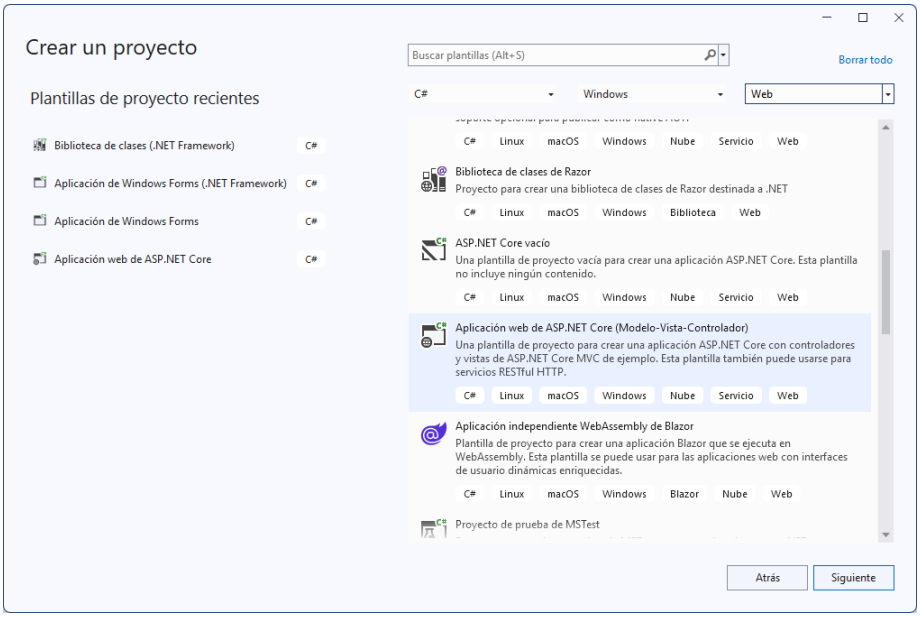
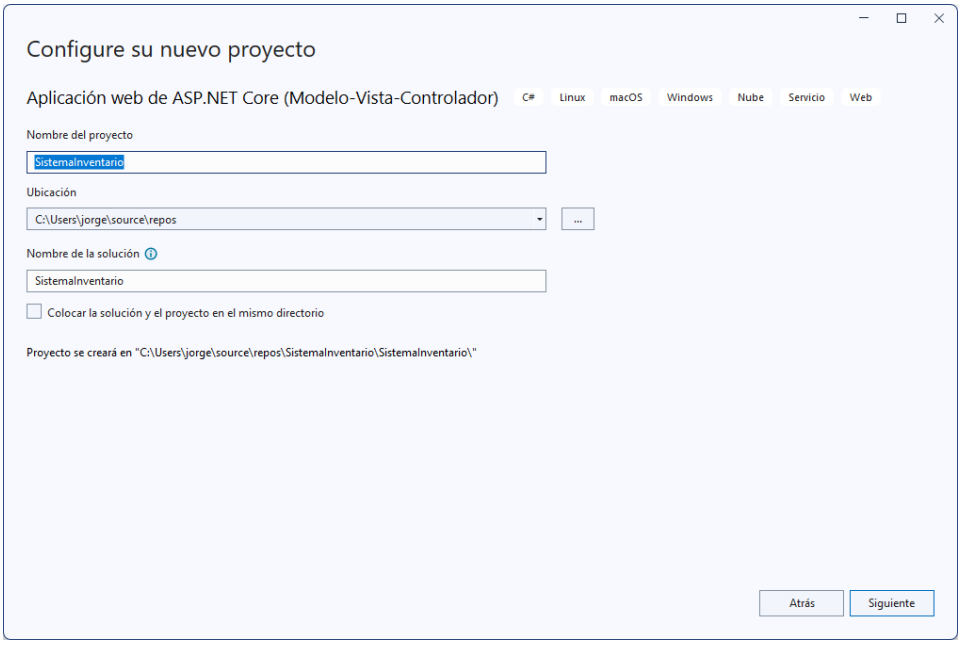
PROYECTO ASP.Net parte 1

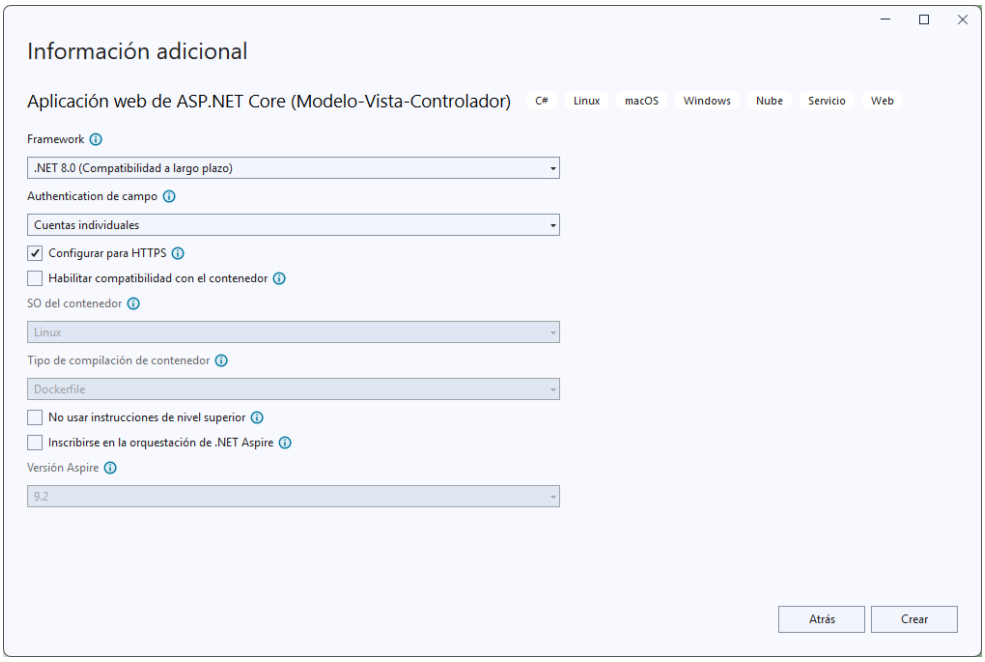
1. Crear nuevo proyecto ASP.Net

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.





