# Angular

* Crear proyecto: ng new nombreProyecto

ng g c nombre\_componente : para generar componentes

* ng g m nombre\_componente\_o\_modulo: para generar modulos
* ng g m nombre\_componente **--routing**: para generar el modulo con su propio routing.

## Formularios dependencias

FormsModule ,ReactiveFormsModulo 🡺 sirven para FormGroup ,FormControls y validators

## Routing Dependencias

HttpClientModule (appModule)

Httpclient

# Agregar el AppRoutingModule

En Angular, la mejor práctica es cargar y configurar el enrutador en un módulo de nivel superior separado que está dedicado al enrutamiento e importado por la raíz AppModule.

Por convención, el nombre de la clase del módulo es AppRoutingModule y pertenece a app-routing.module.ts en la carpeta src/app.

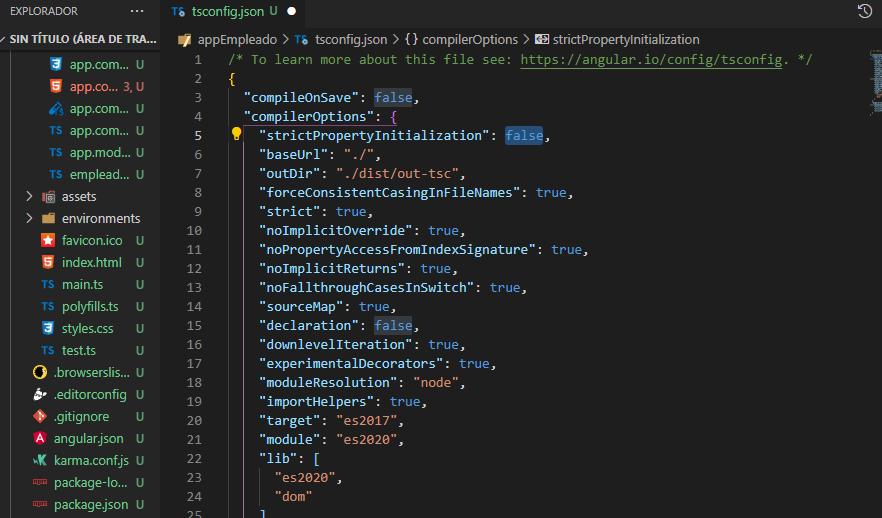
Utiliza el CLI para generarlo.

**ng generate module app-routing --flat --module=app**

--flat coloca el archivo en src/app en lugar de en su propia carpeta.  
--module=app le dice a la CLI que lo registre en el arreglo de importaciones del AppModule.

# Desactivar el tecleado estricto en angular.

* sabe da error para inicializar



Agregar la propiedad   
“strictPropertyInictialization”:false en el archivo de configuración **tsconfig.json**

Eventos

EventEmmiter: $event (debo hacer un casting )

  getValue(event: Event): string {

  return (event.target as HTMLInputElement).value;

Eventos comunes

(focus)="myMethod()"

(submit)="myMethod()"

(scroll)="myMethod()"

(cut)="myMethod()"

(copy)="myMethod()"

(paste)="myMethod()"

(keydown)="myMethod()"

(keyup)="myMethod()"

(mouseenter)="myMethod()"

(mousedown)="myMethod()"

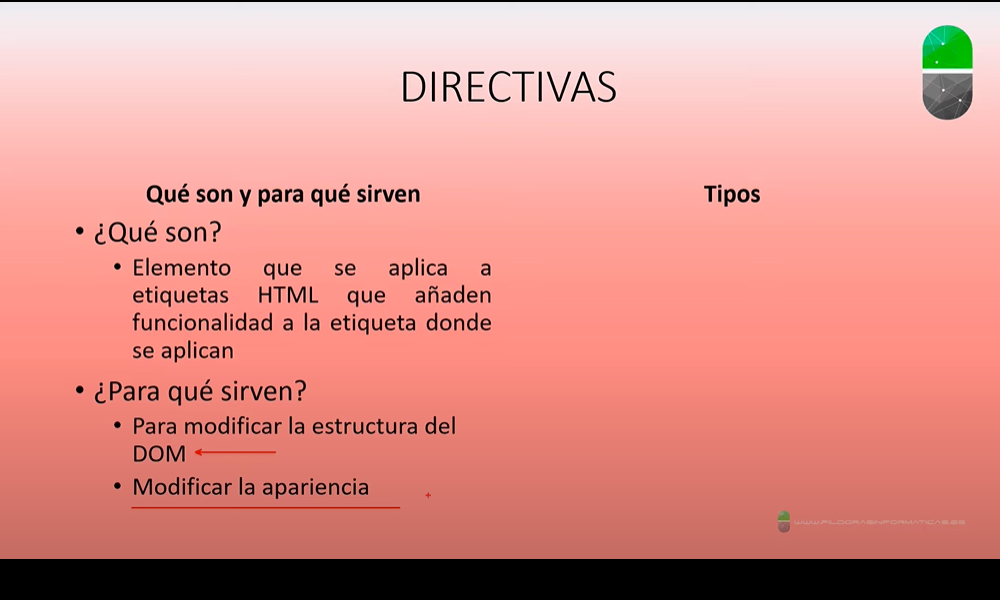
(mouseup)="myMethod()"

