

ProjetoISlv1 - Documentação Completa

Monitorização Industrial com ETL & IoT Simulator Platform

Table of contents

1. Home	3
1.1 ProjetoISiv1 - Monitorização Industrial com ETL e IoT	3
1.2 Contexto e Evolução do Projeto	5
2. Arquitetura	9
2.1 Arquitetura Técnica do Projeto	9
2.2 ETL PROCESSOS	16
2.3 Processos ETL e Transformações	16
2.4 Base de Dados	17
3. Integrações	24
3.1 Integração com API de Preços	24
3.2 Integração Azure IoT Hub	30
3.3 Relatórios e Emails Automáticos	40
4. Simulador IoT	45
4.1 MQTT Simulator	45
4.2 Configuration	47
4.3 MQTT Simulator - Data type <code>math_expression</code>	52
5. Interface	55
5.1 Dashboard de Visualização	55
6. Deployment	61
6.1 Pré-requisitos e Instalação	61
6.2 Troubleshooting	62
7. Projeto	63
7.1 Conclusão e Trabalhos Futuros	63
7.2 Roadmap Completo - Plataforma de Simulação IoT	64

1. Home

1.1 ProjetoISlv1 - Monitorização Industrial com ETL e IoT

Bem-vindo à documentação do **ProjetoISlv1**, um sistema de monitorização industrial que evoluiu para uma plataforma de simulação IoT.

1.1.1 Sobre o Projeto

Este projeto nasceu como trabalho prático da UC de **Integração de Sistemas de Informação (ISI)** da LESI-IPCA e cresceu para algo maior.

Fase 1: Trabalho Académico (Out-Dez 2024)

Sistema de monitorização industrial com recolha automática de dados de estações de produção via MQTT, processamento ETL em Node-RED, e visualização em tempo real.

Objetivo: Aplicar conceitos de integração de sistemas e ETL num contexto industrial prático.

Fase 2: Plataforma IoT Simulator (Jan 2025+)

Evolução para uma plataforma comercial que permite a programadores testar aplicações IoT sem hardware físico, criando sensores virtuais sob demanda.

Objetivo: Validar viabilidade comercial de uma solução SaaS para simulação IoT.

Lê mais: [Contexto e Evolução do Projeto](#)

1.1.2 Stack Tecnológica

Backend: Python, Node-RED, SQLite, Mosquitto MQTT

Frontend: React, Material-UI

DevOps: Docker, FastAPI (Fase 2)

1.1.3 Estrutura da Documentação

Conceito e Arquitetura

- [Contexto do Projeto](#) - Evolução Fase 1 → Fase 2
- [Arquitetura Técnica](#) - Componentes e fluxo de dados
- [Processos ETL](#) - Pipeline de dados

Dados e Integrações

- [Base de Dados](#) - Schema e queries principais
- [Integração API](#) - Preços de peças em tempo real
- [Azure IoT Hub](#) - Telemetria para cloud
- [Relatórios e Emails](#) - Automação de relatórios

Implementação

- [Instalação](#) - Setup do ambiente

- [Dashboard](#) - Interface de visualização
- [Troubleshooting](#) - Problemas comuns

Futuro

- [Fase 2 - IoT Simulator Platform](#) - Roadmap comercial

1.1.4 Quick Start

```
# 1. Clonar repositório
git clone https://github.com/PEQSPC/ProjetoISiv1
cd ProjetoISiv1

# 2. Iniciar broker MQTT
mosquitto -c mosquitto.conf

# 3. Iniciar Node-RED
node-red

# 4. Executar simulador
python simulador/mqtt_publisher.py

# 5. Ver dashboard
cd dashboard && npm start
```

Detalhes completos em [Instalação](#).

1.1.5 O Problema Resolvido

- Antes:** Recolha manual de dados, sem visão integrada, dificuldade em identificar problemas.
- Depois:** Monitorização automática em tempo real, alertas automáticos, relatórios por email, dashboard interativo.
- Métricas:** Produção, paragens, stock, defeitos, eficiência (OEE).

1.1.6 Navegação Recomendada

- Novo no projeto?** 1. [Contexto do Projeto](#) - Entende a visão 2. [Instalação](#) - Configura o ambiente 3. [Dashboard](#) - Vê os dados ao vivo
- Implementador técnico?** 1. [Arquitetura](#) - Compreende o sistema 2. [ETL](#) - Fluxos de processamento 3. [Base de Dados](#) - Modelo de dados
- Interessado no futuro?** 1. [Fase 2 Roadmap](#) - Plataforma comercial

1.1.7 Informação do Projeto

- Instituição:** IPCA - Instituto Politécnico do Cávado e do Ave
- Curso:** Licenciatura em Engenharia de Sistemas Informáticos (LESI)
- UC:** Integração de Sistemas de Informação (ISI)
- Ano Letivo:** 2024/25
- Equipa:** PEQSPC
- Repositório:** github.com/PEQSPC/ProjetoISiv1

Nota: Esta documentação cobre ambas as fases do projeto. Para detalhes da Fase 2 (plataforma comercial), consulta o [roadmap específico](#).

1.2 Contexto e Evolução do Projeto

Esta página explica a jornada do ProjetoISV1 desde um trabalho acadêmico até uma potencial plataforma comercial.

1.2.1 Fase 1: Trabalho Acadêmico ISI (Out-Dez 2024)

Objetivo Original

Aplicar conceitos de **Integração de Sistemas de Informação** num cenário prático de monitorização industrial, focando em processos ETL e comunicação IoT.

Cenário Implementado

Sistema de monitorização de linha de produção com 3 estações que reportam: - Produção (peças fabricadas) - Stock (inventário atual) - Paragens (tempo de inatividade) - Defeitos (peças com problemas)

Arquitetura Fase 1

```
Sensores Simulados (Python MQTT)
  |
Broker MQTT (Mosquitto)
  |
Node-RED (ETL Pipeline)
  |
SQLite Database
  |
Dashboard React + Relatórios Email
```

Componentes Desenvolvidos

- Simulador IoT** - Gera dados sintéticos de sensores via MQTT
- Node-RED Flows** - Processa dados (adiciona timestamps, acumula métricas)
- Integração API** - Enriquece dados com preços de peças externas
- Base de Dados** - Armazena histórico de produção
- Relatórios Automáticos** - Python scripts enviam emails com KPIs
- Dashboard Web** - Visualização em tempo real (React + Material-UI)

Aprendizagens Chave

Técnicas: - Comunicação MQTT em ambiente industrial - Padrão publish-subscribe para desacoplamento - ETL com Node-RED (Extract-Transform-Load) - Sidecar pattern para isolamento de simulações - Persistência de dados em SQLite

Conceituais: - IoT testing requer simuladores robustos - Existe gap no mercado de simulação IoT acessível - Arquitetura local funciona mas não escala para produção - Necessidade de multi-tenancy e isolamento

Limitações Identificadas

- SQLite não adequado para produção multi-utilizador
- Docker local não permite acesso remoto/partilhado
- Falta de autenticação e gestão de utilizadores
- Impossível escalar para múltiplos clientes simultâneos
- Sem API REST para gestão programática

1.2.2 Transição: Da Monitorização à Simulação

Insight Crítico

Durante o desenvolvimento, percebemos:

"Se conseguimos **simular sensores** para testar nosso sistema de monitorização, outros programadores também precisam dessa capacidade. Mas pagar €500+/mês por IoTIFY ou Bevywise não faz sentido para startups."

Oportunidade de Mercado

Problema Real: Programadores precisam testar apps IoT sem comprar hardware.
Soluções Existentes: Caras (€500+/mês) ou complexas (AWS IoT Device Simulator).
Nossa Proposta: Simulação IoT simples e acessível, pay-as-you-go.
Mercado: €714B de IoT em 2025, startups e consultoras são target inicial.

1.2.3 Fase 2: IoT Simulator Platform (Jan 2025+)

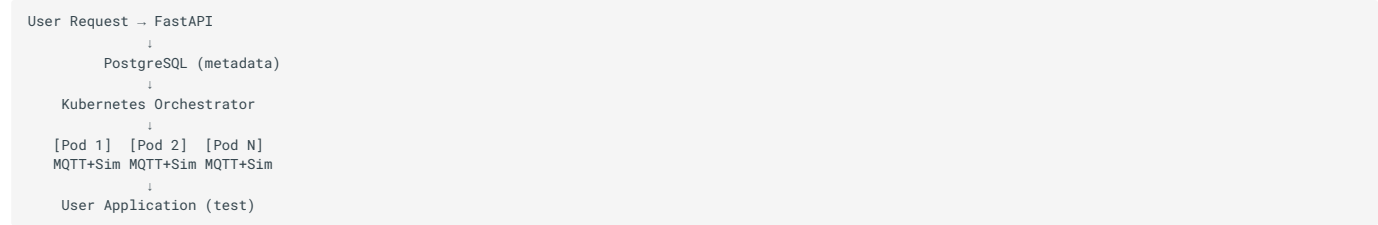
Visão

Plataforma SaaS que permite criar sensores IoT virtuais via API REST, eliminando necessidade de hardware para testes e desenvolvimento.

Transformações Necessárias

Aspecto	Fase 1 (Académico)	Fase 2 (Comercial)
Deploy	Docker local	Kubernetes cloud
DB	SQLite	PostgreSQL
API	Inexistente	FastAPI REST
Auth	Nenhuma	JWT tokens
Preço	Gratuito	Pay-as-you-go
Multi-tenant	Não	Sim (pods isolados)
Escala	1 utilizador	Centenas

Nova Arquitetura (Fase 2)



Conceito-chave: Cada simulação corre num pod isolado com MQTT broker + simulador próprios.

Funcionalidades Planeadas

MVP (3 meses): - Criar/parar simulações via API - Configurar sensores (tipo, frequência, dados) - MQTT isolado por simulação - Monitorização básica de recursos - Pay-as-you-go €0.10/hora

Pós-MVP: - Templates de sensores pré-configurados - Webhooks para eventos - Simulação de cenários complexos - Dashboard de gestão de simulações - Integrações (Azure IoT Hub, AWS IoT Core)

Roadmap Técnico

Ver detalhes completos em [Fase 2 Roadmap](#).

- Q1 2025: MVP funcional, validação de clientes (10 early adopters)
- Q2 2025: Beta pública, landing page, primeiros clientes pagantes
- Q3 2025: Produção estável, features avançadas
- Q4 2025: Escala para 100+ utilizadores

1.2.4 Como a Fase 1 Informou a Fase 2

Validações Técnicas

- 1. Sidecar pattern funciona - Cada simulação isolada num container
- 2. MQTT é protocolo certo - Amplamente adotado em IoT
- 3. Simplicidade vende - Configuração JSON simples é suficiente
- 4. Cleanup automático é crítico - Simulações devem expirar automaticamente

Decisões Arquiteturais

Da experiência Fase 1, decidimos:

- Kubernetes em vez de Docker Compose - Melhor orquestração
- PostgreSQL em vez de SQLite - Suporta concorrência
- FastAPI em vez de Node-RED - Mais controlo programático
- Redis para rate limiting - Evitar abuso da API
- Prometheus/Grafana - Monitorização desde o início

Lições de Produto

- O que funcionou: - Configuração via JSON é intuitiva - Dados sintéticos realistas são valiosos - Visualização em tempo real é esperada
- O que faltou: - Gestão de múltiplas simulações - Autenticação e billing - API para automação - Templates reutilizáveis

1.2.5 Estado Atual (Dez 2024)

Fase 1: ☒ Completa

- Código funcional e documentado
- Apresentação académica aprovada
- Repositório público disponível

Fase 2: ☐ Em Desenvolvimento

- Completado: - Pesquisa de mercado (€714B TAM, gap identificado) - Arquitetura técnica definida - MVP com FastAPI + Docker funcionando - Error handling robusto implementado
- Em Progresso: - Transição para Kubernetes - Sistema de billing - Landing page para validação
- Próximos Passos: - Entrevistas com potenciais clientes - Load testing e isolamento - Deploy em cloud provider - Beta privada com early adopters

1.2.6 Recursos Relacionados

Fase 1: - [Arquitetura Técnica](#) - [Node-RED Flows](#) - [Dashboard](#)

Fase 2: - [IoT Simulator Platform Roadmap](#) - [Repositório API](#) (em construção) - [Landing Page](#) (em construção)

1.2.7 Conclusão

O ProjetoSlv1 começou como exploração acadêmica de ETL e IoT, mas revelou uma oportunidade real de mercado. A experiência técnica da Fase 1 validou conceitos fundamentais que agora evoluem para uma plataforma comercial viável.

Fase 1 ensinou-nos como construir sistemas IoT integrados.

Fase 2 aplica esse conhecimento para resolver um problema real de mercado.

Esta documentação serve ambas as fases, mantendo o histórico acadêmico enquanto suporta o desenvolvimento comercial.

2. Arquitetura

2.1 Arquitetura Técnica do Projeto

A arquitetura do **ProjetoISiv1** implementa um sistema distribuído que suporta todo o ciclo de vida dos dados industriais: **recolha, processamento, armazenamento e visualização**.

2.1.1 Visão Geral da Arquitetura

O sistema segue uma arquitetura orientada a eventos (Event-Driven Architecture) baseada no protocolo MQTT, permitindo comunicação assíncrona e em tempo real entre os componentes.

Princípios Arquiteturais

- **Desacoplamento:** Componentes comunicam através de mensagens MQTT
- **Escalabilidade:** Arquitetura permite adicionar novos sensores sem alterações
- **Modularidade:** Cada componente tem responsabilidades bem definidas
- **Persistência:** Dados armazenados de forma estruturada e auditável
- **Extensibilidade:** Fácil integração com novos sistemas e APIs

2.1.2 Componentes do Sistema

1. Camada de Recolha de Dados

1.1 PUBLISHER - SIMULADOR MQTT (PYTHON)

Responsabilidade: Simular sensores IoT em ambiente industrial

Funcionalidades:

- Geração de dados sintéticos de produção, stock, paragens e defeitos
- Publicação de mensagens JSON em tópicos MQTT específicos por estação
- Configuração flexível através de ficheiros JSON
- Suporte para múltiplos dispositivos e cenários de simulação

Tecnologias: Python 3.10+, biblioteca `paho-mqtt`

Tópicos MQTT:

```
fabrica/estacao/{id}/producao
fabrica/estacao/{id}/stock
fabrica/estacao/{id}/paragem
fabrica/estacao/{id}/defeitos
```

1.2 INTEGRAÇÃO AZURE IOT (FASE 2)

Funcionalidades:

- Conexão com Azure IoT Hub
- Envio de telemetria para a cloud
- Sincronização device-to-cloud

2. Camada de Mensagens

2.1 BROKER MQTT - ECLIPSE MOSQUITTO

Responsabilidade: Intermediário de mensagens entre publishers e subscribers

Características:

- Protocolo MQTT v3.1.1 / v5.0
- Porta padrão: 1883 (não encriptado) / 8883 (TLS)
- Suporte para QoS (Quality of Service) níveis 0, 1 e 2
- Persistência de mensagens opcional

Configuração:

```
listener 1883
allow_anonymous true
persistence true
persistence_location /var/lib/mosquitto/
```

3. Camada de Processamento (ETL)

3.1 NODE-RED - MOTOR ETL

Responsabilidade: Orquestração de fluxos de dados e transformações

Pipeline de Processamento:

1. **Extract (Extração)**
2. Subscrição a tópicos MQTT
3. Receção de payloads JSON
4. Parsing e validação de dados
5. **Transform (Transformação)**
6. Normalização de valores
7. Cálculo de acumulados e agregações
8. Filtragem de outliers
9. Aplicação de regras de negócio
10. Enriquecimento com dados da API
11. **Load (Carregamento)**
12. Inserção em SQLite3
13. Atualização de registos existentes
14. Trigger de jobs de relatório

Nós Principais:

- `mqtt in`: Receção de mensagens
- `function`: Lógica JavaScript para transformações
- `sqlite`: Operações de base de dados
- `exec`: Execução de scripts Python
- `email`: Envio de relatórios

4. Camada de Persistência

4.1 BASE DE DADOS SQLITE3

Responsabilidade: Armazenamento estruturado de dados processados

Estrutura de Tabelas:

```
-- Tabela de Estações
CREATE TABLE estacoes (
  id INTEGER PRIMARY KEY,
  nome TEXT NOT NULL,
  localizacao TEXT,
  tipo_producao TEXT
);

-- Tabela de Dados Acumulados
CREATE TABLE dados_acumulados (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  estacao_id INTEGER,
  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  producao_total INTEGER,
  stock_atual INTEGER,
  tempo_paragem_segundos INTEGER,
  defeitos_total INTEGER,
  FOREIGN KEY (estacao_id) REFERENCES estacoes(id)
);

-- Tabela de Produtos
CREATE TABLE produtos (
  id INTEGER PRIMARY KEY,
  nome TEXT NOT NULL,
  preco_unitario REAL,
  categoria TEXT
);

-- Tabela de Relatórios
CREATE TABLE relatorios (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  data_geracao DATETIME DEFAULT CURRENT_TIMESTAMP,
  tipo TEXT,
  conteudo TEXT,
  enviado BOOLEAN DEFAULT 0
);
```

Características:

- Transações ACID
- Índices para otimização de queries
- Triggers para auditoria automática

5. Camada de Integração

5.1 API EXTERNA - PREÇOS DE PRODUTOS

Responsabilidade: Enriquecimento de dados com informação externa

Funcionalidades:

- Obtenção de preços unitários de peças/produtos
- Cálculo de margens e custos de produção
- Cruzamento com dados de produção
- Utilização de regex para matching de produtos

Fluxo de Integração:

1. Node-RED deteta nova produção
2. Extrai nome/código do produto via regex
3. Consulta API para obter preço
4. Calcula valor total da produção
5. Armazena dados enriquecidos

6. Camada de Relatórios

6.1 PYTHON REPORTING ENGINE

Responsabilidade: Geração e envio de relatórios automáticos

Funcionalidades:

- Consulta de dados agregados da base de dados
- Geração de relatórios em formato HTML/PDF
- Cálculo de KPIs: OEE, produtividade, tempo médio de paragem
- Envio automático por email via SMTP

Bibliotecas Utilizadas:

- `sqlite3`: Conexão à base de dados
 - `smtplib`: Envio de emails
 - `jinja2`: Templates de relatórios (opcional)
 - `matplotlib`: Gráficos e visualizações (opcional)
-

7. Camada de Apresentação

7.1 DASHBOARD WEB (REACT + MATERIAL-UI)

Responsabilidade: Visualização interativa de dados em tempo real

Funcionalidades:

- Monitorização em tempo real de estações de produção
- Gráficos de produção, paragens e eficiência
- Indicadores de performance (KPIs)
- Histórico de produção e tendências
- Interface responsiva e moderna

Tecnologias:

- React 18+
- Material-UI v5
- Axios para chamadas API
- Chart.js / Recharts para gráficos

7.2 GRAFANA (OPCIONAL)

Funcionalidades:

- Dashboards configuráveis
 - Alertas personalizados
 - Visualizações avançadas
-

