



EducaCiência FastCode

Fala Galera,

- Artigo: 48/2021 Data: Fevereiro/2021
- Público-alvo: Desenvolvedores – Iniciantes
- Tecnologia: C# .Net Core 3.1
- Tema: Artigo 48 - BuscaCepCharp_Refit
- Link: <https://github.com/perucello/DevFP>

Neste artigo, abordaremos a criação de uma aplicação em C# que consumirá uma API dos correios gratuita – busca de CEP utilizando uma biblioteca REST => Refit.

Neste momento, abordaremos uma maneira simples de criarmos uma aplicação que fará uma busca por CEP onde interagirá via REST com a API Viacep – API Gratuita

Viacep – <https://viacep.com.br/>

Refit – <https://www.nuget.org/packages/refit/5.2.4> - Biblioteca REST de tipo seguro automático para Xamarin e .NET

Lembrando que os fins são didáticos !

VIACEP
Consulte CEPs de todo o Brasil

Procurando um [webservice](#) gratuito e de alto desempenho para consultar Códigos de Endereçamento Postal (CEP) do Brasil? Utilize o serviço, melhore a qualidade de suas aplicações web e colabore para manter esta base de dados atualizada.

Acessando o webservice de CEP

Para acessar o webservice, um CEP no formato de (8) dígitos deve ser fornecido, por exemplo: "01001000".
Após o CEP, deve ser fornecido o tipo de retorno desejado, que deve ser "json", "xml", "piped" ou "query".

Exemplo de pesquisa por CEP:
viacep.com.br/ws/01001000/json/

Validação do CEP

Quando consultado um CEP de formato inválido, por exemplo: "950100100" (9 dígitos), "95010A10" (alfanumérico), "95010 10" (espaço), o código de retorno da consulta será um 400 (Bad Request). Antes de acessar o webservice, valide o formato do CEP e certifique-se que o mesmo possua (8) dígitos. Exemplo de como validar o formato do CEP em javascript está disponível nos exemplos abaixo.

Quando consultado um CEP de formato válido, porém inexistente, por exemplo: "99999999", o retorno conterá um valor de "erro" igual a "true". Isso significa que o CEP consultado não foi encontrado na base de dados. Veja como manipular este "erro" em javascript nos exemplos abaixo.





Antes de mais nada, utilizaremos da IDE Microsoft Visual Studio Professional 2019

Visual Studio

[Status da licença](#)

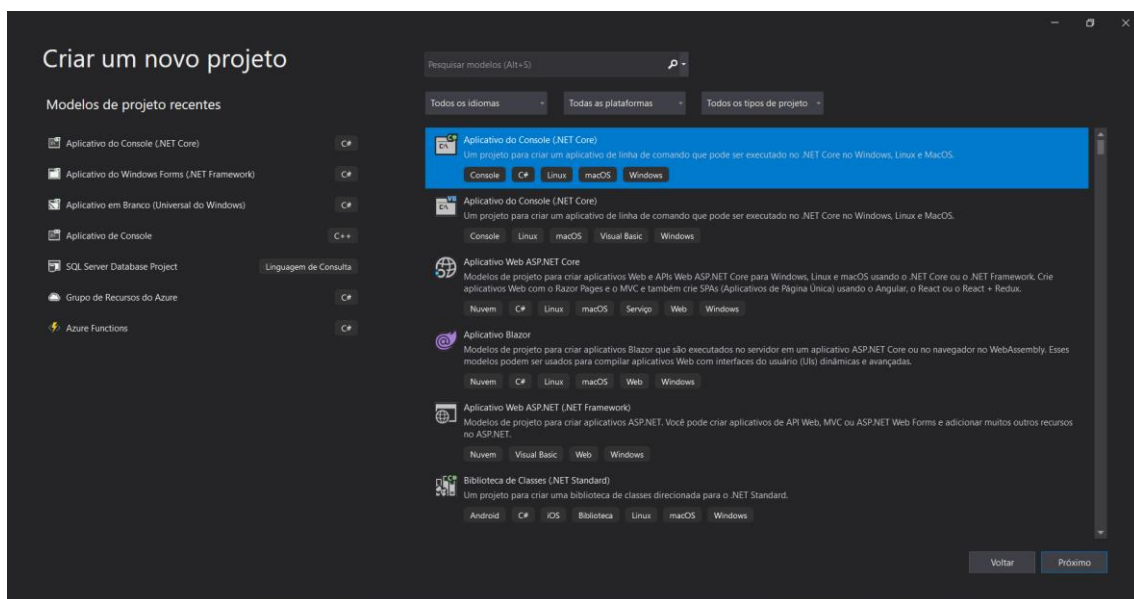
[Termos da licença](#)

Microsoft Visual Studio Professional 2019
Versão 16.6.4
© 2020 Microsoft Corporation.
Todos os direitos reservados.

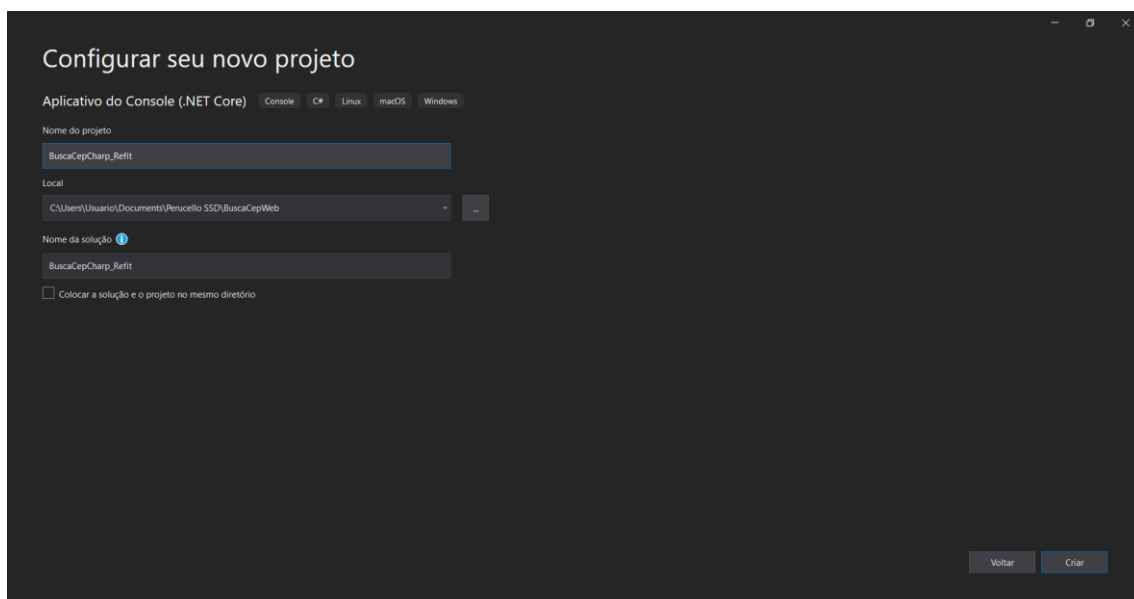
Microsoft .NET Framework
Versão 4.8.04084
© 2020 Microsoft Corporation.
Todos os direitos reservados.