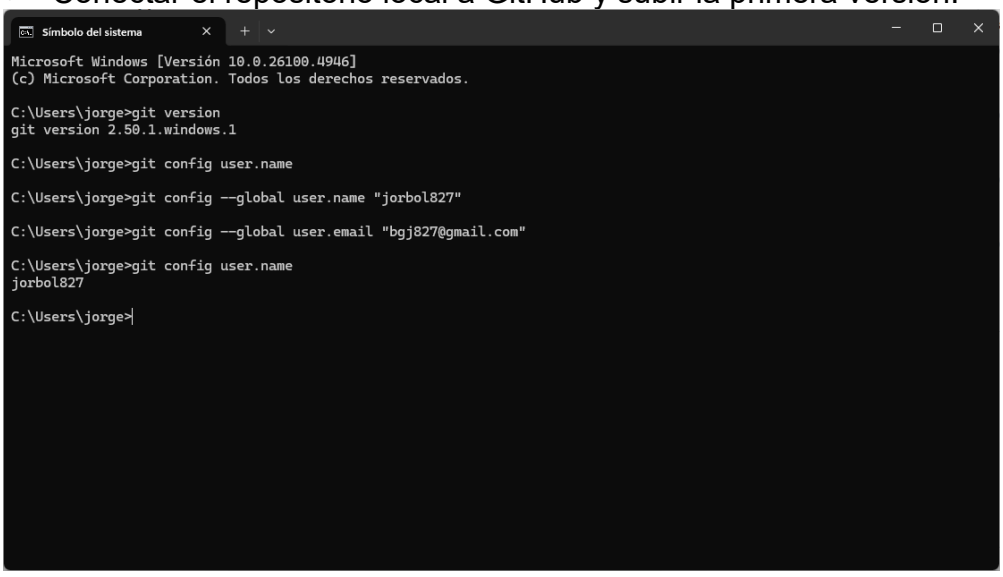


2. Usar Git y GitHub en nuestro Proyecto

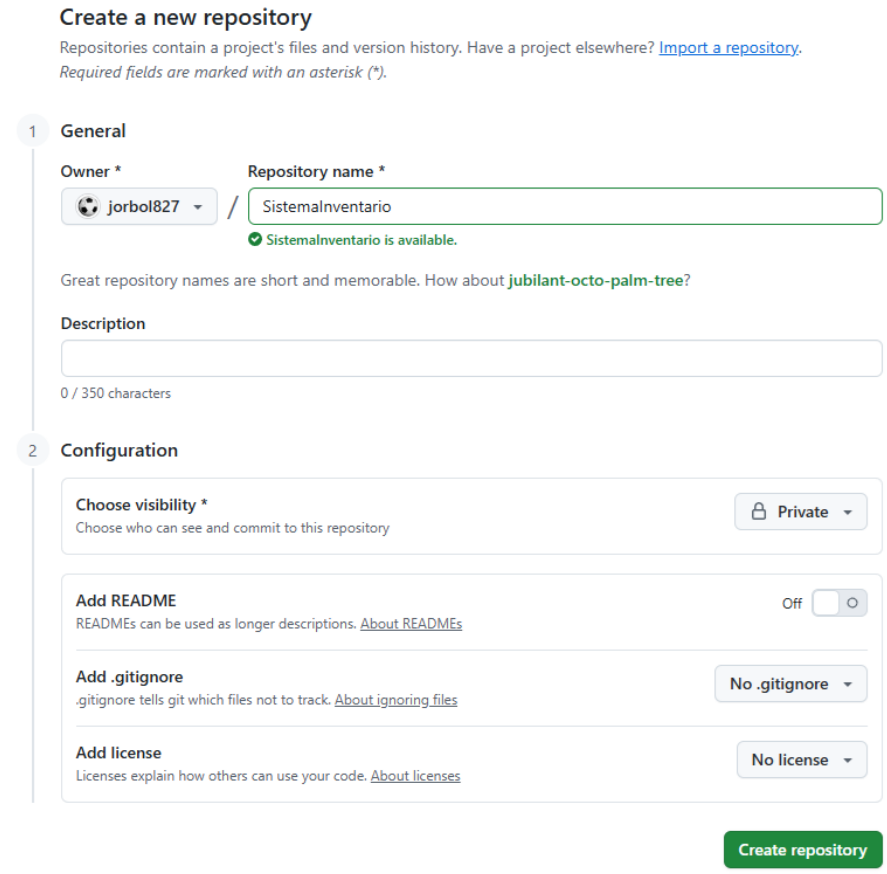
Se recomienda versionar el proyecto con **Git** para llevar control de cambios.

Pasos clave:

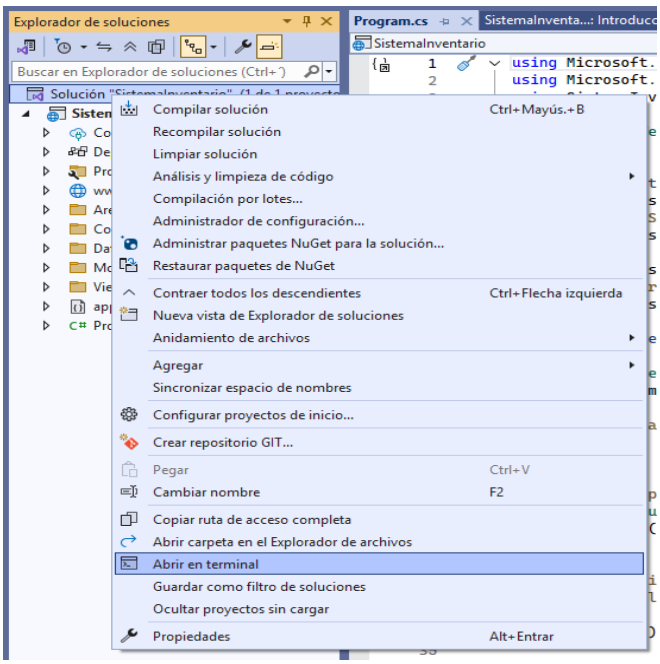
- Inicializar Git en la carpeta del proyecto (git init).
- Crear archivo .gitignore para excluir binarios y configuraciones temporales (dotnet new gitignore).
- Conectar el repositorio local a GitHub y subir la primera versión:



- Creamos nuevo repositorio

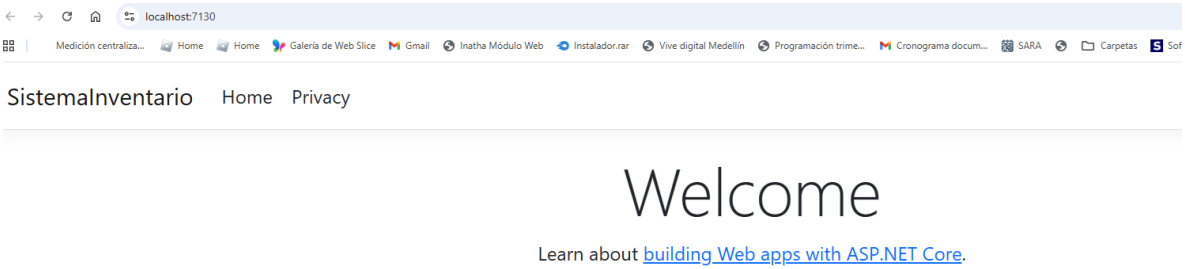


- Abril en terminal (opcional) o en el cmd



- 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/jorbol827/SistemaInventario.git`
  - `git push -u origin main`

