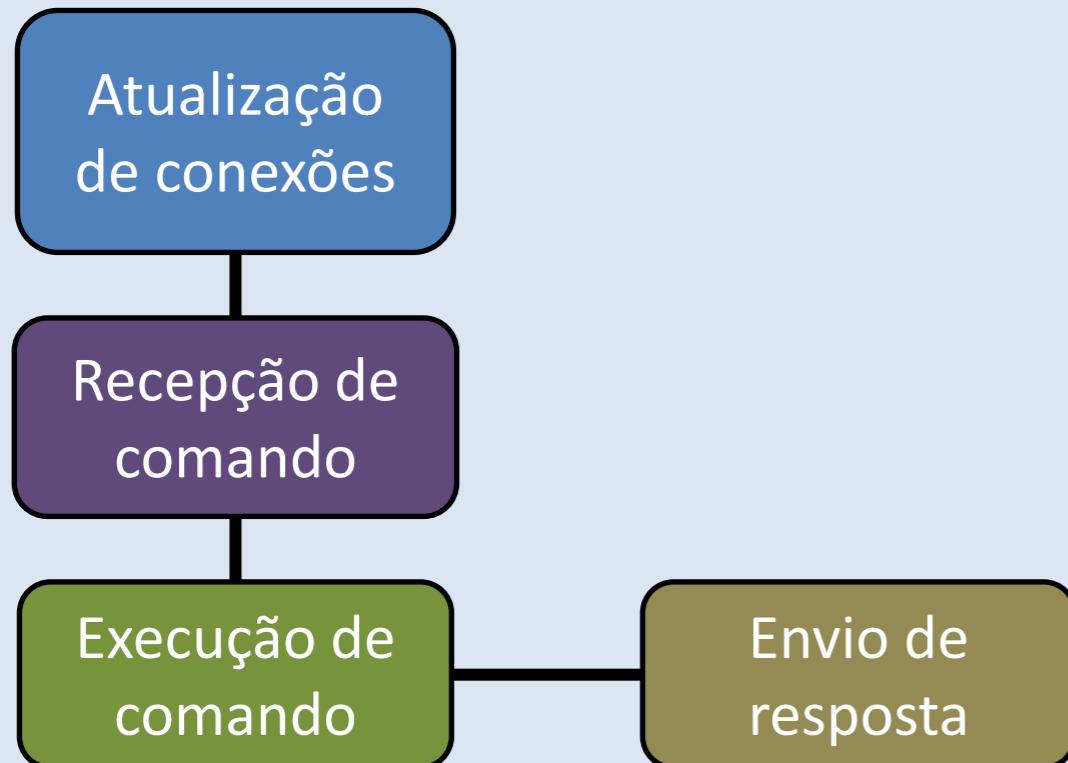
Recepção e inserção de novos clientes
(Vector)

Task responsável por **adicionar** os novos clientes na **lista de clientes conectados** (vector) utilizada pelo ESP.

Código *Diagrama*

handleClients

Task 3: Core 0

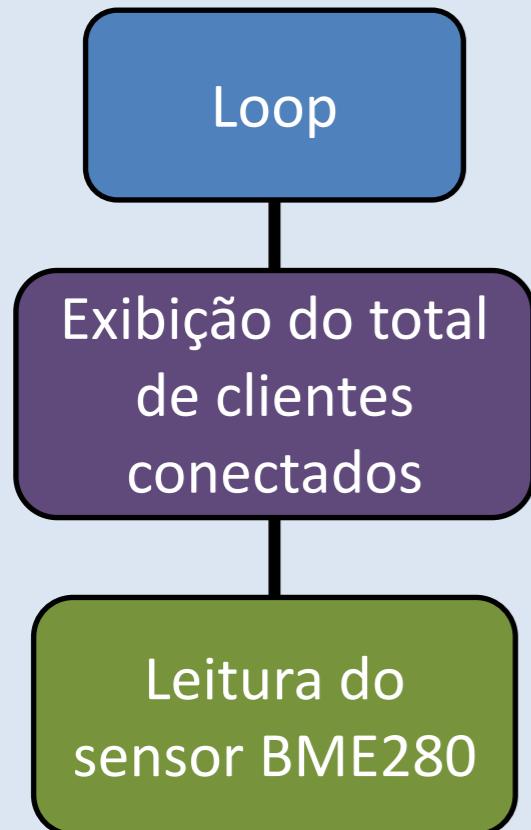


Task responsável por **ler** os comandos
recebidos dos clientes, **executar uma**
ação e **enviar uma resposta**.

