



**Escuela Politécnica Superior de
Alicante**

Diseño de aplicaciones web

**Taller de ASP.net
CORE I**

Contenidos

- Creación de un proyecto
- Un vistazo al proyecto
- Preparando el layout
- Ejemplo sencillo de dinámica de trabajo.
 - Modelo – Servicio – Controlador – Vista
- Patrón listado tabular con ordenación
- Vista detalle
 - Mediante nueva página de Razor
 - Llamadas HTTP para refrescar contenidos en tiempo de cliente.

Creación de un proyecto

Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.

Search for templates (Alt+S) Clear all

C# All platforms Web

 React.js and Redux New
A project template for creating an ASP.NET Core application with React.js and Redux
C# Linux macOS Windows Cloud Service Web

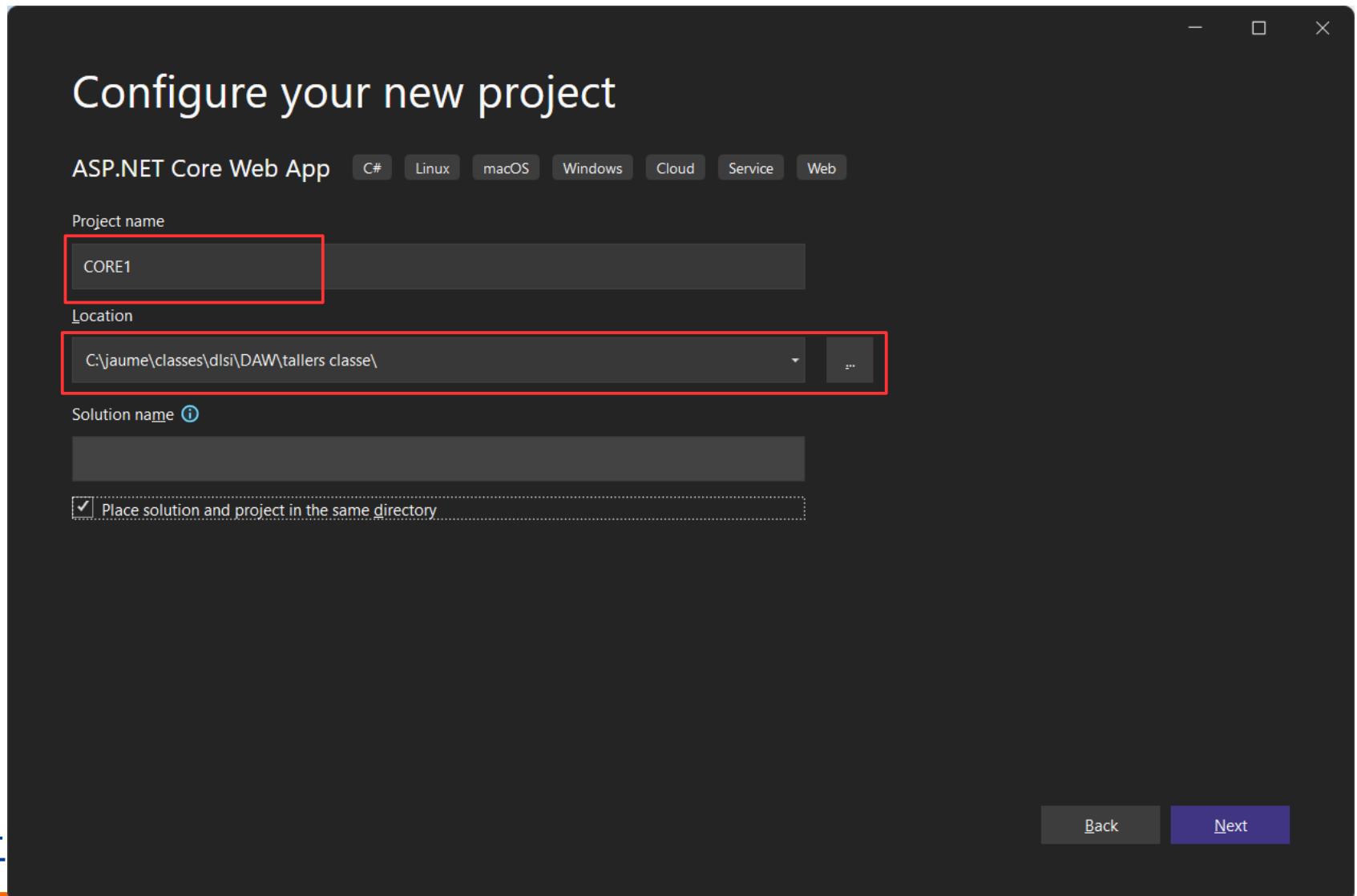
 ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

 Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

 ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI

Back Next

Creación de un proyecto



Creación de un proyecto

Additional information

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Framework ⓘ

.NET 6.0 (Long-term support)

Authentication type ⓘ

None

Configure for HTTPS ⓘ

Enable Docker ⓘ

Docker OS ⓘ

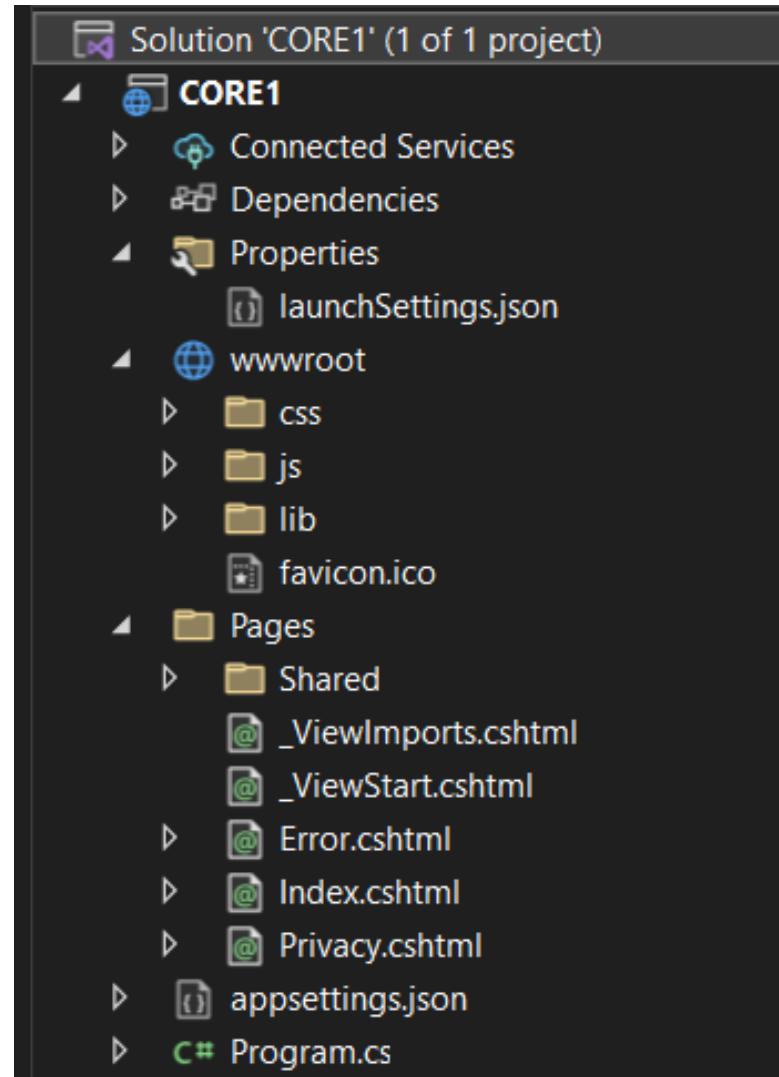
Linux

Do not use top-level statements ⓘ

Back Create

Un vistazo al proyecto

- wwwroot: Raíz del proyecto y contenidos estáticos.
- Ficheros de configuración:
 - Properties/LaunchSettings.json
 - Appsettings.json
 - Program.cs
- Páginas creadas por la plantilla: Index, privacy



