

Escola Superior de Tecnologia de Tomar

IoT

Professor Luís Oliveira

Projeto Final

Trabalho realizado por:

Pedro Ferreira, nº 21264

João Figueiredo, nº 24179

Tomar, Maio 2024

Engenharia Informática

Introdução

Este projeto final de Internet das Coisas (IoT) envolve a implementação de um sistema de monitorização ambiental utilizando um sensor BME280, um Raspberry Pi Pico W, e o protocolo MQTT (Mosquitto). O sensor BME280, conhecido pela sua precisão e fiabilidade, é capaz de medir temperatura, pressão atmosférica e humidade relativa do ar. Este sensor está conectado a um Raspberry Pi Pico W, que atua como a unidade central de processamento e comunicação.

Além da infraestrutura física e de comunicação, foi desenvolvido um site com um sistema de autenticação para acesso seguro. Os utilizadores autenticados podem visualizar os dados recolhidos pelo sensor em tempo real através de uma interface web amigável. Este site não só apresenta os valores atuais de temperatura, pressão e humidade, mas também oferece uma API que permite a integração e o acesso programático a esses dados.

Este projeto demonstra a integração de hardware e software num sistema IoT completo, desde a recolha de dados com sensores até à apresentação e análise de informações através de uma plataforma web segura e eficiente.

Conexão do Raspberry Pi Pico W à Internet.

Para configurar a conexão do Raspberry Pi Pico W à internet via Wi-Fi, foi utilizada a biblioteca Wifi Manager (WifiMgr). Esta biblioteca simplifica o processo de conexão, permitindo que o dispositivo se conecte a uma rede Wi-Fi escolhida pelo usuário através de uma interface gráfica intuitiva.

Funcionamento do Wifi Manager

O Wifi Manager foi projetado para facilitar a conexão de dispositivos IoT a redes Wi-Fi. Quando o Raspberry Pi Pico W é inicializado, ele verifica se já está configurado para se conectar a uma rede Wi-Fi. Se não estiver, o Wifi Manager cria um ponto de acesso temporário e um servidor web simples. O utilizador pode então conectar-se a este ponto de acesso usando um smartphone, tablet ou computador e aceder à interface gráfica através de um browser.

Na **Figura 1** é possível verificar a interface gráfica que é fornecida quando se tenta conectar o Raspberry a uma nova internet. Nesta interface, o utilizador pode visualizar todas as redes Wi-Fi disponíveis e selecionar a rede desejada. Após inserir a senha da rede, o Wifi Manager tenta conectar o dispositivo à internet. Se a conexão for bem-

sucedida, as credenciais da rede são armazenadas no dispositivo para futuras conexões automáticas.



Figura 1 - Interface Wifimgr

Integração com o Projeto

A utilização do Wifi Manager foi inspirada pelo tutorial encontrado no site [Random Nerd Tutorials](#), que fornece um guia passo a passo sobre como implementar esta biblioteca em dispositivos ESP32 e ESP8266. Embora o tutorial seja específico para esses dispositivos, os princípios e a implementação são aplicáveis ao Raspberry Pi Pico W com pequenas adaptações.

Uma vez que o Raspberry Pi Pico W está conectado à internet via Wi-Fi, ele pode comunicar com o broker MQTT. O Mosquitto foi escolhido como o broker MQTT devido à sua robustez e facilidade de uso. O Mosquitto atua como um intermediário para mensagens entre dispositivos IoT, permitindo que o Raspberry Pi Pico W envie dados do sensor BME280 para outros dispositivos ou aplicações que estejam inscritos nos tópicos MQTT correspondentes.

Instalar o mosquitto e garantir autenticação

Para ser possível utilizar um broker MQTT baseado em mosquitto é necessário instalar o mosquitto e fazer algumas alterações ao ficheiro “*mosquitto.conf*” que se encontra na pasta “*C:\Program Files\mosquitto*”.

Na **Figura 2** estão os settings necessários para que seja utilizada autenticação ao fazer a conexão ao broker, com este setting será necessário que as credenciais usadas estejam presentes no ficheiro “passwd”.

```
# acl_file
allow_anonymous true
# allow_zero_length_clientid
# auto_id_prefix
password_file C:\Program Files\mosquitto\passwd
# plugin
```

Figura 2 - Garantir autenticação

Para criar um user é necessário correr o seguinte comando “.mosquitto_passwd.exe -c <caminho ficheiro> <utilizador>”, como visto na **Figura 3**. Neste caso foi criado um utilizador com o nome joao e foi escolhida uma password, este utilizador ficou guardado no ficheiro “passwd”.

```
C:\Program Files\mosquitto>.mosquitto_passwd.exe -c .\passwd joao
Password:
Reenter password:
```

Figura 3 - Criação de um utilizador

Na **Figura 4** pode-se verificar que foi descomentada a linha de listener para garantir que o broker funciona no porto 1883.

```
# listener port-number [ip address/host name/unix socket path]
listener 1883
```

Figura 4 - Garantir que o porto do broker é o 1883

Com estas configurações o mosquitto está pronto para ter um sub ativo utilizando autenticação.

Utilização do ngrok para criar um túnel

Ao fazer a integração do Ngrok no projeto, foi obtido um acesso remoto simplificado ao dispositivo IoT. Isto implica a possibilidade de supervisionar, controlar e depurar os dispositivo a partir de qualquer local, sem necessidade de estar fisicamente presente no seu local de instalação. Esta flexibilidade é crucial para a eficácia do desenvolvimento e teste de sistemas IoT, pois elimina a necessidade de configurações de rede complexas e simplifica o acesso remoto.

Com o ngrok gratuito é possível criar um túnel TCP no porto 1883 que permite então receber mensagens do raspberry pi esteja ele onde estiver.

A instalação do ngrok é simples e bastante rápida. Basta criar uma conta no website deles e fazer o download do ngrok.exe, como visto na **Figura 5**.

