

# Semana 3

## Curso de Angular



### **Semana 3** **Componentes anidados.** **Servicios.**

Desarrollo de aplicaciones web con Angular  
Cefire 2017/2018  
Autor: Arturo Bernal Mayordomo

## Index

Componentes anidados.....	3
Anidando ProductItemComponent.....	3
Anidando StartRateComponent.....	5
Comunicación entre componentes anidados.....	7
Paso de datos a un componente hijo → @Input.....	7
Paso de datos a un componente padre → @Output.....	8
Servicios. Inyección de dependencias.....	11

# Componentes anidados

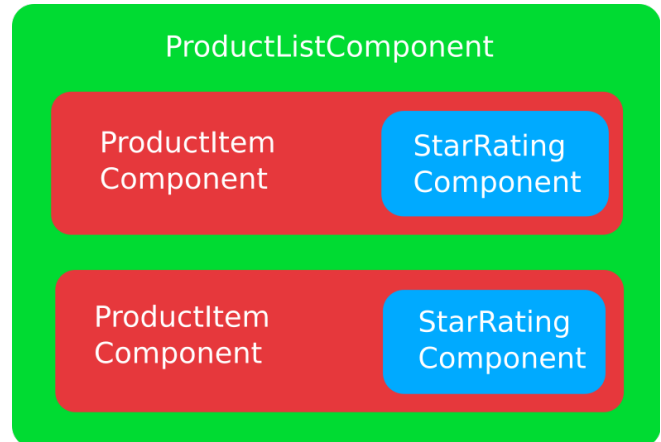
Un componente anidado, o componente hijo, es aquel que representa un fragmento de la vista global. Por ejemplo, cada producto dentro de la lista, podría separarse en un nuevo componente. En este caso, le vamos a llamar **product-item**:

**ng g component product-item**

Además, vamos a crear otro componente que servirá para puntuar un producto y se situará dentro del mismo. Este componente se llamará **star-rating**.

Por ahora, todos los componentes los estamos creando en la raíz (**src/app**):

**ng g component star-rating**



## Anidando ProductItemComponent

Vamos a implementar el componente **product-item** que representa un producto de la lista. Por ahora, se usarán datos fijos del producto hasta que sepamos como obtener dicha información del componente padre (**product-list**).

```
import { IProduct } from '../interfaces/i-product';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'product-item',
  templateUrl: './product-item.component.html',
  styleUrls: ['./product-item.component.css']
})
export class ProductItemComponent implements OnInit {
  product: IProduct = {
    id: 1,
    description: 'SSD 250GB hard drive',
    available: new Date('2016-10-03'),
    price: 75,
    imageUrl: 'assets/ssd.jpg',
    rating: 5
  };

  showImage = true;

  constructor() {}

  ngOnInit() {}
}
```

Como puedes ver, almacenamos en el componente la información necesaria para mostrar un producto. Ahora vamos a implementar la plantilla asociada (**product-item.component.html**).

```
<tr>
  <td>
    <img [src]="product.imageUrl" *ngIf="showImage" alt=""
```

```

        [title]="product.desc | uppercase">
    </td>
    <td>{{ product.desc }}</td>
    <td>{{ product.price | currency:'EUR':'symbol' }}</td>
    <td>{{ product.avail | date:'dd/MM/y' }}</td>
</tr>

```

Simplemente, hemos copiado el trozo de HTML correspondiente a la fila de un producto en la tabla de productos. Ahora sólo queda sustituir dicha fila en el componente padre por el selector del componente actual (**product-list.component.html**):



```

<tbody>
  <product-item *ngFor="let product of products | productFilter:filterSearch">
  </product-item>
</tbody>

```

**Mueve** también los **estilos CSS** al nuevo componente (los que le afecten, claro).

Hide images

	Product	Price	Available
	SSD hard drive	€75.00	03/10/2016
	SSD hard drive	€75.00	03/10/2016

