# Relatório de Entrega Técnica

## Documento de Serviços

**Faculdade:** CESAR School
**Nome do(a) Aluno(a):** André Carvalho - avgcf@cesar.school
Arthur Lins - alg2@cesar.school
Pedro Oliveira - popg@cesar.school
**Título do Projeto/Demanda:** Detector de Luminosidade com ESP8266 e Raspberry Pi
**Data:** 03/06/2025

# Sumário

# Introdução

A Internet das Coisas (IoT) tem se tornado cada vez mais presente no nosso dia a dia, conectando objetos físicos à internet para coletar e trocar informações de forma automática. Essa tecnologia permite que sensores e dispositivos se comuniquem entre si e com sistemas maiores, gerando soluções eficientes para problemas do cotidiano, como o controle de luz, temperatura, segurança, entre outros.

No contexto atual, em que a automação e o uso inteligente da energia são temas cada vez mais relevantes, perceber a necessidade de uma iluminação mais eficiente em ambientes foi o ponto de partida para este projeto. Em muitos lugares, como salas, quartos ou corredores, a luz fica acesa mesmo sem necessidade, ou então é preciso que alguém atue manualmente para ligá-la. Isso gera desperdício de energia e desconforto. A motivação do grupo, portanto, foi pensar em uma solução prática que automatizasse esse processo de forma simples e acessível.

Dessa forma, o projeto tem como objetivo montar um sistema simples de Internet das Coisas (IoT), usando um sensor de luz (LDR), um microcontrolador ESP8266 e uma Raspberry Pi. A ideia principal é detectar o nível de luminosidade do ambiente e, quando estiver abaixo de um certo limite, acender automaticamente um LED. Paralelamente, os dados do sensor são enviados via Wi-Fi para a Raspberry Pi, usando o protocolo MQTT.

Além disso, para permitir o acompanhamento em tempo real do comportamento da luz no ambiente, o grupo desenvolveu um painel gráfico (dashboard) que exibe essas informações de forma visual. Esse painel foi implementado em Python, utilizando a biblioteca tkinter para a interface e seaborn para gerar os gráficos a partir das leituras recebidas.

O projeto envolve os seguintes elementos principais:

- Comunicação via Wi-Fi entre o ESP8266 e a Raspberry Pi
- Uso do protocolo MQTT para envio e recepção de dados
- Sensor LDR que transmite leituras de luminosidade em tempo real
- Acionamento automático do LED em ambientes escuros
- Visualização gráfica dos dados em um dashboard na Raspberry Pi

Essa experiência permitiu ao grupo aplicar conceitos de sensores, redes sem fio, protocolos de comunicação e programação em um sistema integrado e funcional

## Restrições técnicas e orçamentárias

- Uso obrigatório do ESP8266 (ou ESP32) e da Raspberry Pi
- Comunicação entre os dispositivos feita via Wi-Fi
- Envio dos dados realizado com protocolo MQTT
- Código da ESP desenvolvido na Arduino IDE
- Dashboard implementado e executado diretamente na Raspberry Pi em ambiente local

# Metodologia

**Diagrama do Sistema**

O sistema foi dividido nos seguintes blocos funcionais:

Sensor LDR → ESP8266 → (Wi-Fi + MQTT) → Raspberry Pi → Dashboard (tkinter + seaborn)

**Lista de Hardware**

- ESP8266 NodeMCU
- Raspberry Pi 3 B+
- Sensor LDR
- LED
- Resistor 220 ohms
- Protoboard
- Jumpers (fios de conexão)

**Lista de Software**

- Arduino IDE
- Python 3

- Bibliotecas: paho-mqtt, pandas, seaborn, tkinter
- MQTT Broker (Mosquitto)

**Fluxo de Comunicação**

O ESP8266 é o dispositivo responsável por coletar os dados do sensor LDR, interpretar esses dados e agir localmente (acendendo o LED) se a luminosidade estiver baixa. Ao mesmo tempo, ele envia os dados via Wi-Fi para um broker MQTT instalado na Raspberry Pi. A Pi atua como central receptora, processando os dados e exibindo-os por meio de um painel gráfico desenvolvido em Python.

Esse fluxo garante autonomia ao microcontrolador e centralização das informações na Raspberry Pi, permitindo o monitoramento remoto e a expansão futura do sistema.

## Problemas atuais do contexto

Atualmente, em locais com variação de luz natural (como salas com janelas), é difícil controlar automaticamente a iluminação. Ou se deixa uma luz acesa o tempo todo, ou depende de alguém para ligar/desligar.