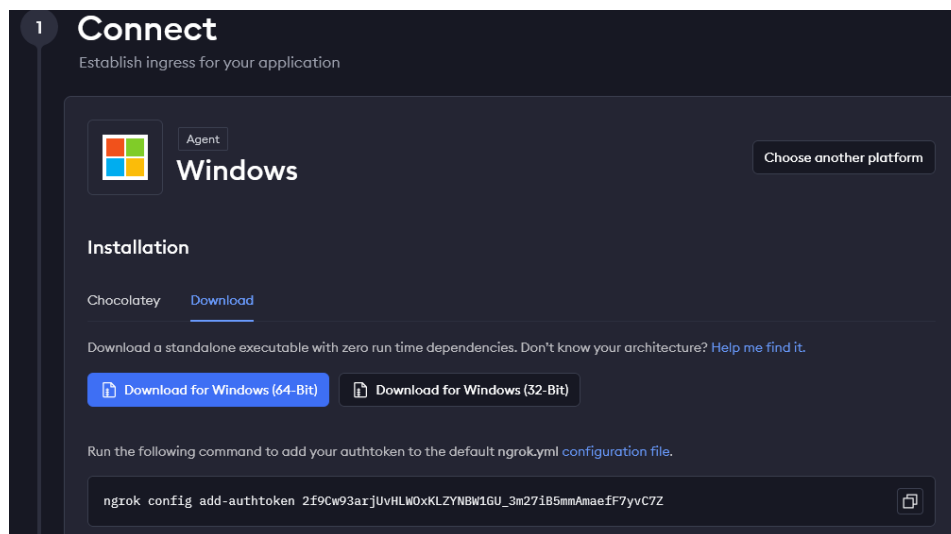


Figura 5 - Instalação do ngrok

Ao executar o ngrok, é aberta uma janela de um terminal onde se pode escrever o seguinte comando “ngrok tcp 1883” que devolverá um address e um porto ao qual será conectado o Raspberry Pi Pico W.

Na **Figura 6** é possível verificar que foi disponibilizado um address que está de momento a apontar para o localhost no porto 1883, porto este que é onde o broker mosquito está em escuta.

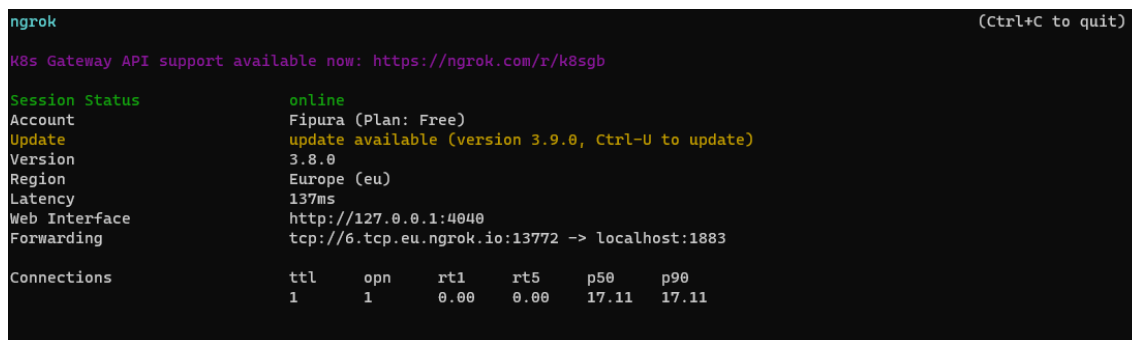


Figura 6 - TCP ngrok

Explicação do código que é executado no Raspberry

Com o mosquitto e o ngrok configurados foi então desenvolvido o código de micropython que permitirá ao Raspberry Pi Pico W comunicar com o mosquitto, utilizando o address que o ngrok disponibiliza e enviar os dados que são lidos pelo sensor BME280.

O código está dividido em partes, na **Figura 7**, são apresentadas as configurações iniciais de variáveis, imports de bibliotecas necessárias, configuração do LED como saída e inicialização do sensor BME280.

```
from machine import Pin, I2C
import wifimgr
import time
import machine
from umqtt.simple import MQTTClient
import utime
import BME280

# Configura o pino LED como saída
led = machine.Pin("LED", machine.Pin.OUT)

# Inicializa a interface I2C com os pinos SCL e SDA especificados
i2c = I2C(id=0, scl=Pin(5), sda=Pin(4), freq=10000)

# Inicializa o sensor BME280 utilizando a interface I2C
bme = BME280.BME280(i2c=i2c)

# Define o intervalo de leitura em segundos
readDelay = 10
```

Figura 7 - Configurações iniciais - Código, Parte 1

Na **Figura 8**, é demonstrada a parte do código que tenta ligar o Raspberry Pi Pico W à internet, a biblioteca wifiMgr possui um método “get_connection()” que tentará conectar-se a uma rede, caso já tenha sido conectado anteriormente ou irá permitir ao utilizador, através de uma interface amigável conectar o dispositivo à sua rede.

```
# Conecta-se à rede WiFi utilizando a biblioteca wifimgr
wlan = wifimgr.get_connection()
if wlan is None:
    # Se a conexão falhar, imprime uma mensagem de erro e entra num ciclo infinito
    print("Failed to initialize connection")
    while True:
        pass

print("WiFi Connected")

# Configurações do servidor MQTT
mqtt_server = '0.tcp.eu.ngrok.io'
client_id = 'teste'
topic_pub = b'test'
led_state = "OFF"
led.low() # Desliga o LED inicialmente
```

Figura 8 - Conectar à internet - Código, Parte 2

Nas **Figuras 9 e 10**, estão presentes as partes do código que irão conectar o Raspberry Pi Pico W ao MQTT Broker. Na **Figura 9** o mqtt_server será o address disponibilizado pelo ngrok e na **Figura 10** o porto na variável cliente é o porto que o ngrok disponibiliza.

