

Programação .NET

Prof. Ricardo Frohlich da Silva

ASP.NET Core MVC

- É uma estrutura específica dentro do ASP.NET Core que segue o padrão de arquitetura MVC.
- É projetado especificamente para o desenvolvimento de aplicativos web que seguem o padrão MVC.
- Fornece componentes específicos para Model, View e Controller, facilitando a organização e manutenção do código.

ASP.NET Core MVC

- O padrão MVC (Model-View-Controller) é um padrão de arquitetura de software amplamente utilizado para desenvolvimento de aplicativos web.
- Ele divide uma aplicação em três componentes principais: Model, View e Controller, cada um com responsabilidades distintas.

A Model

- O Model representa a estrutura de dados do aplicativo.
- Ele é responsável pela lógica de negócios e pela interação com o banco de dados.
- Em ASP.NET Core, um modelo pode ser uma classe simples que representa uma entidade de banco de dados. Por exemplo, considere um modelo de Produto:

```
public class Produto
{
    0 referências
    public int Id { get; set; }
    0 referências
    public string Nome { get; set; }
    0 referências
    public decimal Preco { get; set; }
}
```

A View

- A View é responsável pela apresentação da interface do usuário.
- Ela exibe os dados do modelo aos usuários e também recebe entradas do usuário.
- Em ASP.NET Core, as visualizações são criadas usando o mecanismo de marcação Razor, que combina HTML com código C#.

A View

- Por exemplo, uma visualização para exibir detalhes de um produto pode ser criada assim:

```
@model Produto

<h2>Detalhes do Produto</h2>
<div>
  <h4>@Model.Nome</h4>
  <p>Preço: @Model.Preco</p>
</div>
```

A Controller

- O Controller é responsável por manipular as solicitações do cliente, interagir com os modelos e selecionar as visualizações apropriadas para retornar ao navegador do cliente.
- Em ASP.NET Core, os controladores são classes que herdam da classe Controller.
- Cada ação do controlador corresponde a uma solicitação HTTP.

A Controller

- Por exemplo, um controlador para lidar com operações de produtos pode ser definido assim:

```
public class ProdutoController : Controller
{
    private readonly ProdutoService _produtoService;

    0 referências
    public ProdutoController(ProdutoService produtoService)
    {
        _produtoService = produtoService;
    }

    0 referências
    public IActionResult Details(int id)
    {
        var produto = _produtoService.GetProdutoPorId(id);
        if (produto == null)
        {
            return NotFound();
        }
        return View(produto);
    }
}
```


A Controller

- Neste exemplo, o controlador ProdutoController possui uma ação Details que recebe um identificador de produto como parâmetro.
- Ele utiliza um serviço ProdutoService para obter os detalhes do produto com o identificador fornecido e retorna a visualização correspondente com os detalhes do produto.

```
public class ProdutoController : Controller
{
    private readonly ProdutoService _produtoService;

    0 referências
    public ProdutoController(ProdutoService produtoService)
    {
        _produtoService = produtoService;
    }

    0 referências
    public IActionResult Details(int id)
    {
        var produto = _produtoService.GetProdutoPorId(id);
        if (produto == null)
        {
            return NotFound();
        }
        return View(produto);
    }
}
```

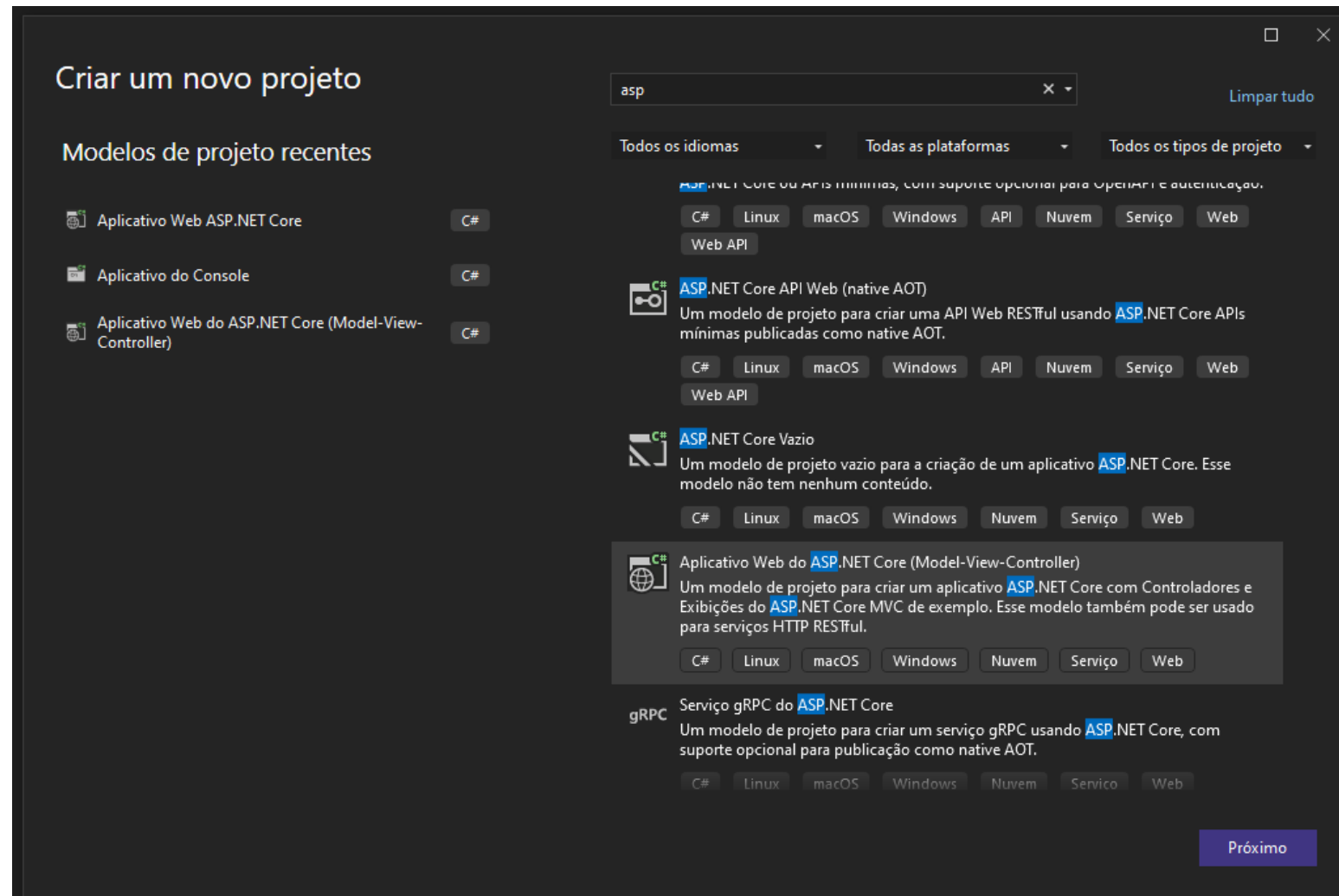
MVC

- O padrão MVC separa a aplicação em três componentes principais, cada um com responsabilidades distintas:
 - Model: Representa a estrutura de dados e a lógica de negócios.
 - View: Responsável pela apresentação da interface do usuário.
 - Controller: Responsável por manipular as solicitações do cliente e coordenar as interações entre o modelo e a visualização.
- Ao utilizar o padrão MVC em ASP.NET Core, você pode criar aplicativos web de forma organizada e escalável, mantendo uma separação clara entre a lógica de negócios, a apresentação e o controle de fluxo. Isso facilita a manutenção do código e o desenvolvimento de novos recursos.

Criando primeiro projeto

- No Visual Studio, clique em "Arquivo" > "Novo" > "Projeto".
- No Visual Studio, na janela "Novo Projeto", escolha "ASP.NET Core (Model-View-Controller)".

Criando primeiro projeto



Entity Framework

- O Entity Framework (EF) é um framework de mapeamento objeto-relacional (ORM) desenvolvido pela Microsoft.
- Sua função principal é permitir que os desenvolvedores trabalhem com dados em um banco de dados usando objetos específicos de domínio, eliminando a necessidade de escrever diretamente consultas SQL.
- Ele mapeia entidades de banco de dados para classes .NET e vice-versa, facilitando a interação entre o banco de dados e o código da aplicação.

Entity Framework

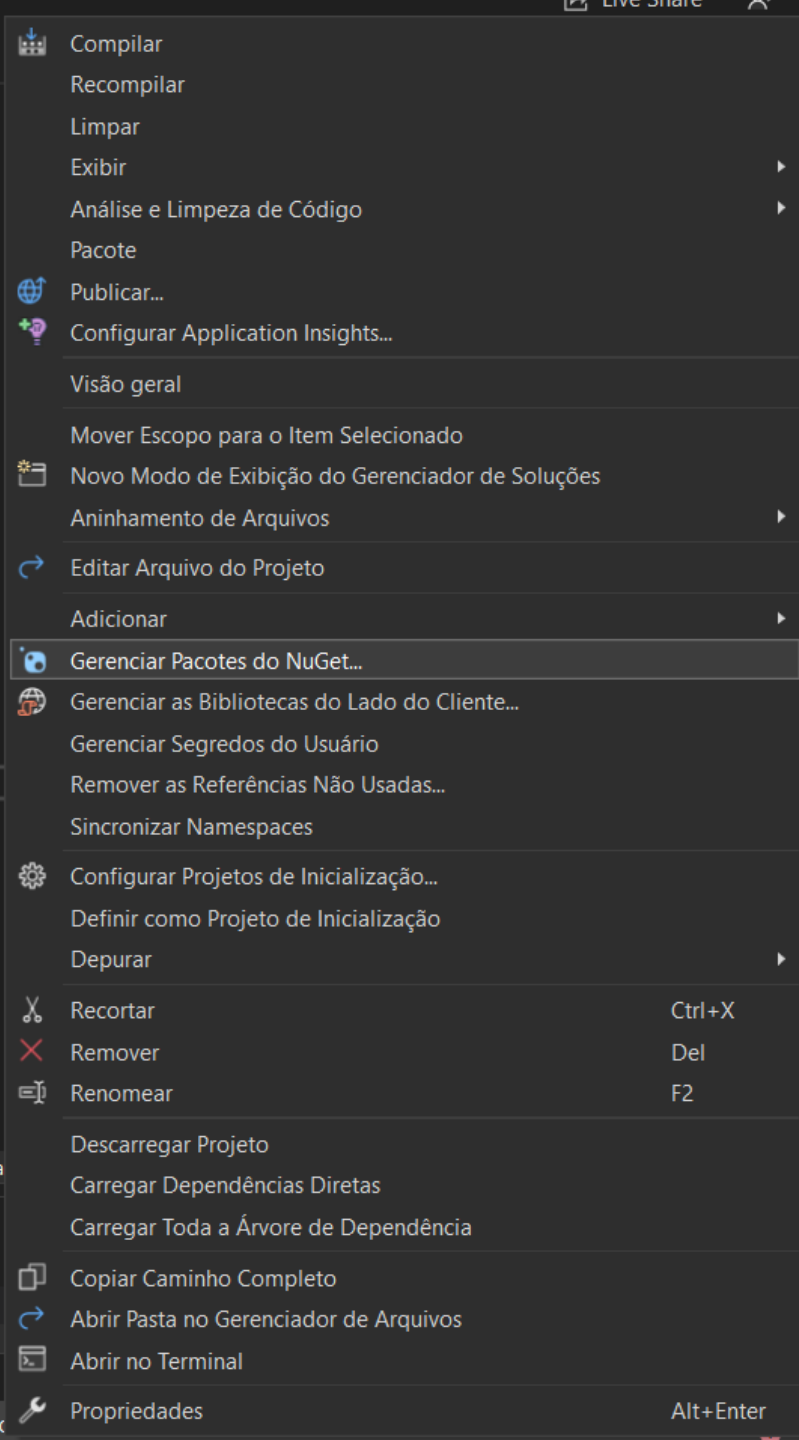
- Componentes Principais de uma Aplicação EF Core:
 - DbContext: A classe que gerencia as conexões com o banco de dados e o mapeamento das entidades para as tabelas.
 - Entidades: Classes que representam as tabelas do banco de dados.
 - Migrations: Uma forma de versionar e aplicar alterações no esquema do banco de dados, controlando o histórico das mudanças.

Entity Framework

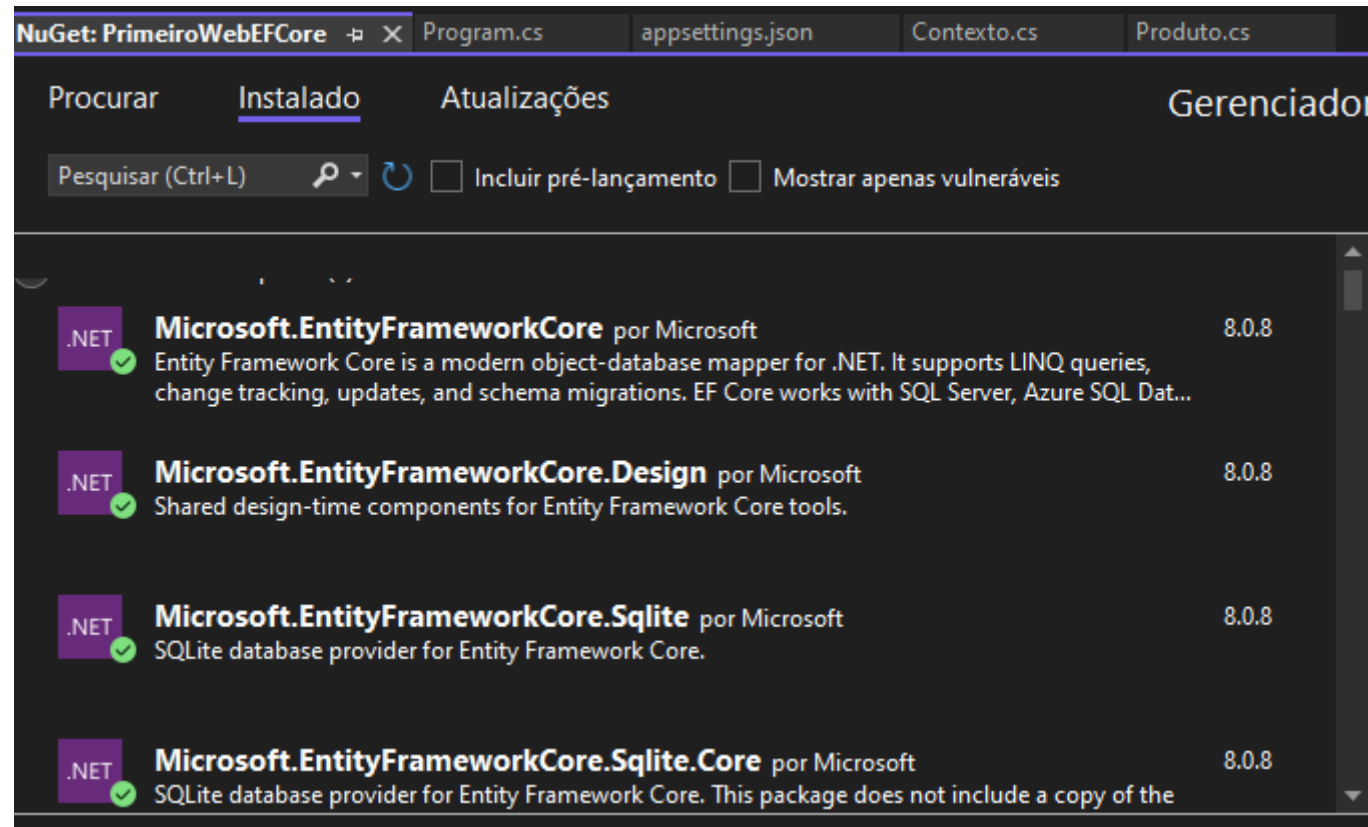
- Exemplo de uma Aplicação EF Core:
 - Projeto: Gerenciamento de Produtos.
 - Entidade: Produto (com propriedades Id, Nome, Preço e Categoria).
 - DbContext: Classe que gerencia a conexão e o mapeamento das entidades para o banco de dados.

