Relatório de Mini Projeto Módulo II

Conectividade e Redes

Sistema de Monitoramento de Consumo de Água com IoT

Autores:

- Gabriel Pires(gpac@cesar.school)
- Luis Mingati (locm@cesar.school)
- João Ventura (jvpv@cesar.school)

Professor: Jymmy Barreto (CESAR School)

Local: Recife
Ano: 2025

Sumário

---

1. Introdução

1.1 Contexto

O trabalho realizado busca criar uma maneira de monitorar o consumo de água de um estabelecimento qualquer através de sensores cuja informação é coletada por uma placa ESP8266 que resulta em dashboards intuitivos e alertas de e-mail.

1.2 Motivação

A motivação do projeto é proveniente do entendimento de que as ações baseadas em dados têm a maior probabilidade de acerto e, portanto, quis-se aplicar essa filosofia inerente ao IoT a um problema atual sério como é o gasto de água.

1.3 Objetivos

O trabalho tem como objetivo criar uma plataforma simples, mas que traga valor a um usuário final, deixando-o ver informações relevantes para o seu consumo de água. Essa telemetria tem como propósito final desperdiçar menos água ao permitir que o usuário final tenha uma maior consciência do seu consumo, trazendo uma economia maior ao usuário e um impacto positivo ao meio ambiente.

---

2. Metodologia

2.1 Diagrama do Sistema

O sistema de monitoramento de consumo de água foi desenvolvido seguindo uma arquitetura distribuída em camadas, conforme apresentado no diagrama da Figura 1. A solução integra componentes de hardware para coleta de dados, middleware para processamento em tempo real, backend para persistência e frontend para visualização.

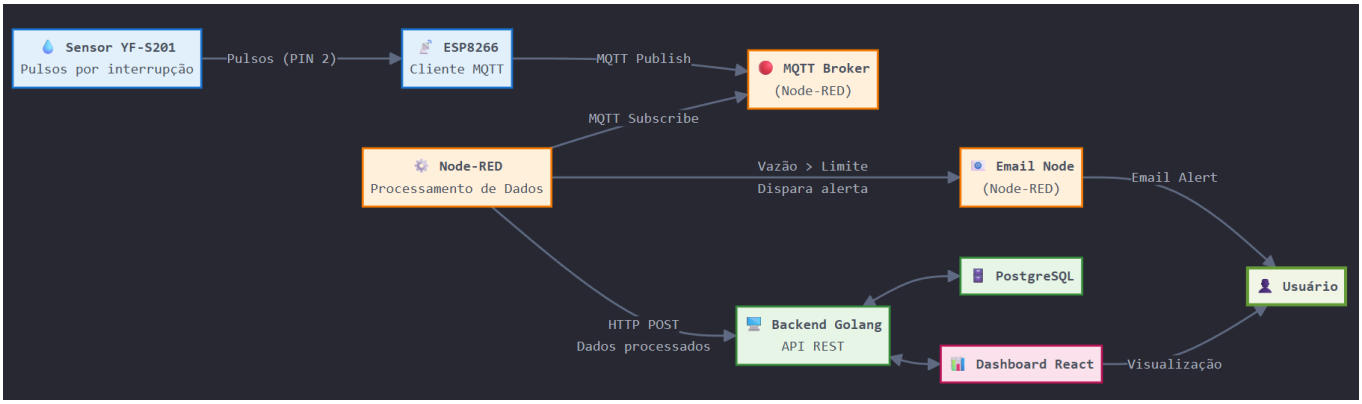


Figura 1: Arquitetura geral do sistema de monitoramento de consumo de água

A arquitetura é composta por cinco camadas principais:

- **Sensoriamento:** ESP8266 + Sensor YF-S201
- **Comunicação:** Protocolo MQTT via Wi-Fi
- **Processamento:** Node-RED
- **Armazenamento:** Backend Golang + PostgreSQL
- **Apresentação:** Dashboard React

Esta estrutura garante escalabilidade, manutenibilidade e separação clara de responsabilidades.

