

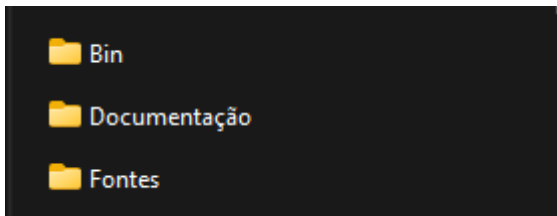
**Linguagem: Object Pascal (Delphi)**

**Tecnologia: POO**

**Padrão de Projeto: MVC**

**Sistema P/ Consulta de Endereços Baseada no CEP**

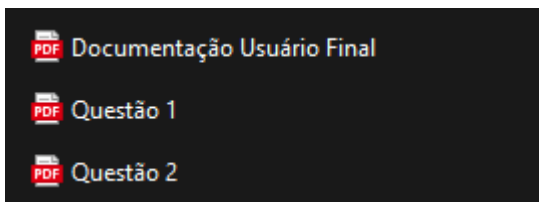
Estrutura de Diretórios:



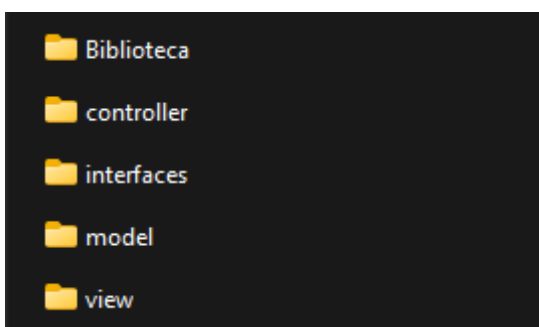
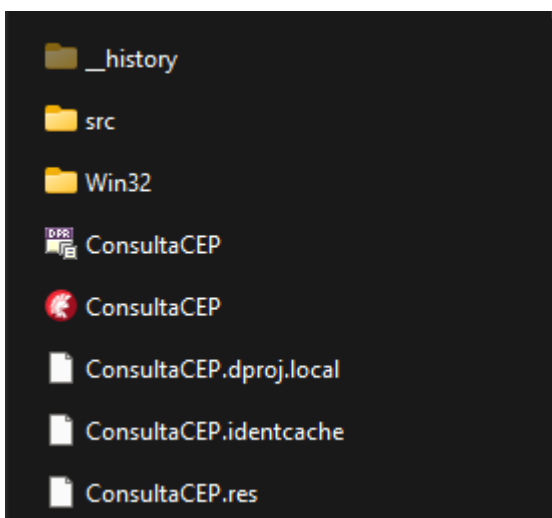
Bin: Executável e outros arquivos pertinentes p/ distribuição dos clientes e usuários.



Documentação: Arquivos destinados a leitura e entendimento de toda estrutura do sistema.


















Fontes: Código fonte do sistema subdividido por outros diretórios.



## Do Desenvolvimento

Foi desenvolvido um método p/ realizar consulta /p dados de endereço utilizando o parâmetro CEP, a classe de consulta foi desenvolvida criando e destruindo os componentes em tempo de projeto evitando a adição dos componentes nos formulários e utilizando a reutilização dos métodos em todas as consultas.

