

Manual de Angular 2



Alberto Basalo
Miguel Angel Alvarez



desarrolloweb.com/manuales/manual-angular-2.html

Introducción: Manual de Angular 2

Comenzamos la redacción a nuestro Manual de Angular 2, la evolución del framework Javascript más popular para el desarrollo de aplicaciones del lado del cliente.

En esta página irás viendo los artículos del Manual de Angular según los vayamos publicando a lo largo de los próximos meses.

Nuestro objetivo es recorrer las piezas principales de este potente framework y estudiar los mecanismos para el desarrollo de aplicaciones web universales con Angular 2. De momento encontrarás una estupenda introducción al desarrollo con el nuevo Angular, y abordaremos muchos más detalles en breve.

Encuentras este manual online en:

<http://desarrolloweb.com/manuales/manual-angular-2.html>

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Alberto Basalo

Alberto Basalo es experto en Angular y otras tecnologías basadas en Javascript, como NodeJS y MongoDB. Es director de Ágora Binaria, empresa dedicada al desarrollo de aplicaciones y a la formación a través de Academia Binaria.



Miguel Angel Alvarez

Miguel es fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.



Introducción al desarrollo con Angular 2

En esta primera parte del manual vamos a dar los primeros pasos para desarrollar aplicaciones con Angular 2. Veremos cómo crear el esqueleto de la aplicación básica y conoceremos los principales componentes de todo proyecto.

Introducción a Angular 2

Qué es Angular 2. Por qué se ha decidido escribir desde cero el popular framework Javascript y qué necesidades y carencias resuelve con respecto a su antecesor.

Angular 2 ha cambiado tanto que hasta el nombre es distinto. Lo conocíamos como "AngularJS" y ahora es sólo "Angular". No deja de ser una anécdota que hayan eliminado el "JS" hasta del nombre del dominio, pero es representativo. No porque ahora Angular no sea Javascript, sino porque es evolución radical.

Angular 2 es otro framework, no simplemente una nueva versión. A los que no conocían Angular 1 ésto les será indiferente, pero los que ya dominaban este framework sí deben entender que el conocimiento que necesitan adquirir es poco menos que si comenzasen desde cero. Obviamente, cuanto más experiencia en el desarrollo se tenga, más sencillo será lanzarse a usar Angular 2 porque muchas cosas sonarán de antes.

En este artículo encontrarás un poco de historia relacionada con Angular 1 y 2, junto con los motivos por los que Angular 2 es otro framework totalmente nuevo, que rompe compatibilidad hacia atrás.



Retos y necesidades de la nueva versión de Angular

Desde su creación hace ya más de 4 años, Angular ha sido el framework preferido por la mayoría de los desarrolladores Javascript. Este éxito ha provocado que los desarrolladores quieran usar el framework para más y más cosas.

De ser una plataforma para la creación de Web Apps, ha evolucionado como motor de una enorme cantidad de proyectos del ámbito empresarial y de ahí para aplicaciones en la Web Mobile Híbrida, llevando la tecnología al límite de sus posibilidades.

Es el motivo por el que comenzaron a detectarse problemas en Angular 1, o necesidades donde no se

alcanzaba una solución a la altura de lo deseable. Son las siguientes.

Javascript: Para comenzar encontramos problemas en la creación de aplicaciones debido al propio Javascript. Es un lenguaje con carácter dinámico, asíncrono y de complicada depuración. Al ser tan particular resulta difícil adaptarse a él, sobre todo para personas que están acostumbradas a manejar lenguajes más tradicionales como Java o C#, porque muchas cosas que serían básicas en esos lenguajes no funcionan igualmente en Javascript.

Desarrollo del lado del cliente: Ya sabemos que con Angular te llevas al navegador mucha programación que antes estaba del lado del servidor, comenzando por el renderizado de las vistas. Esto hace que surjan nuevos problemas y desafíos. Uno de ellos es la sobrecarga en el navegador, haciendo que algunas aplicaciones sean lentas usando Angular 1 como motor.

Por otra parte tenemos un impacto negativo en la primera visita, ya que se tiene que descargar todo el código de la aplicación (todas las páginas, todas las vistas, todas las rutas, componentes, etc), que puede llegar a tener un peso de megas.

Nota: A partir de la segunda visita no es un problema, porque ya están descargados los scripts y cacheados en el navegador, pero para un visitante ocasional sí que representa un inconveniente grande porque nota que la aplicación tarda en cargar inicialmente.

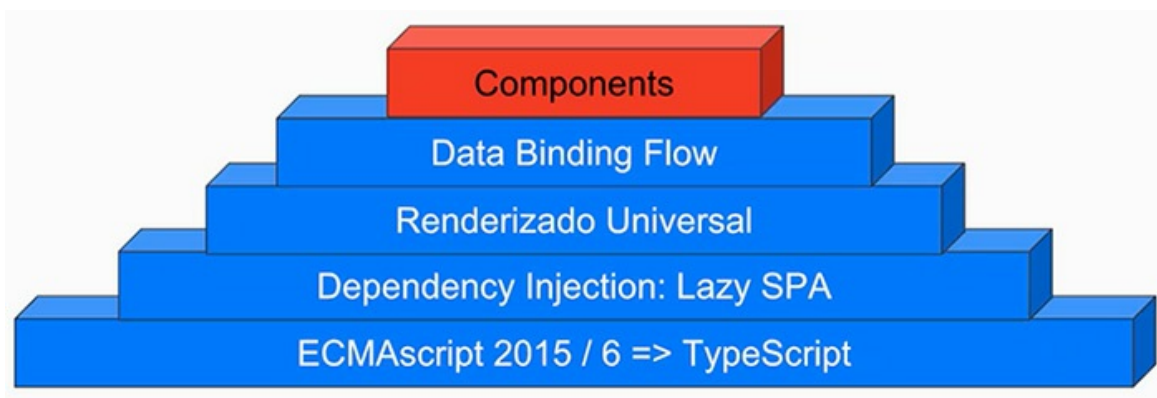
Los intentos de implementar Lazy Load, o carga perezosa, en el framework en su versión 1.x no fueron muy fructíferos. Lo ideal sería que no fuese necesario cargar toda tu aplicación desde el primer instante, pero es algo muy difícil de conseguir en la versión precedente por el propio inyector de dependencias de Angular 1.x.

Otro de los problemas tradicionales de Angular era el impacto negativo en el SEO, producido por un renderizado en el lado del cliente. El contenido se inyecta mediante Javascript y aunque se dice que Google ha empezado a tener en cuenta ese tipo de contenido, las posibilidades de posicionamiento de aplicaciones Angular 1 eran mucho menores. Nuevamente, debido a la tecnología de Angular 1, era difícil de salvar.

Soluciones implementadas en el nuevo Angular 2

Todos esos problemas, difíciles de solucionar con la tecnología usada por Angular 1, han sido los que han impulsado a sus creadores a desarrollar desde cero una nueva versión del framework. La nueva herramienta está pensada para dar cabida a todos los usos dados por los desarrolladores, llevar a Javascript a un nuevo nivel comparable a lenguajes más tradicionales, siendo además capaz de resolver de una manera adecuada las necesidades y problemas de la programación del lado del cliente.

En la siguiente imagen puedes ver algunas de las soluciones aportadas en Angular 2.



TypeScript / Javascript: Como base hemos puesto a Javascript, ya que es el inicio de los problemas de escalabilidad del código. Ayuda poco a detectar errores y además produce con facilidad situaciones poco deseables.

Nota: Con ECMAScript 6 ya mejora bastante el lenguaje, facilitando la legibilidad del código y solucionando diversos problemas, pero todavía se le exige más. Ya puestos a no usar el Javascript que entienden los navegadores (ECMAScript 5), insertando la necesidad de usar un transpilador como [Babel](#), podemos subir todavía un poco de nivel y usar TypeScript.

Angular 2 promueve el uso de TypeScript a sus desarrolladores. El propio framework está desarrollado en TypeScript, un lenguaje que agrega las posibilidades de ES6 y el futuro ES7, además de un tipado estático y ayudas durante la escritura del código, el refactoring, etc. pero sin alejarte del propio Javascript (ya que el código de Javascript es código perfectamente válido en TypeScript).

La sugerencia de usar TypeScript para desarrollar en Angular es casi una imposición porque la documentación y los generadores de código están pensados en TypeScript. Se supone que en futuro también estarán disponibles para Javascript, pero de momento no es así. De todos modos, para la tranquilidad de muchos, TypeScript no agrega más necesidad de procesamiento a las aplicaciones con Angular 2, ya que este lenguaje solamente lo utilizas en la etapa de desarrollo y todo el código que se ejecuta en el navegador es al final Javascript, ya que existe una transpilación previa.

Nota: Puedes saber más sobre este superset de Javascript en el artículo de [introducción a TypeScript](#).

Lazy SPA: Ahora el inyector de dependencias de Angular no necesita que estén en memoria todas las clases o código de todos los elementos que conforman una aplicación. En resumen, ahora con Lazy SPA el framework puede funcionar sin conocer todo el código de la aplicación, ofreciendo la posibilidad de cargar más adelante aquellas piezas que no necesitan todavía.

Renderizado Universal: Angular nació para hacer web y renderizar en HTML en el navegador, pero ahora el renderizado universal nos permite que no solo se pueda renderizar una vista a HTML. Gracias a ésto, alguien podría programar una aplicación y que el renderizado se haga, por ejemplo, en otro lenguaje nativo para un dispositivo dado.

Otra cosa que permite el renderizado universal es que se use el motor de renderizado de Angular del lado del servidor. Es una de las novedades más interesantes, ya que ahora podrás usar el framework para renderizar vistas del lado del servidor, permitiendo un mejor potencial de posicionamiento en buscadores de los contenidos de una aplicación. Esta misma novedad también permite reducir el impacto de la primera visita, ya que podrás tener vistas "precocinadas" en el servidor, que puedes enviar directamente al cliente.

Data Binding Flow: Uno de los motivos del éxito de Angular 1 fue el data binding, pero éste tenía un coste en tiempo de procesamiento en el navegador, que si bien no penalizaba el rendimiento en todas las aplicaciones sí era un problema en aquellas más complejas. El flujo de datos ahora está mucho más controlado y el desarrollador puede direccionarlo fácilmente, permitiendo optimizar las aplicaciones. El resultado es que en Angular 2 las aplicaciones pueden llegar a ser hasta 5 veces más rápidas.

Componentes: La arquitectura de una aplicación Angular ahora se realiza mediante componentes. En este caso no se trata de una novedad de la versión 2, ya que en la versión de Angular 1.5 ya se introdujo el [desarrollo basado en componentes](#).

Sin embargo, la componetización no es algo opcional como en Angular 1.5, sino es una obligatoriedad. Los componentes son estancos, no se comunican con el padre a no ser que se haga explícitamente por medio de los mecanismos disponibles, etc. Todo esto genera aplicaciones más mantenibles, donde se encapsula mejor la funcionalidad y cuyo funcionamiento es más previsible. Ahora se evita el acceso universal a cualquier cosa desde cualquier parte del código, vía herencia o cosas como el "Root Scope", que permitía en versiones tempranas de Angular modificar cualquier cosa de la aplicación desde cualquier sitio.

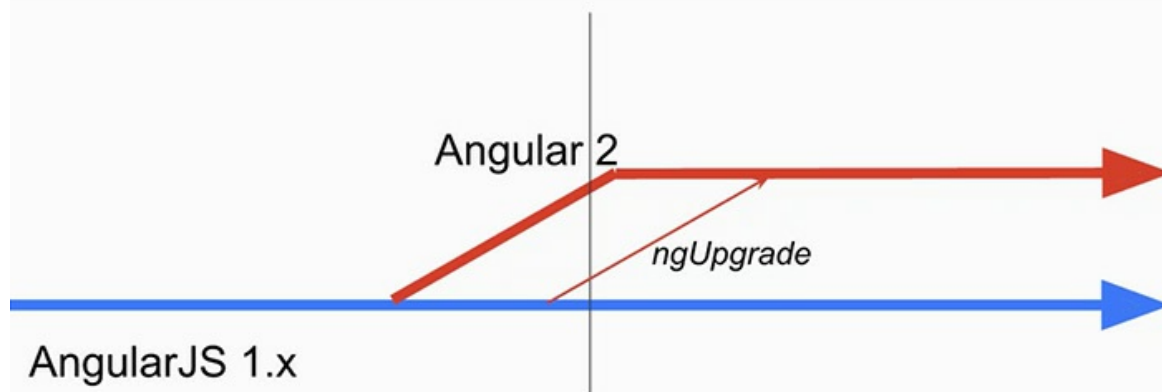
Evolución de las versiones 1 y 2 de Angular

Las versiones de AngularJS 1.x siguen vivas y continuarán dando soporte desde el equipo de Angular. Por tanto, se prevé que después de la actual 1.5 seguirán lanzando actualizaciones, nuevas versiones, etc.

