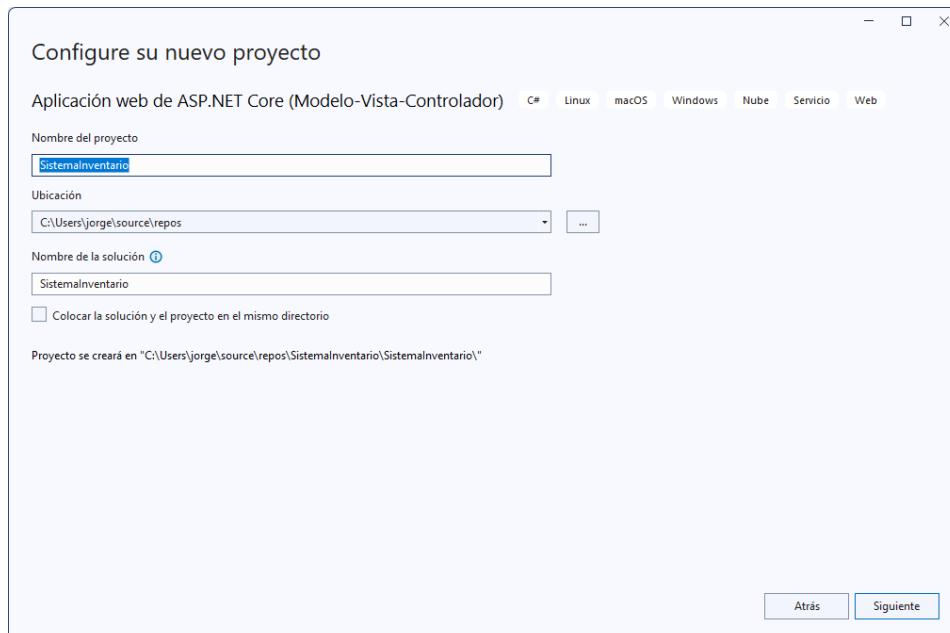
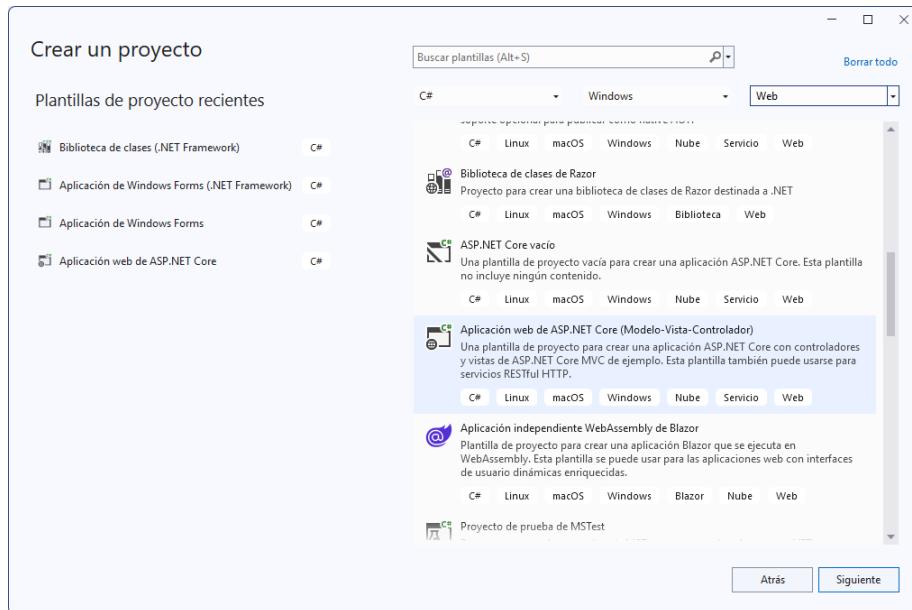


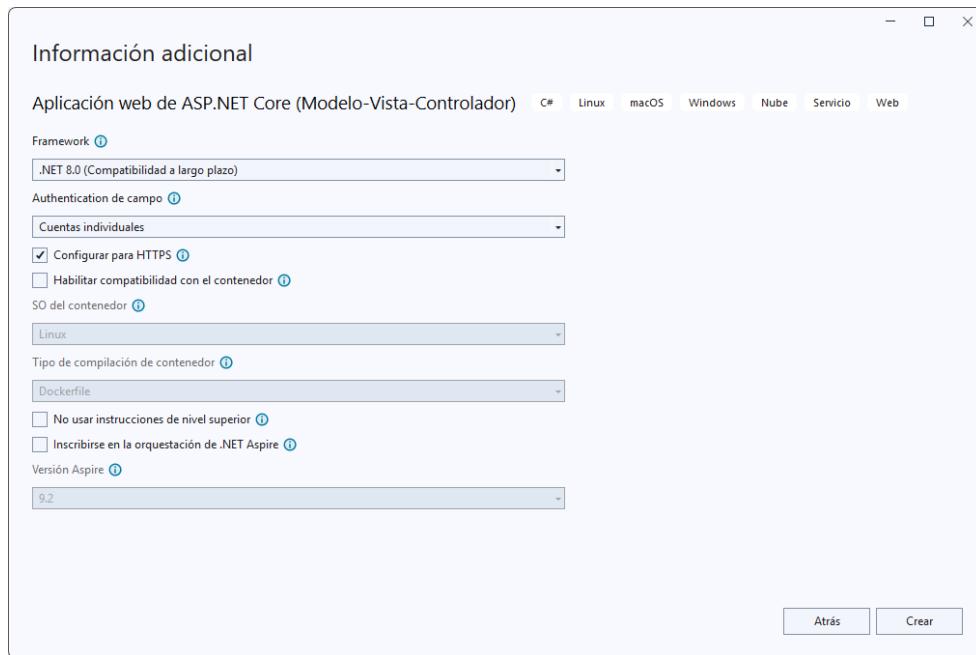
PROYECTO ASP.Net parte 1

1. Crear nuevo proyecto ASP.Net

Se inicia un nuevo proyecto en Visual Studio seleccionando la plantilla ASP.NET Core Web App (Model-View-Controller).

Este será el proyecto principal, que después organizaremos en varias capas (acceso a datos, modelos y utilidades) para aplicar la arquitectura en capas.





2. Usar Git y GitHub en nuestro Proyecto

Se recomienda versionar el proyecto con **Git** para llevar control de cambios.

Pasos clave:

- Inicializar Git en la carpeta del proyecto (git init).
- Crear archivo .gitignore para excluir binarios y configuraciones temporales (dotnet new gitignore).
- Conectar el repositorio local a GitHub y subir la primera versión:

```
C:\Users\jorge>git init
Microsoft Windows [Versión 10.0.26100.4946]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\jorge>git version
git version 2.50.1.windows.1

C:\Users\jorge>git config user.name
C:\Users\jorge>git config --global user.name "jorbo1827"
C:\Users\jorge>git config --global user.email "bgj827@gmail.com"
C:\Users\jorge>git config user.name
jorbo1827

C:\Users\jorge>git add -A
[master (root-commit) 3a2f3d1] First commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md

C:\Users\jorge>
```

- Creamos nuevo repositorio

Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).
Required fields are marked with an asterisk (*).

