UD 2. ASP.NET Core. Características. El lenguaje C#.

A lo largo del primer trimestre hemos utilizado diferentes modos de crear nuestro proyecto para empezar a trabajar:

- Cuando no conocíamos los elementos de una aplicación ASP.NET MVC comenzamos utilizando la plantilla que proporcionaba un proyecto base MVC, y eliminamos los elementos innecesarios.

- A continuación, para introducir los elementos esenciales de una aplicación ASP.NET MVC, pasamos a utilizar la plantilla de proyecto vacío, añadiendo los elementos necesarios por nosotros mismos.

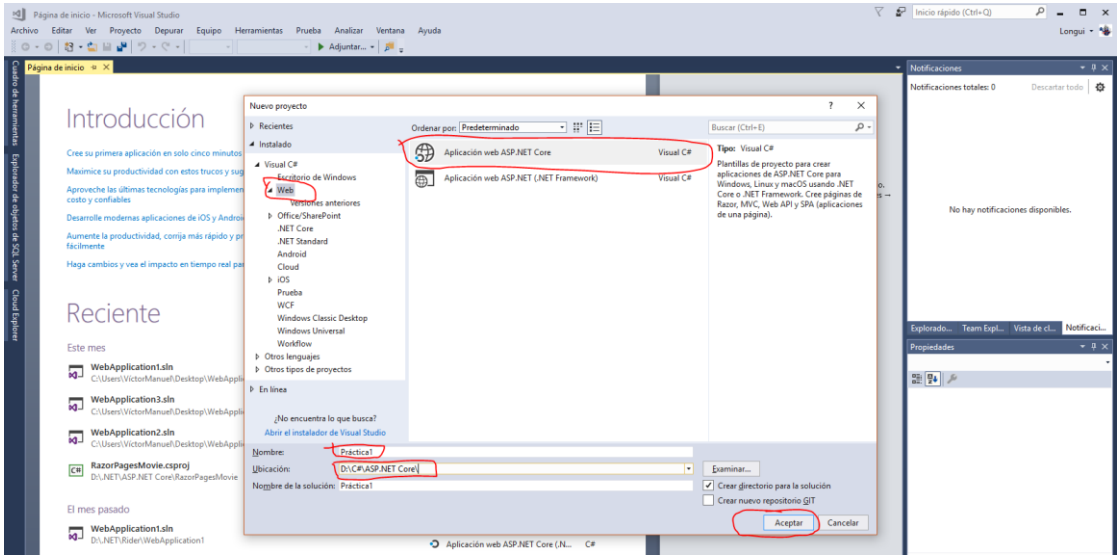
En el punto en que nos encontramos hemos aprendido cómo es la estructura de un proyecto y la multitud de convenciones sobre las que descansa. Además hemos ido poco a poco adquiriendo más conocimientos y ahora somos capaces de realizar más tareas y emplear más elementos, como soporte para elementos estáticos, una plantilla para nuestras vistas, vistas de inicio e importación, vistas compartidas y ViewComponents, modelos y ViewModels, formularios y validación, etc..

Todo esto hace, que a diferencia de nuestros primeros ejercicios, muy simples, y que requerían de poco más que añadir soporte a MVC en nuestro proyecto, en los ejercicios que haremos a partir de ahora, añadir todos los elementos necesarios a una plantilla vacía puede requerir un esfuerzo importante. Por lo tanto, la recomendación a partir de ahora y hasta el final del curso, es partir de una plantilla completa, de proyecto MVC, que nos proporciona todos los elementos necesarios, siendo menor el inconveniente de eliminar lo que no nos hace falta, ahora que somos capaces de reconocer qué es cada cosa.

