

1. Documentação Técnica

Visão Geral do Projeto:

O projeto tem como finalidade executar um serviço utilizando Horse para a disponibilidade de um endpoint local e quando consultado será verificado em uma lista de endpoints verificando o primeiro que contiver uma resposta. Para este caso em específico a nossa consulta será para o Cep verificando se obtem resposta em algum dos endpoints da lista. Quando retornado sua resposta é devolvida para a consulta principal ou seja para o nosso endpoint criado com o Horse que disponibilizamos localmente. Dessa forma podemos centralizar uma lista de endpoints e caso algum esteja fora do ar ele irá verificar no próximo da lista, devolvendo para nossa aplicação principal a resposta independente de saber qual a api de cep consultada.

Arquitetura do Sistema:

A arquitetura foi baseada no modelo de serviço e mensageria, devido a necessidade de trabalhar com uma espécie de fila trazendo a primeira resposta encontrada da nossa lista de apis para o cep. Entendemos que para este projeto não houve a necessidade de trabalhar com threads diretamente aguardando qual dos endpoints consultados trouxesse a resposta primeiro, nem com uma fila especificamente como no caso de utilizações com o RabbitMQ, devido o tamanho do projeto e também a sua utilização, ainda que esteja trabalhando de uma forma simplificada com uma lista se mantém performático sem complexidade para uma implementação mais robusta caso uma necessidade futura.

Tecnologias e Ferramentas Usadas:

Para a sua utilização trabalhamos com o Boss que é um gerenciador de dependências muito conhecido e utilizado, através dele fizemos a associação com o Horse para sua utilização.

Neste projeto também para versionamento de códigos utilizamos o SourceTree, Git e GitHub para sua disponibilização de códigos.

Na criação de modelos a serem trabalhados inicialmente até seu modelo final foi utilizado o draw.io.

Os fontes e o diagrama será possível acesar através do repositório do projeto no GitHub link: <https://github.com/felipescarvalho/TrySearchCep>

Instruções de Utilização:

Para a utilização do centralizador de consultas para o cep temos o serviço que subirá na porta local configurada 9000.

Então dentro da pasta do sistema “TrySearchCep\Project” temos o executável PrjTrySearchCep.exe. Assim que executado ele irá subir o endpoint <http://localhost:9000/searchcep/:cep> que está já disponível para fazer a consulta aguardando apenas a informação do cep como exemplo <http://localhost:9000/searchcep/03737010>. Com isso ele irá fazer a consulta na lista de endpoints

“<https://viacep.com.br/ws/>”

“<https://cep.awesomeapi.com.br/json/>”

<https://cdn.apicep.com/file/apicep/>

Após retorno ele devolverá na saída da nossa rota

“<http://localhost:9000/searchcep/03737010>” um texto com o json onde constará as informações do cep.

Json de exemplo retornado do “<https://viacep.com.br/ws/>” :

```
{
  "cep": "05424020",
  "address_type": "Rua",
  "address_name": "Professor Carlos Reis",
  "address": "Rua Professor Carlos Reis",
  "state": "SP",
  "district": "Pinheiros",
  "lat": "-23.57022",
  "lng": "-46.69684",
  "city": "São Paulo",
  "city_ibge": "3550308",
  "ddd": "11"
}
```

Obs: As apis trazem jsons parecidos porém alguns campos podem ser diferentes, basta fazer uma validação simples para alguns dos campos.

JSONs de retorno:

“<https://cep.awesomeapi.com.br/json/>”

```
{
  "cep": "05424020",
  "address_type": "Rua",
  "address_name": "Professor Carlos Reis",
  "address": "Rua Professor Carlos Reis",
  "state": "SP",
  "district": "Pinheiros",
  "lat": "-23.57022",
  "lng": "-46.69684",
  "city": "São Paulo",
  "city_ibge": "3550308",
  "ddd": "11"
}
```