A azul estão sublinhadas funcionalidades que o MQTT disponibiliza juntamente com o Raspberry. Uma das funcionalidades a notar é a `set_last_will` que permite ao responsável pelo MQTT receber uma mensagem, neste caso “Disconnected” caso o Raspberry Pi Pico W deixe de estar conectado e o `keepalive` que acaba a conexão com o dispositivo caso não responda.

```
# Configurações do servidor MQTT
mqtt_server = '0.tcp.eu.ngrok.io'
client_id = 'teste'
topic_pub = b'test'
led_state = "OFF"
led.low() # Desliga o LED inicialmente
```

Figura 9 - Variáveis para o MQTT - Código, Parte 3

```
# Função para conectar ao servidor MQTT
def mqtt_connect():
    client = MQTTClient(client_id, mqtt_server, port=10916, user="Pedro", password="test", keepalive=3600)
    client.set_callback(mqtt_callback) # Define a função de callback para mensagens MQTT
    client.set_last_will("test", "Disconnected") # Define a mensagem de última vontade
    client.connect() # Conecta ao servidor MQTT
    print('Connected to %s MQTT Broker' % (mqtt_server))
    client.subscribe(topic_pub) # Subcreve ao tópico definido
    return client

# Função para tentar reconectar ao servidor MQTT em caso de falha
def reconnect():
    print('Failed to connect to the MQTT Broker. Reconnecting...')
    time.sleep(5)
    machine.reset() # Reinicia a máquina

# Tenta conectar ao servidor MQTT, e se falhar, tenta reconectar
try:
    client = mqtt_connect()
except OSError as e:
    reconnect()

last_send_time = utime.time() # Armazena o tempo atual
```

Figura 10 - Conectar ao MQTT - Código, Parte 4

De notar que na **Figura 10** também foi sublinhada a função “*set_callback()*” esta função chama uma outra função no código, **Figura 11**.

```
# Função de callback para processar mensagens recebidas pelo MQTT
def mqtt_callback(topic, msg):
    global led_state
    print("Received message: %s" % msg)
    if msg == b"ON":
        led.high() # Liga o LED
        led_state = "ON"
    elif msg == b"OFF":
        led.low() # Desliga o LED
        led_state = "OFF"
```

Figura 11 - Função *mqtt_callback* - Código, Parte 5

O Raspberry Pi Pico W vai ficar à escuta de mensagens recebidas, ao receber uma mensagem vai verificar se é ON ou OFF e se for uma das duas liga ou desliga o LED da board do Raspberry Pi Pico W.

```
last_send_time = utime.time() # Armazena o tempo atual

# Ciclo principal
while True:
    client.check_msg() # Verifica mensagens recebidas pelo MQTT
    current_time = utime.time() # Obtém o tempo atual

    # Se o tempo decorrido for maior ou igual ao intervalo de leitura
    if current_time - last_send_time >= readDelay:
        tempC = bme.temperature # Lê a temperatura do sensor
        hum = bme.humidity # Lê a humidade do sensor
        pres = bme.pressure # Lê a pressão do sensor
        # Formata a mensagem a ser enviada com os valores lidos e o estado do LED
        topic_msg = "{}/{}/{}/{}".format(tempC, hum, pres, led_state)
        print(topic_msg) # Imprime a mensagem no terminal
        client.publish(topic_pub, topic_msg, retain=True) # Publica a mensagem no tópico MQTT
        last_send_time = current_time # Atualiza o tempo da última leitura
```

Figura 12 - Loop principal - Código, Parte 6

Neste loop principal o Raspberry Pi Pico W fica à escuta de mensagens (para ligar ou desligar o led) e vai contando o tempo, ao passarem 10 segundos, os dados lidos pelo sensor BME são enviados para o tópico ao qual o Raspberry Pi Pico W está subscrito divididos por uma “/”. Na função publish do cliente existe um “*retain=True*” (**Figura 13**), para garantir que as mensagens publicadas são retained no MQTT. Isto é essencial pois permite a disponibilidade imediata dos dados, a redução de latência, persistência do estado do sistema e a resiliência a falhas de conexão.

Esta prática assegura que todos os dispositivos e aplicações conectados ao broker MQTT tenham acesso contínuo e atualizado às informações críticas do sensor BME280 e outros componentes do sistema.


```
while True:
    client.publish(topic_pub, topic_msg, retain=True)
    time.sleep(3)
```

Figura 13 - retain na função publish

Automação da criação de um sub

Para a criação do sub no mosquitto, visto ser sempre diferente o address e o porto que o ngrok fornece, foi criado um script que pede ao utilizador o address e o porto do ngrok, **Figura 14**.

```
@echo off
set /p address="Enter MQTT broker address: "
set /p port="Enter port: "
cd /d "C:\Program Files\mosquitto"
mosquitto_sub -h %address% -p %port% -t "test" -u "Pedro" -P "test"
pause
```

Figura 14 - Script que permite a criação de um sub

De momento o script está preparado para utilizar o tópico test, o utilizador Pedro e a password test na criação do sub.

Na **Figura 15** pode ser verificado o script a pedir o address do broker (ngrok address) e o porto, e após a inserção dos mesmos começam a chegar os valores que o Raspberry Pi Pico W envia, valores separados pelo “/”.

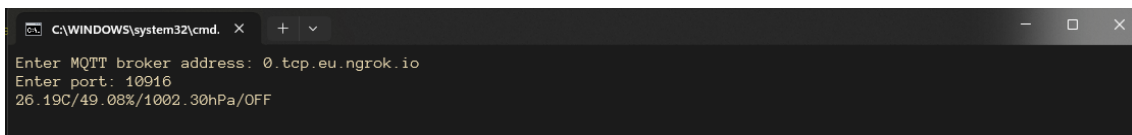


Figura 15 - Execução do script, criação do sub

De notar que pode ser necessário correr este script em modo de administrador.

