



# UT02\_ 02 Persistencia de Datos

Desarrollo Web en  
Entorno Servidor



# Contenidos

- Introducción
- Tipos de SGBD a usar
- Entity Framework Core
  - ¿Qué es EF Core?
  - ¿Qué es un ORM?
  - Code First Vs Database First
- Code First
  - Crear un modelo
  - Scaffold de un proyecto (Elementos creados)
  - Migraciones
  - Seeders



# Introducción

Hasta ahora hemos trabajado con datos que tan solo han existido en la memoria volátil del ordenador.

Esto provoca que los datos se pierdan cuando se reinicia la aplicación.

Para evitar esto, toda aplicación del lado servidor utiliza BBDD para hacer que la información con la que trabaja sea persistente.

Gracias a esto los datos perdurarán en el tiempo más allá del tiempo que la aplicación esté en ejecución.

Las aplicaciones, una vez conectadas al SGBD, podrán realizar operaciones CRUD de los modelos con los que trabajan.



# Entity Framework Core

Entity Framework Core (en adelante EF Core) es un moderno **ORM** (Object-Relation Mapper) para .NET

Esta herramienta permite mapear las clases de los modelos con las tablas de la base de datos:

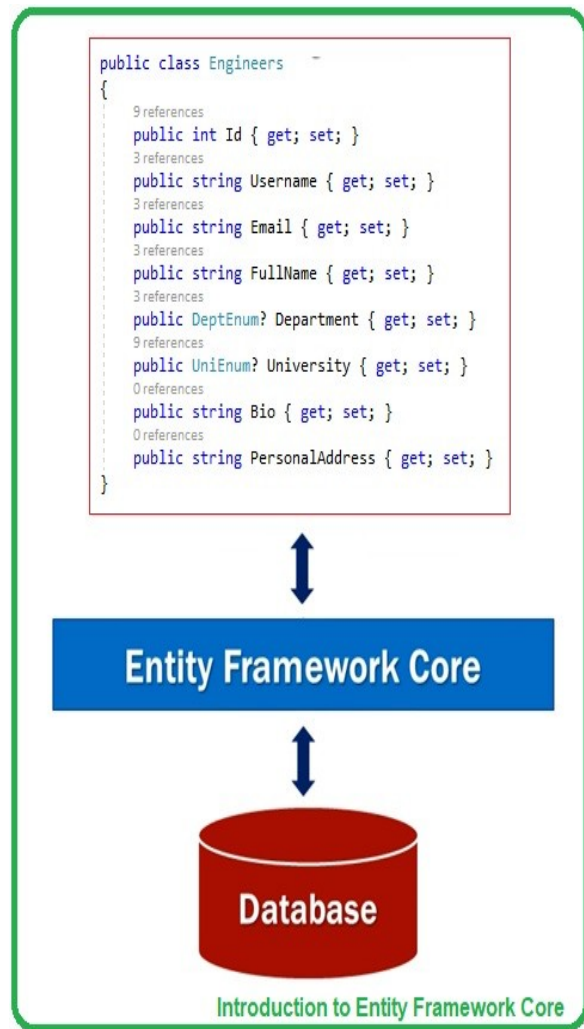
- Permite a los desarrolladores trabajar con una BD usando objetos .NET
- Permite prescindir de la mayor parte del código de acceso a datos que normalmente es necesario escribir.

Se compone de un conjunto de librerías que tendrán que ser importadas en el proyecto que necesite utilizarlo.

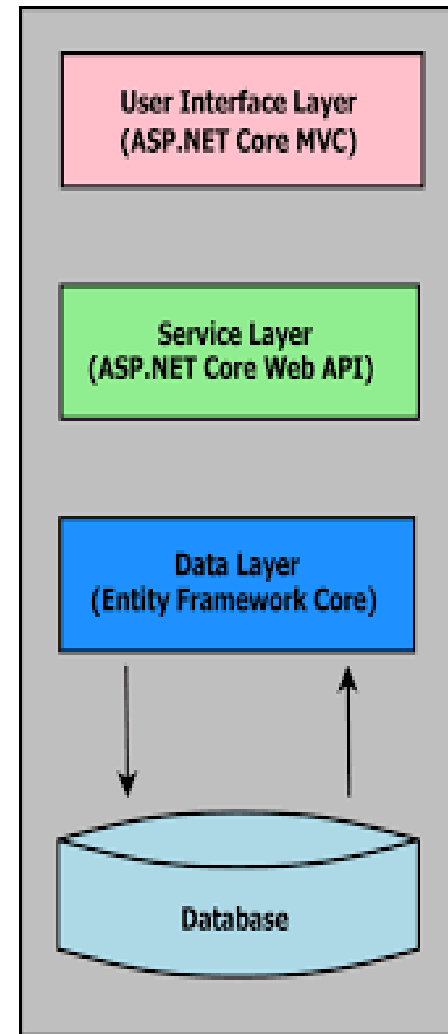
Es el software que hace de puente entre la aplicación y la BD, facilitando la recuperación y almacenamiento de la información.