3. Enrutamiento en MVC

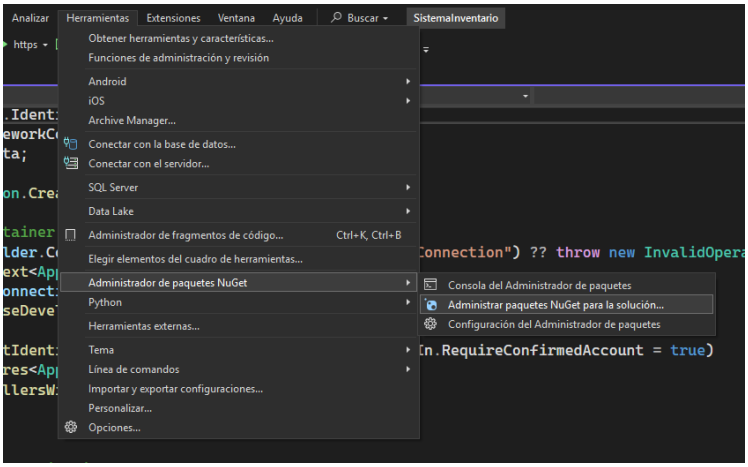


Configurar el Proyecto

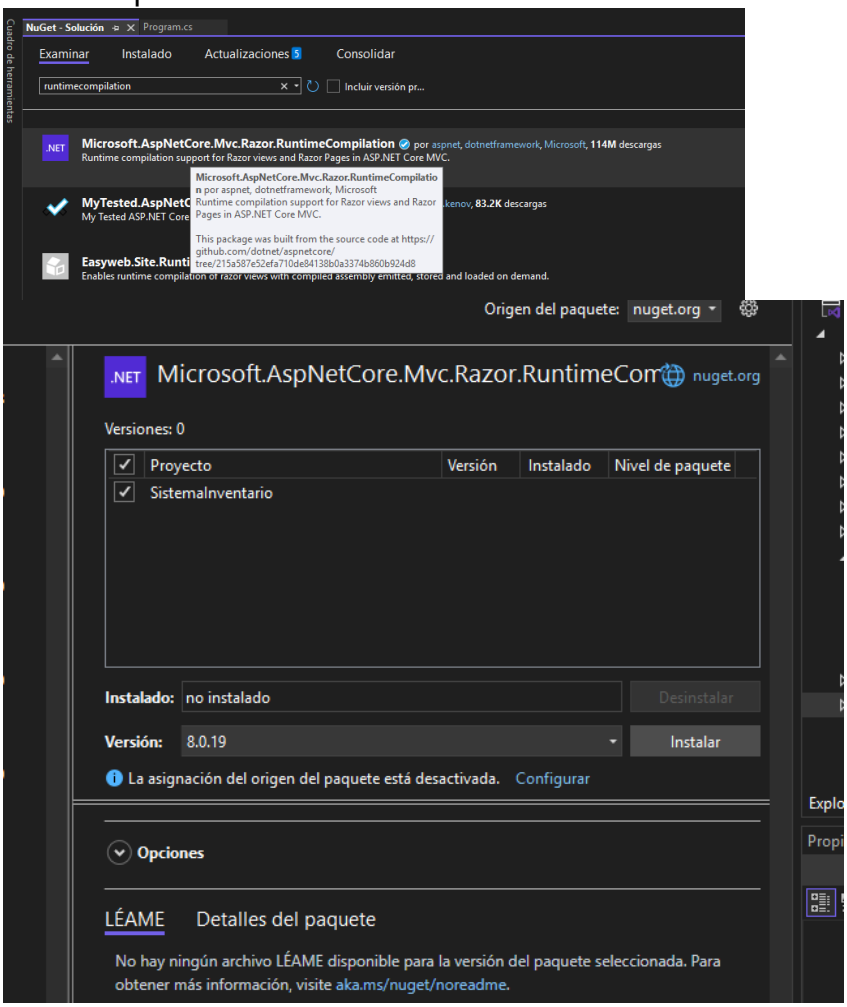
Para habilitar **compilación en caliente de vistas**, se instala el paquete:

Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation.

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

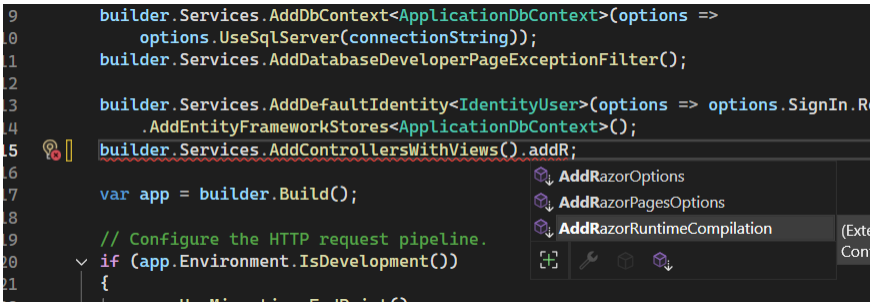


- En la pestaña Examinar



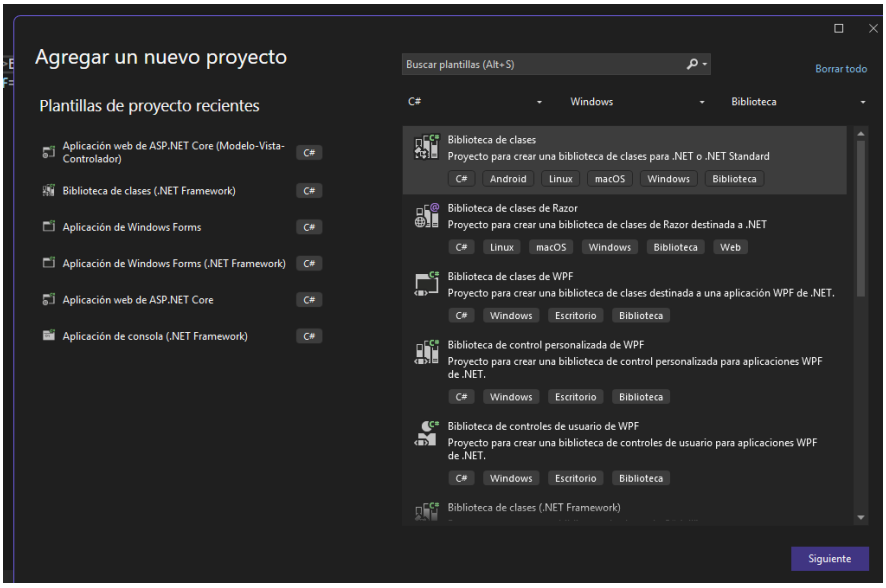
- Agregamos el servicio AddRazorRuntimeCompilation() en Program.cs y verificamos funcionamiento

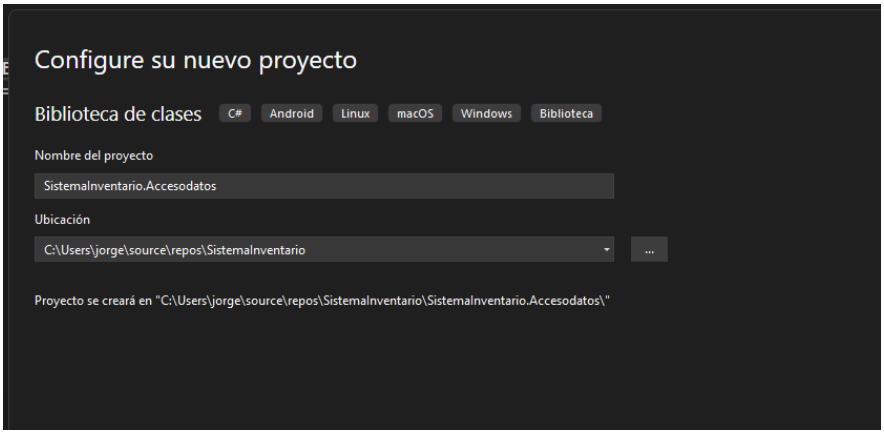
```
builder.Services.AddControllersWithViews()
    .AddRazorRuntimeCompilation();
```



2. Crear Proyectos dentro de la Solución

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





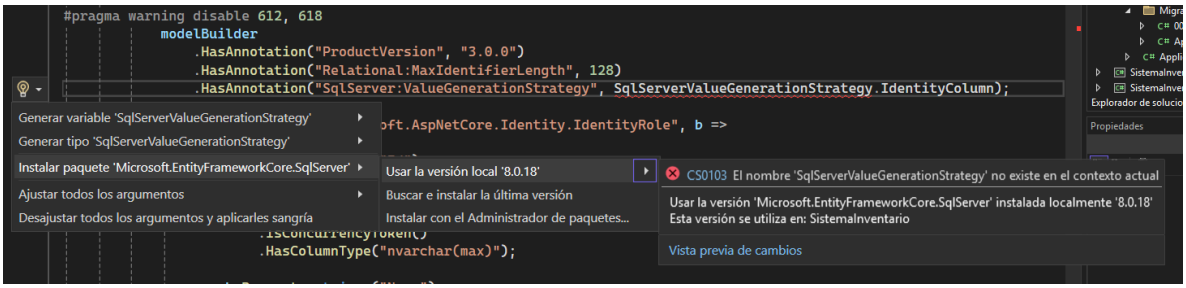
- Eliminar la clase Class1.cs que se crea en el nuevo proyecto.

