

**UNIVERSIDADE DE SOROCABA**

**PRÓ-REITORIA DE GRADUAÇÃO E ASSUNTOS ESTUDANTIS**

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**Juan Dantas Morales**

**Miguel de Oliveira Jardim**

**COORDENAÇÃO DE AÇÕES COM MQTT**

SISTEMA COM PROTOCOLO MQTT

**Sorocaba/SP 2025**

SISTEMA MQTT

**RESUMO**

Em nosso projeto, formamos um sistema de monitoramento e controle da temperatura de uma estufa utilizando o protocolo MQTT. O programa sensor.py é responsável por simular um sensor de temperatura, conectando-se ao broker MQTT e publicando, a cada cinco segundos, valores aleatórios que variam entre 15 °C e 30 °C no tópico estufa/temperatura. Já o cliente.py atua como o centro de controle do sistema: ele também se conecta ao broker, recebe as leituras publicadas pelo sensor e as compara com limites pré-estabelecidos, definidos entre 18 °C e 25 °C. Quando a temperatura ultrapassa 25 °C, o cliente envia um comando para ligar o ventilador, e quando a temperatura cai abaixo de 18 °C, ele envia o comando para desligá-lo, além de publicar o status correspondente. Por fim, o ventilador.py representa o atuador do sistema: ao se conectar ao broker e ouvir o tópico de comando, ele reage às mensagens enviadas pelo cliente, exibindo quando deve ligar ou desligar.

Assim, em conjunto, os três programas simulam um sistema automatizado de climatização para uma estufa, no qual o sensor gera dados de temperatura, o cliente processa essas informações e toma decisões, e o ventilador executa a ação necessária para manter a temperatura dentro da faixa adequada.

**Palavras-chave:** sensor.py, cliente.py, ventilador.py

**SUMÁRIO**

[1 INTRODUÇÃO 2](#_Toc198114907)

[2 objetivos 3](#_Toc198114908)

[3 revisão de literatura 4](#_Toc198114909)

[4 DESENVOLVIMENTO 5](#_Toc198114910)

[4.1 OWASP Top 10 5](#_Toc198114911)

[4.2 A APLICAÇÃO DESENVOLVIDA 5](#_Toc198114912)

[5 RESULTADOS 6](#_Toc198114913)

[6 CONCLUSÃO 7](#_Toc198114914)

[Referências 8](#_Toc198114915)

[anexo a – exemplo 9](#_Toc198114916)

# INTRODUÇÃO

Este documento detalha o desenvolvimento e a operação de um sistema de monitoramento de temperatura e controle de ventilação para uma estufa agrícola. O projeto foi concebido para utilizar o protocolo **MQTT (Message Queuing Telemetry Transport)**, estabelecendo uma arquitetura eficiente e de baixo consumo de banda, ideal para aplicações de Internet das Coisas (IoT).

# objetivos

Durante o desenvolvimento de uma aplicação, sendo ela um software para instalação ou uma aplicação web, é de suma importância a execução de testes, com o intuito de encontrar erros e vulnerabilidades. Com isso, esse documento, tem como objetivo apresentar a maneira que os testes são feitos na aplicação web que está sendo desenvolvida, sendo nesse caso, baseados na OWASP TOP 10, usando aplicações gratuitas na execução dos testes.

Assim os objetivos centrais desse documento estão voltados realização de testes que buscam garantir a segurança de todas as áreas críticas da aplicação web, garantindo que todas sejam avaliadas, utilizando como base referencial a lista OWASP TOP 10 para identificação de vulnerabilidades assegurando que os testes consigam captar os erros mais relevantes da aplicação desenvolvida.

Além disso, serão utilizadas aplicações gratuitas de teste de segurança, as quais, permitem execução de testes de segurança eficazes, fazendo com que não sejam necessários altos investimentos. Acrescentando registros e análises dos resultados obtidos durante os testes, facilitando a identificação de falhas e a implementação de correções necessárias.

Finalizando com métodos de melhorias contínua que permita a atualização constante dos testes e a adaptação às novas ameaças e vulnerabilidades que possam surgir no ambiente digital.

# revisão de literatura

# VISÃO GERAL DO SISTEMA

Os três programas implementam um sistema de IoT simples para uma estufa: um sensor publica a temperatura via MQTT, um cliente de controle recebe essas leituras e decide ligar/desligar a ventilação, e um atuador (ventilador) escuta os comandos e “executa” a ação (nesta simulação, apenas imprime no console). Eles usam o mesmo broker MQTT (host localhost, porta 1883) e se comunicam por tópicos.