8. API REST - Gestão de Simulações (Fase 2)

Responsabilidade: Orquestração de simuladores via API

Endpoints:

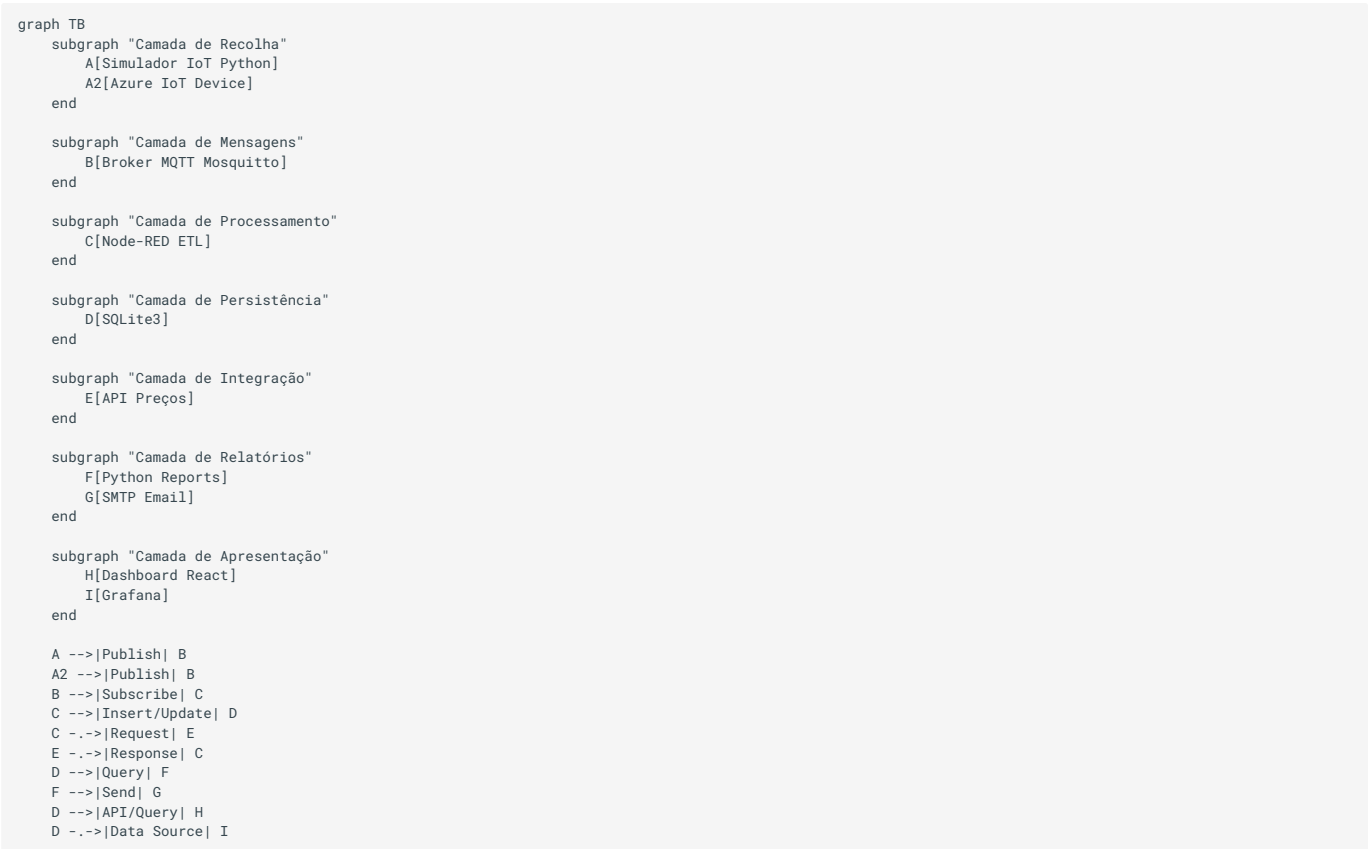
POST	/simulations	# Criar nova simulação
GET	/simulations	# Listar simulações
GET	/simulations/{id}	# Detalhes de simulação
DELETE	/simulations/{id}	# Parar simulação
GET	/simulations/{id}/logs	# Logs de simulação

Funcionalidades:

- Gestão de simuladores em containers Docker
- Configuração dinâmica de sensores
- Monitorização de estado
- Persistência em SQLite

Tecnologia: FastAPI + Uvicorn

2.1.3 Fluxo Global de Dados



2.1.4 Padrões Arquiteturais Implementados

1. Publish-Subscribe Pattern

- Desacoplamento entre produtores e consumidores de dados
- Comunicação assíncrona via MQTT

2. ETL Pipeline Pattern

- Separação clara entre extração, transformação e carregamento

- Processamento sequencial com validação em cada etapa

3. Repository Pattern

- Abstração de acesso a dados via SQLite
 - Centralização de queries e lógica de persistência
-

2.1.5 Características Não-Funcionais

Performance

- Processamento em tempo real com latência < 100ms
- Suporte para 100+ mensagens/segundo

Escalabilidade

- Arquitetura horizontal para múltiplas estações
- Possibilidade de federação de brokers MQTT

Disponibilidade

- Persistência de mensagens no broker
- Reconexão automática de clientes

Segurança

- Autenticação MQTT (futuro)
- Encriptação TLS para comunicações (futuro)
- Validação de inputs em todas as camadas

Manutenibilidade

- Código modular e documentado
 - Configuração externa (JSON/YAML)
 - Logs estruturados
-

2.1.6 Tecnologias Utilizadas

Componente	Tecnologia	Versão	Licença
Simulador IoT	Python	3.10+	PSF
Broker MQTT	Mosquitto	2.0+	EPL/EDL
Motor ETL	Node-RED	3.0+	Apache 2.0
Base de Dados	SQLite	3.40+	Public Domain
Frontend	React	18+	MIT
UI Components	Material-UI	5+	MIT
API Framework	FastAPI	0.100+	MIT
Documentação	MkDocs	1.5+	BSD

2.1.7 Próximos Passos

Ver [Fase 2 - Roadmap](#) para melhorias planeadas:

- Migração para SQL Server
- Containerização completa com Docker Compose
- Implementação de Machine Learning para previsões
- Sistema de alertas em tempo real (SMS/Telegram)

2.2 ETL PROCESSOS

``markdown

2.3 Processos ETL e Transformações

O **Node-RED** atua como o motor central de **ETL** (Extract, Transform, Load).

2.3.1 Estratégia

- 1. **Extração (Extract)**
- 2. Dados recolhidos de sensores MQTT.
- 3. Publicação periódica dos valores de produção, stock e paragem.
- 4. **Transformação (Transform)**
- 5. Normalização dos dados.
- 6. Cálculo de acumulados.
- 7. Validação e filtragem de outliers.
- 8. **Carregamento (Load)**
- 9. Inserção e atualização dos dados em **SQLite3**.
- 10. Geração de logs e relatórios.

2.3.2 Exemplos de Transformações

Tipo	Operação	Descrição
Limpeza	Eliminar valores nulos	Ignorar mensagens sem payload válido
Normalização	Converter tipos	Garantir que todos os valores são <code>float</code>
Agregação	Somar produção acumulada	<code>acumulado += msg.payload.producao</code>
Regex	Extrair ID do produto	<code>/produto-(\d+)/</code>
Filtro	Ignorar paragens < 2s	<code>if (msg.payload.paragem >= 2)</code>

2.3.3 Jobs Node-RED

Os fluxos Node-RED incluem:

- 1. **MQTT In** → **Function** → **SQLite Out**
- 2. **Function** → **Python Exec** → **Email Out**

Esses fluxos são documentados visualmente no ficheiro `flows.json` e explicados no relatório final.

2.4 Base de Dados

O **ProjetoISiv1** utiliza **SQLite3** como sistema de gestao de base de dados para armazenar dados de monitorizacao industrial processados pelo sistema ETL.

2.4.1 Visao Geral

Tecnologia

SQLite 3.40+ - Base de dados relacional embutida, serverless e self-contained

Vantagens da Escolha

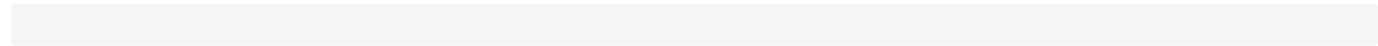
- **Simplicidade:** Nao requer servidor dedicado
- **Portabilidade:** Ficheiro unico `.db` facilmente transferivel
- **Performance:** Rapida para operacoes de leitura/escrita locais
- **Zero Configuracao:** Pronta a usar sem setup complexo
- **ACID Compliant:** Garante integridade transacional
- **Adequada para Prototipos:** Ideal para desenvolvimento e demonstracao

Limitacoes

- Nao adequada para alta concorrencia (multiplas escritas simultaneas)
- Sem controlo de acessos granular
- Para producao enterprise, considerar migracao para SQL Server/PostgreSQL (ver [Fase 2](#))

2.4.2 Estrutura da Base de Dados

Diagrama Entidade-Relacionamento



2.4.3 Tabelas

1. estacoes

Armazena informacoes sobre as estacoes de producao monitorizadas.

```
CREATE TABLE estacoes (  
  id INTEGER PRIMARY KEY,  
  nome TEXT NOT NULL,  
  localizacao TEXT,  
  tipo_producao TEXT,  
  capacidade_maxima INTEGER,  
  status TEXT DEFAULT 'ativa',  
  data_instalacao DATE,  
  ultima_manutencao DATE  
);
```

CAMPOS (ESTACOES)

Campo	Tipo	Descrição
id	INTEGER (PK)	Identificador unico da estacao
nome	TEXT (NOT NULL)	Nome descritivo da estacao
localizacao	TEXT	Localizacao fisica (ex: "Nave A - Setor 2")
tipo_producao	TEXT	Tipo de produto fabricado
capacidade_maxima	INTEGER	Capacidade maxima de producao (pecas/hora)
status	TEXT	Status operacional: 'ativa', 'inativa', 'manutencao'
data_instalacao	DATE	Data de instalacao da estacao
ultima_manutencao	DATE	Data da ultima manutencao

EXEMPLO DE DADOS (ESTACOES)

```
INSERT INTO estacoes (id, nome, localizacao, tipo_producao, capacidade_maxima) VALUES
(1, 'Estacao de Montagem A1', 'Nave A - Setor 1', 'Montagem de Componentes', 100),
(2, 'Estacao de Soldadura B2', 'Nave B - Setor 2', 'Soldadura de Pecas', 80),
(3, 'Estacao de Inspecao C1', 'Nave C - Controlo Qualidade', 'Inspecao Visual', 120);
```

2. dados_acumulados

Regista metricas de producao acumuladas de cada estacao ao longo do tempo.

```
CREATE TABLE dados_acumulados (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  estacao_id INTEGER NOT NULL,
  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  producao_total INTEGER DEFAULT 0,
  stock_atual INTEGER DEFAULT 0,
  tempo_paragem_segundos INTEGER DEFAULT 0,
  defeitos_total INTEGER DEFAULT 0,
  eficiencia_percentual REAL,
  observacoes TEXT,
  FOREIGN KEY (estacao_id) REFERENCES estacoes(id)
);
```

CAMPOS (DADOS_ACUMULADOS)

Campo	Tipo	Descrição
id	INTEGER (PK)	Identificador unico do registo
estacao_id	INTEGER (FK)	Referencia a estacao
timestamp	DATETIME	Data e hora do registo
producao_total	INTEGER	Total de pecas produzidas (acumulado)
stock_atual	INTEGER	Nivel de stock atual (pecas)
tempo_paragem_segundos	INTEGER	Tempo total de paragem em segundos
defeitos_total	INTEGER	Total de pecas defeituosas
eficiencia_percentual	REAL	Taxa de eficiencia calculada (OEE)
observacoes	TEXT	Notas adicionais

INDICES

```
CREATE INDEX idx_estacao_timestamp ON dados_acumulados(estacao_id, timestamp);
CREATE INDEX idx_timestamp ON dados_acumulados(timestamp DESC);
```

EXEMPLO DE DADOS (DADOS_ACUMULADOS)

```
INSERT INTO dados_acumulados (estacao_id, producao_total, stock_atual, tempo_paragem_segundos, defeitos_total, eficiencia_percentual) VALUES
(1, 450, 120, 180, 5, 94.5),
(2, 380, 95, 240, 8, 91.2),
(3, 520, 150, 120, 2, 97.8);
```

3. produtos

Catalogo de produtos/pecas fabricadas com informacao de precos.

```
CREATE TABLE produtos (
  id INTEGER PRIMARY KEY,
  codigo TEXT UNIQUE NOT NULL,
  nome TEXT NOT NULL,
  descricao TEXT,
  preco_unitario REAL NOT NULL,
  custo_producao REAL,
  categoria TEXT,
  unidade TEXT DEFAULT 'unidade',
  ativo BOOLEAN DEFAULT 1,
  data_criacao DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

CAMPOS (PRODUTOS)

Campo	Tipo	Descricao
id	INTEGER (PK)	Identificador unico do produto
codigo	TEXT (UNIQUE)	Codigo do produto (SKU)
nome	TEXT (NOT NULL)	Nome do produto
descricao	TEXT	Descricao detalhada
preco_unitario	REAL	Preco de venda unitario (EUR)
custo_producao	REAL	Custo de producao unitario (EUR)
categoria	TEXT	Categoria do produto
unidade	TEXT	Unidade de medida
ativo	BOOLEAN	Se o produto esta ativo no catalogo
data_criacao	DATETIME	Data de criacao do registo

EXEMPLO DE DADOS (PRODUTOS)

```
INSERT INTO produtos (codigo, nome, preco_unitario, custo_producao, categoria) VALUES
('PCB-001', 'Placa de Circuito Impresso Tipo A', 15.50, 8.20, 'Eletronica'),
('MTL-045', 'Suporte Metalico de Fixacao', 3.80, 1.90, 'Mecanica'),
('PLT-112', 'Tampa Plastica de Protecao', 2.10, 0.95, 'Plasticos');
```

4. relatorios

Registo de relatorios gerados automaticamente pelo sistema.

```
CREATE TABLE relatorios (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  data_geracao DATETIME DEFAULT CURRENT_TIMESTAMP,
  tipo TEXT NOT NULL,
  periodo_inicio DATE,
  periodo_fim DATE,
  conteudo TEXT,
  formato TEXT DEFAULT 'HTML',
  enviado BOOLEAN DEFAULT 0,
  destinatarios TEXT,
  caminho_ficheiro TEXT
);
```

CAMPOS (RELATORIOS)

Campo	Tipo	Descricao
id	INTEGER (PK)	Identificador unico do relatorio
data_geracao	DATETIME	Data e hora de geracao
tipo	TEXT	Tipo: 'diario', 'semanal', 'mensal', 'alerta'
periodo_inicio	DATE	Data de inicio do periodo analisado
periodo_fim	DATE	Data de fim do periodo analisado
conteudo	TEXT	Conteudo do relatorio (HTML/texto)
formato	TEXT	Formato: 'HTML', 'PDF', 'JSON'
enviado	BOOLEAN	Se foi enviado por email
destinatarios	TEXT	Emails dos destinatarios (CSV)
caminho_ficheiro	TEXT	Caminho do ficheiro gerado

EXEMPLO DE DADOS (RELATORIOS)

```
INSERT INTO relatorios (tipo, periodo_inicio, periodo_fim, formato, enviado) VALUES
('diario', '2025-12-25', '2025-12-25', 'HTML', 1),
('semanal', '2025-12-19', '2025-12-25', 'PDF', 0);
```

2.4.4 Views e Consultas uteis

View: Resumo de Producao Diaria

```
CREATE VIEW v_resumo_diario AS
SELECT
  e.nome AS estacao,
  DATE(d.timestamp) AS data,
  SUM(d.producao_total) AS total_producao,
  SUM(d defeitos_total) AS total_defeitos,
  SUM(d.tempo_paragem_segundos) AS total_paragem,
  AVG(d.eficiencia_percentual) AS eficiencia_media
FROM dados_acumulados d
JOIN estacoes e ON d.estacao_id = e.id
GROUP BY e.nome, DATE(d.timestamp);
```

View: Top Produtos por Margem

```
CREATE VIEW v_margem_produtos AS
SELECT
  codigo,
  nome,
  preco_unitario,
  custo_producao,
  (preco_unitario - custo_producao) AS margem,
  ROUND(((preco_unitario - custo_producao) / preco_unitario * 100), 2) AS margem_percentual
FROM produtos
WHERE ativo = 1
ORDER BY margem DESC;
```

Consulta: Performance de Estacoes (ultima Semana)

```
SELECT
  e.nome,
  COUNT(d.id) AS registros,
  SUM(d.producao_total) AS producao_total,
  SUM(d defeitos_total) AS defeitos_total,
  ROUND(AVG(d.eficiencia_percentual), 2) AS eficiencia_media,
  SUM(d.tempo_paragem_segundos) / 3600.0 AS horas_paragem
FROM dados_acumulados d
JOIN estacoes e ON d.estacao_id = e.id
WHERE d.timestamp >= datetime('now', '-7 days')
```

```
GROUP BY e.nome
ORDER BY producao_total DESC;
```

Consulta: Alertas de Baixa Eficiencia

```
SELECT
  e.nome,
  d.timestamp,
  d.eficiencia_percentual,
  d.defeitos_total,
  d.tempo_paragem_segundos
FROM dados_acumulados d
JOIN estacoes e ON d.estacao_id = e.id
WHERE d.eficiencia_percentual < 85
ORDER BY d.timestamp DESC
LIMIT 10;
```

2.4.5 Triggers

Trigger: Atualizar Eficiencia Automaticamente

```
CREATE TRIGGER calcular_eficiencia
AFTER INSERT ON dados_acumulados
BEGIN
  UPDATE dados_acumulados
  SET eficiencia_percentual =
    CASE
      WHEN NEW.producao_total > 0 THEN
        ((NEW.producao_total - NEW.defeitos_total) * 100.0 / NEW.producao_total)
      ELSE 0
    END
  WHERE id = NEW.id;
END;
```

Trigger: Log de Auditoria

```
CREATE TABLE audit_log (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  tabela TEXT,
  operacao TEXT,
  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  dados_anteriores TEXT,
  dados_novos TEXT
);

CREATE TRIGGER audit_producao
AFTER UPDATE ON dados_acumulados
BEGIN
  INSERT INTO audit_log (tabela, operacao, dados_anteriores, dados_novos)
  VALUES ('dados_acumulados', 'UPDATE',
    json_object('producao', OLD.producao_total),
    json_object('producao', NEW.producao_total));
END;
```

2.4.6 Operacoes Comuns

Inserir Novo Registo de Producao

```
INSERT INTO dados_acumulados (estacao_id, producao_total, stock_atual, tempo_paragem_segundos, defeitos_total)
VALUES (1, 458, 125, 190, 6);
```

Atualizar Status de Estacao

```
UPDATE estacoes
SET status = 'manutencao', ultima_manutencao = CURRENT_DATE
WHERE id = 2;
```

Consultar Producao de Hoje