Vamos nessa !

Vamos criar nosso Projeto, para isso vamos seguir os seguintes passos:



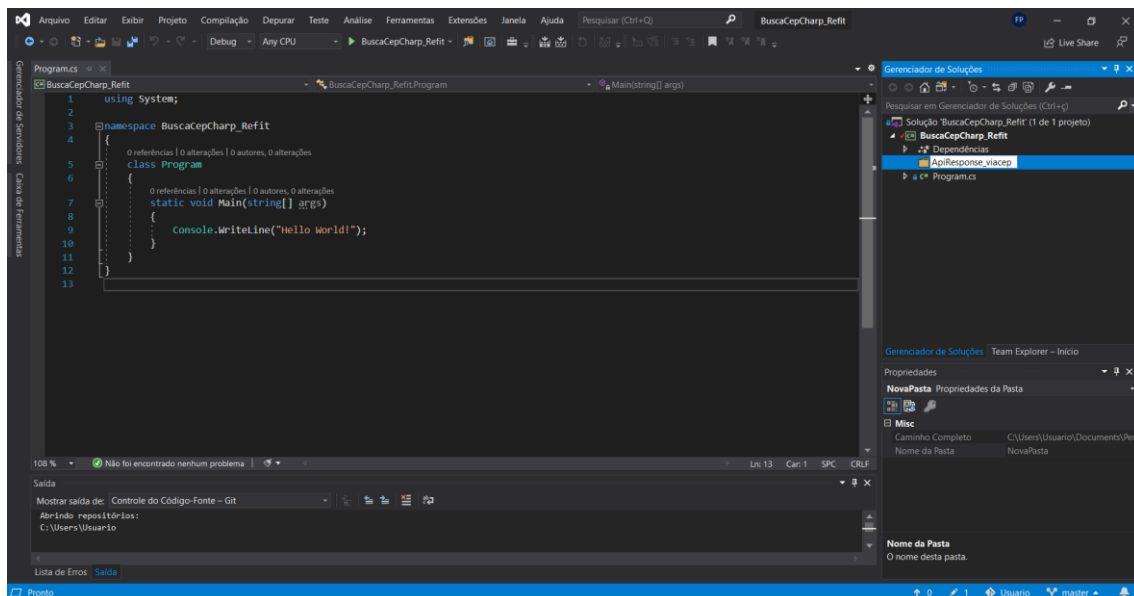
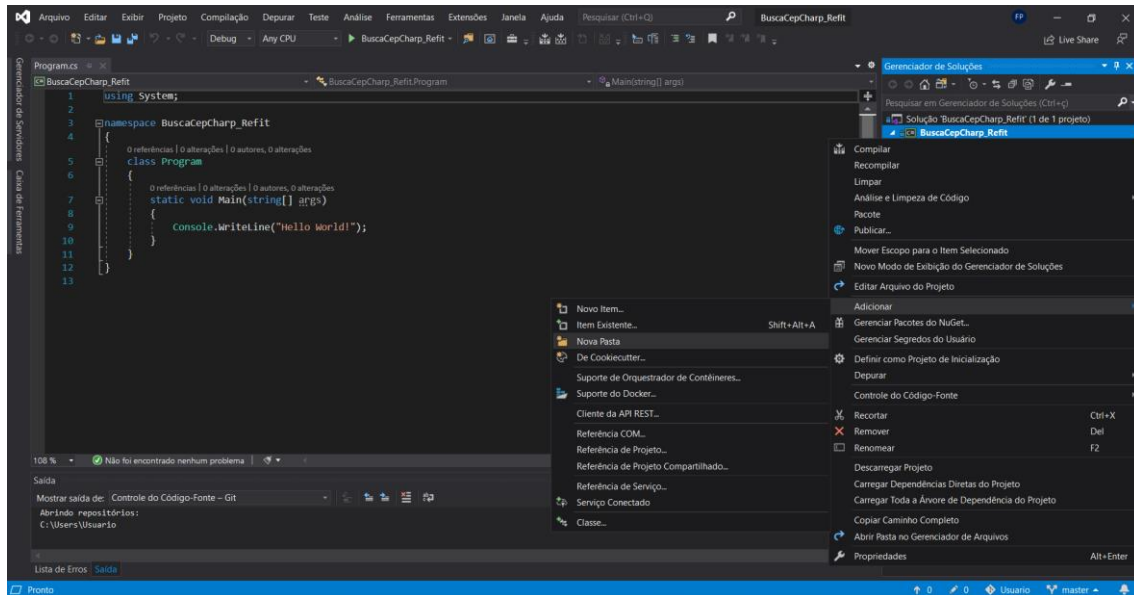
Vamos nomear nosso Projeto como demonstro abaixo:





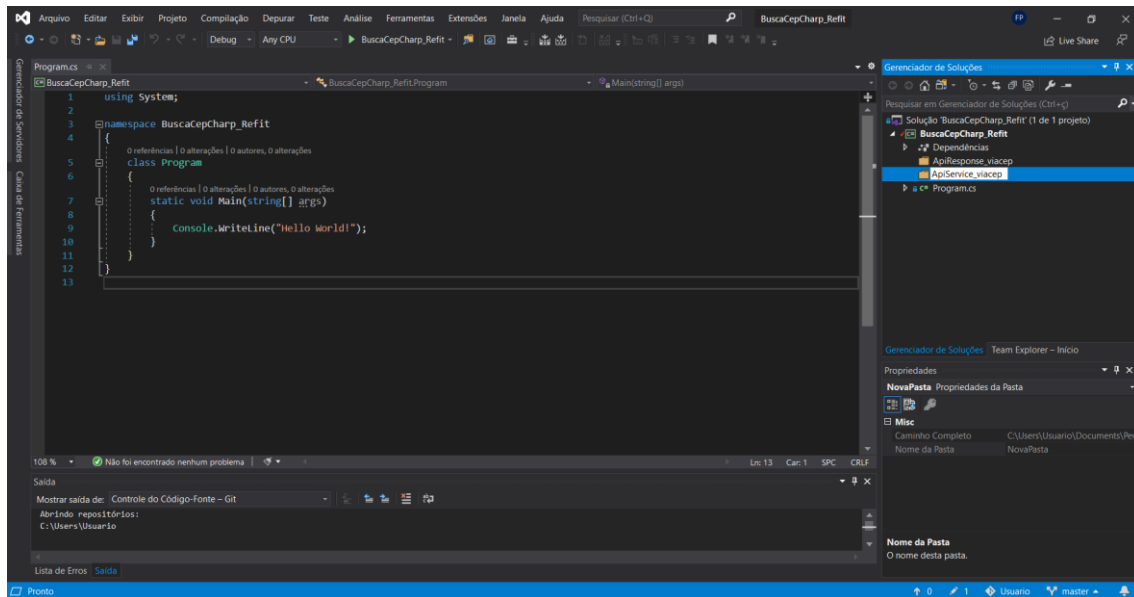
Pronto, temos já nosso projeto criado, agora iremos criar toda nossa estrutura das Pastas e criaremos também nossas classes onde implementaremos nosso código !

