Más sobre componentes

Cómo mejorar nuestros componentes

Tipado fuerte e interfaces

Encapsulando estilos

Enlaces al ciclo de vida

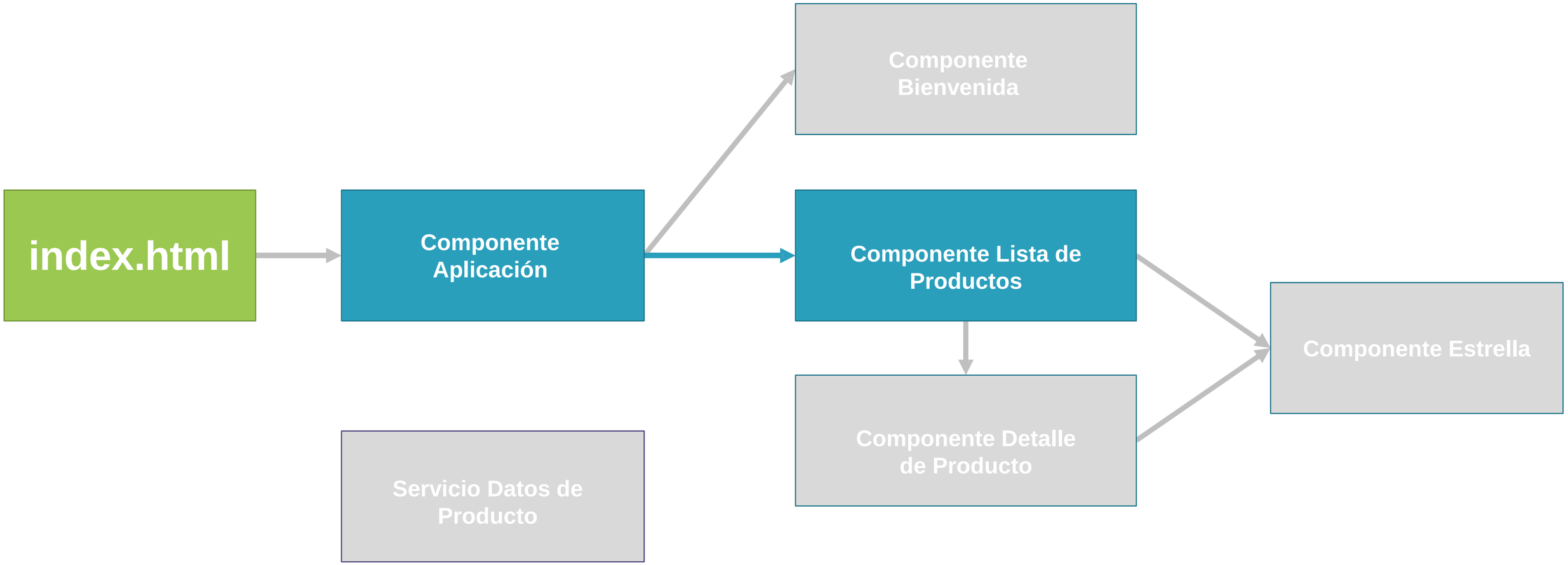
Pipes personalizados

Componentes anidados

Resumen

- **Definir un interface**
- **Encapsulando los estilos de componentes**
- **Utilización de los enlaces del ciclo de vida**
- **Crear un pipe personalizado**

Arquitectura de la aplicación



Tipado fuerte

```
export class ProductListComponent {
    pageTitle: string = 'Product List';
    showImage: boolean = false;
    listFilter: string = 'cart';
    message: string;

    products: any[] = […];

    toggleImage(): void {
        this.showImage = !this.showImage;
    }

    onRatingClicked(message: string): void {
         this.message = message;
    }
}
```

Un interface es una
especificación que identifica
un conjunto relacionado de
propiedades y métodos

Dos formas de utilizar un interface

```
export interface IProduct {
  productId: number;
  productName: string;
  productCode: string;
  releaseDate: string;
  price: number;
  description: string;
  starRating: number;
  imageUrl: string;
}
```

Como un tipo

```
products: IProduct[] = [];
```

```
export interface DoTiming {
    count: number;
    start(index: number): void;
    stop(): void;
}
```

Como un conjunto de características

```
export class myComponent
             implements DoTiming {
  count: number = 0;
  start(index: number): void {
     ...
  }
  stop(): void {
     ...
  }
}
```

Declarando un interface como un tipo de datos

```
export interface IProduct {
    productId: number;
    productName: string;
    productCode: string;
    releaseDate: Date;
    price: number;
    description: string;
    starRating: number;
    imageUrl: string;
}
```

