El gran libro de Angular

Miquel Boada Oriols
Juan Antonio Gómez Gutiérrez

· 100 ejercicios prácticos

Marcombo

Alfaomega

El gran libro de Angular

Miquel Boada Oriols y Juan Antonio Gómez Gutiérrez

El gran libro de Angular

Miquel Boada Oriols y Juan Antonio Gómez Gutiérrez

Marcombo

Alfaomega

ALFAOMEGA

Empresas del Grupo

Colombia: Alfaomega Colombiana S.A.
Calle 62 No.20-46 esquina, Bogotá
Teléfono (57-1) 746 0102 Fax: (57-1) 210 0122
cliente@alfaomegacolombiana.com

México: Alfaomega Grupo Editor S.A. de C.V.
Calle Doctor Olvera No. 74, Colonia Doctores,
Delegación Cuaúhtemoc, Ciudad de México
C.P. 06720 • teléfono (52-55) 5089 7740
Fax (52-55) 5575 2420
Sin costo 01-800-020-4396
libreriapitagoras@alfaomega.com.mx

Argentina: Alfaomega Grupo Editor Argentino S.A.
Av. Córdoba 1215, Piso 10
Capital Federal, Buenos Aires
Teléfono/Fax: (54-11) 4811 7183 / 8352 / 0887
ventas@alfaomegaeditor.com.ar

Chile: Alfaomega Grupo Editor S.A.
Av. Providencia 1443. Oficina 24, Santiago
Teléfonos (56-2) 2235 4248 / 2947 9351 / 2235 5786
agechile@alfaomega.cl

www.alfaomega.com.co

El gran libro de Angular
Bogotá, 2019

Revisor técnico: Pablo Martínez Izurzu
Diseño de la cubierta: Enedenú Diseño Gráfico
Maquetación: ArteMio

ISBN: 978-958-778-495-4 (Edición Colombia)
ISBN: 978-84-267-2604-9 (Edición España)

Hecho en Colombia
Printed and made in Colombia

DEDICATORIA DE MIQUEL BOADA

A Irela, compañera de vida, gracias por estar a mi lado y apoyarme en los retos que he afrontado. Te quiero.

A mis padres, Lluís y Concepció, gracias por todo vuestro amor, esfuerzo y dedicación. Sin vosotros dificilmente hubiera llegado a escribir este libro.

También dedico el libro a mis hermanas Montse e Imma, y a todas aquellas personas que forman parte de mi vida.

DEDICATORIA DE JUANTO

Dedico este libro a mi mujer Victoria y a mi hijo Pablo ya que son la principal fuente de amor y ánimo que me motiva a emprender cualquier reto.

También quisiera dedicarlo a mis padres y a mi hermana Sebi (especialmente) ya que son los que más me han animado a emprender, entre otros, el apasionante camino de las publicaciones.

Quiero agradecer a mis amigos Víctor G. y Jordi P., el cariño y reconocimiento (por otra parte, mutuo) que siempre me ha ayudado a seguir esforzándome.

Por último, quisiera agradecer a mis buenos compañeros Jordi B. y Xavi T., todo el cariño mostrado durante una buena parte de mi carrera profesional.

A vosotros…

AYUDAS EN LA WEB

Descargue gratis las ayudas de este libro en:

http://libroweb.alfaomega.com.mx/book/el_gran_libro_de_angular

CONTENIDO

HERRAMIENTAS INDIRECTAS

BOOTSTRAP

Conocer Angular

001 Introducción

Angular es una plataforma que permite desarrollar aplicaciones web en la sección cliente utilizando **HTML** y **JavaScript** para que el cliente asuma la mayor parte de la lógica y descargue al servidor con la finalidad de que las aplicaciones ejecutadas a través de Internet sean más rápidas. El hecho de estar mantenido por **Google**, así como una serie de innumerables razones técnicas, ha favorecido su rápida adopción por parte de la comunidad de desarrolladores.

Importante

Angular es una plataforma que permite desarrollar aplicaciones web utilizando **HTML** y **JavaScript** en la parte cliente descargando al servidor de buena parte del trabajo, con lo que se consigue una mayor velocidad en la ejecución y, por tanto, un mayor rendimiento.

Permite la creación de aplicaciones web de una sola página (**SPA: *single-page application***) realizando la carga de datos de forma asíncrona.

Además de mejorar el rendimiento de las aplicaciones web, su utilización en dispositivos móviles está optimizada ya que, en ellos, los ciclos de CPU y memoria son críticos para su óptimo funcionamiento. Ello permite el desarrollo de aplicaciones móviles híbridas con **Ionic 2**.

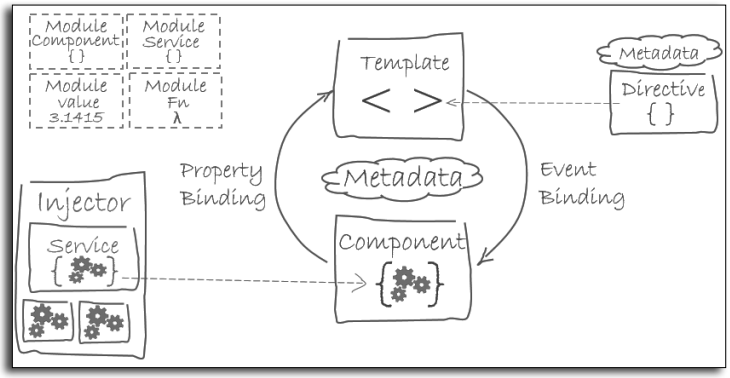
Gracias al uso de componentes, se puede encapsular mejor la funcionalidad facilitando el mantenimiento de las aplicaciones.

