Instituto Superior Técnico

Aplicações e Computação para a Internet das Coisas

20xx-20xx

3º Laboratório: A Internet das Coisas

Grupo:		
Aluno 1:		
Aluno 2:		

Objetivo

O objetivo deste trabalho é conectar dois controladores ESP32 através da internet, separando os atuadores e sensores mas continuando a permitir que os valores dos sensores controlem os atuadores. Para interligar os controladores vai ser usada a plataforma de IoT ThingsBoard.

Descrição

Este trabalho baseia-se nas montagens dos laboratórios anteriores, mas com uma diferença principal:

- Os Sensores estarão ligados a um controlador ESP32.
- Os Atuadores estarão ligados a outro controlador ESP32.

Assim a funcionalidade do sistema será idêntica à do sistema implementado no 2º laboratório:

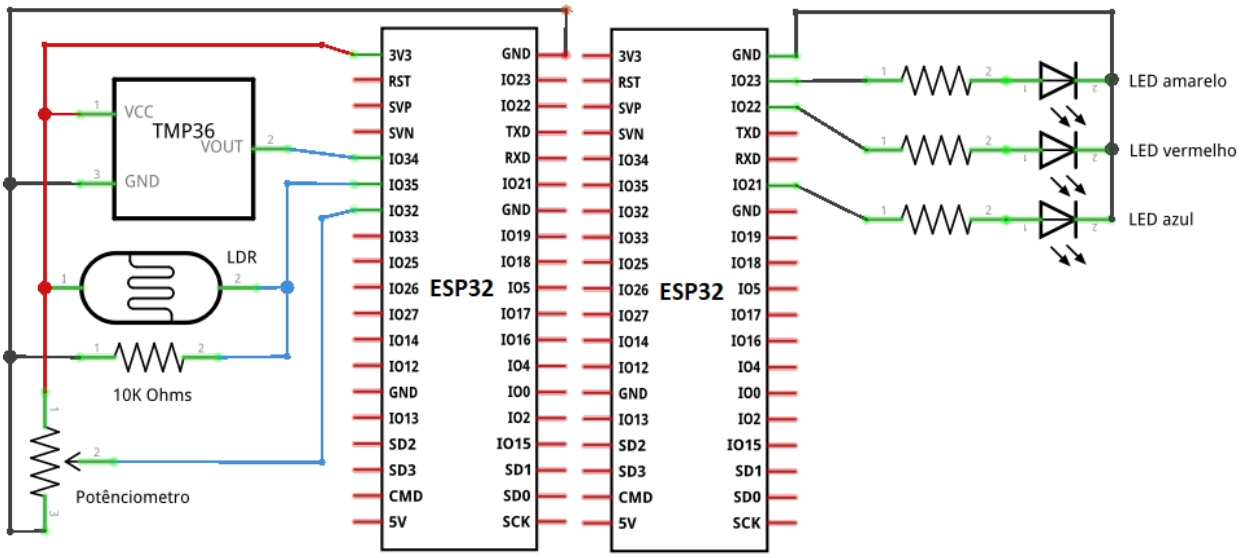
"*Construa um sistema embebido (...) para simultaneamente controlar 3 LEDs (os atuadores) dependendo do estado de 3 atuadores - temperatura, ângulo de rotação (potenciômetro) e intensidade da luz*

- ***Temperatura*** *- O LED amarelo associado à temperatura deve estar ligado quando a temperatura lida for maior que 26ºC (valor que eventualmente pode ser redefinido no laboratório de acordo com as condições ambientais).*
- ***Rotação*** *- O LED verde controlado pelo potenciômetro deve piscar com um período de tempo a variar continuamente entre 0.2 e 2 segundos, dependendo do ângulo de rotação do potenciômetro.*
- ***Luz*** *- O LED azul deve ajustar continuamente a sua própria intensidade com base na intensidade de luz medida no ambiente em que se encontra. O seu comportamento deve ser:*
    - *Escuridão - LED totalmente ligada*
    - *Luminoso - LED desligada*"

Para implementar esta funcionalidade será agora necessário ligar ambos os ESP32 a uma

rede WiFi de modo a que estes tenham acesso ao mesmo servidor do ThingsBoard.

