**PREGUNTAS ANGULAR**

**¿Que es Angular?**

Angular es un framework de ingeniería de software de código abierto mantenido por Google, que sirve para desarrollar aplicaciones web de estilo Single Page Application (SPA) y Progressive Web App (PWA). Sirve tanto para versiones móviles como de escritorio.

Angular tiene un nivel de complejidad de medio a elevado y ofrece soluciones robustas, escalables y optimizadas para lograr un estilo de codificación homogéneo y de gran modularidad. Su desarrollo se realiza por medio de TypeScript o JavaScript. En este último se ofrecen diversas herramientas adicionales al lenguaje como tipado estático o decoradores. Su nombre «Angular» proviene del concepto de paréntesis angulares (< >) que se utilizan en HTML.

Para comprender de mejor forma qué es Angular, hemos listado algunas de las características principales del marco de trabajo que te servirán para conocer mejor su funcionalidad y la manera en que puedes utilizarlo en el desarrollo de algún proyecto web próximo.

1. Uso de DOM regular

Un Document Object Model (DOM) es un documento XML o HTML que tiene una estructura de árbol en la que cada nodo representa una parte del documento. Angular hace uso de DOM regular, lo que permite una mejor organización conforme avanza el desarrollo web.

2. Enlace de datos o data binding

El enlace de datos es un proceso con el que los usuarios pueden manipular elementos de una página web a través de un navegador. Entre sus principales ventajas es que no requiere secuencias de comandos ni programaciones complejas, además de que emplea HTML dinámico. También permite una mejora en la visualización de una página web, sobre todo cuando contiene una gran cantidad de datos.

3. Compatibilidad móvil y de escritorio

Como ya lo mencionamos anteriormente, Angular funciona tanto para el desarrollo de aplicaciones móviles como de escritorio. Esto también significa que puede ejecutarse en la mayoría de navegadores web.

4. Velocidad y rendimiento

Angular cuenta con código de generación que permite convertir tus plantillas en códigos altamente optimizados. Esto te ofrece todos los beneficios del código escrito a mano con la productividad de un marco.

De igual manera, allana el camino para los sitios que optimizan para SEO y sus aplicaciones angulares se cargan rápidamente. Esto ofrece una división de código automática para que solo cargues el código que necesitas para representar la vista que requieres.

5. Productividad

Angular permite la creación rápida de vistas de interfaz de usuarios con una sintaxis de plantilla muy sencilla y eficaz. Además, con sus herramientas de líneas de comandos puedes comenzar a construir en menor tiempo y agregar componentes, pruebas e implementaciones al instante.

6. Enlace bidireccional de datos

Angular enlaza JavaScript y HTML, y una de sus principales ventajas es que el código de ambos está sincronizado, lo que ahorra mucho tiempo para los desarrolladores web.

7. Directivas

Los archivos HTML se ven ampliados gracias a directivas habilitadas por los desarrolladores cuando agregan el prefijo ng- a los atributos HTML. Existen diferentes tipos de directivas muy útiles que puedes usar para diferentes acciones como vincular el contenido de un elemento HTML a los datos de aplicación, o especificar que el contenido de un texto debe reemplazarse con una plantilla.

8. Pruebas

Angular hacer uso del framework de prueba llamado Jasmine, el cual proporciona diversas funcionalidades para escribir diferentes tipos de casos de prueba. De igual manera, el marco admite pruebas unitarias y de integración.

Ahora que ya sabes qué es Angular, es importante mencionar que cuenta con una gran serie de módulos habituales para el desarrollo de proyectos web, por lo que no es necesario que empieces desde cero. Esta no es la única ventaja de utilizarlo y te mencionamos a continuación más beneficios.

1. Alta calidad de la aplicación

Si bien Angular es una plataforma compleja y difícil de aprender para algunos desarrolladores web principiantes, esto, a la vez, también representa una enorme ventaja, pues el éxito del producto está prácticamente asegurado. Puedes crear cualquier cosa que se te ocurra y hacerla realidad con las funciones integradas del marco de trabajo.

