Aplicativo de monitoramento de temperatura para Raspberry Pi CM4

Elaborador:	Ian Francisco Roland do Nascimento
Orientador:	Prof. Esp. Wilton Ruffato Wonrath

Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Ian Francisco Roland do Nascimento

Aplicativo de monitoramento de temperatura para Raspberry Pi CM4

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas na área de concentração em TI.

Orientador: Prof. Esp. Wilton Ruffato Wonrath

Este trabalho corresponde à versão final do Trabalho de Conclusão de Curso apresentado por Ian Francisco Roland do Nascimento e orientado pelo Prof. Esp. Wilton Ruffato Wonrath

Americana, SP

2025

Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

NASCIMENTO, Ian Francisco Roland do

Aplicativo de monitoramento de temperatura para Raspberry Pi CM4. / Ian Francisco Roland do Nascimento – Americana, 2025.

31f.

Relatório técnico (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana Ministro Ralph Biasi – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Esp. Wilton Ruffato Wonrath

1. Análise de dados 2. Desenvolvimento de software 3. Sistemas embarcados. I. NASCIMENTO, Ian Francisco Roland do II. WONRATH, Wilton Ruffato III. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana Ministro Ralph Biasi

CDU: 681516
681.3.05
681518

Elaborada pelo autor por meio de sistema automático gerador de ficha catalográfica da Fatec de Americana Ministro Ralph Biasi.

Fatec Americana Ministro Ralph Biasi

Centro Paula Souza

SÃO PAULO GOVERNO DO ESTADO

Ian Francisco Roland do Nascimento

Aplicativo de Monitoramento de Temperatura para Raspberry Pi CM4

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana Ministro Ralph Biasi.
Área de concentração: Análise e Desenvolvimento de Sistemas.

Americana, 14 de junho de 2025.

Banca Examinadora:

Wilton Ruffato Wonrath
Especialista
Fatec Americana "Ministro Ralph Biasi"

Rafael Rodrigo Martinati
Mestre
Fatec Americana "Ministro Ralph Biasi"

Lucas Serafim Parigotto
Especialista
Fatec Americana "Ministro Ralph Biasi"

SUMÁRIO

Lista de Figuras

Lista de Tabelas

1 OBJETIVO

O monitoramento contínuo da temperatura em sistemas embarcados, como o *Raspberry Pi CM4*, é essencial para garantir a estabilidade operacional, prevenir danos ao hardware e evitar falhas críticas em aplicações que demandam alto desempenho. Tradicionalmente, soluções de monitoramento podem envolver ferramentas que exigem grande potência computacional, interfaces pouco intuitivas ou a ausência de integração com sistemas web modernos, o que limita a eficiência e a acessibilidade dos dados em tempo real.

Este relatório técnico tem como objetivo desenvolver um software para monitoramento de temperatura em sistemas embarcados, utilizando o Raspberry Pi CM4 integrado a CM4 IO Board como plataforma de hardware. A solução proposta integra um backend em Python com o framework Flask para captura e processamento de dados, e um frontend em React para visualização interativa e em tempo real por meio de uma interface web. Além disso, busca-se atingir um software com design moderno e integração direta com ventoinhas de refrigeração, visando aprimorar a confiabilidade e a eficiência do monitoramento.

A metodologia adotada inclui o desenvolvimento orientado a testes, a arquitetura cliente-servidor para comunicação entre o *Raspberry Pi CM4* e a interface web, e o uso de módulos específicos para leitura da temperatura (como *Subprocess* do Python).

Espera-se que este projeto contribua para a otimização de sistemas de monitoramento em ambientes embarcados, oferecendo uma alternativa escalável, de baixo custo e de fácil adaptação para cenários acadêmicos, industriais ou de automação residencial. Como perspectivas futuras, destacam-se a inclusão de notificações por e-mail, análises preditivas de falhas via machine learning e suporte a múltiplos dispositivos simultâneos.

2 DESENVOLVIMENTO

O sistema foi implementado utilizando um Raspberry Pi Compute Module 4 (CM4) acoplado à CM4 I/O Board, plataforma que oferece interfaces físicas robustas (GPIO, USB, Ethernet) e suporte a dissipação térmica, essencial para operações contínuas. O sistema operacional baseado em Linux (Raspberry Pi OS Lite) foi configurado para inicialização mínima, otimizando recursos computacionais.

A leitura da temperatura do processador é realizada via módulo subprocess do Python, executando o comando embutido ['vcgencmd', 'measure_temp'] para capturar dados diretamente do sensor interno do Raspberry Pi. Esse método dispensa o uso de sensores externos, reduzindo custos e complexidade de hardware. Os dados são coletados em intervalos regulares (ex.: a cada 5 segundos) e processados pelo backend desenvolvido em Flask, que disponibiliza uma API REST com o endpoint /temperature_and_datetime para consulta em formato JSON.

O sistema proposto implementa controle ativo por meio de um limiar (threshold) configurável, acionando automaticamente uma ventoinha conectada ao conector dedicado da Raspberry Pi CM4 I/O Board quando a temperatura atinge valores críticos. Essa funcionalidade garante automação na prevenção de superaquecimento, reduzindo riscos de danos ao hardware.

O frontend, desenvolvido em React, realiza requisições periódicas ao backend via fetch API, garantindo a atualização dinâmica da interface sem recarregamento da página. A biblioteca Shadcn (que utiliza da biblioteca recharts.js) é utilizada para plotar gráficos em tempo real, exibindo variações históricas da temperatura (armazenadas temporariamente em memória RAM, já que o sistema não emprega banco de dados). A interface permite ligar ou desligar uma ventoinha integrada ao CM4 I/O Board via I2C com possibilidade de controle das rotações por meio do driver PWM (modulação por largura de pulso).

2.1 Softwares Similares

No cenário de monitoramento de sistemas embarcados, diversas soluções são utilizadas para coleta e análise de dados térmicos. Os softwares mais relevantes para comparação estão listados a seguir.

Open Hardware Monitor (S1) é uma ferramenta *open-source* que monitora temperatura, voltagem e velocidade de ventoinhas em tempo real. Compatível com Windows e Linux, opera sem necessidade de instalação e oferece suporte a sensores de GPUs e CPUs.

HWMonitor Pro (S2): Versão profissional do HWMonitor, permite monitoramento remoto, geração de gráficos históricos e armazenamento de dados em banco de dados. Indicado para ambientes corporativos que exigem relatórios detalhados.

HWiNFO (S3): Software avançado para diagnóstico de hardware, com suporte a logs em CSV e integração com sensores externos via protocolos como I²C. Amplamente utilizado em servidores e sistemas Linux, incluindo Raspberry Pi.

