



© JMA 2020. All rights reserved

Conocimientos

- Mínimos:
 - HTML, CSS, DOM, XML
 - JavaScript
 - C#
 - ADO.NET
- Deseables
 - ASP.NET
 - Entity Framework
 - LINQ
 - JSON, AJAX
 - JQuery,
 - Patrones:
 - MVC, MVP, MVVM
 - DI, IoC
 - Nuget

© JMA 2020. All rights reserved

INTRODUCCIÓN

© JMA 2020. All rights reserved

Introducción

- ASP.NET Core es un marco multiplataforma de código abierto y de alto rendimiento que tiene como finalidad compilar aplicaciones modernas conectadas a Internet y habilitadas para la nube.
- Con ASP.NET Core se puede hacer lo siguiente:
 - Crear servicios y aplicaciones web, aplicaciones de Internet de las cosas (IoT) y back-ends móviles.
 - Usar las herramientas de desarrollo favoritas en Windows, macOS y Linux.
 - Efectuar implementaciones locales y en la nube.
 - Ejecutar en .NET Core.
- Millones de desarrolladores usan o han usado ASP.NET 4.x para crear aplicaciones web. ASP.NET Core es un nuevo diseño de ASP.NET 4.x que incluye cambios en la arquitectura que dan como resultado un marco más sencillo y modular.

© JMA 2020. All rights reserved

Ventajas

- De código abierto y centrado en la comunidad.
- Marco común para crear API Web y una interfaz de usuario web.
- Razor Pages hace que la codificación de escenarios centrados en páginas sean más sencillos y productivos.
- Blazor permite usar C# en el explorador, además de JavaScript. Comparta la lógica de aplicación del lado cliente y servidor escrita toda con .NET.
- Integración de marcos del lado cliente modernos y flujos de trabajo de desarrollo.
- Compatibilidad con el hospedaje de servicios de llamada a procedimiento remoto (RPC) con gRPC.
- Capacidad para desarrollarse y ejecutarse en Windows, macOS y Linux.
- Capacidad para hospedarse en: Kestrel, IIS, HTTP.sys, Nginx, Apache, Docker
- Un sistema de configuración basado en el entorno y preparado para la nube.
- Una canalización de solicitudes HTTP ligera, modular y de alto rendimiento.
- Inyección de dependencias integrada.
- Diseñado para capacitar las pruebas.
- Control de versiones en paralelo.
- Herramientas que simplifican el desarrollo web moderno.

© JMA 2020. All rights reserved

ASP.NET Core MVC

- Proporciona características para crear aplicaciones web y API Web (servicios REST).
- El patrón Modelo-Vista-Controlador (MVC) permite que se puedan hacer pruebas automatizadas en las aplicaciones web y en las API Web.
- Razor Pages es un modelo de programación basado en páginas que facilita la compilación de interfaces de usuario web y hace que sea más productiva.
- El marcado de Razor proporciona una sintaxis productiva para Razor Pages y las vistas de MVC.
- Los HTML Helpers y los asistentes de etiquetas permiten que el código de servidor participe en la creación y la representación de elementos HTML en los archivos de Razor.
- La compatibilidad integrada para varios formatos de datos y la negociación de contenidos permite que las API Web lleguen a una amplia gama de clientes, como los exploradores y los dispositivos móviles.
- El enlace de modelo asigna automáticamente datos de solicitudes HTTP a parámetros de método de acción.
- La validación de modelos efectúa tanto la validación del lado cliente como del lado servidor de forma automática.
- SignalR simplifica las funcionalidades Web en tiempo real a las aplicaciones.

© JMA 2020. All rights reserved

Implementaciones de .NET

- Una aplicación de .NET se desarrolla y se ejecuta en una o varias implementaciones de .NET. Las implementaciones de .NET incluyen .NET Framework, .NET Core y Mono. Hay una especificación de API común a todas las implementaciones de .NET que se denomina .NET Standard.
- .NET Standard es un conjunto de API que se implementan mediante la biblioteca de clases base de una implementación de .NET, que constituyen un conjunto uniforme de contratos contra los que se compila el código. Estos contratos se implementan en cada implementación de .NET. Esto permite la portabilidad entre diferentes implementaciones de .NET, de forma que el código se puede ejecutar en cualquier parte.
- .NET Standard es también una plataforma de destino. Si el código tiene como destino una versión de .NET Standard, se puede ejecutar en cualquier implementación de .NET que sea compatible con esa versión de .NET Standard.

© JMA 2020. All rights reserved

Implementaciones de .NET

- Cada implementación de .NET incluye los siguientes componentes:
 - Uno o varios entornos de ejecución. Ejemplos: CLR para .NET Framework, CoreCLR y CoreRT para .NET Core.
 - Una biblioteca de clases que implementa .NET Standard y puede implementar API adicionales. Ejemplos: biblioteca de clases base de .NET Framework, biblioteca de clases base de .NET Core.
 - Opcionalmente, uno o varios marcos de trabajo de la aplicación. Ejemplos: ASP.NET, Windows Forms y Windows Presentation Foundation (WPF) se incluyen en .NET Framework y .NET Core.
 - Opcionalmente, herramientas de desarrollo. Algunas herramientas de desarrollo se comparten entre varias implementaciones.
- Hay cuatro implementaciones principales de .NET que Microsoft desarrolla y mantiene activamente:
 - .NET Framework, .NET Core, Mono y UWP.

© JMA 2020. All rights reserved

.NET Framework

- También conocida como .NET "Full Framework" o .NET "Tradicional".
- Se trata de la plataforma .NET "de toda la vida", aparecida oficialmente en 2001. Monolítica (instalas todo su núcleo o no instalas nada), pero tremendamente capaz, ya que con ella puedes crear aplicaciones de cualquier tipo: de consola, de escritorio, para la web, móviles... Casi cualquier cosa que puedas imaginar.
- Eso sí, se trata de aplicaciones que se ejecutarán solamente sobre Windows y está optimizado para crear aplicaciones de escritorio de Windows.
- Las versiones 4.5 y posteriores implementan .NET Standard, de forma que el código que tiene como destino .NET Standard se puede ejecutar en esas versiones de .NET Framework y existen miles de componentes de terceros para poder extenderla.
- Con ella puedes utilizar también infinidad de lenguajes de programación: C#, Visual Basic .NET, F#, C++, Python... ¡Hasta Cobol!
- La utilizan miles de empresas en todo el mundo. No va a cambiar mucho en los próximos años.

© JMA 2020. All rights reserved

.NET Core

- Es una nueva plataforma, escrita desde cero con varios objetivos en mente, siendo los principales:
 - Más ligera y "componentizable": de modo que con nuestra aplicación se distribuya exclusivamente lo que necesitamos, no la plataforma completa. En cuanto a los lenguajes disponibles, podremos utilizar C#, F# y Visual Basic .NET.
 - Multi-plataforma: las aplicaciones que creemos funcionarán en Windows, Linux y Mac, no solo en el sistema de Microsoft.
 - Alto rendimiento: no es que .NET tradicional no tuviese buen rendimiento, pero es que .NET Core está específicamente diseñada para ello. .NET Core tiene un desempeño más alto que la versión tradicional (minimiza dependencias y servicios adicionales) lo cual es muy importante para entornos Cloud, en donde esto se traduce en mucho dinero al cabo del tiempo.
 - Pensada para la nube: cuando .NET se diseñó a finales de los años 90 el concepto de nube ni siquiera existía. .NET Core nace en la era Cloud, por lo que está pensada desde el principio para encajar en entornos de plataforma como servicio y crear aplicaciones eficientes para su funcionamiento en la nube (sea de Microsoft o no).

© JMA 2020. All rights reserved

Mono

- Mono es una implementación de .NET que se usa principalmente cuando se requiere un entorno de ejecución pequeño. Es el entorno de ejecución que activa las aplicaciones Xamarin en Android, macOS, iOS, tvOS y watchOS, y se centra principalmente en una superficie pequeña. Mono también proporciona juegos creados con el motor de Unity.
- Admite todas las versiones de .NET Standard publicadas actualmente.
- Históricamente, Mono implementaba la API de .NET Framework más grande y emulaba algunas de las funciones más populares en Unix. A veces, se usa para ejecutar aplicaciones de .NET que se basan en estas capacidades en Unix.
- Mono se suele usar con un compilador Just-In-Time, pero también incluye un compilador estático completo (compilación Ahead Of Time) que se usa en plataformas como iOS.

© JMA 2020. All rights reserved

Plataforma universal de Windows (UWP)

- UWP es una implementación de .NET que se usa para compilar aplicaciones Windows modernas y táctiles, así como software para Internet de las cosas (IoT).
- Se ha diseñado para unificar los diferentes tipos de dispositivos de destino (Windows 10), incluidos equipos, tabletas, teléfonos e incluso la consola Xbox.
- UWP proporciona muchos servicios, como una tienda de aplicaciones centralizada, un entorno de ejecución (AppContainer) y un conjunto de API de Windows para usar en lugar de Win32 (WinRT).
- Pueden escribirse aplicaciones en C++, C#, Visual Basic y JavaScript. Al usar C# y Visual Basic, .NET Core proporciona las API de .NET.

© JMA 2020. All rights reserved

.NET Core o .NET Framework

- Usar .NET Core para la aplicación de servidor cuando:
 - Tenga necesidades multiplataforma.
 - Tenga los microservicios como objetivo.
 - Vaya a usar contenedores de Docker.
 - Necesite sistemas escalables y de alto rendimiento.
 - Necesite versiones de .NET en paralelo por aplicación.
- Usar .NET Framework para su aplicación de servidor cuando:
 - La aplicación usa actualmente .NET Framework (la recomendación es extender en lugar de migrar).
 - La aplicación usa bibliotecas .NET de terceros o paquetes de NuGet que no están disponibles para .NET Core.
 - La aplicación usa tecnologías de .NET que no están disponibles para .NET Core.
 - La aplicación usa una plataforma que no es compatible con .NET Core. Windows, macOS y Linux admiten .NET Core.

© JMA 2020. All rights reserved

Herramientas

- Profesionales:
 - Visual Studio 2019 con la carga de trabajo de “Desarrollo multiplataforma de .NET Core”
- Open source:
 - Visual Studio Code (<https://code.visualstudio.com/>)
 - Extensión C# para Visual Studio Code (<https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp>)
 - SDK de .NET Core 3.0 o versiones posteriores (<https://dotnet.microsoft.com/download>)
 - La interfaz de la línea de comandos (CLI) de .NET Core es una cadena de herramientas multiplataforma para desarrollar, compilar, ejecutar y publicar aplicaciones .NET Core.
 - dotnet new webapp
 - dotnet build --output /build_output
 - dotnet watch run

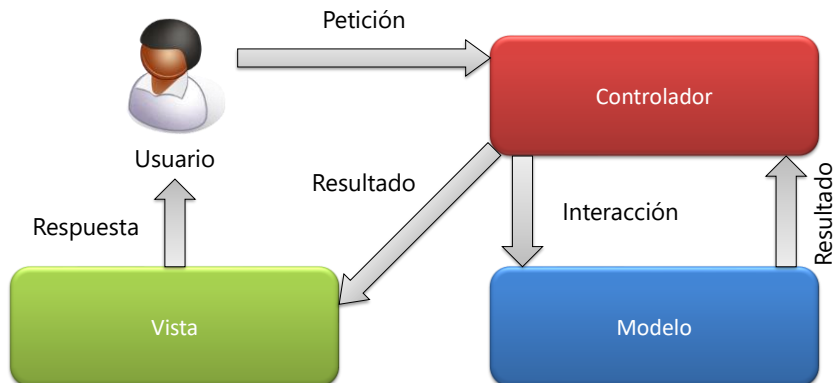
© JMA 2020. All rights reserved

El patrón MVC

- El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software (presentación) que separa los datos y la lógica de negocio de una aplicación del interfaz de usuario y del módulo encargado de gestionar los eventos y las comunicaciones.
- Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones, prueba y su posterior mantenimiento.
- Para todo tipo de sistemas (Escritorio, Web, Movil, ...) y de tecnologías (Java, Ruby, Python, Perl, Flex, SmallTalk, .Net ...)

© JMA 2020. All rights reserved

El patrón MVC en ASP.NET



© JMA 2020. All rights reserved

El patrón MVC



- Representación de los **datos del dominio**
- Lógica de **negocio**
- Mecanismos de **persistencia**



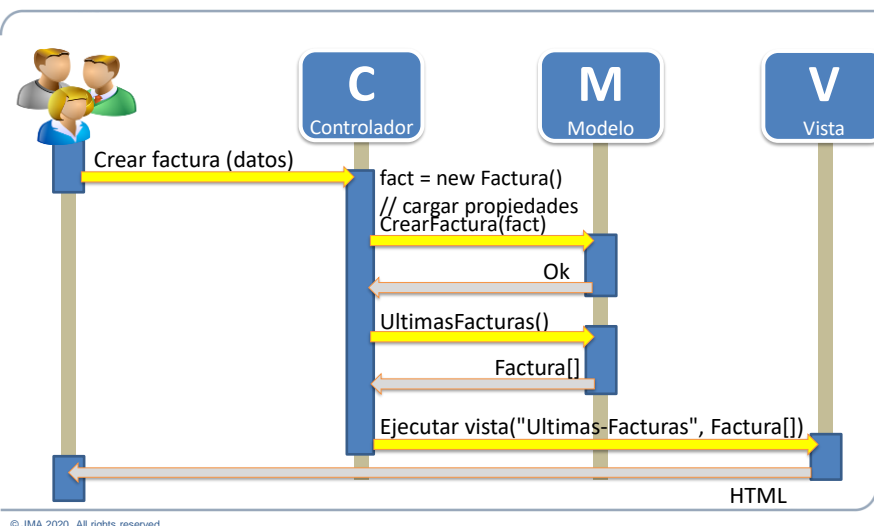
- **Interfaz** de usuario
- Incluye elementos de **interacción**



- **Intermediario** entre Modelo y Vista
- **Mapea acciones** de usuario → acciones del Modelo
- **Selecciona** las vistas y les **suministra** información

© JMA 2020. All rights reserved

El patrón MVC



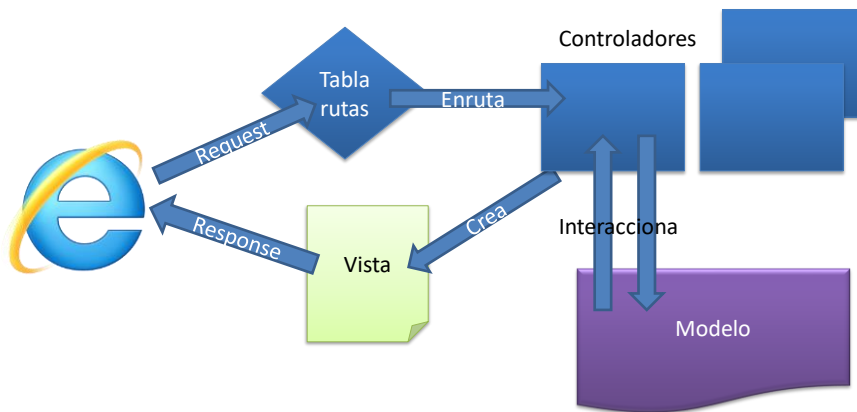
© JMA 2020. All rights reserved

Componentes principales en MVC

- Routing:
 - Route tables
- Model:
 - Business Logic & Domain Entities
- Controllers:
 - Presentation Logic
- Views:
 - Presentation Model, Model state

© JMA 2020. All rights reserved

Modelo, Vista, Controlador



© JMA 2020. All rights reserved

Tabla de rutas

- Dada una URL decide qué acción de qué controlador procesa esta acción
- Sólo tiene en cuenta la URL (nada de parámetros POST, query string, ...)
- Ruta por defecto:
 - `/controller/action/id`
- Ruta con área:
 - `nombreArea/controller/action/id`

© JMA 2020. All rights reserved

Controladores

- Los controladores son los componentes que controlan la interacción del usuario, trabajan con el modelo y por último seleccionan una vista para representar la interfaz de usuario.
- Exponen acciones que se encargan de procesar las peticiones
- Cada acción debe devolver un resultado, que es algo que el framework debe hacer (mandar una vista, un fichero binario, un 404, ...)
- Hablan con el modelo pero son «tontos»

© JMA 2020. All rights reserved

Modelo

- Los objetos de modelo son las partes de la aplicación que implementan la lógica del dominio de datos de la aplicación.
- A menudo, los objetos de modelo recuperan y almacenan el estado del modelo en una base de datos.
- Encapsula toda la lógica de nuestra aplicación
- Responde a peticiones de los controladores

© JMA 2020. All rights reserved

Vistas

- Las vistas son los componentes que muestra la interfaz de usuario de la aplicación.
- Normalmente, esta interfaz de usuario se crea a partir de los datos de modelo.
- Se encarga únicamente de temas de presentación.
- Es «básicamente» código HTML (con un poco de server-side)
- NO acceden a BBDD, NO toman decisiones, NO hacen nada de nada salvo...
- ... mostrar información

© JMA 2020. All rights reserved

Características

- El modelo MVC ayuda a crear aplicaciones que separan los diferentes aspectos de la aplicación (lógica de entrada, lógica de negocios y lógica de la interfaz de usuario), a la vez que proporciona un vago acoplamiento entre estos elementos.
- El modelo especifica dónde se debería encontrar cada tipo de lógica en la aplicación.
 - La lógica de la interfaz de usuario pertenece a la vista.
 - La lógica de entrada pertenece al controlador.
 - La lógica de negocios pertenece al modelo.
- Esta separación ayuda a administrar la complejidad al compilar una aplicación, ya que le permite centrarse en cada momento en un único aspecto de la implementación.
- El acoplamiento vago entre los tres componentes principales de una aplicación MVC también favorece el desarrollo paralelo.

© JMA 2020. All rights reserved

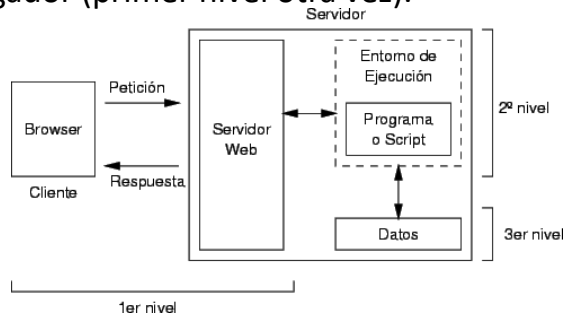
Puntos fuertes de MVC...

- Puntos fuertes de MVC...
 - Modelo muy simple de entender
 - Modelo muy cercano a la web
 - Admite una buena separación de responsabilidades
 - Permite la automatización de las pruebas unitarias de modelos y controladores.
- ... y no tan fuertes
 - Mucha menos abstracción que WebForms
 - Curva de aprendizaje más alta

© JMA 2020. All rights reserved

Arquitectura Web

- Una aplicación Web típica recogerá datos del usuario (primer nivel), los enviará al servidor, que ejecutará un programa (segundo y tercer nivel) y cuyo resultado será formateado y presentado al usuario en el navegador (primer nivel otra vez).



© JMA 2020. All rights reserved

Single-page application (SPA)

- Un single-page application (SPA), o aplicación de página única es una aplicación web o es un sitio web que utiliza una sola página con el propósito de dar una experiencia más fluida a los usuarios como una aplicación de escritorio.
- En un SPA todo el código de HTML, JavaScript y CSS se carga de una sola vez o los recursos necesarios se cargan dinámicamente cuando lo requiera la página y se van agregando, normalmente como respuesta de las acciones del usuario.
- La página no se tiene que cargar otra vez en ningún punto del proceso, tampoco se transfiere a otra página, aunque las tecnologías modernas (como el `pushState()` API del HTML5) pueden permitir la navegabilidad en páginas lógicas dentro de la aplicación.
- La interacción con las aplicaciones de página única pueden involucrar comunicaciones dinámicas con el servidor web que está por detrás, habitualmente utilizando AJAX o WebSocket (HTML5).

© JMA 2020. All rights reserved

Desarrollo del lado del cliente

- ASP.NET Core se integra perfectamente con bibliotecas y plataformas de trabajo populares del lado cliente.
- Dispone de plantillas para crear proyectos:
 - JQuery con Bootstrap.
 - Angular
 - React
 - Blazor (WebAssembly)
- ASP.NET Core admite la creación de servicios RESTful, lo que también se conoce como API Web, mediante C#. Esto permite la creación del back-end de aplicaciones SPA o AJAX y de arquitecturas de microservicios con clientes web, escritorio y móviles.

