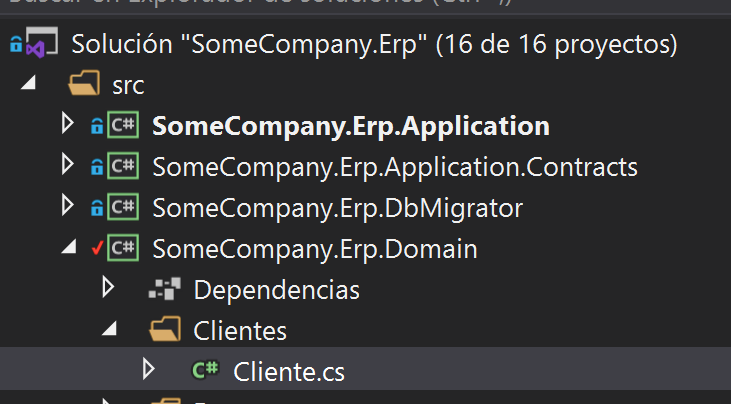
Temer en cuenta que el nombre de la solución tiene el siguiente formato: <Company>.<Solution> = SomeCompany.Erp, este es el nombre que se encontrará como referencia en los namespaces y proyectos.

Después de crear la aplicación con el CLI ir a \*.DbMigrator y cambiar el ConnectionString en appsettings.json. Hacer los mismo con el proyecto \*.Web

Registrar entitdades en Entity Framework

1. Las entidades se definen en Domain

* Para mantener el orden, se les puede asignar una carpeta con su nombre en plural, esto creará un namespace para nuestra entidad. Si la entidad se llama "Cliente", entonces la carpeta que lo contendrá se llamará "Clientes"



using System;

using Volo.Abp.Domain.Entities.Auditing;

namespace SomeCompany.Erp.Clientes

{

public class Cliente : AuditedEntity<Guid> // El Id es Guid y auto generado por abp

{

public string Nombre { get; set; }

public string Ruc { get; set; }

public string Dni { get; set; }

public string Ce { get; set; } // Carnet de extranjeria char(12)

}

}

2. Luego La entiddad debe registrar en el DbContext.

Este DbContext se encuentra en el proyecto EntityFrameworkCore

using Microsoft.EntityFrameworkCore;

using SomeCompany.Erp.Users;

using Volo.Abp.Data;

using Volo.Abp.EntityFrameworkCore;

using Volo.Abp.EntityFrameworkCore.Modeling;

using Volo.Abp.Identity;

using Volo.Abp.Users.EntityFrameworkCore;

using SomeCompany.Erp.Clientes;

namespace SomeCompany.Erp.EntityFrameworkCore

{

/\* This is your actual DbContext used on runtime.

\* It includes only your entities.

\* It does not include entities of the used modules, because each module has already

\* its own DbContext class. If you want to share some database tables with the used modules,

\* just create a structure like done for AppUser.

\*

\* Don't use this DbContext for database migrations since it does not contain tables of the

\* used modules (as explained above). See ErpMigrationsDbContext for migrations.

\*/

[ConnectionStringName("Default")]

public class ErpDbContext : AbpDbContext<ErpDbContext>

{

public DbSet<AppUser> Users { get; set; }

public DbSet<Cliente> Clientes { get; set; } // Agregar nueva entidad aqui

…

3. Finalmente, conectar el modelo a la base de datos. En ErpDbContextModelCreatingExtensions.cs agregar la entidad Cliente

using Microsoft.EntityFrameworkCore;

using Volo.Abp;

using Volo.Abp.EntityFrameworkCore.Modeling;

using SomeCompany.Erp.Clientes;

namespace SomeCompany.Erp.EntityFrameworkCore

{

public static class ErpDbContextModelCreatingExtensions

{

public static void ConfigureErp(this ModelBuilder builder)

{

Check.NotNull(builder, nameof(builder));

/\* Configure your own tables/entities inside here \*/

//builder.Entity<YourEntity>(b =>

//{

// b.ToTable(ErpConsts.DbTablePrefix + "YourEntities", ErpConsts.DbSchema);

// b.ConfigureByConvention(); //auto configure for the base class props

// //...