Código *Diagrama*

loop

Task 4: Core 1



A função loop é responsável por **ler** o sensor de temperatura e umidade BME280 e **exibir** os valores no display a cada **5 segundos**, além de exibir a **quantidade de clientes conectados**.

Código Declarações e variáveis

```
#include <Adafruit_GFX.h> // Biblioteca de gráficos
#include <Adafruit_ST7735.h> // Biblioteca do hardware ST7735
#include <Fonts/FreeSerif9pt7b.h> // Fonte Serif que é usada no display
#include <WiFi.h> // Lib WiFi
#include <Wire.h> // Necessário apenas para o Arduino 1.6.5 e posterior
#include <SPI.h> // Lib de comunicação SPI
#include <Adafruit_BME280.h> // Lib do sensor BME
#include <vector> // Lib com as funções de vetor (vector)

const char* ssid = "SSID"; // Coloque o nome da sua rede wifi aqui
const char* password = "PASSWORD"; // Coloque a sua senha wifi aqui

const IPAddress IP = IPAddress(111,111,1,111); // IP fixo que o ESP utilizará
const IPAddress GATEWAY = IPAddress(111,111,1,1); // Gateway
const IPAddress SUBNET = IPAddress(255,255,255,0); // Máscara
const int port = 80; // Porta

WiFiServer server(port); // Objeto WiFi Server, o ESP será o servidor
std::vector<WiFiClient> clients; // Vetor com os clientes que se conectarão no ESP

// Objeto do sensor BME280
Adafruit_BME280 bme; //SDA = GPIO21, SCL = GPIO22
```

Código *Declarações e variáveis (Continuação)*

```
// ESP32-WROOM
#define TFT_CS 12 // CS
#define TFT_DC 14 // A0
#define TFT_MOSI 27 // SDA
#define TFT_CLK 26 // SCK
#define TFT_RST 0 // RESET
#define TFT_MISO 0 // MISO

// Objeto do display tft
Adafruit_ST7735 display = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST);

// Pinos ligados nos relés
const int relay1 = 17, relay2 = 5, relay3 = 18, relay4 = 19;
// Pino ligado no reed switch
const int reedSwitch = 15;

// Variável que obtém a temperatura e umidade
String climate = "";
// Flag que indica se o display está ocupado
bool displayIsBusy = false;
// Altura da fonte que é usada no display (9 + 5 espaçamento)
int fontHeight = 14;
```

Código *Declarações e variáveis (Continuação)*

```
// Task de leitura do sensor reed switch
void taskReedSwitch(void *);

// Task que insere novos clientes (recém conectados) no vector
void taskNewClients(void *);

// Task que recebe executa comandos dos clientes
void handleClients(void *);
```

Código Setup

```
void setup()
{
    // Iniciamos a serial com 9600 de velocidade
    Serial.begin(9600);

    // Iniciamos o display com fundo preto
    display.initR(INITR_BLACKTAB);
    // Exibimos a mensagem "Starting..."
    showDisplay("Starting...", true);

    // Configuramos os pinos dos relés e reed switch (pinMode)
    setPins();
    // Iniciamos o sensor BME280
    bmeBegin();
    // Iniciamos o servidor (WiFi Server)
    serverBegin();
    // Criamos 3 tasks (mais detalhes no escopo da função)
    createTasks();
}
```

Código Loop

```
→ void loop()
{
    // Exibimos o total de clientes conectados
    → showClients();
    // Lemos o sensor BME280 obtendo a temperatura e umidade
    → readBMESensor();
    // Aguardamos 5 segundos
    delay(5000);
}
```