La novedad es que ahora comienza en paralelo la vida de Angular 2 y seguirá evolucionando por su camino. Obviamente, la primera noticia que esperamos todos es que se presente la versión definitiva (ten en cuenta que en el momento de escribir estas líneas Angular 2 está solo en versiones RC, Release Candidate).

De este modo, debe quedar claro, para las personas que tengan aplicaciones en Angular 1.x, que no necesitan actualizarlas en una fecha determinada. Todavía existe un buen tiempo por delante en el que sus proyectos van a estar perfectamente soportados y el código de las aplicaciones perfectamente válido.

Evolución



Han anunciado además que en algún momento habrá algún sistema para hacer el upgrade de las aplicaciones de Angular 1.x a la 2.x. Esto podría permitir Incluso una convivencia, en una misma aplicación, de partes desarrolladas con Angular 1 y otras con la versión 2. La fecha de lanzamiento de ese hipotético "ngUpgrade" está sin confirmar y obviamente tampoco será magia. Veremos en el próximo próximo año lo que sucede, una vez que esa evolución de las versiones 1 y 2 estén conviviendo.

Conclusión

Se espera un futuro muy prometedor a Angular 2. Sus novedades son importantes y permitirán afrontar el futuro sobre una base tecnológica capaz de resolver todas las necesidades y retos actuales. En el Manual de Angular estaremos explicando en los próximos meses cómo usar este framework para desarrollar todo tipo de aplicaciones basadas en Javascript. Si además quieres aprender ya mismo Angular 2, tutorizado por nosotros te recomendamos también el [curso completo de Angular 2 que encuentras en EscuelaIT](#).

Para aquel desarrollador que empieza desde cero en Angular 2 será un bonito camino que le permitirá crecer profesionalmente y ampliar seriamente sus capacidades y el rango de proyectos que sea capaz de acometer. El recorrido puede ser difícil al principio, pero la recompensa será grande.

Por su parte, quien ya tenga experiencia en Angular 1.x siempre le será positiva, sobre todo para los que (a partir de la 1.5) comenzaron a usar componentes. Aunque en Angular 2 cambie la manera de realizar las cosas, le resultará todo más familiar y por tanto su curva de aprendizaje será más sencilla.

Este artículo es obra de *Alberto Basalo*
Fue publicado por primera vez en 09/06/2016
Disponible online en <http://desarrolloweb.com/articulos/introduccion-angular2.html>

Angular CLI

Qué es Angular CLI, el intérprete de línea de comandos de Angular 2 que te facilitará el inicio de proyectos y la creación del esqueleto, o scaffolding, de la mayoría de los componentes de una aplicación Angular.

Después de la [introducción a las características de Angular 2](#) del pasado artículo, en esta ocasión te presentamos una introducción Angular CLI, una de las herramientas esenciales para desarrollar con el nuevo framework Angular 2. Es un paso esencial y necesario antes de comenzar a ver código, puesto que necesitamos esta herramienta para poder iniciar nuestra primera aplicación Angular 2.

Debido a la complejidad de Angular 2, aunque el ejemplo que deseemos desarrollar se trate de un sencillo "Hola mundo", comenzar usando Angular CLI nos ahorrará escribir mucho código y nos permitirá partir de un esquema de aplicación avanzado y capaz de facilitar los flujos de desarrollo, depuración, testing o deploy. Así que vamos con ello.



Qué es Angular CLI

Dentro del ecosistema de Angular 2 encontramos una herramienta fundamental llamada "Angular CLI" (Command Line Interface). Es un producto que en el momento de escribir este artículo todavía se encuentra en fase beta, pero que ya resulta fundamental para el trabajo con el framework.

Angular CLI no es una herramienta de terceros, sino que nos la ofrece el propio equipo de Angular. En resumen, nos facilita mucho el proceso de inicio de cualquier aplicación con Angular, ya que en pocos minutos te ofrece el esqueleto de archivos y carpetas que vas a necesitar, junto con una cantidad de herramientas ya configuradas. Además, durante la etapa de desarrollo nos ofrecerá muchas ayudas, generando el "scaffolding" de muchos de los componentes de una aplicación. Durante la etapa de producción o testing también nos ayudará, permitiendo preparar los archivos que deben ser subidos al servidor, transpilar las fuentes, etc.

Node y npm

Angular CLI es una herramienta NodeJS, es decir, para poder instalarla necesitaremos contar con NodeJS instalado en nuestro sistema operativo, algo que podemos conseguir muy fácilmente yendo a la página de <https://nodejs.org> y descargando el instalador para nuestro sistema.

Además se instala vía "npm". Por npm generalmente no te tienes que preocupar, pues se instala al instalar NodeJS. No obstante es importante que ambas versiones, tanto la de la plataforma Node como el gestor de paquetes npm, se encuentren convenientemente actualizados. En estos momentos como requisito nos piden tener Node 4 o superior.

Nota: Puedes saber la versión de Node que tienes instalada, así como la versión de npm por medio de los comandos:

```
node -v
```

```
npm -v
```

No tienes que saber Node para desarrollar con Angular, pero sí necesitas tenerlo para poder instalar y usar Angular CLI, además de una serie de herramientas fantásticas para desarrolladores.

Instalar Angular CLI

Esto lo conseguimos desde el terminal, lanzando el comando:

```
npm install -g angular-cli
```

Durante el proceso de instalación se instalará el propio Angular CLI junto con todas sus dependencias. La instalación puede tardar varios minutos dependiendo de la velocidad de tu conexión a Internet.

Una vez instalado dispondrás del comando "ng" a partir del cual lanzarás cualquiera de las acciones que se pueden hacer mediante la interfaz de comandos de Angular. Puedes comenzar lanzando el comando de ayuda:

```
ng --help
```

Nota: ng (que se lee "enji") es el apelativo familiar de "Angular" que se conoce desde el inicio del framework.

También encontrarás una excelente ayuda si entras en la [página de Angular CLI](#), navegando por sus secciones, o bien en el propio [repositorio de GitHub angular-cli](#).

Crear el esqueleto de una aplicación Angular 2

Uno de los comandos que puedes lanzar con Angular CLI es el de creación de un nuevo proyecto Angular 2. Este comando se ejecuta mediante "new", seguido del nombre del proyecto que queramos crear.

```
ng new mi-nuevo-proyecto-angular2
```

Lanzado este comando se creará una carpeta igual que el nombre del proyecto indicado y dentro de ella se generarán una serie de subcarpetas y archivos que quizás por su número despiquen a un desarrollador que se inicia en Angular 2. Si es así no te preocupes porque poco a poco nos iremos familiarizando con el código generado.

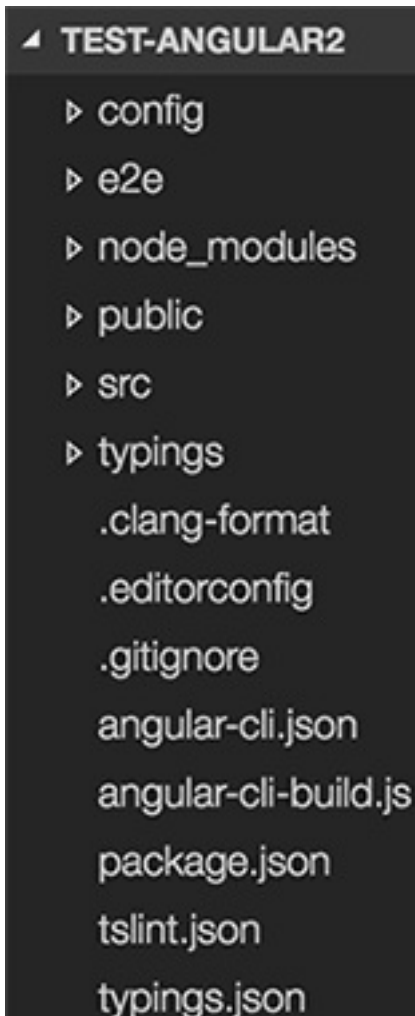
Además, como hemos dicho, se instalarán y se configurarán en el proyecto una gran cantidad de herramientas útiles para la etapa del desarrollo front-end. De hecho, gran cantidad de los directorios y archivos generados al crear un nuevo proyecto son necesarios para que estas herramientas funcionen. Entre otras cosas tendremos:

- Un servidor para servir el proyecto por HTTP
- Un sistema de live-reload, para que cuando cambiamos archivos de la aplicación se refresque el navegador
- Herramientas para testing
- Herramientas para despliegue del proyecto
- Etc.

Una vez creado el proyecto inicial podemos entrar en la carpeta con el comando `cd`.

```
cd mi-nuevo-proyecto-angular2
```

Una vez dentro de esa carpeta encontrarás un listado de archivos y carpetas similar a este:



```
▲ TEST-ANGULAR2
├─ config
├─ e2e
├─ node_modules
├─ public
├─ src
├─ typings
├─ .clang-format
├─ .editorconfig
├─ .gitignore
├─ angular-cli.json
├─ angular-cli-build.js
├─ package.json
├─ tslint.json
└─ typings.json
```

Servir el proyecto desde un web server

Angular CLI lleva integrado un servidor web, lo que quiere decir que podemos visualizar y usar el proyecto sin necesidad de cualquier otro software. Para servir la aplicación lanzamos el comando "serve".

```
ng serve
```

Eso lanzará el servidor web y lo pondrá en marcha. Además, en el terminal verás como salida del comando la ruta donde el servidor está funcionando. Generalmente será algo como esto (pero te sugerimos verificar el puerto en la salida de tu terminal):

```
http://localhost:4200/
```

En la siguiente imagen ves la salida del terminal nuestro.

```
mcMiguel:test-angular2 midesweb$ ng serve
Could not start watchman; falling back to NodeWatcher for file system events.
Visit http://ember-cli.com/user-guide/#watchman for more info.
Livereload server on http://localhost:49152
Serving on http://localhost:4200/

Build successful - 831ms.

Slowest Trees                                | Total
-----+-----
BroccoliTypeScriptCompiler                  | 405ms
vendor                                      | 337ms

Slowest Trees (cumulative)                   | Total (avg)
-----+-----
BroccoliTypeScriptCompiler (1)              | 405ms
vendor (1)                                  | 337ms
```

Podrías modificar el puerto perfectamente si lo deseas, simplemente indicando el puerto deseado con la opción --port:

```
ng serve --port 4201
```

Problema Angular CLI con Broccoli en Windows

El problema más típico que nos podemos encontrar al usar Angular CLI es que no tengamos permisos de administrador. Al intentar poner en marcha el servidor recibirás un error como este:

The Broccoli Plugin: [BroccoliTypeScriptCompiler] failed with: operation not permitted.

Es muy sencillo de solucionar en Windows, ya que simplemente necesitamos abrir el terminal en **modo administrador** (botón derecho sobre el icono del programa que uses para línea de comandos y "abrir como administrador". Eso permitirá que Angular CLI disponga de los permisos necesarios para realizar las tareas

que requiere.

ACTUALIZACIÓN: Esto lo han cambiado en una versión beta de Angular CLI (Beta 6), por lo que ya no hace falta privilegios de administrador para usar las herramientas de línea de comandos de Angular 2.

En Linux o Mac, si te ocurre algo similar, simplemente tendrías que lanzar los comandos como "sudo".

Angular CLI tiene mucho más

En esta pequeña introducción solo te hemos explicado cómo iniciar un nuevo proyecto de Angular 2 y cómo servirlo después por medio del comando serve. Pero lo cierto es que detrás de Angular CLI hay muchas otras instrucciones de gran utilidad. Principalmente, como hemos comentado, encontrarás una gran cantidad de comandos que permiten crear el esqueleto de componentes, directivas, servicios, etc.

A medida que vayamos adentrándonos en el desarrollo con Angular 2 iremos aprendiendo de una forma sencilla y práctica todas las posibilidades que permite esta herramienta. De momento te recomendamos documentarte en el mencionado repositorio de Github.

Este artículo es obra de *Alberto Basalo*
Fue publicado por primera vez en 14/06/2016
Disponible online en <http://desarrolloweb.com/articulos/angular-cli.html>

Análisis de las carpetas de un proyecto básico con Angular 2

Angular 2 exige un marco de trabajo más concreto y avanzado. Vemos los archivos y carpetas que nos hacen falta para comenzar un proyecto básico.

En el pasado artículo abordamos la herramienta [Angular CLI](#) y ahora vamos a introducirnos en el resultado que obtenemos mediante el comando "ng new", que vimos servía para generar el esqueleto básico de una aplicación Angular 2.