2. Desarrollo multiplataforma

Angular ofrece soluciones de aplicaciones web progresivas angulares que puedes ejecutar en plataformas móviles, pero el marco puede utilizarse muy bien en aplicaciones móviles nativas.

Anteriormente, los desarrolladores front-end usaban la fórmula Ionic + Angular para el desarrollo multiplataforma, pero hoy en día es más común utilizar Angular + NativeScript. ¿Por qué? NativeScript te brinda acceso a las API nativas para desarrollar aplicaciones que pueden ejecutarse tanto en iOS como en Android.

3. Proceso de desarrollo web más rápido

Angular te permite crear aplicaciones de forma más rápida y eficiente, pues goza de ventajas técnicas como las siguientes:

* Documentación detallada. La documentación de Angular está cuidadosamente escrita y dotada de una gran variedad de ejemplos de código para tener mayor claridad en el proceso y permitir que los desarrolladores encuentren soluciones rápidas a cualquier problema conforme crean una aplicación.
* Interfaz de línea de comandos de Angular. La CLI (por sus siglas en inglés) facilita el trabajo de los desarrolladores al ofrecer un conjunto de herramientas de codificación. Además, esta línea de comandos puede ampliarse con bibliotecas de terceros para resolver problemas de software inusuales o muy complicados.
* Enlace de datos bidireccional. Como ya lo mencionamos, Angular cuenta con esta característica, lo que permite el ahorro de tiempo al automatizar algunos procesos de generación de código.
* Soporte de Google. Cuando definimos qué es Angular también destacamos que lo mantiene Google, por lo que puedes estar seguro de que recibirás soporte en tiempo y forma a cualquier duda o problema que presentes.

4. Aplicaciones web más ligeras

Versiones anteriores de Angular (las cuales te contaremos de ellas más adelante) presentaban problemas en el tamaño de las aplicaciones, lo que no permitía una carga rápida. Actualmente, se han hecho mejoras en los módulos de carga diferida y con el renderizador de Ivy, que permite hacer paquetes más pequeños para acelerar la aplicación.

5. Código legible y comprobable

Podemos resaltar que Angular es un marco de trabajo consistente gracias a sus elementos estructurales como módulos, componentes, directivas, tuberías y servicios; así como la posibilidad de escribir aplicaciones con la arquitectura tradicional MVVM (Model-View-ViewModel) y MVC (Model-View-Controller). Ambas sirven para el mejoramiento de reutilización de código.

Versiones de Angular y usos

Una vez que ya tienes claro qué es Angular, sus características y principales ventajas, te hablaremos un poco de sus versiones y los usos de cada una.

Es importante conocer las versiones de Angular existentes, pues sus números de versión indican el nivel de cambios que introduce cada una. En este sentido, existen tres partes: major, minor y patch. Angular representa lo anterior como, por ejemplo, versión 7.2.11, en donde se indica de forma desglosada que la versión principal es 7, la versión secundaria es 2 y el nivel de parche es 11. Cada número de versión se incrementa según el nivel de cambio incluido. A continuación, te las explicamos con mayor detalle.

1. Major realease

El Major release o lanzamiento importante se compone de características significativas, en las que se espera una mínima asistencia del desarrollador durante la actualización. Dentro de esta actualización es posible que debas ejecutar secuencias de comandos, refactorizar el código, ejecutar pruebas adicionales y aprender nuevas API.

2. Minor release

El lanzamiento menor contiene características más pequeñas que, si bien no requieren tu asistencia como desarrollador, sí puedes modificar de forma opcional tus aplicaciones y bibliotecas para usar nuevas API, funciones y capacidades agregadas a la nueva versión.

3. Patch release

El lanzamiento de parche se trata de una versión de corrección de errores de bajo riesgo. Para este proceso de actualización no se requiere la asistencia del desarrollador.