Um diagrama do circuito a montar está representado na seguinte figura.



Referências

1. ThingsBoard: https://thingsboard.io/docs/
2. SensorBox: https://github.com/JoaoSaramago/SensorBox-ThingsBoard
3. ThingsBoard Arduino SDK: https://github.com/thingsboard/thingsboard-arduino-sdk
4. ThingsBoard REST API: https://demo.thingsboard.io/swagger-ui.html
5. Repositório do ThingsBoard Python REST Client: https://github.com/thingsboard/thingsboard-python-rest-client
6. Página Web do ThingsBoard Python REST Client: https://thingsboard.io/docs/reference/rest-api/
7. ThingsBoard WebSocket API: https://thingsboard.io/docs/user-guide/telemetry/#websocket-api
8. ThingsBoard usar RPCs: https://thingsboard.io/docs/user-guide/rpc/
9. Python WebSocket: https://pypi.org/project/websocket-client/

Recomendações

De modo a realizar o seu trabalho em segurança e não estragar nenhum equipamento usado, lembre-se de ter em conta as recomendações abaixo mencionadas. Enquanto trabalha, preencha as caixas para ter a certeza que cumpre todas as medidas de segurança.

Trabalhe sempre com os circuitos desconectados da sua fonte de alimentação.	
Chame o professor, ou o responsável pelo laboratório, antes de conectar os circuitos à sua fonte de alimentação.	
Verifique se o circuito está bem montado (sensores, resistências, capacitores, etc.) para prevenir curto-circuito ou estragos no equipamento. (Ex. nunca conecte um LED diretamente ao pino do controlador e ao GND, ou VCC.)	
Evite dobrar os terminais dos componentes o máximo possível. Se necessário, (ex. resistências) dobre o terminal a aproximadamente 5mm do corpo do componente.	

Este trabalho deverá ser implementado em 2 sessões de laboratório.

Tenha atenção ao número de leituras dos sensores que envia por segundo para o servidor do ThingsBoard.

Caso as medições feitas pelo ADC se tornem demasiado instáveis devido a interferências com o WiFi e instabilidade na alimentação do ESP32, considere utilizar métodos para suavizar os valores enviados, por exemplo, fazer várias leituras consecutivas e enviar uma média destas.

ThingsBoard

O ThingsBoard é uma plataforma de IoT open-source. Permite interligar e gerir dispositivos como sensores e atuadores, guardar, visualizar e processar dados recolhidos e criar novas aplicações com base nos dados recolhidos.

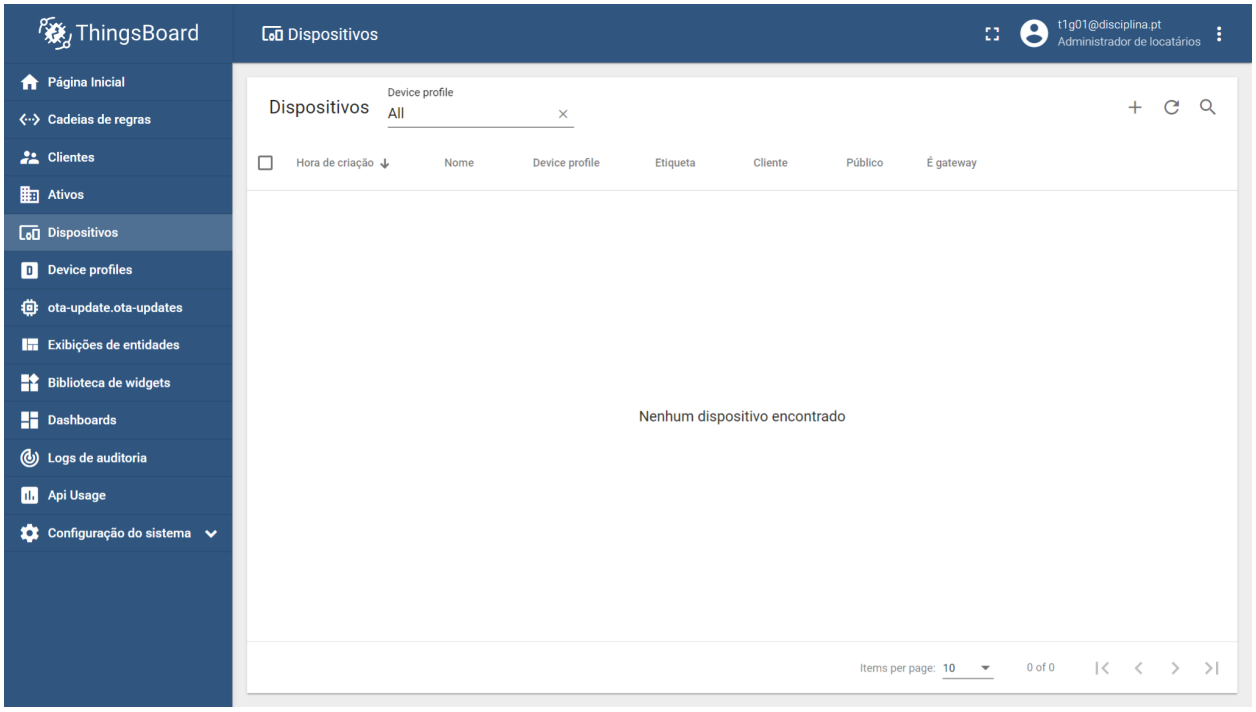
Os dispositivos comunicam diretamente com a plataforma usando MQTT, HTTP ou CoAP, no entanto é possível usar *gateways* para traduzir outros protocolos para MQTT e serem interpretados pelo sistema. Neste trabalho o protocolo de comunicação usado será o MQTT.