Parecería ser la continuación de **AngularJS,** pero, en realidad, más que una nueva versión es realmente un framework o plataforma diferente. El hecho de utilizar componentes como concepto único en lugar de controladores, directivas y servicios de forma específica, como sucedía en **AngularJS**, simplifica mucho las cosas.

Angular está orientado a objetos, trabaja con clases y favorece el uso del patrón **MVC** (**Modelo-Vista-Controlador**).

Permite el uso de **TypeScript** (lenguaje desarrollado por **Microsoft**) con las ventajas que supone poder disponer de un tipado estático y objetos basados en clases. Todo ello, gracias a la especificación **ECMAScript 6**, que es la base sobre la que se apoya **TypeScript**. Gracias a un compilador **(transpilador**) de **TypeScript**, el código escrito en este lenguaje se traducirá a **JavaScript** original.

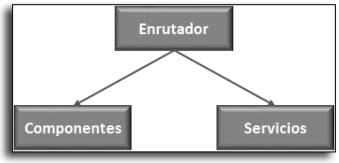
En la web de **Angular** podemos observar el diagrama de arquitectura que muestra cómo se relacionan los elementos principales de una app los cuales son los siguientes:



- Modules.
- Components.
- Templates.
- Metadata.
- Data binding.
- Directives.
- Services.
- Dependency injection.

Una visión más sencilla de la arquitectura **MVC** nos permitiría enumerar los siguientes elementos:

- **Vistas:** Componentes.
- **Capa de control:** Router.
- **Backend:** Servicios.



A lo largo del libro, analizaremos cada uno de estos elementos y veremos, entre otras muchas cosas, que los componentes son la suma de los templates, las clases y los metadatos.

Veremos también que la inyección de dependencias favorece el **testing** y reduce el acoplamiento entre clases. Con **Lazy SPA** no es necesario tener en memoria todos los elementos que componen una aplicación, de forma que podemos cargarla por partes a medida que los necesitemos.

Podemos ejecutar **Angular** en el servidor gracias a **Angular Universal**, con lo que podremos reducir el tiempo de espera que se produce en la primera visita ya que, en lugar de cargar todo lo necesario para ejecutar la aplicación, se pueden enviar vistas "prefabricadas" directamente al cliente.

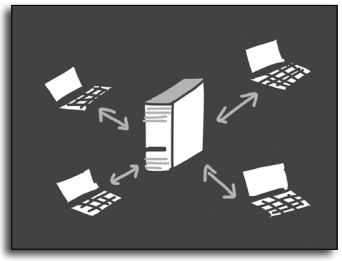
002 Introducción a las aplicaciones SPA

Angular es un framework principalmente enfocado a la creación de aplicaciones web de tipo single-page application (SPA). En este capítulo haremos un breve repaso a los distintos tipos de aplicación web y, a continuación, analizaremos con más detalle en qué consisten las aplicaciones single-page application (SPA).

Importante

Angular es un framework que ofrece todas las herramientas necesarias para crear aplicaciones de tipo single-page application (SPA).

Una aplicación web la podemos definir como toda aquella aplicación proporcionada por un servidor web y utilizada por los usuarios a través de un cliente web (browsers o navegadores). En otras palabras, son aquellas aplicaciones codificadas en un lenguaje soportado por los navegadores web para que puedan ejecutarse desde allí.



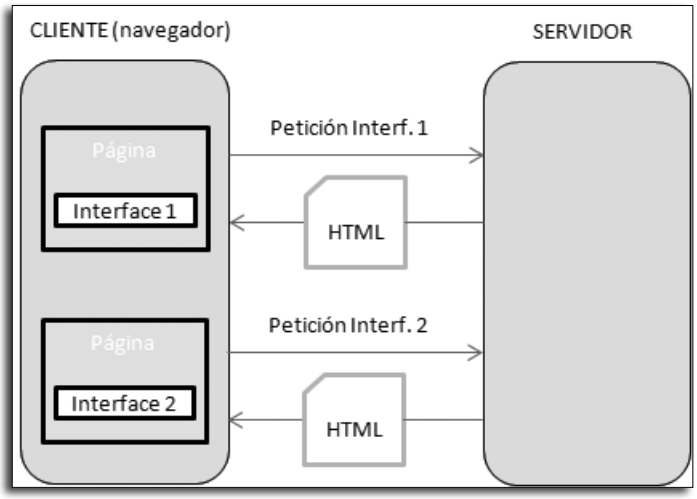
Toda aplicación web tiene una arquitectura básica de tipo cliente-servidor. Esto es así porque toda aplicación web sigue un modelo de aplicación distribuida en la que, por una parte, está el servidor que provee de recursos y servicios y, por la otra, está el cliente que los demanda. A partir de aquí, según lo estática o dinámica que sea cada parte, tenemos distintos tipos de aplicaciones web:

- Cliente y servidor estáticos: no hay ningún tipo de dinamismo y el servidor siempre devuelve los mismos recursos sin ningún tipo de cambio. Más que de aplicaciones web, aquí hablaríamos de páginas web.
- Cliente estático y servidor dinámico: el servidor devuelve recursos dinámicos, por ejemplo, páginas web con el resultado de consultas a base de datos, etc.
- Cliente/servidor dinámicos: el cliente es dinámico porque las páginas web recibidas del servidor incluyen JavaScript, que se ejecuta en el propio navegador dando todo tipo de funcionalidades diversas.