No obstante, antes de entrar en materia, vamos a traer dos puntos interesantes. Uno de ellos es una reflexión previa sobre la complejidad de un proyecto "vacío" para una aplicación Angular 2. El otro es una de las preguntas típicas que se hacen cuando una persona se inicia con cualquier herramienta nueva: el editor que se recomienda usar.



Profesionalización

Todas las mejoras que nos ofrece Angular 2 tienen un coste a nivel técnico. Anteriormente (versiones 1.x) podíamos comenzar una aplicación de Angular con un código de inicio muy sencillo y sin necesidad de configurar diversas herramientas frontend. Básicamente podíamos enlazar Angular desde su CDN y con un script muy elemental empezar a usarlo.

Esto ya no es así. Con Angular 2 el número de piezas que necesitamos integrar, incluso en aplicaciones tan sencillas como un "hola mundo", es mucho mayor.

En aplicaciones grandes, donde existía un entorno bien definido por parte del equipo de desarrollo de esa aplicación (linter, transpiler, loader, tester, deployer...) se equilibra. Es decir, Angular no nos exige nada nuevo que ya no estuvieran usando equipos de desarrollo de aplicaciones grandes. Pero lo cierto es que aplicaciones pequeñas tendrán bastante más complejidad. Todo esto nos lleva a que desarrolladores ocasionales encontrarán más difícil el uso del framework en sus pasos iniciales, pero el esfuerzo sin duda merecerá la pena porque los pondrá a un nivel muy superior y eso redundará en su beneficio profesional.

¿Qué editor de código usar con Angular 2?

Se puede usar cualquier editor de código. Es una aplicación HTML y Javascript / TypeScript, por lo que puedes usar cualquier editor de los que vienes usando para cualquiera de estos lenguajes.

Como recomendación se sugiere usar un editor ligero, pero que te facilite la programación. Entre los que encontramos de código libre y gratuitos para cualquier uso están Brackets, Atom o Visual Studio Code. Éste último es quizás el más indicado, porque ya viene configurado con una serie de herramientas útiles y clave para desarrollar con Angular 2 como es el "intellisense" de TypeScript. De todos modos, a través de plugins podrás hacer que tu editor preferido también sea capaz de mostrarte las ayudas en tiempo de programación del compilador de TypeScript.

Archivos y carpetas del proyecto

Impacta un poco que, recién creado un proyecto para Angular 2 por medio de Angular CLI, veamos en la carpeta de archivos varias subcarpetas, y varias de ellas con el contenido de cientos de ficheros. No obstante, no todo es código de tu aplicación, sino muchas veces son carpetas para crear la infraestructura de todo el tooling NodeJS incluido para gestionar una aplicación Angular 2.

Ahora conoceremos las partes que nos hacen falta para comenzar, aunque sin entrar en demasiados detalles.

Todos los archivos del raíz: Seguro que muchos de los lectores reconocen muchos de los archivos que hay dentro, como package.json (descriptor de dependencias npm) o .gitignore (archivos y carpetas que git debería ignorar de este proyecto cuando se añada al repositorio). En resumen, todo lo que encontraremos en esta raíz no son más que archivos que definen nuestro proyecto y configuran el entorno para diversas herramientas.

Nota: Observarás que no hay un index.html, porque esta no es la carpeta raíz de los archivos que se deben servir por el servidor web.

src: Es la carpeta más interesante para ti como desarrollador, ya que es el lugar donde colocarás el código

fuente de tu proyecto. En realidad más en concreto la carpeta "app" que encontrarás dentro de "src" es donde tienes que programar tu aplicación. Observarás que ya viene con diversos contenidos, entre otras cosas el index.html que debe servir como página de inicio. No obstante, no es exactamente el directorio raíz de publicación, porque al desplegar el proyecto los resultados de compilar todos los archivos se llevarán a la carpeta "dist".

En la carpeta src es donde vas a realizar todo tu trabajo como desarrollador. Seguramente otros muchos archivos te resulten familiares, como el favicon.ico.

Verás además varios archivos .ts, que son código fuente TypeScript. Como quizás sepas, los archivos .ts solo existen en la etapa de desarrollo, es decir, en el proyecto que el navegador debe consumir no encontrarás archivos .ts, básicamente porque el navegador no entiende TypeScript. Esos archivos son los que se compilarán para producir el código .js que sí entienda el navegador.

Nota: Si todavía te encuentras reticente al uso de TypeScript no te preocupes, ya que cualquier código Javascript que pongas en ese fichero es código TypeScript válido. Por tanto tú podrías perfectamente escribir cualquier código Javascript dentro de los archivos .ts y todo irá perfectamente. Si además conoces algo de TypeScript y lo quieres usar para facilitarte la vida en tiempo de desarrollo, tanto mejor para ti.

dist: Es la versión de tu aplicación que subirás al servidor web para hacer público el proyecto. En dist aparecerán todos los archivos que el navegador va a necesitar y nunca código fuente en lenguajes no interpretables por él. (Observa que no hay archivos .ts dentro de dist). Ojo, pues muy probablemente tengas que iniciar el servidor web integrado en Angular CLI para que aparezca la carpeta "dist" en el directorio de tu proyecto. Puedes obtener más información sobre cómo lanzar el servidor web en el [artículo de Angular CLI](#).

Public: Es donde colocas los archivos estáticos del proyecto, imágenes y cosas similares que se conocen habitualmente como "assets". Estos archivos también se moverán a "dist" para que estén disponibles en la aplicación una vez subida al servidor web desde donde se va a acceder.

e2e: Es para el desarrollo de las pruebas. Viene de "end to end" testing.

node_modules: Son los archivos de las dependencias que mantenemos vía npm. Por tanto, todas las librerías que se declaren como dependencias en el archivo package.json deben estar descargados en esta carpeta node_modules. Esta carpeta podría haber estado dentro de src, pero está colgando de la raíz porque vale tanto para las pruebas, como para la aplicación cuando la estás desarrollando.

tmp: Es una carpeta que no tocaremos, con archivos temporales que generará Angular CLI cuando esté haciendo cosas.

Typings: Esto son definiciones de tipos usados por las librerías que usa un proyecto en Angular 2. Estos tipos te sirven para que el editor, gracias a TypeScript, pueda informarte con el "intellisense" en el acto de escribir código, sobre las cosas relacionadas con esas librerías.

De momento eso es todo, esperamos que esta vista de pájaro te sirva de utilidad para reconocer la estructura básica de un proyecto a desarrollar con Angular 2. En el siguiente artículo entraremos en detalle

ya sobre el código, analizando por dentro algunas de estas carpetas y archivos generados desde Angular CLI y realizando nuestras primeras pruebas.

Este artículo es obra de *Alberto Basalo*
Fue publicado por primera vez en 20/06/2016
Disponible online en <http://desarrolloweb.com/articulos/analisis-carpetas-proyecto-angular2.html>

Zambullida en el código del proyecto inicial de Angular 2

Comenzamos a analizar el código de un proyecto básico con Angular 2, generado con Angular CLI. Prestamos atención a SystemJS, el index de la aplicación y al componente principal, que podremos editar para comprobar el funcionamiento.

En este artículo vamos a analizar el código de la aplicación inicial que se instala al hacer un nuevo proyecto con Angular CLI. Ten en cuenta que en artículos anteriores del [Manual de Angular 2](#) hemos abordado ya en una primera instancia la estructura de carpetas generada por Angular CLI, además de explicar cómo crearla a través de comandos de consola.

Por tanto, nos centraremos en entender cuál es el flujo de ejecución del código, junto con los archivos que se van incluyendo y recorriendo al ejecutar la aplicación. Obviamente, habrá cosas en las que no podamos entrar todavía en muchos detalles, pero no hay que preocuparse porque serán materia de estudio en sucesivos artículos.



Archivo package.json

Vamos a comenzar por analizar este archivo, ya que muchas cosas son declaradas inicialmente en él. Al analizar este archivo encontraremos el origen de muchas de las librerías que usa Angular 2.

Como debes saber, package.json es un archivo en el que se declaran las dependencias de una aplicación, gestionadas vía npm. Su código es un JSON en el que se declaran varias cosas.

Inicialmente hay que ver la propiedad "dependencies". En ella encontrarás la librería Angular separada en varios módulos. Esta es una de las características de la nueva plataforma de desarrollo promovida por Angular 2, la modularización del código. Encontrarás que una aplicación Angular 2 necesita ya de entrada diversos módulos como son "common", "compiler", "core", etc.

Luego hay una serie de librerías de terceros, no creadas directamente por el equipo de Angular, o bien creadas por ellos mismos pero con un enfoque universal (capaz de usarse en otros tipos de proyectos). Son

"es6-shim", "rxjs", etc.

Todos estos módulos antes se incluían por medio de scripts (etiqueta SCRIPT) en el HTML de la página. Pero en esta versión y gracias a Angular CLI ya viene incorporada una mejor manera de incluir módulos Javascript, por medio de "SystemJS". Observarás que el propio SystemJS está incluido como dependencias "systemjs" y con él podríamos incluir todo tipo de código, no solo JS, sino otros ficheros necesarios como CSS.

Nota: Al describir la [estructura de carpetas del proyecto Angular 2](#) ya explicamos que todas las dependencias de package.json se las instala npm en la carpeta "node_modules".

Entendiendo lo básico de SystemJS

Para comenzar por nuestro análisis del código, vamos a abrir el archivo "src/index.html".

Como dijimos, la carpeta src es donde se encuentran las fuentes de tu proyecto. En ella encontramos un index.html que es la raíz de la aplicación. Todo empieza a ejecutarse a través de este archivo.

Te llamará la atención el código siguiente:

```
{{#each scripts.polyfills}}<script src="{{.}}"></script>{{/each}}
```

Ese código hace un recorrido por una serie de librerías y las va colocando dentro de etiquetas SCRIPT para incluir su código en la aplicación. De momento lo que debes saber sobre este código es que por medio de él se cargan librerías externas que necesita Angular y que hemos visto declaradas en las "dependencias" del archivo package.json. Lo que nos interesa saber es que así se está cargando, entre otras, la librería SystemJS.

Ahora, en el final de index.html encontrarás allí un script. Cuando llegamos a este punto, SystemJS ya está cargado y en este Script se realiza una inicialización de esta librería para cargar los elementos necesarios para comenzar a trabajar:

```
<script>
  System.import('system-config.js').then(function () {
    System.import('main');
  }).catch(console.error.bind(console));
</script>
```

Encontramos el objeto "System", que es una variable global definida por la librería SystemJS. Por medio del método "import" consigue cargar módulos. Aprenderás que se está cargando inicialmente un archivo llamado "system-config.js".

Luego vemos el método then() y catch(). Estos métodos los deberías de reconocer, pues son pertenecientes a un patrón bien conocido por los desarrolladores Javascript, el de promesas. El método then() se ejecutará cuando se termine de cargar el módulo "system-config.js" y el método catch() se ejecutaría en caso que no

se haya podido cargar ese archivo. Gracias a `then()`, después de haber cargado "system-config.js" entonces se cargará "main", que enseguida veremos qué es.

En este punto te preguntará ¿Dónde está system-config.js?. Quizás no lo encuentres, pero veas en la misma carpeta "src" el archivo system-config.ts. Ese es un archivo TypeScript que contiene el código de system-config.js antes de transpilar con el TypeScript Compiler.

Nota: TypeScript es un lenguaje para el programador. Lo que usa el navegador es Javascript. El TypeScript compiler se encargará de hacer esa conversión del .ts a un .js.

El archivo system-config.ts generalmente no lo vas a tener que tocar, porque Angular CLI te lo irá actualizando. Si lo abres sin duda irás reconociendo algunas líneas, cosas que necesitará SystemJS. No vamos a entrar ahora en el detalle, de momento quédate con que es código generado.

Por su parte la referencia a "main" que teníamos antes en los import del index.html `System.import('main')`, es un import. No le ponen ni siquiera la extensión del archivo "main.js" y esto es porque "main" es un alias declarado en el system-config.ts. Fíjate en el código del archivo, en estas líneas:

```
// Apply the CLI SystemJS configuration.
System.config({
  map: {
    '@angular': 'vendor/@angular',
    'rxjs': 'vendor/rxjs',
    'main': 'main.js'
  },
  packages: cliSystemConfigPackages
});
```

El objeto "map" tiene una lista de alias y archivos que se asocian. Siendo que "main" corresponde con "main.js". Nuevamente, no encontrarás un "main.js" entre los archivos del proyecto, en la carpeta "src", porque lo que tendremos es un main.ts que luego se convertirá en el main.js.

Ahora puedes abrir main.js. Verás que su código nuevamente hace uso de SystemJS, realizando diversos imports. Estos imports son como los que conoces en ECMAScript 6 y básicamente lo que te traen son objetos de diversas librerías.