2.2 Lista de Hardware e Software

2.2.1 Hardware Utilizado

- **ESP8266:** Microcontrolador com conectividade Wi-Fi integrada, responsável pela coleta e transmissão dos dados do sensor
- **Sensor YF-S201:** Sensor de vazão por interrupção baseado em efeito Hall, capaz de medir fluxo de água através de pulsos gerados pela rotação de uma turbina interna
- **Infraestrutura de Rede:** Roteador Wi-Fi para conectividade entre os dispositivos

2.2.2 Software e Tecnologias

- **Visual Studio Code + PlatformIO:** Ambiente de desenvolvimento integrado para programação do ESP8266, oferecendo recursos avançados de debugging e gerenciamento de dependências
- **Node-RED:** Plataforma de desenvolvimento baseada em fluxo para integração de dispositivos IoT
- **MQTT Broker:** Servidor de mensageria para comunicação publisher/subscriber
- **Golang:** Linguagem de programação utilizada no desenvolvimento do backend e API REST
- **PostgreSQL:** Sistema de gerenciamento de banco de dados relacional para persistência dos dados
- **React:** Biblioteca JavaScript para desenvolvimento do frontend e dashboards interativos
- **Resend:** Serviço de envio de emails para sistema de alertas
- **Protocolo HTTP/HTTPS:** Para comunicação entre Node-RED e backend

2.3 Fluxo de Comunicação (MQTT + Wi-Fi)

2.3.1 Coleta e Transmissão de Dados

O processo inicia-se com o sensor YF-S201 conectado ao pino 2 da ESP8266, gerando pulsos por interrupção conforme o fluxo de água passa pela turbina interna. A ESP8266, programada como cliente MQTT, processa estes pulsos e calcula o volume instantâneo de água consumida.

A cada 250 milissegundos, a ESP8266 publica os dados de volume no tópico MQTT configurado, utilizando a conexão Wi-Fi local. Esta frequência de amostragem garante precisão adequada para detecção de consumo normal e anomalias como vazamentos.

2.3.2 Processamento em Node-RED

O Node-RED atua como subscriber MQTT, recebendo continuamente os dados de volume publicados pela ESP8266. O fluxo implementado inclui:

- **Rate Limit Control:** Controle de taxa para evitar sobrecarga do sistema
- **Anomaly Router:** Algoritmo de detecção que compara o volume atual com limites pré-estabelecidos
- **Processamento Dual:** Quando detectada anomalia (vazão acima do limite), o sistema bifurca o fluxo para duas ações paralelas

