API – é uma forma de comunicação entre programas e computadores, em outras palavras é um software que fornece informações para outro software.

Forma de integração entre sistemas.

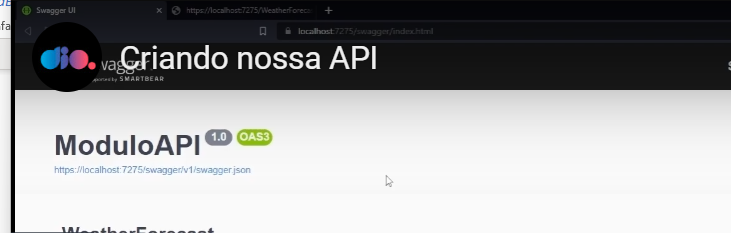
Diagrama

Descrição gerada automaticamente

Criar uma nova api

Dotnet new webapi – cria API

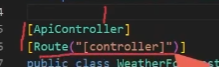
dotnet watch run executa uma api (A função watch, toda alteração que ser feito na API não vai precisar pausar o projeto e rodar denovo, ele fica observando as mudança da sua API e caso tenha alguma mudança ele recompila automaticamente o seu projeto.



Swagger serve para auxiliar na documentação da APi e testes, API não é obrigada a ter um Swagger.

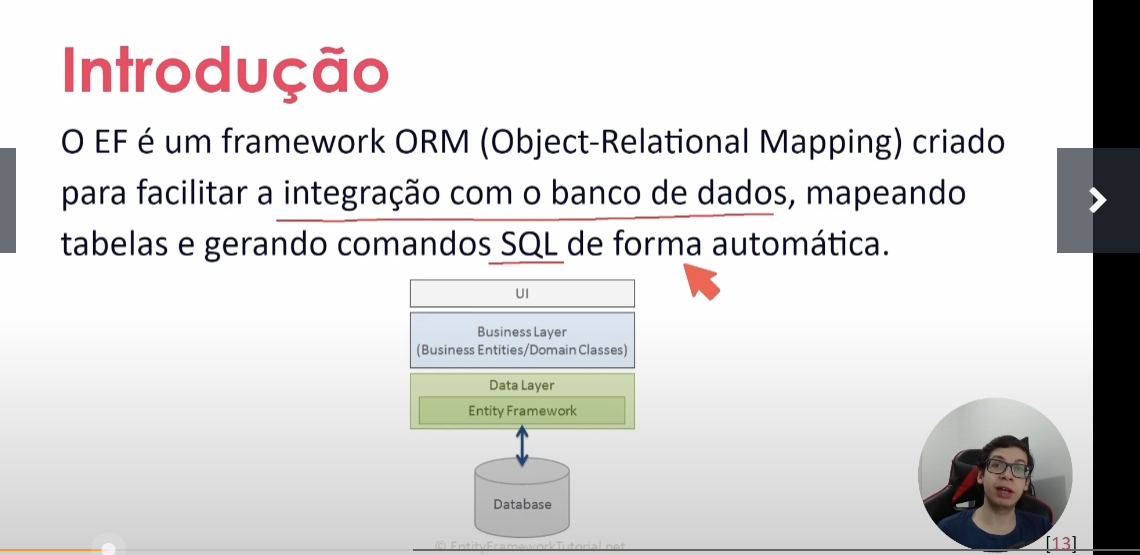
 uma controller nada mais é do que uma classe que vai armazenar suas requisições HTTP e vai disponibilizar seus end point, por exemplo, API que atualizo meus usuários eu vou criar um usuariocontroller( tudo que eu fizer com relação a usuário eu vou fazer na minha usuário controller) se fosse produtos, seriam em outro controller por exemplo produtocontroller. Um controller pode ser entendimento como um agrupamentos de classes em comum, de domínios em comum.

Quando for criar a classe, precisa ter controller no final para ser fácil manutenção do código e identificar a função.