© JMA 2020. All rights reserved

Creación de proyectos

ARQUITECTURA DE UNA APLICACIÓN MVC

© JMA 2020. All rights reserved

Tipos de Proyectos

- Aplicaciones web (Razor Pages)
 - Razor Pages facilita la programación de escenarios centrados en páginas, que no necesitan la complejidad de controladores y vistas.
- Aplicaciones web (Modelo-Vista-Controlador)
 - ASP.NET Core MVC es un completo marco de trabajo para compilar aplicaciones web y API mediante el patrón de diseño Modelo-Vista-Controlador.
- API
 - ASP.NET Core admite la creación de servicios RESTful, lo que también se conoce como API Web
- SPA
 - ASP.NET Core MVC con ClientApp en Angular, React, React con Redux
- Blazor
 - Es una plataforma de trabajo para la creación de interfaces de usuario web interactivas del lado cliente con .NET usando WebAssembly

© JMA 2020. All rights reserved

Estructura de un proyecto MVC

- 📁 wwwroot: Contiene recursos estáticos, como imágenes o archivos HTML, JavaScript y CSS.
- 📁 Controllers: Contiene las clases con los controladores que responden a la entrada desde el navegador, decidir qué hacer con él y devolver la respuesta al usuario.
- 📁 Models: Contiene las clases del modelo.
- 📁 Data: Contiene las clases del EF para el mantenimiento del contexto.
- 📁 Views: Contiene las plantillas de interfaz de usuario, estructuradas en subcarpetas por controlador.
- 📁 Views/Shared: Contiene las plantillas y elementos compartidos.
- 📁 Areas: Contienen las carpetas de las áreas y sus subcarpetas (Controllers, Views, Models).
- 📄 Program.cs: Contiene un método Main, el punto de entrada administrado de la aplicación.
- 📄 Startup.cs: Configura el comportamiento de la aplicación, como el enrutamiento entre páginas.
- 📄 appsettings.json: Fichero de configuración externa, dispone de versiones appsettings.Production.json o appsettings.Development.json

© JMA 2020. All rights reserved

Clase Startup

- Las aplicaciones de ASP.NET Core utilizan una clase Startup, que se denomina Startup por convención:
 - Incluye opcionalmente un método ConfigureServices para configurar los servicios de la aplicación. Un servicio es un componente reutilizable que proporciona funcionalidades de la aplicación. Los servicios se registran en ConfigureServices y se usan en la aplicación a través de la inyección de dependencias (DI) o ApplicationServices.
 - Incluye un método Configure para crear la canalización de procesamiento de solicitudes de la aplicación.
- La clase Startup se especifica cuando se crea el host de la aplicación. Habitualmente, la clase Startup se especifica en el generador de host del main mediante una llamada al método WebHostBuilderExtensions.UseStartup<TStartup>.

© JMA 2020. All rights reserved

Inyección de dependencias (servicios)

- ASP.NET Core incluye un marco de inyección de dependencias (DI) integrado que pone a disposición los servicios configurados a lo largo de una aplicación. Por ejemplo, un componente de registro es un servicio.
- Se agrega al método Startup.ConfigureServices el código para configurar (o registrar) servicios. Por ejemplo:

```
public void ConfigureServices(IServiceCollection services) {  
    services.AddDbContext<MovieContext>(options =>  
        options.UseSqlServer(Configuration.GetConnectionString("MovieContext")));  
}
```
- Los servicios se suelen resolver desde la inyección de dependencias mediante la inyección de constructores. Con la inyección de constructores, una clase declara un parámetro de constructor del tipo requerido o una interfaz. El marco de inyección de dependencias proporciona una instancia de este servicio en tiempo de ejecución.

```
public class IndexModel : PageModel {  
    private readonly MovieContext _context;  
    public IndexModel(MovieContext context) { _context = context; }
```

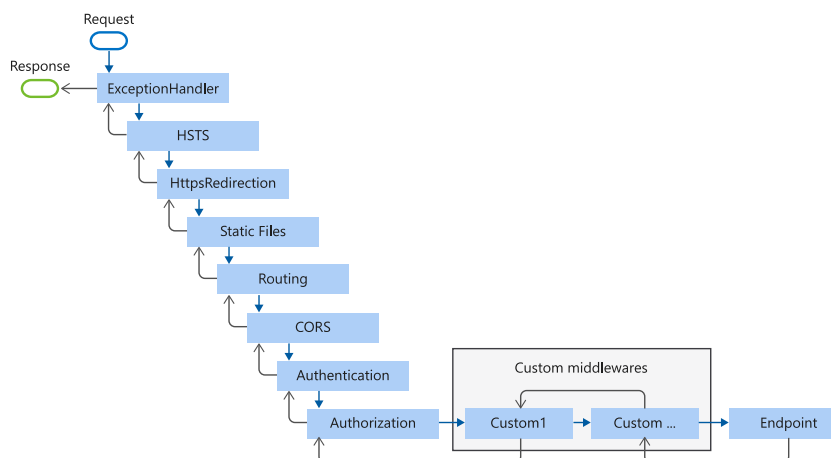
© JMA 2020. All rights reserved

Middleware

- La canalización de control de solicitudes se compone de una serie de componentes de software intermedio.
 - Cada componente lleva a cabo las operaciones en un contexto HttpContext e invoca el middleware siguiente de la canalización, o bien finaliza la solicitud.
 - Normalmente, se agrega un componente de software intermedio a la canalización al invocar un método de extensión Use... en el método Startup.Configure.
- ```
public void Configure(IApplicationBuilder app) {
 app.UseHttpsRedirection();
 app.UseStaticFiles();
 app.UseRouting();
}
```
- ASP.NET Core incluye un amplio conjunto de middleware integrado. También se pueden escribir componentes de middleware personalizados.

© JMA 2020. All rights reserved

## Orden del middleware

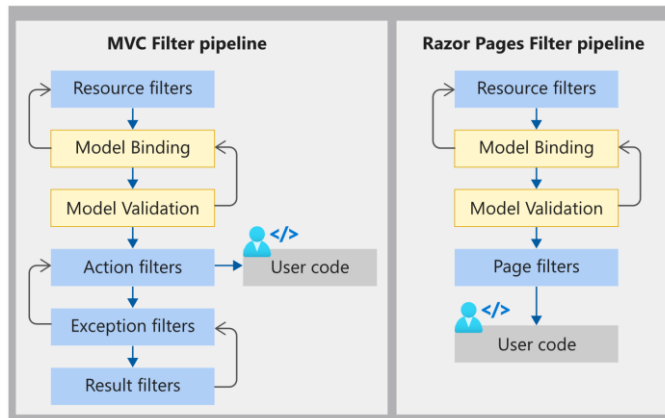


© JMA 2020. All rights reserved

# Punto de conexión

## MVC Endpoint

(called by the Endpoint Middleware)



© JMA 2020. All rights reserved

## Configuración y entornos

- ASP.NET Core proporciona un marco de configuración que obtiene la configuración como pares nombre/valor de un conjunto ordenado de proveedores de configuración. Hay disponibles proveedores de configuración integrados para una gran variedad de orígenes, como archivos .json y .xml, variables de entorno y argumentos de línea de comandos. De manera predeterminada, las aplicaciones de ASP.NET Core están configuradas para leer desde appsettings.json, variables de entorno, la línea de comandos, etc.
- Los entornos de ejecución, como Development, Staging y Production, son un concepto de primera clase en ASP.NET Core. Especificando el entorno que ejecuta una aplicación en la variable de entorno ASPNETCORE\_ENVIRONMENT, ASP.NET Core lee dicha variable de entorno al inicio de la aplicación y almacena el valor en una implementación IWebHostEnvironment. Esta implementación está disponible en cualquier parte de una aplicación a través de la inyección de dependencias (DI).

© JMA 2020. All rights reserved

# Registro y Control de errores

- ASP.NET Core es compatible con una API de registro que funciona con una gran variedad de proveedores de registro integrados y de terceros. Los proveedores disponibles incluyen:
  - Consola
  - Depuración
  - Seguimiento de eventos en Windows
  - Registro de errores de Windows
  - TraceSource
  - Azure App Service
  - Azure Application Insights
- ASP.NET Core tiene características integradas para controlar los errores, tales como:
  - Una página de excepciones para el desarrollador
  - Páginas de errores personalizados
  - Páginas de códigos de estado estáticos
  - Control de excepciones de inicio

© JMA 2020. All rights reserved

# Enrutamiento

- Una ruta es un patrón de dirección URL que se asigna a un controlador. El controlador normalmente es una página de Razor, un método de acción en un controlador MVC o un software intermedio. El enrutamiento de ASP.NET Core le permite controlar las direcciones URL usadas por la aplicación.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {
 app.UseRouting();
 app.UseEndpoints(endpoints => {
 endpoints.MapControllerRoute(
 name: "default",
 pattern: "{controller=Home}/{action=Index}/{id?}");
 endpoints.MapRazorPages();
 });
}
```
- La raíz web es la ruta de acceso base para los archivos de recursos estáticos públicos: Hojas de estilo (.css), JavaScript (.js), Imágenes (.png, .jpg), ...
- De manera predeterminada, los archivos estáticos solo se sirven desde el directorio raíz web y sus subdirectorios. La ruta de acceso raíz web se establece de manera predeterminada en {raíz del contenido}/wwwroot.
- En los archivos Razor (.cshtml), la virgulilla (~/) apunta a la raíz web. Una ruta de acceso que empieza por ~/ se conoce como ruta de acceso virtual.

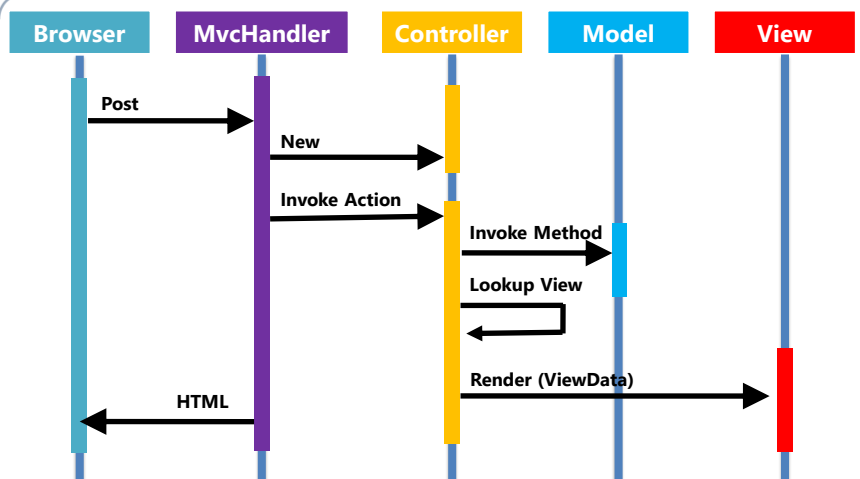
© JMA 2020. All rights reserved

## Fases de ejecución de MVC

1. Recibir la primera solicitud para la aplicación
2. Realizar el enrutamiento
3. Crear el controlador de solicitudes de MVC
4. Crear el controlador
5. Ejecutar el controlador
6. Invocar la acción
7. Ejecutar el resultado
8. Enviar el resultado

© JMA 2020. All rights reserved

## El framework en funcionamiento



© JMA 2020. All rights reserved

## Principales resultados de la acción

| Resultado             | Descripción                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------------------|
| ViewResult            | Representa una vista como una página web.                                                                        |
| PartialViewResult     | Representa una vista parcial, que define una sección de una vista que se puede representar dentro de otra vista. |
| RedirectResult        | Redirecciona a otro método de acción utilizando su URL.                                                          |
| RedirectToRouteResult | Redirecciona a otro método de acción.                                                                            |
| ContentResult         | Devuelve un tipo de contenido definido por el usuario.                                                           |
| JsonResult            | Devuelve un objeto JSON serializado.                                                                             |
| FileResult            | Devuelve la salida binaria para escribir en la respuesta.                                                        |
| EmptyResult           | Representa un valor devuelto que se utiliza si el método de acción debe devolver un resultado null (vacío).      |
| StatusCodeResult      | Proporciona un modo para devolver un resultado de la acción con un código de estado de respuesta HTTP.           |

© JMA 2020. All rights reserved

## NuGet Package Manager

- Gestor de paquetes para desarrolladores
- Se instala como extensión
- Cuenta con una amplia base de datos actualizada de paquetes
- Simplifica el uso de componentes externos.
  - Localización
  - Descarga (¡con dependencias!)
  - Instalación / desinstalación
  - Configuración
  - Actualización
- Dispone de la consola de administración de paquetes.

© JMA 2020. All rights reserved

---

Introducción

# MODELOS

---

© JMA 2020. All rights reserved

## Modelo

- El modelo es la parte de la aplicación que es responsable de la aplicación básica o la lógica de negocio.
- Los objetos de modelo normalmente obtienen acceso a los datos desde un almacén persistente, como SQL Server, y realizan la lógica de negocios en esos datos.
- Los modelos son específicos de la aplicación y, por consiguiente, el marco de ASP.NET Core MVC no impone ninguna restricción sobre los tipos de objetos de modelo que se pueden generar.

---

© JMA 2020. All rights reserved

# Opciones

- Se puede utilizar objetos DataReader o DataSet de ADO.NET, EF o puede utilizar un conjunto personalizado de objetos de dominio. También puede utilizar una combinación de tipos de objeto para trabajar con datos.
- El modelo no es de una clase o interfaz determinada.
- Una clase forma parte del modelo no porque implemente una determinada interfaz o derive de una determinada clase base, sino debido al rol que la clase representa en la aplicación ASP.NET Core MVC y dónde la clase se encuentra en la estructura de carpetas de la aplicación.
- Una clase de modelo en una aplicación ASP.NET Core MVC no administra directamente la entrada del explorador ni genera la salida HTML al explorador.

© JMA 2020. All rights reserved

# Lógica del dominio

- Los objetos del modelo son las partes de la aplicación que implementan la lógica del dominio, también conocida como la lógica de negocio.
- La lógica del dominio administra los datos que se pasan entre la base de datos y la interfaz de usuario.
- Por ejemplo, en un sistema de inventario, el modelo lleva un seguimiento de los elementos en almacenamiento y la lógica para determinar si un artículo está en inventario.
- Además, el modelo formaría parte de la aplicación que actualiza la base de datos cuando un artículo se vende y distribuye fuera del almacén.
- A menudo, el modelo también almacena y recupera el estado modelo en una base de datos.

© JMA 2020. All rights reserved



# Arquitectura

- La carpeta recomendada para colocar las clases de modelo es la carpeta Models, proporcionada por Visual Studio en la plantilla de la aplicación ASP.NET Core MVC.
- Sin embargo, también es habitual colocar las clases modelo en un ensamblado independiente, para que puedan reutilizarse estas clases en aplicaciones diferentes.
- Utilizar las clases modelo con un controlador normalmente consiste en crear instancias de las clases modelo en acciones de controlador, llamando a métodos de los objetos modelo y extrayendo los datos adecuados de estos objetos para mostrarlos en vistas.
- Este es el enfoque recomendado para implementar las acciones. También mantiene la separación entre los elementos lógicos de la aplicación, lo que facilita probar la lógica de la aplicación sin tener que probarla mediante la interfaz de usuario.

© JMA 2020. All rights reserved

Introducción

## CONTROLADORES

© JMA 2020. All rights reserved

## Fundamentos

- El marco de ASP.NET Core MVC asigna direcciones URL a las clases a las que se hace referencia como controladores.
- Los controladores procesan solicitudes entrantes, controlan los datos proporcionados por el usuario y las interacciones, y ejecutan la lógica de la aplicación adecuada.
- Una clase de controlador llama normalmente a un componente de vista independiente para generar el marcado HTML para la solicitud.

© JMA 2020. All rights reserved

## Implementación

- La clase base para todos los controladores es la clase ControllerBase, que proporciona el control general de MVC.
- La clase Controller hereda de ControllerBase y es la implementación predeterminada de un controlador.
- La clase Controller es responsable de las fases del procesamiento siguientes:
  - Localizar el método de acción adecuado para llamar y validar que se le puede llamar.
  - Obtener los valores para utilizar como argumentos del método de acción.
  - Controlar todos los errores que se puedan producir durante la ejecución del método de acción.
- Todas las clases de controlador deben residir en la carpeta Controllers y llevar el sufijo "Controller" en su nombre.

© JMA 2020. All rights reserved

# Métodos de acción

- El controlador define los métodos de acción: métodos públicos de la clase.
- Los controladores pueden incluir tantos métodos de acción como sea necesario.
- Los métodos de acción tienen normalmente una asignación unívoca con las interacciones del usuario.
- Los métodos del acción se invocan mediante solicitudes HTTP.
- Cada invocación cuenta con una dirección URL que MVC analiza usando las reglas de enrutamiento y determina la ruta de acceso del controlador.
- Con la dirección URL, el controlador determina el método de acción adecuado para administrar la solicitud.

© JMA 2020. All rights reserved

## VISTAS

© JMA 2020. All rights reserved

## Vista

- La vista se encarga de la presentación de los datos y de la interacción del usuario.
- Una vista es una plantilla HTML con marcado Razor incrustado, código que interactúa con el marcado HTML para generar una página web que se envía al cliente.
- En ASP.NET Core MVC, las vistas son archivos .cshtml que usan el lenguaje de programación C# en el marcado Razor. Por lo general, los archivos de vistas se agrupan en carpetas con el nombre del controlador de la vista. Dichas carpetas se almacenan en una carpeta llamada Views que está ubicada en la raíz de la aplicación

© JMA 2020. All rights reserved

## Vista

- Los diseños (layout o páginas maestras) permiten cohesionar las secciones de las páginas web y reducir la repetición del código. Los diseños suelen contener encabezado, elementos de menú y navegación y pie de página.
- Las vistas parciales administran las partes reutilizables de las vistas para reducir la duplicación de código.
- Los componentes de la vista se asemejan a las vistas parciales en que permiten reducir el código repetitivo, pero son adecuados para ver contenido que requiere la ejecución de código en el servidor con el fin de representar la página web.

© JMA 2020. All rights reserved

## Vista

- Una página de vista es una instancia de la clase `RazorPage<>`. Hereda de la clase `RazorPage` e implementa la interfaz `IRazorPage`.
- La clase `RazorPage` define una propiedad `ViewData` que devuelve un objeto `ViewDataDictionary`. Esta propiedad contiene los datos que la vista debe mostrar.
- Si la vista es fuertemente tipada, la propiedad `Model` contiene el modelo del tipo suministrado al genérico.

© JMA 2020. All rights reserved

## Razor

- Razor es una sintaxis de marcado para insertar código basado en servidor en páginas html.
- La sintaxis Razor consta de marcado Razor, C# y HTML.
- Los archivos que contienen Razor generalmente tienen una extensión de archivo `.cshtml`. Razor también se utiliza en archivos de componentes Razor (`.Razor`).
- La representación de HTML a partir del marcado Razor no es diferente de representar HTML desde un archivo HTML, el servidor no ha cambiado el formato HTML en los archivos `.cshtml`.
- Sintaxis compacta y limpia
  - Menos **directivas**
  - **Integración** código-marcado más suave

© JMA 2020. All rights reserved

## Sintaxis Razor (C#)

- Selector de código Razor: @
- Bloque: @{ ... }
- Comentario: @\* ... \*@
- Instrucciones terminadas en ;
- Sensible a mayúsculas y minúsculas
- Acepta constantes literales de cadenas:  
`var myFilePath = @"C:\MyFolder\";`
- Acepta nombre cualificados: @class
- Para escapar un símbolo @ en Razor se usa un segundo símbolo @.
- Operadores  
– . () [] = + - \* / += -= == != < > <= >= ! && ||

© JMA 2020. All rights reserved

## Variables

- Se declaran en un bloque
- Con tipo y sin tipo explícito
- Se recomienda la inferencia de tipos:  
`@{ var variable=4; }`
- Mostrar valores (inmerso en HTML o dentro de un bloque de código):  
`<li>@variable</li> <li>@(variable+1)</li>`  
`<li>@objeto.propiedad</li>`  
`<li>@variable.metodo(1, 1)</li>`  
`<li>@(a + b)</li>`

© JMA 2020. All rights reserved

# Conversión

- Métodos de conversión y prueba:
  - AsBool(), IsBool(), AsInt(), IsInt(), AsFloat(), IsFloat(), AsDecimal(), IsDecimal(), AsDateTime(), IsDateTime(), ToString()
- HTML Encoding automático
- Conversión automática de rutas de acceso virtuales a físicos y URLs
  - `var dataFilePath = "~/dataFile.txt";`
  - `<p>@Server.MapPath(dataFilePath)</p>`  
C:\Websites\MyWebSite\datafile.txt
  - ``

© JMA 2020. All rights reserved

# Bucles y lógica condicional

- Dentro de un bloque de código como una instrucción mas.
- Fuera de un bloque, auto comienzan el bloque:

```
@foreach (var myItem in Request.ServerVariables){
 @myItem
}
```

  - Para representar el resto de una línea completa como HTML dentro de un bloque de código, se usa la sintaxis @:
- Condicionales: if, switch
- Bucles: while, for, foreach, do while
- Error: try, catch, finally
- @using: espacio de nombre y uso

© JMA 2020. All rights reserved

## HTML Helpers

- Son pequeños métodos que nos ayudan a escribir el HTML en las vistas.
- Son definidos como métodos extensores de la clase `HtmlHelper`.
- La vista expone una propiedad de tipo `HtmlHelper` con nombre `Html`.
- Esto nos permite usar una sintaxis del tipo `Html.MiHelper(argumentos)`.
- La gracia de los helpers es que podemos crear los nuestros y usarlos siguiendo la misma sintaxis.

© JMA 2020. All rights reserved

## Arquitectónicos

| Nombre         | Descripción                                                                                                                                                      |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Action         | Invoca el método de acción secundario especificado y devuelve el resultado como una cadena HTML.                                                                 |
| ActionLink     | Devuelve un elemento delimitador (elemento a) que contiene la ruta de acceso virtual de la acción especificada.                                                  |
| RouteLink      | Devuelve un elemento delimitador (elemento a) que contiene la ruta de acceso virtual de la acción especificada.                                                  |
| Partial        | Presenta la vista parcial especificada como una cadena codificada en HTML.                                                                                       |
| RenderAction   | Invoca un método de acción secundario especificado y presenta el resultado alineado en la vista primaria.                                                        |
| RenderPartial  | Presenta la vista parcial especificada utilizando la aplicación auxiliar HTML especificada.                                                                      |
| BeginForm      | Escribe una etiqueta de apertura <code>&lt;form&gt;</code> para la respuesta. Cuando el usuario envíe el formulario, un método de acción procesará la solicitud. |
| BeginRouteForm | Escribe una etiqueta de apertura <code>&lt;form&gt;</code> para la respuesta. Cuando el usuario envíe el formulario, el destino de ruta procesará la solicitud.  |
| EndForm        | Presenta la etiqueta <code>&lt;/form&gt;</code> de cierre para la respuesta.                                                                                     |



# Plantillas

| Nombre          | Descripción                                                                                                   |
|-----------------|---------------------------------------------------------------------------------------------------------------|
| Display         | Devuelve el formato HTML para cada propiedad del objeto representado por una expresión de cadena.             |
| DisplayFor      | Devuelve el formato HTML de cada propiedad en el objeto representado por la expresión Expression.             |
| DisplayForModel | Devuelve el formato HTML para cada propiedad en el modelo.                                                    |
| Editor          | Devuelve un elemento input HTML para cada propiedad del objeto representado por la expresión.                 |
| EditorFor       | Devuelve un elemento input de HTML para cada propiedad en el objeto representado por la expresión Expression. |
| EditorForModel  | Devuelve un elemento input de HTML para cada propiedad en el modelo.                                          |

# Visualización

| Nombre         | Descripción                                                                                                |
|----------------|------------------------------------------------------------------------------------------------------------|
| Label          | Devuelve un elemento label de HTML y el nombre de la propiedad representada por la expresión especificada. |
| LabelFor       | Devuelve un elemento label de HTML y el nombre de la propiedad representada por la expresión especificada. |
| LabelForModel  | Devuelve un elemento label de HTML y el nombre de la propiedad representada por el modelo.                 |
| DisplayText    | Devuelve el formato HTML para cada propiedad del objeto representado por la expresión especificada.        |
| DisplayTextFor | Devuelve el formato HTML para cada propiedad del objeto representado por la expresión especificada.        |

## Texto

| Nombre      | Descripción                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------|
| TextArea    | Devuelve el elemento textarea especificado utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario. |
| TextAreaFor | Devuelve un elemento textarea HTML para cada propiedad del objeto representado por la expresión especificada.                       |
| TextBox     | Devuelve un elemento input de texto utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario.        |
| TextBoxFor  | Devuelve un elemento input de texto para cada propiedad del objeto representada por la expresión especificada.                      |
| Password    | Devuelve un elemento input de contraseña utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario.   |
| PasswordFor | Devuelve un elemento input de contraseña para cada propiedad del objeto representado por la expresión especificada.                 |

© JMA 2020. All rights reserved

## Selección

| Nombre          | Descripción                                                                                                                                                   |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CheckBox        | Devuelve un elemento input de casilla utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario.                                |
| CheckBoxFor     | Devuelve un elemento input de casilla para cada propiedad del objeto representado por la expresión especificada.                                              |
| RadioButton     | Devuelve un elemento input de botón de radio que se utiliza para presentar opciones mutuamente excluyentes.                                                   |
| RadioButtonFor  | Devuelve un elemento input de botón de radio para cada propiedad del objeto representado por la expresión especificada.                                       |
| DropDownList    | Devuelve un elemento select de una sola selección utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario                     |
| DropDownListFor | Devuelve un elemento select HTML para cada propiedad en el objeto representado por la expresión especificada, usando los elementos de la lista especificados. |
| ListBox         | Devuelve un elemento select de varias selecciones utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario.                    |
| ListBoxFor      | Devuelve un elemento select HTML para cada propiedad del objeto representado por la expresión especificada y usando los elementos de lista especificados.     |

© JMA 2020. All rights reserved

# Especiales

| Nombre               | Descripción                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| Hidden               | Devuelve un elemento input oculto utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario.           |
| HiddenFor            | Devuelve un elemento input oculto de HTML para cada propiedad del objeto representada por la expresión especificada.                 |
| Validate             | Recupera los metadatos de validación para el modelo especificado y aplica cada regla al campo de datos.                              |
| ValidateFor          | Recupera los metadatos de validación para el modelo especificado y aplica cada regla al campo de datos.                              |
| ValidationMessage    | Muestra un mensaje de validación si existe un error para el campo especificado en el objeto ModelStateDictionary.                    |
| ValidationMessageFor | Devuelve el formato HTML para un mensaje de error de validación para cada campo de datos representado por la expresión especificada. |
| ValidationSummary    | Devuelve una lista desordenada (elemento ul) de los mensajes de validación que están en el objeto ModelStateDictionary.              |

© JMA 2020. All rights reserved

## Ejemplos de Helpers

```
@using (Html.BeginForm()) {
 @Html.AntiForgeryToken()
 <div class="form-horizontal">
 <h4>Customer</h4>
 <hr />
 @Html.ValidationSummary(true, "", new { @class = "text-danger" })
 @Html.HiddenFor(model => model.CustomerID)
 <div class="form-group">
 @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
 <div class="col-md-10">
 @Html.EditorFor(model => model.Name)
 @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
 </div>
 </div>
 <div class="form-group">
 <div class="col-md-offset-2 col-md-10">
 <input type="submit" value="Save" class="btn btn-default" />
 </div>
 </div>
 </div>
}
<div>@Html.ActionLink("Back to List", "Index")</div>
```

© JMA 2020. All rights reserved

# Asistentes de etiquetas

- Los asistentes de etiquetas son una alternativa moderna a los HTML Helpers (no los reemplazan), se crean en C# y tienen como destino elementos HTML en función del nombre de elemento, el nombre de atributo o la etiqueta principal.
- Los asistentes de etiquetas proporcionan:
  - Una experiencia de desarrollo compatible con HTML, usan la sintaxis de los atributos o etiquetas HTML.
  - Un completo entorno de IntelliSense para crear marcado HTML y Razor
  - Una forma de ser más productivo y generar código más sólido, confiable y fácil de mantener con información que solo está disponible en el servidor

© JMA 2020. All rights reserved

## Ejemplos de asistentes

```
<div class="row">
 <div class="col-md-4">
 <form asp-action="Edit">
 <div asp-validation-summary="ModelOnly" class="text-danger"></div>
 <input type="hidden" asp-for="Id" />
 <div class="form-group">
 <label asp-for="Nombre" class="control-label"></label>
 <input asp-for="Nombre" class="form-control" />

 </div>
 <div class="form-group">
 <input type="submit" value="Save" class="btn btn-primary" />
 </div>
 </form>
 </div>
</div>
<div>
 <a asp-action="Index">Back to List
</div>
```

© JMA 2020. All rights reserved

# Asistentes de etiquetas

Atributo	Descripción
asp-controller	El nombre del controlador.
asp-action	El nombre del método de acción.
asp-area	El nombre del área.
asp-page	Nombre de la página.
asp-page-handler	Nombre del controlador de la página.
asp-route	Nombre de la ruta.
asp-route-{value}	Un valor único de ruta de dirección URL. Por ejemplo, asp-route-id="1234".
asp-all-route-data	Todos los valores de ruta.
asp-fragment	El fragmento de dirección URL.
asp-for	Vincular al modelo (establece el type en los INPUT)
asp-validation-for	Vincular al mensaje de validación del modelo
asp-validation-summary	Muestra un resumen de los errores de validación (All or ModelOnly)
asp-items	Vincular la lista con los elementos <optgroup> y <option>.

© JMA 2020. All rights reserved

# Asistentes de etiquetas

- La caché proporciona la capacidad para mejorar el rendimiento de la aplicación de ASP.NET Core al permitir almacenar en memoria caché su contenido en el proveedor de caché interno de ASP.NET Core.  
`<cache expires-after="@TimeSpan.FromSeconds(120)">  
@DateTime.Now  
</cache>`
- Para representar un componente Blazor:  
`<component type="typeof(Counter)" render-mode="ServerPrerendered" />`
- Para representar condicionalmente un contenido en función del entorno actual:  
`<environment names="Staging,Production">  
    <strong>EnvironmentName is Staging or Production</strong>  
</environment>`
- Para representar una vista parcial:  
`<partial name="_ProductViewDataPartial" for="Product" view-data="ViewData">`

© JMA 2020. All rights reserved

---

Ampliación

## MODELOS

---

© JMA 2020. All rights reserved

## Code First

- Instalación de Entity Framework Core
  - Install-Package Microsoft.EntityFrameworkCore.SqlServer
  - Install-Package Microsoft.EntityFrameworkCore.Tools
- Creación de las entidades (modelos) en la carpeta Models (nombres en singular)
- Creación de una clase de contexto de base de datos (hereda de DbContext o uno de sus herederos) en la carpeta Data, agregando un DbSet por entidad (nombres en plural)

```
public class ApplicationDbContext : IdentityDbContext {
 public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
 : base(options) {
 }
 public DbSet<Producto> Productos { get; set; }
}
```
- Generar y revisar el archivo de migración:  
Add-Migration InitialCreate
- Actualizar (o crear) la base de datos  
Update-Database

---

© JMA 2020. All rights reserved

# Database First

- Instalación de Entity Framework Core
  - Install-Package Microsoft.EntityFrameworkCore.SqlServer
  - Install-Package Microsoft.EntityFrameworkCore.Tools
- La ingeniería inversa: Database First
  - get-help scaffold-dbcontext –detailed
- Generar clases:
  - Scaffold-DbContext "Data Source=.;Initial Catalog=AdventureWorksLT2017;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False" Microsoft.EntityFrameworkCore.SqlServer -DataAnnotations -ContextDir Infrastructure.Data.UnitOfWork -Context TiendaDbContext -OutputDir Domain.Entities -Schemas SalesLT

© JMA 2020. All rights reserved

# Database First

- La cadena de conexión en appsettings.json:

```
{
 "ConnectionStrings": {
 "TiendaConnection": "Data Source=.;Initial Catalog=AdventureWorksLT2017;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
 },
}
```
- La inyección de dependencias en Startup.ConfigureServices:

```
services.AddDbContext<TiendaDbContext>(options =>
 options.UseSqlServer(
 Configuration.GetConnectionString("TiendaConnection"));
```
- Eliminar el método OnConfiguring con la cadena de conexión de la clase TiendaDbContext.

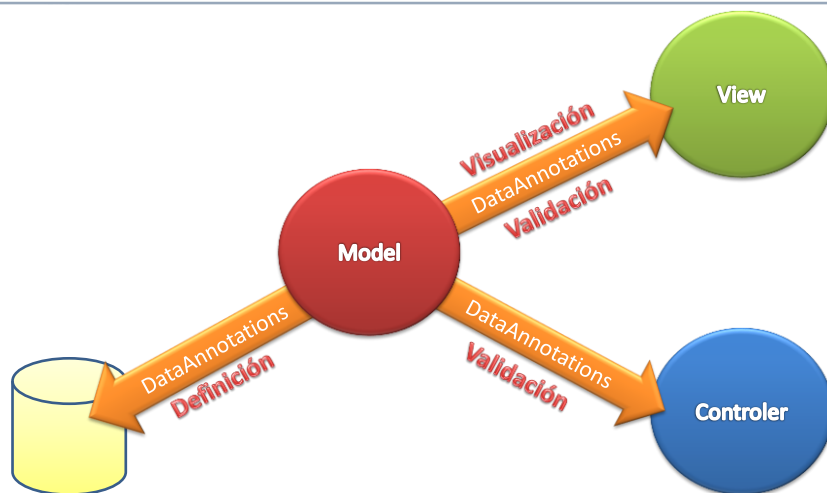
© JMA 2020. All rights reserved

# DataAnnotations

- System.ComponentModel.DataAnnotations;
- Permiten definir los metadatos de los modelos de datos para su uso por entidades externas.
- Principio DRY: No repetir
- Cuenta con decoradores para:
  - Definición del modelo de datos: claves (PK), asociaciones (FK), simultaneidad, ...
  - Interfaz de usuario y localización: Título, descripción, formato, solo lectura, ...
  - Validaciones y errores personalizados: Obligatoriedad, longitudes, formatos, ...

© JMA 2020. All rights reserved

## Contexto Declarativo



© JMA 2020. All rights reserved



# Visualización

Decorador	Descripción
DataType	Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos.
Display	Permite especificar las cadenas traducibles de los tipos y miembros de las clases parciales de entidad.
DisplayFormat	Especifica el modo en que los datos dinámicos de ASP.NET muestran y dan formato a los campos de datos.
ScaffoldColumn	Especifica si una clase o columna de datos usa la técnica scaffolding.
ScaffoldTable	Especifica si una clase o tabla de datos usa la técnica scaffolding.
UIHint	Especifica la plantilla o el control de usuario que los datos dinámicos usan para mostrar un campo de datos.
HiddenInput	(Microsoft.AspNetCore.Mvc) Indica que una propiedad se debería presentar como un elemento input oculto.

© JMA 2020. All rights reserved

## [Display]

- **Name:** valor que se usa para mostrarlo en la interfaz de usuario (Título, Etiqueta, ...).
- **Description:** valor que se usa para mostrar una descripción en la interfaz de usuario.
- **Prompt:** valor que se usará para establecer la marca de agua para los avisos en la interfaz de usuario.
- **ShortName:** valor que se usa para la etiqueta de columna de la cuadrícula.
- **GroupName:** valor que se usa para agrupar campos en la interfaz de usuario.
- **Order:** número del orden de la columna.

© JMA 2020. All rights reserved

# Tipos asociados

Asociado	Descripción
DateTime	Representa un instante de tiempo, expresado en forma de fecha y hora del día.
Date	Representa un valor de fecha.
Time	Representa un valor de hora.
Duration	Representa una cantidad de tiempo continua durante la que existe un objeto.
PhoneNumber	Representa un valor de número de teléfono.
Currency	Representa un valor de divisa.
Text	Representa texto que se muestra.
Html	Representa un archivo HTML.
MultilineText	Representa texto multilínea.
EmailAddress	Representa una dirección de correo electrónico.
Password	Representa un valor de contraseña.
Url	Representa un valor de dirección URL.
ImageUrl	Representa una URL en una imagen.
CreditCard	Representa un número de tarjeta de crédito.
PostalCode	Representa un código postal.
Upload	Representa el tipo de datos de la carga de archivos.

© JMA 2020. All rights reserved

# Validación

Decorador	Descripción
Required	Especifica que un campo de datos necesita un valor.
DataType	Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos. Decoradores especializados: CreditCard, EmailAddress, EnumDataType, FileExtensions, Phone, Url
Compare	Compara dos propiedades.
Range	Especifica las restricciones de intervalo numérico para el valor de un campo de datos.
StringLength	Especifica la longitud mínima y máxima de caracteres que se permiten en un campo de datos. Decoradores especializados: MinLength, MaxLength
RegularExpression	Especifica que un valor de campo de datos debe coincidir con la expresión regular especificada.
CustomValidation	Especifica un método de validación personalizado que se utiliza para validar una propiedad o una instancia de clase.

© JMA 2020. All rights reserved

# Validación manual de objetos

- En System.ComponentModel.DataAnnotations, la clase estática Validator ofrece métodos que permiten realizar las comprobaciones de forma directa sobre objetos o propiedades concretas.

```
IEnumerable<ValidationResult> getValidationErrors(object obj) {
 var validationResults = new List<ValidationResult>();
 var context = new ValidationContext(obj, null, null);
 Validator.TryValidateObject(obj,
 context,
 validationResults,
 true);
 return validationResults;
}
```

© JMA 2020. All rights reserved

## IValidatableObject

- Obliga a implementar un único método, llamado Validate(), que determina si el objeto especificado es válido.
- Será invocado automáticamente por TryValidateObject() siempre que no encuentre errores al comprobar las restricciones especificadas mediante anotaciones.
- Devolverá una lista de objetos ValidationResult con los resultados de las comprobaciones.
- En las clases prescriptivas (EF) se aplica con una partial class.  

```
public partial class Persona : IValidatableObject {
 public IEnumerable<ValidationResult> Validate(ValidationContext
 validationContext) {
 if (FechaNacimiento.Date.CompareTo(DateTime.Today) == 1)
 yield return new ValidationResult("Todavía no ha nacido", new[]
 { nameof(FechaNacimiento) });
 }
}
```
- Interfaces relacionados: IDataErrorInfo, INotifyDataErrorInfo

© JMA 2020. All rights reserved

## Anotar clases ya existentes

- Se requiere una clase auxiliar con las propiedades a decorar decoradas.
- Declarativa: `[MetadataType(typeof(tipo))]`
  - Se aplica con una partial class (en el mismo ensamblado que la clase a anotar).
- Imperativa: `System.ComponentModel.TypeDescriptor` y `AssociatedMetadataTypeTypeDescriptionProvider`:

```
var descriptionProvider = new
 AssociatedMetadataTypeTypeDescriptionProvider(
 typeof(Friend), typeof(FriendMetadata));
TypeDescriptor.AddProviderTransparent(
 descriptionProvider, typeof(Friend));
```

© JMA 2020. All rights reserved

## Validación del lado cliente

- La validación del lado cliente impide realizar el envío hasta que el formulario sea válido, el botón Enviar ejecuta JavaScript para enviar el formulario o mostrar mensajes de error, evitando un recorrido de ida y vuelta innecesario en el servidor cuando hay errores de entrada en un formulario.
- El script de validación discreta de jQuery es una biblioteca de front-end personalizada de Microsoft que se basa en el conocido complemento de validación de jQuery. Los asistentes de etiquetas y los HTML Helper usan los atributos de validación y escriben metadatos de las propiedades del modelo para representar atributos data- de HTML 5 para los elementos de formulario que necesitan validación.
- Se deben activar las siguientes referencias a script:
  - `_Layout.cshtml`

```
<script src="~/lib/jquery/dist/jquery.min.js"></script>
```
  - `_ValidationScriptsPartial.cshtml`

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js">
</script>
```

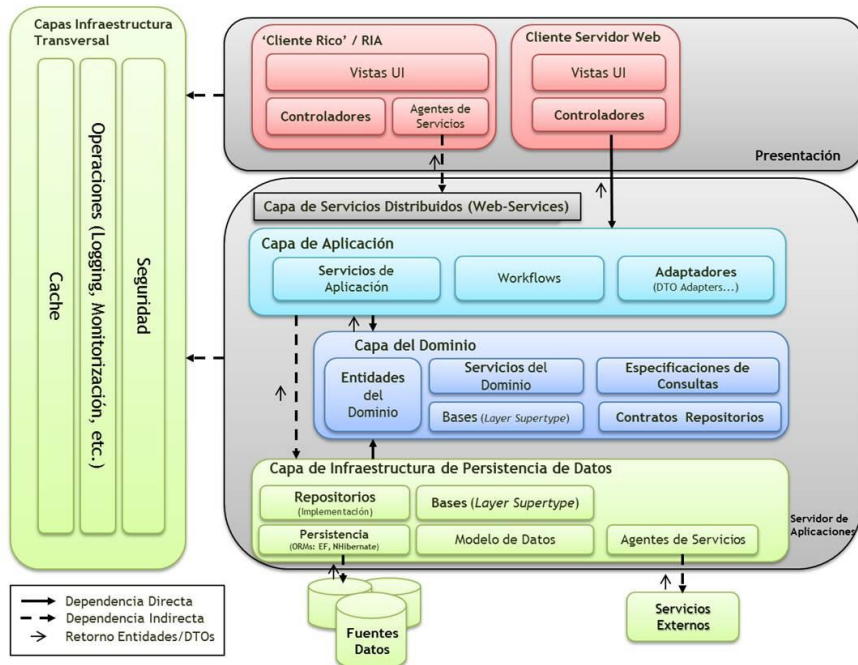
© JMA 2020. All rights reserved

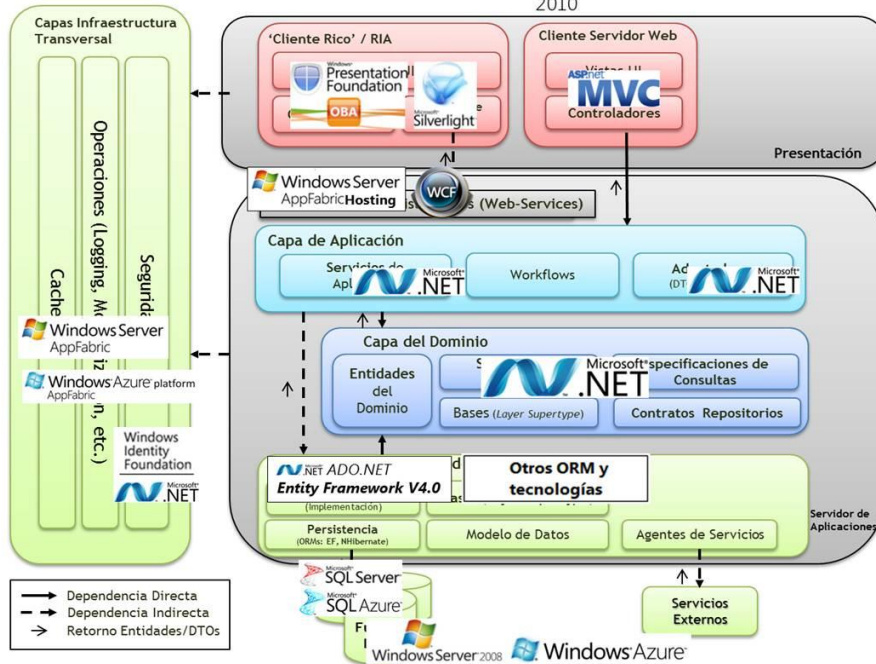
# ModelView

- Los modelos representan entidades de dominio (capa de dominio), que pueden requerir ampliaciones para su uso en capa de presentación.
- Los modelos de la capa de presentación se denominan comúnmente como ModelViews, modelos de las vistas.
- No se deben confundir con los ViewModels del patrón MVVM.
- Los ModelViews son clases (con el sufijo ModelView) que suelen encapsular al modelo de dominio (entidad) a través de una propiedad.
- Los controladores pasan los ModelViews a las vistas en sustitución de los modelos.