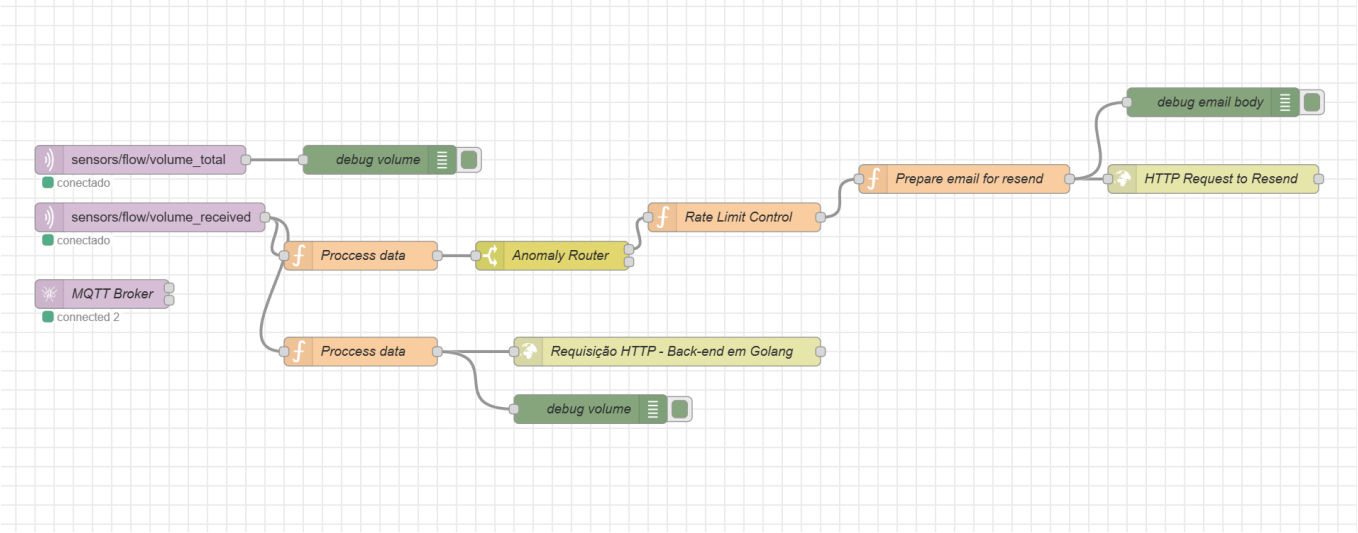


Figura 2: Fluxo de processamento implementado no Node-RED

2.3.3 Sistema de Alertas e Persistência

Quando identificado um possível vazamento, o Node-RED dispara automaticamente:

- **Alerta por Email:** Preparação e envio de email via serviço Resend para notificação imediata ao usuário
- **Requisição HTTP:** POST para o endpoint `/volume` do backend Golang, transmitindo os dados processados

2.3.4 Backend e Armazenamento

O backend desenvolvido em Golang recebe as requisições HTTP POST contendo os dados de volume, processa as informações e persiste no banco PostgreSQL. A API REST disponibiliza endpoints para consulta histórica e estatísticas de consumo.

2.3.5 Visualização Frontend

O dashboard React consome a API REST do backend, apresentando ao usuário visualizações interativas do consumo de água, incluindo gráficos temporais, estatísticas de uso e histórico de alertas de vazamento.

Esta arquitetura assegura comunicação confiável, processamento em tempo real e interface intuitiva para monitoramento eficiente do consumo hídrico.

---

3. Resultados

3.1 Arquitetura do Dashboard

O dashboard é dividido em duas camadas principais: o backend em Go e o frontend em React (com ShadcnUI e Recharts). A ideia central é captar leituras de vazão instantânea vindas do Node-RED, armazená-las em um banco PostgreSQL e, a partir daí, apresentar diferentes visões de consumo e custo ao usuário.

3.2 Funcionamento

- Existe uma tabela no PostgreSQL que recebe leituras periódicas de vazão (cada registro inclui timestamp e valor associado)
- Para testes e demonstração, foram inseridos 72.000 registros falsos que simulam leituras até um ano no passado, garantindo que os gráficos históricos mostrem comportamento realista para dados de até 1 ano atrás
- Todas as consultas ao banco são escritas em arquivos SQL e geradas em código Go pela ferramenta SQLC, o que garante tipagem forte e queries otimizadas

3.3 Definição de Períodos de Análise

O sistema aceita diferentes janelas de tempo: segundo, minuto, hora, dia, semana, mês e ano. Cada janela possui um número fixo de "pontos" para exibir nos gráficos:

- **Últimos 60 segundos** (um ponto por segundo)
- **Últimos 60 minutos** (um ponto por minuto)
- **Últimas 24 horas** (um ponto por hora)
- **Últimos 30 dias** (um ponto por dia)
- **Últimas 21 semanas** (um ponto por semana)
- **Últimos 12 meses** (um ponto por mês)
- **Últimos 10 anos** (um ponto por ano)

Essa definição prévia de quantos pontos usar facilita o agrupamento no banco e evita complexidade adicional no frontend.

3.4 Atualização em Tempo Real

Apesar de considerar SSE (Server-Sent Events) ou WebSockets para o "push" de dados, optou-se por um mecanismo de polling simples: o frontend faz uma requisição a cada segundo para obter os dados em "tempo real". Dessa forma, evita-se a complexidade de manter conexões abertas e toda a lógica de gerenciamento de sockets.

3.5 Camada Frontend em React

A interface foi construída com componentes do ShadcnUI para botões, seletores de período e seletores de data, tornando a experiência consistente e responsiva. Os gráficos são renderizados com Recharts, aproveitando componentes como LineChart e BarChart para exibir séries temporais e valores acumulados.

