Cesar School - 2025.2

Iumos MQTT – Iluminação Inteligente por Presença com ESP32 e MQTT

Bruno Ribeiro
Ian Nunes
Paulo Portella
Vinicius Petribu

04/12/2025

RESUMO

Este relatório apresenta o desenvolvimento do **lumosMQTT**, um sistema IoT de monitoramento de presença e controle inteligente de iluminação, construído para a disciplina de Sistemas Embarcados da CESAR School. O projeto integra um módulo **ESP32** com **sensor PIR** e **LED controlado por PWM**, comunicação sem fio via **Wi-Fi** e **protocolo MQTT**, um backend em **Flask + SQLite** e um dashboard web em **React** para visualização em tempo real das métricas coletadas.

Cada detecção de movimento registrada pelo sensor PIR gera um evento enviado em formato JSON para um broker MQTT (Mosquitto). O backend consome esses eventos, persiste-os em um banco de dados SQLite e calcula indicadores como total de detecções, atividades no dia, distribuição horária, sessões de presença, tempo ocioso e estimativa de energia economizada. O dashboard lumosMQTT apresenta essas métricas em uma interface única, com cards de resumo, gráficos e tabela de eventos recentes, permitindo acompanhar o comportamento de uso de um ambiente em tempo real.

Os resultados demonstram a capacidade do sistema de monitorar mais de 600 detecções, identificar horários de pico de ocupação e estimar economia de energia acima de 80% em comparação a um cenário de iluminação sempre ligada. O projeto cumpre os requisitos da disciplina ao integrar microcontrolador, sensores, atuadores, comunicação MQTT, servidor de aplicação, persistência de dados e interface gráfica.

1 INTRODUÇÃO

A popularização de dispositivos conectados e sensores de baixo custo tem impulsionado soluções de **Internet das Coisas (IoT)** voltadas à automação de ambientes residenciais e corporativos. Um dos problemas recorrentes nesses cenários é o **desperdício de energia elétrica** causado por iluminação mantida ligada mesmo quando não há pessoas presentes no ambiente. Sistemas de detecção de presença e controle automático de iluminação podem reduzir significativamente esse desperdício, aumentando a eficiência energética e o conforto dos usuários.

Neste contexto, foi desenvolvido o **lumosMQTT**, um sistema IoT que utiliza um **ESP32** equipado com **sensor PIR** para detectar movimento e controlar o brilho de um LED por **PWM**. Sempre que um movimento é detectado, o dispositivo envia um evento via **MQTT** para um backend que armazena as informações em um banco de dados e gera métricas avançadas. Um dashboard web em **React** fornece visualização em tempo real desses dados, permitindo analisar padrões de uso e estimar economia de energia.

O projeto foi concebido para atender ao escopo do “Projeto IoT com ESP32” da disciplina de Sistemas Embarcados, integrando **dispositivo embarcado, comunicação sem fio, protocolo MQTT, servidor de aplicação, banco de dados e dashboard**.

1.1 Objetivo geral

Desenvolver um sistema IoT de **monitoramento de movimento e controle inteligente de iluminação**, utilizando ESP32, MQTT, backend em Flask e dashboard web em React, capaz de registrar eventos de presença, calcular métricas de uso e estimar economia de energia em tempo real.

1.2 Objetivos específicos

- Utilizar o **ESP32** como nó embarcado responsável por leitura de sensor e acionamento de atuador.
- Detectar movimento por meio de um **sensor PIR** conectado ao microcontrolador.
- Controlar o brilho de um **LED via PWM**, alternando entre modo de alto brilho quando há movimento recente e modo de baixo brilho em períodos de ociosidade.
- Enviar eventos de detecção de movimento em formato **JSON** por **MQTT** a um broker Mosquitto.
- Persistir os eventos em um **banco de dados SQLite** e calcular métricas, incluindo:
    - total de detecções;
    - atividades do dia;
    - distribuição por horário;
    - sessões de presença;
    - tempos ociosos;
      estimativa de energia consumida e economizada.

- Exibir todas as métricas em um **dashboard web** em tempo real, com cards de resumo, gráficos e tabela de eventos recentes.
- Organizar o projeto em um **repositório GitHub** com estrutura modular, arquivos de configuração (.env) e suporte a execução via **Docker Compose**.

2 METODOLOGIA

Esta seção descreve a arquitetura do sistema, os componentes de hardware e software utilizados, o fluxo de comunicação e os principais módulos implementados.