Vistas compartidas: plantilla _layout

- Inicio del documento y cabeceras

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ ViewData["Title"] - CORE1</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="~/CORE1.styles.css" asp-append-version="true" />
</head>
```

- Header: encabezado del documento y menú

```
<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-page="/Index">CORE1</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-page="/Index">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>
```

Vistas compartidas: plantilla _layout

- Contenido, pie de página, scripts:

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

<footer class="border-top footer text-muted">
    <div class="container">
        © 2021 - TallerCORE - <a asp-area="" asp-page="/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>

@RenderSection("Scripts", required: false)
</body>
</html>
```

Reestructurando el layout

- Vamos a crear tres vistas parciales para: la cabecera, el menú y el pie de página.
- Vista parcial: `views/shared/_head.cshtml`

```
<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-page="/Index">TallerCORE</a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse"
        aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <partial name="_menu.cshtml" />
    </div>
  </nav>
</header>
```

Reestructurando el layout

- Vista parcial: views/shared/_menu.cshtml

```
<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-page="/Index">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-page="/Tiendas">Tiendas</a>
    </li>
    <!--<li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
    </li>-->
  </ul>
</div>
```

Reestructurando el layout

- Vista parcial: views/shared/_foot.cshtml

```
<footer class="border-top footer text-muted">
  <div class="container">
    © 2021 - TallerCORE - <a asp-area="" asp-page="/Privacy">Privacy</a>
  </div>
</footer>
```

- Añadiremos una región nueva para estilos CSS in-line

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - TallerCORE</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
  @RenderSection("styles", required: false)
</head>
```

Reestructurando el layout

- La plantilla '_layout' quedará así:

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

    <partial name="_foot.cshtml" />

    <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>

    @RenderSection("Scripts", required: false)
</body>
</html>
```

Modelo, view-model, vista

- Añadiremos un modelo, un servicio y una página nueva para ofrecer un listado de tiendas.
- El modelo contendrá los datos de una tienda (nombre, dirección, localidad, teléfono, e-mail)
- El servicio obtendrá los datos y los devolverá en forma de lista.
- La nueva página invocará al servicio y devolverá los datos a la vista para que los muestre en pantalla.

Modelo, view-model, vista

- Para añadir un modelo, crearemos una nueva carpeta 'Models' y en su interior una nueva clase: 'Tienda.cs', con el siguiente contenido:

```
namespace TallerCORE.Models
{
    12 referencias
    public class Tienda
    {
        2 referencias
        public string nombre { get; set; }
        1 referencia
        public string direccion { get; set; }
        1 referencia
        public string localidad { get; set; }
        1 referencia
        public string email { get; set; }
        1 referencia
        public string telefonos { get; set; }
        3 referencias
        public Tienda(string n="", string d="", string l="", string e="", string t="")
        {
            nombre = n;
            direccion = d;
            localidad = l;
            email = e;
            telefonos = t;
        }
    }
}
```