1 General

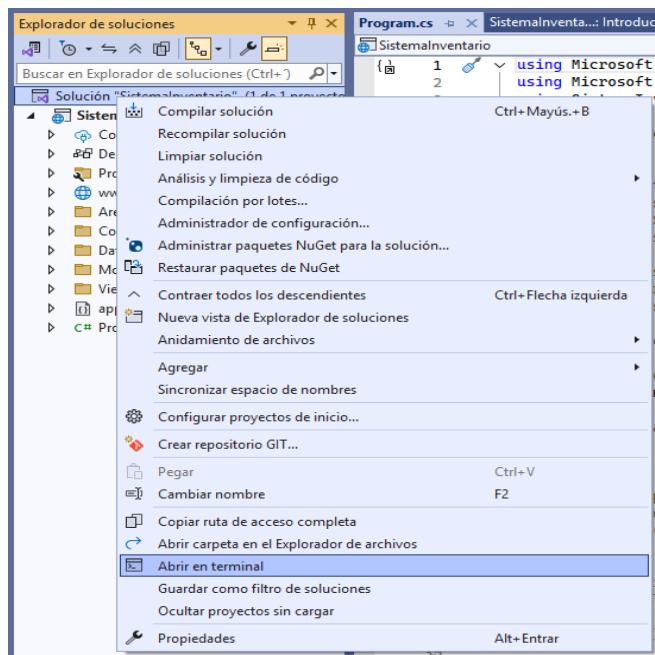
Owner *	Repository name *
 jorbol827	Sistematicventario
✓ Sistematicventario is available.	
Great repository names are short and memorable. How about jubilant-octo-palm-tree?	
Description	
<input type="text"/>	
0 / 350 characters	

2 Configuration

Choose visibility *	 Private
Choose who can see and commit to this repository	
Add README	Off <input checked="" type="checkbox"/>
READMEs can be used as longer descriptions. About READMEs	
Add .gitignore	No .gitignore
.gitignore tells git which files not to track. About ignoring files	
Add license	No license
Licenses explain how others can use your code. About licenses	

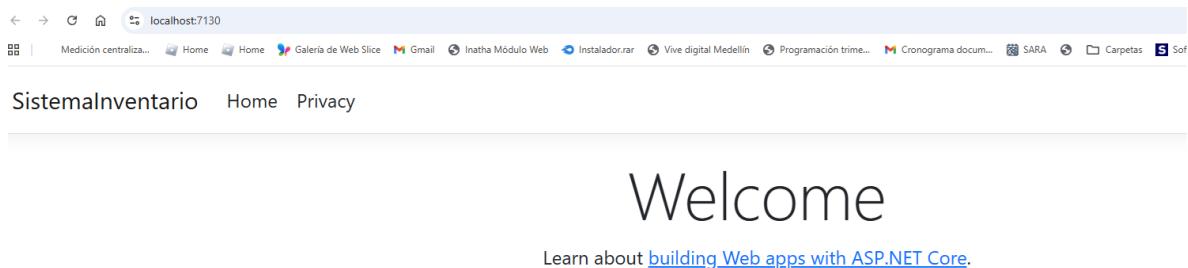
[Create repository](#)

- Abril en terminal (opcional) o en el cmd



- Ejecutar los comandos de git en
 - git init
 - dotnet new gitignore
 - git commit -m “Crear Proyecto”
 - git branch -M main
 - git remote add origin <https://github.com/jorbol1827/SistemaInventario.git>
 - git push -u origin main

3. Enrutamiento en MVC

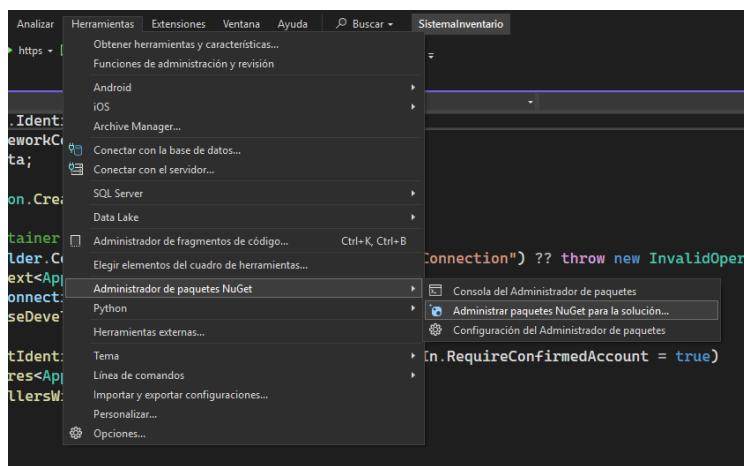


Configurar el Proyecto

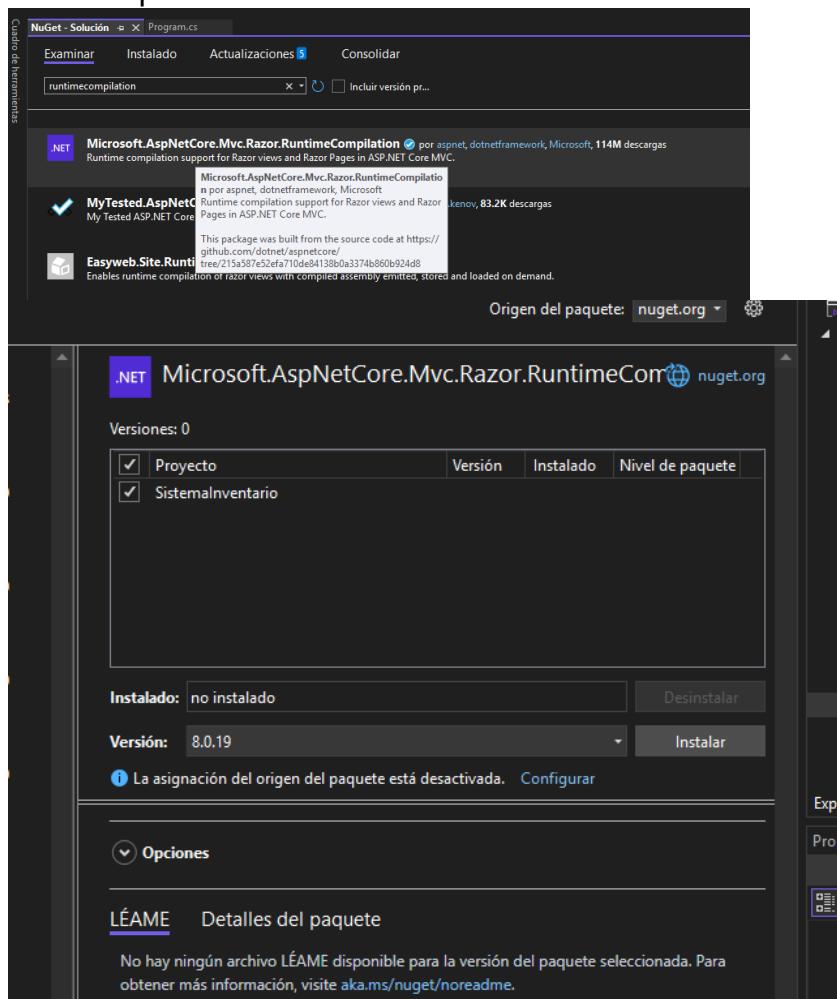
Para habilitar **compilación en caliente de vistas**, se instala el paquete:

Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation.