2.1 Arquitetura geral do sistema

A arquitetura do lumosMQTT segue o modelo típico de aplicações IoT com três camadas: dispositivo embarcado, infraestrutura de comunicação e camada de aplicação.

Visão geral:

- **Camada embarcada:** ESP32 com sensor PIR (entrada) e LED controlado por PWM (saída).
- **Camada de comunicação:** rede Wi-Fi local e broker **Mosquitto MQTT**.
- **Camada de aplicação:** backend em **Flask** com banco **SQLite** e dashboard em **React**.

O fluxo de dados pode ser resumido da seguinte forma:

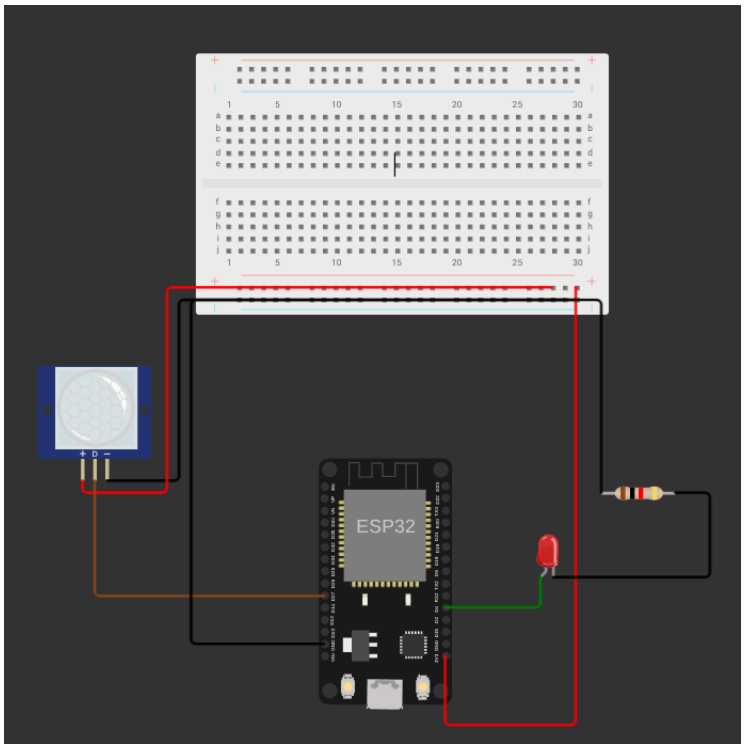
1. O ESP32 lê periodicamente o estado do sensor PIR.
2. Quando ocorre uma borda de subida (transição de LOW para HIGH), o firmware registra o evento e publica uma mensagem JSON no tópico lumosMQTT/motion.
3. O backend Flask está inscrito nesse tópico, recebe as mensagens, grava cada evento na tabela motion_events do SQLite e recalcula métricas.
4. O dashboard React consome periodicamente o endpoint REST /api/metrics e atualiza a interface com as informações mais recentes.

2.2 Hardware utilizado

O sistema utiliza os seguintes componentes de hardware:

- **ESP32 DevKit v1** – microcontrolador principal, responsável pela leitura do sensor e controle do LED.
- **Sensor PIR** (ligado ao GPIO 27) – detecta movimento no ambiente.
- **LED** (ligado ao GPIO 4) – utilizado como atuador de iluminação, com controle de brilho via PWM.
- **Componentes auxiliares** – resistores, jumpers e protoboard para montagem do circuito.

O sensor PIR é configurado com um período de estabilização inicial (≈ 20 segundos) antes de iniciar a detecção, conforme recomendado pelos fabricantes.



2.3 Software e ferramentas

Foram utilizadas as seguintes ferramentas de software:

- **PlatformIO** (C++/Arduino) para desenvolvimento do firmware do ESP32.
- **Mosquitto** como broker **MQTT**, com configuração de listeners para IPv4 (ESP32) e IPv6 (backend).
- **Python 3 + Flask** para o backend HTTP e integração com MQTT.
- **SQLite** como banco de dados local para armazenamento dos eventos de movimento.
- **React + Vite + TypeScript + Tailwind CSS** para o dashboard web.
- **Docker e Docker Compose** para orquestração dos serviços (Mosquitto, backend e frontend).
- **GitHub** para versionamento, organização do código e documentação.

2.4 Organização do repositório

O repositório segue uma estrutura modular:

- README.md e README_EN.md – documentação principal em português e inglês.
- docs/ – inclui o PDF do enunciado do projeto e imagens do dashboard.

