











EF Core - Code First

```
public class Car
{
    0 referencias
    public int Id { get; set; }
    [Required(ErrorMessage = "Campo Obligatorio")]
    [MaxLength(15, ErrorMessage = "El campo no puede tener más de 15 caracteres")]
    [MinLength(3, ErrorMessage = "El campo ha de tener mínimo 3 caracteres")]
    [DisplayName("Modelo")]
    0 referencias
    public string Model { get; set; }
    [Required(ErrorMessage = "Campo Obligatorio")]
    [MaxLength(15, ErrorMessage = "El campo no puede tener más de 15 caracteres")]
    [MinLength(3, ErrorMessage = "El campo ha de tener mínimo 3 caracteres")]
    [DisplayName("Marca")]
    0 referencias
    public string Brand { get; set; }
    [StringLength(7, ErrorMessage = "La matrícula ha de tener 7 caracteres", MinimumLength = 7)]
    [DisplayName("Matrícula")]
    0 referencias
    public string CarCode { get; set; }
    [DataType(DataType.Date)]
    0 referencias
    public DateTime PurchaseDate { get; set; }
    [Required(ErrorMessage = "Campo Obligatorio")]
    [Range(1, 9, ErrorMessage = "El número de asientos ha de estar entre 1 y 9")]
    [DisplayName("Número de Asientos")]
    0 referencias
    public int SeatNum { get; set; }
}
```



	Microsoft.EntityFrameworkCore.InMemory by Microsoft	6.0.25
	In-memory database provider for Entity Framework Core (to be used for testing purposes).	8.0.0
	Microsoft.EntityFrameworkCore.SqlServer por Microsoft	6.0.9
	Microsoft SQL Server database provider for Entity Framework Core.	
	Microsoft.EntityFrameworkCore.Tools por Microsoft	6.0.9
	Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	
	Microsoft.VisualStudio.Web.CodeGeneration.Design por Microsoft	6.0.10
	Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	



Crear contexto en Data

```
public partial class MyDbContext:DbContext
{
    public MyDbContext (DbContextOptions<MyDbContext> options) :
    base(options)
    {

    }

    public DbSet<Game>? Game { get; set; }
    public DbSet<Genre>? Genre { get; set; }

    protected override void OnModelCreating (ModelBuilder modelBuilder)
    {
        // Seed data
        modelBuilder.Entity<Genre>().HasData
        (
            new Genre { Id = 1, Name = "Shooter" },
            new Genre { Id = 2, Name = "RPG" }
        );

        modelBuilder.Entity<Game>().HasData
        (
            new Game { Id = 1, Title = "Doom", GenreId = 1 },
            new Game { Id = 2, Title = "Final Fantasy", GenreId = 2 }
        );

        OnModelCreatingPartial(modelBuilder);
    }

    partial void OnModelCreatingPartial (ModelBuilder modelBuilder);
}
```