<https://cdn.apicep.com/file/apicep/>

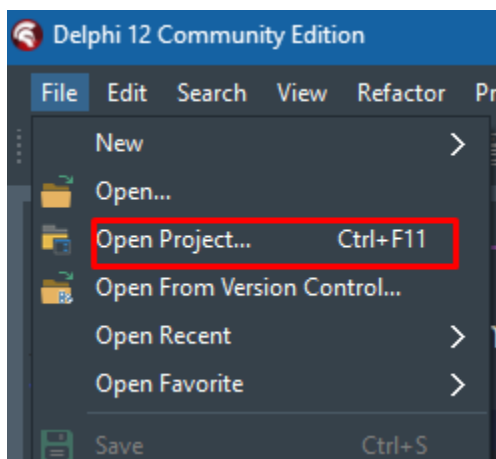
```
{  
  "code": "03737-000",  
  "state": "SP",  
  "city": "São Paulo",  
  "district": "Vila Buenos Aires",  
  "address": "Rua Jardim das Margaridas",  
  "status": 200,  
  "ok": true,  
  "statusText": "ok"  
}
```

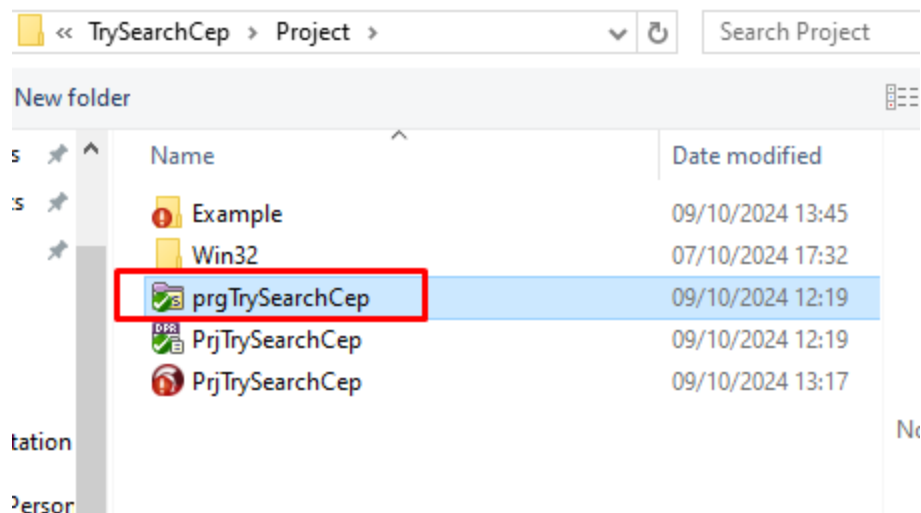
Para a validação e exemplificação de como funciona foi criado um projeto de exemplo que está contido também dentro da pasta “TrySearchCep\Project\Example PrjExample.exe”.

Caso não encontrado diretamente dentro da pasta procure dentro da “TrySearchCep\Project\Example win32” pois pode ser que esteja configurado para gerar o executável diretamente lá, dentro da pasta “debug” ou da “release”.

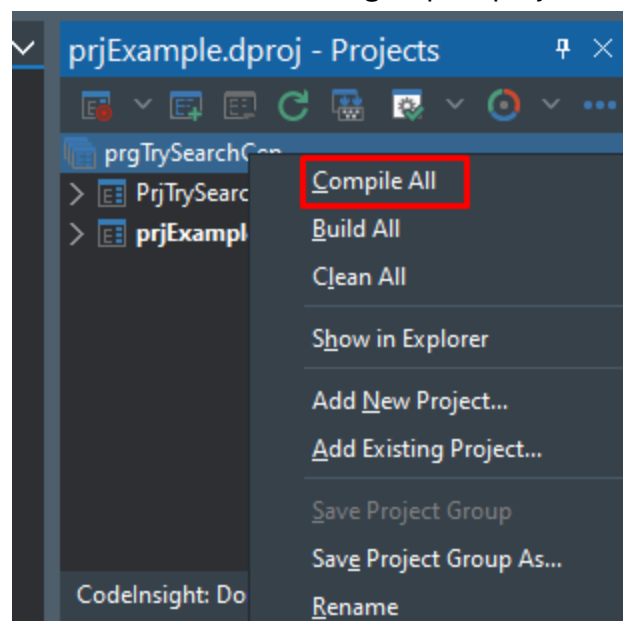
Para a geração de ambos executáveis basta abrir o projetGroup “prgTrySearchCep” no delphi e dar um compile all.