Si bien Angular fue desarrollado por Google para uso interno en sus sitios web y servicios, hoy en día una gran variedad de sitios web de muchas empresas se construyen bajo este marco. Y no es ninguna sorpresa, pues como ya pudiste comprobarlo, es un framework que te permite obtener soluciones web, móviles y de escritorio de alto de rendimiento.

Angular, al ser un marco de trabajo desarrollado en 2009, ha tenido una serie de actualizaciones importantes a lo largo de los años. Dentro de esta evolución hemos visto pasar Angular 1, conocido como AngularJS, para después contar con las versiones, 2, 3, 4, 5, 6 y 7.

Actualmente, la versión más actualizada y usada en estos momentos es Angular 12, que si bien no contiene muchos cambios, sí promueve una nueva forma de hacer las cosas.

Entre sus principales ventajas e innovaciones se encuentran las siguientes:

* Versión Ivy. En versiones anteriores de Angular, era aún utilizado el View Engine, pero en Angular 12 se reemplazó totalmente por Ivy. Esta es la primera entrada a una evolución completa que deja atrás funciones y soluciones obsoletas de versiones antiguas para los desarrolladores.
* Coalescencia nula. El operador de coalescencia nula permite reducir algunas líneas de código para simplificar la lógica de la plantilla y mejorar su lectura.
* Mejoramiento en internacionalización de mensajes. La versión Angular 12 corrige problemas con la lógica que se usa para generar identificadores de diferentes mensajes dentro de la aplicación.

**Modulos:**

Las aplicaciones Angular son modulares y Angular tiene su propio sistema de modularidad llamado NgModules. Los NgModules son contenedores para un bloque cohesivo de código dedicado a un dominio de aplicación, un flujo de trabajo o un conjunto de capacidades estrechamente relacionadas. Pueden contener componentes, proveedores de servicios y otros archivos de código cuyo alcance está definido por el NgModule que los contiene. Pueden importar la funcionalidad que se exporta desde otros NgModules y exportar la funcionalidad seleccionada para que la utilicen otros NgModules.

Cada aplicación Angular tiene al menos una clase NgModule, el módulo raíz, que se llama convencionalmente AppModule y reside en un archivo llamado app.module.ts.

**Modulo:** Es una clase con el decorator NgModule, organizar las piezas o bloques de nuestra aplicación y poder extender ciertas capacidades de nuestra aplicación, como librerías externas.

**Metadatos de NgModule:**

Un NgModule está definido por una clase decorada con @NgModule(). El decorador @NgModule() es una función que toma un único objeto de metadatos, cuyas propiedades describen el módulo. Las propiedades más importantes son las siguientes.

* declarations: Los componentes, directivas, y pipes que pertenecen a este NgModule. Son Privados por defecto y no se pueden redeclarar.
* exports: El subconjunto de declaraciones que deberían ser visibles y utilizables en las plantillas de componentes de otros NgModules.
* imports: Otros módulos cuyas clases exportadas son necesarias para las plantillas de componentes declaradas en este NgModule.
* providers: Creadores de servicios que este NgModule aporta a la colección global de servicios; se vuelven accesibles en todas las partes de la aplicación. (También puedes especificar proveedores a nivel de componente, que a menudo es preferido).
* bootstrap: La vista principal de la aplicación, llamado el componente raíz, que aloja todas las demás vistas de la aplicación. Sólo el NgModule raíz debe establecer la propiedad bootstrap.

**Decorador:**

Un decorador que aparece inmediatamente antes de una definición de clase, que declara que la clase es del tipo dado y proporciona metadatos adecuados para el tipo.

Los siguientes decoradores pueden declarar tipos de clase Angular:

* @Component()
* @Directive()
* @Pipe()
* @Injectable()
* @NgModule()

**directiva**

Una clase que puede modificar la estructura del DOM o modificar atributos en el DOM y el modelo de datos de componentes. Una definición de clase directiva está precedida inmediatamente por un decorador @Directive() que proporciona metadatos.