//});

builder.Entity<Cliente>(b =>

{

b.ToTable(ErpConsts.DbTablePrefix + "Clientes",

ErpConsts.DbSchema);

b.ConfigureByConvention(); //auto configure for the base class props

b.Property(x => x.Nombre).IsRequired().HasMaxLength(50);

b.Property(x => x.Ruc).IsFixedLength(true).HasMaxLength(11); // char(11)

b.Property(x => x.Dni).IsFixedLength(true).HasMaxLength(8); // char(8)

b.Property(x => x.Ce).IsFixedLength(true).HasMaxLength(12); // char(12)

});

}

}

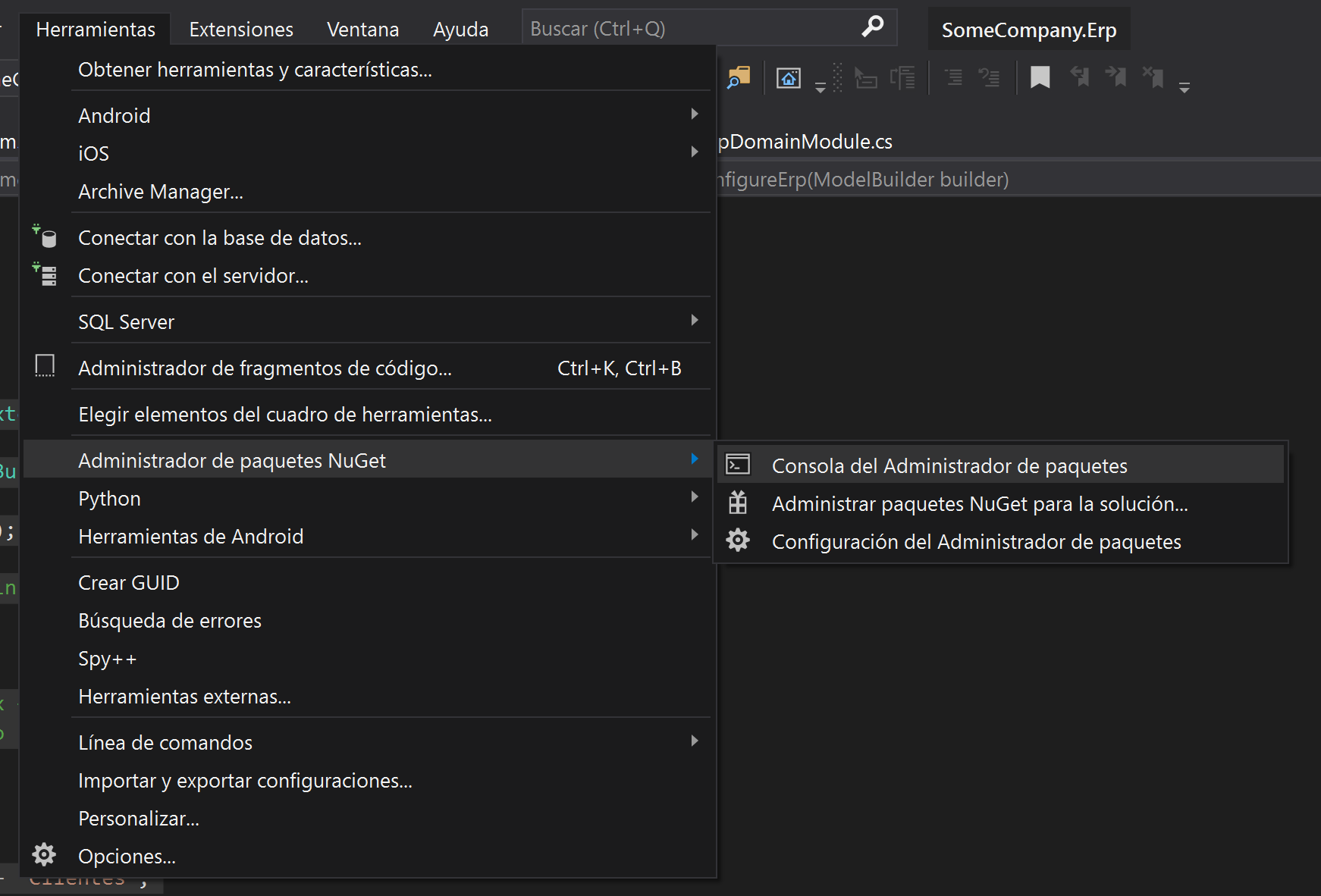