Como puedes observar, a nivel de estructura, hay un problema con las filas de la tabla. Esto ocurre porque el elemento **<product-item>** está situado entre la tabla y cada fila y el navegador no detecta que dichas filas pertenecen a la tabla.

```

▼ <table _ngcontent-spw-2 class="table table-striped">
  ► <thead _ngcontent-spw-2>...</thead>
  ▼ <tbody _ngcontent-spw-2>
    <!--template bindings={
      "ng-reflect-ng-for-of": "[object Object],[object
    }-->
    ▼ <product-item _ngcontent-spw-2 _ngghost-spw-3> ==
      ▼ <tr _ngcontent-spw-3>
        ► <td _ngcontent-spw-3>...</td>
        <td _ngcontent-spw-3>SSD hard drive</td>
        <td _ngcontent-spw-3>€75.00</td>
        <td _ngcontent-spw-3>03/10/2016</td>
      </tr>
    </product-item>

```

Esto se soluciona con CSS. En lugar de usar una tabla con sus correspondientes etiquetas podemos usar elementos con propiedades CSS que indiquen que se comporten como fila, columna, etc. En este caso Bootstrap nos proporciona unas clases (**row** y **col**) para crear una estructura similar a una tabla:

```

#product-list.component.html

<div *ngIf="products && products.length">
  <div class="row headers no-gutters">
    <div class="col-2">
      <button class="btn btn-sm"

```

```

[ngClass]="{'btn-danger': showImage, 'btn-primary': !showImage}"
(click)="toggleImage()">
  {{showImage?'Hide':'Show'}} images
</button></div>
<div class="col-4">{{headers.desc}}</div>
<div class="col">{{headers.price}}</div>
<div class="col">{{headers.avail}}</div>
</div>
<product-item class="row no-gutters"
  *ngFor="let product of products | productFilter:filterSearch">
</product-item>
</div>

```

#### #product-list.component.css

```

.headers {
  font-weight: bold;
  border-bottom: 1px solid #CCC;
  border-top: 1px solid #CCC;
  padding: 5px;
  margin-bottom: 5px;
}

```

```

product-item:nth-child(odd) {
  background: #EEE;
}

```

#### #product-item.component.html

```

<div class="col-2 pl-2">
  <img [src]="product.imageUrl" *ngIf="showImage" alt=""
    [title]="product.desc | uppercase">
</div>
<div class="col-4">{{ product.desc }}</div>
<div class="col">{{ product.price | currency:'EUR':'symbol' }}</div>
<div class="col">{{ product.avail | date:'dd/MM/y' }}</div>

```

#### #product-item.component.css



```

div:first-child img {
  height: 40px;
}

div:not(:first-child) {
  display: flex;
  justify-content: center;
  flex-direction: column;
}

```

Mucho mejor ahora!

Ocultar imágenes	Producto	Precio	Disponible
	Disco duro SSD 240GB	€75.00	03/10/2016
	Disco duro SSD 240GB	€75.00	03/10/2016

## Anidando StartRateComponent

El siguiente paso es implementar el componente **star-rate**, que representará un sistema de puntuación (del 1 al 5) basado en estrellas. Para mostrar las estrellas podríamos recurrir a los caracteres unicode correspondientes, o instalar una fuente de iconos como [Font Awesome](#):

## npm install font-awesome

E importarla en **styles.css** (añadir ../ al principio) o en **angular.json**:

```
"node_modules/font-awesome/css/font-awesome.css"
```

Vamos a ver como quedaría el componente. En principio sólo necesitamos un dato. La puntuación actual del 1 al 5 (**rating**):

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'star-rating',
  templateUrl: './star-rating.component.html',
  styleUrls: ['./star-rating.component.css']
})
export class StarRatingComponent implements OnInit {
  rating: number = 4;
  ...
}
```

En la plantilla, vamos a generar 5 estrellas (iconos). Para ello, vamos a usar una directiva **\*ngFor** con un array fijo del 1 al 5. La diferencia es la clase de cada elemento generado (**fa-star** → estrella rellena, **fa-star-o** → estrella vacía), que asignaremos aprovechando la directiva **ngClass**.

#star-rating.component.html



```
<div class="star-container">
  <span *ngFor="let star of [1,2,3,4,5]" class="fa"
    [ngClass]="{'fa-star': star <= rating, 'fa-star-o': star > rating}"></span>
</div>
```

En el componente padre (product-item), añadimos una nueva columna al final con el componente de puntuación que acabamos de crear:

#product-item.component.html