Sendo assim, criaremos nosso Pacote “ApiResponse_viacep”



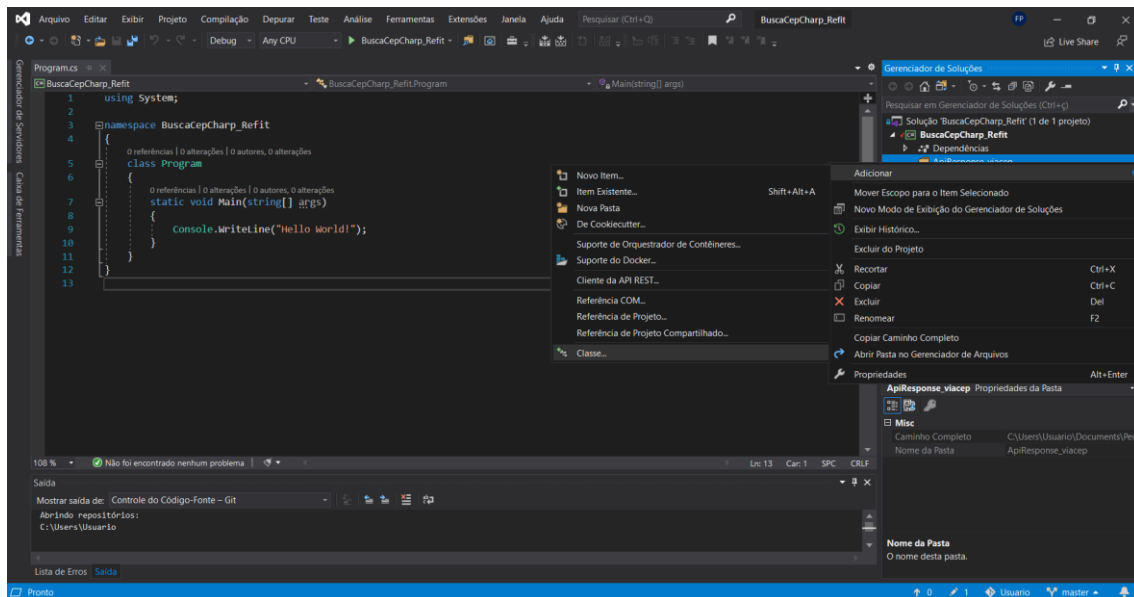


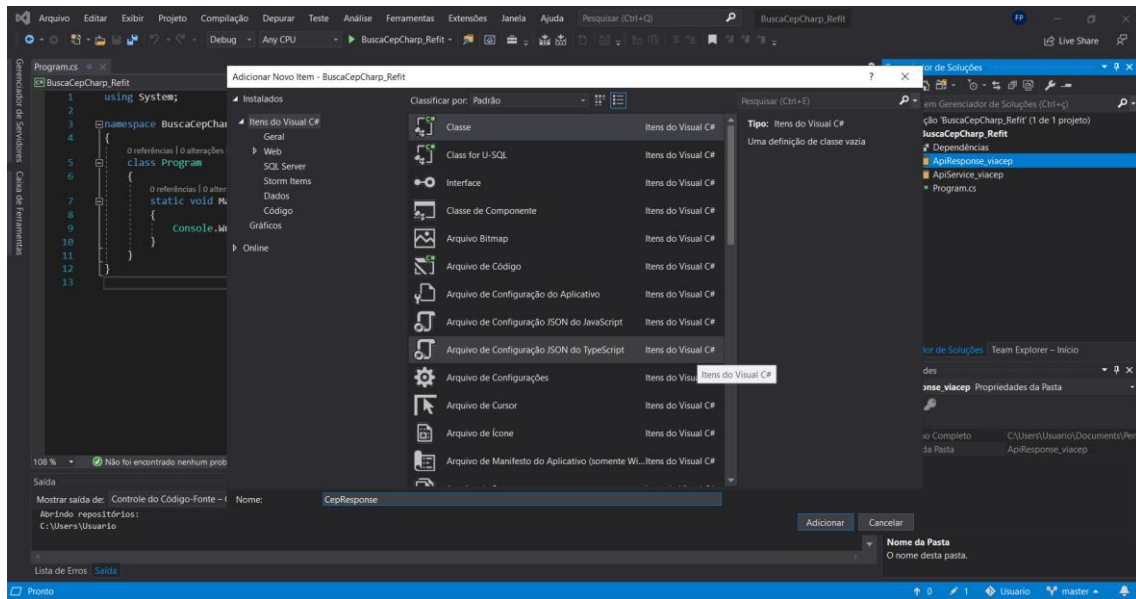
Agora criaremos nosso pacote “ApiService_viacep”



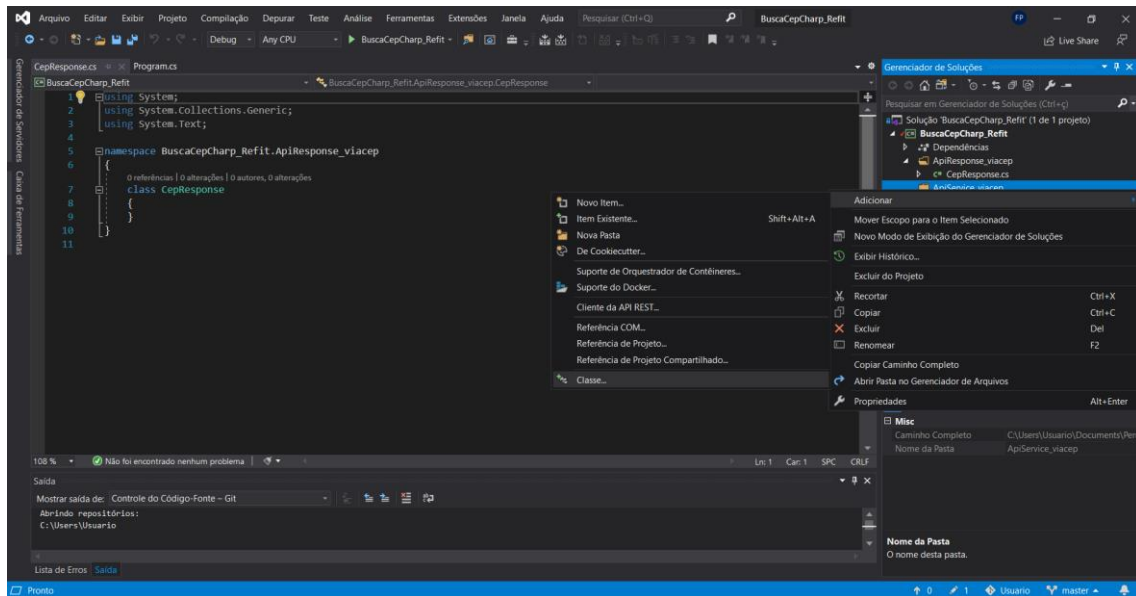
Com nossos Pacotes criados, vamos criar nossas classes

