Desafio BC

Documento Visão Geral  
*Esse documento tem por finalidade explanar como foi desenvolvido a solução.*

É abordado sobre o problema, uma possível solução estrutural, como as duas Apis estão estruturadas, e de que forma foi implantado o banco de dados.

# Problema

*Segundo a empresa, o problema proposto é: “A empresa possui o cadastro via APP, e nessa onda, habilitamos o cadastro via API para Onboard de nossos clientes.Tendo esse problema, você deverá criar uma API para o cadastro destes cliente”*.

# Solução Arquitetura

Eu optei por criar duas Apis, que são: Api.RegisterCustomer e Api.PostalCode conforme pode ser visto na **Figura 1**.

A Api.PostalCode tem responsabilidade em enviar requisição a um link externo para obter o endereço completo. Já Api.RegisterCustomer tem um escopo um pouco maior, é possível cadastrar e buscar por clientes.

Diagrama

Descrição gerada automaticamente

Figura 1: Solução arquitetural para o problema proposto.

Na minha opnião, por desconhecer o sistema por completo e em virtude de se ter uma maior escalabilidade, portanto é interessante a criação de duas Apis que funcione de forma isolada, com escopo de funções menores e em virtude de indisponibilidade do serviço, as demais partes do sistema continuarão a funcionar. Ressaltando que o desenvolvimento é rápido e o custo é menor ao comparar-se com um sistema maior e uma api de responsabilidade única.

# Estrutura da Aplicação

Eu acredito que a estrutura da aplicação deve ser simples e muito bem feita, em razão da Api ter poucas resonsabilidades. Desta forma, as Apis estão estruturadas seguindo o conceito de arquitetura limpa, o projeto é fácil de fácil entendimento por outros desenvolvedores, acabo ganhando também tempo de desenvolvimento e trazendo lucros maiores para a empresa.

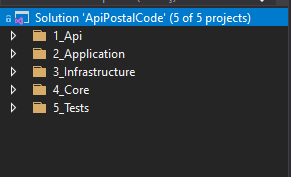
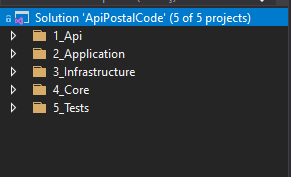
 

Figura 2: Estrutura das Apis com o conceito de arquitetura limpa.

Foi utilizado o Visual Studio Enterprise 2019, os projetos foram criandos com WebApi .net 5 e o versionamento de código feito no [GitHub](https://github.com/Jeffconexion/Challenge_BC)

# Banco de Dados

Para realização da persistência dos dados, foi utilizado o Sql Server Express, sem a geração do migration pelo Visual Studio, dessa forma seguindo o conceito de database first. Entretanto, tomando os devidos cuidados para trazer as regras de negócio para responsabilida da aplicação. A seguir um dicionário de dados e os scripts para executar no Sql Server.

Na **Tabela 1 e Tabela 2**, mostro as tabelas que foram criadas como regra de negócio para aplicação. Inclusive foi criado uma view, com os campos essenciais das três tabelas, ganhando dessa forma um maior desempenho ao realizar uma consulta.

Tabela 1: Descrição das tabelas e view.

|  |  |
| --- | --- |
| **Tabela** | **Descrição** |
| TB\_CUSTUMER | Tabela para registrar o novo cliente cadastrado no sistema. |
| TB\_ADDRESS | Tabela para registrar o endereço mediante o cep informado. |
| TB\_STATUS\_ADDRESS | Tabela para registrar o status da TB\_ADDRESS. |
| VW\_FULLDATA\_CUSTUMER | View com os campos essenciais das 3 tabelas. |

Tabela 2: Campos e descrições da view criada.

|  |  |  |
| --- | --- | --- |
| **Tabela** | **Coluna** | **Descrição** |
| VW\_FULLDATA\_CUSTUMER | name | Nome do cliente |
|  | tax\_id | Cpf ou Cnpj validado na aplicação. |
|  | phone\_number | Telefone do cliente. |
|  | created\_at | Data de criação da conta. |
|  | status | Para o Cep informado vai ser atribuído “Approved” caso contrário “Pending” |
|  | state | Estado do cliente |
|  | city | Cidade do cliente |
|  | district | Bairro do cliente |
|  | address | Endereço do cliente |
|  | code | Cep do cliente |

Na **Tabela 3**, Mostro todas as colunas, tipo, tamanho e o significado de cada coluna para cada tabela.

Tabela 3: Tabelas, colunas, tipo, tamanho e descrições de cada campo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tabela** | **Coluna** | **Tipo** | **Tamanho Máximo** | **Descrição** |
| TB\_CUSTUMER | id | uniqueidentifier | 16 | Guid gerado na aplicação. |
| TB\_CUSTUMER | name | varchar | 60 | Nome do cliente |
| TB\_CUSTUMER | tax\_id | varchar | 25 | Cpf ou Cnpj validado na aplicação. |
| TB\_CUSTUMER | password | varchar | 30 | Senha mínimo caracters de 8, 1 letra Maíscula, 1 Minúscula, Caracter Especial. Validado na aplicação |
| TB\_CUSTUMER | phone\_number | varchar | -1 | Telefone do cliente. |
| TB\_CUSTUMER | created\_at | datetime | 8 | Data de criação da conta. |
| TB\_STATUS\_ADDRESS | id | uniqueidentifier | 16 | Guid gerado na aplicação. |
| TB\_STATUS\_ADDRESS | status | varchar | 10 | Para o Cep informado vai ser atribuído “Approved” caso contrário “Pending” |
| TB\_ADDRESS | id | uniqueidentifier | 16 | Guid gerado na aplicação. |
| TB\_ADDRESS | id\_custumer | uniqueidentifier | 16 | Chave estrangeira que referência a TB\_CUSTUMER. |
| TB\_ADDRESS | id\_status\_address | uniqueidentifier | 16 | Chave estrangeira que referência a TB\_STATUS\_ADDRESS |
| TB\_ADDRESS | state | varchar | 4 | Estado do cliente |
| TB\_ADDRESS | city | varchar | 200 | Cidade do cliente |
| TB\_ADDRESS | district | varchar | 200 | Bairro do cliente |
| TB\_ADDRESS | address | varchar | 200 | Endereço do cliente |
| TB\_ADDRESS | code | varchar | 200 | Cep do cliente |

Em **Script 1**, está o passo a passo para criação do banco de dados, tabelas e view.





Script : Script utilizado para persistir dados da Api.RegisterCustomer.