Modelo, view-model, vista

- Ahora añadiremos un servicio, para ello crearemos una nueva carpeta 'Services' y en su interior una nueva clase: 'Service.cs'
- Ésta contendrá una función estática para obtener los datos de las tiendas: getTiendas.

```
namespace TallerCORE.Services
{
    1 referencia
    public class Service
    {
        1 referencia
        public static List<Tienda> getTiendas()
        {
            List<Tienda> list = new List<Tienda>();

            Tienda t1 = new Tienda("Central", "calle del pez, 5", "Alacant", "central@empresa.com", "965656565");
            list.Add(t1);

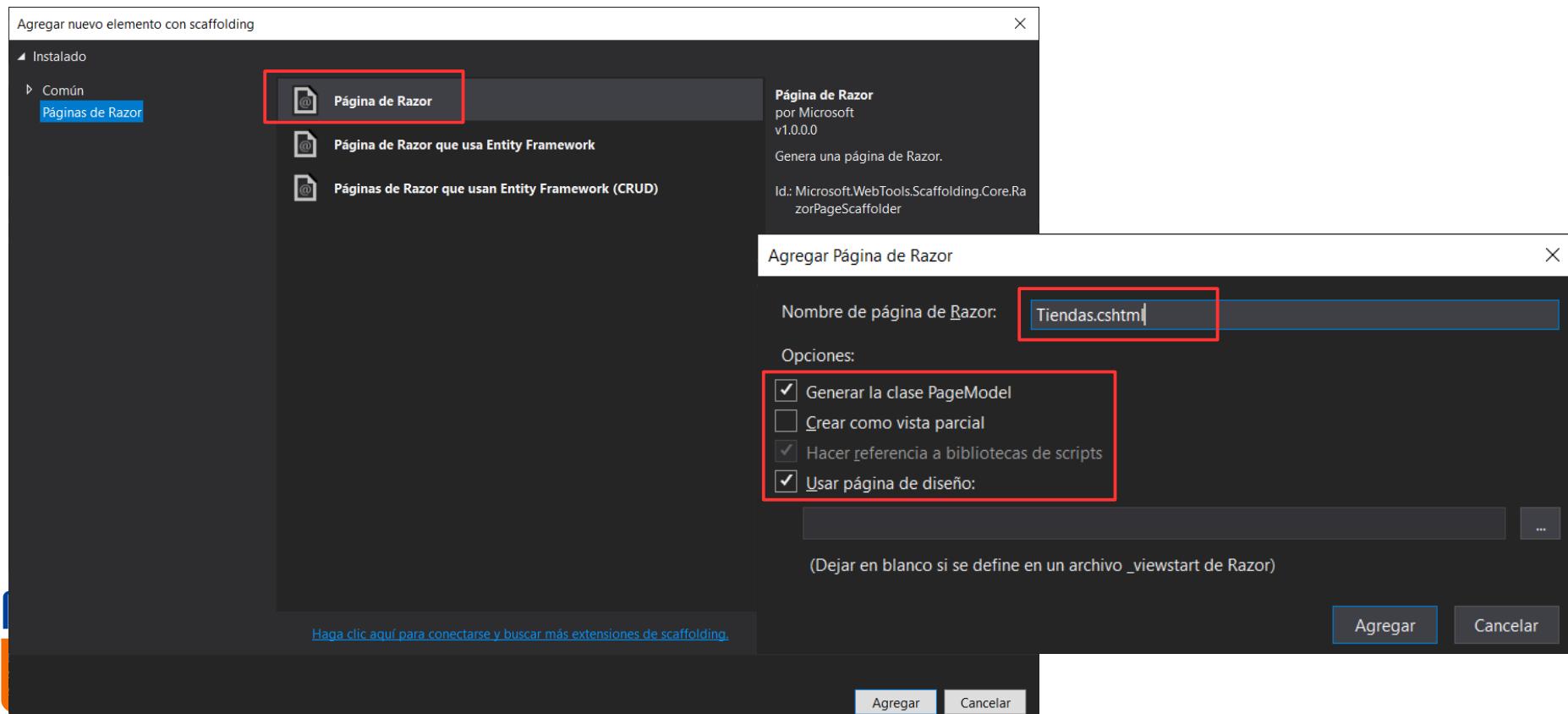
            Tienda t2 = new Tienda("Playa", "calle del mar, 15", "Playa Sant Joan", "playa@empresa.com", "965222324");
            list.Add(t2);

            Tienda t3 = new Tienda("Alcoi", "calle de la montaña, 50", "Alcoi", "alcoi@empresa.com", "965605040");
            list.Add(t3);

            return list;
        }
    }
}
```

Modelo, view-model, vista

- A continuación añadiremos una nueva página: Tiendas.cshtml.
 - Generaremos la clase 'PageModel' y usaremos la plantilla de diseño.



