

T4 - ENG SOFT - PEDRO LUCAS

Atividade T4 — Interfaces (Front-end) sobre XML + API T3

1) Arquitetura das Interfaces

A interface foi construída como uma "Single Page Application" (SPA) contida em `index.html`, `styles.css` e `main.js`. A navegação interna (controlada por JavaScript) alterna entre duas views principais:

1. View 1: Dashboard (Página Inicial)

- **Objetivo:** Fornecer uma visão geral e em tempo real do estado da estufa.
- **Consumo API:** `GET /api/consulta` (busca todos os dados do T3).
- **Componentes:**
 - **KPI Cards:** "Temperatura Média", "Umidade do Solo", "Luminosidade" e "Total de Leituras".
 - **Gráfico de Linha:** Um gráfico (Chart.js) mostrando o histórico de leituras de temperatura.
 - **Tabela "Últimas Leituras":** Uma lista das 10 leituras mais recentes de qualquer sensor, ordenadas por data.

2. View 2: Enviar Dados (Simulador)

- **Objetivo:** Simular um dispositivo IoT enviando um novo pacote de dados XML para o back-end (T3).
- **Consumo API:** `POST /api/xml`.
- **Componentes:**
 - Uma área de texto (`<textarea>`) para colar o `cliente.xml`.
 - Um botão "Importar Exemplo (T2)" para preencher a área de texto.
 - Um botão "Validar no Cliente" (executa as funções da Seção 4).
 - Uma área de status para mostrar erros de validação (do cliente ou do servidor T3).

- Um botão "Enviar ao T3" (executa o `fetch POST`).

2) Estrutura de Pastas

A estrutura de pastas desenvolvida para o front-end foi:

```
frontend/  
├── index.html    # Estrutura principal da SPA (HTML)  
├── styles.css    # Folha de estilos (Aparência, layout, correção do gráfico)  
└── main.js      # Lógica da aplicação (Navegação, Validação T4, API Fetch T3)
```

3) Importação do XML de Exemplo (T2)

Para que o JavaScript possa ler e validar o XML, ele primeiro é convertido de uma string de texto (vinda do `<textarea>`) para um objeto DOM (Document Object Model) usando o `DOMParser`.

```
// O Parser converte a string em um documento XML  
const parser = new DOMParser();  
const xmlDoc = parser.parseFromString(xmlText, "application/xml");  
  
// Verifica se o XML é bem formado (primeira validação)  
const parserError = doc.getElementsByTagName("parsererror");  
if (parserError.length > 0) {  
    // Erro de Sintaxe  
} else {  
    // XML bem formado, pronto para a validação de regras  
}
```

4) Validação no Cliente (contrato T2)

Antes de enviar o XML ao T3, o `main.js` executa uma validação no cliente (T4) que espelha as regras do T2 e T3. Isto dá feedback instantâneo ao usuário e evita chamadas de API desnecessárias.

- **Validação 1: Sintaxe XML** (Ver Seção 3, usando `DOMParser`).

- **Validação 2: Regras de Negócio (Faixas):** Verifica se os valores (`<valor>`) estão dentro das faixas permitidas (ex: temperatura [-10, 60]).
- **Validação 3: Integridade Referencial (ID/IDREF):** Verifica se todo `leitura/@sensorRef` aponta para um `sensor/@id` que existe no documento.

```
/**
 * (Função do main.js)
 * Valida o XML contra as regras do T2.
 * @param {XMLDocument} xmlDoc - O documento XML parseado.
 * @returns {string[]} - Uma lista de strings de erro.
 */
function validarClienteXML(xmlDoc) {
  const erros = [];
  const limites = {
    "temperatura": [-10.0, 60.0],
    "umidadeAr": [0.0, 100.0],
    "umidadeSolo": [0.0, 100.0],
    "luminosidade": [0.0, 200000.0]
  };

  // Validação 3: ID/IDREF
  const sensorIds = new Set();
  xmlDoc.querySelectorAll("sensores sensor").forEach(s => sensorIds.add(
    s.getAttribute("id")));
  if (sensorIds.size === 0) erros.push("Nenhum sensor (<sensor>) definido.");

  xmlDoc.querySelectorAll("leituras leitura").forEach((leitura, i) => {
    const sensorRef = leitura.getAttribute("sensorRef");
    const valorNode = leitura.querySelector("valor");

    if (!sensorRef || !sensorIds.has(sensorRef)) {
      erros.push(`Leitura[${i}]: 'sensorRef' ("${sensorRef || 'nulo'}") é inválido.`);
    }

    if (!valorNode) {
      erros.push(`Leitura[${i}]: Tag <valor> ausente.`);
    }
  });
}
```

```

        return;
    }

    // Validação 2: Faixas
    const valor = parseFloat(valorNode.textContent);
    const sensorTipo = xmlDoc.querySelector(`sensor[id="${sensorRef}"]`)??.getAttribute("tipo");

    if (sensorTipo && limites[sensorTipo]) {
        const [min, max] = limites[sensorTipo];
        if (isNaN(valor) || valor < min || valor > max) {
            erros.push(`Leitura[${i}]: Valor ${valor} para '${sensorRef}' (${sensorTipo}) fora da faixa [${min}, ${max}].`);
        }
    }
});
return erros;
}