NuGet

- Gerenciador de pacotes para a plataforma de desenvolvimento .NET da Microsoft.
- Facilita a instalação, atualização e gerenciamento de bibliotecas e ferramentas em projetos .NET.
- O NuGet automatiza o processo de baixar e configurar dependências de projetos, permitindo que os desenvolvedores adicionem funcionalidades extras sem precisar configurar manualmente bibliotecas de terceiros.



Entity Framework



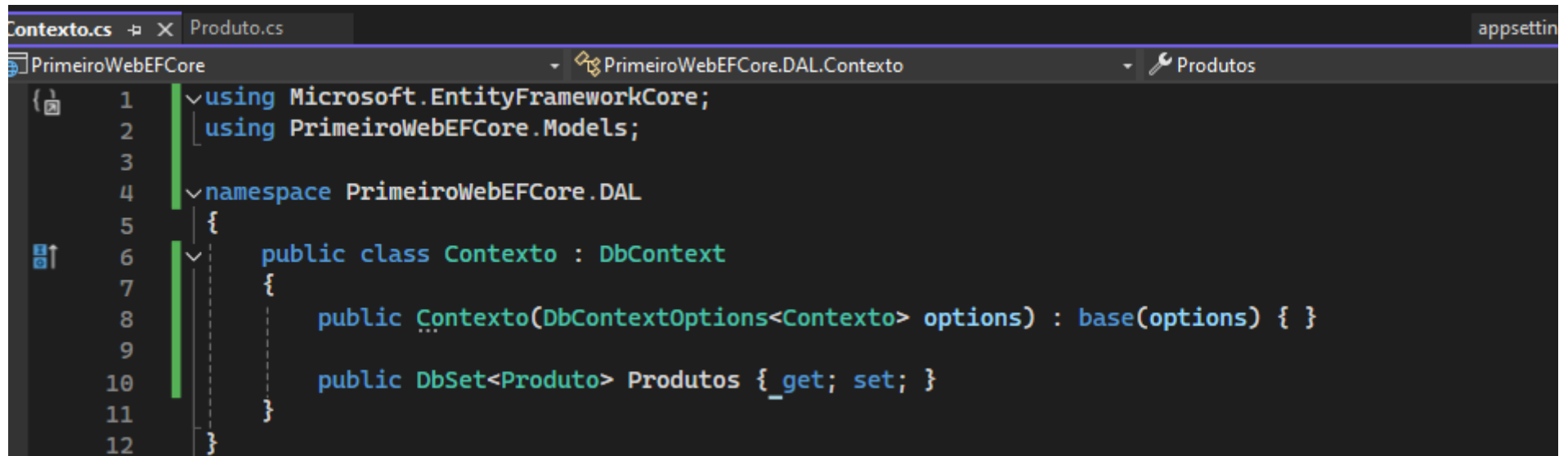
Entity Framework

```
public class Produto
{
    [Key] // Define explicitamente que 'Id' é a chave primária
    public int Id { get; set; }

    [Required(ErrorMessage = "O nome do produto é obrigatório")]
    [StringLength(100, ErrorMessage = "O nome do produto deve ter no máximo 100 caracteres")]
    public string Nome { get; set; }

    [Required(ErrorMessage = "O preço do produto é obrigatório")]
    [Range(0.01, 9999.99, ErrorMessage = "O preço deve estar entre 0.01 e 9999.99")]
    [Column(TypeName = "decimal(10, 2)")] // Configura a precisão do campo no banco de dados SQLite
    public decimal Preco { get; set; }
}
```

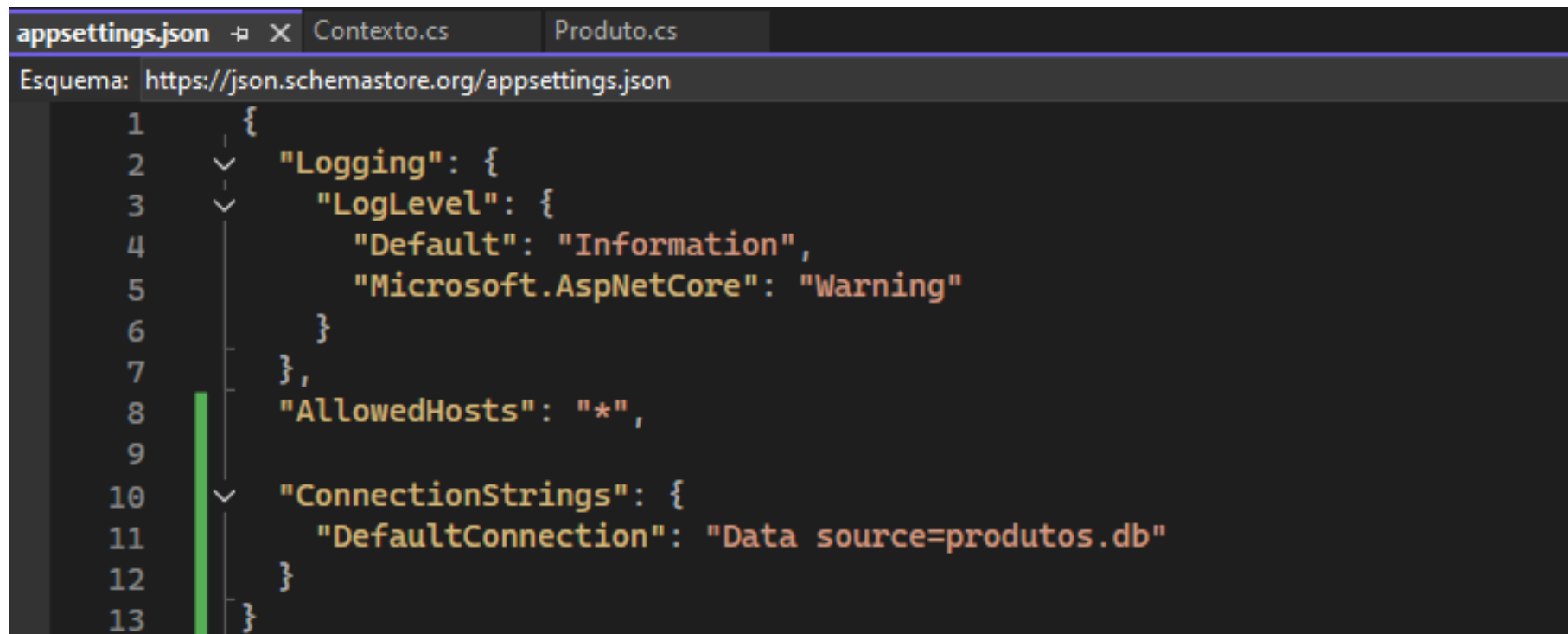
Entity Framework



```
Contexto.cs x Produto.cs appsetin
PrimeiroWebEFCore
PrimeiroWebEFCore.DAL.Contexto
Produtos

1  using Microsoft.EntityFrameworkCore;
2  using PrimeiroWebEFCore.Models;
3
4  namespace PrimeiroWebEFCore.DAL
5  {
6      public class Contexto : DbContext
7      {
8          public Contexto(DbContextOptions<Contexto> options) : base(options) { }
9
10         public DbSet<Produto> Produtos { _get; set; }
11     }
12 }
```

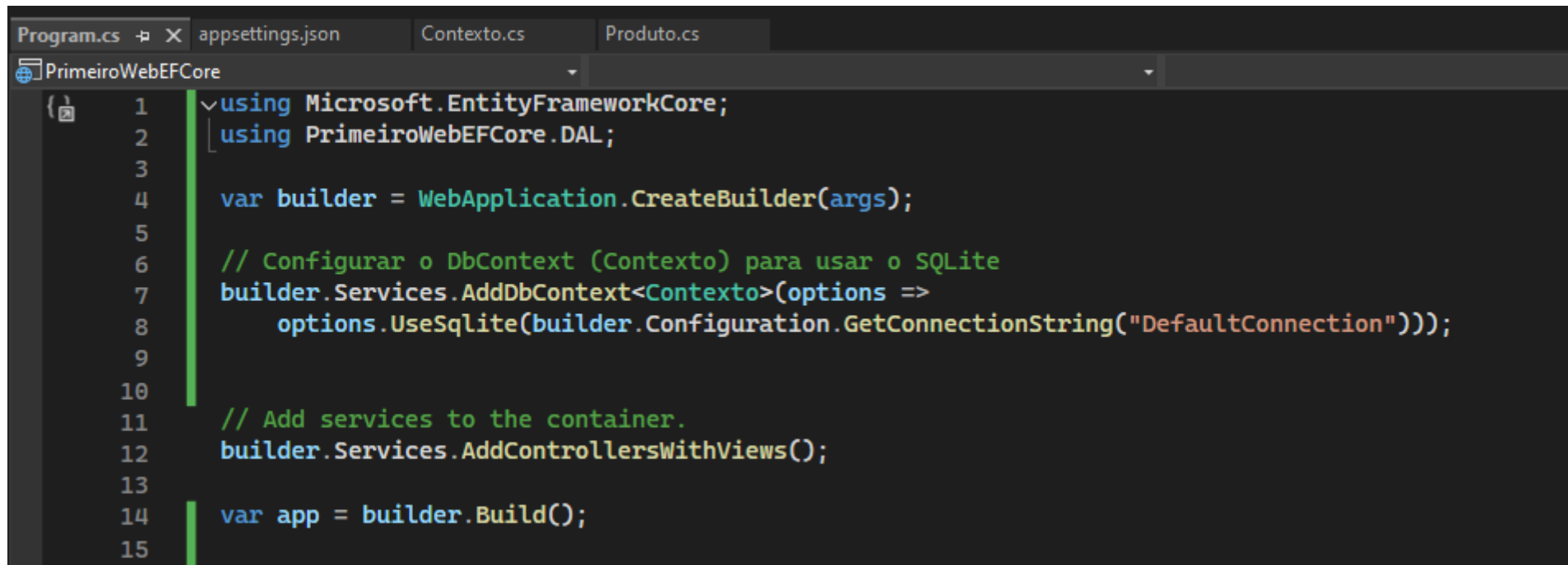
Entity Framework



The screenshot shows a code editor with three tabs: `appsettings.json`, `Contexto.cs`, and `Produto.cs`. The `appsettings.json` tab is active, displaying a JSON configuration file. Above the JSON content, it shows the schema reference: `Esquema: https://json.schemastore.org/appsettings.json`. The JSON content is as follows:

```
1  {
2    ∨  "Logging": {
3      ∨    "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9
10   ∨  "ConnectionStrings": {
11     "DefaultConnection": "Data source=produtos.db"
12   }
13 }
```

Entity Framework



```
Program.cs  X appsettings.json Contexto.cs Produto.cs
PrimeiroWebEFCore
1  using Microsoft.EntityFrameworkCore;
2  using PrimeiroWebEFCore.DAL;
3
4  var builder = WebApplication.CreateBuilder(args);
5
6  // Configurar o DbContext (Contexto) para usar o SQLite
7  builder.Services.AddDbContext<Contexto>(options =>
8      options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
9
10
11 // Add services to the container.
12 builder.Services.AddControllersWithViews();
13
14 var app = builder.Build();
15
```

Entity Framework

```
C:\Users\ricar\source\repos\PrimeiroEFMVC>dotnet tool install --global dotnet-ef
Você pode invocar a ferramenta usando o comando a seguir: dotnet-ef
A ferramenta 'dotnet-ef' (versão '8.0.8') foi instalada com êxito.
```

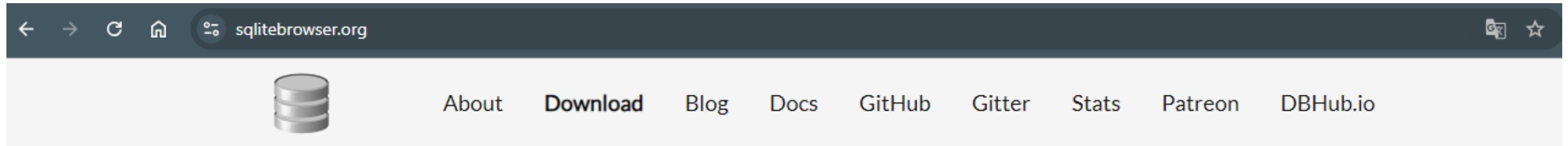
```
PS C:\Users\ricar\source\repos\PrimeiroWebEFCore\PrimeiroWebEFCore> dotnet add package Microsoft.EntityFrameworkCore.Design
```

```
PS C:\Users\ricar\source\repos\PrimeiroWebEFCore\PrimeiroWebEFCore> dotnet ef migrations add InitialCreate
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'
PS C:\Users\ricar\source\repos\PrimeiroWebEFCore\PrimeiroWebEFCore> dotnet ef database update
Build started...
Build succeeded.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
```