Modelo, view-model, vista

- En el código del controlador 'Tiendas.cshtml.cs' se accederá a los datos y se almacenarán en el 'viewmodel':

The screenshot shows a code editor with a dark theme. On the left, there's a navigation pane with a tree structure. The main area displays the following C# code:

```
namespace TallerCORE
{
    public class TiendasModel : PageModel
    {
        [BindProperty]
        public List<Tienda> list { get; set; }

        public void OnGet()
        {
            this.list = Service.getTiendas();
        }
    }
}
```

Two specific parts of the code are highlighted with red boxes:

- A red box surrounds the line `[BindProperty]`.
- A red box surrounds the line `this.list = Service.getTiendas();`.

Modelo, view-model, vista

- En la vista 'Tiendas.cshtml' se recorrerán los datos obtenidos y se mostrarán:

```
@page
@model TallerCORE.TiendasModel
 @{
    ViewData["Title"] = "Tiendas";
}

<h1>Tiendas</h1>

<ul>
    @foreach (Models.Tienda t in Model.list)
    {
        <li>@t.nombre (<a href="mailto:@t.email">@t.email</a>)</li>
    }
</ul>
```

Modelo, view-model, vista

- Añadimos la nueva opción en el menú de la vista parcial '_menu.cshtml' de la plantilla 'Layout.cshtml':

```
<div class="container">
    <a class="navbar-brand" asp-area="" asp-page="/Index">TallerCORE</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse"
        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-page="/Index">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-page="/Tiendas">Tiendas</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
            </li>
        </ul>
    </div>
</div>
</header>
```

Patrón listado tabular con ordenación

- Perfeccionamos el código añadiendo un mensaje si no hay datos y cambiando el listado por una tabla:

```
=@if (Model.list.Count > 0)
{
    <table class="table table-striped table-bordered">
        <thead>
            <tr>
                <th>Nombre</th><th>Dirección</th><th>Población</th>
            </tr>
        </thead>
        <tbody>
            @foreach (Models.Tienda t in Model.list)
            {
                <tr>
                    <td>@t.nombre</td>
                    <td>@t.direccion</td>
                    <td>@t.localidad</td>
                </tr>
            }
        </tbody>
    </table>
} else
{
    <div class="alert alert-danger">No se han encontrado datos</div>
}
```

Patrón listado tabular con ordenación

- Vamos a implementar un sistema de ordenación por nombre y población de la tienda. En sentido ascendente y descendente.
- Primero, añadiremos dos variables al modelo: orden y sentido.
- Despues debemos modificar la función 'OnGet':

```
public void OnGet(string orden="N", string sentido="A")
{
    IQueryables<Tienda> aux = Service.getTiendas().AsQueryable();

    if (String.IsNullOrEmpty(this.orden))
    {
        this.orden = orden;
    }
    if (String.IsNullOrEmpty(this.sentido))
    {
        this.sentido = sentido;
    }

    switch (orden + sentido)
    {
        case "NA":
            aux = aux.OrderBy(t => t.nombre);
            break;
        case "ND":
            aux = aux.OrderByDescending(t => t.nombre);
            break;
    }

    case "PA":
        aux = aux.OrderBy(t => t.localidad);
        break;
    case "PD":
        aux = aux.OrderByDescending(t => t.localidad);
        break;
    }

    this.sentido=(sentido=="A"? "D": "A");
    this.orden = orden;

    this.list = aux.ToList();
}
```

Patrón listado tabular con ordenación

- Ahora, debemos modificar la cabecera de la tabla del listado de forma que dos de las columnas (nombre y localidad) sean clicables y enlacen con la propia página enviando dos datos: orden y sentido.

```
<table class="table table-striped table-bordered">
  <thead>
    <tr>
      <th><a asp-page=".~/Tiendas" asp-route-orden="N" asp-route-sentido="@Model.sentido">Nombre</a></th>
      <th>Dirección</th>
      <th><a asp-page=".~/Tiendas" asp-route-orden="P" asp-route-sentido="@Model.sentido">Población</a></th>
    </tr>
  </thead>
  <tbody>
    @foreach (Models.Tienda t in Model.list)
    {
      <tr>
        <td>@t.nombre</td>
        <td>@t.direccion</td>
        <td>@t.localidad</td>
      </tr>
    }
  </tbody>
</table>
```

