Smart Classroom Monitoring

Sistemas Embarcados - Cesar school

Felipe Rodrigues S. Azevedo, Alec C. Theotonio, Eurivaldo José de Vasconcelos F, Pedro Wanderley de Lira S, Lucas Ferraz S. Filgueiras.

frsa@cesar.school, act@cesar.school, ejvf@cesar.school, pwls@cesar.school, lfsf@cesar.school

Recife, Pernambuco.

2025

1. INTRODUÇÃO.....1
1.1 CONTEXTO.....1
1.2 MOTIVAÇÃO.....1
1.3 OBJETIVOS.....2
2. METODOLOGIA.....2
2.1. DIAGRAMA DO SISTEMA.....2
2.2. LISTA DE HARDWARE E SOFTWARE.....3
2.3. FLUXO DE COMUNICAÇÃO E PROCESSAMENTO.....5
3. RESULTADOS.....5
3.1. TÓPICOS E DADOS COLETADOS.....5
3.2. LÓGICA DE CONTAGEM DE PESSOAS.....6
3.3. INFRAESTRUTURA.....7
4. CONCLUSÃO.....9
4.1. DESAFIOS E APRENDIZADOS.....9
4.2. MELHORIAS FUTURAS.....10
5. APÊNDICE.....11

1. INTRODUÇÃO

1.1 CONTEXTO

A modernização dos ambientes de ensino, conhecida como *Smart Classroom*, consiste na integração de tecnologias inteligentes para aprimorar tanto a eficiência energética quanto o conforto dos usuários. Nesse contexto, o monitoramento de variáveis ambientais — como temperatura, umidade e luminosidade — aliado à detecção precisa de ocupação, torna-se essencial para uma gestão mais eficiente dos espaços. Essas informações permitem otimizar recursos, automatizar processos e criar ambientes mais adequados às necessidades dos estudantes e professores

1.2 MOTIVAÇÃO

A gestão manual de variáveis ambientais em ambientes educacionais — como iluminação, temperatura, umidade e ocupação — tende a ser imprecisa e dependente exclusivamente da percepção humana, o que pode resultar em uso ineficiente dos recursos e diminuição do conforto dos usuários. Nesse contexto, torna-se essencial a implementação de sistemas automatizados capazes de realizar monitoramento contínuo e coleta estruturada de dados em tempo real.

O presente projeto surge da necessidade de integrar tecnologias modernas de Internet das Coisas (IoT), sistemas embarcados e comunicação sem fio para permitir uma análise mais precisa das condições ambientais e do padrão de ocupação. A adoção do protocolo MQTT, aliada ao uso do ESP32 como plataforma central de sensores e atuadores, possibilita a criação de uma infraestrutura distribuída, confiável e de baixo custo.

Assim, este sistema se justifica pela oportunidade de demonstrar, na prática, a aplicação de conceitos de automação inteligente — alinhada a cenários de Smart Building e cidades inteligentes — ao mesmo tempo em que oferece uma solução escalável e aderente às demandas contemporâneas de eficiência energética, conforto ambiental e gestão baseada em dados.

1.3 OBJETIVOS

Desenvolver um sistema IoT para monitoramento ambiental e detecção de ocupação em salas de aula, integrando sensores, comunicação sem fio e visualização dos dados em tempo real.

Objetivos Específicos

- Implementar um firmware modular e eficiente utilizando FreeRTOS, permitindo a execução concorrente e determinística das rotinas de leitura dos diferentes sensores.

- Realizar o monitoramento contínuo das variáveis ambientais — temperatura, umidade e luminosidade — bem como do fluxo de pessoas por meio de sensores ópticos, identificando eventos de entrada e saída.

- Estabelecer uma infraestrutura de comunicação confiável baseada no protocolo MQTT, empregando um broker Mosquitto em container Docker para garantir escalabilidade e interoperabilidade do sistema.

- Disponibilizar os dados coletados em uma interface de acompanhamento em tempo real, possibilitando sua análise e uso para tomadas de decisão.