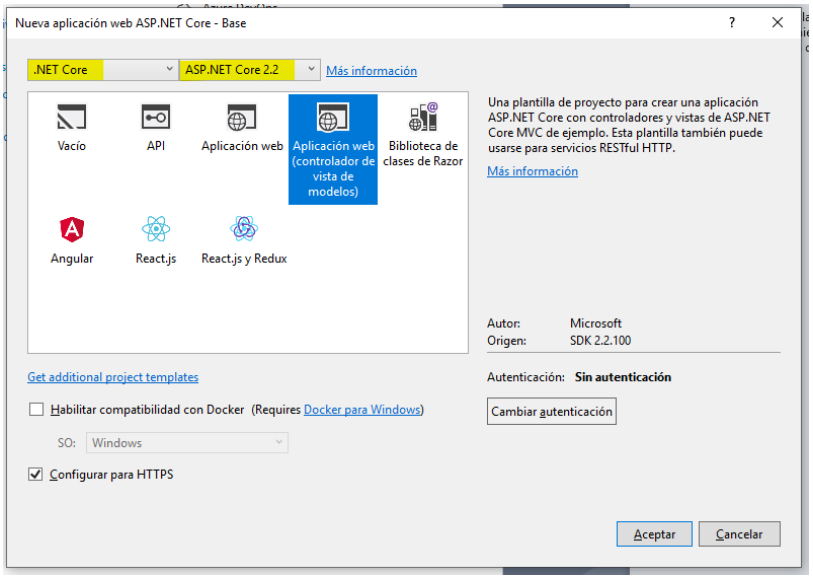
En este documento haremos una nueva revisión sobre los elementos que contiene cada plantilla, que nos servirá de base para este momento y unidades posteriores.

Creación de proyecto de trabajo

Para la creación de un proyecto base sobre el que desarrollar nuestra aplicación comenzaremos creando un nuevo proyecto de Aplicación web ASP.NET Core.

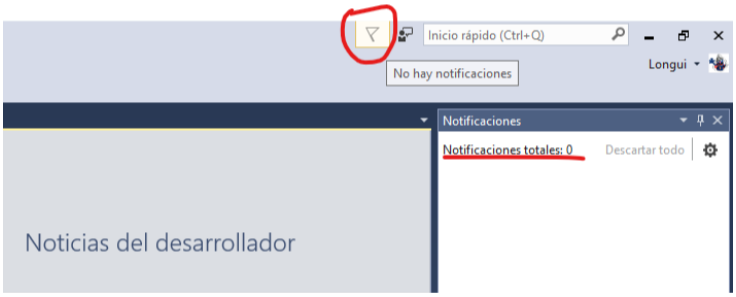


A continuación seleccionaremos la plantilla **Aplicación web (controlador de vista de modelos)**, y que está seleccionado tanto la plataforma .NET Core como la última versión de ASP.NET Core, que es la 2.2 en este momento. Es recomendable actualizar tanto la plataforma .NET Core a su última versión como Visual Studio, ya que desde la aparición de ASP.NET Core 2.2, el IDE ha recibido varias actualizaciones, estando actualmente en la versión 15.9.4, y ha modificado la plantilla de forma importante. Este documento ha sido realizado con estas nuevas actualizaciones, por lo que habrá diferencias con proyectos anteriores.



Para actualizar .NET Core, descargaremos e instalaremos el ejecutable desde:
https://dotnet.microsoft.com/download

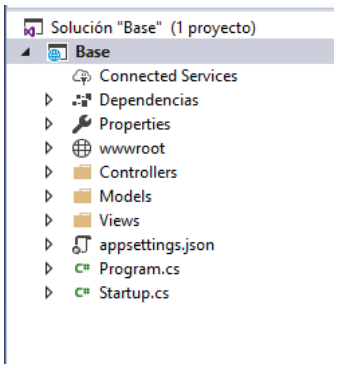
Para actualizar Visual Studio, cada vez que hay una actualización disponible, se iluminará el indicador de notificaciones, en la parte superior, y al pulsarlo se mostrarán las notificaciones de actualización disponibles, permitiendo descargar y aplicar la actualización.



Revisión de los elementos del proyecto

A continuación, en lugar de describir que vamos a eliminar o mantener, se revisarán los elementos contenidos en el proyecto, enumerando de forma breve los que ya conocemos y comentando los que no, para que podamos tener la capacidad de decidir cuáles de estos elementos nos interesan y de cuáles vamos a prescindir.