Nueva página: vista detalle

- Vamos a añadir una página para el detalle de cada tienda.
- Primero añadiremos un nuevo campo 'id' al modelo 'Tienda' que permita identificar cada una de forma única.
 - Tendremos que modificar el servicio y la función

```
public class Tienda
{
    2 referencias
    public decimal id { set; get; }
    4 referencias
    public string nombre { get; set; }
    2 referencias
    public string direccion { get; set; }
    4 referencias
    public string localidad { get; set; }
    1 referencia
    public string email { get; set; }
    1 referencia
    public string telefonos { get; set; }
    4 referencias
    public Tienda(decimal i=0, string n="", string d="", string l="", string e="", string t="")
    {
        id = i;
        nombre = n;
        direccion = d;
        localidad = l;
        email = e;
        telefonos = t;
    }
}
```

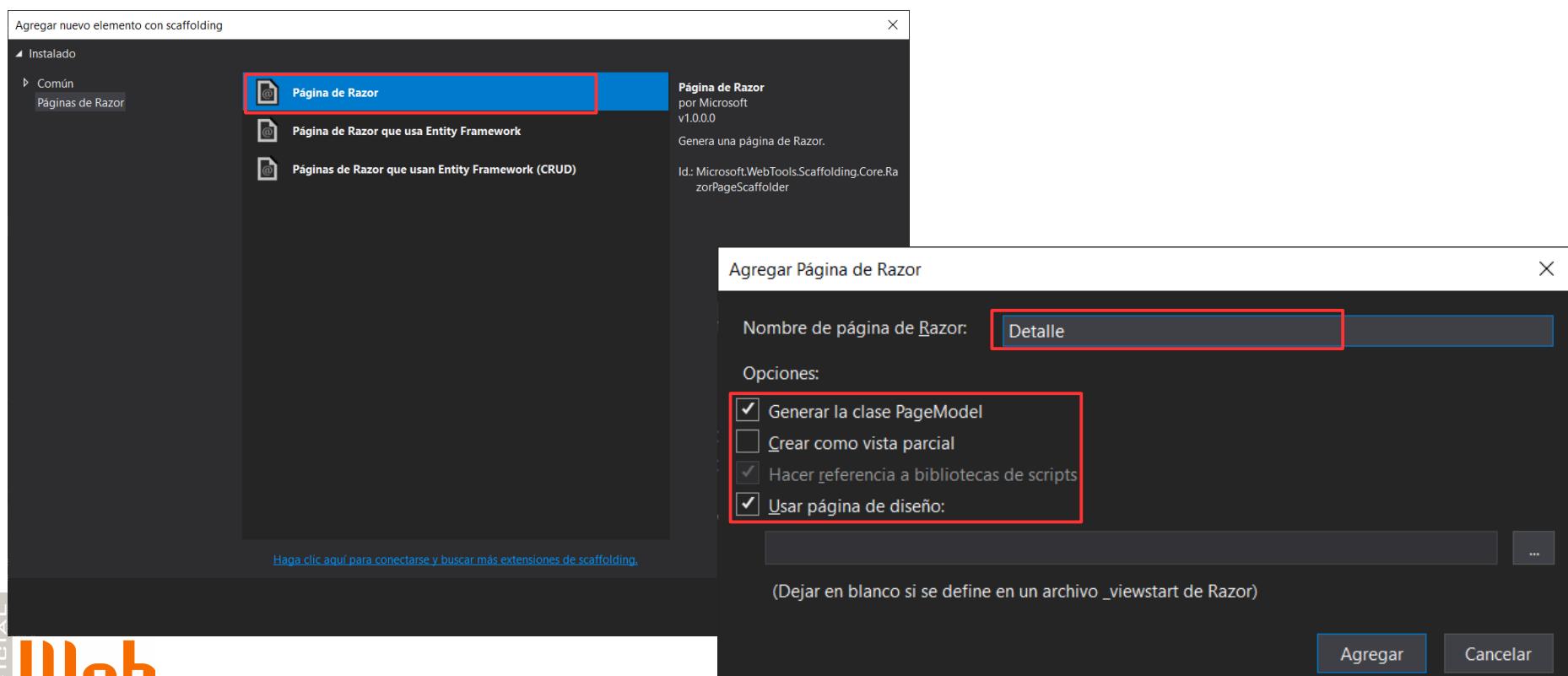
Nueva página: vista detalle

- En la vista añadiremos una columna al listado tabular y en cada fila un botón que enlace a una nueva página 'Detalle' enviando el 'id' de la tienda.

```
<table class="table table-striped table-bordered">
  <thead>
    <tr>
      <th><a asp-page=".Tiendas" asp-route-orden="N" asp-route-sentido="@Model.sentido">Nombre</a></th>
      <th>Dirección</th>
      <th><a asp-page=".Tiendas" asp-route-orden="P" asp-route-sentido="@Model.sentido">Población</a></th>
      <th>&nbsp;</th>
    </tr>
  </thead>
  <tbody>
    @foreach (Models.Tienda t in Model.list)
    {
      <tr>
        <td>@t.nombre</td>
        <td>@t.direccion</td>
        <td>@t.localidad</td>
        <td>
          <a asp-page=".Detalle" asp-route-id="@t.id" class="btn btn-info">Detalle</a>
        </td>
      </tr>
    }
  </tbody>
</table>
```

Nueva página: vista detalle

- Ahora añadimos una nueva página de Razor, llamada 'Detalle', generaremos su clase 'PageModel' y usaremos la página de diseño del proyecto.