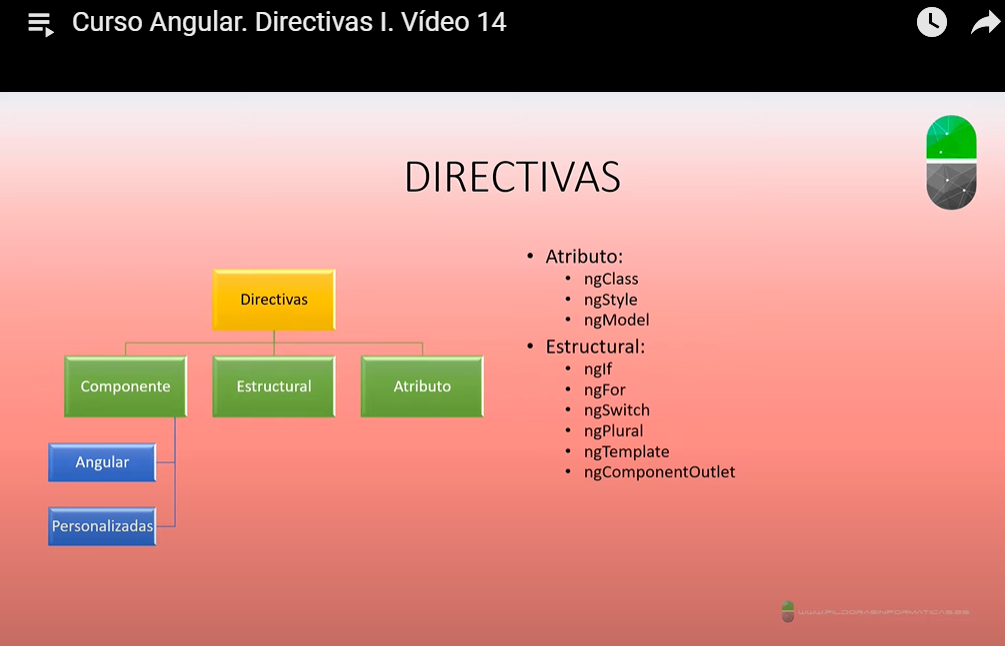
(click)="myMethod()"

(dblclick)="myMethod()"

(drop)="myMethod()"

# Directivas





* \*ngIf
* \*ngFor

# Instalar bootstrap Jquery y popper.js

* npm I bootstrap jquery popper.js 🡪
* **Luego actualizar popper**

popper esta en la nueva version **@popperjs/core**

Luego ir 🡪 angular.json y agregar estas lineas

  "styles": [

              "src/styles.css",

              "node\_modules/bootstrap/dist/css/bootstrap.min.css"