Quando o usuário escolhe um período (por exemplo, "última semana" ou um range de datas), o React faz requisições aos endpoints de total e histórico para obter:

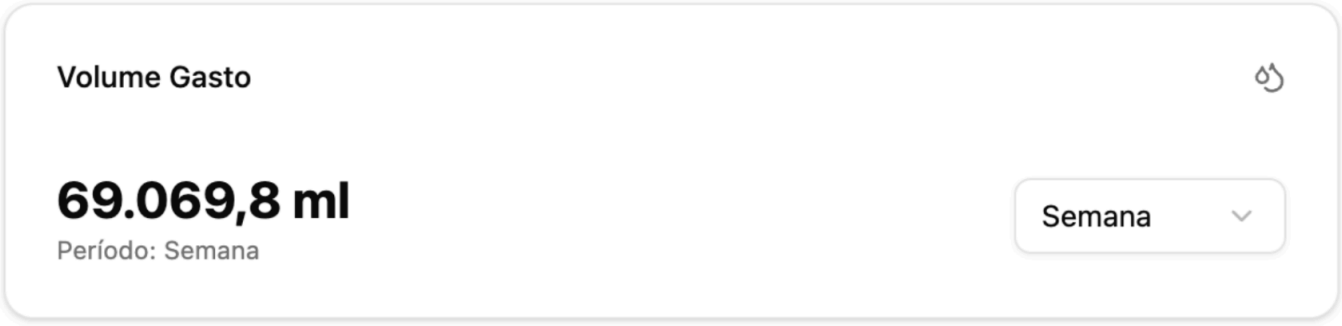
- Volume total no período selecionado
- Valor total no período selecionado
- Dados de série histórica de volume (lista de pontos)
- Dados de série histórica de valor (lista de pontos)

Simultaneamente, um timer de um segundo dispara pedidos ao endpoint de tempo real, utilizando um gráfico que exibe os últimos 60 segundos de leitura.

3.6 Cinco Visões Disponíveis

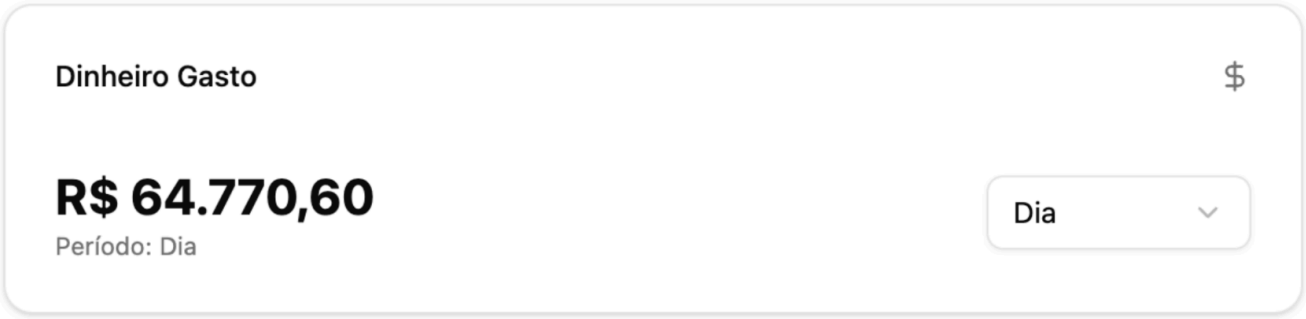
3.6.1 Volume Total no Período Escolhido

Um card grande que mostra, em ml, quanto foi consumido no intervalo especificado (minuto, hora, dia, semana, mês ou ano).