Encontrarás este código en main.js, o algo similar:

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { enableProdMode } from '@angular/core';
import { TestAngular2AppComponent, environment } from '../app';
```

Por ejemplo, estás diciéndole que importe el objeto "bootstrap" de la librería "@angular/platform-browser-dynamic". Esa librería está declarada dentro de "system-config.ts"

Luego verás otras líneas, que es el bootstrap, o inicio de la aplicación Angular. No vamos a entrar en detalle, pero es lo equivalente al "ng-app" que colocabas antes en el código HTML de tu index. Esto, o algo parecido ya se podía hacer con Angular 1 y se conocía como el arranque manual de Angular.

Componente principal de una aplicación

Seguimos analizando index.html y encontramos en el código, en el cuerpo (BODY) una etiqueta que llamará la atención porque no es del HTML tradicional. Es el uso de un componente y su código será algo como:

```
<mi-nombre-proyecto-app>Loading...</mi-nombre-proyecto-app>
```

Ese es el componente raíz de nuestra aplicación Angular 2. Hablaremos de componentes con detalle más adelante. De momento para lo que te interesa a ti, que es reconocer el flujo de ejecución básico, hay que decir que su código está en la carpeta "src/app".

En esa carpeta encontrarás varios archivos del componente que analizaremos con calma más adelante. De momento verás un archivo ".html" que contiene la vista de este componente y un archivo ".css" que contiene el CSS. Si tu componente se llamaba "mi-nombre-proyecto-app", estos archivos se llamarán "mi-nombre-proyecto.component.html" y "mi-nombre-proyecto.component.css".

Para terminar esta primera zambullida al código te recomendamos editar esos archivos. Es código HTML y CSS plano, por lo que no tendrás ningún problema en colocar cualquier cosa, siempre que sea HTML y CSS correcto, claro está.

Para quien use Angular 1 ya reconocerá una estructura como esta:

```
{{title}}
```

Es una expresión que Angular 2 sustituirá por un dato que se declara en el archivo .ts del componente. De momento lo dejamos ahí.

Ejecutar el proyecto

Para comprobar si tus cambios en el HTML del componente han tenido efecto puedes probar el proyecto. La ejecución de esta aplicación ya la vimos en el artículo de [Angular CLI](#), por lo que no necesitarás mayores explicaciones. De todos modos, como resumen, sigue los pasos:

Desde la raíz del proyecto, con el terminal, ejecutamos el comando:

```
ng serve
```

Luego nos dirigimos a la URL que nos indican como resultado de la ejecución del comando, que será algo como:

```
http://localhost:4200/
```

Entonces deberías ver la página funcionando en tu navegador, con el HTML editado tal como lo has dejado en el archivo .html del componente principal.

En futuros artículos profundizaremos sobre muchos de los puntos relatados aquí. De momento creemos que esta introducción al código debe aclararte muchas cosas, o plantearte muchas otras dudas.

Si es tu caso, no te preocupes por sentirte despistado por tantos archivos y tantas cosas nuevas, pues poco a poco iremos familiarizándonos perfectamente con toda esta infraestructura. Para tu tranquilidad, decir que esta es la parte más compleja y que, a partir de aquí, las cosas serán más agradecidas. Si además vienes de Angular 1, empezarás a reconocer mejor las piezas que antes existían en el framework.

Este artículo es obra de *Alberto Basalo*

Fue publicado por primera vez en 29/06/2016

Disponible online en <http://desarrolloweb.com/articulos/codigo-proyecto-inicial-angular2.html>

Introducción a los componentes en Angular 2

Un primer acercamiento al mundo de los componentes en Angular 2, a través del componente inicial que se crea en todo nuevo proyecto. Aprenderás a reconocer sus partes y realizaremos unas modificaciones.

En Angular 2 se desarrolla en base a componentes. Desde la aplicación más básica de Angular 2, el Hola Mundo, todo tiene que comenzar por un componente. Nuestra aplicación "componetizada" constará la verdad de un árbol de componentes de n niveles, desde un componente principal a sus hijos, nietos, etc.

Por si no lo sabes, los componentes son como etiquetas HTML nuevas que podemos inventarnos para realizar las funciones que sean necesarias para nuestro negocio. Pueden ser cosas diversas, desde una sección de navegación a un formulario, o un campo de formulario. Para definir el contenido de esta nueva etiqueta, el componente, usas un poco de HTML con su CSS y por supuesto, un poco de Javascript para definir su funcionalidad.

Básicamente eso es un componente, una de las piezas fundamentales de las aplicaciones en Angular, que nos trae diversos beneficios que mejoran sensiblemente la forma de organización de una aplicación, su mantenimiento, reutilización del código, etc.



Para comenzar a introducirnos en el desarrollo en base a componentes vamos a realizar en este primer artículo un análisis del componente inicial, que contiene de toda aplicación Angular y que podemos encontrar en el proyecto básico creado vía [Angular CLI](#).

Localizar el componente inicial

En el [Manual de Angular 2](#) hemos visto que nuestra aplicación se desarrolla en el directorio "src". Allí encontramos el archivo index.html raíz de la aplicación.. Si lo abres verás que no tiene ningún contenido en sí. Apenas encontrarás el uso de un componente, una etiqueta que no pertenece al HTML y que se llama de una manera similar al nombre que le diste al proyecto, cuando lo creaste con Angular CLI.

```
<proyecto-angular2-app>Loading...</proyecto-angular2-app>
```

Este es el componente donde tu aplicación Angular 2 va a vivir. Todos los demás componentes estarán debajo de éste, unos dentro de otros. Todo lo que ocurra en tu aplicación, estará dentro de este componente.

El texto que está dentro del componente (Loading...) es lo que aparecerá en el navegador mientras no carga la página. Una vez que la aplicación se inicie, Angular lo sustituirá por el contenido que sea necesario.

Nota: Si al arrancar la aplicación (ng-serve) ves que ese mensaje de "Loading..." tarda en irse es porque estás en modo de desarrollo y antes de iniciarse la app tienen que hacerse varias tareas extra, como transpilado de código, que no se necesitarán hacer cuando esté en producción.

Entendiendo el código del componente

El código de este componente está generado de antemano en la carpeta "src/app". Allí encontrarás varios ficheros que forman el componente completo, separados por el tipo de código que colocarás en ellos.

proyecto-angular2.component.html: Equivale a lo que conocemos por "vista" en la arquitectura MVC.

proyecto-angular2.component.css: Permite colocar estilos al contenido, siendo que éstos están encapsulados en este componente y no salen para afuera.

proyecto-angular2.component.ts: Es el archivo TypeScript, que se traducirá a Javascript antes de entregarse al navegador. Sería el equivalente al controlador en el MVC.

proyecto-angular2.component.spec.ts: Un archivo TypeScript destinado a tareas de testing de componentes.

El archivo HTML, o lo que sería la vista, admite toda clase de código HTML, con etiquetas estándar y el uso de otros componentes. Además podemos colocar expresiones (entre dobles llaves), expresar bindig entre componentes, eventos, etc.

Nota: Los que no conozcan de todo eso que estamos hablando no se preocupen, porque lo veremos

más adelante.

Entre el HTML de la vista encontrarás:

```
{{title}}
```

Eso es una expresión. Angular lo sustituirá por el contenido de una variable "title" antes de mostrarlo al cliente. Esa variable se define en lo que sería el controlador en el patrón MVC, solo que en Angular 2 no se le llama controlador, o "controller". Ahora es una clase que podemos encontrar en el archivo TypeScript "proyecto-angular2.component.ts". Si lo abres encontrarás varias cosas.

- El import de "component" dentro de @angular/core
- Una función decoradora que hace la acción de registrar el componente
- La clase que hace las veces de controlador

La función decoradora observarás que declara diversas cuestiones.

```
@Component({
  moduleId: module.id,
  selector: 'proyecto-angular2-app',
  templateUrl: 'proyecto-angular2.component.html',
  styleUrls: ['proyecto-angular2.component.css']
})
```

Una de ellas es el "selector" de este componente, o el nombre de la etiqueta que se usará cuando se desee representar. Luego está asociada la vista (propiedad "templateUrl") al archivo .html del componente y su estilo (propiedad "styleUrls") a un array de todas las hojas de estilo que deseemos.

En la clase del componente, que se debe colocar con un export para que se conozca fuera de este módulo, es la parte que representa el controlador en una arquitectura MVC. En ella colocaremos todas las propiedades y métodos que se deseen usar desde la vista.

```
export class ProyectoAngular2AppComponent {
  title = 'proyecto-angular2 works!';
}
```

Esas propiedades representan el modelo de datos y se podrán usar expresiones en las vistas para poder visualizarlas.

Nota: Observa además que el nombre de la clase de este componente tiene una forma especial. Mientras que el nombre de la etiqueta del componente (su "selector") tiene las palabras separadas por comas, aquí tenemos una notación "CamelCase" típica de Javascript. Esto es una constante. En el HTML que no se reconocen mayúsculas y minúsculas se separa por guiones, colocando todo en minúscula y los mismos nombres en Javascript se escriben con "CamelCase", todo junto y con la

primera letra de cada palabra en mayúscula.

Alterando el código de nuestro componente

Para terminar este artículo vamos a hacer unos pequeños cambios en el código del componente para comprobar si la magia de Angular está funcionando.

Algo muy sencillo sería comenzar por crear una nueva propiedad en la clase del componente. Pero vamos además a colocar un método para poder usarlo también desde la vista.

```
export class ProyectoAngular2AppComponent {
  title = 'Manual de Angular 2 de DesarrolloWeb.com';
  visible = false;
  decirAdios() {
    this.visible = true;
  }
}
```

Nota: Esta clase "class" se escribe en un archivo TypeScript, pero realmente lo que vemos es todo Javascript válido en ES6. TypeScript entiende todo ES6 e incluso algunas cosas de ES7.

Ahora vamos a ver el código HTML que podría tener nuestra vista.