1. **Instalar Razor Runtime Compilation:** en Herramientas/Administrador de paquetes NuGet.



- En la pestaña Examinar



- Agregamos el servicio AddRazorRuntimeCompilation() en Program.cs y verificamos funcionamiento

```
builder.Services.AddControllersWithViews()
    .AddRazorRuntimeCompilation();
```

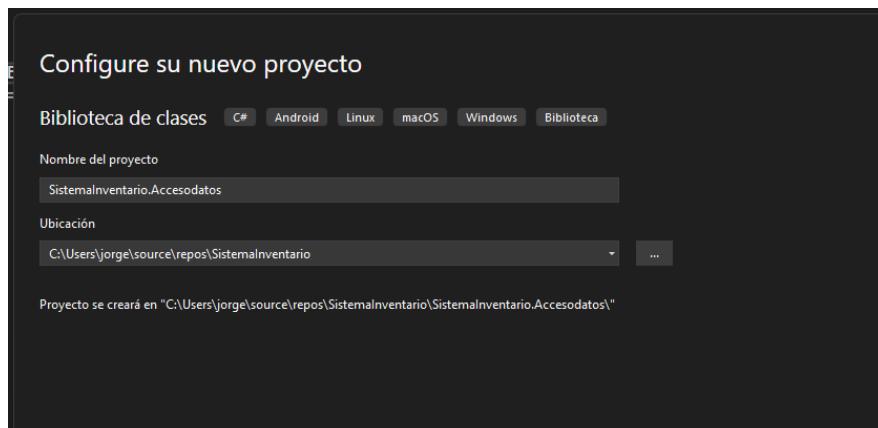
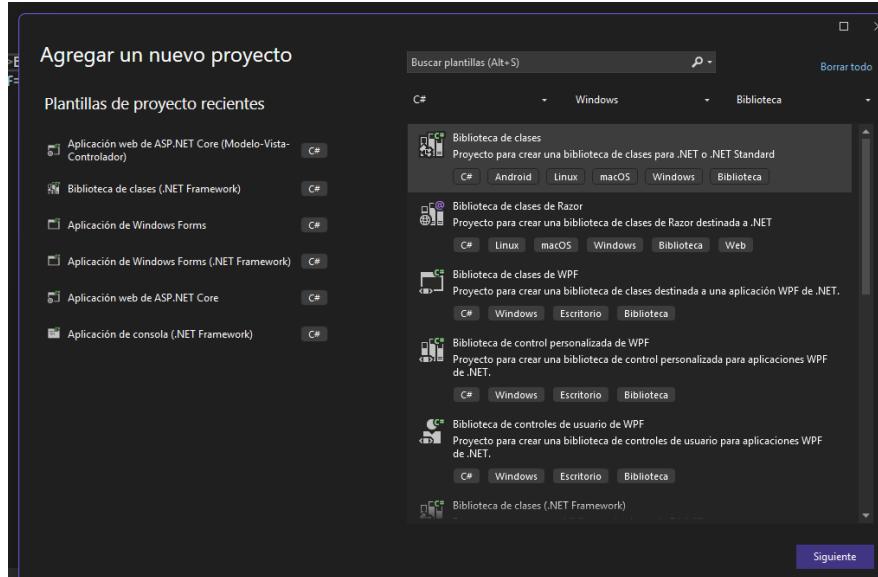
```
9     builder.Services.AddDbContext<ApplicationContext>(options =>
10         options.UseSqlServer(connectionString));
11     builder.Services.AddDatabaseDeveloperPageExceptionFilter();
12
13     builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.R
14         .AddEntityFrameworkStores<ApplicationContext>());
15     builder.Services.AddControllersWithViews().AddR
16
17     var app = builder.Build();
18
19     // Configure the HTTP request pipeline.
20     if (app.Environment.IsDevelopment())
21     {
```

A tooltip appears over the `AddRazorRuntimeCompilation()` call, listing its parameters:

- AddRazorOptions
- AddRazorPagesOptions
- AddRazorRuntimeCompilation

2. Crear Proyectos dentro de la Solución

- Agregamos un nuevo proyecto tipo Biblioteca de Clase llamado SistematicInventory.Accesodatos.

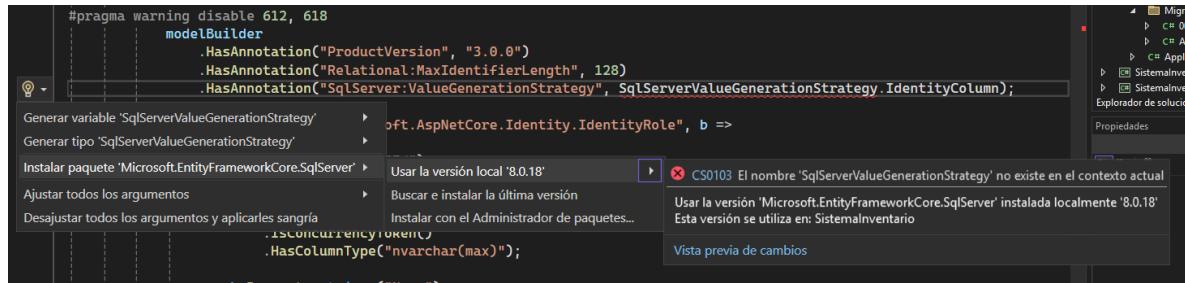


- Eliminar la clase `Class1.cs` que se crea en el nuevo proyecto.
- Agregamos otro nuevo proyecto tipo Biblioteca de Clase llamado `SistematicInventory.Modelos` y eliminamos la clase.
- Agregamos otro nuevo proyecto tipo Biblioteca de Clase llamado `SistematicInventory.Utilidades` y eliminamos la clase.

3. Separar la capa de Datos

- Movemos la carpeta `Data` al proyecto `SistematicInventory.AccesoDatos` y eliminamos la carpeta `Data` del proyecto `SistematicInventory`.

- Dentro del proyecto SistematicInventory.AccesoDatos, abrimos los archivos dentro de Data/Migrations y corregimos los errores que presentan, instalando las dependencias. Después de corregir los errores eliminamos los archivos.