Para demonstração da last will, foi interrompida a execução do código do Raspberry Pi Pico W, na **Figura 16** é possível verificar a mensagem de last will a ser enviada para o MQTT Broker.

```
C:\WINDOWS\system32\cmd. X + v
Enter MQTT broker address: 0.tcp.eu.ngrok.io
Enter port: 10916
26.27C/49.70%/1002.22hPa/OFF
26.86C/49.36%/1001.23hPa/OFF
Disconnected
```

Figura 16 - Mensagem de Last Will

Configuração do sensor BME280

A configuração do BME280 foi simples, pois a biblioteca Adafruit BME280 permite fazer “plug and play”. Esta biblioteca foi desenvolvida por uma das companhias que produz os sensores.

Na **Figura 17** está apresentado o diagrama de conexão do sensor BME280 ao Raspberry Pi Pico W

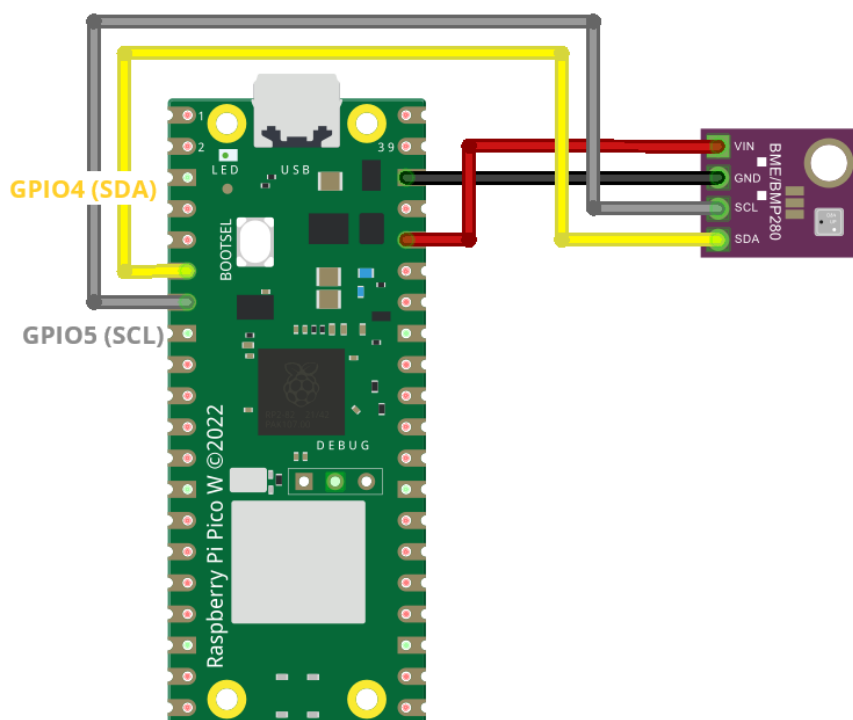


Figura 17 - Conexão do sensor BME280 ao Raspberry Pi Pico W

Site para a visualização dos valores

Com o Raspberry Pi Pico W a comunicar com o MQTT Broker faltava uma forma prática e amigável de verificar os dados que eram enviados. Para isso foi desenvolvido um site que permite aos utilizadores autenticarem-se, visualizar os dados coletados pelo sensor BME280 e interagir com o sistema de forma segura e intuitiva. Abaixo, é detalhado o processo de criação do site e os recursos necessários para o mesmo.

A estrutura, estilo e interatividade do site foram desenvolvidas com HTML, CSS e JavaScript. O backend do site foi implementado com Node.js, permitindo a execução de JavaScript no servidor. Foram utilizadas as seguintes frameworks, Express.js para Node.js, que simplifica a criação de servidores web e APIs. A autenticação foi implementada com Passport.js, um middleware que permite o login utilizando Google OAuth 2.0. Para a comunicação em tempo real entre o servidor e o cliente, foi utilizada a biblioteca Socket.io. Além disso, para conectar e interagir com o broker MQTT, foi utilizada a biblioteca MQTT.js. Estas tecnologias integradas permitiram a criação de um site funcional e interativo, capaz de comunicar eficazmente com o sistema IoT descrito previamente.

Autenticação

Um dos objetivos deste site era garantir a autenticação no mesmo utilizando o OAuth 2.0.

Para o OAuth 2.0 foi utilizada a Google como o Provider do serviço, ou seja, ao tentar fazer uma conexão ao site, o utilizador será redirecionado para a google e deverá utilizar a sua conta para realizar a autenticação. Para isto foi utilizada a biblioteca “*Passport.js*”.

Na **Figura 18** está representado o ficheiro que gere a autenticação. Tudo o que está a ser feito aqui é a utilização de credenciais geradas pela google de modo que o utilizador possa ser autenticado através da mesma. E ao fazer a autenticação, será redirecionado para um url no site “*/auth/google/callback*” que por sua vez redirecionará o utilizador para a dashboard.

```
const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth20').Strategy;

const GOOGLE_CLIENT_ID = '566786276014-drcgkj209ag0q5f9dqqtdggvv2s8cvsc.apps.googleusercontent.com';
const GOOGLE_CLIENT_SECRET = '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';

passport.use(
  new GoogleStrategy(
    {
      clientId: GOOGLE_CLIENT_ID,
      clientSecret: GOOGLE_CLIENT_SECRET,
      callbackURL: 'http://localhost:3000/auth/google/callback',
    },
    (request, accessToken, refreshToken, profile, done) => {
      return done(null, profile);
    }
  )
);

passport.serializeUser(function(user, done) {
  done(null, user);
});

passport.deserializeUser(function(obj, done) {
  done(null, obj);
});

module.exports = passport;
```

Figura 18 - Auth.js, ficheiro que gere a autenticação

O que o utilizador vê é uma página principal com um botão que o redirecionará para a google de modo que se possa autenticar, **Figura 19**.