}

**Resumen**

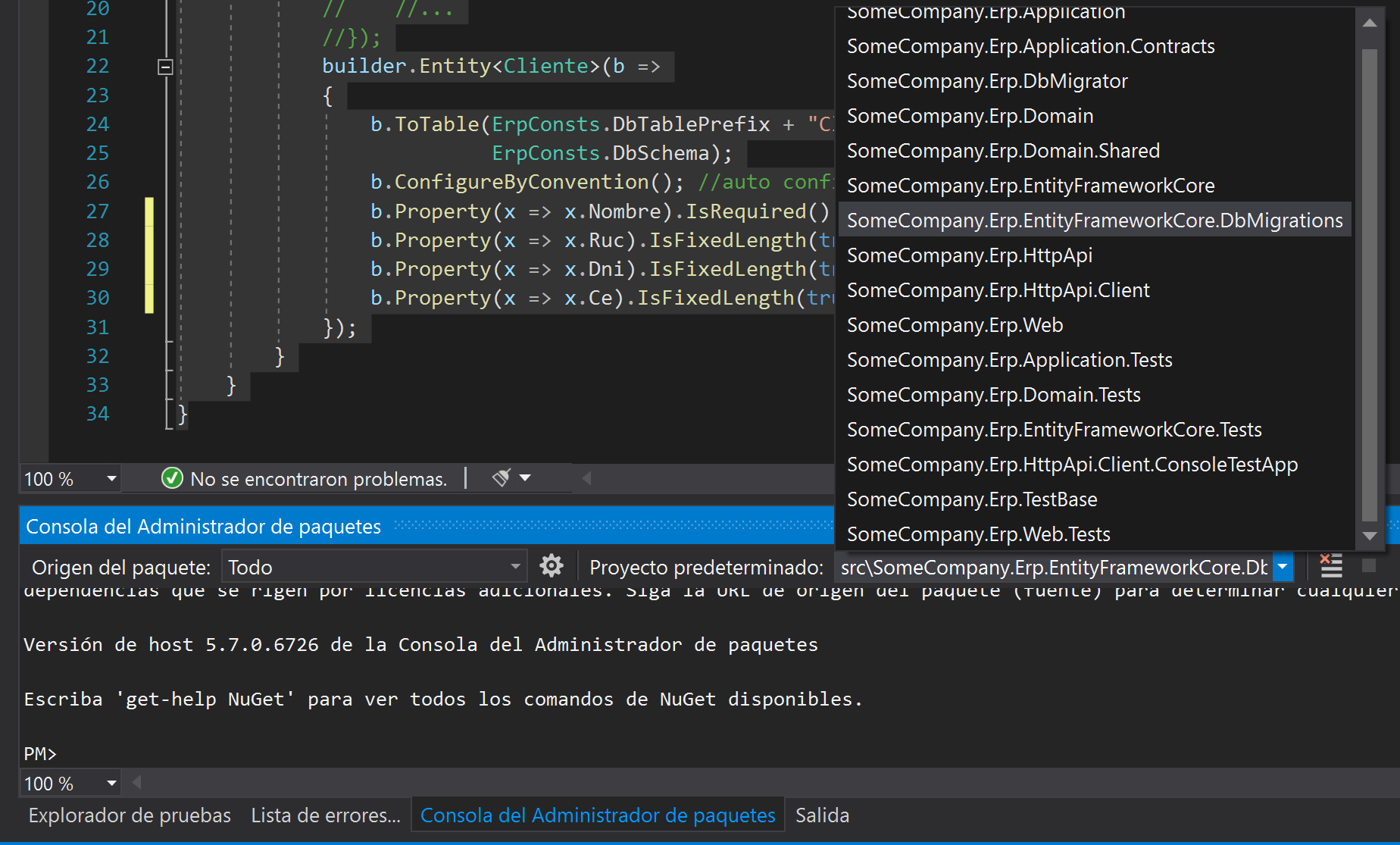
* En la parte 1 se ha creado una entidad Cliente con algunos campos que serán accesibles dentro del código.
* En la parte 2, se ha añadido al dbcontext y ahora entity framework (EF) sabe cómo acceder a los datos de los clientes a través de la clase Cliente.
* Y en la parte 3, se le dice a EF como debe crear la tabla clientes en la DB, esta parte es importante tenerla clara ya que las migraciones se hacen a partir de aquí.