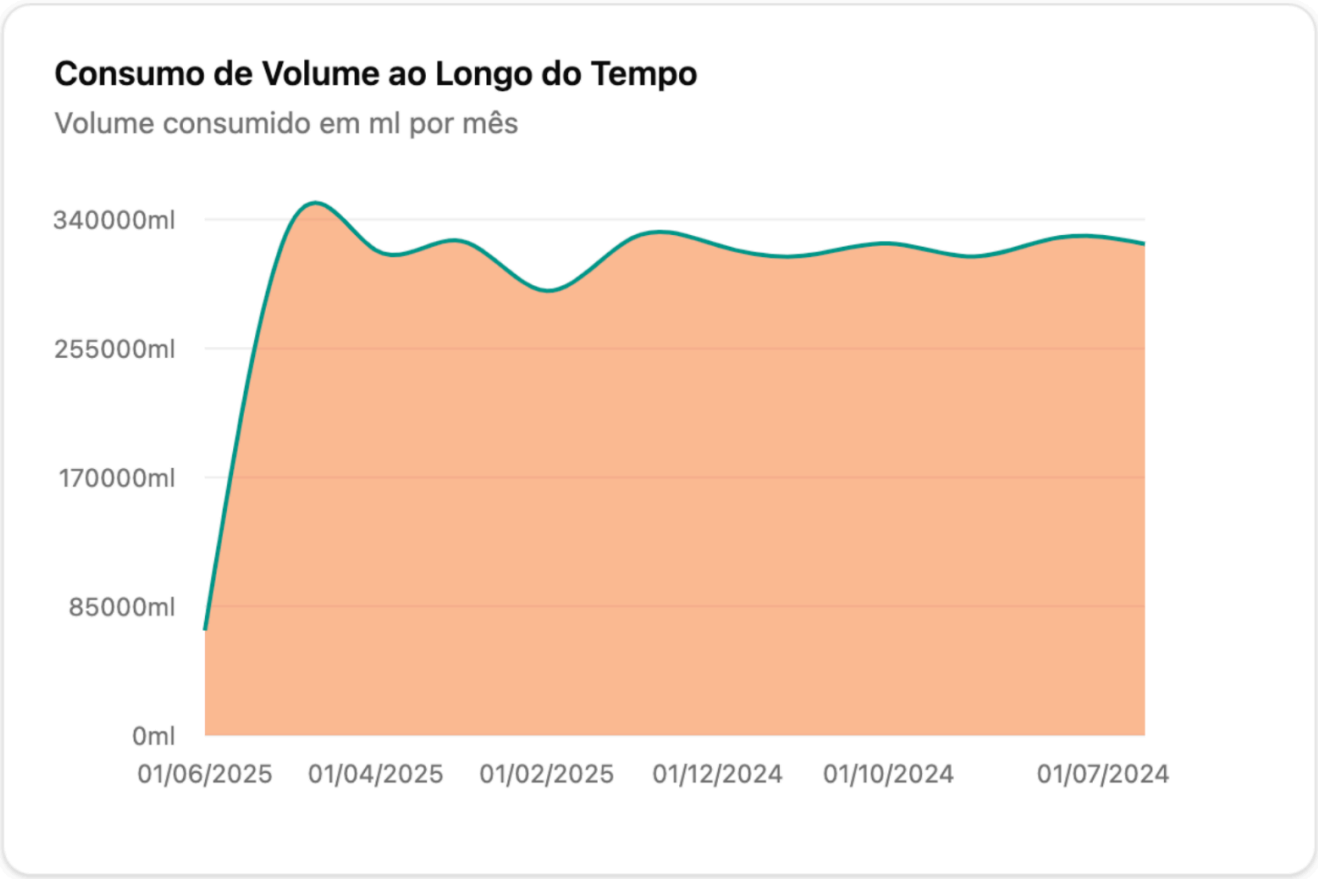
3.6.2 Valor Total no Período Escolhido

Outro card que exibe a soma monetária correspondente, permitindo avaliar custos de forma imediata.