Hoy en día, la mayoría de aplicaciones web disponen de una parte cliente dinámica con el objetivo de disminuir la comunicación con el servidor y mejorar la fluidez y experiencia del usuario. Sin embargo, según hasta donde se lleve esta técnica, podemos hablar de grandes patrones de diseño:

- Multi page web applications (MPA).
- Single-page application (SPA).

Las aplicaciones multi page web applications (MPA) corresponderían al tipo tradicional de aplicación web. Su característica principal es que la mayoría de acciones de usuario se transforman en solicitudes al servidor de páginas completas (incluyendo diseño, css [hojas de estilo], JavaScript y contenido).



Este tipo de diseño es perfectamente funcional para cualquier tipo de aplicación; sin embargo, para aplicaciones complejas pueden existir problemas de lentitud en la navegación. Por ejemplo, una aplicación con una rica interfaz de usuario seguramente tendrá páginas muy complejas y, en este contexto, generar estas páginas en el servidor, transferirlas al cliente y visualizarlas en el navegador, puede requerir un tiempo considerable que afecte negativamente a la experiencia de usuario.

Sin embargo, para mejorar este posible problema de fluidez, las aplicaciones MPA suelen hacer uso de AJAX. AJAX no es una tecnología en sí misma, sino que es un conjunto de tecnologías independientes (JavaScript, XML, etc.) usadas con el fin de permitir refrescar de forma asíncrona partes de una página sin necesidad de recargarla toda de nuevo.

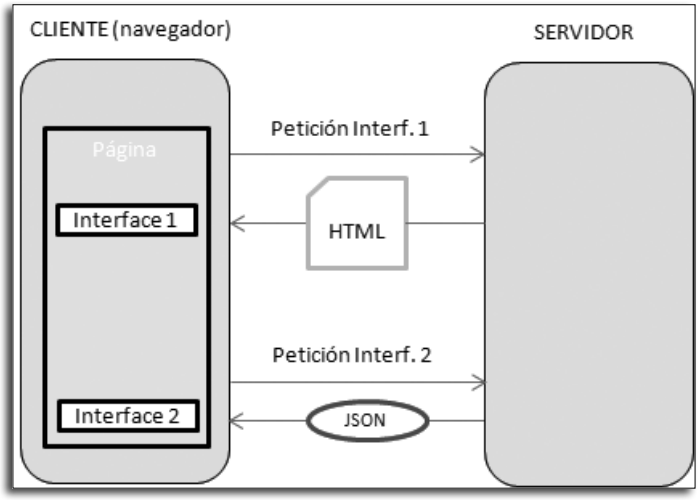
Por otra parte, el patrón de diseño single-page application (SPA) es una evolución del patrón de diseño MPA + AJAX, pero llevando al extremo el uso de AJAX. Hasta el punto de que en el cliente se carga una única página que se modifica desde el propio cliente (navegador) según las acciones de usuario. Por tanto, toda la navegación por las distintas pantallas o interfaces de la aplicación se realizará sin salir de esa única página.

Una de las principales ventajas de las aplicaciones SPA respecto las MPA es la mejora de experiencia de usuario debido a la reducción en el tiempo de respuesta ante las acciones del usuario. Esto se consigue gracias a que:

- Ya no se crean páginas completas por cada acción del usuario.
- Solo se intercambia la información necesaria con el servidor.

En las aplicaciones SPA, la responsabilidad del aspecto de la aplicación recae principalmente en la parte cliente, mientras que el servidor tiene la función de ofrecer al cliente una API de servicios para dar acceso a la base de datos de la cual se alimenta la aplicación. Los datos intercambiados entre cliente y servidor suelen estar en formato JSON, que es un formato más óptimo que el tradicional XML.

En las aplicaciones Web, la programación del lado cliente se realiza con html5 + JavaScript. Sin embargo, no suele usarse estos lenguajes de forma directa, sino que suele hacerse mediante librerías (jQuery) y/o frameworks (Angular, Backbone.js, Ember.js, etc.). Las librerías sirven para facilitar el uso de JavaScript aportando todo tipo de herramientas, pero los frameworks, aparte de aportar herramientas, aportan patrones de diseño para el desarrollo de aplicaciones complejas. Angular es uno de estos framework, quizás uno de los más usados y, como veremos a lo largo del libro, aporta todo lo necesario para crear y mantener aplicaciones single-page application (SPA).



003 Breve historia de Angular

El concepto de las single-page applications (SPA) empezó a debatirse en 2003, pero no fue hasta 2005 que se utilizó el termino por primera vez. Como vemos, se trata de un tipo de aplicación relativamente nueva, sin embargo, su uso se está extendiendo rápidamente y a día de hoy ya detectamos su presencia en aplicaciones tan conocidas como Gmail, Outlook, Facebook y Twitter.