Os utilizadores da plataforma estão organizados em 3 níveis de privilégios:

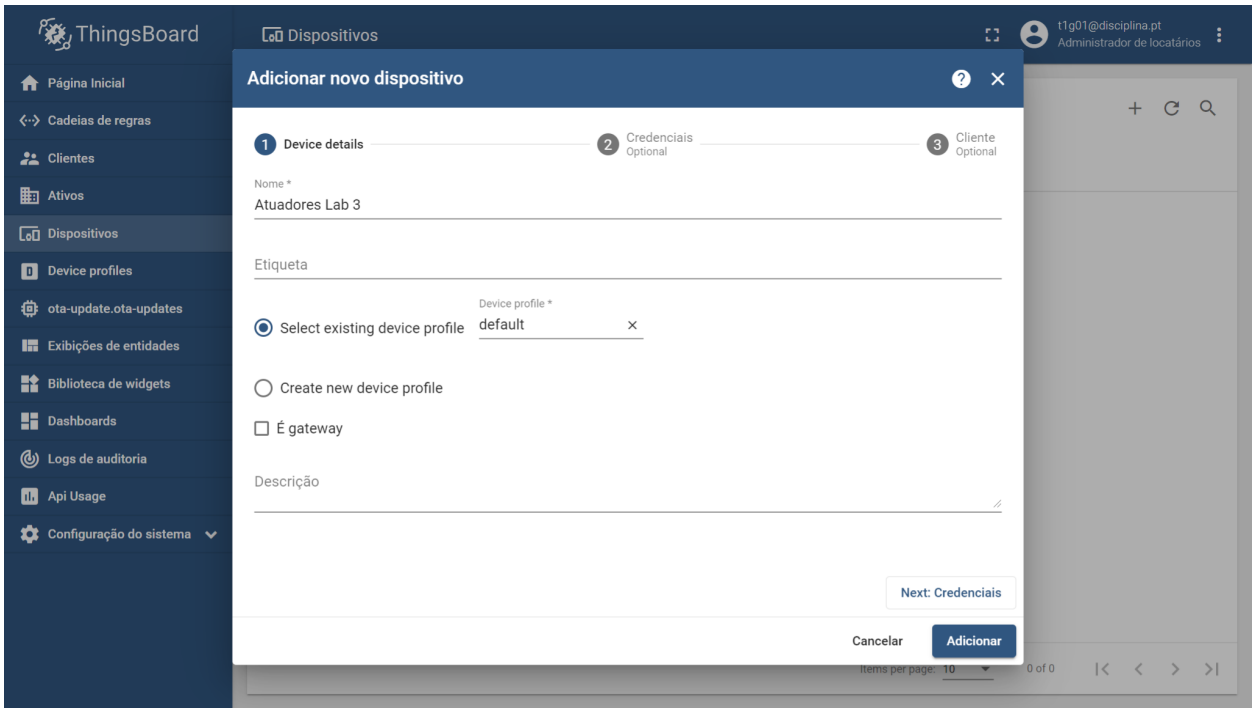
- O **Administrador do Sistema** é o responsável pelo servidor e empresas que usam o sistema para alojar os seus dispositivos.
- O **Locatário(Tenant)** é equiparado a uma unidade de negócio, empresa que tem ou produz dispositivos.
- O **Cliente** é o consumidor final, capaz de visualizar os seus painéis de controlo e de visualizar e gerir dispositivos a que tem acesso.