```
SELECT * FROM dados_acumulados
WHERE DATE(timestamp) = DATE('now')
ORDER BY timestamp DESC;
```

Eliminar Dados Antigos (Mais de 1 Ano)

```
DELETE FROM dados_acumulados
WHERE timestamp < datetime('now', '-1 year');
```

2.4.7 Backup e Manutencao

Criar Backup

```
# Backup completo
sqlite3 industrial_monitoring.db ".backup backup_${date +%Y%m%d}.db"

# Backup em SQL
sqlite3 industrial_monitoring.db ".dump" > backup.sql
```

Restaurar Backup

```
# Restaurar de ficheiro .db
sqlite3 industrial_monitoring.db ".restore backup_20251226.db"

# Restaurar de SQL
sqlite3 new_database.db < backup.sql
```

Otimizar Base de Dados

```
-- Reindexar
REINDEX;

-- Vacuum (compactar e otimizar)
VACUUM;

-- Analisar estatisticas para optimizacao de queries
ANALYZE;
```

Verificar Integridade

```
PRAGMA integrity_check;
PRAGMA foreign_key_check;
```

2.4.8 Conexao e Acesso

Python

```
import sqlite3

conn = sqlite3.connect('industrial_monitoring.db')
cursor = conn.cursor()

cursor.execute("SELECT * FROM estacoes")
estacoes = cursor.fetchall()

conn.close()
```

Node-RED

Utiliza o no `sqlite` configurado com o caminho da base de dados.

CLI

```
sqlite3 industrial_monitoring.db
```

2.4.9 Migracao Futura

Para ambientes de producao, planeia-se migrar para **SQL Server** ou **PostgreSQL** (ver [Fase 2](#)).

Vantagens da Migracao

- Melhor suporte para concorrencia
 - Replicacao e alta disponibilidade
 - Controlo de acessos granular
 - Performance em grandes volumes
 - Ferramentas empresariais de gestao
-

2.4.10 Referencias

- [SQLite Documentation](#)
- [SQLite Tutorial](#)
- [DB Browser for SQLite](#) - GUI para explorar a base de dados

3. Integrações

3.1 Integração com API de Preços

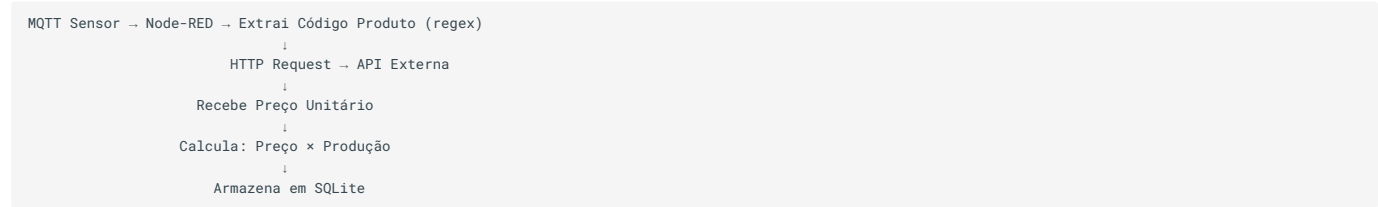
Sistema de enriquecimento de dados de produção com preços de peças através de API externa.

3.1.1 Objetivo

Calcular o **valor monetário da produção** em tempo real, combinando: - Dados de produção (quantas peças foram produzidas) - Preços unitários obtidos via API externa

Resultado: Métricas financeiras automáticas (receita estimada, margem, custo de defeitos).

3.1.2 Arquitetura



3.1.3 API Externa

Endpoint Utilizado

Assumindo uma API REST que retorna preços de produtos:

```
GET https://api.empresa.com/produtos/{codigo}/preco
```

Resposta:

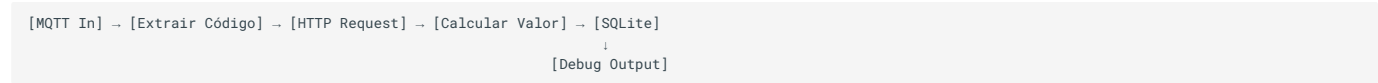
```
{
  "codigo": "PCB-001",
  "nome": "Placa de Circuito Tipo A",
  "preco_unitario": 15.50,
  "moeda": "EUR",
  "ultima_atualizacao": "2024-12-28T10:00:00Z"
}
```

Alternativa: API Pública Mockada

Para testes, pode-se usar: - [JSONPlaceholder](#) - [Mocky.io](#) - API própria com Express/FastAPI

3.1.4 Implementação no Node-RED

Flow Completo



1. Extrair Código do Produto

Node: function - "Extrair Código Produto"

```
// Entrada: msg.payload = {codigo_produto: "PCB-001", dados: {...}}
// Saída: msg.codigo_produto = "PCB-001"

if (msg.payload.codigo_produto) {
  msg.codigo_produto = msg.payload.codigo_produto;

  // Guardar dados originais para uso posterior
  flow.set('ultimo_payload', msg.payload);

  return msg;
} else {
  node.warn("Código de produto não encontrado");
  return null;
}
```

Configuração: - Nome: "Extrair Código Produto" - Outputs: 1

2. Consultar API de Preços

Node: http request - "Buscar Preço API"

Configuração: - Method: GET - URL: `https://api.empresa.com/produtos/{{codigo_produto}}/preco` - Return: parsed JSON object - Headers: - Authorization: Bearer {{env.API_TOKEN}} - Content-Type: application/json

Alternativa com Template URL:

Se a API requerer formato diferente:

```
// Node function antes do HTTP request
msg.url = `https://api.empresa.com/produtos/${msg.codigo_produto}/preco`;
return msg;
```

3. Calcular Valor da Produção

Node: function - "Calcular Valor Total"

```
// msg.payload agora tem dados da API (preco_unitario)
// Recuperar dados originais do flow context

let dados_producao = flow.get('ultimo_payload');
let preco_unitario = msg.payload.preco_unitario;

if (!preco_unitario || !dados_producao) {
  node.warn("Dados incompletos para cálculo");
  return null;
}

// Calcular valores
let producao = dados_producao.dados.producao;
let defeitos = dados_producao.dados.defeitos;

let valor_producao = producao * preco_unitario;
let valor_perdido_defeitos = defeitos * preco_unitario;
let valor_liquido = valor_producao - valor_perdido_defeitos;

// Montar payload para BD
msg.payload = {
  estacao: dados_producao.estacao,
  codigo_produto: dados_producao.codigo_produto,
  timestamp: dados_producao.timestamp,
  producao: producao,
  preco_unitario: preco_unitario,
  valor_producao: valor_producao.toFixed(2),
  defeitos: defeitos,
  valor_perdido: valor_perdido_defeitos.toFixed(2),
  valor_liquido: valor_liquido.toFixed(2)
};

// Query SQL para inserção
msg.topic = `
INSERT INTO producao_valores
(estacao_id, codigo_produto, timestamp, producao, preco_unitario,
valor_producao, defeitos, valor_perdido, valor_liquido)
VALUES
('${msg.payload.estacao}', '${msg.payload.codigo_produto}',
 '${msg.payload.timestamp}', ${msg.payload.producao},
 ${msg.payload.preco_unitario}, ${msg.payload.valor_producao},`
```

```
{msg.payload.defeitos}, ${msg.payload.valor_perdido},
${msg.payload.valor_liquido})
`;

return msg;
```

4. Armazenar em Base de Dados

Node: `sqlite` - "Gravar Valores Produção"

Configuração: - Database: `/path/to/industrial_monitoring.db` - SQL Query: `msg.topic` (definido na função anterior)

3.1.5 Regex para Produtos Diversos

Se a API retorna múltiplos produtos e você precisa filtrar:

Cenário: API retorna lista de 100 produtos, você quer apenas IDs específicos.

Node: `function` - "Filtrar Produtos Relevantes"

```
// msg.payload = [{id: 1, preco: 10}, {id: 2, preco: 20}, ...]
// Produtos que interessam: IDs 5, 12, 23

let produtos_interesse = [5, 12, 23];
let regex_ids = new RegExp(`^(${produtos_interesse.join('|')})$`);

let produtos_filtrados = msg.payload.filter(produto => {
  return regex_ids.test(produto.id.toString());
});

if (produtos_filtrados.length === 0) {
  node.warn("Nenhum produto relevante encontrado");
  return null;
}

// Guardar no contexto para uso posterior
flow.set('produtos_precos', produtos_filtrados);

msg.payload = produtos_filtrados;
return msg;
```

3.1.6 Tabela de Base de Dados

Schema SQL

```
CREATE TABLE producao_valores (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  estacao_id INTEGER,
  codigo_produto TEXT,
  timestamp DATETIME,
  producao INTEGER,
  preco_unitario REAL,
  valor_producao REAL,
  defeitos INTEGER,
  valor_perdido REAL,
  valor_liquido REAL,
  FOREIGN KEY (estacao_id) REFERENCES estacoes(id)
);

CREATE INDEX idx_producao_valores_timestamp
ON producao_valores(timestamp DESC);
```

Consultas Úteis

Valor total produzido hoje:

```
SELECT SUM(valor_liquido) as receita_diaria
FROM producao_valores
WHERE DATE(timestamp) = DATE('now');
```

Top 3 produtos por receita (última semana):

```
SELECT
  codigo_produto,
  SUM(valor_producao) as receita_total,
  SUM(valor_perdido) as perdas_efeitos
FROM producao_valores
WHERE timestamp >= datetime('now', '-7 days')
GROUP BY codigo_produto
ORDER BY receita_total DESC
LIMIT 3;
```

Custo de defeitos por estação:

```
SELECT
  e.nome,
  SUM(pv.valor_perdido) as custo_defeitos
FROM producao_valores pv
JOIN estacoes e ON pv.estacao_id = e.id
WHERE DATE(pv.timestamp) = DATE('now')
GROUP BY e.nome;
```

3.1.7 Gestão de Erros

Tratamento de Falhas na API**Node:** catch - "Erro API"

Adicionar node catch que captura erros do HTTP request:

```
// Node function conectado ao catch
if (msg.error) {
  node.error(`API falhou: ${msg.error.message}`);

  // Usar preço padrão/último conhecido
  let ultimo_preco = flow.get(`preco_${msg.codigo_produto}`) || 0;

  msg.payload = {
    preco_unitario: ultimo_preco,
    fonte: "cache",
    aviso: "API indisponível - usando último preço conhecido"
  };

  return msg;
}
```

Cache de Preços

Evitar chamadas repetidas para o mesmo produto:

```
// No início do flow de preços
let codigo = msg.codigo_produto;
let cache_preco = flow.get(`preco_cache_${codigo}`);
let cache_timestamp = flow.get(`preco_cache_ts_${codigo}`);

// Cache válido por 1 hora
if (cache_preco && cache_timestamp) {
  let agora = Date.now();
  let diff_minutos = (agora - cache_timestamp) / 1000 / 60;

  if (diff_minutos < 60) {
    node.warn(`Usando preço em cache para ${codigo}`);
    msg.payload = {preco_unitario: cache_preco};
    return msg;
  }
}

// Se não há cache válido, prosseguir para HTTP request
return msg;
```

Após receber resposta da API:

```
// Guardar em cache
flow.set(`preco_cache_${msg.codigo_produto}`, msg.payload.preco_unitario);
flow.set(`preco_cache_ts_${msg.codigo_produto}`, Date.now());
```

3.1.8 Rate Limiting

Se a API tem limite de requests:

```
// Antes do HTTP request
let ultima_chamada = flow.get('api_ultima_chamada') || 0;
let agora = Date.now();
let delay_minimo = 1000; // 1 segundo entre chamadas

if (agora - ultima_chamada < delay_minimo) {
  let esperar = delay_minimo - (agora - ultima_chamada);
  node.warn('Rate limit: aguardar ${esperar}ms');

  // Agendar retry
  setTimeout(() => {
    node.send(msg);
  }, esperar);

  return null;
}

flow.set('api_ultima_chamada', agora);
return msg;
```

3.1.9 Variáveis de Ambiente

Configurar credenciais da API:

Ficheiro: settings.js (Node-RED)

```
functionGlobalContext: {
  API_BASE_URL: process.env.API_BASE_URL || 'https://api.empresa.com',
  API_TOKEN: process.env.API_TOKEN || '',
}
```

Uso no flow:

```
let api_url = global.get('API_BASE_URL');
let token = global.get('API_TOKEN');

msg.url = `${api_url}/produtos/${msg.codigo_produto}/preco`;
msg.headers = {
  'Authorization': `Bearer ${token}`
};

return msg;
```

3.1.10 Exemplo de Flow JSON

```
[
  {
    "id": "api_flow_start",
    "type": "mqtt in",
    "topic": "linha_producao/estacao/+",
    "broker": "mqtt_broker",
    "wires": [["extrair_codigo"]]
  },
  {
    "id": "extrair_codigo",
    "type": "function",
    "name": "Extrair Código",
    "func": "msg.codigo_produto = msg.payload.codigo_produto;\nflow.set('ultimo_payload', msg.payload);\nreturn msg;",
    "wires": [["http_get_preco"]]
  },
  {
    "id": "http_get_preco",
    "type": "http request",
    "method": "GET",
    "url": "https://api.empresa.com/produtos/{codigo_produto}/preco",
    "wires": [["calcular_valor"]]
  },
  {
    "id": "calcular_valor",
    "type": "function",
    "name": "Calcular Valor",
    "func": "// código da função anterior",
    "wires": [["sqlite_insert"]]
  },
  {
    "id": "sqlite_insert",
    "type": "sqlite insert",
    "name": "Inserir no SQLite",
    "func": "flow.set('ultimo_codigo', msg.payload);\nreturn msg;",
    "wires": []
  }
]
```

```
    "id": "sqlite_insert",
    "type": "sqlite",
    "db": "industrial_monitoring.db",
    "wires": [[]]
  }
}
```

3.1.11 Métricas Disponíveis

Após implementação completa, terá acesso a:

1. **Valor de Produção por Estação** (€/dia)
2. **Custo de Defeitos** (€ perdidos)
3. **ROI de Manutenções** (comparar antes/depois)
4. **Produtos Mais Lucrativos**
5. **Tendências de Margem**

Estas métricas podem ser visualizadas no dashboard ou incluídas nos relatórios automáticos.

3.1.12 Troubleshooting

API retorna erro 401: - Verificar token de autenticação - Confirmar validade do token - Verificar headers da request

Valores sempre zero: - Verificar se API está retornando dados - Debug do node HTTP request - Confirmar parsing JSON correto

Performance lenta: - Implementar cache de preços - Reduzir frequência de chamadas - Considerar batch requests se API suportar

3.1.13 Melhorias Futuras

- Webhook quando preços mudam (em vez de polling)
 - Batch requests para múltiplos produtos
 - Fallback para múltiplas APIs (redundância)
 - Machine learning para prever preços
-

Esta integração transforma dados operacionais em métricas financeiras, permitindo decisões de negócio baseadas em dados em tempo real.

3.2 Integração Azure IoT Hub

Sistema de envio de telemetria para a cloud usando Azure IoT Hub, permitindo monitorização centralizada e análise de dados em escala.

3.2.1 Visão Geral

Objetivo: Conectar sensores IoT simulados ao Azure IoT Hub para processamento na cloud.

Biblioteca: `azure-iot-device` (Python SDK oficial da Microsoft)

Arquitetura Dual:

```
Simulador IoT
├─> MQTT Local (Mosquitto) → Node-RED → Dashboard/DB Local
└─> Azure IoT Hub → Stream Analytics → Azure SQL/Storage
```

Casos de Uso: - Monitorização multi-site centralizada - Backup de dados na cloud - Análise com Azure Stream Analytics - Dashboards em Azure IoT Central - Integração com Power BI

3.2.2 Instalação

```
pip install azure-iot-device
```

Dependências:

```
azure-iot-device==2.13.0
paho-mqtt==1.6.1
```

3.2.3 Configuração Azure IoT Hub

1. Criar IoT Hub (Azure Portal)

1. Aceder ao [Azure Portal](#)
2. Criar recurso → Internet of Things → IoT Hub
3. Configurar:
4. **Nome:** `producao-industrial-hub`
5. **Região:** West Europe
6. **Tier:** F1 (gratuito) ou S1 (produção)
7. Criar

2. Registar Device

Via Portal: 1. IoT Hub → Devices → Add Device 2. Device ID: `estacao-1` 3. Authentication: Symmetric Key (auto-generate) 4. Copiar **Primary Connection String**

Via Azure CLI:

```
az iot hub device-identity create \
  --hub-name producao-industrial-hub \
  --device-id estacao-1

az iot hub device-identity connection-string show \
  --hub-name producao-industrial-hub \
  --device-id estacao-1
```

Connection String (guardar de forma segura):

```
HostName=producao-industrial-hub.azure-devices.net;DeviceId=estacao-1;SharedAccessKey=xxxxxxxxxxxx
```

3.2.4 Implementação Python

Exemplo Básico: Enviar Telemetria

```
# azure_iot_sender.py

from azure.iot.device import IoTHubDeviceClient, Message
import json
import time
from datetime import datetime
import os

class AzureIoTSender:
    def __init__(self, connection_string):
        """
        Inicializar cliente Azure IoT

        Args:
            connection_string: Connection string do device no Azure IoT Hub
        """
        self.client = IoTHubDeviceClient.create_from_connection_string(
            connection_string
        )
        self.connected = False

    def connect(self):
        """Conectar ao Azure IoT Hub"""
        try:
            self.client.connect()
            self.connected = True
            print(f"✓ Conectado ao Azure IoT Hub")
        except Exception as e:
            print(f"✗ Erro ao conectar: {e}")
            self.connected = False

    def send_telemetry(self, data):
        """
        Enviar telemetria para Azure IoT Hub

        Args:
            data: Dicionário com dados do sensor
        """
        if not self.connected:
            print("✗ Não conectado ao Azure IoT Hub")
            return False

        try:
            # Adicionar timestamp se não existir
            if 'timestamp' not in data:
                data['timestamp'] = datetime.now().isoformat()

            # Criar mensagem
            message = Message(json.dumps(data))

            # Definir propriedades da mensagem
            message.content_type = "application/json"
            message.content_encoding = "utf-8"

            # Adicionar propriedades customizadas
            message.custom_properties["estacao_id"] = str(data.get("estacao", "unknown"))
            message.custom_properties["tipo_alerta"] = self._check_alert(data)

            # Enviar
            self.client.send_message(message)
            print(f"✓ Telemetria enviada: Estação {data.get('estacao')}")

            return True

        except Exception as e:
            print(f"✗ Erro ao enviar telemetria: {e}")
            return False

    def _check_alert(self, data):
        """Verificar se há condições de alerta"""
        dados = data.get('dados', {})

        if dados.get('defeitos', 0) > 10:
            return "defeitos_alto"
        elif dados.get('paragem', 0) > 600: # 10 min
            return "paragem_longa"
        elif dados.get('stock', 0) < 20:
            return "stock_baixo"
```