Com esse projeto, conseguimos automatizar esse processo de forma simples e barata. Além disso, a visualização em tempo real ajuda a entender como a luz varia ao longo do tempo, o que pode ser útil em várias situações (monitoramento, economia de energia, automação residencial etc.).

Durante o desenvolvimento, enfrentamos um problema com **mal contato** tanto na ligação do **sensor LDR com a protoboard** quanto na conexão dos **fios com a ESP8266**. Por conta disso, inicialmente os dados de luminosidade não estavam sendo lidos corretamente, o que impossibilitou o funcionamento do sistema. Após identificar os pontos de falha, refizemos as conexões com mais firmeza, o que resolveu o problema e estabilizou a leitura do sensor.

# Proposta de solução preliminar

Para resolver o desafio proposto pela disciplina, desenvolvemos um sistema de monitoramento de luminosidade baseado em Internet das Coisas (IoT), utilizando como base a comunicação entre um microcontrolador (ESP8266) e uma central de controle (Raspberry Pi 3 B+), conectadas via Wi-Fi e utilizando o protocolo MQTT.

Nosso objetivo era criar um sistema que conseguisse **medir a intensidade da luz ambiente**, **enviar esses dados para um servidor** e **tomar decisões automáticas** com base nessas informações, como acender um LED em locais escuros. Além disso, queríamos que os dados estivessem acessíveis em tempo real por meio de um **painel visual (dashboard) em Python**,

para que qualquer pessoa pudesse acompanhar a variação de luminosidade com gráficos claros.

### Montagem do circuito

A montagem do circuito foi feita em uma protoboard, onde conectamos o sensor LDR ao pino A0 da ESP8266, responsável por ler sinais analógicos. Em paralelo, ligamos um LED ao pino D1, passando por um resistor de 220 ohms, que serve para proteger o LED contra excesso de corrente.

Durante a fase de testes, encontramos dificuldades com **mal contato nas conexões**, especialmente entre o sensor e a protoboard, além da ligação com o próprio ESP8266. Isso impedia que o sensor de luminosidade funcionasse corretamente e, consequentemente, que os dados fossem enviados. O problema foi resolvido ao reforçar as conexões, trocando os fios e ajustando melhor os pinos.

## Funcionamento do sistema

### Conexão à rede

Assim que é ligado, o ESP8266 executa a função setup_wifi() para se conectar à rede Wi-Fi local, utilizando o nome da rede (SSID) e a senha definidos no código. Ele tenta a conexão de forma contínua até obter sucesso, e imprime o IP atribuído no monitor serial. Essa etapa é essencial para que o dispositivo possa se comunicar com o broker MQTT.

### Conexão ao MQTT Broker

Com a rede conectada, o ESP8266 estabelece comunicação com o broker MQTT que está rodando localmente na Raspberry Pi. Essa conexão é feita na porta 1883, padrão do protocolo MQTT. Caso a conexão seja perdida por qualquer motivo, o código executa automaticamente a função reconnect(), garantindo a reconexão de forma resiliente e contínua.

### Leitura dos dados

A cada 2 segundos, o ESP faz uma leitura do sensor LDR conectado ao pino A0. O valor lido é analógico e varia de 0 (ambiente completamente escuro) até 1023 (ambiente totalmente iluminado). Esse valor representa a intensidade da luz no momento da leitura.

### Envio dos dados

Após a leitura, o valor é preparado para envio. Dois dados são gerados:

- O valor bruto da leitura (sensor/light)
- O valor convertido para uma escala percentual de 0 a 100% (sensor/light_percent)

Esses dados são publicados via MQTT em dois tópicos separados. Isso facilita tanto o uso técnico (com o valor real do sensor) quanto a visualização simplificada (com a porcentagem), especialmente no painel gráfico.

**Decisão automática**

Depois de enviar os dados, o próprio ESP8266 toma uma decisão local:
Se a luminosidade estiver abaixo de um valor-limite definido (threshold = 500), o LED conectado ao pino D1 é aceso automaticamente. Isso é feito com um simples digitalWrite. O estado do LED também é publicado em um terceiro tópico (sensor/led_state), com os valores “ON” ou “OFF”. Isso torna o sistema reativo em tempo real e com retorno visual.