- esp32-esp8266/ – firmware do ESP32 (PlatformIO), com:
    - src/main.cpp – código principal;
    - include/env.h – constantes e tópicos MQTT;
    - platformio.example.ini – modelo de configuração sem credenciais.
- backend/ – backend Flask, contendo:
    - app.py – aplicação HTTP e cliente MQTT;
    - database.py – funções de acesso ao SQLite;
    - mosquitto.conf – configuração do broker;
    - requirements.txt, Dockerfile e .env.example.
- frontend/ – dashboard React:
    - src/App.tsx, componentes e páginas;
    - Dockerfile e .env.example.
- docker-compose.yml – orquestração dos serviços.

2.5 Firmware do ESP32

O firmware foi desenvolvido em C++ utilizando a framework Arduino com **FreeRTOS**. Os principais parâmetros de configuração são:

- PIN_PIR = 27 e PIN_LED = 4;
- PWM configurado em 5 kHz, resolução de 8 bits (0–255);
- níveis de brilho:
    - BRIGHT_HIGH = 255 (alto brilho);
    - BRIGHT_LOW = 60 (modo econômico).
- janela de movimento MOTION_WINDOW_MS = 3000 ms, utilizada para manter o LED em alto brilho por alguns segundos após a última detecção.

Uma tarefa FreeRTOS dedicada faz a leitura do sensor PIR a cada ≈ 300 ms, detecta bordas de subida e publica eventos MQTT em TOPIC_MOTION = "lumosMQTT/motion". A função updateLedBrightness() decide se o LED deve permanecer em alto ou baixo brilho com base no tempo decorrido desde a última detecção.
O ESP32 utiliza Wi-Fi de 2,4 GHz para conectar-se ao broker. As credenciais de rede e o endereço do broker são configuradas por meio de um arquivo platformio.ini local, copiado a partir de platformio.example.ini, evitando exposição de dados sensíveis no repositório.

2.6 Backend Flask, MQTT e banco de dados

O backend é responsável por:

- conectar-se ao broker MQTT;
- assinar o tópico de movimento;
- persistir cada evento no banco de dados;
- calcular métricas agregadas;
- expor uma **API REST** para o dashboard.

Funções auxiliares em database.py realizam operações como inserção de eventos, contagem diária, distribuição horária, cálculo de pico de atividade e recuperação de eventos para exportação.

A rota principal /api/metrics consolida as informações do dia atual e retorna um JSON contendo, entre outros:

- totalDetections – total de detecções;
- activitiesToday – detecções no dia corrente;
- detectionsByDay – séries dos últimos sete dias;
- hourlyDistribution – distribuição por hora do dia;
- peakHours – intervalo de maior atividade;
- sessionsToday – número de sessões, duração média e máxima;
- idleMetrics – maior período sem movimento e idade do último evento;
- energyMetrics – tempo em alto e baixo brilho, energia consumida e percentual de economia;
- trends – comparação com o dia anterior e com a média semanal.

2.7 Dashboard web em React

O dashboard lumosMQTT foi implementado como uma SPA (Single Page Application) em React, com design responsivo em tema escuro/claro. As principais seções são:

- **Header:** título do sistema, status de conexão (Online/Offline) com base em /api/health e timestamp da última atualização.

- **Cards de resumo:** exibem total de detecções, atividades de hoje, energia economizada, sessões de hoje, horário de pico e tempo inativo.
- **Gráficos (Recharts):**
  - “Detecções por dia” – série temporal dos últimos sete dias;
  - “Distribuição horária hoje” – barras com contagem de eventos por hora.
- **Tendências:** cards comparando o volume de detecções do dia com o dia anterior e com a média semanal.