2. METODOLOGIA

2.1. DIAGRAMA DO SISTEMA

A solução proposta adota uma arquitetura IoT orientada a eventos, estruturada segundo o modelo *Publish/Subscribe*, de forma a garantir baixo acoplamento entre dispositivos, escalabilidade e comunicação assíncrona. A arquitetura é organizada em três camadas principais:

- **Camada de Borda (Edge):** Constituída por um dispositivo ESP32 executando o sistema operacional FreeRTOS, responsável pela aquisição periódica dos dados ambientais, pela detecção de ocupação e pela

implementação da lógica de pré-processamento local.

- **Camada de Transporte:** Composta pela rede Wi-Fi local e pelo protocolo MQTT, operando na porta padrão 1883, garantindo a comunicação leve, confiável e orientada a tópicos entre os sensores e o servidor.

- **Camada de Servidor:** Implementada por meio do broker Eclipse Mosquitto, executado em um ambiente containerizado utilizando Docker e orquestrado via Docker Compose, proporcionando fácil implantação, isolamento e flexibilidade na configuração dos serviços.

Essa organização modular permite a integração eficiente dos sensores, assegurando fluxo contínuo de dados e facilitando futuras expansões do sistema

2.2. LISTA DE HARDWARE E SOFTWARE

Hardware Utilizado

- Microcontrolador**:** ESP32 DevKit V1, responsável pela execução do firmware, coleta de dados e comunicação via Wi-Fi.

- Sensores Ambientais:

    - *DHT11* — responsável pela leitura de temperatura e umidade.

    - *LDR* — utilizado para medir a luminosidade ambiental.

- Sensores de Ocupação:

    - Dois módulos *TCRT-5000* (sensores infravermelhos reflexivos), empregados para detecção de fluxo de pessoas (entrada e saída).

- Atuadores:
    - LED indicador de ocupação, utilizado para sinalização visual do estado de presença detectado pelo sistema.

Software Utilizado

- Ambiente de Desenvolvimento: PlatformIO integrado ao Visual Studio Code, utilizando o framework Arduino para geração do firmware.
- Sistema Operacional Embarcado: FreeRTOS, empregado para gerenciar multitarefa, sincronização e comunicação entre tasks no ESP32.
- Infraestrutura de Comunicação: Broker MQTT Eclipse Mosquitto, executado em container Docker (*imagem* `eclipse-mosquitto:latest`).
- Bibliotecas e Dependências**:**
    - *PubSubClient* — implementação do protocolo MQTT para o ESP32.
    - *DHT Sensor Library* e *Adafruit Unified Sensor* — responsáveis pelo tratamento e abstração do sensor DHT11.

Essa combinação de hardware e software garante a confiabilidade das leituras, a flexibilidade no desenvolvimento e a integração eficiente entre as diferentes camadas do sistema IoT.

2.3. FLUXO DE COMUNICAÇÃO E PROCESSAMENTO

O firmware foi estruturado de forma a garantir comunicação eficiente entre as diferentes rotinas, evitando bloqueios e preservando a responsividade do sistema. Para isso, utiliza-se o mecanismo de **filas** (*QueueHandle_t*) do FreeRTOS, que realiza a troca assíncrona de mensagens entre as tasks responsáveis pela leitura dos sensores e a task dedicada à transmissão dos dados via MQTT.

Durante a inicialização, o sistema realiza uma verificação da disponibilidade física de cada sensor (DHT11, LDR e TCRT-5000). Apenas os módulos detectados como operacionais têm suas respectivas tasks ativadas, evitando desperdício de processamento e garantindo robustez à solução.

A detecção de ocupação é conduzida pela *taskTCRT*, que implementa uma **máquina de estados** capaz de identificar a ordem de acionamento dos dois sensores ópticos. As sequências S1→S2 e S2→S1 correspondem, respectivamente, a eventos de entrada e saída, os quais são enviados para a fila `queueEntradaSaida`, permitindo que outras tasks processem essas informações sem perda de eventos.

A transmissão dos dados é realizada pela *taskMQTT*, responsável por consumir as filas de mensagens — tanto de variáveis ambientais quanto de eventos de ocupação — e publicar os valores nos tópicos configurados no broker. Essa separação entre aquisição e transmissão garante que operações de rede, que podem apresentar latência, não interfiram na leitura de sensores críticos nem comprometam o funcionamento do sistema em tempo real.