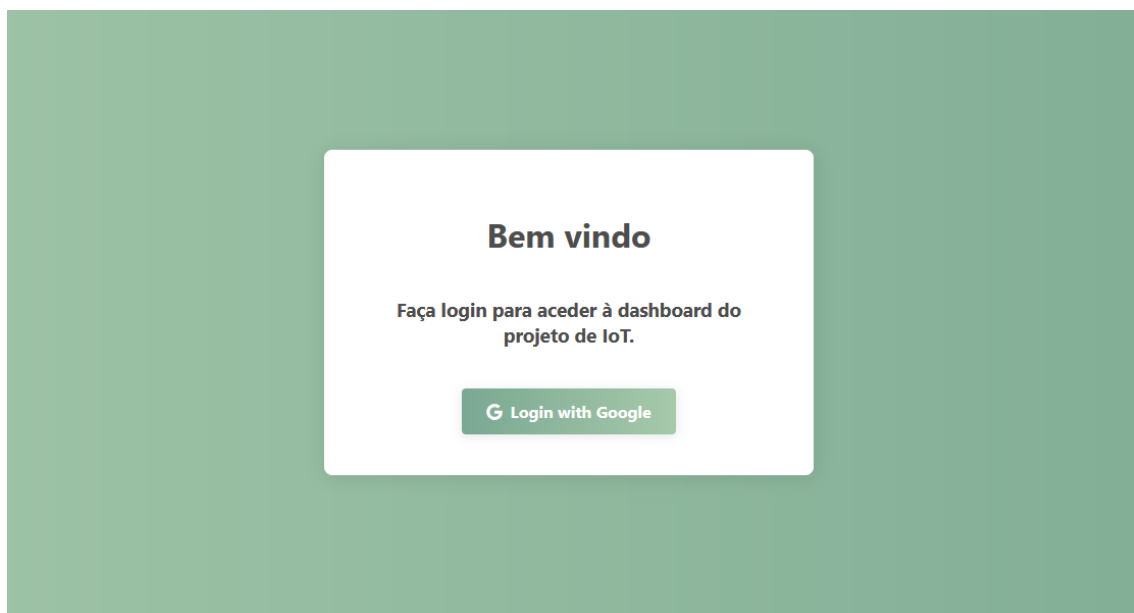


Figura 19 - Página principal

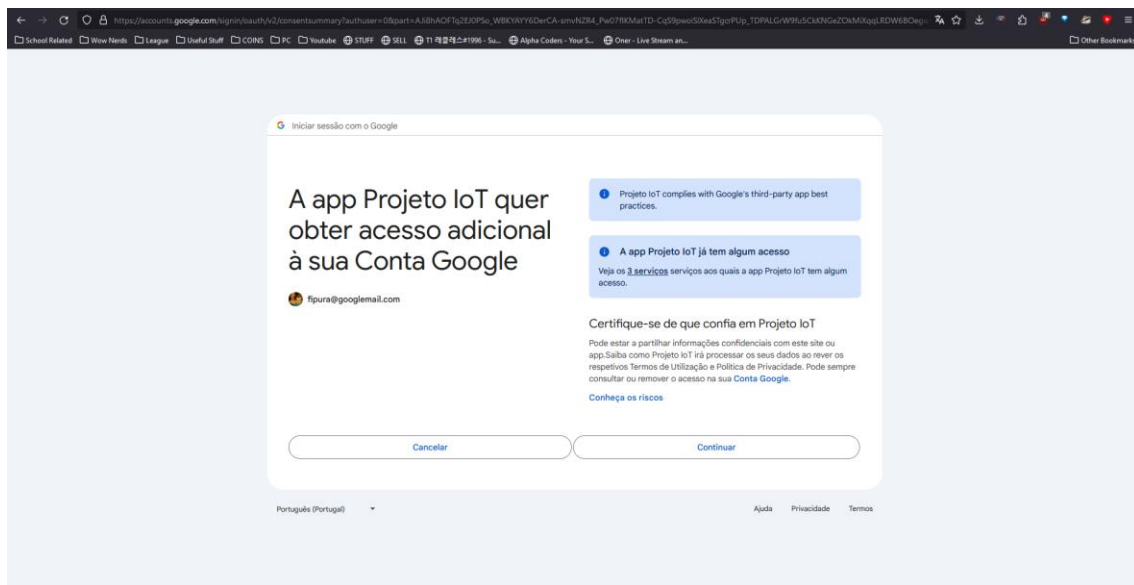


Figura 20 - Autenticação com a google

Conexão com o MQTT Broker - Site

Para que seja possível a apresentação dos dados no dashboard do site é necessário que haja uma conexão ao MQTT Broker. Para isto foi utilizada a biblioteca “*MQTT.js*”, ao fornecer o porto, o address, o tópico e as credenciais necessárias para estabelecer uma conexão com o Mosquitto o servidor fica com acesso aos dados que o BME280 IÊ, nas **Figuras 21** e **22** é apresentada a função que cria a conexão com o MQTT Broker.

```
function connectToMqttBroker(mqttBroker, mqttPort) {
  return new Promise((resolve, reject) => {
    mqttClient = mqtt.connect(`mqtt://${mqttBroker}`, {
      port: mqttPort,
      username: MQTT_USERNAME,
      password: MQTT_PASSWORD,
      reconnect: false
    });

    mqttClient.on("connect", () => {
      console.log("Connected to MQTT broker");
      mqttClient.subscribe("test", (err) => {
        if (err) {
          console.error("Error subscribing to MQTT topic:", err);
          reject(err);
        } else {
          isConnectedToMqtt = true;
          resolve();
        }
      });
    });
  });
}
```

