Capítulo 3
Novedades de ASP.NET Core

1. Introducción

ASP.NET existe desde 2002 y se han introducido muchos cambios en el framework desde su primera versión. Es importante recordar una cosa sobre ASP.NET Core: la nueva plataforma web de Microsoft no es en absoluto una continuación de la versión 4.6 del framework que ya conocemos, sino más bien una renovación que debería marcar una nueva era para la tecnología de Microsoft en la Web moderna.

Algunos dirán que el framework no ha cambiado tanto (especialmente la parte MVC), pero es realmente 'bajo el capó' donde los cambios han sido más profundos, empezando por el espacio de nombres `System.Web`, que ya no existe. A continuación, anunciado como multiplataforma, ASP.NET Core es más modular que en años anteriores. A través de ***NuGet*** para los componentes del servidor y después a través de ***Grunt*** o ***Gulp*** para la parte cliente del sitio web, el nuevo framework también se beneficia de un nuevo runtime, llamado ***CoreCLR***, que permite ejecutar una aplicación web de Microsoft en Linux o Mac.

2. Nuevas herramientas de código abierto

ASP.NET Core viene con toda una nueva gama de herramientas de código abierto para gestionar nuevos proyectos web. Afortunadamente para los desarrolladores, todas estas herramientas se han reunido en una única interfaz de línea de comandos: dotnet.

2.1 El entorno de ejecución dotnet

dotnet ha sido diseñado para que las aplicaciones .NET funcionen en plataformas Windows, Mac y Linux, sin tener que desarrollar un runtime distinto para cada una de ellas. Es tanto un entorno de ejecución como un SDK, con todo lo necesario para hacer funcionar aplicaciones web ASP.NET multiplataforma.

Totalmente orientado ***package-first***, Microsoft ha llevado el concepto de modularidad un paso más allá, permitiendo incluso que el entorno de ejecución incorpore, gestione y cree automáticamente los paquetes que necesita a través de NuGet. dotnet se puede usar en diferentes frameworks (.NET Core o el framework .NET Full) y generar paquetes NuGet directamente. Además, dotnet incluye el nuevo motor de ejecución CoreCLR, diseñado específicamente para los problemas de compatibilidad en otras plataformas.

dotnet está integrado en Visual Studio 2015 para ofrecer una experiencia más rica al desarrollador, pero el entorno de ejecución también se puede controlar desde la línea de comandos. En un proyecto ASP.NET Core, puede añadir herramientas de Microsoft y utilizar dotnet para controlar el proyecto desde la línea de comandos. Estas herramientas se añaden al mismo tiempo que el paquete NuGet en el archivo *.csproj:*

```
<ItemGroup>
    <DotNetCliToolReference Inclu-de="Microsoft.VisualStudio.Web.
CodeGeneration.Tools" Version="2.0.4" />
    <DotNetCliToolReference Include="BundlerMinifier.Core"
Version="2.0.238" />
  </ItemGroup>
```

Los comandos declarados se pueden utilizar, por ejemplo, para lanzar una herramienta de generación de código, antes conocida como *scaffolding*. Esto se puede utilizar para generar el controlador y las vistas que corresponden a un modelo de datos muy específico. El comando se puede utilizar a través de `dotnet <command>`:

```
dotnet Microsoft.VisualStudio.Web.CodeGeneration.Tools
```

Dentro de la propia aplicación, es posible utilizar servicios de dotnet a través de ciertas interfaces C# disponibles mediante inyección de dependencias. Servicios como `IServiceEnvironment` se pueden utilizar para modificar la configuración del entorno de ejecución mientras se ejecuta la propia aplicación.

La utilidad `dotnet` también contiene una lista de comandos predefinidos para realizar determinadas acciones en el servidor ASP.NET Core:

– `dotnet` new: inicializa un proyecto sencillo de consola C#.
– `dotnet restore`: restaura las dependencias del proyecto según *.csproj*.
– `dotnet build`: crea la aplicación .NET Core.
– `dotnet publish`: publica una aplicación .NET Core.
– `dotnet run`: lanza la aplicación desde el código fuente.
– `dotnet test`: ejecuta las pruebas utilizando el *test runner* definido en el *.csproj*.
– `dotnet pack`: crea un paquete NuGet a partir del código fuente.

Esta herramienta admite varios procesos para lanzar la aplicación ASP.NET Core. El primero consiste simplemente en restaurar los paquetes necesarios para la aplicación y, a continuación, lanzar el proyecto desde el código fuente.

```
dotnet new
dotnet restore
dotnet run
```

Sin embargo, la utilidad también se puede utilizar para lanzar la aplicación directamente desde una DLL, una vez generado el proyecto.

```
dotnet build
dotnet run bin/Debug/netcoreapp1.0/test-app.dll
```

Con los dos procesos anteriores, la herramienta demuestra que es capaz de ejecutar varios tipos de aplicaciones:

– Aplicaciones "portátiles", es decir, aplicaciones que dependen de la versión de .NET Core instalada en la máquina. Esto significa que este tipo de aplicaciones se podrán ejecutar en distintas instalaciones de .NET Core, independientemente de los sistemas operativos utilizados. Las aplicaciones "portátiles" también permiten centralizar las librerías .NET: por tanto, solo incrustarán librerías externas.
– Aplicaciones "autónomas", es decir, aplicaciones que contienen todas las dependencias que necesitan para funcionar, incluido el runtime de .NET Core, que es parte integrante de la aplicación. Esto facilita el despliegue de la aplicación, pero hace que el paquete sea más pesado. Además, la aplicación debe especificar la versión del runtime que se utilizará en el *.csproj*.