3. RESULTADOS

3.1. TÓPICOS E DADOS COLETADOS

O sistema foi configurado para publicar periodicamente os dados coletados pelo ESP32 por meio do protocolo MQTT, utilizando uma estrutura de tópicos organizada e coerente com o modelo *Publish/Subscribe*. Cada tópico representa um tipo de informação monitorada, permitindo que diferentes clientes

MQTT — como dashboards, sistemas de registro ou aplicações analíticas — consumam os dados de forma modular.

A Tabela a seguir apresenta os tópicos definidos, a descrição dos dados transmitidos e exemplos dos *payloads* publicados pelo dispositivo:

Tópico MQTT	**Descrição do Dado**	**Exemplo de Payload**
sala/status	Status da conexão	"ESP32 Online"
sala/temperatura	Temperatura em °C	"24.50"
sala/umidade	Umidade Relativa %	"58.00"
sala/luminosidade	Nível de luz (0-100%)	"75.00"
sala/pessoas	Contador de pessoas	"3"
sala/ocupacao	Status binário	"OCUPADA" ou "VAZIA"

3.2. LÓGICA DE CONTAGEM DE PESSOAS

A detecção de fluxo de pessoas foi estruturada por meio de uma máquina de estados implementada na *taskTCRT*, responsável por monitorar continuamente o padrão de acionamento dos dois sensores TCRT-5000 instalados no ponto de passagem. A lógica avalia a ordem em que cada sensor é ativado, permitindo identificar corretamente a direção do movimento:

- Entrada**:** ocorre quando o Sensor 1 (externo) é acionado antes do Sensor 2 (interno), resultando no incremento do contador de pessoas.

- Saída**:** ocorre quando o Sensor 2 é ativado antes do Sensor 1, resultando no decremento do contador.

Para garantir precisão e evitar contagens indevidas, foi implementado um mecanismo de **debounce via software**, fundamental devido à natureza assíncrona e sensível dos sensores infravermelhos. Utilizou-se um tempo mínimo de estabilização (*DEBOUNCE_DELAY = 50 ms*), durante o qual a leitura só é considerada válida se o estado do sensor permanecer consistente. Esse procedimento evita oscilações momentâneas que poderiam gerar falsos positivos.

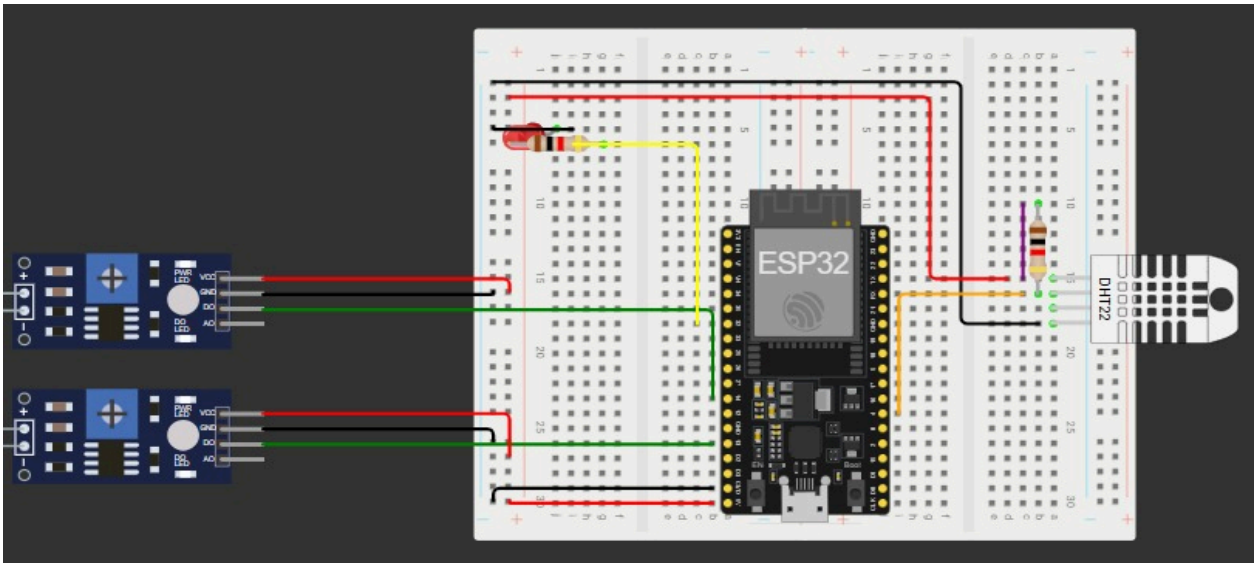
Após a atualização do contador, o sistema avalia o estado de ocupação: quando `salaOcupada` é verdadeiro, o LED conectado ao GPIO 2 é acionado automaticamente, fornecendo ao usuário uma indicação visual imediata da presença de pessoas no ambiente. Essa integração entre máquina de estados, debounce e sinalização local assegura um funcionamento robusto e confiável do sistema de detecção.

