RELATÓRIO CARDIO IA

ARMAZENAMENTO E PROCESSAMENTO LOCAL (EDGE COMPUTING):

Nome: Diego Nunes Veiga

RM:560658

Turma: 2TIAOR

1. OBJETIVO

O objetivo deste projeto é demonstrar a aplicação prática dos conceitos de Edge Computing em um sistema embarcado baseado no ESP32, voltado para o monitoramento de sinais vitais em aplicações de saúde digital.
A proposta consiste em criar um nó de borda capaz de coletar dados de sensores, armazenar localmente em memória flash utilizando o sistema de arquivos SPIFFS, e sincronizar automaticamente esses dados com a nuvem assim que a conectividade for restabelecida.

Além disso, o projeto deve garantir resiliência offline, permitindo a coleta contínua mesmo em situações de falha de rede — um requisito essencial para sistemas críticos como os de monitoramento médico.

2. Explicação do Desenvolvimento do Software

O sistema foi desenvolvido e testado na plataforma Wokwi, utilizando a placa ESP32. Dois sensores foram integrados ao projeto:

- **DHT22**, responsável pela medição de temperatura e umidade;
- **Potenciômetro**, atuando como segundo sensor analógico, simulando um sinal fisiológico, como a oxigenação do sangue.

Além dos sensores, foi utilizada uma chave deslizante (slide switch) para simular o estado da conectividade Wi-Fi. Quando a chave está desligada (nível lógico baixo), o sistema opera em modo offline, armazenando as leituras localmente em um arquivo chamado /buffer.txt. Quando a chave é ligada (nível lógico alto), o sistema entra em modo online, e todas as leituras armazenadas são sincronizadas via Serial.println(), simulando o envio à nuvem. Após o envio completo, o buffer local é apagado automaticamente.

Durante a inicialização, o programa configura os pinos de entrada e saída, inicializa os sensores, monta o sistema de arquivos SPIFFS e exibe mensagens de status no monitor serial.
O loop principal é responsável por realizar a leitura contínua dos sensores e aplicar análises simples para identificar o estado do paciente. A temperatura é classificada como "baixa", "normal", "febre" ou "febre alta", enquanto o valor do potenciômetro é convertido em porcentagem, sendo classificado como "normal", "leve", "moderada" ou "grave".

Cada leitura gera um registro JSON, que contém o timestamp, os valores dos sensores e seus respectivos estados. Esse registro é gravado em uma nova linha do arquivo de buffer e, quando o sistema está online, também é impresso no terminal serial. O programa foi modularizado em diversas funções (subalgoritmos), separando a leitura dos sensores, a análise dos dados e o gerenciamento do sistema de arquivos, o que torna o código mais legível e facilita futuras expansões.

O sistema também inclui uma política de armazenamento limitado, configurada para até 300 KB. Ao atingir esse limite, o arquivo é automaticamente excluído, garantindo que o ESP32 continue operando sem sobrecarregar a memória. Essa abordagem imita um buffer circular, típico em aplicações de Edge Computing, onde o nó prioriza a coleta contínua em detrimento do armazenamento infinito, mantendo sempre as informações mais recentes.

Em testes realizados na simulação do Wokwi, o sistema apresentou comportamento estável: os dados eram gravados corretamente no modo offline, sincronizados quando a conectividade era restabelecida e o arquivo era apagado conforme previsto. A alternância entre os modos foi controlada pela chave deslizante, simulando de forma simples e eficaz as falhas e retomadas de conexão à nuvem.

3. SUBALGORITMOS E EXPLICAÇÕES

O software foi estruturado em subalgoritmos específicos para modularizar e simplificar o funcionamento. A seguir, estão descritas as principais funções:

- **LeituraWifi(int pin)**
  Realiza a leitura do pino associado à chave de conectividade, retornando true (online) ou false (offline). É o gatilho principal que determina o comportamento de sincronização dos dados.

- **LeituraTemperatura()**
  Faz a leitura do sensor **DHT22**, retornando o valor da temperatura em graus Celsius. Possui tratamento para valores inválidos (NaN), assegurando a confiabilidade das medições.