Nueva página: vista detalle

- En la página 'Detalle' obtendremos una tienda individual con la clausula 'where' de Linq:

```
public class DetalleModel : PageModel
{
    2 referencias
    public Tienda tienda { get; set; }
    0 referencias
    public void OnGet(decimal id)
    {
        this.tienda = Service.getTiendas().Where(t => t.id == id).FirstOrDefault();
    }
}
```

Nueva página: vista detalle

- Diseñaremos la vista 'Detalle.cshtml' para mostrar los datos de la tienda recibida en el modelo:

```
<h1>Datos de nuestra tienda @Model.tienda.nombre</h1>
<table class="table table-bordered table-striped">
    <tbody>
        <tr>
            <td><strong>Nombre:</strong></td>
            <td>@Model.tienda.nombre</td>
        </tr>
        <tr>
            <td><strong>Dirección:</strong></td>
            <td>@Model.tienda.direccion (@Model.tienda.localidad)</td>
        </tr>
        <tr>
            <td><strong>E-mail:</strong></td>
            <td><a href="mailto:@Model.tienda.email">@Model.tienda.email</a></td>
        </tr>
        <tr>
            <td><strong>Teléfonos:</strong></td>
            <td>@Model.tienda.telefonos</td>
        </tr>
    </tbody>
</table>
<a asp-page=".~/Tiendas" class="btn btn-success">Volver al listado</a>
```

Detalle de tienda alternativa con llamada http asíncrona

- Para realizar llamadas AJAX (invocaciones HTTP asíncronas) en un proyecto ASP.net CORE Razor Pages tenemos dos opciones:
 - Usar manejadores con nombre en la propia página
 - Usar una página Razor dedicada para las peticiones HTTP o AJAX.
- La primera opción es más sencilla pero no cumple con el principio de cohesión, pues estamos mezclando llamadas de distintos tipos en el mismo controlador o PageModel.
- Con la segunda opción mantenemos la cohesión y podemos agrupar las llamadas HTTP en un mismo controlador.

