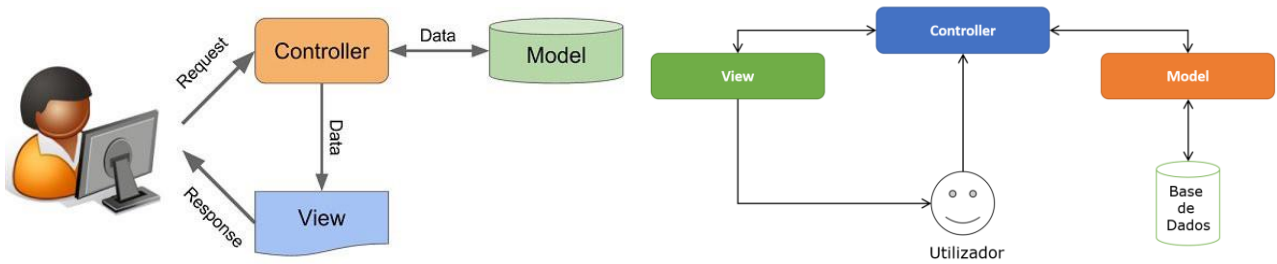


:: ASP.NET Core MVC :: Base dados – Code First :: Exemplo

Objetivo:

- Usar a Entity Framework
- Criar um projeto Web ASP.NET Core MVC
- Base de dados - Abordagem Code First
- CRUD (Create, Read, Update, Delete)



Criar aplicação ASP.NET Core

1. Criar o projeto ASP.NET Core baseado no Template MVC (01-ASP.NET Core- Conceitos fundamentais) ou Empty (02-ASP.NET Core-Projeto de raiz)

Obs: pode usar o projeto modelo disponível no Moodle (00_AspNetCore_MVC_base_bootstrap.zip).

Configurar aplicação para: ASP.NET Core MVC

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();
    app.UseStaticFiles();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

ViewImports

Views_ViewImports.cshtml

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

ViewStart

Views_ViewStart.cshtml

```
@{
    Layout = "_Layout";
}
```

Criar o Layout

wwwroot\css\site.css

```
.rodape {
    text-align: center;
    /*Colocar o rodapé na base (fundo do browser)*/
    position: absolute;
    bottom: 0;
    width: 100%;
}
```

Views\Shared_Layout.cshtml
<pre> <!DOCTYPE html> <html> <head> <meta name="viewport" content="width=device-width" /> <title>@ViewBag.Title</title> <link href="/css/site.css" rel="stylesheet" /> <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-+0n0xVW2eSR5OomGNYDnhzAbdSOXxcvSN1TPPrVMTNDbiYZCxbY0017+AMvyTG2x" crossorigin="anonymous"> <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.bundle.min.js" integrity="sha384- gtEjrD/SeCtmISKJKNUAAKMoLD0//ElJ19smozuHV6z3Iehds+3U1b9Bn9Plx0x4" crossorigin="anonymous"></script> </head> <body> <header> <nav class="navbar navbar-expand-lg navbar-light bg-light"> <div class="container-fluid"> ASP.NET Core <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation"> </button> <div class="collapse navbar-collapse" id="navbarNav"> <ul class="navbar-nav"> <li class="nav-item"> Home <li class="nav-item"> Students <li class="nav-item"> Courses <li class="nav-item"> Enrollments </div> </div> </nav> </header> <div class="container"> @RenderBody() </div> <footer class="rodape"> <hr /> &copy; 2021-TPW </footer> </body> </html> </pre>

2. Adicionar um Controller (03-ASP.NET Core-Controllers)

Controllers\HomeController.cs
<pre> namespace _07_ASP.NET_core_Forms.Controllers { public class HomeController : Controller { public IActionResult Index() { return View(); } } } </pre>

3. Adicionar uma View para a ação Index (04-ASP.NET Core-Views)

Views\Home\index.cshtml
<pre> @{ ViewData["Title"] = "Index"; } <h1>Index</h1> </pre>

Code First

Neste exemplo vamos usar a abordagem **Code First** que permite a criação de uma nova base de dados e posterior atualização a partir do modelo de dados (classes).