```

5) Exportação do **cliente.xml** (exemplo)

Este é o *payload* de texto (string) que o front-end envia no corpo da requisição `POST /api/xml`. Este exemplo é carregado no `<textarea>` através do botão "Importar Exemplo (T2)".

```

<?xml version="1.0" encoding="UTF-8"?>
<estufa id="estufaPrincipal"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"(http://www.w3.org/2001/XMLSchema-instance)"
  xsi:noNamespaceSchemaLocation="estufa.xsd">
  <sensores>
    <sensor id="tNorte" tipo="temperatura" unidade="C">
      <localizacao>Parede Norte, 1.5m altura</localizacao>
    </sensor>
    <sensor id="uSoloB1" tipo="umidadeSolo" unidade="%">
      <localizacao>Bancada 1, Vaso 3</localizacao>
    </sensor>
  </sensores>
</estufa>

```

```

    <sensor id="luzTeto" tipo="luminosidade" unidade="lux">
      <localizacao>Acima da Bancada 1</localizacao>
    </sensor>
  </sensores>
  <leituras>
    <leitura sensorRef="tNorte">
      <dataHora>2025-10-30T10:30:00-03:00</dataHora>
      <valor>24.5</valor>
    </leitura>
    <leitura sensorRef="uSoloB1">
      <dataHora>2025-10-30T10:30:00-03:00</dataHora>
      <valor>55.2</valor>
    </leitura>
    <leitura sensorRef="luzTeto">
      <dataHora>2025-10-30T10:30:00-03:00</dataHora>
      <valor>115000</valor>
    </leitura>
  </leituras>
</estufa>

```

6) Integração com a API T3 (fetch)

O `main.js` usa a API `fetch` para se comunicar com o back-end (T3), que está a rodar em `http://localhost:5000`.

Chamada `POST /api/xml` (Envio de dados):

```

// (Função do main.js)
async function enviarXML(xmlString) {
  try {
    const response = await fetch(`${API_URL}/api/xml`, {
      method: "POST",
      headers: {
        "Content-Type": "application/xml"
      },
      body: xmlString // Envia a string XML validada
    });
  }
}

```

```

const result = await response.json();

if (!response.ok) {
  // Erro 400 (XSD ou Regra) ou 500
  setStatus(`Erro do Servidor (T3):\n${result.code}\n${result.message || result.details}`, 'error');
} else {
  // Sucesso 201
  setStatus(`Sucesso (T3)!\nXML salvo com ID:\n${result.id}`, 'success');
}
} catch (error) {
  // Erro de CORS ou T3 desligado
  setStatus('Erro de Rede: Não foi possível conectar ao T3.', 'error');
}
}