Irá receber credenciais de acesso ao sistema como um Locatário, pessoa que pode criar os seus dispositivos e vender o acesso aos seus dados a clientes. A versão do ThingsBoard utilizada foi a 3.3.1.

Para criar dispositivos inicie sessão com os seus dados e selecione, na barra lateral, **Dispositivos**.

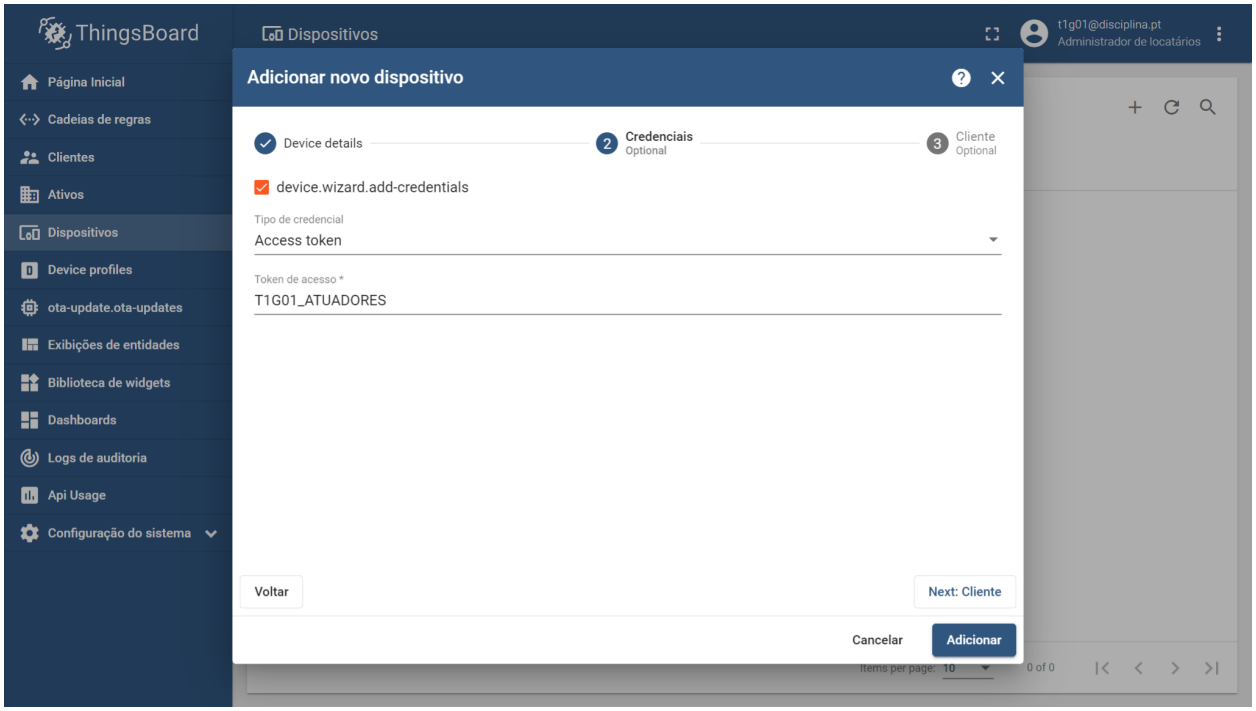


Clique no símbolo **+** e escolha **Adicionar Novo Dispositivo**.