Detalle de tienda alternativa con llamada HTTP

- Vamos a implementar la llamada HTTP siguiendo la primera opción: manejador con nombre en la propia página.
- Primero añadiremos un nuevo botón en cada tienda en el listado tabular, para permitir invocar esta nueva opción.

```
<tr>
    <td>@t.nombre</td>
    <td>@t.direccion</td>
    <td>@t.localidad</td>
    <td>
        <a asp-page=".~/Detalle" asp-route-id="@t.id" class="btn btn-info">Detalle</a>
        &ampnbsp
        <button onclick="abrirDetalle(@t.id)" class="btn btn-secondary">Detalle2</button>
    </td>
</tr>
```

Detalle de tienda alternativa con llamada HTTP

- En la vista 'Tiendas.cshtml' añadiremos un diálogo modal:

```
<div class="modal modal-info" tabindex="-1" role="dialog" id="detalleTienda">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Datos de la tienda</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <table class="table table-bordered table-striped">
                    <tbody>
                        <tr>
                            <td><strong>Nombre:</strong></td><td id="modalNombre"></td>
                        </tr>
                        <tr>
                            <td><strong>Dirección:</strong></td><td id="modalDireccion"></td>
                        </tr>
                        <tr>
                            <td><strong>E-mail:</strong></td><td id="modalEmail"></td>
                        </tr>
                        <tr>
                            <td><strong>Teléfonos:</strong></td><td id="telefonos"></td>
                        </tr>
                    </tbody>
                </table>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Cerrar</button>
            </div>
        </div>
    </div>
</div>
```

Detalle de tienda alternativa con llamada HTTP

- Finalmente, añadiremos la sección Scripts con la función 'abrirDetalle':

```
@section Scripts {
<script type="text/javascript">
    function abrirDetalle(id) {
        $.ajax({
            url: '/Tiendas?handler=Detalle',
            data: {
                id: id
            }
        })
        .done(function (result) {
            $("#tituloModal").html(result.nombre);
            $("#modalNombre").html(result.nombre);
            var direccion = result.direccion + " (" + result.localidad + ")";
            $("#modalDireccion").html(direccion);
            var email = "<a href='mailto:" + result.email + "'>" + result.email + "</a>";
            $("#modalEmail").html(email);
            $("#modalTelefonos").html(result.telefonos);
            $('#detalleTienda').modal('show');
        });
    }
</script>
}
```

Detalle de tienda alternativa con llamada HTTP

- Solo nos queda implementar la parte de back-end, añadiendo al PageModel 'Tiendas.cshtml.cs' el método que responderá la llamada HTTP:

```
0 referencias
public JsonResult OnGetDetalle(decimal id)
{
    Tienda tienda = Service.getTiendas().Where(t => t.id == id).FirstOrDefault();
    return new JsonResult(tienda);
}
```

Ejercicio:

- A partir del proyecto del taller, vamos a añadir las siguientes mejoras o modificaciones:
 - Añadir un campo dirección web (llamado 'url') y refactorizar el código para incluirlo en el proyecto.
 - Añadir código que muestre un mensaje de error para el caso de que no encuentre la tienda en el detalle.
 - Comprobar si la llamada HTTP da error y mostrar el mensaje oportuno.

Ejercicio adicional:

- Añadir el patrón de filtrado de datos a la tabla de tiendas, permitiendo filtrar por población, para ello:
 - Añadir un modelo 'Localidad'
 - Modificar el modelo 'Tienda' para incorporar una localidad en 'layering'
 - Modificar el servicio para incorporar los cambios anteriores y para añadir el método 'getPoblaciones'
 - Añadir un desplegable de localidades en la página de tiendas cuyas opciones se obtengan desde la fuente de datos (servicio/modelos)
 - Vincular un evento al desplegable anterior para realizar la llamada al back-end y obtener las tiendas filtradas.

Referencias

- Tutorial sobre ASP.net CORE:
 - <https://docs.microsoft.com/es-es/aspnet/core/data/ef-rp/intro?view=aspnetcore-6.0&tabs=visual-studio>
- Artículo sobre Llamadas AJAX con Razor Pages:
 - <https://www.thereformedprogrammer.net/asp-net-core-razor-pages-how-to-implement-ajax-requests/>