```

Chamada `GET /api/consulta` (Carregar o Dashboard):

```

// (Função do main.js)
async function carregarDashboard() {
  try {
    // Chama o GET /api/consulta do T3
    const response = await fetch(`${API_URL}/api/consulta`);
    if (!response.ok) {
      throw new Error(`Falha no T3: ${response.statusText}`);
    }
    const data = await response.json();

    // ... (chama funções de renderização) ...
    renderizarKPIs(data.leituras);
    renderizarTabela(data.leituras);
    renderizarGrafico(data.leituras);

  } catch (error) {
    // Erro de CORS ou T3 desligado
  }
}

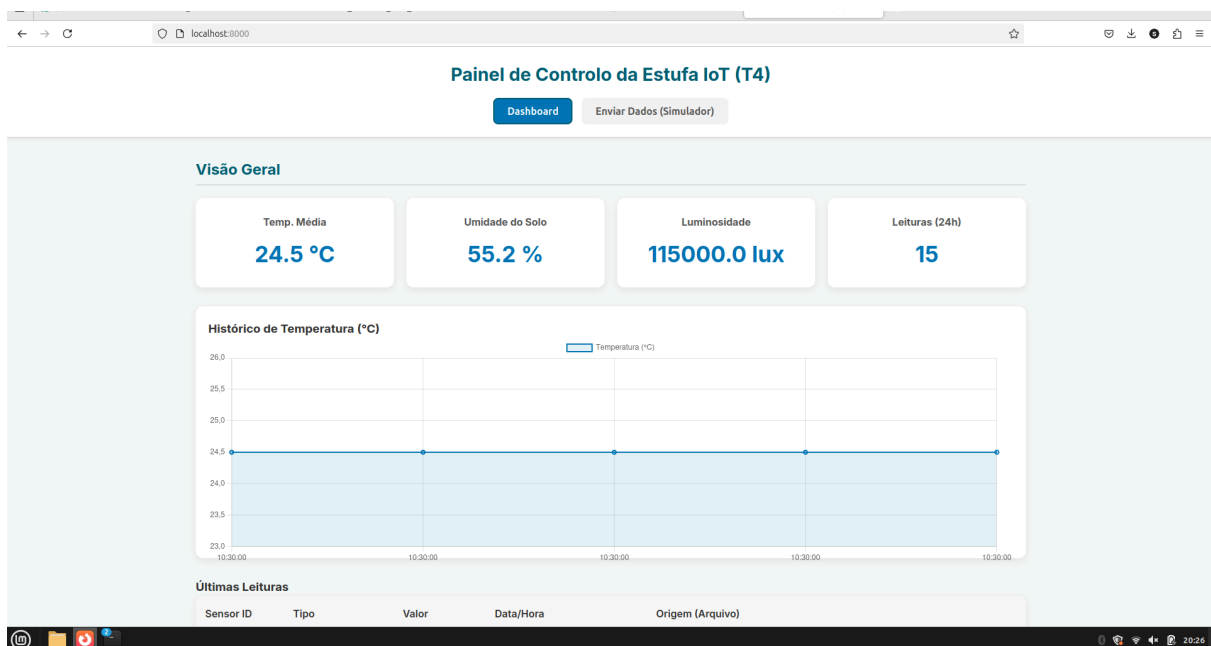
```

