Tema 7 – Reusabilidad de componentes

- Características de un componente reutilizable
- Comunicación mediante propiedades Input y Output
- Componentes plantilla
- Prácticas:
  - Componente “Favorito”
  - Componente “CabeceraEstandar”
- Preparar una librería de componentes

Como en cualquier entorno de desarrollo, es importante poder reutilizar el código entre aplicaciones y en la misma aplicación. Esto en Angular se realiza creando **módulos y componentes reusables.**

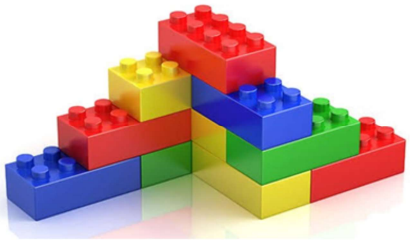
En este tema vamos a ver cómo Angular nos facilita la creación de componentes reutilizables.

Veremos qué características debe tener un componente reutilizable, cómo podemos comunicarnos con un componente mediante propiedades de entrada y salida, y veremos como diseñar componentes que actúen como plantillas para contenido HTML.

Construiremos 2 componentes diseñados para ser reutilizados, y los usaremos en la aplicación de Datos Deportivos.

Finalmente veremos como preparar una librería propia con los componentes desarrollados, de forma que podamos usarlos de forma independiente en nuestras aplicaciones.

Qué es un componente reutilizable



Un componente que realiza una funcionalidad genérica, concreta y limitada, que puede ser utilizado sin modificación en múltiples aplicaciones a través de una API.

Características de un componente reutilizable

Especializado

- Debe tener una responsabilidad concreta y limitada...
- ...suficientemente genérica para su uso en múltiples escenarios

Configurable

- El contexto y datos que necesite los obtiene mediante parámetros de entrada y/o propiedades de configuración
- Sus dependencias se basan en interfaces, no en implementaciones concretas
- Dispone de una API para la realización de sus funciones y el acceso a su estado.
- No requiere modificación de su código para su uso

Independiente

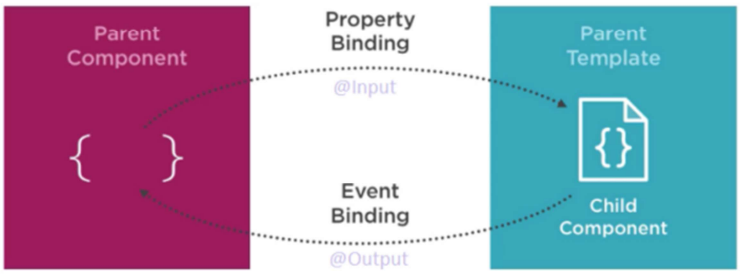
- Sin acoplamiento con el resto de la aplicación
- No accede ni modifica directamente componentes externos
- Comunica sus cambios de estado mediante eventos
- Empaquetado en su propio módulo junto con sus dependencias directas

Confiable

- Testado (test unitarios y de integración)
- API documentada
- Mantenimiento correctivo y evolutivo (empresa, comunidad)

Comunicación mediante propiedades Input y Output

- Angular usa:
  - Propiedades **Input** para implementar **parámetros** de entrada al componente
  - Propiedades **Output** para implementar **eventos** de salida del componente