# Entity Framework Core



Re  
f1



Re  
f2



# Entity Framework Core

EF Core puede tener acceso a muchas bases de datos diferentes a través de bibliotecas de complementos denominadas proveedores de bases de datos (database providers).

Como veremos más adelante, estas bibliotecas se importan en los proyectos .NET mediante **paquetes NuGet**.

Algunas de las bases de datos que se pueden usar con EF Core son:

- SQL Server
- Mysql
- PostgreSQL
- Oracle
- SQLite, [etc.](#)

Es el que usaremos



# Entity Framework Core

## SQL Server

Es un sistema de gestión de base de datos (SGBD) relacional, desarrollado por Microsoft.

Existen varias ediciones. Algunas son:

- SQL Server en Azure
- SQL Server Enterprise
- SQL Server Developer
- SQL Server Express

Usaremos SQL Server Express LocalDB



# Entity Framework Core

## SQL Server Express LocalDB

Esta versión tiene las siguientes características:

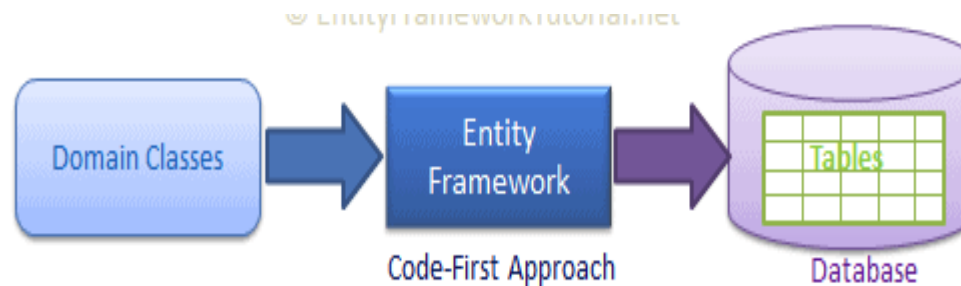
- Es una versión ligera del motor de base de datos SQL Server Express, instalada de forma predeterminada con Visual Studio
- Se inicia a petición mediante una cadena de conexión
- Está destinado al desarrollo de programas. Se ejecuta en modo usuario, sin necesidad de una configuración compleja
- De forma predeterminada, crea archivos *.mdf* en el directorio *C:/Usuarios/{user}*



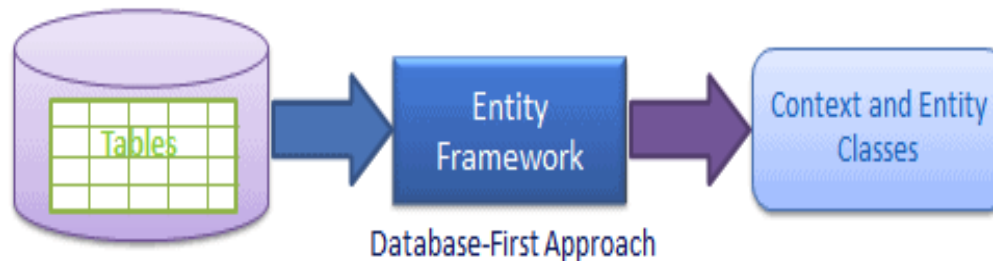
# Entity Framework Core

EF Core permite trabajar de dos formas con la BD:

- **Code First:** Se definen las clases de los modelos, con sus relaciones, y se generan automáticamente las tablas en la base de datos



- **Database First:** Partiendo de una base de datos relacional, se utiliza el ORM para generar las clases de los modelos de la aplicación.





# Entity Framework Core

## ¿Code First ó Database First?

Hay que elegir una de las dos en función de algunos aspectos:

- **Code First**

- El equipo de desarrollo maneja mejor el código que las bases de datos
- El proyecto no trabaja sobre una BD previa, sino que la crea desde el inicio

- **Database First**

- El equipo de desarrollo se defiende bien con las bases de datos
- El proyecto se ha de adaptar a una base de datos existente



# EF Core - Code First

Partiremos de un nuevo proyecto ASP.NET Core MVC

Varios de los conceptos tratados en este apartado también los usaremos cuando trabajemos de la forma Database First

Para trabajar de esta manera veremos los siguientes aspectos:

- Creación del modelo
- Paquetes NuGet
- Scaffolding de vistas y controlador
- Migraciones
- Seeder



# EF Core - Code First

## Creación del Modelo

En este tipo de aproximación partiremos de un determinado modelo.

La creación de un modelo es algo que ya se ha tratado anteriormente.

Todos los atributos de validación que puedan tener los campos del modelo se utilizarán para la creación de las columnas de la tabla correspondiente, según lo considere EF Core.

