**Projeto API**

**1: Tipo de projeto:**

**ASP.NET Core Web API (Versão 7):**

* O projeto foi desenvolvido usando ASP.NET Core Web API, versão 7.

**2. Configuração do Entity Framework:**

**2.0: Criar Models (dominio):**

* Models/Classe das entidades:
  + Classes: ClienteMod, ItensVend,Vendedor, Venda, Item.
  + Relacionamentos das classes.

**2.1: Criar a ViewModel:**

* ViewModel/Classe das entidades (para configurar no Service e não usar direto as models):
  + Com relacionamento das classes.

**2.2: Instalar os pacotes NuGet (para cada projeto):**

* package Microsoft.EntityFrameworkCore;
* package Microsoft.EntityFrameworkCore.Design;
* package Microsoft.EntityFrameworkCore.SqlServer;
* package Microsoft.EntityFrameworkCore.Tools;

**2.3: Criar DbContext(dominio):**

* Data/Classe de Contexto (ex: AplicacaoDbContext):
  + Criar uma classe de contexto recebendo **DbContext**, utilizando o construtor(quando code first):
    - public AplicacaoDbContext(DbContextOptions<AplicacaoDbContext> options) : base(options) { }
    - O public **DSet<>** setando a Model que quer transformar em Tabela.

**2.4: Criar ConnectionStrings:**

* Conexão com o banco de dados > **appsettings.json**:
  + Ex: "ConnectionStrings": {

"ClientesAPI": "Data Source=KEILALYNCAS;Initial

Catalog=ClienteAPI;Integrated Security=True;Encrypt=False" }

**2.5: Injeção de Dependência:**

* Receber a **ConnectionString** na configuração da aplicação(**program**):
  + Selecionar quem é o DBContext usado na aplicação, informando ao options que vai usar o SQLServer, passando qual o nome da conectionString que sera utilizada:

builder.Services.AddDbContext<AplicacaoDbContext> (options =>

{

options.UseSqlServer(builder.Configuration.GetConnectionString("ClientesVendas"));

});

**3: Criar a Repository:**

* Criar pasta Repository, com classe do **Repositorys** e **Interface**:
  + Criar a Interface do Repository com todos os métodos que precisam ser implementados.

**3.0** **InterfaceService:**

* Na Interface do Repository implementar os métodos:
  + GetAll,
  + GetById,
  + CreateAsync,
  + UpdateAsync,
  + DeteClienteById

**3.1 Classe repositorys:**

* Cria a propriedade que ele ficará dentro,
* Criar um construtor com Injeção de Dependência com a aplicação appDbContext,
* Configurar os métodos da Interface do Repository

**3: Criar a Service:**

* Pasta para cada entidade, com classe do **Service** mais a **Interface**:
  + Criar a Interface do Service com todos os métodos que precisam ser implementados.

**3.0** **InterfaceService:**

* Na Interface do Service implementar os métodos:
  + GetAll,
  + GetById,
  + CreateAsync,
  + UpdateAsync,
  + DeteClienteById

**3.1 Classe Service:**

* Cria a propriedade que ele ficará dentro,
* Criar um construtor com a Injeção de Dependência do Repository,
* Configurar as funções dos métodos da Interface Service

**4: Criar a Controller:**

* Criar Controllers/Classe: (no add do visual studio pode criar com controller/API empty)
* Cria a propriedade que ele ficará dentro,
* Fazer referência com o **ControllerBase**(proprio do entity framework)
* Criar um construtor com Injeção de Dependência com Interface do Service,

**4.0 Aplicar na Program:**

* Na Program avisar que a Interface do Service se comunica com o Service,

**4.0: Criar os verbos HTTP:**

* **GET:**
  + Utilizado para obter informações ou realizar consultas no servidor.
    - Utilizando o Task com assincrono para List/IEnumerable de ClienteViewModel
* **DELETE:**
  + Utilizado para solicitar a remoção de um recurso específico no servidor.
    - Ex: /api/clientes/{id}
* **POST:**
  + Criação de um novo recurso no servidor.
    - Ex: /api/clientes
* **PUT:** 
  + Atualização de um recurso existente com dados fornecidos.
    - Ex: /api/clientes/{id}

**5: Para cada endpoint**

**4.0: Swagger: (**exemplo de tabela de clientes)

* Consulta:
  + Endpoint: /api/clientes
  + Método HTTP: GET
  + Descrição: Este endpoint permite consultar todos os clientes cadastrados.
  + Como Testar (Swagger):
    - Acesse o Swagger da API.
    - Vá para o endpoint /api/clientes.
    - Clique no botão "Try it out".
    - Execute a solicitação.

* Exclusão:
  + Endpoint: /api/clientes/{id}
  + Método HTTP: DELETE
  + Descrição: Este endpoint permite excluir um cliente específico com base no ID.
  + Como Testar (Swagger):
    - Acesse o Swagger da API.
    - Vá para o endpoint /api/clientes/{id}.
    - Insira o ID do cliente desejado.
    - Clique no botão "Try it out".
    - Execute a solicitação.