Una clase de directiva generalmente está asociada con un elemento o atributo HTML, y ese elemento o atributo a menudo se conoce como la directiva misma. Cuando Angular encuentra una directiva en una plantilla HTML, crea la instancia de clase de directiva coincidente y le da a la instancia control sobre esa parte del DOM del navegador.

Hay tres categorías de directivas:

* Componentes usa @Component() (una extensión de @Directive()) para asociar una plantilla con una clase.
* Directivas de atributo modifica el comportamiento y la apariencia de los elementos de la página.
* Directivas estructurales modifican la estructura del DOM.

Angular proporciona una serie de directivas integradas que comienzan con el prefijo ng. También puedes crear nuevas directivas para implementar tu propia funcionalidad. Asocia un selector (una etiqueta HTML como <my-directive>) con una directiva personalizada, extendiendo así la sintaxis de plantilla que puede usar en tus aplicaciones.

**Componente:**

Una clase con el @Component() decorador que lo asocia con un compañero plantilla. Juntos, la clase de componente y la plantilla definen una vista. Un componente es un tipo especial de directiva. El decorador @Component() extiende el decorador @Directive() con características orientadas a plantillas.

Una clase de componente Angular es responsable de exponer los datos y manejar la mayor parte de la visualización de la vista y la lógica de interacción del usuario a través de enlace de datos.

**directivas de atributos:**

Una categoría de directiva que puede escuchar y modificar el comportamiento de otros elementos HTML, atributos, propiedades y componentes. Generalmente están representados como atributos HTML, de ahí el nombre.

Obtén más información en Directivas de atributos.

**directivas estructurales**

Una categoría de directiva que es responsable de dar forma al diseño HTML modificando el DOM, es decir, agregando, eliminando o manipulando elementos y sus elementos secundarios.

Obtén más información en Directivas estructurales.

**enlace de datos**

Un proceso que permite a las aplicaciones mostrar valores de datos a un usuario y responder al usuario acciones (como clics, toques y pulsaciones de teclas).

En el enlace de datos, declara la relación entre un widget HTML y una fuente de datos y deja que el framework maneje los detalles.

El enlace de datos es una alternativa para insertar manualmente los valores de datos de la aplicación en HTML, adjuntando oyentes de eventos, extrayendo valores modificados de la pantalla y actualizar los valores de los datos de la aplicación.

Lee sobre las siguientes formas de enlace en la Sintaxis de plantilla de Angular:

* **Interpolación:** La interpolación le permite incorporar cadenas calculadas en el texto entre las etiquetas de elementos HTML y dentro de las asignaciones de atributos. Las expresiones de plantilla son lo que usas para calcular esas cadenas.

****

* **Enlace de propiedad:** Utilice el enlace de propiedades para establecer las propiedades de los elementos de destino o los decoradores de la directiva @Input().

****

* **Enlace de evento:** El enlace de eventos le permite escuchar ciertos eventos, como pulsaciones de teclas, movimientos del ratón, clics y toques

****

* **Enlace de atributo:** Establece el valor de un atributo directamente con un enlace de atributo. Esta es la única excepción a la regla de que un enlace establece una propiedad de destino y el único enlace que crea y establece un atributo.

****

* **Enlace de clase:** Aquí se explica cómo configurar el atributo class sin un enlace en HTML simple.

****

* **Enlace de estilo:** Aquí se explica cómo configurar el atributo style sin un enlace en HTML simple.

****

* **Enlace de datos bidireccional con ngModel:** La directiva NgModel le permite mostrar una propiedad de datos y actualizar esa propiedad cuando el usuario realiza cambios

****

**Directivas estructurales integradas**

Las directivas estructurales son responsables del diseño HTML. Forman o remodelan la estructura del DOM, normalmente agregando, eliminando y manipulando los elementos del host a los que están unidos.