            ],

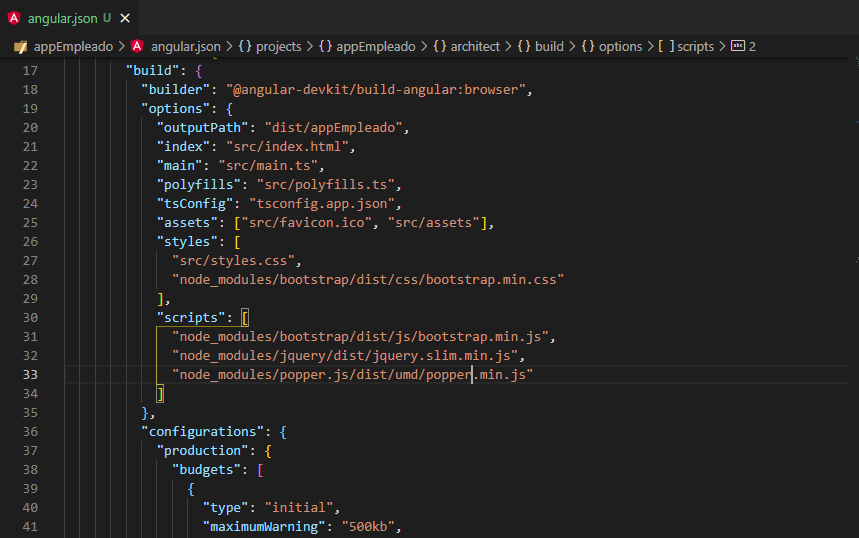
            "scripts": [

              "node\_modules/bootstrap/dist/js/bootstrap.min.js",

              "node\_modules/jquery/dist/jquery.slim.min.js",

              "node\_modules/popper.js/dist/umd/popper.min.js"

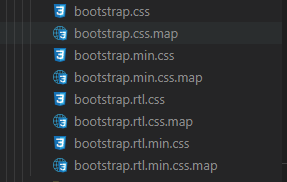
            ]

Ver imágenes extras

|  |
| --- |
| La de jquery |

La de boostrap

|  |
| --- |
| La de Popper |



menu intelligence: Ctrl+barraEspaciadora,

**[ngStyle]**

La condición style se mustra en rojo si el cargo es ==director

<p \*ngIf="bandera; else sinRegistrar">

  El usuario {{nombre}} {{apellido }} que tiene el cargo de

  <span [ngStyle]="{ color: cargo == 'director' ? 'red' : 'black' }">{{

    cargo

  }}</span>

  se ha Registrado con exito 🚀`

</p

Para poder dar mas configuración a cada clase hacemos asi

1. vamos a app.component.css
2. modificamos el archivo

.paraDirectores {

  /\* que sea rojo , negrita y subrayado \*/

  color: red;

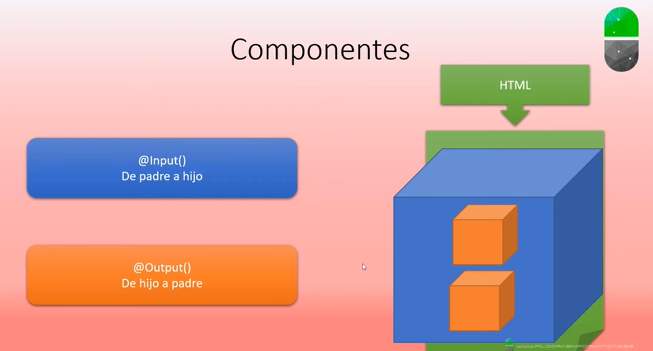
  font-weight: bold;

  text-decoration: underline;

}

1. en el app.component.html
2. <p \*ngIf="bandera; else sinRegistrar">
3. {{ mensaje }} que tiene el cargo de
4. <span [ngClass]="{ paraDirectores: cargo == 'directores' }" >{ cargo }}
5. </span>
6. se ha Registrado con exito 🚀`
7. </p>

# Comunicación entre componentes



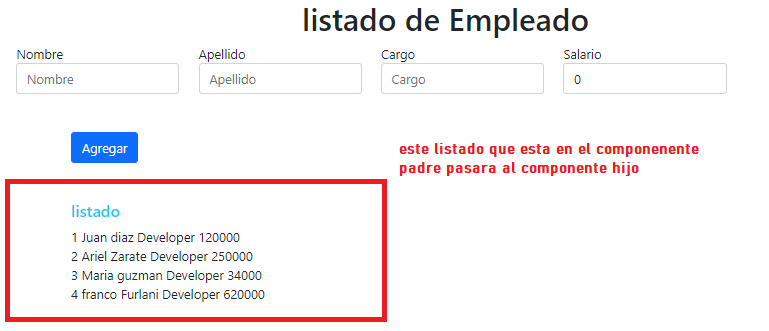
## COMUNICACIÓN DE PADRE A HIJO

## @Input

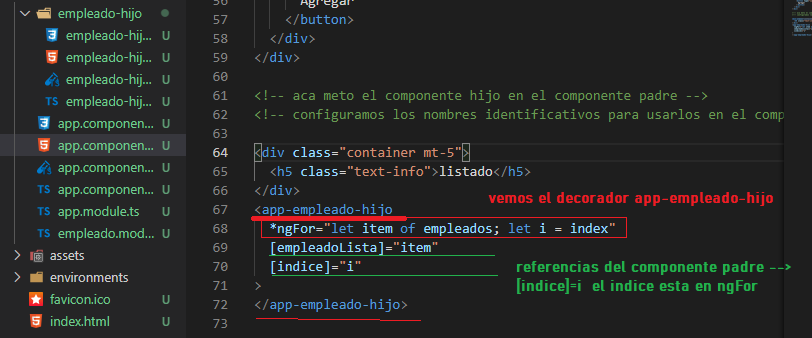
Vamos a modificar el componenente padre de un archivo HTML para que cierta parte del mismo pertenezca al hijo