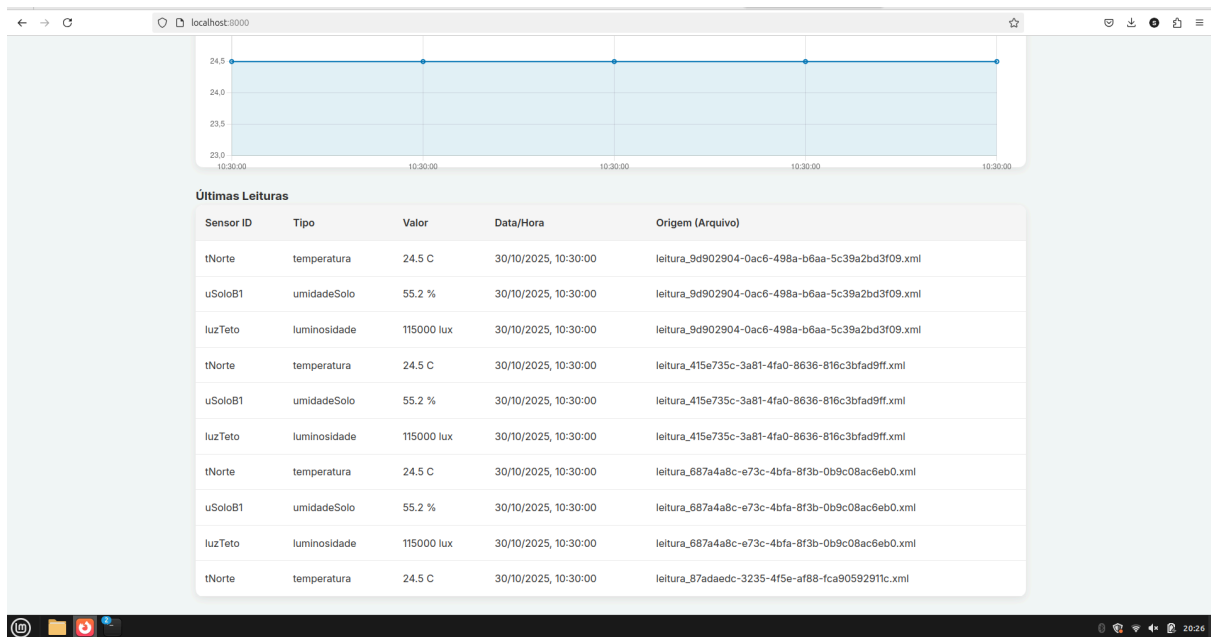
```
}
```

7) Indicadores (gráfico/tabela)

O Dashboard (View 1) é a principal tela de consumo de dados, focada em legibilidade e alertas visuais. Os dados são carregados do T3 (`GET /api/consulta`) e renderizados nos seguintes componentes:

1. **Indicadores KPI (Cards):** Médias calculadas (Temp, Solo, Luz) e contagem total de leituras.
2. **Gráfico de Linha (Histórico):** Gráfico (Chart.js) mostrando a variação da temperatura ao longo do tempo.
3. **Tabela (Log de Eventos):** Tabela com as 10 leituras mais recentes, mostrando a origem (ID do arquivo salvo no T3).





8) Manual de Uso (Tutorial)

Este é o fluxo de ponta a ponta para testar o sistema (T4 → T3 → T4):

- Iniciar Back-end (T3):** No terminal 1, navegar para `meu-projeto-t3`, ativar o `venv` e executar `python backend/app/api.py`. O servidor T3 estará a rodar em `http://localhost:5000`.
- Iniciar Front-end (T4):** No terminal 2, navegar para `meu-projeto-t3/frontend` e executar `python -m http.server 8000`. O servidor T4 estará a rodar em `http://localhost:8000`.
- Aceder ao Sistema:** Abrir o navegador e aceder a `http://localhost:8000`.
- Ver Dashboard Vazio:** O Dashboard (View 1) será carregado. Se a pasta `backend/data` do T3 estiver vazia, ele mostrará "Nenhum dado encontrado".
- Simular Dispositivo:** Navegar para a view "Enviar Dados (Simulador)".
- Importar Exemplo:** Clicar em "Importar Exemplo (T2)". A área de texto será preenchida com o `cliente.xml`.
- Validar (Cliente):** Clicar em "1. Validar no Cliente". O painel de status exibirá "XML VÁLIDO (Cliente)!...". O botão "Enviar ao T3" será ativado.
- Enviar ao T3:** Clicar em "2. Enviar ao T3". O front-end executa o `fetch POST`. O T3 valida (XSD e Regras), salva o ficheiro em `backend/data/` e retorna `HTTP 201`. O painel de status exibirá "Sucesso (T3)!...".
- Ver Dashboard com Dados:** Navegar de volta para a view "Dashboard".

10. **Resultado:** A página fará uma nova chamada `GET /api/consulta`. Desta vez, o T3 encontrará o XML salvo, e o front-end exibirá os dados nos KPIs, no gráfico de temperatura e na tabela de últimas leituras.

9) Checklist

- ✓ ~~Navegação e views descritas~~
- ✓ ~~Validações no cliente documentadas~~
- ✓ ~~Exemplo de cliente.xml colado~~
- ✓ ~~Exemplos de fetch para a API T3~~
- ✓ ~~Manual de uso completo~~
- ✓ ~~Estrutura de pastas definida~~
- ✓ ~~Indicadores/Layout descritos (e prints anexados na Seção 7)~~