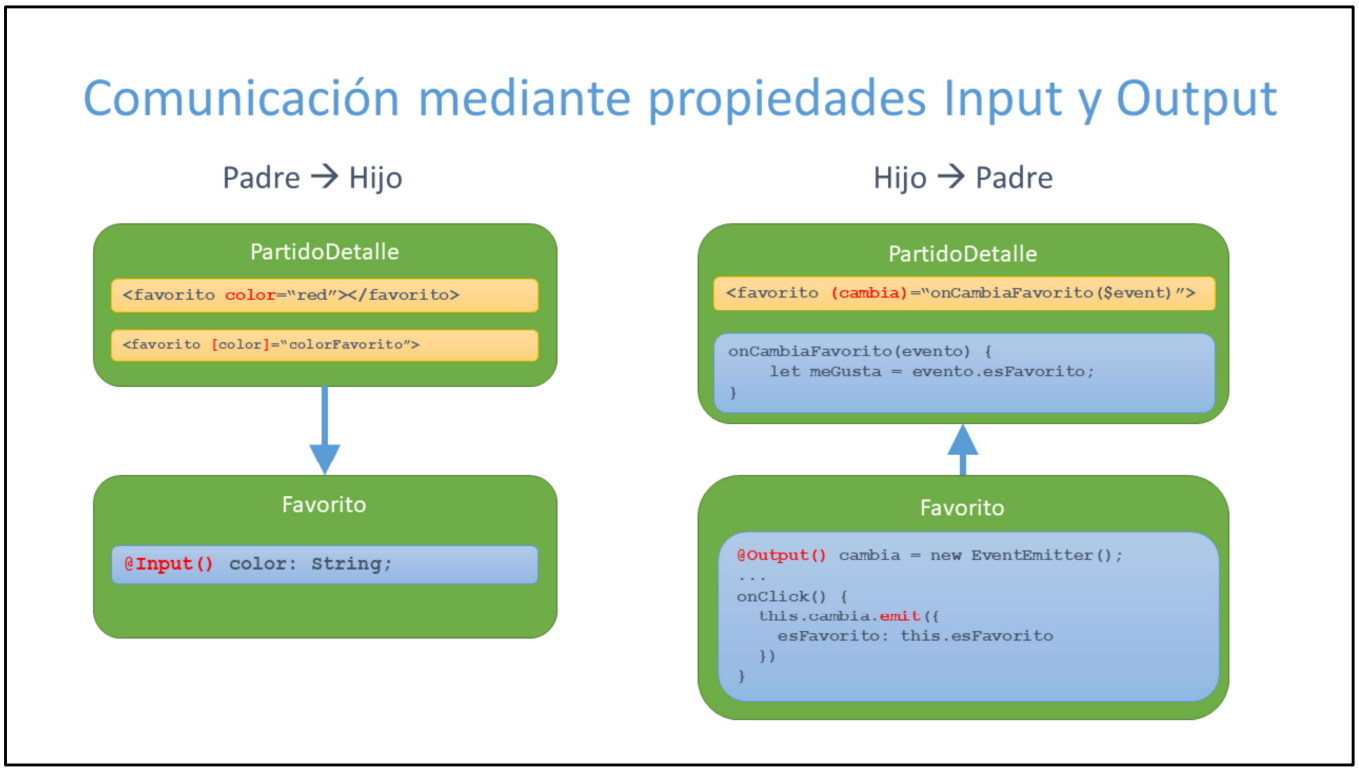


https://hahoangv.wordpress.com/

2 de las características fundamentales de un componente reutilizable son la capacidad de recibir parámetros de entrada, y la de comunicar sus cambios de estado mendiante eventos.

Para implementar los parámetros de entrada se usan las “Input properties”, y para implementar los eventos se usan las “Output properties”.

Como se puede ver en el gráfico, el componente padre puede usar el binding de propiedades para establecer el valor de los parámetros de entrada, y por otra parte usar el binding de eventos para ser informado de los cambios de estado del componente hijo.



Poniendo un ejemplo práctico, vamos a usar uno de los componentes que vamos a desarrollar luego, llamado “Favorito”.

Este componente implementa una estrella que se puede pulsar para marcar algo como favorito. Queremos que disponga de una propiedad mediante la que podamos establecer el color de la estrella cuando está marcada.

Esa propiedad la llamaremos “color” y será de tipo string. En el componente Favorito la definimos como una propiedad string y el decorador o anotación @Input. Eso le indica a Angular que es una propiedad de entrada.

El componente PartidoDetalle quiere usar el componente Favorito para indicar si un partido en concreto es o no favorito. Puede usar la propiedad “color” del componente bien directamente o dinámicamente mediante binding de propiedades.

Por otra parte, el componente Favorito debe emitir un evento cuando se modifique su estado, de forma que PartidoDetalle pueda usar el binding de eventos y realizar una acción específica según el estado del componente Favorito.

Para ello, definimos la propiedad “cambia” con el decorador @Output, y la inicializamos con un nuevo objeto del tipo “EventEmitter”. Cuando el componente deba lanzar el

evento, por ejemplo cuando se haga click en la estrella, debe llamar al método “emit” de la propiedad Output, pasándole como parámetro un objeto que representará la información del evento.

