

Pedro Inácio Rodrigues Pontes

Prática 8: APIs REST

Belo Horizonte, Brasil

2025

1 Introdução

O objetivo deste projeto é desenvolver um programa de console em C que realize o monitoramento contínuo de temperatura através de requisições periódicas a um serviço REST local. O programa deve solicitar ao usuário a unidade de temperatura desejada (Celsius, Kelvin ou Fahrenheit) e o intervalo em segundos para cada nova leitura, realizando então requisições HTTP GET ao endpoint `http://localhost:5000/temperatura/unidade` em intervalos regulares. A cada resposta bem-sucedida, o sistema deve capturar o horário local exato, comparar o valor atual com a leitura anterior para determinar se a temperatura subiu, desceu ou permaneceu igual, e apresentar essas informações no console com formatação colorida: vermelho para indicar aumento de temperatura, azul para diminuição e cor padrão para ausência de alteração. O programa deve incluir tratamento robusto de erros para requisições HTTP falhadas ou respostas JSON inválidas, exibindo mensagens apropriadas em cor amarela, e permitir interrupção controlada pelo usuário através de comandos como Ctrl+C, finalizando com uma mensagem de encerramento adequada.

2 Desenvolvimento

2.1 RestServer

Esta classe foi fornecida pronta pelo professor e representa um servidor REST simples que simula leituras de temperatura. O servidor utiliza o framework ASP.NET Core para criar um endpoint HTTP que retorna valores de temperatura calculados matematicamente com base no horário atual, simulando variações realistas ao longo do dia.

- **Construtor RestServer** Inicializa o servidor web através do *WebApplicationBuilder* e configura o endpoint `/temperatura/{unidade}`. O endpoint aceita requisições GET onde o parâmetro *unidade* especifica a escala de temperatura desejada (celsius, kelvin ou fahrenheit).
- **Algoritmo de Simulação** A temperatura base é calculada usando uma função senoidal que varia entre 20°C e 30°C ao longo de 24 horas, simulando o ciclo natural de temperatura diária. A fórmula utilizada é: $25.0 + 5.0 * \sin((2/24) * \text{hora_atual})$. Um componente de ruído aleatório é adicionado para tornar as leituras mais realistas.
- **Conversão de Unidades** O servidor suporta três unidades de temperatura:
 - Celsius: valor base (sem conversão)
 - Kelvin: temperatura em Celsius + 273.15
 - Fahrenheit: (temperatura em Celsius \times 9/5) + 32

- **Método StartAsync** Inicia o servidor de forma assíncrona, permitindo que ele aceite requisições HTTP na porta configurada (padrão 5000).
- **Resposta JSON** Retorna um objeto JSON contendo a unidade solicitada e o valor da temperatura arredondado para duas casas decimais. Em caso de unidade inválida, retorna um erro HTTP 400 (Bad Request).

2.1.1 Demonstração

Estrutura da classe RestServer:

```
public class RestServer
{
    WebApplicationBuilder builder;
    WebApplication app;

    public RestServer(string[]? args = null)
    {
        // Configura o do servidor e endpoint
    }

    public async void StartAsync()
    {
        // Inicializa o assncrona do servidor
    }
}
```

2.2 Temperatura

Esta classe representa um modelo de dados simples (DTO - Data Transfer Object) criado especificamente para deserializar a resposta JSON retornada pelo servidor REST. Ela serve como uma estrutura de mapeamento entre o formato JSON recebido nas requisições HTTP e os objetos C utilizados pelo programa cliente.

- **Propriedade unidade** Propriedade do tipo *string* nullable que armazena a unidade de temperatura retornada pelo servidor (celsius, kelvin ou fahrenheit). O uso de tipo nullable permite tratamento adequado de respostas JSON incompletas ou malformadas.
- **Propriedade valor** Propriedade do tipo *double* nullable que contém o valor numérico da temperatura na unidade especificada. O tipo nullable oferece flexibilidade para lidar com cenários onde o valor pode estar ausente na resposta JSON.

- **Padrão de Propriedades Automáticas** Ambas as propriedades utilizam o padrão de propriedades automáticas com getters e setters públicos, facilitando a serialização/deserialização automática pelo sistema de JSON do .NET.

2.2.1 Demonstração

Estrutura da classe Temperatura:

```
public class Temperatura
{
    public string? unidade { get; set; }
    public double? valor { get; set; }
}
```

2.3 Program

Esta classe representa o ponto de entrada da aplicação e implementa toda a lógica principal do programa de monitoramento de temperatura. É responsável por coordenar a inicialização do servidor, configuração dos parâmetros de monitoramento, execução do ciclo de requisições periódicas e tratamento da interrupção controlada pelo usuário.