## SENSOR.PY – SIMULADOR DE SENSOR DE TEMPERATURA

* **Conexão MQTT:** Cria um cliente Paho MQTT (ID SensorTemperatura) e conecta no broker localhost:1883. Define um callback simples para informar se a conexão deu certo (código de retorno rc == 0).
* **Publicação periódica:** Entra num while True gerando uma temperatura aleatória (15.00 a 30.00 °C) com duas casas decimais, publica no tópico estufa/temperatura a cada 5 segundos e imprime no console o valor publicado. Em KeyboardInterrupt (Ctrl+C), encerra a simulação e desconecta.

### OBSERVAÇÕES TÉCNICAS ÚTEIS

* A publicação acontece continuamente; em projetos reais, é comum iniciar o loop de rede (client.loop\_start()) para garantir o processamento de keep-alive e fila de mensagens em background. Aqui, como há publicações frequentes, tende a funcionar, mas o uso explícito do loop melhora a robustez.
* O uso de um tópico claro e específico (estufa/temperatura) facilita o desacoplamento entre produtor (sensor) e consumidores (lógica/atuadores).

## CLIENTE.PY – LÓGICA DE CONTROLE

### TÓPICOS E LIMITES

* Leitura de temperatura: estufa/temperatura
* Comando para ventilação: estufa/ventilacao/comando
* Status publicado: estufa/ventilação/status
* Além disso, fixa faixa ideal de temperatura [18.0, 25.0] °C e mantém um estado interno ventilador\_ligado (boolean) para não reenviar comandos repetidos.

Assinatura e processamento: Ao conectar com sucesso, assina estufa/temperatura. No callback on\_message:

1. Converte o payload recebido para float (tratando ValueError se vier dado inválido).
2. Decide:
   1. Se > 25.0 °C e ventilador desligado, publica "ligar" no tópico de comando e publica status "ligado"; atualiza ventilador\_ligado = True.
   2. Se < 18.0 °C e ventilador ligado, publica "desligar" e o status "desligado"; atualiza ventilador\_ligado = False.
   3. Caso contrário, apenas imprime que a “temperatura ideal” (isto é, não faz ação de comutação).
3. O cliente roda loop\_forever(), mantendo a conexão e processando mensagens continuamente.

### OBSERVAÇÕES TÉCNICAS ÚTEIS

* Prevenção de “chattering”: O estado ventilador\_ligado evita mandar o mesmo comando repetidamente a cada nova leitura enquanto a condição permanecer verdadeira.
* Histerese / faixa morta: Já existe uma espécie de histerese operacional porque o liga ocorre em > 25 e o desliga em < 18; como as bordas são afastadas, reduz pulsos liga/desliga se a leitura oscilar — ainda assim, em cenários reais, você pode querer uma lógica explícita de histerese (ex.: liga > 25 e só desliga quando < 23) para suavizar transições.
* Status dedicado: Publicar também no tópico de status (estufa/ventilacao/status) permite que outros sistemas (dashboards, logs, alarmes) acompanhem o estado sem precisarem inferir pelo último comando.
* Mensagens de log: Há uma pequena inconsistência textual, quando a temperatura está < 18 °C e o ventilador está ligado, a mensagem impressa começa com “Temperatura ideal. Desligando…” (poderia ser “Temperatura baixa. Desligando…”); é só ajuste de texto, a lógica está correta.

## VENTILADOR.PY – SIMULADOR DE ATUADOR

* Assinatura de comandos: Conecta ao mesmo broker com o ID VentiladorSimulado, assina o tópico estufa/ventilacao/comando e fica aguardando.
* Ação ao receber: No on\_message, lê o payload: se "ligar", imprime [VENTILADOR] LIGAR!; se "desligar", imprime [VENTILADOR] DESLIGAR!; qualquer outra coisa, informa “comando desconhecido”. Roda loop\_forever() para manter-se sempre escutando.

### OBSERVAÇÕES TÉCNICAS ÚTEIS

Aqui o “atuador” é apenas um print, mas em hardware real você acionaria um GPIO (Raspberry Pi) ou uma interface de relé para energizar um motor/contator. A estrutura do código já isola a camada de controle (quem decide) da camada de execução (quem faz).

## COMO TODOS SE COMUNICAM

1. sensor.py publica temperatura em estufa/temperatura a cada 5 s.
2. cliente.py está inscrito nesse tópico, lê a última medida, compara com os limites e publica em estufa/ventilacao/comando "ligar" ou "desligar" quando precisa (e também atualiza o status).
3. ventilador.py está inscrito no tópico de comando e executa a ação recebida (na simulação, imprime).

# MANUAL DA APLICAÇÃO

Este projeto implementa um sistema de automação para uma estufa agrícola, utilizando o protocolo MQTT (Message Queuing Telemetry Transport) para coordenar a ação de um ventilador com base na leitura de um sensor de temperatura. O objetivo principal é manter a temperatura ideal dentro do intervalo de 18°C a 25°C.

O sistema é composto por três scripts Python que se comunicam via um Broker MQTT central, seguindo o padrão publicador/assinante:

* Sensor (sensor.py): Simula a leitura e publica a temperatura no tópico estufa/temperatura.
* Cliente de Controle (cliente.py): Assina o tópico de temperatura, analisa os dados e, se a temperatura estiver fora da faixa ideal, envia comandos ("ligar" ou "desligar") para o tópico estufa/ventilacao/comando.
* Sistema de Ventilação (ventilador.py): Simula o atuador, assinando o tópico estufa/ventilacao/comando e respondendo aos comandos recebidos.

## REQUISITOS E CONFIGURAÇÃO INICIAL

Para rodar o projeto, você precisa dos seguintes requisitos instalados e configurados.

### Instalação do Python e Biblioteca Paho-MQTT

Você deve ter o Python 3 instalado em seu sistema. Em seguida, a biblioteca MQTT para Python (Paho-MQTT) deve ser instalada.

Instalação da biblioteca: Use o gerenciador de pacotes do Python (pip) para instalar a biblioteca Paho-MQTT.

### Instalação e Início do Broker MQTT (Mosquitto)

O broker é o servidor que centraliza toda a comunicação e deve estar ativo antes de qualquer script Python.

Instalação do Mosquitto: Instale o broker Mosquitto, que é leve e amplamente utilizado em projetos IoT.

Iniciar o Broker: O Mosquitto geralmente inicia automaticamente como um serviço. Para fins de teste e visualização do log, você pode iniciá-lo manualmente em um terminal ou usar um terminal dedicado para esta tarefa. Você deve manter este terminal aberto enquanto o projeto estiver em execução.

## GUIA DE EXECUÇÃO DO SISTEMA

O sistema exige que cada componente (o broker e os três scripts Python) seja executado simultaneamente em terminais separados.

### PREPARAÇÃO

Abra três terminais adicionais (além do terminal do Mosquitto).

Em cada um desses três terminais, navegue até a pasta onde você salvou os arquivos sensor.py, cliente.py, e ventilador.py.

### SEQUÊNCIA DE INICIALIZAÇÃO

Inicie cada componente em seu próprio terminal, seguindo a ordem abaixo.

## INICIAR O VENTILADOR (TERMINAL 1)

O ventilador é o primeiro a ser iniciado, ficando em modo de escuta.

* Comando de Execução: Rode o script ventilador.py.
* Saída Esperada: Simulador de ventilador conectado e aguardando comandos...

1. INICIAR O CLIENTE DE CONTROLE (TERMINAL 2)

O cliente de controle se conecta e se inscreve no tópico de temperatura.

* Comando de Execução: Rode o script cliente.py.
* Saída Esperada: Cliente de controle conectado. Aguardando dados...

1. INICIAR O SENSOR (TERMINAL 3)

O sensor inicia o fluxo de dados, publicando a temperatura periodicamente.

* Comando de Execução: Rode o script sensor.py.
* Saída Esperada: Sensor conectado ao broker MQTT com sucesso. O sensor começará a publicar temperaturas a cada 5 segundos.

### OBSERVAÇÃO E FLUXO DE DADOS

* O Terminal 3 publica a temperatura.
* O Terminal 2 recebe essa temperatura e avalia a regra (18°C a 25°C).
* Se a regra for violada, o Terminal 2 envia um comando.
* O Terminal 1 recebe o comando do cliente de controle e imprime a confirmação da ação (LIGAR ou DESLIGAR).
* O terminal do Mosquitto registrará todas as conexões e o tráfego de mensagens.

## FINALIZAR A EXECUÇÃO

Para parar o sistema, vá em cada um dos terminais onde os scripts Python estão rodando e pressione a combinação de teclas: Control + C.

# RESULTADOS

# CONCLUSÃO

O projeto demonstrou com sucesso a aplicação do protocolo MQTT como uma solução robusta. O sistema de monitoramento de temperatura e controle de ventilação opera de forma eficiente, desacoplando o sensor, o controlador e o atuador. A arquitetura de publicação/assinatura adotada não apenas atende a todos os requisitos propostos, mas também estabelece uma base sólida para futuras expansões, como a integração de múltiplos sensores ou sistemas de irrigação, validando o MQTT como a espinha dorsal ideal para ambientes de IoT.

Referências

anexo a – exemplo