© JMA 2020. All rights reserved

## Arquitectura N-Capas con Orientación al Dominio





## Repositorio

- Un repositorio separa la lógica empresarial de las interacciones con la base de datos subyacente y centra el acceso a datos en un área, lo que facilita su creación y mantenimiento.
- El repositorio pertenecen a la capa de infraestructura y devuelve los objetos del modelo de dominio.
- Deberían implementar el patrón de doble herencia.
- Forma parte de los Domain Driven Design patterns: Domain Entity, Value-Object, Aggregates, Repository, Unit of Work, Specification, Dependency Injection, Inversion of Control (IoC).

# Ventajas del Repositorio

- Proporciona un punto de sustitución para las pruebas unitarias. Es fácil probar la lógica empresarial sin una base de datos y otras dependencias externas.
- Las consultas y los modelos de acceso a datos duplicados se pueden quitar y refactorizar en el repositorio.
- Los métodos del controlador pueden usar parámetros fuertemente tipados, lo que significa que el compilador encontrará errores de tipado de datos en cada compilación en lugar de realizar la búsqueda de errores de tipado de datos en tiempo de ejecución durante las pruebas.
- El acceso a datos está centralizado, lo que brinda las siguientes ventajas:
  - Mayor separación de intereses (SoC), uno de los principios de MVC, lo que facilita más el mantenimiento y la legibilidad.
  - Implementación simplificada de un almacenamiento en caché de datos centralizado.
  - Arquitectura más flexible y con menor acoplamiento, que se puede adaptar a medida que el diseño global de la aplicación evoluciona.
- El comportamiento se puede asociar a los datos relacionados (calcular campos, aplicar relaciones, reglas de negocios complejas entre los elementos de datos de una entidad, ...).
- Un modelo de dominio se puede aplicar para simplificar una lógica empresarial compleja.

© JMA 2020. All rights reserved

## DTO

- Un objeto de transferencia de datos (DTO) es un objeto que define cómo se enviarán los datos a través de la red.
- Su finalidad es:
  - Desacoplar del nivel de servicio de la capa de base de datos.
  - Quitar las referencias circulares.
  - Ocultar determinadas propiedades que los clientes no deberían ver.
  - Omitir algunas de las propiedades con el fin de reducir el tamaño de la carga.
  - Eliminar el formato de grafos de objetos que contienen objetos anidados, para que sean más conveniente para los clientes.
  - Evitar el "exceso" y las vulnerabilidades por publicación.

© JMA 2020. All rights reserved

---

Ampliación

## CONTROLADORES

---

© JMA 2020. All rights reserved

### Métodos de acción

- Todos los métodos públicos de la clase son métodos de acción
- Para evitar que un método público sea de acción se decora con [NonAction]
- Pueden tener parámetros.
- Se invocan mediante una URL (GET):
  - ~/controlador/método/VPP?NP=VP&...
  - VPP: Valor del parámetro predeterminado (opcional)
  - NP=VP: Pares Nombre Parámetro = Valor Parámetro, opcionales, el primero precedido por ? y el resto por &.

---

© JMA 2020. All rights reserved



## Parámetros de los métodos de acción

- Número de parámetros:
  - Ninguno
  - Uno, predeterminado: Por defecto “id” (definido al mapear la ruta).  
~/controlador/método/1
  - Uno, no predeterminado: Requiere el par Nombre=Valor.  
~/controlador/método/?p1=1
  - Varios: el primero puede ser el predeterminado, el resto requieren los pares Nombre=Valor  
~/controlador/método/1?p2=2&p3=3  
~/controlador/método/?p1=1&p2=2
- Los valores de los parámetros de método de acción se asignan automáticamente, se mapean por nombre.

© JMA 2020. All rights reserved

## Métodos de acción

- Por defecto los parámetros se reciben vía GET, para indicar la recepción vía POST se decoran con [HttpPost].
- El decorador [ActionName] permite asignar un nombre al método de acción distinto al del nombre del método en la clase.
- [ValidateAntiForgeryToken] previene la falsificación de solicitud entre sitios (CSRF)

© JMA 2020. All rights reserved

## Valor devuelto por el métodos de acción

- Se pueden crear métodos de acción que devuelven un objeto de cualquier tipo, como una cadena, un entero o un valor booleano.
- La mayoría de los métodos de acción devuelven una instancia que implemente IActionResult.
- La clase derivada de ActionResult determina el resultado de la acción.
- El controlador hereda métodos auxiliares que simplifican y automatizan la creación de los objetos ActionResult devueltos.
- El resultado más frecuente consiste en llamar al método View. El método View devuelve una instancia de la clase ViewResult, que se deriva de ActionResult.

© JMA 2020. All rights reserved

## Tipos derivados de ActionResult

Tipo	Método auxiliar	Descripción
ViewResult	View	Representa una vista como una página web.
PartialViewResult	PartialView	Representa una vista parcial, que define una sección de una vista que se puede representar dentro de otra vista.
RedirectResult	Redirect	Redirecciona a otro método de acción utilizando su URL.
RedirectToRouteResult	RedirectToAction RedirectToRoute	Redirecciona a otro método de acción.
StatusCodeResult	StatusCode NotFound ...	Proporciona un modo para devolver un resultado de la acción con un código de estado de respuesta HTTP.

© JMA 2020. All rights reserved

## Tipos derivados de ActionResult

Resultado	Método auxiliar	Descripción
ContentResult	Content	Devuelve un tipo de contenido definido por el usuario.
JsonResult	Json	Devuelve un objeto JSON serializado.
FileResult	File	Devuelve la salida binaria para escribir en la respuesta.
EmptyResult		Representa un valor devuelto que se utiliza si el método de acción debe devolver un resultado null (vacío).

© JMA 2020. All rights reserved

## Pasar datos a la vista

- El marco de ASP.NET Core MVC proporciona contenedores de nivel de página que pueden pasar datos entre controladores y vistas, débilmente tipados (sin tipo) o fuertemente tipados (modelo).
- La propiedad ViewData es un diccionario (claves/valor) para datos de vista.
- La propiedad ViewBag gestiona el diccionario de datos de vista dinámico (declaración al vuelo).  
`ViewBag.p = 1; → ViewData["p"] = 1;`
- Vista y controlador comparten las dos propiedades (mismo nombre y al llamar al método View del controlador se asignan a la vista)

© JMA 2020. All rights reserved

## Pasar datos fuertemente tipados

- La vista debe heredar de `RazorPage<TModel>` y establecer la propiedad `Model` (empezar por):  
`@model Domain.Entity`
- El modelo se pasa como parámetro al llamar al método `View` del controlador.
- La propiedad `ModelState` obtiene el objeto de diccionario de estados del modelo que contiene el estado del modelo y la validación de enlace del modelo.
- El modelo se puede recibir automáticamente:  
`[HttpPost][ValidateAntiForgeryToken]`  
`public ActionResult Edit(Entity ent) {`  
 `if (ModelState.IsValid) {`
- O manualmente utilizando los métodos del controlador:
  - `TryUpdateModel`: Actualiza la instancia de modelo especificada con los valores del proveedor de valores actual del controlador.
  - `TryValidateModel`: Valida la instancia de modelo especificada.

© JMA 2020. All rights reserved

## Pasar el estado entre métodos de acción

- Los métodos de acción pueden que tengan que pasar datos a otra acción, cuando se produce un error al exponer un formulario, o si el método debe redirigir a métodos adicionales.
- La propiedad `TempData` es un diccionario (claves/valor) para datos temporales.
- El valor de la propiedad `TempData` se almacena en el **estado de la sesión** (proveedor de datos temporales predeterminado) y se conserva hasta que se lea o hasta que se agote el tiempo de espera de la sesión.
- El proveedor de datos temporales se establece en `Controller.TempDataProvider`.
- Cualquier método de acción al que se llame después de establecer su valor puede obtener valores del objeto y, a continuación, procesarlos o mostrarlos.
- Conservar `TempData` de esta manera, habilita escenarios como la redirección, porque los valores de `TempData` están disponibles para más de una única solicitud.

© JMA 2020. All rights reserved

# Errores

- Para agregar el mensaje de error especificado a la colección de errores para el diccionario de modelo-estado asociado a la clave especificada.
  - ModelState.AddModelError()
- Para excepciones no tratada se pueden crear filtros de excepciones.
- Para excepciones no tratada se pueden habilitar la página en Startup.Configure:

```
if (env.IsDevelopment()) {
 app.UseDeveloperExceptionPage();
 app.UseDatabaseErrorPage();
} else {
 app.UseExceptionHandler("/Home/Error");
}
```
- Las páginas de excepciones para el desarrollador solo deben habilitarse cuando la aplicación se ejecute en el entorno de desarrollo. Es un riesgo de seguridad compartir públicamente información detallada sobre las excepciones cuando la aplicación se ejecute en producción.

© JMA 2020. All rights reserved

## Contexto de ejecución

Propiedad	Descripción
ControllerContext	Obtiene o establece el contexto del controlador.
HttpContext	Obtiene la información específica de HTTP sobre una solicitud HTTP individual.
ModelState	Obtiene el objeto de diccionario de estados del modelo que contiene el estado del modelo y la validación de enlace del modelo.
Request	Obtiene el objeto HttpRequestBase de la solicitud HTTP actual.
Response	Obtiene el objeto HttpResponseBase de la respuesta HTTP actual.
RouteData	Obtiene los datos de ruta de la solicitud actual.
TempData	Obtiene o establece el diccionario para datos temporales.
TempDataProvider	Obtiene el objeto de proveedor de datos temporales que se utiliza para almacenar los datos para la solicitud siguiente.
Url	Obtiene el objeto auxiliar de direcciones URL que se usa para generar las direcciones URL mediante el enrutamiento.
User	Obtiene la información de seguridad del usuario para la solicitud HTTP actual.

© JMA 2020. All rights reserved

# Enlazado

- Las capacidades de binding de ASP MVC conforman un potente mecanismo mediante el cual, de manera automática, se obtienen valores para los parámetros de las acciones que se ejecutan buscándolos en la petición que llega del cliente, así como las propiedades del modelo.
- El binding es el proceso que trata de emparejar los valores y parámetros enviados desde el cliente con los parámetros de las acciones que se ejecutan en el servidor.
- Una vez selecciona una acción, se analizan los nombres de los parámetros de la acción y trata de encontrar parejas entre ellos y los campos de formularios, parámetros de ruta, parámetros GET o archivos adjuntos (siguiendo el orden indicado).

© JMA 2020. All rights reserved

# Enlazado

- Por defecto, convenio de nombres:
  - Nombre GET → parámetro del método de acción
  - Nombre POST → parámetro del método de acción
- Tipos complejos:
  - Nombre POST → propiedad del parámetro
- Tipos complejos que contienen propiedades complejas:
  - Nombre POST → propiedad de la propiedad del parámetro
- Tipo lista
  - Nombre POST[n] → parámetro[n]
- Ficheros
  - Nombre POST → parámetro `HttpPostedFileBase`

© JMA 2020. All rights reserved

## Model Binders

- Extensión del framework que permite crear instancias de clases en base a valores enviados por request.
  - Al action llega el objeto instanciado y no los valores del request
- Nos permite participar del ciclo de vida de creación de la instancia permitiéndonos por ejemplo validar los atributos y agregar mensajes de error invalidando el modelo

© JMA 2020. All rights reserved

## Personalizar el enlazado

- El decorador Bind se usa para proporcionar detalles cómo debe producir el enlace del modelo a un parámetro.
  - Prefix: Prefijo que se va a usar cuando se represente el marcado para enlazar a un argumento de acción o a una propiedad compleja de modelo.
  - Exclude: Lista delimitada por comas de nombres de propiedades para las que no se permite el enlace.
  - Include: Lista delimitada por comas de nombres de propiedades para las que se permite el enlace (el resto se excluye).
- La creación de Binders personalizados (clase que implemente `IModelBinder`) proporciona un control total del enlazado. Se decora el parámetro con:  
`[ModelBinder(typeof(MiBinder))]`

© JMA 2020. All rights reserved

# Enrutamiento

- Las rutas mapean las URL a los métodos de acción de los controladores y sus parámetros.
- Separadas en fragmentos (/), compuestas por variables {...} y/o constantes.
- Ruta por defecto (Startup.Configure):

```
app.UseRouting();
app.UseEndpoints(endpoints => {
 endpoints.MapControllerRoute(
 name: "default",
 pattern: "{controller=Home}/{action=Index}/{id?}");
});
```
- Un parámetro puede ser opcional (aparecer o no):
  - Con valor por defecto: {action=Index}
  - Sin valor: {id?}

© JMA 2020. All rights reserved

## Rutas personalizadas

- Soporta tantas rutas como sean necesarias pero deben estar registradas.
- Los nombres de ruta proporcionan un nombre lógico a la ruta. La ruta con nombre se puede usar para la generación de direcciones URL.
- El orden del mapeo importa, las mas especificas primero y las mas generales las últimas (la ruta por defecto la última)
- Toda ruta debe conocer su {controller} y su {action}, en caso de no aparecer en el patrón se deben definir sus valores por defecto.

```
endpoints.MapControllerRoute(name: "blog",
 pattern: "articulos/{**articulo}",
 defaults: new { controller = "Blog", action = "Article" });
```
- MVC enlaza por nombre: nombre variable → nombre parámetro.
- El resto de las variables pueden ser:
  - Obligatorias: sin valor por defecto, deben aparecer en la URL.
  - Opcionales con valor: tienen valor por defecto, si no aparecen en la URL el parámetro toma el valor por defecto.
- Un segmento se puede dividir en varios parámetros: files/{filename}.{ext?}

© JMA 2020. All rights reserved



## Rutas personalizadas

- El asterisco \* y el asterisco doble \*\* se puede usar como prefijo de un parámetro de ruta para enlazar con el resto del URI. Debe ser el último parámetro. Con un solo \* se sustituyen los / por su escape %2F.
- Las restricciones de ruta se ejecutan cuando se busca la coincidencia y antes de que se conviertan en tokens de valores de ruta. Las restricciones se establecen a continuación del nombre separándolas con dos puntos:  
`{id:int:min(1)}`
- Las restricciones a los parámetros disponibles son:
  - `{x:alpha}` `{x:bool}` `{x:datetime}` `{x:decimal}` `{x:double}` `{x:float}` `{x:guid}` `{x:int}`  
`{x:length(6)}` `{x:length(1,20)}` `{x:long}` `{x:max(10)}` `{x:maxlength(10)}` `{x:min(10)}`  
`{x:minlength(10)}` `{x:range(10,50)}` `{x:required}` `{x:regex(^d{3}-d{3}-d{4}$)}`
- Mediante expresiones regulares se pueden establecer restricciones más complejas a los valores de las variables y se pueden crear restricciones de ruta personalizadas mediante la implementación de la interfaz `IRouteConstraint`.
- Para restringir el verbo HTTP utilizado se configuran las rutas sustituyendo el método `MapControllerRoute` por `MapGet`, `MapPost`, ...

© JMA 2020. All rights reserved

## Atributos de Enrutado

- El enrutamiento mediante atributos (anotaciones) utiliza un conjunto de atributos para asignar las plantillas de ruta directamente a controladores y acciones.
- Para activar el sondeo de atributos (si no hay rutas previas):  
`app.UseEndpoints(endpoints => { endpoints.MapControllers(); });`
- Las rutas de atributo admiten la misma sintaxis en línea que las rutas convencionales para especificar parámetros opcionales, valores predeterminados y restricciones.
- Para establecer la ruta particular para una acción:  
`[Route("reviews/{reviewId}/edit")]`  
`public ActionResult Edit(int reviewId) { ... }`
- Las rutas se pueden asociar a verbos HTTP con las especializaciones del atributo `Route`: `HttpGet`, `HttpPost`, `HttpPut`, `HttpDelete`, `HttpHead`, `HttpPatch`

© JMA 2020. All rights reserved

# Atributos de Enrutado

- Para que el enrutamiento mediante atributos sea menos repetitivo, los atributos de ruta del controlador se combinan con los atributos de ruta de las acciones individuales (la ruta en el controlador hace que todas las acciones usen el enrutamiento mediante atributos).  

```
[Route("Home")]
public class HomeController : Controller {
 [Route("")] [Route("Index")] [Route("/")]
 public IActionResult Index() { ... }
 [Route("About")]
 public IActionResult About() { ... }
```
- Las aplicaciones pueden mezclar el uso de enrutamiento convencional con el de atributos. Es habitual usar las rutas convencionales para controladores que sirven páginas HTML para los exploradores, y mediante atributos para los controladores que sirven las API de REST.

© JMA 2020. All rights reserved

# Áreas

- Las áreas son una característica de MVC que se usa para organizar la funcionalidad relacionada en un grupo independiente:
  - Espacio de nombres de enrutamiento para las acciones de controlador.
  - Estructura de carpetas para las vistas.
- El uso de áreas permite que una aplicación tenga varios controladores con el mismo nombre, siempre y cuando tengan áreas diferentes. El uso de áreas crea una jerarquía para el enrutamiento mediante la adición del parámetro de ruta "área", a controller y action.  

```
app.UseEndpoints(endpoints => {
 endpoints.MapControllerRoute(name: "areas",
 pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}");
});
```
- El atributo `[Area]` es lo que denota que un controlador es parte de un área y, sin el, los controladores no son miembros de ningún área y no coinciden cuando el enrutamiento proporciona el valor de área en la ruta.  

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

© JMA 2020. All rights reserved

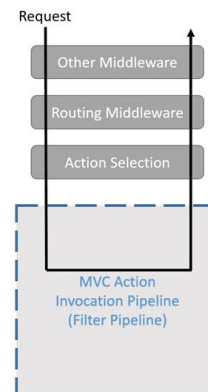
# Filtrado

- Los controladores definen métodos de acción que normalmente tienen una relación uno a uno con las posibles interacciones del usuario, como hacer clic en un vínculo o enviar un formulario.
- A veces se desea ejecutar la lógica antes de llamar a un método de acción o después de ejecutar un método de acción.
- Los filtros son clases personalizadas que proporcionan un método declarativo y de programación para agregar el comportamiento previo y posterior a la acción a los métodos de acción del controlador.

© JMA 2020. All rights reserved

# Filtros

- Los filtros en ASP.NET Core permiten que se ejecute el código antes o después de determinadas fases de la canalización del procesamiento de la solicitud.
- Se pueden crear filtros personalizados que se encarguen de cuestiones transversales, como el control de errores, el almacenamiento en caché, la configuración, la autorización y el registro. Los filtros evitan la duplicación de código. Así, por ejemplo, un filtro de excepción de control de errores puede consolidar el control de errores.
- Los filtros se ejecutan dentro de la canalización de invocación de acciones de ASP.NET Core, a veces denominada canalización de filtro.
- La canalización de filtro se ejecuta después de que ASP.NET Core seleccione la acción que se va a ejecutar.



© JMA 2020. All rights reserved

# Tipos de filtros

- Filtros de autorización: se ejecutan en primer lugar y sirven para averiguar si el usuario está autorizado para realizar la solicitud, pueden cortocircuitar la canalización si una solicitud no está autorizada.
- Filtros de recursos:
  - Se ejecutan después de la autorización.
  - OnResourceExecuting ejecuta código antes que el resto de la canalización del filtro.
  - OnResourceExecuted ejecuta el código una vez que el resto de la canalización se haya completado.
- Filtros de acción:
  - Ejecutan código inmediatamente antes y después de llamar a un método de acción.
  - Pueden cambiar los argumentos pasados a una acción.
  - Pueden cambiar el resultado devuelto de la acción.
  - No se admiten en páginas Razor.
- Filtros de excepciones: aplican directivas globales a las excepciones no controladas que se producen antes de que se escriba el cuerpo de respuesta.
- Filtros de resultados: ejecutan código inmediatamente antes y después de la ejecución de resultados de acción. Se ejecutan solo cuando el método de acción se ha ejecutado correctamente y son útiles para la lógica que debe regir la ejecución de la vista o el formateador.