Creando migraciones

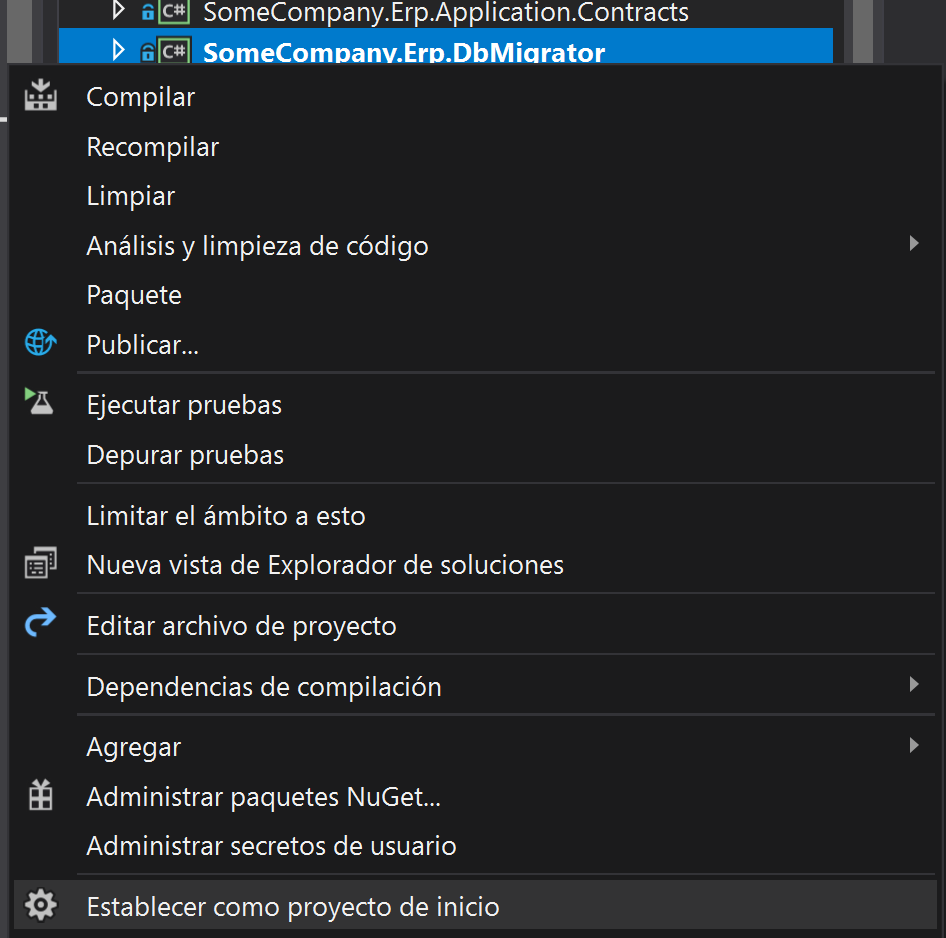
1. Abrir la consola del administrador de paquetes



1. Asegurarse que el proyecto predeterminado en la consola apunte a DbMigrations



1. Asegurar de que Migrator esté seleccionado como proyecto de inicio y ejecutar el siguiente comando en la cnsola del administrador de paquetes



Add-Migration "Creando\_Entidad\_Clientes" -Verbose

El parámetro Verbose sirve para que nos de más detalles del procedimiento en caso de fallas y así poder saber que debemos arreglar.

