Estruturas de Dados Utilizadas:

Listas (List<SensorData>) para armazenar e manipular coleções de dados de sensores

Dicionários (Dictionary<int, string> e Dictionary<string, List<(string, double)>>) para mapear sensores para cômodos e cômodos para dispositivos

Tuplas ((string, double)) para representar pares de nome e consumo de dispositivos

Padrões Algorítmicos

Iteração e Acumulação: Usado para calcular somas de consumo

Filtro e Mapeamento: Usado para transformar e selecionar dados

Busca de Máximo/Mínimo: Usado para encontrar o cômodo com maior consumo

Agrupamento e Agregação: Usado para processar dados de sensores por grupos

Resumo das Funções e Algoritmos

LoadSensorData(): Implementa um algoritmo de processamento de arquivo CSV com validação e conversão de dados.

CalculateCurrentConsumption(): Implementa um algoritmo de cálculo com iteração aninhada e regras condicionais para ajustar o consumo com base em dados dos sensores.

EstimateMonthlyEnergyCost(): Utiliza o resultado do cálculo de consumo atual para estimar o custo mensal de energia.

CalculateRoomMonthlyConsumption(): Implementa um algoritmo complexo com múltiplas regras condicionais para diferentes tipos de dispositivos e padrões de uso.

FindHighestConsumptionRoom(): Implementa um algoritmo de busca do máximo para identificar o cômodo com maior consumo de energia.

GetLatestSensorData(): Utiliza LINQ para agrupar e selecionar os dados mais recentes de cada sensor.

GetSensorData(): Utiliza LINQ para filtrar e ordenar dados de um sensor específico.

GetRoomName() e GetRoomDevices(): Implementam algoritmos de busca em dicionário com tratamento de caso padrão.

Início do Código:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace SmartHomeDashboard
{
    public class SmartHomeMonitor
    {
        // Classe para representar dados do sensor
        public class SensorData
        {
            public DateTime Timestamp { get; set; }
            public int SensorId { get; set; }
            public double Temperature { get; set; }
            public double Humidity { get; set; }
            public int Movement { get; set; }
        }

        // Mapeamento de sensores para cômodos
        private Dictionary<int, string> sensorToRoom = new Dictionary<int,
string>()
        {
            { 1, "Sala" },
            { 2, "Cozinha" },
            { 3, "Quarto Principal" },
            { 4, "Banheiro" },
            { 5, "Escritório" }
        };
```

```
        // Lista para armazenar todos os dados dos sensores
        private List<SensorData> allSensorData = new List<SensorData>();

        // Preço do kWh
        private const double PRICE_PER_KWH = 0.75; // R$ 0,75 por kWh

        // Consumo base por dispositivo em cada cômodo (em Watts)
        private Dictionary<string, List<(string, double)>> roomDevices = new
Dictionary<string, List<(string, double)>>()
        {
            { "Sala", new List<(string, double)> { ("TV", 120), ("Iluminação", 60), ("Ar
Condicionado", 1200) } },
            { "Cozinha", new List<(string, double)> { ("Geladeira", 300),
("Microondas", 1000), ("Iluminação", 80) } },
            { "Quarto Principal", new List<(string, double)> { ("Iluminação", 40), ("Ar
Condicionado", 900), ("Carregadores", 20) } },
            { "Banheiro", new List<(string, double)> { ("Iluminação", 40), ("Chuveiro
Elétrico", 4500), ("Ventilador", 65) } },
            { "Escritório", new List<(string, double)> { ("Computador", 200),
("Iluminação", 60), ("Impressora", 45) } }
        };

        // Construtor
        public SmartHomeMonitor()
        {
            LoadSensorData();
        }

        /* FUNÇÃO: LoadSensorData()
         * ALGORITMO: Processamento de arquivo CSV
         * - Lê um arquivo CSV linha por linha
         * - Analisa (parsing) e converte dados em diferentes formatos
         * - Realiza validação de dados
```

```
         * - Armazena os dados em uma estrutura de dados (lista)
         * - Ordena os dados cronologicamente
         */
        public void LoadSensorData()
        {
            try
            {
                // Caminho para o seu arquivo CSV
                string csvFilePath = "D:/Banco de Dados/BD_Casa_Inteligente
(1).csv";

                // Ler todas as linhas do arquivo
                string[] lines = System.IO.File.ReadAllLines(csvFilePath);

                // Pular o cabeçalho se existir
                for (int i = 1; i < lines.Length; i++)
                {
                    string line = lines[i];
                    string[] values = line.Split(',');

                    if (values.Length >= 5)
                    {
                        SensorData data = new SensorData();