- **LeituraOximetro(int pin)**
  Captura o valor analógico do potenciômetro e converte para porcentagem (0 a 100%), representando a oxigenação simulada do paciente.

- **AnaliseTemperatura(float temp)**
  Classifica o valor de temperatura em quatro categorias:
  - < 35°C → "baixa";
  - 35°C–37,5°C → "normal";
  - 37,5°C–39°C → "febre";
  - 39°C → "febre alta".

- **AnaliseOxigenio(float oxi)**
  Classifica o valor percentual de oxigenação em:
  - ≥ 95% → "normal";
  - 93%–94% → "leve";
  - 90%–92% → "moderada";
  - < 90% → "grave".

- **MontaRegistroJSON(float temp, String statusTemp, float oxi, String statusOxi)**
  Cria um registro no formato JSON contendo:
  - Timestamp (em milissegundos desde o boot);
  - Temperatura e status;
  - Nível de oxigenação e status.
    O JSON é gravado no buffer local e, quando o sistema está online, também é enviado ao terminal serial.

- **FS_Init()**
  Inicializa o sistema de arquivos SPIFFS, garantindo que esteja montado corretamente. Caso ocorra falha, realiza uma formatação e tenta montar novamente.

- ****FS_AppendLine(const char path, String line)***
  Abre o arquivo /buffer.txt e adiciona uma nova linha no final (modo append). Cada linha contém uma amostra de dados em formato JSON.

- ****FS_EnforceLimit(const char path, size_t maxBytes)***
  Verifica o tamanho atual do arquivo. Se ultrapassar 300 KB, o arquivo é apagado, evitando que o sistema de arquivos fique cheio e comprometendo novas gravações.

- ***SyncBufferParaNuvem(const char path)**
  Responsável pela sincronização dos dados com a nuvem simulada. Lê linha por linha do arquivo de buffer e envia via Serial.println(). Após o envio completo, remove o arquivo do SPIFFS.

- **StatusTemperatura() e StatusOxigenio()**
  Exibem no terminal o valor lido e o estado classificado, facilitando o acompanhamento visual durante a simulação.

- **LimpaSerial()**
  Imprime várias quebras de linha para limpar a tela do terminal serial, organizando a visualização dos dados.

4. RESULTADOS OBTIDOS

Os testes realizados no Wokwi confirmaram que o sistema funciona conforme os objetivos propostos. No modo offline, as leituras foram armazenadas corretamente no arquivo /buffer.txt e o tamanho do arquivo aumentava a cada ciclo de coleta. Quando o modo online era ativado, o sistema lia todo o conteúdo armazenado, enviava linha a linha via Serial.println() e, ao final da transmissão, apagava o arquivo de buffer. Essa operação foi repetida diversas vezes, confirmando a resiliência offline e o comportamento de sincronização automática.

A leitura e classificação de temperatura e oxigenação também se mostraram consistentes. As mensagens exibidas no monitor serial apresentavam claramente o status de cada variável e indicavam o estado de conectividade, facilitando o entendimento do fluxo de operação. A política de armazenamento limitado de 300 KB funcionou corretamente, impedindo travamentos e garantindo que o sistema pudesse operar indefinidamente, mesmo em cenários prolongados de desconexão.

5. CONCLUSÃO

O projeto atendeu integralmente aos objetivos da Parte 1, implementando um sistema de Edge Computing funcional e resiliente. O nó de borda desenvolvido demonstrou capacidade de coleta contínua, armazenamento local seguro e sincronização automática com a nuvem simulada, mesmo em situações de falha de conectividade. A modularização por meio dos subalgoritmos facilitou a leitura, manutenção e escalabilidade do código, permitindo que futuras versões integrem protocolos reais de comunicação, como MQTT, e sistemas de visualização em nuvem, como Node-RED ou Grafana Cloud.

O sistema está, portanto, totalmente preparado para evoluir para a Parte 2 – Fog/Cloud Computing, completando a arquitetura IoT com envio real de dados e visualização em dashboards.