**CENTRO PAULA SOUZA**

**FATEC**

# DOCUMENTAÇÃO: SENTINELA VERDE

# Arthur Leite

# David Batista

# Gabriel Alves

# Matheus Henrique

# Maria Eduarda

# Nicolas Ferreira

# Rafael Botelho

Rio Claro – SP 2025

## SUMÁRIO

1. INTRODUÇÃO ...........................................................................................................
2. REQUISITOS DO SISTEMA .....................................................................................
   1. REQUISITOS FUNCIONAIS (RF)..........................................................................
   2. Requisitos não funcionais (RNF) ............................................................

3 ARQUITETURA E DESENVOLVIMENTO...............................................................

3.1 ARQUITETURA GERAL DO SISTEMA

3.2 FLUXO DE DADOS

3.3 METODOLOGIA E GESTÃO DE PROJETO

4 DETALHAMENTO DOS MÓDULOS DE SOFTWARE..................................................

4.1 FIRMWARE DO HARDWARE (arduino.txt)

4.2 CLIENTE DE API EXTERNA (api\_client.py)

4.3 NÚCLEO DE PROCESSAMENTO (backend.py)

4.4 INTERFACE GRÁFICA (frontend.py)

4.5 ORQUESTRADOR DA APLICAÇÃO (main\_app.py)

5 RESULTADOS E DISCUSSÃO

5.1 RESULTADOS OBTIDOS

5.2 DESAFIOS TÉCNICOS ENFRENTADOS

6 GUIA DE INSTALAÇÃO E EXECUÇÃO

7 CONCLUSÃO

REFERÊNCIAS

1. **INTRODUÇÃO**

O projeto sentinela verde surgiu como proposta de um trabalho de conclusão de nota para o primeiro semestre da faculdade Fatec Rio claro, o objetivo do mesmo era ser realizado uma ideia que utilizasse dos conceitos de I.A (inteligência artificial), e que fosse inovador em seu setor ou nicho escolhido. Em nosso respectivo grupo foi pensado diversas ideias para tal, a primeira consistia em um aplicativo gamificado de educação financeira que com o tempo poderia virar um banco ou aplicativo de investimento. Tal ideia foi logo descartada devido a altíssima complexidade e uso dos sistemas de I.O.T (internet das coisas) na qual todas a matérias do curso se juntaram para serem utilizadas no projeto. Após um longo brainstorm e certa atividade que utilizasse as Ods da ONU, pensamos em um projeto que usa se da ods 13 “combate as alterações climáticas”, na qual um sistema consegue prever e medir a qualidade dor ar e seus respectivos poluentes. O seguinte trabalho teve por volta de 2 a 3 meses de desenvolvimento.

1. **REQUISITOS DO SISTEMA**

A seguir, serão listados os requisitos que nortearam o desenvolvimento do sistema assim como o seu devido modo de se comportar.

* 1. Requisitos funcionais (RF)

Os seguintes requisitos servem para listar as tarefas e funcionalidades que o sistema deve executar para ser completo e eficiente.

* **RF01:** O sistema por meio de sensores como MQ-135 e DHT11 deve coletar dados do ambiente que estiver presente assim como a, concentração de gás, temperatura e umidade.
* **RF02:** O sistema deve transmitir os dados coletados para a aplicação de software de forma rápida e eficiente via protocolo MQTT (Message Queuing Telemetry Transport).
* **RF03:** O sistema deve obter dados de fontes de notícias confiáveis, por meio da API WAQI, assim medindo os particulados de PM2.5 e PM10.
* **RF04:** O sistema deve armazenar de forma persistente todos os dados coletados.
* **RF05:** O sistema deve aplicar um modelo de Machine Learning para classificar a qualidade do ar atual.
* **RF06:** O sistema deve aplicar um modelo estatístico para prever as tendências da qualidade do ar para as próximas 24 horas.
* **RF07:** O sistema deve exibir todos os dados e análises em um dashboard interativo.
* **RF08:** O dashboard deve indicar visualmente o nível de qualidade do ar por meio de cores.
* **RF09:** O dashboard deve exibir um gráfico com as previsões de concentração de poluentes.
  1. Requisitos não funcionais (RNF)

Os requisitos não funcionais definem as qualidades e seu padrão de eficácia e funcionamento do sistema.