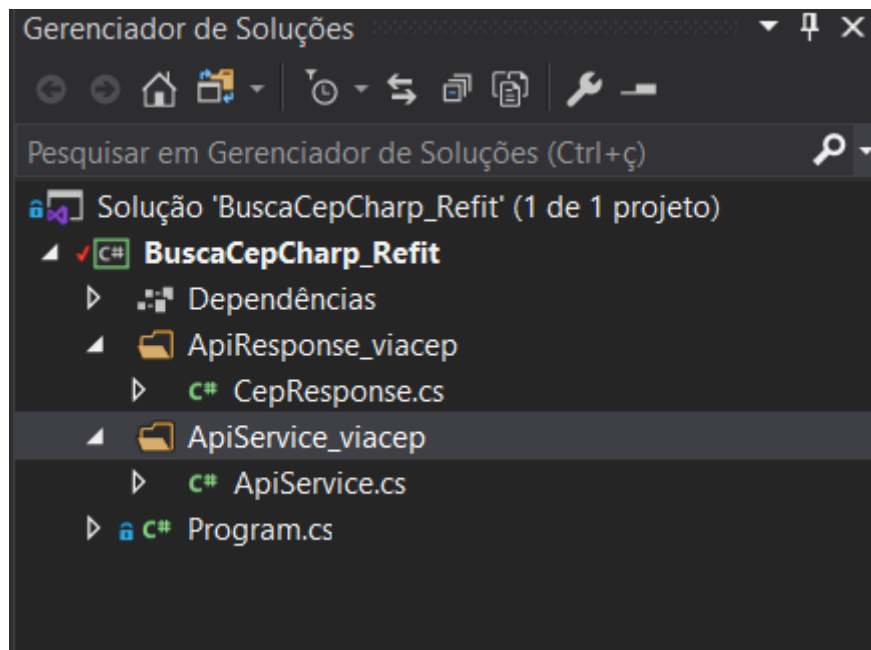
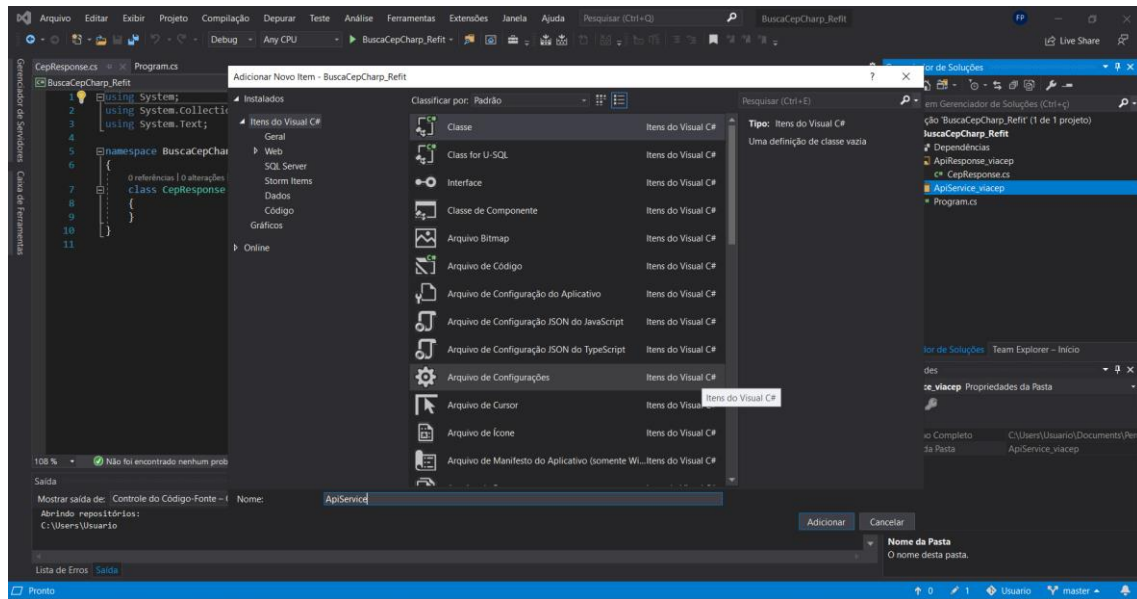
⇒ CepResponse.cs