Figura 21 - Conexão ao MQTT Broker, Parte 1

```
mqttClient.on("error", (err) => {
  console.error("MQTT connection error:", err);
  reject(err);
  mqttClient.end();
});

mqttClient.on("message", function (topic, message) {
  console.log("Received message:", message.toString());
  const parts = message.toString().split("/");
  const temperature = parseFloat(parts[0]);
  const humidity = parseFloat(parts[1]);
  const pressure = parseFloat(parts[2]);
  const ledState = parts[3];

  if (!isNaN(temperature) && !isNaN(humidity) && !isNaN(pressure)) {
    lastValueTemp = temperature; // Update lastValue to temperature
    lastValueHum = humidity; // Update lastValue to humidity
    lastValuePress = pressure; // Update lastValue to pressure
    data.push([humidity, temperature, pressure]); // Push an array with humidity, temperature, and pressure
    console.log(
      "Temperature:",
      temperature,
      "Humidity:",
      humidity,
      "Pressure:",
      pressure
    );
    io.emit("data", data);
    io.emit("mqttData", { temperature, humidity, pressure, ledState });
  }
});
});
```

Figura 22 - Conexão ao MQTT Broker, Parte 2

Na **Figura 22** é configurado o que deve acontecer ao ser recebida uma mensagem do Raspberry Pi Pico W, os dados vêm separados por "/" como dito previamente, isto é útil porque ao receber uma mensagem é feito um split() da mesma através de "/" o que dividirá a mensagem em temperatura, humidade, pressão e o estado do led. Com estes valores são populadas as variáveis lastValueTemp, lastValueHum e lastValuePress que serão utilizadas na API. É também enviado para o dashboard através do io.emit() os dados recebidos.

Na **Figura 23** é demonstrada a página para a qual o utilizador será redirecionado após a sua autenticação com a Google. Devido a estar a ser utilizada a versão gratuita do ngrok o address e o porto irão estar sempre a ser alterados, então, para que o utilizador tenha uma interface amigável foi criada esta página para que seja fácil fazer a conexão com o MQTT Broker e utilizar então a função mostrada neste tópico.

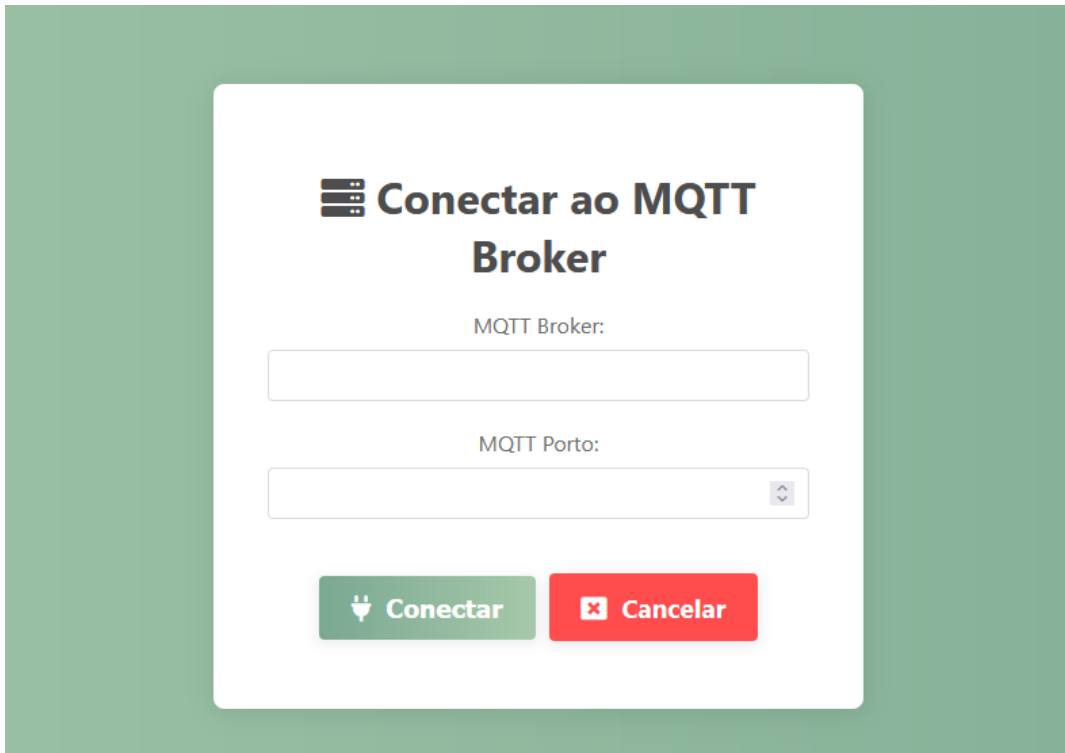
A screenshot of a web form titled "Conectar ao MQTT Broker". The form has a white background and is centered on a green gradient background. It contains two input fields: "MQTT Broker:" and "MQTT Porto:". Below the inputs are two buttons: a green "Conectar" button with a plug icon and a red "Cancelar" button with a close icon.

Figura 23 - Página que pede dados do Broker

Comunicação em tempo real com a dashboard

Na **Figura 22** foi demonstrado que o `io.emit` envia dados para o dashboard. Isto é possível devido à `socket.io`. `Socket.io` permite uma comunicação bidirecional para qualquer plataforma. Neste caso estão a ser criadas `WebSockets` para a comunicação entre o servidor (`server.js`) e a dashboard (cliente).

Na **Figura 24** é apresentado o método que envia do servidor para o cliente todos os dados recebidos do Raspberry Pi Pico W.

```
io.on("connection", (socket) => {  
  console.log("A user connected");  
  
  sendWeatherData();  
  socket.emit("data", data);  
  socket.emit("mqttData", { moistureValue: lastValuehum });  
  socket.emit("mqttData", { temperatureValue: lastValuetemp });  
  socket.emit("mqttData", { pressureValue: lastValuepress });  
  
  socket.on("disconnect", () => {  
    console.log("User disconnected");  
  });  
});
```

Figura 24 - Transmissão dos dados para o cliente, utilizando `socket.io`

Na **Figura 25** é possível verificar a dashboard à qual o utilizador tem acesso após se autenticar e inserir os dados corretos relativamente ao MQTT Broker.