- Agregamos otro nuevo proyecto tipo Biblioteca de Clase llamado SistemaInventario.Modelos y eliminamos la clase.
- Agregamos otro nuevo proyecto tipo Biblioteca de Clase llamado SistemaInventario.Utilidades y eliminamos la clase.

3. Separar la capa de Datos

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

- Dentro del proyecto SistemaInventario.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 SistemaInventario.AccesoDatos.Data

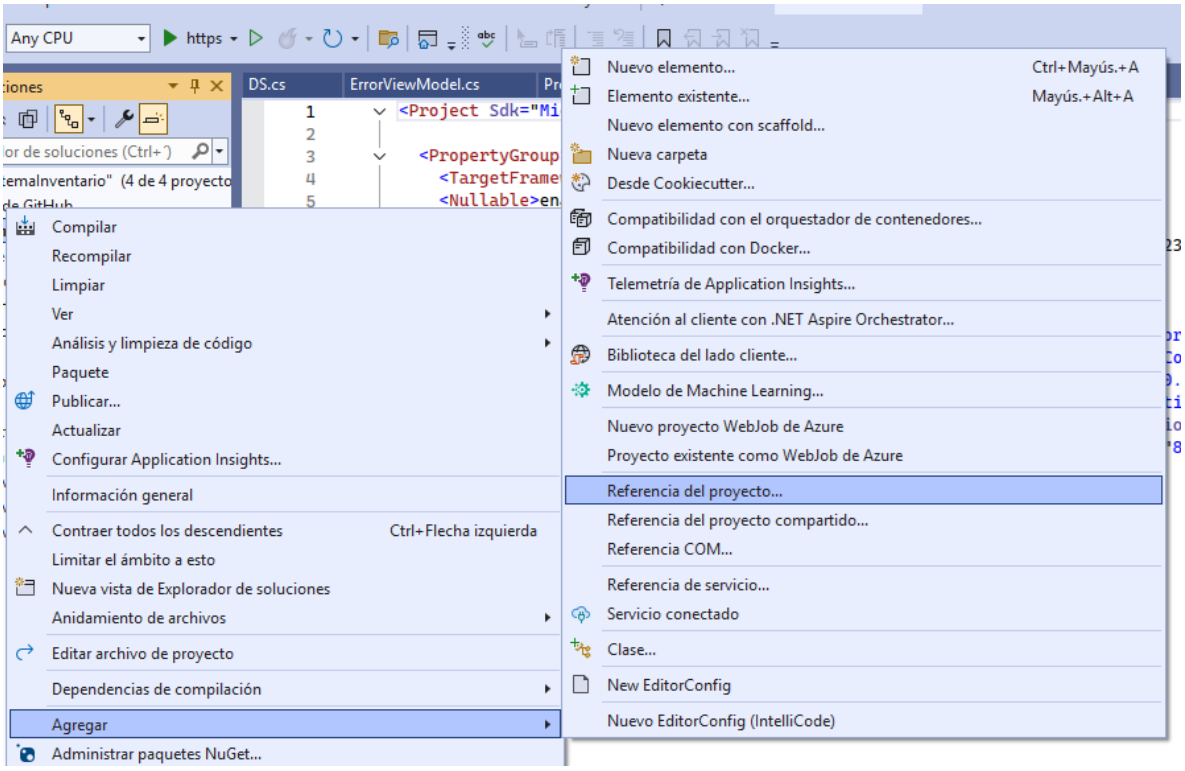
4. Separar Modelos y Agregar Referencias

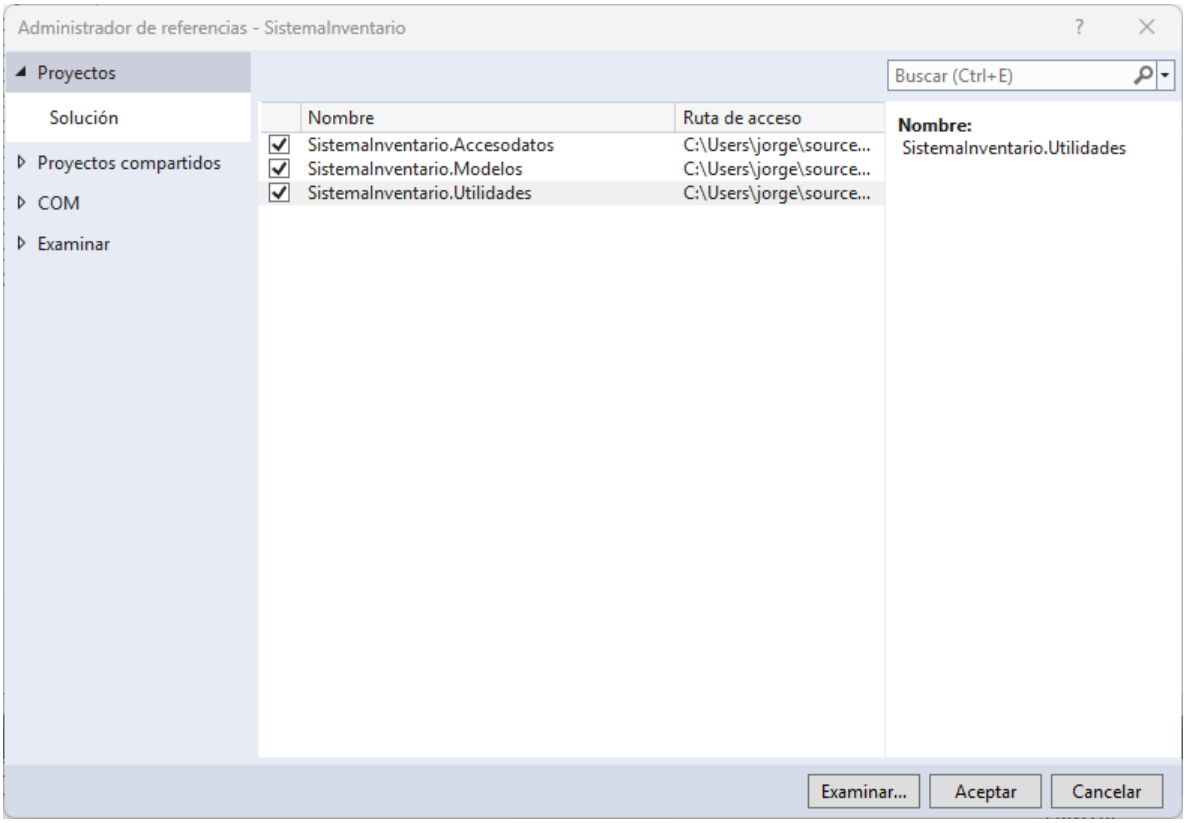
- Movemos la carpeta Models al proyecto SistemaInventario.Modelos y eliminamos la carpeta Models del proyecto SistemaInventario.
- Movemos el archivo ErrorViewModel.cs al raíz del directorio del proyecto SistemaInventario.Modelos y eliminamos la carpeta Models
- En el proyecto SistemaInventario.Modelos creamos carpeta ViewModels y movemos el archivo ErrorViewModel.cs a esa carpeta.