```

        return "normal"

    def disconnect(self):
        """Desconectar do Azure IoT Hub"""
        if self.connected:
            self.client.disconnect()
            self.connected = False
            print("✓ Desconectado do Azure IoT Hub")

# Exemplo de uso
if __name__ == "__main__":
    # Connection string do device (usar variável de ambiente)
    CONN_STRING = os.getenv('AZURE_IOT_CONNECTION_STRING')

    if not CONN_STRING:
        print("Erro: Definir AZURE_IOT_CONNECTION_STRING")
        exit(1)

    # Criar cliente
    azure_sender = AzureIoTSender(CONN_STRING)
    azure_sender.connect()

    # Enviar dados de exemplo
    dados_exemplo = {
        "estacao": 1,
        "dados": {
            "producao": 458,
            "stock": 120,
            "paragem": 180,
            "defeitos": 5
        }
    }

    azure_sender.send_telemetry(dados_exemplo)

    # Desconectar
    azure_sender.disconnect()

```

3.2.5 Integração com Simulador MQTT Existente

Simulador Dual: MQTT Local + Azure IoT

```

# mqtt_publisher_azure.py

import paho.mqtt.client as mqtt
from azure.iot.device import IoTHubDeviceClient, Message
import json
import time
import random
from datetime import datetime
import os

class DualPublisher:
    """Publica para MQTT local E Azure IoT Hub simultaneamente"""

    def __init__(self, mqtt_broker, mqtt_port, azure_conn_string, estacao_id):
        # MQTT Local
        self.mqtt_client = mqtt.Client()
        self.mqtt_broker = mqtt_broker
        self.mqtt_port = mqtt_port

        # Azure IoT
        self.azure_client = IoTHubDeviceClient.create_from_connection_string(
            azure_conn_string
        )

        self.estacao_id = estacao_id
        self.running = False

    def connect(self):
        """Conectar a ambos os endpoints"""
        # Conectar MQTT local
        try:
            self.mqtt_client.connect(self.mqtt_broker, self.mqtt_port, 60)
            self.mqtt_client.loop_start()
            print(f"✓ Conectado ao MQTT local ({self.mqtt_broker}:{self.mqtt_port})")
        except Exception as e:
            print(f"✗ Erro MQTT local: {e}")

        # Conectar Azure IoT
        try:
            self.azure_client.connect()
            print(f"✓ Conectado ao Azure IoT Hub")
        except Exception as e:
            print(f"✗ Erro Azure IoT: {e}")

    def publish_data(self, data):
        """Publicar dados para ambos os destinos"""

```



```

timestamp = datetime.now().isoformat()

payload = {
    "estacao": self.estacao_id,
    "timestamp": timestamp,
    "dados": data
}

# Publicar no MQTT local
try:
    topic = f"linha_producao/estacao/{self.estacao_id}"
    self.mqtt_client.publish(topic, json.dumps(data), qos=2)
    print(f"MQTT: {topic}")
except Exception as e:
    print(f"X Erro MQTT publish: {e}")

# Publicar no Azure IoT Hub
try:
    message = Message(json.dumps(payload))
    message.content_type = "application/json"
    message.content_encoding = "utf-8"
    message.custom_properties["estacao_id"] = str(self.estacao_id)

    self.azure_client.send_message(message)
    print(f"Azure: Estação {self.estacao_id}")
except Exception as e:
    print(f"X Erro Azure publish: {e}")

def simulate_production(self, interval=60):
    """Simular produção continuamente"""
    self.running = True

    print(f"\n▶ Iniciando simulação (Estação {self.estacao_id})")
    print(f"Intervalo: {interval}s | Ctrl+C para parar\n")

    try:
        while self.running:
            # Gerar dados sintéticos
            data = {
                "producao": random.randint(80, 120),
                "stock": random.randint(50, 200),
                "paragem": random.randint(0, 300),
                "defeitos": random.randint(0, 8)
            }

            # Publicar
            self.publish_data(data)

            # Aguardar
            time.sleep(interval)

        except KeyboardInterrupt:
            print("\n■ Simulação interrompida")
            self.running = False

def disconnect(self):
    """Desconectar de ambos"""
    self.running = False

    try:
        self.mqtt_client.loop_stop()
        self.mqtt_client.disconnect()
        print("✓ MQTT local desconectado")
    except:
        pass

    try:
        self.azure_client.disconnect()
        print("✓ Azure IoT desconectado")
    except:
        pass

# Executar
if __name__ == "__main__":
    # Configuração
    MQTT_BROKER = os.getenv('MQTT_BROKER', 'localhost')
    MQTT_PORT = int(os.getenv('MQTT_PORT', 1883))
    AZURE_CONN = os.getenv('AZURE_IOT_CONNECTION_STRING')
    ESTACAO_ID = int(os.getenv('ESTACAO_ID', 1))

    if not AZURE_CONN:
        print("Erro: Definir AZURE_IOT_CONNECTION_STRING")
        exit(1)

    # Criar publisher dual
    publisher = DualPublisher(MQTT_BROKER, MQTT_PORT, AZURE_CONN, ESTACAO_ID)

    # Conectar
    publisher.connect()

    # Simular
    try:
        publisher.simulate_production(interval=60)

```

```
finally:
    publisher.disconnect()
```

Variáveis de Ambiente

```
# .env
export MQTT_BROKER="localhost"
export MQTT_PORT="1883"
export AZURE_IOT_CONNECTION_STRING="HostName=...azure-devices.net;DeviceId=estacao-1;SharedAccessKey=..."
export ESTACAO_ID="1"
```

3.2.6 Device Twin (Propriedades do Device)

Ler e Atualizar Propriedades

```
# device_twin_manager.py

from azure.iot.device import IoTHubDeviceClient
import json

class DeviceTwinManager:
    def __init__(self, connection_string):
        self.client = IoTHubDeviceClient.create_from_connection_string(
            connection_string
        )
        self.client.connect()

    def get_desired_properties(self):
        """Obter configurações enviadas pela cloud"""
        twin = self.client.get_twin()
        desired = twin['desired']

        print("Configurações desejadas (cloud → device):")
        print(json.dumps(desired, indent=2))

        return desired

    def update_reported_properties(self, properties):
        """Reportar estado do device para a cloud"""
        self.client.patch_twin_reported_properties(properties)
        print(f"✓ Propriedades reportadas atualizadas: {properties}")

    def get_full_twin(self):
        """Obter twin completo"""
        twin = self.client.get_twin()

        print("\nDevice Twin completo:")
        print(json.dumps(twin, indent=2))

        return twin

# Exemplo de uso
if __name__ == "__main__":
    import os

    conn_string = os.getenv('AZURE_IOT_CONNECTION_STRING')
    manager = DeviceTwinManager(conn_string)

    # Ler configurações da cloud
    desired = manager.get_desired_properties()

    # Reportar estado atual do device
    reported = {
        "firmware_version": "1.2.0",
        "capacidade_maxima": 500,
        "status": "operacional",
        "ultima_manutencao": "2024-12-20",
        "temperatura_sensor": 42.5
    }

    manager.update_reported_properties(reported)

    # Ver twin completo
    manager.get_full_twin()
```

Configurar via Azure Portal

1. IoT Hub → Devices → estacao-1 → Device Twin

2. Adicionar desired properties:

```
{
  "desired": {
    "frequencia_envio": 60,
    "modo_producao": "alta_velocidade",
    "threshold_alerta_defeitos": 10,
    "threshold_paragem": 600
  }
}
```

1. No simulador, ler estas configs e ajustar comportamento:

```
desired = manager.get_desired_properties()
frequencia = desired.get('frequencia_envio', 60)
threshold_defeitos = desired.get('threshold_alerta_defeitos', 10)

# Usar nas simulações
publisher.simulate_production(interval=frequencia)
```

3.2.7 Direct Methods (Comandos Remotos)

Executar Comandos da Cloud

```
# direct_methods_handler.py

from azure.iot.device import IoTHubDeviceClient, MethodResponse
import json
import time

class CommandHandler:
    def __init__(self, connection_string):
        self.client = IoTHubDeviceClient.create_from_connection_string(
            connection_string
        )
        self.producao_ativa = True
        self.frequencia = 60

        # Registrar handler de métodos
        self.client.on_method_request_received = self.method_request_handler

    def method_request_handler(self, method_request):
        """Handler para processar comandos da cloud"""
        print(f"\n📡 Comando recebido: {method_request.name}")
        print(f"Payload: {method_request.payload}")

        # Processar comando
        if method_request.name == "parar_producao":
            response = self._parar_producao()

        elif method_request.name == "iniciar_producao":
            response = self._iniciar_producao()

        elif method_request.name == "ajustar_frequencia":
            frequencia = method_request.payload.get("frequencia", 60)
            response = self._ajustar_frequencia(frequencia)

        elif method_request.name == "obter_status":
            response = self._obter_status()

        elif method_request.name == "resetar_contadores":
            response = self._resetar_contadores()

        else:
            response = {
                "result": f"Comando desconhecido: {method_request.name}",
                "status": 404
            }

        # Enviar resposta
        method_response = MethodResponse.create_from_method_request(
            method_request,
            status=response["status"],
            payload=response
        )

        self.client.send_method_response(method_response)
        print(f"✅ Resposta enviada: {response['result']}\n")

    def _parar_producao(self):
        self.producao_ativa = False
        return {
            "result": "Produção parada com sucesso",
            "status": 200,
            "producao_ativa": False
        }
```

```

    }

    def _iniciar_producao(self):
        self.producao_ativa = True
        return {
            "result": "Produção iniciada com sucesso",
            "status": 200,
            "producao_ativa": True
        }

    def _ajustar_frequencia(self, nova_freq):
        self.frequencia = nova_freq
        return {
            "result": f"Frequência ajustada para {nova_freq} segundos",
            "status": 200,
            "frequencia": nova_freq
        }

    def _obter_status(self):
        return {
            "result": "Status obtido",
            "status": 200,
            "producao_ativa": self.producao_ativa,
            "frequencia": self.frequencia,
            "timestamp": time.time()
        }

    def _resetar_contadores(self):
        # Lógica para resetar contadores
        return {
            "result": "Contadores resetados",
            "status": 200
        }

    def start_listening(self):
        """Iniciar escuta de comandos"""
        self.client.connect()
        print("✓ Conectado ao Azure IoT Hub")
        print("👂 Aguardando comandos da cloud...\n")

        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print("\n■ Parando escuta de comandos")
            self.client.disconnect()

# Executar
if __name__ == "__main__":
    import os

    conn_string = os.getenv('AZURE_IOT_CONNECTION_STRING')
    handler = CommandHandler(conn_string)
    handler.start_listening()

```

Invocar Comando via Azure CLI

```

# Parar produção
az iot hub invoke-device-method \
  --hub-name producao-industrial-hub \
  --device-id estacao-1 \
  --method-name parar_producao

# Ajustar frequência
az iot hub invoke-device-method \
  --hub-name producao-industrial-hub \
  --device-id estacao-1 \
  --method-name ajustar_frequencia \
  --method-payload '{"frequencia": 30}'

# Obter status
az iot hub invoke-device-method \
  --hub-name producao-industrial-hub \
  --device-id estacao-1 \
  --method-name obter_status

```

3.2.8 Monitorização de Mensagens

Ver Telemetria em Tempo Real

```

# Azure CLI - Monitorizar mensagens
az iot hub monitor-events \
  --hub-name producao-industrial-hub \
  --device-id estacao-1

```

```
# Ou para todos os devices
az iot hub monitor-events \
  --hub-name producao-industrial-hub
```

Azure IoT Explorer (GUI)

1. Instalar: [Azure IoT Explorer](#)
2. Conectar ao IoT Hub
3. Selecionar device
4. Tab "Telemetry" → Start
5. Visualizar mensagens em tempo real

3.2.9 Processamento na Cloud

Azure Stream Analytics

Input: Azure IoT Hub

Output: Azure SQL Database / Blob Storage / Power BI

Query SQL:

```
-- Calcular médias por estação (janela de 5 minutos)
SELECT
    estacao,
    System.Timestamp() AS window_end,
    AVG(dados.producao) AS producao_medio,
    AVG(dados.stock) AS stock_medio,
    SUM(dados.defeitos) AS total_defeitos,
    AVG(dados.paragem) AS paragem_medio
INTO
    [output-sql]
FROM
    [input-iotHub]
TIMESTAMP BY timestamp
GROUP BY
    estacao,
    TumblingWindow(minute, 5)

-- Alertas de defeitos altos
SELECT
    estacao,
    dados.defeitos AS defeitos,
    timestamp
INTO
    [output-alerts]
FROM
    [input-iotHub]
WHERE
    dados.defeitos > 10
```

Armazenar em Azure SQL

Tabela:

```
CREATE TABLE TelemetriaProducao (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    EstacaoId INT,
    Timestamp DATETIME2,
    Producao INT,
    Stock INT,
    Paragem INT,
    Defeitos INT,
    ReceivedAt DATETIME2 DEFAULT GETUTCDATE()
);

CREATE INDEX idx_estacao_timestamp
ON TelemetriaProducao(EstacaoId, Timestamp DESC);
```

3.2.10 Segurança

Boas Práticas

1. **Não hardcodar connection strings** - Usar variáveis de ambiente
2. **Rotação de chaves** - Regenerar Shared Access Keys periodicamente
3. **Usar X.509 Certificates** em produção (em vez de Symmetric Keys)
4. **Implementar Device Provisioning Service (DPS)** para escala

Connection String em Variável de Ambiente

```
import os
from dotenv import load_dotenv

load_dotenv() # Ler .env

CONN_STRING = os.getenv('AZURE_IOT_CONNECTION_STRING')

if not CONN_STRING:
    raise ValueError("AZURE_IOT_CONNECTION_STRING não definida")
```

.env (não commitar no Git):

```
AZURE_IOT_CONNECTION_STRING=HostName=...
```

.gitignore:

```
.env
*.pyc
```

3.2.11 Casos de Uso

1. Multi-Site Monitoring

Centralizar dados de múltiplas fábricas:

```
Fábrica Lisboa → Azure IoT Hub
Fábrica Porto → Azure IoT Hub → Análise → Dashboard Power BI
Fábrica Braga → Azure IoT Hub
```

2. Backup na Cloud

- Dados locais (SQLite) para operação imediata
- Dados na cloud (Azure SQL) para histórico e análise

3. Machine Learning

- Azure Stream Analytics → Azure ML
- Prever falhas de equipamento
- Otimizar schedules de produção

4. Alertas Avançados

- Logic Apps triggam emails/SMS em defeitos altos
 - Integração com Microsoft Teams
 - Webhooks para sistemas externos
-

3.2.12 Custos Azure IoT Hub

Tier	Mensagens/dia	Preço/mês	Casos de Uso
F1 (Free)	8.000	€0	Desenvolvimento/Testes
S1 (Standard)	400.000	~€21	Produção pequena
S2	6M	~€630	Produção média
S3	300M	~€6.300	Enterprise

Cálculo: - 3 estações × 1 msg/min × 60 min × 24h = **4.320 msg/dia** - Cabe no **F1 gratuito** ou **S1** com folga

3.2.13 Troubleshooting

Erro de autenticação: - Verificar connection string correta - Confirmar device está registado no IoT Hub - Regenerar chave se necessário

Mensagens não chegam: - Verificar se device está conectado - Monitorizar com `az iot hub monitor-events` - Verificar quotas do tier F1 (8k msg/dia)

Timeout de conexão: - Verificar firewall/proxy - Porta 8883 (MQTT) ou 443 (HTTPS) aberta - Testar conectividade: `telnet producao-industrial-hub.azure-devices.net 8883`

3.2.14 Próximos Passos

1. **Implementar retry logic** para reconexões
2. **Batch messages** para otimizar quotas
3. **Migrar para X.509 certificates** (mais seguro)
4. **Implementar Device Provisioning Service** para auto-registro
5. **Criar dashboards no Power BI** conectados ao Azure SQL

3.2.15 Recursos

- [Azure IoT Hub Docs](#)
- [Python SDK Reference](#)
- [Azure IoT Explorer](#)
- [Pricing Calculator](#)

Esta integração adiciona capacidades enterprise ao ProjetoSIv1, permitindo escalar de um protótipo local para uma solução cloud-ready com monitorização centralizada e análise avançada.

3.3 Relatórios e Emails Automáticos

Sistema de geração e envio automático de relatórios de produção por email.

3.3.1 Visão Geral

Objetivo: Enviar relatórios diários/semanais com KPIs de produção sem intervenção manual.

Tecnologia: Python scripts + SQLite + SMTP

Trigger: Agendado via cron ou executado por Node-RED

3.3.2 Tipos de Relatórios

1. Relatório Diário

Resumo das últimas 24h de produção.

Conteúdo: - Total produzido por estação - Tempo de paragens - Taxa de defeitos - Eficiência (OEE)

Destinatários: Supervisores de produção

Frequência: Todos os dias às 7h00

2. Relatório Semanal

Análise agregada da semana anterior.

Conteúdo: - Tendências de produção - Comparação entre estações - Top 5 problemas - Recomendações

Destinatários: Gestão

Frequência: Segundas-feiras às 9h00

3. Alertas em Tempo Real

Notificações imediatas de anomalias.

Triggers: - Paragem > 10 minutos - Defeitos > 5% da produção - Eficiência < 80% - Stock crítico (< 20 unidades)

Método: Email ou SMS (futuro)

3.3.3 Estrutura do Relatório

Template HTML

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body { font-family: Arial, sans-serif; }
    .kpi { display: inline-block; padding: 20px; margin: 10px; background: #f0f0f0; border-radius: 8px; }
    .kpi-value { font-size: 32px; font-weight: bold; color: #1976d2; }
    .kpi-label { font-size: 14px; color: #666; }
    .alert { background: #ffebee; border-left: 4px solid #f44336; padding: 10px; margin: 10px 0; }
    table { width: 100%; border-collapse: collapse; }
    th, td { padding: 12px; text-align: left; border-bottom: 1px solid #ddd; }
    th { background: #1976d2; color: white; }
  </style>
</head>
<body>
  <h1>Relatório de Produção - {data}</h1>
```



```

<div class="kpis">
  <div class="kpi">
    <div class="kpi-value">{producao_total}</div>
    <div class="kpi-label">Peças Produzidas</div>
  </div>
  <div class="kpi">
    <div class="kpi-value">{oeo}%</div>
    <div class="kpi-label">OEE</div>
  </div>
  <div class="kpi">
    <div class="kpi-value">{defeitos}%</div>
    <div class="kpi-label">Taxa de Defeitos</div>
  </div>
</div>

{alertas_html}

<h2>Produção por Estação</h2>
<table>
  <tr>
    <th>Estação</th>
    <th>Produção</th>
    <th>Paragens (min)</th>
    <th>Defeitos</th>
    <th>Eficiência</th>
  </tr>
  {tabela_estacoes}
</table>
</body>
</html>