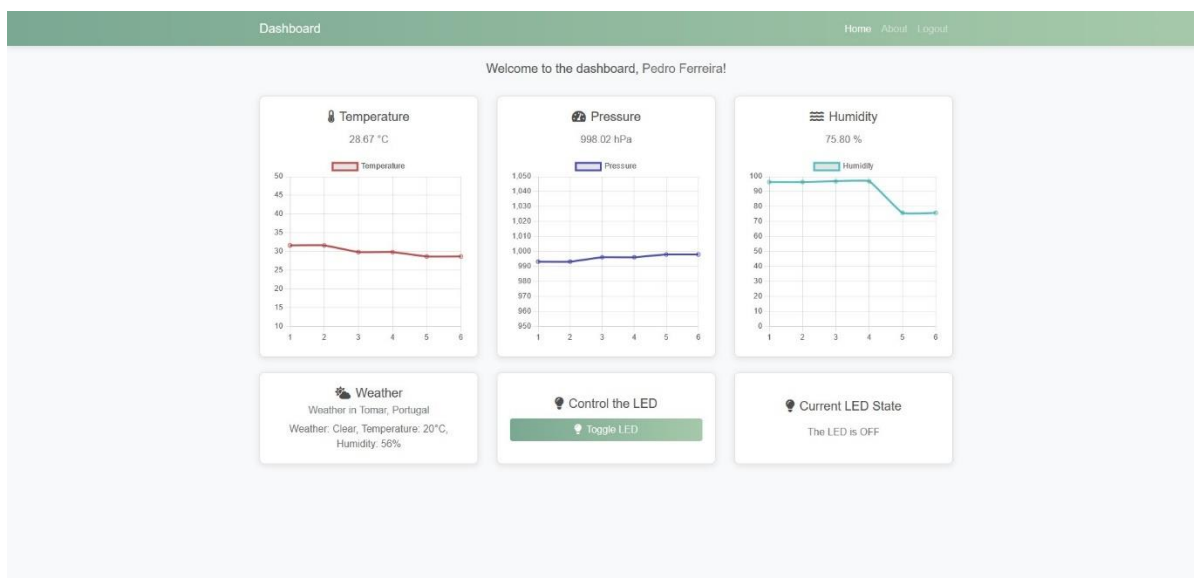


Figura 25 - Dashboard do utilizador

Nesta dashboard, o utilizador tem acesso a três gráficos que mostra a temperatura, pressão e humidade consoante os valores chegam, tem também acesso ao tempo em tomar que está a ser apresentado com a utilização de uma API aberta de temperatura e consegue interagir com o LED presente na board do Raspberry Pi Pico W, vendo se está ligado ou desligado através da dashboard. Graças ao Passport.js e ao OAuth 2.0 é possível dar as boas-vindas ao utilizador pelo seu nome.

Rotas do servidor

Por fim o servidor tem rotas criadas, nas **Figura 26** estão presentes as rotas que servem de autenticação.

```
app.get("/auth/google", (req, res) => {
  passport.authenticate("google", { scope: ["email", "profile"] })(req, res);
});

app.get("/auth/google/callback", (req, res) => {
  passport.authenticate("google", {
    successRedirect: "/connectmqtt.html",
    failureRedirect: "/auth/failure",
  })(req, res, () => {
    res.redirect("/dashboard.html");
  });
});
```

Figura 26 - Rotas de autenticação

A rota `"/auth/google"` permite à biblioteca `"Passport.js"` pedir a autenticação do utilizador à google. A Rota `"/auth/google/callback"` verifica se a resposta da google foi bem-sucedida e reencaminha o utilizador para a página de conexão ao MQTT Broker em caso de sucesso, ou para a página de erro em caso de falha.

```
app.get("/logout", function (req, res, next) {
  req.logout(function (err) {
    if (err) {
      return next(err);
    }
    res.redirect("/");
    req.session.destroy();
    if(mqttClient){
      console.log("Closing MQTT connection");
      mqttClient.end();
      mqttClient = null;
    }
  });
});
```

Figura 27 - Rota de logout

Na **Figura 27** está presente a rota de logout que redirecionará o utilizador novamente para a página de login, destruindo a sua sessão criada com a google e a conexão feita com o MQTT Broker.

Nas **Figuras 28 e 29** estão as rotas para as páginas principais do website, tais como o index, a página de erro, a dashboard, a página de conexão ao MQTT Broker e a página de about.

```
app.get("/auth/failure", (req, res) => {
  res.sendFile(__dirname + "/public/failure.html");
});

// Place the protected routes after the middleware setup
app.get("/dashboard.html", ensureAuthenticated, ensureConnection, (req, res) => {
  console.log("DASHBOARD");
  res.sendFile(__dirname + "/public/dashboard.html");
});

app.get("/connectmqtt.html", ensureAuthenticated, (req, res) => {
  res.sendFile(__dirname + "/public/connectmqtt.html");
});

app.get("/about.html", ensureAuthenticated, ensureConnection, (req, res) => {
  res.sendFile(__dirname + "/public/about.html");
});
```

Figura 28 - Rotas para páginas HTML

```
app.get("/", (req, res) => {  
  res.sendFile(__dirname + "/public/index.html");  
});
```

Figura 29 - Rota para o index

Na **Figura 28** é possível verificar que algumas das rotas tem o uso de duas funções, “*ensureAuthenticated*” e “*ensureConnection*”. Estas duas funções são utilizadas para a autorização e segurança no website, verificam se existe uma conexão com o MQTT Broker e se há uma autenticação feita com a Google, **Figura 30**.

```
function ensureAuthenticated(req, res, next) {  
  if (req.isAuthenticated()) {  
    return next();  
  }  
  console.log('User not authenticated, redirecting to login page');  
  /*res.redirect('/');*/  
  return res.sendFile(__dirname + "/public/unauthorized.html");  
}  
  
function ensureConnection(req, res, next) {  
  if (isConnectedToMqtt) {  
    return next();  
  }  
  return res.sendFile(__dirname + "/public/connectmqtt.html");  
}
```

Figura 30 - Funções de segurança e autorização

Como foi falado da página About, a mesma é apenas uma página HTML que explica o âmbito do site e dá ao utilizador uma forma de conhecer melhor os estudantes que realizaram o trabalho, **Figura 31**.