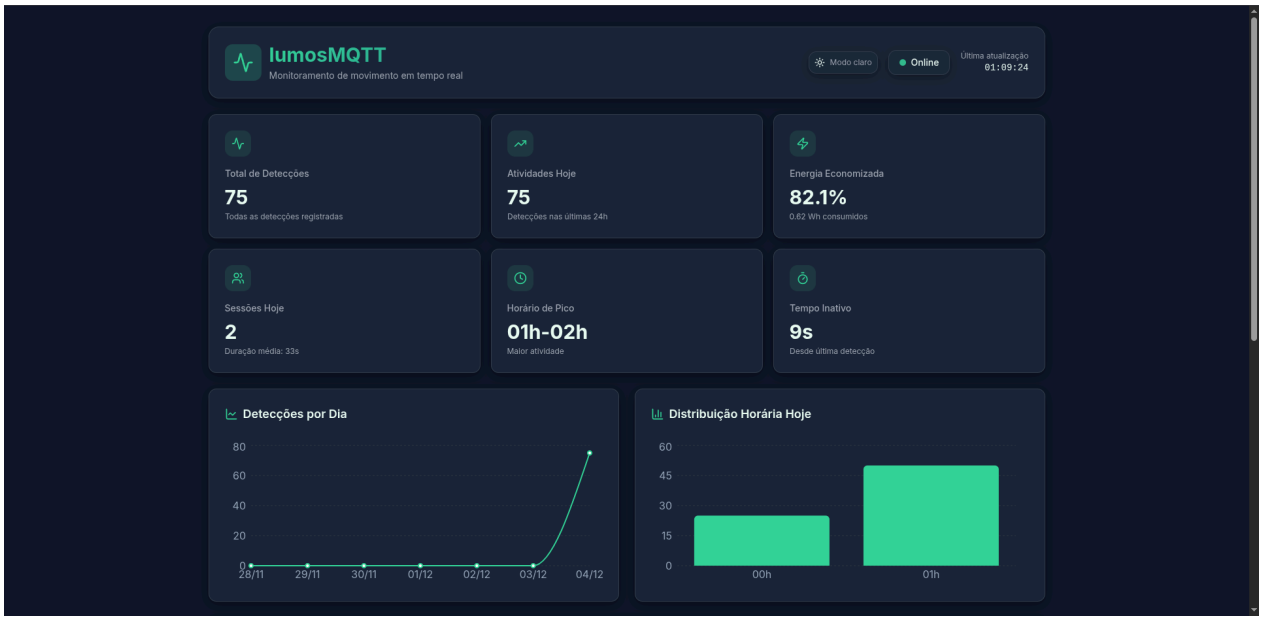
- **Tabela de eventos recentes:** lista dos eventos mais recentes com ID, horário formatado e timestamp; oferece botões para exportar o histórico completo em **JSON** ou **CSV**.

O frontend consome a API a partir da variável VITE_API_BASE_URL, definida em .env.

3 RESULTADOS

Esta seção apresenta os principais resultados obtidos após a implantação do sistema, com base em dados reais coletados pelo lumosMQTT.

3.1 Coleta de eventos e métricas de presença



Durante o período analisado, o sistema registrou 75 detecções totais, sendo 75 atividades apenas nas últimas 24 horas, o que indica que todo o movimento capturado ocorreu no dia atual. No mesmo intervalo, foram contabilizadas 2 sessões de presença, com duração média de 33 segundos, sugerindo interações rápidas no ambiente monitorado.

O painel também identificou um horário de pico entre 01h e 02h, destacando esse período como o de maior concentração de movimentos. A métrica de energia revelou uma economia de 82,1%, correspondendo a apenas 0,62 Wh consumidos, demonstrando alta eficiência energética do sistema.

A funcionalidade de “tempo inativo” indicou que haviam se passado 9 segundos desde a última detecção no momento do snapshot, permitindo avaliar períodos de baixa utilização. A distribuição horária mostra maior número de detecções entre 01h e 02h, reforçando o padrão observado no horário de pico.

3.2 Economia de energia estimada

Com base na reconstrução das janelas de alto brilho (LED em BRIGHT_HIGH) e baixo brilho (BRIGHT_LOW), o backend estima o consumo diário de energia e o percentual de economia obtido pela estratégia de iluminação inteligente.

Em **um dos cenários registrados**, o sistema apresentou:

- **Energia economizada:** aproximadamente **81,5%** em relação a um cenário de LED sempre em alto brilho.
- **Energia utilizada:** cerca de **7,52 Wh** no período analisado.

Esses resultados demonstram o potencial da abordagem para reduzir desperdícios de energia em ambientes que ficam ociosos por longos períodos.

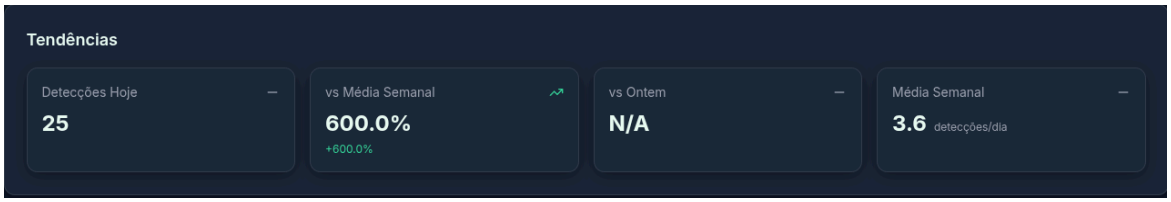
3.3 Distribuições e tendências

Os gráficos do dashboard mostraram que **em um dos cenários testados**:

- **Curva de detecções por dia** com crescimento acentuado a partir de determinado dia de testes, alcançando valores próximos de 380 eventos no último dia observado.
- **Distribuição horária** com concentração de eventos nas primeiras horas da manhã e pico entre 10h e 11h, com barras significativamente mais altas nesse intervalo.