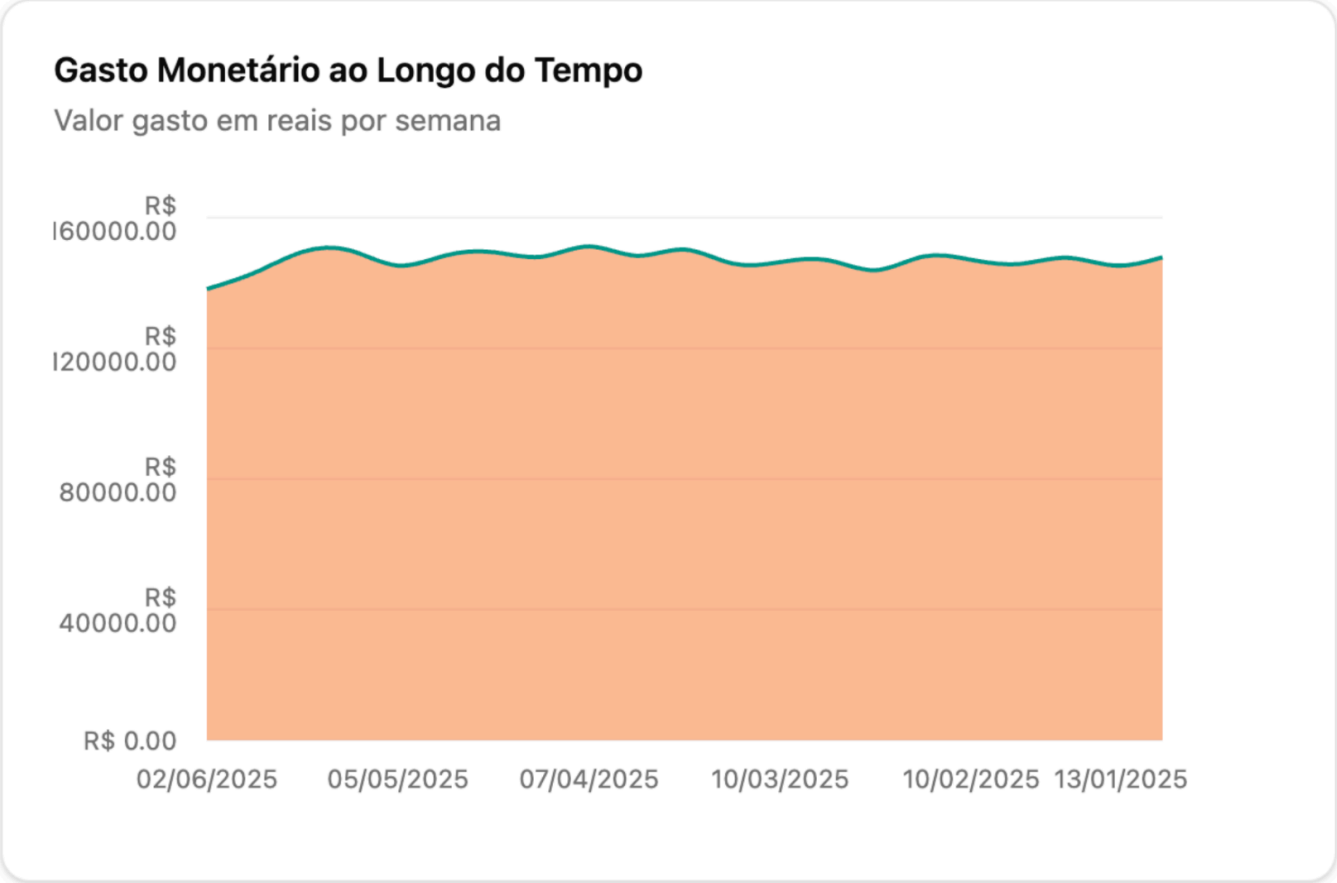
3.6.3 Gráfico de Volume ao Longo do Tempo

Uma série temporal que mostra como o consumo variou dentro da janela.