```
...
<div class="col"><ap-star-rating></ap-star-rating></div>
```

Y este es el resultado (ten en cuenta que por ahora todos los datos son fijos). En breve veremos como se pasa información entre componentes.

Ocultar imágenes	Producto	Precio	Disponible	Puntuación
	Disco duro SSD 240GB	€75.00	03/10/2016	★★★★☆
	Disco duro SSD 240GB	€75.00	03/10/2016	★★★★☆

# Comunicación entre componentes anidados

En la sección anterior vimos como anidar componentes. Sin embargo, todavía no hemos alcanzado la funcionalidad necesaria como mostrar los datos de cada producto, ocultar/mostrar imágenes, mostrar la puntuación, etc., ya que no sabemos como se comunican los componentes todavía.

## Paso de datos a un componente hijo → @Input

Para indicar que un componente recibe datos de entrada por parte del componente padre, creamos una propiedad y la precedemos con el decorador **@input()**. Esto le dice a Angular, que el valor de la propiedad será obtenido a partir de un atributo con el mismo nombre, en el selector HTML del componente actual.

En este caso, como ejemplo, vamos a pasar al componente **product-item**, los datos del producto a mostrar y el booleano que indica si la imagen debe mostrarse:

#product-list.component.html

```
<product-item class="row no-gutters"
  [product]="product" [showImage]="showImage"
  *ngFor="let product of products | productFilter:filterSearch">
</product-item>
```

Como se puede observar, tenemos un atributo llamado **product**, cuyo valor es el objeto del producto a mostrar. También hay otro atributo llamado **showImage**, que recibe el booleano que indica si las imágenes se muestran o no. Como vimos al principio del curso, los corchetes se usan para vincular el valor a una propiedad/método del componente.

Ahora vinculamos estos atributos en el componente hijo con el decorador **@input()**:

```
import { Component, Input, OnInit } from '@angular/core';
import { IProduct } from './iproduct';



@Component({
  selector: 'product-item',
  templateUrl: './product-item.component.html',
  styleUrls: ['./product-item.component.css']
})
export class ProductItemComponent implements OnInit {

  @Input() product: IProduct;
  @Input() showImage: boolean;

  constructor() { }

  ngOnInit() { }
}
```

¡Ahora muestra los productos correctamente!

Ocultar imágenes	Producto	Precio	Disponible	Puntuación
	Disco duro SSD 240GB	€75.00	03/10/2016	★★★★☆
	Placa base LGA1151	€96.95	15/09/2016	★★★★☆

Para terminar, vamos a hacer lo mismo con el componente **star-rating**, enviando la valoración a mostrar desde el componente padre:

#product-item.component.html



```
<div class="col">
  <star-rating [rating]="product.rating"></star-rating>
</div>
```

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

@Component({
  selector: 'star-rating',
  templateUrl: './star-rating.component.html',
  styleUrls: ['./star-rating.component.css']
})
export class StarRatingComponent implements OnInit {

  @Input() rating: number;
  ...
}
```

¡Y ya tenemos los productos con su valoración correcta!

Ocultar imágenes	Producto	Precio	Disponible	Puntuación
	Disco duro SSD 240GB	€75.00	03/10/2016	★★★★★
	Placa base LGA1151	€96.95	15/09/2016	★★★★☆

## Paso de datos a un componente padre → @Output

Vamos a ver ahora la situación inversa. Cuando el componente hijo necesita informar al padre de algún cambio (por ejemplo, que ha sido borrado para eliminarlo del array). Para esto tenemos el decorador **@Output()**.

Antes de nada vamos a implementar la funcionalidad de que, cuando situemos el puntero encima de una estrella, cambiar la puntuación del producto (a esa estrella). Como no debemos modificar el valor al que referencian las propiedades de entrada (**@Input**), ya que las devincularíamos del padre y dejarían de actualizarse automáticamente, crearemos una propiedad auxiliar a la que llamaremos **auxRating** y que inicializaremos al mismo valor que recibimos del componente padre.