Loop

Exibição do total
de clientes
conectados

Leitura do
sensor BME280

Código *Setup - createTasks*

```
// Criamos as 3 Tasks
void createTasks()
{
    //Criamos a task de leitura do reed switch
    xTaskCreatePinnedToCore
    (
        taskReedSwitch, //Função que será executada
        "readReedSwitch", //Nome da tarefa
        10000, //Tamanho da pilha
        NULL, //Parâmetro da tarefa
        2, //Prioridade da tarefa
        NULL, //Caso queria manter uma referência para a tarefa que vai ser criada (no caso não precisamos)
        0 //Número do core que será executada a tarefa (usamos o core 0 para o loop ficar livre com o core
        1)
    );

    //Criamos a task que insere os novos clientes no vector
    xTaskCreatePinnedToCore(taskNewClients,"taskNewClients",10000,NULL,2,NULL,0);

    //Criamos a task que recebe e executa os comandos dos clients conectados
    xTaskCreatePinnedToCore(handleClients,"handleClients",10000,NULL,2,NULL,0);
}
```

Código Task 1: Core 0 - taskReedSwitch

```
// Task que lê o reed switch
void taskReedSwitch(void* p)
{
    // Valor que representa 1ms, usado nos vTaskDelays
    TickType_t taskDelay = 1 / portTICK_PERIOD_MS;
    // O loop deve ser infinito (para chamarmos constantemente o vTaskDelay que, por sua vez,
    // alimenta o Watchdog e evita a reinicialização indesejada do ESP pelo Watchdog)
    while (true)
    {
        // Se o reed switch estiver com sinal alto
        → if(digitalRead(reedSwitch) == HIGH)
        {
            // Exibe mensagem na serial e no display
            Serial.println("reed high");
            showDisplay("Reed switch\nActivated!", true);

            // Envia a mensagem "The sensor has been activated!" para todos os clientes conectados
            for(int i=0; i<clients.size(); i++)
            {
                if(clients[i])
                    → clients[i].print("The sensor has been activated!");

                vTaskDelay(taskDelay);
            }
        }
    }
}
```

Leitura reed
switch

Envio de alerta
para os clients

Código Task 1: Core 0 – taskReedSwitch (Continuação)

```
// Aguardamos até que o sensor seja desativado
while(digitalRead(reedSwitch) == HIGH)
    vTaskDelay(taskDelay);
}
else
{
    // Se o sensor for desativado, exibimos na serial
    Serial.println("reed low");
    // E aguardamos até que ele volte a ser ativado
    while(digitalRead(reedSwitch) == LOW)
        vTaskDelay(taskDelay);
}
// Importante: A task deve conter um delay para alimentar o Watchdog
vTaskDelay(taskDelay);
}
```

Código Task 2: Core 0 – taskNewClients

```
// Task que insere novos clientes conectados no vector
void taskNewClients(void *p)
{
    // Objeto WiFiClient que receberá o novo cliente conectado
    WiFiClient newClient;
    // Tempo esperado no delay da task (1 ms)
    TickType_t taskDelay = 1 / portTICK_PERIOD_MS;

    while(true)
    {
        // Se existir um novo client atribuimos para a variável
        → newClient = server.available();
        // Se o client for diferente de nulo
        if(newClient)
        {
            // Inserimos no vector
            → clients.push_back(newClient);
            // Exibimos na serial indicando novo client e a quantidade atual de clients
            Serial.println("New client! size:"+String(clients.size()));
        }
        // Aguardamos 1ms
        vTaskDelay(taskDelay);
    }
}
```

Verificação de novos clientes

Recepção e inserção de novos clientes (Vector)

Código Task 3: Core 0 – handleClients

```
// Função que verifica se um cliente enviou um comando
void handleClients(void *p)
{
    String cmd; // String que receberá o comando
    TickType_t taskDelay = 1 / portTICK_PERIOD_MS; // Tempo aguardado pela task (1 ms)
    while(true) // Loop infinito
    {
        // Atualizamos o vector deixando somente os clientes conectados
        → refreshConnections();
        // Percorremos o vector
        for(int i=0; i<clients.size(); i++)
        {
            if(clients[i].available()) // Se existir dados a serem lidos
            {
                → cmd = clients[i].readStringUntil('\n'); // Recebemos a String até o '\n'
                // Executamos um comando, enviando por parametro a String cmd, e o client que nos enviou o comando
                → → executeCommand(cmd, clients[i]);
            }
            vTaskDelay(taskDelay); // Delay de 1 ms
        }
        vTaskDelay(taskDelay); // Delay de 1 ms
    }
}
```