El auge de este tipo de aplicaciones se debe en parte a las ventajas que aportan respecto a otras soluciones, pero, sobre todo, a la gran inversión que están haciendo grandes empresas tecnológicas para desarrollar frameworks que permitan su desarrollo. Este es el caso de Google con el framework Angular, o Facebook con el framework React.js.

Importante

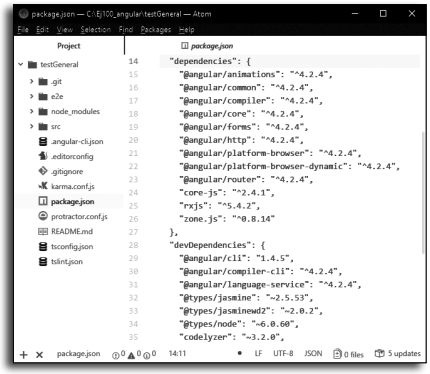
El framework Angular se rediseñó por completo en la versión 2.0, y pasó de llamarse AngularJS a Angular. Para no dejar colgados a los desarrolladores de AngularJS, Google sigue dando soporte y mantenimiento a AngularJS. En la actualidad, AngularJS se encuentra en la versión 1.6.4, y Angular en la versión 4.0.

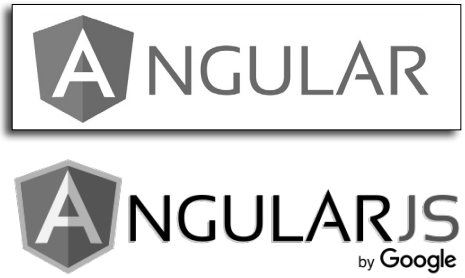
AngularJS fue desarrollado en 2009 por Miško Hevery y Adams Abrons. Originalmente era el software detrás de un servicio de almacenamiento online de archivos JSON, pero, poco tiempo después, se abandonó el proyecto y se liberó AngularJS como una biblioteca de código abierto. Adams Abrons dejó el proyecto, pero Miško Hevery, como empleado de Google, continuó el desarrollo y mantenimiento del framework.

Desde entonces se han ido lanzando muchas versiones de AngularJS, pero, sobre todas ellas, cabe destacar la salida de la **versión 2.0** por los siguientes motivos:

- **Se rediseñó por completo todo el framework** (nueva arquitectura basada en componentes, etc.).
- Se introdujo el uso de **TypeScript** de Microsoft (superconjunto de JavaScript) como lenguaje de programación.
- Se cambió el nombre del framework. Pasó de llamarse **AngularJS** a **Angular.**
- Los desarrolladores anunciaron que darían soporte y mantenimiento tanto para AngularJS (versiones 1.X.Y) como para Angular (versiones superiores), pero que una y otra seguirían ciclos de vida independientes. Actualmente, **AngularJS se encuentra en la versión 1.6.4**, mientras que **Angular se halla en la versión 4.0.**

En un primer momento, la salida de Angular tuvo muchas críticas por parte de los desarrolladores que usaban AngularJS, ya que veían como la actualización de sus desarrollos al nuevo framework requería un trabajo muy laborioso de migración. Por otra parte, también veían que su experiencia y conocimiento no les servía de mucho con los nuevos conceptos de Angular.





Sin embargo, a medida que los desarrolladores apreciaron las ventajas de la nueva versión, la situación mejoró. También ayudaron las distintas iniciativas de Google, como anunciar que seguirían dando soporte y mantenimiento a las antiguas versiones del framework (AngularJS) o la de documentar un protocolo de migración de los desarrollos al nuevo Angular (https://angular.io/guide/upgrade).

A continuación, sin entrar en demasiado detalle, podremos analizar algunas de las diferencias entre AngularJS y Angular mediante un sencillo ejemplo. El ejemplo consiste en una aplicación web que muestra un "Hola mundo". Primero lo veremos implementado en AngularJS y luego en Angular.

Ejemplo AngularJS

- index.html:

```
<!DOCTYPE html>
<html ng-app="testAngularJS">
  <head>
    <link rel="stylesheet" type="text/css" href="bootstrap.min.css" />
    <script type="text/javascript" src="angular.min.js"></script>
    <script type="text/javascript" src="app.js"></script>
  </head>
  <body>
    <entity-directive1></entity-directive1>
  </body>
</html>
```

- app.js: En este archivo tenemos definido el módulo y la directiva de tipo "entidad" a los que se hacía referencia en el index.html.

```
(function() {
    var app = angular.module('testAngularJS', []);
    app.directive('entityDirective1', function() {
      return {
        restrict: 'E',
        template: '<h1>{{title}}</h1>',
        scope: {},
        link: function(scope, element){
          scope.title='AngularJS: Hola mundo!!';
        }
      };
    });
  })();
```

Ejemplo Angular. Aparte de los archivos indicados a continuación, el proyecto incluye muchos otros archivos relacionados con la carga de bibliotecas necesarias y el uso de TypeScript. En la imagen anterior puede ver algunas de estas dependencias referenciadas en el archivo **package.json** de un proyecto Angular.

- index.html:

```
…
<body>
  <app-root>Loading...</app-root>
</body>
…
```

- *app.component.ts:* Definición del componente usado en el index.html.

