UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Desarrollo con Angular

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Unidad 1: Introducción a Angular

Presentación:

En la presente unidad incursionamos en el maravilloso mundo de Angular, framework desarrollado en javascript, utilizando typescript para facilitar el proceso durante la etapa de desarrollo.

Objetivos:

Que los participantes:

- Entiendan qué es Angular.
- Comprendan las ventajas y desventajas dadas por el framework
- Realicen una primera aplicación utilizando componentes en Angular.

Bloques temáticos:

- Angular.
- Angular CLI.
- Crear nuestra primera aplicación.
- Estructura de directorios.
- Estructura de directorios - src.
- Sintaxis para las vistas.
- Componentes.
- Decorador de componentes.
- Crear un nuevo componente.
- Usar un componente creado.

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

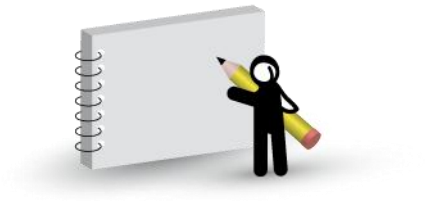
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***



Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Angular

Introducción

Angular 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 haya eliminado el "JS" hasta del nombre del dominio, pero es representativo. No porque ahora Angular no sea Javascript, sino porque es evolución radical.

Angular es otro framework, no simplemente una nueva versión. A los que no conocían Angular 1 esto 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, cuanta más experiencia en el desarrollo se tenga, más sencillo será lanzarse a usar Angular porque muchas cosas sonarán de antes.

Problemas de angular 1

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.

Soluciones implementadas en Angular

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.

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, 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.

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 esto, 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 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.

Angular CLI

Comenzar a través de un comando

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

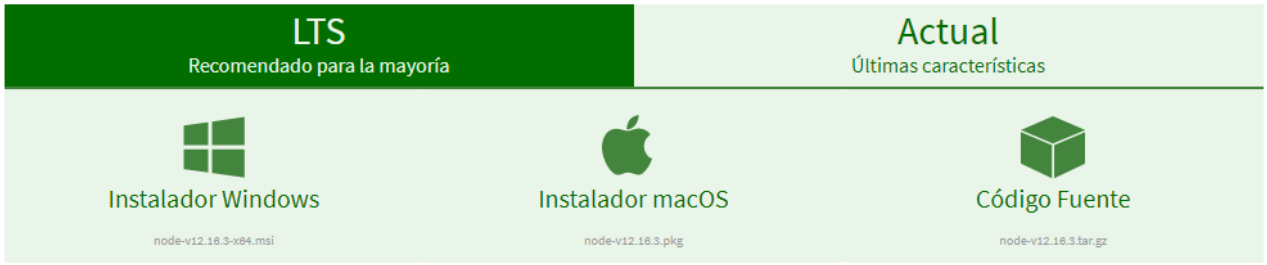
Angular CLI no es una herramienta de terceros, sino que nos la ofrece el propio equipo de Angular. Nos facilita mucho el proceso de inicio de cualquier aplicación con Angular, ya que en pocos minutos te ofrece el esqueleto de archivos y carpetas que vas a necesitar, junto con una cantidad de herramientas 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.

Instalar Node JS y NPM

Instalar node JS

Para poder instalar Angular CLI debemos instalar previamente node js y el gestor de paquetes NPM.