Para la vista será igual pero por dentro se estará pasando información del componente padre al hijo a través de @input

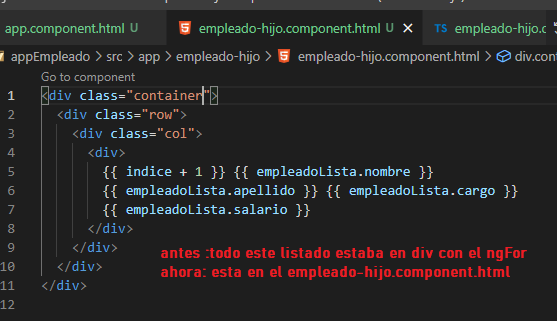
DEBEMOS TENER CREADO UN COMPONENENTE HIJO: **ng g c componenteHijo**



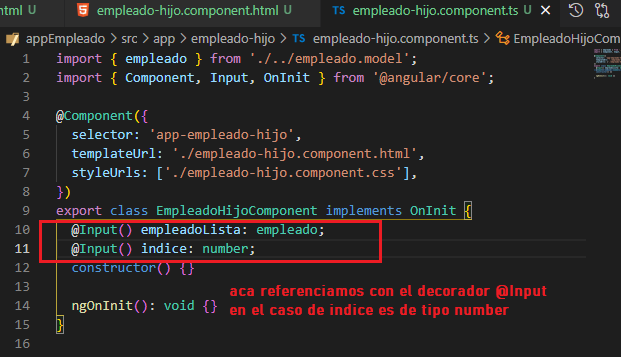
**App.component.html**



**Componente hijo html**

****

**componente Hijo del ts**

**+-**

## COMUNICACIÓN DE HIJO AL PADRE

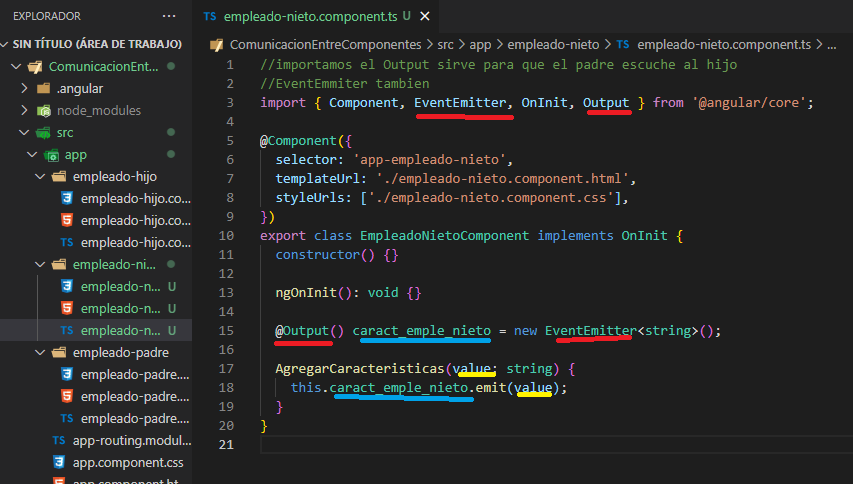
Para usar , debe configurar el padre y el hijo.@[Output](https://angular.io/api/core/Output)()

1. Importar [Output](https://angular.io/api/core/Output)y [EventEmitter](https://angular.io/api/core/EventEmitter)en la clase de componente secundario:



1. En la clase de componentes, decora una propiedad con . El siguiente ejemplo tiene un tipo de , lo que significa que es un evento.@[Output](https://angular.io/api/core/Output)()newItemEvent @[Output](https://angular.io/api/core/Output)()[EventEmitter](https://angular.io/api/core/EventEmitter)



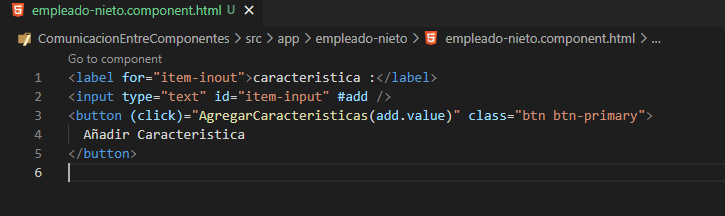


## Configurando la plantilla del niño

La plantilla del niño tiene dos controles. El primero es un HTML <input>con una variable de [referencia de plantilla](https://angular.io/guide/template-reference-variables)#newItem , donde el usuario escribe el nombre de un elemento. La valuepropiedad de la #newItemvariable almacena lo que el usuario escribe en el archivo <input>.

