

1. A Entrega Padrão:

- Crie uma requisição do tipo GET.
- Tente buscar o CEP da nossa faculdade (ou um famoso, como o
- da Sé em SP: 01001000).

Pergunta: Qual foi o Status Code retornado? O JSON veio completo?

O status code retornado foi 200, JSON veio completo

```
{  
  "cep": "01001-000",  
  "logradouro": "Praça da Sé",  
  "complemento": "lado ímpar",  
  "unidade": "",  
  "bairro": "Sé",  
  "localidade": "São Paulo",  
  "uf": "SP",  
  "estado": "São Paulo",  
  "regiao": "Sudeste",  
  "ibge": "3550308",  
  "gia": "1004",  
  "ddd": "11",  
  "siafi": "7107"  
}
```

2. O Caso do CEP Inexistente:

- Tente fazer um GET para um CEP que não existe (ex: 99999999).

Desafio: Observe o JSON retornado. O ViaCEP retorna um erro 404 ou ele retorna um JSON com um campo "erro": true? Por que você acha que eles escolheram essa estratégia?

```
{  
  "erro": true  
}
```

status code 200, O ViaCEP retorna "erro": true ao invés de 404 porque:

A requisição ela funcionou corretamente

Só que o CEP não existe

Isso facilita o tratamento no sistema.

Mudando o Formato (O Camaleão): O ViaCEP é incrível porque ele aceita outros formatos além do JSON. Tente mudar o final da URL de /json/ para /xml/.

O que mudou na visualização dos dados? Qual formato parece mais fácil de ler no C#?

Veio em XML ao invés de JSON, o formato mais fácil de ler é o **JSON** Porque o C# trabalha muito bem com JSON usando o **System.Text.Json**

```
<?xml version="1.0" encoding="UTF-8"?>

<xmlcep>

<cep>01001-000</cep>

<logradouro>Praça da Sé</logradouro>

<complemento>lado ímpar</complemento>

<unidade></unidade>

<bairro>Sé</bairro>

<localidade>São Paulo</localidade>

<uf>SP</uf>

<estado>São Paulo</estado>

<regiao>Sudeste</regiao>

<ibge>3550308</ibge>

<gia>1004</gia>

<ddd>11</ddd>

<siafi>7107</siafi>

</xmlcep>
```

Passo 3: O Relatório do Engenheiro

Após realizar os testes, responda:

1. Verbo HTTP: Qual método você usou? Por que não usamos POST para consultar um CEP?

Usei o método GET.

Não usamos POST porque

GET ele consulta dados
POST ele envia dados

Consultar CEP não envia dados novos.

2. Análise de Dados: Copie o JSON do seu CEP e identifique:

- Qual é a Chave (Key) que guarda o nome da cidade?

A chave KEY é "**localidade**"

- Qual é a Chave que guarda o nome da rua?

A chave KEY é "**logradouro**"

3. Integração C#: Se você fosse criar uma classe em C# para receber esses dados, como ficaria a sua propriedade para a chave localidade?

Use o que aprendemos sobre [JsonPropertyName].

```
C# lista_01.cs ×
C# lista_01.cs
1  using System.Text.Json.Serialization;
2
3  Windsurf: Refactor | Explain
4  public class Endereco
5  {
6      [JsonPropertyName("cep")]
7      public string Cep { get; set; }
8
9      [JsonPropertyName("logradouro")]
10     public string Logradouro { get; set; }
11
12     [JsonPropertyName("complemento")]
13     public string Complemento { get; set; }
14
15     [JsonPropertyName("unidade")]
16     public string Unidade { get; set; }
17
18     [JsonPropertyName("bairro")]
19     public string Bairro { get; set; }
20
21     [JsonPropertyName("localidade")]
22     public string Localidade { get; set; }
23
24     [JsonPropertyName("uf")]
25     public string Uf { get; set; }
26
27     [JsonPropertyName("estado")]
28     public string Estado { get; set; }
29
30     [JsonPropertyName("regiao")]
31     public string Regiao { get; set; }
32
33     [JsonPropertyName("ibge")]
34     public string Ibge { get; set; }
35
36     [JsonPropertyName("gia")]
37     public string Gia { get; set; }
38
39     [JsonPropertyName("ddd")]
40     public string Ddd { get; set; }
41
42     [JsonPropertyName("siafi")]
43     public string Siafi { get; set; }
}
```

