



Desarrollo de Software Empresarial

CICLO 02 -2024

Guía 1: Introducción a ASP.NET MVC – Parte #1

Competencias

- Que el estudiante aprenda el uso del IDE Visual Studio para el diseño de aplicaciones web.
- Que el estudiante obtenga conocimientos básicos sobre la utilización del patrón MVC.
- Adquirir conocimientos y habilidades en la construcción de páginas web con ASP.NET MVC.

Introducción Teórica

Introducción a ASP.NET MVC en .NET Core

ASP.NET Core MVC es un marco de trabajo robusto para la creación de aplicaciones web utilizando la arquitectura Model-View-Controller (MVC) en el entorno de .NET Core. Esta guía práctica se enfoca en los fundamentos esenciales de MVC, proporcionando una comprensión clara de cómo estructurar y desarrollar aplicaciones web mediante la separación de responsabilidades entre modelos, vistas y controladores.

Controladores

Un controlador es una clase que gestiona las solicitudes HTTP entrantes, actuando como el intermediario entre el usuario y el sistema. Su función principal es recibir las solicitudes del navegador web, procesar la lógica de la aplicación correspondiente y luego determinar la respuesta adecuada. Los controladores contienen métodos llamados "acciones" que responden a diferentes tipos de solicitudes HTTP, como GET y POST. Estas acciones pueden interactuar con los modelos para obtener o manipular datos y luego devolver una vista al usuario. En resumen, los controladores en MVC manejan cómo se procesan y responden las solicitudes del usuario.

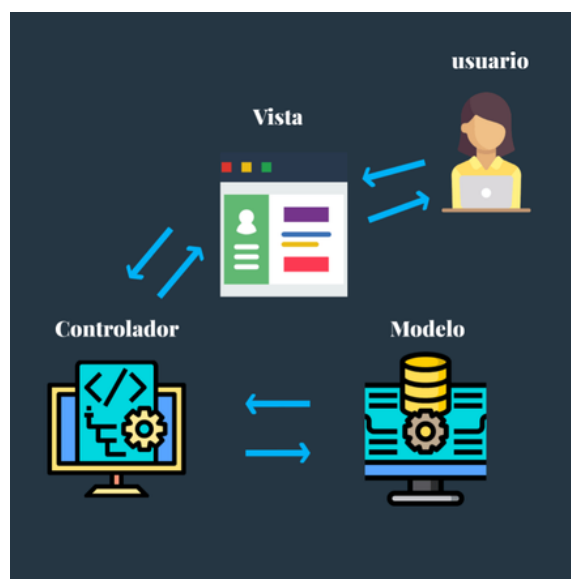
Vistas

Representa la interfaz de usuario visible para el usuario final. Es un archivo de plantilla que combina HTML con código C# (utilizando Razor) para generar la página que se muestra al usuario. Las vistas son responsables de presentar los datos al usuario de manera visualmente

atractiva y estructurada. Pueden contener elementos como formularios, tablas y elementos de estilo que muestran la información proporcionada por los controladores y modelos. Las vistas no contienen lógica de negocio compleja, pero pueden incluir lógica simple para iterar sobre datos o mostrar contenido condicional basado en la lógica del controlador.

Modelo

Representa la estructura de datos y la lógica de negocio de la aplicación. Es responsable de encapsular los datos que la aplicación maneja y definir la lógica para su manipulación. Los modelos suelen ser clases que contienen propiedades y métodos para interactuar con la capa de datos subyacente o para aplicar reglas de negocio específicas. En un contexto sin base de datos, el modelo puede simular datos estáticos o utilizar estructuras de datos simples para demostrar conceptos. Los controladores utilizan los modelos para obtener y manipular datos antes de enviarlos a las vistas para su presentación.



Material es y Equipos

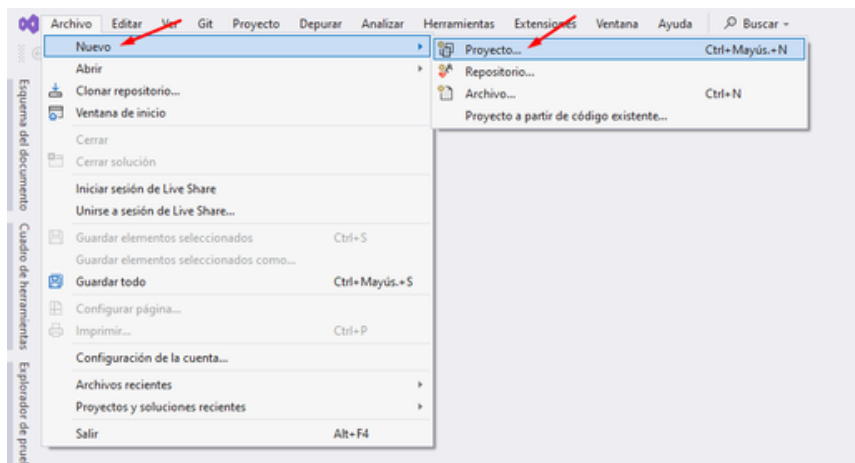
1. Guía No 1.
2. Computadora con programa Microsoft Visual Studio 2022.

Procedimiento

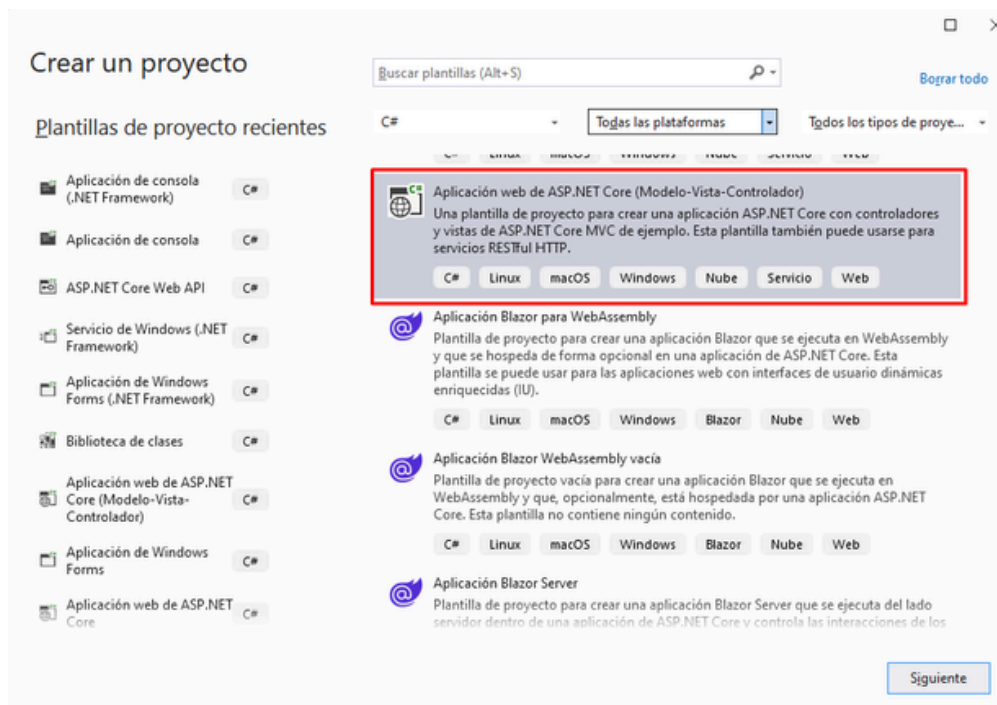
EJEMPLO No. 1: Creación de un nuevo proyecto ASP MVC

Para la creación del Proyecto, por favor siga las indicaciones que se detallan a continuación. 1.

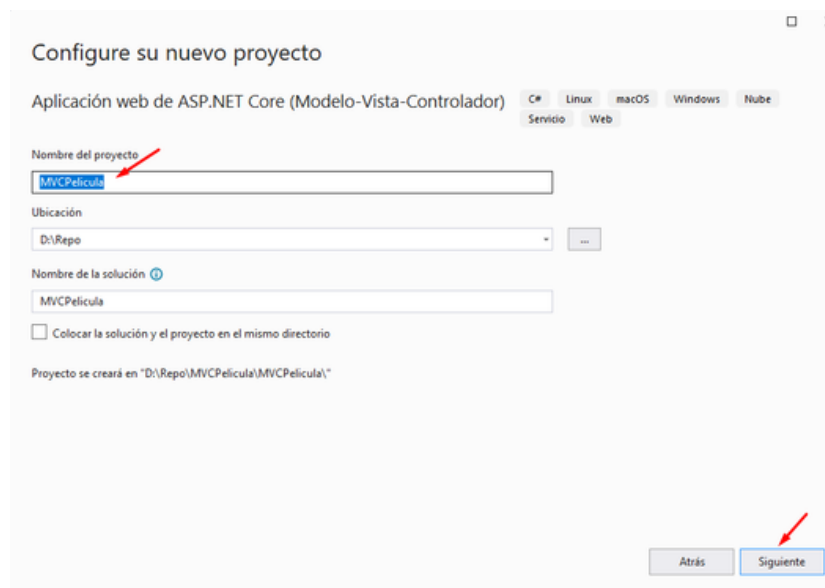
1. Inicie Visual Studio.NET
2. En el menú **"Archivo (FILE)"**, seleccione **"Nuevo (New)"** y, después seleccione la opción **"Proyecto (Project)"**.



En el panel “**Lenguaje**”, seleccione “**C#**”, y busque y seleccione el tipo “**Aplicación Web ASP.NET Core (Modelo-Vista-Controlador)**”.



Nombre a su proyecto “**MVCPelicula**”. Luego, haga clic en el botón “**Siguiente**”.