También, para cuando quitemos el cursor del ratón del componente, restableceremos el valor de **auxRating** a la puntuación original:

```
export class StarRatingComponent implements OnInit {
  private auxRating: number;
  @Input() rating: number;

  constructor() { }

  restoreRating() {
    this.auxRating = this.rating;
  }
}
```



```

    }

    ngOnInit() {
        this.restoreRating();
    }
}

```

En la plantilla, cuando el cursor del ratón esté encima de una estrella (evento **mouseenter**), cambiaremos el valor de **auxRating** (es el usado para mostrar la puntuación ahora) a la posición de la estrella actual. Cuando el ratón salga del componente (evento **mouseleave**), restableceremos el valor de **auxRating**.

```

<div class="star-container" (mouseleave)="restoreRating()">
  <span *ngFor="let star of [1,2,3,4,5]" class="fa"
    [ngClass]="{'fa-star': star <= auxRating, 'fa-star-o': star > auxRating}"
    (mouseenter)="auxRating = star"></span>
</div>

```

Funciona, pero aún falta que cuando hagamos clic sobre una estrella, establezcamos una nueva puntuación. Modificar la variable de entrada (**rating**) desvinculará dicha variable del **rating**, además de no modificar su valor en el objeto producto que contiene el padre (el valor se pasa por copia).

```

<div class="star-container" (mouseleave)="restoreRating()">
  <span *ngFor="let star of [1,2,3,4,5]" class="fa"
    [ngClass]="{'fa-star': star <= auxRating, 'fa-star-o': star > auxRating}"
    (mouseenter)="auxRating = star" (click)="setRating()"></span>
</div>

```

Lo que hará el método **setRating()** es emitir un evento al componente padre con la nueva puntuación actual (establecida en **auxRating**). Para ello, creamos un “emisor de eventos” → **EventEmitter<number>** con el decorador **@Output()**. El evento se llamará **ratingChanged**.

```

import { Component, Input, Output, EventEmitter, OnInit } from '@angular/core';

@Component({
  selector: 'star-rating',
  templateUrl: './star-rating.component.html',
  styleUrls: ['./star-rating.component.css']
})
export class StarRatingComponent implements OnInit {

  ...
  @Output() ratingChanged = new EventEmitter<number>();
  ...

  setRating() {
    this.ratingChanged.emit(this.auxRating);
  }
  ...
}

```

En el componente padre (**product-item**) escuchará cuando se produzca este evento (**ratingChanged**). Accedemos al valor emitido (la nueva puntuación) con la variable especial **\$event**. Cabe destacar que en este caso, como modificamos la propiedad de un **objeto** no hace falta que lo haga el componente padre (**product-list**) que contiene el array de objetos, ya que los objetos **siempre** se pasan por referencia y no por valor (copia) como sí ocurre con la puntuación (número).

```
<div class="col">
  <ap-star-rating [rating]="product.rating"
    (ratingChanged)="changeRating($event)"></ap-star-rating>
</div>
```

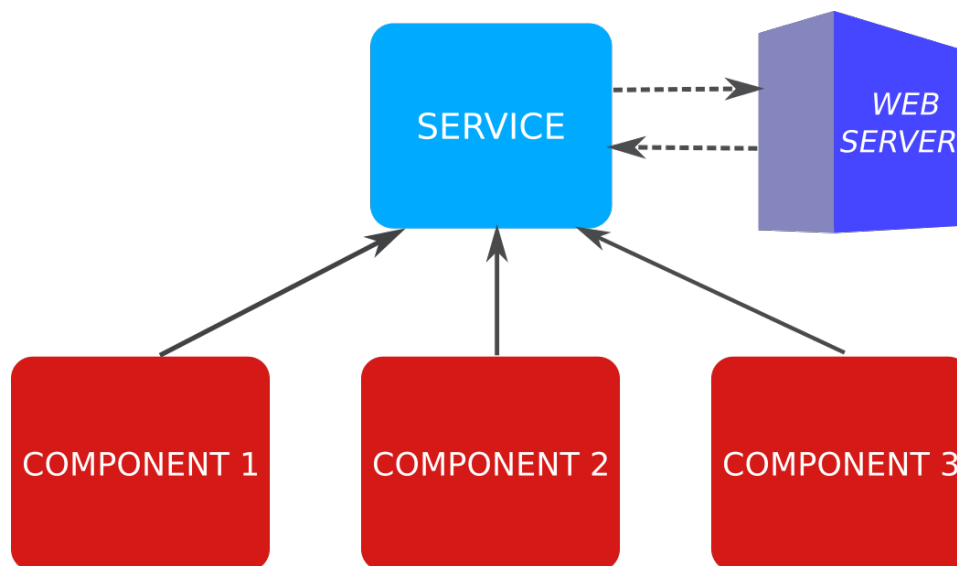
```
export class ProductItemComponent implements OnInit {
  ...
  changeRating(rating: number) {
    this.product.rating = rating;
  }
  ...
}
```