La estructura de nuestro proyecto será la que ya conocemos:



La primera ventaja que nos ofrece esta plantilla es que ya contiene todas las carpetas necesarias para los elementos más importantes, siguiendo las convenciones de nombres, incluyendo la carpeta de controladores, de modelos, de vistas y el contenido estático. Realmente las carpetas por si solas no son una gran ventaja, pero veremos como contienen gran cantidad de elementos que no tendremos que añadir nosotros mismos.

Startup.cs.

En esta clase es donde se configuran elementos esenciales para nuestra aplicación. Esta tarea se realiza en dos métodos importantes que nos permiten añadir determinados servicios y configurar éstos, junto con otros aspectos de la aplicación.

Comenzamos por *ConfigureServices*:

```
// This method gets called by the runtime. Use this method to add services to the container.
0 referencias | 0 excepciones
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });


    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

La primera parte configura la política de cookies de nuestra aplicación, especificando aspectos tales como si se requerirá la autorización por parte del usuario. Este elemento no es necesario para nuestros ejercicios, pero si que es recomendable para una aplicación real, de modo que podemos mantenerlo o no, una vez que conocemos su utilidad. En la documentación de ASP.NET podemos obtener más información al respecto.

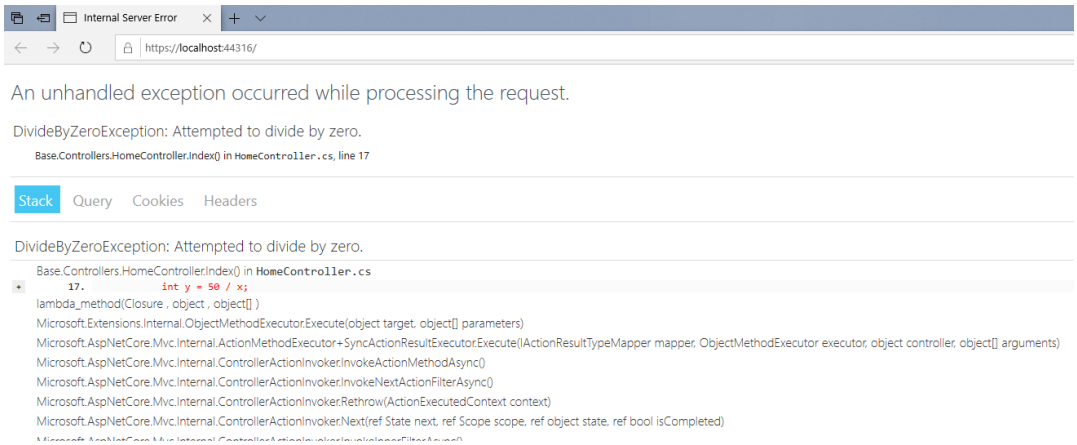
A continuación pasamos a analizar el método Configure, donde tienen lugar gran parte de las tareas de configuración inicial de nuestra aplicación.

En este caso vamos a comenzar por el final, donde indicamos que nuestra aplicación utilizará el servicio MVC, y definimos el mapa de enrutado tal y como se indicó en la unidad 3. Por defecto la estructura siempre será *(controlador)/(acción)/(id?)* pero tanto la estructura, como los valores por defecto pueden ser modificados según nuestras necesidades. Por defecto, es decir, cuando no se especifica un

valor en la URL, el controlador sería *Home*, el método de acción *Index*, y el valor del parámetro id no es obligatorio, por lo que será nulo si no se indica.

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 referencias | 0 excepciones
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios,
        // see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

La primera parte del método configura la respuesta de nuestra aplicación en caso de error. Si la aplicación se está ejecutando en desarrollo, es decir, cuando estamos trabajando en la creación de la misma desde Visual Studio, y la ejecutamos para probarla, y se produce un error, se utilizará la página de excepciones para desarrollador. Esta página es muy útil ya que nos ofrece información detallada del error que se ha producido, y esto ayuda para la depuración y detección del error. A continuación se muestra un ejemplo de página de error:



Podemos ver que el error se produce en el controlador *HomeController*, concretamente en la línea 17, y se trata de una división por 0. Esta información es muy útil para un desarrollador, ya que permite localizar fácilmente los errores, pero JAMÁS debe mostrarse dicha información en una aplicación en explotación. Uno de los mayores errores en seguridad de una aplicación web tiene que ver con las páginas de error, ya que muestran a un posible atacante gran información sobre la plataforma en la que se está ejecutando la aplicación y el modo en que está desarrollada, haciéndola muy insegura.

La práctica correcta consiste en que en explotación, las aplicaciones web deben mostrar una página de error diferente, habitualmente una página común, que siga la estética del resto de la aplicación, y donde no se muestren detalles comprometidos sobre nuestra aplicación.

En nuestro caso se indica que cuando se produzca cualquier error en nuestra aplicación, el encargado de manejarlo será el método *Error* del controlador *Home*. De nuevo se trata de una característica que no es necesaria para nuestros ejercicios pero si recomendable para cualquier aplicación.

La parte central del método especifica, por este orden, que se fuercen todas las peticiones a https por motivos de seguridad, que se habilite el uso de contenido estático y que nuestra aplicación haga uso de la política de cookies especificada en el método *ConfigureServices*.

Controllers.

En la carpeta *Controllers* contamos con un controlador Home, que maneja 3 posibles acciones, la acción por defecto *Index* y la acción *Privacy*, que se limitan a mostrar las vistas correspondientes. El método Index lo mantendremos y lo reutilizamos para nuestra aplicación. *Privacy* muestra una vista que contiene la política de privacidad de nuestra web. En absolutamente innecesario para nuestros ejercicios pero podría resultarnos útil para una aplicación.

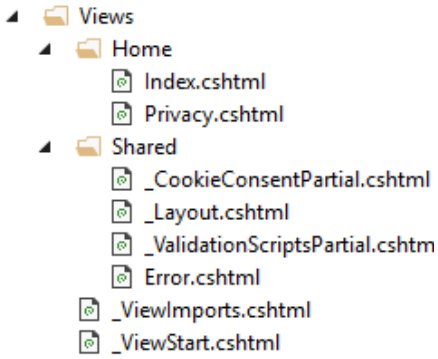
```
namespace Base.Controllers
{
    0 referencias
    public class HomeController : Controller
    {
        0 referencias | 0 solicitudes | 0 excepciones
        public IActionResult Index()
        {
            return View();
        }