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

@Component({
  selector: 'app-root',
  template: '<h1>{{title}}</h1>',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular: Hola mundo!!!!';
}
```

004 Instalación

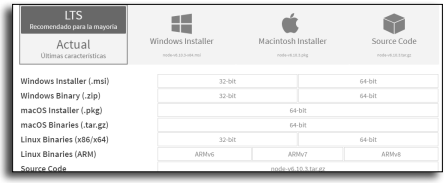
Para trabajar con **Angular** necesitamos realizar diversas instalaciones, algunas de las cuales son imprescindibles, como por ejemplo Node.js, y otras son opcionales, aunque recomendadas, ya que nos facilitarán enormemente el trabajo. Por ejemplo, el usuario puede utilizar el editor que más le guste, pero recomendamos emplear algunos editores que aportan un sinfín de utilidades a la hora de comprobar la sintaxis de nuestras sentencias o de aportar bloques de código a modo de plantilla.

Importante

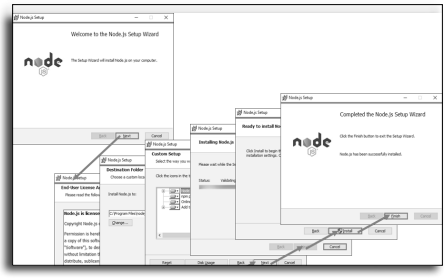
Procure instalar la última versión de los productos indicados para mantenerse actualizado y disfrutar de las últimas actualizaciones en todo momento.

1. En primer lugar instalaremos **Node.js**. Para ello podemos descargarlo desde la siguiente página:

   https://nodejs.org/es/download/

   En esta página, seleccionaremos la opción que más nos interese según nuestro SO y el número de bits (**32** o **64**). En nuestro caso, seleccionamos la opción **Windows Installer (.msi) 64-bit** y descargamos el instalador en una carpeta desde la que realizaremos posteriormente la instalación.

   Una vez descargado, ejecutaremos el instalador y podemos aceptar todos los valores que nos propone por defecto hasta que lleguemos a la pantalla final y pulsemos **Finish**.

2. Seguidamente, instalaremos **TypeScript**. Para ello, podemos acudir al siguiente link:

   https://www.typescriptlang.org/

Al pulsar sobre el botón **download** aparece una pantalla en la que se muestra la sentencia a ejecutar para instalar **TypeScript**

```
npm install -g typescript
```

A continuación, copiaremos dicha sentencia y abriremos una ventana de **CMD** (**Ejecutar->CMD**) para pegarla y ejecutarla.

3. A continuación, instalaremos **Angular-Cli**. Para ello, acudiremos a la siguiente página:

https://cli.angular.io/

En dicha página, ya podemos ver diversas instrucciones entre las que hallamos la siguiente:

```
npm install -g @angular/cli
```

De nuevo, abrimos una ventana **CMD** (**Ejecutar->CMD**) y ejecutaremos la sentencia anterior.

Para obtener más información sobre los detalles de la instalación y prerrequisitos, puede consultar la siguiente página:
https://github.com/angular/angular-cli

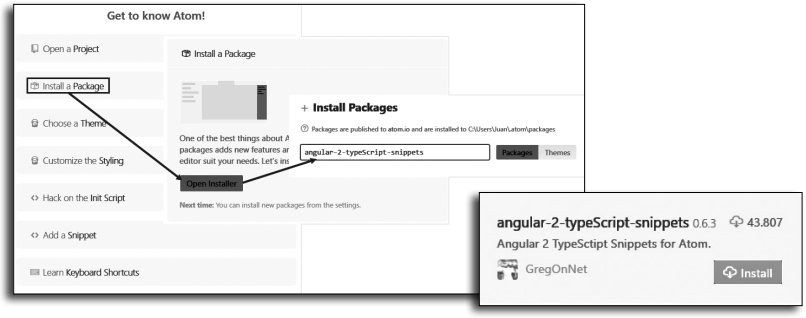
4. Podemos realizar nuestras pruebas en muchos navegadores, pero nosotros realizaremos todas las explicaciones basándonos en **Google Chrome** y en sus herramientas para desarrolladores. Por tanto, sugerimos que utilice este navegador, el cual puede descargar de la siguiente página:

https://www.google.es/chrome/browser/desktop/index.html

5. Por último, tal y como hemos indicado en la introducción, necesitará un editor que le ayude a desarrollar sus componentes con **Angular** y puede utilizar alguno de los siguientes:

Atom	https://atom.io/
Visual Studio Code	https://code.visualstudio.com/download
Sublime Text	https://www.sublimetext.com/3

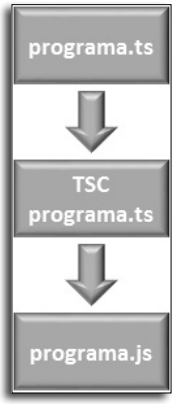
Nosotros utilizaremos **Atom** en nuestras explicaciones y, en este caso, una vez instalado **Atom**, recomendamos añadir una serie de funcionalidades que nos facilitarán el desarrollo con Angular. Para añadir funcionalidades, accederemos a **Help->Welcome Guide** y, seguidamente, accederemos a **Install a Package->Open Installer**. Cuando aparezca la pantalla de **Install Packages**, teclearemos el nombre del paquete que queramos añadir y, una vez que nos muestre el package deseado, pulsaremos sobre **Install**.



Algunas de las funcionalidades recomendadas se hallan en los siguientes packages:

- Angular-2-TypeScript-Snippets,
- Angular2-Snippets,
- Atom-Bootstrap 3,
- Atom-TypeScript,
- file-icons,
- Linter,
- Linter-UI-Default,
- PlatformIO-IDE-Terminal.

005 TypeScript. Introducción (variables, clases, transpilación, etc.)



TypeScript es un lenguaje de programación orientado a objetos fuertemente tipado que se traduce a **JavaScript** añadiéndole características que no posee. La operación de traducir TypeScript a JavaScript se conoce como **transpilación**.

Gracias al uso de TypeScript, es posible localizar errores de sintaxis antes incluso de su ejecución. De ahí que haya ganado mucha aceptación entre los desarrolladores del mundo web.

Importante

TypeScript permite desarrollar un código con menos errores y aprovechar toda la potencia de una programación orientada a objetos.

Podríamos dedicar un libro completo a TypeScript, pero, en este ejercicio, simplemente ofreceremos una pincelada sobre el lenguaje y elaboraremos el clásico programa **Hola Mundo** para que podamos ver cómo se compila y ejecuta.

Los programas escritos con TypeScript se suelen escribir en ficheros con la extensión **ts**.

Una vez escritos, dichos programas se transpilan (compilan) o, lo que es igual, se traducen a JavaScript para que puedan ejecutarse posteriormente. Cualquier instrucción escrita en JavaScript "puro" se copia igual (sin traducción) en el fichero resultante de la transpilación.

Para compilar usaremos el comando **TSC** y para ejecutar emplearemos el comando **node**, como veremos más adelante y el cual ya hemos instalado en capítulos anteriores.

La sintaxis que hallaremos en TypeScript está compuesta por módulos, funciones, variables, sentencias, expresiones y comentarios.

Respecto de las funciones, TypeScript permite definir funciones con nombre, anónimas, tipo Lambda o de flecha, además de permitir la sobrecarga de funciones utilizando diferentes parámetros y/o tipos en sus llamadas.

Asimismo, disponemos de un gran surtido de **operadores**, **sentencias condicionales**, **loops**, **funciones** y **métodos** para **numéricos** y **strings**.

También es posible definir interfaces y clases, incluyendo en las mismas campos, constructores y funciones utilizando la herencia y encapsulación propias de **POO**.

Tipo operador	Operadores
Aritméticos	+, -, *, /, %, ++, --
Relacionales	>, <, >=, <=, ==, !=
Lógicos	&&, ||, !
Asignaciones	=, +=, -=, *=, /=
Condicional	test ? expre1 : expre2
Typeof	Devuelve tipo de datos del operando