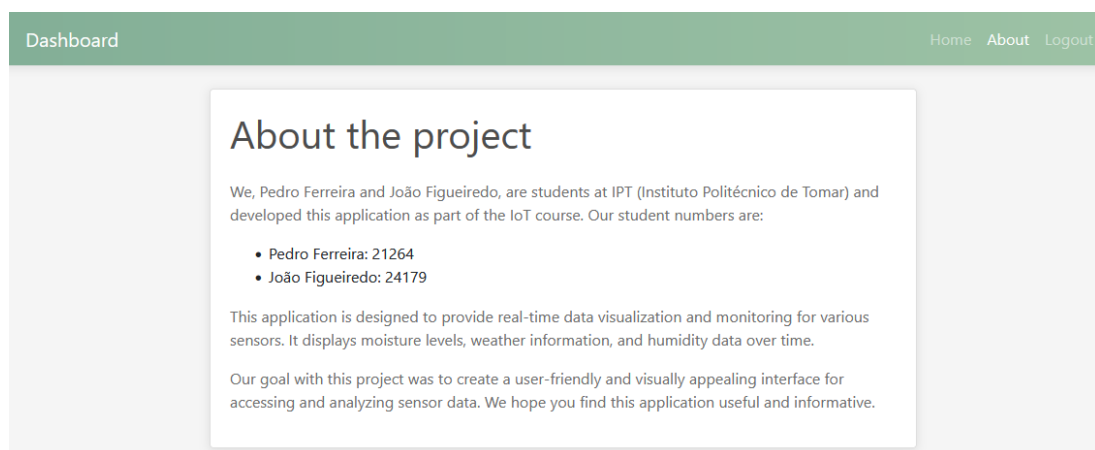


Figura 31 - Página About

Por fim, existem as rotas da API criada. Um dos objetivos deste projeto final de IoT era a criação de uma API que permitisse o uso dos dados lidos pelo sensor a qualquer outra pessoa, para isso foram criados 5 endpoints, 3 deles servem para receber o último valor recebido do sensor BME280, daí a população as variáveis `lastValuetemp`, `lastValuehum` e `lastValuepress` como visto anteriormente na **Figura 22**. Os outros 2 endpoints servem para permitir a interação do utilizador com o LED do Raspberry Pi Pico W, porém também pode servir para uma outra pessoa verificar o estado do led ou até mudar o estado do mesmo, assumindo que tem uma conexão feita.

```
app.get("/api/humidity", (req, res) => {
  if (req.isAuthenticated()) {
    res.json({ humidity: lastValuehum });
  } else {
    return res.sendFile(__dirname + "/public/unauthorized.html");
  }
});

app.get("/api/temperature", (req, res) => {
  if (req.isAuthenticated()) {
    res.json({ temperature: lastValuetemp });
  } else {
    return res.sendFile(__dirname + "/public/unauthorized.html");
  }
});

app.get("/api/pressure", (req, res) => {
  if (req.isAuthenticated()) {
    res.json({ pressure: lastValuepress });
  } else {
    return res.sendFile(__dirname + "/public/unauthorized.html");
  }
});

app.get("/api/ledState", (req, res) => {
  console.log("LEDSTATE");
  if (req.isAuthenticated()) {
    res.json({ ledState });
  } else {
    return res.sendFile(__dirname + "/public/unauthorized.html");
  }
});

app.post("/api/led", (req, res) => {
  if (req.isAuthenticated()) {
    ledState = ledState === "OFF" ? "ON" : "OFF";
    mqttClient.publish("test", ledState);
    res.json({ ledState });
  } else {
    return res.sendFile(__dirname + "/public/unauthorized.html");
  }
});
```

Figura 32 - Endpoints para a API

Desafios e problemas encontrados.

Durante a realização deste projeto, enfrentámos vários desafios e problemas que exigiram soluções criativas e adaptações.

Um dos principais problemas foi a utilização da versão gratuita do ngrok. Esta versão gera endereços e portas distintos a cada sessão, o que dificultava a comunicação consistente com o broker MQTT. Para resolver esta questão, criámos a página “connectmqtt.html”, onde o utilizador pode inserir manualmente o endereço e a porta corretos fornecidos pelo ngrok.

Outro desafio foi o uso de certificados no Raspberry Pi Pico W para garantir uma comunicação segura. Infelizmente, tal como observado durante os laboratórios, não conseguimos encontrar uma solução viável para implementar certificados no dispositivo dentro do prazo do projeto.

Inicialmente, planeávamos permitir o acesso à dashboard a partir de qualquer rede, garantindo uma maior flexibilidade e acessibilidade para os utilizadores. No entanto, a implementação do OAuth 2.0 complicou este plano. A Google exige um endereço específico para redirecionar o utilizador após a autenticação, e como o ngrok gera endereços diferentes em cada sessão é impossível saber para qual endereço deverá ser redirecionado, o utilizador seria, então, redirecionado para o localhost, onde não teria acesso ao servidor a correr no seu próprio computador. Por isso, decidimos manter a implementação do OAuth 2.0, mas evitar expor a dashboard publicamente.

Estas decisões e adaptações foram essenciais para garantir a funcionalidade e segurança do sistema, apesar das limitações técnicas encontradas.