[ApiControll] para identificar que é uma APi controller

Entiry Frameworks com c#



Integração com banco de dados exemplo de pesquisa

C# crud banco de dados sql

C# crud banco de dados Mysql

Instalando um entity framework

Executar o seguinte comando apenas uma vez no visual studio

dotnet tool install --global dotnet-ef

Instalar o pacote em todo projeto( A cada novo projeto precisa ser instalado esse pacote)

dotnet add package Microsoft.EntityFrameworkCore.Design

Pacote SQL server( Instalar a cada novo projeto que utiliza sql server)

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

ou ( caso seja mysql)

dotnet add package Microsoft.EntityFrameworkCore.mysql

criar uma classe contexto é uma classe que centraliza todas as nossas informações em determinado banco de dados





Appsetting.development, usado apenas para desenvolvimento

Appsettings.jason usado no ambiente de produção

"Server=localhost\\sqlexpress;Initial Catalog=Agenda; Integrated Security=True"

ConnectionStrings": de conexão

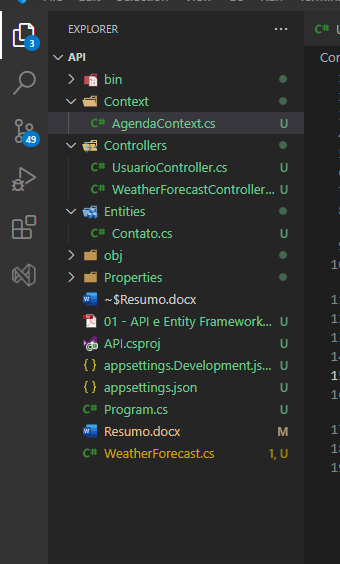
  "ConnectionStrings": {

    "ConexãoPadrão":"Server=localhost\\sqlexpress;Initial Catalog=Agenda; Integrated Security=True"

  }

Catalog=Agenda Banco de dados a ser criado pelo EntityFramework automaticamente

; Integrated Security=True" significa que vamos utilizar a nossa autenticação do Windows, caso esteja trabalhando com sistema de verdade que tenha usuário e senha e onde vai passar.



Appsettings.development.json ( exemplo de conexão com banco de dados)

{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft.AspNetCore": "Warning"

    }

  },

  "ConnectionStrings": {

    "ConexãoPadrão":"Server=localhost\\sqlexpress;Initial Catalog=Agenda; Integrated Security=True"

  }

}

Conceito Migrations

Toda criação de tabela eu tenho que pedir para o entity de maneira antecimada, isso é migrations , é um mapeamentos das nossas classes para poder transformar eles em tabela.