Entity Framework

```
PS C:\Users\ricar\source\repos\PrimeiroWebEFCore\PrimeiroWebEFCore> dotnet ef database update
Build started...
Build succeeded.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (29ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      PRAGMA journal_mode = 'wal';
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (33ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE "__EFMigrationsHistory" (
        "MigrationId" TEXT NOT NULL CONSTRAINT "PK__EFMigrationsHistory" PRIMARY KEY,
        "ProductVersion" TEXT NOT NULL
      );
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT COUNT(*) FROM "sqlite_master" WHERE "name" = '__EFMigrationsHistory' AND "type" = 'table';
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT "MigrationId", "ProductVersion"
      FROM "__EFMigrationsHistory"
```


Entity Framework

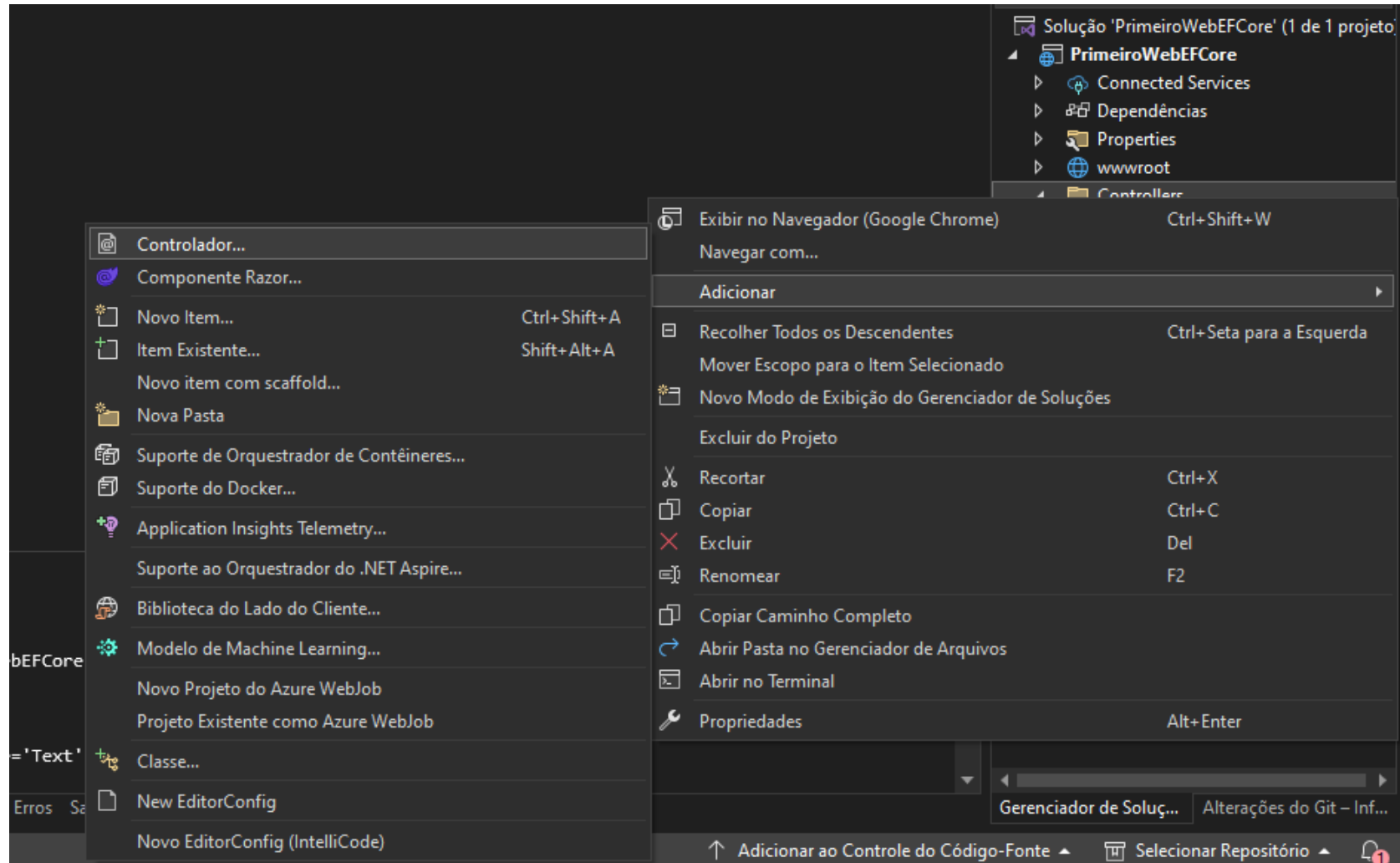


docs [Wiki](#) chat [on gitter](#) downloads [9.9M](#) Qt [qmake](#) [coverity](#) [passing](#) donate [Patreon](#)

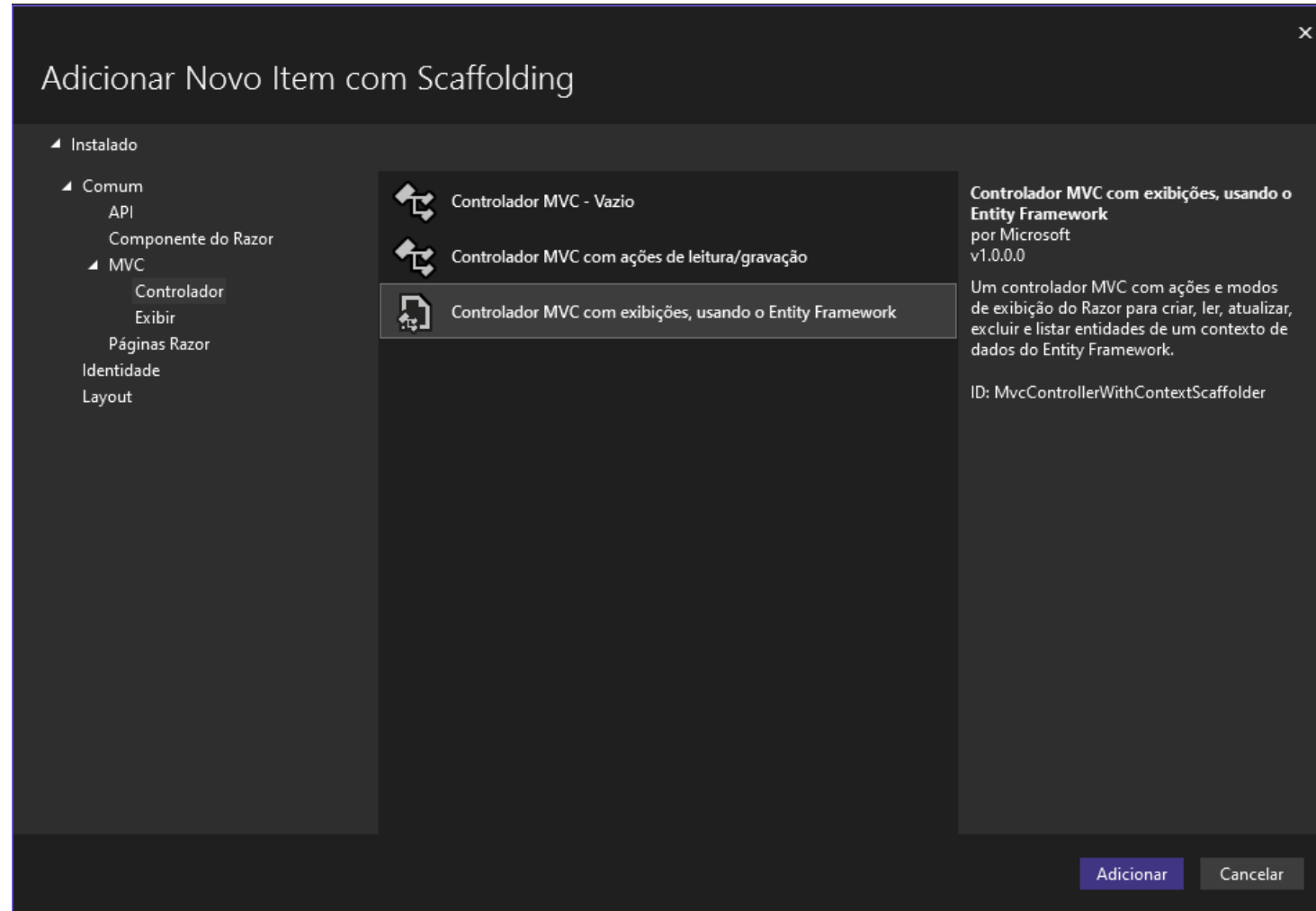
DB Browser for SQLite

DB Browser for SQLite (DB4S) is a high quality, visual, [open source](#) tool designed for people who want to create, search, and edit [SQLite](#) database files. DB4S gives a familiar spreadsheet-like interface on the database in addition to providing a full SQL query facility. It works with [Windows](#), [macOS](#), and most versions of [Linux](#) and [Unix](#). Documentation for the program is on the [wiki](#).

Entity Framework



Entity Framework



Entity Framework

×

Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo

Produto (PrimeiroWebEFCore.Models)

Classe DbContext

Contexto (PrimeiroWebEFCore.DAL)

+

Provedor de banco de dados

Configurado a partir do DbContext selecionado

Exibições

☒ Gerar modos de exibição

☒ Bibliotecas de scripts de referência

☒ Usar uma página de layout

...

(Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

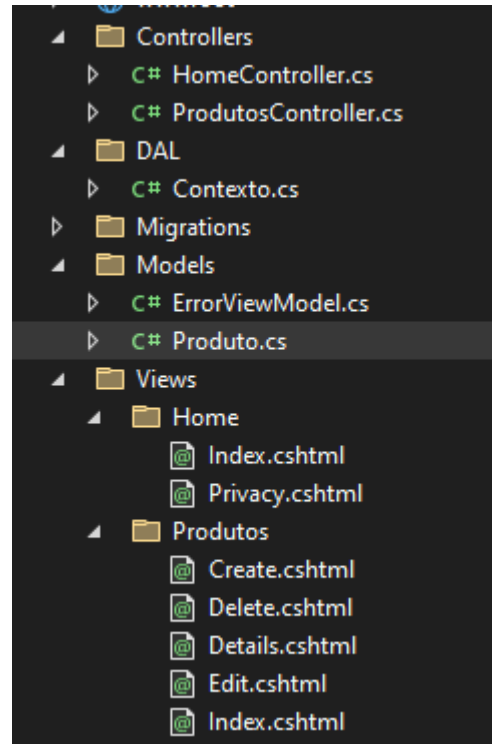
Nome do controlador

ProdutosController

Adicionar

Cancelar

Entity Framework



Entity Framework

```
Edit.cshtml  site.js  Program.cs  Produto.cs  ProdutosController.cs  Create.cshtml  _Layout.cshtml
PrimeiroWebEFCore
7  <h1>Cadastro de Produto</h1>
8
9  <hr />
10 <div class="row">
11   <div class="col-md-4">
12     <form asp-action="Create">
13       <div asp-validation-summary="ModelOnly" class="text-danger"></div>
14       <div class="form-group">
15         <label asp-for="Nome" class="control-label"></label>
16         <input asp-for="Nome" class="form-control" />
17         <span asp-validation-for="Nome" class="text-danger"></span>
18       </div>
19       <div class="form-group">
20         <label asp-for="Preco" class="control-label"></label>
21         <input asp-for="Preco" class="form-control" id="Preco" type="number" step="0.01" />
22         <span asp-validation-for="Preco" class="text-danger"></span>
23       </div>
24       <div class="form-group">
25         <input type="submit" value="Cadastrar" class="btn btn-primary" />
26       </div>
27     </form>
28   </div>
29 </div>
```

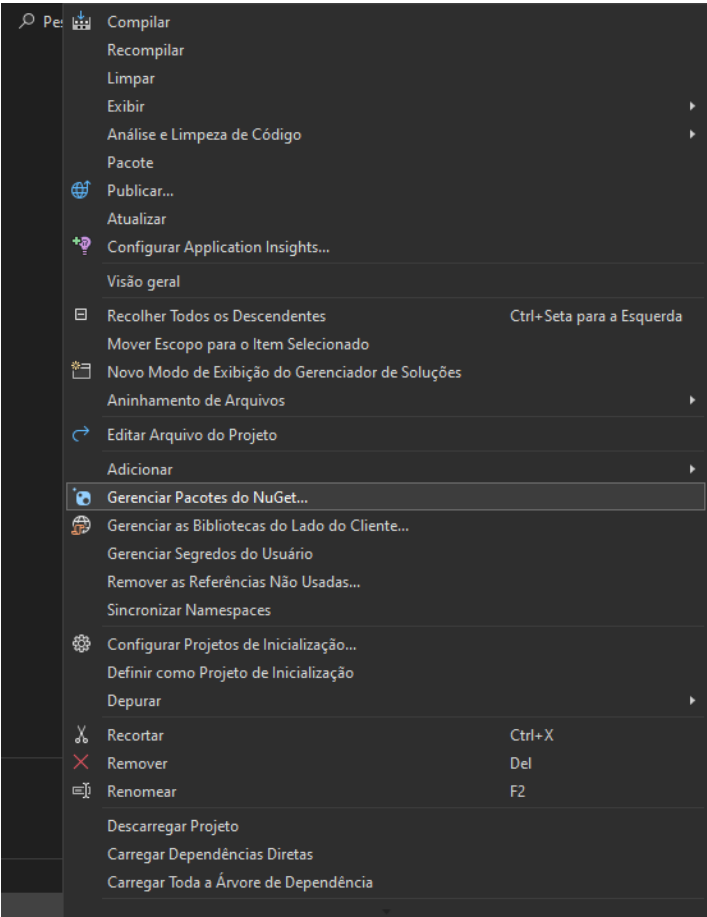
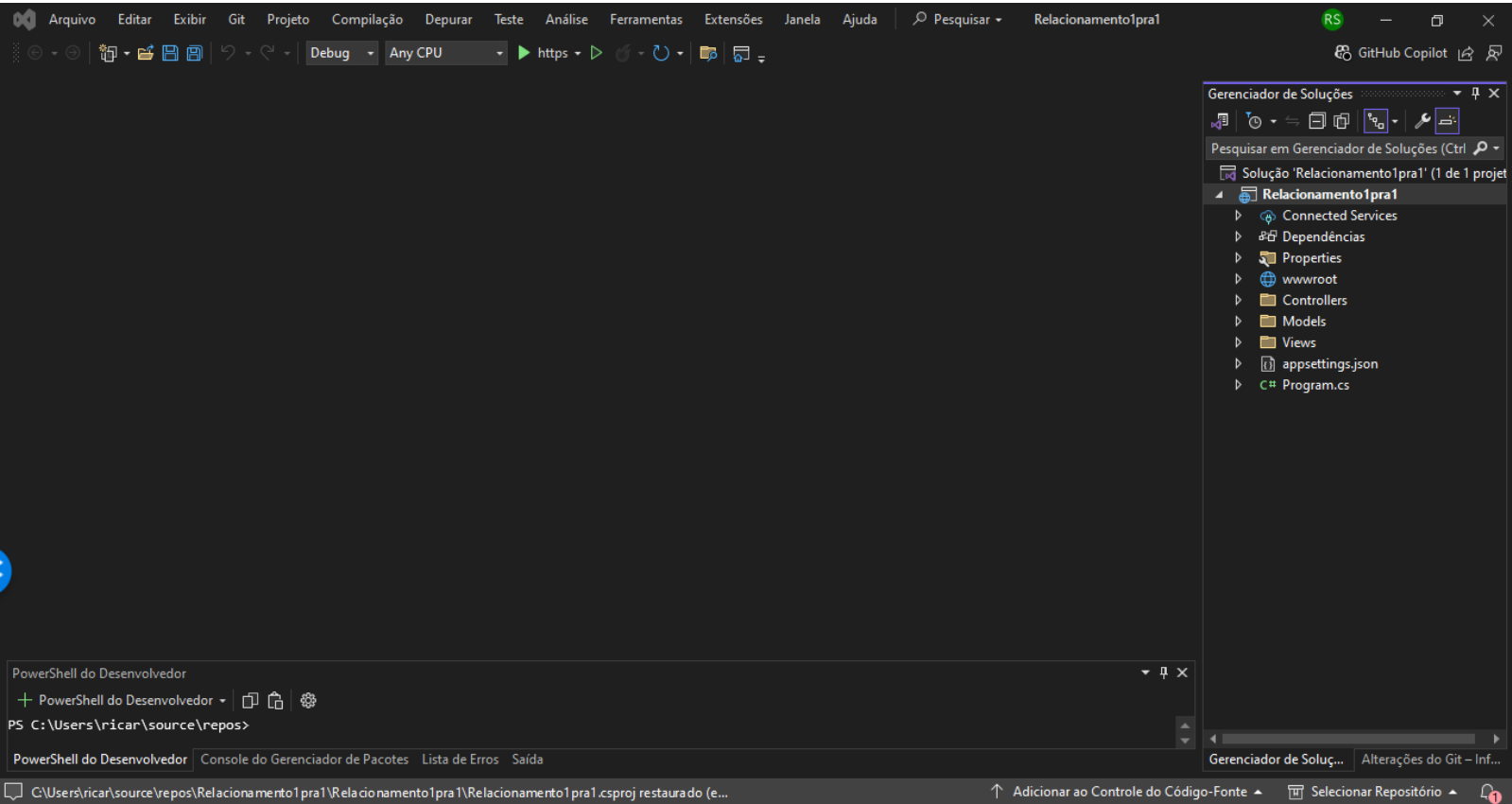
Relacionamento entre entidades