Componente “Favorito”

Definición
El componente Favorito permite marcar un valor como favorito dentro de una colección. Muestra una estrella en forma de silueta (no favorito) o sólida (favorito) con un tamaño y color configurables. Los valores marcados como favoritos se almacenan en el local storage del navegador, bajo una variable con el nombre de la colección.

Propiedad INPUT	Tipo	Descripción
coleccion	string	Nombre de la colección (variable en el local storage). Por defecto “favoritos”.
valor	string	Valor a guardar en la colección. Es obligatorio.
color	string	Color de la estrella cuando es favorito. Por defecto “orange”.
escala	number	Escala de la estrella. Por defecto 1.

Propiedad OUTPUT	Descripción	Evento
cambia	Lanza un evento cuando se modifica el estado de favorito del componente.	FavoritoEvent { coleccion: string, valor: string, esFavorito: boolean };

Componente “Favorito”

Local Storage:

- **fav_partidos:** [{idLocal:”BARÇA”, idVisitante: “ROMA”}]
- **fav_participantes:** [“MADRID”, “BARÇA”, “SEVILLA”]

```
<h2> Lista de partidos </h2>
<ul>
  <li *ngFor="let partido of partidos">
    <favorito coleccion="fav_partidos" [valor]="partido"></favorito>
    <span>{{partido.idLocal}} - {{partido.idVisitante}} </span>
  </li>
</ul>

<h2> Lista de equipos </h2>
<ul>
  <li *ngFor="let participante of participantes">
    <favorito coleccion="fav_participantes" [valor]="participante.id"></favorito>
    <span>{{participante.nombre}} </span>
  </li>
</ul>
```

Componente “Favorito”

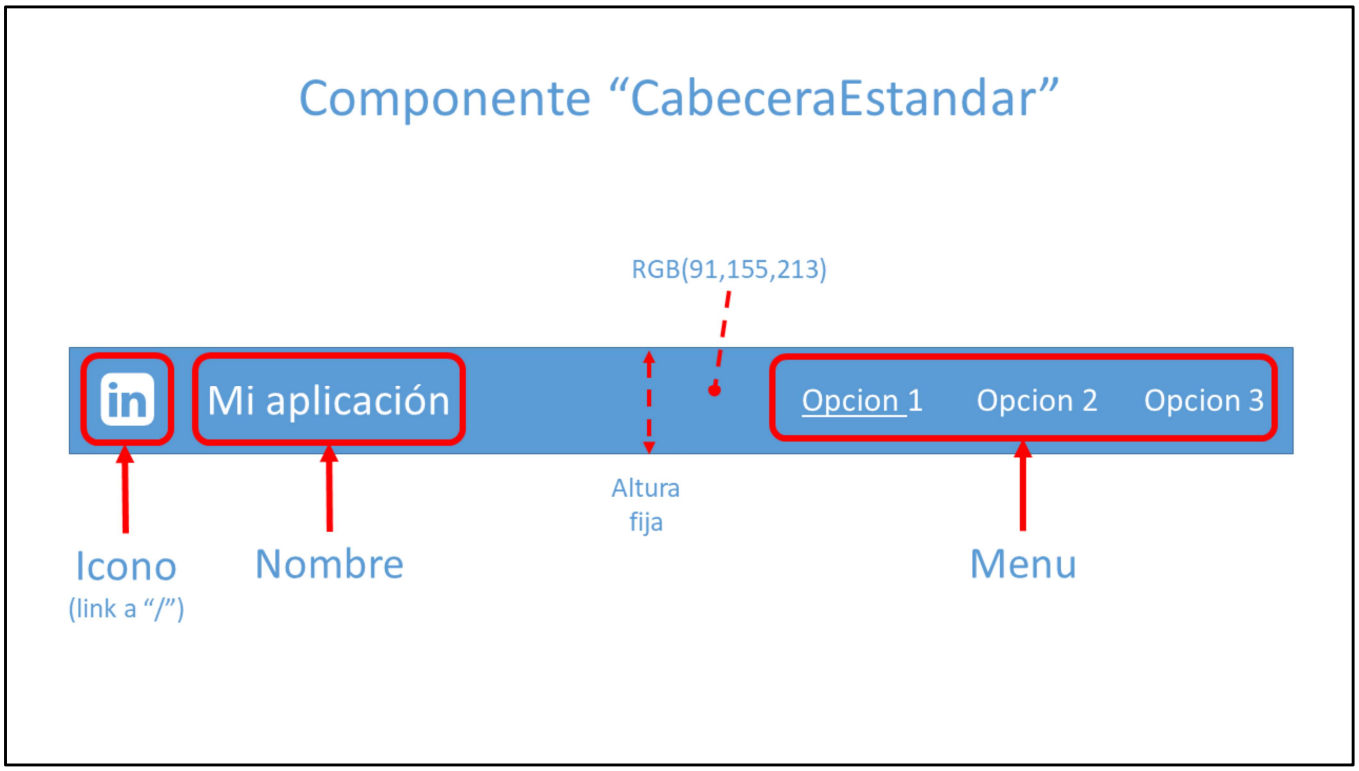
Componentes “plantilla”

- Se usan para estructurar contenido
- Marcan el layout y los elementos funcionales que lo componen
- Alternativa a simples plantillas HTML

Hay casos en los que, además de dar funcionalidad, o en lugar de ello, un componente puede usarse para estructurar contenido.

Es muy habitual que en el ámbito de una organización, existan plantillas para el layout general de las páginas, las cabeceras, menus, paneles específicos, etc. Normalmente esto estará definido en una guía de estilo o similar.

A efectos prácticos, esta estandarización se puede basar en plantillas HTML y CSS, que los equipos de desarrollo estén “obligados” a utilizar. Pero en un enfoque de aplicaciones basadas en componentes, como usa Angular, es preferible construir componentes que implementen los estándares marcados por la organización.



Supongamos que en nuestra organización, las cabeceras de las aplicaciones deben cumplir con unos determinados estándares.

La información “variable” será el logo, el nombre de la aplicación y el menú de navegación, pero otras características son fijas: color, layout, enlace a inicio, etc.

Componente “CabeceraEstandar”

Definición
El componente CabeceraEstandar define una estructura para las cabeceras de las aplicaciones, y permite personalizar el logotipo y el nombre de la aplicación. El contenido dentro del elemento <cabecera-estandar> se añade a la salida como menú de navegación.

Propiedad INPUT	Tipo	Descripción
icono-logo	string	Nombre del icono
nombre-aplicacion	string	Nombre de la aplicación

Propiedad OUTPUT	Descripción	Evento
No tiene		

Componente “CabeceraEstandar”

app.component.html

```
<cabecera-estandar icono-logo="sport_soccer" nombre-app="Datos
deportivos">

   <a href="partidos">Partidos</a>
   <a href="participantes">Participantes</a>
   <a href="sucesos">Sucesos</a>

</cabecera-estandar>
```

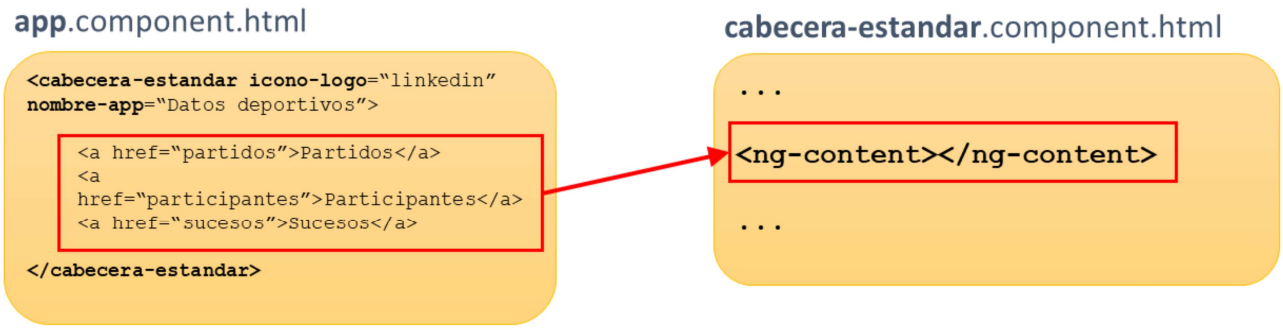
Un componente reusable para esta cabecera podría usarse de la forma que vemos aquí.

