Módulo 8: Backend com C# e Python

Dia 05: Assincronismo e Consumo de APIs

João Quentino

29 de dezembro de 2025

Roteiro do Dia 05

1. Objetivo e Conceitos de Assincronismo
2. Async/Await em Python
3. Async/Await em C#
4. Consumindo APIs: CEP e Clima
5. Consultas em Paralelo
6. Prática: Buscador de CEP e Clima
7. Encerramento

Objetivo da Aula

Ao final desta aula, você será capaz de:

- Entender o problema de bloquear a aplicação em chamadas externas.
- Utilizar `async` e `await` em Python e C#.
- Consumir APIs HTTP (CEP e Clima) em ambas as linguagens.
- Fazer chamadas em paralelo para ganhar tempo.
- Tratar erros básicos em requisições HTTP.

Por que Assíncrono?

- A aplicação não fica parada esperando o servidor responder.
- Melhor uso de recursos: enquanto espera I/O, outra tarefa pode rodar.
- Experiência melhor para o usuário (respostas mais rápidas).

Primeiro contato com async / await (Python)

Função assíncrona simples

```python
import asyncio

async def diga_ola():
    await asyncio.sleep(1)
    print("Ola depois de 1 segundo")

asyncio.run(diga_ola())
```

Primeiro contato com async / await (C#)

Método assíncrono simples

```
using System;
using System.Threading.Tasks;

static async Task DigaOla()
{
    await Task.Delay(1000);
    Console.WriteLine("Ola depois de 1 segundo");
}

await DigaOla();
```

Consumindo API de CEP em Python

Exemplo com httpx

```python
import httpx
import asyncio

async def buscar_cep(cep: str):
    url = f"https://viacep.com.br/ws/{cep}/json/"
    async with httpx.AsyncClient() as client:
        resposta = await client.get(url)
        dados = resposta.json()
        print("Cidade:", dados.get("localidade"))

asyncio.run(buscar_cep("49000000"))
```

Consumindo API de CEP em C#

Exemplo com HttpClient

```
using System;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;

static async Task BuscarCepAsync(string cep)
{
    using var client = new HttpClient();
    var url = $"https://viacep.com.br/ws/{cep}/json/";
    var json = await client.GetStringAsync(url);

    using var doc = JsonDocument.Parse(json);
    var raiz = doc.RootElement;
    Console.WriteLine("Cidade: " + raiz.GetProperty("
        localidade").GetString());
}
```

Consultas paralelas em Python

CEP e Clima ao mesmo tempo

```
async def buscar_tudo():
    tarefa_cep = buscar_cep("49000000")
    tarefa_clima = buscar_clima("Aracaju")

    await asyncio.gather(tarefa_cep, tarefa_clima)
```

Consultas paralelas em C#

CEP e Clima ao mesmo tempo

```
async Task BuscarTudoAsync()
{
    var tarefaCep = BuscarCepAsync("49000000");
    var tarefaClima = BuscarClimaAsync("Aracaju");

    await Task.WhenAll(tarefaCep, tarefaClima);
}
```

Desafio Guiado

Objetivo

Construir um pequeno programa de console que:

- Recebe um CEP e uma cidade.
- Busca o endereço pelo CEP em uma API pública.
- Busca as informações de clima da cidade em outra API.
- Exibe as duas respostas no terminal.

Passos da Prática

- Definir funções/métodos para cada chamada de API.
- Adicionar `async` / `await` e testar de forma sequencial.
- Adaptar para rodar CEP e Clima em paralelo.
- Adicionar tratamento básico de erro (status code, exceções).

Resumo do Dia 05

- Conceito de assincronismo e porque ele é importante.
- Uso básico de `async` / `await` em Python e C#.
- Consumo de APIs HTTP para CEP e Clima.
- Execução de múltiplas chamadas em paralelo.

Próximos Passos

Vamos chamar APIs!

- Explorar mais endpoints de CEP e Clima.
- Encapsular chamadas em serviços reutilizáveis.
- Pensar como essas chamadas entram na nossa API backend.