- Abrimos el archivo ApplicationDbContext.cs, corregimos los errores instalando los paquetes y cambiamos el namespace

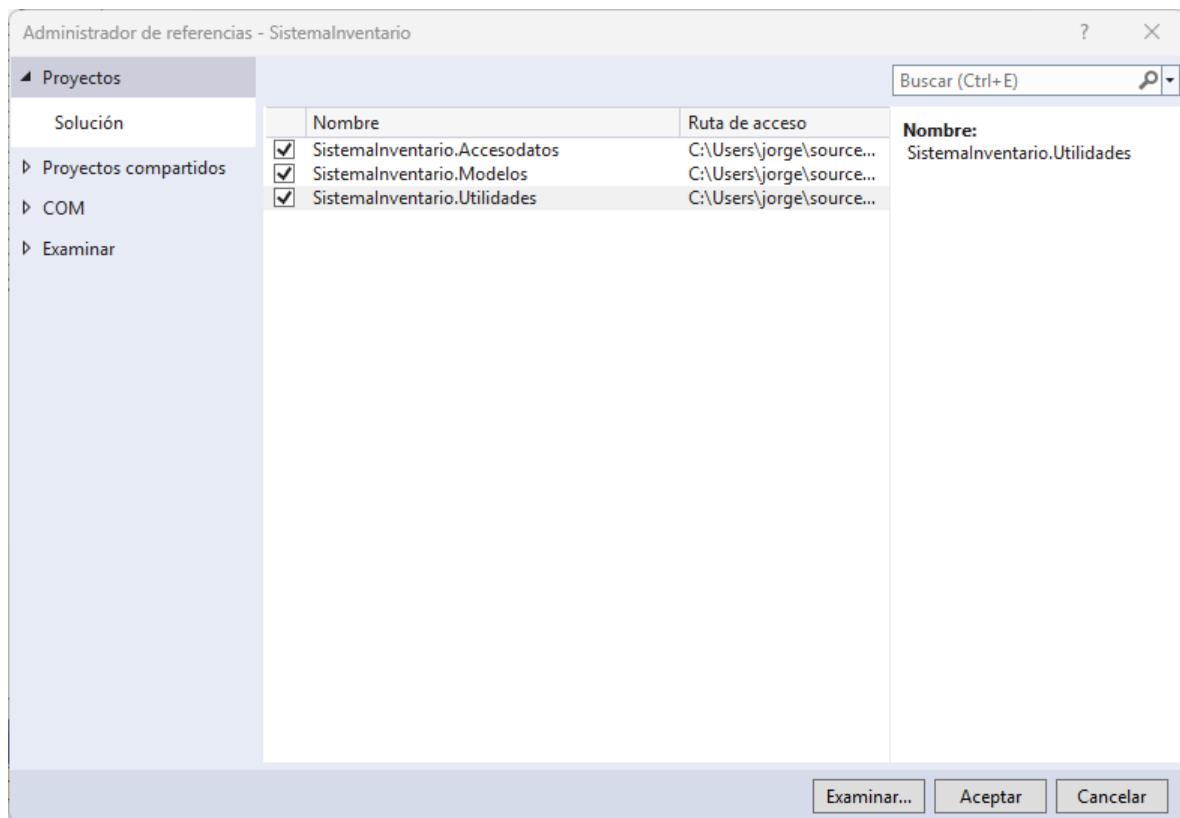
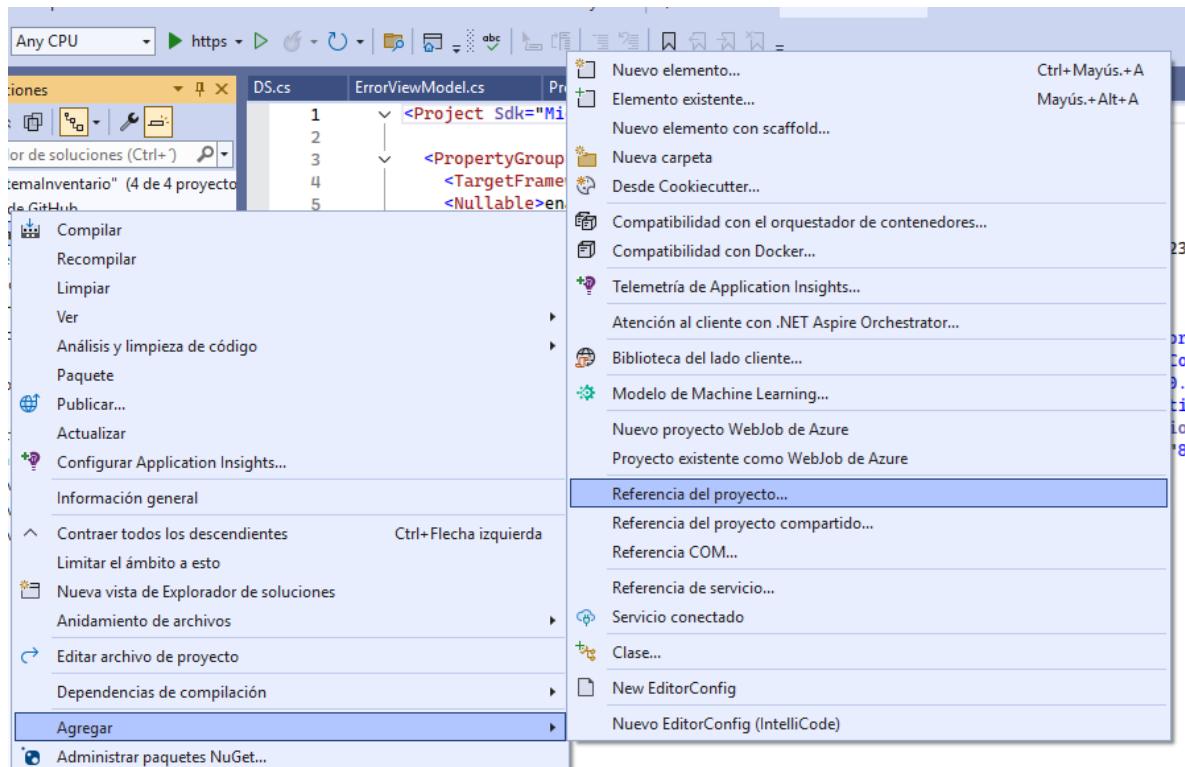
namespace SistematicInventory.AccesoDatos.Data