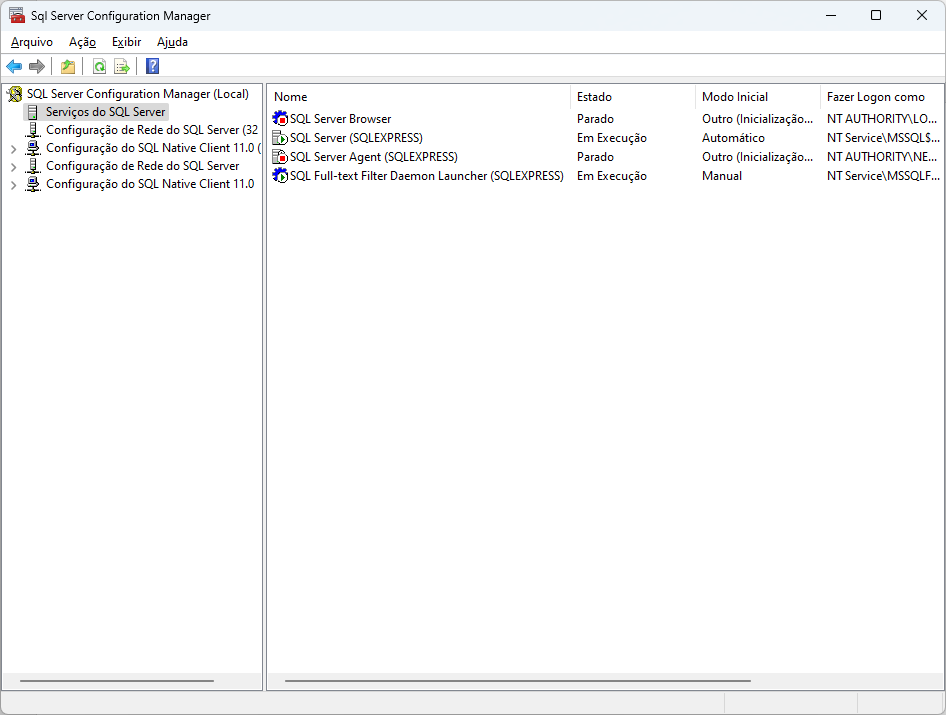
Tudo que tiver

public DbSet<Contato> Contatos{get;set;}

é uma tabela a ser criada n

o banco de dados

verificar se o sqlserver está rodando normalmente



Agora va no terminal e rodar o seguinte comando

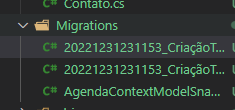
Para criar um migrations rodar o comando

Migrations ferramenta que vai atualizar o banco de dados, add(Adicionar)

dotnet-ef migrations add Nome da migrations ( seja descritiva Exemplo, cliente , contatos)

dotnet-ef migrations add CriaçãoTabelaContato

Automaticamente ele criou uma pasta Migrations



Após criação dos arquivos é necessário criar as tabelas

dotnet-ef database update

Adicionado o comando Encrypt=False para solucionar erro de conexão com o banco de dados sem ser encriptado.

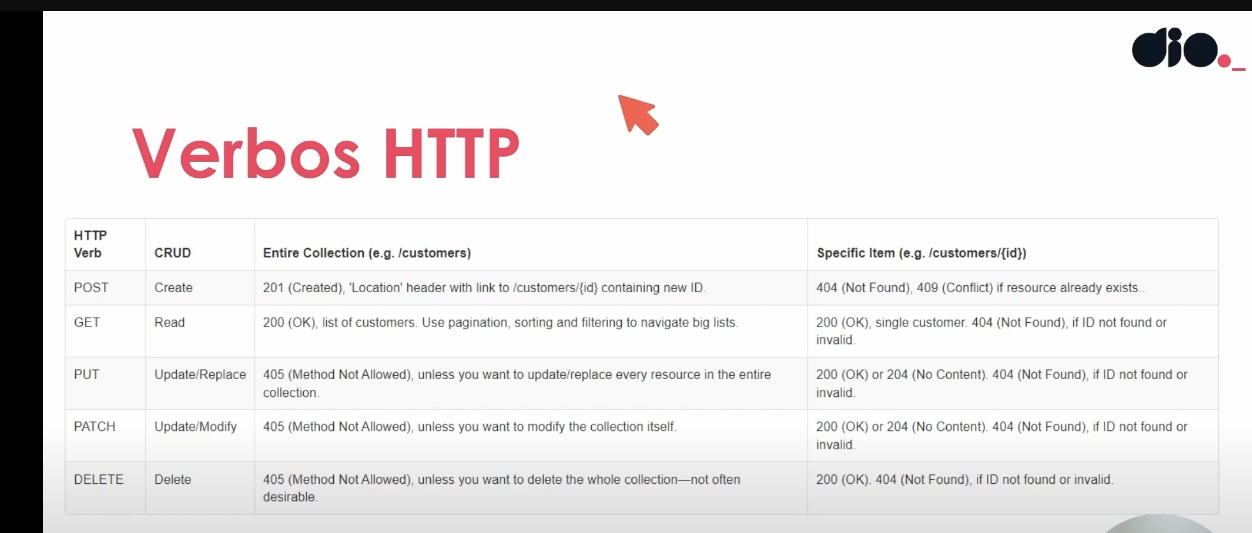
 "ConnectionStrings": {

    "ConexãoPadrão":"Server=localhost\\sqlexpress;Initial Catalog=Agenda;Encrypt=False ; Integrated Security=True "

  }

dotnet watch run

verbos HTTP – significam ações que vao ocorrer na sua API



CRUD SIGNIFICADO - CREATE, READ, UPDATE, DELETE

**POST** – É PARA CRIAR UM NOVO RECURSO OU NOVA INFORMAÇÃO NA API

**GET** – OBTER ALGUMA INFORMAÇÃO – RETORNO DE ALGUMA INFORMAÇÃO, EX id

**PUT** – ESTA RELACIONADO A UMA ATUALIZAÇÃO, ATUALIZAR UM RECURSO RESISTENTE

**PATCH** – ESTA RELACIONADO COM ATUALIZAÇÃO E MODIFICAÇÕES, DIFERENÇA DE PUT PARA PATCH- PUT EU PASSO UMA INFORMAÇÃO COMPLETA O PATCH EU PASSO UMA INFORMAÇÃO PARCIAL, O PATCH EU POSSO ATUALIZAR UM ITEM OU MAIS , POR EXEMPLO TELEFONE, O PUT TEM QUE PASSAR A SUA ENTIDADE INTEIRA