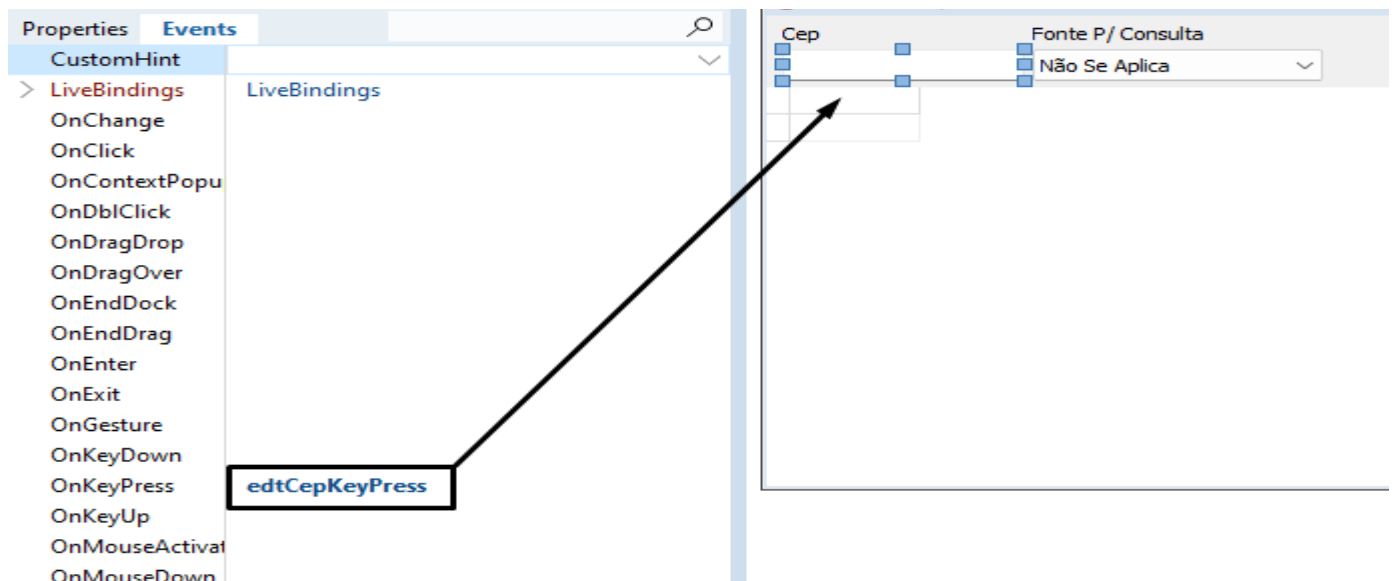
### Estrutura:

- ▼  **ConsultaCEP.exe**
  - >  Build Configurations (Debug)
  - >  Target Platforms (Windows 32-bit)
  - ▼  src
    - ▼  Biblioteca
      -  uFuncoes.pas
    - ▼  controller
      -  Controller.Conexao.pas
    - ▼  interfaces
      -  Interfaces.Conexao.pas
      -  Interfaces.Controller.Conexao.pas
    - ▼  model
      -  Model.Conexao.pas
    - ▼  view
      - >  View.FormPrincipal.pas

## View:

Formulário responsável pela visualização dos clientes e usuários, onde fica de forma flexível e de fácil entendimento o formato da consulta.

No Onkeypress do componente edit foi realizada a programação de busca.



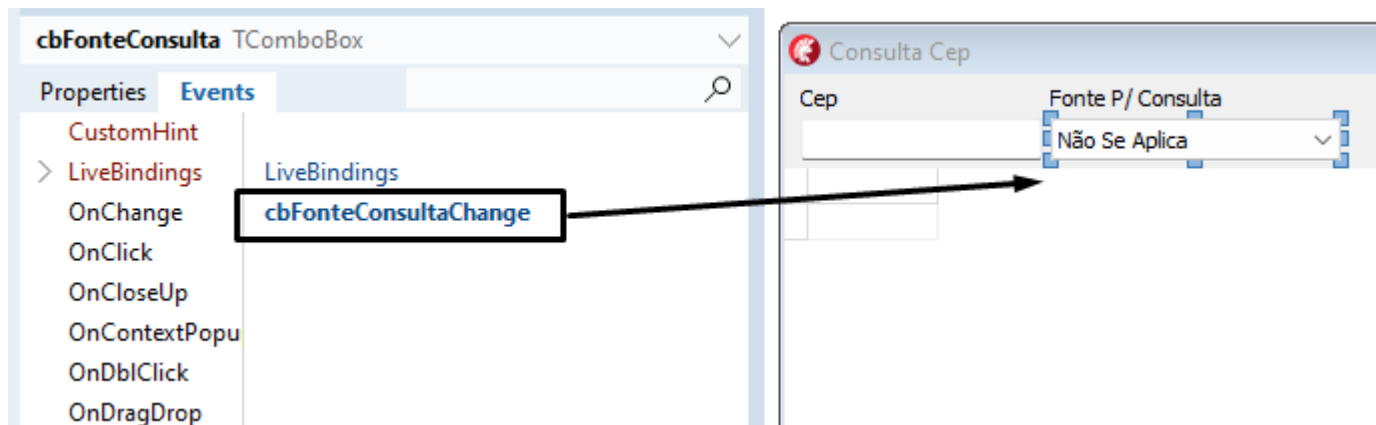
```
procedure TFormConsultaCep.edtCepKeyPress(Sender: TObject; var Key: Char);
begin
    if key = #13 then
    begin
        ConsultaCEP;
    end;
end;
```