Configure su nuevo proyecto

Aplicación web de ASP.NET Core (Modelo-Vista-Controlador)

C# Linux macOS Windows Nube
Servicio Web

Nombre del proyecto
MVCPelicula

Ubicación
D:\Repo

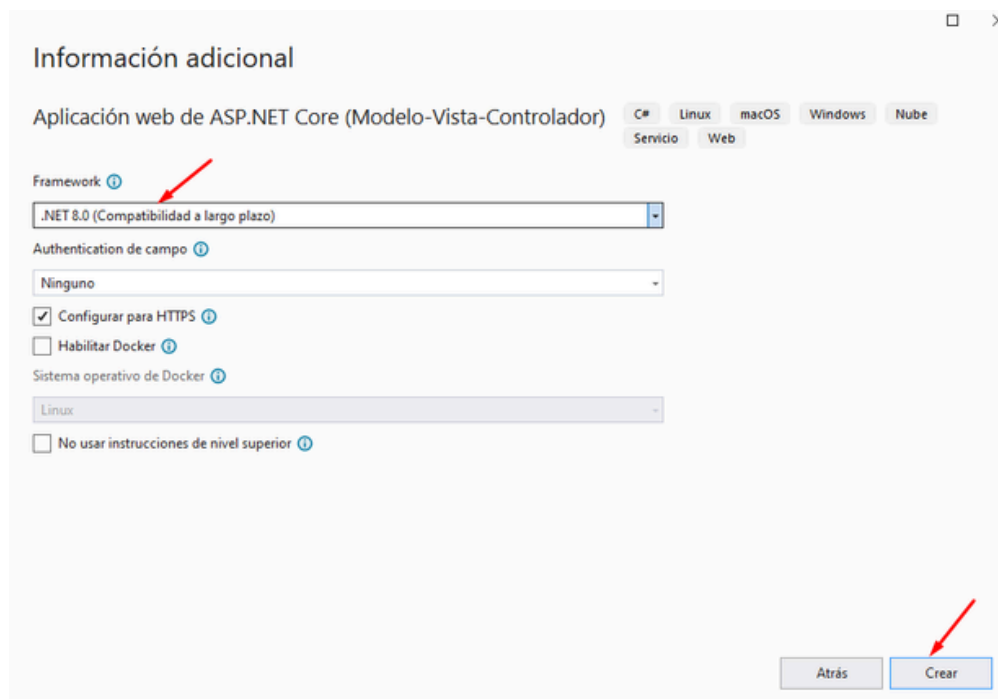
Nombre de la solución
MVCPelicula

☐ Colocar la solución y el proyecto en el mismo directorio

Proyecto se creará en "D:\Repo\MVCPelicula\MVCPelicula\."

Atrás Siguiente

Finalmente, seleccione como Framework “**.NET 8.0**” y de clic en el botón “**Crear**”.



Información adicional

Aplicación web de ASP.NET Core (Modelo-Vista-Controlador)

C# Linux macOS Windows Nube
Servicio Web

Framework
.NET 8.0 (Compatibilidad a largo plazo)

Authentication de campo
Ninguno

☒ Configurar para HTTPS

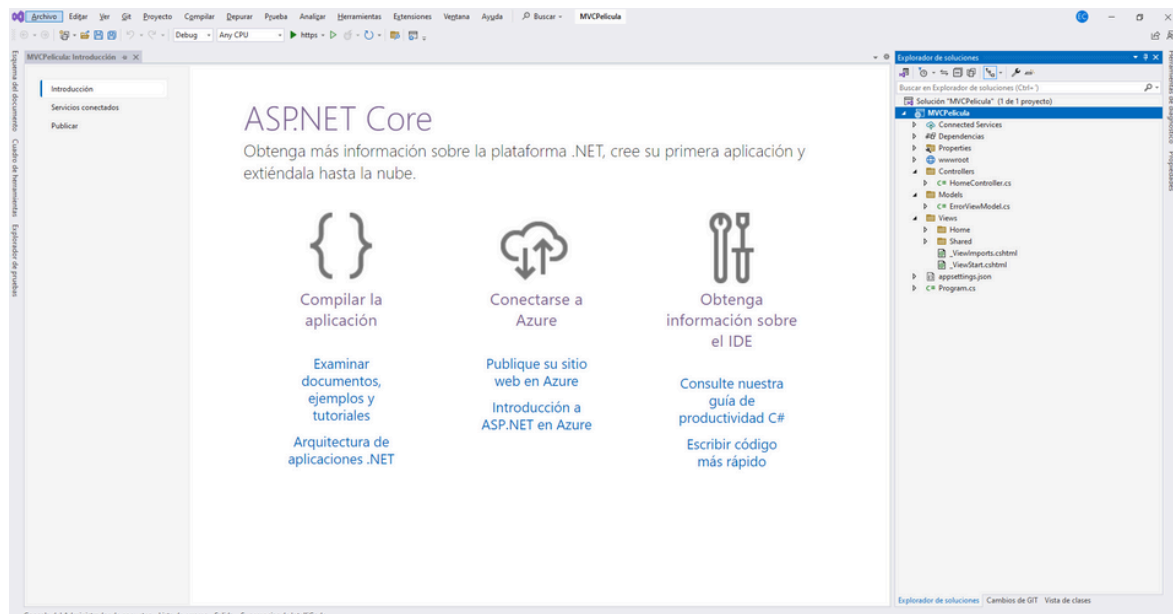
☐ Habilitar Docker

Sistema operativo de Docker
Linux

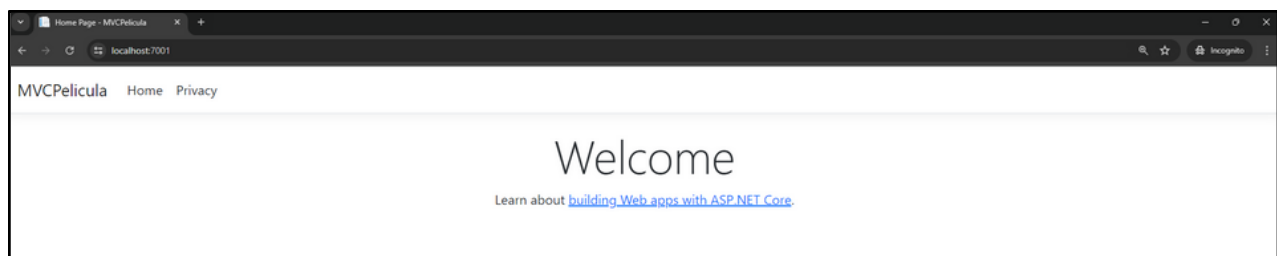
☐ No usar instrucciones de nivel superior

Atrás Crear

Visual Studio usó una plantilla predeterminada para el proyecto ASP.NET MVC que acaba de crear, por lo que ahora tiene una aplicación que funciona sin hacer nada.

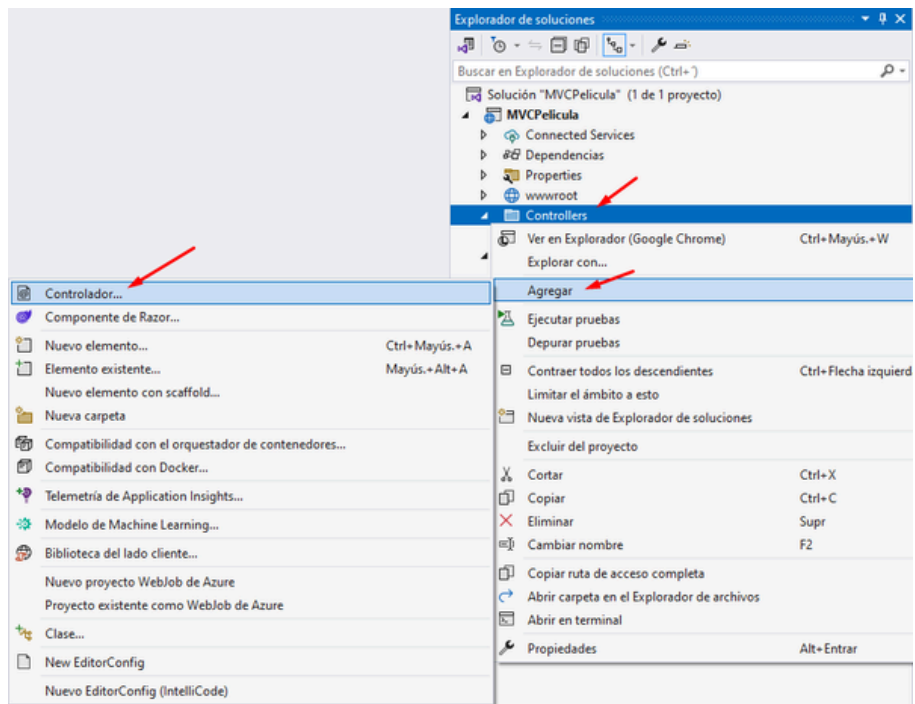


Presionar F5 en Visual Studio para iniciar la depuración, se activa IIS Express, una versión ligera del servidor web de Microsoft, Internet Information Services (IIS). Esto proporciona una experiencia más realista al probar aplicaciones web en comparación con el servidor "Cassini" incluido habitualmente en Visual Studio. IIS Express ejecuta su aplicación web y abre automáticamente una página en un navegador web. La dirección en la barra de direcciones del navegador muestra localhost seguido de un número de puerto, como localhost:port#. Esto indica que la aplicación se está ejecutando en su computadora local. Visual Studio asigna automáticamente un puerto aleatorio para el servidor web cada vez que inicia el proyecto.