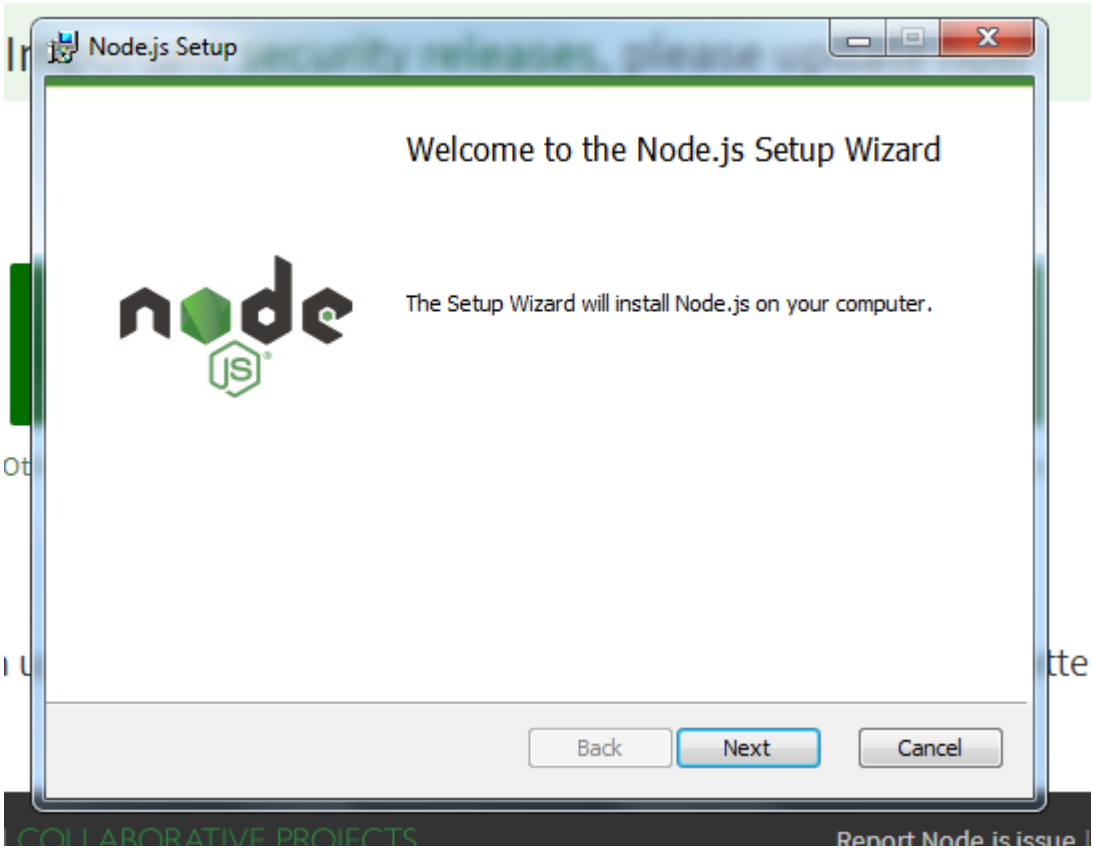
- Ingresar a: https://nodejs.org/en/
- Descargar la versión recomendada