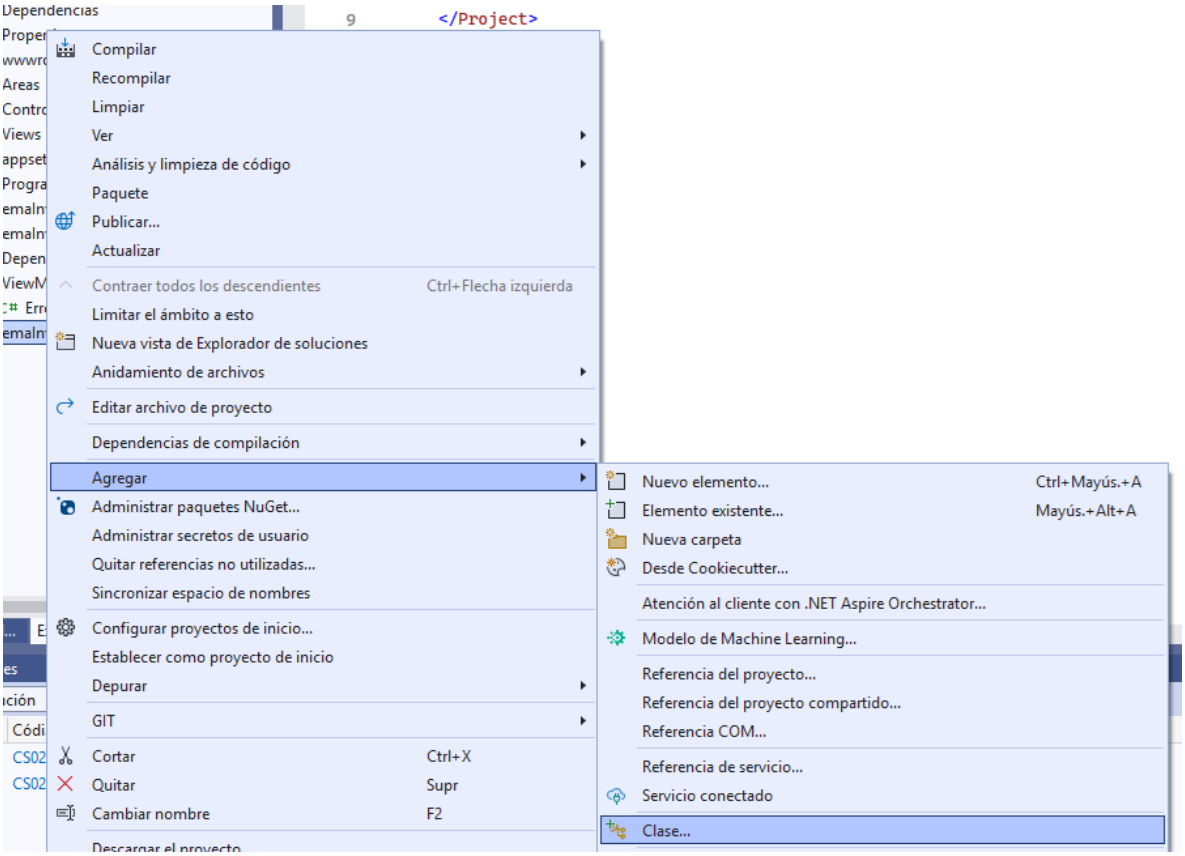
- En el proyecto SistemaInventario.Utilidades creamos una clase estática DS.

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

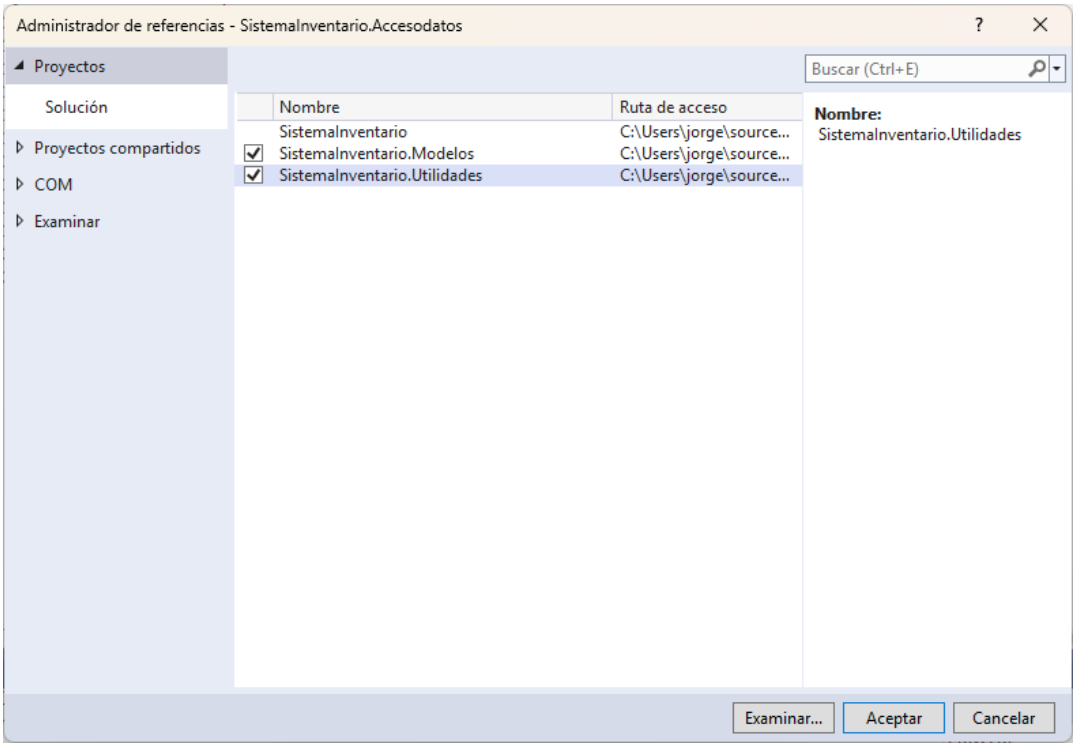
- En el proyecto SistemaInventario agregamos referencias


Any CPU
https
DS.cs
ErrorViewModel.cs
Compilar
Recompilar
Limpiar
Ver
Análisis y limpieza de código
Paquete
Publicar...
Actualizar
Configurar Application Insights...
Información general
Contraer todos los descendientes Ctrl+Flecha izquierda
Limitar el ámbito a esto
Nueva vista de Explorador de soluciones
Anidamiento de archivos
Editar archivo de proyecto
Dependencias de compilación
Agregar
Administrar paquetes NuGet...
Nuevo elemento... Ctrl+Mayús.+A
Elemento existente... Mayús.+Alt+A
Nuevo elemento con scaffold...
Nueva carpeta
Desde Cookiecutter...
Compatibilidad con el orquestador de contenedores...
Compatibilidad con Docker...
Telemetría de Application Insights...
Atención al cliente con .NET Aspire Orchestrator...
Biblioteca del lado cliente...
Modelo de Machine Learning...
Nuevo proyecto WebJob de Azure
Proyecto existente como WebJob de Azure
Referencia del proyecto...
Referencia del proyecto compartido...
Referencia COM...
Referencia de servicio...
Servicio conectado
Clase...
New EditorConfig
Nuevo EditorConfig (IntelliCode)