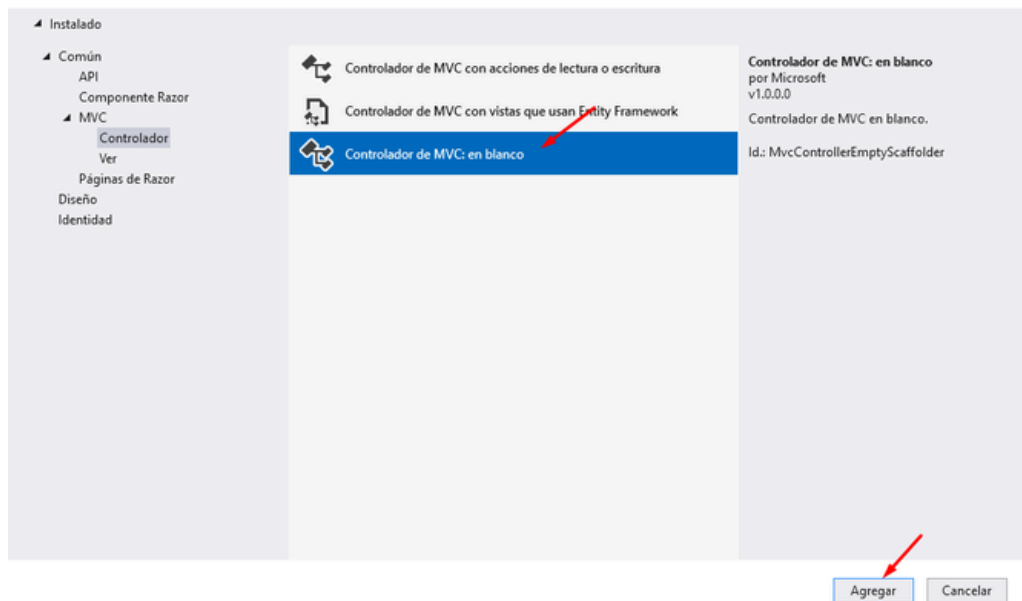
AGREGAR UN CONTROLADOR

Comencemos creando una clase de controlador. En el Explorador de soluciones, haga clic con el botón derecho en la carpeta Controllers y luego haga clic en Agregar, luego en Controlador.

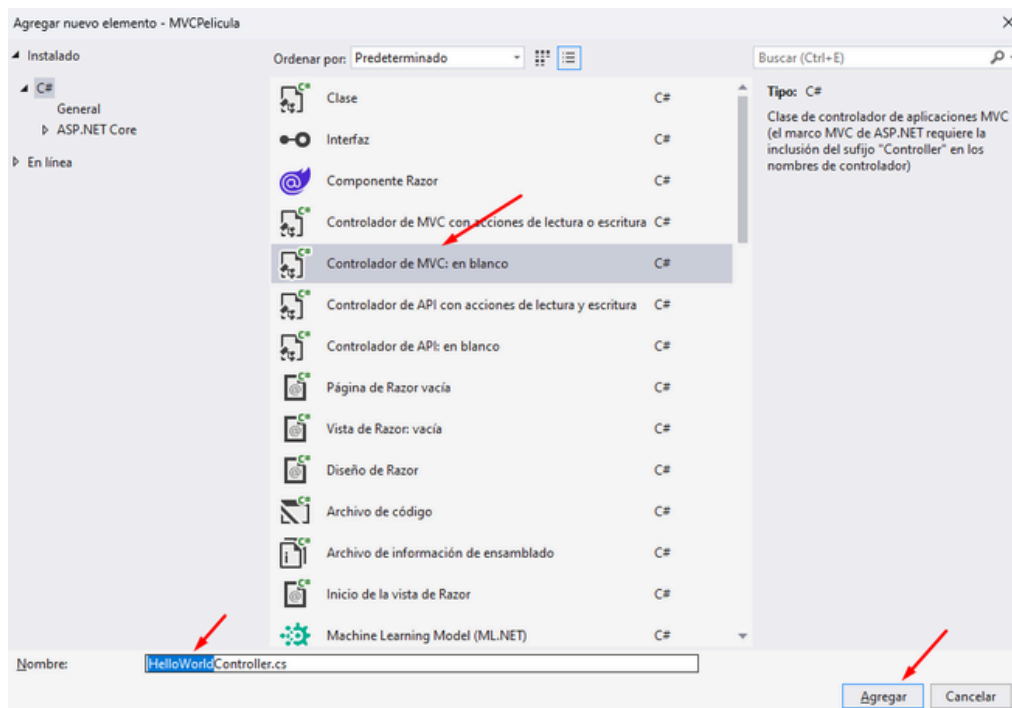


En el cuadro de diálogo **Agregar nuevo elemento con scaffold**, haga clic en **Controlador MVC: en blanco**, y luego haga clic en Agregar.

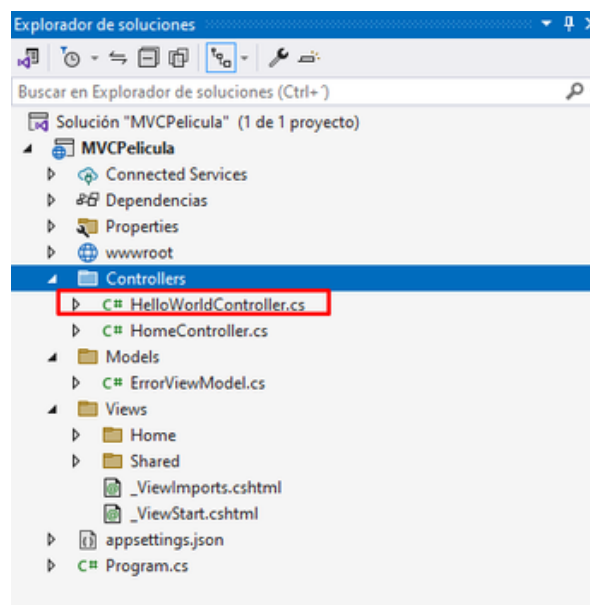
Agregar nuevo elemento con scaffolding



Nombre su nuevo controlador "**HelloWorldController**" y haga clic en **Agregar**.



Observe en el Explorador de soluciones que se ha creado un nuevo archivo llamado **HelloWorldController.cs**



Reemplace el contenido del controlador con el siguiente código.


```

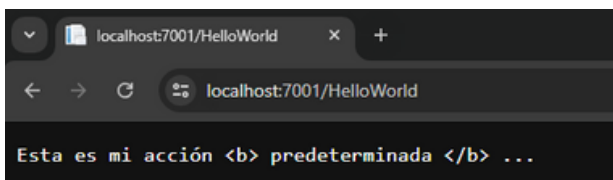
using Microsoft.AspNetCore.Mvc;

namespace MVCPelícula.Controllers
{
    // 0 referencias
    public class HelloWorldController : Controller
    {
        // GET: /HelloWorld/
        // 0 referencias
        public string Index()
        {
            return "Esta es mi acción <b> predeterminada </b> ...";
        }

        // GET: /HelloWorld/Welcome/
        // 0 referencias
        public string Welcome()
        {
            return "Este es el método de acción Bienvenida...";
        }
    }
}

```

Los métodos del controlador devolverán una cadena de HTML como ejemplo. Se nombra el controlador HelloWorldController y se nombra el primer método Index. Invoquemoslo desde un navegador. Ejecute la aplicación (presione F5 o Ctrl + F5). En el navegador, agregue "HelloWorld" a la ruta en la barra de direcciones. La página en el navegador se verá como la siguiente captura de pantalla. En el método anterior, el código devolvió una cadena directamente.



ASP.NET MVC Core invoca diferentes clases de controlador (y diferentes métodos de acción dentro de ellos) dependiendo de la URL entrante. La lógica de enrutamiento de URL predeterminada utilizada por ASP.NET MVC Core, utiliza un formato como este, para determinar qué código invocar:

/[Controller]/[ActionName]/[Parameters]

Estableciendo el formato para el enrutamiento en el archivo Program.cs

```

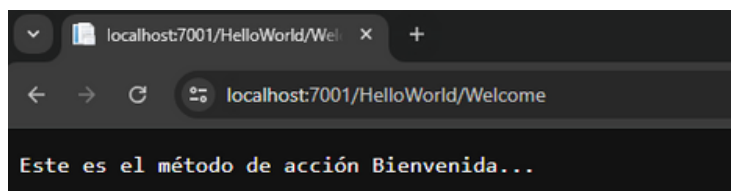
Program.cs
1  var builder = WebApplication.CreateBuilder(args);
2
3  // Add services to the container.
4  builder.Services.AddControllersWithViews();
5
6  var app = builder.Build();
7
8  // Configure the HTTP request pipeline.
9  if (!app.Environment.IsDevelopment())
10 {
11     app.UseExceptionHandler("/Home/Error");
12     // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
13     app.UseHsts();
14 }
15
16 app.UseHttpsRedirection();
17 app.UseStaticFiles();
18
19 app.UseRouting();
20
21 app.UseAuthorization();
22
23 app.MapControllerRoute(
24     name: "default",
25     pattern: "{controller=Home}/{action=Index}/{id?}");
26
27 app.Run();

```

Cuando se ejecuta la aplicación y no se proporciona ningún segmento de URL, el controlador predeterminado es "Inicio" y el método de acción "Index" especificado en la sección de valores predeterminados del código anterior.

La primera parte de la URL, determina la clase de controlador a ejecutar. Entonces /HelloWorld se asigna a la clase HelloWorldController. La segunda parte de la URL determina el método de acción en la clase a ejecutar. Entonces /HelloWorld/Index haría que el método Index de la clase HelloWorldController se ejecute. Tenga en cuenta que solo tuvimos que buscar /HelloWorld y el método Index se usó de forma predeterminada. Esto se debe a que un método llamado Index es el método predeterminado que se invocará en un controlador si no se especifica uno explícitamente. La tercera parte del segmento URL (Parameters) es para datos de ruta. Veremos los datos de la ruta más adelante en esta guía.