1. Devemos instalar o EF (Entity Framework)
2. Criar o modelo de dados
3. Criar e registrar o contexto de base de dados (DbContext)
4. Criar as Migrações ou Class de inicialização

1. Instalar (adicionar ao projeto) o EF (Entity Framework)

Linha de comando (Visual Studio)

Tools | NuGet Package Manager | Package Manager Console

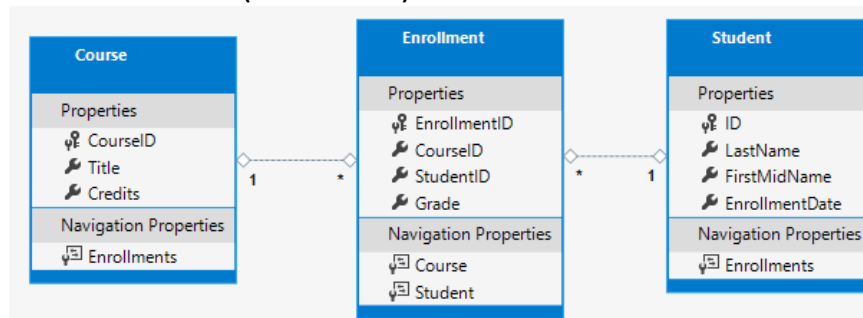
Podemos usar **Tab** para pedir ajuda de contexto

```
Install-Package Microsoft.EntityFrameworkCore -Version 5.0.6
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Microsoft.EntityFrameworkCore.Design
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Ou Assistente

Tools | NuGet Package Manager | Manager Nugets Packages for Solution

2. Criar o Modelo de dados (Entidades)



Relacionamentos:

- Uma relação um-para-muitos entre as entidades **Student** e **Enrollment**. Um **Student** pode estar matriculado em qualquer número de **Course**.
- Uma relação um-para-muitos entre as entidades de **Course** e **Enrollment**. Um **Course** pode ter qualquer número de **Student** matriculados nele.

Criar classe Student.cs , apenas com as propriedades

Models\Student.cs

Student

Properties

ID

LastName

FirstMidName

EnrollmentDate

Navigation Properties

Enrollments

```
public class Student
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }
}
```

Criar classe Enrollment.cs, apenas com as propriedades

Models\Enrollment.cs

Enrollment

Properties

EnrollmentID

CourseID

StudentID

Grade

Navigation Properties

Course

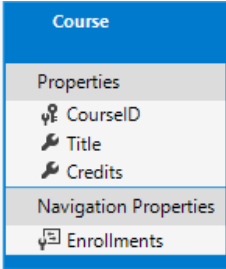
Student

```
public enum Grade
{
    A, B, C, D, F
}

public class Enrollment
{
    public int EnrollmentID { get; set; }
    public int CourseID { get; set; }
    public int StudentID { get; set; }
    public Grade? Grade { get; set; }
}
```

Criar classe Course.cs, apenas com as propriedades

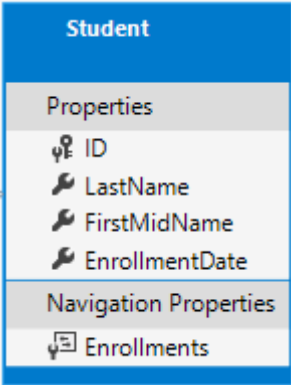
Models\Course.cs



```
public class Course
{
    public int CourseID { get; set; }
    public string Title { get; set; }
    public int Credits { get; set; }
}
```

Criar as ligações entre tabelas (1..1 e 1..n)

Models\Student.cs



```
public class Student
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }

    public ICollection<Enrollment> Enrollments { get; set; }
}
```

Models\Enrollment.cs

Enrollment	
Properties	
PK	EnrollmentID
FK	CourseID
FK	StudentID
FK	Grade
Navigation Properties	
FK	Course
FK	Student