El (click)evento está vinculado al addNewItem()método en la clase del componente secundario. El addNewItem()método toma como argumento el valor de la #newItem.valuepropiedad.

Ejemplo de mi código

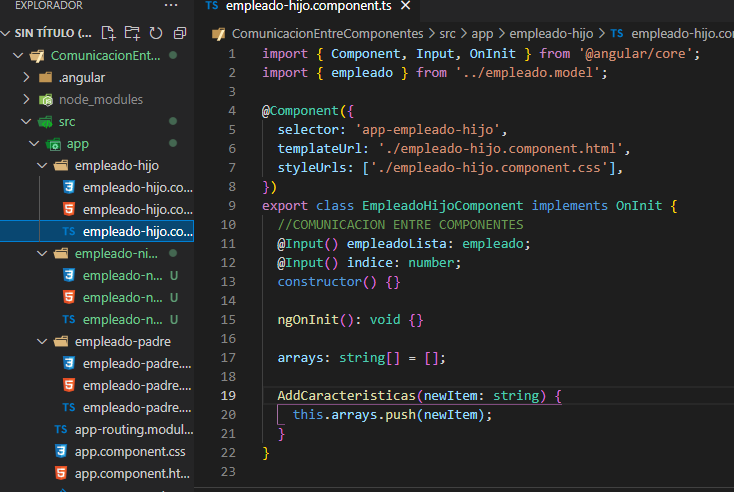


## Configuración del componente principal

En AppComponenteste ejemplo, se presenta una lista de elementos ítems en una matriz y un método para agregar más elementos a la matriz.

El addItem()método toma un argumento en forma de cadena y luego agrega esa cadena a la ítems matriz.

Ejemplo de mi código



## Configuración de la plantilla de los padres

En la plantilla del padre, vincule el método del padre al evento del hijo.

Coloque el selector secundario, aquí <app-item-output>, dentro de la plantilla del componente principal, app.component.html.

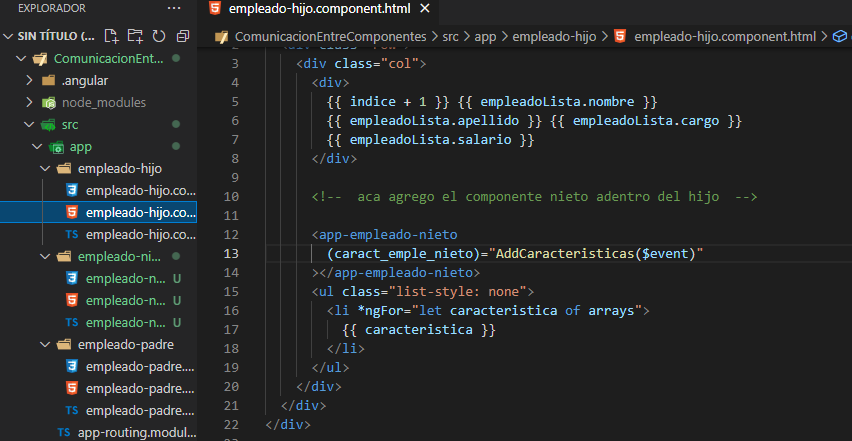
El enlace de eventos (newItemEvent)='addItem($event)', conecta el evento en el elemento secundario, newItemEventcon el método en el elemento principal, addItem().

contiene los $eventdatos que el usuario escribe en la <input>interfaz de usuario de la plantilla secundaria.