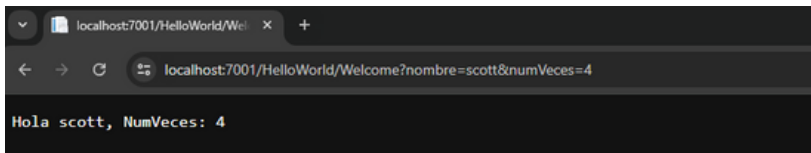
Vaya a <http://localhost:xxxx/HelloWorld/Welcome>. El método Welcome se ejecuta y devuelve la cadena "Este es el método de acción Bienvenido ...". La asignación de MVC predeterminada es /[Controller]/[ActionName]/[Parameters]. Para esta URL, el controlador es HelloWorld y Welcome es el método de acción. Aún no se ha utilizado la parte de la URL para [Parameters].



Modifiquemos ligeramente el ejemplo para que pueda pasar información de parámetros desde la URL al controlador (por ejemplo, /HelloWorld/Welcome?nombre=Juan&numVeces=4). Cambie su método Welcome para incluir dos parámetros como se muestra a continuación. Tenga en cuenta que el código usa la función de parámetro opcional C# para indicar que el parámetro numVeces debe tener un valor predeterminado de 1, si no se pasa ningún valor para ese parámetro.

```
0 referencias
public string Welcome(string nombre, int numVeces = 1)
{
    return HtmlEncoder.Default.Encode($"Hola {nombre}, NumVeces: {numVeces}");
}
```

Ejecute su aplicación y busque la URL de ejemplo (**<http://localhost:xxxxx/HelloWorld/Welcome?nombre=Scott&numveces=4>**). Puede probar diferentes valores para nombre y numVeces en la URL. El sistema de enlace del modelo ASP.NET MVC asigna automáticamente los parámetros con nombre, de la cadena de consulta en la barra de direcciones, a los parámetros de su método.

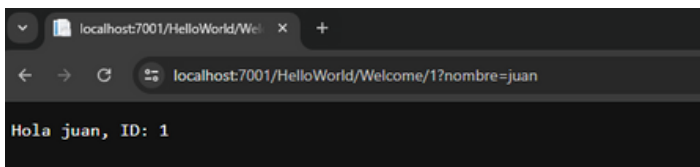


En el ejemplo anterior, el segmento URL (Parameters) no se usa, los parámetros nombre y numVeces se pasan como cadenas de consulta. Los ? (signo de interrogación) en la URL anterior son un separador, y siguen las cadenas de consulta. El carácter & separa las cadenas de consulta.

Reemplace el método de bienvenida con el siguiente código:

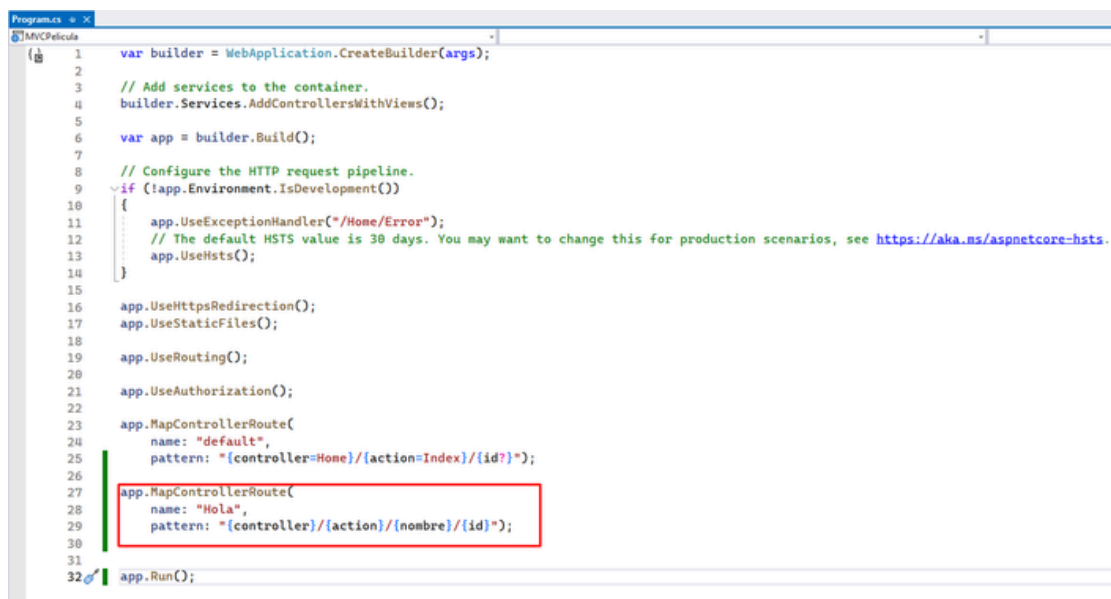
```
0 referencias
public string Welcome(string nombre, int ID = 1)
{
    return HtmlEncoder.Default.Encode($"Hola {nombre}, ID: {ID}");
}
```

Ejecute la aplicación e ingrese la siguiente URL: <http://localhost:xxx/HelloWorld/Welcome/1?name=Juan>

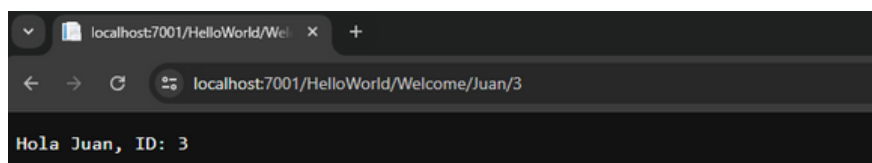


Esta vez, el tercer segmento de URL coincidió con el parámetro ID de la ruta. El método de acción Welcome contiene un parámetro (ID) que coincide con la especificación de URL en el método RegisterRoutes.

En las aplicaciones ASP.NET MVC, es más típico pasar parámetros como datos de ruta (como hicimos con la ID anterior) que pasarlos como cadenas de consulta. También puede agregar una ruta para pasar los parámetros nombre y numVeces en los datos de ruta en la URL. Para esto en el archivo Program.cs, agregue la ruta "Hola":



Ejecute la aplicación y busque localhost:XXXX/HelloWorld/Welcome/Juan/3



Para muchas aplicaciones MVC, la ruta predeterminada funciona bien. Aprenderá más adelante en esta guía a pasar datos utilizando el modelo de carpeta, y no tendrá que modificar la ruta predeterminada para eso.

En estos ejemplos, el controlador ha estado haciendo la parte "VC" de MVC, es decir, la vista y el trabajo del controlador. El controlador está devolviendo HTML directamente. Por lo general, no deseamos que los controladores devuelvan HTML directamente, ya que el código se vuelve muy engorroso. En su lugar, usaremos un archivo de plantilla de vista separado para ayudar a generar la respuesta HTML. Veamos a continuación cómo podemos hacer esto.

AGREGAR UNA VISTA

En esta sección, modificaremos la clase HelloWorldController para usar los archivos de plantilla de vista, para encapsular limpiamente el proceso de generación de respuestas HTML a un cliente.

Crearemos un archivo de plantilla de vista, utilizando el motor de vista Razor. Las plantillas de vista basadas en Razor tienen una extensión de archivo .cshtml y proporcionan una manera elegante de crear resultados HTML usando C#. Razor minimiza la cantidad de caracteres y pulsaciones de teclas necesarias al escribir una plantilla de vista, y permite un flujo de trabajo de codificación rápido y fluido.

Actualmente, el método Index devuelve una cadena con un mensaje codificado en la clase de Controlador. Cambie el método Index para llamar a los métodos controladores View, como se muestra en el siguiente código:

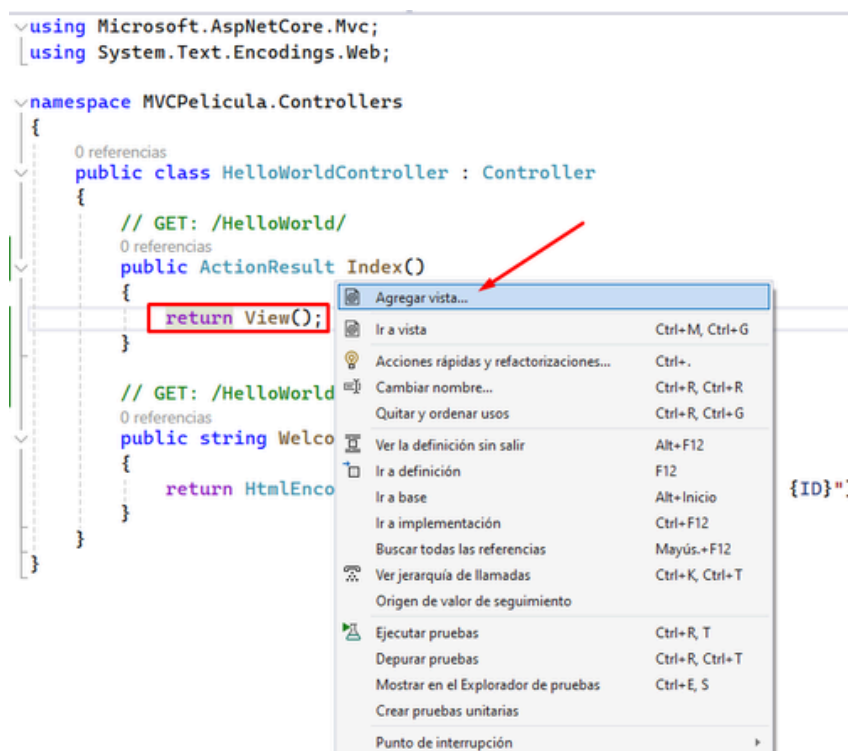
```
using Microsoft.AspNetCore.Mvc;
using System.Text.Encodings.Web;

namespace MVCPelicula.Controllers
{
    0 referencias
    public class HelloWorldController : Controller
    {
        // GET: /HelloWorld/
        0 referencias
        public ActionResult Index()
        {
            return View();
        }

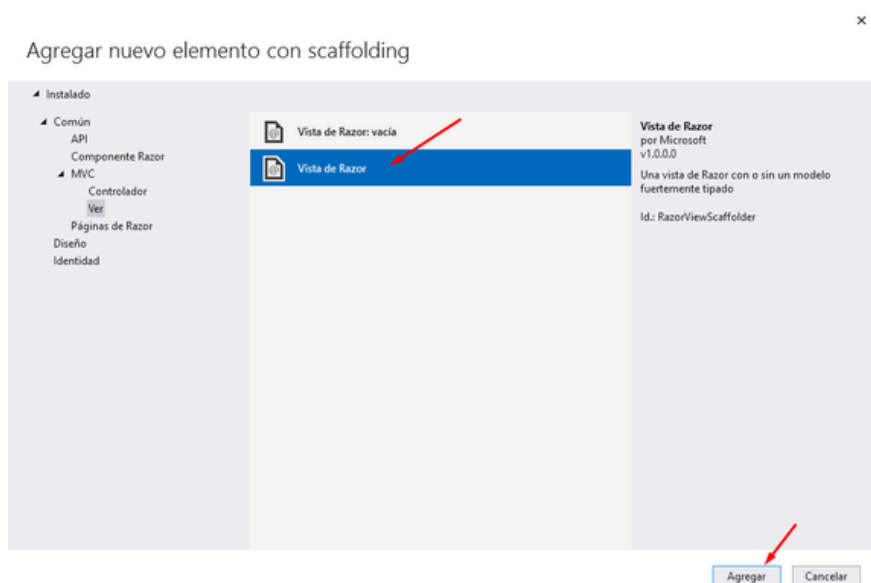
        // GET: /HelloWorld/Welcome/
        0 referencias
        public string Welcome(string nombre, int ID = 1)
        {
            return HtmlEncoder.Default.Encode($"Hola {nombre}, ID: {ID}");
        }
    }
}
```

