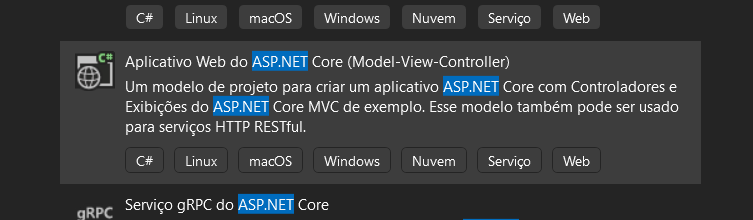
**ASP.NET Core MVC**

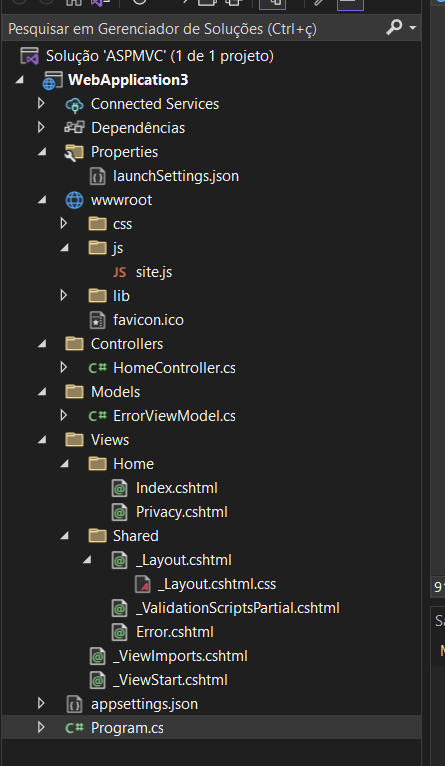
O ASP.NET Core é uma versão recente do ASP.NET, e o projeto ASP.NET Core MVC é um projeto ASP.NET básico que utiliza o padrão MVC.

Agora, descrevendo um pouco mais tecnicamente, o ASP.NET Core MVC é um novo framework de código aberto para a construção de aplicações conectadas pela internet. Ele permite o desenvolvimento e a execução de aplicações em Windows, Mac e Linux, e estas podem ser executadas no .NET Core ou no .NET Framework (versão anterior do .NET).

**Como criar o projeto:**

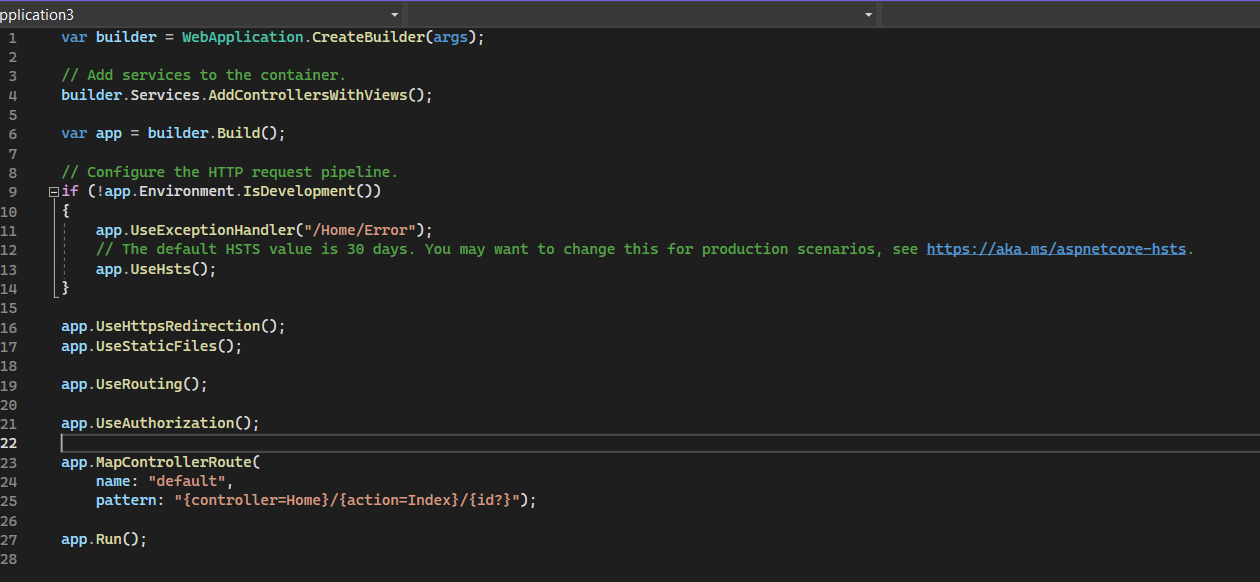
Após termos criado a solução em branco criamos um projeto ASP.NET Core MVC:





**Entendendo a estrutura do projeto:**

Analisando as pastas percebemos logo de cara a pasta dependências onde terá as dependências do projeto, em seguida temos a pasta **Properties** essa pasta ira conter as propriedades do projeto, basicamente ira conter as configurações Relacionadas ao IIS Express, em seguida temos a pasta **wwwroot** essa pasta contém os arquivos secundários ao View, como imagens, css, js e as bibliotecas do FrontEnd, em seguida temos as pastas principais da aplicação os MVC, em seguida temos o arquivo **appsetting.json** onde ira conter as configurações relacionadas ao funcionamento do projeto como a configuração do Entity Framework por exemplo, em seguida temos o arquivo **Program.cs**:



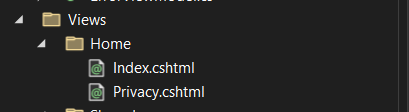
A classe Program.cs possui as configurações para rodar a aplicação, inicialmente o importante para analisarmos é o método **MapControllerRoute** que é responsável por mapear as rotas dos nossos controllers.

Basicamente como está agora ele está definindo que se não for especificado nenhum controller ele será o Home e caso não seja fornecido a action ela será Index.

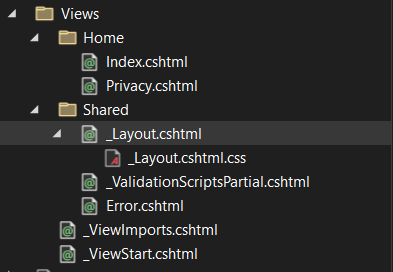
**Analisando os controllers:**

****

Analisando a classe HomeController percebemos que todo Controller deve ter o seu nome seguido da palavra Controller e deve extender a classe Controller.

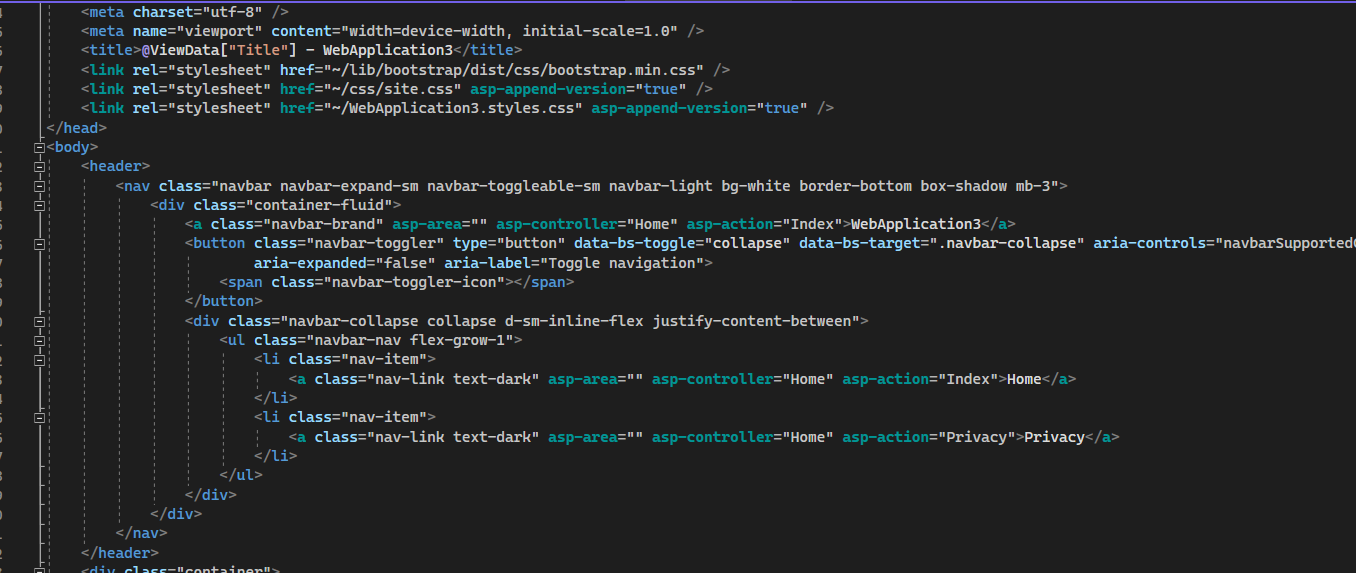
Analisando os métodos da classe percebemos que todos retornam um **IActionResult** e através disso retornam uma view, no caso dos métodos em Home podemos ver essas views na pasta view: 