Seleccionar versiones LTS

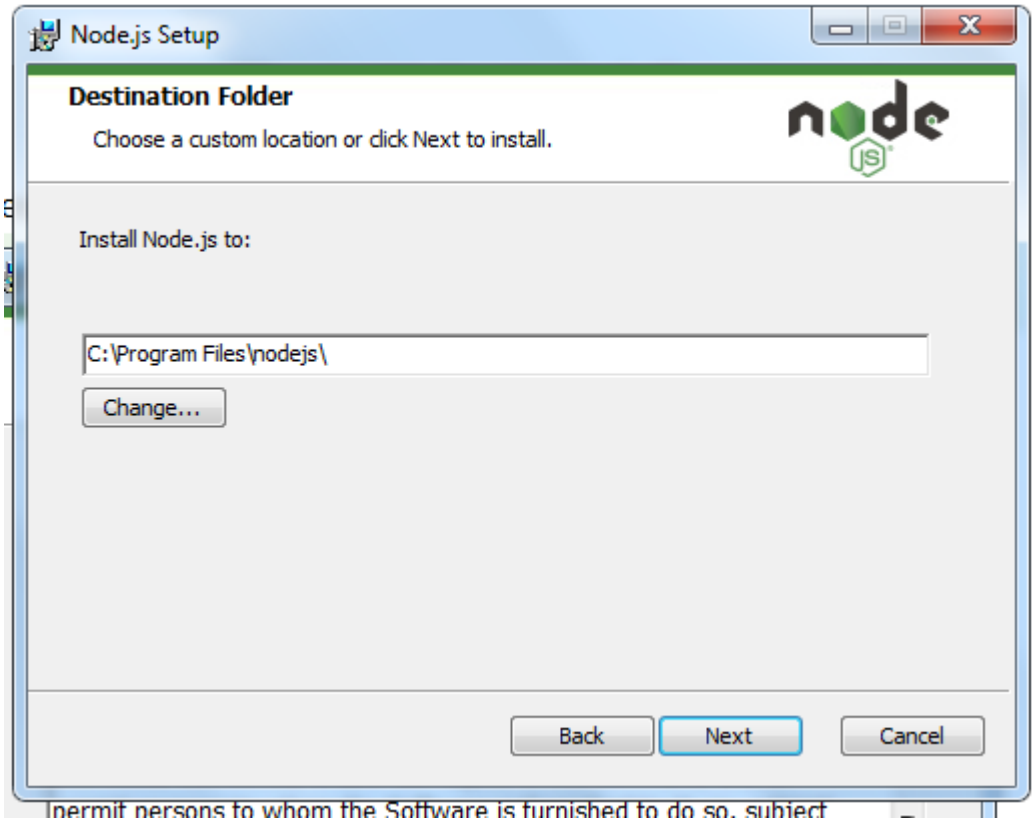
- Ejecutar el archivo descargado

- Hacer clic en “Siguiente” o “Next”

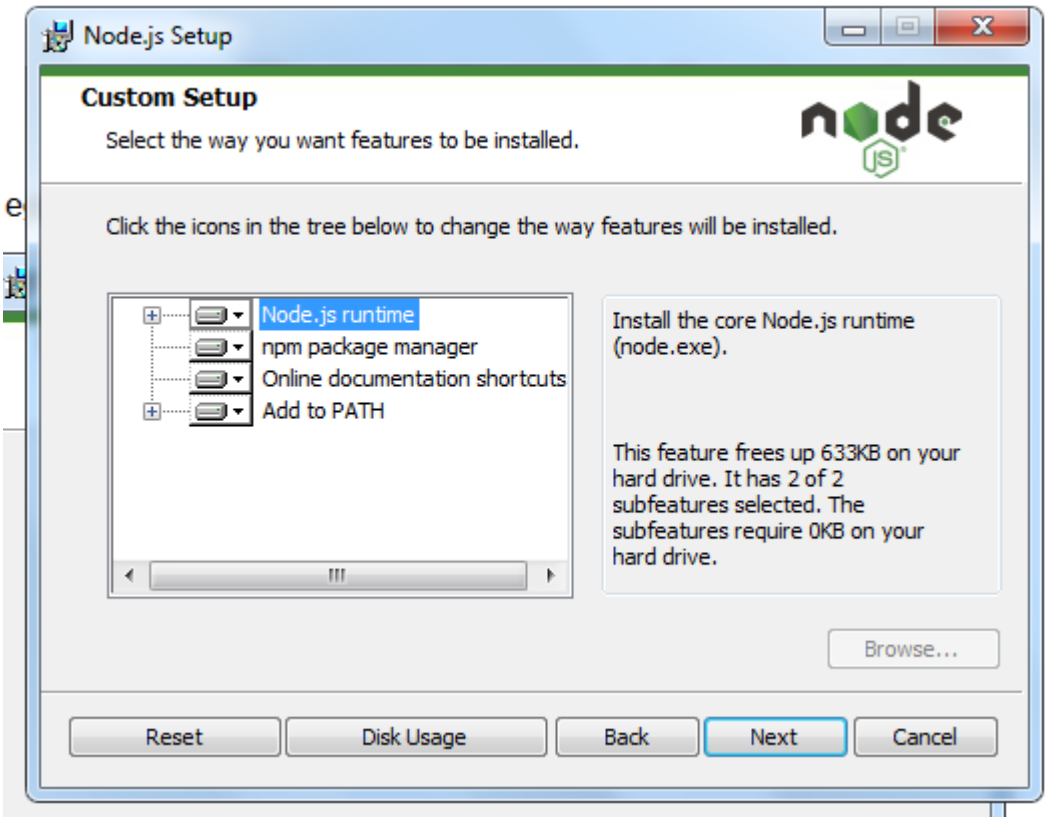


- Aceptar términos y condiciones y apretar “Next”