Exemplo detalhado de uso:

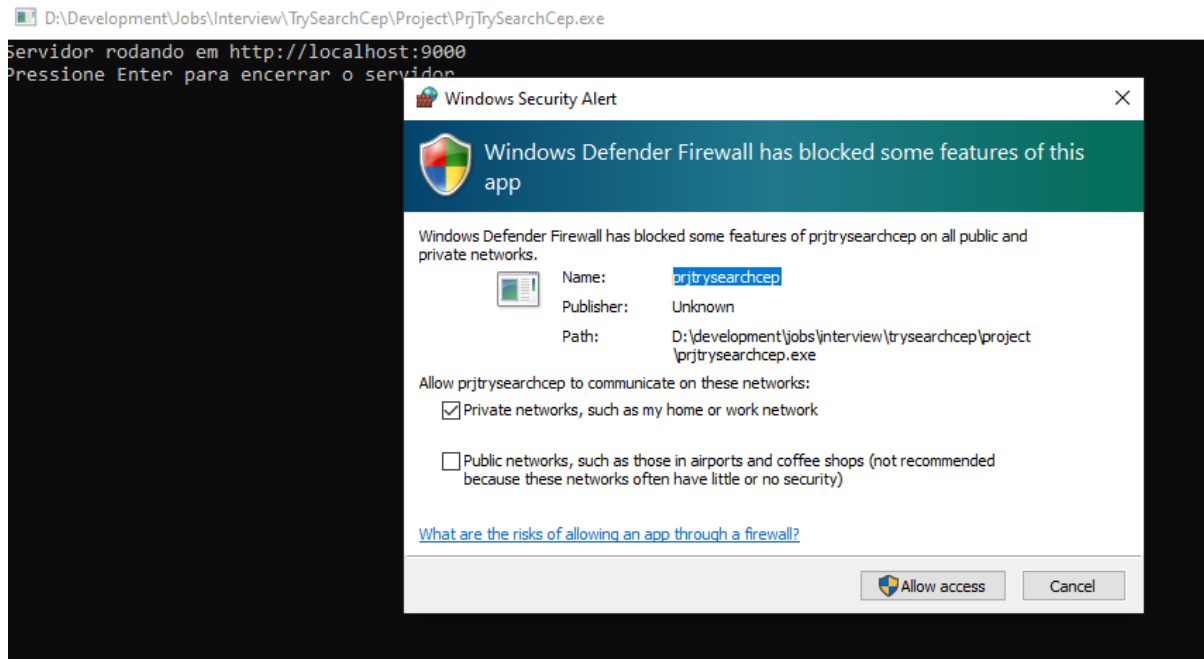




Botão direito em cima do group do projeto

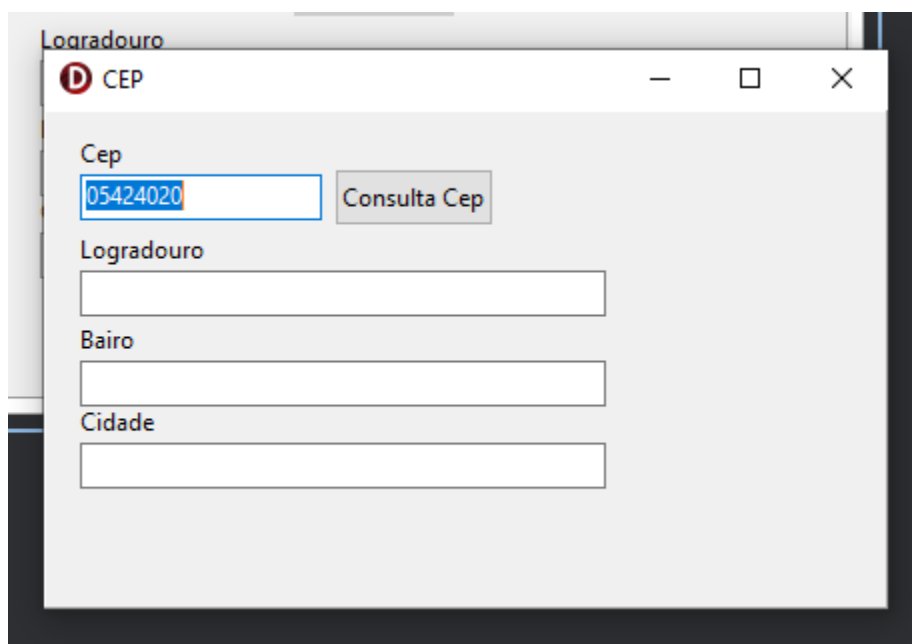


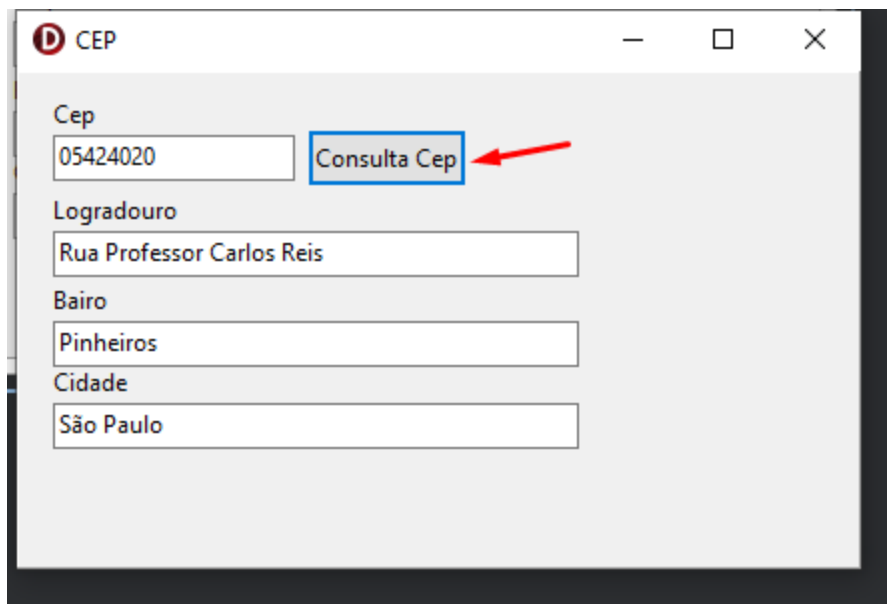
Executa o service para subir a rota com o horse e permita o windows dando acesso



Após isso você terá um serviço rodando escutando na porta 9000 que é a configurada inicialmente, podendo trocar conforme sua necessidade.

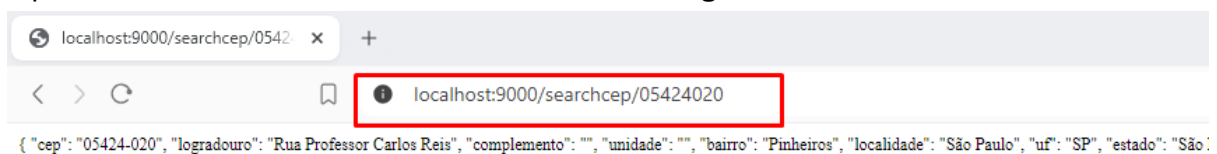
Agora abra o nosso projetinho de exemplo e mande ele consultar





Pronto está feita a consulta verificando na nossa lista de endpoints e trazendo as informações.

É possível verificar também através do nosso navegador



2. Documentação de Código

Comentários no Código:

No nosso console que é onde iremos fazer todas as execuções e validações no nosso view source podemos identificar a nossa classe principal que iremos fazer a chama e criação de tudo a TBuilde.StartEngine.

Obs: A rota de ping/pong deixamos apenas para fins de testes e validação se o servidor está ativo.

```

1 program PrjTrySearchCep;
.
. {$APPTYPE CONSOLE}
.
. {$R *.res}
.
. uses
.   System.SysUtils,
.   System.Classes,