        0 referencias | 0 solicitudes | 0 excepciones
        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        0 referencias | 0 solicitudes | 0 excepciones
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

Models.

En esta carpeta encontraremos un único elemento, un ViewModel utilizado en la vista de error de la aplicación. Si lo mantenemos o no dependerá, obviamente, de que hayamos decidido mantener el código que especifica la utilización de la página de error en *Startup.cs*.

Views.

En mi opinión, simplemente por esta carpeta ya merece la pena partir de esta plantilla en lugar de la plantilla vacía, ya que no sólo tenemos creada la estructura de carpetas de las que cuelgan las diferentes vistas, si no que además especifica una plantilla para las vistas de nuestro proyecto, además de contener una serie de vistas compartidas importantes que de lo contrario tendremos que añadir manualmente.



En la carpeta Home encontramos las vistas referentes al controlador del mismo nombre. Estas páginas no tienen gran contenido, y podremos modificarlas fácilmente para que muestren lo que necesitamos.

En la carpeta raíz encontramos dos elementos importantes que ya conocemos, *_ViewImports.cshtml* y *_ViewStart.cshtml*. El primero contiene por defecto sentencias de importación del namespace del proyecto base y de la carpeta Models, ya que lo normal será que las páginas utilicen objetos del modelo, por lo que al importar este namespace aquí, no tendremos que hacerlo en cada página. además contiene también la importación necesaria para utilizar TagHelpers en nuestras vistas.