* **RNF01:** A atualização dos dados no dashboard deve ocorrer em tempo real.
* **RNF02:** A aplicação deve ser responsiva e não deve ser bloqueada por operações de rede.
* **RNF03:** O sistema deve ser desenvolvido em Python, com suas dependências gerenciadas por um arquivo requirements.txt.
* **RNF04:** O firmware deve ser compatível com a plataforma ESP32.
* **RNF05:** O sistema deve gerar logs de eventos e erros.

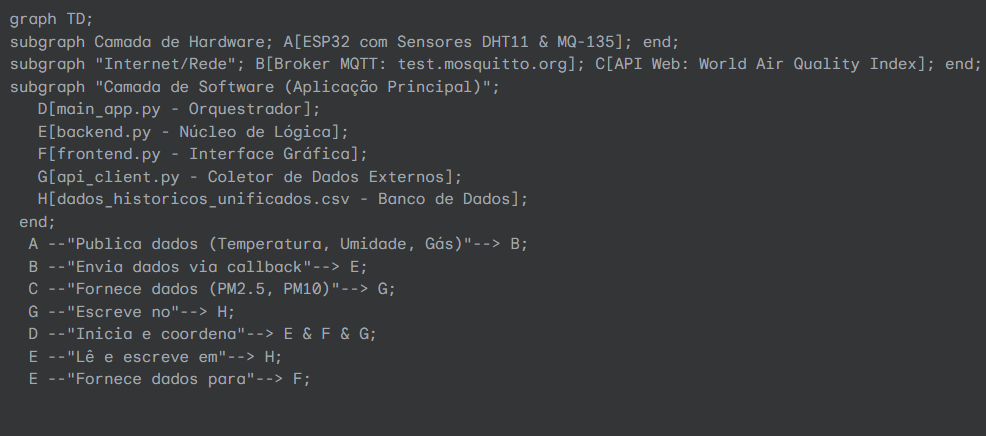
# DESENVOLVIMENTO E APLICAÇÃO

Esta seção descreve a estrutura arquitetônica do sistema e as metodologias empregadas em seu desenvolvimento.

3.1 ARQUITETURA GERAL DO SISTEMA

A arquitetura do Sentinela Verde é modular, composta por camadas de hardware, rede e software. Esta separação de responsabilidades confere ao sistema flexibilidade e escalabilidade.

**Figura 1 – Diagrama da Arquitetura do Sistema Sentinela Verde**



A arquitetura do Sentinela Verde foi projetada em três camadas distintas para garantir modularidade e clareza:

1. Coleta de Dados (Hardware): A base do sistema é um microcontrolador ESP32, responsável por coletar dados do ambiente. Ele utiliza um sensor de gás MQ-135 para medir a concentração geral de gases poluentes (em PPM), e um sensor DHT11/22 para registrar a temperatura e a umidade. Essas informações são enviadas via protocolo MQTT para a aplicação de software, que as integra e armazena.

1. Processamento e Inteligência (Backend): O backend.py é o cérebro da aplicação. Escrito em Python, ele recebe os dados dos sensores e os unifica com medições de material particulado (PM2.5 e PM10) obtidas de uma API externa. A biblioteca Pandas é usada para carregar e validar o conjunto de dados completo. Nesta camada, são aplicadas duas abordagens de classificação da qualidade do ar:

* **Classificação por Regras:** Um sistema transparente que compara as medições com limites pré-definidos no arquivo config.yaml, inspirados em normas de saúde.
* **Classificação por I.A.:** Um modelo de Árvore de Decisão, treinado com a biblioteca Scikit-learn, que identifica padrões nos dados para realizar uma classificação inteligente. Além disso, o backend utiliza a biblioteca Statsmodels para aplicar um modelo de série temporal (Suavização Exponencial) que prevê as concentrações de poluentes para as próximas 24 horas.

1. **Visualização (Frontend):** A interface com o usuário é um dashboard interativo desenvolvido com o framework Flet. O frontend.py se comunica com o backend para solicitar as análises e exibir os resultados de forma gráfica e intuitiva. Painéis e gráficos, gerados com Matplotlib, mostram os dados de monitoramento, o status atual da qualidade do ar e as previsões futuras, com atualização em tempo real.