Leitura Direta via vcgencmd (S4): Método nativo do Raspberry Pi OS para medição da temperatura da CPU, utilizando comandos de terminal. Não possui interface gráfica ou armazenamento de dados, sendo comum em scripts personalizados.

Considerando esses aspectos, a Tabela 1 destaca as diferenças entre os softwares S1, S2, S3 e o sistema desenvolvido neste trabalho (S5):

Tabela 1 - Comparativo de funcionalidades entre softwares de terceiros e o desenvolvido neste trabalho.

Funcionalidade	S1	S2	S3	S4	S5
Código aberto	Sim	Não	Não	Sim	Sim
Armazenamento	Memória ram	Banco de dados	Logs em CSV	Inexistente	Memória ram
Multiplataforma	Sim	Não	Sim	Não	Parcialmente
Controle ativo	Não	Sim	Não	Não	Sim
Custo	Gratuito	Pago	Gratuito	Gratuito	Gratuito
Interface gráfica	Sim	Sim	Sim	Não	Sim

Fonte: Elaborado pelo autor (2025).

2.2 Levantamento de requisitos

A Engenharia de Requisitos (do inglês Requirements Engineering – RE) engloba processos sistemáticos para identificar, analisar, documentar e validar as necessidades de um sistema. um requisito pode ser definido como uma descrição dos serviços oferecidos pelo sistema e das restrições associadas à sua operação (SOMMERVILLE, 2007). Tradicionalmente, os requisitos são classificados em duas categorias: requisitos funcionais (que definem as funcionalidades específicas do sistema) e requisitos não funcionais (relacionados a atributos como desempenho, segurança e usabilidade).

2.2.1 Requisitos funcionais

Os requisitos funcionais definem as funcionalidades que o sistema deve executar, ou seja, descrevem as ações específicas que o software realizará para cumprir seu propósito (SOMMERVILLE, 2007). A Tabela 2 detalha os requisitos funcionais deste projeto, classificados por prioridade (Essencial, Importante, Desejável).

Tabela 2 - Requisitos funcionais do projeto.

Identificação	Requisito Funcional	Prioridade
RF001	Monitorar temperatura em tempo real	Essencial
RF002	Configurar limites de temperatura (threshold)	Essencial
RF003	Ativar/desativar ventoinha via I2C	Essencial
RF004	Visualizar histórico de temperaturas	Importante
RF005	Alertar usuário em caso de temperatura crítica	Essencial
RF006	Acessar interface web em dispositivos remotos	Essencial
RF007	Integrar *backend* (Flask) e *frontend* (React)	Essencial
RF008	Verificar status do hardware (sensor/ventoinha)	Essencial
RF009	Exportar logs de temperatura em CSV	Desejável
RF010	Personalizar intervalo de atualização de dados	Importante

Fonte: Elaborado pelo autor (2025).

2.2.2 Requisitos não funcionais

Requisitos não funcionais referem-se às propriedades gerais do sistema, como desempenho, segurança e usabilidade, que não estão diretamente vinculadas a funcionalidades específicas (SOMMERVILLE, 2007). A **Tabela 3** lista os requisitos não funcionais do projeto, categorizados por domínio e prioridade.

Tabela 3 - Requisitos não funcionais do projeto.

Identificação	Requisito não funcional	Categoria	Prioridade
RNF001	Interface web responsiva e intuitiva	Usabilidade	Essencial
RNF002	Tempo de resposta da API ≤ 2 segundos	Desempenho	Essencial
RNF003	Operação contínua (24/7) sem reinicializações	Confiabilidade	Essencial
RNF004	Comunicação entre *frontend* e *backend* via HTTPS	Segurança	Essencial
RNF005	Compatibilidade com navegadores modernos	Portabilidade	Essencial
RNF006	Consumo de RAM ≤ 30% no Raspberry Pi CM4	Hardware/Software	Importante
RNF007	Uso de bibliotecas Python/React atualizadas	Padrões	Essencial
RNF008	Documentação técnica para replicação do sistema	Manutenibilidade	Desejável
RNF009	Suporte a atualizações via terminal (SSH)	Distribuição	Importante

Fonte: Elaborado pelo autor (2025).

2.3 Recursos e ferramentas

Para o desenvolvimento do sistema de monitoramento térmico, foram selecionados ferramentas e recursos que garantem eficiência, escalabilidade e integração com hardware embarcado. A escolha priorizou tecnologias *open-source* e amplamente documentadas, conforme detalhado a seguir.

1. **Raspberry Pi Compute Module 4 (CM4) + CM4 I/O Board:**
    - **Função**: Plataforma de hardware embarcado para execução do sistema, conectividade GPIO, PWM e interfaceamento com sensores.
    - **Características**: Processador ARM Cortex-A72 de 64 bits, suporte a comunicação via USB, Ethernet, GPIO e PWM (RASPBERRY PI FOUNDATION, 2025).
2. **Python 3.x:**
    - **Função**: Linguagem principal para desenvolvimento do *backend*, integração com sensores e automação de processos.
    - **Recursos**:
        - Módulo **subprocess** para execução do comando vcgencmd measure_temp.
3. **Flask:**
    - **Função**: Framework web em Python para criação da API RESTful, responsável por transmitir dados de temperatura em tempo real ao *frontend*.
    - **Recursos**:
        - Rotas personalizadas (ex: /temperature para consulta de dados).
        - Suporte a *multithreading* para lidar com requisições simultâneas (FLASK, 2025).
        - Integração nativa com bibliotecas Python (ex: subprocess).
4. **React (TypeScript):**
    - **Função**: Construção da interface web responsiva para visualização de dados.
    - **Bibliotecas**:
        - **Recharts.js** para geração de gráficos dinâmicos (RechartsJS, 2025).
        - **Fetch API** para comunicação assíncrona com o *backend*.