**Dashboard (painel visual)**

Na Raspberry Pi, um script em Python (client.py) está em execução contínua e escutando o tópico sensor/light. Sempre que chega um novo dado:

1. O valor é adicionado a um DataFrame do pandas, com a marcação de horário da leitura.

2. Um gráfico de linha é atualizado utilizando o seaborn, com as **últimas 20 leituras** para manter a visualização limpa e leve.

3. O gráfico e o valor atual da luz são exibidos em uma janela feita com tkinter, sem necessidade de navegador ou aplicação externa.

Essa solução oferece uma visualização simples, direta e funcional da luz no ambiente ao longo do tempo.

Esse sistema simula a lógica de um sistema real de automação de iluminação, como os utilizados em casas inteligentes ou escritórios sustentáveis. A principal diferença é que, aqui, o sistema está conectado a sensores e decide com base em **dados em tempo real**, e não em temporizadores fixos. Isso permite maior autonomia, eficiência e economia de energia.

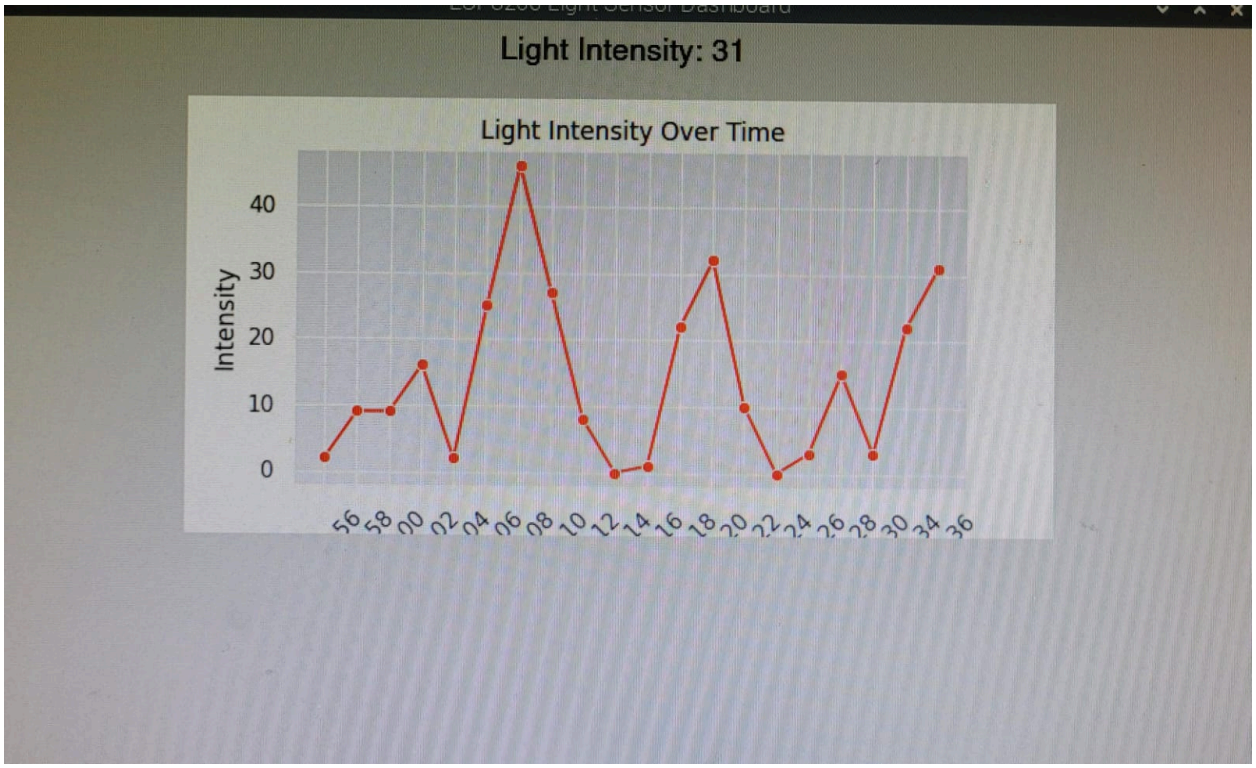
# Resultados

**4.1 Dashboard**

O painel foi desenvolvido com Python e utiliza a biblioteca tkinter para interface e seaborn para plotar um gráfico de linha com as últimas 20 leituras de luminosidade. O painel atualiza automaticamente conforme os dados chegam via MQTT.