```
while(condicion)
{
   // sentencias si condición = true
}
```

```
for (variable in lista)
{
   //sentencias
}
```

```
for (valor inicial; condición para fin; incremento)
{
   //sentencias
}
```

```
do
{
   //sentencias
}while(condicion)
```

```
if (expresión)
{
   // sentencias;
}
```

```
if (expresión)
{
   // sentencias;
}
else
{
   // sentencias;
}
```

```
switch(expresión)
{
  case constante1:
  {
    // sentencias;
    break;
  }
  case constante2:
  {
    // sentencias;
    break;
  }
  default:
  {
    // sentencias;
    break;
  }
}
```

```
if (expresión)
{
   // sentencias;
}
else if (expresión)
{
   // sentencias;
}
else
{
   // sentencias;
}
```

```
function nombre_funcion():tipo_retorno
{
   //sentencias
return valor;
```

Numéricos	String		
• toExponential()	• charAt()	• slice()	
• toFixed()	• charCodeAt()	• split()	
• toLocaleString()	• concat()	• substr()	
• toPrecision()	• indexOf()	• substring()	
• toString()	• lastIndexOf()	• toLocaleLowerCase()	
• valueOf()	• localeCompare()	• toLocaleUpperCase()	
	• match()	• toLowerCase()	
	• replace()	• toString()	
	• search()	• toUpperCase()	
		• valueOf()	

Para acabar con la breve mención al lenguaje, es importante resaltar que el concepto de **namespace** permite organizar mejor el código y sustituye a los antiguos **Internal Modules**.

A continuación, vamos a crear un programa con TypeScript que simplemente muestre el texto **Hola Mundo**:

1. Para ello, nos ubicaremos en nuestro directorio de ejercicios (**Ej100_angular**) y crearemos el directorio **005_HolaMundo_TS**.

2. Seguidamente, abriremos nuestro editor favorito y, dentro del directorio recién creado, crearemos el fichero **005_HMTS.ts** con el siguiente contenido:

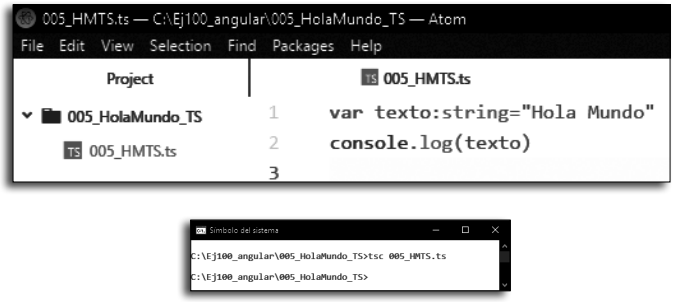
```
Símbolo del sistema
C:\>cd Ej100_angular