Se recomienda instalar la extensión de Chrome [Augury](#) para ayudar en la depuración de aplicaciones Angular. Con dicha extensión podremos observar, entre otras cosas, como cambian las propiedades de los componentes de valor.

## Servicios. Inyección de dependencias.

Un **Servicio** es una clase cuyo propósito es mantener una lógica (y datos) compartidos entre diferentes componentes de la aplicación. Esto es útil tanto para agrupar funcionalidad común de varios componentes, como para compartir datos entre componentes que no tengan relación de parentesco.

También se recomienda su uso para acceder a datos externos (servicios web). Cuando un componente de Angular (o filtro, o directiva, u otro servicio, etc.) necesita usar un servicio, existe un componente interno llamado **inyector de dependencias** (común en muchos frameworks), que nos proveerá el objeto de dicho servicio. Sólo se creará como máximo una instancia de dicho servicio para la aplicación (Singleton).



En nuestra aplicación de ejemplo, vamos a usar un Servicio para almacenar los productos (en el futuro los obtendrá de un servidor web). Crea un directorio llamado **services** (dentro de **src/app**) y ejecuta lo siguiente:

**ng g service products**

Esto creará un archivo llamado **products.service.ts** con la clase del servicio. Esta clase está precedida del decorador **@Injectable()**, para indicar al inyector de dependencias de Angular, que debe proveer un objeto de esta clase cuando cualquier componente lo requiera. Para usar un servicio debemos registrarlo en el módulo de la aplicación (array **providers**):

```
...
import { ProductService } from '../shared/product.service';
...

@NgModule({
  ...
  providers: [ProductsService],
  ...
})
export class AppModule { }
```

Ahora creamos un método en el servicio que simplemente devuelva el array de productos (en el futuro los obtendremos de un servidor web):

```
import { Injectable } from '@angular/core';
import { IProduct } from '../product-item/iproduct';

@Injectable()
export class ProductsService {

  constructor() { }

  getProducts(): IProduct[] {
    return [{
      id: 1,
      desc: 'SSD hard drive',
      avail: new Date('2016-10-03'),
      price: 75,
      imageUrl: 'assets/ssd.jpg',
      rating: 5
    }, {
      id: 2,
      desc: 'LGA1151 Motherboard',
      avail: new Date('2016-09-15'),
      price: 96.95,
      imageUrl: 'assets/motherboard.jpg',
      rating: 4
    }
  ];
}
```

Ahora que los productos están en el servicio, haremos que el componente **product-list** los obtenga de ahí. Para “inyectar” el servicio en el componente, Angular utiliza una característica de TypeScript. Si declaramos un parámetro con el modificador **public** o **private** en el constructor, TypeScript declara un atributo en la clase con el mismo nombre y hace una asignación del parámetro automática.

Simplemente indicando el tipo de parámetro (ProductService), al ser un servicio, Angular lo pasa automáticamente (el objeto del servicio) como parámetro al llamar al constructor. Esto es la inyección de dependencias.

Finalmente, en el método `ngOnInit`, llamamos al método del servicio que nos devuelve los productos. El array debe inicializarse como array vacío, ya que si no, podría fallar el `*ngFor` (el atributo estaría `undefined`) mientras obtenemos los datos.

```
import { Component, OnInit } from '@angular/core';
import { IProduct } from '../product-item/iproduct';
import { ProductService } from '../shared/product.service';

...
export class ProductListComponent implements OnInit {
  ...
  products: IProduct[] = [];

  ...
  constructor(private productService: ProductService) { }
  ...

  ngOnInit() {
    this.products = this.productService.getProducts();
  }
}
```