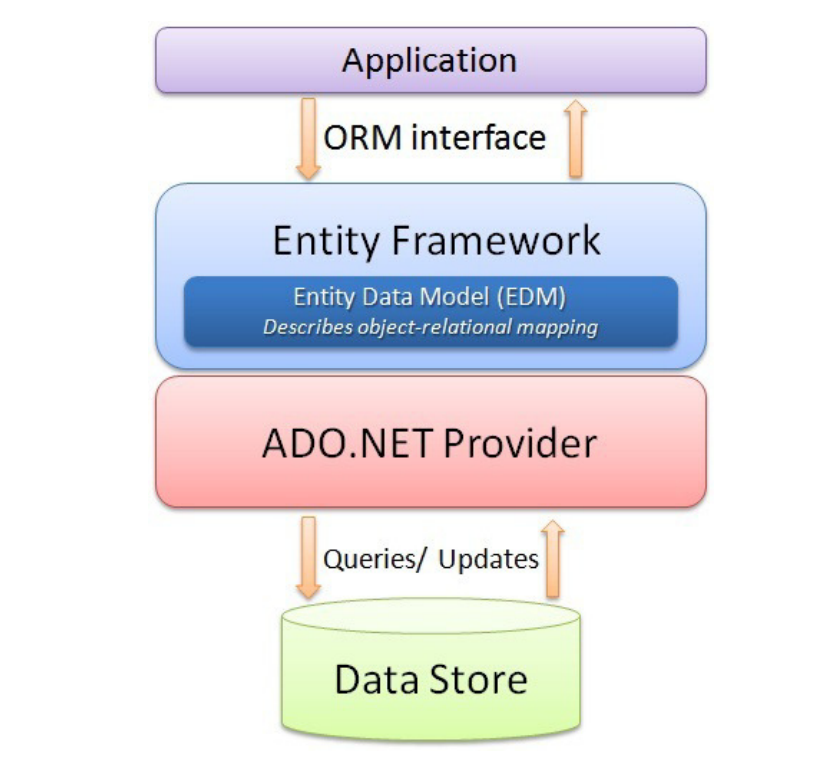
**Entity Framework**

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.

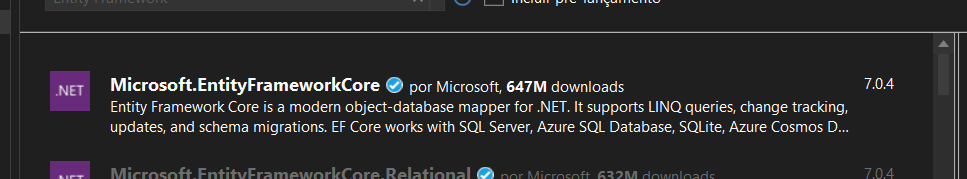
**Migrations:**

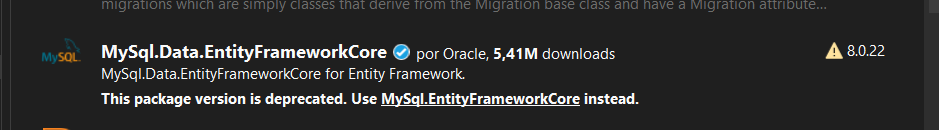
É um recurso que oferece uma maneira de atualizar o banco de dados de forma incremental para manter a sincronia com os modelos de classes.

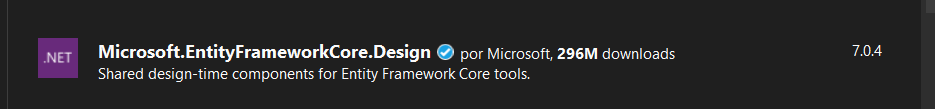
Também é possível através do Migrations fazer o Downgrade caso você deseje voltar para a versão anterior em que se encontrava.