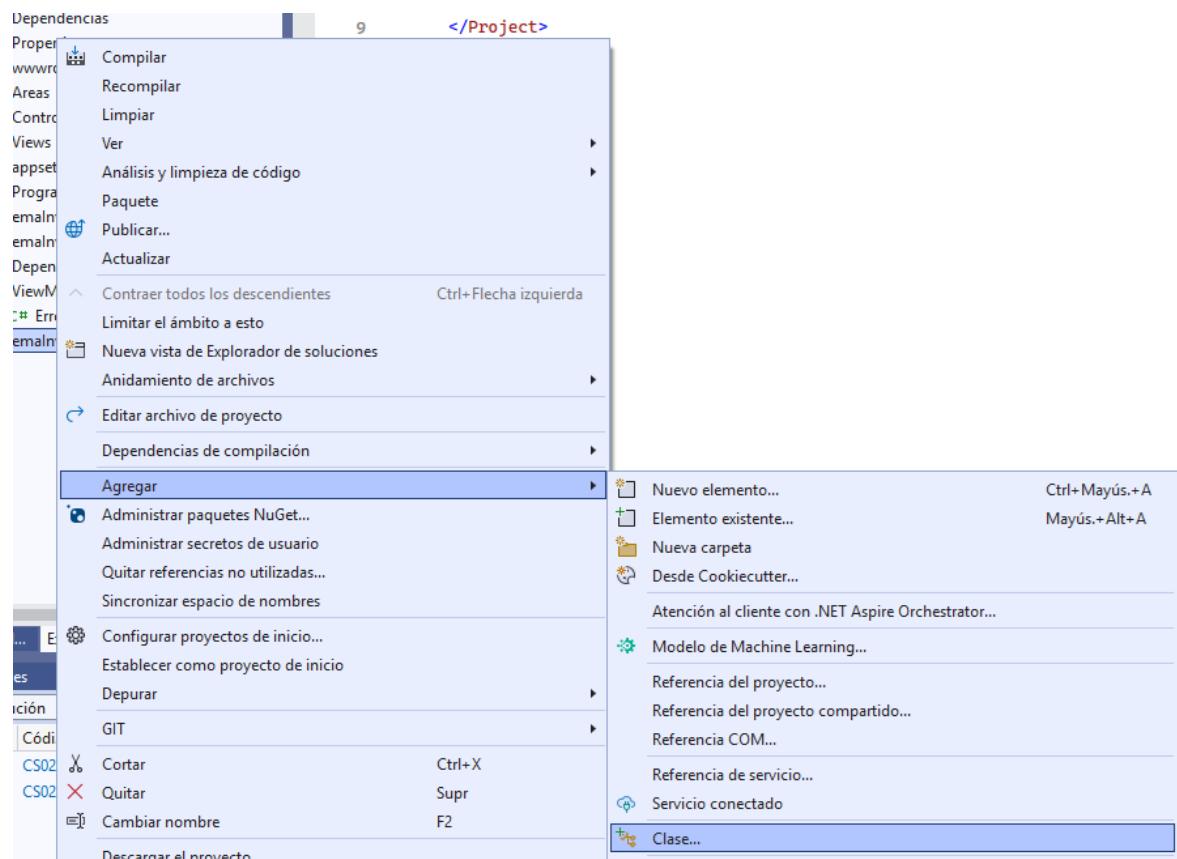
4. Separar Modelos y Agregar Referencias

- Movemos la carpeta Models al proyecto SistematicInventory.Modelos y eliminamos la carpeta Models del proyecto SistematicInventory.
- Movemos el archivo ErrorViewModel.cs al raíz del directorio del proyecto SistematicInventory.Modelos y eliminamos la carpeta Models
- En el proyecto SistematicInventory.Modelos creamos carpeta ViewModels y movemos el archivo ErrorViewModel.cs a esa carpeta.
- En el proyecto SistematicInventory.Utilidades creamos una clase estática DS.

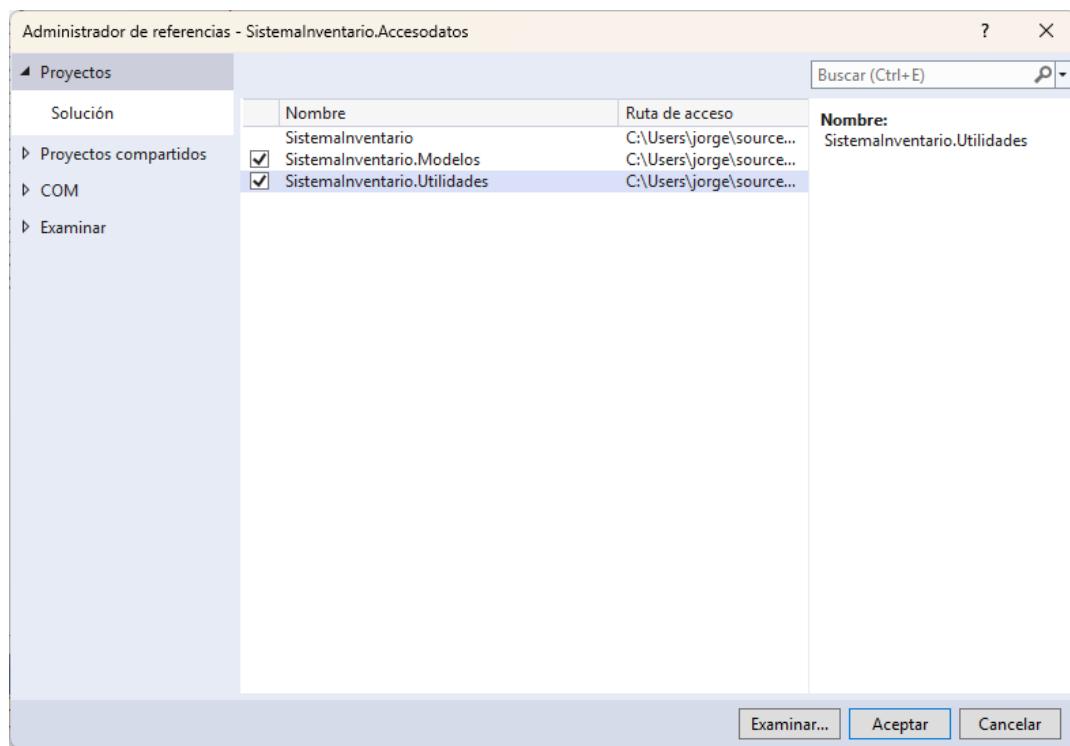
```
namespace SistematicInventory.Utilidades
{
    public static class DS
    {
    }
}
```