Un ejemplo de Modelo, que usaremos para este ejemplo, podría ser el siguiente:



# 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; }
}
```



# EF Core - Code First

## Paquetes NuGet

Los paquetes NuGet son librerías formadas por código compilado (archivos DLL) que podemos incluir en un proyecto .NET para ampliar su funcionalidad.

Son unidades reutilizables que los desarrolladores ponen a disposición de la comunidad para que se usen en los proyectos.

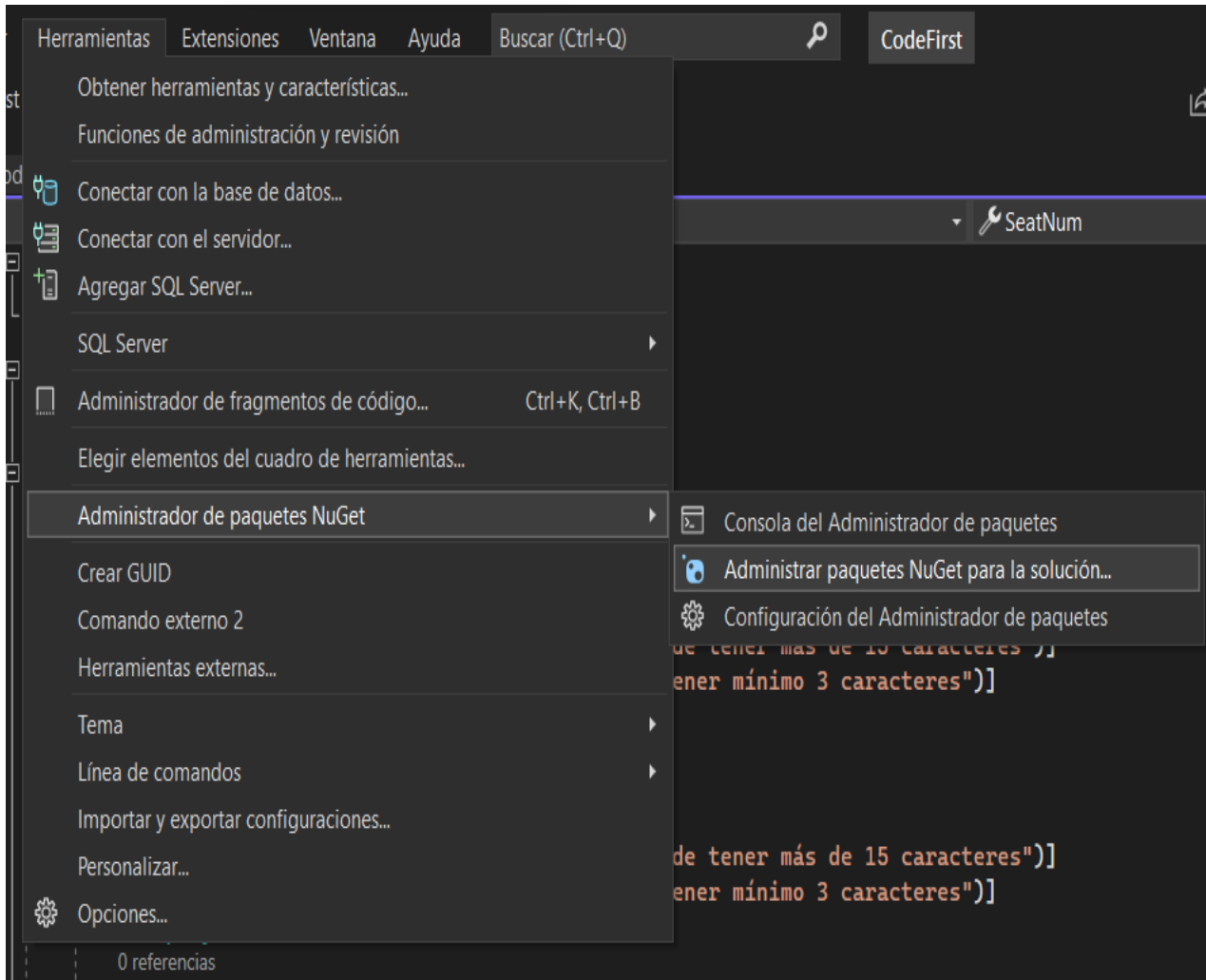
Se pueden instalar de dos formas:

- Usando la Consola de Administrador de paquetes
- Mediante el panel de Administrar paquetes NuGet para la solución

Algunas acciones son necesarias realizarlas por consola, como veremos.



# EF Core - Code First










# EF Core - Code First

## Paquetes NuGet

En la pestaña “Examinar” buscaremos e instalaremos los siguientes 3 paquetes:

 	<b>Microsoft.EntityFrameworkCore.SqlServer</b> por Microsoft Microsoft SQL Server database provider for Entity Framework Core.	6.0.9
 	<b>Microsoft.EntityFrameworkCore.Tools</b> por Microsoft Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	6.0.9
 	<b>Microsoft.VisualStudio.Web.CodeGeneration.Design</b> por Microsoft Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	6.0.10





# EF Core - Code First

## MVC Scaffolding

El *scaffolding* (andamiaje en inglés) es una técnica usada en muchos frameworks MVC para generar código básico para operaciones CRUD.

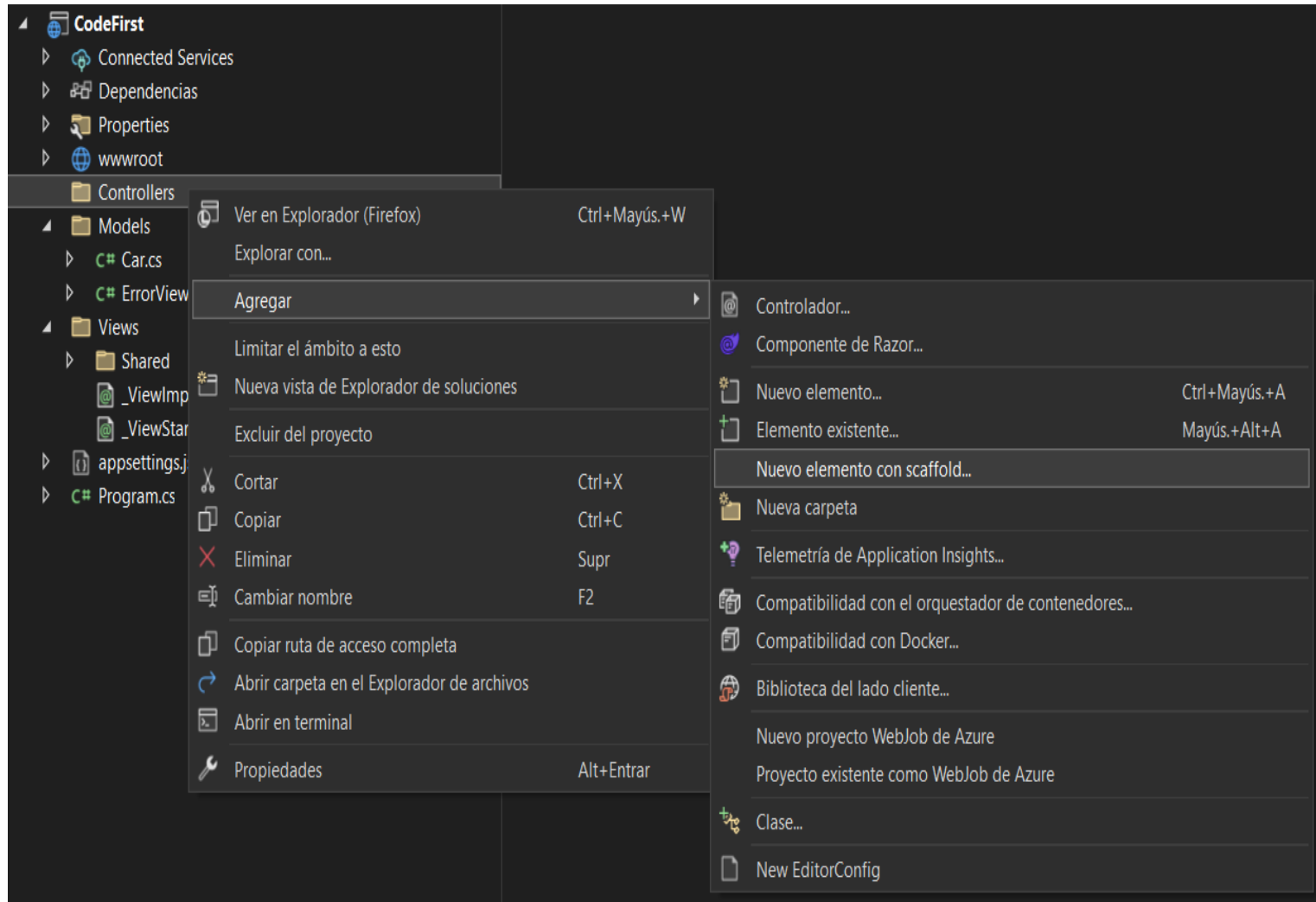
Esto genera el controlador, con sus acciones correspondientes, y las vistas necesarias para poder realizar este tipo de operaciones de persistencia con la base de datos.

Esta acción se realiza en base a un modelo definido y genera una gran cantidad de archivos y código que veremos con detenimiento.

Todo el código generado puede no ser útil, y será el desarrollador quien tenga que realizar las adaptaciones necesarias.