Un elemento “cabecera-estandar”, con 2 atributos “icono-logo” y “nombre-app”, y como contenido del elemento se incluyen los enlaces de navegación.

Ya hemos visto cómo implementar las propiedades de entrada. Para completar el componente debemos saber cómo insertar código HTML suministrado dentro de la etiqueta.

Componente “CabeceraEstandar”

- **ng-content**: Inserta un bloque de contenido en la plantilla HTML del componente



Para insertar contenido directamente desde la plantilla, Angular tiene la directiva “ng-content”. En su forma más simple, se sustituye por todo el contenido que se inserta en la etiqueta del componente.

Librerías propias

- Una librería permite empaquetar módulos, componentes y servicios reusables, e incorporarlos a una aplicación como cualquier otro módulo de node.
- Pueden distribuirse en un equipo, o bien se pueden compartir globalmente en el repositorio de node.js

Crear una librería

1. Crear un espacio de trabajo sin aplicacion

```
> ng new librerias --create-application=false
```

2. Crear la librería dentro del nuevo espacio

```
> cd librerias
> ng generate library libreria-dim
```

3. Copiar los módulos y componentes en la subcarpeta **/projects/libreria-dim/src/lib**

4. Añadir referencias a los módulos y componentes en el fichero **/projects/libreria-dim/src/public-api.ts**

```
export * from './lib/cabecera-estandar/cabecera-estandar.module';
export * from './lib/cabecera-estandar/cabecera-estandar/cabecera-estandar.component';
export * from './lib/favorito/favorito.module'
export * from './lib/favorito/favorito/favorito.component'
```

5. Hacer build de la librería (crea una subcarpeta *dist/libreria-dim*)

```
> ng build libreria-dim
```

Estructura del espacio de trabajo creado:

- **/projects**: como con el comando "new" le hemos dicho que no cree una aplicación, lo que ha creado es un espacio de trabajo, en el que pueden haber varios proyectos. En esta carpeta es donde se van creando los diferentes proyectos.
    - **/librería-dim**: esta carpeta la ha creado el comando ng generate library, y es donde pondremos el código de los componentes y servicios
        - **/src**: De esta carpeta cuelga el código fuente.
            - **/lib**: En esta carpeta pondremos los componentes y servicios. Por defecto Angular nos crea un módulo, un componente y un servicio vacíos.
            - **public-api.ts**: En este fichero se indican los componentes y servicios que se deben exportar en la librería, es decir, los que serán usables por los consumidores externos. Esta exportación es adicional a la exportación de cada componente en su módulo. Angular ha añadido automáticamente los elementos creados.
            - **test.ts**: este fichero está relacionado con los test.

        - **ng-package.json**: Este fichero tiene propiedades que se usan en el proceso de construcción de la librería. Por ejemplo la carpeta de salida

de los ficheros.

Compartir la librería

- **Para compartirla con un equipo:**
  - Comprimir la carpeta "dist/libreria-dim" en formato zip / tar
  - Distribuir el fichero comprimido a quien la quiera usar

- **Para compartirla globalmente:**
  - Añadir información de autor, versión, etc, en el fichero **package.json**
  - Volver a construir la librería usando el atributo –prod, para generar la librería en produccion:

```
> ng build libreria-dim --prod
```

  - Si no se tiene, es necesario registrar un usuario en npm (https://www.npmjs.com/signup), y hacer login con **npm login**
  - Publicar la librería en el **repositorio de Node**: Ejecutar **npm publish** desde la carpeta dist/librería-dim:

```
> npm publish
```

Usar la librería en Mi Aplicacion

1. Instalar en mi proyecto la librería externa
   - Para instalar desde un fichero comprimido:

```
C:\proyectos\MiAplicacion> npm install ruta/a/fichero/libreria-dim.tar
```

   - Para instalar desde la librería publicada en npm:

```
C:\proyectos\MiAplicacion> npm install libreria-dim
```

2. Importar los módulos o componentes usando el nombre de la librería:

```
import { CabeceraEstandarModule } from 'libreria-dim';
...

@NgModule({
...
   imports: [CabeceraEstandarModule]
...
});
```