Palabra clave `export`

Nombre de interfaz

Palabra clave `interface`

Uso de un interface como un tipo de datos

```
import { IProduct } from './product';

export class ProductListComponent {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';

  products: IProduct[] = […];

  toggleImage(): void {
      this.showImage = !this.showImage;
    }
}
```

Manejo de estilos únicos para componentes

- **Las plantillas requieren el uso de estilos únicos**
- **Podemos integrar los estilos directamente en HTML.**
- **Podemos crear una hoja de estilos externa y enlazarla en index.html**
- **Pero, hay una forma mejor…**

Encapsular los estilos de los componentes

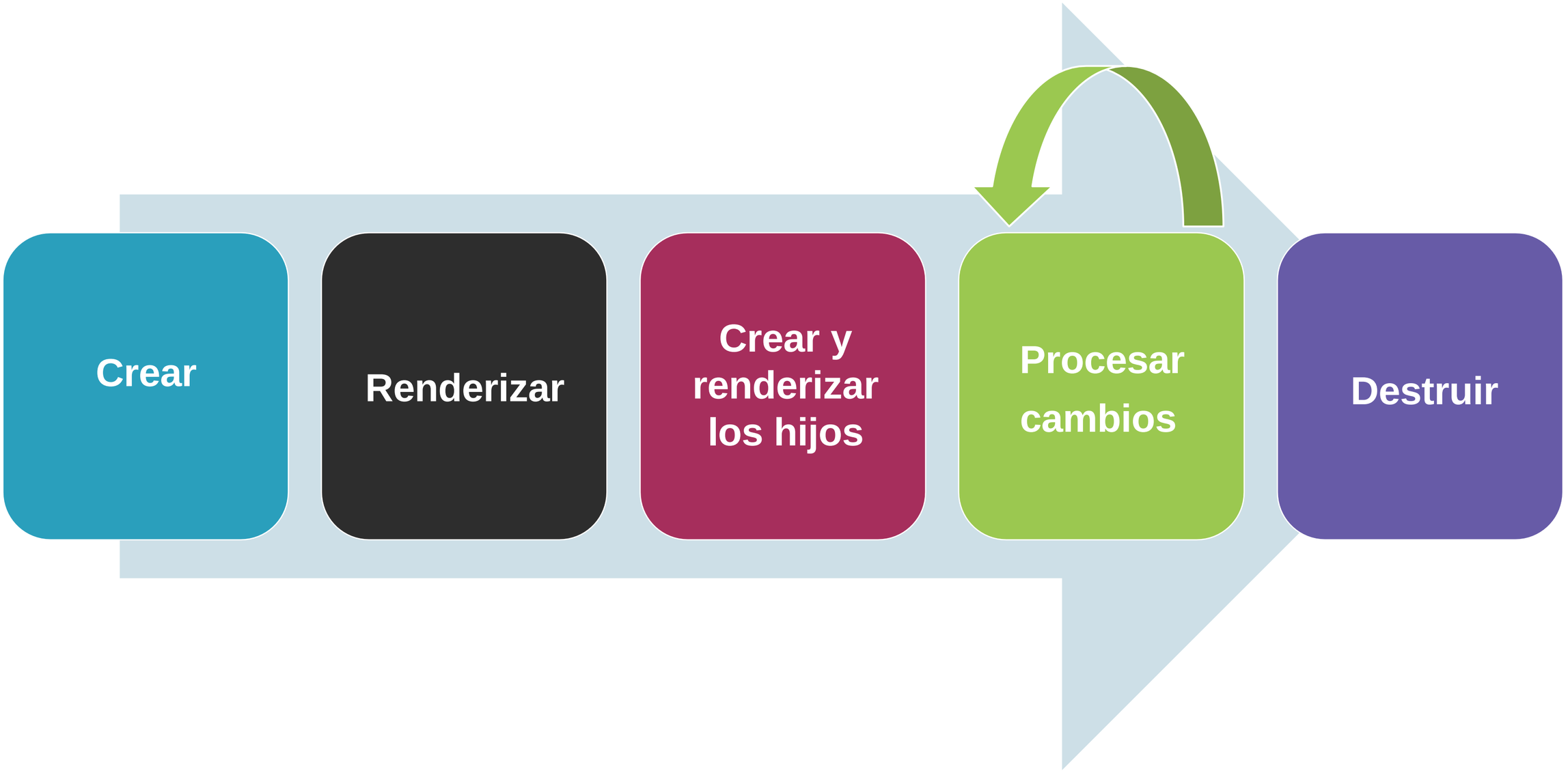
styles

```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styles: ['thead {color: #337AB7;}']})
```

styleUrls

```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styleUrls: ['./product-list.component.css']})
```

Ciclo de vida de los componentes



Un enlace del ciclo de vida es una interfaz que implementamos para escribir código cuando se produce un evento del ciclo de vida de un componente.

Enlaces del ciclo de vida de los componentes

`OnInit`: **Realizar la inicialización de componentes, recuperar datos**

`OnChanges`: **Realizar una acción después de cambiar las propiedades de entrada**

`OnDestroy`: **Realizar limpieza**

Cómo utilizar un enlace al clico de vida

2

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

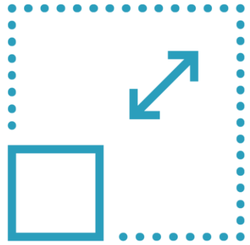
1

```
export class ProductListComponent implements OnInit {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';
  products: IProduct[] = […];
```

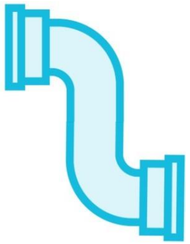
3

```
  ngOnInit(): void {
        console.log('In OnInit');
  }
}
```

Transformación de datos mediante Pipes



Transformar las propiedades enlazadas antes de la visualización



Pipes integradas: fecha, número, decimal, porcentaje, moneda, json, etc.

Pipes personalizados

Crear un pipe personalizado

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
    name: 'convertToSpaces'
})
export class ConvertToSpacesPipe implements PipeTransform {

  transform(value: string,
            character: string): string {
  }
}
```

Uso de un pipe personalizado

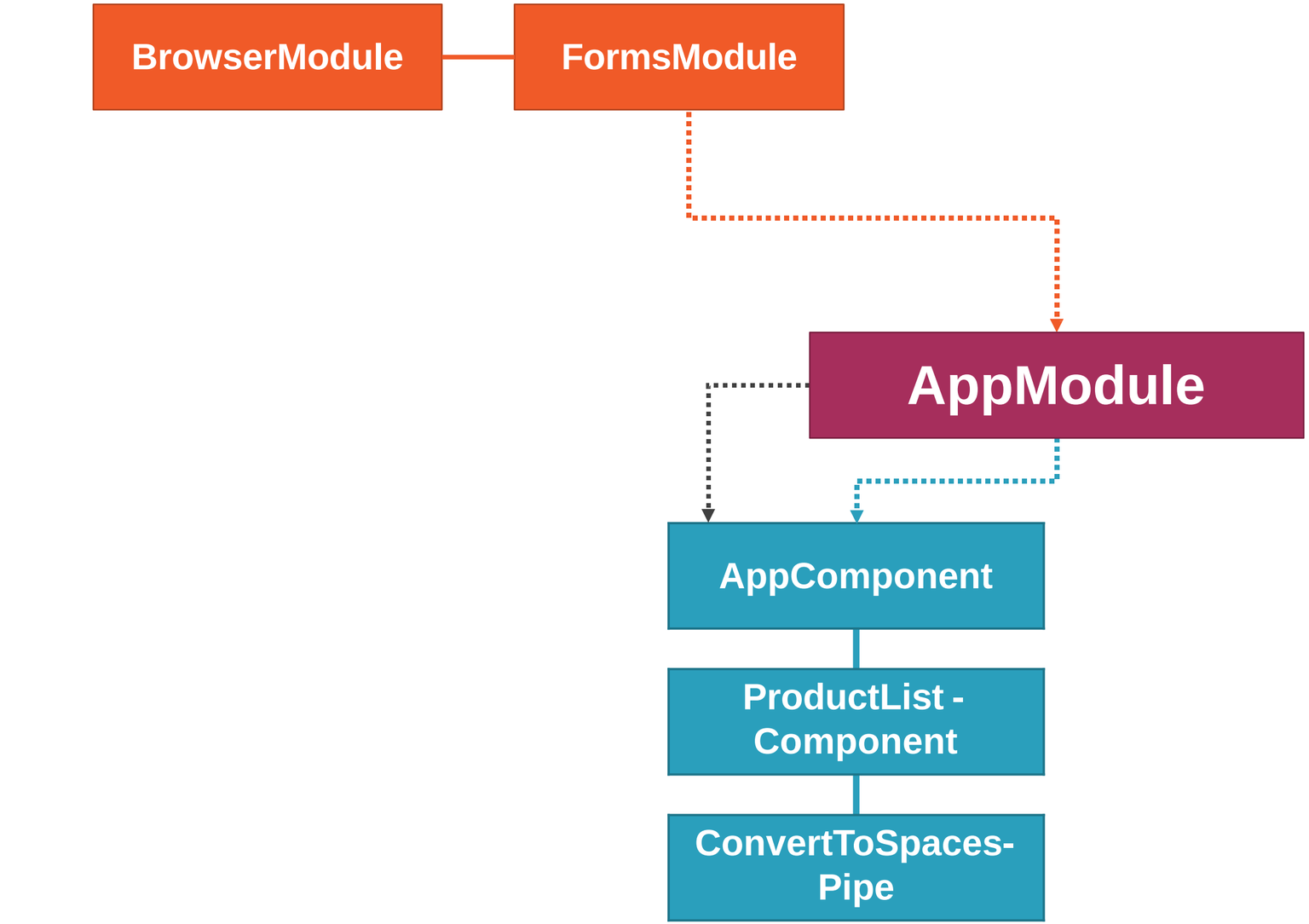
Plantilla

```
<td>{{ product.productCode | convertToSpaces:'-'}}</td>
```

Pipe

```
transform(value: string, character: string): string {


}
```


BrowserModule
FormsModule
AppModule
AppComponent
ProductList - Component
ConvertToSpaces-Pipe
Imports
Exports
Declaraciones
Arranque

Uso de un pipe personalizado

Plantilla

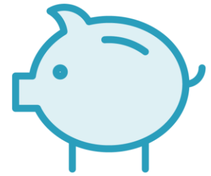
```
<td>{{ product.productCode | convertToSpaces:'-'}}</td>
```

Módulo

```
@NgModule({
  imports: [
      BrowserModule,
      FormsModule ],
  declarations: [
       AppComponent,
       ProductListComponent,
       ConvertToSpacesPipe ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Getters y Setters

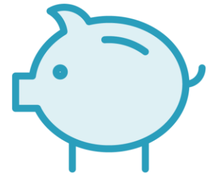
```
amount: number = 0;
```



```
get amount(): number {
    // process the amount
    // return amount from private storage
}
set amount(value: number) {
    // process the amount
    // retain amount in private storage
}
```

Getters y Setters

```
amount: number = 0;
```



```
private _amount: number = 0;
```

```
get amount(): number {
    // process the amount
    // return amount from private storage
    return this._amount;
}
set amount(value: number) {
    // process the amount
    // retain amount in private storage
    this._amount = value;
}
```

Getters y Setters

```
private _amount: number = 0;
```

```
get amount(): number {
    // process the amount
    // return amount from private storage
    return this._amount;
}
set amount(value: number) {
    // process the amount
    // retain amount in private storage
    this._amount = value;
}
```

```
this.amount = 200;
```

```
console.log(this.amount);
```

Filtrado de una lista

```
products: IProduct[] = [...];
```

```
performFilter(): IProduct[] {
    return this.products.filter();
}
```

Una función flecha es una sintaxis compacta para definir una función.

Filtrando una lista

```
products: IProduct[] = [...];
```

```
performFilter(): IProduct[] {
    return this.products.filter((product: IProduct) =>
        product.productName.includes(this.listFilter));
}
```

Funciones flecha

Forma tradicional: Función con nombre (método)

```
capitalizeName(product: IProduct): string {
    return product.productName.toUpperCase();
}
```

Función flecha

```
(product: IProduct) => product.productName.toUpperCase();
```

Función flecha con múltiples sentencias

```
(product: IProduct) => {
    console.log(product.productName);
    return product.productName.toUpperCase();
}
```

Lista de comprobación de interfaces: Interfaz como tipo

- **palabra clave** `interface`
- **Propiedades y sus tipos**
- **Exportarlo**

```
export interface IProduct {
  productId: number;
  productName: string;
  productCode: string;
  ...
}
```

Uso de un interfaz como tipo de datos

```
products: IProduct[] = […];
```

Lista de comprobación de interfaces: Interfaz como conjunto de características

Implementando interfaces:

- **Palabra clave `implements` y nombre de interfaz**
- **Escribir código para cada propiedad y método**

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


export class ProductComponent implements OnInit {
  ngOnInit(): void {
        console.log('In OnInit');
  }
}
```

Styles Checklist: Encapsulating Styles

Lista de comprobación de estilos: Encapsulación de estilos

- **Propiedad** `styles`
  - **Especificar un array de cadenas de estilos**
- **Propiedad** `styleUrls`
  - **Especificar un array de rutas de estilos**