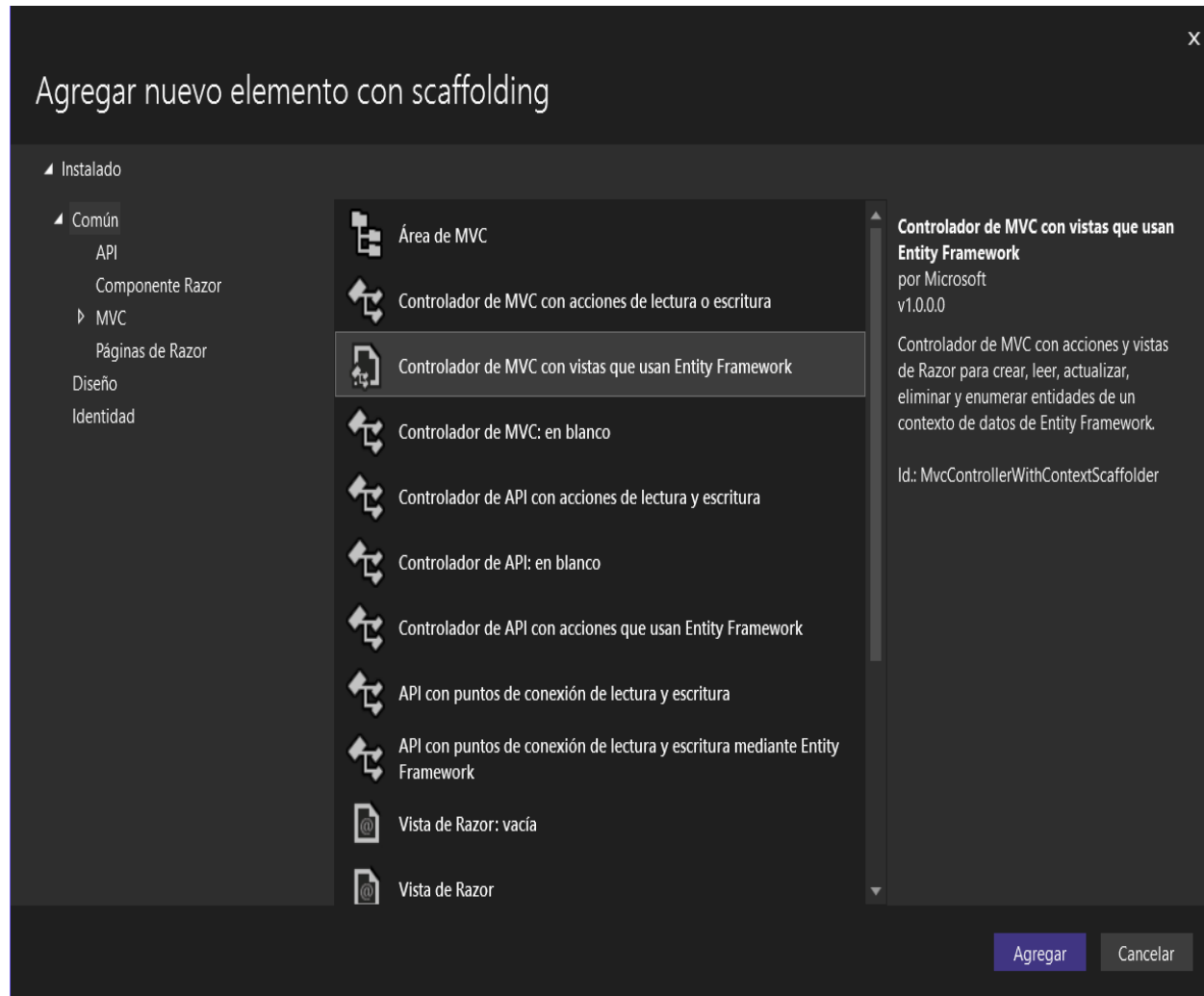


# EF Core - Code First





# 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

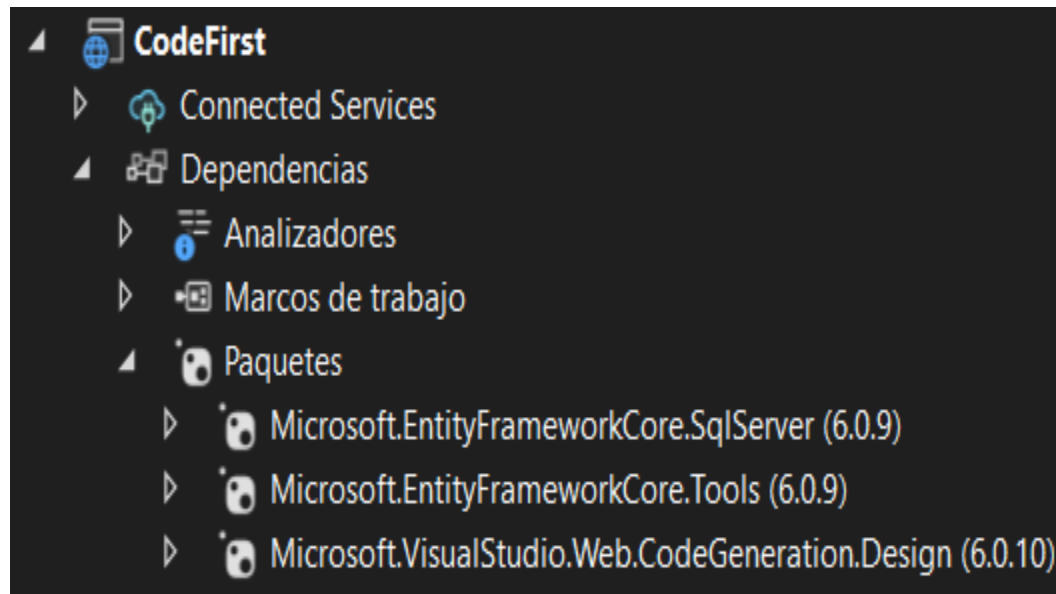
Pulsando + nos muestra la clase de contexto de datos por defecto de la aplicación, que es la que habrá que elegir



# EF Core - Code First

## Paquetes NuGet

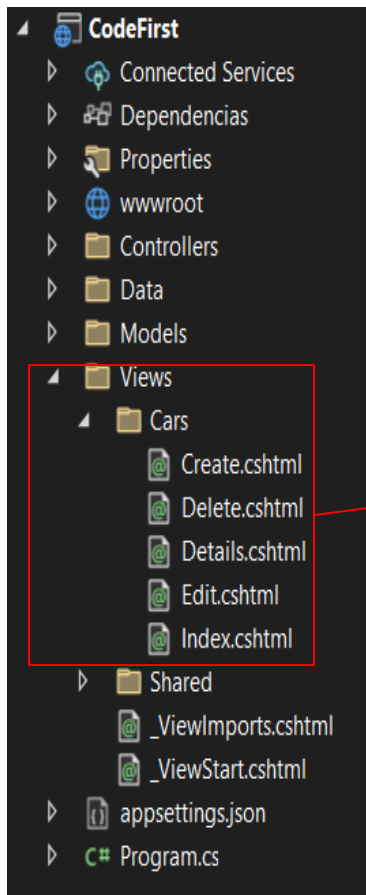
Vemos que tras la instalación aparecen estos 3 paquetes como dependencias de nuestro proyecto:





# EF Core - Code First

## MVC Scaffolding - Elementos Creados/Actualizados



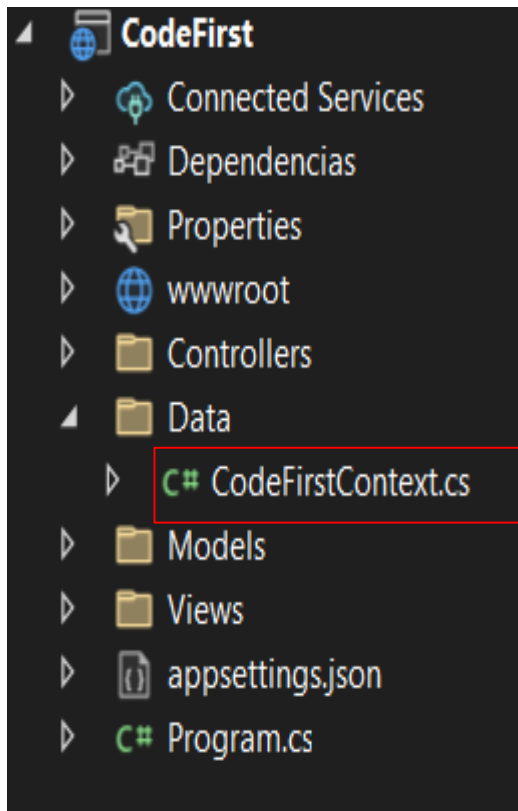
Todas las vistas relacionadas con el modelo que nos van a permitir:

- Listar
- Crear
- Ver detalles
- Editar
- Eliminar



# EF Core - Code First

## MVC Scaffolding - Elementos Creados/Actualizados



Es la clase de tipo DbContext de la aplicación. Es la clase primaria responsable de interactuar con la base de datos.