Atualização
de conexões

Recepção de
comando

Execução de
comando

Envio de
resposta

Código *refreshConnections* (função chamada pela task *handleClients*)

```
// Função que verifica se um ou mais clients se desconectaram do server e, se sim, estes clients
serão retirados do vector
void refreshConnections()
{
    bool flag = false; // Flag que indica se pelo menos um client se desconectou
    std::vector<WiFiClient> newVector; // Objeto que receberá apenas os clients conectados

    for(int i=0; i<clients.size(); i++) // Percorremos o vector
    {
        if(!clients[i].connected()) // Verificamos se o client está desconectado
        {
            // Exibimos na serial que um cliente se desconectou e a posição em que ele está no vector
            Serial.println("Client disconnected! ["+String(i)+"]");
            clients[i].stop(); // Desconectamos o client
            flag = true; // Setamos a flag como true indicando que o vector foi alterado
        }
        else
            newVector.push_back(clients[i]); // Se o client está conectado, adicionamos no newVector
    }
    // Se pelo menos um client se desconectou, atribuimos ao vector "clients" os clients de "newVector"
    if(flag)
        clients = newVector;
}
```

Código *executeCommand* (função chamada pela task *handleClients*)

```
// Função que executa um comando de acordo com a String "cmd"
void executeCommand(String cmd, WiFiClient client)
{
    // String que guarda a resposta que será enviada para o client
    String response = "";
    // Vetor com os 4 pinos dos relés usados neste exemplo, que facilitará a escrita do código fonte
    int relays[4] = {relay1, relay2, relay3, relay4};

    // Se a String estiver vazia, abortamos a função
    if (cmd.equals(""))
        return;

    // Deixamos a string toda em maiúsculo
    cmd.toUpperCase();

    // Exibimos o comando recebido na serial e no display
    Serial.println("Recebido: ["+cmd+"]");
    showDisplay("Command\n received:", true);
    showDisplay(cmd, false);

    // Se o comando for igual a status
    if(cmd.equals("STATUS"))
        response = getStatus(); // Montamos a String de resposta com o status de cada relé
    else
```

Código executeCommad (Continuação)

```
if(cmd.equals("RELAYS ON")) // Se for igual a Relays on, ligamos todos os relés
{
    for(int i=0; i<4; i++)
        digitalWrite(relays[i], LOW); // Lógica inversa

    response = "OK";
}
else
if(cmd.equals("RELAYS OFF")) // Se for igual a Relays off, desligamos todos os relés
{
    for(int i=0; i<4; i++)
        digitalWrite(relays[i], HIGH);

    response = "OK";
} // Se for igual a relay 1 on, relay 1 off, relay 2 on... e assim por diante, executamos uma ação
de acordo com o comando
else
if(cmd.equals("RELAY 1 ON") || cmd.equals("RELAY 2 ON") || cmd.equals("RELAY 3 ON") ||
cmd.equals("RELAY 4 ON") || cmd.equals("RELAY 1 OFF") || cmd.equals("RELAY 2 OFF") ||
cmd.equals("RELAY 3 OFF") || cmd.equals("RELAY 4 OFF"))
{

```

Código executeCommand (Continuação)

```
int index = atoi((cmd.substring(cmd.indexOf(" ")+1, cmd.indexOf(" ")+2)).c_str());  
if(cmd.indexOf("ON")>=0)  
    digitalWrite(relays[index-1], LOW);  
else  
    digitalWrite(relays[index-1], HIGH);  
  
response = "OK";  
}  
else  
if(cmd.equals("CLIMATE")) // Se for igual a climate, atribuimos a String 'response' a última  
leitura de temperatura e umidade feita pelo BME280  
    response = climate;  
else// Se não for nenhum dos comandos acima, retornamos "Comando inválido"  
    response = "Invalid command";  
  
Serial.println("Resposta: ["+response+"]"); // Exibimos a resposta na serial (debug)  
  
// Enviamos para o client passado por parâmetro e exibimos sucesso ou falha na serial  
→ if(client.print(response)>0)  
    Serial.println("Resposta enviada");  
else  
    Serial.println("Erro ao enviar resposta");  
}
```