No Evento foi programado qual tipo de tecla é pressionada e uma procedure que realizar uma ação de validação ou consulta do cep.

```
procedure TFormConsultaCep.ConsultaCEP;
begin
    try
        if (edtCep.Text = '') then
        begin
            ShowMessage('Atenção... Digite um cep válido !');
            edtCep.SetFocus;
            exit;
        end
        else
        begin
            if (cbFonteConsulta.ItemIndex = 0) then
            begin
                edtCep.Text := MascaraCEP(edtCep.Text);
                ShowMessage('Atenção... Selecione uma fonte de busca !');
                cbFonteConsulta.SetFocus;
                exit;
            end
            else
            begin
                TControllerConexao
                .New
                .Conexao
                .RestRequest
                (
                    dsConsulta,
                    cbFonteConsulta.ItemIndex,
                    edtCep.Text
                );
                edtCep.Text := MascaraCEP(edtCep.Text);
            end;
        end;
    except on E: Exception do
    end;
end;
```

O Evento verifica se o campo está vazio ou o combobox igual a 0.

No Evento OnChange do Combobox também realizamos uma validação.



```
procedure TFormConsultaCep.cbFonteConsultaChange(Sender: TObject);  
begin  
40  if (edtCep.Text = '') and (cbFonteConsulta.ItemIndex <> 0) then  
begin  
    ShowMessage('Atenção... Antes de consulta digite um cep válido !');  
    edtCep.SetFocus;  
    exit;  
end;  
    if Assigned(Grid.DataSource.DataSet) then  
begin  
    Grid.DataSource.DataSet.Close;  
    ConsultaCEP;  
50  end  
    else  
        ConsultaCEP;  
end;  
end;
```

Procedure ConsultarCEP

Caso esteja tudo certo com a consulta passamos via parâmetros os seguintes dados para o controller

DataSource

ItemIndex

edtCep

```
begin  
    TControllerConexao  
    .New  
    .Conexao  
    .RestRequest  
    (  
        dsConsulta,  
        cbFonteConsulta.ItemIndex,  
        edtCep.Text  
    );  
    edtCep.Text := MascaraCEP(edtCep.Text);  
end;
```

Dentro do Sistema utilizamos a seguinte estrutura até chegar no Model.

#### Interfaces.Conexao

```
type
  IRequest = interface
    ['{21FD205F-967F-4827-832E-F43936A148DF}']
    function RestRequest(aDataSet : TDataSource; aOrigem : Integer; aCep : String) : IRequest;
  end;
```

#### Interfaces.Controller.Conexao

```
type
  IControllerConexao = interface
    ['{B8452494-CA46-45BE-99EE-A3A6BD1CD8E3}']
    function Conexao : IRequest;
  end;
```

#### Controller.Conexao

```
type
  TControllerConexao = class(TInterfacedObject, IControllerConexao)
  private
  public
    Constructor Create;
    destructor destroy; override;
    class function New : IControllerConexao;
    function Conexao : IRequest;
  end;

implementation

{ TControllerConexao }

function TControllerConexao.Conexao: IRequest;
begin
  Result := TConexao.New;
end;

constructor TControllerConexao.Create;
begin
end;

destructor TControllerConexao.destroy;
begin
  inherited;
end;

class function TControllerConexao.New: IControllerConexao;
begin
  Result := Self.Create;
end;
```