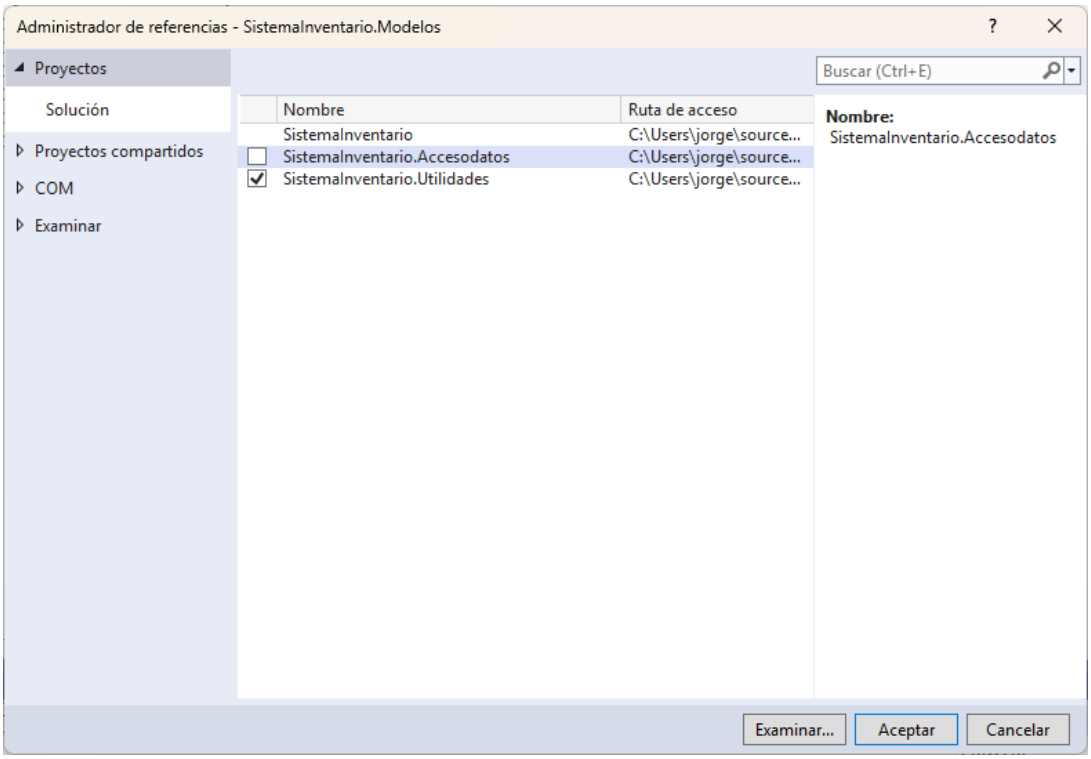

Administrador de referencias - SistemaInventario
Proyectos
Solución
Proyectos compartidos
COM
Examinar
Buscar (Ctrl+E)
Nombre
Ruta de acceso
SistemaInventario.Accesodatos C:\Users\jorge\source...
SistemaInventario.Modelos C:\Users\jorge\source...
SistemaInventario.Utilidades C:\Users\jorge\source...
Nombre:
SistemaInventario.Utilidades
Examinar...
Aceptar
Cancelar



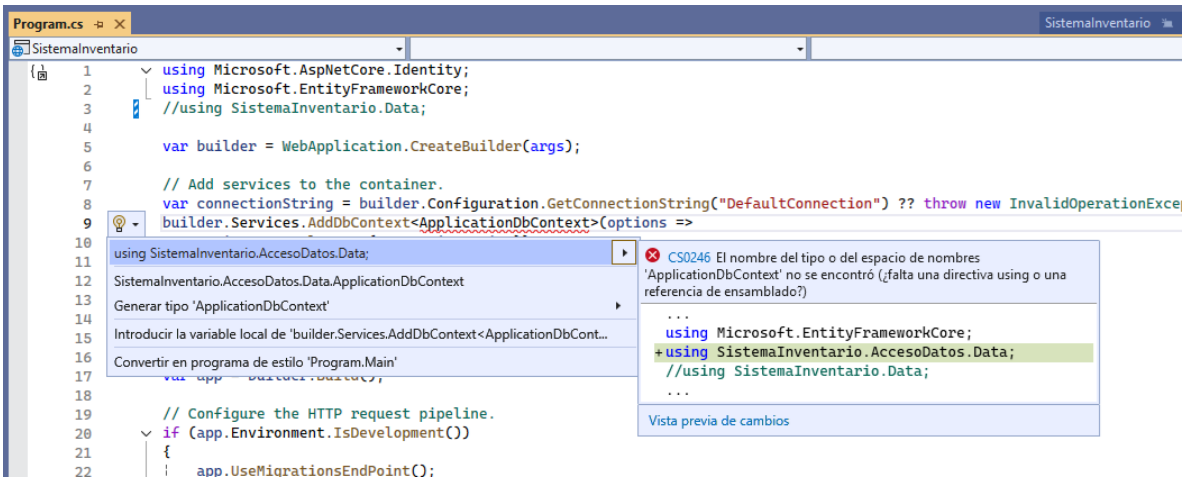
- En el proyecto SistemaInventario.AccesoDatos agregamos referencias



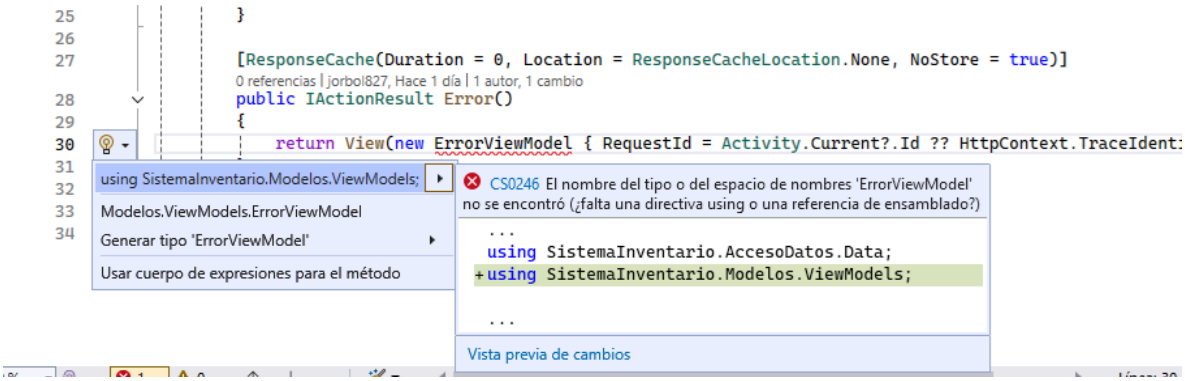
- En el proyecto SistemaInventario.Modelos agregamos referencias



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



- Abrimos Controllers/HomeController.cs y corregimos errores



- En Views/ _ViewImports.cshtml corregimos la referencia.