Estos son los 3 mensajes que nos indican que todo ha salido bien:

Build started...

Build succeeded.  
To undo this action, use Remove-Migration.

Y Ahora aparece en la carpeta Migrations del proyecto DbMigrations un archivo nuevo que termina en “\_Creando\_Entidad\_Clientes.cs”

1. (Opcional) Si se necesita tener datos como para un entorno de pruebas entonces se puede ejecutar un procedimiento bastante común llamado Seeding, existen varias formas de lograrlo, veamos cómo hacerlo con Abp.

* En el proyecto Domain, crear un archivo llamado ErpDataSeederContributor.cs y escribir:

using System;

using System.Threading.Tasks;

using SomeCompany.Erp.Clientes;

using Volo.Abp.Data;

using Volo.Abp.DependencyInjection;

using Volo.Abp.Domain.Repositories;

namespace SomeCompany.Erp

{

public class ErpDataSeederContributor : IDataSeedContributor, ITransientDependency

{

private readonly IRepository<Cliente, Guid> \_clienteRepository;

public ErpDataSeederContributor(IRepository<Cliente, Guid> clienteRepository)

{

\_clienteRepository = clienteRepository;

}

public async Task SeedAsync(DataSeedContext context)

{

if (await \_clienteRepository.GetCountAsync() <= 0)

{

await \_clienteRepository.InsertAsync(

new Cliente

{

Nombre = "Cliente 1",

Ruc = "1234567890A",

Dni = "12345678",

Ce = "1234567890AB"

},

autoSave: true

);

await \_clienteRepository.InsertAsync(

new Cliente

{

Nombre = "Cliente 2",

Ruc = "A0987654321",

Dni = "87654321",

Ce = "BA0987654321"

},

autoSave: true

);