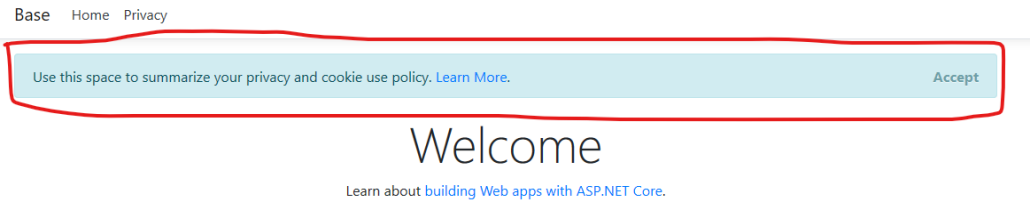
```
@using Base
@using Base.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

En el segundo se especifica que todas nuestras vistas utilizarán por defecto como plantilla la vista *_Layout.cshtml*, evitando también que tengamos que especificarlo en cada página.

Por último, la carpeta Shared, o de vistas compartidas por toda la aplicación, es la que más elementos contiene, y de más complejidad, así que los veremos uno a uno.

La vista parcial *_CookieConsentPartial.cshtml* trabaja conjuntamente con la política de cookies que vimos anteriormente. Esta vista contiene el texto que se mostrará al usuario indicando que nuestra web utiliza cookies y que debe autorizarlo, y el código necesario para ello. Debe ser personalizada con nuestro propio mensaje. Si decidimos no utilizar la política de cookies en nuestra aplicación podremos eliminar esta vista parcial, y acordarnos de quitar de la plantilla _Layout la línea que la muestra.

En las siguientes imágenes podemos ver esta vista parcial en funcionamiento, simplemente muestra un mensaje la primera vez que se ejecuta, y si aceptamos la política nuestro navegador aceptará el uso de cookies, no volviéndose a mostrar:



_ValidationScriptsPartial.cshtml es una vista parcial muy interesante, cuyo uso ya hemos visto en la documentación relativa a la validación de formularios. Contiene la referencia a los scripts necesarios para el funcionamiento de la validación de datos del lado servidor, concretamente la librería jQuery Validation.

```
<environment include="Development">
    <script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>
</environment>
<environment exclude="Development">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.17.0/jquery.validate.min.js"
            asp-fallback-src="~/lib/jquery-validation/dist/jquery.validate.min.js"
            asp-fallback-test="window.jQuery && window.jQuery.validator"
            crossorigin="anonymous"
            integrity="sha256-F6h55Qw6sweK+t7SiOJX+2bpSAa3b/fnlrVCJvmEj1A=">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-unobtrusive/3.2.11/jquery.validate.unobtrusive.min.js"
            asp-fallback-src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"
            asp-fallback-test="window.jQuery && window.jQuery.validator && window.jQuery.validator.unobtrusive"
            crossorigin="anonymous"
            integrity="sha256-9GycpJnliUjJDVDqP0UEu/bsm9U+3dnQUH8+3W10vkY=">
    </script>
</environment>
```

De nuevo, vemos cómo se diferencia entre si estamos trabajando en desarrollo, o se trata de una aplicación en explotación. En el primer caso, vemos como los scripts se cargan desde una carpeta local lib de nuestra aplicación. Esto se hace por motivos de eficiencia, ya que no hay que descargar los scripts, además de que podemos estar trabajando offline, y no podríamos probar nuestra aplicación sin una conexión. De este modo, el proyecto incluye las librerías para que podamos trabajar en modo desconectado, mientras que si la aplicación está en explotación en un servidor, intentará en primer lugar descargarlos del servidor cdn, y si no está disponible utilizará la versión guardada localmente.

El motivo de contener esta importación de scripts en una vista parcial aparte ya se explicó, y no es otro que ofrecer un mecanismo que nos permita que sólo utilicemos estos scripts en las vistas necesarias, donde necesitemos el uso de validación, ya que de lo contrario estaríamos forzando la carga de scripts innecesarios en otras vistas, haciendo la aplicación más ineficiente y pesada.

La vista *Error.cshtml,* si bien se muestra desde una acción del *HomeController*, se ubica en *Shared* ya que se trata de una vista que puede ser llamada desde cualquier lugar de la aplicación, en el momento en que se produce un error. Podemos personalizarla a nuestro gusto, e incluso podemos eliminarla en caso de optar por no incluir la característica de página de error.

Por último, nos encontramos con el fichero más complejo e importante, la plantilla general de la aplicación o *_Layout.cshtml*. Vamos a analizarla en detalle por que su comprensión es fundamental.

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Base</title>

    <environment include="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.3/css/bootstrap.min.css"
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute"
              crossorigin="anonymous"
              integrity="sha256-eSi1q2PG6J7g7ib17yAaWMcrr5GrtohYChqibrV7PBE="/>
    </environment>
    <link rel="stylesheet" href="~/css/site.css" />
</head>
```