```
<h1>
  {{title}}
</h1>
<p [hidden]="!visible">
  Adiós
</p>
<button (click)="decirAdios()">Decir adiós</button>
```

En este HTML hemos incluido más cosas de las que puedes usar desde Angular. Habíamos mencionado la expresión, entre llaves. También encuentras el uso de una propiedad de un elemento, como es "hidden", entre corchetes (nuevo en Angular 2). Además de la declaración de un evento "click" que se coloca entre paréntesis.

En este caso lo que tenemos en la propiedad "visible" de la clase, que hace las veces de controlador, se usa para asignarlo a la propiedad hidden del elemento "p". El método de la clase, decirAdios() se usa para asociarlo como manejador del evento "click".

Hablaremos más adelante de todas estas cosas que puedes colocar en las vistas y algunas otras, junto con las explicaciones sobre la sintaxis que se debe usar para declararlas.

Nota: Al modificar los archivos del componente, cualquiera de ellos, tanto el html, css o ts, se debería refrescar automáticamente la página donde estás visualizando tu proyecto una vez puesto en marcha con el comando "ng serve", gracias al sistema de "live-reload" que te monta Angular CLI en cualquier proyecto Angular 2.

Otra cosa interesante del entorno de trabajo es que, si usas Visual Studio Code u otro editor con los correspondientes plugin TypeScript, te informarán de posibles errores en los archivos .js. Es una ayuda muy útil que aparece según estás escribiendo.

Con esto acabamos nuestro primer análisis y modificaciones en el componente inicial. Estamos seguros que esta última parte, en la que hemos modificado el código del componente básico, habrá resultado ya algo más entretenida.

Este artículo es obra de *Alberto Basalo*
Fue publicado por primera vez en 06/07/2016
Disponible online en <http://desarrolloweb.com/articulos/introduccion-componentes-angular2.html>

Sintaxis para las vistas en Angular 2

Expresiones, binding, propiedades, eventos. Son muchas cosas las que podemos expresar en las vistas HTML de las aplicaciones Angular 2. Te ofrecemos una introducción general.

En los artículos anteriores del [Manual de Angular 2](#) hemos analizado la estructura de una aplicación básica. Una de las cosas fundamentales que encontramos es el componente principal, sobre el cual hemos hecho pequeñas modificaciones para comprobar que las cosas están funcionando y comenzar a apreciar el poder de Angular.

Dentro del componente básico encontramos varios archivos y uno de ellos es el código HTML del componente, al que comúnmente nos referiremos con el nombre de "vista", denominación que viene por el patrón de arquitectura MVC. Pues bien, en ese código HTML, la vista, de manera declarativa podemos definir y usar muchas de las piezas con las que contamos en una aplicación: Vistas, propiedades, eventos, bindeo...

La novedad en Angular 2, para los que vienen de la versión anterior del framework, es que ahora podemos ser mucho más precisos sobre cómo queremos que la información fluya entre componentes, entre la vista y el modelo, etc. En este artículo vamos a ofrecer una primera aproximación general a todo lo que se puede declarar dentro de una vista, teniendo en cuenta que muchas de las cosas necesitarán artículos específicos para abordarlas en profundidad.



Nota: En este artículo, así como a lo largo del manual, hacemos uso constantemente de conceptos del MVC. Recordamos que existe un artículo genérico de lo que es el MVC, [introducción al modelo vista controlador](#). De todos modos, en Angular 2 no se aplica una implementación perfectamente clara sobre el MVC. Existe una separación del código por responsabilidades, lo que ya nos aporta los beneficios de la arquitectura por capas, pero la implementación y características de estas capas no queda tan definida como podría darse en un framework backend. En este artículo y los siguientes verás que hacemos referencia a los modelos y realmente no es que exista una clase o algo concreto donde se coloca el código del modelo. Ese modelo realmente se genera desde el controlador y para ello el controlador puede obtener datos de diversas fuentes, como servicios web. Esos datos, tanto propiedades como métodos, se podrán usar desde la vista. Es algo más parecido a un VW (View Model), o sea, un modelo para la vista, que se crea en el controlador. Por otra parte, tampoco existe un controlador, sino una clase que implementa el View Model de cada componente. En resumen, conocer el MVC te ayudará a entender la arquitectura propuesta por Angular 2 y apreciar sus beneficios, pero debemos mantener la mente abierta para no confundirnos con conocimientos que ya podamos tener de otros frameworks.

Piezas declarables en una vista

Comenzaremos por describir las cosas que disponemos para su declaración en una vista, de modo que todos podamos usar un único vocabulario.

- **Propiedad:** Cualquier valor que podemos asignar por medio de un atributo del HTML. Ese elemento puede ser simplemente un atributo del HTML estándar, un atributo implementado mediante el propio Angular 2 o un atributo personalizado, creado para un componente en específico.
- **Expresión:** Es un volcado de cualquier información en el texto de la página, como contenido a cualquier etiqueta. La expresión es una declaración que Angular procesará y sustituirá por su valor, pudiendo realizar pequeñas operaciones.
- **Binding:** Es un enlace entre el modelo y la vista. Mediante un binding si un dato cambia en el modelo, ese cambio se representa en la vista. Pero además en Angular se introduce el "doble binding", por el cual si un valor se modifica en la vista, también viaja hacia el modelo.
- **Evento:** es un suceso que ocurre y para el cual se pueden definir manejadores, que son funciones que se ejecutarán como respuesta a ese suceso.

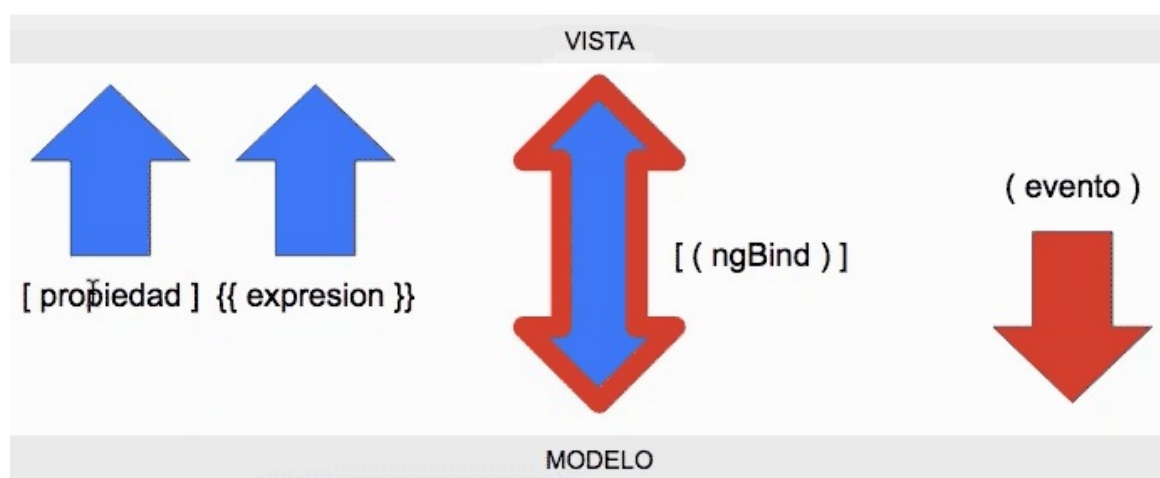
Nota: Generalmente cuando hablemos de "binding" en la mayoría de las ocasiones nos referimos a "doble binding", que es la principal novedad que trajo Angular 1 y que le produjo tanto éxito para este framework.

El problema del doble binding es que tiene un coste en tiempo de procesamiento, por lo que si la aplicación es muy compleja y se producen muchos enlaces, su velocidad puede verse afectada. Es el motivo por el que se han producido nuevas sintaxis para poder expresar bindings de varios tipos, de una y de dos direcciones. Dicho de otra manera, ahora se entrega al programador el control del flujo de la información, para que éste pueda optimizar el rendimiento de la aplicación.

Flujo de la información de la vista al modelo y modelo a vista

El programador ahora será capaz de expresar cuándo una información debe ir del modelo hacia la vista y cuándo debe ir desde la vista al modelo. Para ello usamos las anteriores "piezas" o "herramientas" en el HTML, las cuales tienen definida de antemano un sentido para el flujo de los datos.

En el siguiente diagrama puedes ver un resumen del flujo de la información disponible en Angular, junto con las piezas donde podemos encontrarlo y su sintaxis.



1. Las propiedades tienen un flujo desde el modelo a la vista. Una información disponible en el modelo se puede asignar como valor en un elemento del HTML mediante una propiedad, usando la notación corchetes. Por ej: [propiedad]
2. Las expresiones también viajan desde el modelo a la vista. La diferencia de las propiedades es que en este caso las usamos como contenido de un elemento y además que se expresan con dobles llaves. Por ej: {{expresión}}
3. El binding (a dos sentidos, o doble binding) lo expresamos entre corchetes y paréntesis. En este caso la información fluye en ambos sentidos, desde el modelo a la vista y desde la vista al modelo. Por ej: [(ngBind)]
4. Los eventos no es que necesariamente hagan fluir un dato, pero sí se considera un flujo de aplicación, en este caso de la vista al modelo, ya que se originan en la vista y generalmente sirven para ejecutar métodos que acabarán modificando cosas del modelo. Por ej: (evento)

Nota: Como ves, ahora existen diversas sintaxis para expresar cosas en las vistas. Quizás nos resulte extraño, pero enseguida nos familiarizaremos. La notación más rara es la que usamos para expresar un binding en dos direcciones [(ngBind)], pero una manera sencilla de acordarnos de ella es con su denominación anglosajona "banana in a box". Los paréntesis parecen una banana, dentro de los corchetes, que parecen una caja.

Ejemplos de sintaxis utilizada en vistas de Angular 2

Realmente ya hemos visto ejemplos de buena parte de las piezas posibles a declarar en una vista. Si te fijas en el artículo anterior dedicado a la [Introducción a los componentes en Angular 2](#). Ahora les podemos dar nombres a cada uno de los elementos encontrados.

Propiedades: Era el caso de la propiedad "hidden" que usamos para mostrar / ocultar determinados elementos. En este caso, hidden no es una propiedad estándar del HTML, sino que está generada por Angular 2 y disponible para aplicar tanto en etiquetas HTML comunes como en componentes personalizados.

```
<p [hidden]="!visible">Adiós</p>
```

También podríamos aplicar valores a atributos del HTML con datos que están en propiedades del modelo, aunque no soporta todos los atributos del HTML estándar. Por ejemplo, podríamos asignar una clase CSS (class) con lo que tuviésemos en una propiedad del modelo llamada "clase".

```
<div [class]="clase">Una clase marcada por el modelo</div>
```

O el enlace de un enlace podríamos también definirlo desde una variable del modelo con algo como esto:

```
<a [href]="enlace">Pulsa aquí</a>
```

Pero en general, el uso más corriente que se dará a esta forma de definir una propiedad personalizada de un componente, de modo que podamos personalizar el estado o comportamiento del componente mediante datos que tengamos en el modelo. Veremos este caso más adelante cuando analicemos con mayor detalle los componentes.

Lo que de momento debe quedar claro es que las propiedades van desde el modelo a la vista y, por tanto, si se modifica el valor de una propiedad en el modelo, también se modificará la vista. Pero, si dentro de la vista se modifica una propiedad no viajará al modelo automáticamente, pues el enlace es de una sola dirección.

Expresiones: Es el caso de la propiedad del modelo "title" que se vuelca como contenido de la página en el encabezamiento.

```
<h1>
  {{title}}
</h1>
```

Simplemente existe esa sustitución del valor de la propiedad, colocándose en el texto de la página. El enlace es de una única dirección, desde el modelo a la vista. En la vista tampoco habría posibilidad de modificar nada, porque es un simple texto.

El caso de propiedades del HTML con valores que vienen del modelo también podríamos implementarlo por medio de expresiones, tal como sigue.

```
<a href="{{enlace}}">Clic aquí</a>
```

Nota: Las dos alternativas (usar expresiones con las llaves o los corchetes para propiedades, como hemos visto para el ejemplo de un href de un enlace cuyo valor traes del modelo) funcionan exactamente igual, no obstante para algunos casos será mejor usar la sintaxis de propiedades en vez de la de expresiones, como es el caso del enlace. Algo que entenderemos mejor cuando lleguemos al sistema de rutas.

Eventos: Esto también lo vimos en el artículo anterior, cuando asociamos un comportamiento al botón. Indicamos entre paréntesis el tipo de evento y como valor el código que se debe de ejecutar, o mejor, la función que se va a ejecutar para procesar el evento.

```
<button (click)="decirAdios()">Decir adiós</button>
```

Con respecto a Angular 1.x entenderás que ahora todas las directivas como ng-click desaparecen, dado que ahora los eventos solo los tienes que declarar con los paréntesis. Esto es interesante ya de entrada, porque nos permite definir de una única manera cualquier evento estándar del navegador, pero es más interesante todavía cuando comencemos a usar componentes personalizados, creados por nosotros, que podrán disparar también eventos personalizados. Capturar esos eventos personalizados será tan fácil como capturar los eventos estándar del HTML.

Binding: Para este último caso no hemos visto todavía una implementación de ejemplo, pero lo vamos a conseguir muy fácilmente. Como se dijo, usamos la notación "banana in a box" para conseguir este comportamiento de binding en dos direcciones. Los datos viajan de la vista al modelo y del modelo a la vista.

```
<p>
  ¿Cómo te llamas? <input type="text" [(ngModel)]="quien">
</p>
```

En este caso, desde el HTML estaríamos creando una propiedad dentro del modelo. Es decir, aunque no declaremos la propiedad "quien" en el Javascript, por el simple hecho de usarla en la estructura de binding va a producir que se declare automáticamente y se inicialice con lo que haya escrito en el campo de texto. Si la declaramos, o la inicializamos desde la clase que hace las veces de controlador, tanto mejor, pero no será necesario.

Nota: La notación "banana in a box" tiene una explicación y es que usa tanto el flujo de datos desde el modelo a la vista, que conseguimos con los corchetes para las propiedades, como el flujo desde la vista al modelo que conseguimos con los paréntesis para los eventos.

Para quien conozca Angular 1, ngModel funciona exactamente igual que la antigua directiva. En resumen, le asignamos el nombre de una propiedad en el modelo, en este caso "quien", con la que se va a conocer ese dato. A partir de entonces lo que haya escrito en el campo de texto viajará de la vista al modelo y si cambia en el modelo también se actualizará la vista, produciendo el binding en las dos direcciones.

Si quisiéramos visualizar ese dato en algún otro de la vista, por ejemplo en un párrafo, usaríamos una expresión. Por ejemplo:

```
<p>  
  Hola {{quien}}  
</p>
```

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 19/07/2016
Disponible online en <http://desarrolloweb.com/articulos/sintaxis-vistas-angular2.html>

Los componentes en Angular2

Abordamos el desarrollo basado en componentes con todo detalle. Es la pieza más importante de Angular 2 que nos permitirá no solo estructurar una aplicación de una manera ordenada, sino encapsular funcionalidad y facilitar una arquitectura avanzada y de fácil mantenimiento de los proyectos Javascript con la nueva versión de AngularJS.

El concepto de los componentes en Angular 2 y su arquitectura

Qué es un componente para Angular 2 y cómo usaremos los componentes para realizar la arquitectura de una aplicación.

En este artículo del [Manual de Angular 2](#) queremos abordar el concepto del componente desde un punto de vista teórico, sin entrar a ver cómo se construyen en Angular 2. Esa parte práctica la dejaremos para el próximo artículo, aunque cabe recordar, para los que se impacientan por ver código, que ya dimos una [zambullida en el código de los componentes](#) cuando comenzamos con el "hola mundo".

El objetivo es entender mejor cuál es la arquitectura promovida por Angular 2 para el desarrollo de aplicaciones y qué papel específico desempeñan los componentes. Es importante porque toda aplicación Angular 2 se desarrolla en base a componentes y porque es algo relativamente nuevo en el framework.



Árbol de componentes

Una aplicación Angular 2 se desarrolla a base de crear componentes. Generalmente tendrás un árbol de componentes que forman tu aplicación y cada persona lo podrá organizar de su manera preferida. Siempre existirá un componente padre y a partir de ahí podrán colgar todas las ramas que sean necesarias para crear tu aplicación.

Esto no resultará nada extraño, pues si pensamos en una página web tenemos un mismo árbol de etiquetas, siendo BODY la raíz de la parte del contenido. La diferencia es que las etiquetas generalmente son para mostrar un contenido, mientras que los componentes no solo encapsulan un contenido, sino también una funcionalidad.

En nuestro árbol, como posible organización, podemos tener en un primer nivel los bloques principales de la pantalla de nuestra aplicación.

- Una barra de herramientas, con interfaces para acciones principales (lo que podría ser una barra de navegación, menús, botonera, etc.).
- Una parte principal, donde se desplegarán las diferentes "pantallas" de la aplicación.
- Un área de logueo de usuarios.
- Etc.

Nota: Obviamente, ese primer nivel de componentes principales lo dictará el propio proyecto y podrá cambiar, pero lo anterior nos sirve para hacernos una idea.

Luego, cada uno de los componentes principales se podrá subdividir, si se desea, en nuevos árboles de componentes.

- En la barra de herramientas principal podríamos tener un componente por cada herramienta.
- En el área principal podríamos tener un componente para cada "pantalla" de la aplicación o "vista". A su vez, dentro de cada "vista" o "pantalla" podíamos tener otra serie de componentes que implementen diversas funcionalidades.
- Etc.

Los niveles del árbol serán los que cada aplicación mande, atendiendo a su complejidad, y cada desarrollador estime necesario, en función de su experiencia o preferencias de trabajo. A medida que componetizamos conseguimos dividir el código de la aplicación en piezas menores, con menor complejidad, lo que seguramente sea beneficioso.

Si llegamos a un extremo, y nos pasamos en nuestra ansia de componetizar, quizás obtengamos el efecto contrario. Es decir, acabemos agregando complejidad innecesaria a la aplicación, puesto que existe un coste de tiempo de trabajo y recursos de procesamiento para posibilitar el flujo de comunicación entre componentes.

Componentes Vs directivas

En Angular 2 perdura el concepto de directiva. Pero ahora tenemos componentes y la realidad es que ambos artefactos se podrían aprovechar para usos similares. La clave en este caso es que los componentes son piezas de negocio, mientras que las directivas se suelen usar para presentación y problemas estructurales.

Puedes pensar en un componente como un contenedor donde solucionas una necesidad de tu aplicación. Una interfaz para interacción, un listado de datos, un formulario, etc.

Para ser exactos, en la documentación de Angular 2 nos indican que un componente es un tipo de directiva. Existen tres tipos de directivas:

Componentes: Un componente es una directiva con un template. Habrá muchas en tu aplicación y resuelven necesidades del negocio. **Directivas de atributos:** Cambian la apariencia o comportamiento de

un element. Por ejemplo tenemos ngClass, que nos permite colocar una o más clases de CSS (atributo class) en un elemento. **Directivas estructurales:** Son las que realizan cambios en el DOM del documento, añadiendo, manipulando o quitando elementos. Por ejemplo ngFor, que nos sirve para hacer una repetición (similar al ngRepeat de Angular 1.x), o ngIf que añade o remueve elementos del DOM con respecto a una expresión condicional.

Por tanto, a diferencia de otras librerías como Polymer, donde todo se resuelve mediante componentes, hay que tener en cuenta qué casos de uso son los adecuados para resolver con un componente.

Las partes de un componente

Aunque también hemos analizado anteriormente, cuando [repasamos la aplicación básica de Angular 2](#) generada con [Angular CLI](#), cuáles son las partes fundamentales de un componente, vamos a volver a este punto para poder ver sus piezas de manera global.

Un componente está compuesto por tres partes fundamentales:

- Un template
- Una clase
- Una función decoradora

Las dos primeras partes corresponden con capas de lo que conocemos como MVC. El template será lo que se conoce como vista y se escribe en HTML y lo que correspondería con el controlador se escribe en Javascript por medio de una clase (de programación orientada a objetos).

Por su parte, tenemos el decorador, que es una especie de registro del componente y que hace de "pegamento" entre el Javascript y el HTML.

Todas estas partes son las que vamos a analizar en los próximos artículos con mayor detalle, comenzando por los decoradores, que introduciremos en el próximo artículo.

Este artículo es obra de *Alberto Basalo*
Fue publicado por primera vez en 31/07/2016
Disponible online en <http://desarrolloweb.com/articulos/concepto-teorico-componente-angular2.html>

Decorador de componentes en Angular 2

Qué es un decorador de componentes, qué función tiene y cómo se implementa en un componente básico de Angular 2.

Ahora, una de las funciones básicas que vas a tener que realizar en todo desarrollo con Angular 2 es la decoración de componentes. En sí, no es más que una declaración de cómo será un componente y las diversas piezas de las que consiste.

En el artículo de [introducción a los componentes](#) explicamos solo una de las partes que tiene el archivo .ts con el código Javascript / TypeScript principal de un componente. Lo que vimos hasta ahora era la clase

que, decíamos, hacía las veces de controlador, que se exporta hacia afuera para que el flujo principal de ejecución de una aplicación sea capaz de conocer al componente. Además, contiene lo que se llama una función decoradora que conoceremos a continuación.



Qué es un decorador

Un decorador es una herramienta que tendremos a nuestra disposición en Javascript en un futuro próximo. Es una de las propuestas para formar parte del estándar ECMAScript 2016, conocido también como ES7. Sin embargo, ya están disponibles en TypeScript, por lo que podemos comenzar a usarlos ya en Angular 2.

Básicamente es una implementación de un patrón de diseño de software que en sí sirve para extender una función mediante otra función, pero sin tocar aquella original, que se está extendiendo. El decorador recibe una función como argumento (aquella que se quiere decorar) y devuelve esa función con alguna funcionalidad adicional.

Las funciones decoradoras comienzan por una "@" y a continuación tienen un nombre. Ese nombre es el de aquello que queramos decorar, que ya tiene que existir previamente. Podríamos decorar una función, una propiedad de una clase, una clase, etc.

Mira la primera línea del código del archivo .ts de tu componente principal.

```
import { Component } from '@angular/core';
```

Ese import nos está trayendo la clase Component. En la siguiente línea se decora a continuación, con el correspondiente "decorator". No es nuestro objetivo hablar sobre el patrón decorator en sí, ni ver las posibilidades de esta construcción que seguramente tendremos en el futuro ES7, así que vamos a centrarnos en lo que conseguimos hacer con Angular 2 mediante estos decoradores.

Nota: Uno de los motivos por los que Angular 2 ha tomado TypeScript como lenguaje es justamente por permitir usar decoradores. Con TypeScript podemos realizar la decoración de código de ES7 ya mismo, lo que facilita la decoración del código.

Qué información se agrega por medio del decorador

Angular 2 usa los decoradores para registrar un componente, añadiendo información para que éste sea reconocido por otras partes de la aplicación. La forma de un decorador es la siguiente:

```
@Component({
  moduleId: module.id,
  selector: 'test-angular2-app',
  templateUrl: 'test-angular2.component.html',
  styleUrls: ['test-angular2.component.css']
})
```

Como apreciarás, en el decorador estamos agregando diversas propiedades específicas del componente. Esa información en este caso concreto se conoce como "anotación" y lo que le entregamos son unos "metadatos" (metadata) que no hace más que describir al componente que se está creando. En este caso son los siguientes:

- **moduleId**: esta propiedad puede parecer un poco extraña, porque siempre se le asigna el mismo valor en todos los componentes. Realmente ahora nos importa poco, porque no agrega ninguna personalización. Es algo que tiene que ver con CommonJS y sirve para poder resolver Urls relativas.
- **selector**: este es el nombre de la etiqueta nueva que crearemos cuando se procese el componente. Es la etiqueta que usarás cuando quieras colocar el componente en cualquier lugar del HTML.
- **templateUrl**: es el nombre del archivo .html con el contenido del componente, en otras palabras, el que tiene el código de la vista.
- **styleUrls**: es un array con todas las hojas de estilos CSS que deben procesarse como estilo local para este componente. Como ves, podríamos tener una única declaración de estilos, o varias si lo consideramos necesario.

Nota: Ese código de anotación o decoración del componente es generado por Angular CLI. Además, cuando creemos nuevos componentes usaremos el mismo Angular CLI para obtener el scaffolding (esqueleto) del cual partiremos. Por tanto, no hace falta que memorices la sintaxis para la decoración, porque te la darán hecha. En todo caso tendrás que modificarla si quieres cambiar el comportamiento del componente, los nombres de archivos del template (vista), hojas de estilo, etc.

De momento no necesitamos dar mucha más información sobre los decoradores. Es algo que debemos comenzar a usar para desarrollar componentes en Angular 2, pero no nos tiene que preocupar demasiado todavía, porque de momento no necesitaremos tocar mucho sobre ellos.

Era importante para finalizar nuestro análisis de la aplicación básica de Angular 2. Con esto creemos que debe quedar claro el código del primer componente que nos genera la aplicación de inicio creada por Angular CLI. Ahora que ya sabes lo básico, podemos continuar con otros asuntos que seguro serán más entretenidos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 09/08/2016
Disponible online en <http://desarrolloweb.com/articulos/decorador-componentes-angular2.html>

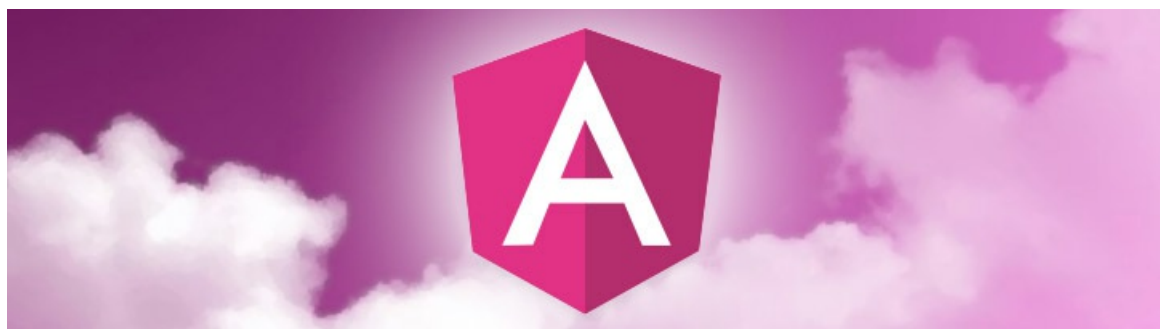
Crear un componente nuevo con Angular 2

En este artículo te vamos a enseñar a crear un nuevo componente con Angular CLI y luego a usarlo en tu aplicación Angular 2.

Después de nuestro repaso teórico a la figura de los [componentes en Angular 2](#) estamos en condiciones de irnos al terreno de lo práctico y ver cómo se generan nuevos componentes en una aplicación.

Si abres el index.html que hay en la carpeta "src" verás como en el body existe un único componente. Ahora, todos los que desarrollemos en adelante estarán dentro de ese primer componente. Esto no es nuevo, puesto que ya se comentó en el [Manual de Angular 2](#), pero está bien recordarlo para que quede claro. Partiremos de esta situación agregando ahora nuevos componentes con los que podremos expandir la aplicación.

Nota: En realidad nadie te obliga a tener un componente único como raíz. Podrías crear un componente y usarlo directamente en el index.html de tu aplicación. Haciendo esto convenientemente no habría ningún problema por ello. Aunque debido a que ese index.html es código generado y generalmente no lo querremos tocar, será más recomendable crear los componentes debajo del componente raíz.



Creamos un componente a través de Angular CLI

Ya usamos [Angular CLI](#) para generar el código de inicio de nuestra aplicación Angular 2. Ahora vamos a usar esta herramienta de línea de comandos para generar el esqueleto de un componente. Desde la raíz del proyecto lanzamos el siguiente comando:

```
ng generate component nombre-del-componente
```

Nota: Existe un alias para la orden "generate" que puedes usar para escribir un poco menos. Sería simplemente escribir "g".