5. **Visual Studio Code:**
    - **Função**: Ambiente de desenvolvimento integrado (IDE) para edição de código Python e TypeScript.
    - **Extensões**:
        - *Python* e *React Snippets* para autocompletar e depuração (MICROSOFT, 2025).
6. **Postman**
    - **Função**: Teste de endpoints da API Flask, validação de respostas HTTP e simulação de requisições (POSTMAN, 2025).
7. **Shadcn**
    - **Função:** Biblioteca de componentes de interface (UI) reutilizáveis e acessíveis, utilizada para construção de elementos visuais personalizáveis no frontend.
    - **Características:**
        - **Componentes "Copy-Paste":** Os elementos são copiados diretamente para o código do projeto, permitindo ajustes estruturais e de estilo sem dependências externas.
        - **Integração com Tailwind CSS:** Utiliza classes utilitárias para estilização, garantindo consistência visual e otimização de performance.
        - **Acessibilidade:** Baseia-se em primitivos do Radix UI, assegurando conformidade com padrões WCAG (ex.: navegação por teclado e leitores de tela).
    - **Aplicação no Projeto**: Componentes como gráficos interativos (Chart.js) e alertas térmicos foram estilizados com Shadcn/ui, aproveitando temas claros/escuros e variáveis CSS.
8. **TailwindCSS (v4)**
    - **Função:** Framework CSS utilitário para estilização responsiva e dinâmica da interface web.
    - **Recursos:**
        - Configuração Simplificada: Prescinde de arquivos de configuração tradicionais em versões recentes, reduzindo complexidade.
        - Temas Customizáveis: Variáveis CSS (ex.: --background, --foreground) permitem adaptação visual para diferentes cenários de monitoramento.

2.4.1 Casos de uso

Os diagramas de caso de uso constituem ferramentas essenciais na engenharia de software para modelagem funcional de sistemas, representando interações entre atores e funcionalidades sob a perspectiva do usuário (SOMMERVILLE, 2007). Nesta abordagem, os elementos são simbolicamente representados como:

- **Atores**: Figuras humanas que interagem com o sistema
- **Funcionalidades**: Elipses contendo descrições de ações
- **Relações**: Linhas que conectam atores a funcionalidades

No sistema de monitoramento térmico desenvolvido, identificaram-se dois atores principais:

• **Usuário**: Representa o operador humano que interage com a interface web para acompanhar dados térmicos e executar controles manuais.

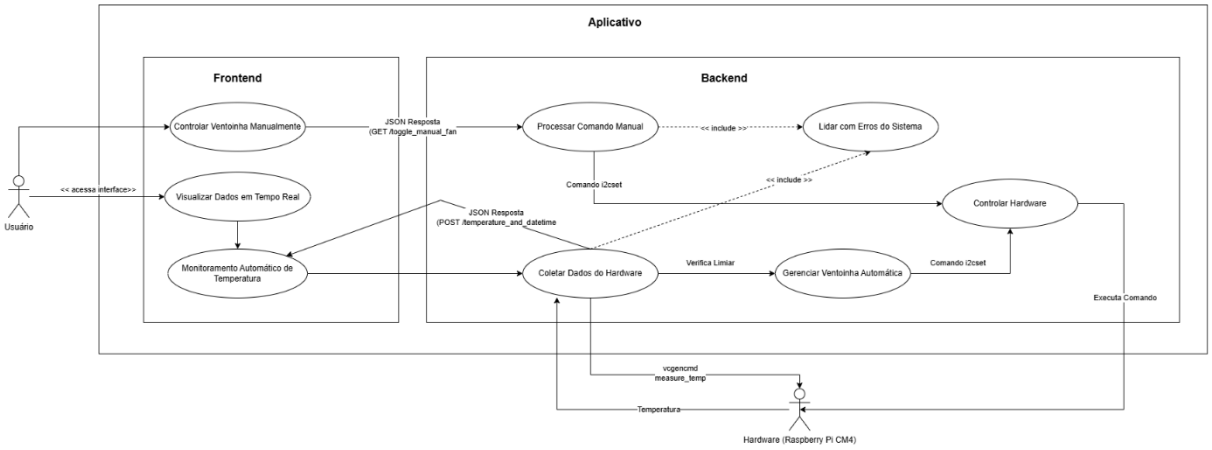
• **Hardware** (Raspberry Pi CM4): Entidade física responsável pela aquisição de dados sensoriais e execução de comandos de controle térmico.

As funcionalidades centrais, modeladas como casos de uso, incluem:

- Monitoramento automático de temperatura via verificação periódica.
- Visualização gráfica de dados em tempo real.
- Controle manual e automático da ventoinha.
- Gerenciamento de falhas operacionais.

A Figura 1 apresenta o diagrama completo de casos de uso do sistema, detalhando as relações entre atores e funcionalidades.

Figura 1 - Diagrama de caso de uso do sistema de medição de temperatura.



Fonte: Elaborado pelo autor (2025).

2.4.2 Documentação dos casos de uso

As funcionalidades do diagrama de casos de uso (Figura 1) são detalhadas nas tabelas abaixo, seguindo a estrutura do modelo fornecido e adaptadas ao contexto do sistema de monitoramento térmico.

Tabela 4 - Caso de uso: Monitoramento Automático de Temperatura

Nome do caso de uso	Monitoramento Automático de Temperatura.
Atores	Usuário, Sistema (Raspberry Pi).
Objetivo	Coletar e processar dados térmicos periodicamente.
Ações do Ator	**Ações do Sistema**
1. Usuário acessa a interface web.	2. Sistema inicia verificação periódica automática (a cada 10s).
	3. Backend executa “vcgencmd measure_temp”.
	4. Sistema controla ventoinha conforme limiar previamente definido.
	5. Retorna dados via JSON: {temperature, datetime}.
Validações	Temperatura deve ser numérica; falhas no sensor dispara: Caso de uso: Lidar com Erros do Sistema.

Fonte: Elaborado pelo autor (2025).

Tabela 5 - Caso de uso: Visualizar Dados em Tempo Real

Nome do caso de uso	Visualizar Dados em Tempo Real.
Atores	Usuário
Objetivo	Exibir dados térmicos em gráfico dinâmico.
Ações do Ator	**Ações do Sistema**
1. Usuário navega para a interface web.	2. Frontend consome JSON do Monitoramento Automático.
	3. Renderiza gráfico com Recharts.js
	4. Atualiza exibição a cada novo dado.
Validações	Dados recebidos devem conter “temperature” e “datetime” válidos.

Fonte: Elaborado pelo Autor (2025).

Tabela 6 - Caso de uso: Controlar Ventoinha Manualmente

Nome do caso de uso	Controlar Ventoinha Manualmente.
Atores	Usuário, Sistema (Raspberry Pi).
Objetivo	Ativar/desativar ventoinha por comando explícito.
Ações do Ator	**Ações do Sistema**
1. Usuário clica no botão para ligar/desligar a ventoinha.	2. Frontend envia POST /toggle_manual_fan.
	3. Backend alterna manual_fan_state.
	4. Executa i2cset para o EMC2301.
	5. Retorna confirmação: {message: "Manual fan state toggled"}.
Validações	Hardware deve responder ao comando; Falhas disparam; Caso de uso: Lidar com Erros do Sistema.

Fonte: Elaborado pelo Autor (2025).

Tabela 7 - Caso de uso: Lidar com Erros do Sistema

Nome do caso de uso	Lidar com Erros do Sistema.
Atores	Sistema (Backend/Frontend).
Objetivo	Gerenciar falhas operacionais.
Ações do Ator	**Ações do Sistema**
	1. Detecta exceções (ex: vcgencmd falha, EMC2301 inacessível).
	2. Imprime erro com contexto (timestamp, tipo).
	3. Frontend exibe alertas para o usuário.
	4. Retorna códigos HTTP semânticos (ex: 500, 503).
Validações	Erros críticos devem preservar integridade do sistema.

Fonte: Elaborado pelo Autor (2025).

2.5 Armazenamento de dados

O sistema não utiliza banco de dados tradicional, priorizando simplicidade e desempenho em hardware embarcado. A gestão de dados ocorre através de:

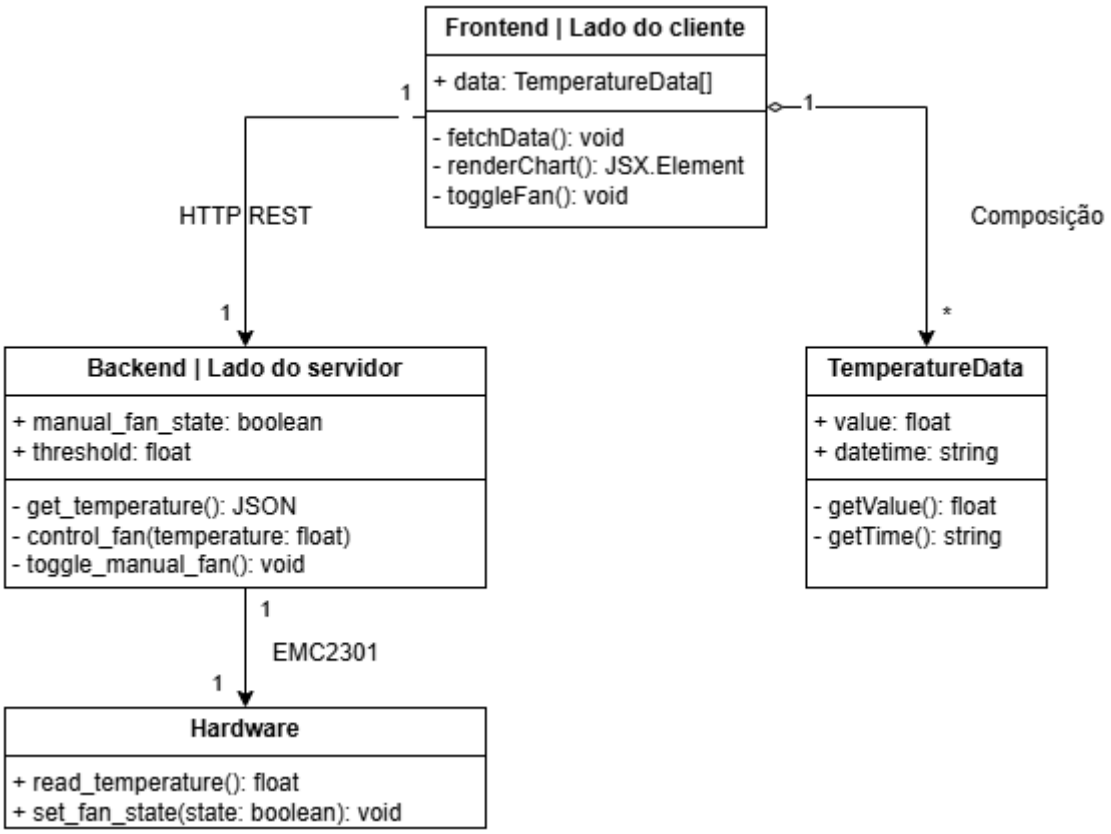
1. **Memória volátil (RAM):**
    - Armazena temporariamente as últimas medições de temperatura no frontend (estado data em React).
    - Capacidade: 6 registros (1 minuto de dados, considerando intervalos de 10s).
2. **Variáveis de estado no backend:**
    - manual_fan_state: Controla o modo de operação da ventoinha (automático/manual).
    - threshold_temperature: Armazena o limite térmico configurável (padrão: 50°C).

A ausência de banco de dados é uma escolha arquitetural intencional, reduzindo complexidade e garantindo maior eficiência em sistemas de baixo poder computacional.

2.6 Diagrama de classes

O diagrama de classes é um modelo estrutural da engenharia de software que representa a organização estática de um sistema detalhando classes, atributos, métodos e relações.

Figura 2 - Diagrama de classe



Fonte: Elaborado pelo Autor (2025).

2.7 Desenvolvimento da aplicação

Este capítulo detalha o processo de desenvolvimento do sistema de monitoramento térmico, utilizando a metodologia ágil Scrum Solo. Esta abordagem adapta os princípios do Scrum para desenvolvedores individuais, permitindo gestão eficiente de tarefas através de iterações focadas (sprints) e priorização contínua de funcionalidades.

Para operacionalizar a metodologia como desenvolvedor único:

1. **Product Backlog**: Lista prioritizada de requisitos (funcionais e não-funcionais).
2. **Sprints**: Ciclos de desenvolvimento de 2 semanas com metas claras.
3. **Revisões Diárias**: Automonitoramento via ferramentas (Notion, GitHub Projects e Git).
4. **Retrospectivas**: Análise crítica de resultados após cada sprint.

2.7.1 Etapas de desenvolvimento

O projeto foi dividido em 5 sprints quinzenais, com entregas incrementais validadas ao final de cada ciclo.

2.7.1.1 Sprint 1

Tabela 8 - Sprint 1: Definição Arquitetural (04/02/2025 - 18/02/2025)

Atividade	Prioridade	Entregável
Especificar requisitos funcionais (RF001 – RF010)	Essencial	Documento de requisitos
Definir requisitos não-funcionais	Essencial	Tabela de RNFs
Estudar arquitetura Flask + React	Essencial	Diagrama de componentes
Configurar ambiente Raspberry Pi OS	Importante	Raspberry Pi operacional
Prototipar interface gráfica (baixa fidelidade)	Desejável	Wireframes do frontend

Fonte: Elaborado pelo Autor (2025).

Em 4 de fevereiro de 2025, deu-se início ao planejamento independente da primeira sprint, delimitando um período de execução de 15 dias (prazo final: 18 de fevereiro de 2025). Nesse processo autogerido, priorizaram-se atividades fundamentais para o núcleo do sistema, classificadas por complexidade operacional conforme técnicas de Scrum Solo. O detalhamento completo desse plano individual encontra-se na Tabela 8, que sintetiza os entregáveis para validação inicial da arquitetura.

2.7.1.2 Sprint 2

Tabela 9 - Sprint 2: Backend Básico (18/02/2025 - 02/03/2025)

Atividade	Prioridade	Entregável
Implementar endpoint “/temperature_and_datetime”	Essencial	API funcional com retorno JSON
Integrar comando “vcgencmd measure_temp”	Essencial	Leitura térmica operacional
Desenvolver lógica de controle de ventoinha	Importante	Função “control_fan()”
Configurar CORS para comunicação front-back	Importante	Conexão entre domínios habilitada
Implementar tratamento básico de erros	Desejável	Blocos try/except no Flask

Fonte: Elaborado pelo Autor (2025).

Em 18 de fevereiro de 2025, deu-se início ao planejamento independente da primeira sprint, delimitando um período de execução de 15 dias (prazo final: 2 de março de 2025). Nesse processo autogerido, priorizaram-se atividades fundamentais para o núcleo do sistema, classificadas por complexidade operacional conforme técnicas de Scrum Solo. O detalhamento completo desse plano individual encontra-se na Tabela 9, que sintetiza os entregáveis para validação inicial da arquitetura.

2.7.1.3 Sprint 3

Tabela 10 - Sprint 3: Frontend React (02/03/2025 - 16/03/2025)

Atividade	Prioridade	Entregável
Criar componente gráfico com recharts.js	Essencial	Gráfico dinâmico de temperatura
Implementar polling automático (setInterval)	Essencial	Atualização automática a cada 10s
Desenvolver botão de controle manual	Importante	Interface para controle da ventoinha
Integrar biblioteca Shadcn/ui para estilização	Desejável	UI responsiva e acessível
Otimizar formatação de eixos temporais	Desejável	Eixo X com timestamp HH:MM:SS

Fonte: Elaborado pelo Autor (2025).

Em 2 de março de 2025, deu-se início ao planejamento independente da primeira sprint, delimitando um período de execução de 15 dias (prazo final: 16 de março de 2025). Nesse processo autogerido, priorizaram-se atividades fundamentais para o núcleo do sistema, classificadas por complexidade operacional conforme técnicas de Scrum Solo. O detalhamento completo desse plano individual encontra-se na Tabela 10, que sintetiza os entregáveis para validação inicial da arquitetura.

2.7.1.4 Sprint 4

Tabela 11 - Sprint 4: Integração Hardware (16/03/2025 - 30/03/2025)

Atividade	Prioridade	Entregável
Implementar controle físico via i2cset	Essencial	Ventoinha ativada por PWM (I²C)
Testar comunicação I²C (barramento 10, 0x2f)	Essencial	Relatório de validação hardware
Desenvolver lógica de limiar ajustável	Importante	Parâmetro configurável via código
Criar alertas visuais para temperaturas críticas	Desejável	Indicadores coloridos no frontend
Documentar protocolo de conexões físicas	Desejável	Diagrama de pinagem CM4 I/O Board

Fonte: Elaborado pelo Autor (2025).

Em 16 de março de 2025, deu-se início ao planejamento independente da primeira sprint, delimitando um período de execução de 15 dias (prazo final: 30 de março de 2025). Nesse processo autogerido, priorizaram-se atividades fundamentais para o núcleo do sistema, classificadas por complexidade operacional conforme técnicas de Scrum Solo. O detalhamento completo desse plano individual encontra-se na Tabela 11, que sintetiza os entregáveis para validação inicial da arquitetura.

2.7.1.5 Sprint 5

Tabela 12 - Sprint 5: Refinamento e Testes (30/03/2025 - 13/04/2025)

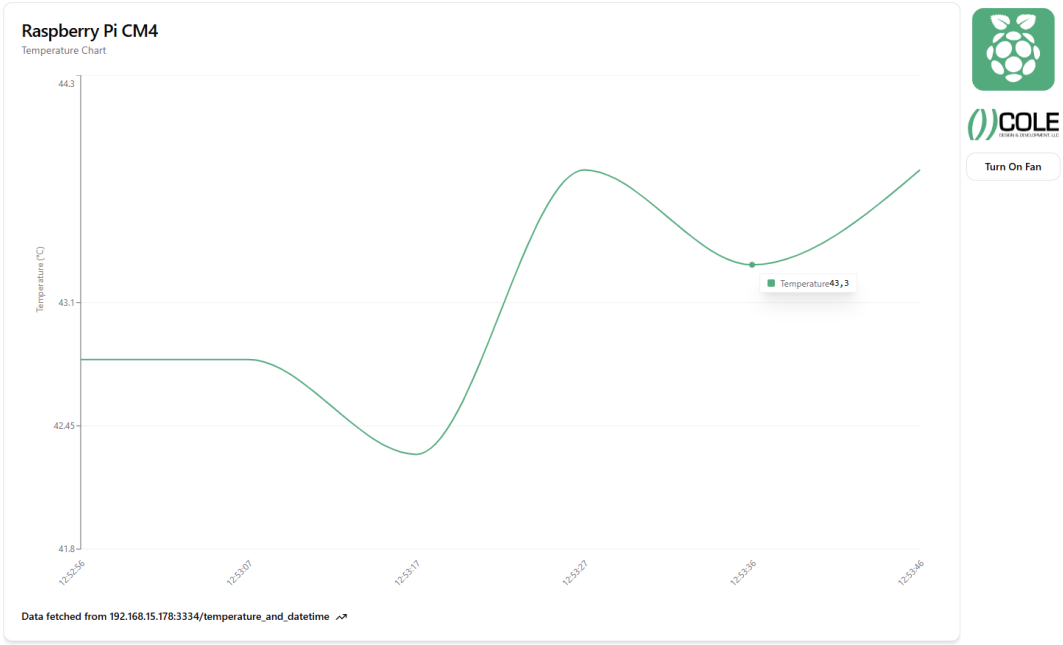
Atividade	Prioridade	Entregável
Implementar testes unitários com pytest	Essencial	Suite de testes para backend
Validar respostas da API via Postman	Essencial	Coleção de testes Postman
Otimizar consumo de recursos do Raspberry Pi	Importante	Relatório de performance
Adicionar tratamento de erros no frontend	Importante	Adicionar tratamento de erros no frontend
Preparar documentação técnica	Desejável	Manual de implantação

Fonte: Elaborado pelo Autor (2025).