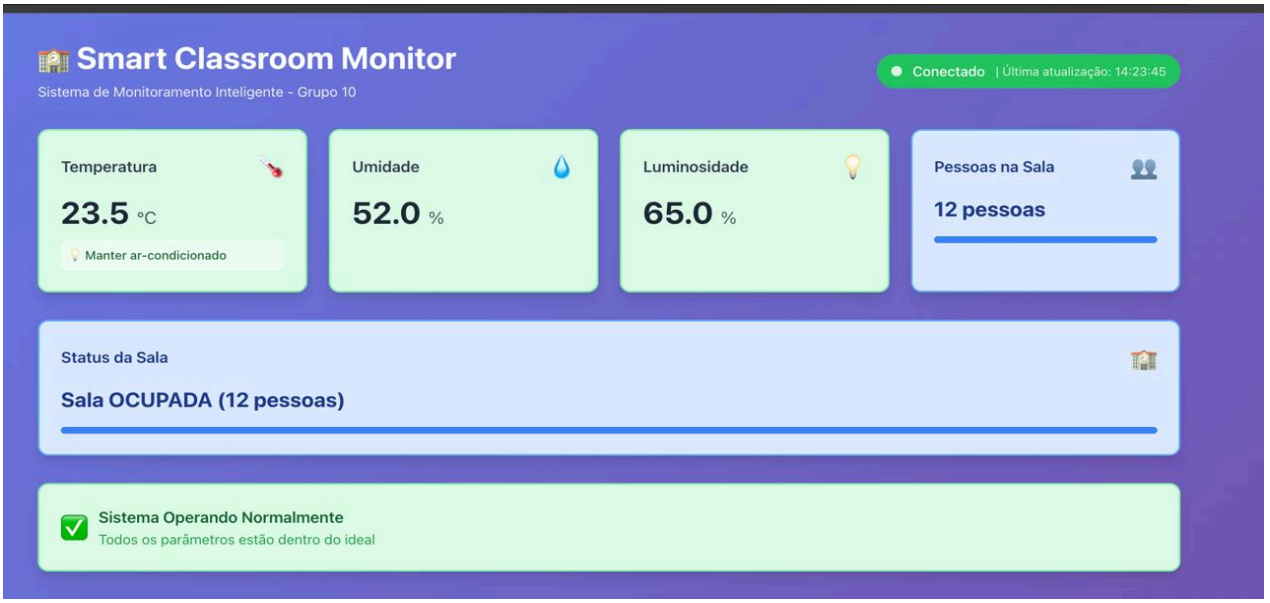
3.3. INFRAESTRUTURA

O broker MQTT utilizado no sistema — **Eclipse Mosquitto** — foi implantado em ambiente containerizado por meio do Docker, garantindo portabilidade, isolamento e facilidade de configuração. A utilização do *Docker Compose* simplifica a orquestração dos serviços, permitindo inicializar toda a infraestrutura com um único comando (docker-compose up), além de facilitar a manutenção, replicação e escalabilidade do ambiente.

Essa abordagem reduz dependências do sistema operacional, assegura reprodutibilidade do ambiente de testes e possibilita que o broker seja executado de forma estável, independentemente da máquina hospedeira. Com isso,

a comunicação entre o ESP32 e o broker torna-se mais confiável, contribuindo para a robustez da solução IoT como um todo.