                        // Tratamento da data
                        if (values[0].Contains("/"))
                        {
                            // Formato de data como "18/1/25 10:28"
                            data.Timestamp = DateTime.Parse(values[0]);
                        }
                        else
```

```
{
    // Formato numérico (Excel)
    double excelDate;
    if (double.TryParse(values[0], out excelDate))
    {
        data.Timestamp = DateTime.FromOADate(excelDate);
    }
    else
    {
        continue; // Pular linha com formato inválido
    }
}

// Converter os outros valores
int sensorId;
double temperature, humidity;
int movement;

if (int.TryParse(values[1], out sensorId) &&
    double.TryParse(values[2], out temperature) &&
    double.TryParse(values[3], out humidity) &&
    int.TryParse(values[4], out movement))
{
    data.SensorId = sensorId;
    data.Temperature = temperature;
    data.Humidity = humidity;
    data.Movement = movement;

    allSensorData.Add(data);
}
```

```
            }
        }

        // Ordenar por data
        allSensorData = allSensorData.OrderBy(d => d.Timestamp).ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine("Erro ao carregar dados: " + ex.Message);
    }
}

/* FUNÇÃO: CalculateCurrentConsumption()
 * ALGORITMO: Cálculo de consumo com iteração aninhada
 * - Itera sobre uma coleção de dados de sensores
 * - Para cada sensor, mapeia para um cômodo
 * - Para cada dispositivo no cômodo, aplica regras de ajuste de consumo
 * - Acumula o consumo total
 */
public double CalculateCurrentConsumption(List<SensorData> latestData)
{
    double totalConsumption = 0;

    foreach (var data in latestData)
    {
        if (sensorToRoom.ContainsKey(data.SensorId))
        {
            string roomName = sensorToRoom[data.SensorId];

            // Obter dispositivos do cômodo
```

```
                var devices = roomDevices[roomName];

                foreach (var device in devices)
                {
                    string deviceName = device.Item1;
                    double baseConsumption = device.Item2;

                    // Ajustar consumo baseado nos sensores
                    double adjustedConsumption = baseConsumption;

                    // Ar condicionado consome mais quando está quente
                    if (deviceName.Contains("Ar Condicionado") &&
data.Temperature > 25)
                    {
                        adjustedConsumption *= 1.5;
                    }

                    // Iluminação só consome quando há movimento
                    if (deviceName.Contains("Iluminação") && data.Movement == 0)
                    {
                        adjustedConsumption *= 0.1; // Standby
                    }

                    // Adicionar ao total (convertendo de W para kW)
                    totalConsumption += adjustedConsumption / 1000;
                }
            }
        }

        return totalConsumption;
    }
```

```
        /* FUNÇÃO: EstimateMonthlyEnergyCost()
         * ALGORITMO: Estimativa de custo mensal
         * - Utiliza o consumo atual calculado
         * - Extrapola para um mês (24 horas x 30 dias)
         * - Aplica o preço por kWh para calcular o custo
         */
        public double EstimateMonthlyEnergyCost(List<SensorData> latestData)
        {
            // Calcular consumo atual
            double currentConsumption =
CalculateCurrentConsumption(latestData);

            // Estimar consumo mensal (assumindo 24 horas por 30 dias)
            double monthlyConsumptionKWh = currentConsumption * 24 * 30;

            // Calcular custo
            return monthlyConsumptionKWh * PRICE_PER_KWH;
        }

        /* FUNÇÃO: CalculateRoomMonthlyConsumption()
         * ALGORITMO: Cálculo complexo com regras condicionais
         * - Calcula o consumo mensal de um cômodo específico
         * - Aplica regras diferentes para cada tipo de dispositivo
         * - Considera padrões de uso variados (horas por dia)
         * - Ajusta o consumo com base em condições ambientais
         */
        public double CalculateRoomMonthlyConsumption(string roomName,
SensorData data)
        {
            double roomConsumption = 0;
```

```
            // Obter dispositivos do cômodo
            var devices = roomDevices[roomName];

            foreach (var device in devices)
            {
                string deviceName = device.Item1;
                double baseConsumption = device.Item2;

                // Ajustar consumo baseado nos sensores
                double adjustedConsumption = baseConsumption;

                // Ar condicionado consome mais quando está quente
                if (deviceName.Contains("Ar Condicionado") && data.Temperature >
25)
                {
                    adjustedConsumption *= 1.5;
                }

                // Iluminação só consome quando há movimento
                if (deviceName.Contains("Iluminação") && data.Movement == 0)
                {
                    adjustedConsumption *= 0.1; // Standby
                }

                // Diferentes padrões de uso para diferentes dispositivos
                double hoursPerDay = 0;

                if (deviceName.Contains("Geladeira"))
                {
                    hoursPerDay = 24; // Ligada o tempo todo
```

```
            }
            else if (deviceName.Contains("Ar Condicionado"))
            {
                hoursPerDay = 8; // 8 horas por dia
            }
            else if (deviceName.Contains("Iluminação"))
            {
                hoursPerDay = data.Movement == 1 ? 6 : 0; // 6 horas se tiver
movimento
            }
            else if (deviceName.Contains("TV"))
            {
                hoursPerDay = 5; // 5 horas por dia
            }
            else if (deviceName.Contains("Computador"))
            {
                hoursPerDay = 8; // 8 horas por dia
            }
            else if (deviceName.Contains("Chuveiro"))
            {
                hoursPerDay = 0.5; // 30 minutos por dia
            }
            else
            {
                hoursPerDay = 2; // Padrão: 2 horas por dia
            }

            // Calcular consumo mensal (kWh)
            double deviceMonthlyConsumption = (adjustedConsumption / 1000) *
hoursPerDay * 30;
            roomConsumption += deviceMonthlyConsumption;
```

```
        }

        return roomConsumption;
    }

    /* FUNÇÃO: FindHighestConsumptionRoom()
     * ALGORITMO: Busca do máximo com comparações sucessivas
     * - Itera sobre todos os cômodos
     * - Calcula o consumo de cada um
     * - Mantém registro do cômodo com maior consumo
     * - Retorna o resultado formatado
     */
    public string FindHighestConsumptionRoom(List<SensorData> latestData)
    {
        string highestRoom = "";
        double highestConsumption = 0;

        foreach (var roomEntry in sensorToRoom)
        {
            string roomName = roomEntry.Value;
            int sensorId = roomEntry.Key;

            var sensorLatestData = latestData.FirstOrDefault(d => d.SensorId ==
sensorId);
            if (sensorLatestData != null)
            {
                double roomConsumption =
CalculateRoomMonthlyConsumption(roomName, sensorLatestData);

                if (roomConsumption > highestConsumption)
                {
```

```
                highestConsumption = roomConsumption;
                highestRoom = roomName;
            }
        }
    }

    return $"{highestRoom} ({highestConsumption:F2} kWh/mês)";
}

/* FUNÇÃO: GetLatestSensorData()
 * ALGORITMO: Agrupamento e seleção usando LINQ
 * - Agrupa os dados por ID do sensor
 * - Para cada grupo, seleciona o registro mais recente
 * - Retorna uma lista com os dados mais recentes de cada sensor
 */
public List<SensorData> GetLatestSensorData()
{
    return allSensorData
        .GroupBy(d => d.SensorId)
        .Select(g => g.OrderByDescending(d => d.Timestamp).First())
        .ToList();
}

/* FUNÇÃO: GetSensorData()
 * ALGORITMO: Filtragem e ordenação usando LINQ
 * - Filtra os dados pelo ID do sensor
 * - Ordena os resultados por data/hora
 * - Retorna uma lista ordenada
 */
public List<SensorData> GetSensorData(int sensorId)
```

```
{
    return allSensorData
        .Where(d => d.SensorId == sensorId)
        .OrderBy(d => d.Timestamp)
        .ToList();
}

/* FUNÇÃO: GetRoomName()
 * ALGORITMO: Busca em dicionário com tratamento de caso padrão
 * - Busca o nome do cômodo no dicionário usando o ID do sensor
 * - Retorna o nome encontrado ou um valor padrão se não encontrar
 */
public string GetRoomName(int sensorId)
{
    if (sensorToRoom.ContainsKey(sensorId))
        return sensorToRoom[sensorId];

    return $"Sensor {sensorId}";
}

/* FUNÇÃO: GetRoomDevices()
 * ALGORITMO: Busca em dicionário com tratamento de caso padrão
 * - Busca a lista de dispositivos no dicionário usando o nome do cômodo
 * - Retorna a lista encontrada ou uma lista vazia se não encontrar
 */
public List<(string, double)> GetRoomDevices(string roomName)
{
    if (roomDevices.ContainsKey(roomName))
        return roomDevices[roomName];
```

```
            return new List<(string, double)>();
        }
    }
}
```