Esta sección es una introducción a las directivas estructurales integradas comunes:

* NgIf-cora o destruye condicionalmente subvistas de la plantilla.
* NgFor—repetir un nodo para cada elemento de una lista.
* NgSwitch—un conjunto de directivas que cambian entre puntos de vista alternativos.

**Inmutabilidad:**

La capacidad de alterar el estado de un valor después de su creación. Formularios reactivos realizan cambios inmutables en ese cada cambio en el modelo de datos produce un nuevo modelo de datos en lugar de modificar el existente. Formularios controlados por plantilla realizan cambios mutables con NgModel y enlace de datos bidireccional para modificar el modelo de datos existente en su lugar.

I**nyectable:**

Una clase Angular u otra definición que proporciona una dependencia utilizando el mecanismo de inyección de dependencia. Una clase inyectable de servicio debe estar marcada por el decorador @Injectable(). Otros elementos, como valores constantes, también pueden ser inyectables.

**ECMAScript**

La especificación oficial del lenguaje JavaScript.

No todos los navegadores son compatibles con el último estándar ECMAScript, pero puedes usar un transpiler (como TypeScript) para escribir código utilizando las últimas funciones, que luego se transpilarán al código que se ejecuta en las versiones que son compatibles con los navegadores.

**Ciclo de Vida del Angular:**

Una interfaz que le permite aprovechar el ciclo de vida de [directivas](https://docs.angular.lat/guide/glossary" \l "directive)y [componentes](https://docs.angular.lat/guide/glossary" \l "component)a medida que se crean, actualizan y destruyen.

Cada interfaz tiene un método de enlace único cuyo nombre es el nombre de la interfaz con el prefijo ng. Por ejemplo, la interfaz [OnInit](https://docs.angular.lat/api/core/OnInit) tiene un método de enlace llamado ngOnInit.

Angular llama a estos métodos de enlace en el siguiente orden:

* ngOnChanges: Cuando cambia un valor de enlace [entrada](https://docs.angular.lat/guide/glossary" \l "input)/[salida](https://docs.angular.lat/guide/glossary" \l "output).
* ngOnInit: Después de los primeros ngOnChanges.
* ngDoCheck: Detección de cambios personalizados del desarrollador.
* ngAfterContentInit: Después del contenido del componente inicializado.
* ngAfterContentChecked: Después de cada verificación del contenido del componente.
* ngAfterViewInit: Después de que se inicializan las vistas de un componente.
* ngAfterViewChecked: Después de cada comprobación de las vistas de un componente.
* ngOnDestroy: Justo antes de que se destruya la directiva.

**Funcion Flecha:**

Una expresión de función flecha es una alternativa compacta a una expresión de función tradicional, pero es limitada y no se puede utilizar en todas las situaciones.

Diferencias y limitaciones:

* No tiene sus propios enlaces a this o super y no se debe usar como métodos.
* No tiene argumentos o palabras clave new.target.
* No apta para los métodos call, apply y bind, que generalmente se basan en establecer un ámbito o alcance
* No se puede utilizar como constructor.
* No se puede utilizar yield dentro de su cuerpo.

param => expresión. (param1, paramN) => expresión

**Promesas:**

Las promesas son objetos de JavaScript que sustituyen de alguna manera a los callbacks. Se utilizan cuando no se puede retornar el valor de una función porque aún no se conoce, pero no podemos dejar que bloqueen la función esperando a que llegue. El objeto promesa en un futuro contendrá el valor que esperamos retornar. Es complicado de entender en un principio, pero si conocemos como funciona, en ocasiones nos puede resultar útil.

Hay que indicar que una Promesa no nos sustituye el hecho de utilizar callbacks para leer su valor, pero lo que nos evita es caer en el temido Callback Hell.

Callback Hell:

Esta base le permite a JavaScript crear asincronía, un concepto que se refiere a la habilidad de programar acciones que pueden ser ejecutadas en tiempos no inmediatos. Para ello, se utilizan promesas, condicionales y métodos de tiempo.