Envio de
resposta

Código *getStatus* (função chamada por “*executeCommand*”)

```
// Função que lê os pinos dos relés e retorna o estado atual em uma String
String getStatus()
{
    String status = "Relay 1: ";

    if(digitalRead(relay1) == LOW)
        status+="ON";
    else
        status+="OFF";

    status += "\n Relay 2: ";

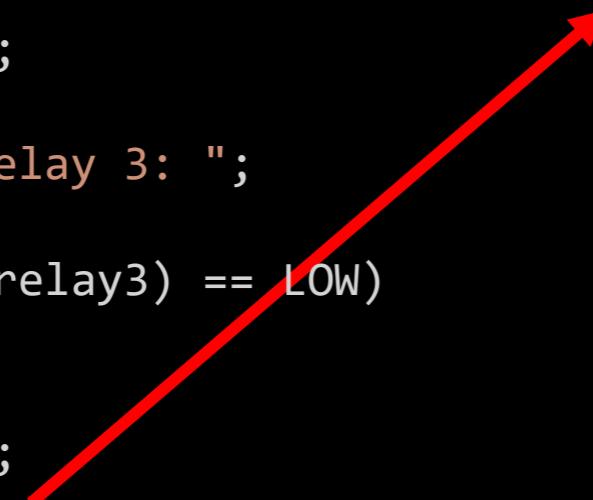
    if(digitalRead(relay2) == LOW)
        status+="ON";
    else
        status+="OFF";

    status += "\n Relay 3: ";

    if(digitalRead(relay3) == LOW)
        status+="ON";
    else
        status+="OFF";

    status += "\n Relay 4 ";
    if(digitalRead(relay4) == LOW)
        status+="ON";
    else
        status+="OFF";
}

return status;
```



Código *Setup - setPins*

```
// Configuramos os pinos INPUT e OUTPUT
void setPins()
{
    pinMode(reedSwitch, INPUT);
    pinMode(relay1, OUTPUT);
    pinMode(relay2, OUTPUT);
    pinMode(relay3, OUTPUT);
    pinMode(relay4, OUTPUT);

    digitalWrite(relay1, HIGH);
    digitalWrite(relay2, HIGH);
    digitalWrite(relay3, HIGH);
    digitalWrite(relay4, HIGH);
}
```

Código *Setup - bmeBegin*

```
// Iniciamos o sensor BME280 exibindo sucesso ou falha
void bmeBegin()
{
    //0x76 = o pino SD0 do sensor deve ser ligado no GND
    //0x77 = o pino SD0 do sensor deve ser ligado no VCC
    if (!bme.begin(0x76))
    {
        Serial.println("Sensor bme failed!");
        showDisplay("Sensor bme failed!", false);
    }
    else
    {
        Serial.println("Sensor bme ok");
        showDisplay("Sensor bme ok", false);
    }
}
```

Código *Setup - serverBegin*

```
// Conectamos no WiFi e iniciamos o servidor
void serverBegin()
{
    WiFi.begin(ssid, password); // Iniciamos o WiFi
    showDisplay("WiFi Connecting", false); //Exibimos na serial e no display
    Serial.println("Connecting to WiFi"); // Enquanto não estiver conectado exibimos um ponto

    while (WiFi.status() != WL_CONNECTED)
    {
        display.print(".");
        Serial.print(".");
        delay(1000);
    }
    // Exibimos na serial e no display a mensagem OK
    Serial.println("OK");
    showDisplay("OK", true);

    // Configuramos o WiFi com o IP definido anteriormente
    WiFi.config(IP, GATEWAY, SUBNET);
    // Iniciamos o servidor
    server.begin(port);
    // Printamos o IP (debug)
    Serial.println(WiFi.localIP());
}
```