}

}

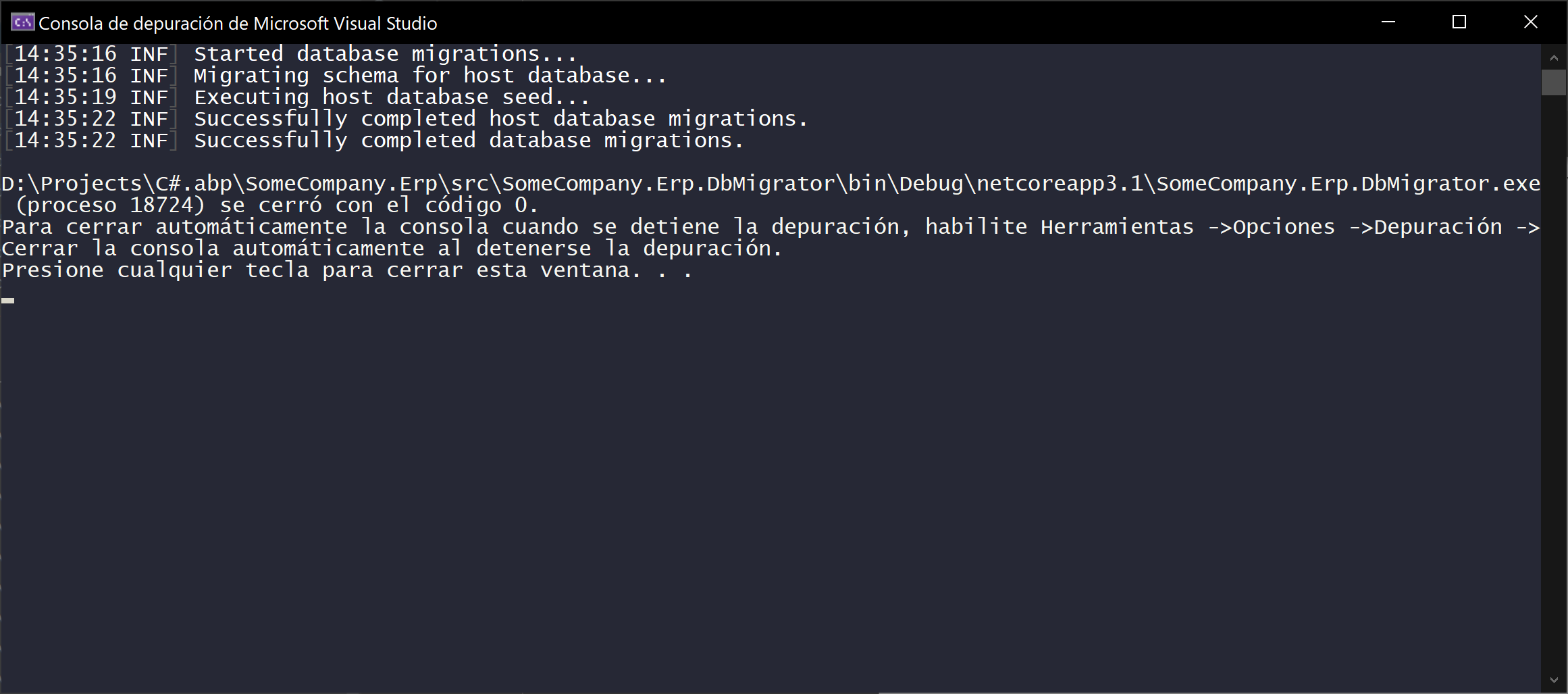
}

}

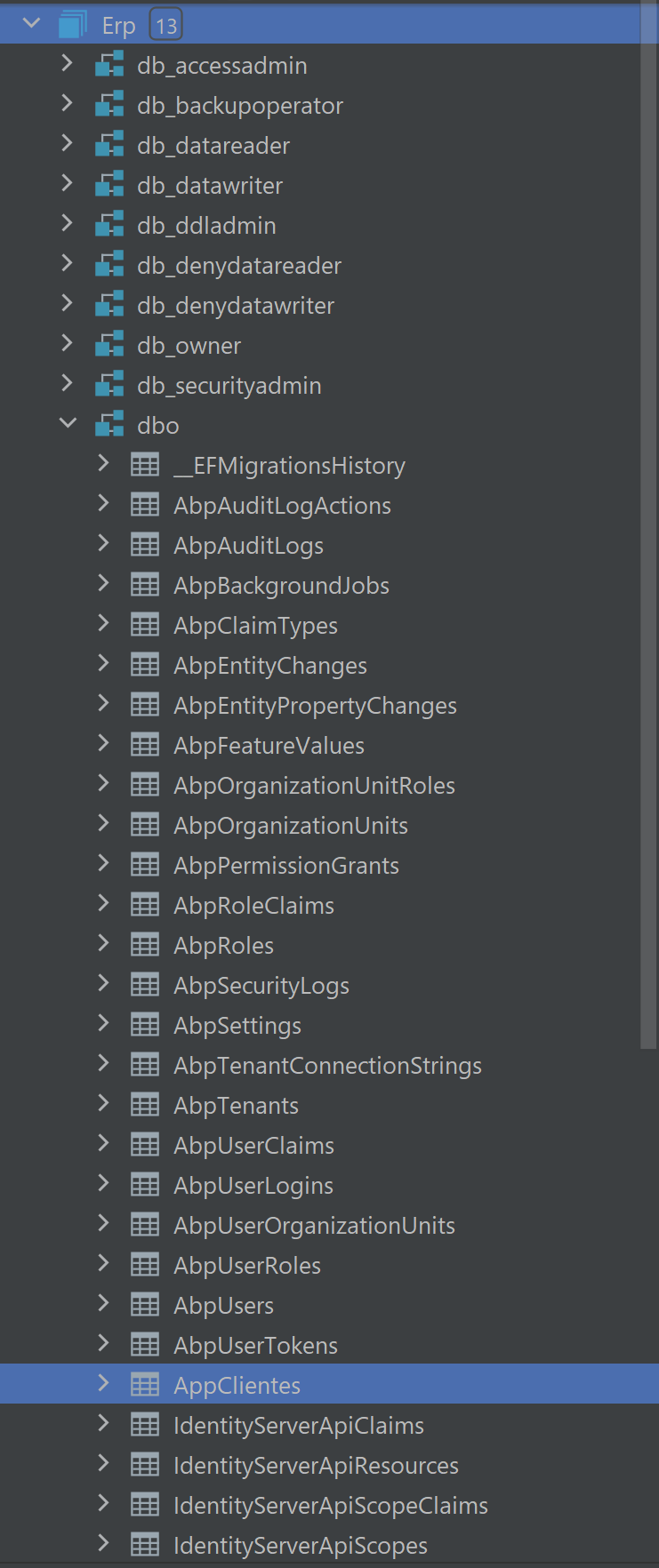
Si no hay datos en esa tabla, entonces inserta 2 registros