**Observables:**

Un observable es, por definición, algo que uno desea observar. El objetivo es ver su condición a lo largo del tiempo y recopilar los cambios que experimentaría.

A diferencia de las promesas, los observables pueden cambiar un número infinito de veces su valor. Un observable puede ser observado y luego liberado, luego observado, mientras que una promesa no puede deshacerse. Los observables son un flujo de datos al que uno puede suscribirse. Todo el proceso está bien dividido en dos partes distintas: el observable y el observador que se suscribe a lo observable.

Podemos observar mil veces un observable, pero una vez mil promesas.

Observable es una clase de TypeScript ubicada en un paquete de la biblioteca rxjs. Esto permite utilizar los asincrónicos y los observables más fácilmente.

miVariable = new Observable(funcion () { } );

La función es una serie de eventos que emitirá el observable. Cada emisión se realiza utilizando el método next().

miObservable = new Observable(obs => {

obs.next(1);

obs.next(2) ;

obs.next(3) ;

});

Aquí, el observable emite el valor 1, luego 2, luego 3.

**Polyfills:**

Polyfills en angular hay unas pocas líneas de código que hacen que su aplicación sea compatible con diferentes navegadores. El código que escribimos está principalmente en ES6 (nuevas características: descripción general y comparación) y no es compatible con IE o firefox y necesita algunas configuraciones de entorno antes de poder ser visto o utilizado en estos navegadores.

Polyfills.tsfue proporcionado por angular para ayudarte a eliminar la necesidad de configurar todo específicamente.

Guard:

Los Guards en Angular, son de alguna manera: middlewares que se ejecutan antes de cargar una ruta y determinan si se puede cargar dicha ruta o no. Existen 4 tipos diferentes de Guards (o combinaciones de estos) que son los siguientes:

* (CanActivate) Antes de cargar los componentes de la ruta.
* (CanLoad) Antes de cargar los recursos (assets) de la ruta.
* (CanDeactivate) Antes de intentar salir de la ruta actual (usualmente utilizado para evitar salir de una ruta, si no se han guardado los datos).
* (CanActivateChild) Antes de cargar las rutas hijas de la ruta actual.

Como middleware, estos componentes se ejecutan de manera intermedia antes de determinadas acciones y si retorna true la ruta seguiría su carga normal, en caso negativo, el Guard retornaría false y la ruta no se cargaría. Generalmente en caso de que no se cumpla la condición del Guard, se suele hacer una redirección a la ruta anterior o a una ruta definida como la interfaz de autenticación.

Una de las ventajas de utilizarlos, es que al no cargar la ruta evitamos que los usuarios vean una interfaz a la que no tienen acceso (aunque sea por unos pocos milisegundos). Por otra parte, con estos componentes es posible estructurar el código de la aplicación de una manera más organizada y donde la lógica de la ruta está en la ruta en sí y la lógica de permisos y acceso a recursos se encuentra aislada en estos componentes.

Interpolacion:

<p>

Nombre: {{ nombre }}

Apellido : {{ apellido }}

Edad: {{ edad + 1 }}

</p>

Template Reference:

<div (keyup)="0">

<label for="">Repetidor:</label>

<input type="text" #entrada id="">

<br>

<label for="">Resultado: {{ entrada.value }}</label>

</div>

Property Binding

HTML:

<button class="btn btn-primary" [disabled]="deshabilitar">AgregarPersona</button>

TS: deshabilitar = true;

Event Binding:

HTML:

<button class="btn btn-primary" (click)="agregarPersona()">Agregar Persona</button>

<p>{{ mensaje }}</p>

TS:

agregarPersona() {

this.mensaje = 'Persona Agregada';

}

Otro Ejemplo…

HTML:

<label>Titulo Persona</label>

<input type="text" class="form-control" (input)='modificarTitulo($event)'>{{ titulo }}