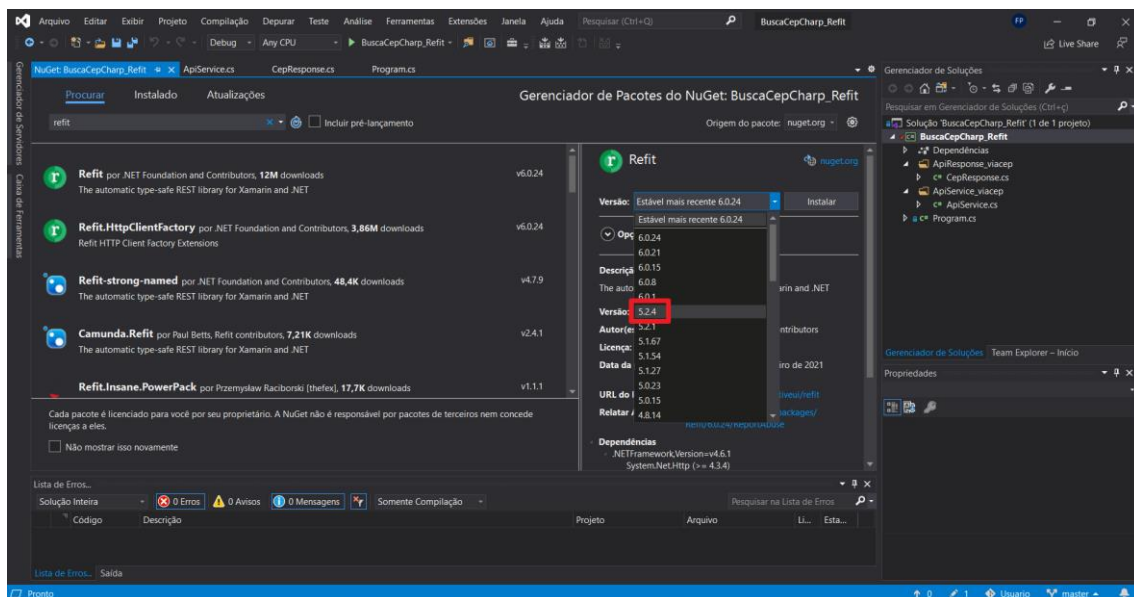
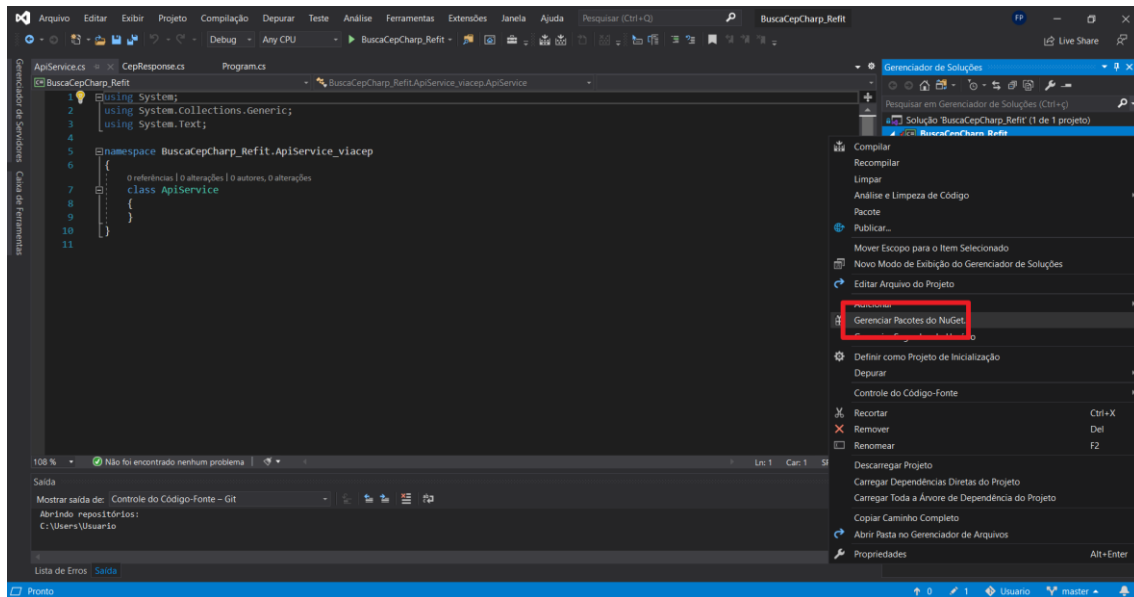
⇒ ApiService.cs

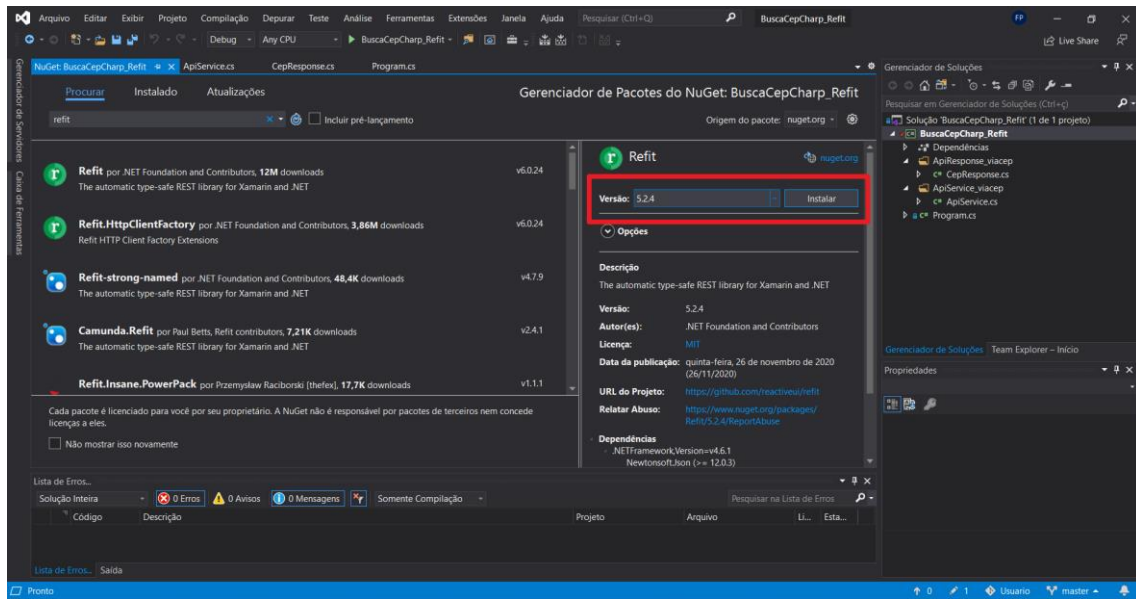






Como mencionamos no tópico, utilizaremos da biblioteca Refit, então vamos instalar seguindo os seguintes procedimentos:





Gerenciador de Pacotes do NuGet: BuscaCepCharp_Refit

Procurar: refit

Instalar

Atualizações

Gerenciador de Pacotes do NuGet: BuscaCepCharp_Refit

Origem do pacote: nuget.org

Versão: 5.2.4

Instalar

Opções

Descrição

The automatic type-safe REST library for Xamarin and .NET

Versão: 5.2.4

Autor(es): .NET Foundation and Contributors

Licença: MIT

Data da publicação: quinta-feira, 26 de novembro de 2020 (26/11/2020)

URL do Projeto: <https://github.com/reactiveui/refit>

Relatar Abuso: <https://www.nuget.org/packages/Refit/5.2.4/ReportAbuse>

Dependências

- NETFramework.Version=v4.6.1
- Newtonsoft.Json (>= 12.0.3)

Lista de Erros...