Escolha o nome que desejar e clique em **Next: Credentials** para definir as credenciais de

acesso para o dispositivo comunicar com o servidor.



Marque a caixa de **Adicionar Credenciais**. No tipo de credencial escolha **Access Token** e escreva um que seja único, em todo o sistema não será possível haver Access Tokens iguais entre dispositivos, mesmo que de diferentes Locatários. Por fim clique em **Adicionar** e deverá voltar à página de dispositivos onde já aparece o novo dispositivo.

SensorBox-ThingsBoard

Para simplificar a conexão dos controladores à plataforma de IoT foi criada uma biblioteca para o Arduino IDE que estende a oficial já existente do ThingsBoard para Arduino. Esta biblioteca possui todas as funcionalidades da biblioteca oficial, simplificando ainda a ligação à rede WiFi, ao servidor do ThingsBoard e a subscrição de mensagens para os atuadores, entre outras funcionalidades.

Esta biblioteca deve ser descarregada do GitHub (ver referências) e instalada no Arduino IDE como .zip.

A biblioteca oficial do ThingsBoard também deve ser instalada como .zip. A versão mais atualizada no momento da escrita é a 0.5.0, no entanto esta tem um problema com a receção de comandos RPC. Assim a versão usada neste trabalho é a deste commit:

https://github.com/thingsboard/thingsboard-arduino-sdk/tree/23aaf47f4838e887493aeafaef92f76e4159bfb7

Uma vez lançada a nova versão da biblioteca oficial já será possível instala-la a partir do gestor de bibliotecas do Arduino IDE com esta correção.

Para usar a biblioteca devem ser instaladas através do gestor de bibliotecas do Arduino IDE as seguintes bibliotecas:

- PubSubClient - versão 2.8.0
- ArduinoJson - versão 6.18.4

No código, a biblioteca deve ser incluída no topo da seguinte forma:

```
#include <SensorBoxTB.h>
```

Depois, definir o nome da rede WiFi, a palavra-passe da mesma, o IP ou URL do servidor do ThingsBoard e o token do dispositivo:

```
char* ssid               = "your-ssid";
char* password           = "your-password";
char* thingsboard_server = "thingsboard.rnl.tecnico.ulisboa.pt";
char* token              = "device_token";
```

Para criar um objeto SensorBox, o termo usado pela biblioteca para definir o dispositivo com sensores e atuadores, use o seguinte:

```
SensorBox sb(ssid, password, thingsboard_server, token);
```

Para inicializar as comunicações, isto é, ligar à rede WiFi e ao servidor do ThingsBoard, use dentro da função `setup()` o seguinte:

```
void setup() {
  ...
  sb.initComms();
  ...
}
```

De modo a manter as comunicações abertas deve ser sempre executado na função `loop()` o seguinte método:

```
void loop() {
  ...
  sb.loop();
}
```

Agora o dispositivo já se encontra conectado à plataforma de IoT e pronto para publicar valores de sensores. O envio de dados dos sensores é feito da seguinte forma para variáveis `float` ou `int`:

```
sb.sendTelemetryFloat("temperature", temperatureValue);
sb.sendTelemetryInt("rotation_angle", rotationAngleValue);
```

Mais tipos de variáveis podem ser enviadas usando os métodos adequados (ver documentação da biblioteca oficial do ThingsBoard).

Para receber informações que manipulem atuadores (ex. ligar um LED) é preciso estar subscrito a essas mensagens. Para isso o ThingsBoard usa RPC, chamada de procedimento remoto, isto é, através da API do ThingsBoard é possível executar funções do controlador e receber os resultados remotamente. As funções a serem executadas devem ser criadas da seguinte forma:

```
RPC_Response processSetLetIntensity(const RPC_Data &data) {
  // exemplo
  led_intensity = data["value"];
  ...
  return RPC_Response("led_intensity", led_intensity);
}
```

A variável `data` contém uma lista da informação recebida do ThingsBoard, necessária para executar a função, neste caso o valor para a intensidade chama-se `value` mas poderia ter sido enviado com qualquer outro nome. No final da função devolvemos o resultado da mesma como se fosse uma variável, neste caso enviamos a variável `led_intensity` com o valor recebido para confirmar que a função foi corretamente executada e que foi lido o valor correto.