```
ng g component nombre-del-componente
```

Además de componentes, la orden generate te permite crear el esqueleto de otra serie de artefactos como

son directivas, servicios, clases, etc.

Reconociendo los archivos del componente

Si observas ahora la carpeta "src/app" encontrarás que se ha creado un directorio nuevo con el mismo nombre del componente que acabamos de crear. Dentro encuentras una serie de archivos que ya te deben de sonar porque los hemos analizado ya para el [componente inicial de las aplicaciones Angular2](#).

Ahora además queremos fijarnos en un archivo que se llama index.ts, que verás que solo tiene un export.

```
export { NombreDelComponenteComponent } from './nombre-del-componente.component';
```

Como ves, está exportando el propio componente, por su nombre. Ese componente es una clase (de OOP) que hay en "nombre-del-componente.component.ts", que a su vez se exportaba también.

```
export class NombreDelComponenteComponent implements OnInit {
```

En resumen, el archivo index.ts únicamente sirve para que, cuando importas un componente desde otro lugar de tu aplicación, no tengas que referirte al archivo "nombre-del-componente.component.ts" con todas sus letras, sino simplemente a la carpeta donde se encuentra.

Componente declarado para SystemJS

En todos los lugares donde, en adelante, deseemos usar ese componente tendremos que importarlo para que se conozca su código. Para ello hemos visto que solo tendremos que indicar el nombre del componente y no el archivo donde se escribió su clase.

Pero por debajo, para realizar todo lo que es la carga de módulos, se utiliza en una aplicación Angular 2 SystemJS. No nos hemos metido en profundidad con este sistema, pero básicamente lo que hace es traernos el código de aquello que necesitamos.

Dentro de system-config.ts se administran todas las librerías que podremos importar con SystemJS. Entre ellas encontrarás que se hace una declaración del nuevo componente que acabamos de generar.

```
// App specific barrels.  
'app',  
'app/shared',  
'app/nombre-del-componente',
```

Esa declaración es la que realmente permite que, cuando importamos el componente, no tengamos que preocuparnos por la estructura de archivos que tiene, simplemente lo importamos con el nombre de la carpeta donde se encuentra.

Nota: Angular CLI es quien maneja y genera el código de system-config.ts, por lo que nosotros no

tendremos que tocar nada generalmente. No obstante, queríamos mostrar este detalle para que no pasase desapercibido.

Javascript de nuestro componente

El archivo "nombre-del-componente.component.ts" contiene el código Javascript del componente.

Nota: Apremiarás que deberíamos decir que contiene el "código TypeScript del componente", dado que en realidad a día de hoy Angular CLI solo tiene la opción de generar código TypeScript. No debe representar un gran problema para ti, porque realmente todo código Javascript es también código [TypeScript](#), al que se le agregan ciertas cosas, sobre todo para el control de tipos.

Debes reconocer ya diversas partes:

- Imports de todo aquello que necesitemos. En principio de la librería `@angular/core`, pero luego veremos que aquí iremos colocando muchos otros imports a medida que vayamos necesitando código de más lugares.
- Decorador del componente, para su registro.
- Clase que hace las veces del controlador, que tengo que exportar.

Además ahora te pedimos simplemente echar un vistazo a la cabecera de la clase, en concreto a su "implements":

```
export class NombreDelComponenteComponent implements OnInit {
```

Ese implements es una interfaz, que no están disponibles todavía en Javascript, ni tan siquiera en ES6, pero que ya son posibles de usar gracias a TypeScript. Es simplemente como un contrato que dice que dentro de la clase del componente vamos a definir la función `ngOnInit()`. Sobre esa función no hablaremos mucho todavía, pero es un lugar donde podremos colocar código a ejecutar cuando se tenga la certeza que el componente ha sido inicializado ya.

Solo a modo de prueba, vamos a crear una propiedad llamada "dato" dentro de nuestro componente recién creado. El código nos quedará algo como esto:

```
import { Component, OnInit } from '@angular/core';
```

```
@Component({
  moduleId: module.id,
  selector: 'app-nombre-del-componente',
  templateUrl: 'nombre-del-componente.component.html',
  styleUrls: ['nombre-del-componente.component.css']
})
export class NombreDelComponenteComponent implements OnInit {
```

```
dato = "Creando componentes para DesarrolloWeb.com";

constructor() {}

ngOnInit() {
  console.log('componente inicializado...');
}

}
```

HTML del componente

El componente que acabamos de crear tiene un HTML de prueba, ya escrito en el archivo "nombre-del-componente.component.html".

Podemos agregarle la expresión para que se vea en el componente la propiedad que hemos generado del lado de Javascript, en la clase del componente. Tendrías algo como esto:

```
<p>
  nombre-del-componente works!
</p>

<p>
  {{ dato }}
</p>
```

Puedes abrir ese HTML y colocar cualquier cosa que consideres, simplemente a modo de prueba, para comprobar que consigues ver ese texto ahora cuando usemos el componente.

Nota: No hemos entrado todavía, pero seguro que alguno ya se lo pregunta o quiere saberlo. El componente tiene un archivo CSS (aunque se pueden definir varios en el decorador del componente) y podemos editarlo para colocar cualquier declaración de estilos. Veremos más adelante, o podrás comprobar por ti mismo, que esos estilos CSS se aplican únicamente al componente donde estás trabajando y no a otros componentes de la aplicación.

Con esto hemos terminado de explicar todo lo relativo a la creación de un componente. El componente está ahí y estamos seguros que estarás ansioso por verlo en funcionamiento en tu proyecto. Es algo que veremos ya mismo, en el próximo artículo del Manual de Angular 2.

Este artículo es obra de *Alberto Basalo*
Fue publicado por primera vez en 26/08/2016
Disponible online en <http://desarrolloweb.com/articulos/crear-componente-nuevo-angular2.html>

Usar un componente en Angular 2

Cómo se usan componentes creados por nosotros en Angular 2, una tarea que si bien es sencilla requiere de varios pasos.

En el pasado artículo realizamos todos los pasos para [crear un componente en Angular 2](#). Realmente vimos que la mayoría del código lo generas desde Angular CLI, lo que acelera mucho el desarrollo y facilita nuestra labor.

Ahora, para terminar nuestra práctica, vamos a aprender a usar el componente que acabamos de crear. Es una tarea sencilla, pero debido a la arquitectura de Angular y el modo de trabajo que nos marca, es algo que tendremos que realizar en varios pasos:



1. Crear el HTML para usar el componente

En el lugar de la aplicación donde lo vayas a usar el componente, tienes que escribir el HTML necesario para que se muestre. El HTML no es más que la etiqueta del componente, que se ha definido en la función decoradora, atributo "selector" ([como vimos al explicar los decoradores](#)).

```
<app-nombre-del-componente></app-nombre-del-componente>
```

Dado que estamos comenzando con Angular 2 y el anterior era el primer componente creado por nosotros mismos, solo lo podremos usar dentro del componente principal (aquel generado por Angular CLI al hacer construir el nuevo proyecto).

El HTML de este componente principal lo encontramos en el archivo "app.component.html". En ese archivo, la vista del componente principal, debemos colocar la etiqueta para permitir mostrar el componente recién creado.

El problema es que esa etiqueta no es conocida por el navegador. Cuando creamos el componente generamos todo el código, en diversos archivos, necesario para dar vida al componente, pero habrá que incluirlo para que el navegador conozca esa etiqueta cuando se vaya a usar. Es lo que hacemos en los siguientes pasos.

2. Importar el código del componente

Como decíamos, desde el componente principal no conocemos el código del componente que se pretende usar. En Angular 2 en general, todo archivo con código Javascript que quieras usar, debe ser importado.

En el componente desde donde pretendas usar ese otro componente necesitas importar su código. Como

verás en el archivo .ts principal de un componente, siempre comienza con un import de librerías que nos aporta Angular. Ahora, después del import de las librerías de Angular, colocaremos el import de nuestro componente:

```
import { NombreDelComponenteComponent } from './nombre-del-componente'
```

Ese import indica que te quieres traer la clase del componente y después del "from" está la ruta desde donde te la traes.

Nota: Recuerda que no necesitas decirle exactamente en qué fichero está la clase NombreDelComponenteComponent, porque existe un archivo index.ts en todo componente generado que hace de raíz y asocia el nombre de carpeta del componente al archivo .ts donde está la clase. Esto lo explicamos en el artículo anterior, cómo crear un componente, en el apartado "Reconociendo los archivos del componente".

3. Declarar que vas a usar este componente

El import permite conocer el código del componente, pero todavía no es suficiente para poder usarlo. Nos queda un último paso.

En el componente desde donde hemos incluido el import del componente, un poco más abajo encuentras la función decoradora del componente. En esa función debes declarar todos los componentes que pretendes usar.

Eso lo hago desde la función decoradora del componente principal y tengo que declarar todos los componentes y directivas a usar dentro del array "directives".

```
directives: [NombreDelComponenteComponent]
```

El decorador completo del componente principal se vería parecido a esto:

```
@Component({
  moduleId: module.id,
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['app.component.css'],
  directives: [NombreDelComponenteComponent]
})
```

Observa que el array "directives" me permite declarar que voy a usar varios componentes. Simplemente separo sus nombres por comas. Lo que indico, como podrás apreciar, es el nombre de la clase del componente que pretendo usar. Esa clase es la que has importado con el correspondiente "import" del paso anterior (punto 2).

Nota: Puede parecer un poco raro que el array se llame "directives" en vez de "components" y eso es porque en ese array podríamos declarar que vamos a usar tanto componentes como directivas. Además debes recordar, porque se mencionó en el artículo de teoría de los componentes en Angular 2, que un componente no es más que un tipo específico de directiva.

Y eso es todo! Al guardar los archivos se debería recargar de nuevo la aplicación en el navegador y deberías ver el HTML escrito para el componente que estás usando y que acabamos de crear.

El proceso puede parecer un tanto laborioso, pero a medida que se vaya repitiendo observaremos que no hay tanto trabajo. La parte más aburrida de escribir de un componente, el esqueleto o scaffolding, ya te lo dan hecho gracias a Angular CLI. Así que nos queda simplemente hacer la parte del componente que corresponde a las necesidades de la aplicación.

Con los conocimientos que hemos ido proporcionando en los anteriores capítulos del Manual de Angular 2, estamos seguros de que podrás colocar más código, tanto en el HTML como en la clase para proporcionar como práctica otros tipos de funcionalidad.

Este artículo es obra de *Alberto Basalo*
Fue publicado por primera vez en 31/08/2016
Disponible online en <http://desarrolloweb.com/articulos/usar-componente-angular2.html>

Directiva ngClass en Angular 2

Estudio de la directiva ngClass de Angular 2, ejemplo práctico de un componente que usa esa directiva.

Después de varios artículos un tanto densos de en el [Manual de Angular 2](#) vamos a estudiar algunas cosas un poco más ligeras, que nos permitirán practicar con el framework sin añadir mucha complejidad a lo que ya conocemos. Para ello vamos a hacer una serie de pequeños artículos acerca de las directivas de Angular 2 más útiles en el día a día del desarrollo de aplicaciones.

Comenzamos con la directiva ngClass, que nos permite alterar las clases CSS que tienen los elementos de la página. Lo veremos en el marco del desarrollo de aplicaciones Angular con un ejemplo básico.

Si vienes de Angular 1.x verás que las directivas han perdido un poco de protagonismo en el desarrollo en Angular 2. Muchas de las antiguas directivas han desaparecido y su uso ha sido sustituido por otras herramientas diversas. Como norma ahora en Angular 2 las directivas se usarán para solucionar necesidades específicas de manipulación del DOM y otros temas estructurales.

Nota: Los componentes también pueden ser considerados como un tipo de directiva, aunque en este caso se usan para resolver problemas de negocio, como ya se introdujo en el artículo sobre las [características básicas de los componentes](#).



No todos los problemas de clases se necesitan resolver con ngClass

Lo primero es decir que la directiva ngClass no es necesaria en todos los casos. Las cosas más simples ni siquiera la necesitan. El atributo "class" de las etiquetas si lo pones entre corchetes funciona como propiedad a la que le puedes asignar algo que tengas en tu modelo. Esto lo vimos en el [artículo sobre la sintaxis de las vistas](#).

```
<h1 [class]="claseTitular">Titular</h1>
```

En este caso, "claseTitular" es una variable del modelo, algo que me pasa el controlador que tendrás asociado a un componente.