2.2 La utilidad dotnet restore

Un proyecto web ASP.NET Core tiene varias dependencias de paquetes externos, a menudo de NuGet. Para poner en marcha una aplicación web, primero hay que restaurar los paquetes necesarios para que el proyecto funcione correctamente. Esta es una de las novedades de ASP.NET Core: el proyecto solo incorpora lo que necesita, por lo que ha sido necesario diseñar una herramienta que permita restaurar estas dependencias, de forma multiplataforma y totalmente transparente. dotnet restore permite llevar a cabo esta restauración.

Integrado en .NET Core e instalado con dotnet, dotnet restore permite analizar el proyecto ASP.NET Core y recuperar de la nube los paquetes que necesita. Visual Studio 2015 utiliza automáticamente esta utilidad cuando se actualizan las dependencias del proyecto. Sin embargo, puede iniciar el proceso manualmente mediante el siguiente comando:

```
> dotnet restore
```

Observación

Hay que tener en cuenta, sin embargo, que en la consola desde la que se lanza el comando es necesario estar ubicado en la raíz del proyecto. dotnet restore inspeccionará el archivo .csproj para identificar las dependencias a restaurar.

Gracias a dotnet restore y a su facilidad de uso, es muy sencillo restaurar las dependencias de un proyecto ASP.NET Core, independientemente de la plataforma.

2.3 Gestión de paquetes NuGet con dotnet pack

La herramienta `dotnet` pack se utiliza para generar un paquete NuGet a partir de un proyecto .NET. Se puede utilizar de varias maneras para gestionar los paquetes NuGet. En primer lugar, podemos utilizar el siguiente comando, en la raíz de un proyecto .NET, para simplemente generar un paquete NuGet:

```
dotnet pack mi_proyecto.csproj
```

Este comando generará un paquete NuGet a partir del proyecto especificado y lo guardará en el directorio `bin\Debug` o `bin\Release`, dependiendo del modo de compilación utilizado.

Para incluir información sobre la versión y los metadatos en el paquete NuGet generado, podemos utilizar las opciones --`version` y --`metadata`:

```
dotnet pack mi_proyecto.csproj --version 1.2.3
--metadata authors="John Doe"
```

En este ejemplo, hemos especificado la versión `1.2.3` del paquete y añadido un metadato `authors` con el valor "`John Doe`".

Para publicar el paquete NuGet generado en un repositorio NuGet, podemos utilizar la opción --`publish`, especificando la URL del repositorio:

```
dotnet pack mi_proyecto.csproj --publish https://mynugetrepo.com
```

Este comando publicará el paquete NuGet generado en el repositorio NuGet especificado.

Para más información, puede consultar la documentación oficial o utilizar el comando `dotnet pack --help` para mostrar la lista completa de opciones disponibles:

– `--output`: directorio de salida de los paquetes generados.
– `--no-build`: genera un paquete NuGet sin lanzar la generación del proyecto .NET.
– `--no-restore`: genera un paquete NuGet sin restaurar los paquetes NuGet del proyecto.
– `--include-symbols`: incluye símbolos de compilación junto al paquete generado en la carpeta de salida.
– `--include-source`: incluye archivos PDB y archivos fuente. Las fuentes irán a la carpeta `src`.
– `--serviceable`: define el nivel de mantenimiento del paquete.
– `--nologo`: no muestra el logotipo cuando se inicia el comando.
– `--interactive`: permite esperar la interacción del usuario si es necesario.
– `--verbosity`: indica la verbosidad de los registros que se muestran al ejecutar el comando.
– `--version-suffix`: define el valor de la propiedad `$(VersionSuffix)`que se utilizará al generar el proyecto.
– `--configuration`: define la configuración que se utilizará durante la generación. Los valores pueden ser `Debug` o `Release`.
– `--use-current-runtime`: define si se debe utilizar el runtime actual como runtime de destino.

El comando `dotnet` pack es muy útil para gestionar paquetes NuGet para proyectos .NET y permite a los desarrolladores crear y publicar paquetes fácilmente.