En program.cs

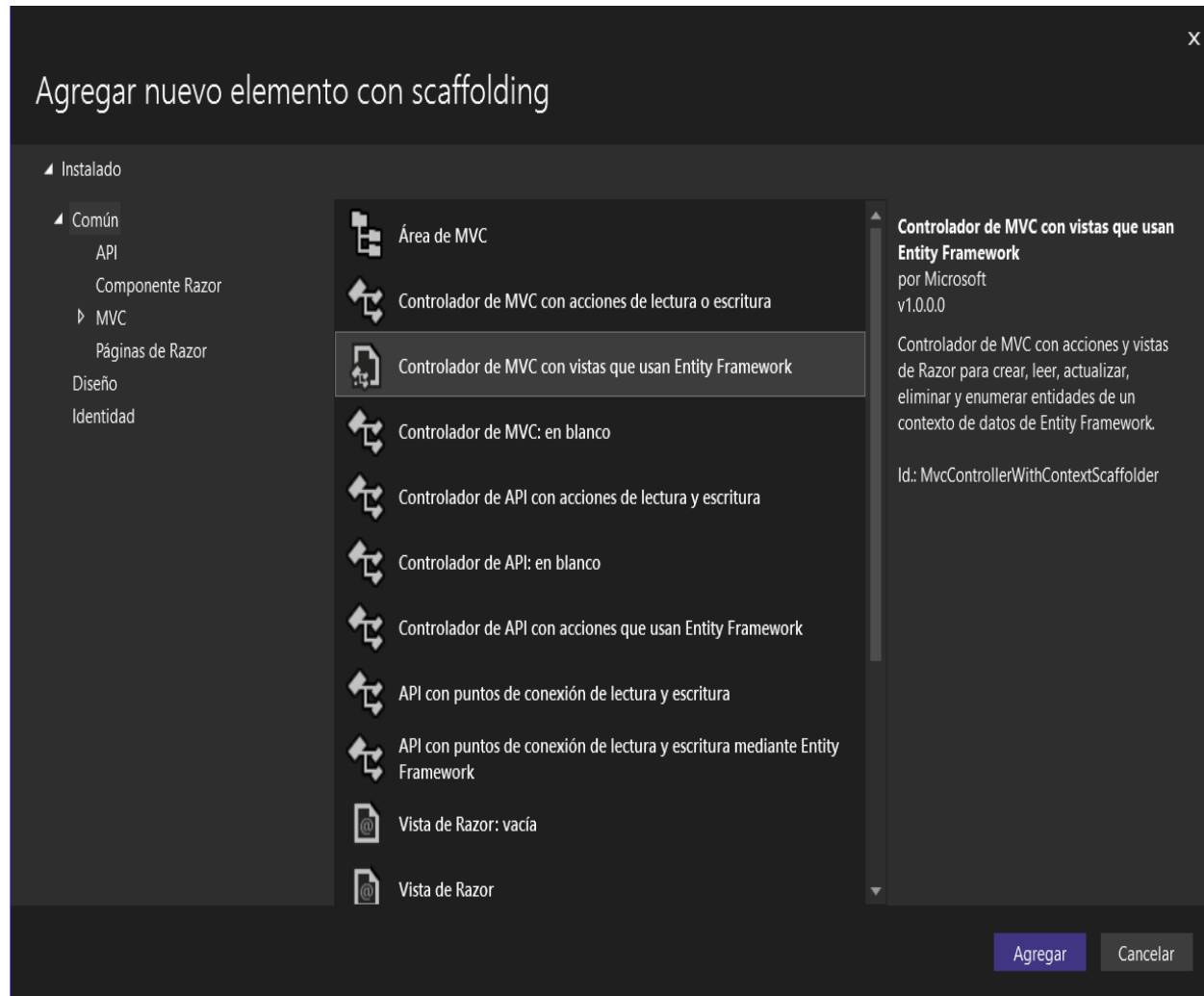
```
builder.Services.AddDbContext<MyDbContext>(opt
ions =>
{
    options.UseInMemoryDatabase("GameGenre");
});
```

Para crear controladores

- 1.- Creas un controlador que te de error, para crear context en appsettings.json
- 2.- Ahora si creas lo controlades con el data creado.
- 3.- Comentas in memory y haces options.SqlServer...



EF Core - Code First





EF Core - Code First

Agregar Controlador de MVC con vistas que usan Entity Framework

Clase de modelo: Car (CodeFirst.Models)

Clase de contexto de datos: CodeFirst.Data.CodeFirstContext

Vistas

- ☒ Generar vistas
- ☒ Hacer referencia a bibliotecas de scripts
- ☒ Usar página de diseño

(Dejar en blanco si se define en un archivo _viewstart de Razor)

Nombre de controlador: CarsController

Agregar Cancelar

Modelo sobre el que se va a realizar el Scaffold

Pulsando + nos muestra la clase de contexto de datos por defecto de la aplicación, que es la que habrá que elegir