En el método Index anterior se usa una plantilla de vista para generar una respuesta HTML al navegador. Los métodos de controlador (también conocidos como métodos de acción), como el método Index anterior, generalmente devuelven un ActionResult (o una clase derivada de ActionResult), no tipos primitivos como string.

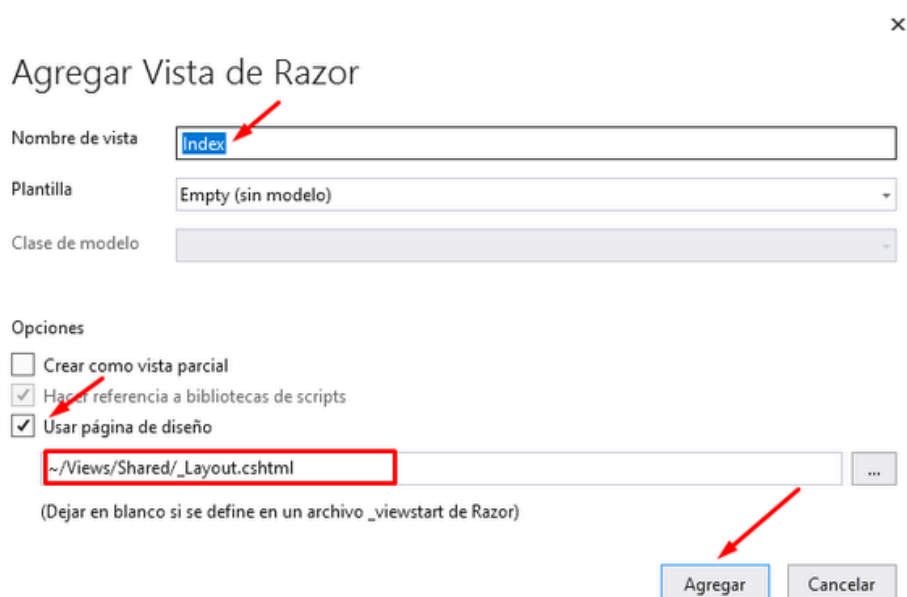
Haga clic con el botón derecho sobre la línea de código "return View();" que esta dentro del método Index, y de clic a la opción **Agregar Vista...**



En el cuadro de diálogo **Agregar nuevo elemento con scaffold**, seleccione **Vista de Razor** y de clic en **Agregar**.



En el cuadro de diálogo Especificar nombre para elemento, para este caso dejé el nombre **Index** que aparece por defecto, Luego marque la opción “**Usar página de diseño**” y en el cuadro de texto bajo a este, ingrese la ruta de la plantilla MVC, la cual se encuentra en el archivo **_Layout.cshtml**. Finalmente, de clic en **Aceptar**.



Agregar Vista de Razor

Nombre de vista:

Plantilla:

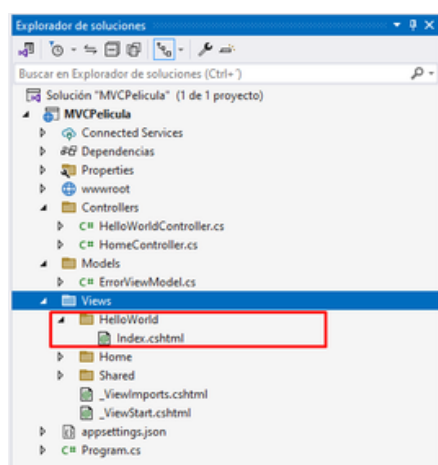
Clase de modelo:

Opciones

- ☐ Crear como vista parcial
- ☒ Hacer referencia a bibliotecas de scripts
- ☒ Usar página de diseño

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

En el explorador de soluciones, expanda la carpeta Views y verá como se ha creado la ruta **HelloWorld/Index.cshtml**

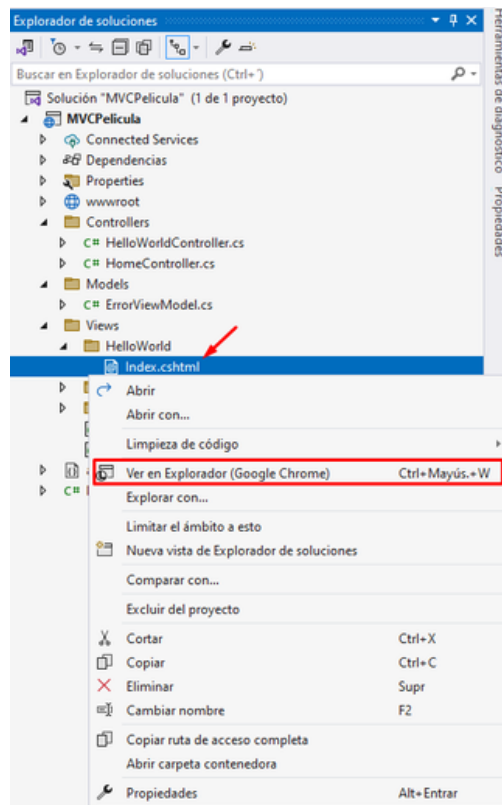


Abra el archivo **Index.cshtml** y agregué el siguiente código:

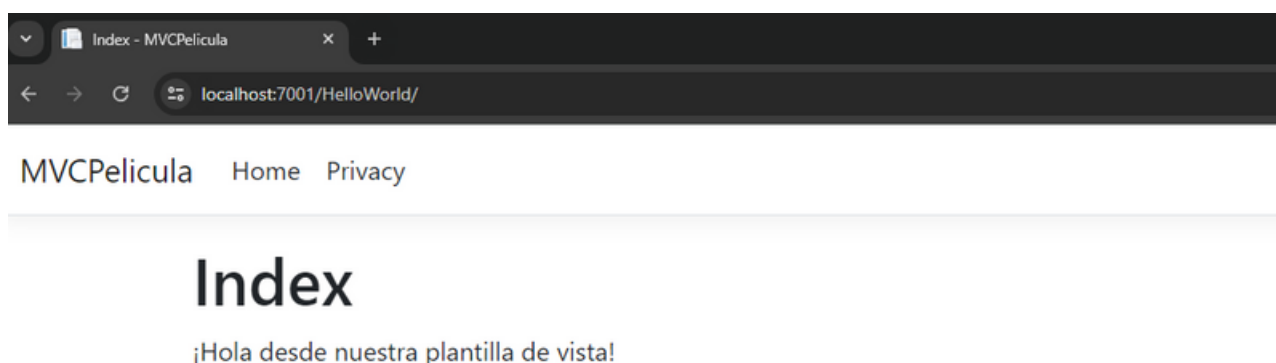


```
1
2  @{
3      ViewData["Title"] = "Index";
4      Layout = "~/Views/Shared/_Layout.cshtml";
5  }
6
7  <h1>Index</h1>
8
9  <p>¡Hola desde nuestra plantilla de vista!</p>
```

Haga clic derecho en el archivo Index.cshtml y seleccione **Ver en el explorador**.



Alternativamente, ejecute la aplicación y busque el controlador HelloWorld (<http://localhost:xxxx/HelloWorld>). El método Index en su controlador no hizo mucho trabajo; simplemente ejecutó la declaración `return View ()`, que especificaba que el método debería usar un archivo de plantilla de vista para responder al navegador. Debido a que no especificó explícitamente el nombre del archivo de plantilla de vista a usar, ASP.NET MVC de manera predeterminada utilizó el archivo de vista Index.cshtml en la carpeta `\Views\HelloWorld`. La siguiente imagen muestra la cadena codificada en la vista "¡Hola desde nuestra plantilla de vista!".