- O Entity Framework (EF) Core suporta três tipos principais de relacionamentos entre entidades, que refletem as associações comuns em modelos de banco de dados relacionais:
 - um-para-um (1:1)
 - um-para-muitos (1) e;
 - muitos-para-muitos (N)
- Cada um desses relacionamentos é implementado de maneiras diferentes no EF Core e depende de como as entidades estão modeladas.

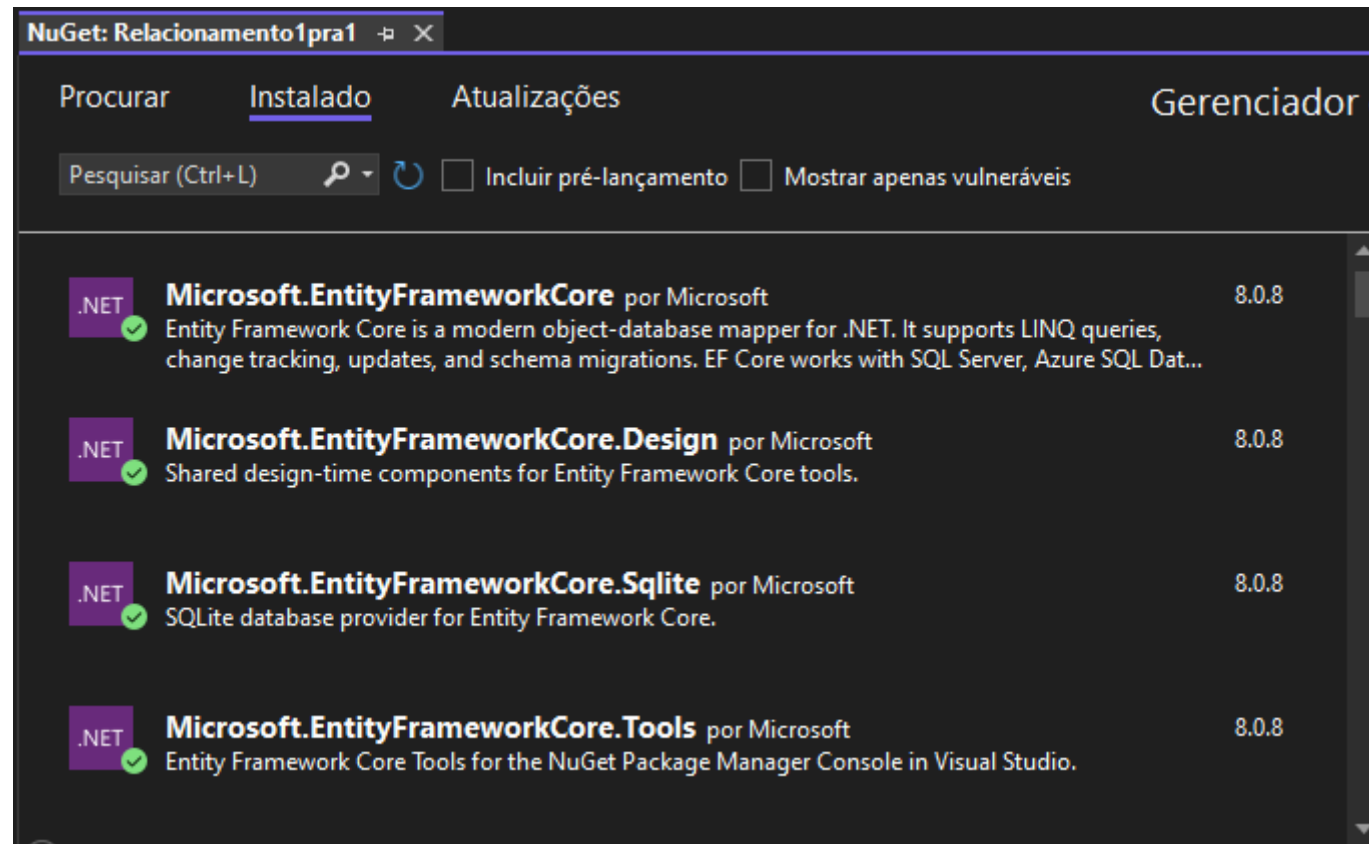
Relacionamento entre entidades

- Relacionamento Um-para-Um (1:1)
 - No relacionamento um-para-um, cada entidade em uma tabela está associada a exatamente uma entidade em outra tabela.
- Exemplo: Vamos supor que você tenha uma entidade Pessoa e uma entidade Endereco, onde cada pessoa possui exatamente um endereço.

Relacionamento entre entidades



Relacionamento entre entidades



Relacionamento entre entidades

- Classe Cliente

```
public class Pessoa
{
    [Key]
    public int PessoaId { get; set; }
    [Required (ErrorMessage = "O nome do cliente é obrigatório")]
    [StringLength(100, ErrorMessage = "O nome do cliente deve ter no máximo 100 caracteres")]
    public string Nome { get; set; }
    //Fazendo relacionamento utilizando Data Annotations
    public Endereco Endereco { get; set; }
}
```

Relacionamento entre entidades

- Classe Endereco

```
public class Endereco
{
    [Key]
    [ForeignKey("Pessoa")]
    public int EnderecoId { get; set; }
    [Required(ErrorMessage = "O nome do cliente é obrigatório")]
    [StringLength(100, ErrorMessage = "O nome do cliente deve ter no máximo 100 caracteres")]
    public string Rua { get; set; }
    public int Nro { get; set; }
    public string Complemento { get; set; }
    [Required(ErrorMessage = "O nome da cidade é obrigatório")]
    [StringLength(100, ErrorMessage = "O nome da cidade deve ter no máximo 100 caracteres")]
    public string Cidade { get; set; }
    public string Estado { get; set; }
    public string CEP { get; set; }

    public int PessoaId { get; set; }
    public Pessoa? Pessoa { get; set; } // Definindo a outra parte do relacionamento
}
```

Fluent API

- Fluent API é uma maneira de configurar o comportamento do Entity Framework (EF) Core usando métodos encadeados em vez de anotações de dados (Data Annotations).
- Enquanto as anotações de dados são aplicadas diretamente nas classes e propriedades do modelo, a Fluent API é definida dentro do método `OnModelCreating` da classe `DbContext`, oferecendo maior flexibilidade e controle sobre a configuração do modelo.

Fluent API

- Principais Características da Fluent API:
 - Configuração Detalhada:
 - Permite configurar aspectos do mapeamento que não são possíveis com anotações de dados.
 - Por exemplo, você pode configurar nomes de tabelas, chaves compostas, relações complexas entre entidades, propriedades de índice, entre outros.
 - Métodos Encadeados:
 - Usa métodos que podem ser encadeados uns nos outros, permitindo uma configuração fluida e legível.
 - Por exemplo, você pode definir um relacionamento e configurar restrições ou comportamentos adicionais tudo em uma linha de código.

Fluent API

- Principais Características da Fluent API:
 - Flexibilidade:
 - Ao usar a Fluent API, você não está limitado às convenções padrão ou às limitações das anotações de dados.
 - Você pode configurar qualquer aspecto do modelo de dados de forma mais granular.

Fluent API

- Quando Usar Fluent API:
 - Requisitos de Mapeamento Avançado:
 - Quando você precisa configurar propriedades complexas, como chaves compostas, tabelas de junção personalizadas, ou índices.
 - Nomeação e Convenções Personalizadas:
 - Se você precisa configurar nomes de tabelas ou colunas que não seguem as convenções padrão do EF Core.
- Configurações Globais:
 - Quando você deseja aplicar configurações globais a entidades ou propriedades, como definir uma convenção para todos os campos string para serem mapeados para um tipo específico de coluna no banco de dados.

Fluent API x Data Annotations

- Fluent API:
 - Oferece maior flexibilidade para configurar relacionamentos complexos e regras avançadas.
 - Permite definir configurações que não são possíveis (ou são difíceis) de serem feitas apenas com Data Annotations, como configurações de índice composto, diferentes convenções de nomenclatura, e regras de mapeamento complexas.
 - É ideal quando você precisa de controle granular sobre como as entidades são mapeadas para o banco de dados.
- Data Annotations:
 - Fornece uma forma mais simples e direta de configurar relacionamentos, mas com menos flexibilidade.
 - São limitadas na capacidade de configurar certas opções avançadas, como múltiplas chaves estrangeiras, renomeação de colunas ou propriedades compostas.
 - É mais adequado para configurações básicas ou quando o modelo é relativamente simples.

Relacionamento entre entidades

- Criando uma classe Contexto

```
public class Contexto : DbContext
{
    public DbSet<Pessoa> Pessoas { get; set; }
    public DbSet<Endereco> Enderecos { get; set; }

    public Contexto(DbContextOptions<Contexto> options) : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        //Usando Fluent API para definir o relacionamento entre as classes
        modelBuilder.Entity<Pessoa>()
            .HasOne(p => p.Endereco)
            .WithOne(e => e.Pessoa)
            .HasForeignKey<Endereco>(e => e.PessoaId);

        base.OnModelCreating(modelBuilder);
    }
}
```

appsettings.json

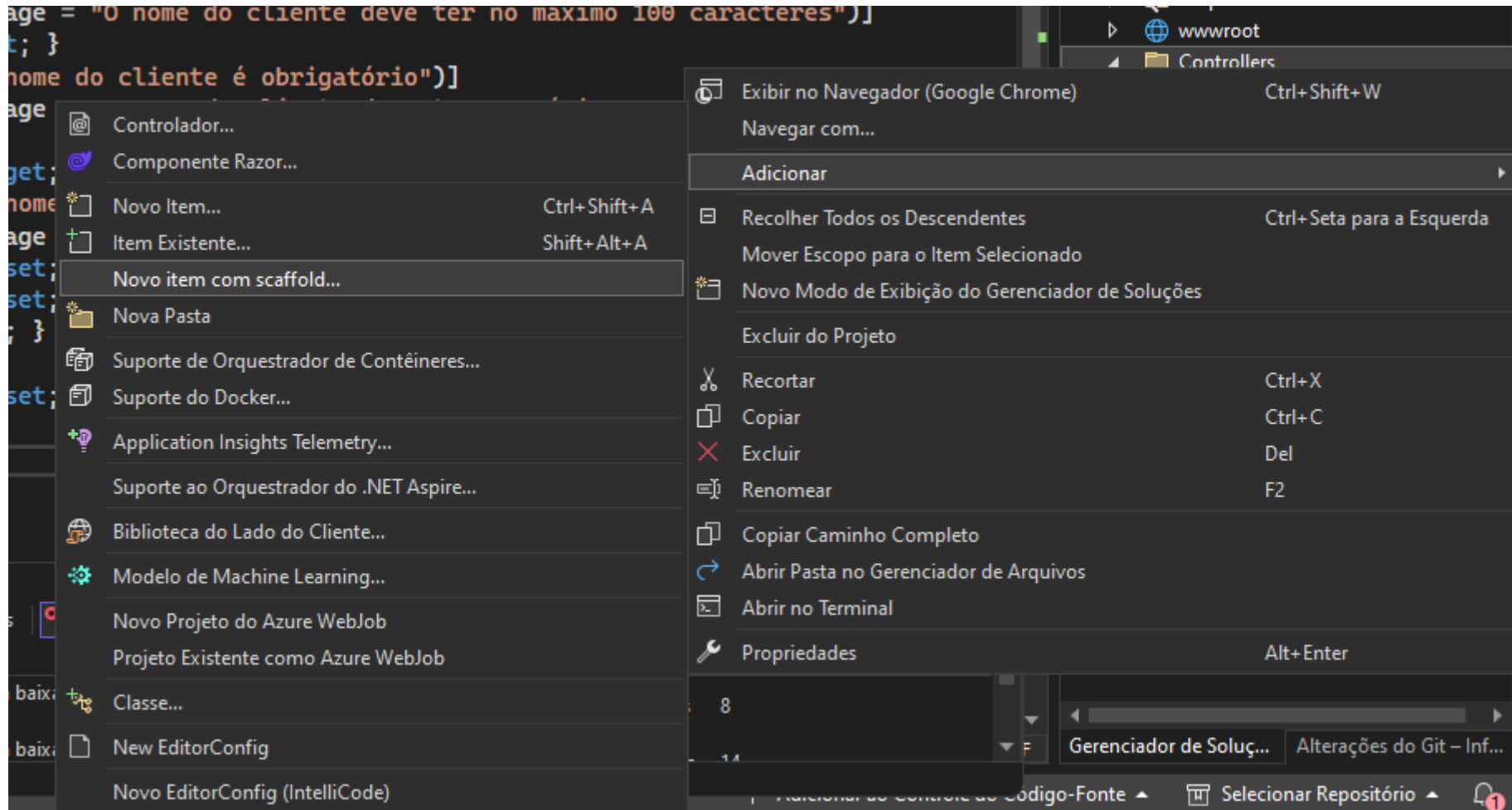
Esquema: <https://json.schemastore.org/appsettings.json>