- En el proyecto SistematicInventory agregamos referencias

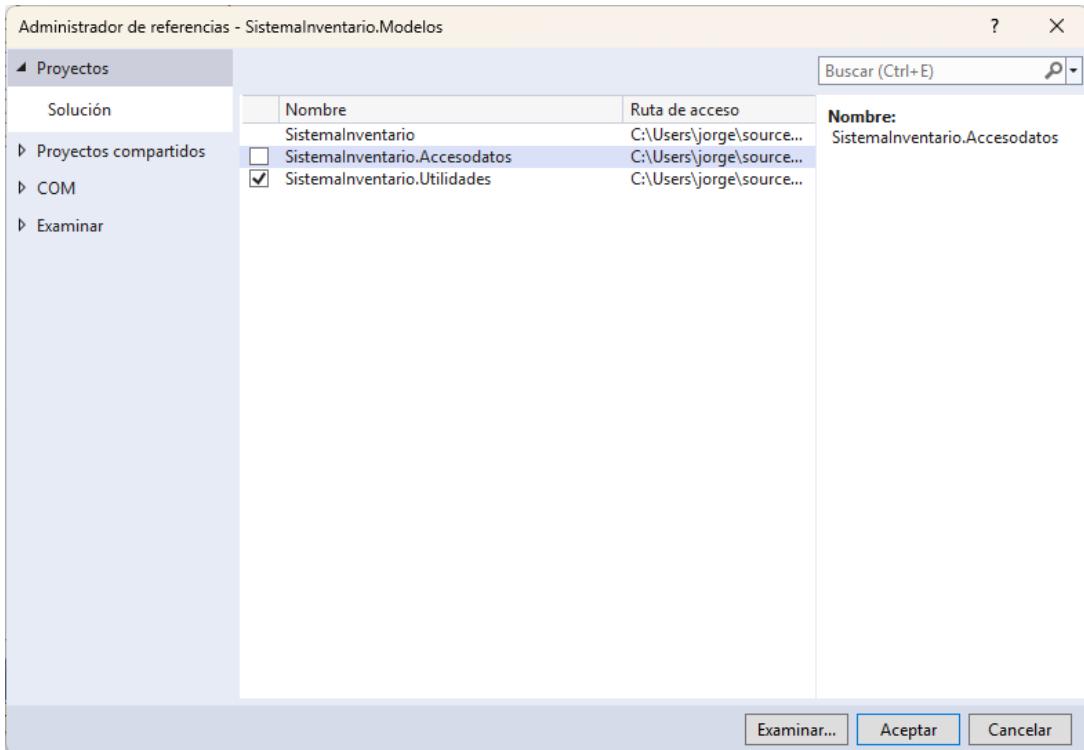




- En el proyecto SistemaInventario.AccesoDatos agregamos referencias



- En el proyecto SistematicInventory.Modelos agregamos referencias



- Compilamos para verificar errores.
 - Abrimos Program.cs y corregimos errores

The screenshot shows the 'Program.cs' file in Visual Studio. The cursor is at the end of the line 'builder.Services.AddDbContext<ApplicationDbContext>(options =>'. A tooltip box is open, displaying error CS0246: 'El nombre del tipo o el espacio de nombres 'ApplicationDbContext' no se encontró (falta una directiva using o una referencia de ensamblado?)'. Below the tooltip, there is a preview of the code with the missing 'using' statement added: 'using Microsoft.EntityFrameworkCore; + using SistemaInventario.AccesoDatos.Data; //using SistemaInventario.Data;'. The code editor shows the following code:

```
1  using Microsoft.AspNetCore.Identity;
2  using Microsoft.EntityFrameworkCore;
3  //using SistemaInventario.Data;
4
5  var builder = WebApplication.CreateBuilder(args);
6
7  // Add services to the container.
8  var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException();
9  builder.Services.AddDbContext<ApplicationDbContext>(options =>
10
11    using SistemaInventario.AccesoDatos.Data;
12    SistemaInventario.AccesoDatos.Data.ApplicationDbContext
13    Generar tipo 'ApplicationDbContext'
14
15    Introducir la variable local de 'builder.Services.AddDbContext<ApplicationDbContext>()
16    Convertir en programa de estilo 'Program.Main'
17    var app = builder.Build();
18
19    // Configure the HTTP request pipeline.
20    if (app.Environment.IsDevelopment())
21    {
22        app.UseMigrationsEndPoint();
```

- Abrimos Controllers/HomeController.cs y corregimos errores

The screenshot shows a code editor in Visual Studio with the following code:

```

25 }
26 }
27 [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
28 0 referencias | jorbo1827, Hace 1 día | 1 autor, 1 cambio
29 public IActionResult Error()
30 {
31     return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
32     using SistemaInventario.Modelos.ViewModels;
33     Modelos.ViewModels.ErrorViewModel
34     Generar tipo 'ErrorViewModel'
35     Usar cuerpo de expresiones para el método

```

A tooltip from the code editor is displayed, showing a warning message:

CS0246 El nombre del tipo o el espacio de nombres 'ErrorViewModel' no se encontró (falta una directiva using o una referencia de ensamblado?)

The tooltip also shows the corrected code:

```

...
using SistemaInventario.AccesoDatos.Data;
+using SistemaInventario.Modelos.ViewModels;
...
```

At the bottom of the tooltip, it says "Vista previa de cambios".

- En Views/_ViewImports.cshtml corregimos la referencia.

```
@using SistemaInventario.Modelos
```

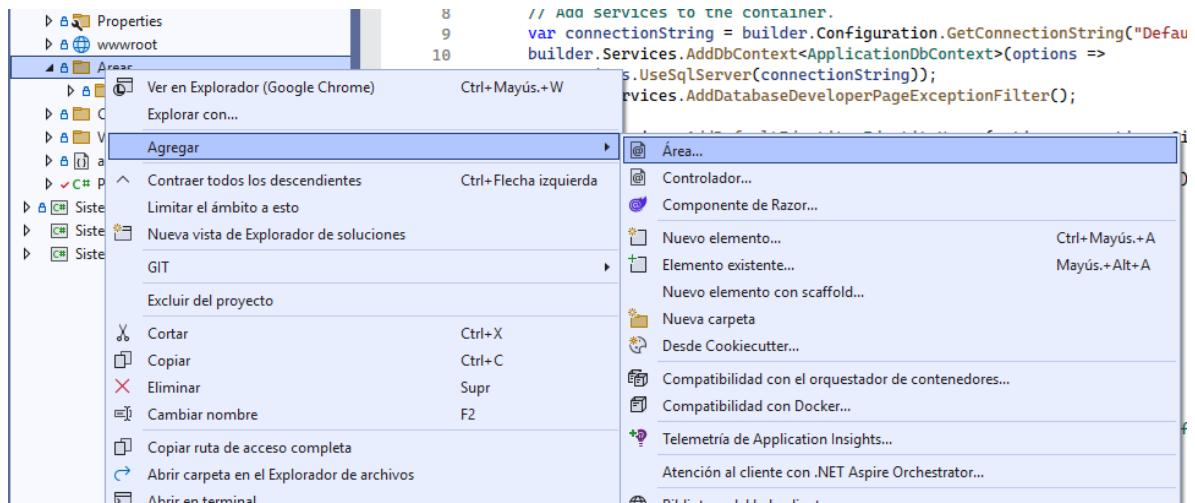
- En View/Shared/Error.cshtml corregimos la referencia.

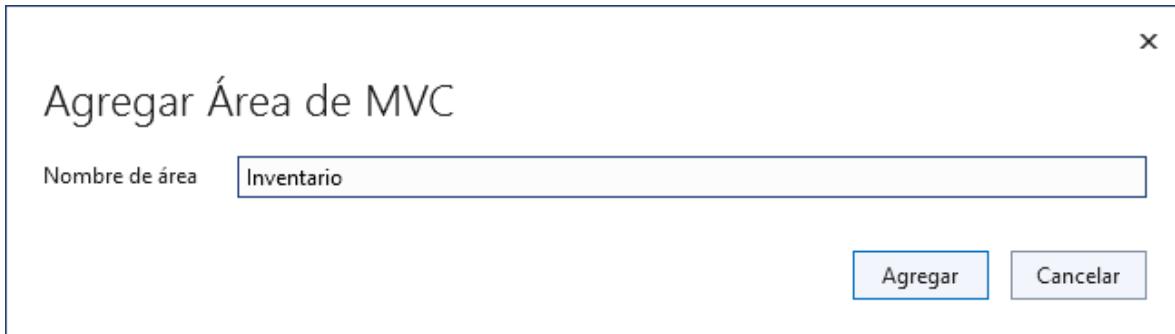
```
@using SistemaInventario.Modelos.ViewModels
@model ErrorViewModel
```

- Compilamos para verificar errores.

5. Agregar Área Inventory

- En el Proyecto SistemaInventario/Areas agregamos un área de MVC Inventory.