3.2 FLUXO DE DADOS

O fluxo de informações no sistema é projetado para ser contínuo. O hardware realiza a coleta e a publicação dos dados via MQTT. Concomitantemente, o cliente de API coleta dados externos para o armazenamento. O backend, por sua vez, recebe os dados do MQTT, realiza a unificação com os dados da API, armazena o registro consolidado, executa a análise de I.A. e, ao término do ciclo, notifica o frontend para que este proceda com a atualização da interface gráfica.

3.3 METODOLOGIA DE GESTÃO DO PROJETO

O processo de desenvolvimento foi conduzido em conformidade com a metodologia ágil Scrum. O trabalho foi segmentado em quatro *sprints* quinzenais. A plataforma Trello foi utilizada para o gerenciamento do backlog do produto e para o acompanhamento das tarefas.

Para o desenvolvimento do projeto foi preciso antes pensar na forma que ele captaria os dados e sua linguagem. A língua de programação utilizada foi o Python que está alinhada com as tecnologias de análise de dados e machine learning junto com o algoritmo de árvore de decisão. A realização deste projeto também utilizou da metodologia de projetos ágeis como o Scrum, sendo dividida em 4 sprints onde foi definido objetivos e metas a serem alcançadas, por meio de plataformas de organização como o Trello.

**4 DETALHAMENTO DOS MÓDULOS DE SOFTWARE**

Esta seção fornece uma análise técnica de cada componente de software do sistema, detalhando suas responsabilidades e implementações.

4.1 FIRMWARE DO HARDWARE (arduino.txt)

O firmware, desenvolvido em C++ para a plataforma ESP32, é o componente responsável pela interface direta com o ambiente físico. Sua lógica de operação é contida principalmente nas funções setup() e loop(). Na inicialização (setup), o dispositivo estabelece a conexão com a rede Wi-Fi e configura o cliente MQTT (Message Queuing Telemetry Transport) para se conectar ao broker público test.mosquitto.org. O loop principal (loop) é executado continuamente; a cada 60 segundos, ele realiza a leitura do sensor de temperatura e umidade DHT11 e do sensor de gás MQ-135. O valor analógico do MQ-135 é convertido para uma estimativa em Partes por Milhão (PPM) e, juntamente com os demais dados, é publicado em um tópico MQTT dedicado. O código também implementa uma rotina de reconexão automática, garantindo a resiliência da comunicação em caso de falhas na rede.

4.2 CLIENTE DE API EXTERNA (api\_client.py)

Este módulo em Python opera como um coletor de dados autônomo. Sua função primordial é enriquecer as medições locais com dados de referência de fontes externas. A função fetch\_air\_quality\_data utiliza a biblioteca requests para realizar uma chamada HTTP à API do World Air Quality Index (WAQI), buscando especificamente os níveis de material particulado PM2.5 e PM10 para a cidade de Rio Claro. O tratamento de erros de conexão e de respostas inválidas da API está implementado para garantir robustez.

Os dados obtidos são, então, processados pela função append\_api\_data\_to\_csv, que os anexa, juntamente com um timestamp atual, ao arquivo dados\_historicos\_unificados.csv, mantendo assim a base de dados histórica sempre atualizada.

4.3 NÚCLEO DE PROCESSAMENTO (backend.py)

O backend.py representa o núcleo lógico e analítico do sistema. A classe central, SentinelaVerde, orquestra o fluxo de processamento de dados. O módulo inclui um MQTTClient que se inscreve no tópico do hardware e, ao receber uma mensagem, aciona o método process\_mqtt\_message. Este método, por sua vez, decodifica os dados e inicia o ciclo de análise. O ciclo consiste em: i) unificar os dados recebidos com os dados da API; ii) persistir o registro completo no arquivo CSV; iii) carregar o conjunto de dados histórico completo; iv) treinar o modelo de classificação (Árvore de Decisão da biblioteca scikit-learn) e o modelo de previsão (Suavização Exponencial da biblioteca statsmodels); e v) armazenar os resultados. Ao final, ele traz uma função de callback para notificar o frontend sobre a disponibilidade de novos dados.

4.4 INTERFACE GRÁFICA (frontend.py)

Desenvolvido com o framework Flet, este módulo é exclusivamente responsável pela camada de apresentação. Ele constrói a interface gráfica do usuário (GUI), que é composta por um dashboard interativo organizado em "cards". Cada card exibe uma métrica específica, como a concentração de gás, PM2.5, PM10, temperatura e umidade. O principal componente de feedback visual é o painel de "Qualidade do Ar", que altera dinamicamente sua cor e ícone com base na classificação fornecida pelo backend.

