Configurar uma aplicação .Net C# para conectar ao banco de dados Oracle – Parte 2

Autor(es)

Thiago Keller Torquato Vicco

@2TDS-2024

Sumário

Introdução

Nessa apostila iremos continuar a configuração do Entity Framework Core. Lembrando que essa apostila é somente um passo a passo para a configuração, nas aulas teremos explicações mais profundas e outros conteúdos, **por isso é de grande importância a sua presença nas aulas.**

Instalando Packages

Para a continuar a conexão do banco de dados e uso do **EntityFrameworkCore** instalaremos os seguintes pacotes via nuget.

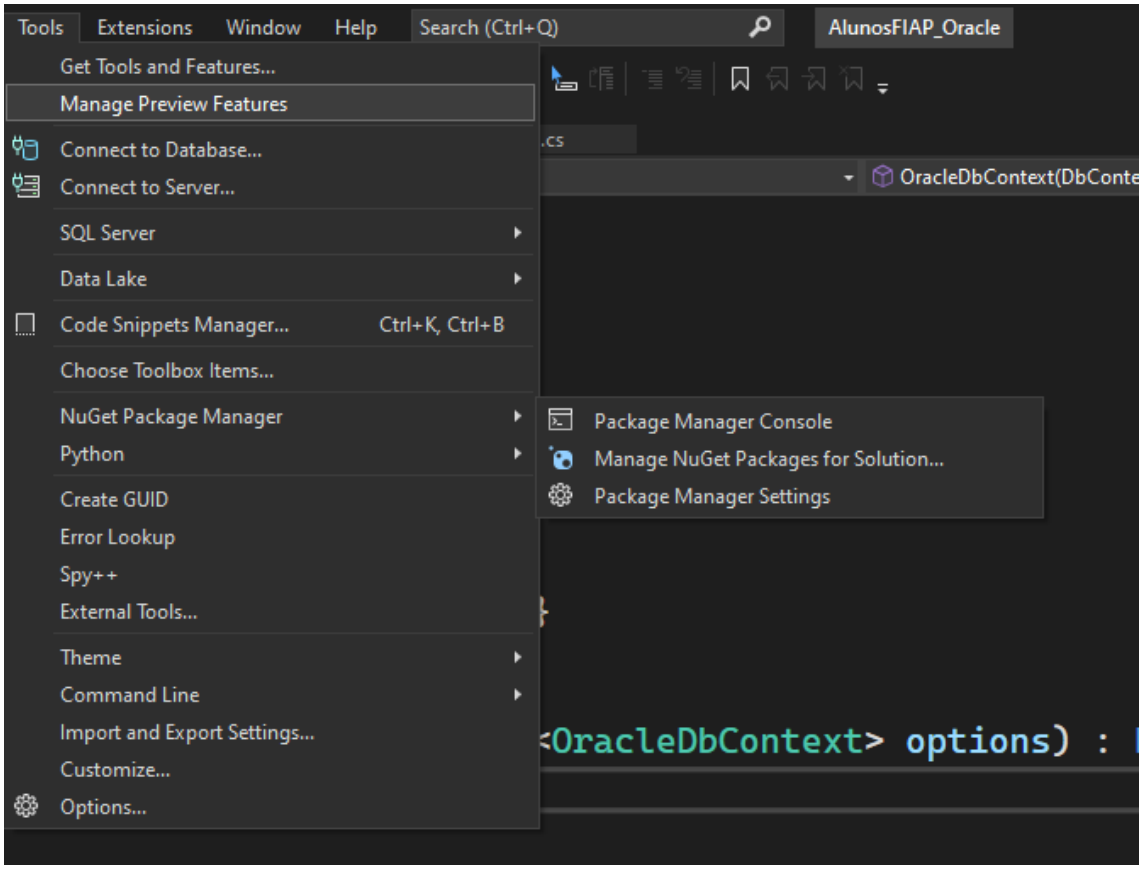
- Microsoft.EntityFrameworkCore.Tools
- Microsoft.EntityFrameworkCore.Design

Adicionando Migration

Agora que temos nosso projeto com as classes necessárias, precisamos criar uma migration.

Saiba mais sobre migration: Migrations Overview - EF Core | Microsoft Learn

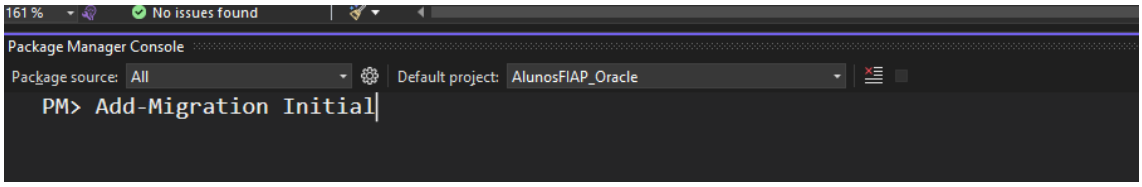
- Acesse o menu “Tools/Nuget Package Manager/Package Manager Console



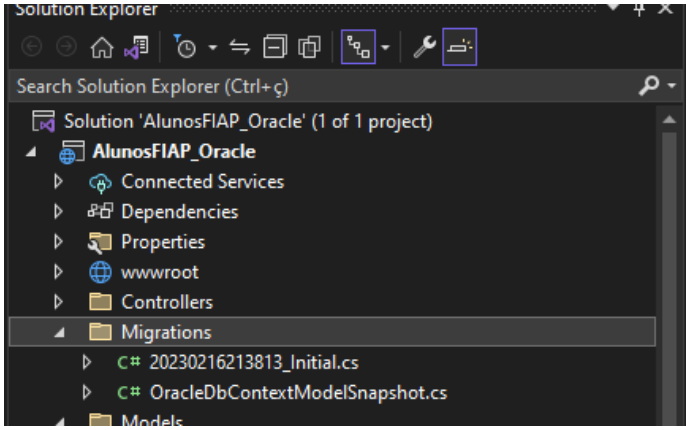
Ao clicar, irá abrir uma nova aba na parte inferior da sua IDE.