Solução Inteira

0 Erros

0 Avisos

0 Mensagens

Somente Compilação

Projeto

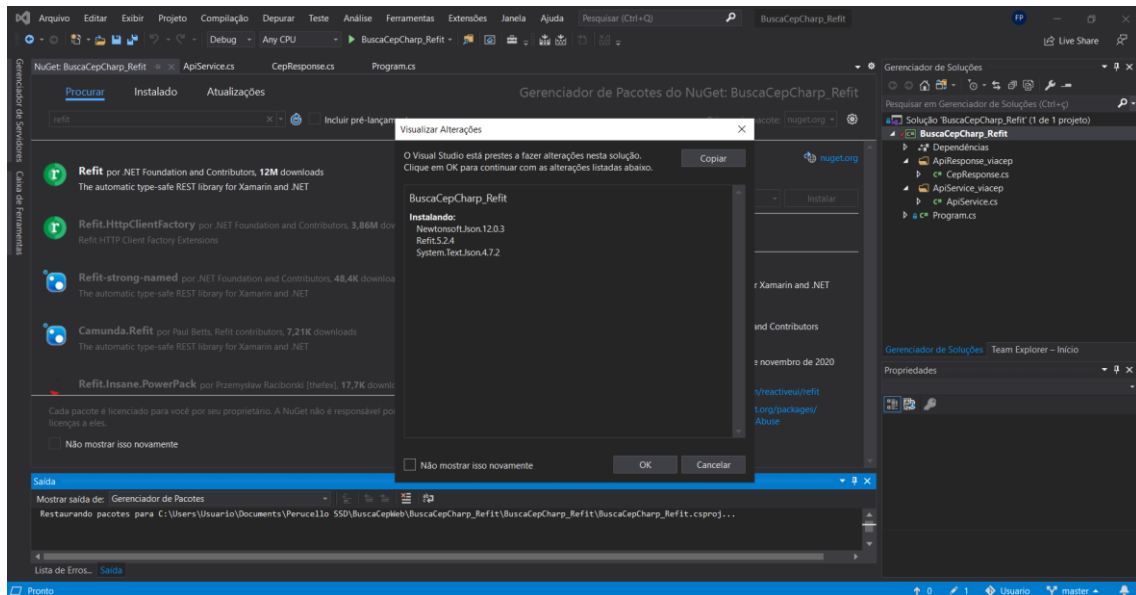
Arquivo

Esta...

Lista de Erros...

Saída

Pronto



Gerenciador de Pacotes do NuGet: BuscaCepCharp_Refit

Procurar: refit

Instalar

Atualizações

Gerenciador de Pacotes do NuGet: BuscaCepCharp_Refit

Origem do pacote: nuget.org

Versão: 5.2.4

Instalar

Opções

Descrição

The automatic type-safe REST library for Xamarin and .NET

Versão: 5.2.4

Autor(es): .NET Foundation and Contributors

Licença: MIT

Data da publicação: quinta-feira, 26 de novembro de 2020 (26/11/2020)

URL do Projeto: <https://github.com/reactiveui/refit>

Relatar Abuso: <https://www.nuget.org/packages/Refit/5.2.4/ReportAbuse>

Dependências

- NETFramework.Version=v4.6.1
- Newtonsoft.Json (>= 12.0.3)

Lista de Erros...

Solução Inteira

0 Erros

0 Avisos

0 Mensagens

Somente Compilação

Projeto

Arquivo

Esta...

Lista de Erros...

Saída

Pronto

Visualizar Alterações

O Visual Studio está prestes a fazer alterações nesta solução. Clique em OK para continuar com as alterações listadas abaixo.

Copiar

Instalando:

- Newtonsoft.Json.12.0.3
- Refit.5.2.4
- System.Text.Json.4.7.2

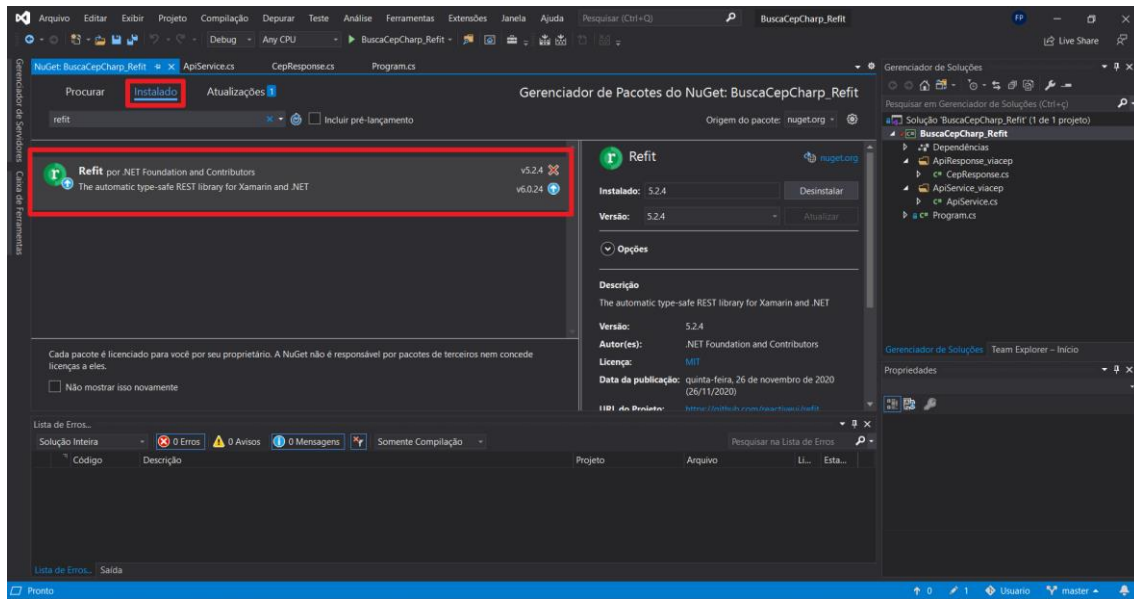
OK

Cancelar

Saída

Mostrar saída de: Gerenciador de Pacotes

Restaurando pacotes para C:\Users\Usuario\Documents\Peruella\SSD\BuscaCepCharp\BuscaCepCharp_Refit\BuscaCepCharp_Refit\BuscaCepCharp_Refit.csproj...



Como proposta, utilizaremos da API Gratuita Viacep (link na pagina 1) e sendo assim, acesse o link para mapearmos quais objetos esta API nos dispõe:

Formatos de Retorno

Veja exemplos de acesso ao webservice e os diferentes tipos de retorno:

```
JSON
URL: viacep.com.br/ws/01001000/json/
UNICODE: viacep.com.br/ws/01001000/json/unicode/

{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "3508308",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7107"
}
```

Nota-se que temos alguns objetos que a disponibiliza para busca, mas não utilizaremos de todos, iremos manipular apenas Logradouro, Bairro, Cidade e Estado.