Para a exibição de gráficos de previsão, foi implementada a função gerar\_imagem\_grafico\_base64, que utiliza a biblioteca Matplotlib responsável por renderizar o gráfico em memória, codificá-lo em Base64 e inseri-lo em um componente de imagem do Flet. A atualização da interface é reativa, ocorrendo apenas quando o frontend é notificado pelo backend através do mecanismo de callback.

4.5 ORQUESTRADOR DA APLICAÇÃO (main\_app.py)

O script main\_app.py constitui o ponto de entrada (entrypoint) da aplicação, e sua responsabilidade primária é a orquestração dos componentes de software e o gerenciamento da concorrência. A arquitetura implementada representa uma evolução significativa em relação a um modelo de leitura de arquivo estático, adotando o protocolo MQTT para comunicação sem fio em tempo real.

Para gerenciar a natureza concorrente das tarefas — como a renderização da interface gráfica e a escuta de mensagens de rede —, o sistema emprega multithreading. A biblioteca threading é utilizada para iniciar fluxos de execução paralelos: a thread principal gerencia a interface do Flet, mantendo-a responsiva, enquanto threads secundárias são delegadas para o cliente MQTT e para o coletor de dados da API. Esta última utiliza a biblioteca schedule para execuções periódicas. Certa passagem de tarefas de rede a threads de segundo plano é fundamental para atender ao requisito de não bloqueio (RNF02), garantindo que a experiência do usuário permaneça fluida e que a interface não congele durante operações de I/O.

**5 RESULTADOS E DISCUSSÃO**

Esta seção apresenta os resultados práticos obtidos com o protótipo funcional do sistema e discute os principais desafios técnicos que foram superados durante o ciclo de desenvolvimento.

5.1 RESULTADOS OBTIDOS

O desenvolvimento do projeto resultou em um protótipo funcional que atende a todas as demandas esperadas e atribuídas ao mesmo. Os resultados mais significativos e interessantes são:

* **Sistema de Monitoramento Integrado:** Foi implementado com êxito um sistema coeso capaz de capturar, unificar e armazenar dados de múltiplas fontes assíncronas (sensores locais via MQTT e API externa via HTTP).
* **Dashboard Operacional e em Tempo Real:** A interface gráfica se mostrou funcional, exibindo os dados de forma clara e atualizando-se em tempo real após o processamento de novas medições, fornecendo um feedback visual imediato sobre a qualidade do ar.
* **Validação dos Modelos de I.A.:** O sistema demonstrou a viabilidade de aplicar modelos de machine learning em um fluxo contínuo de dados. O modelo de classificação oferece um diagnóstico imediato, enquanto o modelo de previsão fornece uma estimativa de tendências, agregando valor analítico à solução.
* **Comprovação da Arquitetura:** A arquitetura proposta, baseada em comunicação desacoplada via MQTT e processamento concorrente com multithreading, provou ser uma abordagem eficaz para a construção de uma aplicação de IoT responsiva e robusta.

5.2 DESAFIOS TÉCNICOS ENFRENTADOS

Durante a construção e idealização, a equipe passou por diversos desafios técnicos e lógicos como:

A integração de fontes de dados assíncronas vindas do MQTT (frequentes e em tempo real) com dados da API (periódicos e menos frequentes) exigiu esforço no desenvolvimento de uma lógica para o preenchimento de valores ausentes (forward fill), a fim de garantir a integração dos conjuntos de dados utilizados para treinar os modelos.

A calibração de sensores de baixo custo como o sensor MQ-135, que por sua natureza, mede uma gama geral de gases. A obtenção de leituras precisas para poluentes específicos demandou uma pesquisa e estudo melhor dos sensores na qual não sabíamos direito como configurar e como obter seus dados.

O desenvolvimento do código responsável pelo backend também foi um dos grandes desafios enfrentado pelo grupo, visto que o desenvolvimento da árvore de decisão necessitou um aprofundamento sobre machine learning e seu modo de funcionar. Outro desafio foi a integração com ESP32 visto que a sua linguagem é o C++, logo foi preciso entender melhor e pesquisar para que assim pudéssemos desenvolver seus códigos e assim chegarmos aos resultados esperados.