@using SistemaInventario.Modelos

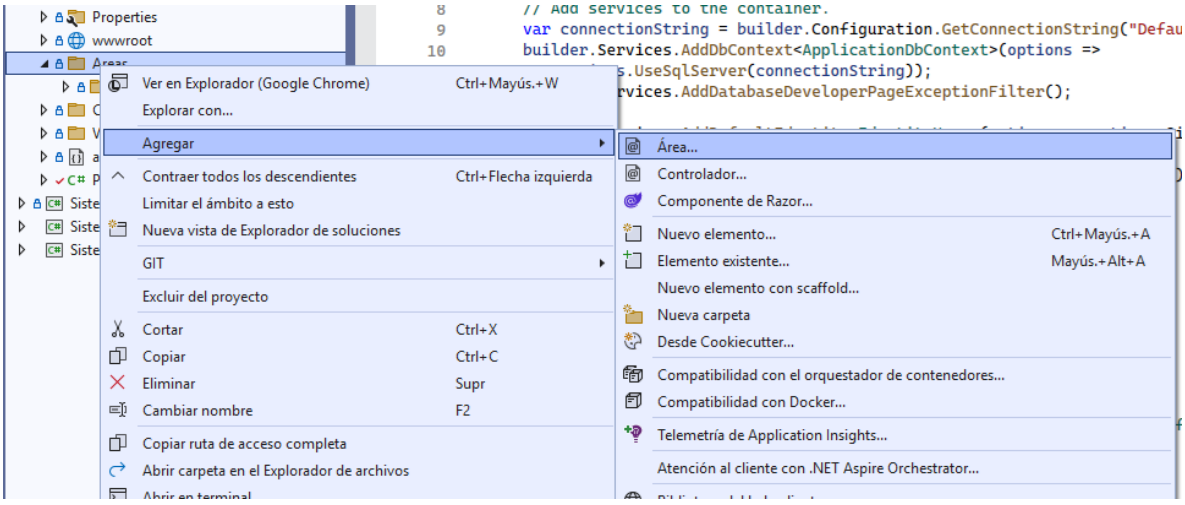
- En View/Shared/Error.cshtml corregimos la referencia.

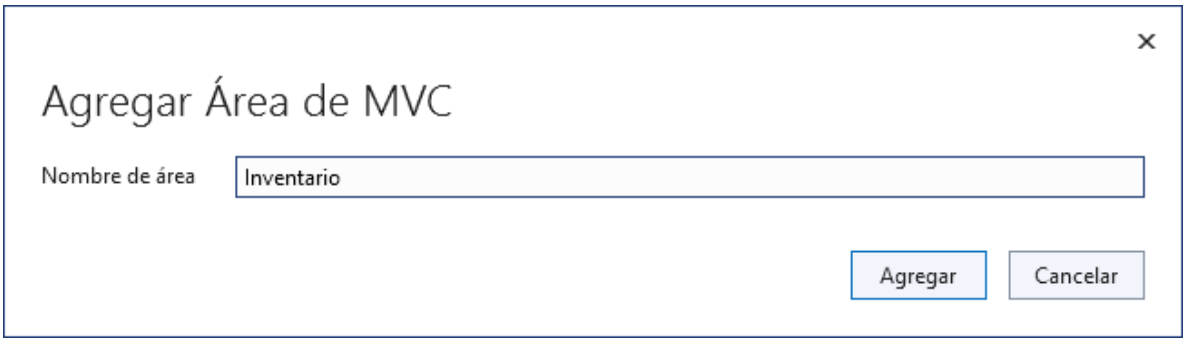
@using SistemaInventario.Modelos.ViewModels
@model ErrorViewModel

- Compilamos para verificar errores.

5. Agregar Área Inventario

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





- En Programs.cs adicionamos el área Inventario.

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

- En el área Inventario eliminamos las carpetas Data y Models.
- Movemos HomeControllar.cs de la carpeta Controllers a la carpeta Inventario/Controllers y eliminamos la carpeta Controllers.

- Modificamos el archivo HomeControllar.cs.