4. CONCLUSÃO

4.1. DESAFIOS E APRENDIZADOS

A implementação do FreeRTOS desempenhou papel fundamental na coordenação das tarefas concorrentes, especialmente diante da natureza assíncrona dos sensores utilizados. Um dos principais desafios enfrentados foi o tratamento de *ruído* e oscilações nos sensores infravermelhos TCRT-5000, que poderiam gerar leituras instáveis e contagens incorretas. Esse problema foi mitigado por meio da aplicação de um mecanismo de *debounce* via software, utilizando uma janela de verificação temporal de 50 ms (*DEBOUNCE_DELAY*), garantindo maior confiabilidade na detecção da sequência dos sensores.

Outro ponto relevante foi a configuração da infraestrutura de comunicação. A adoção do Docker, em conjunto com o *Docker Compose*, simplificou a implantação e gerenciamento do broker MQTT, reduzindo complexidades associadas à configuração manual do ambiente de servidor. Esse

processo evidenciou a importância das práticas de containerização para o desenvolvimento de soluções reprodutíveis, portáveis e de fácil manutenção.

Esses desafios e soluções contribuíram para o amadurecimento do projeto, além de reforçarem a importância de arquiteturas bem estruturadas e de práticas robustas no desenvolvimento de sistemas IoT.

4.2. MELHORIAS FUTURAS

Para a evolução do sistema e ampliação de suas funcionalidades, algumas melhorias podem ser implementadas em versões futuras do projeto:

- **Implementar autenticação no broker MQTT:** adicionar mecanismos de segurança baseados em usuário e senha, desabilitando a configuração *allow_anonymous* atualmente ativa, a fim de garantir maior proteção na comunicação entre dispositivos e o servidor.

- **Integrar um display OLED local:** permitir a exibição direta da contagem de pessoas e demais informações relevantes no próprio ambiente monitorado, reduzindo a dependência de uma interface externa ou conexão com o dashboard.

- **Calibrar limiares via mensagens MQTT:** tornar a configuração dos valores de temperatura mínima e máxima (TEMP_MIN/TEMP_MAX) dinâmica, permitindo que o ajuste seja realizado remotamente por meio de mensagens *downlink*, aumentando a flexibilidade e adaptabilidade do sistema.

Esses aprimoramentos contribuiriam para tornar o sistema mais seguro, informativo e configurável, ampliando seu potencial de uso em diferentes cenários reais.

5. APÊNDICE

APÊNDICE A – CÓDIGOS E CONFIGURAÇÕES

- Github: https://github.com/alecct812/Project-embedded-systems---Smart-Classroom-Monioring
- Firmware ESP32: arquivo esp32-esp8266/src/main.cpp, contendo configuração de pinos, tarefas FreeRTOS (taskDHT11, taskLDR, taskTCRT, taskMQTT), conexão Wi-Fi, lógica de contagem de pessoas e publicação de sensores via cliente MQTT. https://github.com/alecct812/Project-embedded-systems---Smart-Classroom-Monioring/blob/main/esp32-esp8266/src/main.cpp
- Infraestrutura (Docker): arquivo raspberry-pi/docker-compose.yml, com a definição do serviço mosquitto, mapeamento de portas (1883 para MQTT e 9001 para WebSocket) e configuração de volumes persistentes. https://github.com/alecct812/Project-embedded-systems---Smart-Classroom-Monioring/blob/main/raspberry-pi/docker-compose.yml
- Configuração do broker MQTT: arquivo raspberry-pi/mosquitto/config/mosquitto.conf, incluindo permissão de acesso anônimo, *listener* na porta 1883 (acessível externamente) e configurações de persistência e logs. https://github.com/alecct812/Project-embedded-systems---Smart-Classroom-Monioring/blob/main/raspberry-pi/mosquitto/config/mosquitto.conf
- Ambiente e Dependências: arquivo Projeto/platformio.ini, servindo de configuração para o ambiente de compilação (env:esp32dev) e listando as bibliotecas necessárias (Adafruit DHT, PubSubClient, Adafruit Unified Sensor). https://github.com/alecct812/Project-embedded-systems---Smart-Classroom-Monioring/blob/main/esp32-esp8266/platformio.ini
- Configuração do servidor proxy MQTT para WebSocket https://github.com/alecct812/Project-embedded-systems---Smart-Classroom-Monioring/blob/main/raspberry-pi/mqtt-proxy-server.js