Para o dispositivo saber a que RPCs deve subscrever cria-se a lista com estes, sendo o primeiro argumento o nome público do processo que é chamado pela API e o segundo argumento é a função que é desencadeada por esta chamada:

```
RPC_Callback callbacks[] = {
  {"setLetIntensity", processSetLetIntensity},
};
```

Com fim a subscrever aos RPCs executa-se o seguinte método na função `setup()`, após se inicializar as comunicações:

```
sb.subscribeActuatorsCmds(callbacks, COUNT_OF(callbacks));
```

Assim, já é possível acionar os atuadores remotamente.

Interligação de sensores e atuadores

Os dispositivos são feitos para publicar dados e/ou acionar atuadores, o processamento dos dados e decisão sobre o que deve ser feito é idealmente realizado por uma aplicação fora dos controladores, usando a API do ThingsBoard. Assim podemos usar diretamente a REST API em HTTP ou usar as bibliotecas, nas linguagens de programação disponíveis, que simplificam o acesso à REST API. Uma das bibliotecas existentes é em Python. (Ver referências)

Na API o início de sessão já é feito com as credenciais do utilizador em vez das dos dispositivos, pois agora estamos a agir como utilizadores da plataforma.

Para comandar um atuador é preciso o ID do dispositivo que o contém, este pode ser encontrado na página de administração do próprio ou pode ser obtido através da API. A função

a chamar é a `handle_two_way_device_rpc_request(device_id, body)` onde o `body` deve estar em formato de dicionário, por exemplo :

```
{
  "method": "setLetIntensity",
  "params": {
    "value": 200
  }
}
```

E o `device_id` deve ser um objeto `DeviceId`, por exemplo:

```
DeviceId('DEVICE', deviceID_string)
```

Para subscrever os dados dos sensores é preciso abrir um WebSocket, o qual esta biblioteca Python não faz, assim terá de usar a REST API diretamente em conjunto com uma biblioteca de WebSockets de Python (sugestão nas referências). Para abrir a conexão com WebSocket use o seguinte endereço:

```
"wss://thingsboard.rnl.tecnico.ulisboa.pt/api/ws/plugins/telemetry?to
ken=" + token
```

O token do utilizador pode ser obtido através da biblioteca do ThingsBoard pois já foi feito o início de sessão, este devolve um token que é depois usado pela API durante o resto da aplicação. Pode ser obtido desta forma:

```
token = rest_client.configuration.api_key["X-Authorization"]
```

Após a ligação ser efetuada podemos então subscrever aos sensores do dispositivo que desejamos. A estrutura da mensagem de subscrição deverá ser:

```
{
  "tsSubCmds": [
    {
      "entityType": "DEVICE",
      "entityId": sensors_deviceID_str,
      "scope": "LATEST_TELEMETRY",
      "cmdId": 1
    }
  ],
  "historyCmds": [],
  "attrSubCmds": []
}
```

Onde a variável `sensors_deviceID_str` é o ID do dispositivo com os sensores. Tendo isto sido feito com sucesso a aplicação deverá começar a receber as mensagens dos sensores que podem ser tratadas e desencadear o comportamento desejado dos atuadores.

Programe a aplicação

Forneça o código criado, tanto dos dois controladores como da aplicação, adequadamente estruturado e comentado.

Evite usar a função `delay` pois esta pára a execução do programa durante o intervalo de tempo especificado.

Resultados

1. Descreva as modificações necessárias ao código do laboratório 2 para cumprir as funcionalidades deste laboratório.
    a. Descreva alguma mudança feita no modelo de comportamento de software utilizado (como a mudança de tarefas no *Round-Robin loop*).
    b. Descreva outras implementações mudadas (implementação de tarefas).
2. Avalie o desempenho das comunicações no seu sistema (fluxo de dados, latência).