Sendo assim, na nossa classe CepResponse.cs insira os seguintes códigos:

```
using System;
using System.Collections.Generic;
using System.Text;
using Newtonsoft.Json;

namespace CepConsoleApi
{
    class CepResponse
    {
        // link do via cep de consulta => https://viacep.com.br/

        [JsonProperty("cep")]
        public string Cep { get; set; } //prop tab tab
    }
}
```

```

[JsonProperty("logradouro")]
public string Logradouro { get; set; }

[JsonProperty("complemento")]
public string Complemento { get; set; }

[JsonProperty("bairro")]
public string Bairro { get; set; }

[JsonProperty("localidade")]
public string Localidade { get; set; }

[JsonProperty("uf")]
public string Uf { get; set; }

[JsonProperty("unidade")]
public string Unidade { get; set; }

[JsonProperty("ibge")]
public string Ibge { get; set; }

[JsonProperty("gia")]
public string Gia { get; set; }

}
}

```

Agora , temos criado a classe ApiService.cs, mas a transformaremos em uma “interface” , para isso, acesse a classe e insira os seguintes códigos:

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using Refit;

namespace CepConsoleApi
{
    interface ApiService
    {
        [Get("/ws/{cep}/json")]
        Task<CepResponse> GetAddressAsync(string cep);
    }
}

```

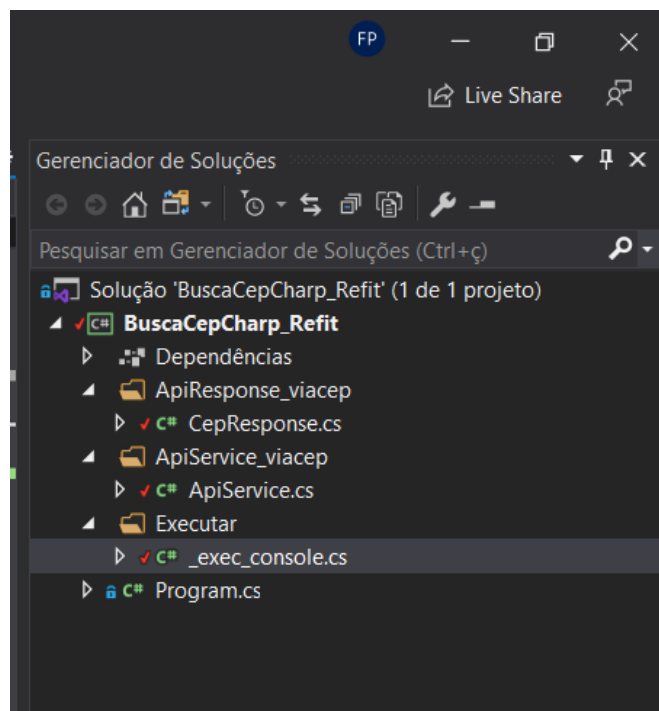
Pronto, preparamos nossa classe que contem nossa regra de negocio com o webservice Viacep e tambem temos nossa interface criada.

Agora, precisamos preparar nosso metodo main, ou seja , o metodo que fará nossa pesquisa ! Mas , temos a ideia neste artigo de abordarmos duas maneiras, uma busca por Objeto e uma Busca via console !

Sendo assim, iremos começar mapeando por objeto, e para simplificar nossa arquitetura, iremos seguir implementar um Pacote Executar e uma classe que terá nosso metodo para que no nosso Main seja invodada.

Portanto, agora que explicamos, vamos seguir os seguintes passos:

- ⇒ Criar pacote “Executar”
- ⇒ Criar classe “_exec_console.cs”



Agora, vamos implementar nosso método na classe “_exec_console.cs”, para isso insira o seguinte código:

```
using CepConsoleApi;
using Refit;
using System;
using System.Collections.Generic;
using System.Text;

namespace BuscaCepWeb.Executar
{
    public class _exec_console
    {
        public _exec_console() { }
    }
}
```

```
public async System.Threading.Tasks.Task BuscarCepPorConsoleAsync()
{
    try
    {
        string cep = "";
        var cepClient = RestService.For<ApiService>("https://viacep.com.br/");
        Console.WriteLine("Informe o cep: ");

        string cepBuscar = Console.ReadLine().ToString();
        Console.WriteLine("Consultando " + cepBuscar);

        var address = await cepClient.GetAddressAsync(cepBuscar);

        Console.WriteLine(
            $"\\nLogradouro: {address.Logradouro }, " +
            $"\\nBairro: {address.Bairro}, " +
            $"\\nCidade: {address.Localidade}, " +
            $"\\nEstado: {address.Uf}"
        );

        Console.ReadKey();

    }
    catch (Exception e)
    {
        Console.WriteLine("Falha de Exception " + e.Message);
    }
}
}
```

Pronto, agora iremos ao nosso main e chamar nosso método criado, para isso , acesse a classe **Program.cs** e insira o seguinte código:

```
using BuscaCepWeb.Executar;
using System;

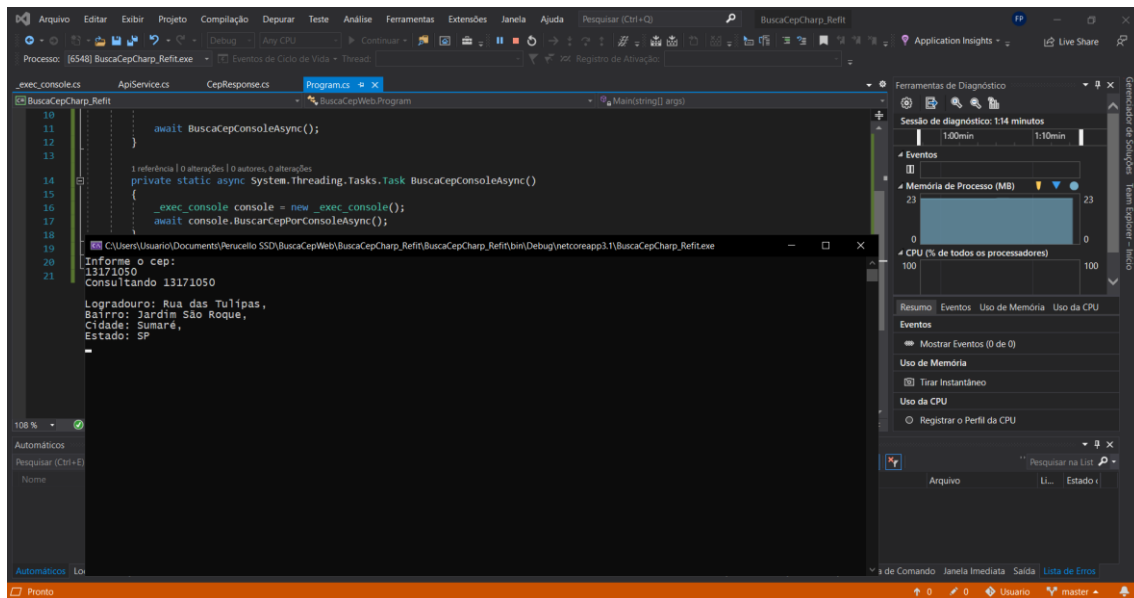
namespace BuscaCepWeb
{
    class Program
    {