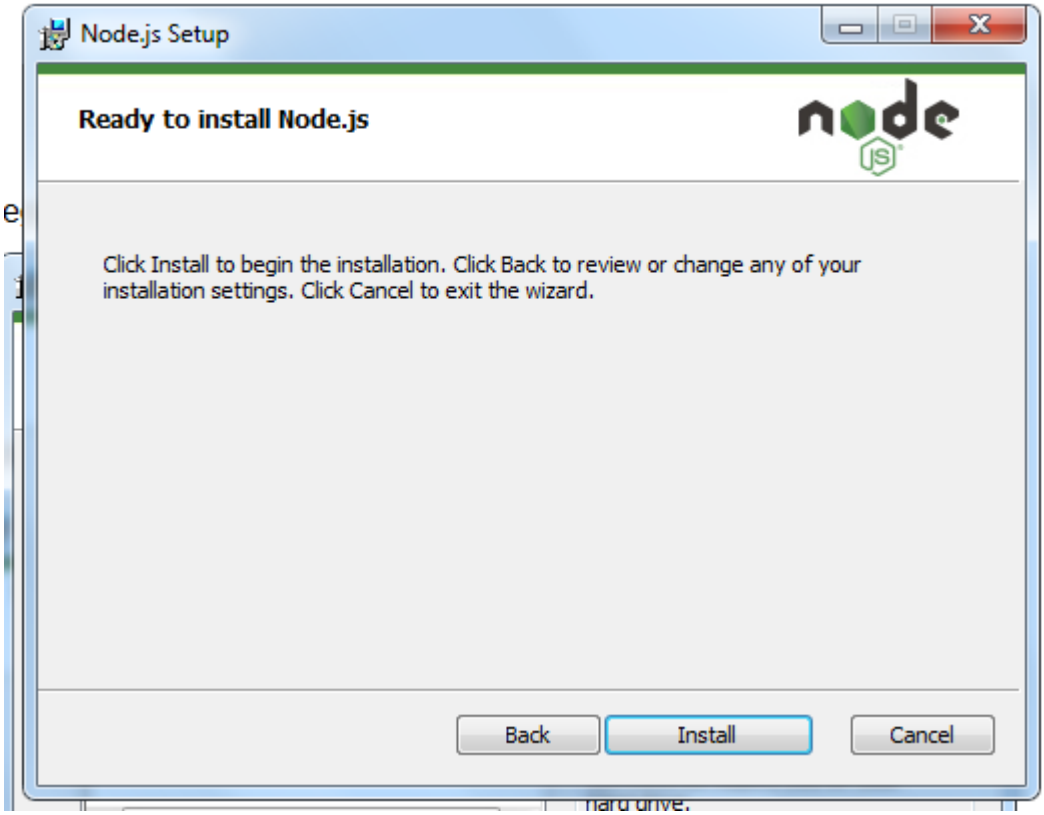
- Elegir directorio de instalación



- Elegir configuraciones a instalar (Dejar todas por defecto).



- Finalmente apretar en “Install”.



Instalar Angular CLI

Install

Para instalar angular CLI debes ejecutar en la consola lo siguiente:

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

npm install –g @angular/cli

Durante este proceso se instalara la herramienta Angular CLI con todas las dependencias necesarias.

ng

Mediante el comando **ng** podrás lanzar cualquiera de las acciones disponibles en angular, por ejemplo

ng –help

Podemos encontrar más información sobre cómo utilizar angular CLI en https://cli.angular.io/