- **Propriedades Estáticas** A classe utiliza propriedades estáticas para armazenar o estado global da aplicação: *UnidadeDesejada* (string para a unidade de temperatura), *IntervaloDeRequisicao* (int para o intervalo em segundos), *Url* (string para o endpoint construído) e *AnteriorTemp* (double nullable para comparação entre leituras).
- **Método Main** Ponto de entrada assíncrono que coordena toda a execução do programa. Inicializa o servidor REST local, configura o tratamento de interrupção via Ctrl+C usando *CancellationTokenSource*, realiza a validação das entradas do usuário (unidade e intervalo), constrói a URL do endpoint e executa o loop principal de monitoramento até que seja cancelado.
- **Método ExibirTemperatura** Método assíncrono responsável por executar uma única iteração do ciclo de monitoramento. Realiza a requisição HTTP GET ao servidor, deserializa a resposta JSON usando *JsonSerializer*, compara o valor atual com a temperatura anterior para determinar a tendência (subida, descida ou estabilidade), atualiza o estado da temperatura anterior e exibe o resultado formatado no console com timestamp e símbolos indicativos.
- **Tratamento de Cancelamento** Implementa um sistema robusto de cancelamento baseado em *CancellationToken*, permitindo interrupção controlada tanto pelo usuário (Ctrl+C) quanto por exceções do sistema, garantindo que o programa finalize adequadamente em todas as situações.

- **Validação de Entrada** Inclui validação básica dos parâmetros fornecidos pelo usuário, verificando se a unidade de temperatura está entre as opções válidas (celsius, fahrenheit, kelvin) e se o intervalo de requisição é um valor não-negativo.

2.3.1 Demonstração

Estrutura da classe Program com trechos importantes:

```
public class Program
{
    static string? UnidadeDesejada { get; set; }
    static int IntervaloDeRequisicao { get; set; }
    static string? Url { get; set; }
    static double? AnteriorTemp { get; set; }

    public static async Task Main(string[] args)
    {
        RestServer server = new();
        server.StartAsync();

        using var cts = new CancellationTokenSource();
        Console.CancelKeyPress += (sender, e) =>
        {
            e.Cancel = true;
            cts.Cancel();
            Console.WriteLine("\nParando o
                               monitoramento...");
        };

        [...]
        if (UnidadeDesejada != "celsius" && UnidadeDesejada
            != "fahrenheit" && UnidadeDesejada != "kelvin")
        {
            Console.WriteLine("Selecione uma unidade
                               válida");
            return;
        }

        [...]
        while (!cts.Token.IsCancellationRequested)
        {
```

```
        await ExibirTemperatura(httpClient, cts.Token);
    }
}

static async Task ExibirTemperatura(HttpClient
    httpClient, CancellationToken cancellationToken)
{
    HttpResponseMessage response = await
        httpClient.GetAsync(Url, cancellationToken);
    string jsonString = await
        response.Content.ReadAsStringAsync();
    var temperatura =
        JsonSerializer.Deserialize<Temperatura>(jsonString);

    [...]
    if (temperatura.valor > AnteriorTemp)
    {
        symbol = "    ";
    }
    else if (temperatura.valor < AnteriorTemp)
    {
        symbol = "    ";
    }

    Console.WriteLine($"[{DateTime.Now:HH:mm:ss}] Temperatura:
        {temperatura.valor}  {temperatura.unidade}
        {symbol}");
    await Task.Delay(IntervaloDeRequisicao * 1000,
        cancellationToken);
}
}
```

3 Resultados

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5086
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /home/pedro/School/3ano/TAP00/ServidorTemp/ServidorTemp
Digite a unidade de medida desejada
Celsius
Digite o intervalo entre as requisições
2
[20:21:22]Temperatura: 21.83°celsius
[20:21:24]Temperatura: 21.19°celsius ↓
[20:21:26]Temperatura: 21°celsius ↓
[20:21:28]Temperatura: 21.22°celsius ↑
[20:21:30]Temperatura: 21.62°celsius ↑
[20:21:32]Temperatura: 21.29°celsius ↓
[20:21:34]Temperatura: 21.48°celsius ↑
[20:21:36]Temperatura: 21.89°celsius ↑
[20:21:38]Temperatura: 21.11°celsius ↓
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...