Proposta de resolução

- Código atuadores:

```
#include <SensorBoxTB.h>

const char* ssid               = "your-ssid";
const char* password           = "your-password";
const char* thingsboard_server = "host-or-ip";
const char* token              = "T1G01_ATUADORES";

const int tempLedPin = 23;
const int potLedPin = 22;
const int lightLedPin = 21;

// PWM setup
const int pwmChannel = 0;
const int pwmFreq = 5000;
const int pwmResolution = 8;

unsigned long prevBlinkCheck = 0;

int blinkInterval = 200;

RPC_Response setLedTempAlarm(const RPC_Data &data) {
  bool ledTempState = data["value"];
  digitalWrite(tempLedPin, ledTempState);
  return RPC_Response("yellow_led_alarm", ledTempState);
}

RPC_Response setLedIntensity(const RPC_Data &data) {
  int intensity = data["value"];
  ledcWrite(pwmChannel, intensity);
  return RPC_Response("blue_led_intensity", intensity);
}

RPC_Response setLedBlinkInterval(const RPC_Data &data) {
  blinkInterval = data["value"];
  return RPC_Response("red_led_blink_interval", blinkInterval);
}

RPC_Callback callbacks[] = {
  {"setLedTempAlarm", setLedTempAlarm},
  {"setLedIntensity", setLedIntensity},
  {"setLedBlinkInterval", setLedBlinkInterval},
};

SensorBox sb(ssid, password, thingsboard_server, token);

void setup() {
  Serial.begin(115200);
```

```
  pinMode(tempLedPin, OUTPUT);
  pinMode(potLedPin, OUTPUT);
  pinMode(lightLedPin, OUTPUT);

  ledcSetup(pwmChannel, pwmFreq, pwmResolution);
  ledcAttachPin(lightLedPin, pwmChannel);

  sb.initComms();
  sb.subscribeActuatorsCmds(callbacks, COUNT_OF(callbacks));
}

void loop() {
  setLedBlinkIntervalTask();
  sb.loop();
}

void setLedBlinkIntervalTask() {
  if (millis() - prevBlinkCheck > blinkInterval) {
    prevBlinkCheck = millis();
    digitalWrite(potLedPin, LOW);
  } else if (millis() - prevBlinkCheck > blinkInterval/2) {
    digitalWrite(potLedPin, HIGH);
  }
}
```

- Código sensores:

```
#include <SensorBoxTB.h>

const char* ssid               = "your-ssid";
const char* password           = "your-password";
const char* thingsboard_server = "host-or-ip";
const char* token              = "T1G01_SENSORES";

const int tempSensorPin = 34;
const int lightSensorPin = 35;
const int potSensorPin = 32;

// Calibration
const float tempOffset = 0.0;
const int potMin = 0;
const int potMax = 4095;
const int lightSensorMin = 0;
const int lightSensorMax = 4095;

// Multi-Sampling
const int n_samples = 20;
```

```
const int tempTaskDelay = 1000;
const int lightTaskDelay = 500;
const int blinkTaskDelay = 500;

unsigned long lastTimeTemp = 0;
unsigned long lastTimeLight = 0;
unsigned long lastTimeBlink = 0;

SensorBox sb(ssid, password, thingsboard_server, token);

void setup() {
  Serial.begin(115200);
  sb.initComms();
}

void loop() {
  readBlinkIntervalTask();
  readLightTask();
  readTempTask();
  sb.loop();
}

void readTempTask() {
  if ( millis() - lastTimeTemp < tempTaskDelay ) {
    return;
  }

  int reading = multiSampling(tempSensorPin);
  float temperature = (((reading / 4096.0) * 3.3) - 0.5) *100;
  temperature = temperature + tempOffset;
  temperature = roundf(temperature * 10) / 10;  // Round to one decimal place
  Serial.print("Temperature: "); Serial.println(temperature);
  sb.sendTelemetryFloat("temperature", temperature);
  lastTimeTemp = millis();
}

void readLightTask() {
  if ( millis() - lastTimeLight < lightTaskDelay ) {
    return;
  }

  int reading = multiSampling(lightSensorPin);
  int intensity = map(reading, lightSensorMin, lightSensorMax, 0, 255);
  Serial.print("Intensity: "); Serial.println(intensity);
  sb.sendTelemetryInt("light_intensity", intensity);
  lastTimeLight = millis();
}

void readBlinkIntervalTask() {
  if ( millis() - lastTimeBlink < blinkTaskDelay ) {
    return;
```

```
  }

  int reading = multiSampling(potSensorPin);
  int readBlinkInterval = map(reading, potMin, potMax, 200, 2000);
  Serial.print("Read Blink Interval (ms): "); Serial.println(readBlinkInterval);
  sb.sendTelemetryInt("blink_interval", readBlinkInterval);
  lastTimeBlink = millis();
}

// Measures da analog pin n_samples times and returns the mean of those readings
int multiSampling(int pin) {
  int sum = 0;
  int reading = 0;
  for (int i=0; i<n_samples; i++){
    reading = analogRead(pin);
    sum += reading;
  }
  return sum / n_samples;
}
```

- Código aplicação:

```
import json
import logging
import websocket
import _thread
# Importing models and REST client class from Community Edition version
from tb_rest_client.rest_client_ce import *
from tb_rest_client.rest import ApiException


logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s - %(levelname)s - %(module)s - %(lineno)d -
%(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S')

# ThingsBoard server url
base_url = "thingsboard.rnl.tecnico.ulisboa.pt"
server_url = "https://" + base_url

username = "t1g01@disciplina.pt"
password = "t1g01!"
actuators_deviceID_str = "f7a0e3c0-0f66-11ec-822a-d79efbc35f5b"
sensors_deviceID_str = "69f47780-1006-11ec-943a-d79efbc35f5b"


# Creating the REST client object with context manager to get auto token refresh
with RestClientCE(base_url=server_url) as rest_client:
    actuators_deviceID = DeviceId('DEVICE', actuators_deviceID_str)

    def on_message(ws, message):
```

```
        msg = json.loads(message)
        sensors_data = msg["data"]

        if 'temperature' in sensors_data:
            temp = sensors_data["temperature"][0][1]
            print("Temperature:", temp)
            temp_alarm = True if float(temp) > 22.0 else False
            body_alarm = {
                "method": "setLedTempAlarm",
                "params": {
                    "value": temp_alarm
                }
            }
            rest_client.handle_two_way_device_rpc_request(actuators_deviceID, body_alarm)

        if 'blink_interval' in sensors_data:
            interval = sensors_data["blink_interval"][0][1]
            print("Blink Interval:", interval)
            body_blink = {
                "method": "setLedBlinkInterval",
                "params": {
                    "value": interval
                }
            }
            rest_client.handle_two_way_device_rpc_request(actuators_deviceID, body_blink)

        if 'light_intensity' in sensors_data:
            intensity = sensors_data["light_intensity"][0][1]
            intensity = 255 - int(intensity)
            print("Light Intensity:", intensity)
            body_intensity = {
                "method": "setLedIntensity",
                "params": {
                    "value": intensity
                }
            }
            rest_client.handle_two_way_device_rpc_request(actuators_deviceID,
body_intensity)


    def on_error(ws, error):
        print(error)


    def on_close(ws, close_status_code, close_msg):
        print("### websocket closed ###")


    def on_open(ws):
        def run(*args):
            sub_cmd = {
```

```
                "tsSubCmds": [
                    {
                        "entityType": "DEVICE",
                        "entityId": sensors_deviceID_str,
                        "scope": "LATEST_TELEMETRY",
                        "cmdId": 1
                    }
                ],
                "historyCmds": [],
                "attrSubCmds": []
            }
            sub_cmd_str = json.dumps(sub_cmd)
            ws.send(sub_cmd_str)
            print("Subscribed to sensors")

        _thread.start_new_thread(run, ())

    try:
        rest_client.login(username=username, password=password)
        print("Inicio de sessao feito com sucesso!")

        token = rest_client.configuration.api_key["X-Authorization"]
        ws_url = "wss://" + base_url + "/api/ws/plugins/telemetry?token=" + token
        ws = websocket.WebSocketApp(ws_url, on_open=on_open, on_message=on_message,
on_error=on_error, on_close=on_close)

        ws.run_forever(sslopt={"cert_reqs": ssl.CERT_NONE})

    except ApiException as e:
        logging.exception(e)
```