Cambio de vistas y páginas de diseño

Si queremos personalizar el nombre de la aplicación, título de página, o cambiar la plantilla actual por defecto, esto se realiza modificando el archivo `_Layout.cshtml`.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>@ViewData["Title"] - MVPelícula</title>
7   <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
8   <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
9   <link rel="stylesheet" href="~/MVPelícula.styles.css" asp-append-version="true" />
10 </head>
11 <body>
12   <header>
13     <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
14       <div class="container-fluid">
15         <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">MVPelícula</a>
16         <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse"
17           aria-expanded="false" aria-label="Toggle navigation">
18           <span class="navbar-toggler-icon"></span>
19         </button>
20         <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
21           <ul class="navbar-nav flex-grow-1">
22             <li class="nav-item">
23               <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
24             </li>
25             <li class="nav-item">
26               <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
27             </li>
28           </ul>
29         </div>
30       </div>
31     </nav>
32   </header>
33   <div class="container">
34     <main role="main" class="pb-3">
35       @RenderBody()
36     </main>
37   </div>
38   <footer class="border-top footer text-muted">
39     <div class="container">
40       <div class="row">
41         <div class="col">
42           &copy; 2024 - MVPelícula - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
43         </div>
44       </div>
45     </div>
46   </footer>
47   <script src="~/lib/jquery/dist/jquery.min.js"></script>
48   <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
49   <script src="~/js/site.js" asp-append-version="true"></script>
50   @await RenderSectionAsync("Scripts", required: false)
51 </body>
52 </html>
```

Las plantillas de diseño le permiten especificar el diseño del contenedor HTML de su sitio en un lugar y luego aplicarlo en varias páginas de su sitio. Encuentra la línea `@RenderBody()`. `RenderBody` es un marcador de posición donde aparecen todas las páginas específicas de vista que crea "wrapped" en la página de diseño.

Cambie el contenido del elemento del título. Luego cambie el `NavLink` en la plantilla de diseño de **"MVPelícula"** a **"Películas SV"** y el controlador de Home a Películas. El archivo de diseño completo se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - Películas SV</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
  <link rel="stylesheet" href="~/MVPelícula.styles.css" asp-append-version="true" />
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
      <div class="container-fluid">
        <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Películas SV</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse"
          aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
          <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>
  <footer class="border-top footer text-muted">
    <div class="container">
      <div class="row">
        <div class="col">
          &copy; 2024 - MVPelícula - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </div>
      </div>
    </div>
  </footer>
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```



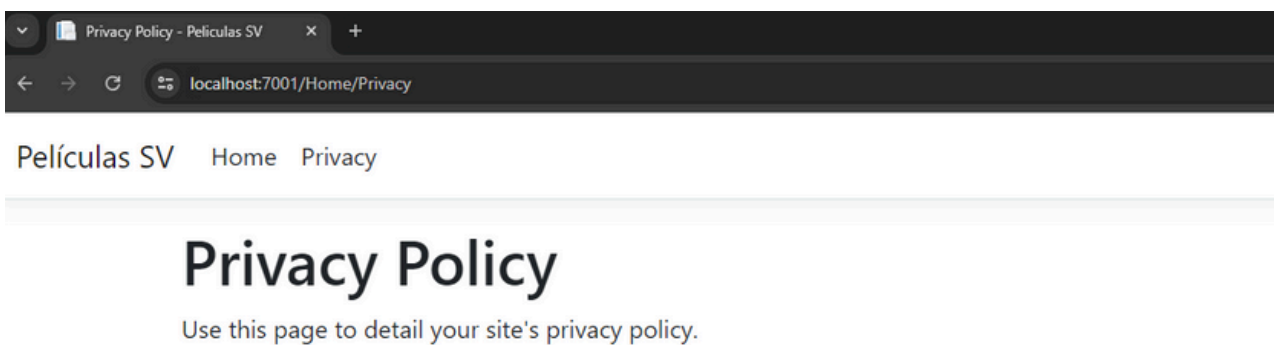
```

        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
      </li>
    </ul>
  </div>
</div>
</nav>
</header>
<div class="container">
  <main role="main" class="pb-3">
    @RenderBody()
  </main>
</div>

<footer class="border-top footer text-muted">
  <div class="container">
    &copy; 2024 - MVCPelícula - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
  </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>

```

Ejecute la aplicación y observe que ahora dice "Películas SV". Haga clic en el enlace **Privacy** de y verá cómo esa página muestra "Películas SV" también. Pudimos hacer el cambio una vez en la plantilla de diseño y hacer que todas las páginas del sitio reflejen el nuevo título.



Cuando creamos por primera vez el archivo Views\HelloWorld\Index.cshtml, contenía el siguiente código:

```

@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

```

El código Razor anterior está configurando explícitamente la página de diseño, tal cual nosotros indicamos al crear la vista.

Examine el archivo Views_ViewStart.cshtml, que contiene exactamente el mismo marcado de Razor. El archivo Views_ViewStart.cshtml define el diseño común que utilizarán todas las vistas, por lo tanto, puede comentar o eliminar ese código del archivo Views\HelloWorld\Index.cshtml.

```
@{
    ViewData["Title"] = "Index";
    //Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Index</h1>

<p>¡Hola desde nuestra plantilla de vista!</p>
```

Puede usar la propiedad Layout para establecer una vista de diseño diferente, o configurarla para que no se use ningún archivo de diseño (null).

Ahora, cambiemos el título de la vista de índice. Abra MvcPelicula\Views\HelloWorld\Index.cshtml. Hay dos lugares para hacer un cambio: primero, el texto que aparece en el título del navegador, y luego en el encabezado secundario (el elemento <h2>). Los hará ligeramente diferentes para que pueda ver qué parte del código cambia qué parte de la aplicación.

```
@{
    ViewData["Title"] = "Lista de Peliculas";
}

<h1>Mi lista de Peliculas</h1>

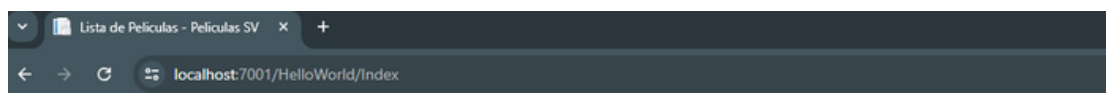
<p>¡Hola desde nuestra plantilla de vista!</p>
```

Para indicar el título HTML para mostrar, el código anterior establece una propiedad Title del objeto ViewData (que se encuentra en la plantilla de vista Index.cshtml). Tenga en cuenta que la plantilla de diseño (Views\Shared_Layout.cshtml) usa este valor en el elemento <title> como parte de la sección <head> del HTML que modificamos anteriormente.

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Peliculas SV</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/MvcPelicula.styles.css" asp-append-version="true" />
</head>
```

Con este enfoque ViewData, puede pasar fácilmente otros parámetros entre su plantilla de vista y su archivo de diseño.

Ejecute la aplicación. Observe que el título del navegador, el encabezado principal y los encabezados secundarios han cambiado. (Si no ve cambios en el navegador, es posible que esté viendo contenido en caché. Presione Ctrl + F5 en su navegador para forzar la carga del servidor). El título del navegador se crea con el ViewData["Title"] que configuramos en la plantilla de la vista index.cshtml y la "Aplicación de Película" adicional agregada en el archivo de diseño. Observe también cómo el contenido de la plantilla de vista Index.cshtml se fusionó con la plantilla de vista _Layout.cshtml y se envió una única respuesta HTML al navegador. Las plantillas de diseño hacen que sea realmente fácil realizar cambios que se apliquen en todas las páginas de su aplicación.



Películas SV Home Privacy

Mi lista de Películas

¡Hola desde nuestra plantilla de vista!

Sin embargo, nuestro pequeño "dato" (en este caso, el mensaje "¡Hola desde nuestra plantilla de vista!") está codificado. La aplicación MVC tiene una "V" (vista) y usted tiene un "C" (controlador), pero todavía no tiene una "M" (modelo). En la siguiente guía, veremos cómo crear una base de datos y recuperar datos del modelo.

Pasar datos del controlador a la vista

Sin embargo, antes de ir a una base de datos y hablar sobre modelos, primero hablemos sobre cómo pasar información del controlador a una vista. Las clases de controlador se invocan en respuesta a una solicitud de URL entrante. Una clase de controlador es donde se escribe el código que maneja las solicitudes entrantes del navegador, recupera los datos de una base de datos y, en última instancia, decide qué tipo de respuesta enviar al navegador. Las plantillas de vista se pueden usar desde un controlador para generar y formatear (dar formato) una respuesta HTML al navegador.

Los controladores son responsables de proporcionar los datos u objetos necesarios para que una plantilla de vista responda al navegador. Una práctica recomendada es que: una plantilla de vista nunca debe realizar la lógica de negocio o interactuar con una base de datos directamente. En cambio, una plantilla de vista debería funcionar solo con los datos que le proporciona el controlador. Mantener esta "separación de responsabilidades" ayuda a mantener su código limpio, comprobable y más fácil de mantener.

Actualmente, el método de acción `Welcome` en la clase `HelloWorldController` toma los parámetros `nombre` y `numVeces` y luego envía los valores directamente al navegador. En lugar de que el controlador presente esta respuesta como una cadena, cambiemos el controlador para que use una plantilla de vista. La plantilla de vista generará una respuesta dinámica, lo que significa que debe pasar los bits de datos apropiados del controlador a la vista para generar la respuesta. Puede hacer esto haciendo que el controlador ponga los datos dinámicos (parámetros) que la plantilla de vista necesita en un objeto `ViewBag` al que la plantilla de vista puede acceder.

