TOTVS

Documentação para o desenvolvedor:

Projeto API.CEP;

O Projeto API.CEP conta com alguns padrões de projeto, como:
Singleton, Aplicado na Controller para garantir apenas uma instância.
MVC, Repository e Services. Estes padrões ajudam a desacoplar a lógica da camada Model da aplicação.
Ganhando então a Repository, que foi aplicada para consumir a API de terceiros. E a Services, que pode ganhar regras de negócios, antes de devolver uma resposta para o client.
Com a aplicação da Repository e Services, a Model passa a ser uma DTO apenas, ou Entity, como preferir.

Para montar a API, foi utilizado Horser, escutando na porta 9000.
Aplicado paralelismo ao Horse, para liberar o terminar, assim, pode ser encerrada a API, ao pressionar qualquer tecla do terminal.

A arquitetura do projeto ficou da seguinte forma:

Controller > Services > Repository > Model.

Aplicado Modules, para libs extras, como GerarLog, e ClientSock, que consome a API de terceiros.

Utilizado Generics, para que o Socket consiga retornar qualquer classe, assim, evitando escrita excessiva para cada Model. Este feito acontece com a junção à classe Repository.

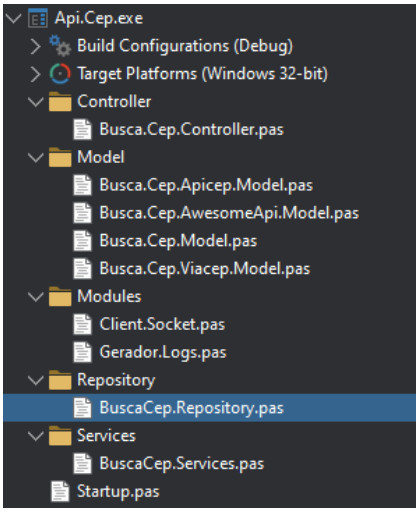
Classe repository, o fluxo de comunicação com as APIs externas, onde, ao falhar a primeira API, buscará a segunda, caso falhe, buscará a terceira.

Nas camadas Repository e Services foram utilizadas interfaces, assim ganhando com o garbage collector e apoio ao Generics.

Rota da API.
http://localhost:9000/BuscarCep/{cep}

Exemplo: localhost:9000/BuscarCep/12929-605

Padronização

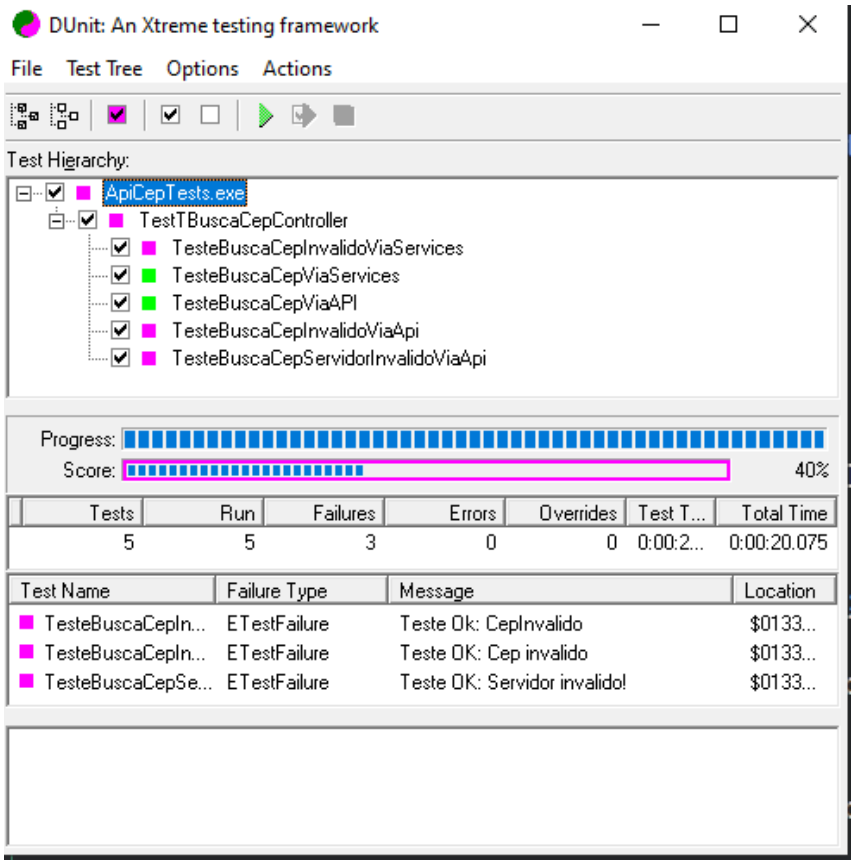


Projeto: ApiCepTest

Aplicação de testes unitários.
Utilizando o Dunit do Delphi, criando testes que falharão e testes que serão um sucesso.

Os erros são em cenários controlados e esperados.
O teste se dividiu em dois critérios, testes via API e teste via código, isolando a API.

Via API, o REST chama a API, via rota, e passa os parâmetros necessários.
Via Services, isolado a camada da API, e chamado o método pela Service.



Projeto: BuscaCep.Client

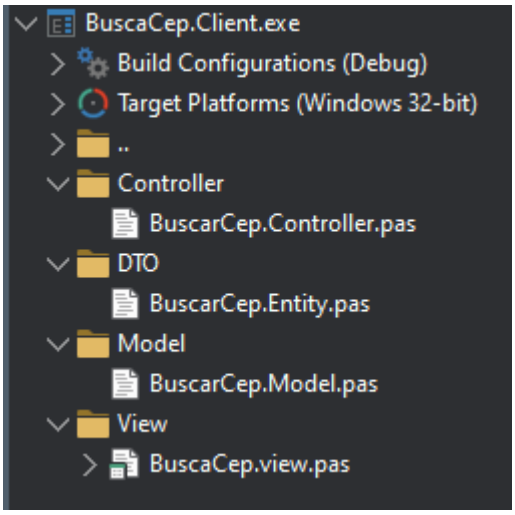
Para o Client, foi utilizado FMX com Skia.
Padrão de projeto MVC.

Sendo a distribuição.
View, cuidando somente do que é visual ao usuário.
Controller, criando a rota.
Model, Aqui, como é o padrão MVC apenas, ficou com a lógica.
DTO, utilizado para transferência de dados, para o objeto sendo apoio para model, e renderizando na view.

Foi reaproveitado os códigos da API, Model. Pois cada API Externa tem sua própria model, na view, foi unificada em uma única Model. Mediante um parse.
Reaproveitado a Modules da Api também, para gerar logs.

Padrão aplicado no Client.

View > Controller > Model > Dto;



Foi criada uma pasta de código compartilhado, que o Client e a ApiCepTeste compartilham, para chamar a ApiCep.