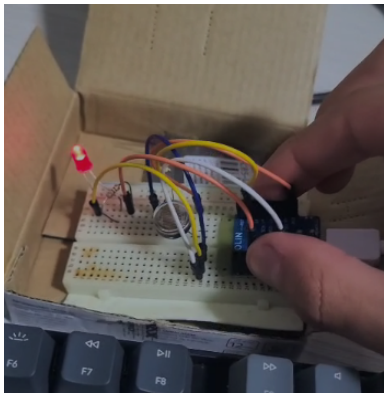
**4.2 Dados Coletados**

Os dados de luminosidade são lidos a cada 2 segundos. Cada valor é armazenado com a hora exata da leitura e adicionado a um DataFrame do pandas. Isso permite observar a variação da luz ambiente ao longo do tempo. O gráfico gerado mostra essas variações em tempo real, facilitando a análise.




### 4.3 Evidências de Funcionamento

Durante os testes, foi possível observar o LED acendendo em ambientes com pouca luz, e os valores registrados sendo atualizados corretamente na interface gráfica. O comportamento do sistema foi consistente após a resolução de um problema inicial de mau contato entre os fios e a protoboard. Um vídeo foi gravado demonstrando o funcionamento completo do sistema, desde a leitura do sensor até a exibição dos dados no dashboard.



**LINK PARA VÍDEO DO PROJETO RODANDO**

# Conclusão

O desenvolvimento deste projeto possibilitou a aplicação prática de diversos conceitos estudados na área de Internet das Coisas (IoT), com foco em sensores físicos, comunicação sem fio e visualização de dados em tempo real. A experiência proporcionou uma compreensão mais concreta de como integrar hardware e software em um sistema funcional e com respostas automáticas baseadas em condições do ambiente.

Durante a montagem e os testes, um dos principais desafios enfrentados foi o mau contato entre os fios da protoboard, especialmente na ligação entre o sensor LDR e a ESP8266. Esse problema fazia com que os valores de luminosidade não fossem captados corretamente, o que impedia o funcionamento do sistema. Após revisar e reforçar todas as conexões, o circuito passou a operar de forma estável, o que também evidenciou a importância da montagem física para o sucesso de projetos com componentes eletrônicos.

Com o sistema funcionando, foi possível observar o LED sendo acionado automaticamente em ambientes escuros e, ao mesmo tempo, os dados sendo enviados para a Raspberry Pi e atualizados em tempo real em um gráfico na interface gráfica. A combinação entre o controle local (LED) e a visualização centralizada (dashboard) mostrou como soluções simples podem ser úteis no dia a dia.

Algumas melhorias que poderiam ser implementadas futuramente incluem:

- Armazenamento histórico das leituras, para análise posterior ou geração de relatórios;
- Envio de alertas automáticos, via e-mail, app ou mensagens, em caso de baixa luminosidade;
- Criação de um painel web acessível remotamente, ampliando a acessibilidade do sistema;
- Adição de novos sensores, como temperatura ou movimento, e outros atuadores, como relés para controlar lâmpadas reais.

Esse projeto foi importante para consolidar o aprendizado técnico da equipe, além de demonstrar como tecnologias simples, quando bem integradas, podem gerar soluções eficientes para problemas do cotidiano.

# Apêndice

## Código-fonte da ESP8266

Abaixo está o código completo usado na ESP8266. Ele foi desenvolvido na Arduino IDE, utilizando as bibliotecas ESP8266WiFi para conexão com a rede e PubSubClient para comunicação MQTT.

O código é responsável por:

- Conectar o microcontrolador à rede Wi-Fi
- Ler o sensor LDR
- Acionar o LED conforme a luminosidade
- Enviar os dados via MQTT

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Warpedro1";
const char* password = "guerra123";
const char* mqtt_server = "192.168.161.217";

// Cliente MQTT
WiFiClient espClient;
PubSubClient client(espClient);

// Pinos
const int ledPin = D1;
const int ldrPin = A0;
int threshold = 500; // valor-limite de luz

void setup_wifi() {
  Serial.begin(9600);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }

  Serial.println("WiFi conectado");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);

    if (client.connect(clientId.c_str())) {
      Serial.println("Conectado ao MQTT Broker");
    } else {
      Serial.print("Tentando novamente em 5s. Código de erro: ");
      Serial.println(client.state());
      delay(5000);
    }
  }
}

void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }

  client.loop();

  int ldrValue = analogRead(ldrPin);
  String payload = String(ldrValue);
  String lightPercent = String(map(ldrValue, 0, 1024, 0, 100));

  Serial.print("Luminosidade: ");
  Serial.print(ldrValue);
  Serial.print(" | ");
  Serial.print(lightPercent);
```