```
public enum Grade
{
    A, B, C, D, F
}

public class Enrollment
{
    public int EnrollmentID { get; set; }
    public int CourseID { get; set; }
    public int StudentID { get; set; }
    public Grade? Grade { get; set; }

    public Student Student { get; set; }
    public Course Course { get; set; }
}
```

Models\Course.cs

Course	
Properties	
PK	CourseID
FK	Title
FK	Credits
Navigation Properties	
FK	Enrollments

```
public class Course
{
    public int CourseID { get; set; }
    public string Title { get; set; }
    public int Credits { get; set; }

    public ICollection<Enrollment> Enrollments { get; set; }
}
```

3. Criar o contexto de base de dados (DbSchoolContext)

Data\DbSchoolContext.cs
<pre> public class DbSchoolContext : DbContext { public DbSchoolContext(DbContextOptions<DbSchoolContext> options) : base(options) { } public DbSet<Course> Courses { get; set; } public DbSet<Enrollment> Enrollments { get; set; } public DbSet<Student> Students { get; set; } protected override void OnModelCreating(ModelBuilder modelBuilder) { modelBuilder.Entity<Course>().ToTable("Course"); modelBuilder.Entity<Enrollment>().ToTable("Enrollment"); modelBuilder.Entity<Student>().ToTable("Student"); } } </pre>

4. Registrar o Contexto de base de dados

ConnectionString

Data Source=(LocalDB)\MSSQLLocalDB;Initial Catalog= DbBlogs;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite; MultiSubnetFailover=False

Por questões de segurança, e também para mais fácil gestão, vamos colocar a Connection String num ficheiro externo de configurações: appsettings.json
A base dados será criada com o nome de “DbUniversity”

appsettings.json
<pre> { "Logging": { "LogLevel": { "Default": "Information", "Microsoft": "Warning", "Microsoft.Hosting.Lifetime": "Information" } }, "AllowedHosts": "*", "ConnectionStrings": { "SchoolConnection": "Data Source=(LocalDB)\\MSSQLLocalDB;Initial Catalog= DbUniversity; Integrated Security=True; Connect Timeout=30; Encrypt=False; TrustServerCertificate=False;ApplicationIntent=ReadWrite; MultiSubnetFailover=False" } } </pre>

Startup.cs

```
public IConfiguration Configuration { get; }
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<DbBlogContext>(options => options.UseSqlServer(Configuration.GetConnectionString("SchoolConnection")));

    services.AddControllersWithViews();
}
```

Neste exemplo, **em vez de usar migrações de dados** (para criar a BD e respetivas tabelas), vamos usar uma classe para inicializar a Base dados.

Migrações

Comando Add-Migration

```
Add-Migration InitialMigration -o Data/Migrations
```

Comando Update-Database

```
Update-Database
```

Classe para inicializar a Base dados

Data\DbInitializer.cs

```
public class DbInitializer
{
    public static void Initialize(DbSchoolContext context)
    {
        context.Database.EnsureCreated();
    }
}
```

Program.cs

```
public static void Main(string[] args)
{
    //CreateHostBuilder(args).Build().Run();

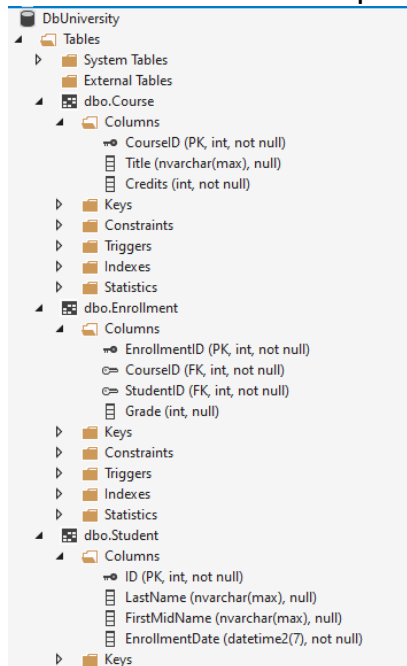
    var host = CreateHostBuilder(args).Build();