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9    "ConnectionStrings": {
10     "DefaultConnection": "Data Source=bdPessoas.db"
11   }
12 }
```

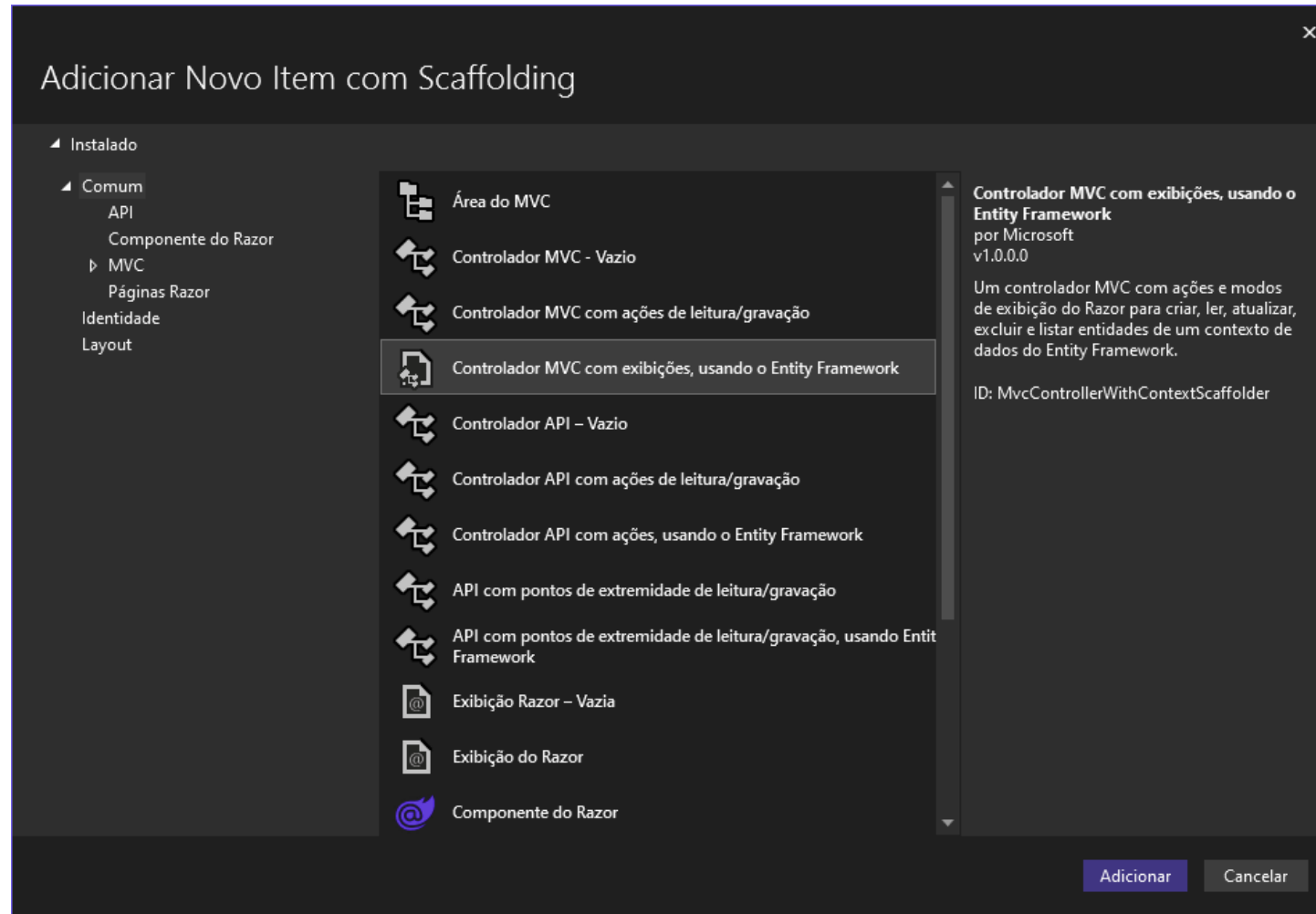
```
Program.cs  appsettings.json
Relacionamento1pra1

1 using Microsoft.EntityFrameworkCore;
2 using Relacionamento1pra1.DAL;
3
4
5 var builder = WebApplication.CreateBuilder(args);
6
7 // Add services to the container.
8 builder.Services.AddControllersWithViews();
9
10 builder.Services.AddDbContext<Contexto>(options =>
11     options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
12
13 var app = builder.Build();
14
```

Criando agora as controllers e views



Criando agora as Controllers e Views



Criando agora as Controllers e Views

×

Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo

Pessoa (Relacionamento1pra1.Models)

Classe DbContext

Contexto (Relacionamento1pra1.DAL)

+

Provedor de banco de dados

Configurado a partir do DbContext selecionado

Exibições

☒ Gerar modos de exibição

☒ Bibliotecas de scripts de referência

☒ Usar uma página de layout

...

(Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

Nome do controlador

PessoasController

Adicionar

Cancelar

Alterando o _Layout para ter o item no menu

```
_Layout.cshtml* Contexto.cs 20240829111943_initial.cs Pessoa.cs Endereco.cs
[C#] Relacionamento1pra1
13 <navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3>
14 <div class="container-fluid">
15 <div class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Relacionamento1pra1</div>
16 <div class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-co
17 <div class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-co
18 <div class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-co
19 <div class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-co
20 <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
21 <ul class="navbar-nav flex-grow-1">
22 <li class="nav-item">
23 <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
24 </li>
25 <li class="nav-item">
26 <a class="nav-link text-dark" asp-area="" asp-controller="Pessoas" asp-action="Index">Pessoas</a>
27 </li>
28 </ul>
29 </div>
```


Executando o projeto a 1ª vez

- Mas ao adicionar uma pessoa, não sai disso. Não aceita o cadastro

Relacionamento1pra1 Home Pessoas

Create

Pessoa

Nome

Create

[Back to List](#)

- Isso ocorre pois ele espera um endereço

Executando o projeto a 1ª vez

- E aqui ele verifica se o modelo é válido:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("PessoaId, Nome")] Pessoa pessoa)
{
    if (ModelState.IsValid)
    {
        _context.Add(pessoa);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(pessoa);
}
```


- E não aceita o cadastro sem endereço

Alterando a model

- Precisamos alterar a classe para informar que o Endereço é opcional

```
public class Pessoa
{
    [Key]
    public int PessoaId { get; set; }
    [Required (ErrorMessage = "O nome do cliente é obrigatório")]
    [StringLength(100, ErrorMessage ="O nome do cliente deve ter no máximo 100 caracteres")]
    public string Nome { get; set; }
    //Fazendo relacionamento utilizando Data Annotations
    public Endereco? Endereco { get; set; } //Adicionando o ? para informar que o endereço é opcional
}
```

Agora, vamos adicionar a Controller e a View Endereço



Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo: Endereco (Relacionamento1pra1.Models)

Classe DbContext: Contexto (Relacionamento1pra1.DAL) +

Provedor de banco de dados: Configurado a partir do DbContext selecionado

Exibições

- ☒ Gerar modos de exibição
- ☒ Bibliotecas de scripts de referência
- ☒ Usar uma página de layout

... (Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

Nome do controlador: EnderecosController

Adicionar Cancelar

Adicionando no menu o acesso para o cadastro de Endereços

```
EnderecosController.cs  PessoasController.cs  _Layout.cshtml  Contexto.cs  20240829111943_initial.cs  Pessoa.cs  Endereco.cs
C# Relacionamento1pra1
13  <navbar navbar-expand-sm navbar-togglerable-sm navbar-light bg-white border-bottom box-shadow mb-3">
14  <div class="container-fluid">
15  <div class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Relacionamento1pra1</div>
16  <div class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="
17  <span class="navbar-toggler-icon"></span>
18  <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
19  <div class="navbar-nav flex-grow-1">
20  <li class="nav-item">
21  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
22  </li>
23  <li class="nav-item">
24  <a class="nav-link text-dark" asp-area="" asp-controller="Pessoas" asp-action="Index">Pessoas</a>
25  </li>
26  <li class="nav-item">
27  <a class="nav-link text-dark" asp-area="" asp-controller="Enderecos" asp-action="Index">Enderecos</a>
28  </li>
29  </div>
30  </div>
31  </div>
```

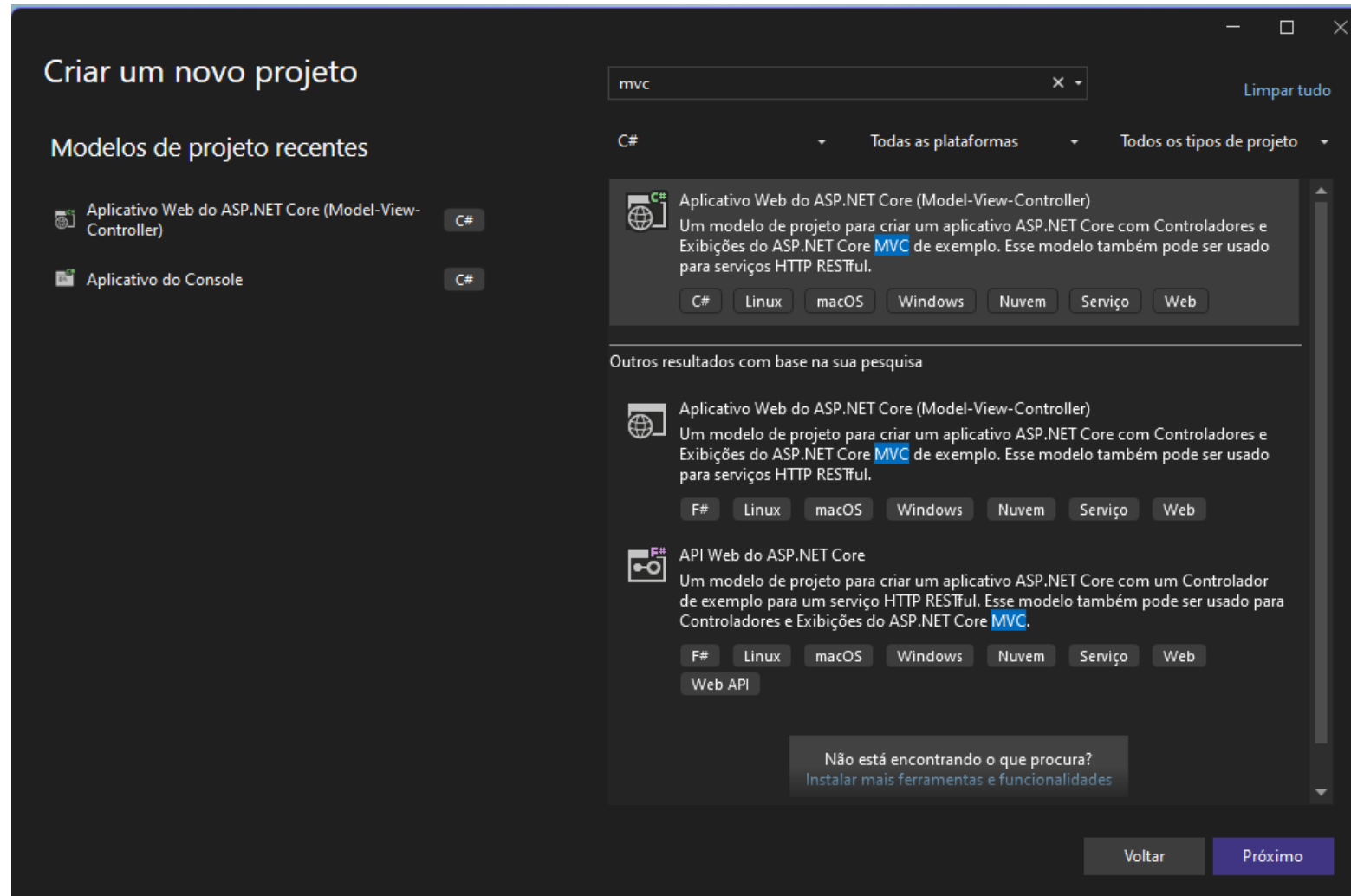
Projeto

- https://github.com/ricardofrohlich/ProjetoMVC_EF_Relacionamento1pra1

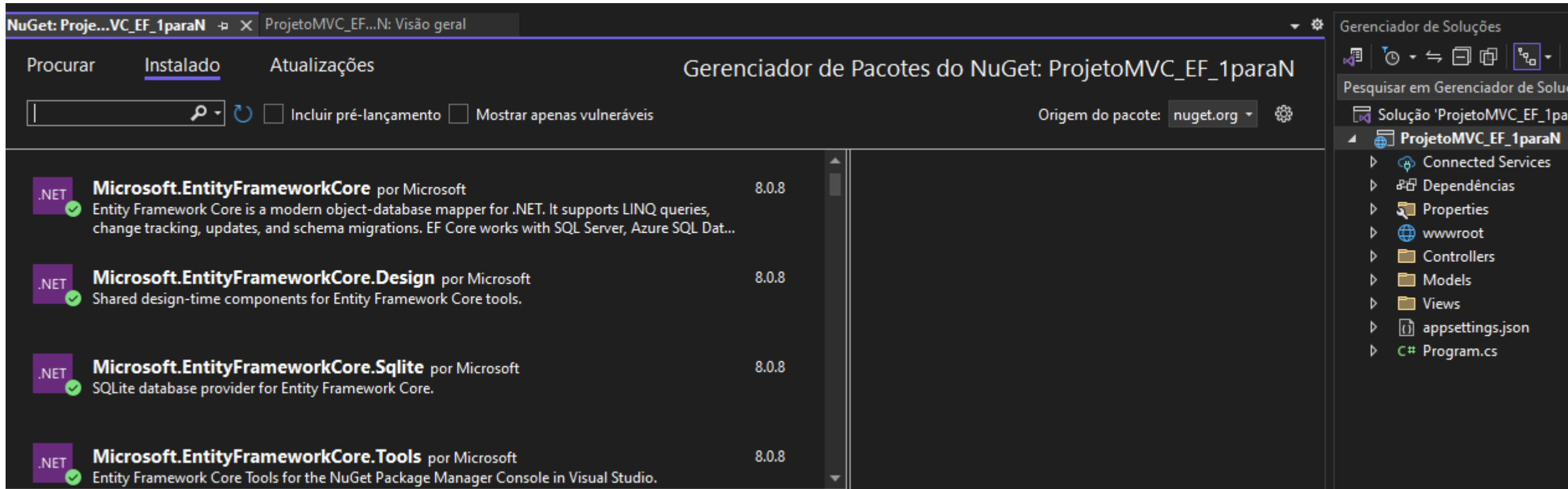
Relacionamento entre entidades

- Relacionamento Um-para-Muitos (1)
 - Esse é o relacionamento mais comum, onde uma entidade está relacionada a várias outras.
 - Um registro na tabela "um" pode ter muitos registros na tabela "muitos", mas os registros na tabela "muitos" podem estar relacionados apenas a um registro na tabela "um".
 - Exemplo:
 - Um Professor pode dar aula em várias Turmas.
 - Uma Turma tem apenas um Professor.