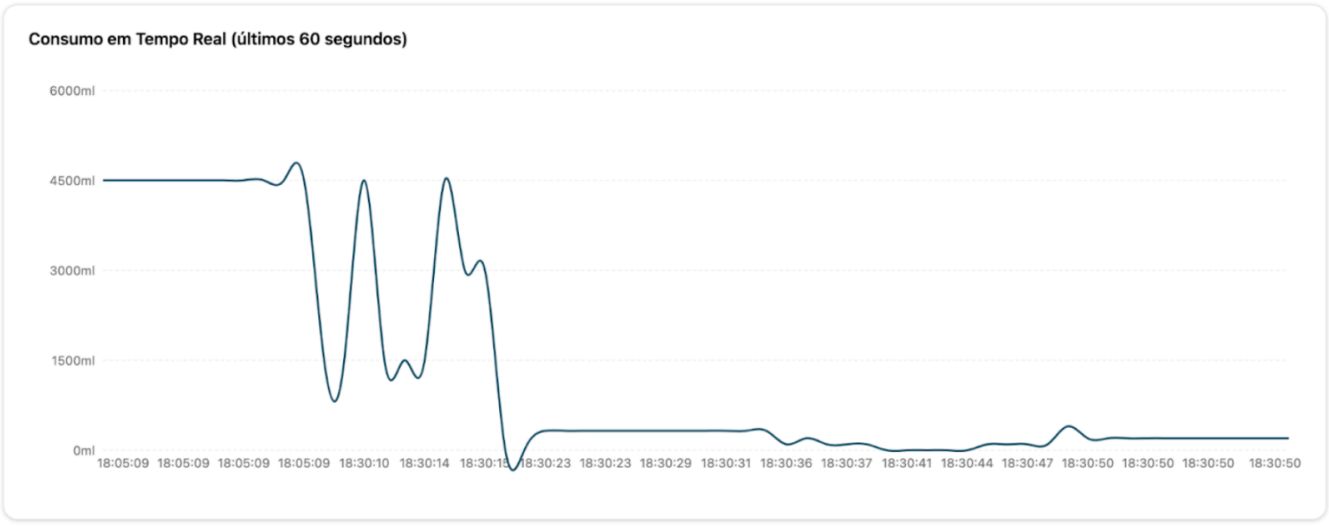
3.6.4 Gráfico de Valor ao Longo do Tempo

Mesma ideia do volume histórico, mas focado no aspecto financeiro, permitindo comparar visualmente evolução de custo.



3.6.5 Gráfico em Tempo Real (Últimos 60s)

Um gráfico que se atualiza a cada segundo, exibindo volume ou valor dos últimos 60 segundos, ideal para identificar picos súbitos ou quedas rápidas.



---

4. Conclusão

4.1 Desafios Enfrentados

4.1.1 Problemas de Hardware

Foram experimentadas ao longo do projeto algumas falhas de hardware que dificultaram o desenvolvimento. Inicialmente pensou-se em utilizar a placa ESP8266 em conjunto com uma Raspberry Pi, porém, a placa estava defeituosa e optou-se por utilizar apenas a ESP8266. Além disso, uma vez que o projeto estava rodando em uma versão básica, ele de repente parou de funcionar da noite para o dia, isso fez com que a equipe procurasse uma solução de software quando na verdade tratava-se de um problema de hardware: os fios estavam com defeito.

4.1.2 Desafios de Software

Experimentaram-se também desafios no âmbito de software. A princípio houve algumas dificuldades com o SQL, alguns erros de digitação significaram tempo gasto em debugging de lógica quando na realidade não havia nenhum. Além disso, houve também problemas na hora de juntar os projetos: algumas dependências não coincidiam e foi gasto tempo tentando fazê-las encaixar.

4.2 Aprendizados e Resultados

A equipe sentiu que o trabalho em equipe fluiu bem e a atividade foi produtiva para trabalhar as habilidades interpessoais na sua matriz particular de projeto de tecnologia, que requer delegação de tarefas junto com ferramentas de versionamento. O projeto foi produtivo também para consolidar os conhecimentos de IoT aprendidos na sala de aula e para o entendimento de que soluções de TI que podem ter um impacto grande podem ser realizadas sem muito esforço.

4.3 Propostas de Melhoria

Como melhorias propõe-se:

1. **Estudo em cenários reais:** Para ver quais desafios surgem nesse contexto e ter evidência empírica de como melhorar o produto
2. **Fabricação de peças:** Articular os componentes eletrônicos em um artefato que faça sentido para o seu uso final, como por exemplo um estojo resistente à umidade

---

5. Apêndice

Código-fonte: https://github.com/gabrielpires-1/IoT-WaterMonitor/tree/main

---