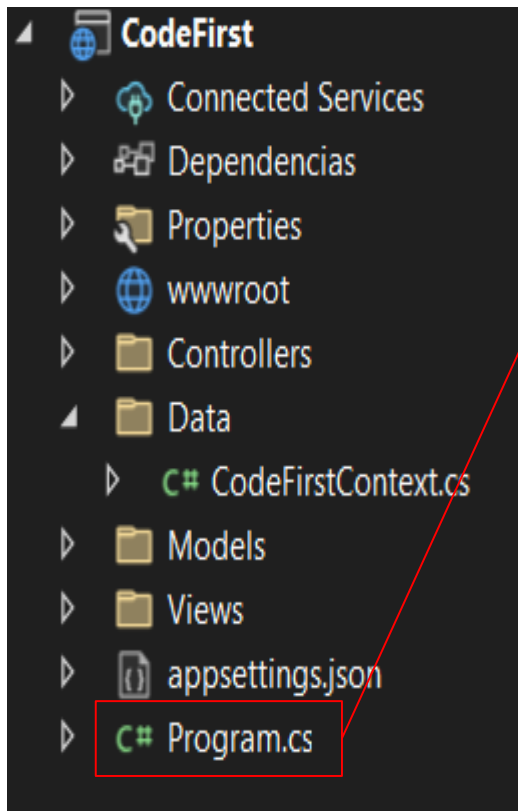
Se encarga de las siguientes acciones sobre la base de datos:

- Consultas
- Seguimiento de cambios
- Operaciones de persistencia CUD
- Almacenamiento en caché
- Gestionar relaciones (CF o DBF)
- Materialización de objetos



# EF Core - Code First

## MVC Scaffolding - Elementos Creados/Actualizados



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

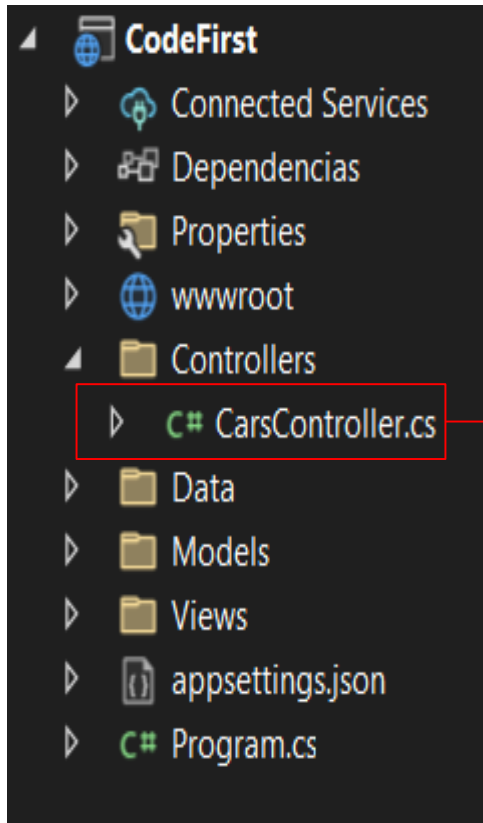
```
builder.Services.AddDbContext<CodeFirstContext>(options =>  
    options.UseSqlServer(builder.Configuration.GetConnectionString("CodeFirstC
```





# EF Core - Code First

## MVC Scaffolding - Elementos Creados/Actualizados



Controlador del modelo del que hemos hecho el *scaffold*.

Contiene las siguientes acciones para realizar sobre el modelo

- Listar
- Crear
- Ver detalles
- Editar
- Eliminar



# 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());
    }
}
```

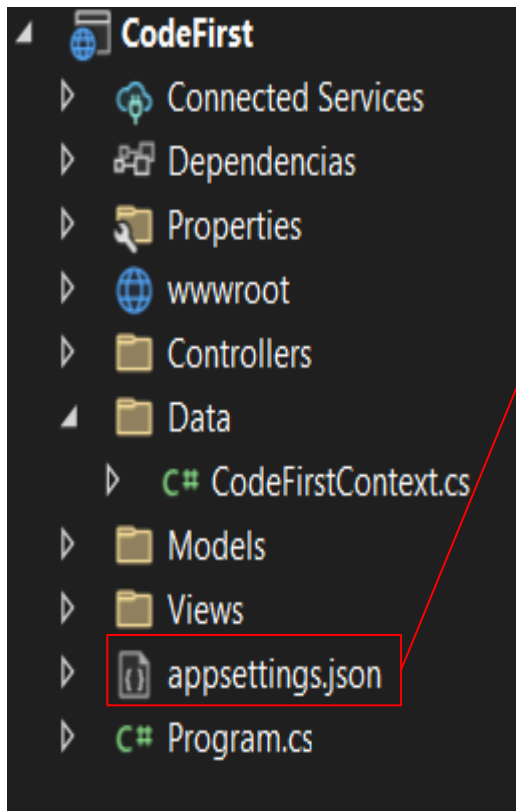
El controlador declara una referencia al DbContext de la aplicación y la inicializa en el constructor.  
Se utilizará para realizar los accesos a la Base de Datos

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;  
}
```



# EF Core - Code First

## SQL Express LocalDB

Para consultar el contenido de la base de datos tenemos dos formas:

- Explorador de objetos de SQL Server (Herramienta de Visual Studio)
- Sql Server Management Studio

Ambas son válidas para consultar el contenido de nuestra base de datos local de SQL Express



# EF Core - Code First

## SQL Express LocalDB - Consultar estado del servicio

```
PS C:\Users\asanhid> sqllocaldb info
MSSQLLocalDB
PS C:\Users\asanhid> sqllocaldb info MSSQLLocalDB
Name:                MSSQLLocalDB
Version:              15.0.4153.1
Shared name:
Owner:                AzureAD\asanhid
Auto-create:          Yes
State:                Running
Last start time:      03/10/2022 17:18:06
Instance pipe name:  np:\\.\pipe\LOCALDB#42D22625\tsql\query
```



# EF Core - Code First

## Migraciones

Si consultamos la base de datos en este punto, incluso tras haber compilado y ejecutado la aplicación, veremos que no hay ninguna base de datos ni tablas creadas.

Para ello hemos de crear la migración inicial, que se encargará de crear la base de datos con el nombre del proyecto y una primera tabla en base al modelo creado.

Las migraciones son un conjunto de herramientas que crean y actualizan una base de datos para que coincida con el modelo de datos.



# EF Core - Code First

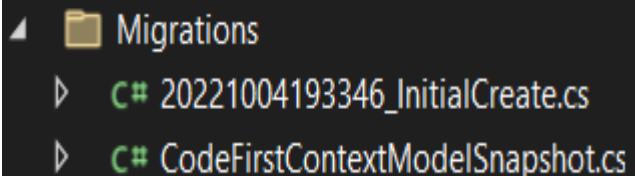
## Migraciones

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



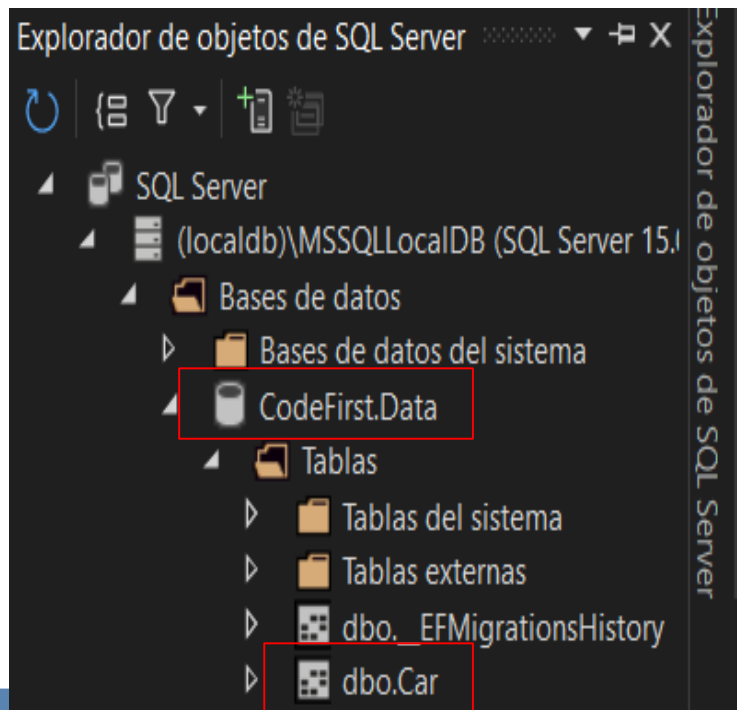
```
Migrations  
└─ C# 20221004193346_InitialCreate.cs  
└─ C# CodeFirstContextModelSnapshot.cs
```



# 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.







# EF Core - Code First

## Migraciones

En este punto, si modificamos la ruta por defecto de la aplicación y la ejecutamos veremos que se carga un listado vacío de elementos del modelo sobre el que hemos hecho *scaffolding*.

Es posible insertar datos por defecto de un modelo en su tabla correspondiente mediante un nuevo elemento denominado *seeder*.



# EF Core - Code First

## ***Seeders***

Los *seeders* nos permiten insertar registros en determinadas tablas que sean necesarios para el funcionamiento de la aplicación.

Para sembrar nuestra base de datos hay que realizar dos acciones:

- Crear una clase *SeedData* mediante la cual se usará el DbContext para añadir registros a las tablas que se deseen.
- Modificar el archivo Program.cs para que se siembren datos en la base de datos al inicializar la aplicación



# EF Core - Code First

## Seeders - Clase SeedData

```
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);  
}
```

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



# Ejercicio

Realizar las siguientes acciones:

- Crear la clase de un modelo con 4 campos, variados en su tipo, más el campo `id`
- Añadir atributos de validación de los campos del modelo
- Generar el *scaffold* en base a ese modelo en la aplicación
- Crear una migración que cree una base de datos con el nombre del proyecto, en nuestro SQL LocalDB, y una tabla vacía con los campos de nuestro modelo.
- Crear un *seeder* que incluya 3 registros en la tabla de nuestro modelo al iniciar la aplicación, en caso de que esta estuviera vacía.