Volveremos al archivo `HelloWorldController.cs` y cambiaremos el método `Welcome` para agregar un valor de `Mensaje` y `NumVeces` al objeto `ViewData`. `ViewData` es un objeto dinámico, lo que significa que puede poner lo que quiera en él; el objeto `ViewData` no tiene propiedades definidas hasta que coloque algo dentro de él. El sistema de enlace del modelo ASP.NET MVC asigna automáticamente los parámetros con nombre (`nombre` y `numVeces`) de la cadena de consulta en la barra de direcciones, a los parámetros de su método. El archivo completo `HelloWorldController.cs` tiene este aspecto:

```

using Microsoft.AspNetCore.Mvc;

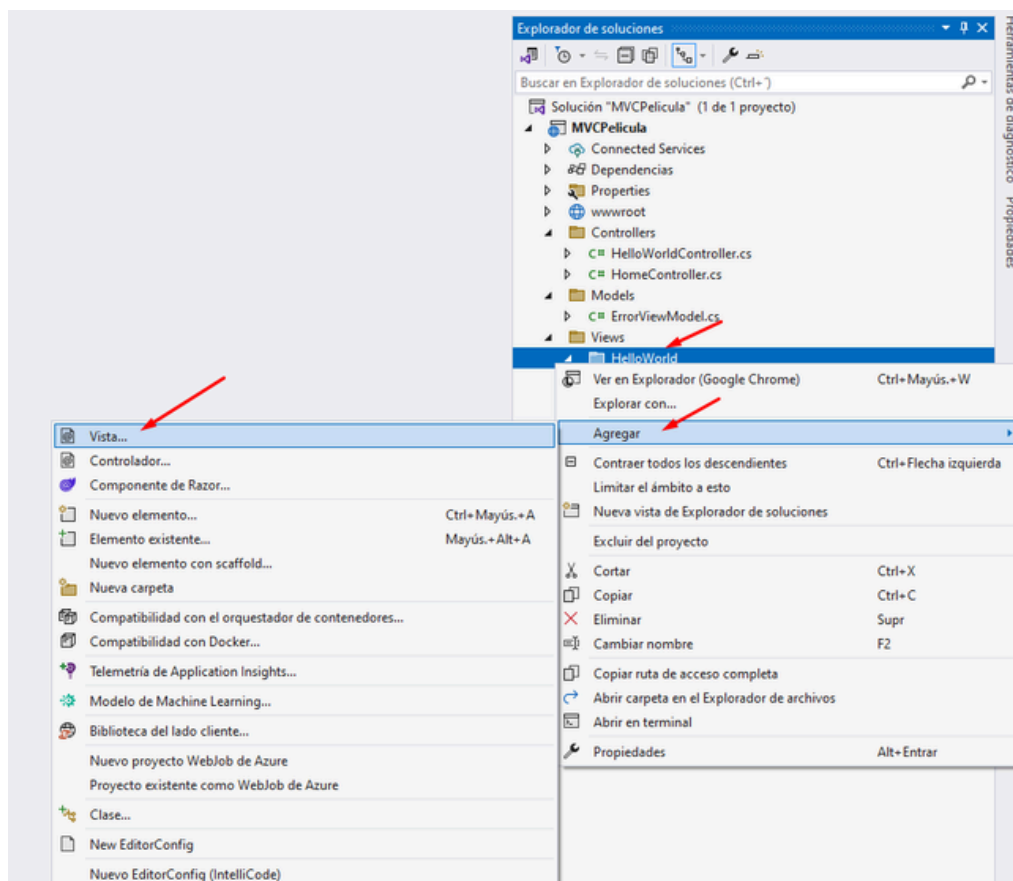
namespace MVCPelicula.Controllers
{
    0 referencias
    public class HelloWorldController : Controller
    {
        // GET: /HelloWorld/
        0 referencias
        public ActionResult Index()
        {
            return View();
        }

        // GET: /HelloWorld/Welcome/
        0 referencias
        public ActionResult Welcome(string nombre, int numVeces = 1)
        {
            ViewData["nombre"] = "Hola " + nombre;
            ViewData["numVeces"] = numVeces;

            return View();
        }
    }
}

```

Ahora el objeto ViewData contiene datos que se pasarán a la vista automáticamente. A continuación, necesitamos una plantilla de vista para el método Welcome. En el menú Compilar, seleccione Compilar Solución (o Ctrl + Shift + B) para asegurarse de que el proyecto se compila. Haga clic con el botón derecho en la carpeta Views\HelloWorld y haga clic en Agregar, luego haga clic en Vista.



En el cuadro de dialogo, seleccionamos la opción **Vista de Razor** (Como ya lo hicimos cuando se creó el Index). Luego exspecificamos el nombre para la vista, ingrese el nombre **Welcome** y haga clic en **Aceptar**.

Agregar Vista de Razor

Nombre de vista:

Plantilla:

Clase de modelo:

Opciones:

- ☐ Crear como vista parcial
- ☐ Hacer referencia a bibliotecas de scripts
- ☒ Usar página de diseño

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

Se crea el archivo MvcPelícula\Views\HelloWorld\Welcome.cshtml. Reemplace el código en el archivo Welcome.cshtml. Esto creará un bucle que diga "Hola" tantas veces como el usuario diga. El archivo completo Welcome.cshtml se muestra a continuación.

```
@{
    ViewData["Title"] = "Bienvenido";
}

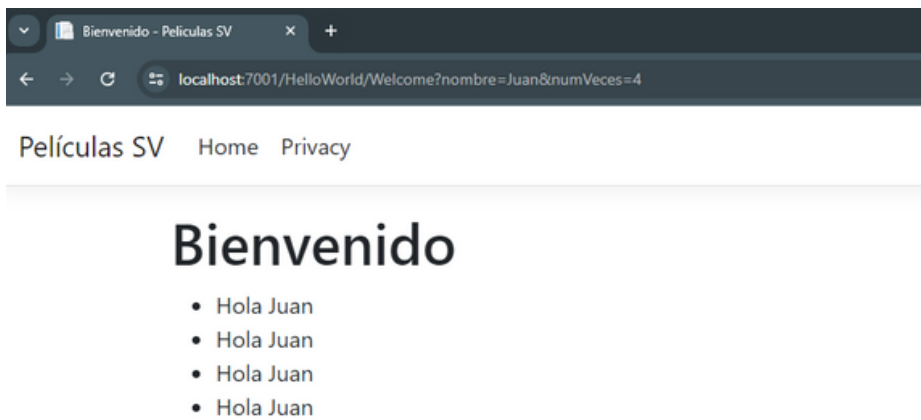
<h1>Bienvenido</h1>

<ul>
    @{
        int numVeces = (int)ViewData["numVeces"];
        for (int i = 0; i < numVeces; i++)
        {
            <li>@ViewData["nombre"]</li>
        }
    }
</ul>
```

Ejecute la aplicación y busque la siguiente URL:

<http://localhost:xxxxx/HelloWorld/Welcome?nombre=Juan&numVeces=4>

Ahora los datos se toman de la URL y se pasan al controlador utilizando el modelo de carpeta. El controlador empaqueta los datos en un objeto ViewData y pasa ese objeto a la vista. La vista luego muestra los datos como HTML para el usuario.



En el ejemplo anterior, usamos un objeto ViewData para pasar datos del controlador a una vista. Más adelante en la segunda parte, utilizaremos un modelo de vista para pasar datos de un controlador a una vista. El enfoque del modelo de vista para pasar datos, es generalmente preferido sobre el enfoque de la bolsa de vista.

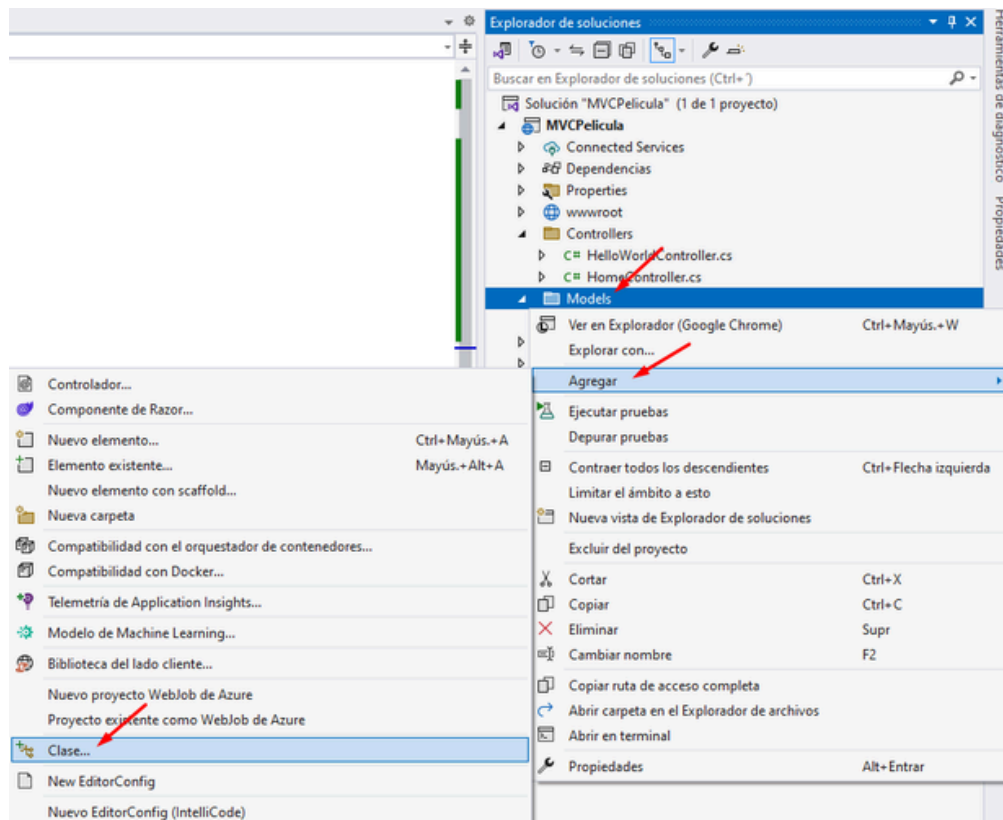
Lo anterior era una especie de "M" para el modelo, pero no del tipo de base de datos. Tomemos lo que hemos aprendido y creemos una base de datos de películas.