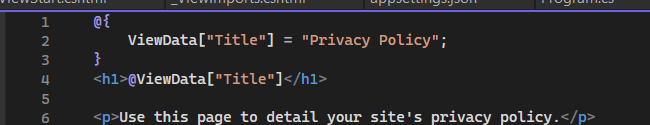
**Entendendo a view:**

****

Podemos analisar que todo controller terá a sua pasta em View com as suas páginas correspondentes, analisando percebemos que os arquivos com “\_” serão compartilhados entre as Views, dessa forma temos o \_Layout.cshtml que irá conter a estrutura html padrão, dessa forma não precisamos escrever toda a estrutura html em cada View, pois ela já é declarada em \_Layout.cshtml e será compartilhada entre as Views.

**Analisando a estrutura cshtml:**

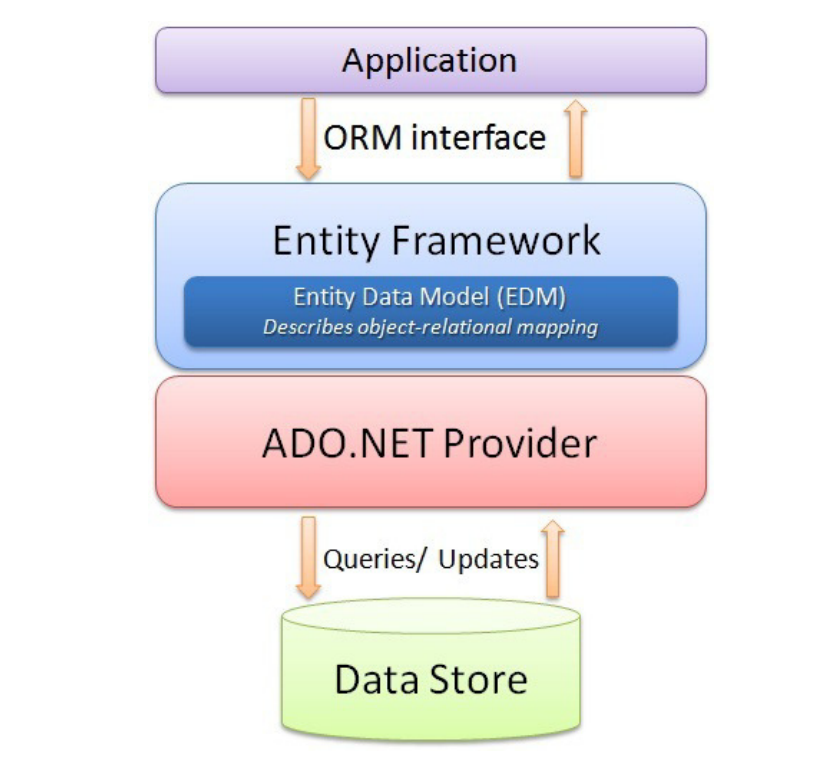
****

****

Podemos perceber que a estrutura cshtml é semelhante ao que é o jsp para o java, temos código html com código c#, podemos analisar isso através da anotação “@” e alguns métodos como “asp-area”, “asp-controller” e “asp-action”.

**Entity Framework**

O Entity Framework é um poderoso Object Relational Mapper (ORM), que gera objetos de negócios e entidades de acordo com as tabelas do banco de dados. Veja o diagrama de sua estrutura:



O Entity Framework possui 3 linhas de atuação principal:

**.Database First 🡪** Primeiro é feito a database no banco de dados e depois é interligada no Entity Framework.

**.Model First 🡪** Primeiro é feito o modelo no sistema e a partir dele podemos gerar nossa base de dados

**.Code First 🡪** Primeiro é feito o modelo no sistema e depois é de responsabilidade do Entity Framework criar a database.

**Data Annotations:**

Os data Anotations é um recurso que permite que você adicione atributos e métodos em nossas classes para alterar convenções padrão e personalizar alguns comportamentos.

**Prindipais Atributos:**

**🡪 Required:** Significa campo obrigatório**.**

**🡪 RegularExpression:** Valida o campo por expressão regular.

**🡪 Display:** Nome a ser mostrado em todas as interfaces de usuário.

**🡪 StringLength:** Determina a quantidade máxima de caracteres que poderá ser informada.

**🡪 MinLength:** Determina a quantidade mínima de caracteres que poderá ser informada**.**

**🡪DisplayFormat:** Formato a ser exibido nas interfaces de usuário.

**🡪Range:** Define a faixa de dados aceita pela propriedade.

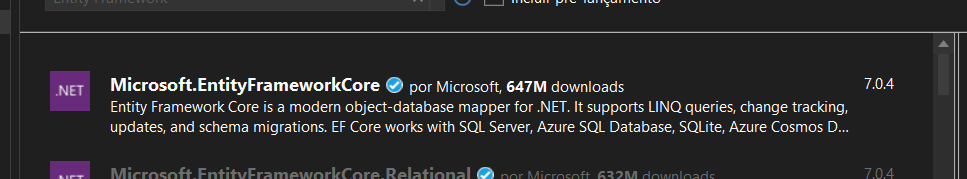
**Migrations:**

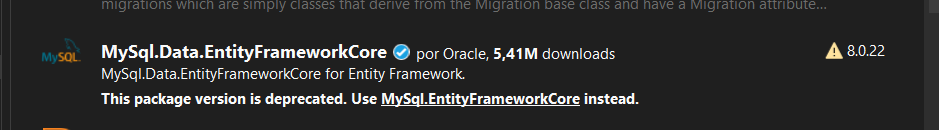
É um recurso que oferece uma maneira de atualizar o banco de dados de forma incremental para manter a sincronia com os modelos de classes.

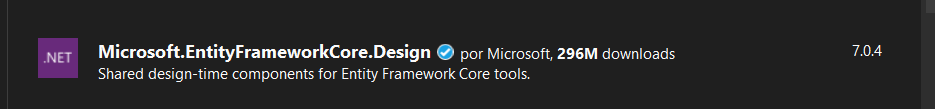
Também é possível através do Migrations fazer o Downgrade caso você deseje voltar para a versão anterior em que se encontrava.

**Adicionando o Entity Framework ao projeto:**

Primeiramente vamos adicionar as dependências do Entity Framework através do gerenciador NuGet:

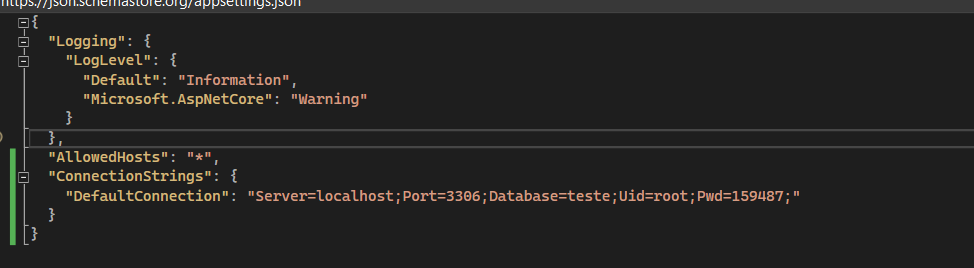




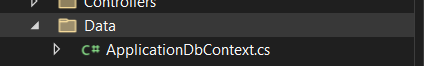




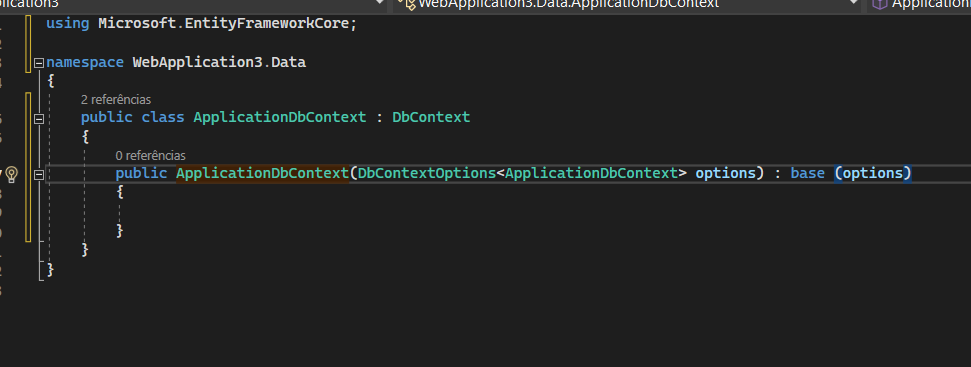
Após adicionarmos as dependências do Entity Framework devemos criar a string de conexão do banco de dados no arquivo appsettings.json:



Após adicionarmos a string de conexão devemos criar a classe de contexto do banco de dados:

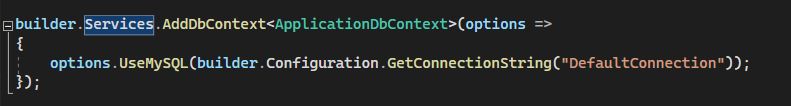


**Criando a classe de contexto do banco de dados:**

****

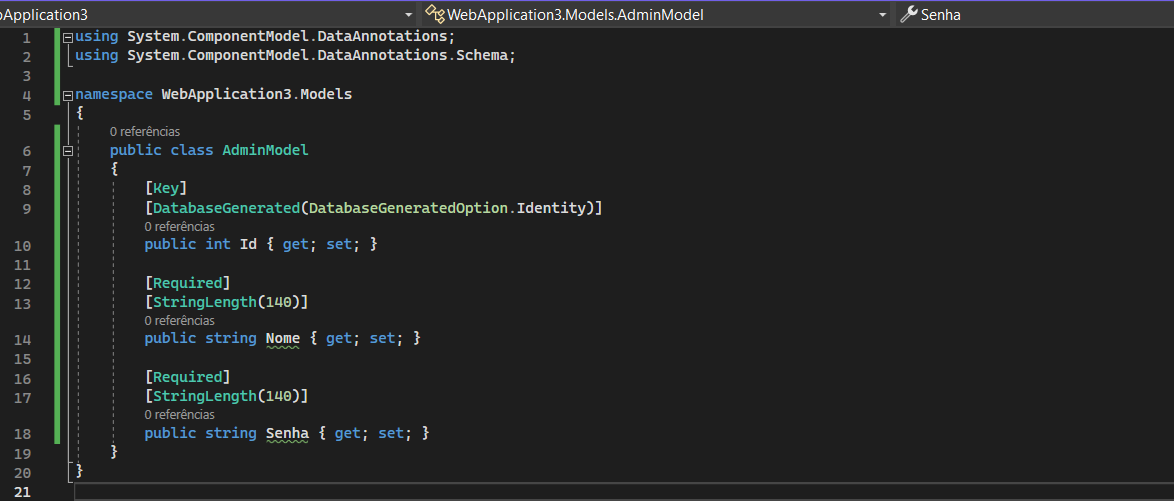
Devemos criar uma classe que extende a classe DbContext e que tenha um constructor que receba a classe DbContextOptions com type igual a classe do contexto, esse constructor deve extender a base(options) do DbContext.

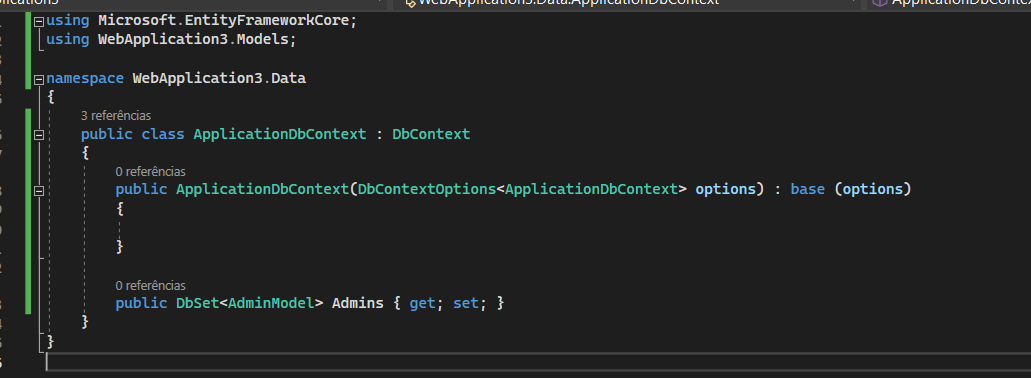
Em seguida devemos aplicar esse DbContext com a string de conexão, isso é feito adicionando uma configuração em builder na classe Program.cs:



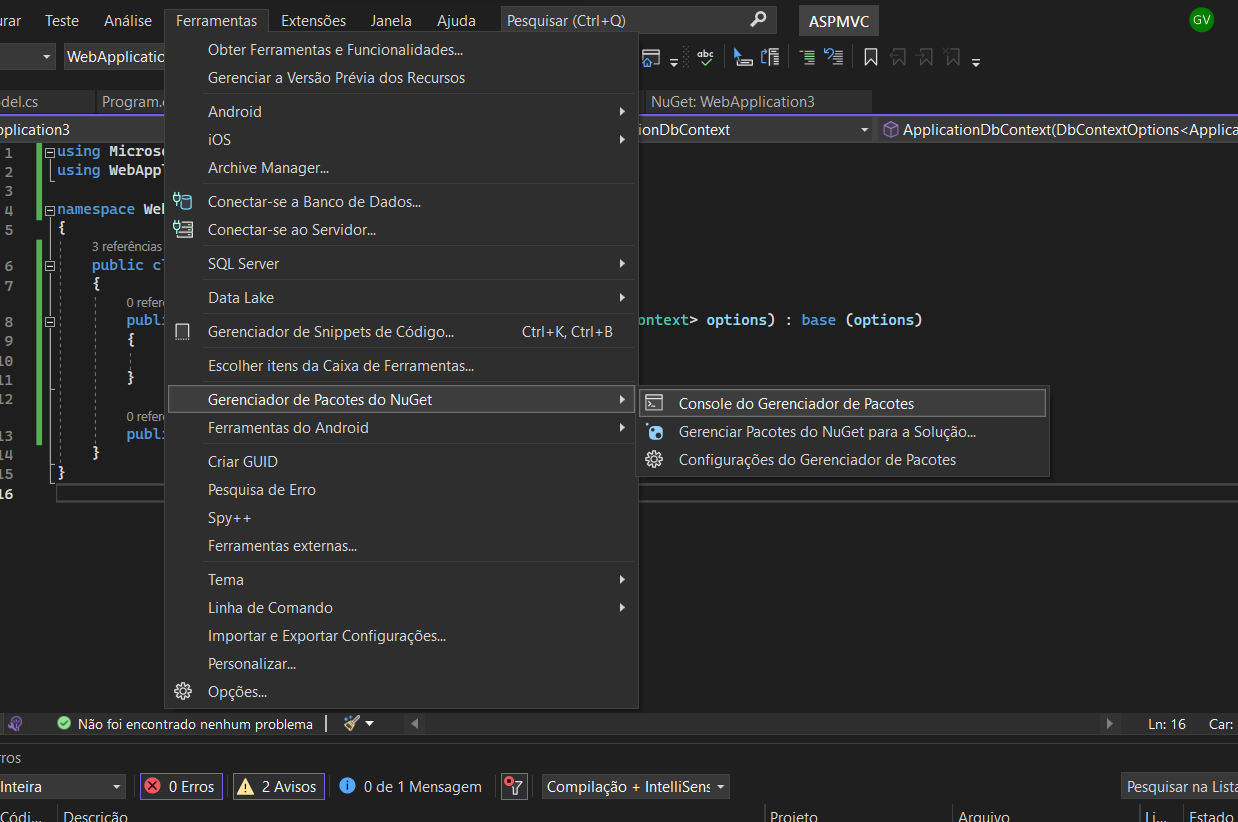
**Migrations - Model First:**

Após configurarmos o DbContext, vamos adicionar uma tabela ao nosso banco de dados:



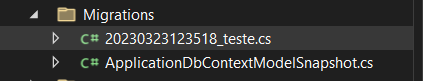


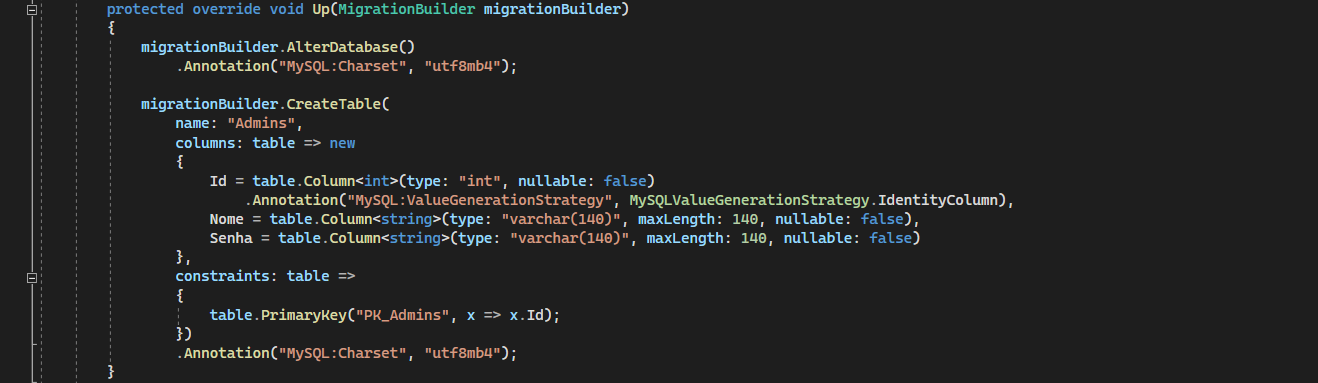
Após isso vamos usar o migrations:





Após isso se tudo estiver correto sera criado uma pasta migrations com os arquivos da migração:

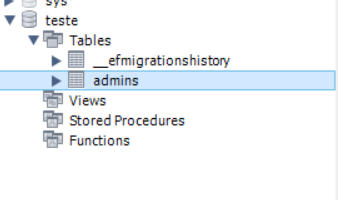




Após isso só precisamos usar o comando:

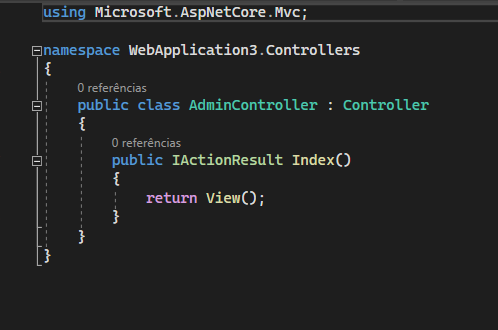


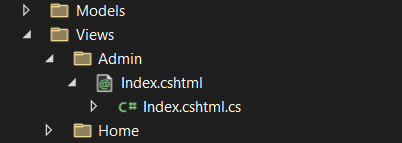
E a tabela será criada no banco de dados:



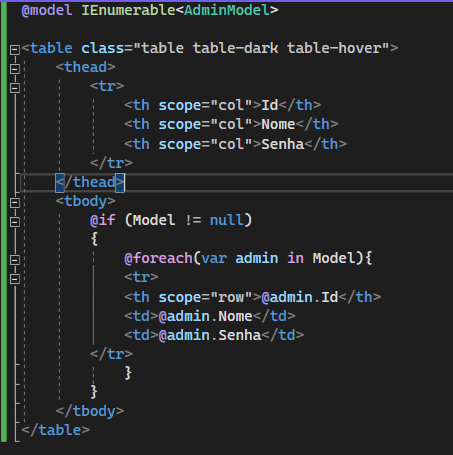
**Aplicando o contexto no controller:**

Vamos aplicar a entidade que criamos em uma view e pegar os dados, para isso vamos primeiro criar o controller de admin:

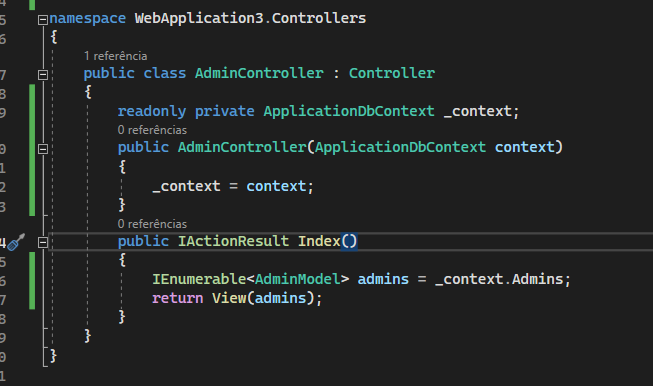




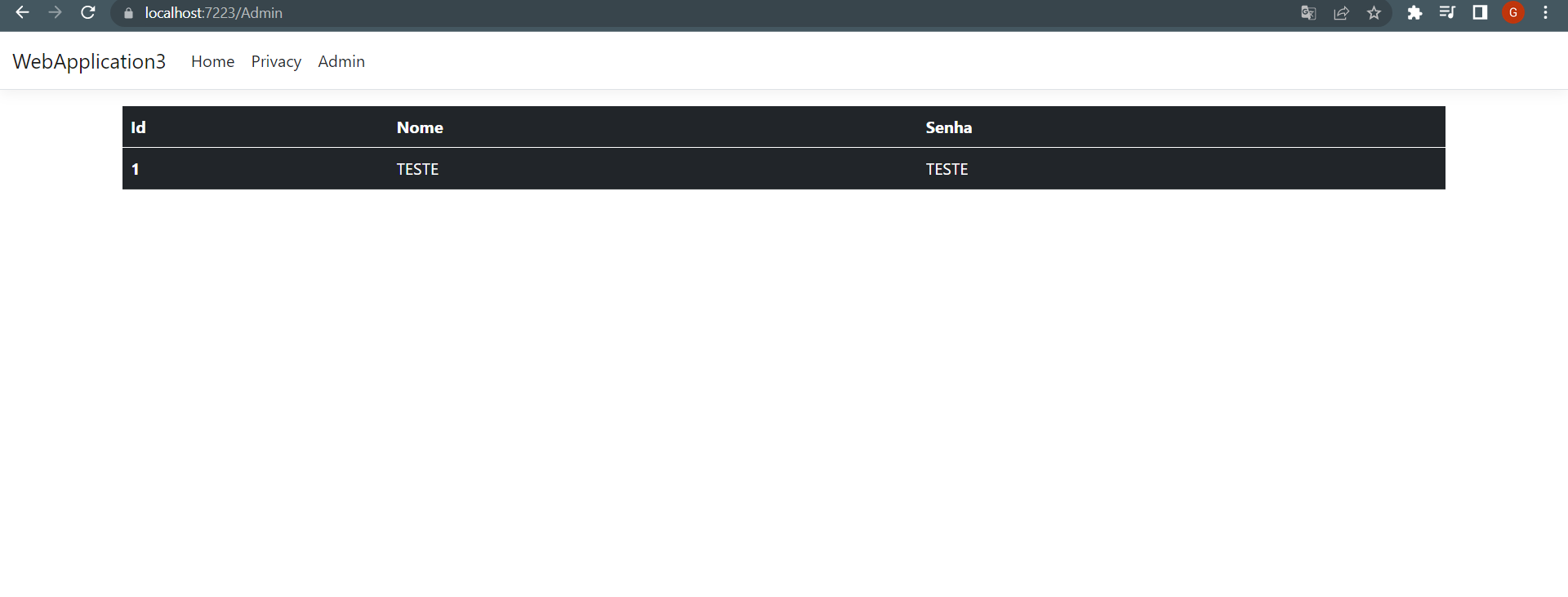
Agora vamos criar um layout de tabela para os dados:



Perceba que usamos um parâmetro IEnumerable<AdminModel> que deve ser passado pela classe controler:



Perceba usamos o constructor para pegar o contexto e armazenar em uma variável, e através dessa variável pegamos os “Admins” e passamos para a view.



**Migrations-DatabaseFirst:**

Podemos importar um banco de dados para o código através de um comando assim:

Scaffold-DbContext "Server=localhost; Port=3306; Database=teste; Uid=root;Pwd=159487;" MySql.EntityFrameworkCore -OutputDir Models