        static async System.Threading.Tasks.Task Main(string[] args)
```

```
{
    await BuscaCepConsoleAsync();
}

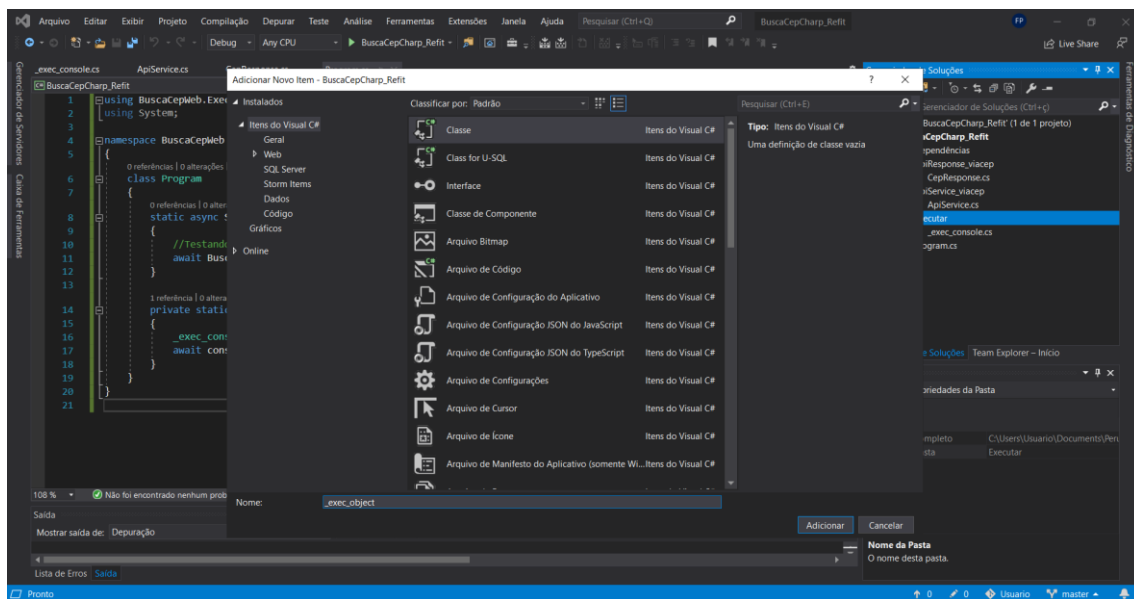
private static async System.Threading.Tasks.Task BuscaCepConsoleAsync()
{
    _exec_console console = new _exec_console();
    await console.BuscarCepPorConsoleAsync();
}
}
```

Agora vamos executar:
CEP de busca , digitamos como exemplo 13171050



Nota-se que funcionou perfeitamente a pesquisa onde passamos o CEP via console.

Agora vamos criar um método para que o CEP seja um objeto mocado, sendo assim iremos criar uma classe no nosso pacote Executar, sendo assim, faça como no exemplo:



Após criar a classe “_exec_object.cs” insira o seguinte código:

```
using CepConsoleApi;
using Refit;
using System;
using System.Collections.Generic;
using System.Text;

namespace BuscaCepWeb.Executar
{
    public class _exec_object
    {

        public _exec_object() {}

        public async System.Threading.Tasks.Task BuscarCepPorObjetoAsync()
        {
            try
            {
                string cepBuscar = "13171050";
                var cepClient = RestService.For<ApiService>("https://viacep.com.br/");
                Console.WriteLine("Consultando " + cepBuscar);

                var address = await cepClient.GetAddressAsync(cepBuscar);

                Console.WriteLine(
                    $"\\nLogradouro: {address.Logradouro}, " +
                    $"\\nBairro: {address.Bairro}, " +
                    $"\\nCidade: {address.Localidade}, " +
                    $"\\nEstado: {address.Uf}"
                );

                Console.ReadKey();
            }
        }
    }
}
```



```

    }
    catch (Exception e)
    {

        Console.WriteLine("Falha de Exception " + e.Message);
    }
}
}
}

```

Feito isso, precisamos agora , ir no nosso Program.cs e chamar este método que acabamos de criar, então vamos nessa:

```

using BuscaCepWeb.Executar;
using System;

namespace BuscaCepWeb
{
    class Program
    {
        static async System.Threading.Tasks.Task Main(string[] args)
        {
            /*
            //Testando Busca de CEP por argumento
            await BuscaCepConsoleAsync();
            */

            //Testando Busca de CEP por Objeto
            await BuscaCepObjectAsync();
        }

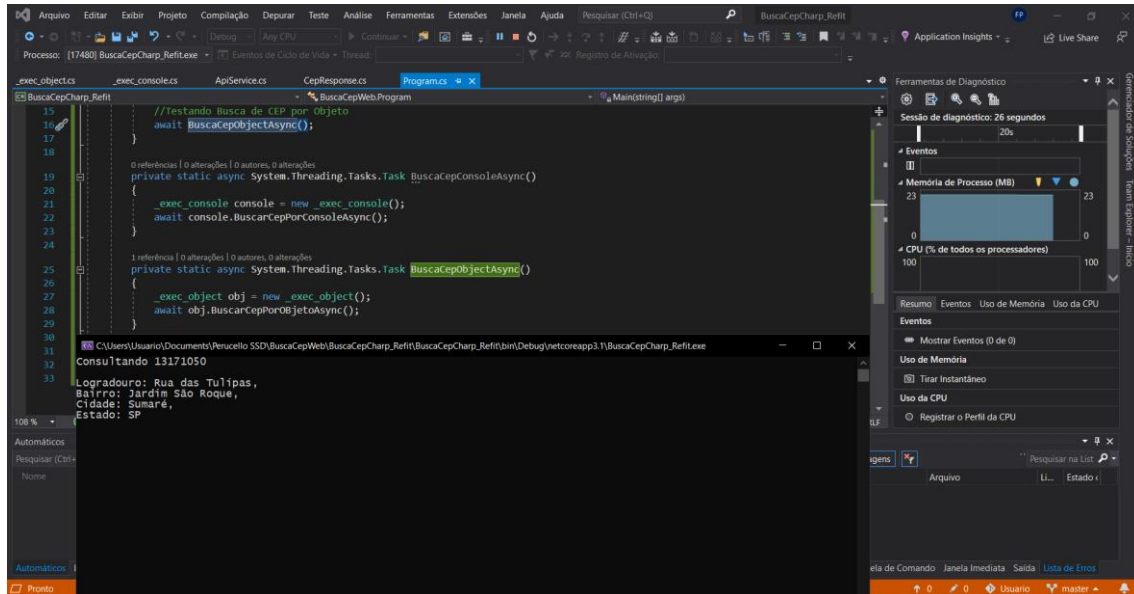
        private static async System.Threading.Tasks.Task BuscaCepConsoleAsync()
        {
            _exec_console console = new _exec_console();
            await console.BuscarCepPorConsoleAsync();
        }

        private static async System.Threading.Tasks.Task BuscaCepObjectAsync()
        {
            _exec_object obj = new _exec_object();
            await obj.BuscarCepPorObjetoAsync();
        }
    }
}

```



Agora, vamos executar , lembrando que o CEP informado está mocado.



Nossa API funcionou como nossa proposta, sendo assim finalizamos este artigo, onde os códigos estão disponíveis no GitHub para consumo.

Até mais !
Espero ter ajudado !