Para ver el funcionamiento, agregue lo siguiente a la plantilla de los padres:@[Output](https://angular.io/api/core/Output)()

Itera sobre los elementos de la matriz . Cuando ingresa un valor en el elemento secundario y hace clic en el botón, el elemento secundario emite el evento y el método del elemento principal empuja el valor a la matriz y el nuevo elemento aparece en la lista.\*[ngFor](https://angular.io/api/common/NgForOf) items<input>addItem()items

Ejemplo de mi código



# Servicios

|  |  |
| --- | --- |
| C:\Users\ariel\Pictures\CursoAngularPildoras Informaticas\servicios1.png | C:\Users\ariel\Pictures\CursoAngularPildoras Informaticas\datos servicio inyeccion de dependecia.png |

## Inyección de dependencia en Angular

Las dependencias son servicios u objetos que una clase necesita para realizar su función. La inyección de dependencia, o DI, es un patrón de diseño en el que una clase solicita dependencias de fuentes externas en lugar de crearlas.

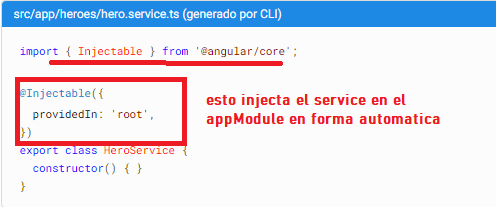
El marco DI de Angular proporciona dependencias a una clase en la creación de instancias. Utilice Angular DI para aumentar la flexibilidad y modularidad en sus aplicaciones.

## Creación de un servicio inyectable

Para generar una nueva HeroService clase en la src/app/héroes

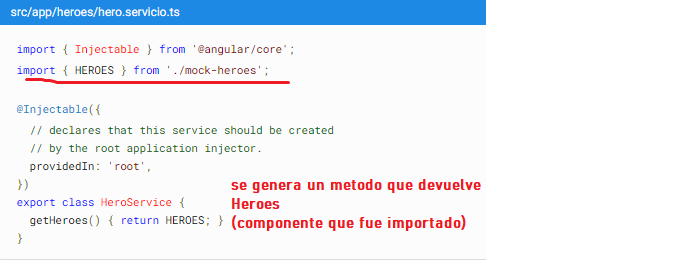
carpeta, use el siguiente comando de [CLI angular .](https://angular.io/cli)

ng generate service heroes/hero



El decorador especifica que Angular puede usar esta clase en el sistema DI. Los metadatos, significa que el es visible en toda la aplicación.@[Injectable](https://angular.io/api/core/Injectable)()providedIn: 'root'HeroService

A continuación, para obtener los datos simulados del héroe, agregue un getHeroes()método que devuelva los héroes de mock.heroes.ts.



Para mayor claridad y facilidad de mantenimiento, se recomienda que defina componentes y servicios en archivos separados.

Si combina un componente y un servicio en el mismo archivo, es importante definir primero el servicio y luego el componente. Si define el componente antes que el servicio, Angular devuelve un error de referencia nula en tiempo de ejecución.

## Uso de servicios en otros servicios

Cuando un servicio depende de otro servicio, siga el mismo patrón que inyectando en un componente. En el siguiente ejemplo HeroServicedepende de un Logger servicio para reportar sus actividades.

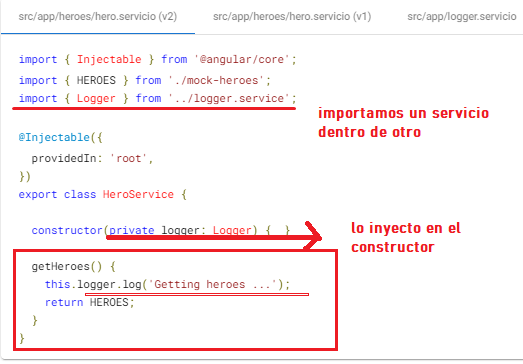
Primero, importe el Loggerservicio. A continuación, inyecte el Loggerservicio en HeroService constructor()especificándolo private logger: Logger entre paréntesis.

Cuando crea una clase que constructor()tiene parámetros, especifique el tipo y los metadatos sobre esos parámetros para que Angular pueda inyectar el servicio correcto.

Aquí, constructor()especifica un tipo de Loggery almacena la instancia de Logger en un campo privado llamado logger.

Las siguientes pestañas de código presentan el Loggerservicio y dos versiones de HeroService. La primera versión de HeroServiceno depende del Loggerservicio. La segunda versión revisada depende del Logger servicio.

# Routing



## Servicios de inyección

Inyectar servicios da como resultado hacerlos visibles para un componente.

Para inyectar una dependencia en un componente constructor(), proporcione un argumento de constructor con el tipo de dependencia. El siguiente ejemplo especifica el HeroServiceen el HeroListComponentconstructor. El tipo de heroServicees HeroService.