.   Horse,
20   Builder.Impl in '..\src\Implementation\Builder.Impl.pas',
.   Registers.API.Impl in '..\src\Implementation\Api\Registers.API.Impl.pas',
.   Controller.Manager.Cep.Impl in '..\src\Implementation\Controller\Controller.Manager.Cep.Impl.pas',
.   CircuitBreaker.Impl in '..\src\Implementation\CircuitBreaker.Impl.pas';
.
. procedure Ping(Req: THorseRequest; Res: THorseResponse; Next: TNextProc);
. begin
.   Res.Send('Pong');
. end;
20
. begin
.   try
.     THorse.Get('/ping', ping);
.
.     TBUILDER.StartEngine;
.
.     WriteLn('Servidor rodando em http://localhost:9000');
.     WriteLn('Pressione Enter para encerrar o servidor...');
.     THorse.Listen(9000);
30
.     ReportMemoryLeaksOnShutdown := true;
.     ReadLn;
.   except
.     on E: Exception do
.       WriteLn(E.ClassName, ': ', E.Message);
.   end;
. end;

```

Assim que acessamos ela temos acesso ao a nossa classe utilizada para registrar o nosso endpoint.

```

20 { TBUILDER }
.
. class procedure TBUILDER.RegistersEndPoints;
. begin
.   TRegisterApi.RegistersEndPoints;
. end;
.
27 class procedure TBUILDER.StartEngine;
. begin
.   RegistersEndPoints;
30 end;
.
. end.