    CreateDbIfNotExists(host);

    host.Run();
}

private static void CreateDbIfNotExists(IHost host)
{
    using (var scope = host.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        try
        {
            var context = services.GetRequiredService<DbSchoolContext>();
            DbInitializer.Initialize(context);
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred creating the DB.");
        }
    }
}
```

Se correremos a aplicação a BD será criada com as respetivas tabelas.



Podemos povoar as tabelas (ao criar a BD) com registos

Data\DbInitializer.cs
<pre> public class DbInitializer { public static void Initialize(DbSchoolContext context) { context.Database.EnsureCreated(); // Look for any students. if (context.Students.Any()) { return; // DB has been seeded } var students = new Student[] { new Student{FirstMidName="Carson", LastName="Alexander", EnrollmentDate=DateTime.Parse("2005-09-01")}, new Student{FirstMidName="Meredith", LastName="Alonso", EnrollmentDate=DateTime.Parse("2002-09-01")}, new Student{FirstMidName="Arturo", LastName="Anand", EnrollmentDate=DateTime.Parse("2003-09-01")}, new Student{FirstMidName="Gytis", LastName="Barzdukas", EnrollmentDate=DateTime.Parse("2002-09-01")}, new Student{FirstMidName="Van", LastName="Li", EnrollmentDate=DateTime.Parse("2002-09-01")}, new Student{FirstMidName="Peggy", LastName="Justice", EnrollmentDate=DateTime.Parse("2001-09-01")}, new Student{FirstMidName="Laura", LastName="Norman", EnrollmentDate=DateTime.Parse("2003-09-01")}, new Student{FirstMidName="Nino", LastName="Olivetto", EnrollmentDate=DateTime.Parse("2005-09-01")} }; foreach (Student s in students) { context.Students.Add(s); } context.SaveChanges(); var courses = new Course[] { new Course{CourseID=1050, Title="Chemistry", Credits=3}, new Course{CourseID=4022, Title="Microeconomics", Credits=3}, new Course{CourseID=4041, Title="Macroeconomics", Credits=3}, new Course{CourseID=1045, Title="Calculus", Credits=4}, new Course{CourseID=3141, Title="Trigonometry", Credits=4}, new Course{CourseID=2021, Title="Composition", Credits=3}, new Course{CourseID=2042, Title="Literature", Credits=4} }; foreach (Course c in courses) { context.Courses.Add(c); } context.SaveChanges(); var enrollments = new Enrollment[] { new Enrollment{StudentID=1, CourseID=1050, Grade=Grade.A}, new Enrollment{StudentID=1, CourseID=4022, Grade=Grade.C}, new Enrollment{StudentID=1, CourseID=4041, Grade=Grade.B}, new Enrollment{StudentID=2, CourseID=1045, Grade=Grade.B}, new Enrollment{StudentID=2, CourseID=3141, Grade=Grade.F}, new Enrollment{StudentID=2, CourseID=2021, Grade=Grade.F}, new Enrollment{StudentID=3, CourseID=1050}, new Enrollment{StudentID=4, CourseID=1050}, new Enrollment{StudentID=4, CourseID=4022, Grade=Grade.F}, new Enrollment{StudentID=5, CourseID=4041, Grade=Grade.C}, new Enrollment{StudentID=6, CourseID=1045}, new Enrollment{StudentID=7, CourseID=3141, Grade=Grade.A}, }; foreach (Enrollment e in enrollments) { context.Enrollments.Add(e); } context.SaveChanges(); } } </pre>