Comenzamos por la cabecera, donde ya podemos ver algunos elementos interesantes. En la etiqueta <meta> especificamos la escala y el ancho de la página, así como el título general de nuestras vistas, que podemos personalizar. Tanto la plantilla de nuestra aplicación como diversas páginas de datos que más adelante veremos como Visual Studio puede generar por nosotros, utilizan la librería Bootstrap, ya que esta librería está integrada en los proyectos ASP.NET. Desde la ultima versión de Visual Studio se utiliza en concreto Bootstrap 4. Es importante por lo tanto conservar la importación de esta librería. Vemos de nuevo como en un entorno de desarrollo cargaremos la versión local de la hojas de estilo de Bootstrap mientras que en explotación intentaremos cargarla desde el servidor cdn. Por último cargamos también la hoja de estilos de nuestra aplicación, que podemos personalizar para cambiar la apariencia de la misma.

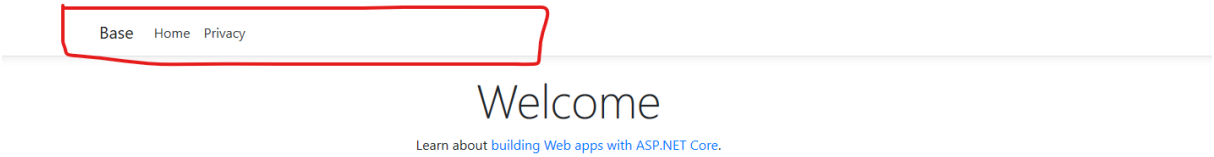
A continuación, analizamos por partes el cuerpo de nuestra página, vemos que está dividido en tres partes fundamentales, una cabecera, una parte principal y un pie.

```
<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Base</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="
                    aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                <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>
```

A continuación analizamos por partes el cuerpo de nuestra página, vemos que está dividido en tres partes fundamentales, una cabecera, una parte principal y un pie.

A continuación analizamos por partes el cuerpo de nuestra página, vemos que está dividido en tres partes fundamentales, una cabecera, una parte principal y un pie.

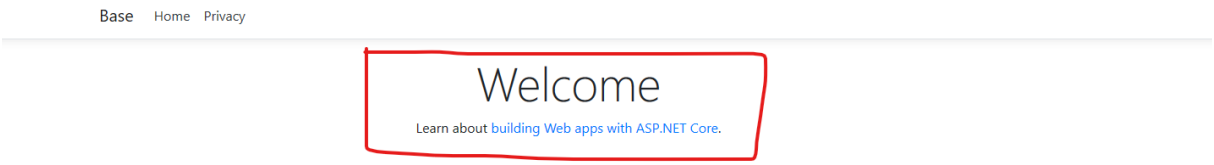
La cabecera muestra una barra de navegación o menú con enlaces a diversas partes de nuestra aplicación, en concreto a la vista inicial por defecto, a la vista Home y a Privacy. Podemos personalizar este menú para contener enlaces a las partes de la aplicación que deseemos, modificar su aspecto, o para los ejercicios sencillos, podemos prescindir completamente de este menú o de la cabecera completa.