Assim como o backend, o frontend foi um grande desafio visto que foi neste projeto que houve um primeiro contato com ele. Certos softwares como Figma foram utilizados de início, porém devido a certa complexidade foi abandonada e utilizamos de outros métodos para o desenvolvimento da dashboard e gráficos. Além disto a integração por parte do código backend foi um processo desafiador assim como a integração com a API, para conseguir consertar foi utilizado do auxílio de inteligências artificiais que corrigiam e revelavam onde e como consertar os erros.

**6 GUIA DE INSTALAÇÃO E EXECUÇÃO**

Esta seção será responsável por guiar como replicar o ambiente de desenvolvimento e a execução do sistema Sentinela Verde. O processo está dividido em duas etapas principais: a configuração do componente de hardware e a preparação do ambiente de software.

Para a correta implementação, são necessários pré-requisitos de hardware e software. No que tange ao hardware, é imprescindível dispor de uma placa de desenvolvimento ESP32, dos sensores DHT11 e MQ-135, e de componentes eletrônicos associados para a montagem do circuito. Os pré-requisitos de software incluem a presença de um interpretador Python (versão 3.9 ou superior), do sistema de controle de versão Git e da IDE do Arduino com as bibliotecas DHT sensor library, Adafruit Unified Sensor e PubSubClient devidamente instaladas.

O procedimento de configuração do componente de hardware inicia-se com a abertura do código-fonte do firmware (arduino.txt) na IDE do Arduino. Neste arquivo, devem ser modificadas as variáveis ssid e password para que correspondam às credenciais da rede Wi-Fi local. Subsequentemente, o código é compilado e carregado na placa ESP32. A verificação do sucesso da operação pode ser realizada por meio do Monitor Serial da IDE, que exibirá o status da conexão e a confirmação da publicação dos primeiros dados via MQTT.

Paralelamente, a configuração do ambiente de software em um computador local requer, primeiramente, a clonagem do repositório do projeto utilizando o Git. Uma vez clonado, é uma prática recomendada navegar até o diretório raiz do projeto e criar um ambiente virtual Python, a fim de isolar as dependências. Após a criação e ativação deste ambiente, todas as bibliotecas necessárias são instaladas de forma automatizada, por meio da execução do comando pip install -r requirements.txt. Este comando lê o arquivo de requisitos e instala todas as dependências listadas, como Flet, Pandas, Scikit-learn e Paho-MQTT.

Finalizadas as etapas de configuração, a execução da aplicação é iniciada. É fundamental assegurar que o dispositivo ESP32 esteja energizado e transmitindo dados. Com o ambiente virtual Python ativado no terminal, o orquestrador principal da aplicação é invocado pelo comando python main\_app.py. Como resultado, a interface gráfica do Sentinela Verde será iniciada, realizando um primeiro ciclo de análise e passando a aguardar novas medições para proceder com as atualizações em tempo real.

**7 CONCLUSÃO**

Por fim o projeto Sentinela Verde alcançou com êxito os seus objetivos esperados, culminando em um protótipo funcional que atesta a viabilidade de um sistema de monitoramento da qualidade do ar de baixo custo, inteligente e em tempo real. A arquitetura modular e a adoção de tecnologias de código aberto demonstraram ser adequadas aos requisitos definidos, resultando em uma solução robusta e escalável. Conclui-se que o sistema não apenas satisfaz os requisitos acadêmicos, mas também se apresenta como uma ferramenta com significativo potencial de aplicação prática para a conscientização e a gestão ambiental.

**REFERÊNCIAS**

FLET. **Flet: Build Flutter apps in Python**. Disponível em: <https://flet.dev/>. Acesso em: 12 jun. 2025.

OPEN KNOWLEDGE FOUNDATION. **World Air Quality Index Project**. Disponível em: <https://aqicn.org/api/>. Acesso em: 12 jun. 2025.

PAHO. **paho-mqtt: The official Eclipse Paho MQTT Python client library**. Disponível em: <https://pypi.org/project/paho-mqtt/>. Acesso em: 12 jun. 2025.

VAN DER WALT, S.; COLBERT, S. C.; VAROQUAUX, G. The NumPy array: a structure for efficient numerical computation. **Computing in Science & Engineering**, v. 13, n. 2, p. 22-30, 2011.

WES, McKinney. Data Structures for Statistical Computing in Python. In: **PROCEEDINGS OF THE 9TH PYTHON IN SCIENCE CONFERENCE**. Austin, TX, 2010. p. 51-56.