**Adicionando o Entity Framework ao projeto:**

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

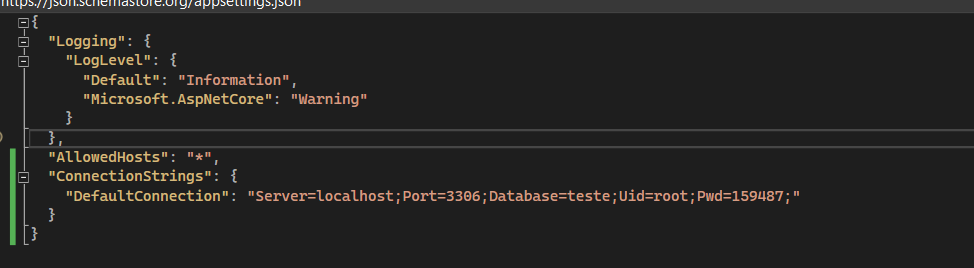




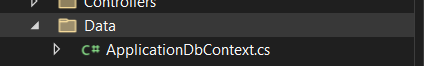




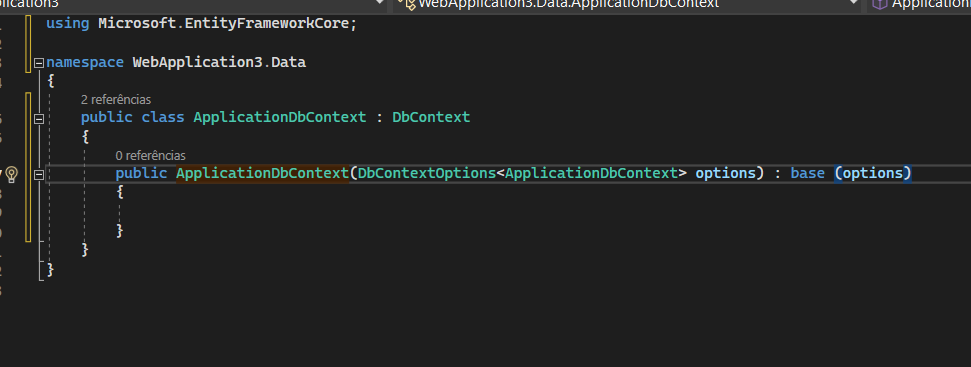
Após adicionarmos as dependências do Entity Framework devemos criar a string de conexão do banco de dados no arquivo appsettings.json:



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

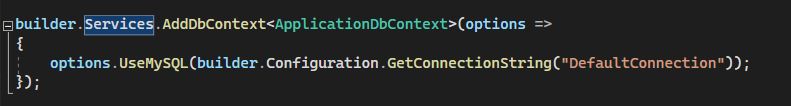


**Criando a classe de contexto do banco de dados:**

****

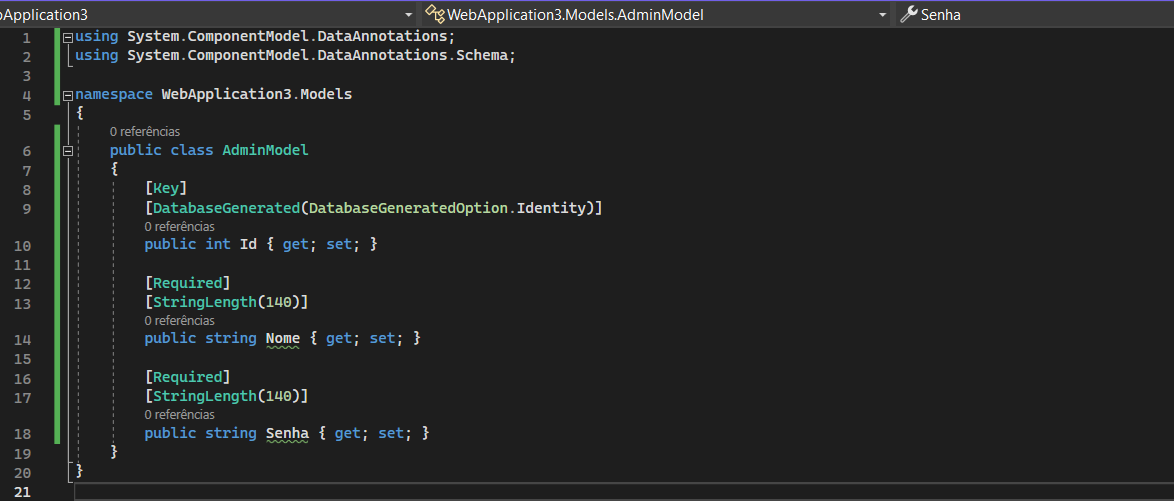
Devemos criar uma classe que extende a classe DbContext e que tenha um constructor que receba a classe DbContextOptions com type igual a classe do contexto, esse constructor deve extender a base(options) do DbContext.