Criando um novo projeto



Adicionar as dependências do EF



Models

```
public class Turma
{
    public int TurmaId { get; set; }
    public string Nome { get; set; }

    public int ProfessorId { get; set; }

    public Professor? Professor { get; set; }
}
```

```
public class Professor
{
    public int ProfessorID { get; set; }
    public string Nome { get; set; }

    //Agora, vai ser diferente. Precisamos de uma lista
    public ICollection<Turma>? Turmas { get; set; }
}
```

Contexto

```
public class Contexto : DbContext
{
    public DbSet<Professor> Professores { get; set; }
    public DbSet<Turma> Turmas { get; set; }

    public Contexto(DbContextOptions<Contexto> options) : base(options)
    {
    }

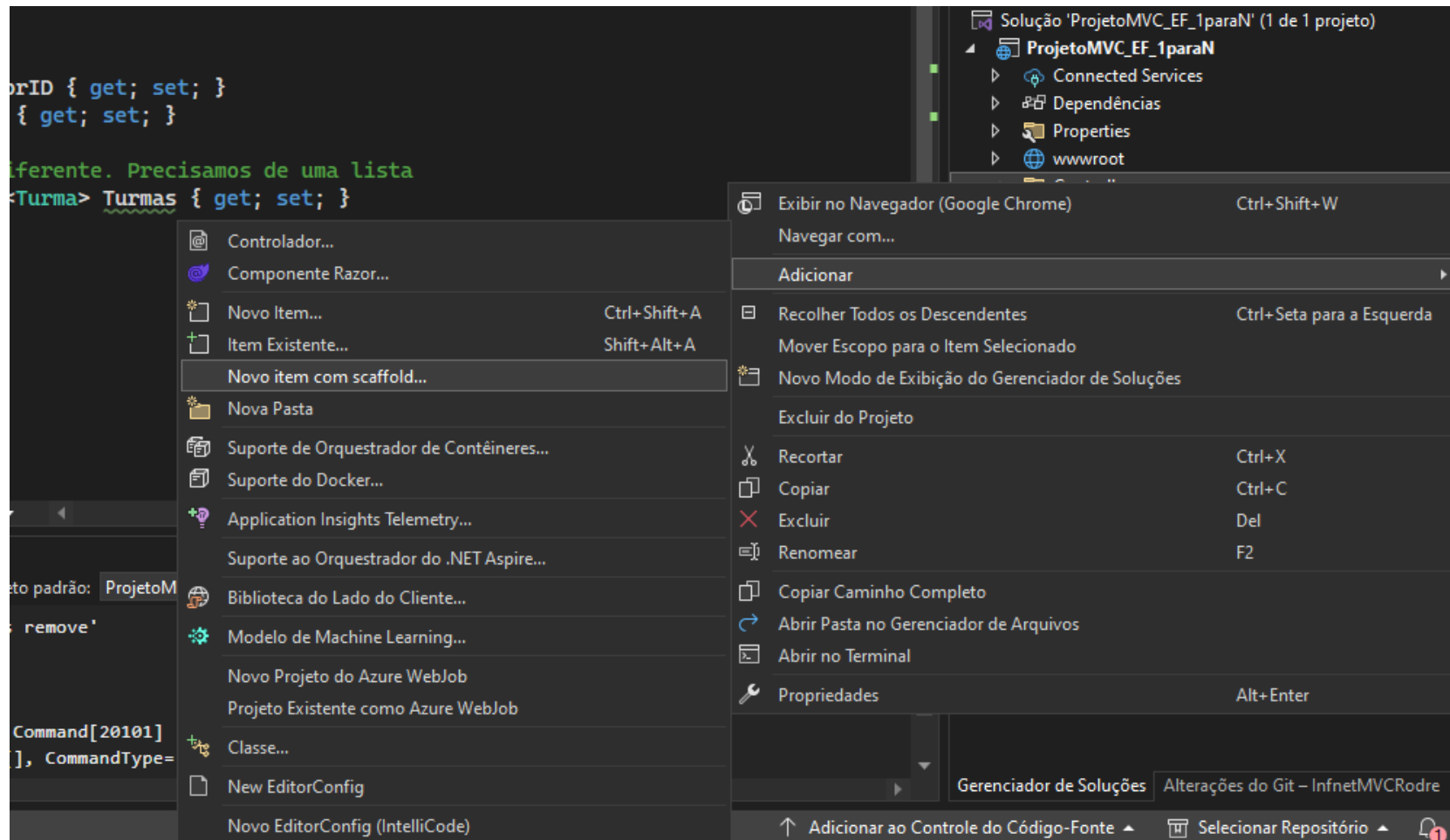
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Turma>()
            .HasOne(t => t.Professor)
            .WithMany(p => p.Turmas)
            .HasForeignKey(t => t.ProfessorId)
            .OnDelete(DeleteBehavior.Cascade);

        base.OnModelCreating(modelBuilder);
    }
}
```

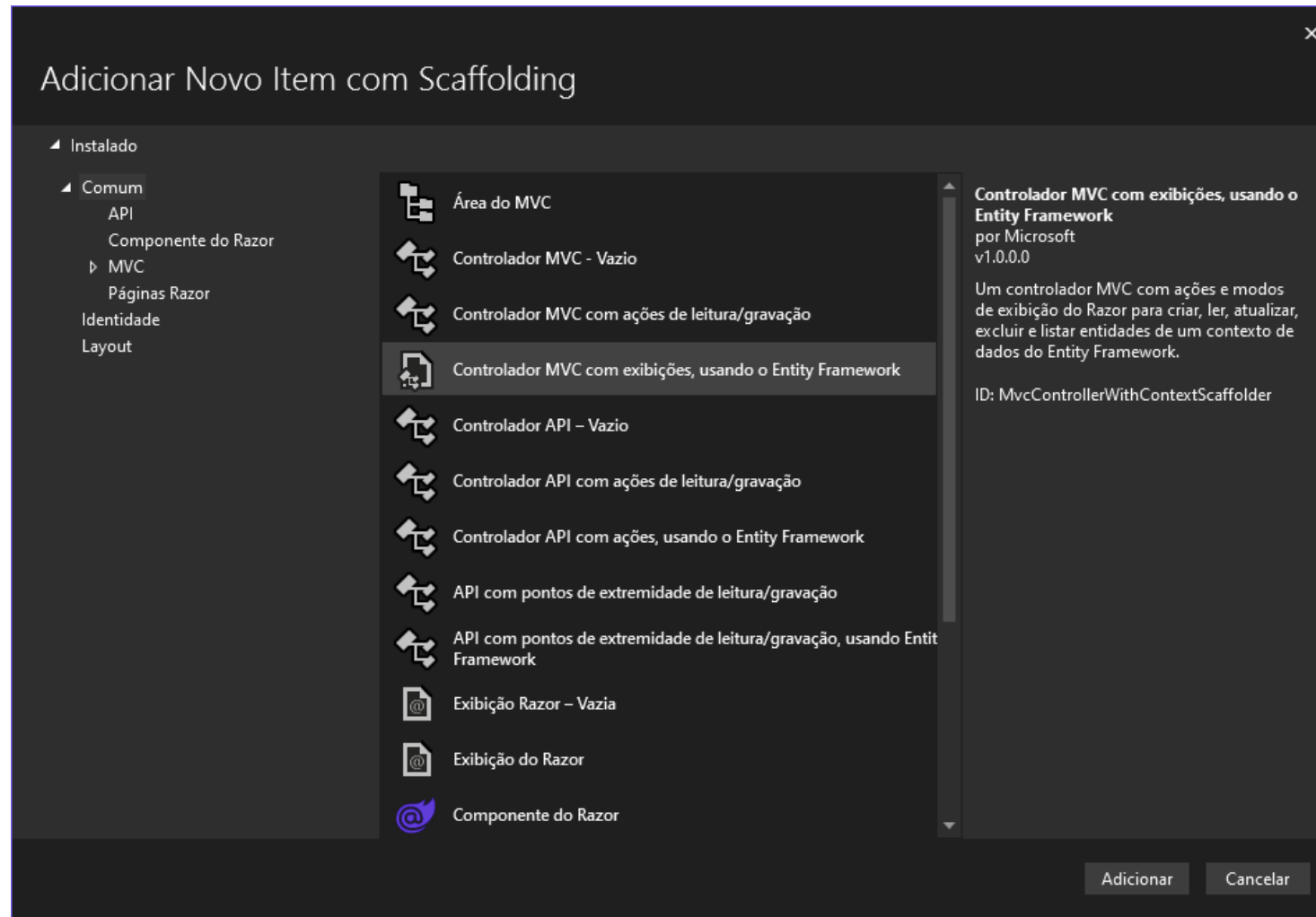
Fazendo a migration

```
Unrecognized command or argument 'add migration'  
PM> dotnet ef migrations add inicial  
Build started...  
Build succeeded.  
Done. To undo this action, use 'ef migrations remove'  
PM> dotnet ef database update  
Build started...  
Build succeeded.
```

Vamos adicionar os controladores



Adicionando Controlador Professor



Adicionando Controlador Professor

×

Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo

Professor (ProjetoMVC_EF_1paraN.Models)

Classe DbContext

Contexto (ProjetoMVC_EF_1paraN.DAL)

+

Provedor de banco de dados

Configurado a partir do DbContext selecionado

Exibições

☒ Gerar modos de exibição

☒ Bibliotecas de scripts de referência

☒ Usar uma página de layout

...

(Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

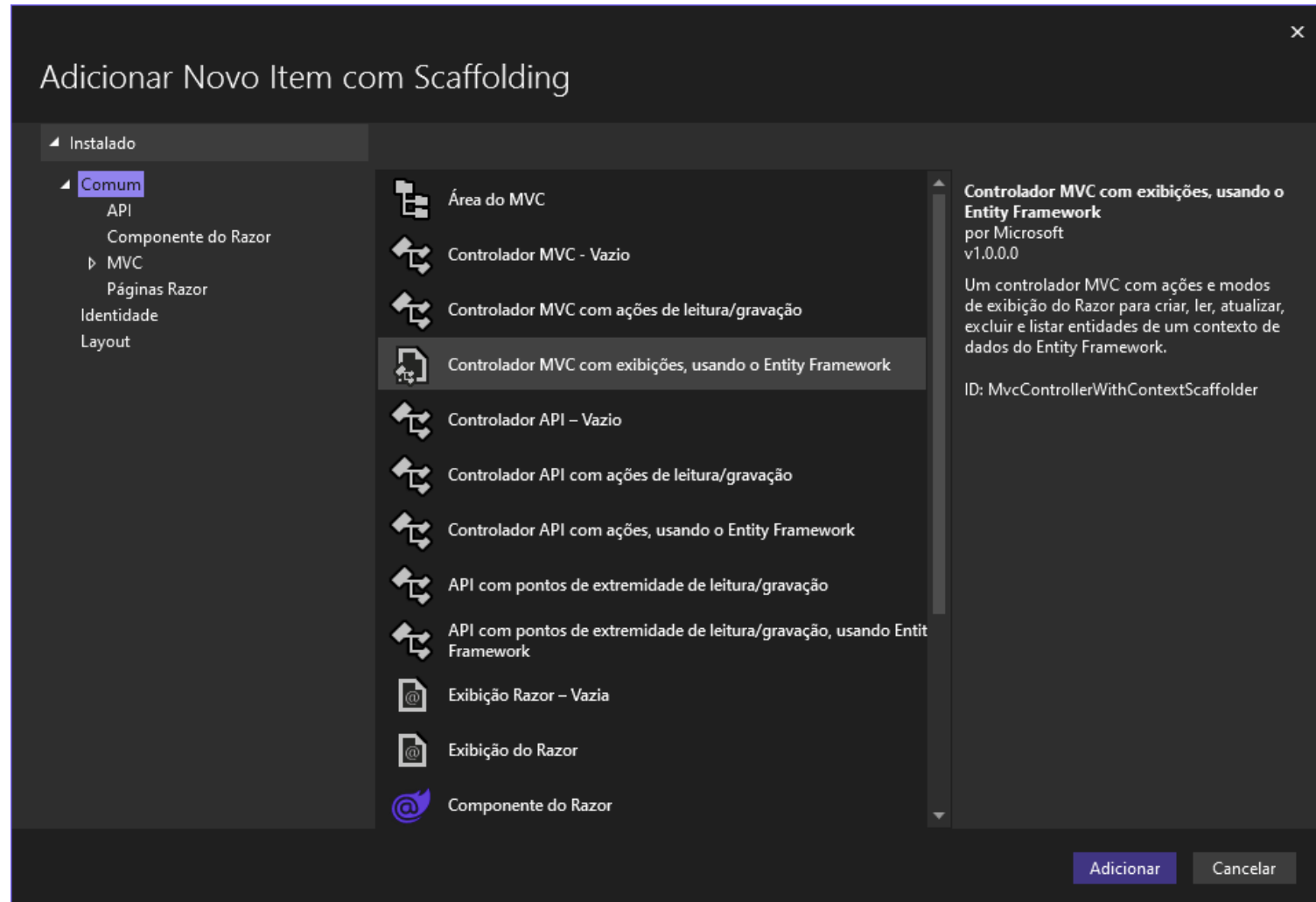
Nome do controlador

ProfessoresController

Adicionar

Cancelar

Adicionando Controlador Turma



Adicionando Controlador Turma

×

Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo

Turma (ProjetoMVC_EF_1paraN.Models)

Classe DbContext

Contexto (ProjetoMVC_EF_1paraN.DAL)

+

Provedor de banco de dados

Configurado a partir do DbContext selecionado

Exibições

☒ Gerar modos de exibição

☒ Bibliotecas de scripts de referência

☒ Usar uma página de layout

...

(Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

Nome do controlador

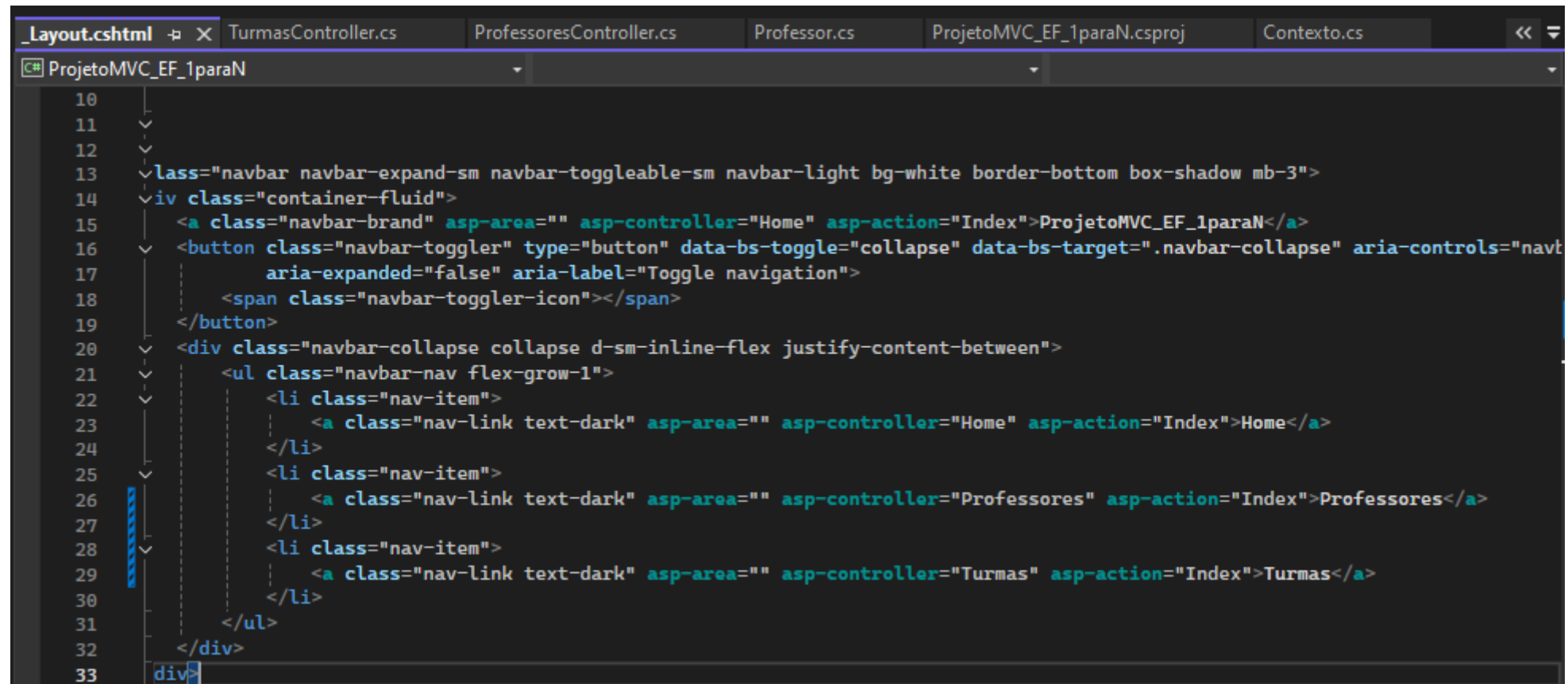
TurmasController

Adicionar

Cancelar

Vamos testar

- Adicionar no navbar para melhor acessar:



```
10
11
12
13 <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
14 <div class="container-fluid">
15 <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">ProjetoMVC_EF_1paraN</a>
16 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navb
17     aria-expanded="false" aria-label="Toggle navigation">
18     <span class="navbar-toggler-icon"></span>
19 </button>
20 <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
21 <ul class="navbar-nav flex-grow-1">
22 <li class="nav-item">
23     <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
24 </li>
25 <li class="nav-item">
26     <a class="nav-link text-dark" asp-area="" asp-controller="Professores" asp-action="Index">Professores</a>
27 </li>
28 <li class="nav-item">
29     <a class="nav-link text-dark" asp-area="" asp-controller="Turmas" asp-action="Index">Turmas</a>
30 </li>
31 </ul>
32 </div>
33 </div>
```

Inserindo professor

Index

[Create New](#)

Nome	
Ricardo	Edit Details Delete
Astolfo	Edit Details Delete
Fernando	Edit Details Delete

Inserindo turma

Index

[Create New](#)

Nome	Professor	
Desenvolvimento Web com .NET e Bases de dados	1	Edit Details Delete
Desenvolvimento Web com .NET e Bases de dados - Turma Noite	3	Edit Details Delete

Mas notem que tanto ao inserir quanto na apresentação, é mostrado somente o ID do professor, para isso, precisamos mudar algumas coisas...

-
- Na camada Controller a gente tá enviando somente o ID do professor:

```
48 // GET: Turmas/Create
49 public IActionResult Create()
50 {
51     ViewData["ProfessorId"] = new SelectList(_context.Professores, "ProfessorID", "ProfessorID");
52     return View();
53 }
```

- Consequentemente na View, Create.cshtml, estamos recebendo somente o ID do professor

```
9      <h4>Turma</h4>
10     <hr />
11     <div class="row">
12     <div class="col-md-4">
13         <form asp-action="Create">
14             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
15             <div class="form-group">
16                 <label asp-for="Nome" class="control-label"></label>
17                 <input asp-for="Nome" class="form-control" />
18                 <span asp-validation-for="Nome" class="text-danger"></span>
19             </div>
20             <div class="form-group">
21                 <label asp-for="ProfessorId" class="control-label"></label>
22                 <select asp-for="ProfessorId" class="form-control" asp-items="ViewBag.ProfessorId"></select>
23             </div>
24             <div class="form-group">
25                 <input type="submit" value="Create" class="btn btn-primary" />
26             </div>
27         </form>
28     </div>
29 </div>
```

-
- Pra isso, basta a gente alterar o código na controller pra isso:

```
// GET: Turmas/Create
public IActionResult Create()
{
    ViewData["ProfessorId"] = new SelectList(_context.Professores, "ProfessorID", "Nome");
    return View();
}
```

- Isso significa que seguimos mandando o ID como identificador, mas será apresentado o nome do professor

No Index também aparece o ID

Index

[Create New](#)

Nome	Professor	
Desenvolvimento Web com .NET e Bases de dados	1	Edit Details Delete
Desenvolvimento Web com .NET e Bases de dados - Turma Noite	3	Edit Details Delete
Microserviços	1	Edit Details Delete

Para isso, vamos alterar o código aqui:

```
<@foreach (var item in Model) {  
<tr>  
<td>  
    @Html.DisplayFor(modelItem => item.Nome)  
</td>  
<td>  
    @Html.DisplayFor(modelItem => item.Professor.ProfessorID)  
</td>  
<td>  
    <a asp-action="Edit" asp-route-id="@item.TurmaId">Edit</a> |  
    <a asp-action="Details" asp-route-id="@item.TurmaId">Details</a> |  
    <a asp-action="Delete" asp-route-id="@item.TurmaId">Delete</a>  
</td>  
</tr>  
}
```

Para isso, vamos alterar o código aqui:

```
h (var item in Model) {  
  <tr>  
    <td>  
      @Html.DisplayFor(modelItem => item.Nome)  
    </td>  
    <td>  
      @Html.DisplayFor(modelItem => item.Professor.ProfessorID) - @Html.DisplayFor(modelItem => item.Professor.Nome)  
    </td>  
    <td>  
      <a asp-action="Edit" asp-route-id="@item.TurmaId">Edit</a> |  
      <a asp-action="Details" asp-route-id="@item.TurmaId">Details</a> |  
      <a asp-action="Delete" asp-route-id="@item.TurmaId">Delete</a>  
    </td>  
  </tr>  
}
```

Resultado

Index

[Create New](#)

Nome	Professor	
Desenvolvimento Web com .NET e Bases de dados	1 - Ricardo	Edit Details Delete
Desenvolvimento Web com .NET e Bases de dados - Turma Noite	3 - Fernando	Edit Details Delete
Microserviços	1 - Ricardo	Edit Details Delete

Github do projeto

- https://github.com/ricardofrohlich/ProjetoMVC_EF_1paraN

Relacionamento Muitos-para-Muitos (N)

- O relacionamento muitos-para-muitos (N) ocorre quando uma entidade pode estar relacionada a várias outras entidades e vice-versa.
- No banco de dados, esse tipo de relacionamento é representado por uma tabela de junção que contém as chaves estrangeiras de ambas as entidades.
- Exemplo:
 - Um Estudante pode estar matriculado em vários Cursos.
 - Um Curso pode ter vários Estudantes matriculados.
 - No Entity Framework Core (EF Core), o relacionamento muitos-para-muitos é implementado utilizando uma tabela associativa que o próprio EF Core pode gerenciar automaticamente.

Relacionamento Muitos-para-Muitos (N)

	A	B	C	D	E	F	G	H
1	Estudantes							
2		1 Ricardo						
3		2 Rafael						
4		3 Guilherme						
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								

< >

Estudante

Curso

EstudantesCursos

+

	A	B	C	D	
1	Curso				
2		1 Ciência da computação			
3		2 Engenharia da Computação			
4		3 Análise e desenvolvimento de sistemas			
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					

< >

Estudante

Curso

EstudantesCursos

+

Relacionamento Muitos-para-Muitos (N)

	A	B	C	D	E	
1	EstudanteID	CursosID				
2		1	1			
3		2	1			
4		1	3			
5		1	2			
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						

< >

Estudante | Curso | EstudantesCursos | +

Criando o projeto

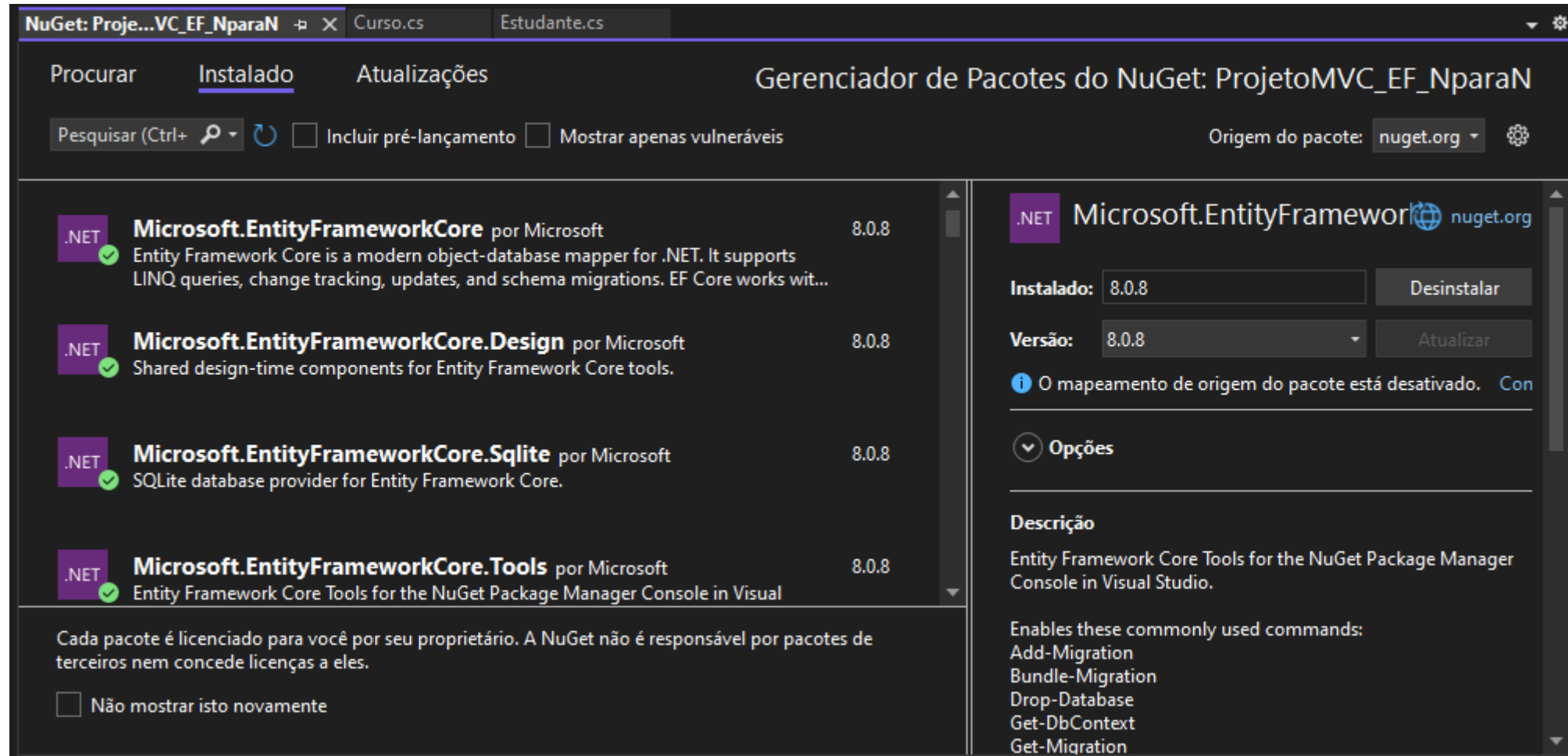
```
public class Curso
{
    public int CursoID { get; set; }
    public string NomeCurso { get; set; }
    public ICollection<EstudantesCursos>? EstudantesCursos { get; set; }
}
```

```
public class Estudante
{
    public int EstudanteId { get; set; }
    public string Nome { get; set; }
    public ICollection<EstudantesCursos>? EstudantesCursos { get; set; }
}
```

```
public class EstudantesCursos //Classe Junção -> Ter 1 estudante -> 1 curso
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int EstudantesCursosId { get; set; }
    public int CursoId { get; set; }
    public Curso? Curso { get; set; }

    public int EstudanteID { get; set; }
    public Estudante? Estudante { get; set; }
}
```