```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styleUrls: ['./product-list.component.css']})
```

Lista de comprobación de enlaces del ciclo de vida: Uso de enlaces del ciclo de vida

- Importar el interfaz del ciclo de vida
- Impementar el interfaz del ciclo de vida
- Escribir el código para el enlace del ciclo de vida

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

export class ProductComponent implements OnInit {
  ngOnInit(): void {
        console.log('In OnInit');
  }
}
```

Lista de comprobación de pipes: Crear un pipe personalizado

- Crear una clase que implemente PipeTransform
- Escribir código para el método Transform
- Decorar la clase con el decorador Pipe

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
    name: 'spacePipe'
})
export class SpacePipe implements PipeTransform {
  transform(value: string,
            character: string): string { ... }
}
```

Lista de comprobación de pipes: Utilizar un pipe personalizado

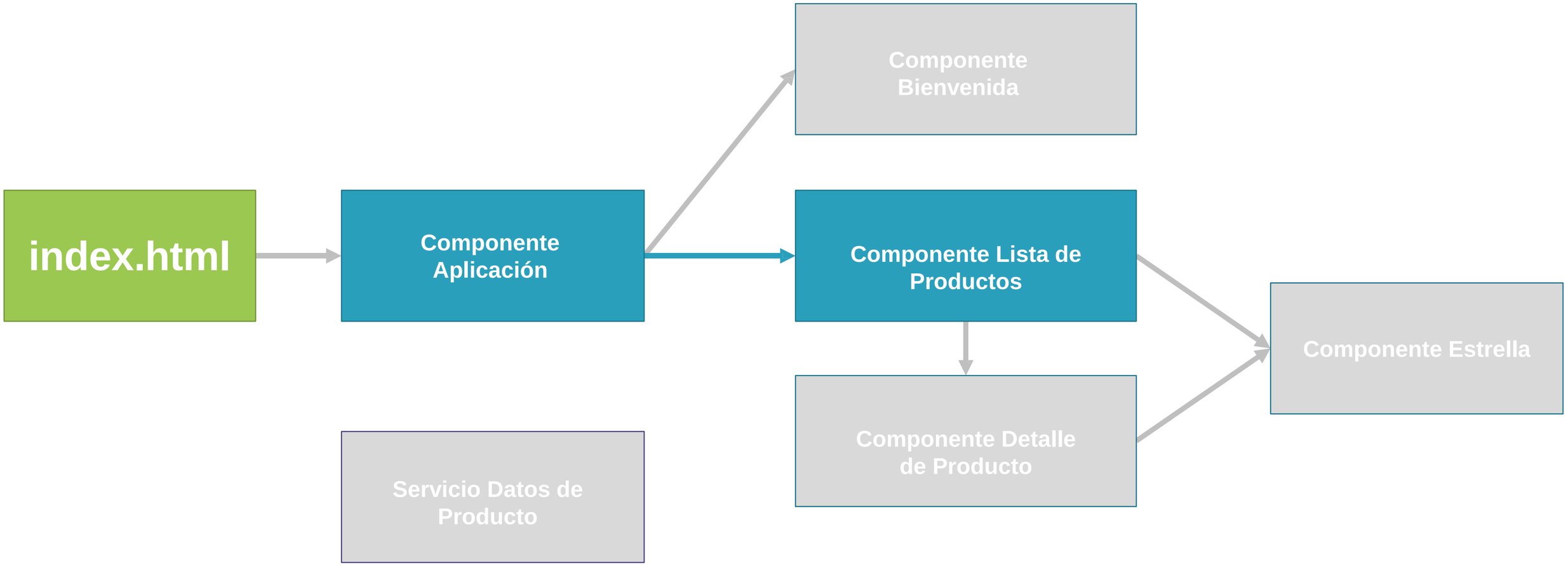
- **Agregar el pipe al array de declaraciones del módulo Angular**

```
@NgModule({
  imports: [...],
  declarations: [
       AppComponent,
       ProductListComponent,
       SpacePipe ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

- **Utilizar el pipe en una plantilla**
  - **Carácter de pipe**
  - **Nombre de pipe**
  - **Argumentos del pipe (separados por comas)**

```
{{ product.productCode | spacePipe:'-'}}
```

Arquitectura de la aplicación



A continuación …

Crear componentes anidados

Product List

Filter by:

Show Image

Product	Code	Available	Price	5 Star Rating
Leaf Rake	gdn 0011	March 19, 2021	$19.95	★★★½
Garden Cart	gdn 0023	March 18, 2021	$32.99	★★★★½
Hammer	tbx 0048	May 21, 2021	$8.90	★★★★★
Saw	tbx 0022	May 15, 2021	$11.55	★★★★½
Video Game Controller	gmg 0042	October 15, 2020	$35.95	★★★★½