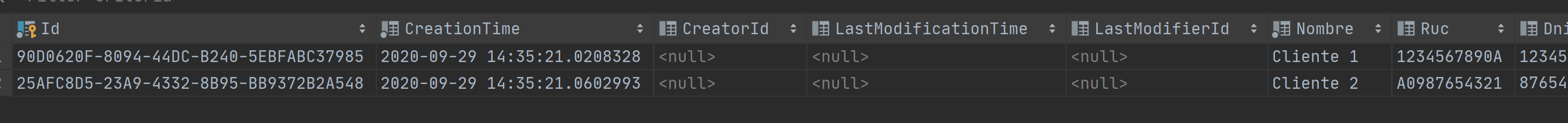
1. Asegurar de que Migratos esté como proyecto de inicio y presionar F5.

Deberíamos tener una consola más o menos así:



Al terminar la ejecución deberíamos tener la base de datos “Erp” creada junto con todas las tablas que utiliza Abp para funcionar y nuestra tabla “AppClientes” con los 2 registros que guardamos en el Seeding





**Resumen**

* Ahora que EF sabe como acceder a la base de datos desde el código, solo lo utilizamos para sincronizar la estructura de la base de datos con el código, además con el programa de migraciones podemos olvidarnos de hacer los cambios y diferencias entre versiones de la base de datos ya que esta se efecturará automáticamente

Application Service

Más información de como funcionan los Dto’s: https://docs.abp.io/en/abp/latest/Data-Transfer-Objects

1. Crear el dto de Clientes. Definimos ClienteDto.cs en una carpeta llamada Clientes dentro del proyecto \*.Application.Contracts

using System;

using Volo.Abp.Application.Dtos;

namespace SomeCompany.Erp.Clientes

{

public class ClienteDto : AuditedEntityDto<Guid>

{

public string Nombre { get; set; }

public string Ruc { get; set; }

public string Dni { get; set; }

public string Ce { get; set; }

}

}

1. Definimos el mapeador de ClienteDto en ErpApplicationAutoMapperProfile.cs del proyecto \*.Application

using AutoMapper;

using SomeCompany.Erp.Clientes;

namespace SomeCompany.Erp

{

public class ErpApplicationAutoMapperProfile : Profile

{

public ErpApplicationAutoMapperProfile()

{

/\* You can configure your AutoMapper mapping configuration here.

\* Alternatively, you can split your mapping configurations

\* into multiple profile classes for a better organization. \*/

CreateMap<Cliente, ClienteDto>();

}

}

}

// Continuar desde https://docs.abp.io/en/abp/latest/Tutorials/Part-1?UI=MVC&DB=EF#createupdatebookdto