Projeto ISI - Documentação Completa

Trabalho da Disciplina de Integração de Sistemas de Informação (ISI)

PEQSPC

None

Table of contents

1. ProjetoISIv1 — Monitorização Industrial com ETL e IoT

Para entender todos os detalhes técnicos e de configuração do projeto, lê toda a documentação disponível no site gerado pelo **MkDocs**.

1.1 Introdução

O **ProjetoISIv1** é um trabalho prático da Unidade Curricular **Integração de Sistemas de Informação (ISI)** — Licenciatura em Engenharia de Sistemas Informáticos (2025/26).
O objetivo principal é aplicar e experimentar **ferramentas de ETL (Extract, Transform, Load)** em processos de integração de dados no contexto de **monitorização industrial com sensores IoT**.

1.2 Problema

A recolha de dados das estações de produção era feita manualmente e de forma isolada, o que impedia:

- A **monitorização em tempo real**;
- A identificação de **anomalias**;
- A integração e análise de métricas de **produção, paragem, stock e defeitos**.

1.3 Objetivos

- Centralizar dados de produção, stock e paragens num repositório único;
- Automatizar a recolha de dados via **sensores MQTT**;
- Normalizar e validar dados antes do armazenamento;
- Enriquecer as leituras com **dados externos via API**;
- Gerar relatórios automáticos e enviar por email;
- Visualizar indicadores em tempo real com **React + UI Builder + Grafana**.

1.4 Estrutura da Documentação

1.5 Sobre o Projeto

- **Nome dos Autores:** PEQSPC
- **Curso e unidade curricular** LESI-IPCA ISI(Integracao de Sistemas de Informaçao)
- **Ano letivo (2025/26)**
- **Link para o repositório Git** Github

2. Arquitetura Técnica do Projeto

A arquitetura técnica do **ProjetoISIv1** suporta todo o ciclo de recolha, transformação e visualização dos dados industriais.

2.1 Componentes Principais

2.1.1 1. Publisher (Simulador MQTT em Python)

- Gera dados de produção, stock, paragens e defeitos.
- Publica mensagens MQTT em tópicos por estação.

2.1.2 2. Broker MQTT — Eclipse Mosquitto

- Atua como intermediário entre sensores (publishers) e Node-RED (subscriber).
- Porta padrão: `1883`.

2.1.3 3. Node-RED (ETL)

- Recebe dados MQTT.
- Processa e acumula valores.
- Guarda resultados em **SQLite3**.
- Envia relatórios e aciona scripts Python.

2.1.4 4. Base de Dados SQLite3

- Armazena dados acumulados e históricos.
- Tabelas: `estacoes`, `dados_acumulados`, `relatorios`, `produtos`.

2.1.5 5. Enriquecimento via API

- Obtém **preços das peças** e cruza com dados de produção.
- Utiliza **expressões regulares (Regex)** para filtrar os produtos relevantes.

2.1.6 6. Python Reporting

- Gera relatórios automáticos sobre margens e desempenho.
- Envia por email.

2.1.7 7. Dashboard (React + Material UI)

- Exibe produção, paragens e eficiência em tempo real.
- Interface moderna criada com UI Builder.

2.2 Fluxo Global de Dados

```
graph TD
  A[Sensores Python] --> B[Broker MQTT (Mosquitto)]
  B --> C[Node-RED ETL]
  C --> D[SQLite3]
```

```
    D --> E[API Preços]
    D --> F[Python Reporting]
    F --> G[Email Automático]
    D --> H[Dashboard React]
```

3. ETL PROCESSOS

```markdown
# 4. Processos ETL e Transformações

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

---

## 4.1 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.

---

## 4.2 Exemplos de Transformações

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

---

## 4.3 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.
```

5. Integração com API

Neste houve uma integraçao com uma API para integrar os preços da peças feitas nas estaçoes para adicionar ao relatorio.

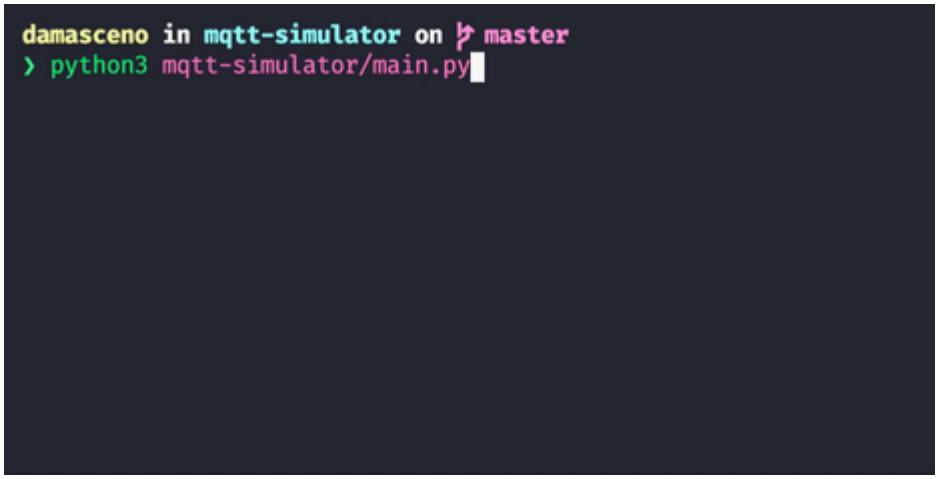
6. Base de Dados

7. Simulador de Sensores

7.1 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



7.1.1 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

7.1.2 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>
```

7.1.3 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
        }
      ]
    }
  ]
}
```

7.1.4 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 }

7.2 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.

7.2.1 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 paho.mqtt.client `protocol` param. Version of the MQTT protocol to use for this client. Can be either `3` (MQTTv31), `4` (MQTTv311) or `5` (MQTTv5)
`TLS_CA_PATH`	string	None	Sets the paho.mqtt.client.tls_set `ca_certs` param. String path to the Certificate Authority certificate file
`TLS_CERT_PATH`	string	None	Sets the paho.mqtt.client.tls_set `certfile` param. String path to the PEM encoded client certificate file
`TLS_KEY_PATH`	string	None	Sets the paho.mqtt.client.tls_set `keyfile` param. String path to the PEM encoded client private keys file
`AUTH_USERNAME`	string	None	Sets the paho.mqtt.client.username_pw_set `username` param. Username to authenticate with
`AUTH_PASSWORD`	string	None	Sets the paho.mqtt.client.username_pw_set `password` param. Password to authenticate with
`CLEAN_SESSION`	bool	True	Sets the paho.mqtt.client `clean_session` param. Boolean that determines the client type. This property is ignored if `PROTOCOL_VERSION` is `5`.
`RETAIN`	bool	False	Sets the paho.mqtt.client.publish `retain` 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 paho.mqtt.client.publish `qos` 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

7.2.2 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

7.2.3 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 *Pythonic* way. Also accept functions from Math modules	if `TYPE` is `"math_expression"`
`INTERVAL_START`	number	Minimum value that the `MATH_EXPRESSION`'s variable `x` can assume	if `TYPE` is `"math_expression"`
`INTERVAL_END`	number	Maximum value that the `MATH_EXPRESSION`'s variable `x` can assume	if `TYPE` is `"math_expression"`
`MIN_DELTA`	number	Minimum value that can be added to the `MATH_EXPRESSION`'s variable `x` 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 `x` 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 `len(values) - 1`. Only valid if `TYPE` is `"raw_values"`
`RESTART_ON_END`	bool		

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

> ***NOTE:*** Access math_expression.md file for more explanations and a example of `TYPE: "math_expression"`.

7.3 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:

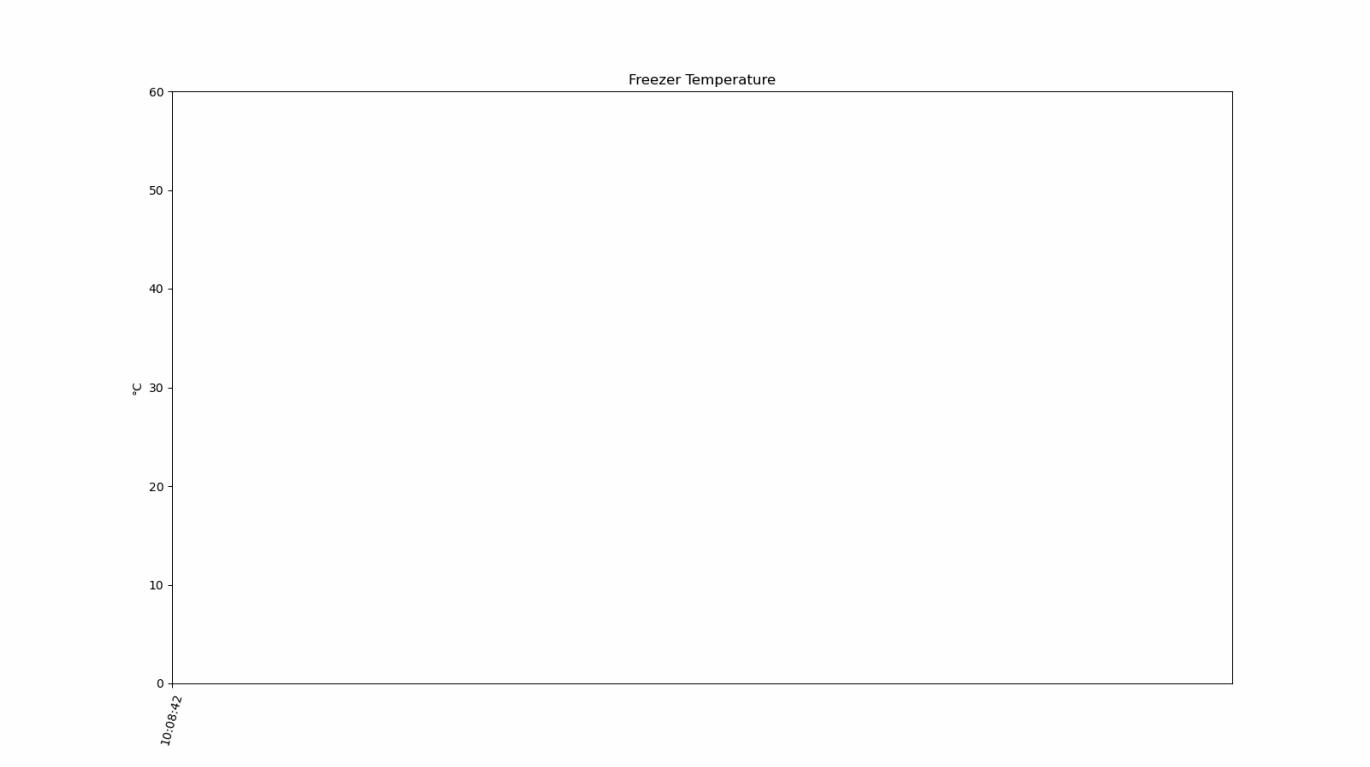
```
{
    ...
    "TYPE": "math_expression",
    "MATH_EXPRESSION": "x**2",
    "INTERVAL_START": 30,
    "INTERVAL_END": 40,
    "MIN_DELTA": 0.1,
    "MAX_DELTA": 0.2,
    ...
}
```

- `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 works 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`.

7.3.1 Example 1 - Freezer Temperature

In the example below the `MATH_EXPRESION` = $2x²+1$, `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.

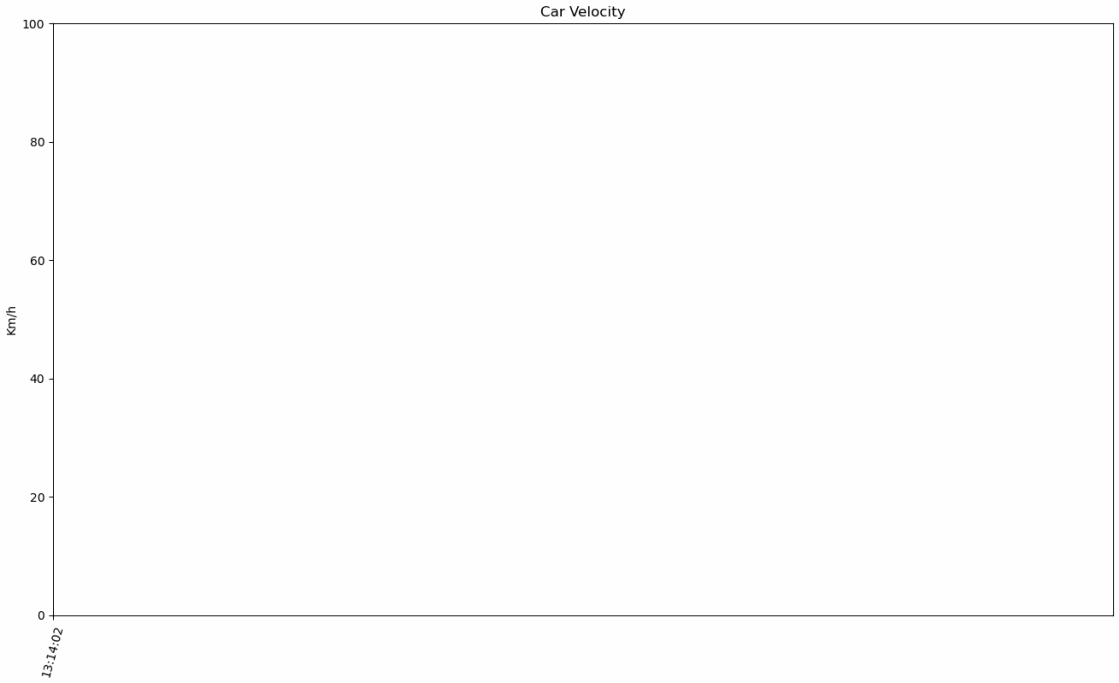
```
{
    "TYPE": "single",
    "PREFIX": "freezer",
    "TIME_INTERVAL": 6,
    "DATA": [
        {
            "NAME": "temperature",
            "TYPE": "math_expression",
            "RETAIN_PROBABILITY": 0.1,
            "MATH_EXPRESSION": "2*math.pow(x,2)+1",
            "INTERVAL_START": 0,
 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



7.3.2 Example 2 - Car Velocity

In the example below the `MATH_EXPRESION` = $\sqrt{75x}$, `INTERVAL_START = 0`, `INTERVAL_END = 100`, `MIN_DELTA = 10` and `MAX_DELTA = 10`, so it is expected that the generated values are between 0 and 87. As `REATIN_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
        }
    ]
}
```


Car Velocity
Km/h
100
80
60
40
20
0
13:14:02

8. Dashboard

9. Relatórios e Email

10. 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.

10.1 Pré-requisitos

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

10.1.1 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

10.1.2 Bibliotecas Python

Instala as bibliotecas necessárias:

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

10.1.3 Dependências Node.js

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

10.1.4 Node-RED

Instala o Node-RED globalmente:

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

Depois, inicia com:

```
node-red
```

10.1.5 Comandos Git

Geeks git Commands

10.1.6 Gerar Documentação com MkDocs

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

11. Troubleshooting

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

12. Conclusão e Trabalhos Futuros

O **ProjetoISIv1** demonstrou a aplicação prática de conceitos de **Integração de Sistemas de Informação** com recurso a ferramentas de ETL, comunicação IoT e visualização web.

12.1 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.

12.2 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).

13. 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)

13.1 📊 Visão Geral

```
Fase 1 (MVP)          Fase 2 (Kubernetes)     Fase 3 (Produção)
Semanas 1-4           Semanas 5-10            Semanas 11-16
    |                      |                       |
    ├─ API REST            ├─ K8s Client           ├─ CI/CD
    ├─ Docker Local        ├─ Pods Dinâmicos       ├─ Monitoring
    ├─ SQLite              ├─ PostgreSQL           ├─ Auth/Billing
    └─ Demo Funcional      └─ Multi-User           └─ Deploy Cloud
```

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

13.2 🎯 FASE 1 - MVP Funcional (4 semanas)

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

13.2.1 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

13.2.2 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
```

13.2.3 Semana 3: Documentação e Cliente de Teste

Dia 15-16: README.md Completo

- [ ] Secção "Instalação" passo-a-passo
- [ ] Secção "Configuração" (Docker, SQLite)
- [ ] Secção "Uso" com exemplos de curl/Postman
- [ ] Secção "Arquitetura" com diagrama
- [ ] Secção "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
```

13.2.4 Executar

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

Aceder: http://localhost:8000/docs

13.3 Uso

13.3.1 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": [...]
  }'
```

13.3.2 Listar Simulações

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

13.3.3 Ver Logs

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

13.4 Arquitetura

```
┌──────────────┐
│   Cliente    │
└──────┬───────┘
       │ POST /simulations
       ▼
┌──────────────────┐
│  FastAPI REST    │
│  :8000           │
└──┬───────────────┘
   │
   ├──► Docker Engine (cria container)
   │
   └──► SQLite (guarda metadata)
```

13.5 Exemplos de Config

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

13.6 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
        }
      ]
    }
  ]
}
```

13.6.1 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

13.7 Arquitetura

```
┌──────────┐
│ Cliente  │
└────┬─────┘
     │ POST /simulations
     ▼
┌──────────────┐
│ FastAPI      │──► SQLite (metadata)
│ REST API     │
└──────┬───────┘
       │
       └──► Docker Engine
            │
            └──► mqtt-simulator container
                 (publica MQTT)
```

13.8 Stack Tecnológica

Backend: - Python 3.11 - FastAPI (API REST) - SQLAlchemy + SQLite - Docker SDK
```

Simulador: - mqtt-simulator (imagem existente) - MQTT protocol - Random walk data generation

Tools: - Swagger UI (documentação automática) - Postman (testes)

13.9 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

13.10 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

13.11 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)

13.12 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

---

13.13 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
```
```

13.14 🎓 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

---

13.15 🚀 FASE 2 - Kubernetes Multi-User (6 semanas)

Objetivo: Migrar para Kubernetes, suportar múltiplos utilizadores
Pré-requisitos: Fase 1 100% funcional, conhecimento básico K8s

13.15.1 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())
```

13.15.2 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

13.15.3 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

13.16 🏭 FASE 3 - Production Ready (6 semanas)

Objetivo: Sistema pronto para deploy real
Pré-requisitos: Fase 2 funcional, cluster K8s real

13.16.1 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

13.16.2 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

13.16.3 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)

13.17 Milestones e Entregáveis

13.17.1 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

---

13.17.2 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

---

13.17.3 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

---

13.18 Checklists de Validação

13.18.1 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
```

13.18.2 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
```

13.18.3 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
```

13.19 Métricas de Sucesso

13.19.1 Fase 1 (MVP)

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

13.19.2 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

13.19.3 Fase 3 (Produção)

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

13.20 Próximos Passos Imediatos (Esta Semana)

13.20.1 Segunda-feira

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

13.20.2 Terça-feira

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

13.20.3 Quarta-feira

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

13.20.4 Quinta-feira

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

13.20.5 Sexta-feira

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

13.21 Recursos Úteis

13.21.1 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

13.21.2 Tutoriais

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

13.21.3 Tools

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

13.22 🎬 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.