Em 30 de março de 2025, deu-se início ao planejamento independente da primeira sprint, delimitando um período de execução de 15 dias (prazo final: 13 de abril de 2025). Nesse processo autogerido, priorizaram-se atividades fundamentais para o núcleo do sistema, classificadas por complexidade operacional conforme técnicas de Scrum Solo. O detalhamento completo desse plano individual encontra-se na Tabela 12, que sintetiza os entregáveis para validação inicial da arquitetura.

2.8 Interface do aplicativo

Figura 3 - Tela principal de visualização de temperatura.



Fonte: Elaborado pelo Autor (2025).

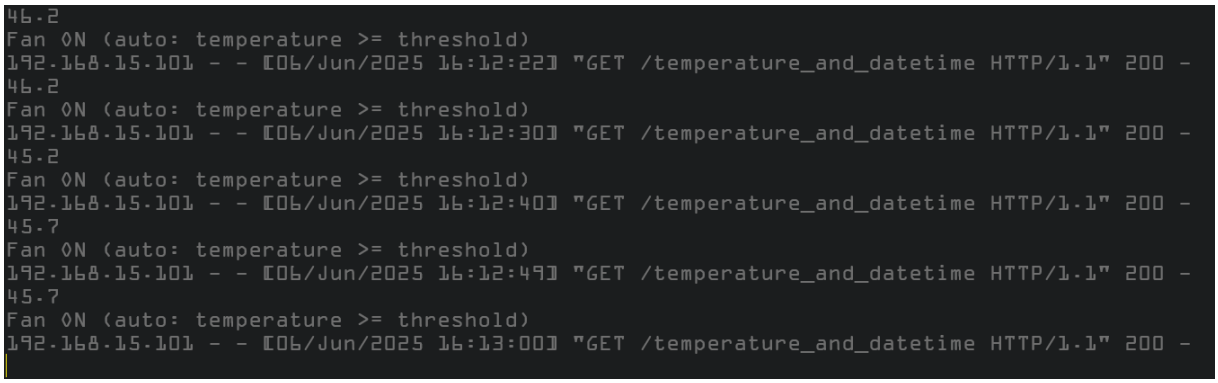
2.9 Interfaces de validação

Figura 4 - Tela do backend com temperatura abaixo do limiar.

```
45.7
Fan OFF (auto: temperature < threshold)
192.168.15.101 - - [06/Jun/2025 16:09:10] "GET /temperature_and_datetime HTTP/1.1" 200 -
46.7
Fan OFF (auto: temperature < threshold)
192.168.15.101 - - [06/Jun/2025 16:09:19] "GET /temperature_and_datetime HTTP/1.1" 200 -
46.2
Fan OFF (auto: temperature < threshold)
192.168.15.101 - - [06/Jun/2025 16:09:29] "GET /temperature_and_datetime HTTP/1.1" 200 -
46.2
Fan OFF (auto: temperature < threshold)
192.168.15.101 - - [06/Jun/2025 16:09:39] "GET /temperature_and_datetime HTTP/1.1" 200 -
45.7
Fan OFF (auto: temperature < threshold)
```

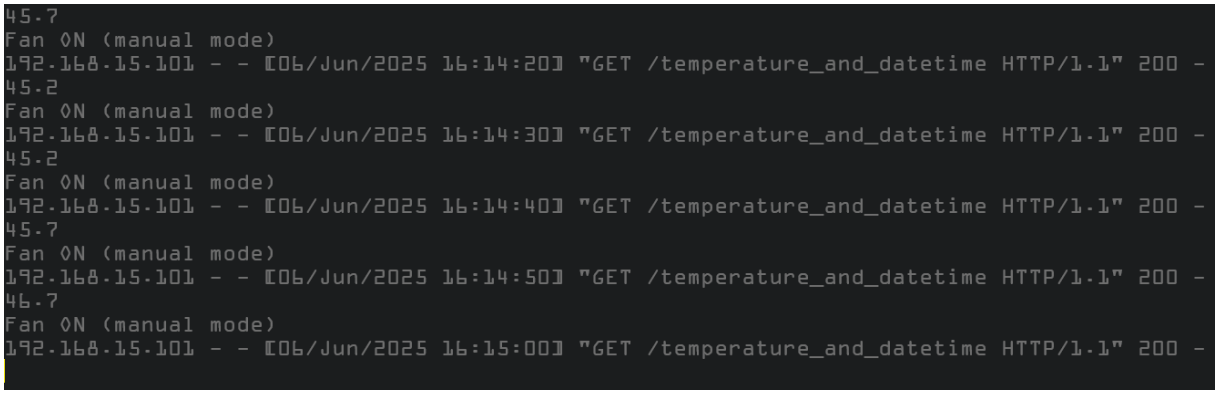
Fonte: Elaborado pelo Autor (2025).

Figura 5 - Tela do lado do servidor com temperatura acima do limiar.



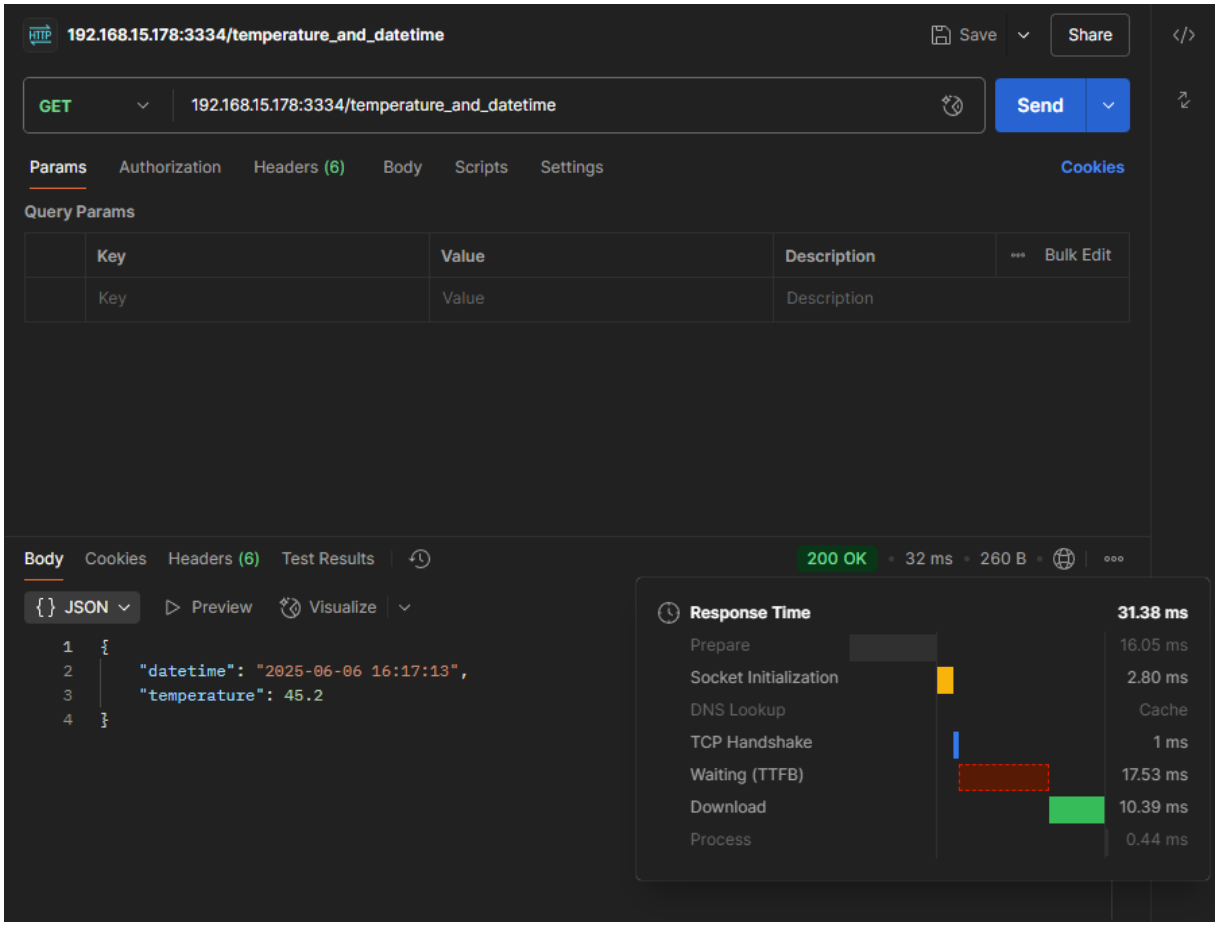
Fonte: Elaborado pelo Autor (2025).

