Documentação Completa: Projeto SafeAlert e Protótipo de Monitoramento

1. Recursos Principais

- **Vídeo de Apresentação:** Assista à demonstração do projeto no YouTube
- **Repositório do Código-Fonte:** Acesse o código do protótipo no GitHub

2. Visão Geral do Projeto: SafeAlert

O projeto **SafeAlert** nasce como uma resposta direta e inovadora a um dos maiores desafios socioambientais do Brasil: as enchentes. Conforme detalhado no documento de apresentação, o problema é grave, com 93% dos municípios brasileiros atingidos por desastres naturais entre 2013 e 2022, resultando em mais de 4,2 milhões de desabrigados e prejuízos que ultrapassam R$ 215 bilhões.

2.1. Problema

A crescente frequência e intensidade de eventos climáticos extremos, como as catastróficas enchentes no Rio Grande do Sul em 2024, demonstram a vulnerabilidade de milhões de brasileiros e a necessidade urgente de sistemas de alerta mais eficazes.

2.2. Solução Proposta

A solução é a **SafeAlert**, uma plataforma web colaborativa projetada para ser um centro de informações e alertas sobre riscos de enchentes. Seus principais pilares são:

- **Colaboração em Tempo Real:** Permitir que cidadãos reportem incidentes como alagamentos e árvores caídas, enriquecendo os dados do sistema.
- **Alertas Inteligentes:** Integrar dados de usuários, sensores IoT (como o protótipo desenvolvido) e fontes meteorológicas para gerar alertas textuais, claros e precisos.
- **Evacuação Segura:** Fornecer orientações para que os moradores de áreas de risco possam evacuar de forma segura e antecipada.

2.3. Público-Alvo

A plataforma se destina a cidadãos em áreas de risco, Defesa Civil, autoridades públicas, empresas e qualquer pessoa que precise de informações confiáveis sobre segurança hídrica.

3. Protótipo Técnico: Monitor de Nível de Rio com ESP32 e

Node-RED

Para validar a coleta de dados de sensores, foi desenvolvido um protótipo que simula o monitoramento do nível de um rio. Este sistema utiliza hardware simulado na plataforma Wokwi, que se comunica com uma lógica de processamento de dados no Node-RED através do protocolo MQTT.

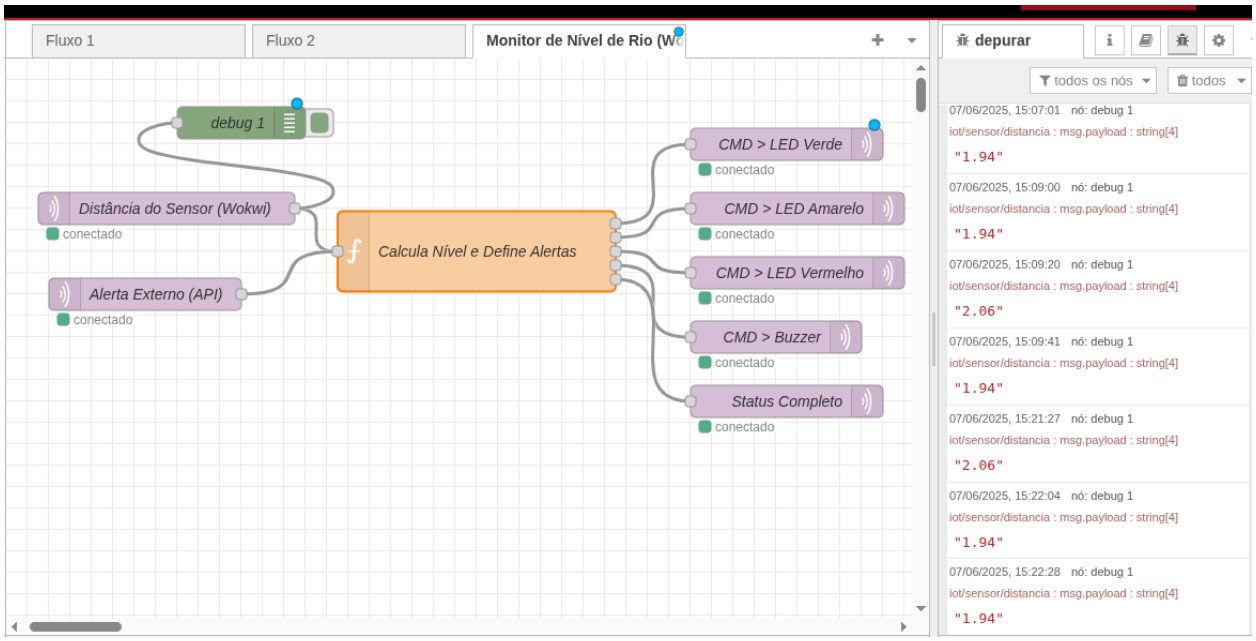
3.1. Arquitetura do Protótipo

O sistema é composto por duas partes principais:

- **Simulação no Wokwi:** Um microcontrolador ESP32 é conectado a um sensor ultrassônico (HC-SR04) para medir a distância até a superfície da água. LEDs (verde, amarelo, vermelho) e um buzzer atuam como sinalizadores de status. O ESP32 se conecta ao Wi-Fi e envia os dados para um broker MQTT.
- **Processamento no Node-RED:** Um fluxo no Node-RED recebe os dados do sensor, calcula o nível real da água, determina o estado de alerta (Normal, Alerta, Perigo) e envia comandos de volta para o ESP32 para acionar os LEDs e o buzzer correspondentes.

4. Análise do Fluxo no Node-RED

O coração da lógica do protótipo reside no Node-RED. O fluxo em operação, definido pelo arquivo node.json, pode ser visualizado abaixo.