- En Programs.cs adicionamos el área Inventory.

```
app.MapControllerRoute  
    name: "default",  
    pattern: "{area=Inventory}/{controller=Home}/{action=Index}/{id?}");  
app.MapRazorPages();
```

- En el área Inventory eliminamos las carpetas Data y Models.
- Movemos HomeController.cs de la carpeta Controllers a la carpeta Inventory/Controllers y eliminamos la carpeta Controllers.
- Modificamos el archivo HomeController.cs.

```
[Area("Inventory")]  
public class HomeController : Controller  
{  
    .  
    .  
    .
```

- Movemos la carpeta Home de la carpeta Views a la carpeta Inventory/Views.
- Copiamos los archivos _ViewImports.cshtml y _ViewStart.cshtml a la carpeta Inventory/Views
- Modificamos _ViewStart.cshtml

```
@{  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}
```

6. Agregar Área Admin

- En el área Admin eliminamos las carpetas Data y Models
- Copiamos los archivos _ViewImports.cshtml y _ViewStart.cshtml de la carpeta Inventory/Views a la carpeta Admin/Views

7. Agregar Menú Desplegable

- Abrimos el archivo View/Shared/_Layout.cshtml para agregar el menú.

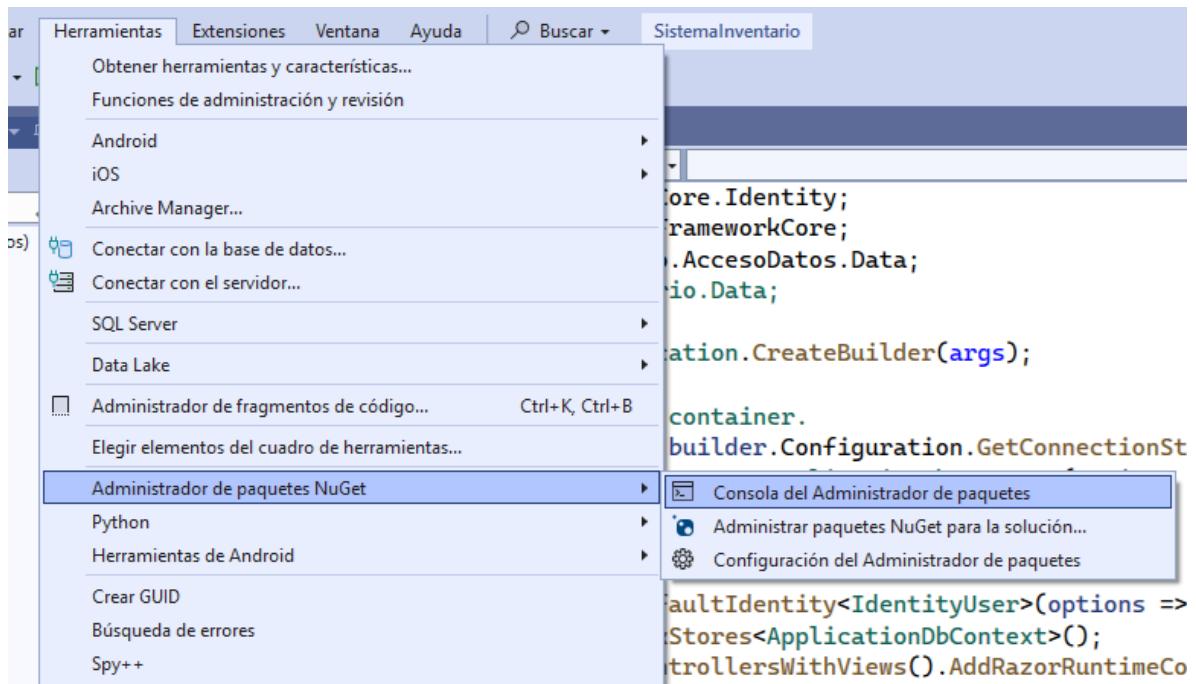
```
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown" aria-expanded="false">
        Contenido
    </a>
    <ul class="dropdown-menu">
        <li><a class="dropdown-item" asp-area="Admin" asp-controller="Bodega" asp-action="Index">Bodegas</a></li>
    </ul>
</li>
```

8. Migracion Inicial de Base de datos

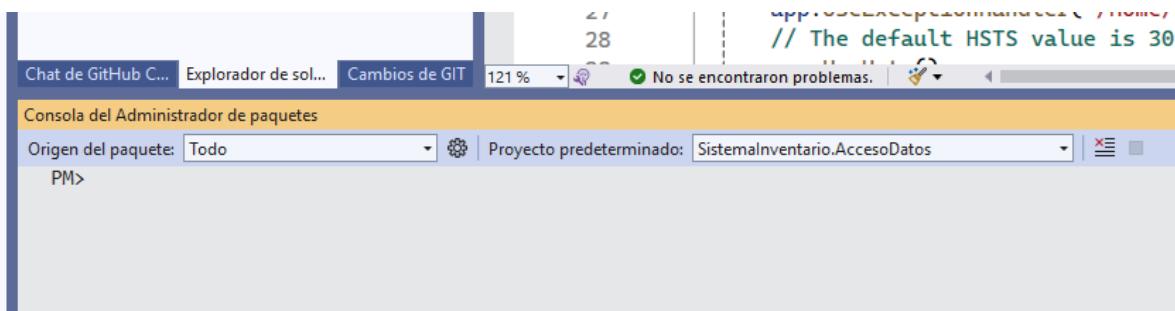
- Modificamos el archivo appsettings.json con los datos de la conexión a sql server

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=jorge-
win11\\SQLEXPRESS;Database=SistematicInventoryDB;Trusted_Connection=True;MultipleActiveResultSets=true
;TrustServerCertificate=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

- Abrimos la terminal del Administrador de paquetes NuGet



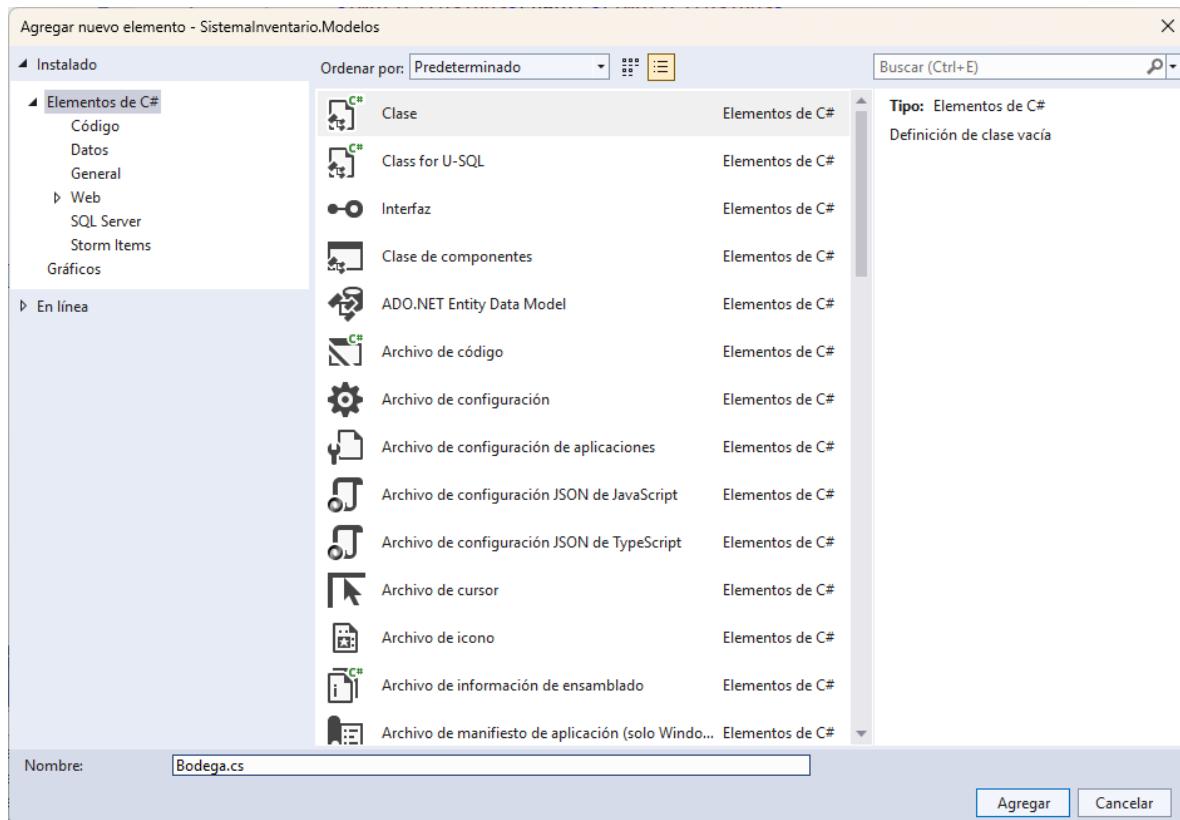
- Seleccionamos el SistematicInventory.AccesoDatos