* Alteração e inclusão no banco utilizando o Entity:
  + Inclusão (POST - /api/clientes)
    - Descrição: Este endpoint permite adicionar um novo cliente ao banco de dados.
    - Como Testar:
      * Acesse o Swagger da API.
      * Navegue até o endpoint /api/clientes.
      * Clique no botão "Try it out".
      * Preencha os detalhes do novo cliente no corpo da solicitação.
      * Execute a solicitação.
  + Alteração (PUT - /api/clientes/{id})
    - Descrição: Este endpoint permite atualizar as informações de um cliente existente com base no ID.
    - Como Testar:
      * Acesse o Swagger da API.
      * Vá para o endpoint /api/clientes/{id}.
      * Insira o ID do cliente desejado.
      * Clique no botão "Try it out".
      * Preencha os detalhes atualizados do cliente no corpo da solicitação.
      * Execute a solicitação.

**4.0: Postman: (**exemplo de tabela de clientes)

* Consulta:
  + Endpoint: <http://localhost:5164/api/clientes>
  + Método HTTP: GET
  + Como Testar:
    - Abra o Postman.
    - Selecione o método GET.
    - Insira a URL <http://localhost:5164/api/clientes>.
    - Clique em "Send".

* Exclusão:
  + Endpoint: http://localhost:5164/api/clientes/{id}
  + Método HTTP: DELETE
  + Como Testar:
    - Selecione o método DELETE.
    - Insira a URL http://localhost:5164/api/clientes/{id}, substituindo {id} pelo ID do cliente desejado.
    - Clique em "Send".

* Alteração e inclusão no banco utilizando o Entity:
  + Inclusão (POST - /api/clientes)
    - Endpoint: <http://localhost:5164/api/clientes>
    - Método HTTP: POST
    - Como Testar:
      * Abra o Postman.
      * Selecione o método POST.
      * Insira a URL http://localhost:5164/api/clientes.
      * No corpo da solicitação, forneça os detalhes do novo cliente em formato JSON.
      * Clique em "Send".
  + Alteração (PUT - /api/clientes/{id})
    - Endpoint (Alteração): [http://localhost:5164/api/clientes/{id](http://localhost:5164/api/clientes/%7Bid)}
    - Método HTTP: PUT
    - Como Testar:
      * Abra o Postman.
      * Selecione o método PUT.
      * Insira a URL http://localhost:5164/api/clientes/{id}, substituindo {id} pelo ID do cliente desejado.
      * No corpo da solicitação, forneça os detalhes atualizados do cliente em formato JSON.
      * Clique em "Send".

**Entidades (Entities):** Representam os objetos principais do domínio e refletem a estrutura dos dados no banco de dados. São usadas principalmente na camada de persistência.

**DTOs (Data Transfer Objects):** São usados para transferir dados entre diferentes partes da aplicação, geralmente entre a camada de controle e os serviços. Permitem uma comunicação eficiente entre camadas.

**ViewModels:** São usadas na camada de apresentação para representar os dados exibidos nas visualizações. Contêm apenas os dados necessários para renderizar uma determinada visualização e são projetadas para atender às necessidades específicas de exibição da interface do usuário.

**Seguem algumas dicas para as api`s q vocês estão utilizando:**

* **GET**:
* **Descrição**: Usado para recuperar recursos existentes.
* **Exemplo**: Obter uma lista de todos os itens em um recurso ou obter detalhes de um item específico.
* **Resposta**: Retorne o recurso solicitado no corpo da resposta. Para listas, o corpo pode conter uma matriz JSON de recursos. Para solicitações de detalhes de um único item, retorne um objeto JSON representando o recurso.
* **POST**:
* **Descrição**: Usado para criar novos recursos ou executar operações complexas.
* **Exemplo**: Adicionar um novo item a uma coleção de recursos ou consultar com filtros complexos.
* **Resposta**: Retorne o recurso recém-criado no corpo da resposta, juntamente com o código de status HTTP 201 Created. O cabeçalho Location deve conter o URI do recurso recém-criado.
* **PUT**:
* **Descrição**: Usado para atualizar um recurso existente por completo.
* **Exemplo**: Atualizar todas as propriedades de um recurso existente.
* **Resposta**: Não é necessário retornar o corpo da resposta, a menos que seja necessário fornecer informações adicionais. Um código de status HTTP 200 OK é comum para indicar que a atualização foi bem-sucedida.
* **DELETE**:
* **Descrição**: Usado para excluir um recurso existente.
* **Exemplo**: Remover um item de uma coleção de recursos.
* **Resposta**: Não é necessário retornar um corpo na resposta. Um código de status HTTP 204 No Content é comum para indicar que a exclusão foi bem-sucedida.

É importante lembrar que as respostas HTTP devem ser consistentes e informativas para facilitar o entendimento e o uso da API pelos clientes. Além disso, os códigos de status HTTP devem ser usados de acordo com as especificações padrão para garantir a interoperabilidade e a confiabilidade da API.

* A camada de aplicação depende da camada de serviço.
* A camada de serviço depende da camada de domínio e da camada de repository.
* A camada de repository depende da camada de domínio.