Projeto Final de Robôs Movíveis Inteligentes

Grupo 4

Objetivo:
Desenvolver um sistema de monitoramento que registra dados de temperatura e umidade, salvando-os em um arquivo CSV. Em caso de desconexão da rede Wi-Fi, os dados são registrados localmente e, quando a conexão for restabelecida, eles são enviados para um banco de dados MongoDB via uma API desenvolvida em Python. A aplicação também deve permitir consultas dos dados via requisições utilizando Swagger.

Componentes:

1. **ESP32/ESP8266**: microcontrolador principal.
2. **Sensor de Umidade e Temperatura** (ex: DHT11 ou DHT22): para medir temperatura e umidade.
4. **Conexão Wi-Fi**: para enviar dados para o MongoDB.
5. **Servidor Python**: API para processar e armazenar os dados no MongoDB.

Funcionalidades:

1. Monitoramento de Temperatura e Umidade:

- O sistema lê os dados de temperatura e umidade usando o sensor DHT11/DHT22.
- Os dados são registrados periodicamente em um arquivo CSV com a data e hora da coleta.

2. Data Logger Local (CSV):

- Se a conexão Wi-Fi estiver disponível, os dados são enviados diretamente ao MongoDB via a API Python.
- Se a conexão Wi-Fi for perdida, os dados são armazenados em um arquivo CSV local. Cada registro no CSV contém:
  - **Temperatura**
  - **Umidade**

- **Hora da Coleta**
- **Timestamp da Desconexão da Rede (se aplicável)**
- **Timestamp da Reconexão da Rede (se aplicável)**

3. Reconexão e Envio de Dados ao MongoDB:

- Quando a conexão Wi-Fi for restabelecida, o ESP32 enviará os dados salvos localmente para o MongoDB por uma API Python.
- A API gerenciará o armazenamento dos dados, garantindo que os registros de desconexão e reconexão também sejam enviados para o MongoDB no banco de dados **Robos_I** na tabela **Grupo_4**
- O sistema em ESP deve verificar frequentemente a disponibilidade da rede, para reconectar.

4. Interface API via Swagger:

- Uma API Python será criada usando frameworks como Flask ou FastAPI, integrando o Swagger para facilitar o uso de requisições REST.
- Através da interface do Swagger, será possível:
  - Consultar dados históricos de temperatura e umidade.
  - Filtrar dados por período, visualizando informações de quando houve falha e reconexão na rede.
  - Consultar o status atual do sistema (última leitura de temperatura e umidade).
- Exemplo de endpoints:
  - GET /dados: retorna todos os dados de temperatura e umidade.
  - GET /dados: start={data}&end={data}: Retorna dados dentro de um período específico.
  - GET /status: retorna o status atual do sistema.

Passos para o Desenvolvimento:

1. Desenvolvimento do Código em MicroPython:

- **Leitura de Sensores**: Implemente a função de leitura do sensor de umidade e temperatura.
- **Verificação da Conexão Wi-Fi**: Escreva uma função que verifique se a rede Wi-Fi está disponível.
- **Armazenamento Local (CSV)**: Se a rede Wi-Fi estiver indisponível, grave os dados no formato CSV.
- **Envio de Dados para API Python**: Quando a conexão Wi-Fi for restaurada, envie os dados do CSV para o servidor Python via uma requisição HTTP POST.

3. Desenvolvimento da API em Python:

- Use Flask ou FastAPI para criar uma API RESTful.
- Implemente rotas que permitam receber dados do ESP32 e armazená-los no MongoDB.
- Adicione integração com Swagger para documentar e testar a API.

5. Testes e Validação:

- Teste o sistema verificando o comportamento durante períodos de desconexão da rede Wi-Fi.
- Verifique se os dados são armazenados no CSV corretamente e se, após a reconexão, eles são enviados para o MongoDB.
- Valide o funcionamento da API via Swagger, testando diferentes endpoints e visualizando os dados.