C:\Ej100_angular>mkdir 005_HolaMundo_TS

C:\Ej100_angular>
```

```
var texto:string=”Hola Mundo”
console.log(texto)
```

3. Una vez creado y guardado el fichero, accederemos a la ventana de **CMD** y nos ubicaremos en nuestro nuevo directorio (**005_HolaMundo_TS**) para teclear lo siguiente:



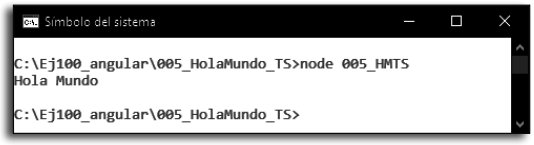

```
tsc 005_HMTS.ts
```

4. Observaremos que, si no se ha producido ningún error, la compilación se realiza de forma silenciosa devolviendo el control al prompt de la ventana de **CMD**.

5. En este punto, procederemos a la ejecución de nuestro programa tecleando lo siguiente:

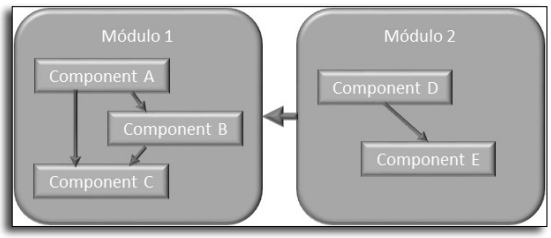
```
node 005_HMTS
```

6. Observaremos como, efectivamente, se muestra el texto **Hola Mundo** en pantalla.



006 Definición de elementos en una aplicación

En el capítulo de introducción, enumeramos los elementos que había que tener en cuenta en Angular. A continuación, los definiremos brevemente.

Elemento	Comentario
Módulo	Conjunto de códigos dedicados a resolver un objetivo que generalmente exporta una clase. Cada aplicación posee al menos una clase de módulo que denominamos "**módulo raíz**" y que, por defecto, posee el nombre de **AppModule**. En el mismo podemos apreciar una clase con un decorador (**@NgModule**) y una serie de propiedades entre las que destacamos las siguientes: • **Declarations:** define las vistas que pertenecen a este módulo. • **Exports:** declaraciones que han ser visibles para componentes de otros módulos. • **Imports:** clases que otros módulos exportan para poderlas utilizar en el módulo actual. • **Providers:** servicios usados de forma global y accesibles desde cualquier parte de la aplicación. • **Bootstrap:** la vista principal o componente raíz, que aloja el resto de vistas de la aplicación. Propiedad establecida solo por el módulo raíz. Un módulo puede agrupar diversos componentes relacionados entre sí. 

Componente	Elemento que controla una zona de espacio de la pantalla que representa la "vista". Define las propiedades y métodos que usa el propio template y contiene la lógica y la funcionalidad que utiliza la vista. Pueden tener atributos tanto de entrada (decorador **@Input**) como de salida (decorador **@Output**). También podemos definir al componente como: **Componente = template + metadatos + clase**
Template	Define la vista de un **Componente** mediante un fragmento de **HTML**.
Metadatos	Permiten decorar una clase y configurar así su comportamiento. Extiende una función mediante otra sin tocar la original.
Data binding	Permite intercambiar datos entre el template y la clase del componente que soporta la lógica del mismo. Existen 4 tipos de **Data binding** (que analizaremos en profundidad más adelante) y que son: • **Interpolación:** muestra el valor de una propiedad en el lugar donde se incluya **{{ propiedad }}**. • **Property binding:** traspaso del objeto del componente padre al componente hijo (**[objHijo]="objPadre"**). • **Event binding:** permite invocar un método del componente pasándole un argumento al producirse un evento (p. ej., **(click)="hazAlgo(obj)"**). • **Two-way binding:** binding bidireccional que permite combinar **event** y **Property binding** (p. ej., **input [(ngModel)]= "obj"**).
Directiva	Permite añadir comportamiento dinámico a HTML mediante una etiqueta o selector. Existen directivas estructurales (***ngFor** o ***ngIf**) o de atributos que modifican el aspecto o comportamiento de un elemento **DOM** (**NgSwitch**, **NgStyle** o **NgClass**).
Servicio	Son clases que permiten realizar acciones concretas que ponen a disposición de cualquier componente. Por ejemplo, permiten obtener datos, resolver lógicas especiales o realizar peticiones a un servidor.
Dependency injection	Permite suministrar funcionalidades, servicios, etc., a un componente sin que sea el propio componente el encargado de crearlos. Utiliza el decorador **@Injectable()**. Generalmente proporciona servicios y facilita los test y la modularidad del código.

Al crear una aplicación, por defecto se crea el módulo **app.module.ts**, como veremos en ejercicios posteriores.

Importante

Cree módulos para separar los grupos de componentes que dan soporte a una lógica determinada y que pueden reutilizarse en diversas aplicaciones.

En el mismo, podemos apreciar diferentes partes correspondientes a:

- Los "**imports**":
  - **BrowserModule:** usado para renderizar la aplicación en el navegador.
  - **NgModule:** decorador necesario para el módulo.
  - **AppComponent:** componente principal del módulo.

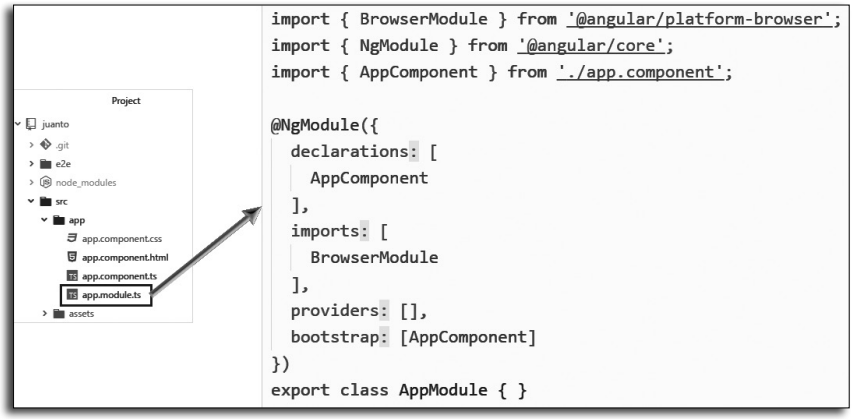
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
```