- Ejecutamos los siguientes comandos para la migración:
 - add-migration AgregarMigracionIdentidad
 - update-database
- Verificar en SQL Server la base de datos creada.

9. Agregar Modelo Bodega

- En SistematicInventory.Modelos agregamos una nueva clase llamada Bodega.cs.



Bodega.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistematicInventory.Modelos
{
    public class Bodega
    {
        [Key]
        public int Id { get; set; }
        [Required(ErrorMessage = "Nombre es requerido")]
        [MaxLength(60, ErrorMessage = "Nombre debe ser maximo 60 caracteres")]
        public string Nombre { get; set; }
        [Required(ErrorMessage = "Descripcion es requerido")]
        [MaxLength(100, ErrorMessage = "Descripcion debe ser maximo 100 caracteres")]
        public string Descripcion { get; set; }
        [Required(ErrorMessage = "Estado es requerido")]
        public bool Estado { get; set; }
    }
}
```

- En el archivo SistematicInventory.AccesoDatos/ApplicationDbContext.cs agregamos el modelo Bodega.

```
using SistematicInventory.Modelos;
.
.
.
public DbSet<Bodega> Bodegas { get; set; }
```

10. Fluent API.

- Modificamos el archivo SistematicInventory.AccesoDatos/Data/ApplicationDbContext.cs agregando el método OnModelCreating.

```
using System.Reflection;
.
.
.
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
    builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
}
```

- Creamos una carpeta llamada Configuracion en el proyecto SistematicInventory.AccesoDatos
- En Configuracion creamos una clase llamada BodegaConfiguracion.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using SistematicInventory.Modelos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistematicInventory.AccesoDatos.Configuracion
{
    public class BodegaConfiguracion : IEntityTypeConfiguration<Bodega>
    {
        public void Configure(EntityTypeBuilder<Bodega> builder)
        {
            builder.Property(x => x.Id).IsRequired();
            builder.Property(x => x.Nombre).IsRequired().HasMaxLength(60);
            builder.Property(x => x.Descripcion).IsRequired().HasMaxLength(100);
            builder.Property(x => x.Estado).IsRequired();
        }
    }
}
```

```
}
```

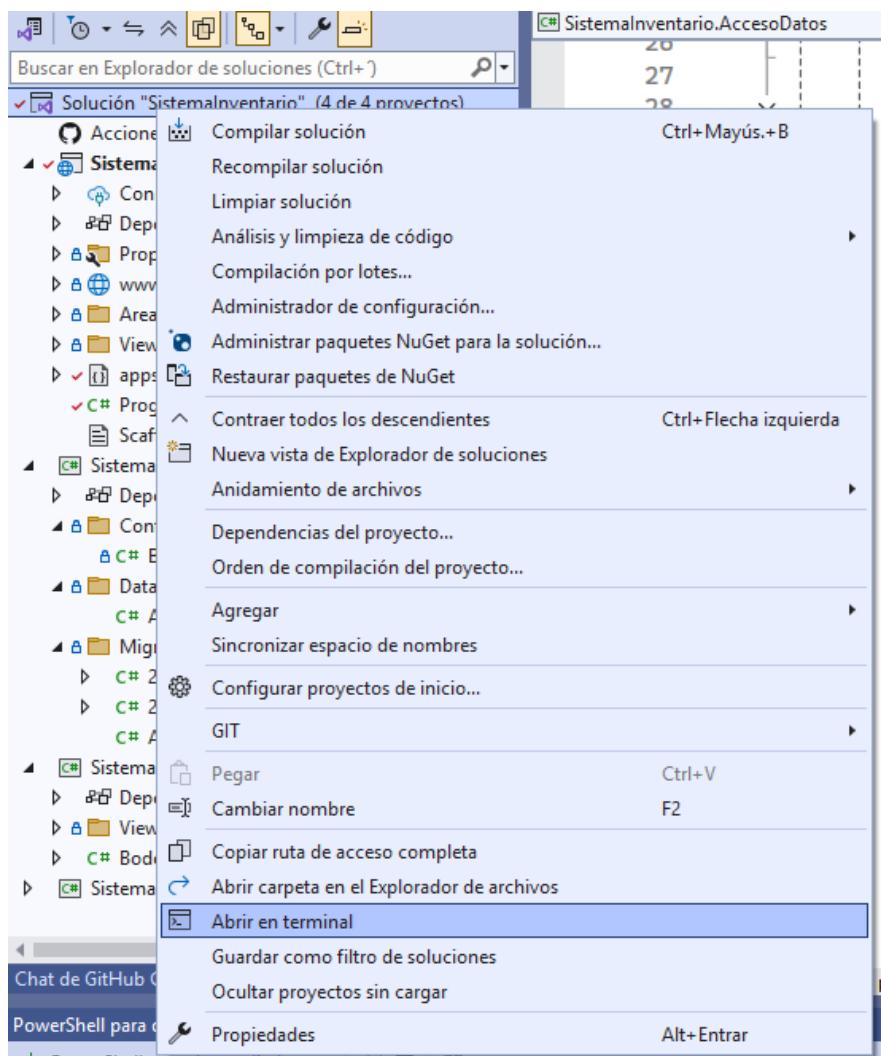
- En el terminal del Administrador de paquetes NuGet ejecutamos:

- add-migration AgregarBodegaMigracion
 - update-database

- Verificar en SQL Server la base de datos creada

11. Subir cambios a GitHub (opcional)

- Abrimos terminal de soluciones



- Ejecutar comandos:

- git add .
 - git commit -m "Configuracion del proyecto"
 - git push -u origin main