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

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

7. Agregar Menú Desplegable

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

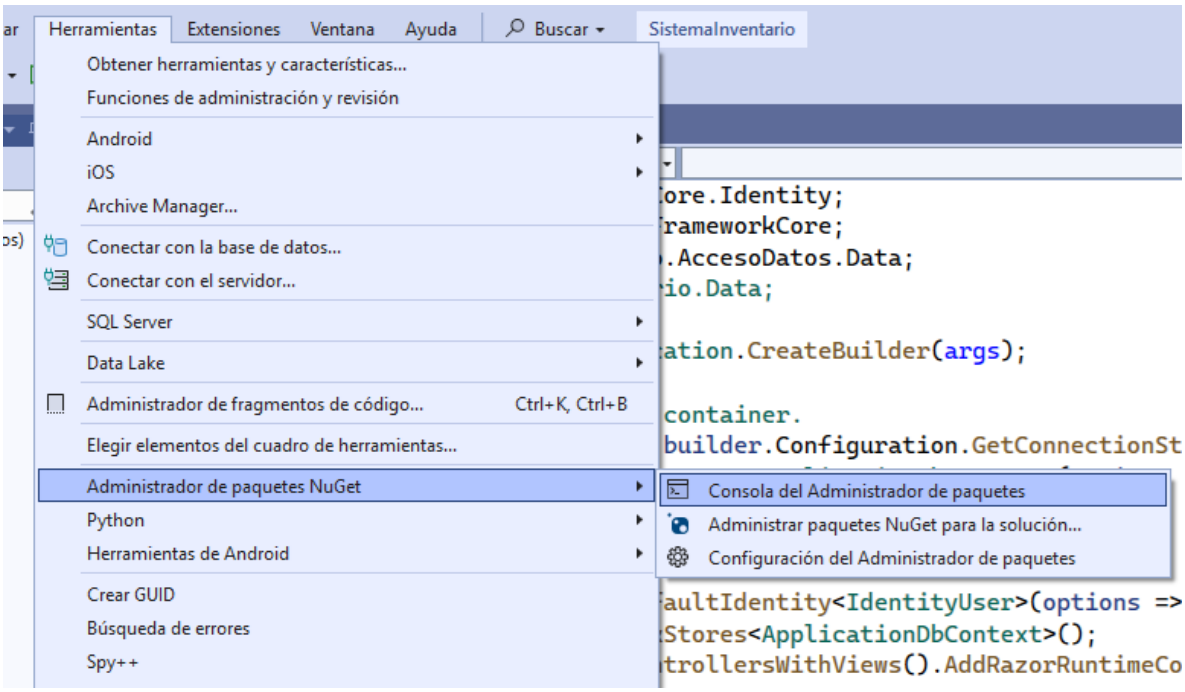
```
<li class="nav-item dropdown">
   <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown" aria-expanded="false">
      Contenido
   </a>
   <ul class="dropdown-menu">
      <li><a class="dropdown-item" asp-area="Admin" asp-controller="Bodega" asp-
action="Index">Bodegas</a></li>
 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

8. Migracion Inicial de Base de datos

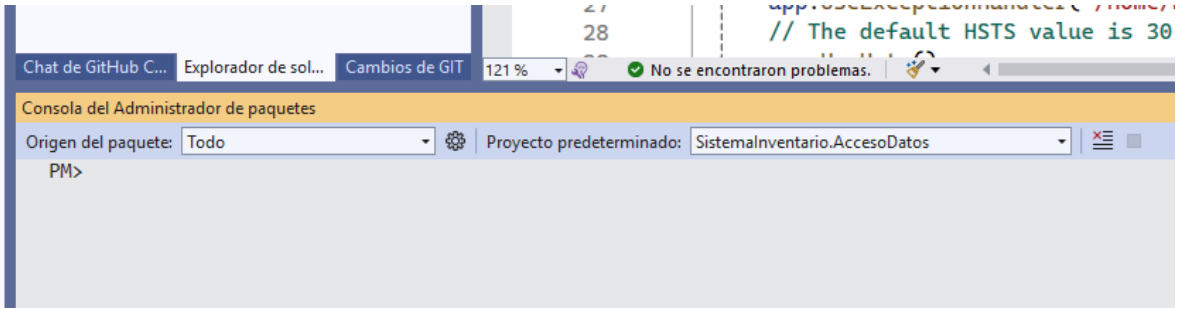
- Modificamos el archivo appsettings.json con los datos de la conexión a sql server

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

- Abrimos la terminal del Administrador de paquetes NuGet



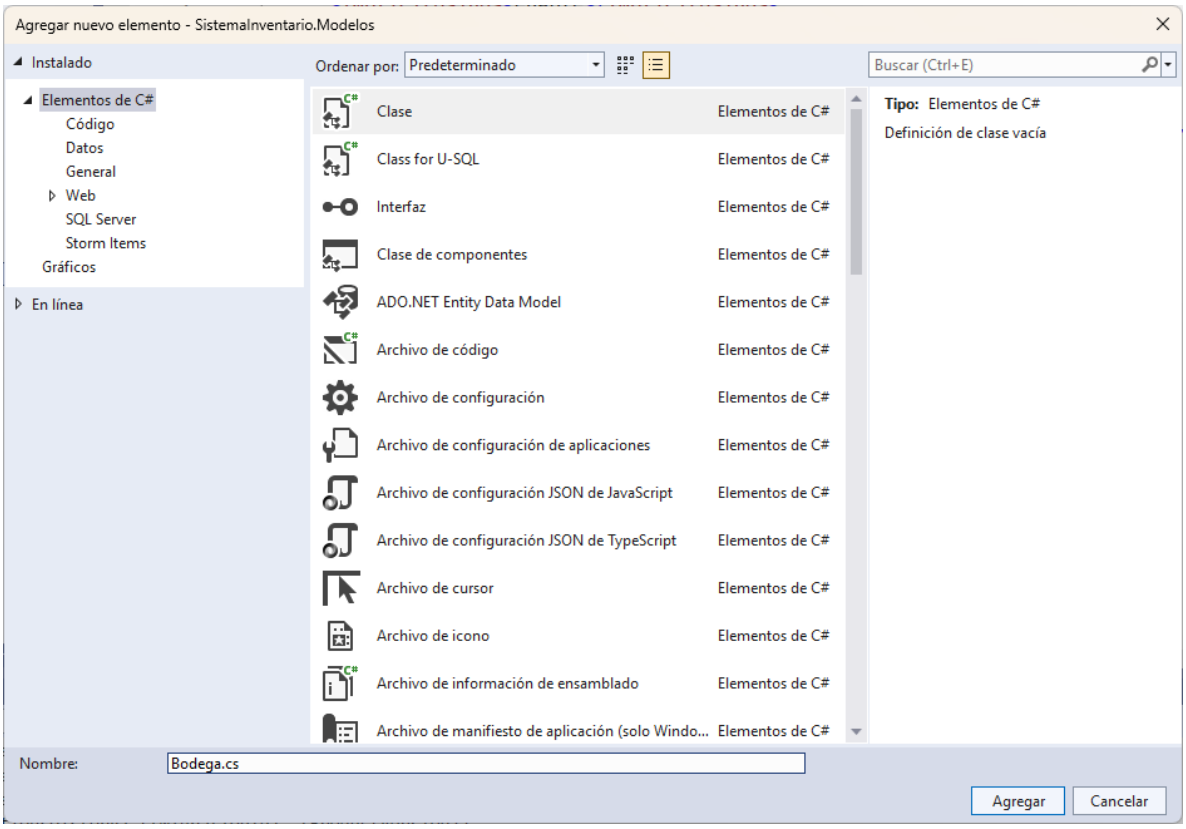
- Seleccionamos el SistemaInventario.AccesoDatos



- 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 SistemaInventario.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 SistemaInventario.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(10, 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 SistemaInventario.AccesoDatos/Data/ApplicationDbContext.cs agregamos el modelo Bodega.

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

10. Fluent API.

- Modificamos el archivo SistemaInventario.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 SistemaInventario.AccesoDatos

- En Configuracion creamos una clase llamada BodegaConfiguracion.cs

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

namespace SistemaInventario.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();
      }
```

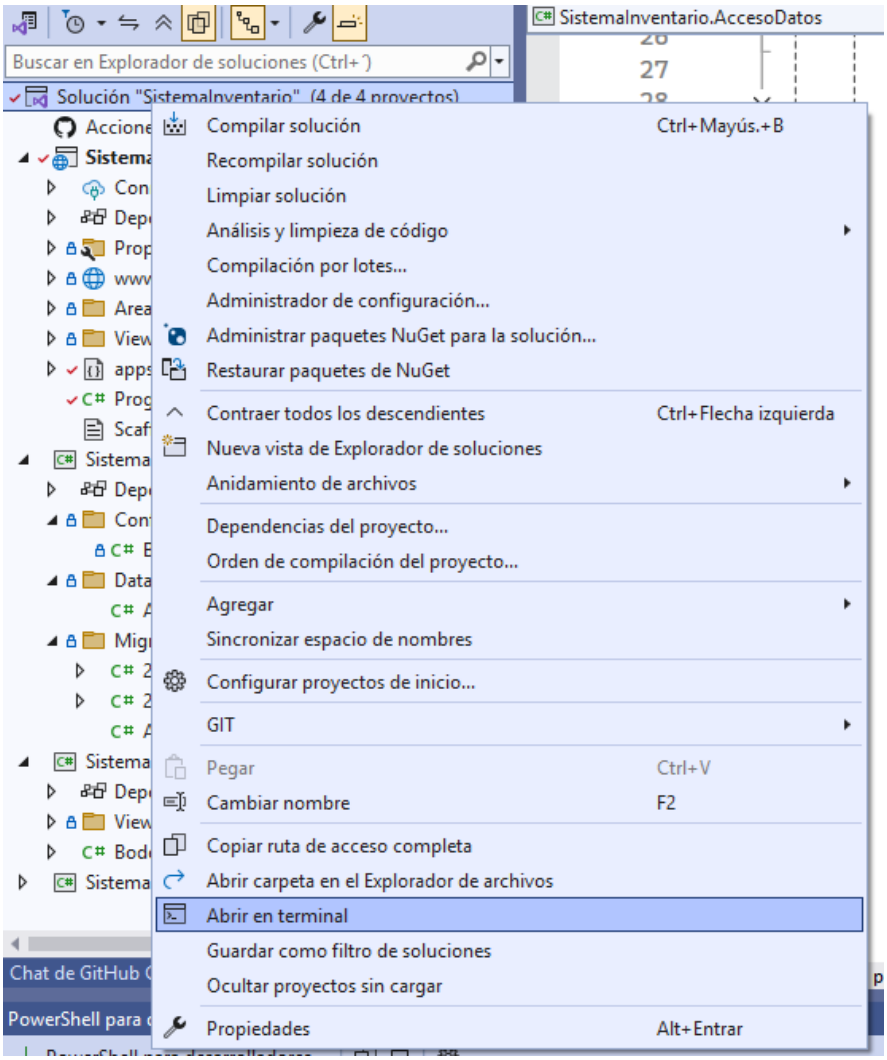
```
    }
}
```

- 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