© JMA 2020. All rights reserved

## Ámbito y Orden de ejecución de filtros

- Un filtro se puede agregar a la canalización en uno de tres ámbitos posibles:

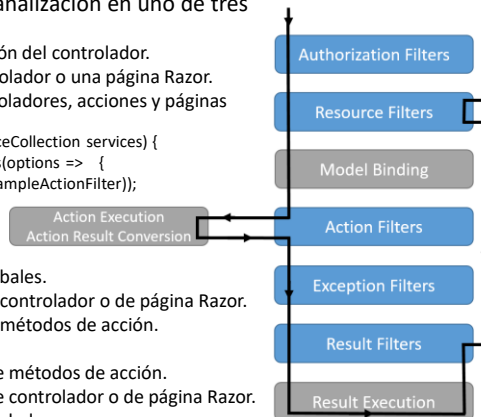
- Mediante un atributo en una acción del controlador.
- Mediante un atributo en un controlador o una página Razor.
- Globalmente para todos los controladores, acciones y páginas

Razor:

```
public void ConfigureServices(IServiceCollection services) {
 services.AddControllersWithViews(options => {
 options.Filters.Add(typeof(MySampleActionFilter));
 });
}
```

- La secuencia de ejecución:

1. El código anterior de los filtros globales.
2. El código anterior de los filtros de controlador o de página Razor.
3. El código anterior de los filtros de métodos de acción.
4. Código de la acción.
5. El código posterior de los filtros de métodos de acción.
6. El código posterior de los filtros de controlador o de página Razor.
7. El código posterior de los filtros globales.



© JMA 2020. All rights reserved

# Ciclo de ejecución de una acción

- Sincrona:
  - OnActionExecuting: Se le llama antes de invocar al método de acción.
    - Se puede implementar a través de un filtro global, de controlador o de acción.
    - Se puede sobrescribir en el controlador, con ámbito de controlador.
  - OnActionExecuted: Se le llama después de ejecutar el método de acción.
    - Se puede implementar a través de un filtro global, de controlador o de acción.
    - Se puede sobrescribir en el controlador, con ámbito de controlador.
- Asíncrona:
  - OnActionExecuting: Se le llama antes de invocar asincrónicamente al método de acción, el callback next se invoca después de ejecutar el método de acción.
    - Se puede implementar a través de un filtro global, de controlador o de acción.
    - Se puede sobrescribir en el controlador, con ámbito de controlador.

© JMA 2020. All rights reserved

## Algunos filtros proporcionados en ASP.NET Core MVC

- **Authorize:** Restringe el acceso mediante autenticación y opcionalmente mediante autorización.
- **AllowAnonymous:** Marca controladores y acciones para omitir Authorize durante la autorización.
- **RequireHttps:** Obliga a reenviar las solicitudes HTTP no seguras sobre HTTPS.
- **AcceptVerbs:** Especifica a qué verbos HTTP responderá un método de acción (Con **HttpGet**, **HttpPost**, **HttpPut**, **HttpHead**, **HttpPatch**, **HttpOptions** y **HttpDelete** el método solo administra las solicitudes HTTP del verbo).
- **ValidateAntiForgeryToken:** Impide la falsificación de una solicitud.
- **IgnoreAntiforgeryToken:** Ignora la validación contra la falsificación de una solicitud.
- **RequestSizeLimit:** Limita el tamaño máximo del cuerpo de una solicitud:

© JMA 2020. All rights reserved

# Inversión de Control

- Inversión de control (Inversion of Control en inglés, IoC) es un concepto junto a unas técnicas de programación:
  - en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales,
  - en los que la interacción se expresaba de forma imperativa haciendo llamadas a procedimientos (procedure calls) o funciones.
- Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones.
- En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir.

© JMA 2020. All rights reserved

# Inyección de Dependencias

- Inyección de Dependencias (en inglés Dependency Injection, DI) es un patrón de arquitectura orientado a objetos, en el que se inyectan objetos a una clase en lugar de ser la propia clase quien cree el objeto, básicamente recomienda que las dependencias de una clase no sean creadas desde el propio objeto, sino que sean configuradas desde fuera de la clase.
- La inyección de dependencias (DI) procede del patrón de diseño más general que es la Inversión de Control (IoC).
- Al aplicar este patrón se consigue que las clases sean independientes unas de otras e incrementando la reutilización y la extensibilidad de la aplicación, además de facilitar las pruebas unitarias de las mismas mediante la suplantación.

© JMA 2020. All rights reserved

# Inyección en ASP.NET Core

- ASP.NET Core admite el patrón de diseño de software de inyección de dependencias (DI), que es una técnica para conseguir la inversión de control (IoC) entre clases y sus dependencias. Una dependencia es un objeto (servicio) del que depende otro objeto.
- El ciclo de vida de la inyección de dependencias es:
  - registrar, resolver y eliminar
- El marco de ASP.NET Core permite administrar el ciclo mediante el registro, resolución y eliminación de dependencia, facilitando el uso de la inyección de dependencia en las aplicaciones.
- Permite dos tipos de inyección
  - inyección del constructor (es la más frecuente)
  - inyección de la llamada a un método
- Para casos excepcionales, permite la resolución manual de dependencias.

© JMA 2020. All rights reserved

## Registro

- Los servicios se pueden insertar en el constructor Startup y en el método Startup.Configure.
- Los servicios se pueden registrar con una de las siguientes duraciones:
  - Transitorio (AddTransient): se crean cada vez que el contenedor del servicio los solicita, no se cachea, indicada para servicios sin estado ligeros.
  - Con ámbito (AddScoped): se crean una vez por solicitud del cliente (conexión), indicada para servicios con estado por cliente, se eliminan al final de la solicitud.
  - Singleton (AddSingleton): se crean la primera vez que se solicitan (o al proporcionar una instancia en el registro, rara vez es necesario), instancia única para toda la aplicación y todas las solicitudes de cliente, deben ser seguros para los subprocesos y se suelen usar en servicios sin estado.
- El marco ASP.NET Core usa una convención para registrar un grupo de servicios relacionados: un único método de extensión Add{GROUP\_NAME} para registrar todos los servicios requeridos por una característica del marco. Por ejemplo, el método de extensión AddControllers registra los servicios necesarios para los controladores MVC.

© JMA 2020. All rights reserved

# Registro

- Aunque se puede registrar un tipo concreto, se recomienda el uso de interfaces o clases base para abstraer la implementación de las dependencias.

```
public void ConfigureServices(IServiceCollection services) {
 // Tipo concreto
 services.AddSingleton<MyClass>();
 // Interfaz
 services.AddSingleton<IMyDependency, MyClass>();
 // Instancia concreta
 services.AddSingleton<IMyDependency>(new MyClass(...));
 services.AddSingleton(new MyClass(...));
 // Duraciones
 services.AddTransient<IOperationTransient, Operation>();
 services.AddScoped<IOperationScoped, Operation>();
 services.AddSingleton<IOperationSingleton, Operation>();
}
```

© JMA 2020. All rights reserved

# Resolver

- Los servicios se agregan como un parámetro del constructor de la clase y el marco de trabajo asume la responsabilidad de crear u obtener una instancia de la dependencia y de desecharla cuando ya no es necesaria.

```
public class HomeController : Controller {
 private readonly IMyDependency _myDependency;
 public HomeController(IMyDependency myDependency) {
 _myDependency = myDependency;
 }
}
```

- Mediante el uso del patrón de DI, el controlador:
  - No usa el tipo concreto, solo la interfaz que implementa. Esto facilita el cambio de la implementación que el controlador utiliza sin modificar el controlador.
  - No crea una instancia, la crea el contenedor de DI.
- No es raro usar la inyección de dependencias de forma encadenada. Una dependencia solicitada puede, a su vez, solicitar sus propias dependencias. El contenedor resuelve las dependencias del grafo y devuelve la instancia totalmente formada.

© JMA 2020. All rights reserved



# Resolver

- El atributo [FromServices] permite la inyección de un servicio directamente en un método de acción sin usar la inyección de constructores:  
`public IActionResult About([FromServices] IMyDependency myDependency) {`
- Para la inyección de servicios desde el `main()` es necesario crear un elemento `IServiceScope` con `IServiceScopeFactory.CreateScope` para resolver un servicio con ámbito dentro del ámbito de la aplicación. Este método resulta útil para tener acceso a un servicio con ámbito durante el inicio para realizar tareas de inicialización.

```
public class Program {
 public static void Main(string[] args) {
 var host = CreateHostBuilder(args).Build();
 using (var serviceScope = host.Services.CreateScope())
 {
 var services = serviceScope.ServiceProvider;
 try {
 var myDependency = services.GetRequiredService<IMyDependency>();
 myDependency.WriteMessage("Call services from main");
 } catch (Exception ex) {
 var logger = services.GetRequiredService<ILogger<Program>>();
 logger.LogError(ex, "An error occurred.");
 }
 }
 host.Run();
 }
 public static IHostBuilder CreateHostBuilder(string[] args) =>
 Host.CreateDefaultBuilder(args).ConfigureWebHostDefaults(webBuilder => {
 webBuilder.UseStartup<Startup>();
 });
}
```

© JMA 2020. All rights reserved

## PÁGINAS MAESTRAS, PLANTILLAS Y CSS

© JMA 2020. All rights reserved

## Página Maestra

- Contiene las partes comunes de todas las paginas (concepto de layout).
- La vista es parte del cuerpo y se introduce a través de `@RenderBody()`
- Por defecto esta establecido en `/Views/_ViewStart.cshtml`  
`@{ Layout = "~/Views/Shared/_Layout.cshtml"; }`
- Para establecerlo a nivel de vista:
  - `@{ this.Layout = "...URL..."; }`
  - `@{ this.Layout = null; }`

© JMA 2020. All rights reserved

## Partes de Vistas

- Comúnmente conocidas como vistas parciales.
- Contienen un fragmento reutilizable de la vista: Etiquetas Razor y marcas HTML.
- Se almacenan en ficheros separados.
- Para facilitar su identificación, habitualmente su nombre va precedido por `_`
- No deben gestionar la página maestra (layout)
- Se insertan en la vista con:
  - `<partial name="_MyPartial" />`
  - `@Html.Partial("_MyPartial")`
  - `@await Html.PartialAsync("_PartialName")`
  - `@Html.Action("MetodoAcción")`
    - Marcado con `[ChildActionOnly]` y `return PartialView(...);`

© JMA 2020. All rights reserved

## Secciones

- Las secciones establecen puntos de representación en la página maestra que se diseñan en las vistas.
- Pueden ser opcionales u obligatorias
- En la página maestra:

```
@RenderSection("featured", required: false)
@if (IsSectionDefined("featured"))
```
- En la vista:

```
@section featured {
...
}
```
- O la página maestra puede incluir vistas parciales:

```
@Html.Partial("_LoginPartial")
```

© JMA 2020. All rights reserved

## Entornos

- El asistente de etiquetas de entorno representa condicionalmente el contenido incluido en función del entorno de hospedaje actual.

```
<environment include="Development">
 <script src="~/lib/jquery/dist/jquery.js"></script>
 <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.js">
</script>
 <script src="~/js/site.js" asp-append-version="true"></script>
</environment>
<environment exclude="Development">
 <script src="~/lib/vendors.bundle.min.js"></script>
 <script src="~/js/site.min.js" ></script>
</environment>
```

© JMA 2020. All rights reserved

# Plantillas

- Representación visual (HTML) de los tipos de datos.
- Plantillas específicas:
  - `<ControllerName>/DisplayTemplates/<TemplateName>.cshtml`  
Utilizadas por los extensores Display de HtmlHelper
  - `<ControllerName>/EditorTemplates/<TemplateName>.cshtml`  
Utilizadas por los extensores Editor de HtmlHelper
- Plantillas comunes:
  - `Shared/DisplayTemplates/<TemplateName>.cshtml`
  - `Shared/EditorTemplates/<TemplateName>.cshtml`
- La plantilla utilizada viene determinada por el `TemplateName`, que representa el nombre de un:
  - Tipo específico: Tipo de datos de la propiedad.
  - Tipo asociado: Modifica con `[DataType(...)]` el tipo de la propiedad.
  - Plantilla: Se establece con `[UIHint("...")]` en la propiedad.
  - Modelo: Tipo de la clase Model (`DataTemplate`).

© JMA 2020. All rights reserved

## Tipos asociados

Asociado	Descripción
Custom	Representa un tipo de datos personalizado.
DateTime	Representa un instante de tiempo, expresado en forma de fecha y hora del día.
Date	Representa un valor de fecha.
Time	Representa un valor de hora.
Duration	Representa una cantidad de tiempo continua durante la que existe un objeto.
PhoneNumber	Representa un valor de número de teléfono.
Currency	Representa un valor de divisa.
Text	Representa texto que se muestra.
Html	Representa un archivo HTML.
MultilineText	Representa texto multilínea.
EmailAddress	Representa una dirección de correo electrónico.
Password	Representa un valor de contraseña.
Url	Representa un valor de dirección URL.
ImageUrl	Representa una URL en una imagen.
CreditCard	Representa un número de tarjeta de crédito.
PostalCode	Representa un código postal.
Upload	Representa el tipo de datos de la carga de archivos.

© JMA 2020. All rights reserved

# EditorTemplates

```
@model DateTime
@{
 List<SelectListItem> days = new List<SelectListItem>();
 for (int i = 1; i <= 31; i++) {
 days.Add(new SelectListItem() { Text = i.ToString(), Value = i.ToString(), Selected = (i == Model.Day ? true : false) });
 }
 List<SelectListItem> months = new List<SelectListItem>();
 for (int i = 1; i <= 12; i++) {
 months.Add(new SelectListItem() { Text = i.ToString(), Value = i.ToString(), Selected = (i == Model.Month ? true : false) });
 }
 List<SelectListItem> years = new List<SelectListItem>();
 int prevYearCount = ViewBag.PreviousYearCount ?? 100;
 int nextYearCount = ViewBag.NextYearCount ?? 0;
 for (int i = Model.Year - prevYearCount; i <= Model.Year + nextYearCount; i++) {
 years.Add(new SelectListItem() { Text = i.ToString(), Value = i.ToString(), Selected = (i == Model.Year ? true : false) });
 }
}
@Html.DropDownList("days", days) @Html.DropDownList("months", months) @Html.DropDownList("years", years)
```

© JMA 2020. All rights reserved

RWD – Responsive Web Design

## DISEÑO WEB ADAPTATIVO

© JMA 2020. All rights reserved

# Diseño Adaptativo

- Es un enfoque de diseño destinado a la elaboración de sitios/aplicaciones para proporcionar un entorno óptimo de:
  - Lectura Fácil
  - Navegación correcta con un número mínimo de cambio de tamaño
  - Planificaciones y desplazamientos
- Con la irrupción multitud de nuevos dispositivos y el que el acceso a internet se realiza ya mayoritariamente desde dispositivos diferentes a los tradicionales ordenadores ha obligado a seguir dicho enfoque en las aplicaciones WEB.
- Contempla la definición de múltiples elementos antes de la realización de la programación real.
  - Elementos de página en las unidades de medidas correctas
  - Imágenes flexibles
  - Utilización de CSS dependiendo de la aplicación

© JMA 2020. All rights reserved

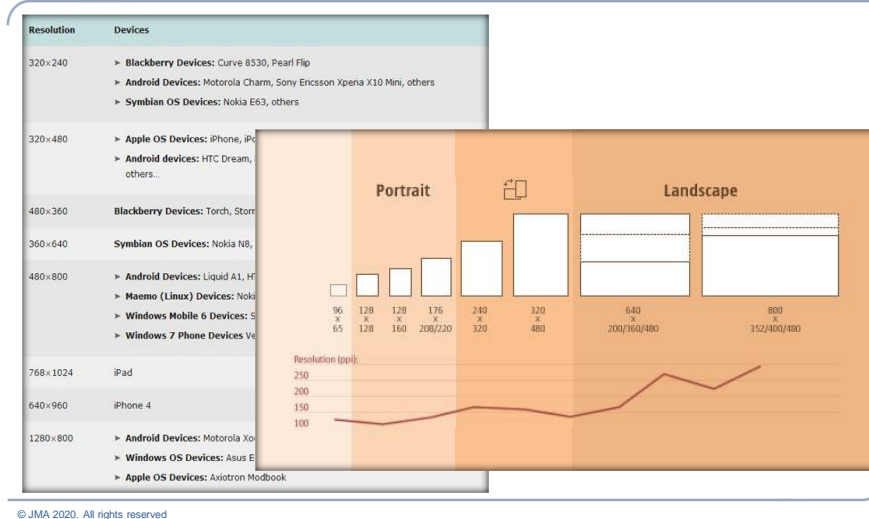
# Resolución

- Los dispositivos móviles tienen una característica distintiva, y es su resolución de pantalla.
- Es necesario conocer cuales son las resoluciones más comunes en este tipo de dispositivos móviles, de los gadgets más utilizados, etc.
- Las resoluciones van cambiando de forma muy rápida y en dispositivos nuevos



© JMA 2020. All rights reserved

# Resolución



© JMA 2020. All rights reserved

# Resolución

- También deberemos tener en cuenta la resoluciones de otros dispositivos como:
  - Tablets
  - TV SmartTV
  - Pizarras electrónicas, etc



**Pizarras** 10.1" y 11.6" (2560x1440, 1920x1080, 1366x768), 17" (1920x1080)

**PC** 12" (1280x800), 14" (1920x1080, 1366x768), 15.6" (1920x1080)

**Family hub** 23" (1920x1080), 27" (2560x1440)

© JMA 2020. All rights reserved

# Orientación de Página

- La orientación del papel es la forma en la que una página rectangular está orientada y es visualizada.
- Los dos tipos más comunes son:
  - Landscape (Horizontal)
  - Portrait (Vertical)



© JMA 2020. All rights reserved

## Recomendaciones de Diseño

1. Utilizar porcentajes y “ems” como unidad de medida en lugar de utilizar los valores determinados como definición de pixel.

**Las ems son unidades relativas,**  
así que más exactamente 1 em equivale  
al cien por cien del tamaño inicial de  
fuente.

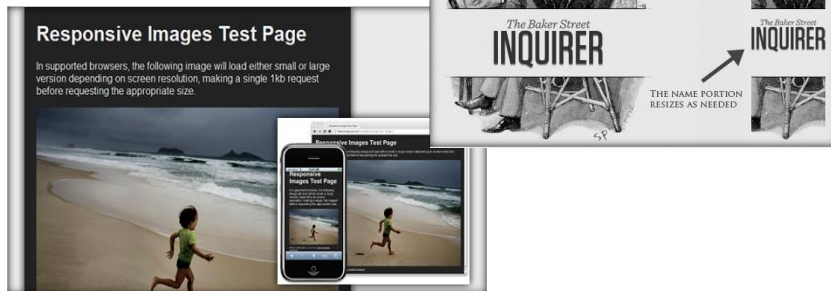
© JMA 2020. All rights reserved



## Recomendaciones de Diseño

### 2. Determinar el tamaño y definición de las imágenes a utilizar

- Recortar, Ajustar, etc



© JMA 2020. All rights reserved

## Recomendaciones de Diseño

### 3. El contenido y funcionalidad BÁSICA debe de ser accesible por todos los navegadores



© JMA 2020. All rights reserved

## Recomendaciones de Diseño

### 4. Definición correcta de contenidos en función del dispositivo



© JMA 2020. All rights reserved

## Ventajas

- Soporte de dispositivos móviles.
- Con una sola versión en HTML y CSS se cubren todas las resoluciones de pantalla.
- Mejora la experiencia de usuario.
- Se reducen los costos de creación y mantenimiento cuando el diseño de las pantallas es similar entre dispositivos de distintos tamaños.
- Evita tener que desarrollar aplicaciones específicas para cada sistema operativo móvil.
- Facilita la referenciación y posicionamiento en buscadores, versión única contenido/página.

© JMA 2020. All rights reserved

# Representación adaptable

- Etiqueta meta de viewport  
`<meta name="viewport" content="width=device-width" />`
- Posicionamiento relativo (flujo) frente a absoluto.
- Unidades relativas frente a fijas.
- Consultas de medios de CSS 3  
`@media only screen and (max-width: 850px) {  
 header { float: none; }  
}`
- Filtros de propiedades por navegador en CSS3:  
`-webkit-box-sizing: border-box;`

© JMA 2020. All rights reserved

REpresentational State Transfer

## SERVICIOS RESTFUL

© JMA 2020. All rights reserved

# REST (REpresentational State Transfer)

- Un **estilo de arquitectura** para desarrollar aplicaciones web distribuidas que se basa en el uso del protocolo HTTP e Hypermedia.
- Definido en el 2000 por Roy Fielding, para no reinventar la rueda, se basa en aprovechar lo que ya estaba definido en el HTTP pero que no se utilizaba.
- El HTTP ya define 8 métodos (algunas veces referidos como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado:
  - HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT
- El HTTP permite en el encabezado transmitir la información de comportamiento:
  - Accept, Content-type, Response (códigos de estado), Authorization, Cache-control, ...

© JMA 2020. All rights reserved

## Uso de la cabecera

- **Request:** Método /uri?parámetros
  - GET: Recupera el recurso
    - Todos: Sin parámetros
    - Uno: Con parámetros
  - POST: Crea un nuevo recurso
  - PUT: Edita el recurso
  - DELETE: Elimina el recurso
- **Accept:** Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- **Content-type:** Indica en que formato está codificado el cuerpo, utilizando MIME
- **Response:** Código de estado con el que el servidor informa del resultado de la petición.