```

3.3.4 Implementação Python

Script Principal: `report_generator.py`

```

import sqlite3
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime, timedelta
import os

class ReportGenerator:
    def __init__(self, db_path='industrial_monitoring.db'):
        self.db_path = db_path
        self.smtp_server = os.getenv('SMTP_SERVER', 'smtp.gmail.com')
        self.smtp_port = int(os.getenv('SMTP_PORT', '587'))
        self.email_from = os.getenv('EMAIL_FROM')
        self.email_password = os.getenv('EMAIL_PASSWORD')

    def get_daily_data(self):
        """Busca dados das últimas 24h"""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        query = """
        SELECT
            e.nome,
            SUM(d.producao_total) as total_producao,
            SUM(d.tempo_paragem_segundos)/60 as paragens_min,
            SUM(d.defeitos_total) as total_defeitos,
            AVG(d.eficiencia_percentual) as eficiencia
        FROM dados_acumulados d
        JOIN estacoes e ON d.estacao_id = e.id
        WHERE d.timestamp >= datetime('now', '-1 day')
        GROUP BY e.nome
        """

        cursor.execute(query)
        results = cursor.fetchall()
        conn.close()

        return results

    def calculate_kpis(self, data):
        """Calcula KPIs agregados"""
        total_producao = sum(row[1] for row in data)
        total_defeitos = sum(row[3] for row in data)
        avg_eficiencia = sum(row[4] for row in data) / len(data) if data else 0
        taxa_defeitos = (total_defeitos / total_producao * 100) if total_producao > 0 else 0

        return {
            'producao_total': total_producao,
            'oeo': round(avg_eficiencia, 1),
            'defeitos': round(taxa_defeitos, 2)
        }

    def generate_html_report(self, data, kpis):

```

```

"""Gera relatório HTML"""
# Template HTML (simplificado para exemplo)
template = """
<html>
<body>
<h1>Relatório Diário - {data}</h1>
<p>Produção Total: {producao_total} peças</p>
<p>OEE: {oe}</p>
<p>Taxa Defeitos: {defeitos}%</p>

<h2>Detalhes por Estação</h2>
<table border="1">
  <tr><th>Estação</th><th>Produção</th><th>Paragens</th><th>Defeitos</th></tr>
  {rows}
</table>
</body>
</html>
"""

rows = ""
for row in data:
    rows += f"<tr><td>{row[0]}</td><td>{row[1]}</td><td>{row[2]:.0f}min</td><td>{row[3]}</td></tr>"

return template.format(
    data=datetime.now().strftime('%Y-%m-%d'),
    producao_total=kpis['producao_total'],
    oe=kpis['oe'],
    defeitos=kpis['defeitos'],
    rows=rows
)

def send_email(self, html_content, recipients):
    """Envia email com relatório"""
    msg = MIME multipart('alternative')
    msg['Subject'] = f"Relatório de Produção - {datetime.now().strftime('%Y-%m-%d')}"
    msg['From'] = self.email_from
    msg['To'] = ', '.join(recipients)

    html_part = MIMEText(html_content, 'html')
    msg.attach(html_part)

    with smtplib.SMTP(self.smtp_server, self.smtp_port) as server:
        server.starttls()
        server.login(self.email_from, self.email_password)
        server.send_message(msg)

def run_daily_report(self):
    """Executa relatório diário completo"""
    data = self.get_daily_data()
    kpis = self.calculate_kpis(data)
    html = self.generate_html_report(data, kpis)

    recipients = os.getenv('REPORT_RECIPIENTS', '').split(',')
    self.send_email(html, recipients)

    print(f"Relatório enviado para {len(recipients)} destinatários")

# Uso
if __name__ == '__main__':
    generator = ReportGenerator()
    generator.run_daily_report()

```

3.3.5 Configuração SMTP

Gmail (Recomendado)

1. Criar App Password: [Google Account Security](#)
2. Configurar variáveis de ambiente:

```

export SMTP_SERVER="smtp.gmail.com"
export SMTP_PORT="587"
export EMAIL_FROM="producao@empresa.com"
export EMAIL_PASSWORD="xxxx xxxx xxxx xxxx" # App Password
export REPORT_RECIPIENTS="supervisor@empresa.com,gestor@empresa.com"

```

Outlook

```

export SMTP_SERVER="smtp-mail.outlook.com"
export SMTP_PORT="587"

```

Gmail Empresarial (Google Workspace)

Mesmo que Gmail pessoal, mas usar email empresarial.

3.3.6 Agendamento Automático

Cron (Linux/Mac)

Editar crontab:

```
crontab -e
```

Adicionar:

```
# Relatório diário às 7h
0 7 * * * cd /path/to/project && python report_generator.py

# Relatório semanal às segundas 9h
0 9 * * 1 cd /path/to/project && python weekly_report.py
```

Windows Task Scheduler

- 1. Abrir Task Scheduler
- 2. Create Basic Task
- 3. Trigger: Daily, 7:00 AM
- 4. Action: Start a program
- 5. Program: python.exe
- 6. Arguments: C:\path\to\report_generator.py

Node-RED (Recomendado)

Usar node inject com cron:

```
{
  "id": "schedule_daily_report",
  "type": "inject",
  "cron": "0 7 * * *",
  "name": "Trigger Daily Report",
  "wires": [[ "exec_python_report" ]]
}
```

3.3.7 Alertas em Tempo Real

Integração Node-RED

Flow para detectar anomalias:

```
// Node: function "Detect Anomalies"
if (msg.payload.dados.paragem > 600) { // 10 min
  msg.alert = {
    tipo: "PARAGEM_LONGA",
    estacao: msg.payload.estacao,
    duracao: msg.payload.dados.paragem,
    severidade: "alta"
  };
  return msg;
}

if (msg.payload.dados.defeitos / msg.payload.dados.producao > 0.05) {
  msg.alert = {
    tipo: "DEFEITOS_ELEVADOS",
    estacao: msg.payload.estacao,
    taxa: (msg.payload.dados.defeitos / msg.payload.dados.producao * 100).toFixed(2),
    severidade: "média"
  };
  return msg;
}
```

```
}  
  
return null; // Sem alerta
```

Conectar a node `exec` que executa:

```
python send_alert_email.py --tipo "${alert.tipo}" --estacao "${alert.estacao}"
```

3.3.8 Exemplo de Email Recebido

Assunto: Relatório de Produção - 2024-12-28

Relatório Diário de Produção

KPIs Gerais:

- ✓ Produção Total: 1,358 peças
- ✓ OEE: 94.5%
- ⚠ Taxa de Defeitos: 1.2%

Produção por Estação:

Estação	Produção	Paragens	Defeitos	Eficiência
Estação A1	458	3 min	5	95.2%
Estação B2	420	8 min	8	92.1%
Estação C1	480	2 min	3	96.8%

Alertas:

- ⚠ Estação B2: Tempo de paragem acima do normal

3.3.9 Melhorias Futuras

- Gráficos embutidos (matplotlib → imagem inline)
- Comparação com dia/semana anterior
- Previsões baseadas em tendências
- Anexar PDF do relatório
- Integração Telegram/Slack para alertas
- Dashboard web para histórico de relatórios

3.3.10 Troubleshooting

Emails não enviados: - Verificar App Password do Gmail - Testar SMTP manualmente: `telnet smtp.gmail.com 587` - Verificar firewall/antivírus

Dados vazios no relatório: - Confirmar que há dados nas últimas 24h no SQLite - Verificar query SQL no script

Cron não executa: - Verificar logs: `grep CRON /var/log/syslog` - Testar script manualmente primeiro - Usar caminhos absolutos no cron

3.3.11 Referências

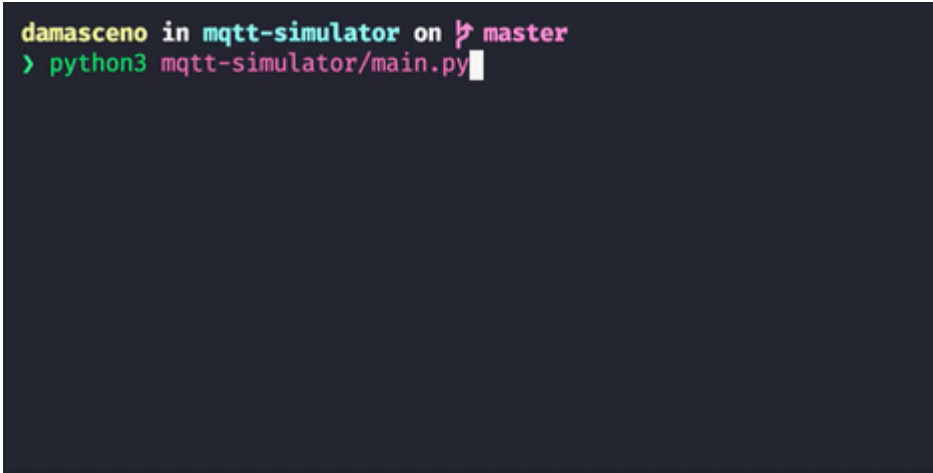
- [Python smtplib](#)
- [Gmail SMTP](#)
- [Cron Syntax](#)


4. Simulador IoT

4.1 MQTT Simulator

A lightweight, easy-to-configure MQTT simulator written in [Python 3](#) for publishing JSON objects to a broker, simulating sensors and devices.

[Features](#) • [Getting Started](#) • [Configuration](#) • [Main contributors](#)



```
damasceno in mqtt-simulator on  master  
> python3 mqtt-simulator/main.py
```

4.1.1 Features

- Lightweight and easy-to-configure simulator for publishing data to an MQTT broker
- Simple setup with a single JSON configuration file
- Publish data on predefined fixed topics
- Publish data on multiple topics that have a variable id or items at the end
- Simulated random variation of data based on configurable parameters
- Real-time event logging during simulation

4.1.2 Getting Started

Running using Python

Run the simulator with the default settings file (`config/settings.json`):

```
python3 mqtt-simulator/main.py
```

Or specify a custom settings file:

```
python3 mqtt-simulator/main.py -f <path/settings.json>
```

To install all dependencies with a virtual environment before using:

```
python3 -m venv venv  
source venv/bin/activate  
pip3 install -r requirements.txt
```

Running using uv

Run the simulator with [uv](#), a fast Python package and project manager - no need to manually setup a virtual environment:

```
uv run mqtt-simulator/main.py -f <path/settings.json>
```

Running using Docker

Additionally, you can run the simulator via [Docker](#) using the provided `Dockerfile`.

Build the image:

```
docker build -t mqtt-simulator .
```

Run the container:

```
docker run mqtt-simulator -f <path/settings.json>
```

4.1.3 Configuration

See the [configuration documentation](#) for detailed usage instructions.

You can also check a full settings file example at: [settings.json](#).

Below is a minimal configuration file that connects to the `mqtt.eclipseprojects.io` broker and publishes data to the `/place/roof` and `/place/basement` topics. The simulator generates `temperature` variations based on the provided parameters:

```
{
  "BROKER_URL": "mqtt.eclipseprojects.io",
  "TOPICS": [
    {
      "TYPE": "list",
      "PREFIX": "place",
      "LIST": ["roof", "basement"],
      "TIME_INTERVAL": 8,
      "DATA": [
        {
          "NAME": "temperature",
          "TYPE": "float",
          "MIN_VALUE": 20,
          "MAX_VALUE": 55,
          "MAX_STEP": 3,
          "RETAIN_PROBABILITY": 0.5,
          "INCREASE_PROBABILITY": 0.6
        }
      ]
    }
  ]
}
```

4.1.4 Sensores Simulados

Sensor	Tipo de dado	Unidade	Intervalo de emissão	Exemplo de payload
Produção	Peças por minuto	pcs/min	5 s	{ "estacao": 1, "producao": 58 }
Paragem	Tempo parado	segundos	evento	{ "estacao": 1, "paragem": 15 }
Stock	Nível de stock	%	10 s	{ "estacao": 1, "stock": 73 }
Qualidade	Peças defeituosas	%	10 s	{ "estacao": 1, "defeitos": 4 }

4.2 Configuration

The MQTT Simulator configuration consists of 3 main sections: **Broker**, **Topics**, and **Data**.

Quick Navigation:

[Broker settings](#) • [Topics settings](#) • [Data settings](#)

You can also check a full settings file example at: [settings.json](#).

4.2.1 Broker settings

The **Broker settings** section is located at the root level of the JSON configuration file and defines the fundamental MQTT connection parameters:

```
{
  "BROKER_URL": "mqtt.eclipse.org",
  "BROKER_PORT": 1883,
  "TOPICS": [
    ...
  ]
}
```

Key	Type	Default	Description
BROKER_URL	string	localhost	The broker URL where the data will be published
BROKER_PORT	number	1883	The port used by the broker
PROTOCOL_VERSION	number	4	Sets the <code>paho.mqtt.client</code> <code>protocol</code> param. Version of the MQTT protocol to use for this client. Can be either <code>3</code> (MQTTv31), <code>4</code> (MQTTv311) or <code>5</code> (MQTTv5)
TLS_CA_PATH	string	None	Sets the <code>paho.mqtt.client.tls_set</code> <code>ca_certs</code> param. String path to the Certificate Authority certificate file
TLS_CERT_PATH	string	None	Sets the <code>paho.mqtt.client.tls_set</code> <code>certfile</code> param. String path to the PEM encoded client certificate file
TLS_KEY_PATH	string	None	Sets the <code>paho.mqtt.client.tls_set</code> <code>keyfile</code> param. String path to the PEM encoded client private keys file
AUTH_USERNAME	string	None	Sets the <code>paho.mqtt.client.username_pw_set</code> <code>username</code> param. Username to authenticate with
AUTH_PASSWORD	string	None	Sets the <code>paho.mqtt.client.username_pw_set</code> <code>password</code> param. Password to authenticate with
CLEAN_SESSION	bool	True	Sets the <code>paho.mqtt.client</code> <code>clean_session</code> param. Boolean that determines the client type. This property is ignored if <code>PROTOCOL_VERSION</code> is <code>5</code> .
RETAIN	bool	False	Sets the <code>paho.mqtt.client.publish</code> <code>retain</code> param. If set to true, the message will be set as the "last known good"/retained message for the topic
QOS	number	2	Sets the <code>paho.mqtt.client.publish</code> <code>qos</code> param. Quality of service level to use
TIME_INTERVAL	number	10	Time interval in seconds between submissions towards the topic
TOPICS	array\	None	Specification of topics and how they will be published

4.2.2 Topics settings

The **TOPICS** key is a list. Each topic entry is an `object` containing parameters that define how topics will be structured and published:

```
{
  "TYPE": "multiple",
  "PREFIX": "place",
  "RANGE_START": 1,
  "RANGE_END": 2,
  "TIME_INTERVAL": 25,
  "DATA": [
    ...
  ]
}
```

Key	Type	Description	Required
TYPE	string	It can be "single", "multiple" or "list"	yes
PREFIX	string	Prefix of the topic URL, depending on the TYPE it can be concatenated to /<id> or /<item>	yes
LIST	array\	When the TYPE is "list" the topic prefix will be concatenated with /<item> for each item in the array	if TYPE is "list"
RANGE_START	number	When the TYPE is "multiple" the topic prefix will be concatenated with /<id> where RANGE_START will be the first number	if TYPE is "multiple"
RANGE_END	number	When the TYPE is "multiple" the topic prefix will be concatenated with /<id> where RANGE_END will be the last number	if TYPE is "multiple"
CLEAN_SESSION	bool	Overwrites the broker level config value and applies only to this Topic	no
RETAIN	bool	Overwrites the broker level config value and applies only to this Topic	no
QOS	number	Overwrites the broker level config value and applies only to this Topic	no
TIME_INTERVAL	number	Overwrites the broker level config value and applies only to this Topic	no
PAYLOAD_ROOT	object	The root set of params to include on all messages	optional
DATA	array\	Specification of the data that will form the JSON to be sent in the topic	yes

4.2.3 Data settings

The key **DATA** inside TOPICS is a list. Each data entry is an `object` containing parameters that define individual data properties and how values are simulated:

```
{
  "NAME": "temperature",
  "TYPE": "float",
  "INITIAL_VALUE": 35,
  "MIN_VALUE": 20,
  "MAX_VALUE": 55,
  "MAX_STEP": 0.2,
  "RETAIN_PROBABILITY": 0.5,
  "RESET_PROBABILITY": 0.1,
  "INCREASE_PROBABILITY": 0.7,
}
```



```
"RESTART_ON_BOUNDARIES": true  
}
```

Key	Type	Description	Required
NAME	string	JSON property name to be sent	yes
TYPE	string	It can be "int", "float", "bool", "math_expression" or "raw_values"	yes
INITIAL_VALUE	same that is returned according to TYPE	Initial value that the property will assume when the simulation starts. If not specified: random for "int", "float" or "bool", and determined by other parameters for "math_expression" or "raw_values"	optional
RETAIN_PROBABILITY	number	Number between 0 and 1 for the probability of the value being retained and sent again	optional, default is 0
RESET_PROBABILITY	number	Number between 0 and 1 for the probability of the value being reset to INITIAL_VALUE	optional, default is 0
MIN_VALUE	number	Minimum value that the property can assume	if TYPE is "int" or "float"
MAX_VALUE	number	Maximum value that the property can assume	if TYPE is "int" or "float"
MAX_STEP	number	Maximum change that can be applied to the property from a published data to the next	if TYPE is "int" or "float"
INCREASE_PROBABILITY	number	Number between 0 and 1 for the probability of the next value being greater than the previous one	optional, default is 0.5 (same probability to increase or decrease). Only valid if TYPE is "int" or "float"
RESTART_ON_BOUNDARIES	bool	When true and the value reaches MAX_VALUE or MIN_VALUE the next value will be the INITIAL_VALUE	optional, default is false. Only valid if TYPE is "int" or "float"
MATH_EXPRESSION	string	Math expression written in a <i>Pythonic</i> way. Also accept functions from Math modules	if TYPE is "math_expression"
INTERVAL_START	number	Minimum value that the MATH_EXPRESSION's variable <code>x</code> can assume	if TYPE is "math_expression"
INTERVAL_END	number	Maximum value that the MATH_EXPRESSION's variable <code>x</code> can assume	if TYPE is "math_expression"
MIN_DELTA	number	Minimum value that can be added to the MATH_EXPRESSION's variable <code>x</code> from a published data to the next	if TYPE is "math_expression"
MAX_DELTA	number	Maximum value that can be added to the MATH_EXPRESSION's variable <code>x</code> from a published data to the next	if TYPE is "math_expression"
INDEX_START	number	The index to start publishing from the VALUES array	optional, default is 0. Only valid if TYPE is "raw_values"
INDEX_END	number	The index to end publishing from the VALUES array	optional, default is <code>len(values) - 1</code> . Only valid if TYPE is "raw_values"
RESTART_ON_END	bool		

Key	Type	Description	Required
		When true and the index of the <code>VALUES</code> array reaches <code>INDEX_END</code> , the next index will be <code>INDEX_START</code> . Otherwise, the param will become inactive and won't be sent after reaching <code>INDEX_END</code>	optional, default is false. Only valid if <code>TYPE</code> is <code>"raw_values"</code>
<code>VALUES</code>	<code>array\</code>	The values to be published in array order	if <code>TYPE</code> is <code>"raw_values"</code>
<code>VALUE_DEFAULT</code>	<code>object</code>	The default value params used or overwritten by params in <code>VALUES</code>	optional, default is <code>{}</code> . Only valid if <code>TYPE</code> is <code>"raw_values"</code> and <code>VALUES</code> is an array\

NOTE: Access [math_expression.md](#) file for more explanations and a example of `TYPE: "math_expression"`.