**DELETE** – ELE VAI DELETAR ALGUM RECURSO RESISTENTE

**Recaptular o funcionamento da API – passo a passo**

**Appsettings.development.js**

**Ele precisa ter essa string de conexão**

**Precisa ter obrigatoriamente o** "ConnectionStrings":

"ConexãoPadrão":" pode ser o nome que deseja como padrão

  "ConnectionStrings": {

    "ConexãoPadrão":"Server=localhost\\sqlexpress;Initial Catalog=Agenda;Encrypt=False ; Integrated Security=True "

  }

**Temos também a pasta entidade - Contato**

****

namespace API.Entities

{

    public class Contato

    {

        public int Id { get; protected set; } //Utilizado protected set para o Entity reconhecer como uma ID

        public string Nome { get; set; }

        public string Telefone { get; set; }

        public bool Ativo { get; set; }

    }

**O Contato é nossa entidade ou seja é uma classe uma classe que vamos trabalhar na API e ao mesmo tempo é uma tabela do banco de dados, a classe Contatos tem todos atributos para criar uma tabela no banco de dados.**

**Nos temos também um Context - AgendaContext**

****

namespace API.Context

{

    public class AgendaContext :DbContext // importar a classe dbcontext

    {

        public AgendaContext(DbContextOptions<AgendaContext>options) : base(options) // aqui onde recebe a conexão com banco

        //faz a ligação do banco de dados, vai passar para calsse pai do DbContext

        {

        }

        public DbSet<Contato> Contatos{get;set;} // Dbset ele esta representado por uma classe e tb representado por uma tabela do banco de dados

    }

}

public class AgendaContext :DbContext – QUE HERDA DO DbContext – ou seja esse nosso contexto é a nossa classe que vai acessar o nosso banco de dados é onde utilizaremos para chamar a conexão com o banco por isso tem que herdar de DbContext e também precisa ter o construtor abaixo:

**public AgendaContext(DbContextOptions<AgendaContext>options) : base(options)**

**Recebendo um options e passando para classe** DbContext

{

}

Por sua vez a Dbcontext tem varias propriedade por exemplo do tipo dbset

public DbSet<Contato> Contatos{get;set;}

Ou seja vou ter um dbset do tipo contato que eu chamei de **Contatos**

**Por exemplo eu vou acessar os registro da minha tabela Contato**

**Se a classe não estiver com dbset ela não vai ser indentificado como tabela, ela precisa ser definida no contexto**

**Por sua vez acessamos o Progran.cs**

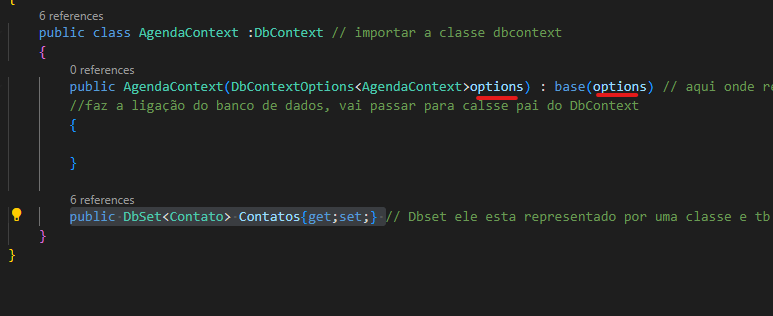


Para passar a configuração para o dbcontext, ou seja ele precisa de uma conexão do banco de dados para se conectar

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

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

utilizamos builder.Services.AddDbContext do tipo <AgendaContext> ( que é o nosso contexto agenda que criamos) e passamos o options, o nosso options ele vai receber em



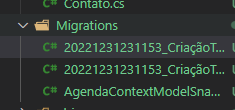
Então eu estou falado para o sistema Options utilize sql server ( options.UseSqlServer) porque estou utilizando banco de dados sql server então foi colocado builder.Configuration.GetConnectionString("ConexãoPadrão")

Que é a nossa conexão padrão criada em appsettins.development.json

Por sua vez criamos as nossas **Migrations**

**Exemplo de criação**

**dotnet-ef migrations add CriaçãoTabelaContato**



Temos uma migrations que nada mais é um código para espelhar as configurações no banco de dados.