© JMA 2020. All rights reserved

# Peticiones

Request: GET /users  
Response: 200  
content-type:application/json  
Request: GET /users/11  
Response: 200  
content-type:application/json  
Request: POST /users  
Response: 201 Created  
content-type:application/json  
body  
Request: PUT /users/11  
Response: 200  
content-type:application/json  
body  
Request: DELETE /users/11  
Response: 204 No Content

*Fake Online REST API  
for Testing and Prototyping*  
<https://jsonplaceholder.typicode.com/>

© JMA 2020. All rights reserved

## Patrón Agregado (Aggregate)

- Una Agregación es un grupo de objetos asociados que deben tratarse como una unidad a la hora de manipular sus datos.
- El patrón Agregado es ampliamente utilizado en los modelos de datos basados en Diseños Orientados al Dominio (DDD).
- Proporciona un forma de encapsular nuestras entidades y los accesos y relaciones que se establecen entre las mismas de manera que se simplifique la complejidad del sistema en la medida de lo posible.
- Cada Agregación cuenta con una Entidad Raíz (root) y una Frontera (boundary):
  - La Entidad Raíz es una Entidad contenida en la Agregación de la que colgarán el resto de entidades del agregado y será el único punto de entrada a la Agregación.
  - La Frontera define qué está dentro de la Agregación y qué no.
- La Agregación es la unidad de persistencia, se recupera toda y se almacena toda.

© JMA 2020. All rights reserved

---

# SERVICIOS WEB API

---

© JMA 2020. All rights reserved

## Introducción

---

- ASP.NET Core admite la creación de servicios RESTful, lo que también se conoce como API Web, mediante C#. Para gestionar las solicitudes, una API Web usa controladores.
- Los controladores de una API Web son clases que se derivan de ControllerBase, mediante la cual podemos crear una auténtica capa de servicios REST de manera muy similar a como veníamos desarrollando los controladores hasta ahora.
- Es posible construir una capa de servicios simplemente retornando un JsonResult en las distintas acciones de los controladores.
- La Web API se encarga de realizar este trabajo de forma específica, ofreciendo unas funcionalidades más avanzadas y apropiadas para crear este tipo de capas de servicios de tipo REST.

```
[ApiController]
```

```
[Route("api/[controller]")]
```

```
public class AlumnosController : ControllerBase
```

---

© JMA 2020. All rights reserved

# Verbos HTTP

- No se basan en los métodos de acción, se basa en los verbos HTTP (8 verbos):
  - GET: Pide una representación del recurso especificado.
  - HEAD: Pide una representación del recurso especificado.
  - POST: Envía los datos en el cuerpo de la petición para que sean procesados.
  - PUT: Sube, carga o realiza un upload a un recurso especificado.
  - DELETE: Borra el recurso especificado.
  - TRACE: Solicita al servidor que envíe de vuelta en el mensaje completado el cuerpo de entidad con la traza de la solicitud.
  - OPTIONS: Devuelve los métodos HTTP que el servidor soporta para un URL específico.
  - CONNECT: Se utiliza para saber si se tiene acceso a un host.

© JMA 2020. All rights reserved

# RESTful - CRUD

- Acrónimo de Create, Read, Update, Delete (altas, bajas, modificaciones y consultas)
- Se debe crear un método de acción por cada verbo salvo la consulta que requiere dos métodos: uno sin parámetros para obtener todos y otro con parámetro para obtener solo uno.
- La correspondencia entre verbos HTTP y métodos se puede realizar de dos formas según se usen:
  - Prefijos: El nombre del método comienza con el nombre del verbo que implementa.
  - Decoradores: Hay libertad en la elección del nombre del método pero deben ir precedidos por el decorador del correspondiente verbo.

© JMA 2020. All rights reserved

## Verbos y métodos

Acción	Verbo	Prefijo	Decorador	Parámetros	Retorno
Consulta todos	GET	Get	HttpGet	Ninguno	IEnumerable<Entidad> ó HttpResponseException
Consulta uno	GET	Get	HttpGet	Id(Clave)	Entidad ó HttpResponseException
Añadir	POST	Post	HttpPost	Entidad	HttpResponseMessage con HttpStatusCode
Modificar	PUT	Put	HttpPut	Entidad	HttpResponseMessage con HttpStatusCode
Borrar	DELETE	Delete	HttpDelete	Id(Clave)	HttpResponseMessage con HttpStatusCode

© JMA 2020. All rights reserved

## Seguridad

- La ejecución de aplicaciones JavaScript puede suponer un riesgo para el usuario que permite su ejecución.
- Por este motivo, los navegadores restringen la ejecución de todo código JavaScript a un entorno de ejecución limitado.
- Las aplicaciones JavaScript no pueden establecer conexiones de red con dominios distintos al dominio en el que se aloja la aplicación JavaScript.
- Los navegadores emplean un método estricto para diferenciar entre dos dominios ya que no permiten ni subdominios ni otros protocolos ni otros puertos.
- Si el código JavaScript se descarga desde la siguiente URL:  
<http://www.ejemplo.com>
- Las funciones y métodos incluidos en ese código no pueden acceder a:
  - <https://www.ejemplo.com/scripts/codigo2.js>
  - <http://www.ejemplo.com:8080/scripts/codigo2.js>
  - <http://scripts.ejemplo.com/codigo2.js>
  - <http://192.168.0.1/scripts/codigo2.js>
- La propiedad document.domain se emplea para permitir el acceso entre subdominios del dominio principal de la aplicación.

© JMA 2020. All rights reserved



# CORS

- Un recurso hace una solicitud HTTP de origen cruzado cuando solicita otro recurso de un dominio distinto al que pertenece y, por razones de seguridad, los exploradores restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script.
- XMLHttpRequest sigue la política de mismo-origen, por lo que, una aplicación usando XMLHttpRequest solo puede hacer solicitudes HTTP a su propio dominio. Para mejorar las aplicaciones web, los desarrolladores pidieron a los proveedores de navegadores que permitieran a XMLHttpRequest realizar solicitudes de dominio cruzado.
- El Grupo de Trabajo de Aplicaciones Web del W3C recomienda el nuevo mecanismo de Intercambio de Recursos de Origen Cruzado (CORS, Cross-origin resource sharing). Los servidores deben indicar al navegador mediante cabeceras si aceptan peticiones cruzadas y con que características:
  - "Access-Control-Allow-Origin", "\*"
  - "Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept"
  - "Access-Control-Allow-Methods", "GET, POST, PUT, DELETE"
- Soporte: Chrome 3+ Firefox 3.5+ Opera 12+ Safari 4+ Internet Explorer 8+

© JMA 2020. All rights reserved

## Desactivar la seguridad de Chrome

- Pasos para Windows:
  - Localizar el acceso directo al navegador (icono) y crear una copia como "Chrome Desarrollo".
  - Botón derecho -> Propiedades -> Destino
  - Editar el destino añadiendo el parámetro al final. ej: "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-web-security
  - Aceptar el cambio y lanzar Chrome
- Para desactivar parcialmente la seguridad:
  - allow-file-access
  - allow-file-access-from-files
  - allow-cross-origin-auth-prompt
- Referencia a otros parametros:
  - <http://peter.sh/experiments/chromium-command-line-switches/>

© JMA 2020. All rights reserved

# Habilitar CORS en ASP.NET Core

- Crear la política en `Startup.ConfigureServices` (sin nombre sería la predeterminada):

```
services.AddCors(options => { options.AddPolicy("AllowAll", builder => builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader()); });
```
- Habilitarla mediante middleware para toda la aplicación en `Startup.Configure`:

```
app.UseCors("AllowAll");
```
- Habilitarla en el enrutamiento de puntos de conexión:

```
endpoints.MapControllerRoute(name: "apis", pattern: "api/{controller:exists}/{id?}", defaults: new { area = "Apis" }).RequireCors("AllowAll");
```
- Habilitarla con el atributo `[EnableCors]` para controladores o métodos de acción concretos.

```
[ApiController]
[EnableCors("AllowAll")]
public class AlumnosController : ControllerBase
```

© JMA 2020. All rights reserved

AJAX

## CÓDIGO EN EL LADO DEL CLIENTE

© JMA 2020. All rights reserved

## JavaScript no obstrusivo

- Todas las aplicaciones que se desarrollen deberían cumplir la normativa de accesibilidad vigente.
- Una de las normas indica que no se debe introducir eventos dentro de los elementos HTML.
- Los validadores lo detectan y avisan que no se podrá interactuar con determinados elementos que el usuario que accede desconoce, que no puede visualizar correctamente, etc.
- Alternativa: Usar atributos personalizados (no estándar) dentro de la etiqueta HTML que serán sustituidos al cargar el documento por el código JavaScript correspondiente (jquery.unobtrusive):
  - Validaciones: data-val-number, data-val-required, data-valmsg-for, data-valmsg-replace, ...

© JMA 2020. All rights reserved

## Estrategias en el uso de Ajax

- Para realizar la invocación las estrategias son:
  - Uso de hipervínculos
  - Uso de formularios
  - Uso del JQuery Ajax o fetch.
- Para el tratamiento de la respuesta se pueden emplear también dos estrategias:
  - Vistas parciales: La invocación devuelve una vista parcial que sustituye al contenido actual.
  - JSON: La invocación devuelve JSON que debe ser tratado en JavaScript para actualizar el contenido de la página actual.

© JMA 2020. All rights reserved

# Uso con JQuery AJAX

```
$.ajax({
 type: "PUT",
 url: "/api/miwebapi/" + $('#editId').val(),
 data: $('#editForm').serialize(),
 success: function (result) {
 if (result) {
 ...
 }
 }
});

$(document).ready(function () {
 $("#btnPartial").click(function () {
 $.ajax({ type: "GET", url: "texto", success: procesaPartialView });
 });
});
```

© JMA 2020. All rights reserved

## AJAX con PartialView

- En el controlador: Crear un método de acción que debe devolver PartialViewResult (se puede declarar como IActionResult) utilizando el método PartialView del controlador:

```
public PartialViewResult metodoAccion(...) {
 ...
 return PartialView(data);
}
```

- En la vista:

- Preparar e inicializar el contenedor del resultado.
- Crear una función Java que asigne el resultado:

```
<script type="text/javascript">
function procesaPartial(data) {
 $("#rsltAjax").html(data);
}
</script>
```
- Crear el formulario o enlace de invocación.

© JMA 2020. All rights reserved

# JSON

- JSON, acrónimo de JavaScript Object Notation, es un formato ligero de texto para el intercambio de datos (<http://www.json.org/>).
- JSON es un subconjunto de la notación literal de objetos de JavaScript (pero independiente) que no requiere el uso de XML (de mayor potencia y complejidad).
- Las implementaciones de XMLHttpRequest permiten cualquier codificación basada en texto, incluyendo: texto plano, XML, JSON y HTML.
- En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval().

© JMA 2020. All rights reserved

## Sintaxis JSON

- JSON permite definir dos tipos de estructuras básicas:
  - Colecciones de pares nombre/valor: para representar objetos, registros, estructuras, diccionarios, tablas hash, listas con clave, o matrices asociativa.  
`{ nombre : valor, ... }`
  - Listas ordenadas de valores: para representar tablas, vectores, listas o secuencias.  
`[ valor, ... ]`
- Un valor puede ser una cadena (entre comillas dobles y pueden secuencias de escape), un número (el punto como separador decimal), **true**, **false**, **null** u otras estructuras (estructuras anidadas).
- En JavaScript:  

```
miVariable = eval('(' + datosJSON + ')');
```

© JMA 2020. All rights reserved

# AJAX con JSON

- En el controlador: Crear un método de acción que debe devolver JsonResult (se puede declarar como IActionResult) utilizando el método Json del controlador:

```
public JsonResult metodoAccion(...) {
 ...
 return Json(data);
}
```

- En la vista:

- Preparar e inicializar el contenedor del resultado.
- Crear una función Java que procese el resultado (convertir JSON en HTML):

```
<script type="text/javascript">
function procesaJSON(data) {
 var target = $("#rsltAjax");
 target.empty();
 for (var i = 0; i < data.length; i++) {
 var modelo = data[i];
 target.append(...);
 }
</script>
```

- Crear el formulario o enlace de invocación.

© JMA 2020. All rights reserved

## Aplicaciones en una sola página

- Una aplicación de página única (SPA) es un tipo popular de aplicación web debido a su experiencia de usuario mejorada.
- Están disponibles plantillas de proyecto que habilitan la integración de bibliotecas o marcos de SPA del lado cliente, como Angular o React, con el marcos del lado servidor ASP.NET Core.
- Hay disponibles plantillas de terceros para otros marcos SPA populares.
- En el lado servidor se crearan los Web API de tramitación de datos.

© JMA 2020. All rights reserved

# Aplicaciones en tiempo real

- ASP.NET Core SignalR es una biblioteca de código abierto que simplifica la incorporación de funcionalidades Web en tiempo real a las aplicaciones. La funcionalidad web en tiempo real permite que el código del lado servidor Inserte contenido a los clientes al instante.
- Buenos candidatos para SignalR :
  - Aplicaciones que requieren actualizaciones desde el servidor con mucha frecuencia.
  - Paneles y aplicaciones de supervisión.
  - Aplicaciones de colaboración.
  - Aplicaciones que requieren notificaciones.
- Algunos ejemplos son correo electrónico, chat, juegos, redes sociales, votaciones, subastas, venta instantáneas, mapas, GPS, paneles empresariales, alertas, notificaciones, reuniones, pizarras, ...
- SignalR proporciona una API para crear llamadas a procedimiento remoto (RPC) de servidor a cliente. Las RPC llaman a funciones de JavaScript en los clientes desde el código de .NET Core del lado servidor.

© JMA 2020. All rights reserved

## SignalR para ASP.net Core

- Estas son algunas características:
  - Controla la administración de conexiones automáticamente.
  - Envía mensajes a todos los clientes conectados simultáneamente. Por ejemplo, un salón de chat.
  - Envía mensajes a clientes o grupos de clientes específicos.
  - Escalado para controlar el aumento del tráfico.
- SignalR elige automáticamente el mejor método de transporte que se encuentra dentro de las capacidades del servidor y del cliente.
- Admite las siguientes técnicas para controlar la comunicación en tiempo real (en orden de reserva correcta):
  - WebSockets
  - Eventos enviados por el servidor (server-sent event)
  - Sondeo largo (long polling)

© JMA 2020. All rights reserved

---

# SEGURIDAD DE UNA APLICACIÓN WEB

---

© JMA 2020. All rights reserved

## Características

---

- ASP.NET Core contiene características para administrar la autenticación, autorización, protección de datos, cumplimiento de HTTPS, secretos de aplicación, protección de vulnerabilidades y administración de CORS.
  - La autenticación es un proceso en el que un usuario proporciona credenciales que después se comparan con las almacenadas en un sistema operativo, base de datos, aplicación o recurso. La autorización se refiere al proceso que determina las acciones que un usuario puede realizar.
  - ASP.NET Core y EF contienen características que ayudan a proteger las aplicaciones y evitar las infracciones de seguridad:
    - Ataques de scripts de sitios (XSS)
    - Ataques por inyección de código SQL
    - Ataques de falsificación de solicitud entre sitios (XSRF/CSRF)
    - Ataques de redireccionamiento abierto
- 

© JMA 2020. All rights reserved



# Autenticación

- En ASP.NET Core, la autenticación se controla mediante `IAuthenticationService`, elemento que el middleware de autenticación emplea para conseguirlo. El servicio de autenticación usa controladores de autenticación registrados para completar las acciones relacionadas con la autenticación.
- Los controladores de autenticación registrados y sus opciones de configuración se denominan "esquemas".
- Para especificar esquemas de autenticación, es necesario registrar servicios de autenticación en `Startup.ConfigureServices`:
  - Mediante una llamada a un método de extensión específico del esquema tras una llamada a `services.AddAuthentication` (por ejemplo, `AddJwtBearer` o `AddCookie`). Estos métodos de extensión usan `AuthenticationBuilder.AddScheme` para registrar esquemas con la configuración adecuada.
  - Con menos frecuencia, mediante una llamada directa a `AuthenticationBuilder.AddScheme`.

© JMA 2020. All rights reserved

# Autenticación

- Si se usan varios esquemas, las directivas de autorización (o los atributos de autorización) pueden especificar el esquema (o esquemas) de autenticación del que dependen para autenticar al usuario.
- En algunos casos, la llamada a `AddAuthentication` se realiza automáticamente mediante otros métodos de extensión (al usar ASP.NET Core Identity, se llama a `AddAuthentication` de manera interna).
- Para agregar el middleware de autenticación en `Startup.Configure`, se llama al método de extensión `UseAuthentication` en el elemento `IApplicationBuilder` de la aplicación. La llamada a `UseAuthentication` registra el middleware que usa los esquemas de autenticación previamente registrados, debe hacerse antes de registrar cualquier middleware que requiera autenticación. Al usar el enrutamiento de punto de conexión, la llamada a `UseAuthentication` debe ir:
  - Después de `UseRouting`, para que la información de ruta esté disponible para las decisiones de autenticación.
  - Antes de `UseEndpoints`, para que los usuarios se autenticuen como paso previo para tener acceso a los puntos de conexión.

© JMA 2020. All rights reserved

# Opciones de autenticación

- Autenticación de Windows
- Autenticación de formularios
  - Cuentas de usuario individuales (ASP.NET Core Identity):
    - Los usuarios pueden crear una cuenta con la información de inicio de sesión almacenada en Identity o pueden utilizar un proveedor de inicio de sesión externo.
    - Entre los proveedores de inicio de sesión externos admitidos se incluyen Facebook, Google, cuenta de Microsoft y Twitter.
  - Profesionales y educativas desde instancias de Azure Active Directory (individuales, organización única, varias organizaciones)

© JMA 2020. All rights reserved

## Autenticación Windows

- La autenticación de Windows (también conocida como autenticación de negociación, Kerberos o NTLM) puede configurarse para ASP.NET Core aplicaciones hospedadas con IIS o HTTP.sys.
- Ventajas:
  - Usa la infraestructura de Windows existente
  - Controla el acceso a información confidencial y se integra con el sistema operativo
  - Mayor nivel de seguridad
- Desventajas:
  - No admite todos los tipos de clientes
  - Requiere relaciones de confianza
- Habitualmente se restringe a Intranet.
- Se configura en Startup.ConfigureServices :  
`services.AddAuthentication(IISDefaults.AuthenticationScheme);`

© JMA 2020. All rights reserved

## Autenticación de formularios

- **Ventajas:**
  - Usa su propia infraestructura que se puede personalizar
  - Admite todos los tipos de clientes (cualquier S.O. y dominio, incluyendo la autenticación delegada)
- **Desventajas:**
  - Menor nivel de seguridad
  - Se basa en cookies
  - Requiere comunicaciones seguras dado que el usuario y la contraseña se transmiten en abierto ([RequireHttps])
- Recomendable para su uso en Internet.
- Las plantillas de MVC suministran el controlador, los modelos y las vistas para su gestión y personalización.

© JMA 2020. All rights reserved

## ASP.NET Core Identity

- Identity se puede configurar mediante una base de datos de SQL Server para almacenar nombres de usuario, contraseñas y datos de perfil. Como alternativa, se puede usar otro almacén persistente, por ejemplo, Azure Table Storage.
- El proceso de scaffolding del proyecto genera automáticamente la base de datos, los modelos, vistas y controladores para dar soporte a Identity.
- El código fuente se suministra como una biblioteca de clases Razor (RCL). Las vistas se puede extraer para su personalización con:
  - Explorador de soluciones, haga clic con el botón derecho en el proyecto > Agregar > Nuevo elemento con scaffolding > Identidad > Agregar y, en el cuadro de diálogo Agregar identidad , seleccionando las views o partialViews deseadas.
- El punto de partida para la personalización del modelo es derivar del tipo de contexto adecuado (habitualmente ApplicationDbContext) y extender los tipos base.

© JMA 2020. All rights reserved

# Modelos de Identity

Entidad	Tipo genérico	Descripción
User	IdentityUser	Representa al usuario.
Role	IdentityRole	Representa un rol.
UserClaim	IdentityUserClaim	Representa una demanda que un usuario posee.
UserToken	IdentityUserToken	Representa un token de autenticación para un usuario.
UserLogin	IdentityUserLogin	Asocia un usuario a un inicio de sesión.
RoleClaim	IdentityRoleClaim	Representa una demanda que se concede a todos los usuarios de un rol.
UserRole	IdentityUserRole	Entidad de combinación que asocia usuarios y roles.

© JMA 2020. All rights reserved

## Configurar servicios Identity

- Las opciones se establecen en `Startup.ConfigureServices`:  

```
services.AddDefaultIdentity<IdentityUser>(options => {
 options.Password.RequireNonAlphanumeric = false;
 options.Password.RequiredLength = 6;
 options.Password.RequiredUniqueChars = 1;
 options.SignIn.RequireConfirmedAccount = true;
 options.User.RequireUniqueEmail = false;
}).AddEntityFrameworkStores<ApplicationDbContext>();
```
- Se puede configurar las características de los usuarios y las contraseñas, su algoritmo hash, bloqueo ante fallos, tokens, ...
- En `Startup.Configure`:  

```
app.UseAuthentication();
```

© JMA 2020. All rights reserved

# OAuth/OpenID

- ASP.NET Core permite la delegación, mediante OAuth 2.0, de la autenticación en credenciales de Microsoft Live, Facebook, Twitter y Google. El hecho de permitir a los usuarios iniciar sesión con sus credenciales: resulta muy práctico para los usuarios y transfiere muchas de las complejidades de administrar el proceso de inicio de sesión a un tercero.
- Como paso previo es necesario ir a los diferentes proveedores a registrar la aplicación/URL para obtener un identificador y una contraseña/código secreto (según el proveedor).

```
services.AddAuthentication()
 .AddMicrosoftAccount(microsoftOptions => { ... })
 .AddGoogle(googleOptions => { ... })
 .AddTwitter(twitterOptions => { ... })
 .AddFacebook(facebookOptions => { ... });
```
- A través de paquetes NuGet de terceros hay proveedores externos para: LinkedIn, Instagram, Reddit, GitHub, Yahoo, Tumblr, Pinterest, Flickr, Vimeo, ...