```
export class PruebaComponent implements OnInit {  
  claseTitular: string = "class1";  
  
  cambiaEstado() {  
    this.claseTitular = "class2"  
  }  
  
  ngOnInit() {  
  }  
}
```

La class del H1 valdrá lo que haya en la variable claseTitular. Cuando alguien llame al método cambiaEstado() se modificaría el valor de esa variable y por tanto cambiaría la clase en el encabezamiento.

Si esto ya resuelve la mayoría de las necesidades que se nos ocurren ¿para qué sirve entonces ngClass?

Asignar clases CSS con ngClass

La directiva ngClass es necesaria para, de una manera cómoda asignar cualquier clase CSS entre un grupo de posibilidades. Puedes usar varios valores para expresar los grupos de clases aplicables. Es parecido a como funcionaba en Angular 1.x.

A esta directiva le indicamos como valor:

1. Un array con la lista de clases a aplicar. Ese array lo podemos especificar de manera literal en el HTML.

```
<p [ngClass]="['negativo', 'off']">Pueden aplicarse varias clases</p>
```

O por su puesto podría ser el nombre de una variable que tenemos en el modelo con el array de clases creado mediante Javascript.

```
<p [ngClass]="arrayClases">Pueden aplicarse varias clases</p>
```

Ese array lo podrías haber definido del lado de Javascript.

```
clases = ['positivo', 'si'];
```

2. Un objeto con propiedades y valores (lo que sería un literal de objeto Javascript). Cada nombre de propiedad es una posible clase CSS que se podría asignar al elemento y cada valor es una expresión que se evaluará condicionalmente para aplicar o no esa clase.

```
<li [ngClass]="{positivo: cantidad > 0, negativo: cantidad < 0, off: desactivado, on: !desactivado }">Línea</li>
```

Ejemplo de aplicación de ngClass

Vamos a hacer un sencillo ejemplo de un componente llamado "BotonSino" que simplemente muestra un mensaje "SI" o "NO". Al pulsar el botón cambia el estado. Cada estado se representa además con una clase que aplica un aspecto.

Nota: No vamos a explicar las partes del componente porque se vieron con detalle en los artículos anteriores del Manual de Angular 2. Consultar toda la parte de desarrollo de componentes para más información.

Nuestro HTML es el siguiente:

```
<p>
  <button
    [ngClass]="{si: estadoPositivo, no: !estadoPositivo}"
    (click)="cambiaEstado()"
    >{{texto}}</button>
</p>
```

La etiqueta button tiene un par de atributos que son los que hacen la magia. Entre corchetes se aplica la directiva "ngClass", con un valor de objeto. Entre paréntesis se aplica el evento "click", con la invocación de la función encargada de procesar la acción. Además, el texto del botón es algo que nos vendrá de la variable "texto".

Nuestro Javascript será el siguiente:

```
import { Component, OnInit } from '@angular/core';

@Component({
  moduleId: module.id,
  selector: 'app-boton-sino',
  templateUrl: 'boton-sino.component.html',
  styleUrls: ['boton-sino.component.css']
})
export class BotonSinoComponent implements OnInit {

  texto: string = "SI";
  estadoPositivo: boolean = true;

  cambiaEstado() {
    this.texto = (this.estadoPositivo) ? "NO" : "SI";
    this.estadoPositivo = !this.estadoPositivo;
  }

  ngOnInit() {
  }

}
```

Como sabes, este código TypeScript es la mayoría generado por Angular CLI. Lo que hemos hecho nosotros es lo que está dentro de la clase BotonSinoComponent.

En ella creamos las propiedades necesarias en la vista y el método que se encarga de procesar el cambio de estado.

Nuestro CSS:

Lógicamente, para que esto funcione necesitaremos declarar algunos estilos sencillos, al menos los de las clases que se usan en el HTML.

```
button {
  padding: 15px;
  font-size: 1.2em;
  border-radius: 5px;
  color: white;
  font-weight: bold;
  width: 70px;
  height: 60px;
}

.si{
  background-color: #6c5;
}

.no{
  background-color: #933;
}
```

Con esto es todo! Es un ejemplo muy sencillo que quizás para los que vienen de Angular 1.x resulte demasiado básico, pero seguro que lo agradecerán quienes estén comenzando con Angular en estos momentos. Más adelante lo complicaremos algo. Realmente la dificultad mayor puede ser [seguir los pasos para la creación del componente](#) y luego los [pasos para su utilización](#), pero eso ya lo hemos explicado anteriormente.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 13/09/2016
Disponible online en <http://desarrolloweb.com/articulos/directiva-ngclass-angular2.html>

Directiva ngFor de Angular 2

Explicamos la directiva **ngFor**, o ***ngFor**, que nos permite repetir una serie de veces un bloque de HTML en aplicaciones Angular 2.

En este artículo vamos a conocer y practicar con una directiva de las más importantes en Angular 2, que es la directiva **ngFor**, capaz de hacer una repetición de elementos dentro de la página. Esta repetición nos permite recorrer una estructura de array y para cada uno de sus elementos replicar una cantidad de elementos en el DOM.

Para los que vengan de Angular 1 les sonará, puesto que es lo mismo que ya se conoce de **ngRepeat**, aunque cambian algunas cosillas sobre la sintaxis para la expresión del bucle, así como algunos mecanismos como la ordenación.

Uso básico de ngFor

Para ver un ejemplo de esta directiva en funcionamiento estamos obligados a crear de antemano un array con datos, que deben ser enviados a la vista, para que ya en el HTML se pueda realizar esa repetición. Como todo en Angular 2 se organiza mediante un componentes, vamos a crear un componente que contiene lo necesario para poder usar esta el **ngFor**.



Comenzamos con el código de la parte de TypeScript, que es donde tendremos que crear los datos que estarán disponibles en la vista.

```
import { Component, OnInit } from '@angular/core';
```

```
@Component({
  moduleId: module.id,
  selector: 'app-listado-preguntas',
  templateUrl: 'listado-preguntas.component.html',
  styleUrls: ['listado-preguntas.component.css']
})
export class ListadoPreguntasComponent implements OnInit {

  preguntas: string[] = [
    "¿España ganará la Euro 2016?",
    "¿Hará sol el día de mi boda?",
    "¿Estás aprendiendo Angular 2 en DesarrolloWeb?",
    "¿Has hecho ya algún curso en EscuelaIT?"
  ];

  ngOnInit() {
  }

}
```

Este código nos debe de sonar, pues es el boilerplate de un componente (creado mediante Angular CLI) al que le hemos agregado la declaración de una propiedad de tipo array de strings.

Nota: Hemos asignado el valor de ese array de manera literal, pero lo normal sería que lo obtengas de alguna fuente como un servicio web, API, etc.

Luego, veamos el código HTML de este componente, que es donde colocamos la directiva ngFor.

```
<p *ngFor="let pregunta of preguntas">
  {{pregunta}}
</p>
```

Es algo muy sencillo, simplemente tenemos un párrafo que se repetirá un número de veces, una por cada elemento del array de preguntas. En este caso es un párrafo simple, pero si dentro de él tuviéramos más elementos, también se repetirían. Lo que tenemos que analizar con detalle es el uso de la directiva, aunque creemos que se auto-explica perfectamente.

```
*ngFor="let pregunta of preguntas"
```

Lo primero que verás es un símbolo asterisco (*) que quizás parezca un poco extraño. No es más que "azúcar sintáctico" para recordarnos que estas directivas (las comenzadas por el asterisco) afectan al DOM, produciendo la inserción, manipulación o borrado de elementos del mismo.

Nota: En las explicaciones que encontrarás en la documentación de Angular 2 sobre el origen del asterisco en el nombre de la directiva nos mencionan detalles acerca de su implementación a bajo nivel, indicando que para ello se basan en el tag [TEMPLATE](#), uno de las especificaciones nativas de [Web Components](#).

Como valor de la directiva verás que se declara de manera interna para este bucle una variable "pregunta", que tomará como valor cada uno de los valores del array en cada una de sus repeticiones.

Nota: "let" es una forma de declarar variables en Javascript ES6. Quiere decir que aquella variable sólo tendrá validez dentro del bloque donde se declara. En este caso "de manera interna" nos referimos a que "pregunta" solo tendrá validez en la etiqueta que tiene el ngFor y cualquiera de sus los elementos hijo.

Recorrido a arrays con objetos con ngFor

Generalmente ngFor lo usarás para recorrer arrays que seguramente tendrán como valor en cada una de sus casillas un objeto. Esto no cambia mucho con respecto a lo que ya hemos visto en este artículo, pero sí es una bonita oportunidad de aprender algo nuevo con TypeScript.

De momento veamos las cosas sin TypeScript para ir progresivamente. Así sería cómo quedaría la declaración de nuestro array, al que todavía no indicaremos el tipo para no liarnos.

```
preguntasObj = [  
  {  
    pregunta: "¿España ganará la Euro 2016?",  
    si: 22,  
    no: 95  
  },  
  {  
    pregunta: "¿Estás aprendiendo Angular 2 en DesarrolloWeb??",  
    si: 262,  
    no: 3  
  },  
  {  
    pregunta: "¿Has hecho ya algún curso en EscuelaIT??",  
    si: 1026,  
    no: 1  
  }  
]
```

Como ves, lo que antes era un array de strings simples ha pasado a ser un array de objetos. Cada uno de los objetos nos describen tanto la pregunta en sí como las respuestas positivas y negativas que se han recibido hasta el momento.

Ahora, al usarlo en la vista, el HTML, podemos mostrar todos los datos de cada objeto, con un código que

podría ser parecido a este:

```
<p *ngFor="let objPregunta of preguntasObj">
  {{objPregunta.pregunta}}:
  <br>
  <span class="si">Si {{objPregunta.si}}</span> /
  <span class="no">No {{objPregunta.no}}</span>
</p>
```

Como ves en este código, dentro del párrafo tengo acceso a la pregunta, que al ser un objeto, contiene diversas propiedades que uso para mostrar los datos completos de cada ítem.

Modificación implementando Interfaces TypeScript

Como has visto, no existe mucha diferencia con respecto a lo que teníamos, pero ahora vamos a darle un uso a TypeScript que nos permitirá experimentar algo que nos aporta el lenguaje: interfaces.

En este caso vamos a usar las interfaces simplemente para definir un tipo de datos para las preguntas, un esquema para nuestros objetos pregunta. En este caso solo lo vamos a usar para que, a la hora de escribir código, el editor nos pueda ayudar indicando errores en caso que la interfaz no se cumpla. Así, a la hora de escribir código podremos estar seguros que todas las preguntas con las que trabajemos tengan los datos que son necesarios para la aplicación.

Nota: Las interfaces que se sabe son mecanismos para solventar las carencias de herencia múltiple, en este caso las vamos a usar como una simple definición de tipos.

Algo tan sencillo como esto:

```
interface PreguntasInterface {
  pregunta: string;
  si: number;
  no: number;
}
```

Permite a TypeScript conocer el esquema de un objeto pregunta. Ahora, apoyándonos en esa interfaz podrás declarar tu array de preguntas de esta manera.

```
preguntasObj: PreguntasInterface[] = [
  {
    pregunta: "¿Te gusta usar interfaces?",
    si: 72,
    no: 6
  }
]
```

Ahora estamos indicando el tipo de los elementos del array, diciéndole que debe concordar con lo definido en la interfaz. ¿Te gusta? Quizás ahora no aprecies mucha diferencia, pero esto se puede usar para varias cosas, significando una ayuda en la etapa de desarrollo, y sin afectar al rendimiento de la aplicación, puesto que las interfaces en TypeScript una vez transpilado el código no generan código alguno en Javascript.

Nota: Lo normal es que coloques el código de la interfaz en un archivo independiente y que hagas el correspondiente "import". Recordando que Angular CLI tiene un comando para generar interfaces que te puede resultar útil. De momento si lo deseas, a modo de prueba lo puedes colocar en el mismo archivo que el código TypeScript del componente.

Quizás para los que no estén acostumbrados a TypeScript sea difícil hacerse una idea exacta sobre cómo te ayudaría el editor de código por el simple hecho de usar esa interfaz. Para ilustrarlo hemos creado este vídeo en el que mostramos las ayudas contextuales cuando estamos desarrollando con Visual Studio Code.

Para ver este vídeo es necesario visitar el artículo original en:
<http://desarrolloweb.com/articulos/directiva-ngfor-angular2.html>

Hablaremos más sobre interfaces en otras ocasiones. Ahora el objetivo era simplemente ver una pequeña muestra de las utilidades que podría aportarnos.

Como habrás visto no es difícil entender esta directiva, pero ten en cuenta que hemos visto lo esencial sobre las repeticiones con ngFor. Hay mucho más que habría que comentar en un futuro, acerca de usos un poco más avanzados como podría ser la ordenación de los elementos del listado.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en 27/09/2016
Disponible online en <http://desarrolloweb.com/articulos/directiva-ngfor-angular2.html>