André Ferreira da Silva RA: 25027670
Cauan Moreira da Silva RA:25027645
Victor Bancatelli Lucena Lopes RA:25027658

Projeto interdisciplinar (Algoritmo e Logica de programação)
Entrega 2

O projeto a seguir retrata de um trecho do nosso código, no caso, a classe SensorService, que representa uma parcela dos comando utilizado para formar nosso dashboard.

INICIO DO EXEMPLO:

```
// Define a classe responsável por processar dados dos sensores
public class SensorService
{
    // Método principal que processa os sensores em um intervalo de tempo e
retorna um resumo de consumo
    public SensorResponse ProcessarSensores(DateTime inicio, DateTime fim)
    {
        // Dicionário que armazena o consumo agrupado por dia
        var consumoPorDia = new Dictionary<DateTime, double>();

        // Instancia o acesso ao banco de dados
        var db = new BancoDados();

        // Busca os sensores no intervalo de tempo fornecido
        List<Sensor> sensores = db.ObterSensoresPorPeriodo(inicio, fim);

        // Se houver menos de dois sensores, não é possível calcular o
consumo, retorna resposta vazia
        if (sensores.Count < 2)
            return new SensorResponse();

        // Percorre os sensores, exceto o último (pois será comparado com o
próximo)
        for (int i = 0; i < sensores.Count - 1; i++)
        {
            // Sensor atual
            var atual = sensores[i];

            // Próximo sensor na lista
            var proximo = sensores[i + 1];

            // Calcula o intervalo de tempo em horas entre os dois sensores
            double horas = (proximo.TimeStamp - atual.TimeStamp).TotalHours;

            // Se o intervalo for menor ou igual a zero, ignora (possível
erro ou dados fora de ordem)
            if (horas <= 0) continue;

            // Calcula a potência com base no tipo (id) do sensor
            double potencia = CalcularPotencia(atual.Id_sensor);

            // Calcula o consumo como potência multiplicada pelas horas
            double consumo = potencia * horas;

            // Obtém apenas a data (sem hora) do timestamp do sensor atual
            DateTime dia = atual.TimeStamp.Date;
```

```
            // Se a data ainda não estiver no dicionário, adiciona com valor
inicial zero
            if (!consumoPorDia.ContainsKey(dia))
                consumoPorDia[dia] = 0;

            // Soma o consumo calculado ao total do dia correspondente
            consumoPorDia[dia] += consumo;
        }

        // Retorna um objeto SensorResponse preenchido com os dados
calculados:
        return new SensorResponse
        {
            // Dicionário com consumo agrupado por dia
            ConsumoPorDia = consumoPorDia,

            // Soma total do consumo no período
            ConsumoTotalMes = consumoPorDia.Values.Sum(),

            // Consumo médio diário
            ConsumoMediaDia = consumoPorDia.Values.Any() ?
consumoPorDia.Values.Average() : 0,

            // Consumo máximo em um único dia
            ConsumoMax = consumoPorDia.Values.Any() ?
consumoPorDia.Values.Max() : 0,

            // Consumo mínimo em um único dia
            ConsumoMin = consumoPorDia.Values.Any() ?
consumoPorDia.Values.Min() : 0
        };
    }

    // Método auxiliar que determina a potência de um sensor com base no seu
ID
    private double CalcularPotencia(int idSensor) =>
        idSensor switch
        {
            1 or 2 => 1.5,    // Sensores com ID 1 ou 2 consomem 1.5 kW
            3 => 0.05,        // Sensor com ID 3 consome 0.05 kW
            4 => 3.0,         // Sensor com ID 4 consome 3.0 kW
            5 => 7.0,         // Sensor com ID 5 consome 7.0 kW
            _ => 0.0          // Outros sensores desconhecidos consomem 0.0
kW
        };
}
FIM DO EXEMPLO
```