Código *Loop – showClients e readBMEsensors*

```
// Função que exibe a quantidade de clientes conectados no servidor do ESP
void showClients()
{
    showDisplay("Clients: "+String(clients.size()), true);
}
void readBMESensor() // Função que lê a temperatura e umidade do sensor BME280
{
    float t = bme.readTemperature();
    float h = bme.readHumidity();
    if(isnan(t) || isnan(h))
    {
        Serial.println("Erro ao ler sensor");
        showDisplay("Climate:\n-\n-", false);
        climate = "Climate:\n -\n -";
    }
    else
    {
        Serial.println((String(t))+";"+(String(h)));
        showDisplay("\nClimate:\n"+String(t)+" C", false);
        showDisplay(String(h)+" %", false);
        climate = "Climate:\n "+String(t)+" C";
        climate += "\n "+String(h)+" %";
    }
}
```

Código *showDisplay*

```
// Verifica se o display está ocupado e exibe a mensagem no display
// Se clear estiver como true então limpamos o display antes da exibição
void showDisplay(String msg, bool clear)
{
    if(!displayIsBusy)
    {
        displayIsBusy = true;
        if(clear)
            resetDisplay();

        display.println(msg);

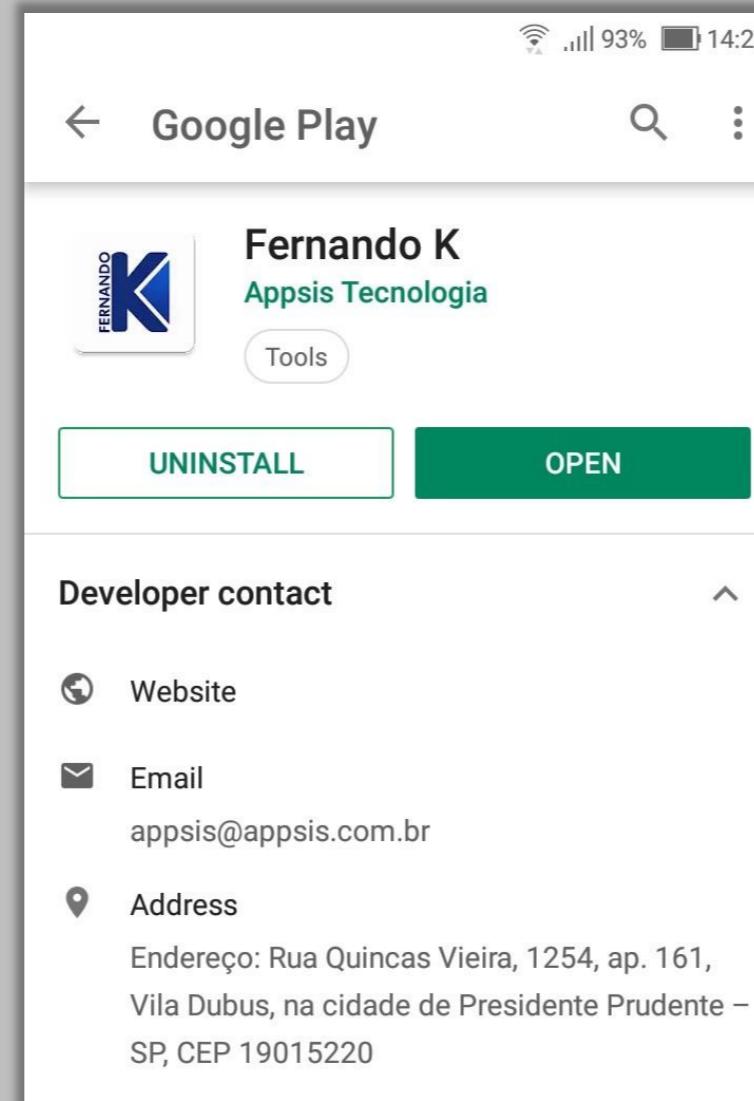
        displayIsBusy = false;
    }
}
```

Código *resetDisplay*

```
// Limpa e reconfigura o display
void resetDisplay()
{
    display.setFont(&FreeSerif9pt7b);
    display.fillScreen(ST77XX_BLACK);
    display.setTextColor(ST7735_WHITE);
    display.setCursor(0,fontHeight);
    display.setTextSize(1);
}
```

Link para download do App

<https://play.google.com/store/apps/details?id=com.appfernandok>



Em www.fernandok.com

Download arquivos PDF e INO do código fonte