Crear nuestra primera aplicación

new

El comando **ng new *nombre-app*** nos creara un nuevo proyecto Angular. 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.

```
C:\sites\angular>ng new angularNueve
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

Seleccionar Y (para instalar el modulo de angular routing) y luego elegir CSS

En este caso se creó la aplicación **angularNueve**

Serve

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".

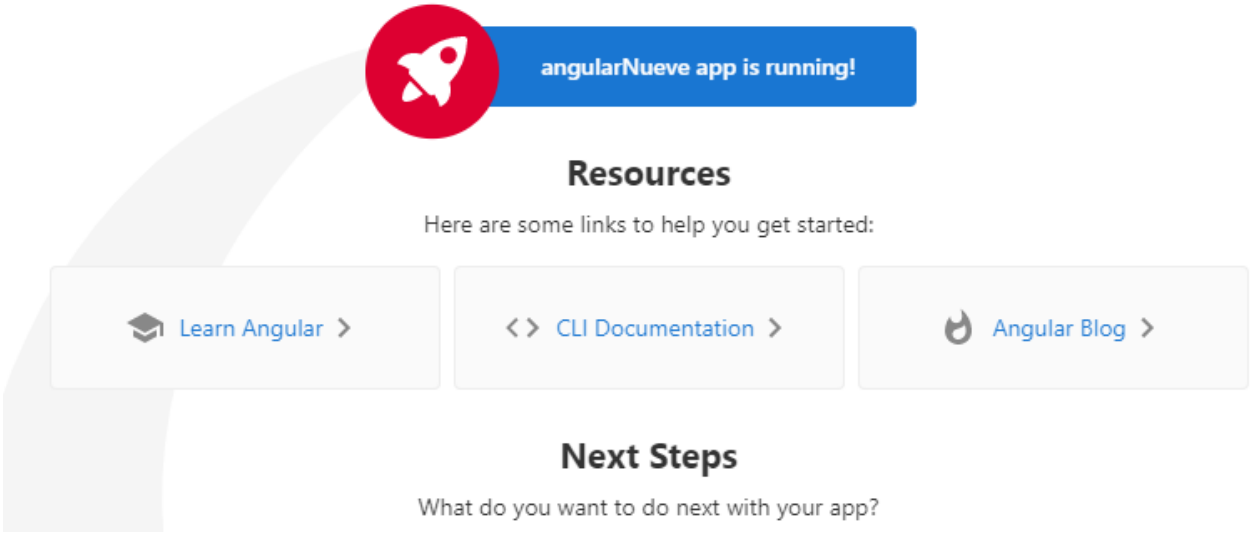
Para lanzar la aplicación en el navegador debemos ingresar al directorio en el cual se creó la misma. Siguiendo el ejemplo anterior debemos hacer (desde línea de comandos) **cd angularNueve**

Una vez dentro del directorio ejecutamos **ng serve**

```
C:\sites\angular\angularNueve>ng serve
Compiling @angular/core : es2015 as esm2015
Compiling @angular/animations : es2015 as esm2015
```

La aplicación por defecto se lanzara en **http://localhost:4200/,** en caso de querer modificar el puesto debemos ejecutar **ng serve –port 4201** (o el numero de puerto que queramos configurar).

Abrimos el navegador y colocamos la URL mencionada anteriormente, debemos ver algo como esto:



¡Ya tenemos nuestra primer aplicación en Angular funcionando!

Estructura de directorios

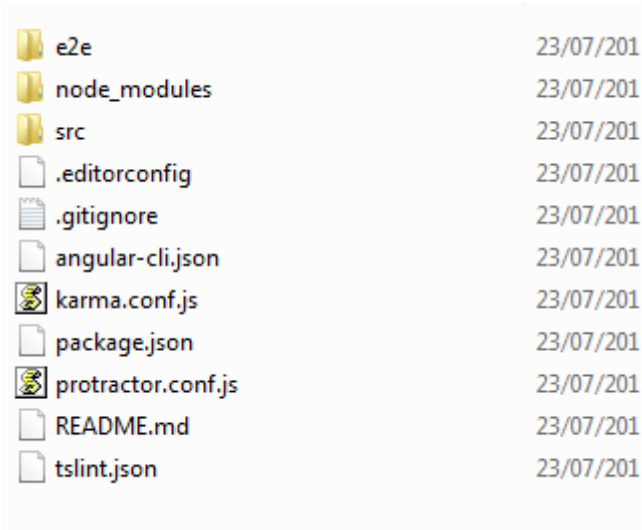
¿Qué editor de código usar con Angular?

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 gratuito 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 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

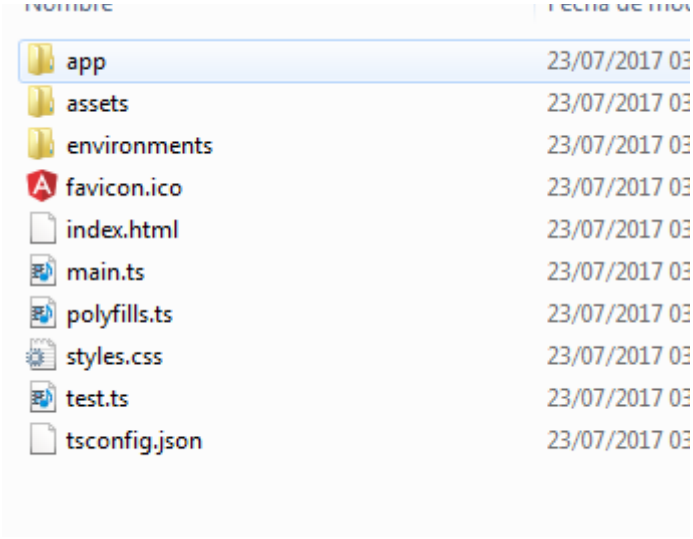
Archivos del directorio raíz: Seguro que muchos de los lectores reconocen muchos de los archivos que hay dentro, como package.json (descriptor de dependencias npm) o .gitignore (archivos y carpetas que git debería ignorar de este proyecto cuando se añada al repositorio). En resumen, todo lo que encontraremos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



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.


app 23/07/2017 03
assets 23/07/2017 03
environments 23/07/2017 03
favicon.ico 23/07/2017 03
index.html 23/07/2017 03
main.ts 23/07/2017 03
polyfills.ts 23/07/2017 03
styles.css 23/07/2017 03
test.ts 23/07/2017 03
tsconfig.json 23/07/2017 03

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.

.

package.json

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, la modularización del código. Encontrarás que una aplicación Angular necesita ya de entrada diversos módulos como son "common", "compiler", "core", etc.