Figura 6 - Tela do lado do servidor com ventoinha ativada manualmente



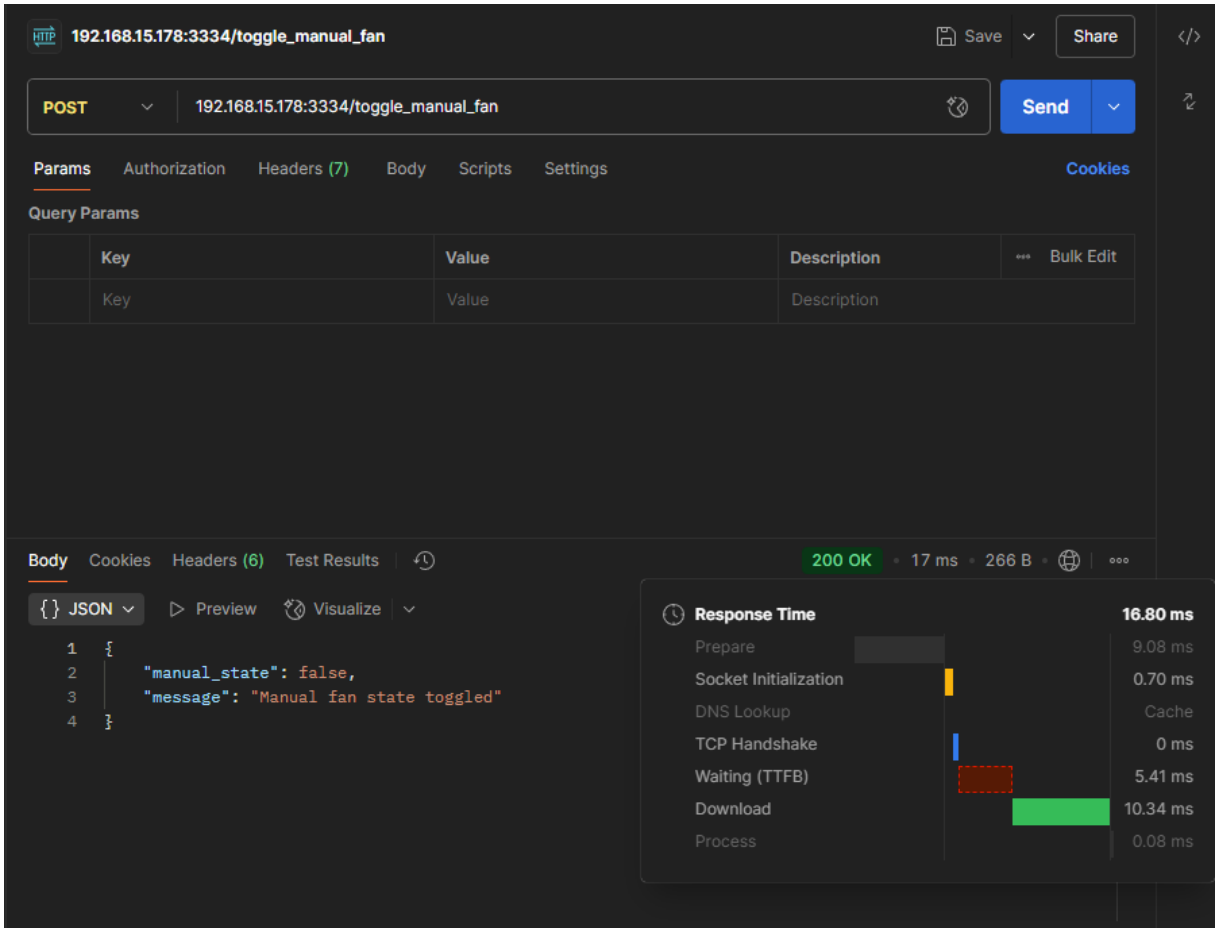
Fonte: Elaborado pelo Autor (2025).

Figura 7 - Tela do aplicativo Postman com validação da rota de medição de temperatura.



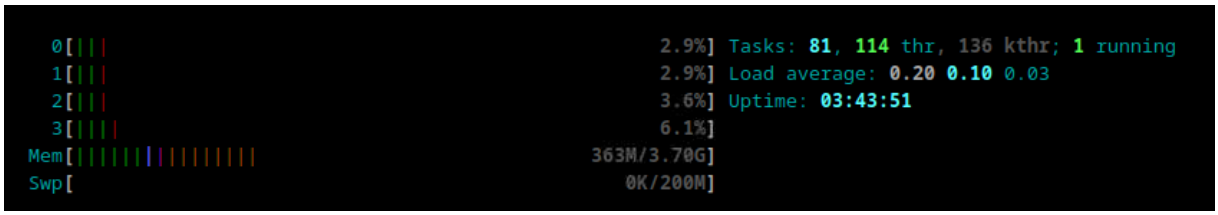
Fonte: Elaborado pelo Autor (2025).