Criando o projeto

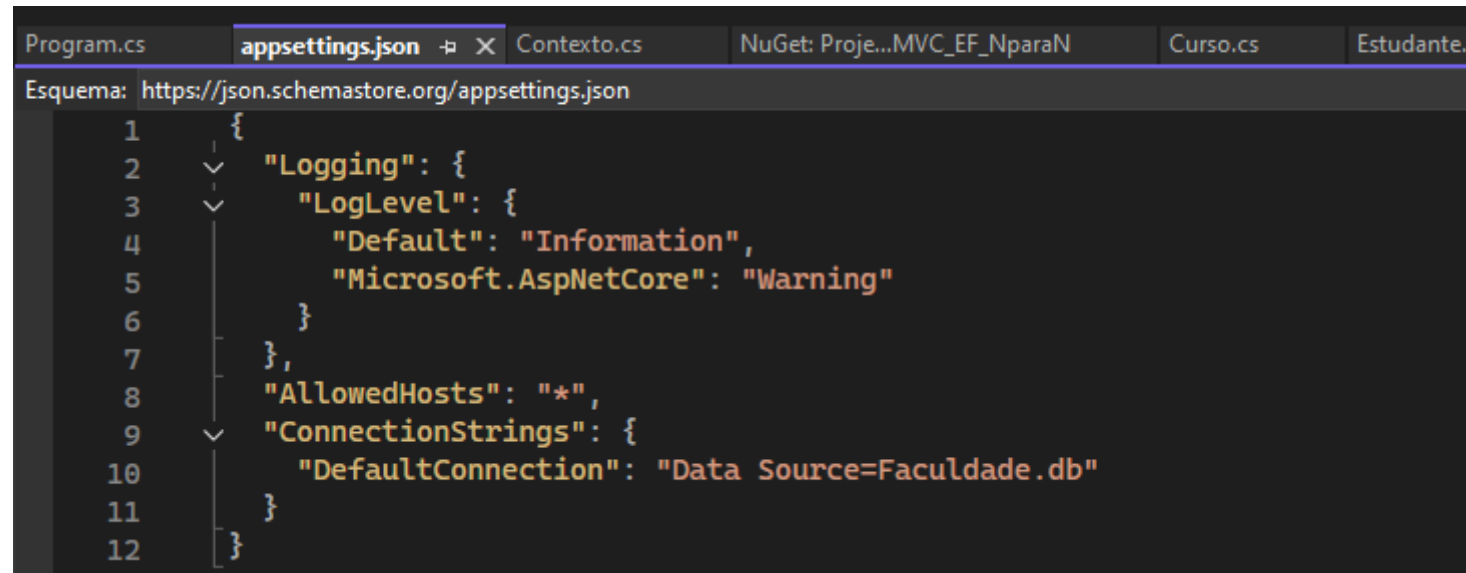


Criando a contexto

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<EstudantesCursos>()
        .HasKey(e => e.EstudantesCursosId);

    //chave estrangeira dentro da tabela EstudanteCursos referente ao Curso
    modelBuilder.Entity<EstudantesCursos>()
        .HasOne(ec => ec.Curso)
        .WithMany(c => c.EstudantesCursos)
        .HasForeignKey(ec => ec.CursoId);
    //chave estrangeira dentro da tabela EstudanteCursos referente ao Estudante
    modelBuilder.Entity<EstudantesCursos>()
        .HasOne(ec => ec.Estudante)
        .WithMany(e => e.EstudantesCursos)
        .HasForeignKey(ec => ec.EstudanteID);
}
```

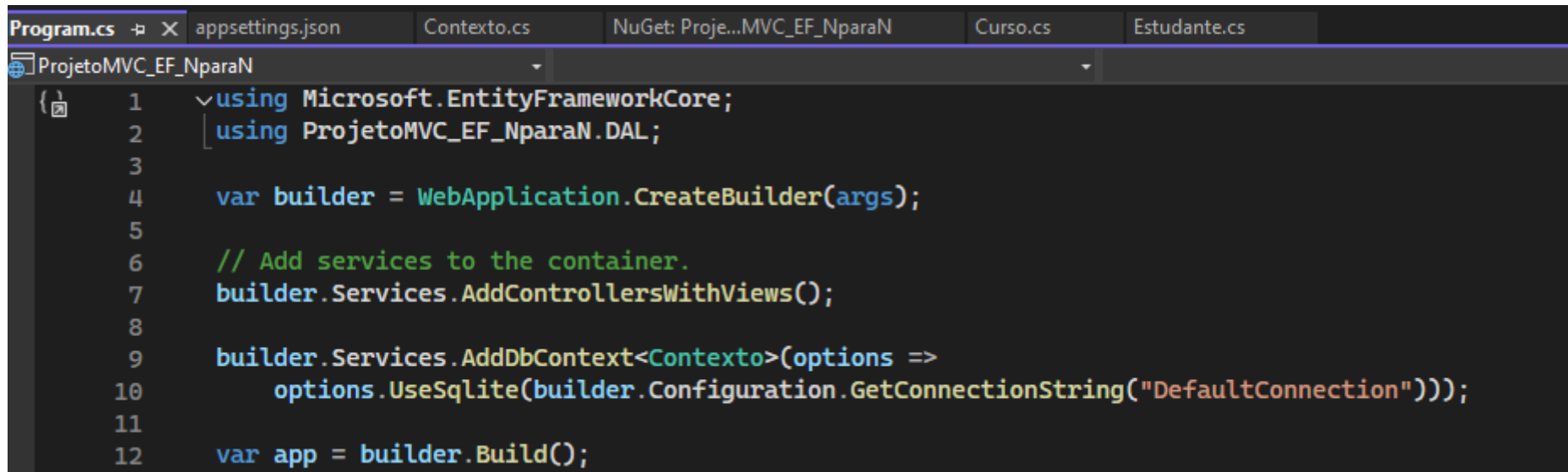
Configurações de “conexão” com o banco de dados



The screenshot shows an IDE window with several tabs: Program.cs, appsettings.json (selected), Contexto.cs, NuGet: Proje...MVC_EF_NparaN, Curso.cs, and Estudante. The left sidebar displays a tree view of the appsettings.json file, with lines 1 through 12 numbered. The main editor area shows the following JSON configuration:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=Faculdade.db"
  }
}
```

Chamando a Contexto no início da execução

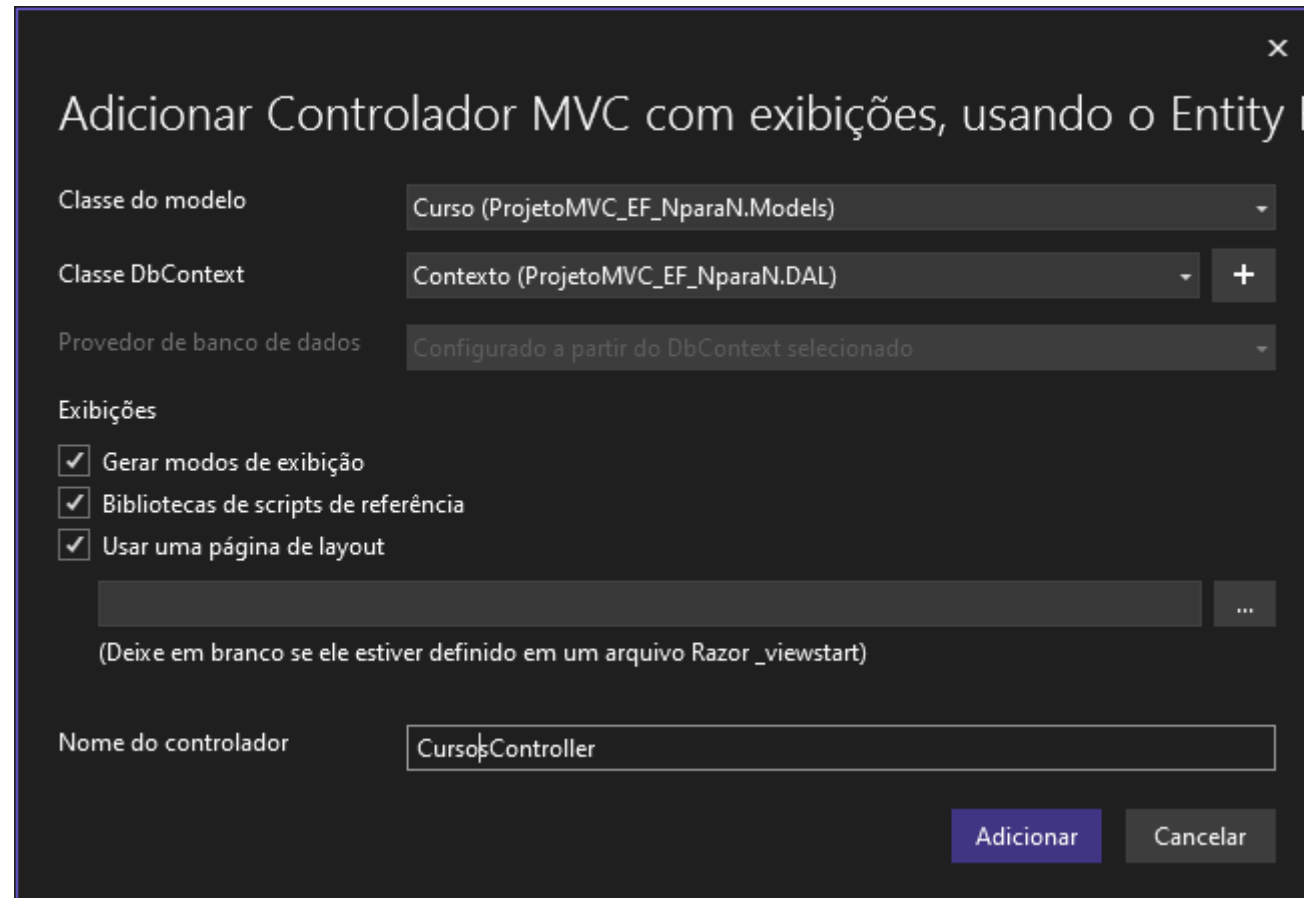


```
Program.cs  X appsettings.json Contexto.cs NuGet: Proje...MVC_EF_NparaN Curso.cs Estudante.cs
ProjetoMVC_EF_NparaN
1  using Microsoft.EntityFrameworkCore;
2  using ProjetoMVC_EF_NparaN.DAL;
3
4  var builder = WebApplication.CreateBuilder(args);
5
6  // Add services to the container.
7  builder.Services.AddControllersWithViews();
8
9  builder.Services.AddDbContext<Contexto>(options =>
10     options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
11
12  var app = builder.Build();
```

Agora, vamos fazer a migration

```
PM> add-migration inicial  
Build started...  
Build succeeded.  
To undo this action, use Remove-Migration.  
PM> update-database  
Build started...  
Build succeeded.
```

Vamos criar os itens com Scaffold



Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo: Curso (ProjetoMVC_EF_NparaN.Models)

Classe DbContext: Contexto (ProjetoMVC_EF_NparaN.DAL) +

Provedor de banco de dados: Configurado a partir do DbContext selecionado

Exibições

- ☒ Gerar modos de exibição
- ☒ Bibliotecas de scripts de referência
- ☒ Usar uma página de layout

... (Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

Nome do controlador: CursoController

Adicionar Cancelar

Vamos criar os itens com Scaffold

×

Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo

Estudante (ProjetoMVC_EF_NparaN.Models) ▾

Classe DbContext

Contexto (ProjetoMVC_EF_NparaN.DAL) ▾

+

Provedor de banco de dados

Configurado a partir do DbContext selecionado ▾

Exibições

☒ Gerar modos de exibição

☒ Bibliotecas de scripts de referência

☒ Usar uma página de layout

...

(Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

Nome do controlador

EstudantesController

Adicionar

Cancelar

Vamos criar os itens com Scaffold

×

Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo

EstudantesCursos (ProjetoMVC_EF_NparaN.Models) ▾

Classe DbContext

Contexto (ProjetoMVC_EF_NparaN.DAL) ▾ +

Provedor de banco de dados

Configurado a partir do DbContext selecionado ▾

Exibições

☒ Gerar modos de exibição

☒ Bibliotecas de scripts de referência

☒ Usar uma página de layout

...

(Deixe em branco se ele estiver definido em um arquivo Razor _viewstart)

Nome do controlador

EstudantesCursosController

Adicionar

Cancelar

Adicionando o acesso no NavBar

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Cursos" asp-action="Index">Cursos</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Estudantes" asp-action="Index">Estudantes</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="EstudantesCursos" asp-action="Index">EstudantesCursos</a>
    </li>
  </ul>
</div>
```

```
Index.cshtml  Estudante.cs  Curso.cs  _Layout.cshtml  EstudantesCursosController.cs  EstudantesController.cs
C# ProjetoMVC_EF_NparaN
11  <table class="table">
12  <thead>
13  <tr>
14  <th>
15  @Html.DisplayNameFor(model => model.Curso)
16  </th>
17  <th>
18  @Html.DisplayNameFor(model => model.Estudante)
19  </th>
20  <th></th>
21  </tr>
22  </thead>
23  <tbody>
24  @foreach (var item in Model) {
25  <tr>
26  <td>
27  @Html.DisplayFor(modelItem => item.Curso.CursoID) - @Html.DisplayFor(modelItem => item.Curso.NomeCurso)
28  </td>
29  <td>
30  @Html.DisplayFor(modelItem => item.Estudante.EstudanteId) - @Html.DisplayFor(modelItem => item.Estudante.Nome)
31  </td>
32  <td>
33  @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
34  @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
35  @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
36  </td>
37  </tr>
38  </tbody>
</table>
```

```
Index.cshtml | Estudante.cs | Curso.cs | _Layout.cshtml | EstudantesCursosController.cs | EstudantesController.cs
ProjetoMVC_EF_NparaN | ProjetoMVC_EF_NparaN.Controllers.EstudantesCursos | Create(EstudantesCursos estudantesCursos)

42     {
43         return NotFound();
44     }
45
46     return View(estudantesCursos);
47 }
48
49 // GET: EstudantesCursos/Create
50 public IActionResult Create()
51 {
52     ViewData["CursoId"] = new SelectList(_context.Cursos, "CursoID", "NomeCurso");
53     ViewData["EstudanteID"] = new SelectList(_context.Estudantes, "EstudanteId", "Nome");
54     return View();
55 }
```



localhost:7079/EstudantesCursos



ProjetoMVC_EF_NparaN Home Cursos Estudantes EstudantesCursos

Index

[Create New](#)

Curso	Estudante	
1 - Ciência da Computação	1 - Ricardo Frohlich da Silva	Edit Details Delete
2 - Engenharia da Computação	2 - Rafael Silveira	Edit Details Delete