```
{
  "name": "angular-ejemplo",
  "version": "0.0.0",
  "license": "MIT",
  "angular-cli": {},
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "test": "ng test",
    "pree2e": "webdriver-manager update --standalone false --gecko false",
    "e2e": "protractor"
  },
  "private": true,
  "dependencies": {
    "@angular/common": "^2.3.1",
    "@angular/compiler": "^2.3.1",
    "@angular/core": "^2.3.1",
    "@angular/forms": "^2.3.1",
    "@angular/http": "^2.3.1",
    "@angular/platform-browser": "^2.3.1",
    "@angular/platform-browser-dynamic": "^2.3.1",
    "@angular/router": "^3.3.1",
    "core-js": "^2.4.1",
    "rxjs": "^5.0.1",
    "ts-helpers": "^1.1.1",
    "zone.js": "^0.7.2"
  },
  "devDependencies": {
```

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".

Estructura de directorios - src

Index.html

De lo que es Angular lo único interesante es el componente **<app-root>Loading...</app-root>**. Todo, incluida la aplicación principal, debe ser definido y declarado como un componente.

De hecho definiremos las aplicaciones Angular como árboles de componentes. Y todo árbol debe tener una raíz. Mientras Angular no entre en funcionamiento, el usuario verá el mensaje de Loading... después la magia de Angular lo sustituirá por el contenido del componente **app-root** predefinido por el generador.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>AngularEjemplo</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>Loading...</app-root>
</body>
</html>
```

main.ts

Centrándonos en el código que habremos de mantener fíjate en la línea **import { AppModule } from './app/';.** Le indica a WebPack que importe el contenido de la carpeta **./app/.** Para ello buscará en dicho directorio un archivo index.ts. Ese fichero sirve de índice y contiene las instrucciones para exportar el código interesante del resto de la carpeta. En nuestro caso son el módulo y el componente raíz.

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Módulo raíz

Las aplicaciones Angular están pensadas para crecer. Para ello es fundamental cierto grado de modularidad. Dentro del archivo **app/app.module.ts** te encontraras con algo como lo siguiente:

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

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

Un módulo no es más que una clase contenedora. Cada módulo puede incluir múltiples componentes y servicios. Normalmente un módulo dependerá de otros. El módulo raíz declara un componente especial para el arranque de la aplicación: **El componente raíz**

Componente Raíz

Buceando a mayor profundidad nos encontramos con el resto del contenido de la carpeta **./app/.** Son archivos con nombres tipo **app.component.*** y se usan para definir un componente.

Los componentes son los bloques de construcción de Angular que representan regiones de la pantalla. Las aplicaciones se definen como árboles de componentes. Nuestra aplicación es un árbol que tiene una raíz, habitualmente llamado **app** y que es común a cualquier desarrollo.