Parando o monitoramento...
Monitoramento finalizado.
```

Figura 1 – Execução do programa com unidade Celsius

```
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5086
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /home/pedro/School/3ano/TAP00/ServidorTemp/ServidorTemp
Digite a unidade de medida desejada
fahrenheit
Digite o intervalo entre as requisições
1
[20:25:59]Temperatura: 70.4°fahrenheit
[20:26:01]Temperatura: 69.78°fahrenheit ↓
[20:26:02]Temperatura: 71.14°fahrenheit ↑
[20:26:03]Temperatura: 70.53°fahrenheit ↓
[20:26:04]Temperatura: 71.3°fahrenheit ↑
[20:26:05]Temperatura: 71.18°fahrenheit ↓
[20:26:06]Temperatura: 70.55°fahrenheit ↓
[20:26:07]Temperatura: 71.14°fahrenheit ↑
[20:26:08]Temperatura: 71.36°fahrenheit ↑
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...

Parando o monitoramento...
Monitoramento finalizado.
```

Figura 2 – Execução do programa com unidade Fahrenheit

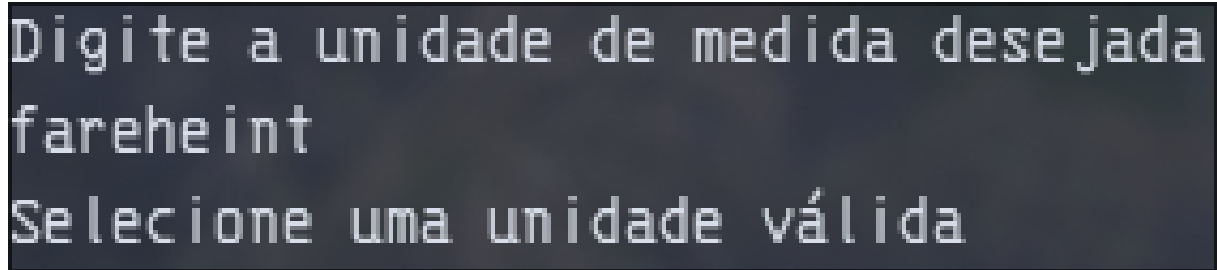

```
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5086
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /home/pedro/School/3ano/TAP00/ServidorTemp/ServidorTemp
Digite a unidade de medida desejada
KelvIn
Digite o intervalo entre as requisições
5
[20:26:32]Temperatura: 294.98°kelvin
[20:26:37]Temperatura: 294.68°kelvin ↓
[20:26:42]Temperatura: 294.28°kelvin ↓
[20:26:47]Temperatura: 295.07°kelvin ↑
[20:26:52]Temperatura: 294.82°kelvin ↓
[20:26:57]Temperatura: 294.23°kelvin ↓
[20:27:02]Temperatura: 294.63°kelvin ↑
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...

Parando o monitoramento...
Monitoramento finalizado.
```

Figura 3 – Execução do programa com unidade Kelvin

```
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5086
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /home/pedro/School/3ano/TAP00/ServidorTemp/ServidorTemp
Digite a unidade de medida desejada
celsius
Digite o intervalo entre as requisições
-1
Selecione um intervalo válido
```

Figura 4 – Tratamento de erro para intervalo inválido



```
Digite a unidade de medida desejada
fareheint
Selecione uma unidade válida
```

Figura 5 – Tratamento de erro para unidade inválida

4 Conclusão

Todos os objetivos do projeto foram alcançados com sucesso. O desenvolvimento do programa de monitoramento de temperatura proporcionou aprendizado prático sobre APIs REST, tanto na perspectiva de criação quanto de consumo.

Na criação de APIs REST, foi possível compreender como estruturar endpoints HTTP utilizando ASP.NET Core, definindo rotas parametrizadas e retornando dados em formato JSON. A implementação do servidor demonstrou a simplicidade de mapear URLs para funções que processam requisições e geram respostas padronizadas.

No consumo de APIs REST, o projeto evidenciou o uso do HttpClient para realizar requisições GET, o tratamento de respostas HTTP e a deserialização de JSON para objetos C#. A experiência mostrou a importância do gerenciamento adequado de recursos HTTP e do tratamento de erros de comunicação.

O trabalho consolidou conhecimentos fundamentais sobre comunicação entre aplicações através de protocolos HTTP, demonstrando como sistemas distribuídos podem trocar informações de forma estruturada e eficiente utilizando o padrão REST.