4.1. Entradas (Inputs)

Existem duas fontes de dados que alimentam o sistema, ambas usando o protocolo MQTT:

1. **Distância do Sensor (Wokwi)**: Este nó (mqtt in) se inscreve no tópico iot/sensor/distancia. Ele recebe a medição bruta da distância (em cm) que o ESP32 envia a cada poucos segundos.
2. **Alerta Externo (API)**: Este nó (mqtt in) se inscreve no tópico iot/api/alerta. Ele permite que um sistema externo (como uma API da Defesa Civil ou um comando manual) envie um nível de alerta (0 para normal, 1 para alerta, 2 para perigo), sobrepondo-se à leitura do sensor se necessário.

4.2. Processamento Central

- **Calcula Nível e Define Alertas**: Este é o nó de função principal (function) e contém toda a lógica de negócio do protótipo. Ele executa as seguintes ações:
    1. **Armazena o Estado:** Utiliza variáveis de contexto para guardar o último valor recebido do sensor e do alerta externo.
    2. **Calcula o Nível da Água:** A lógica é nivelAgua = ALTURA_SENSOR - distancia. A ALTURA_SENSOR (definida como 300 cm no código) é a altura fixa do sensor em relação ao leito do rio. Assim, quanto menor a distância medida, maior o nível da água.
    3. **Define o Status de Alerta:** Com base no nível da água calculado e no alerta externo, ele aplica as seguintes regras:
        - **PERIGO**: Se o nivelAgua ultrapassar o NIVEL_ALERTA (220 cm) OU se o alertaApi for 2. Neste caso, o LED vermelho e o buzzer são ativados.
        - **ALERTA**: Se o nivelAgua ultrapassar o NIVEL_NORMAL (150 cm) OU se o alertaApi for 1. O LED amarelo é ativado.
        - **NORMAL**: Em todos os outros casos. O LED verde é ativado.

4.3. Saídas (Outputs)

O nó de função tem 5 saídas, cada uma conectada a um nó de publicação MQTT (mqtt out) para controlar os atuadores no Wokwi ou para depuração:

- **CMD > LED Verde**: Publica 1 ou 0 no tópico iot/led/verde.
- **CMD > LED Amarelo**: Publica 1 ou 0 no tópico iot/led/amarelo.
- **CMD > LED Vermelho**: Publica 1 ou 0 no tópico iot/led/vermelho.
- **CMD > Buzzer**: Publica 1 ou 0 no tópico iot/buzzer/estado.
- **Status Completo**: Publica um objeto JSON no tópico iot/status/dados com todas as informações, útil para monitoramento e depuração.

5. Código-Fonte Explicado

5.1. Código do Node-RED (Lógica Principal)

Este é o código JavaScript contido no nó de função "Calcula Nível e Define Alertas", extraído do arquivo node.json.

```
// --- Constantes de Configuração (ajuste conforme necessário) ---
const ALTURA_SENSOR = 300.0; // Altura do sensor em relação ao leito do rio (cm)
const NIVEL_NORMAL = 150.0;    // Nível máximo para status 'Normal' (cm)
const NIVEL_ALERTA = 220.0;    // Nível máximo para status 'Alerta' (cm)

// --- Leitura e Armazenamento de Dados ---
let distancia = context.get('distancia') || 300;
let alertaApi = context.get('alertaApi') || 0;

if (msg.topic === 'iot/sensor/distancia') {
    distancia = parseFloat(msg.payload);
    context.set('distancia', distancia);
} else if (msg.topic === 'iot/api/alerta') {
    alertaApi = parseInt(msg.payload, 10);
    context.set('alertaApi', alertaApi);
}

// --- Lógica de Cálculo e Decisão ---
let nivelAgua = ALTURA_SENSOR - distancia;
if (nivelAgua < 0) { nivelAgua = 0; }

let estado = { normal: 0, alerta: 0, perigo: 0, buzzer: 0 };

if ((nivelAgua > NIVEL_ALERTA && nivelAgua <= ALTURA_SENSOR) || alertaApi === 2) {
    estado.perigo = 1;
    estado.buzzer = 1;
} else if ((nivelAgua > NIVEL_NORMAL && nivelAgua <= NIVEL_ALERTA) || alertaApi ===
1) {
    estado.alerta = 1;
} else {
    estado.normal = 1;
}

// --- Preparação das Mensagens de Saída ---
let dadosOut = {
```

```
    payload: {
        distancia: distancia,
        nivelAgua: nivelAgua.toFixed(2),
        alertaApi: alertaApi,
        status: estado
    }
};

return [
    { topic: 'iot/led/verde', payload: estado.normal },
    { topic: 'iot/led/amarelo', payload: estado.alerta },
    { topic: 'iot/led/vermelho', payload: estado.perigo },
    { topic: 'iot/buzzer/estado', payload: estado.buzzer },
    { topic: 'iot/status/dados', payload: JSON.stringify(dadosOut.payload) }
];
```

5.2. Código do ESP32 (sketch.ino - Funcionalidade Descrita)

O código C++ para o ESP32, inferido a partir dos arquivos do projeto, executa as seguintes funções:

- **Inclusão de Bibliotecas:** WiFi.h e PubSubClient.h.
- **Configuração de Pinos:** Define os pinos para o sensor HC-SR04, LEDs e buzzer.
- **Conexão Wi-Fi e MQTT:** Conecta-se à rede e ao broker broker.hivemq.com.
- **Assinatura de Tópicos:** Inscreve-se nos tópicos de comando para os LEDs e o buzzer.
- **Função de Callback:** Processa as mensagens recebidas para ligar/desligar os atuadores.
- **Loop Principal:** Mede a distância periodicamente, publica o resultado em iot/sensor/distancia e mantém a conexão MQTT ativa.