Cada componente a su vez está formado por tres partes:

1. **La vista**: es el código que se renderizará para los usuarios. Esta plantilla estará en un fichero de extensión .html.
2. **La clase controladora**: En ES6 usaremos clases para declarar los controladores que exponen datos y funcionalidad a la vista.
3. **Metadata**: Se declara como un decorador, una función especial de TypeScript, que recibe un objeto de configuración. Esto acompaña al controlador en un fichero de extensión .ts

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'angularNueve';
}
```

Seguro que la parte más novedosa es **@Component({...}).** Es el equivalente a los antiguos Objetos de Definición de Directivas. Lo que hace es asociar al controlador una plantilla HTML **app.component.html** y un selector para ser invocado desde otra vista **<app-root></app-root>**.

La función decoradora (@component) se encarga declara de diversas cuesiones.

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 AppComponent {
  title = 'angularNueve';
}
```

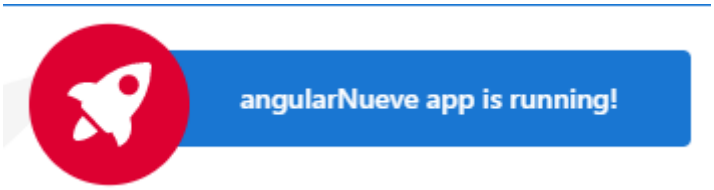
Esas propiedades representan el modelo de datos y se podrán usar expresiones en las vistas para poder visualizarlas.

app.component.html

```
        </div>
      </a>
      <a href="https://github.com/angular/angu
        <svg class="material-icons" xmlns="htt
      </a>
    </footer>

    <svg id="clouds" alt="Gray Clouds Backgrounc
      <path id="Path_39" data-name="Path 39" d="
    </svg>

  </div>

  <!-- * * * * * * * * * * * * * * * * * * * *
  <!-- * * * * * * * * * * The content above *
  <!-- * * * * * * * * * * is only a placeholder
  <!-- * * * * * * * * * * and can be replaced.
  <!-- * * * * * * * * * * * * * * * * * * * *
  <!-- * * * * * * * * * * End of Placeholder *
  <!-- * * * * * * * * * * * * * * * * * * * *



  <router-outlet></router-outlet>
```

No eliminar ***<router-outlet></router-outlet>***

Estas son cosas que te resultarán muy familiares como la interpolación **{{ title }}** que permite mostrar



Sintaxis para las vistas

Piezas declarables de una vista

Comenzaremos por describir las cosas que disponemos para su declaración en una vista:

- **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 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.

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.

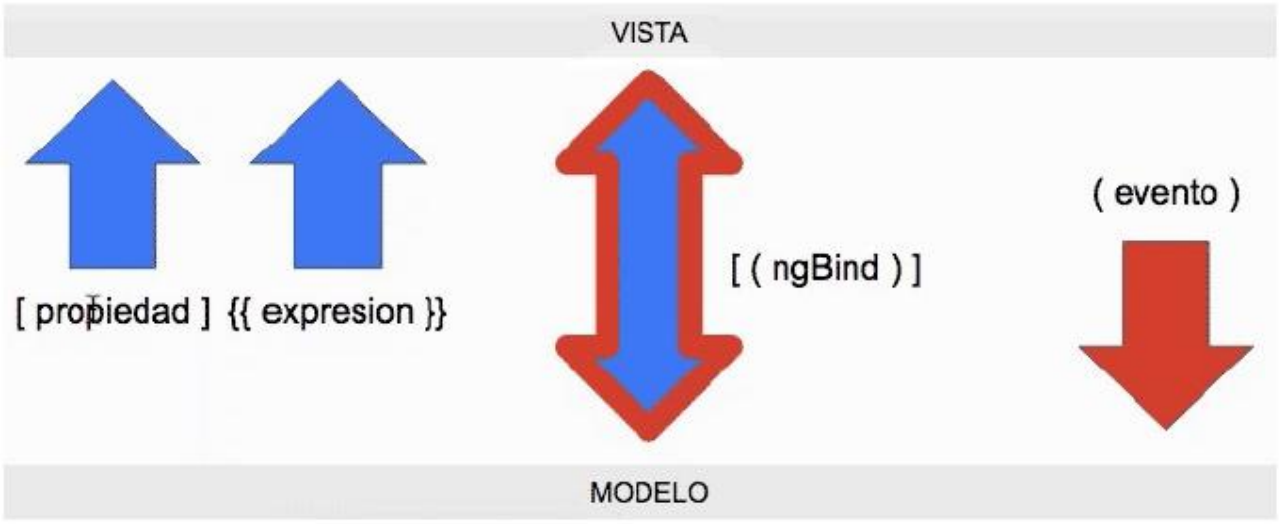
Dicho de otra manera, ahora se entrega al programador el control del flujo de la información, para que éste pueda optimizar el rendimiento de la aplicación.