Para criar uma nova *migration* devemos usar o comando “Add-Migration <Nome>”.

Obs.: Troque a variável nome conforme necessário.



Seu projeto será compilado e o Entity irá gerar uma nova **migration** dentro da pasta **“Migrations”**



Esse novo arquivo **(*_<NOME_MIGRATION>.cs)** conterá todas as instruções que serão interpretadas pelo **EntityFrameworkCore** e executadas na sua instancia do banco de dados.

Criando o Banco

Utilizando o mesmo console (“Tools/Nuget Package Manager/Package Manager Console) utilizado anteriormente, iremos rodar o comando **“Update-Database”**. Esse comando irá “ler” e “interpretar” as ***migrations*** e irá aplicar na sua instancia de banco de dados.

Update-Database

```
PM> Update-Database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (257ms) [Parameters=[], CommandType='Text', CommandTimeout='0']
      SELECT t.table_name FROM user_tables t WHERE t.table_name = N'__EFMigrationsHistory'
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (132ms) [Parameters=[], CommandType='Text', CommandTimeout='0']
      Begin
      BEGIN
      EXECUTE IMMEDIATE 'CREATE TABLE
      "__EFMigrationsHistory" (
          "MigrationId" NVARCHAR2(150) NOT NULL,
          "ProductVersion" NVARCHAR2(32) NOT NULL,
          CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY ("MigrationId")
      )';
      END;

      End;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (16ms) [Parameters=[], CommandType='Text', CommandTimeout='0']
      SELECT t.table_name FROM user_tables t WHERE t.table_name = N'__EFMigrationsHistory'
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (19ms) [Parameters=[], CommandType='Text', CommandTimeout='0']
      SELECT "MigrationId", "ProductVersion"
      FROM "__EFMigrationsHistory"
      ORDER BY "MigrationId"
Microsoft.EntityFrameworkCore.Migrations[20402]
      Applying migration '20230217122401_Initial'.
Applying migration '20230217122401_Initial'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (25ms) [Parameters=[], CommandType='Text', CommandTimeout='0']
      BEGIN
      EXECUTE IMMEDIATE 'CREATE TABLE
      "Alunos" (
          "Id" NUMBER(10) GENERATED BY DEFAULT ON NULL AS IDENTITY NOT NULL,
          "Nome" NVARCHAR2(2000) NOT NULL,
          CONSTRAINT "PK_Alunos" PRIMARY KEY ("Id")
      )';
      END;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (32ms) [Parameters=[], CommandType='Text', CommandTimeout='0']
      INSERT INTO "__EFMigrationsHistory" ("MigrationId", "ProductVersion")
      VALUES (N'20230217122401_Initial', N'7.0.3')
Done.
PM>
```

Injeção de Dependência?

Container de injeção de dependência (IoC) é um mecanismo que automatiza o fornecimento de dependências entre os diferentes componentes de um aplicativo. Em vez de as classes criarem suas próprias dependências, o container é responsável por criar, gerenciar e injetar as instâncias corretas nos lugares apropriados.

Benefícios

- **Baixo Acoplamento**: Os containers de injeção de dependência reduzem o acoplamento entre componentes. As classes não precisam se preocupar em criar ou encontrar suas dependências, permitindo que mudanças em uma classe não afetem diretamente outras.
- **Reusabilidade**: Dependências podem ser configuradas uma vez e reutilizadas em todo o aplicativo. Isso promove a consistência e evita a duplicação de configurações.
- **Testabilidade**: Facilita a realização de testes unitários, pois as dependências podem ser facilmente substituídas por *mocks*.
- **Configuração Centralizada**: A configuração das dependências é centralizada em um local, tornando mais fácil gerenciar mudanças e atualizações.
- **Flexibilidade**: Permite a alteração de dependências sem alterar o código das classes que as usam, tornando a aplicação mais adaptável a mudanças futuras.

Ciclos de Vida

- **AddScoped**: Instâncias são criadas uma vez por solicitação (aplicativo da web) e compartilhadas entre componentes durante a solicitação.
- **AddTransient**: Uma nova instância é criada sempre que é solicitada.
- **AddSingleton**: Uma única instância é criada e compartilhada durante toda a vida do aplicativo.

Próximos passos

Nas próximas aulas nós iremos criar e configurar os repositórios e controladores.

Não percam!!!!

Links interessantes

Curso Online Entity Framework Core: banco de dados de forma eficiente | Alura

Visão geral do Entity Framework Core – EF Core | Microsoft Learn

Se chegaram a essa parte com sucesso, meus parabéns!! Em breve iremos melhorar essa aplicação.

LET'S ROCK THE FUTURE