```
  Serial.println("%");

  client.publish("sensor/light", payload.c_str());
  client.publish("sensor/light_percent", lightPercent.c_str());

  if (ldrValue < threshold) {
    digitalWrite(ledPin, HIGH);
    client.publish("sensor/led_state", "ON");
  } else {
    digitalWrite(ledPin, LOW);
    client.publish("sensor/led_state", "OFF");
  }

  delay(2000);
}
```

Esse código foi testado e validado com sucesso. Em conjunto com o dashboard, conseguimos observar a luminosidade do ambiente sendo atualizada em tempo real, enquanto o LED respondia automaticamente aos níveis mais baixos de luz. O vídeo do projeto em funcionamento comprova que os objetivos foram atingidos.

## Client.py

Script Python responsável por:

- Conectar-se ao broker MQTT
- Escutar os tópicos de luminosidade
- Armazenar os dados com timestamp
- Atualizar um gráfico em tempo real
- Exibir os dados em uma janela tkinter

```
import tkinter as tk

from tkinter import ttk

import paho.mqtt.client as mqtt
```

```
import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

from datetime import datetime

import pandas as pd


# Cria uma base de dados inicial vazia

data = pd.DataFrame(columns=["timestamp", "light"])


# Configuração da janela gráfica com tkinter

root = tk.Tk()

root.title("ESP8266 Light Sensor Dashboard")


# Elemento de texto que mostra o valor atual da luz

label_var = tk.StringVar()

label_var.set("Waiting for data...")

label = ttk.Label(root, textvariable=label_var, font=("Helvetica",
16))

label.pack(pady=10)


# Configuração do gráfico

sns.set(style="darkgrid")

fig, ax = plt.subplots(figsize=(6, 3))

canvas = FigureCanvasTkAgg(fig, master=root)

canvas.get_tk_widget().pack(padx=10, pady=10)
```

```
# Função para atualizar o gráfico sempre que chegar um novo dado

def update_plot():

    if not data.empty:

        ax.clear()

            sns.lineplot(data=data, x="timestamp", y="light", ax=ax,
marker="o", color="red")

        ax.set_title("Light Intensity Over Time")

        ax.set_xlabel("Time")

        ax.set_ylabel("Intensity")

        ax.tick_params(axis='x', rotation=45)

        canvas.draw()


# Função chamada quando conecta ao broker MQTT

def on_connect(client, userdata, flags, rc):

    print("Connected to MQTT broker")

      client.subscribe("sensor/light")   # Escuta o tópico de dados
brutos


# Função chamada sempre que uma nova mensagem chega no tópico

def on_message(client, userdata, msg):

    value = msg.payload.decode()

    try:

        light = int(value)

        timestamp = datetime.now().strftime("%H:%M:%S")
```

```
        global data
        # Adiciona a nova leitura à base de dados
                data  =  pd.concat([data,  pd.DataFrame([{"timestamp":
timestamp, "light": light}])], ignore_index=True)

        # Limita a base às últimas 20 leituras
        if len(data) > 20:
            data = data.tail(20)

        # Atualiza o texto e o gráfico
        label_var.set(f"Light Intensity: {light}")
        update_plot()
    except ValueError:
        print(f"Invalid data received: {value}")

# Configura o cliente MQTT
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect("localhost", 1883, 60)  # Broker rodando localmente na
Pi

# Loop do MQTT rodando em paralelo com a interface gráfica
def mqtt_loop():
    client.loop(timeout=1.0)
```

```
    root.after(500, mqtt_loop)  # Verifica por novas mensagens a cada
0.5s



root.after(500, mqtt_loop)

root.mainloop()
```

Esse script é o responsável por exibir o painel gráfico do projeto, recebendo os dados de luminosidade enviados pelo ESP8266. Ele mantém um histórico dos dados e atualiza a interface a cada nova leitura recebida.

## Script de Automação (run_project.sh)

```
#!/bin/bash

cd ~/Documents/bolinho

source ~/Documents/bolinho/bolovenv

~/Documents/bolinho/bolovenv/bin/python3
~/Documents/bolinho/client.py
```

Esse script automatiza a execução do sistema ao iniciar o ambiente virtual Python e rodar o painel.

## Configurações do broker MQTT

- Broker utilizado: Mosquitto
- Hospedado na Raspberry Pi, porta 1883
- Tópicos utilizados:
    - sensor/light
    - sensor/light_percent
    - sensor/led_state