TS:

modificarTitulo(*event*: Event) {

console.log('Entrando a Evento Modificar Titulo');

this.titulo = (<HTMLInputElement>event.target).value;

}

Two-Way Binding:

HTML:

<label>Titulo Persona</label>

<input type="text" class="form-control" [(ngModel)]="titulo">{{ titulo }}

TS:

titulo = 'Ingeniero';

**Directivas:**

Las Directivas extienden la funcionalidad del HTML usando para ello una nueva sintaxis. Con ella podemos usar lógica que será ejecutada en el DOM (Document Object Model).

Cada Directiva que usamos tiene un nombre, y determina donde puede ser usada, sea en un elemento, atributo, componente o clase.

Se dividen en tres tipos diferentes:

* Directivas de Atributo
* Directivas de estructurales
* Componentes

**Directivas de Atributo**

Alteran la apariencia o comportamiento de un elemento del DOM y son usados como atributos de los elementos.

Entre la directivas de atributo, encontramos:

* ngModel: Implementa binding
* ngClass: permite añadir/eliminar varias clases
* ngStyle: permite asignar estilos inline

**Directivas Estructurales**

Alteran la estructura del DOM, agregando, eliminando y manipulando los elementos host a los que están unidos.

Algunas directivas estructurales tienen un asterisco (\*), que precede al nombre del atributo de la directiva.

En las directivas estructurales podemos encontrar las siguientes:

\*ngIf: Nos permite incluir condicionales de lógica en nuestro código, como por ejemplo evaluar sentencias, hacer comparaciones, mostrar u ocultar secciones de código, y entre las muchas condiciones que deseemos crear, para que se renderice nuestro HTML, cumpliendo la sentencia a evaluar. Con el \*ngIf, podemos evaluar sentencias con un simple If, podemos evaluar el else, para que no cumpliéndose la primera condición que se evalúa nuestro código ejecute otra acción en el caso contrario y podemos además incluir el then, para que cumpliendose la condición afirmativa (if), podamos añadir más flexibilidad a nuestro código incluyéndole un camino afirmativo adicional.

\*ngFor: Permite ejecutar bucles, los bucles son los que conocemos en lógica de programación como: for, while, foreach, etc. Con esta directiva estructural podemos evaluar de acuerdo a nuestra condición n veces.

ngSwitch: esta directiva es similar al \*ngIf, y es como el switch en lógica de programación. En esta directiva se pueden crear los diferentes casos que deseamos evaluar y cuando se cumple la condición esperada, oculta/muestra el HTML. Nos permite mantener nuestro código más limpio, si necesitamos evaluar varias sentencias.

ngPlural: es una directiva que permite agregar o remover elementos del DOM, basado en un valor númerico. Para usar esta directiva, se debe proporcionar un elemento contenedor que establezca el atributo [ngPlural] en una expresión de cambio. Los elementos internos con un [ngPluralCase] ​​se mostrarán en función de su expresión. Si [ngPluralCase] ​​se establece en una expresión (que comience con = o ‘>’ o ‘<’ etc.), el elemento se mostrará, si el valor es igual a la expresión.

Para mostrar valores por defecto se puede usar el string “other”.

ngTemplate: esta directiva como su nombre lo indica es un template en Angular. El contenido de esta etiqueta puede reutilizarse en otros templates. Dentro de la etiqueta ng-template tenemos acceso a las mismas variables de contexto que son visibles en el template exterior, como por ejemplo la variable ‘noSuperHeroes’. Esto se debe a que las instancias de ng-template tienen acceso al mismo contexto en el cual están integradas. Además cada template también puede definir su set de variables.

ngComponentOutlet: nos permite crear componentes dinámicos.

**Directivas de Componente**

Las Directivas de Componente son directivas con un Template. Los componentes tienen decoradores “@Component”, el componente es un decorador @Directive que es extendido con características propias de los templates.