Definição do projeto e arquitetura do sistema

Equipe:	**1**
Nome:	**Amanda Segura Mendes de Oliveira**
Nome:	**Ana Clara Godoy Ensides**
Nome:	**Beatriz Martuscelli da Silva Prado**
Nome:	**Felipe Pellegrini Kumagae**
Nome:	**Guilherme Yuiti de Queiroz Barbosa**
Nome:	**Hebert de Oliveira Brito**
Nome:	**Maysa Gabriela Lucas Izaias**
Nome:	**Rafael Utsunomya Machado**

1. Arquitetura Geral do Sistema

A arquitetura do projeto é um sistema ciberfísico que integra um loop de controle de hardware com o objetivo de controlar o nível do tanque 3 (intermediário) a partir da vazão fornecida pela bomba do tanque 1, considerando a interligação hidráulica entre ambos. Além disso, esse controle é feito com um gêmeo digital para supervisão e registro de dados.

a. Loop de Controle Físico

1. **Planta (Hardware):** O Kit Amira (DTS200) é a planta física, com tanques, bombas e sensores. Os sensores e atuadores operam com sinais de -10V a 10V.
2. **Condicionamento de Sinais:** Um módulo de condicionamento converte os sinais do Kit Amira (leitura/escrita de -10V a 10V) para níveis compatíveis com o microcontrolador (0V a 3V).
3. **Controlador (IoT):** O ESP32 atua como o cérebro do sistema. Ele é responsável por:
    - Ler os sinais de nível (0V-3V).
    - Implementar o controlador (ex: PID) em tempo real para ajustar a bomba P1.
    - Enviar o sinal de controle (0V-3V) de volta ao condicionador.

b. Arquitetura do Gêmeo Digital

1. **Coleta de Dados (ESP32):** O mesmo ESP32 lê os níveis reais dos tanques, formata os dados em JSON e os envia periodicamente (via HTTP POST ou MQTT) pela rede Wi-Fi.
2. **Backend (API):** Um servidor (rodando PHP, Python ou Node.js) recebe os dados do ESP32. A API é responsável por validar e gravar os dados no banco.
3. **Banco de Dados:** Um banco de dados MySQL (XAMPP) armazena os dados recebidos. As tabelas devem incluir medições, tanques e eventos para manter um registro histórico.

4. **Frontend (Dashboard Web):** Uma interface de supervisão web (HTML/CSS/JS) consome a API do backend para exibir os níveis dos tanques, indicadores e histórico em tempo real.

2. Estrutura de Pastas do Repositório

A estrutura de pastas para o projeto é a que segue abaixo. O repositório já foi colocado no GitHub.

```
None
Projeto-Integrador-Tanques/
│
├── firmware/                    # Código do ESP32 e simuladores
simples
│   ├── main.ino                  # (esqueleto do firmware real)
│   └── simulador.py              # script que envia dados
simulados p/ API
│
├── backend/                     # API e integração MySQL
│   ├── app.py                    # pode começar em Flask,
FastAPI ou PHP
│   ├── requirements.txt          # dependências mínimas
│   └── README.md
│
├── database/                    # Modelagem e scripts SQL
│   ├── schema.sql                # CREATE TABLES básicas
│   ├── seed.sql                  # dados simulados
│   └── diagramas/
│       └── er.mmd                # modelo
entidade-relacionamento
│
├── frontend/                    # Dashboard web
│   ├── index.html                # protótipo inicial
│   ├── script.js                 # leitura e visualização básica
de dados
│   └── style.css
│
├── shared_data/                 # Dados simulados comuns (ESP ↔
API ↔ Front)
│   └── dados_simulados.json
│
├── docs/                        # Documentação do projeto
```

```
|   ├── README.md                 # resumo do escopo, equipe e
divisão de tarefas
|   └── arquitetura_geral.mmd     # diagrama simples de blocos
|
└── test/                        # Scripts e logs de testes
    ├── test_api.http             # requisições REST para testar
a API
    └── logs/.gitkeep
```

3. Especificação de API e Formato de Dados

a. Formato de Dados (ESP32 para Backend)

Os dados devem ser estruturados em formato JSON. O formato deve ser um array de objetos, onde cada objeto representa uma medição:

```JSON
[
  {"tanque": 1, "nivel": 78.5, "timestamp": "2025-10-22
14:00:00"},
  {"tanque": 2, "nivel": 43.2, "timestamp": "2025-10-22
14:00:00"},
  ...
]
```

b. Endpoints da API (Backend para Frontend)

O backend deve prover endpoints de consulta para o frontend, incluindo:

- **POST /api/medicoes**: Endpoint de recebimento para o ESP32 enviar os dados JSON (via HTTP POST).
- **GET /api/medicoes/atual:** Consulta o último valor de nível registrado para os tanques.
- **GET /api/medicoes/historico**: Consulta o histórico de medições, permitindo filtros (ex: por tanque ou período).
- **GET /api/alertas:** Consulta os eventos ou alertas (ex: nível muito alto/baixo).

4. Diagrama de Fluxo de Dados

O fluxo de dados no sistema completo pode ser descrito em duas partes:

a. Fluxo de Controle (Tempo Real)

1. **Leitura:** Nível do Tanque 1 (h1) e Tanque 2 (h2) é lido pelos sensores do Kit Amira.
2. **Conversão 1:** Sinais (-10V a 10V) são convertidos para (0V a 3V) pelo Condicionador.
3. **Processamento:** ESP32 lê os sinais (0V-3V) e aplica a lógica do controlador PID para calcular o sinal de controle necessário para a Bomba P1.
4. **Conversão 2:** O sinal de controle (0V-3V) do ESP32 é convertido de volta para (-10V a 10V).
5. **Atuação:** O sinal de controle ajusta a Bomba P1 , que altera a vazão de entrada q1 no Tanque 1.

b. Fluxo de Dados (Digital Twin)

1. **Coleta:** O ESP32 captura os valores de nível (h1, h2) lidos no "Fluxo de Controle".
2. **Formatação:** Os dados são estruturados em JSON.
3. **Envio:** O ESP32 envia o JSON via Wi-Fi (HTTP POST) para o endpoint POST /api/medicoes do Backend.
4. **Armazenamento:** A API do Backend recebe , valida e insere os dados nas tabelas medicoes do banco MySQL.
5. **Visualização (Pull):**
    - O Frontend (Dashboard) solicita dados ao Backend periodicamente (ex: GET /api/medicoes/atual e GET /api/medicoes/historico).
    - O Backend consulta o MySQL e retorna os dados ao Frontend.
    - O Frontend atualiza os gráficos e indicadores para visualização em tempo real.