La parte central o principal es en principio la más sencilla, aunque también la más importante. Contiene únicamente dos elementos: la vista parcial de consentimiento de política de cookies, en caso de que decidamos mantenerla y la etiqueta <main>, donde "inyectaremos" cada una de nuestras vistas particulares, tal y como se explicó en la unidad 3, en el apartado referente a vistas y la plantilla Layout.

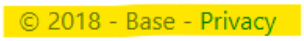
```
<div class="container">
    <partial name="_CookieConsentPartial" />
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>
```

En la imagen siguiente podemos ver como se ha incluido el contenido de la vista Home dentro de la plantilla.



A continuación nos encontramos con un pie con información general de la aplicación, que también podremos decidir entre personalizar o eliminar:

```
<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2018 - Base - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
</footer>
```



Por último, nos encontramos con la importación de las librerías JavaScript necesarias. Por defecto se utilizan tanto las librerías de jQuery como Bootstrap, ya que nuestra plantilla necesita de estas librerías, ofreciendo soporte para su uso en todas nuestras vistas. Una vez más se utilizan diferentes mecanismos en función de si estamos trabajando en desarrollo o en un entorno final de producción. Se importa también un fichero JavaScript genérico llamado site.js, que nos permite incluir nuestro propio código JavaScript en la aplicación. La última parte permite incluir código de una vista parcial, pero en este caso especificamos que no es obligatorio que incluyamos dicha vista parcial en todas nuestras vistas. De este modo, podremos utilizar, en aquellas páginas que lo necesitemos, otros scripts como pueden ser los incluidos en la vista parcial *_ValidationScriptsPartial.cshtml*, en los formularios, por ejemplo.

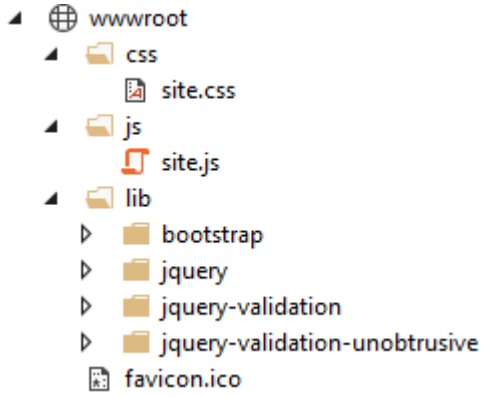
```
<environment include="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.js"></script>
</environment>
<environment exclude="Development">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"
            asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
            asp-fallback-test="window.jQuery"
            crossorigin="anonymous"
            integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.3/js/bootstrap.bundle.min.js"
            asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"
            asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
            crossorigin="anonymous"
            integrity="sha256-E/V4cWE4qvAeO5MOhjtGtqDzPndRO1LBk81J/PR7CA4=">
    </script>
</environment>
<script src="~/js/site.js" asp-append-version="true"></script>

@RenderSection("Scripts", required: false)
```

Contenido estático (wwwroot).

La última carpeta que vamos a analizar, es la que contiene el contenido estático de nuestra aplicación. Todos los ficheros incluidos en esta carpeta estarán disponibles tal cual en nuestro servidor, y es el lugar donde incluiremos cualquier contenido que nos haga falta, habitualmente imágenes o cualquier tipo de documento necesario.

Nuestra plantilla incluirá por defecto dentro de la carpeta de contenido estático todas las hojas de estilos y ficheros JavaScript explicados anteriormente en la sección dedicada a la carpeta vistas, y especialmente a la plantilla o Layout.

En la imagen podemos ver como encontramos la hoja de estilos y el fichero de código JavaScript específicos de nuestra aplicación, además de las librerías Bootstrap para la presentación de nuestras vistas, y las librerías jQuery Validation para la validación del lado cliente.



Conclusión.

Teniendo en cuenta la cantidad de elementos que se incluyen en la plantilla de aplicación MVC, la recomendación es partir de ésta para la creación de nuestros nuevos proyectos. En cuanto a qué mantener y que no, particularmente pienso que la mejor opción y las más simple es mantener todo excepto quizás la vista de política del sitio, y para los ejercicios más simples eliminar el menú superior ya que puede resultar innecesario.