As métricas de tendência (trends) indicaram, por exemplo, que:

- o número de detecções do dia pode ser mais de **600% maior** que a média semanal;



Essas informações permitem uma análise mais rica do comportamento do ambiente ao longo do tempo.

3.4 Interface gráfica e experiência de uso

O dashboard lumosMQTT apresenta as informações de forma clara e organizada, com:

- cards informativos em destaque na parte superior;
- gráficos alinhados na seção intermediária;
- tabela detalhada de eventos na parte inferior, incluindo funcionalidade de exportação em **JSON** e **CSV**.

A alternância entre modos claro/escuro e a indicação de status Online/Offline contribuem para uma melhor experiência de uso, aproximando o projeto de uma aplicação profissional.

4 CONCLUSÃO

O projeto lumosMQTT atingiu o objetivo de implementar um sistema IoT completo para **monitoramento de movimento e controle inteligente de iluminação**, integrando microcontrolador, sensores, atuadores, comunicação MQTT, backend com persistência de dados e dashboard web para visualização em tempo real.

Do ponto de vista de sistemas embarcados, o uso do **ESP32** com **FreeRTOS**, sensor PIR e controle de LED por PWM permitiu explorar conceitos de tarefas, temporização, detecção de bordas e uso eficiente de recursos de hardware. Na camada de comunicação, o protocolo **MQTT** mostrou-se adequado para envio de eventos compactos e desacoplamento entre dispositivo e servidor.

O backend em **Flask + SQLite** possibilitou o armazenamento estruturado dos eventos e a implementação de métricas mais sofisticadas, como sessões de presença, tempos ociosos e estimativa de economia de energia. O dashboard em **React**, por sua vez, forneceu uma interface amigável para análise visual desses dados, aproximando o projeto de um cenário real de monitoramento.

Os resultados obtidos, incluindo economia de energia estimada superior a 80% e identificação clara de padrões de uso por horário, evidenciam a eficácia da solução proposta e reforçam a importância de combinar sistemas embarcados com análise de dados para tomada de decisão.

4.1 Desafios enfrentados

Entre os principais desafios enfrentados, destacam-se:

- ajustes na sensibilidade e estabilização do sensor PIR;
- sincronização de horário via NTP para geração de timestamps confiáveis;
- definição de critérios para agrupamento de eventos em sessões de presença;
- configuração de rede e endereços do broker MQTT em ambientes diferentes (local, Docker, Wi-Fi).

A superação desses desafios contribuiu para o amadurecimento técnico da equipe, tanto em aspectos de hardware quanto de software.

4.2 Melhorias futuras

Como extensões naturais do projeto, são propostas as seguintes melhorias:

- inclusão de sensor de luminosidade (LDR) para combinar presença com luz ambiente na decisão de acender o LED;
- implementação de alertas por e-mail ou aplicativos de mensagem em horários específicos ou situações anômalas;
- adoção de autenticação e, eventualmente, criptografia (TLS) no MQTT, visando ambientes de produção;
- evolução do histórico de eventos com filtros por data e faixa de horário diretamente no dashboard;
- parametrização remota de janelas de movimento, níveis de brilho e limiares de sessão;
- exploração de ferramentas de “web flashing” para facilitar a atualização do firmware do ESP32.

APÊNDICE A – CÓDIGOS E CONFIGURAÇÕES

- **Github:** https://github.com/brunoribeirol/lumosMQTT
  - **Firmware ESP32:** arquivo esp32-esp8266/src/main.cpp, contendo configuração de pinos, tarefas FreeRTOS, conexão Wi-Fi, cliente MQTT e envio de eventos.
  - **Backend:** arquivos backend/app.py e backend/database.py, com implementação da API REST, consumo de mensagens MQTT e acesso ao SQLite.
  - **Configuração do broker MQTT:** arquivo backend/mosquitto.conf, incluindo listeners para IPv4 (porta 1883) e IPv6 (porta 1884).
  - **Dashboard:** componentes React localizados em frontend/src, com implementação dos cards, gráficos e tabela de eventos.
  - **Arquivos de ambiente:** .env.example no backend e frontend, servindo de modelo para configuração de variáveis sensíveis.