© JMA 2020. All rights reserved

## Multi-factor Authentication (MFA)

- Multi-factor Authentication (MFA) es un proceso en el que se solicita un usuario durante un evento de inicio de sesión para otras formas de identificación. Este mensaje puede indicar un código de un teléfono móvil, usar una clave FIDO2 o proporcionar un análisis de huellas digitales. Cuando se requiere una segunda forma de autenticación, se mejora la seguridad. Un atacante no obtiene ni duplica fácilmente el factor adicional.
- MFA requiere al menos dos o más tipos de pruebas para una identidad como algo que conoce, algo que posee o validación biométrica para que el usuario se autentique.
- La autenticación en dos fases (2FA) es como un subconjunto de MFA, pero la diferencia es que MFA puede requerir dos o más factores para demostrar la identidad.
- MFA con SMS aumenta la seguridad de forma masiva en comparación con la autenticación de contraseña (factor único). Sin embargo, ya no se recomienda usar SMS como segundo factor. Existen demasiados vectores de ataque conocidos para este tipo de implementación.

© JMA 2020. All rights reserved

## Autenticación en dos fases

- MFA con TOTP, un algoritmo de contraseña de un solo uso, es una implementación compatible mediante ASP.NET Core Identity. Se puede usar junto con cualquier aplicación de autenticador compatible, lo que incluye:
  - Aplicación Microsoft Authenticator
  - Aplicación de Google Authenticator
- Una aplicación autenticadora proporciona un código de 6 a 8 dígitos que los usuarios deben escribir después de confirmar su nombre de usuario y contraseña.
- El enfoque recomendado del sector para 2FA es TOTP frente SMS.
- Normalmente, una aplicación autenticadora se instala en un smartphone.
- Se puede habilitar la generación de código QR para las aplicaciones de TOTP Authenticator.

© JMA 2020. All rights reserved

## Autorización

- La autorización se refiere al proceso que determina lo que un usuario puede hacer. La autorización es ortogonal e independiente de la autenticación. Sin embargo, la autorización requiere un mecanismo de autenticación. La autenticación es el proceso de determinar quién es un usuario. La autenticación puede crear una o varias identidades para el usuario actual.
- La autorización de ASP.NET Core proporciona un modelo basado en roles, sencillo y declarativo, y un modelo avanzado basado en directivas.
- La autorización se expresa en requisitos y los controladores evalúan las notificaciones de un usuario en relación con los requisitos.
- Las comprobaciones imperativas pueden basarse en directivas simples o directivas que evalúan la identidad del usuario y las propiedades del recurso al que el usuario está intentando obtener acceso.

© JMA 2020. All rights reserved

# Configurar la autorización

- Las opciones se establecen en `Startup.ConfigureServices`:

```
services.AddDefaultIdentity<IdentityUser>()
 .AddRoles<IdentityRole>()
 .AddEntityFrameworkStores<ApplicationDbContext>();
services.AddAuthorization(options => {
 options.AddPolicy("RequireAdministratorRole",
 policy => policy.RequireRole("Administrator"));
});
```
- En `Startup.Configure`:

```
app.UseAuthorization();
```

© JMA 2020. All rights reserved

## Autorización basada en roles

- Las comprobaciones de autorización basadas en roles son declarativas, el desarrollador las inserta en el código, en un controlador o una acción dentro de un controlador, especificando los roles de los que el usuario actual debe ser miembro para tener acceso al recurso solicitado.

```
[Authorize(Roles = "Administrator, PowerUser")]
public class ControlPanelController : Controller {
```
- `[Authorize]`, sin nada, obliga al usuario a estar autenticado.
- Se puede bloquear un controlador, pero permitir el acceso anónimo y no autenticado a acciones individuales.

```
[AllowAnonymous]
public ActionResult Login() {
```
- Los requisitos de rol también se pueden expresar con la sintaxis de directivas:

```
[Authorize(Policy = "RequireAdministratorRole")]
public class ControlPanelController : Controller {
```

© JMA 2020. All rights reserved

# Autorización basada en directivas

- La autorización basada en directivas utiliza requisitos, controladores de requisitos y directivas de autorización.
- Una directiva de autorización se compone de uno o varios requisitos. Se registra como parte de la configuración del servicio de autorización en `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services) {
 services.AddAuthorization(options => {
 options.AddPolicy("AtLeast21", policy =>
 policy.Requirements.Add(new MinimumAgeRequirement(18)));
 });
 services.AddSingleton<IAuthorizationHandler, MinimumAgeHandler>();
}
```
- El servicio principal que determina si la autorización se realiza correctamente es `IAuthorizationService`.
- `IAuthorizationRequirement` es un servicio de marcador sin métodos, con propiedades para la personalización y es el mecanismo para hacer un seguimiento de si la autorización se realiza correctamente.
- El servicio `IAuthorizationHandler` es responsable de comprobar si se cumplen los requisitos asociados por el tipo genérico.

© JMA 2020. All rights reserved

# Autorización basada en directivas

- Un requisito de autorización es una colección de parámetros de datos que una Directiva puede usar para evaluar y configurar la entidad de seguridad de usuario actual.

```
namespace Services.Requirements {
 public class MinimumAgeRequirement : IAuthorizationRequirement {
 public int MinimumAge { get; }
 public MinimumAgeRequirement(int minimumAge) {
 if (minimumAge < 0)
 throw new ArgumentOutOfRangeException();
 MinimumAge = minimumAge;
 }
 }
}
```
- Un controlador de autorización es responsable de la evaluación de las propiedades de un requisito. El controlador de autorización evalúa los requisitos con respecto a un `AuthorizationHandlerContext` proporcionado para determinar si se permite el acceso. Un requisito puede tener varios controladores.
- Un controlador puede implementar `IAuthorizationHandler` para administrar más de un tipo de requisito.

© JMA 2020. All rights reserved



# Autorización basada en directivas

- Un controlador de autorización puede heredar `AuthorizationHandler<TRequirement>`, donde `TRequirement` es el requisito que se debe controlar.  

```
namespace Services.Handlers {
 public class MinimumAgeHandler : AuthorizationHandler<MinimumAgeRequirement> {
 protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
 MinimumAgeRequirement requirement) {
 if (!context.User.HasClaim(c => c.Type == ClaimTypes.DateOfBirth))
 return Task.CompletedTask;
 var dateOfBirth = Convert.ToDateTime(context.User.FindFirst(c => c.Type ==
 ClaimTypes.DateOfBirth).Value);
 if (dateOfBirth.AddYears(requirement.MinimumAge).Date <= DateTime.Today)
 context.Succeed(requirement);
 return Task.CompletedTask;
 }
 }
}
```
- Un controlador indica que se ha realizado correctamente llamando a `context.Succeed(requirement)`, si no se indica nada pasa a otros controladores para que el mismo requisito pueda cumplirse. Para garantizar el fallo, independientemente otros controladores de requisitos, se llama a `context.Fail()`.

© JMA 2020. All rights reserved

# Autorización basada en directivas

- Los controladores de autorización deben registrarse en el inyector de dependencias:  

```
services.AddSingleton<IAuthorizationHandler, MinimumAgeHandler>();
```
- Cuando el cumplimiento de una directiva sea fácil de expresar en el código, es posible proporcionar un `Func<AuthorizationHandlerContext, bool>` al configurar la Directiva con el generador de directivas `RequireAssertion`.  

```
services.AddAuthorization(options => {
 options.AddPolicy("BadgeEntry", policy => policy.RequireAssertion(context =>
 context.User.HasClaim(c => (c.Type == "BadgeId" || c.Type ==
 "TemporaryBadgeId") && c.Issuer == "https://microsoftsecurity")));
});
```
- `AuthorizationPolicyBuilder`, además de `RequireAssertion`, dispone de los atajos: `RequireUserName`, `RequireRole`, `RequireClaim` y `RequireAuthenticatedUser`.  

```
options.AddPolicy("EmployeeOnly", policy => policy.RequireClaim("isEmployee",
 true));
```

© JMA 2020. All rights reserved

# Personalización de controladores

- En la Autorización basada en recursos, la estrategia de autorización depende del recurso al que se tiene acceso y se implementa mediante personalización imperativa de los controladores. La autorización esta implementa como un servicio `IAuthorizationService`, registrado en `Startup.ConfigureServices` y disponible en el controlador a través de la inyección de dependencias:

```
public class DocumentController : Controller {
 private readonly IAuthorizationService _authorizationService;
 public DocumentController(IAuthorizationService authorizationService) {
 _authorizationService = authorizationService;
 }
}
```

- La autorización imperativa se implementa en las acciones:

```
public async Task<IActionResult> Edit(Guid documentId) {
 var document = _documentRepository.Find(documentId);
 if (document == null) return NotFound(); // 404
 if ((await AuthorizationService.AuthorizeAsync(User, Document, "EditPolicy")).Succeeded)
 return View(document); // 200
 if (User.Identity.IsAuthenticated) return Forbid(); // 403
 return new Unauthorized(); // 401
}
```

© JMA 2020. All rights reserved

# Personalización de vistas

- A menudo, un desarrollador desea mostrar, ocultar o modificar de alguna forma una interfaz de usuario basada en la identidad del usuario actual.
- La propiedad `User`, disponible a través del contexto de las vistas y los controladores, obtiene la información de seguridad del usuario para la solicitud HTTP actual:

- `Identity`: Obtiene la identidad de la entidad de seguridad actual.
  - `IsAuthenticated`: indica si la solicitud se ha autenticado.
  - `Name`: Obtiene el nombre del usuario actual.
  - `AuthenticationType`: Obtiene el tipo de autenticación utilizado.
- `IsInRole()`: Determina si la entidad de seguridad actual pertenece al rol especificado.

- Puede tener acceso al servicio de autorización a través de la inyección de dependencias como en los controladores:

```
@using Microsoft.AspNetCore.Authorization
@inject IAuthorizationService AuthorizationService
```

- Para personalizar la vista:

```
@if ((await AuthorizationService.AuthorizeAsync(User, "MyPolicy")).Succeeded) {
```

© JMA 2020. All rights reserved

# Administración de secretos

- Para conectarse con recursos protegidos y otros servicios, las aplicaciones típicamente necesitan usar cadenas de conexión, contraseñas u otras credenciales que contengan información confidencial.
- Estas partes de información sensible se llaman secretos.
- Es una buena práctica no incluir secretos en el código fuente (appsettings.json) y, sobre todo, no almacenar secretos en el sistema de control de versiones.
- En su lugar, debería utilizar el modelo de configuración para leer los secretos desde ubicaciones más seguras.
- Se deben separar los secretos para acceder a los recursos de desarrollo y pre- producción (staging) de los que se usan para acceder en producción, porque diferentes individuos necesitarán acceder a esos conjuntos diferentes de secretos. Para almacenar secretos usados durante el desarrollo, los enfoques comunes son almacenar secretos en variables de entorno. Para un almacenamiento más seguro en entornos de producción, se pueden almacenar secretos en un proveedor de configuración como Azure Key Vault.

© JMA 2020. All rights reserved

# Administración de secretos

- Las variables de entorno se usan para evitar el almacenamiento de secretos de aplicación en código o en archivos de configuración local. Las variables de entorno invalidan los valores de configuración de todos los orígenes de configuración especificados previamente. Si el equipo o el proceso se ve comprometido, las variables de entorno no son de confianza.
- La herramienta Administrador de secretos almacena datos confidenciales durante el desarrollo de un proyecto de ASP.NET Core. En este contexto, una parte de la información confidencial es un secreto de la aplicación, que se almacenan en una ubicación independiente del árbol del proyecto. Los secretos de la aplicación se asocian a un proyecto específico o se comparten entre varios proyectos. Los secretos de la aplicación no se protegen en el control de código fuente. La herramienta Administrador de secretos no cifra los secretos almacenados y no debe tratarse como un almacén de confianza. Solo es para fines de desarrollo. Las claves y los valores se almacenan en un archivo de configuración JSON en el directorio del perfil de usuario.

© JMA 2020. All rights reserved

# Requerir HTTPS

- Se recomienda que todas las aplicaciones Web utilicen HTTPS. Es obligatorio en caso de utilizar autenticación por formularios.
- Para las aplicaciones Web de producción ASP.NET Core se recomienda que usen (Startup.Configure):
  - Middleware de redirección de HTTPS ( UseHttpsRedirection ) para redirigir las solicitudes HTTP a https.  
`app.UseHttpsRedirection();`
  - Middleware de HSTS (UseHsts) para enviar encabezados del Protocolo de seguridad de transporte estricto (HSTS) http a los clientes.  
`app.UseHsts();`

© JMA 2020. All rights reserved

# XSS: Cross-Site Scripting

- Técnica utilizada para inyectar código malicioso ejecutable en las aplicaciones, utiliza las entradas de texto de los formularios que posteriormente se mostrarán en las páginas.
- ASP.NET Core protege por defecto del XSS formateando la salida de los valores como texto (HTML Encoding). La codificación HTML toma caracteres como < y los cambia de forma segura como &lt; haciendo que pierdan su significado HTML.
- Para que se puedan insertar texto enriquecido con HTML en Razor se debe utilizar en las vistas el Helper @Html.Raw(Propiedad).
- Los datos proporcionados por el usuario nunca son confiables para mostrarlos en modo Raw. Si se deben permitir, la validación es imprescindible para limitar los ataques XSS, se recomienda el uso de la biblioteca AntiXSS como HtmlSanitizer (en Nuget Packages) para “sanear” los datos de entrada que puedan contener código malicioso (validación positiva).
- La tendencia actual es utilizar Markdown como opción más segura para aceptar entradas enriquecidas.

© JMA 2020. All rights reserved

# CSRF: Cross Site Request Forgery

- Al contrario del XSS que explota la confianza del usuario en un sitio en particular, explota la confianza del sitio en un usuario en particular.
- El CSRF utiliza a un usuario ya autenticado en un sitio para, a través de este, introducir solicitudes "válidas" al sitio en su nombre sin su conocimiento (siempre que mantenga abierta la sesión) mediante formularios y campos ocultos.
- Mitigación:
  - Habilitar cierre de sesión por el usuario
  - Control estricto del Session.Timeout
  - Supervisión del atributo Referer de la cabecera HTTP.
  - Usar un Token dinámico:
    - En la vista (Helper): @Html.AntiForgeryToken()
    - En el método de acción (Decorador): [ValidateAntiForgeryToken]

© JMA 2020. All rights reserved

## Prevención de ataques de falsificación de solicitud entre sitios (XSRF/CSRF)

- El enfoque más común para defenderse contra ataques CSRF es usar el patrón de token de sincronizador (STP). STP se usa cuando el usuario solicita una página con datos de formulario:
  1. El servidor envía al cliente un token asociado a la identidad del usuario actual: campo HIDDEN y cookie.
  2. El cliente devuelve formulario al servidor, el navegador adjunta cookie.
  3. Si los tokens que el servidor recibe en el formulario y en la cookie no coinciden con la identidad del usuario autenticado, se rechaza la solicitud.
- El mecanismo "Cookie-to-Header Token" se utiliza para prevenir ataques XSRF con los API Web.
  1. El servidor debe establecer un token en una cookie de sesión legible en JavaScript, llamada XSRF-TOKEN, en la carga de la página o en la primera solicitud GET. En las solicitudes posteriores, el cliente debe incluir el encabezado HTTP X-XSRF-TOKEN con el valor recibido en la cookie. El navegador no permite acceder con JavaScript a cookies entre sitios.
  2. El servidor puede verificar que el valor en la cookie coincida con el del encabezado HTTP y, por lo tanto, asegúrese de que sólo el código que se ejecutó en su dominio pudo haber enviado la solicitud.

© JMA 2020. All rights reserved

# Inyección SQL

- Se produce al generar consultas SQL que se concatenan con entradas de usuario no validadas y su posterior ejecución en el motor de la base de datos.
- No se producen cuando se usa EF y LINQ
- Se evitan al utilizar consultas parametrizadas.
- Se debe establecer una política de privilegios mínimos en la base de datos.

© JMA 2020. All rights reserved

## DEPURACIÓN, PRUEBAS UNITARIAS Y REFACTORIZACIÓN

© JMA 2020. All rights reserved

# Depuración

- El depurador permite observar el comportamiento del programa en tiempo de ejecución y encontrar errores lógicos.
- El depurador permite:
  - Control de ejecución: Iniciar, continuar, interrumpir o detener la ejecución. Avanzar paso a paso por la aplicación. Ejecutar un proceso hasta una ubicación especificada. Establecer el punto de ejecución.
  - Puntos de interrupción, puntos de seguimiento, pila de llamada, ...
  - Inspección de variables, registros, memoria, ...
  - Editar y continuar

© JMA 2020. All rights reserved

# Unit testing

- Probar partes del sistema de manera individual asegurando que funcionan correctamente
- Provee un contrato escrito y estricto que una porción de código debe cumplir
- Como resultado podemos encontrar problemas de manera temprana y de manera instantánea

© JMA 2020. All rights reserved

## Herramientas de prueba de Visual Studio

- Las herramientas de prueba de Visual Studio permiten desarrollar y mantener altos estándares de excelencia de código.
- Las pruebas unitarias están disponibles en todas las ediciones de Visual Studio.
- Otras herramientas de pruebas, como Live Unit Testing, IntelliTest y Pruebas automatizadas de IU, solo están disponibles en la edición Visual Studio Enterprise.

© JMA 2020. All rights reserved

## Marcos de pruebas

- Unit Test Frameworks:
  - MSTest, NUnit, xUnit (<https://xunit.net/docs/comparisons>)
- Mocking Frameworks:
  - Microsoft Fakes, Moq, NSubstitute, Rhino Mocks, FakeItEasy, EntityFramework.Testing
- Code Coverage (Métrica de calidad del software: Valor cuantitativo que indica que cantidad del código ha sido ejercitada por un conjunto de casos de prueba):
  - Visual Studio, NCover, NCrunch, OpenCover

© JMA 2020. All rights reserved



# Microsoft.VisualStudio.TestTools

## .UnitTesting

- Atributos que identifican las clases y métodos de prueba ([TestClass] y [TestMethod]).
- Atributos de inicialización y limpieza para ejecutar código antes o después de ejecutar las pruebas unitarias, a fin de asegurarse un estado inicial y final concretos ([TestInitialize] y [TestCleanup]).
- Clases Assert que se pueden utilizar para comprobar las condiciones en las pruebas unitarias.
- El atributo ExpectedException para comprobar si se inicia determinado tipo de excepción durante la ejecución de la prueba unitaria.
- La clase TestContext que almacena la información que se proporciona a las pruebas unitarias, como la conexión de datos para las pruebas controladas por datos y la información necesaria para ejecutar las pruebas unitarias para los servicios Web ASP.NET.

© JMA 2020. All rights reserved

## Proyecto de prueba unitaria

- Las pruebas unitarias a menudo reflejan la estructura del código sometido a pruebas.
- Habitualmente, se crearía un proyecto de prueba unitaria para cada proyecto de código en el producto.
- El proyecto de prueba puede estar en la misma solución que el código fuente o puede estar en una solución independiente.
- En una solución se pueden tener tantos proyectos de prueba unitaria como sea necesario.
- Para crear un proyecto de prueba unitaria, en el menú Archivo > Nuevo > Proyecto, filtrar por tipo Prueba y seleccionar la plantilla de proyecto del marco de pruebas que se desea usar.
- En el proyecto de prueba unitaria, agregar la referencia al proyecto que se quiere probar haciendo clic con el botón derecho en Referencias o Dependencias y eligiendo Agregar referencia.
- En general, es más rápido generar el proyecto de prueba unitaria y los códigos auxiliares de pruebas unitarias a partir del código, haciendo clic con el botón derecho y seleccione *Crear pruebas unitarias* en el menú contextual.

© JMA 2020. All rights reserved

## Ejecutar pruebas unitarias

- Se puede usar el Explorador de pruebas para ejecutar pruebas unitarias en el marco de pruebas integrado (MSTest) o en marcos de pruebas de terceros.
- Se puede agrupar las pruebas en categorías, filtrar la lista de pruebas y crear, guardar y ejecutar listas de reproducción de pruebas.
- También se puede depurar las pruebas, analizar la cobertura de código y el rendimiento de la prueba.
- En Visual Studio Enterprise Edition, con Live Unit Testing se pueden ver los resultados en vivo de las pruebas unitarias.
- Para determinar qué proporción de código del proyecto se está probando realmente mediante pruebas codificadas como pruebas unitarias, se puede utilizar la característica de cobertura de código de Visual Studio. Para restringir con eficacia los errores, las pruebas deberían ensayar una proporción considerable del código.

© JMA 2020. All rights reserved

## Patrones

- Los casos de prueba se pueden estructurar siguiendo diferentes patrones:
  - ARRANGE-ACT-ASSERT: Preparar, Actuar, Afirmar
  - GIVEN-WHEN-THEN: Dado, Cuando, Entonces
  - BUILD-OPERATE-CHECK: Generar, Operar, Comprobar
- Aunque con diferencias conceptuales, todos dividen el proceso en tres fases:
  - Una fase inicial donde montar el escenario de pruebas que hace que el resultado sea predecible.
  - Una fase intermedia donde se realizan las acciones que son el objetivo de la prueba.
  - Una fase final donde se comparan los resultados con el escenario previsto.

© JMA 2020. All rights reserved

## Preparación mínima

- La sección de preparación, con la entrada del caso de prueba, debe ser lo más sencilla posible, lo imprescindible para comprobar el comportamiento que se está probando.
- Las pruebas se hacen más resistentes a los cambios futuros en el código base y más cercano al comportamiento de prueba que a la implementación.
- Las pruebas que incluyen más información de la necesaria tienen una mayor posibilidad de incorporar errores en la prueba y pueden hacer confusa su intención. Al escribir pruebas, el usuario quiere centrarse en el comportamiento. El establecimiento de propiedades adicionales en los modelos o el empleo de valores distintos de cero cuando no es necesario solo resta de lo que se quiere probar.

© JMA 2020. All rights reserved

## Actuación mínima

- Al escribir las pruebas hay que evitar introducir condiciones lógicas como if, switch, while, for, etc.
- Minimiza la posibilidad de incorporar un error a las pruebas.
- El foco está en el resultado final, en lugar de en los detalles de implementación.
- Al incorporar lógica al conjunto de pruebas, aumenta considerablemente la posibilidad de agregar un error. Cuando se produce un error en una prueba, se quiere saber realmente que algo va mal con el código probado y no en el código que prueba. En caso contrario, restan confianza y las pruebas en las que no se confía no aportan ningún valor.
- El objetivo de la prueba debe ser único, si la lógica en la prueba parece inevitable, denota que el objetivo es múltiple y hay que considerar la posibilidad de dividirla en dos o más pruebas diferentes.

© JMA 2020. All rights reserved

## Evitar varias aserciones

- Al escribir las pruebas, hay que intentar incluir solo una aserción por prueba. Los enfoques comunes para usar solo una aserción incluyen:
  - Crear una prueba independiente para cada aserción.
  - Usar pruebas con parámetros.
- Si se produce un error en una aserción, no se evalúan las aserciones posteriores.
- Garantiza que no se estén declarando varios casos en las pruebas.
- Proporciona la imagen exacta de por qué se producen errores en las pruebas.
- Al incorporar varias aserciones en un caso de prueba, no se garantiza que se ejecuten todas. Es un todas o ninguna, se sabe por cual fallo pero no si el resto también falla o es correcto, proporcionando la imagen parcial.
- Una excepción común a esta regla es cuando la validación cubre varios aspectos. En este caso, suele ser aceptable que haya varias aserciones para asegurarse de que el resultado está en el estado que se espera que esté.

© JMA 2020. All rights reserved

## Lenguaje específico del dominio

- La refactorización del código de prueba favorece la reutilización y la legibilidad, simplifican las pruebas.
- Salvo que todos los métodos de prueba usen los mismos requisitos, si se necesita un objeto o un estado similar para las pruebas, es preferible usar métodos auxiliares a los métodos de instalación y desmontaje (si existen):
  - Menos confusión al leer las pruebas, puesto que todo el código es visible desde dentro de cada prueba.
  - Menor posibilidad de configurar mas o menos de lo necesario para la prueba.
  - Menor posibilidad de compartir el estado entre las pruebas, lo que crea dependencias no deseadas entre ellas.
- Cada prueba normalmente tendrá requisitos diferentes para funcionar y ejecutarse. Los métodos de instalación y desmontaje son únicos, pero se pueden crear tantos métodos auxiliares como escenarios reutilizables se necesiten.

© JMA 2020. All rights reserved

## Cubrir aspectos no evidentes

- Las pruebas no deben cubrir solo los casos evidentes, los correctos, sino que deben ampliarse a los casos incorrectos.
- Un juego de pruebas debe ejercitar la resiliencia: la capacidad de resistir los errores y la recuperación ante los mismos.
- En los cálculos no hay que comprobar solamente si realiza correctamente el calculo, también hay que verificar que es el calculo que se debe realizar.
- Los dominios de los datos determinan la validez de los mismos y fijan la calidad de la información, dichos dominios deben ser ejercitados profundamente.

© JMA 2020. All rights reserved

## Respetar los limites de las pruebas

- Las pruebas unitarias ejercitan profundamente los componentes de formar aislada centrándose en la funcionalidad, los cálculos, las reglas de dominio y semánticas de los datos. Opcionalmente la estructura del código, es decir, sentencias, decisiones, bucles y caminos distintos.
- Las pruebas de integración se basan en componentes ya probados (unitaria o integración) o en dobles de pruebas y se centran en la estructura de llamadas, secuencias o colaboración, y la transición de estados.
- Hay muchos tipos de pruebas de sistema y cada uno pone el foco en un aspecto muy concreto, cada prueba solo debe tener un aspecto. Las pruebas funcionales del sistema son las pruebas de integración de todo el sistema centrándose en compleción de la funcionalidades y los procesos de negocio, su estructura, disponibilidad y accesibilidad.

© JMA 2020. All rights reserved

# Pirámide de pruebas



© JMA 2020. All rights reserved

## Principios F.I.R.S.T.

- **Fast:** Los tests deben ser rápidos, del orden de milisegundos, hay cientos sino miles de tests en un proyecto que se deben ejecutar continuamente.
- **Independent:** Los tests no deben depender del entorno ni de ejecuciones de tests anteriores, se pueden ejecutar en cualquier orden.
- **Repeatable:** Los tests deben ser repetibles, para cualquier entorno, y ante la misma entrada de datos, producen los mismos resultados.
- **Self-Validating:** Los tests tienen que ser autovalidados, es decir, NO debe de existir la intervención humana en la validación. El resultado debe ser booleano: pasa o falla.
- **Timely (Oportuno):** Los tests deben crearse en el momento oportuno, antes del código, y ejecutarse en el momento oportuno, después de cada cambio en el código.

© JMA 2020. All rights reserved

# Aislar las pruebas

- Las dependencias externas afectan a la complejidad de la estrategia de pruebas, hay que aislar a las pruebas de las dependencias externas, sustituyendo las dependencias por dobles de prueba, salvo que se este probando específicamente dichas dependencias.
- Siguiendo la misma regla de oro, las pruebas de integración y sistema deben estar aisladas de sus dependencias salvo cuando se estén probando dichas dependencias. Así mismo, el resultado de las pruebas debe ser previsible.
- Entre las ventajas de esta aproximación se encuentran:
  - Devuelven resultados determinísticos
  - Permiten crear o reproducir determinados estados (por ejemplo errores de conexión)
  - Obtienen resultados mucho mas rápidamente y a menor coste, incluso offline.
  - Permiten el inicio temprano de las pruebas incluso cuando las dependencias todavía no están disponibles.
  - Permiten incluir atributos o métodos exclusivamente para el testeo.

© JMA 2020. All rights reserved

# Simulación de objetos

- Las dependencias con sistemas externos afectan a la complejidad de la estrategia de pruebas, ya que es necesario contar con sustitutos de estos servicios externos durante el desarrollo. Ejemplos típicos de estas dependencias son Servicios Web, Sistemas de envío de correo, Fuentes de Datos o simplemente dispositivos hardware.
- Estos sustitutos, muchas veces son exactamente iguales que el servicio original, pero en otro entorno o son simuladores que exponen el mismo interfaz pero realmente no realizan las mismas tareas que el sistema real, o las realizan contra un entorno controlado.
- Para poder emplear la técnica de simulación de objetos se debe diseñar el código a probar de forma que sea posible trabajar con los objetos reales o con los objetos simulados:
  - IoC: Inversión de Control (Inversion Of Control)
  - DI: Inyección de Dependencias (Dependency Injection)
  - Objetos Mock

© JMA 2020. All rights reserved

# Moq

- Moq (pronunciado "Mock-you" o simplemente "Mock") es la biblioteca para mocking más popular y amigable para .NET desarrollada desde cero para aprovechar al máximo los árboles de expresión .NET Linq y las expresiones lambda, lo que la convierte en una de las bibliotecas más productiva, con seguridad de tipos y fácil de refactorizar disponible.
- Admite dobles de pruebas tanto sobre interfaces como sobre clases.
- El API es extremadamente simple y directo, no requiere ningún conocimiento previo o experiencia con conceptos de dobles de pruebas.
- La instalación y descarga se realiza a través de NuGet.

© JMA 2020. All rights reserved

## Desarrollo Guiado por Pruebas (TDD)

- El Desarrollo Guiado por Pruebas, es una técnica de programación (definida por KentBeck); consistente en desarrollar primero el código que pruebe una característica o funcionalidad deseada antes que el código que implementa dicha funcionalidad.
- El objetivo a lograr es que no exista ninguna funcionalidad que no esté avalada por una prueba.
- Lo primero que hay que aprender de TDD son sus reglas básicas:
  - No añadir código sin escribir antes una prueba que falle
  - Eliminar el Código Duplicado empleando Refactorización

© JMA 2020. All rights reserved



# Ritmo TDD

- TDD invita a seguir una serie de tareas ordenadas, que a menudo se denomina ritmo TDD, y que se basa en los siguientes pasos:
  1. Escribir una prueba que demuestre la necesidad de escribir código.
  2. Escribir el mínimo código para que el código de pruebas compile
  3. Implementar exclusivamente la funcionalidad demandada por las pruebas
  4. Mejorar el código (Refactoring) sin añadir funcionalidad
  5. Volver al primer paso
- Este ritmo permite formalizar las tareas que se han de realizar para conseguir un código fácil de mantener, bien diseñado y que se puede probar automáticamente.

© JMA 2020. All rights reserved

## Estrategia RED – GREEN

- Se recomienda una estrategia de test unitarios conocida como **RED** (fallo) – **GREEN** (éxito), es especialmente útil en equipos de desarrollo ágil.
- Una vez que entendamos la lógica y la intención de un test unitario, hay que seguir estos pasos:
  - Escribe el código del test (**Stub**) para que compile (pase de **RED** a **GREEN**)
    - Inicialmente la compilación fallará **RED** debido a que falta código
    - Implementa sólo el código necesario para que compile **GREEN** (aún no hay implementación real).
  - Escribe el código del test para que se **ejecute** (pase de **RED** a **GREEN**)
    - Inicialmente el test fallará **RED** ya que no existe funcionalidad.
    - Implementa la funcionalidad que va a probar el test hasta que se ejecute adecuadamente **GREEN**.
  - **Refactoriza** el test y el código una vez que este todo **GREEN** y la solución vaya evolucionando.

© JMA 2020. All rights reserved

# Ritmo TDD



© JMA 2020. All rights reserved

## Ciclo de vida

1. Crear los proyectos de código fuente y de pruebas.
2. Agregar "Prueba unitaria ..." al proyecto de prueba: NuevoTipoTests
3. Agregar nuevo método de prueba a la clase de prueba recién creada: NuevoMetodoTest
4. Codificar el nuevo método de prueba que compruebe el resultado obtenido para una entrada específica.  