- Al decorador **@NgModule** con su metadata:
  - **Declarations:** AppComponent (vista del módulo por defecto).
  - **Imports:** BrowserModule (clase necesaria para el navegador).
  - **Providers:** en este caso, no disponemos de ninguno.
  - **Bootstrap:** AppComponent (componente principal).
- A su clase: permite cargar la aplicación **(AppModule)**.

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

El archivo total agrupa todas las partes descritas que constituyen el módulo en sí mismo.



007 Definición de un componente

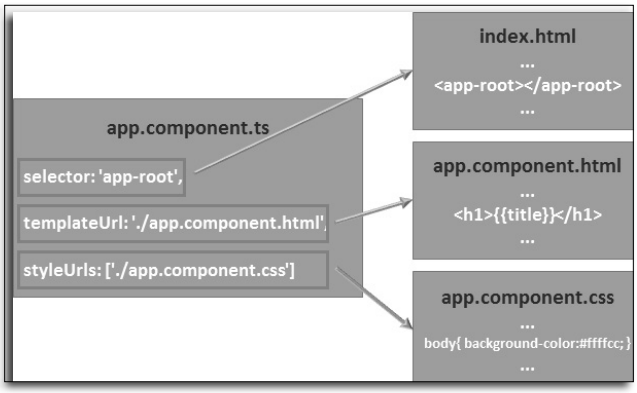
Un componente es una clase a la que se le añade un decorador **(@Component)** y que permite crear nuevas etiquetas **HTML** mediante las cuales podemos añadir funcionalidad a nuestras páginas controlando determinadas zonas de pantalla. Básicamente, los componentes se organizan en forma de árbol teniendo un componente principal donde, por defecto, su propiedad "selector" posee el valor **app-root**, gracias al cual el componente puede incluirse en nuestras páginas **HTML** (p. ej., en **index.html**) mediante el tag **<app-root></app-root>** como veremos más adelante.

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>007 Componentes</title>
  <base href="/">
  <meta name="viewport"
        content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon"
        href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Los componentes se componen de tres partes:

- **Anotaciones:** metadatos que agregamos a nuestro código y que permite definir ciertas propiedades como son:



- **Selector:** nombre a utilizar como tag en HTML para indicar a Angular que cree una instancia del componente cuando se encuentre con dicho tag.
- **Template:** plantilla a utilizar para el componente y que se describe en el propio fichero **ts** del mismo.
- **TemplateUrl:** indica el URL en el que se halla la plantilla en caso de definirse externamente.
- **Style:** definición de estilos para la plantilla.
- **StyleUrl:** array que contiene los diferentes archivos de estilos que pueden utilizarse en el componente en curso.
- **Host:** permite encapsular ciertas partes del componente.
- **Directive:** array con las directivas que pueden utilizarse en el componente.
- **Input:** define variables que recogen información del padre.
- **Output:** define variables que pasan información al padre.

- **Vista:** contiene el template que usaremos para elaborar la pantalla asociada al componente. Esta pantalla podemos definirla en un archivo auxiliar o bien dentro del propio archivo donde definimos la clase controlador encerrando el código HTML entre apóstrofos (`` `<h1>{{title}}</h1>` ``).

app.component.html

```
<div class="container">
  <h1>{{title}}</h1>
</div>
```

- **Controlador:** clase que contendrá la lógica del componente (definiciones, funciones, etc.)

```
export class AppComponent {
  title = '007 Componentes';
}
```

Generalmente, un componente suele llevar asociados diversos archivos con el siguiente cometido:

- **app.component.html:** contiene el código HTML mediante el que fabricamos la pantalla (o **Vista**).

- **app.component.css:** define los estilos que usaremos en la vista del componente.

- **app.component.ts:** archivo que contiene la lógica del componente (**Controlador**) y que se escribe en **TypeScript** para ser traducido a **JavaScript** posteriormente.

- **app.component.spec.ts:** archivo usado para la realización de test unitarios.

Importante

Cree tantos componentes como pantallas necesite y compártalos entre módulos para reutilizarlos.

En ejercicios posteriores, veremos que, al generar una aplicación con determinadas utilidades (p. ej., **Angular-CLI**), se crea un componente por defecto con una serie de bloques como los que se han descrito, cuyo código es similar al siguiente: