Global Solution 2025 - 1º Semestre

Sistema de Monitoramento de Qualidade da Água com IoT e Machine Learning

Link para o Repositório no GitHub:
https://github.com/marcofiap/GS1Sem

Link do Vídeo Demonstrativo (YouTube - Não Listado):
aguardando link...

Integrantes do Grupo:

- Felipe Sabino da Silva - RM 563569
- Juan Felipe Voltolini - RM 562890
- Luiz Henrique Ribeiro de Oliveira - RM 563077
- Marco Aurélio Eberhardt Assumpção - RM 563348
- Paulo Henrique Senise - RM 565781

1. Introdução

Quando começamos a pensar sobre a Global Solution 2025 e o desafio de desenvolver soluções para eventos naturais extremos, nossa primeira reação foi: "por onde começar?". Depois de muito debate no grupo, decidimos focar em algo que nos chamou bastante atenção - a contaminação da água após desastres como enchentes e deslizamentos.

A escolha não foi por acaso. Pesquisando sobre o assunto, descobrimos que esses eventos podem contaminar completamente os reservatórios de água potável, e isso vira um problema de saúde pública. Imaginem pessoas que já perderam suas casas numa enchente tendo que lidar ainda com água contaminada.

Nossa abordagem com os dados:
Decidimos usar dados reais de qualidade da água que mostram como ela fica depois de desastres naturais. Encontramos no kaggle.com informações sobre como esses eventos afetam a infraestrutura hídrica, e isso nos ajudou a entender melhor o problema. O dataset "Water Quality" foi perfeito para nossa ideia porque tem exatamente os parâmetros que ficam alterados quando acontecem enchentes e deslizamentos (pH, turbidez, cloro, etc.).

Daí surgiu nossa solução: um **Sistema de Monitoramento de Qualidade da Água** que junta IoT com Machine Learning para analisar se a água está própria para consumo em tempo real. A ideia é usar tudo que aprendemos até agora - programação em Python, lógica, estruturação de dados e algoritmos de ML.

O que queremos provar com este projeto:
Basicamente, queremos mostrar que dá para criar um sistema que:

- Pega dados de sensores instalados em lugares estratégicos (usando ESP32)
- Processa esses dados com algoritmos de Machine Learning
- Toma decisões automáticas sobre a qualidade da água
- Mostra tudo isso de forma clara para quem precisa tomar decisões rápidas
- Guarda um histórico para analisar tendências depois dos desastres

2. Desenvolvimento

2.1. Como organizamos nossa solução

Primeira coisa que fizemos foi sentar e pensar: "como vamos integrar tudo que aprendemos até agora?". Acabamos dividindo o projeto usando conhecimentos de várias disciplinas:

O que usamos de cada matéria:

- **Machine Learning:** Criamos um modelo RandomForestClassifier que aprende a classificar se a água está boa ou não
- **Banco de Dados:** Montamos um banco Oracle para guardar todas as leituras e histórico
- **ESP32:** Programamos o microcontrolador para ler sensores de pH, turbidez e cloro
- **Lógica e Programação:** Usamos bastante if/else, loops for e while, algoritmos de decisão
- **Estruturação de Dados:** Limpamos e organizamos os dados, fizemos normalização

As partes do nosso sistema:

1. **O hardware (ESP32):**
    - É o "cérebro" que fica no campo lendo os sensores
    - Tem uma telinha OLED que mostra os dados na hora
    - LEDs verde e vermelho para indicar se a água está boa ou ruim
    - Conecta no Wi-Fi para mandar os dados
2. **O backend (Flask em Python):**
    - Recebe os dados que o ESP32 manda
    - Organiza tudo e chama o modelo de ML

- Cuida da comunicação entre todas as partes

3. **A inteligência (Machine Learning):**
   - Modelo que treinamos com dados reais
   - Analisa os parâmetros e decide se a água está potável
   - Funciona em tempo real
4. **O banco de dados (Oracle):**
   - Guarda todas as leituras que fazemos
   - Permite consultar o histórico depois
   - Estrutura simples mas eficiente
5. **A interface (Streamlit):**
   - Dashboard onde dá para ver tudo que está acontecendo
   - Gráficos mostrando como os dados mudam com o tempo
   - Sistema de alertas quando algo está errado

Como tudo se conecta:
O ESP32 lê os sensores, manda via Wi-Fi para nossa API Flask, que processa com o modelo de ML, salva no banco Oracle e mostra no dashboard Streamlit. Parece complicado, mas na prática funciona bem fluido.

2.2. A parte do Machine Learning

Essa foi uma das partes mais desafiadoras do projeto. Usamos o arquivo `src/model/train.py` como base e escolhemos o RandomForestClassifier porque, depois de testar alguns algoritmos, foi o que deu melhores resultados com nossos dados.

Como desenvolvemos o modelo:

1. **Limpeza dos dados:**
   Primeiro tivemos que lidar com dados faltando (valores NaN). Nossa solução foi bem simples:

```
# Se tem dados faltando, preenchemos com a mediana
if dados.isnull().any():
    dados = dados.fillna(dados.median())

# Normalizamos cada coluna
for coluna in dados.columns:
    dados[coluna] = normalizar_dados(dados[coluna])
```

2. **Divisão dos dados:**
   - 80% para treinar o modelo (2.637 amostras)
   - 20% para testar se está funcionando (659 amostras)

3. **Normalização:**
   Usamos o StandardScaler para deixar todas as variáveis na mesma escala, senão o modelo pode dar mais importância para valores maiores.
4. **Treinamento:**
   Treinamos o modelo e testamos com validação cruzada para ter certeza que não estava "decorando" os dados.
5. **Salvamento:**
   No final, salvamos tudo num arquivo `.joblib` que nossa API Flask consegue carregar e usar.

2.3. Programação do ESP32

A programação do ESP32 (arquivo `simularsensor/src/main.cpp`) foi bem interessante. Tivemos que pensar em como fazer ele ler vários sensores, processar os dados e ainda se comunicar via Wi-Fi.

A lógica que implementamos:

```
void loop() {
    // Fica esperando alguém apertar o botão
    if (digitalRead(BOTAO_PIN) == LOW) {

        // Lê todos os sensores em sequência
        for (int i = 0; i < NUM_SENSORES; i++) {
            leituras[i] = lerSensor(sensores[i]);
            delay(100);  // Pequena pausa entre leituras
        }

        // Manda os dados e recebe a resposta
        if (enviarDados(leituras)) {
            if (resposta == "POTAVEL") {
                // Acende LED verde
                digitalWrite(LED_VERDE, HIGH);
                digitalWrite(LED_VERMELHO, LOW);
            } else {
                // Acende LED vermelho
                digitalWrite(LED_VERDE, LOW);
                digitalWrite(LED_VERMELHO, HIGH);
            }
        }
    }

    delay(1000);  // Aguarda 1 segundo antes de verificar novamente
}
```

O que conseguimos implementar:

- Conexão automática no Wi-Fi (com tentativas caso falhe)
- Leitura de múltiplos sensores de forma sequencial
- Comunicação HTTP com nosso backend
- Display em tempo real dos dados no OLED
- Sinalização visual clara com LEDs
- Tratamento de erros quando a conexão cai

2.4. Backend e banco de dados

O backend foi desenvolvido em Flask (`src/api/servidor.py`) de forma bem direta. Criamos um endpoint `/data` que recebe os dados via GET e processa tudo.

O controller (`src/api/controller.py`) é onde acontece a mágica - ele pega os dados do ESP32, chama o modelo de ML e salva tudo no banco.

Nossa estrutura no banco:

```
CREATE TABLE readings (
    id NUMBER PRIMARY KEY,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ph NUMBER(5,2),
    turbidity NUMBER(8,3),
    chlorine NUMBER(8,3),
    prediction VARCHAR2(20),
    confidence NUMBER(5,4)
);
```

A lógica do controller ficou assim:

```
def processar_leitura(dados_sensores):
    # Primeiro validamos se os dados fazem sentido
    if validar_dados(dados_sensores):

        # Checamos se algum valor está muito alto
        for parametro in dados_sensores:
            if parametro['valor'] > limite_critico:
                gerar_alerta(parametro)

        # Fazemos a predição
        resultado = modelo.predict([dados_sensores])

        # Decidimos se é potável ou não
        if resultado[0] == 1:
            status = "POTAVEL"
        else:
            status = "NAO_POTAVEL"

        return status
```

2.5. Interface do usuário

O dashboard em Streamlit (`src/ui/app.py`) foi pensado para ser simples de usar. Queremos que qualquer pessoa consiga entender rapidamente o que está acontecendo com a qualidade da água.

Principais funcionalidades:

- Métricas básicas (quantas leituras, % de água potável, etc.)
- Gráficos que mostram como os parâmetros mudam ao longo do tempo
- Tabela com histórico de todas as leituras
- Possibilidade de testar o modelo manualmente com valores próprios

Exemplo da lógica que usamos:

```
def atualizar_dashboard():
    leituras = obter_leituras_recentes()

    contador_potavel = 0
    contador_nao_potavel = 0

    # Conta quantas são potáveis e quantas não são
    for leitura in leituras:
        if leitura['prediction'] == 'POTAVEL':
            contador_potavel += 1
        else:
            contador_nao_potavel += 1

    # Se muita água não potável, mostra alerta
    porcentagem_nao_potavel = (contador_nao_potavel / len(leituras)) * 100
    if porcentagem_nao_potavel > 30:
        st.error("⚠️ ATENÇÃO: Muita água não potável detectada!")

    return metricas
```

3. Resultados Esperados

Acreditamos que nossa solução pode fazer uma diferença real em situações de desastre. Não é só um projeto acadêmico - é algo que poderia ser usado de verdade.

O que esperamos conseguir:

Sistema de alerta rápido: Quando acontece uma enchente, por exemplo, nosso sistema conseguiria detectar rapidamente se a água ficou contaminada, permitindo que as autoridades tomassem providências antes que alguém consuma água contaminada.

Decisões baseadas em dados: Em vez de depender apenas da experiência humana (que pode falhar em situações de stress), o sistema analisa múltiplos parâmetros ao mesmo tempo e dá uma

resposta objetiva.

Monitoramento 24 horas: O sistema pode ficar funcionando direto, mesmo em lugares onde é difícil mandar técnicos para fazer análises manuais.

Fácil de expandir: A arquitetura que criamos permite instalar vários pontos de monitoramento rapidamente em uma região afetada.

Algumas métricas que conseguimos:

- Precisão do modelo: mais de 85% nos testes
- Tempo de resposta: menos de 2 segundos do sensor até o dashboard
- Sistema consegue processar mais de 1000 leituras por hora

4. Conclusões

Foi um projeto desafiador, mas conseguimos aplicar praticamente tudo que aprendemos durante o curso. O mais legal foi ver como as diferentes disciplinas se conectam na prática - o que às vezes parecia desconectado nas aulas fez muito sentido quando juntamos tudo.

O que conseguimos entregar:

- Sistema completo funcionando com ML em Python e ESP32 com sensores
- Todos os códigos testados e operacionais
- Documentação organizada (Introdução, Desenvolvimento, Resultados e Conclusões)
- Código disponível no GitHub
- Vídeo demonstrativo

Onde nossa solução poderia ser usada:

- Acampamentos de desabrigados após enchentes
- Monitoramento de reservatórios depois de deslizamentos
- Controle de qualidade em distribuição de água emergencial

O que aprendemos:
Este projeto mostrou como a tecnologia pode realmente ajudar em problemas do mundo real. Não é só teoria - é algo que pode salvar vidas. A combinação de IoT, ML e estruturação de dados criou uma solução que pode ser adaptada para diferentes tipos de desastres naturais.

Próximas melhorias:

- Adicionar mais tipos de sensores (metais pesados, bacteriológicos)

- Implementar alertas por SMS para autoridades
- Criar mapas para mostrar a situação geograficamente
- Desenvolver um app para celular

Enfim, este projeto provou que quando juntamos o conhecimento técnico com um problema real, conseguimos criar soluções que podem fazer a diferença. Foi gratificante ver tudo funcionando no final e saber que desenvolvemos algo que poderia ser usado em situações reais de emergência.

Declaração:

Este projeto foi desenvolvido integralmente pelo nosso grupo, com pesquisa própria e implementações originais.