A primeira vez que a aplicação é executada a BD é criada com as respetivas tabelas e com os registos.

Se mudar o modelo:

- Eliminar a base de dados
- E correr novamente a aplicação.

DbUniversity

- Tables
 - System Tables
 - External Tables
 - dbo.Course
 - Columns
 - CourseID (PK, int, not null)
 - Title (nvarchar(max), null)
 - Credits (int, not null)
 - Keys
 - Constraints
 - Triggers
 - Indexes
 - Statistics
 - dbo.Enrollment
 - Columns
 - EnrollmentID (PK, int, not null)
 - CourseID (FK, int, not null)
 - StudentID (FK, int, not null)
 - Grade (int, null)
 - Keys
 - Constraints
 - Triggers
 - Indexes
 - Statistics
 - dbo.Student
 - Columns
 - ID (PK, int, not null)
 - LastName (nvarchar(max), null)
 - FirstMidName (nvarchar(max), null)
 - EnrollmentDate (datetime2(7), not null)
 - Keys

	ID	LastName	FirstMidName	EnrollmentDate
▶	1	Alexander	Carson	01/09/2005 00:00:00
	2	Alonso	Meredith	01/09/2002 00:00:00
	3	Anand	Arturo	01/09/2003 00:00:00
	4	Barzdukas	Gytis	01/09/2002 00:00:00
	5	Li	Yan	01/09/2002 00:00:00
	6	Justice	Peggy	01/09/2001 00:00:00
	7	Norman	Laura	01/09/2003 00:00:00
	8	Olivetto	Nino	01/09/2005 00:00:00
*	NULL	NULL	NULL	NULL

	CourseID	Title	Credits
▶	1045	Calculus	4
	1050	Chemistry	3
	2021	Composition	3
	2042	Literature	4
	3141	Trigonometry	4
	4022	Microeconomics	3
	4041	Macroeconomics	3
*	NULL	NULL	NULL

	EnrollmentID	CourseID	StudentID	Grade
▶	1	1050	4	NULL
	2	1050	3	NULL
	3	2021	2	4
	4	3141	2	4
	5	1045	2	1
	6	4041	1	1
	7	4022	1	2
	8	1050	1	0
	9	4022	4	4
	10	4041	5	2
	11	1045	6	NULL
	12	3141	7	0
*	NULL	NULL	NULL	NULL

Controllers e Views automáticos (Scaffold)

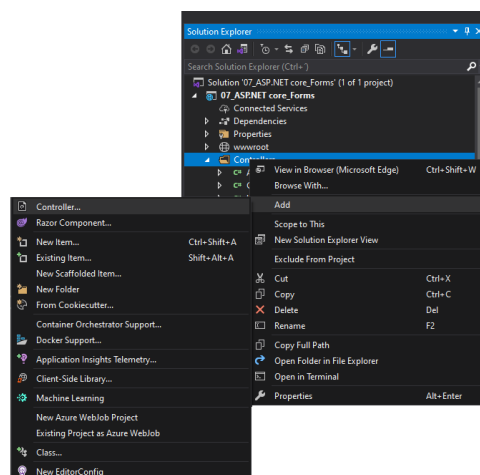
Scaffolding é uma framework de geração de código que adiciona códigos automaticamente e cria páginas de visualização (Views) e controladores (Controllers).

O Scaffolding torna o trabalho do programador mais fácil criando controladores (Controllers) e páginas de visualização (Views) para o modelo de dados (Models).

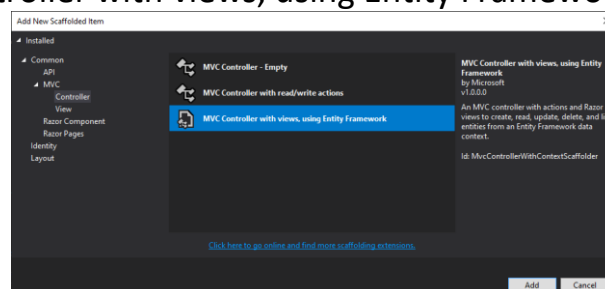
Gera códigos e páginas para a operação CRUD (**C**reate (Criar), **R**ead (Ler), **U**ppdate (Atualizar) e **D**elete (Excluir)).

Controllers e Views automáticos (Entity Framework)

Em Solution Explorer, sobre o nome da pasta “*Controllers*”, fazer Tecla direita do rato e seleccionar **Add -> New Scaffolded ou Controller**, como na figura a seguir:



Selecionar “MVC Controller with views, using Entity Framework”



Para cada classe do modelo de dados, criar um Controller e respectivas Views.