```
[TestMethod()]
public void NuevoMetodoTest() {
 var arrange = new NuevoTipo();
 Assert.AreEqual("OK", arrange.NuevoMetodo());
}
```
5. Generar el tipo a partir del código de prueba colocando el cursor en NuevoTipo y, en el menú de bombilla, Generar nuevo tipo. En el cuadro de diálogo Generar tipo, establecer el proyecto de código fuente como destino.

© JMA 2020. All rights reserved

## Ciclo de vida

6. Generar el método a partir del código de prueba colocando el cursor en NuevoMetodo y, en el menú de bombilla, Generar método.
7. Ejecutar la prueba unitaria, la prueba se ejecuta pero no se supera: Se produce excepción NotImplementedException.
8. A partir del código de prueba colocando el cursor en NuevoMetodo y, clic derecho, Ir a definición. Sustituir el código generado por código constante que supere la aserción.  

```
public string NuevoMetodo() {
 return "OK";
}
```
9. Ejecutar la prueba unitaria, la prueba se ejecuta y se supera. En caso contrario depurar el código del método de prueba y el código constante para volver a ejecutar la prueba unitaria hasta que se supere.
10. Una vez comprobado el correcto funcionamiento, se sustituye el código constante por la implementación real.

© JMA 2020. All rights reserved

## Ciclo de vida

11. Ejecutar la prueba unitaria y hay que modificar la implementación hasta que se supere la prueba, sin modificar el código del método de prueba.
12. Se pueden añadir nuevos métodos de prueba que cubran el casos de prueba completo. Hay que evitar modificar las pruebas existentes que se completan correctamente.
13. Refactorizar el código implementado para mejorarlo sin añadir funcionalidad ni modificar el código del método de prueba.
14. Ejecutar la prueba unitaria que debe seguir siendo superada, en caso contrario habrá modificar código refactorizado hasta que se supere la prueba.

© JMA 2020. All rights reserved

# Refactorizar el código en pruebas

- Una refactorización es un cambio que está pensado para que el código se ejecute mejor o para que sea más fácil de comprender.
- No está pensado para alterar el comportamiento del código y, por tanto, no se cambian las pruebas.
- Se recomienda realizar los pasos de refactorización independientemente de los pasos que amplían la funcionalidad.
- Mantener las pruebas sin cambios aporta la confianza de no haber introducido errores accidentalmente durante la refactorización.

© JMA 2020. All rights reserved

## Refactorización

- La refactorización es el proceso que consiste en mejorar el código una vez escrito cambiando su estructura interna sin modificar su comportamiento externo.
- C# proporciona los siguientes comandos de refactorización en el menú Refactorización:
  - Extraer método
  - Cambiar nombre
  - Encapsular campo
  - Extraer interfaz
  - Promocionar una variable local a parámetro
  - Quitar parámetros
  - Reordenar parámetros

© JMA 2020. All rights reserved

## Probando los modelos

- El patrón MVC propugna una separación clara de los datos y la lógica de negocio de una aplicación del interfaz de usuario y del módulo encargado de gestionar los eventos y las comunicaciones, esto permite la prueba aislada del código independientemente de la presentación.
- Los modelos, con la lógica de negocio y la capa de acceso a datos, contarán con una exhaustiva batería de pruebas unitarias y de integración.
- En las pruebas unitarias, todas las dependencias se sustituirán por dobles de prueba, ASP.NET Core facilita esta sustitución a través de la inyección de dependencias en los constructores.

© JMA 2020. All rights reserved

## Probando los controladores

- Los controladores, por definición, deberían contar con una lógica mínima: delegar en el modelo y generar el ActionResult a devolver por el método de acción (las vistas no llegan a representarse).
- En las pruebas unitarias, todas las dependencias de modelos y servicios se sustituirán por dobles de prueba, ASP.NET Core facilita esta sustitución a través de la inyección de dependencias en los constructores.
- Los ActionResult son objetos que permiten aserciones que verifiquen que los resultados son los esperados a través de las propiedades ViewData, Model, ViewName, StatusCode, ...

© JMA 2020. All rights reserved

## Probando los controladores

```
[Fact]
public async Task Index_ReturnsAViewResult_WithAListOfBrainstormSessionsTest() {
 // Arrange
 var mockRepo = new Mock<IBrainstormSessionRepository>();
 mockRepo.Setup(repo => repo.ListAsync()).ReturnsAsync(GetTestSessions());
 var controller = new HomeController(mockRepo.Object);

 // Act
 var result = await controller.Index();

 // Assert
 var viewResult = Assert.IsType<ViewResult>(result);
 var model = Assert.IsAssignableFrom<IEnumerable<StormSessionViewModel>>(
 viewResult.ViewData.Model);
 Assert.Equal(2, model.Count());
}
```

© JMA 2020. All rights reserved

## Probando las vistas

- Aseguremos que todo lo que llega a la vista esta testeado.
- Probamos lógica de navegación.
- Probamos las validaciones.
- La vista se puede probar navegando el sitio manualmente o con alguna herramienta automatizada.
- A medida que las aplicaciones crecen en tamaño y complejidad, se vuelve imposible depender de pruebas manuales para verificar la corrección de las nuevas características, errores de captura y avisos de regresión.
- Las pruebas unitarias son la primera línea de defensa para la captura de errores, pero a veces las circunstancias requieran la integración entre componentes que no se pueden capturar en una prueba unitarias.
- Las pruebas de extremo a extremo (E2E: end to end) se hacen para encontrar estos problemas.

© JMA 2020. All rights reserved

# Selenium WebDriver

- El Selenium es un conjunto de herramientas para automatizar los navegadores web, robot que simula la interacción del usuario con el navegador, originalmente pensado como entorno de pruebas de software para aplicaciones basadas en la web.
- Como principales herramientas Selenium cuenta con:
  - Selenium IDE:
    - una herramienta para grabar y reproducir secuencias de acciones con el navegador que permite crear pruebas sin usar un lenguaje de scripting para pruebas.
  - Selenium Core:
    - API para escribir pruebas automatizadas y de regresión en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python.
  - WebDriver:
    - interfaces que permite ejecutar las pruebas de forma nativa usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX.
  - Selenium Grid:
    - Permite ejecutar muchas pruebas de un mismo grupo en paralelo o pruebas en múltiples entornos. Tiene la ventaja que un conjunto de pruebas muy grande puede dividirse en varias máquinas remotas para una ejecución más rápida o si se necesitan repetir las mismas pruebas en múltiples entornos.

© JMA 2020. All rights reserved

## DESPLIEGUE Y PUESTA EN PRODUCCIÓN DE APLICACIONES WEB

© JMA 2020. All rights reserved

# Tareas de la publicación

- Copiar la aplicación Web
- Cambiar los valores de appsettings.json que deben ser diferentes en el entorno del destino.
- Propagar los datos o estructuras de datos en las bases de datos que se usan en la aplicación web.
- Configurar valores de IIS en el equipo de destino, como el grupo de aplicaciones, el método de autenticación, si se permite el examen de directorios y el control de errores.
- Instalar certificados de seguridad.
- Establecer valores en el Registro del equipo de destino.
- Instalar ensamblados de aplicación en la GAC en el equipo de destino.

© JMA 2020. All rights reserved

# Metodología

- Pasos (requieren una checklist):
  - Preparación
  - Despliegue
    - Paquetes de implementación web
    - Publicación con un solo clic
  - Verificación
- Existe una extensión de IIS, denominada Web Deploy, que puede automatizar la mayor parte de las tareas de implementación:
  - Compilación y copia de los archivos de la aplicación.
  - Configuración de valores de IIS, como el grupo de aplicaciones, el método de autenticación, si se permite el examen de directorios y el control de errores.
  - Generación y ejecución scripts de bases de datos que se usan para propagar los cambios a los datos o estructuras de la base de datos.
  - Transformación de valores de la configuración como la configuración de depuración, o cadenas de conexión.

© JMA 2020. All rights reserved



## Preparación del archivo appsettings.json

- Para adaptar el fichero de configuración al entorno de destino es necesario:
  - Modificar:
    - Cadenas de conexión.
    - Valores de autenticación, pertenencia ...
    - Configuración de depuración.
    - Configuración de traza.
    - Errores personalizados.
  - Quitar información confidencial.
- Para automatizar el proceso, es posible disponer de versiones del fichero de configuración al entorno adaptadas a los diferentes entornos:
  - appsettings.Development.json
  - appsettings.Staging.json
  - appsettings.Production.json

© JMA 2020. All rights reserved

## Preparación de la base de datos

- Al implementar una aplicación web que usa una base de datos de SQL Server, es posible que se tenga que propagar igualmente estructuras de datos, datos o ambas cosas.
- Para ello, Visual Studio puede crear automáticamente scripts (archivos .sql) desde la base de datos de origen para la base de datos de destino y estos scripts se pueden incluir en el paquete web.
- Para la actualización de la base de datos, también puede incluir scripts de SQL Server personalizados y especificar el orden en el que se deben ejecutar los scripts.
- Web Deploy ejecuta los scripts en el servidor de destino cuando se instala el paquete.

© JMA 2020. All rights reserved

## Preparación Web Deploy

- Especificar opciones de implementación de la aplicación web (pestaña Empaquetar/publicar web de la página Propiedades del proyecto).
  - Si se desea propagar la configuración de IIS del proyecto al entorno del destino, hay que marcar la casilla Incluir la misma configuración de IIS existente en el Administrador de IIS.
- Especificar opciones de implementación de base de datos (pestaña Empaquetar/publicar SQL de la página Propiedades del proyecto).
- Especificar las migraciones de Entity Framework, en caso de ser necesario, que actualice la base de datos.

*Web Deploy debe estar instalado en el equipo de desarrollo (se instala al instalar Visual Studio) y la misma versión de Web Deploy debe estar instalada en el servidor web de destino.*

© JMA 2020. All rights reserved

## Publicación con un solo clic

- Para poder publicar la aplicación web, debe crear un perfil de publicación que especifique cómo se publica. Se pueden crear tantos perfiles como sean necesarios para los diferentes escenarios (producción, preproducción, ...).
- Los posibles métodos de publicación son: Azure, Docker, Servidor Web(con Web Deploy), Sistema de archivos (Carpeta) y Servidor FTP.
- Sistema de archivos (Carpeta) y Servidor FTP solo copian archivos, no propagan configuraciones de IIS ni bases de datos, ni realizan otras tareas que podrían ser necesarias para implementar una aplicación web.
- La publicación con un solo clic está diseñada para simplificar la publicación repetitiva.

© JMA 2020. All rights reserved