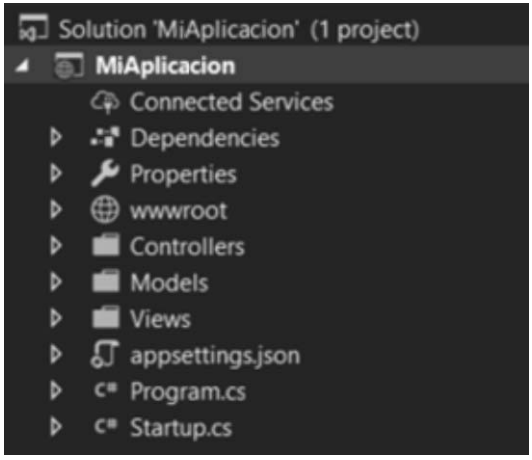
3. La estructura de una solución

Una solución ASP.NET Core es la base de un proyecto web que utiliza tecnologías Microsoft. Permite desplegar rápidamente un sitio y estructurar el código utilizado para ejecutar la aplicación. Una solución puede incluir código tanto del lado del servidor como del lado del cliente, a la vez que incluye mecanismos para separar ambas partes. Este capítulo examinará los diferentes componentes de una solución ASP.NET Core y explicará sus funciones en la configuración o despliegue de la aplicación web.

3.1 Archivos .csproj

Un proyecto ASP.NET Core implementa una nueva filosofía de diseño de aplicaciones web de Microsoft que se inspira en gran medida en el código abierto.

La nueva plantilla tiene este aspecto:



Nueva plantilla de proyecto ASP.NET Core

Lo primero que hay que saber es que la disposición de los proyectos en la solución no es fija. De hecho, mediante un sistema de referencia, es muy fácil modificar la ubicación de los proyectos para seguir sus propias convenciones de nomenclatura y estructura. Un proyecto básico ASP.NET Core MVC se compone de los siguientes elementos:

– **Una solución**: es el vínculo entre todos sus proyectos. El archivo .sln permite agrupar en un único archivo toda la información importante sobre sus proyectos: nombre y ruta del proyecto, configuración de compilación del proyecto (Debug, Release, etc.). También reúne diversa información contextual sobre la solución, como la versión mínima de Visual Studio necesaria para abrir el proyecto. El formato del archivo es muy similar a YAML.

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 15
VisualStudioVersion = 15.0.28010.2016
MinimumVisualStudioVersion = 10.0.40219.1
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") =
"MiAplicacion", "MiAplicacion\MiAplicacion.csproj",
"{GUID-PROJET}
EndProject
Global
   GlobalSection(SolutionConfigurationPlatforms) = preSolution
       Debug|Any CPU = Debug|Any CPU
       Release|Any CPU = Release|Any CPU
   EndGlobalSection
   GlobalSection(ProjectConfigurationPlatforms) = postSolution
       {GUID-PROJET}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
          {GUID-PROJET}.Debug|Any CPU.Build.0 = Debug|Any CPU
       {GUID-PROJET}.Release|Any CPU.ActiveCfg = Release|Any CPU
       {GUID-PROJET}.Release|Any CPU.Build.0 = Release|Any CPU
   EndGlobalSection
   GlobalSection(SolutionProperties) = preSolution
       HideSolutionNode = FALSE
   EndGlobalSection
   GlobalSection(ExtensibilityGlobals) = postSolution
       SolutionGuid = {1D73F40D-E395-4018-B0A3-DD55BE367A54}
   EndGlobalSection
EndGlobal
```

- La sección **Connected Services**: se utiliza para conectar servicios externos al proyecto, como Application Insights o Azure Key Vault.
- La sección **Dependencies**: agrupa todas las dependencias del proyecto y las clasifica en tres categorías: analizadores (analizadores de código, por ejemplo), paquetes NuGet y dependencias del SDK .NET Core.
- La sección **Properties**: incluye un archivo de configuración, **launchSettings.json**, utilizado para configurar los perfiles de inicio del proyecto. Este archivo se describe con más detalle más adelante en esta sección.
- La carpeta **wwwroot**: esta carpeta contiene todos los archivos destinados únicamente al cliente web: archivos CSS, JavaScript, imágenes y recursos de todo tipo.
- Carpetas **Controllers**, **Models** y **Views**: carpetas que contienen el código del proyecto web MVC. La asignación de nombres es muy importante, porque permite al framework encontrar los controladores según las plantillas de enrutamiento definidas en la configuración del proyecto.
- Archivos **appsettings.json**: archivos de configuración del proyecto para diferentes entornos. Estos archivos contendrán, por ejemplo, cadenas de conexión a bases de datos o constantes que se pueden configurar fácilmente fuera del código fuente o que pueden diferir en función del entorno de ejecución de la aplicación.
- La clase **Program.cs**: el punto de entrada de la aplicación. Aquí es donde se lanza el servidor web Kestrel con la configuración inicial de la infraestructura necesaria para que funcione correctamente (puerto, integración con IIS, etc.).
- La clase **Startup.cs**: clase de configuración del proyecto web. Define los servicios que se utilizarán en toda la aplicación. También define el pipeline HTTP para las peticiones entrantes y salientes.