Flujo de información entre el modelo y la 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)**

En el siguiente ejemplo vemos la utilización de piezas de la vista utilizadas:

```
export class AppComponent {
  title = 'app works!';
  visible = false;
  mostrar(){
    this.visible = true;
  }
}
```

app.component.ts

```
<h1>
  {{title}}
</h1>
<p [hidden]="!visible">
Adiós
</p>
<button (click)="mostrar()">Mostrar contenido oculto</button>
```

app.component.html

En este caso vemos la utilización de la propiedad **[hidden]** del evento **(click)** y de la expresión **{{title}}**

Otros ejemplos:

```
export class AppComponent {
  title = 'app works!';
  visible = false;
  clase="css-class";
  mostrar(){
    this.visible = true;
  }
}
```

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

*En este caso se agregar una clase css con el nombre que tenga la variable **clase** declarado en el modelo, en el ejemplo es css-class.*

Ver ejemplo 1

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

*Al igual que el ejemplo anterior, el atributo href tomara el valor definido en la variable de clase **enlace.***

Componentes

Árbol de componentes

Una aplicación Angular 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.

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.

Componentes vs directivas

En Angular 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 nos indican que un componente es un tipo de directiva.

Existen tres tipos de directivas:

- Componentes: Un componente es una directiva con un témplate. 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.

Decorador de componentes

¿Qué es un decorador?

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.

¿Qué información se agrega a través del decorador?

```
@Component({
    moduleId: module.id,
    selector: 'test-Angular-app',
    templateUrl: 'test-Angular.component.html',
    styleUrls: ['test-Angular.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.

Crear un nuevo componente

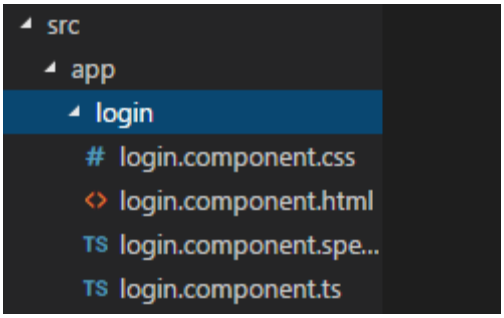
ng generate

Para crear un nuevo componente debemos ejecutar:

```
C:\sites\angular\angularNueve>ng g component login
CREATE src/app/login/login.component.html (20 bytes)
CREATE src/app/login/login.component.spec.ts (621 bytes)
CREATE src/app/login/login.component.ts (271 bytes)
CREATE src/app/login/login.component.css (0 bytes)
UPDATE src/app/app.module.ts (545 bytes)
```

ng generate component *nombre_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.



Usar un componente creado

Crear el HTML

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"

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

  constructor() { }
```

Si queremos utilizar dicho componente debemos agregar en el HTML que queramos la etiqueta **<app-login></app-login>** (Agregar esta etiqueta dentro del html del app.component.ts)

```
<h1>
  Hola {{nombre_persona}}
</h1>
<input [(ngModel)]="nombre_persona"/>

<app-login></app-login>
```

Dado que estamos comenzando con Angular 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.

Bibliografía utilizada y sugerida

https://cli.angular.io/

https://www.typescriptlang.org/

Lo que vimos:

En esta unidad hemos aprendido los conceptos básicos para comenzar a desarrollar nuestras aplicaciones en angular.

Lo que viene:

En la próxima unidad veremos cómo utilizar directivas en angular, validación de formularios y componentes avanzados.