## Model.Conexao

```
type
  TConexao = class(TInterfacedObject, IRequest)
  private
    FConexao: TRESTClient;
    FRequest : TRESTRequest;
    FResponse : TRESTResponse;
    FResponseDataSetAdapter: TRESTResponseDataSetAdapter;
  public
    Constructor Create;
    destructor destroy; override;
    class function New : IRequest;
    function RestRequest(aDataSet : TDataSource; aOrigem : Integer; aCep : String) : IRequest;
  end;
```

### Create

Criamos todos os nossos componentes e passamos todos os parâmetros necessário p/ realizar uma consulta, o sistema foi pensado para ser fechado p/ modificações e aberto p/ novas implementações, o mesmo método faz a consulta de das 3 api's, possibilitando a adição de novas rotas e forma simples e eficiente.

```
constructor TConexao.Create;
begin
  FConexao := TRESTClient.Create(nil);
  FRequest := TRESTRequest.Create(nil);
  FResponse := TRESTResponse.Create(nil);
  FResponse.ContentType := 'application/json';
  FResponseDataSetAdapter := TRESTResponseDataSetAdapter.Create(nil);
  FMemTable := TFDMemTable.Create(nil);

  with FConexao do
  begin
    Accept := 'application/json, text/plain; q=0.9, text/html;q=0.8,';
    AcceptCharset := 'utf-8, *,q=0.8';
    ContentType := 'application/x-www-form-urlencoded';
    SecureProtocols := [THttpSecureProtocol.SSL2, THttpSecureProtocol.SSL3, THttpSecureProtocol.TLS1];
  end;
  with FRequest do
  begin
    Accept := 'application/json, text/plain; q=0.9, text/html;q=0.8,';
    AcceptCharset := 'utf-8, *,q=0.8';
    AssignedValues := [TCustomRESTRequest.TAssignedValue.rvConnectTimeout,
                      TCustomRESTRequest.TAssignedValue.rvReadTimeout];
  end;
  with FResponseDataSetAdapter do
  begin
    AutoUpdate := true;
    Dataset := FMemTable;
    Response := FResponse;
    TypesMode := TJSONTypesMode.JSONOnly;
  end;
end;

destructor TConexao.destroy;
begin
  FreeAndNil(FConexao);
  FreeAndNil(FRequest);
  FreeAndNil(FResponse);
  FreeAndNil(FResponseDataSetAdapter);
  inherited;
end;
```

## RestRequest

Método responsável p/ retornar as consultas realizadas via CEP.

```
function TConexao.RestRequest(aDataSet : TDataSource; aOrigem : Integer; aCep : String) : IRequest;
begin
    try
        if aOrigem = 1 then {ViaCep}
            FConexao.BaseURL := 'http://viacep.com.br/ws/'+RemoverCaracteresEspeciais(aCep)+'/.json';
        if aOrigem = 2 then {apicep}
            FConexao.BaseURL := 'https://cdn.apicep.com/file/apicep/'+MascaraCEP(aCep)+'/.json';
        if aOrigem = 3 then {awesomeapi}
            FConexao.BaseURL := 'https://cep.awesomeapi.com.br/json/'+RemoverCaracteresEspeciais(aCep);

        FRequest.Client := FConexao;
        FRequest.ConnectTimeout := 3000;
        FRequest.Method := rmGET;
        FRequest.Response := FResponse;

        FRequest.Execute;

        if FResponse.StatusCode <> 200 then
            begin
                ShowMessage('Atenção... API Indisponível no momento, selecione outra opção e tente novamente');
                exit
            end
        else
            begin
                if FResponseDataSetAdapter.Active = false then
                    FResponseDataSetAdapter.Active := true;
                if FMemTable.Active = false then
                    FMemTable.Active := true;

                aDataSet.DataSet := FMemTable;
            end;
        except On E: Exception do
            begin
                ShowMessage('Atenção... O Cep digitado é inválido !');
                abort
            end;
        end;
    end;
end;
```

Foi realizada uma validação com o tipo do Status, assim evitando que o usuário final se depare com algum problema de consulta e dando mais opções p/ que o processo de retorno da consulta seja realizado de forma satisfatória.