EF Core - Code First

- 1.- Haces el scaffolding para actualizar el contexto con el objeto ya creado.
- 2.- Comprueba que el objeto esté en el contexto.
- 3.- Add-Migration InitialCreate
- 4.- Borra las tablas que no necesites.
- 5.- Update-Database

```
2 references
public virtual DbSet<Review> Review { get; set; }

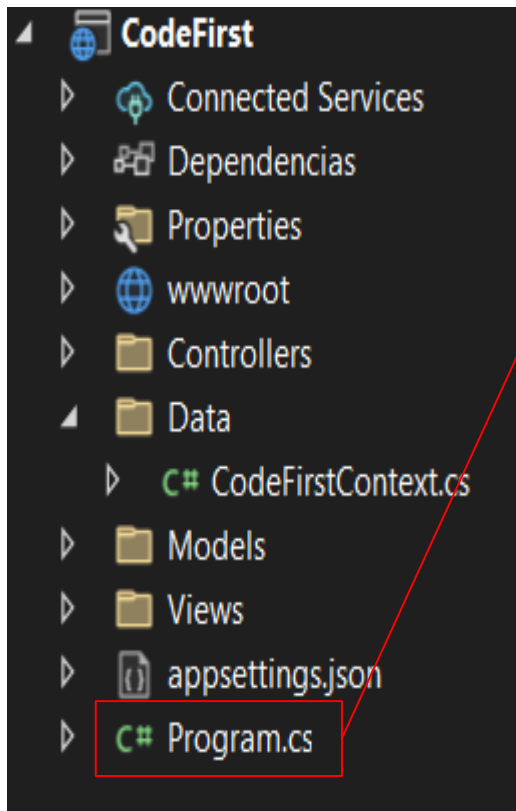
0 references
protected override void OnModelCreating (ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Review>().HasData
    (
        new Review { Id = 1, Title = "Crítica al artista 155", Description = "Esto es una crítica al artista 155", Value = 1, ArtistId = 155 },
        new Review { Id = 2, Title = "Crítica al artista 212", Description = "Esto es una crítica al artista 212", Value = 5, ArtistId = 212 }
    );
}
```

```
modelBuilder.Entity<Review>().HasData
(
    new Review { Id = 1, Title = "Crítica al artista 155", Description = "Esto es una crítica al artista 155", Value = 1,
    ArtistId = 155 },
    new Review { Id = 2, Title = "Crítica al artista 212", Description = "Esto es una crítica al artista 212", Value = 5,
    ArtistId = 212 }
);
```



EF Core - Code First

MVC Scaffolding - Elementos Creados/Actualizados



Se actualiza este archivo para registrar el contexto de la base de datos

```
builder.Services.AddDbContext<CodeFirstContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("CodeFirstC

builder.Services.AddDbContext<ChinookContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("DbContext"));
});
```



EF Core - Code First

MVC Scaffolding - Elementos Creados/Actualizados

```
public class CarsController : Controller
{
    private readonly CodeFirstContext _context;

    0 referencias
    public CarsController(CodeFirstContext context)
    {
        _context = context;
    }

    // GET: Cars
    3 referencias
    public async Task<IActionResult> Index()
    {
        return View(await _context.Car.ToListAsync());
    }
}
```

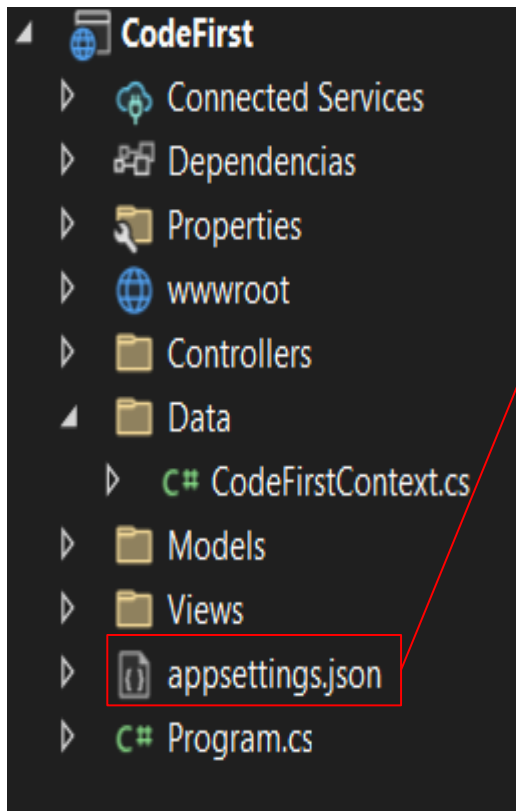
El controlador declara una referencia al DbContext de la aplicación y la inicializa en el constructor.
Se utilizará para realizar los accesos a la Base de Datos

Cuando trabajamos con bases de datos, como las peticiones son asíncronas, todas las acciones del controlador que accedan a esta tendrán que trabajar de forma asíncrona. [Más información.](#) [Y más.](#)



EF Core - Code First

MVC Scaffolding - Elementos Creados/Actualizados



Se agrega la cadena de conexión de la base de datos

```
"ConnectionStrings": {  
  "CodeFirstContext": "Server=(localdb)\\mssqllocaldb;Database=CodeFirst.Data;  
  }  
}  
  
"ConnectionStrings": {  
  "DbContext": "Server=(localdb)\\MSSQLLocalDB; Database=Chinook; Trusted_Connection=True"  
}
```



Add-Migration InitialCreate

Update-Database

EF Core - Code First

Si hay mas de un contexto:

Add-Migration InitialCreate -context NombreContexto

Migraciones

Para realizar la primera migración hay que introducir los siguientes comandos en la Consola de Administrador de paquetes

```
Add-Migration InitialCreate  
Update-Database
```

Este comando crea el archivo de la primera migración con el nombre: `<timestamp>_InitialCreate.cs` en la carpeta Migrations. Esta clase tiene la información para crear las tablas que se definan.