Eu tenho uma entidade chamada contato que por sua vez está como um dbset na minha AgendaContext.

Como meu banco de dados é novos do zero, eu não tenho a tabela Contatos, então para isso o ideal é que não crie tabela com um comando SQl, mas sim deixe o Entity framework criar para você, porque assim ele tem um maior controle das alterações das tabelas.

Para que eu possa criar a tabela de contatos, eu vou utilizar a Migrations

        protected override void Up(MigrationBuilder migrationBuilder) // metodo UP - aplicar mudança nos bancos de dados

        {

            migrationBuilder.CreateTable(

                name: "Contatos",

                columns: table => new

                {

                    Id = table.Column<int>(type: "int", nullable: false)

                        .Annotation("SqlServer:Identity", "1, 1"),

                    Nome = table.Column<string>(type: "nvarchar(max)", nullable: true),

                    Telefone = table.Column<string>(type: "nvarchar(max)", nullable: true),

                    Ativo = table.Column<bool>(type: "bit", nullable: false)

                },

                constraints: table =>

                {

                    table.PrimaryKey("PK\_Contatos", x => x.Id);

                });

        }

Para aplicar a migrations no banco de dados é utilizar o comando de atualização:

**dotnet-ef database update**

Foi intalado o Dotnet Framework



   <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="7.0.1">

      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>

      <PrivateAssets>all</PrivateAssets>

    </PackageReference>

    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="7.0.1" />

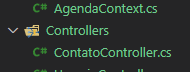
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.2.3" />

Instalado dois pacotes Microsoft.EntityFrameworkCore.SqlServer" e o Designer ="Microsoft.EntityFrameworkCore.Design" , esses pacotes precisam instalar em cada projeto que vai utilizar dotnetframework

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

por ultimo nos temos as nossas controller



Controller nada mais é do que ponto de entrada que nos vamos disponibilizar os nossos métodos, uma controller vai ficar dentro da pasta Controller , vai o final Controller e também ela vai herdar de ControllerBase. E vai ter os atributos APIController e Route

 [ApiController]  // serve para indentificar que é uma APi controller

    [Route("[controller]")] // requisito para criação de API

    public class ContatoController :ControllerBase

e na nossa controller vamos receber no nosso construtor o nosso AgendaContext

ou seja esse contexto que permite acessar o banco de dados

private readonly AgendaContext \_context;

ou seja eu tenho uma propriedade chamada \_context do tipo AgendaContext

vou receber do meu construtor

        public ContatoController(AgendaContext context)

        {

            \_context = context;

        }

Eu vou passar do construtor para \_context ( \_context = context;))

Lembrando o que estou recebendo é sem o underline public ContatoController(AgendaContext context) e que esta na minha classe é com o underline private readonly AgendaContext \_context;

Eu tenho que receber via construtor isso se chama injeção de depedencia

Por fim foi criado vários métodos que retorna um ActionsResult ou seja um resulto HTTP

E temos os verbos

 [HttpGet("{id}")] // estou informando que ele vai receber uma ID

 [HttpPost]// porque estamos enviando uma informação

 [HttpGet("ObterPorNome")]

Eles precisam ser descritivos

public IActionResult ObterPorID(int id)

public IActionResult ObterPorNome(string nome)

em todos eles é chamado \_context

exemplo

            \_context.Contatos.Update(contatoBanco);

            \_context.SaveChanges();

Para todos eles com excessao do get vai utilizar o \_context.SaveChanges(); se não não vai atualizar o banco de dados

Por sua vez tem o tipo de retorno se foi sucesso ou não, se achou algo ele faz a devida atualização incluindo o savechanges.