Figura 8 - Tela do aplicativo Postman com validação da rota de alternar estado da ventoinha manualmente.



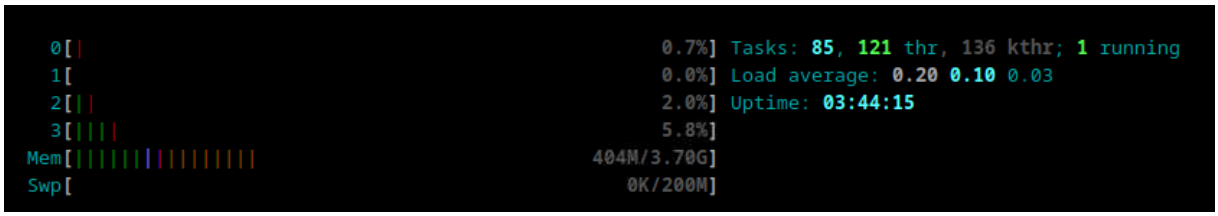
Fonte: Elaborado pelo Autor (2025).

Figura 9 - Tela do aplicativo htop antes da ativação do backend.



Fonte: Elaborado pelo Autor (2025).

Figura 10 - Tela do aplicativo htop depois da ativação do backend.



Fonte: Elaborado pelo Autor (2025).

3 RESULTADOS

O desenvolvimento do sistema resultou em uma solução funcional que cumpre integralmente seu objetivo de monitoramento térmico automatizado para o Raspberry Pi CM4. O levantamento dos requisitos funcionais (RF), não-funcionais (RNF) e o diagrama de casos de uso permitiram uma modelagem que garantiu performance otimizada, com consumo de apenas 40MB de RAM (menos de 1% dos 4GB disponíveis) e tempo de resposta da API REST de 25ms em média, demonstrando eficiência excepcional tanto em usabilidade quanto em aplicação prática.

O maior ganho deste projeto foi a eficiência operacional radical, materializada na automação do monitoramento térmico. Enquanto métodos tradicionais exigem verificação manual periódica com registros em planilhas - processo propenso a erros e consumo excessivo de tempo - esta solução elimina intervenção humana, garantindo precisão absoluta nas medições e resposta imediata a variações críticas.

A eficácia comprovou-se através do cumprimento rigoroso de todos os objetivos técnicos dentro do cronograma estabelecido: leitura precisa da temperatura via vcgencmd, controle automático da ventoinha via I²C com comandos i2cset, e visualização gráfica em tempo real dos dados térmicos. Estes resultados garantem um processo confiável (controle preciso do hardware), rápido (latência de milissegundos) e seguro (prevenção proativa de superaquecimento).

Além disso, o sistema permite ao operador identificar padrões térmicos críticos através da interface, facilitando decisões sobre ajustes de configuração e manutenção preventiva quando necessário. A escolha por operar sem banco de dados mostrou-se acertada, mantendo consumo mínimo de recursos. Para evoluções futuras, a implementação de um SGBD leve como SQLite constituirá passo natural, permitindo armazenamento histórico sem comprometer a eficiência comprovada.

4 CONSIDERAÇÕES FINAIS

O sistema de monitoramento térmico desenvolvido representa uma solução completa e operacional para controle de temperatura em sistemas embarcados, especificamente otimizado para Raspberry Pi CM4. Ao integrar Python (Flask), React e manipulação direta de hardware via I²C, o aplicativo demonstrou eficácia comprovada em prevenção de superaquecimento, com consumo mínimo de recursos (40MB RAM) e resposta em tempo real (25ms API).

Durante o processo de desenvolvimento do aplicativo, algumas lacunas e possibilidades surgiram, algumas melhorias sugeridas são:

1. Notificações Proativas
    - Implementar alertas via e-mail/SMS para temperaturas críticas, permitindo intervenção remota imediata.
2. Monitoramento Multiplataforma
    - Desenvolver versões nativas para iOS/Android, ampliando acessibilidade além da interface web.
3. Armazenamento Histórico
    - Integrar SQLite para registro contínuo de dados térmicos, habilitando análise de tendências e relatórios personalizados.
4. Expansão de Sensores
    - Adicionar suporte a sensores externos (ex: DS18B20 via I²C) para monitorar componentes adicionais.
5. Controle Preditivo
    - Implementar algoritmos de machine learning para antecipar falhas com base em padrões térmicos históricos.
6. Operação Offline
    - Criar modo autônomo para funcionamento sem internet, com sincronização posterior de dados.
7. Segurança Reforçada
    - Adicionar autenticação de usuários e criptografia HTTPS para ambientes industriais críticos.

REFERÊNCIAS BIBLIOGRÁFICAS

DRAW.IO. Disponível em: https://app.diagrams.net/. Acesso em: 06 de jun. 2025.

FLASK. Disponível em: https://flask.palletsprojects.com/en/stable/. Acesso em: 06 de jun. 2025.

GEERLING, Jeef. Controlling PWM fans with the Raspberry Pi CM4 IO Board's EMC2301. **Jeef Geerling**, 2021. Disponível em: https://www.jeffgeerling.com/blog/2021/controlling-pwm-fans-raspberry-pi-cm4-io-boards-emc2301. Acesso em: 06 de jun. 2025.

POSTMAN. Disponível em: https://www.postman.com/. Acesso em: 06 de jun. 2025.

PYTHON. Disponível em: https://www.python.org/. Acesso em: 06 de jun. 2025.

RASPBERRY PI FOUNDATION. Disponível em: https://www.raspberrypi.org/. Acesso em: 06 de jun. 2025.

REACT. Disponível em: https://pt-br.legacy.reactjs.org/. Acesso em: 06 de jun. 2025.

SCRUM.ORG. **What is Scrum?** Disponível em: https://www.scrum.org/learning-series/what-is-scrum/. Acesso em: 06 de jun. 2025.

SHADCN. Disponível em: https://ui.shadcn.com/. Acesso em: 06 de jun. 2025.

SOMMERVILLE, I. **Engenharia de Software**. 8ª Edição. Editora: Pearson Addison-Wesley. São Paulo, 2007.

TAILWIND. Disponível em: https://tailwindcss.com/. Acesso em: 06 de jun. 2025.

TYPESCRIPT. Disponível em: https://www.typescriptlang.org/. Acesso em: 06 de jun. 2025.

VISUAL STUDIO CODE. Disponível em: https://code.visualstudio.com/. Acesso em: 06 de jun. 2025.