4.3 MQTT Simulator - Data type math_expression

For general information on how to configure the MQTT Simulator see the [README.md](#) file.

For TYPE: "math_expression" we need five required configuration parameters: MATH_EXPRESSION, INTERVAL_START, INTERVAL_END, MIN_DELTA and MAX_DELTA. Each of these have some notes:

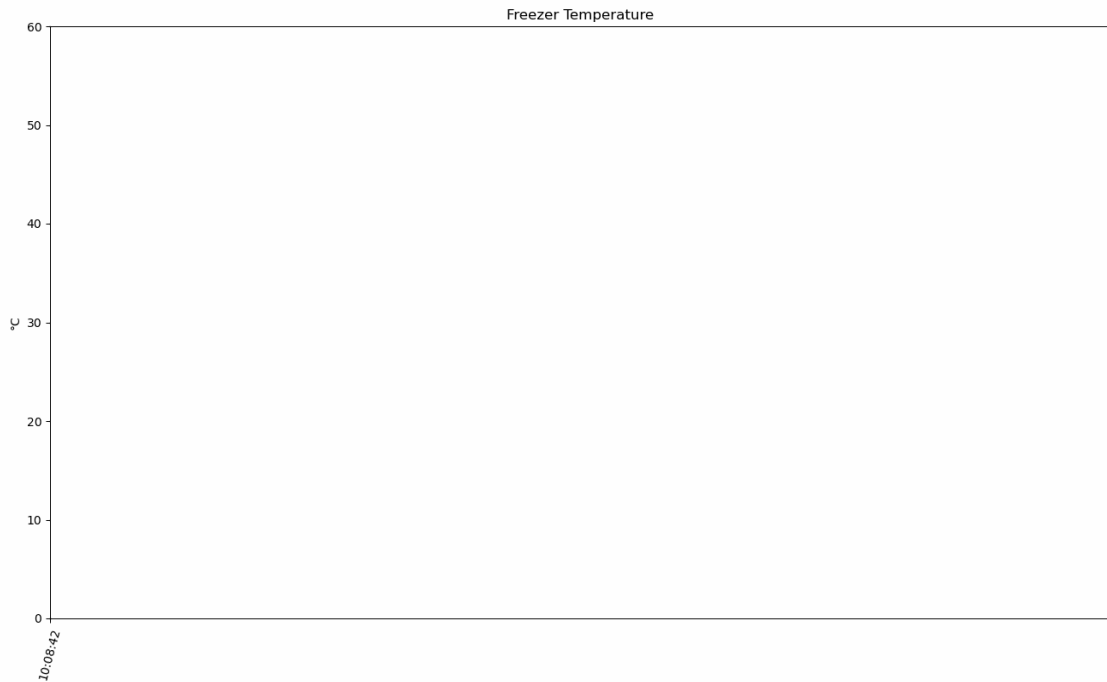
```
{
  ...
  "TYPE": "math_expression",
  "MATH_EXPRESSION": "x**2",
  "INTERVAL_START": 30,
  "INTERVAL_END": 40,
  "MIN_DELTA": 0.1,
  "MAX_DELTA": 0.2,
  ...
}
```

- MATH_EXPRESSION :
-
- The MATH_EXPRESSION 's variable **must** be defined as `x`.
- Any *Pythonic* expression is valid, so, for instance, if you declare it as `x**2` or `math.pow(x,2)` the generated function will be the same.
- INTERVAL_START and INTERVAL_END :
- These parameters will work as the function domain, restricting the value that `x` can reach.
- When the variable `x > INTERVAL_END`, the function will be evaluated, and then the variable will be reset to `x=INTERVAL_START`. So keep in mind that the real interval is `[INTERVAL_START, INTERVAL_END+MAX_DELTA)`.
- MIN_DELTA and MAX_DELTA :
- It is possible to set both with the same value, in this case, it is expected that the curves are more similar between the "loops", and may be identical if `RETAIN_PROBABILITY = 0`.

4.3.1 Example 1 - Freezer Temperature

In the example below the `MATH_EXPRESSION = 2x2+1`, `INTERVAL_START = 0`, `INTERVAL_END = 5`, `MIN_DELTA = 0` and `MAX_DELTA = 0.5`, so it is expected that the generated values are between 1 and 61.5, and the curves should be slightly different.

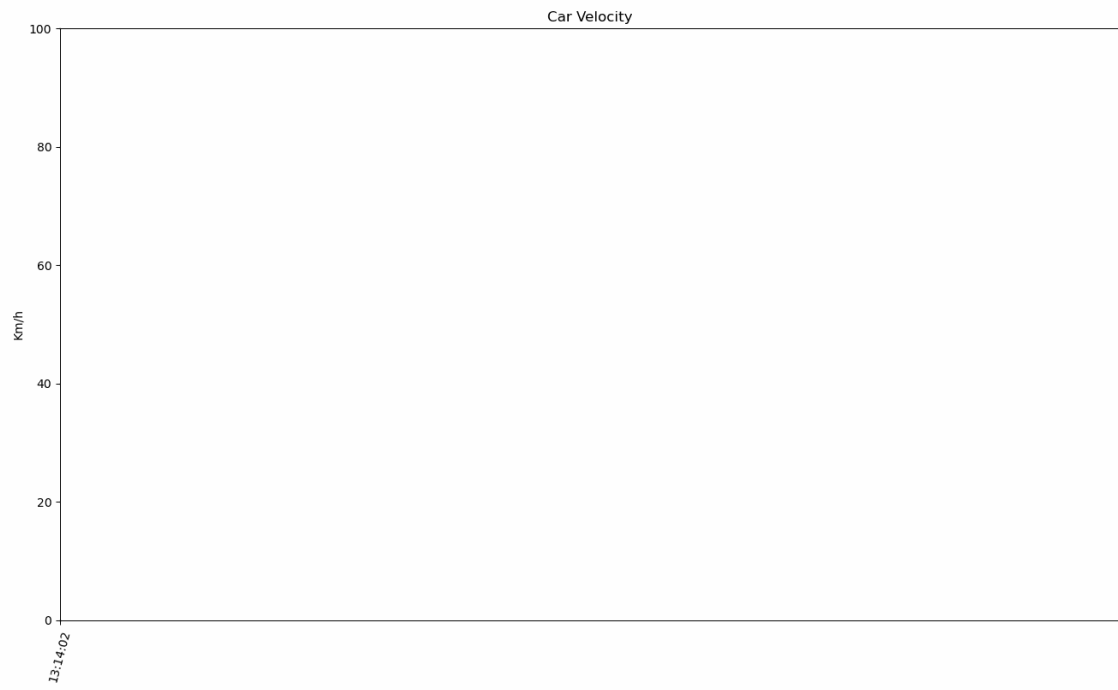
```
{
  "TYPE": "single",
  "PREFIX": "freezer",
  "TIME_INTERVAL": 6,
  "DATA": [
    {
      "NAME": "temperature",
      "TYPE": "math_expression",
      "RETAIN_PROBABILITY": 0.1,
      "MATH_EXPRESSION": "2*math.pow(x,2)+1",
      "INTERVAL_START": 0,
      "INTERVAL_END": 5,
      "MIN_DELTA": 0.5,
      "MAX_DELTA": 0.5
    }
  ]
}
```



4.3.2 Example 2 - Car Velocity

In the example below the $\text{MATH_EXPRESION} = \sqrt{75x}$, $\text{INTERVAL_START} = 0$, $\text{INTERVAL_END} = 100$, $\text{MIN_DELTA} = 10$ and $\text{MAX_DELTA} = 10$, so it is expected that the generated values are between 0 and 87. As $\text{RETAIN_PROBABILITY} = 0$ and the MIN_DELTA and MAX_DELTA are identicals, the curves must be identicals.

```
{
  "TYPE": "single",
  "PREFIX": "car",
  "TIME_INTERVAL": 6,
  "DATA": [
    {
      "NAME": "velocity",
      "TYPE": "math_expression",
      "RETAIN_PROBABILITY": 0,
      "MATH_EXPRESSION": "(75*x)**(1/2)",
      "INTERVAL_START": 0,
      "INTERVAL_END": 100,
      "MIN_DELTA": 10,
      "MAX_DELTA": 10
    }
  ]
}
```



5. Interface

5.1 Dashboard de Visualização

O **Dashboard** do ProjetoISiv1 oferece uma interface web moderna e responsiva para monitorização em tempo real dos dados de produção industrial.

5.1.1 Visão Geral

Tecnologias Utilizadas

Tecnologia	Versão	Propósito
React	18+	Framework frontend
Material-UI (MUI)	5+	Biblioteca de componentes UI
Axios	1.6+	Cliente HTTP para APIs
Chart.js / Recharts	-	Visualizações gráficas
React Router	6+	Navegação entre páginas

Funcionalidades Principais

- Monitorização em tempo real de estações de produção
- Visualização de KPIs (Key Performance Indicators)
- Gráficos interativos de produção, paragens e defeitos
- Histórico de dados com filtros por período
- Interface responsiva (desktop, tablet, mobile)
- Atualizações automáticas via polling/WebSocket

5.1.2 Arquitetura do Dashboard

Estrutura de Componentes

dashboard/	
src/	
components/	
ProductionChart.jsx	# Gráfico de produção
StationStatus.jsx	# Status das estações
DowntimeAnalysis.jsx	# Análise de paragens
DefectsOverview.jsx	# Visão geral de defeitos
KPICard.jsx	# Card de KPI
Header.jsx	# Cabeçalho da aplicação
pages/	
Dashboard.jsx	# Página principal
StationDetails.jsx	# Detalhes de estação
Reports.jsx	# Página de relatórios
services/	
api.js	# Configuração Axios
dataService.js	# Serviços de dados
utils/	
formatters.js	# Formatação de dados
calculations.js	# Cálculos de KPIs
App.jsx	# Componente raiz
index.jsx	# Ponto de entrada
public/	
index.html	
favicon.ico	
package.json	

5.1.3 Componentes Principais

1. ProductionChart - Gráfico de Produção

Exibe a produção ao longo do tempo para cada estação.

Funcionalidades: - Gráfico de linhas/barras interativo - Comparação entre múltiplas estações - Filtros por período (hoje, semana, mês) - Zoom e tooltips detalhados

Exemplo de implementação (Chart.js v3+ / react-chartjs-2):

```
// Exemplo funcional e compatível com Chart.js v3+
// Requer: npm install chart.js react-chartjs-2

import React from 'react';
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  BarElement,
  Title,
  Tooltip,
  Legend
} from 'chart.js';
import { Line } from 'react-chartjs-2';

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  BarElement,
  Title,
  Tooltip,
  Legend
);

const ProductionChart = ({ data }) => {
  const chartData = {
    labels: data.timestamps,
    datasets: [
      {
        label: 'Produção (peças)',
        data: data.production,
        borderColor: 'rgb(75, 192, 192)',
        backgroundColor: 'rgba(75, 192, 192, 0.2)',
        tension: 0.3,
      }
    ]
  };

  const options = {
    responsive: true,
    plugins: {
      legend: { position: 'top' },
      title: { display: false }
    },
    interaction: { mode: 'index', intersect: false },
    scales: {
      x: { display: true },
      y: { display: true, beginAtZero: true }
    }
  };

  return <Line data={chartData} options={options} />;
};

export default ProductionChart;
```

2. StationStatus - Status das Estações

Mostra o status operacional de cada estação em tempo real.

Estados possíveis: - Ativa (verde): Produzindo normalmente - Parada (vermelho): Em paragem - Manutenção (amarelo): Em manutenção - Offline (cinza): Sem comunicação

Exemplo de layout: - Estação A1 — [●] Ativa — Produção: 458 peças | Eficiência: 94% - Estação B2 — [●] Parada — Tempo paragem: 5m 32s

3. DowntimeAnalysis - Análise de Paragens

Analisa os tempos de paragem e identifica padrões.

Métricas exibidas: - Tempo total de paragem - Número de paragens - Duração média de paragem - Paragens por estação (gráfico pizza) - Timeline de paragens

4. DefectsOverview - Visão Geral de Defeitos

Monitoriza a qualidade da produção através de métricas de defeitos.

Visualizações: - Taxa de defeitos (%) - Total de peças defeituosas - Comparação entre estações - Tendência ao longo do tempo - Top 5 estações com mais defeitos

5. KPICard - Cartões de KPI

Exibe indicadores-chave de performance de forma visual.

KPIs Principais:

KPI	Descrição	Cálculo
OEE	Overall Equipment Effectiveness	(Disponibilidade × Performance × Qualidade) × 100%
Taxa de Produção	Peças produzidas/hora	Total Produção / Horas Operação
Taxa de Defeitos	Percentagem de defeitos	(Defeitos / Total Produção) × 100%
Tempo Médio de Paragem	MTTR (Mean Time To Repair)	Total Paragem / Número de Paragens
Disponibilidade	Tempo disponível vs. paragem	((Tempo Total - Paragem) / Tempo Total) × 100%

Exemplo visual: OEE — 94.5% ± 2.3%

5.1.4 Páginas do Dashboard

Página Principal (Dashboard)

Layout: - Cabeçalho: [Logo] Monitorização Industrial | [User] [Settings] - Cards de KPIs (OEE, Produção, Defeitos, Paragens) - Gráfico de Produção (linha) - Status das Estações - Análise de Paragens (barra) - Taxa de Defeitos (pizza)

Página de Detalhes de Estação

Exibe informação detalhada sobre uma estação específica.

Secções: 1. Informação Geral (nome, localização, capacidade) 2. Métricas em Tempo Real 3. Histórico de Produção (últimas 24h) 4. Log de Eventos (paragens, manutenções) 5. Configurações da Estação

Página de Relatórios

Lista relatórios gerados e permite download.

Funcionalidades: - Filtrar por tipo (diário, semanal, mensal) - Filtrar por período - Download em PDF/HTML - Reenviar por email - Gerar novo relatório ad-hoc

5.1.5 Integração com Backend

Configuração da API

Ficheiro: `src/services/api.js`

```
import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://localhost:8000';

const api = axios.create({
  baseURL: API_BASE_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});

export default api;
```

Serviço de Dados

Ficheiro: `src/services/dataService.js`

```
import api from './api';

export const dataService = {
  // Obter dados de todas as estações
  getStations: async () => {
    const response = await api.get('/stations');
    return response.data;
  },

  // Obter dados acumulados
  getAccumulatedData: async (stationId, startDate, endDate) => {
    const response = await api.get('/accumulated-data', {
      params: { stationId, startDate, endDate }
    });
    return response.data;
  },

  // Obter KPIs
  getKPIs: async (stationId) => {
    const response = await api.get(`/kpis/${stationId}`);
    return response.data;
  },
};
```

Atualização em Tempo Real

Opção 1: Polling

```
useEffect(() => {
  const interval = setInterval(async () => {
    const data = await dataService.getStations();
    setStationsData(data);
  }, Number(process.env.REACT_APP_REFRESH_INTERVAL) || 5000);

  return () => clearInterval(interval);
}, []);
```

Opção 2: WebSocket (futuro)

```
useEffect(() => {
  const ws = new WebSocket(process.env.REACT_APP_MQTT_BROKER || 'ws://localhost:9001');

  ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    updateDashboard(data);
  };

  return () => ws.close();
}, []);
```

5.1.6 Instalação e Configuração

1. Instalar dependências

```
cd dashboard
npm install
# instalar chart.js e react-chartjs-2 se não estiverem:
npm install chart.js react-chartjs-2
```

1. Configurar variáveis de ambiente

Crie `.env` :

```
REACT_APP_API_URL=http://localhost:8000
REACT_APP_REFRESH_INTERVAL=5000
REACT_APP_MQTT_BROKER=ws://localhost:9001
```

1. Executar em desenvolvimento

```
npm start
```

Acesse em: <http://localhost:3000>

1. Build para produção

```
npm run build
```

Ficheiros gerados em `build/`.

1. Deploy

Com servidor estático:

```
npm install -g serve
serve -s build -p 3000
```

Com Nginx (exemplo):

```
server {
    listen 80;
    server_name dashboard.exemplo.com;

    root /var/www/dashboard/build;
    index index.html;

    location / {
        try_files $uri /index.html;
    }

    location /api {
        proxy_pass http://localhost:8000;
    }
}
```

5.1.7 Personalização

Tema Material-UI

Ficheiro: `src/theme.js`

```
import { createTheme } from '@mui/material/styles';

const theme = createTheme({
  palette: {
    primary: { main: '#1976d2' },
    secondary: { main: '#dc004e' },
    success: { main: '#4caf50' },
    warning: { main: '#ff9800' },
    error: { main: '#f44336' },
  },
  typography: { fontFamily: 'Roboto, Arial, sans-serif' },
});
```

```
export default theme;
```

Configurar Cores por Status

```
const statusColors = {
  ativa: '#4caf50', // Verde
  parada: '#f44336', // Vermelho
  manutencao: '#ff9800', // Amarelo
  offline: '#9e9e9e', // Cinza
};
```

5.1.8 Grafana (Alternativa)

Como alternativa ao dashboard React, pode-se utilizar **Grafana** para visualizações avançadas.

Vantagens do Grafana: - Dashboards configuráveis sem código - Alertas personalizados - Múltiplas fontes de dados - Painéis pré-construídos - Partilha e exportação de dashboards

Configuração básica: 1. Instalar Grafana 2. Adicionar SQLite (ou outra fonte) como data source 3. Criar dashboard com painéis 4. Configurar refresh automático 5. Configurar alertas

Exemplo Docker:

```
grafana:
  image: grafana/grafana:latest
  ports:
    - "3000:3000"
  volumes:
    - grafana-storage:/var/lib/grafana
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=admin
```

5.1.9 Troubleshooting

Dashboard não carrega dados: - Verificar se a API está a correr (`http://localhost:8000`) - Verificar configuração de CORS no backend - Inspeccionar console do browser para erros

Gráficos não aparecem: - Verificar se os dados têm o formato correto - Instalar dependências de chart: `npm install chart.js react-chartjs-2` - Verificar imports e inicialização do Chart.js (ChartJS.register)

Performance lenta: - Reduzir intervalo de atualização - Implementar paginação de dados - Utilizar React.memo para componentes pesados - Otimizar queries de backend

5.1.10 Próximos Passos

Ver [Fase 2](#) para melhorias planeadas: - WebSocket para atualizações em tempo real - Autenticação de utilizadores - Dashboards personalizáveis por utilizador - Exportação de dados - Notificações push - Modo escuro (dark mode)

5.1.11 Referências

- React Documentation: <https://react.dev/>
- Material-UI: <https://mui.com/>
- Chart.js: <https://www.chartjs.org/>
- Recharts: <https://recharts.org/>
- Grafana: <https://grafana.com/docs/>

6. Deployment

6.1 Pré-requisitos e Instalação

Esta secção descreve as dependências necessárias e o processo de instalação do sistema de **Monitorização Industrial com IoT e ETL**, incluindo a configuração dos componentes MQTT, Node-RED, SQLite, React UI e automação com Python.

6.1.1 Pré-requisitos

Antes de executar o projeto, certifica-te de que tens as seguintes ferramentas instaladas:

Sistema Base

- **Python 3.10+** — para scripts de simulação, integração com API e envio de emails
- **Node.js 18+** — para execução da interface React e Node-RED
- **npm ou yarn** — gestor de pacotes JavaScript
- **SQLite3** — para armazenamento local dos dados processados
- **Mosquitto MQTT Broker** — para gerir a comunicação entre sensores (publishers) e consumidores (subscribers)
- **Git** — para controlo de versões e clonagem do repositório

Bibliotecas Python

Instala as bibliotecas necessárias:

```
pip install paho-mqtt requests smtplib sqlite3
```

Dependências Node.js

```
npm install @mui/material @emotion/react @emotion/styled axios
```

Node-RED

Instala o Node-RED globalmente:

```
npm install -g node-red
```

Depois, inicia com:

```
node-red
```

Comandos Git

[Geeks git Commands](#)

Gerar Documentação com MkDocs

```
pip install mkdocs mkdocs-material  
mkdocs serve
```

6.2 Troubleshooting

Problema	Causa provável	Solução
Node-RED não liga	Porta ocupada	Muda a porta: <code>node-red -p 1881</code>
MQTT não conecta	Broker offline	Reinicia Mosquitto: <code>sudo systemctl restart mosquitto</code>
SQLite bloqueado	Conflito de acesso	Fecha outras conexões e tenta novamente
Emails não enviados	Autenticação falhou	Verifica SMTP e permissões "App Password"

7. Projeto

7.1 Conclusão e Trabalhos Futuros

O **ProjetoISlv1** demonstrou a aplicação prática de conceitos de **Integração de Sistemas de Informação** com recurso a ferramentas de ETL, comunicação IoT e visualização web.

7.1.1 Resultados

- Automação da recolha de dados via sensores simulados;
 - Processamento em tempo real com Node-RED;
 - Armazenamento persistente em SQLite;
 - Geração de relatórios automáticos;
 - Visualização em dashboard web.
-

7.1.2 Trabalhos Futuros

- Migrar a base de dados para **SQL Server**;
 - Usar docker para todo o sistema (Mosquitto, Node-RED, UI, DB);
 - Adicionar predições com **IA/ML**;
 - Criar alertas em tempo real (SMS/Telegram).
-

7.2 Roadmap Completo - Plataforma de Simulação IoT

Projeto: Sistema de Simulação IoT sob Demanda
Objetivo: API REST que provisiona simuladores MQTT dinamicamente
Stack Base: Python + FastAPI + Docker + MQTT
Duração Total: 12-16 semanas (3 fases)

7.2.1 Visão Geral

Fase 1 (MVP) Semanas 1-4	Fase 2 (Kubernetes) Semanas 5-10	Fase 3 (Produção) Semanas 11-16
<ul style="list-style-type: none">API RESTDocker LocalSQLiteDemo Funcional	<ul style="list-style-type: none">K8s ClientPods DinâmicosPostgreSQLMulti-User	<ul style="list-style-type: none">CI/CDMonitoringAuth/BillingDeploy Cloud

Estado Atual: Fase 1 - 70% completo (API + Docker + SQLite funcionais)

7.2.2 FASE 1 - MVP Funcional (4 semanas)

Objetivo: Sistema local funcional para demonstração universitária
Entregáveis: API REST + Docker + SQLite + Documentação + Demo

Semana 1: Core Implementation (ATUAL)

Status: 🔄 Em Progresso

DIA 1-2: FINALIZAR INTEGRAÇÃO DOCKER

- [x] Pull da imagem `ghcr.io/damascenorafael/mqtt-simulator:sha-a73a2e8`
- [x] API cria containers com config JSON
- [x] Adicionar flag `-f` no comando Docker
- [] **Testar:** Container lê config corretamente
- [] **Validar:** `docker logs` mostra tópicos configurados

Código-chave:

```
container = docker_client.containers.run(
    "ghcr.io/damascenorafael/mqtt-simulator:sha-a73a2e8",
    command=["-f", "/config/settings.json"],
    volumes={config_path: {'bind': '/config/settings.json', 'mode': 'ro'}},
    detach=True,
    remove=True
)
```

DIA 3-4: PERSISTÊNCIA SQLITE

- [x] Modelo `Simulation` com SQLAlchemy
- [x] Endpoints CRUD completos
- [] **Corrigir:** Health check database "disconnected"
- [] **Adicionar:** Campo `expires_at` na tabela
- [] **Implementar:** Cleanup de expirados ao startup

Tasks específicas:

```
# Recriar BD com novo campo
rm simulations.db
python3 -c "from database import Base, engine; Base.metadata.create_all(bind=engine)"
```



```
# Adicionar ao database.py
expires_at = Column(DateTime, nullable=False)

# Adicionar ao main.py startup
@app.on_event("startup")
async def startup_event():
    cleanup_expired_simulations()
```

DIA 5-7: AUTO-STOP E CLEANUP

- [x] Thread para auto-stop após duração
- [] **Implementar:** Função `cleanup_expired_simulations()`
- [] **Testar:** Criar simulação 2 min → para automaticamente
- [] **Testar:** Reiniciar API → containers órfãos limpos
- [] **Validar:** BD atualiza status para "expired"

Checklist de Validação: - [] Container para após `duration_minutes` - [] BD atualiza `stopped_at` e `status='expired'` - [] Ficheiro config temporário é removido - [] Containers órfãos limpos ao reiniciar API

Semana 2: Robustez e Error Handling

DIA 8-9: TRATAMENTO DE ERROS

- [] Try/catch robusto em todos endpoints
- [] Validação Pydantic completa (todos campos obrigatórios)
- [] Mensagens de erro descritivas
- [] HTTP status codes corretos (404, 422, 500)

Endpoints a revisar:

```
# POST /simulations
- 422: Config JSON inválido
- 404: Docker image não encontrada
- 500: Erro ao criar container

# GET /simulations/{id}
- 404: Simulação não existe
- 200: Retorna logs mesmo se container parou

# DELETE /simulations/{id}
- 404: Simulação não existe
- 400: Simulação já parada
- 204: Sucesso
```

DIA 10-11: FEATURES ADICIONAIS

- [] Endpoint `GET /simulations?status=running&limit=20`
- [] Endpoint `GET /stats` (total, running, stopped, expired)
- [] Endpoint `GET /simulations/{id}/logs` (últimas 100 linhas)
- [] Response models Pydantic para todos endpoints

Código exemplo:

```
@app.get("/simulations", response_model=List[SimulationListItem])
async def list_simulations(
    status: Optional[str] = None,
    limit: int = Query(default=20, ge=1, le=100),
    offset: int = Query(default=0, ge=0),
    db: Session = Depends(get_db)
):
    query = db.query(Simulation)

    if status:
        query = query.filter(Simulation.status == status)

    total = query.count()
    sims = query.order_by(Simulation.created_at.desc()).offset(offset).limit(limit).all()

    return {
```

```

    "total": total,
    "limit": limit,
    "offset": offset,
    "simulations": [...]
  }

```

DIA 12-14: TESTES MANUAIS COMPLETOS

- [] Criar 5 configs JSON diferentes (single, multiple, list topics)
- [] Testar cada endpoint com Postman/curl
- [] Stress test: criar 10 simulações simultâneas
- [] Load test: criar → parar → criar 50x
- [] Verificar memory leaks (containers não limpos)

Checklist de Testes:

```

# Test 1: Happy path
curl -X POST http://localhost:8000/simulations -d @config.json
# Verificar: container ativo, BD tem entrada, logs funcionam

# Test 2: Config inválido
curl -X POST http://localhost:8000/simulations -d '{"invalid": true}'
# Verificar: retorna 422 com mensagem clara

# Test 3: Listar e filtrar
curl http://localhost:8000/simulations?status=running
# Verificar: só mostra simulações ativas

# Test 4: Auto-stop
# Criar simulação 1 min, esperar → container deve parar

# Test 5: Cleanup após restart
# Criar 3 simulações, matar API (Ctrl+C), reiniciar
# Verificar: simulações expiradas têm status correto

```

Semana 3: Documentação e Cliente de Teste

DIA 15-16: README.MD COMPLETO

- [] Seção "Instalação" passo-a-passo
- [] Seção "Configuração" (Docker, SQLite)
- [] Seção "Uso" com exemplos de curl/Postman
- [] Seção "Arquitetura" com diagrama
- [] Seção "API Reference" (ou link para Swagger)
- [] Screenshots do Swagger UI

Template README.md:

```

# IoT Simulator Platform

API REST para criar simuladores IoT sob demanda usando Docker.

## Features
- ☒ Criação dinâmica de simuladores MQTT
- ☒ Configuração JSON flexível
- ☒ Auto-stop após duração especificada
- ☒ Persistência SQLite
- ☒ Cleanup automático de containers órfãos

## Instalação

### Requisitos
- Python 3.11+
- Docker
- pip

### Setup
```bash
git clone <repo>
cd iot-simulator-api
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

```

Executar

```
uvicorn main:app --reload --port 8000
```

Aceder: <http://localhost:8000/docs>

7.2.3 Uso

Criar Simulação

```
curl -X POST http://localhost:8000/simulations \
-H "Content-Type: application/json" \
-d '{
 "BROKER_URL": "test.mosquitto.org",
 "BROKER_PORT": 1883,
 "TIME_INTERVAL": 10,
 "duration_minutes": 30,
 "TOPICS": [...]
}'
```

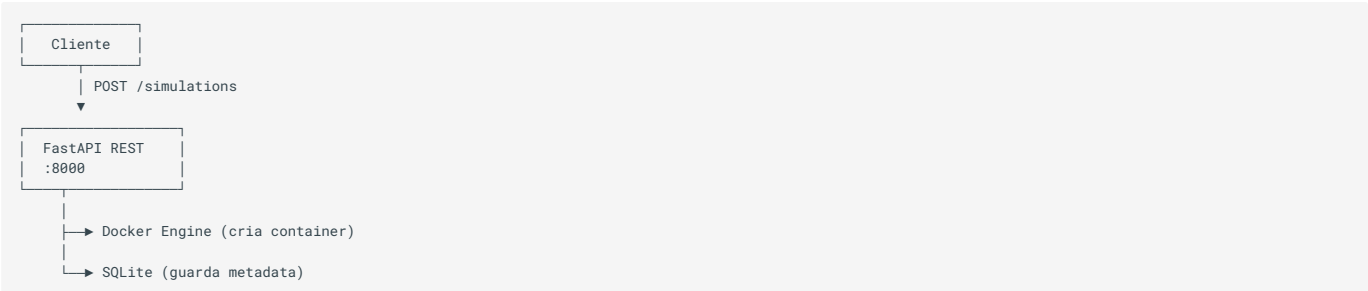
Listar Simulações

```
curl http://localhost:8000/simulations
```

Ver Logs

```
curl http://localhost:8000/simulations/{id}/logs
```

7.2.4 Arquitetura



7.2.5 Exemplos de Config

Ver pasta `examples/` para configs prontos: - `factory.json` - Simulação industrial - `agriculture.json` - Sensores agrícolas - `smart-home.json` - Casa inteligente

7.2.6 Troubleshooting

Erro: Database disconnected

```
rm simulations.db
python3 -c "from database import Base, engine; Base.metadata.create_all(bind=engine)"
```

**Container não para** - Verificar `duration_minutes` foi especificado - Ver logs da API para mensagens "[AUTO-STOP]"

```
Dia 17-18: Cliente HTML de Teste
- [] HTML básico para conectar a broker MQTT
- [] Usar MQTT.js para subscrever tópicos
- [] Mostrar mensagens em tempo real
- [] Adicionar gráfico Chart.js (opcional)

Ficheiro `test-client/index.html`:
```html
<!DOCTYPE html>
<html>
<head>
```

```

<title>IoT Simulator - Test Client</title>
<script src="https://unpkg.com/mqtt@5.3.5/dist/mqtt.min.js"></script>
<style>
  body { font-family: Arial; max-width: 800px; margin: 50px auto; }
  input { padding: 10px; width: 300px; }
  button { padding: 10px 20px; }
  #messages {
    border: 1px solid #ccc;
    height: 400px;
    overflow-y: scroll;
    padding: 10px;
    margin-top: 20px;
    font-family: monospace;
  }
  .message { padding: 5px; border-bottom: 1px solid #eee; }
</style>
</head>
<body>
  <h1> 🍷 IoT Simulator Test Client</h1>

  <div>
    <input id="broker" placeholder="Broker (ex: test.mosquitto.org)" value="test.mosquitto.org">
    <input id="topic" placeholder="Topic pattern (ex: fabrica/#)" value="demo/#">
    <button onclick="connect()">Connect</button>
    <button onclick="disconnect()">Disconnect</button>
    <button onclick="clearMessages()">Clear</button>
  </div>

  <div id="status">Disconnected</div>
  <div id="messages"></div>

  <script>
    let client = null;

    function connect() {
      const broker = document.getElementById('broker').value;
      const topic = document.getElementById('topic').value;

      if (client) client.end();

      document.getElementById('status').textContent = 'Connecting...';

      client = mqtt.connect(`ws://${broker}:8080`);

      client.on('connect', () => {
        document.getElementById('status').textContent = '✅ Connected';
        client.subscribe(topic);
        addMessage(`Subscribed to: ${topic}`, 'info');
      });

      client.on('message', (t, payload) => {
        try {
          const data = JSON.parse(payload.toString());
          addMessage(`${t}: ${JSON.stringify(data, null, 2)}`, 'data');
        } catch {
          addMessage(`${t}: ${payload.toString()}`, 'data');
        }
      });

      client.on('error', (err) => {
        document.getElementById('status').textContent = '❌ Error';
        addMessage(`Error: ${err.message}`, 'error');
      });
    }

    function disconnect() {
      if (client) {
        client.end();
        document.getElementById('status').textContent = 'Disconnected';
        addMessage('Disconnected', 'info');
      }
    }

    function clearMessages() {
      document.getElementById('messages').innerHTML = '';
    }

    function addMessage(msg, type) {
      const div = document.createElement('div');
      div.className = `message ${type}`;
      div.textContent = `[${new Date().toLocaleTimeString()}] ${msg}`;
      document.getElementById('messages').appendChild(div);
      div.scrollIntoView();
    }
  </script>
</body>
</html>

```

DIA 19-21: CONFIGS DE EXEMPLO

- [] examples/factory.json - Linha de produção

- [] `examples/agriculture.json` - Sensores agrícolas
- [] `examples/smart-home.json` - Casa inteligente
- [] `examples/fleet.json` - Gestão de frota (GPS)

Exemplo `examples/factory.json`:

```
{
  "BROKER_URL": "test.mosquitto.org",
  "BROKER_PORT": 1883,
  "TIME_INTERVAL": 5,
  "duration_minutes": 10,
  "TOPICS": [
    {
      "TYPE": "multiple",
      "PREFIX": "fabrica/maquina",
      "RANGE_START": 1,
      "RANGE_END": 5,
      "DATA": [
        {
          "NAME": "producao_unidades",
          "TYPE": "int",
          "MIN_VALUE": 50,
          "MAX_VALUE": 100,
          "MAX_STEP": 5,
          "INCREASE_PROBABILITY": 0.6,
          "RETAIN_PROBABILITY": 0.7
        },
        {
          "NAME": "temperatura_motor",
          "TYPE": "float",
          "MIN_VALUE": 40.0,
          "MAX_VALUE": 85.0,
          "MAX_STEP": 2.5,
          "INCREASE_PROBABILITY": 0.5,
          "RETAIN_PROBABILITY": 0.8
        },
        {
          "NAME": "defeitos",
          "TYPE": "int",
          "MIN_VALUE": 0,
          "MAX_VALUE": 3,
          "MAX_STEP": 1,
          "INCREASE_PROBABILITY": 0.2,
          "RETAIN_PROBABILITY": 0.95
        }
      ]
    }
  ]
}
```

Semana 4: Apresentação e Demo

DIA 22-23: SLIDES DE APRESENTAÇÃO

- [] Slide 1: Título + Nome + Data
- [] Slide 2-3: Problema e Contexto
- [] Slide 4: Solução Proposta
- [] Slide 5: Arquitetura Técnica
- [] Slide 6: Stack Tecnológica
- [] Slide 7-8: Demo ao Vivo
- [] Slide 9: Features Implementadas
- [] Slide 10: Roadmap Futuro (Fase 2-3)
- [] Slide 11: Conclusão
- [] Slide 12: Q&A

Template Slides (Markdown → reveal.js):

```
---
title: Plataforma de Simulação IoT
author: [Teu Nome]
date: 2025
---
```

```
# Plataforma de Simulação IoT
## API REST para Simulação sob Demanda

[Teu Nome]
Projeto ISI - 2024/2025

---

## O Problema

- Desenvolvimento IoT **custa 45.000-500.000 USD**
- Setup de hardware **demora semanas**
- Impossível testar cenários extremos
- Riscos: danificar equipamento caro

**Solução:** Virtualizar sensores

---

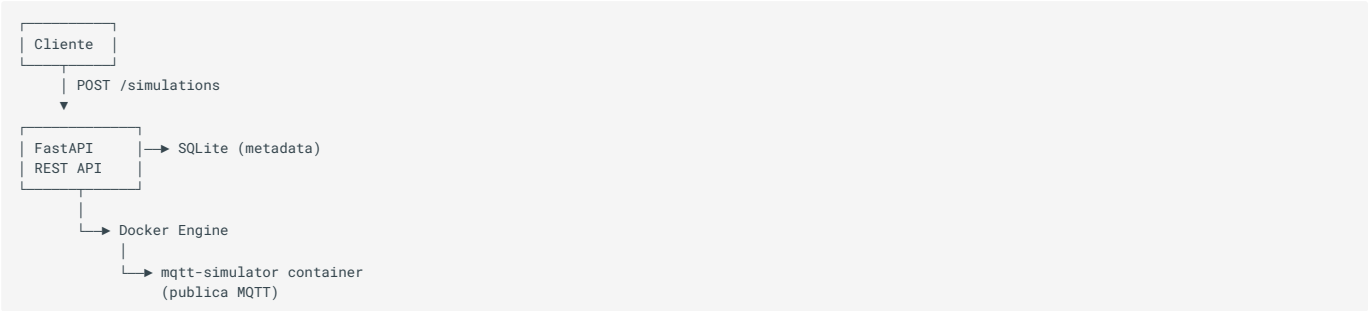
## Solução Proposta

**API REST** que cria **simuladores MQTT** dinamicamente

```bash
POST /simulations + JSON config
↓
Container Docker inicia
↓
Dados MQTT em tempo real
```

**Benefícios:** - Setup em < 1 minuto - Custo: 0 EUR (usa broker público) - Cenários impossíveis no mundo real

7.2.7 Arquitetura



7.2.8 Stack Tecnológica

- Backend:** - Python 3.11 - FastAPI (API REST) - SQLAlchemy + SQLite - Docker SDK
- Simulador:** - mqttsimulator (imagem existente) - MQTT protocol - Random walk data generation
- Tools:** - Swagger UI (documentação automática) - Postman (testes)

7.2.9 Demo ao Vivo

1. Abrir Swagger UI
2. POST criar simulação (factory.json)
3. Ver container ativo: `docker ps`
4. Ver logs: `docker logs <id>`
5. Abrir cliente HTML
6. Ver dados MQTT em tempo real
7. Simulação para automaticamente

### 7.2.10 Features Implementadas

- ☒ API REST completa (CRUD)
- ☒ Persistência SQLite
- ☒ Auto-stop após duração
- ☒ Cleanup automático
- ☒ Swagger UI
- ☒ Tratamento de erros robusto
- ☒ Cliente de teste HTML
- ☒ Configs de exemplo prontos




### 7.2.11 Roadmap Futuro

- Fase 2 (4-6 semanas):** - Kubernetes para multi-user - PostgreSQL - Isolamento por utilizador
- Fase 3 (8+ semanas):** - Autenticação JWT - Billing e quotas - Monitoring (Prometheus/Grafana) - Deploy cloud (AWS/GCP)

### 7.2.12 Conclusão

- Objetivo Alcançado:** - ☒ MVP funcional - ☒ Demonstra conceito - ☒ Código limpo e documentado - ☒ Extensível (roadmap claro)
- Aprendizagens:** - FastAPI + Docker integration - SQLAlchemy ORM - MQTT protocol - Container lifecycle management

### 7.2.13 Perguntas?

-  [teu-email]@example.com
-  GitHub: [link-repo]
-  Documentação: http://localhost:8000/docs

```
Dia 24-25: Script e Ensaio de Demo
- [] Escrever script palavra-a-palavra (7-10 minutos)
- [] Ensaiar 3x sozinho
- [] Gravar vídeo backup (se demo falhar)
- [] Preparar dados de teste (não improvisar)

Script de Demo (7 minutos):
```

[00:00-01:00] Introdução "Bom dia. Hoje vou apresentar uma plataforma de simulação IoT que permite testar aplicações sem hardware físico.

O problema: desenvolver IoT custa entre 45 mil e 500 mil dólares, e demora semanas só para configurar sensores.

A minha solução: API REST que cria simuladores virtuais em segundos."

[01:00-02:00] Mostrar Arquitetura "A arquitetura é simples: FastAPI recebe config JSON, cria container Docker com o simulador, e publica dados via MQTT.

Stack: Python, FastAPI, Docker, SQLite para persistência."

[02:00-05:00] Demo ao Vivo "Vou demonstrar. Aqui está o Swagger UI da API.

1. POST /simulations - vou criar uma simulação de fábrica [Colar factory.json pré-preparado]
2. Executar... sucesso! Retornou simulation\_id e container\_id.
3. Ver container ativo [terminal: docker ps] Aqui está, nome sim-abc123, a correr.
4. Ver logs [terminal: docker logs sim-abc123] Dados sendo publicados: fabrica/maquina1, maquina2...
5. Abrir cliente HTML [browser] Conectar ao broker... subscribed! Dados chegam em tempo real, aqui temperatura, produção, defeitos.
6. Listar simulações [Swagger GET /simulations] Aparece na base de dados, status running.
7. Esta simulação está configurada para 2 minutos... [esperar ou cortar] ...e para automaticamente.
8. Se reiniciar a API [Ctrl+C, uvicorn main:app] Containers órfãos são limpos ao startup."

[05:00-06:00] Código "Rapidamente o código: main.py tem ~300 linhas. POST /simulations valida JSON com Pydantic, cria container Docker, guarda na BD, agenda auto-stop.

Database.py - modelo SQLAlchemy simples, SQLite para persistência."

[06:00-07:00] Conclusão "Resumindo: sistema funcional que demonstra o conceito. MVP completo com persistência, auto-stop, cleanup.

Próximos passos seriam adicionar Kubernetes para multi-user, autenticação, billing.

Mas para esta fase, objetivo alcançado: API REST que cria simuladores sob demanda. Obrigado."

```
Dia 26-28: Buffer e Polimento Final
- [] Corrigir bugs encontrados durante ensaios
- [] Melhorar mensagens de erro
- [] Adicionar logs mais descritivos
- [] Verificar todos requirements.txt
- [] Limpar código commented out
- [] Formatar código (black, autopep8)

Checklist Pré-Apresentação:
```bash
# Código
- [ ] Sem erros ao iniciar: uvicorn main:app
- [ ] Swagger abre: http://localhost:8000/docs
- [ ] Health check OK: curl http://localhost:8000/health
- [ ] Criar simulação funciona com factory.json
- [ ] Cliente HTML conecta e mostra dados

# Documentação
- [ ] README.md completo e sem typos
- [ ] Comentários em funções críticas
- [ ] requirements.txt atualizado

# Demo
- [ ] factory.json testado e funciona
- [ ] Cliente HTML funciona em browser fresh
- [ ] Script impresso ou em tablet
- [ ] Vídeo backup gravado e testado
- [ ] Slides exportados em PDF

# Contingência
- [ ] Laptop carregado
- [ ] Segundo laptop disponível
- [ ] Internet backup (hotspot telemóvel)
- [ ] Docker images pré-downloaded
```

7.2.14 📌 MARCO: Apresentação Fase 1

Entregáveis Mínimos: - ☒ Código fonte funcionando - ☒ README.md - ☒ Slides apresentação - ☒ Demo ao vivo (ou vídeo backup)

Critérios de Sucesso: - API cria simulações com config JSON - Containers Docker iniciam e publicam MQTT - Auto-stop funciona - Persistência SQLite funciona - Cleanup de órfãos funciona ao reiniciar

7.2.15 FASE 2 - Kubernetes Multi-User (6 semanas)

Objetivo: Migrar para Kubernetes, suportar múltiplos utilizadores

Pré-requisitos: Fase 1 100% funcional, conhecimento básico K8s

Semana 5-6: Setup Kubernetes

APRENDER KUBERNETES BÁSICO

- [] Curso online (2-3 dias): Kubernetes for Beginners
- [] Conceitos: Pods, Deployments, Services, Namespaces
- [] kubectl comandos básicos
- [] Instalar minikube ou kind para testes locais

Recursos: - Kubernetes.io tutorials - "Kubernetes Up & Running" (livro) - KodeKloud free tier

SETUP CLUSTER LOCAL

- [] Instalar minikube: `brew install minikube`
- [] Iniciar cluster: `minikube start`
- [] Testar: `kubectl get nodes`
- [] Criar namespace: `kubectl create namespace simulations`

KUBERNETES PYTHON CLIENT

- [] Instalar: `pip install kubernetes`
- [] Testar conexão:

```
from kubernetes import client, config
config.load_kube_config()
v1 = client.CoreV1Api()
print(v1.list_pod_for_all_namespaces())
```

Semana 7-8: Migrar Docker → Kubernetes

CRIAR PODS DINAMICAMENTE

- [] Refactor `create_simulation()` para usar K8s API
- [] Criar Pod spec para simulador
- [] Montar ConfigMap com JSON config
- [] Testar criação de Pod

Código-chave:

```
from kubernetes import client, config

config.load_kube_config()
k8s = client.CoreV1Api()

def create_simulator_pod(sim_id: str, config_json: dict):
    # Criar ConfigMap
    configmap = client.V1ConfigMap(
        metadata=client.V1ObjectMeta(name=f"config-{sim_id}"),
        data={"settings.json": json.dumps(config_json)}
    )
    k8s.create_namespaced_config_map("simulations", configmap)

    # Criar Pod
    pod = client.V1Pod(
        metadata=client.V1ObjectMeta(name=f"sim-{sim_id}"),
        spec=client.V1PodSpec(
            containers=[
                client.V1Container(
                    name="simulator",
                    image="ghcr.io/damascenorafael/mqtt-simulator:sha-a73a2e8",
                    command=["-f", "/config/settings.json"],
                    volume_mounts=[
                        client.V1VolumeMount(
```

```

        name="config",
        mount_path="/config"
    )
    ]
    )
    ],
    volumes=[
        client.V1Volume(
            name="config",
            config_map=client.V1ConfigMapVolumeSource(
                name=f"config-{sim_id}"
            )
        )
    ],
    restart_policy="Never"
)
)

k8s.create_namespaced_pod("simulations", pod)
return f"sim-{sim_id}"

```

EXPOR PODS VIA SERVICE

- [] Criar Service para acesso externo
- [] Testar acesso MQTT via NodePort
- [] Documentar endpoint para utilizadores

CLEANUP EM KUBERNETES

- [] Implementar `delete_pod()` com K8s API
- [] Adicionar `activeDeadlineSeconds` aos Pods
- [] CronJob para limpar Pods expirados

Semana 9-10: PostgreSQL e Multi-User

MIGRAR SQLITE → POSTGRESQL

- [] Instalar PostgreSQL (Docker ou managed)
- [] Migração schema (manual ou Alembic)
- [] Atualizar connection string
- [] Testar CRUD operations

Migration script:

```

# migrate_to_postgres.py
import sqlite3
import psycopg2

# Ler de SQLite
sqlite_conn = sqlite3.connect('simulations.db')
sqlite_cursor = sqlite_conn.cursor()
rows = sqlite_cursor.execute("SELECT * FROM simulations").fetchall()

# Escrever em PostgreSQL
pg_conn = psycopg2.connect("postgresql://user:pass@localhost/iotsim")
pg_cursor = pg_conn.cursor()

for row in rows:
    pg_cursor.execute(
        "INSERT INTO simulations (...) VALUES (...)",
        row
    )

pg_conn.commit()

```

ADICIONAR UTILIZADORES

- [] Tabela `users` (id, email, password_hash)
- [] Endpoint POST `/auth/register`
- [] Endpoint POST `/auth/login` (retorna JWT)
- [] Middleware para verificar JWT
- [] Filtrar simulações por `user_id`

Modelo User:

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    email = Column(String, unique=True, nullable=False)
    password_hash = Column(String, nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)

    simulations = relationship("Simulation", back_populates="user")

class Simulation(Base):
    # ... campos existentes ...
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    user = relationship("User", back_populates="simulations")
```

RATE LIMITING

- ☐ Instalar Redis
- ☐ Implementar rate limiter (5 simulações/hora/user)
- ☐ Middleware para verificar limites
- ☐ Retornar 429 Too Many Requests

7.2.16 FASE 3 - Production Ready (6 semanas)**Objetivo:** Sistema pronto para deploy real**Pré-requisitos:** Fase 2 funcional, cluster K8s real**Semana 11-12: Autenticação e Segurança****JWT COMPLETO**

- ☐ Refresh tokens
- ☐ Logout (blacklist tokens)
- ☐ Password reset flow
- ☐ Email verification (opcional)

RBAC KUBERNETES

- ☐ ServiceAccount para API
- ☐ Role com permissões mínimas
- ☐ RoleBinding
- ☐ Testar permissões

SECRETS MANAGEMENT

- ☐ Kubernetes Secrets para DB password
- ☐ Secrets para JWT secret key
- ☐ Não commitar secrets no Git

Semana 13-14: Monitoring e Observability**PROMETHEUS**

- ☐ Instalar Prometheus no cluster
- ☐ Adicionar metrics na API (prometheus_client)
- ☐ Dashboards Grafana básicos

Métricas importantes:

```
from prometheus_client import Counter, Histogram

simulations_created = Counter('simulations_created_total', 'Total sims')
simulation_duration = Histogram('simulation_duration_seconds', 'Duration')
pods_active = Gauge('pods_active', 'Active pods')
```

LOGGING

- ☐ Structured logging (JSON)
- ☐ Loki para agregação logs
- ☐ Queries úteis em Grafana

Semana 15-16: CI/CD e Deploy**GITHUB ACTIONS**

- ☐ Pipeline: test → build → push image
- ☐ Deploy automático ao push main
- ☐ Rollback automático se falhar

Exemplo `.github/workflows/deploy.yml`:

```
name: Deploy

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build Docker image
        run: docker build -t iotsim-api:${{ github.sha }} .

      - name: Push to registry
        run: docker push iotsim-api:${{ github.sha }}

      - name: Deploy to K8s
        run: |
          kubectl set image deployment/iotsim-api \
            api=iotsim-api:${{ github.sha }}
```

DEPLOY CLOUD

- ☐ Escolher provider (DigitalOcean, Linode, AWS)
- ☐ Criar cluster managed Kubernetes
- ☐ Configurar DNS
- ☐ SSL/TLS (Let's Encrypt)

7.2.17 Milestones e Entregáveis**Milestone 1: MVP Demo (Semana 4)****Entregáveis:** - Código fonte completo - README.md - API funcionando localmente - Demo gravada (backup) - Slides apresentação**Critérios de Sucesso:** - ☐ API cria simulações - ☐ Auto-stop funciona - ☐ Persistência funciona - ☐ Demo de 7 min ensaiada**Milestone 2: Kubernetes Working (Semana 10)****Entregáveis:** - Código migrado para K8s - Manifests YAML - PostgreSQL funcional - Multi-user básico

Critérios de Sucesso: - ☐ Pods criados dinamicamente - ☐ Múltiplos users isolados - ☐ Rate limiting funciona - ☐ Documentação atualizada

Milestone 3: Production Deploy (Semana 16)

Entregáveis: - Sistema deployed em cloud - CI/CD funcional - Monitoring setup - Documentação completa

Critérios de Sucesso: - ☐ API acessível publicamente - ☐ SSL/TLS configurado - ☐ Logs e métricas visíveis - ☐ Zero-downtime deploys

7.2.18 Checklists de Validação

Checklist Fase 1 (antes de apresentar)

Funcionalidade:

- ☐ POST /simulations cria container Docker
- ☐ GET /simulations lista todas
- ☐ GET /simulations/{id} mostra detalhes + logs
- ☐ DELETE /simulations/{id} para container
- ☐ Container para após duration_minutes
- ☐ Containers órfãos limpos ao reiniciar API

Base de Dados:

- ☐ Simulações persistem após restart
- ☐ Status atualiza corretamente (running-expired-stopped)
- ☐ expires_at calculado e usado
- ☐ Ficheiros config temporários limpos

Qualidade:

- ☐ Erros retornam HTTP status corretos
- ☐ Swagger UI funciona e está completo
- ☐ Código sem warnings/erros
- ☐ README.md completo

Demo:

- ☐ factory.json funciona
- ☐ Cliente HTML conecta e mostra dados
- ☐ Script de 7 min ensaiado 3x
- ☐ Vídeo backup gravado

Checklist Fase 2 (antes de migrar para Fase 3)

Kubernetes:

- ☐ Pods criados dinamicamente via K8s API
- ☐ ConfigMaps montados corretamente
- ☐ Pods param após activeDeadlineSeconds
- ☐ CronJob limpa Pods expirados

Multi-User:

- ☐ PostgreSQL substituiu SQLite
- ☐ Tabela users existe e funciona
- ☐ JWT authentication funciona
- ☐ Users só vêem suas simulações
- ☐ Rate limiting por user funciona

Robustez:

- ☐ RBAC K8s configurado
- ☐ Secrets geridos corretamente
- ☐ Error handling robusto
- ☐ Documentação atualizada

Checklist Fase 3 (antes de considerar "pronto")

Produção:

- ☐ Deploy em cloud pública
- ☐ DNS configurado
- ☐ SSL/TLS ativo
- ☐ Logs agregados (Loki/ELK)
- ☐ Métricas visíveis (Grafana)
- ☐ Alertas configurados

DevOps:

- ☐ CI/CD funcional
- ☐ Testes automatizados
- ☐ Rollback funciona
- ☐ Backups database automáticos

Segurança:

- ☐ Secrets não no Git

- [] RBAC mínimo necessário
- [] Network policies configuradas
- [] Security scan passou

7.2.19 Métricas de Sucesso

Fase 1 (MVP)

- **Funcionalidade:** 100% dos endpoints funcionam
- **Performance:** API responde < 200ms (P95)
- **Confiabilidade:** 0 crashes durante demo
- **Documentação:** README completo + comentários

Fase 2 (Kubernetes)

- **Escalabilidade:** Suporta 10+ utilizadores simultâneos
- **Isolamento:** Users não vêem simulações de outros
- **Performance:** 50+ pods criados sem degradação
- **Confiabilidade:** Cleanup 100% eficaz

Fase 3 (Produção)

- **Uptime:** 99%+ SLA
- **Deploy:** < 5 min do commit ao prod
- **Observability:** Logs e métricas 100% cobertura
- **Segurança:** 0 vulnerabilidades críticas

7.2.20 Próximos Passos Imediatos (Esta Semana)

Segunda-feira

- [] Corrigir health check database
- [] Adicionar campo expires_at
- [] Implementar cleanup_expired_simulations()
- [] Testar: criar simulação 2 min → para sozinha

Terça-feira

- [] Testar: reiniciar API → órfãos limpos
- [] Adicionar filtros GET /simulations?status=running
- [] Adicionar GET /stats

Quarta-feira

- [] Começar README.md
- [] Criar factory.json de exemplo
- [] Testar todos endpoints com Postman

Quinta-feira

- [] Cliente HTML básico
- [] Testar cliente com simulação real
- [] Começar slides apresentação

Sexta-feira

- [] Finalizar slides
- [] Ensaiar demo 1x
- [] Identificar gaps/bugs

7.2.21 Recursos Úteis

Documentação Oficial

- FastAPI: <https://fastapi.tiangolo.com>
- Docker SDK: <https://docker-py.readthedocs.io>
- SQLAlchemy: <https://docs.sqlalchemy.org>
- Kubernetes Python: <https://github.com/kubernetes-client/python>

Tutoriais

- FastAPI + Docker: <https://testdriven.io/blog/fastapi-docker/>
- Kubernetes Basics: <https://kubernetes.io/docs/tutorials/>
- MQTT.js: <https://github.com/mqttjs/MQTT.js>

Tools

- Postman: Testar API
- Swagger UI: Documentação automática
- Docker Desktop: Gerir containers
- minikube: Kubernetes local

7.2.22 Conclusão

Este roadmap cobre **12-16 semanas** de desenvolvimento estruturado em 3 fases:

1. **Fase 1 (4 semanas):** MVP funcional para apresentação universitária ← **FOCO ATUAL**
2. **Fase 2 (6 semanas):** Kubernetes + multi-user
3. **Fase 3 (6 semanas):** Produção com CI/CD, monitoring

Recomendação: Completar Fase 1 perfeitamente antes de pensar em Fase 2. Um MVP bem executado vale mais que features incompletas.

Próxima Ação: Seguir checklist "Próximos Passos Imediatos" acima.