```

Dentro da classe de registro dos endpoints temos o método “class

procedure RegisterApiCep” que utilizamos para registrar o nosso para o cep, deixando sempre a possibilidade de expansão, por isso, trabalhado dessa forma e não diretamente tudo dentro de um método.

```

type
  TRegisterApi = class
  strict private
    class procedure RegistersApiCep;
  strict protected
    class procedure DoRegistersEndPoints;
  public
    class procedure RegistersEndPoints;
  end;

implementation

uses
  Horse,
  Controller.Manager.Cep.Impl;

{ TRegisterApi }

class procedure TRegisterApi.DoRegistersEndPoints;
begin
  RegistersApiCep;
end;

class procedure TRegisterApi.RegistersApiCep;
begin
  THorse.Get('/searchcep/:cep',
    procedure (Req: THorseRequest; Res: THorseResponse; Next: TNextProc)
    begin
      var lResponse:= TSearchManagerCep.RequestApi(Req.Params.Items['cep']);
      Res.Send(lResponse);
    end);
end;

class procedure TRegisterApi.RegistersEndPoints;
begin
  DoRegistersEndPoints;
end;

```

Ainda dentro dessa classe temos a chamada para a nossa classe que será a controladore dentro da nossa unit Controller.Manager.Cep.Impl.

Essa unit será a responsável por validar dentro da lista que iremos criar dos endpoints para consulta do cep e retornará para a gente o resultado quando receber da request que estará fazendo.

```

class procedure TRegisterApi.RegistersApiCep;
begin
  THorse.Get('/searchcep/:cep',
    procedure (Req: THorseRequest; Res: THorseResponse; Next: TNextProc)
    begin
      var lResponse:= TSearchManagerCep.RequestApi(Req.Params.Items['cep']);
      Res.Send(lResponse);
    end);
end;

```

Então assim que recebermos uma requisição na nossa rota "<http://localhost:9000/searchcep/:cep>" é repassado o cep para dentro da nossa RequestAPI.


```

1 init CircuitBreaker.Impl;
.
.
. interface
.
. uses
.     Generics.Collections;
.
. type
.
. TClassDefaultApi = class
10 private
.     FURL: string;
.     FMaxFailures: Integer;
.     FFailures: Integer;
.     FLastFailureTime: TDateTime;
.     FTimeout: TDateTime;
.
. public
.     constructor Create(const aBaseUrl: string);
.     function Request(out aError: string): string;
.     function CanAttempt: Boolean;
20     procedure RecordFailure;
.     procedure RecordSuccess;
.
. end;
.
. TClassDeafaultApiList = class
. private
.     FListClassApi: TList<TClassDefaultApi>;
. public
.     property ListClassApi: TList<TClassDefaultApi> read FListClassApi write FListClassApi;
.     constructor Create(const aCEP: string);
30     destructor Destroy; override;
.
. end;
.

```

Iremos entrão criar uma lista de classe

```

. { TClassDeafaultApiList }
.
. constructor TClassDeafaultApiList.Create(const aCEP: string);
. begin
.     FListClassApi := TList<TClassDefaultApi>.create;
.
.     FListClassApi.Add(TClassDefaultApi.Create('https://viacep.com.br/ws/' + aCEP + '/json/'));
.     FListClassApi.Add(TClassDefaultApi.Create('https://cep.awesomeapi.com.br/json/' + aCEP));
.     FListClassApi.Add(TClassDefaultApi.Create('https://cdn.apicep.com/file/apicep/' + aCEP + '.json'));
. end;
.
. destructor TClassDeafaultApiList.Destroy;
.

```

E trabalharemos em cima dela, deixando sempre a possibilidade de expansão.

Depois de criada a nossa lista voltamos e iremos varrer nossa lista de classe para fazer nossas requisições.

```

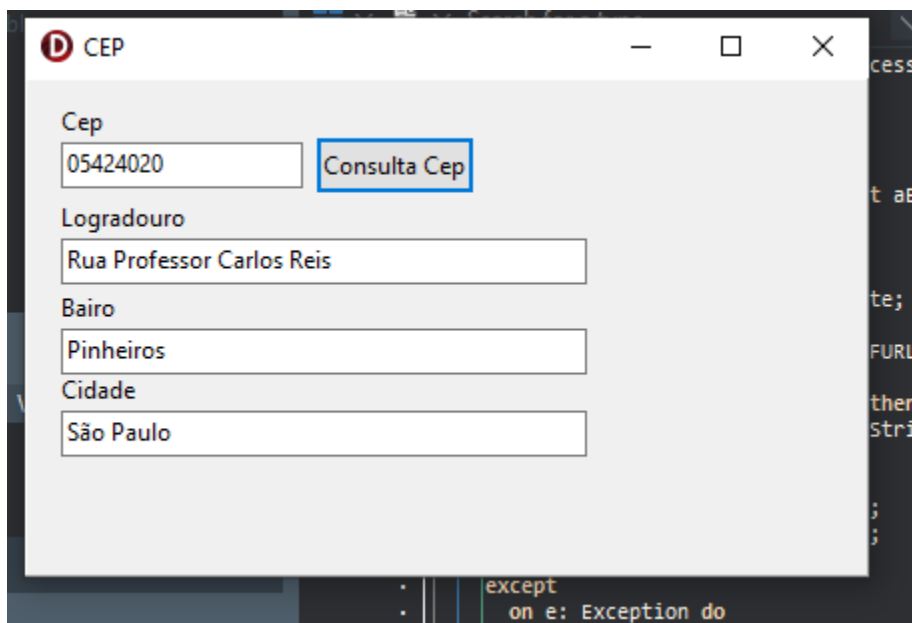
. class function TSearchManagerCep.RequestAPI(const ACEP: string): string;
20 begin
.     var lClassDeafaultApiList := TClassDeafaultApiList.Create(ACEP);
.     try
.         for var lItemConumerApi in lClassDeafaultApiList.ListClassApi do
.             begin
.                 if lItemConumerApi.CanAttempt then
.                     begin
.                         var lError: string := '';
.                         Result := lItemConumerApi.Request(lError);
.                         if (lError.IsEmpty) and (Result <> EmptyStr) then
30                         begin
.                             lItemConumerApi.RecordSuccess;
.                             Exit;
.                         end
.                     else
.                         lItemConumerApi.RecordFailure;
.                     end;
.                 end;
.             end;
.         end;
.     end;
.

```

A requisição foi feita de uma forma bem simples apenas para trazer nossa resposta quando ocorrer, porém, está bem simples para fazer as validações e implementações de logs e validação do json de retorno.

```
function TClassDefaultApi.Request(out aError: string): string;
begin
    var lJSON: string := '';
    var lError: string;
    var lHttpClient:= THttpClient.Create;
    try
        var lResponse:= lHttpClient.Get(FURL);
        try
            if lResponse.StatusCode = 200 then
                lJSON := lResponse.ContentAsString(TEncoding.UTF8)
            else
                case lResponse.StatusCode of
                    400: {implementar retorno};
                    404: {implementar retorno};
                end;
            except
                on e: Exception do
                    begin
                        lError := lResponse.StatusText + ' ' + e.Message;
                    end;
                end;
            finally
                lHttpClient.Free;
            end;
        end;
        lError := aError;
        Result := lJSON;
    end;
```

Quando retornado o nosso json de uma das apis devolvemos para nossa requisição principal, seja ela através do nosso projetinho de exemplo ou via postman.



The image shows a Windows application window titled "CEP". Inside the window, there is a form with four text input fields and one button. The first field is labeled "Cep" and contains the value "05424020". The second field is labeled "Logradouro" and contains "Rua Professor Carlos Reis". The third field is labeled "Bairro" and contains "Pinheiros". The fourth field is labeled "Cidade" and contains "São Paulo". To the right of the "Cep" field is a button labeled "Consulta Cep". The button has a blue border. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Obs: Existem 3 tipos de informações desses JSONs de retorno como informado acima na documentação, dentro da nosso projeto de exemplo tem uma classe basica aonde

faz a validação para tratar o retorno do json, o que pode ser jogado dentro do da nossa aplicação de retorno principal.

3. Boas Práticas

Mantenha sempre a possibilidade de expansão da aplicação como foi feito para manter sempre de uma forma simples e prática para quem for utilizar, seja o projeto completo ou parte dele como base.