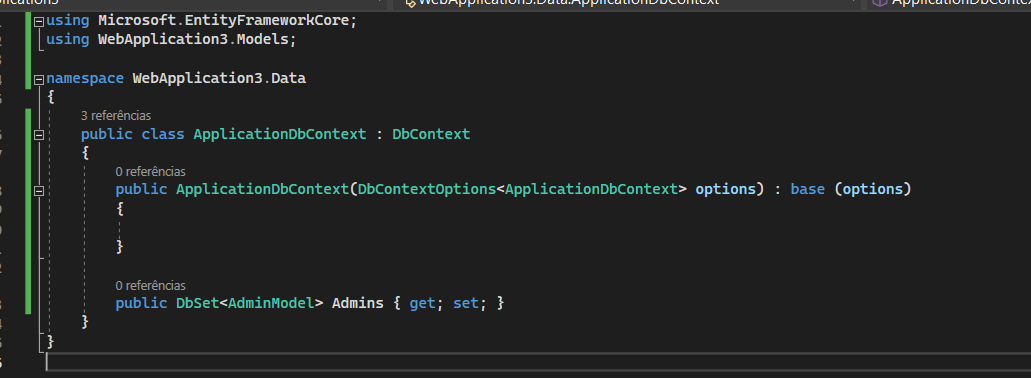
Em seguida devemos aplicar esse DbContext com a string de conexão, isso é feito adicionando uma configuração em builder na classe Program.cs:



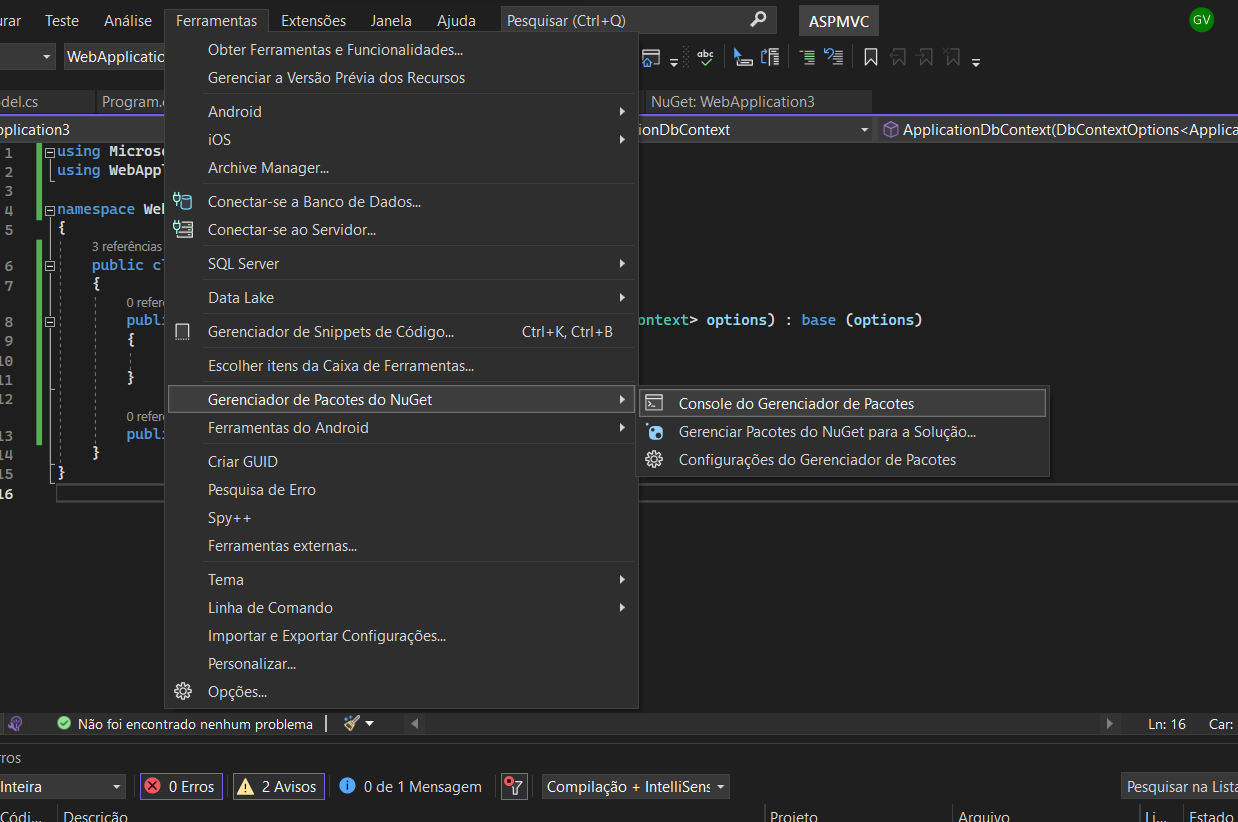
**Migrations - Model First:**

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



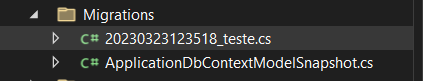


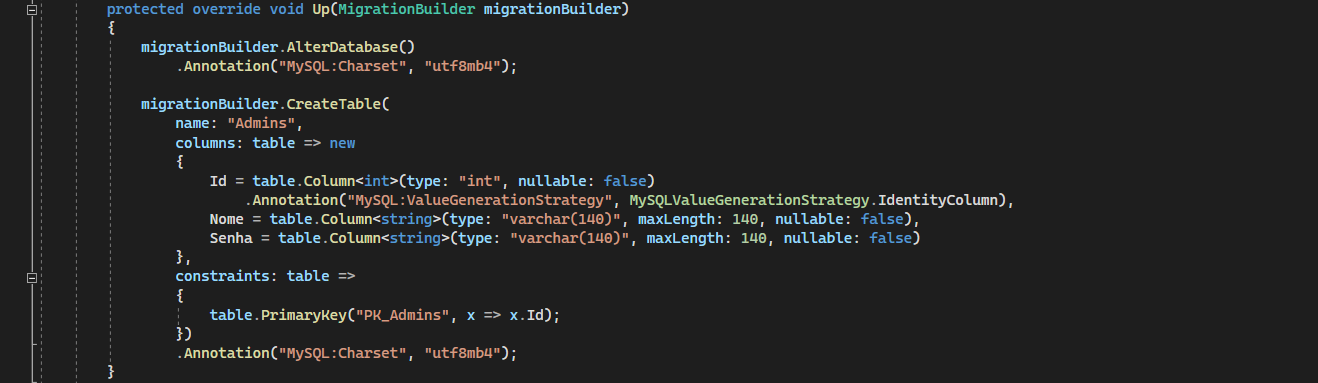
Após isso vamos usar o migrations:





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

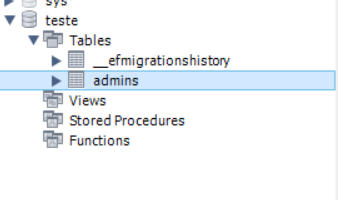




Após isso só precisamos usar o comando:



E a tabela será criada no banco de dados:



**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

**Trabalhando com o contexto:**

**Consultando dados:**

Para buscar uma entidade especifica pelo seu id podemos usar o seguinte comando:

var blog = db.Blogs.Find(blogId);

Para buscar a primeira entidade:

var blog = db.Blogs.First();

Para buscar uma entidade com base em uma propriedade:

var blog = db.Blogs.Where(b => b.Url == "http://sample.com").First();

Para buscar uma lista de entidades com base em alguma propriedade:

var blogs = db.Blogs.Where(b => b.Rating > 3).ToList();

**Adicionar dados:**

Para adicionar uma única entidade podemos utilizar o método:

var blog = new Blog { Url = "http://sample.com" };

db.Blogs.Add(blog);

db.SaveChanges();

Para adicionar varias entidades podemos utilizar o método:

var blogs = new List<Blog>

{

new Blog { Url = "http://sample1.com" },

new Blog { Url = "http://sample2.com" },

new Blog { Url = "http://sample3.com" }

};

db.Blogs.AddRange(blogs);

db.SaveChanges();

**Alterar dados:**

Para alterar os dados de uma entidade ou de varias entidades devemos primeiro buscar essas entidades, alterar seus valores e depois salva-las:

var blog = db.Blogs.First();

blog.Url = "http://newurl.com";

db.SaveChanges();

var blogs = db.Blogs.Where(b => b.Rating > 3).ToList();

foreach (var blog in blogs)

{

blog.Rating++;

}

db.SaveChanges();

**Deletando entidades:**

Para remover uma entidade ou varias também devemos buscar elas primeiro:

var blog = db.Blogs.Find(blogId);

db.Blogs.Remove(blog);

db.SaveChanges();

var blogs = db.Blogs.Where(b => b.Rating < 3).ToList();

db.Blogs.RemoveRange(blogs);

db.SaveChanges();

**Querys personalizadas:**

Podemos utilizar de querys personalizadas também, entretanto é importante nunca combinar a entrada do usuário com o texto do comando SQL, pois isso deixaria o sistema vulnerável a SQLInjection, tendo isso em vista os comandos sql podem ser feitos assim:

var blogId = 1;

db.Database.ExecuteSqlRaw("DELETE FROM Blogs WHERE BlogId = {0}", blogId);