AGREGAR UN MODELO

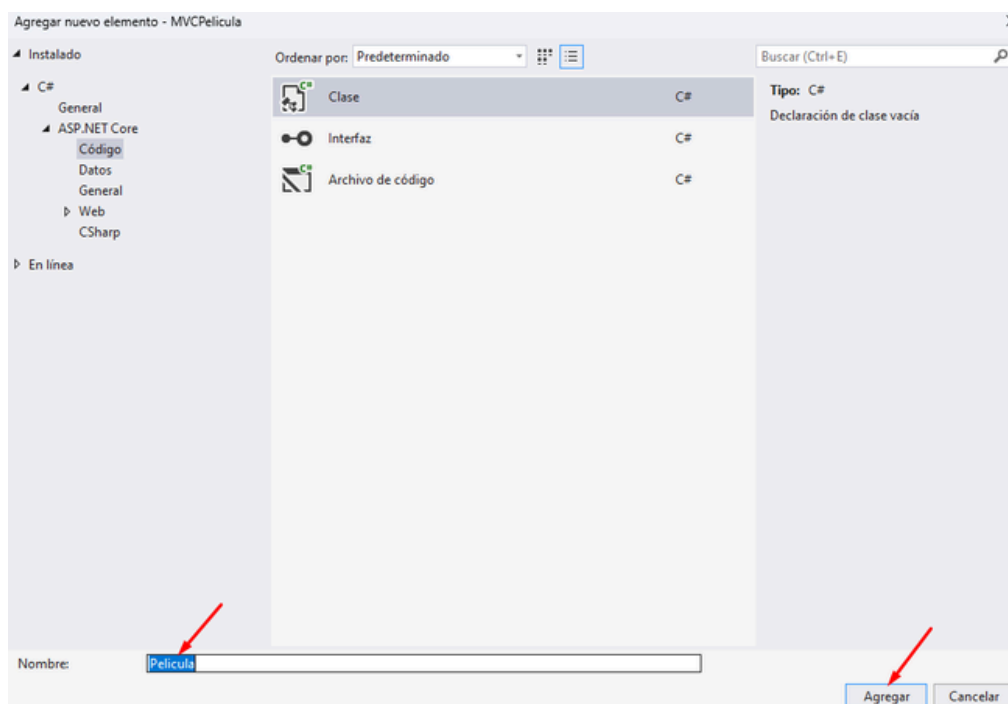
En esta sección, agregaré algunas clases para administrar películas en una base de datos. Estas clases serán la parte "Modelo" de la aplicación ASP.NET MVC. Utilizaremos una tecnología de acceso a datos de .NET conocida como Entity Framework, para, definir y trabajar con estas clases de modelos. El Entity Framework (a menudo denominado EF) admite un paradigma de desarrollo llamado Code First. Code First le permite crear modelos escribiendo clases simples. (También se conocen como clases POCO, de "objetos CLR simples"). Luego, puede crear la base de datos sobre la marcha desde sus clases a través de migraciones, lo que permite un flujo de trabajo de desarrollo muy limpio y rápido.

Agregar clases de modelos

En el Explorador de soluciones, haga clic con el botón derecho en la carpeta Models, seleccione Agregar y luego seleccione Clase.



Ingrese el nombre de la clase "**Película**" y presione el botón agregar.



Agregue las siguientes cinco propiedades a la clase Película:

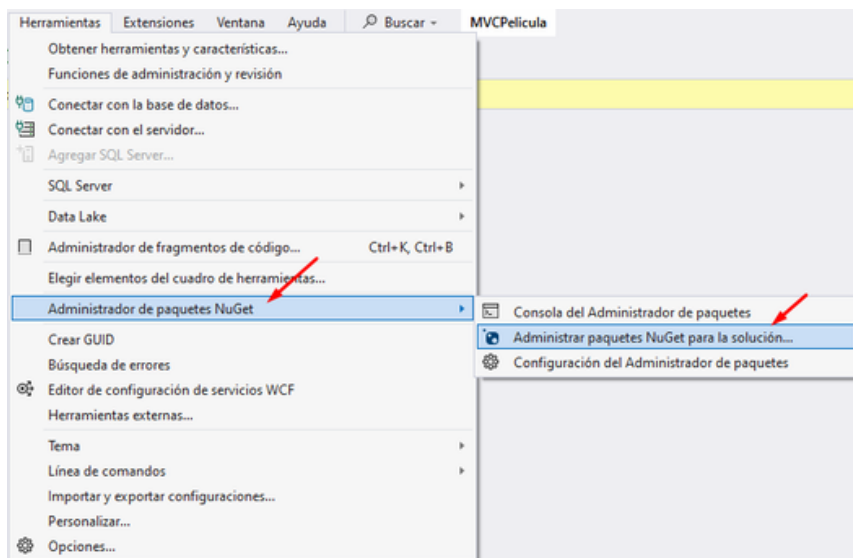
```

namespace MVCPelicula.Models
{
    0 referencias
    public class Pelicula
    {
        0 referencias
        public int ID { get; set; }
        0 referencias
        public string Titulo { get; set; }
        0 referencias
        public DateTime FechaLanzamiento { get; set; }
        0 referencias
        public string Genero { get; set; }
        0 referencias
        public decimal Precio { get; set; }
    }
}

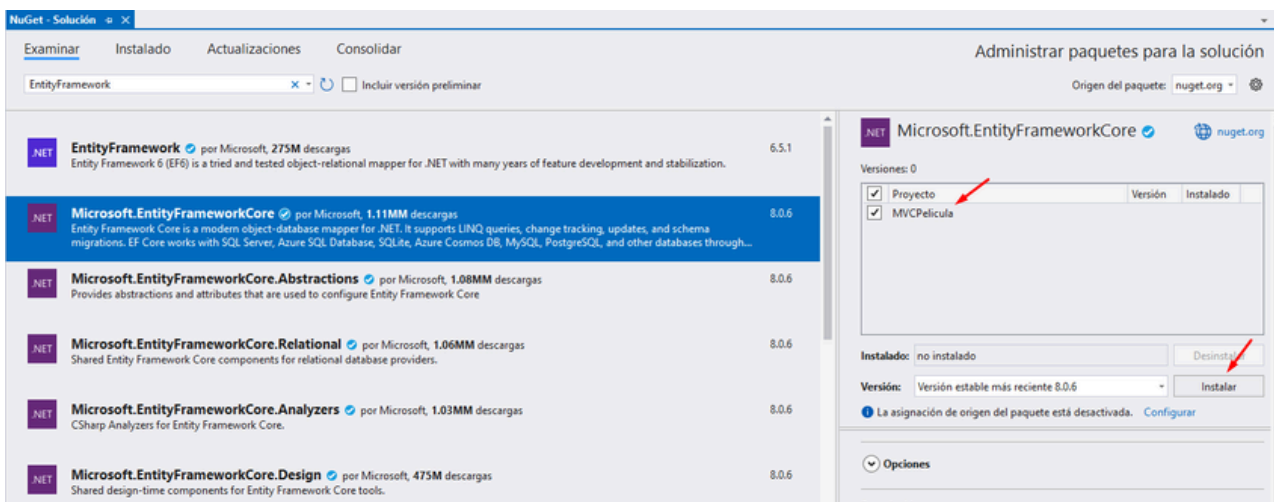
```

Usaremos la clase Película para representar películas en una base de datos. Cada instancia de un objeto Película corresponderá a una fila dentro de una tabla de base de datos, y cada propiedad de la clase Película se asignará a una columna en la tabla.

Nota: Para usar System.Data.Entity y la clase relacionada, debe instalar el paquete NuGet de Entity Framework. Para ello de clic en Herramientas -> Administrador de paquetes NuGet -> Administra paquetes NuGet para la solución...



En la pestalla Explorar busque “EntityFrameworkCore”, una vez lo encuentre, seleccíonelo y de clic en Instalar.



Cree una nueva clase llamada **PeliculasDBContext.cs** dentro de la carpeta **Models** y agregue el siguiente código:

```
using Microsoft.EntityFrameworkCore;

namespace MVCPelícula.Models
{
    1 referencia
    public class PeliculasDBContext: DbContext
    {
        0 referencias
        public PeliculasDBContext(DbContextOptions options) : base(options)
        {
        }

        0 referencias
        public DbSet<Película> Peliculas { get; set; }
    }
}
```

La clase **PeliculasDBContext** representa el contexto de la base de datos de películas de Entity Framework, que maneja la obtención, el almacenamiento y la actualización de instancias de clase Película en una base de datos. PeliculasDBContext deriva de la clase DbContext base proporcionada por el marco de la entidad.

Finalmente hemos agregado un modelo (la M en MVC). En la siguiente guía de laboratorio, trabajaremos con la cadena de conexión de la base de datos.

Desarrollo de Habilidades

Ejercicio No. 1

Modifique la vista HelloWorld/Welcome y el archivo Program.cs para que además de recibir los parámetros de Nombre y NumVeces en la URL reciba el apellido y muestre en la vista el contador de las veces que se ha mostrado el nombre. Ejemplo:

Películas SV Home Privacy

Bienvenido

- Hola Juan Perez, veces mostrado = 1
- Hola Juan Perez, veces mostrado = 2
- Hola Juan Perez, veces mostrado = 3
- Hola Juan Perez, veces mostrado = 4
- Hola Juan Perez, veces mostrado = 5

Ejercicio No. 2

Cree un nuevo Proyecto con el nombre Persona y siga los pasos necesarios para agregar el modelo, los atributos que debe tener la clase persona son DUI, Nombre, Apellido, Fecha de nacimiento, Dirección y correo electrónico, tenga en cuenta que el desarrollo de este ejercicio será necesario para realizar los ejercicios de guías posteriores.