Este comando actualiza la base de datos con lo que se indique en los archivos de migración

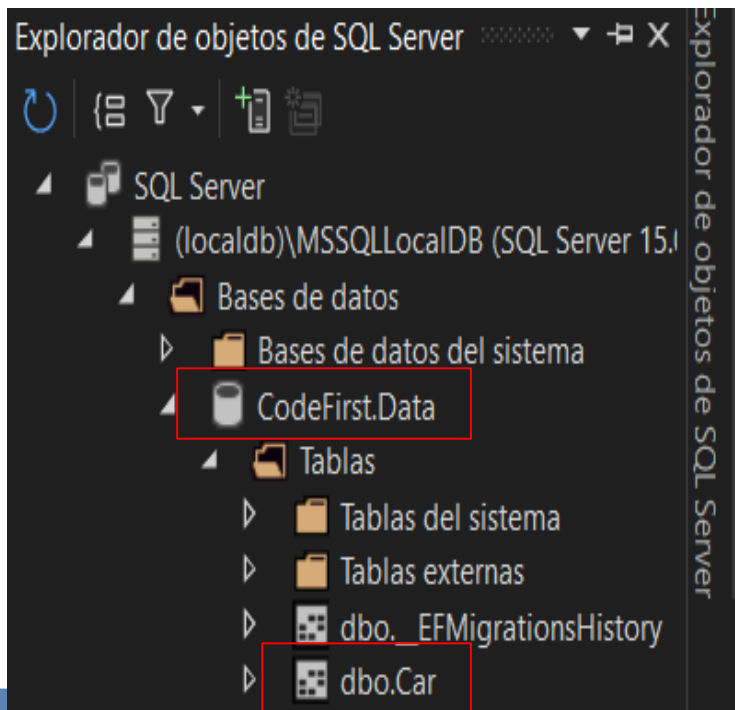
```
Migrations  
└─ C# 20221004193346_InitialCreate.cs  
└─ C# CodeFirstContextModelSnapshot.cs
```



EF Core - Code First

Migraciones

Si ahora exploramos SQL Server LocalDB veremos que se ha creado la base de datos y la tabla del modelo.



Ahora actualizar, el context



EF Core - Code First

Seeders - Clase SeedData

```

public static void Initialize (IServiceProvider
serviceProvider)
{
    using (var context = new ChinookContext
(serviceProvider.GetRequiredService<DbContext>
Options<ChinookContext>>()))
    {
        if (context.Review.Any())
        {
            return;
        }

        context.Review.AddRange
        (
            new Review { Id = 1, Title = "Crítica
al artista 155", Description = "Esto es una crítica
al artista 155", Value = 1, ArtistId = 155 },
            new Review { Id = 2, Title = "Crítica
al artista 212", Description = "Esto es una crítica
al artista 212", Value = 5, ArtistId = 212 }
        );

        context.SaveChanges();
    }
}

```

```

using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using CodeFirst.Data;
using System;
using System.Linq;

namespace CodeFirst.Models
{
    0 referencias
    public static class SeedData
    {
        0 referencias
        public static void Initialize(IServiceProvider serviceProvider)
        {
            using (var context = new CodeFirstContext(
                serviceProvider.GetRequiredService<
                    DbContextOptions<CodeFirstContext>>()))
            {
                // Look for any movies.
                if (context.Car.Any())
                {
                    return; // DB has been seeded
                }

                context.Car.AddRange(
                    new Car
                    {
                        /*Inicializar campos del objeto del modelo*/
                    }
                );
                context.SaveChanges();
            }
        }
    }
}

```



EF Core - Code First

Seeders - Program.cs

Editamos el archivo con el siguiente código.

```
var app = builder.Build();  
  
using (var scope = app.Services.CreateScope())  
{  
    var services = scope.ServiceProvider;  
  
    SeedData.Initialize(services);  
}
```

```
using (var scope =  
app.Services.CreateScope())  
{  
    var services = scope.ServiceProvider;  
    SeedData.Initialize(services);  
}
```

Si ahora ejecutamos la aplicación veremos que nos lista los elementos del modelo que hemos definido en la clase *SeedData*