Angular - Elementos básicos

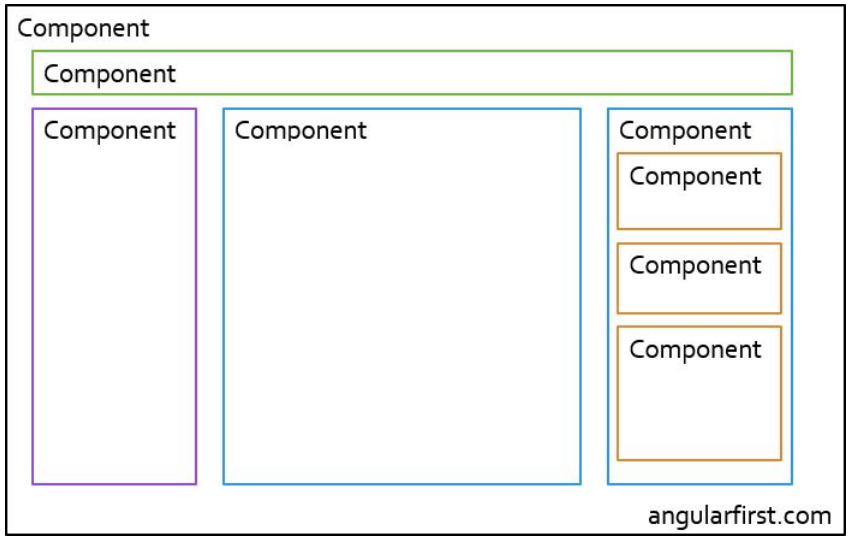
- Módulos
- Componentes
- Decoradores
- Directivas
- Servicios

Angular - Módulos

- Organizan la aplicación en bloques funcionales.
- Cada aplicación tiene, como mínimo, un módulo: el módulo principal o root, llamado **AppModule (App.module.ts**)
- En el módulo se declaran los componentes y, si es necesario, se importan otros módulos necesarios para su funcionamiento

Angular - Componentes

- Angular.io: *Los componentes definen **vistas**, que son conjuntos de elementos de pantalla que Angular puede elegir y modificar según la lógica y los datos de su programa.*
- Se podría definir como una **etiqueta HTML** creada por el programador que consta de una vista y de una lógica:
  - **Vista**: Es una plantilla (*template*) HTML que puede incorporar elementos propios de angular.
  - **Lógica**: Clase Typescript asociada a la vista
- Son los principales bloques de construcción de una aplicación Angular
- Cada componente gestiona una pequeña parte de UI



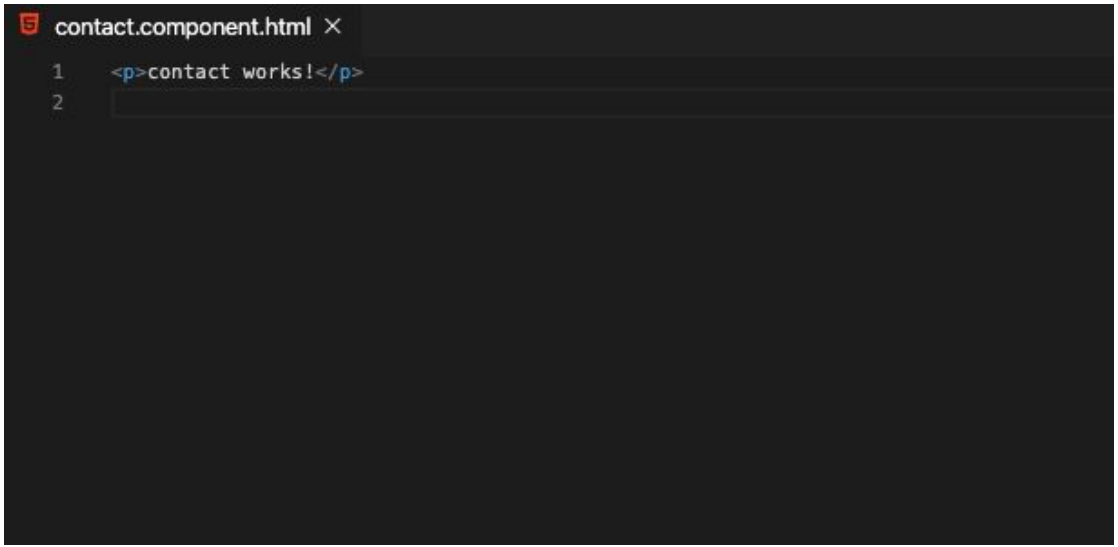
Angular - Componentes

- Angular.io: *Los componentes definen **vistas**, que son conjuntos de elementos de pantalla que Angular puede elegir y modificar según la lógica y los datos de su programa.*

```
moises@MBPro16-MC 02- FirstNgApp % ng generate component myNewComponent
CREATE src/app/my-new-component/my-new-component.component.css (0 bytes)
CREATE src/app/my-new-component/my-new-component.component.html (31 bytes)
CREATE src/app/my-new-component/my-new-component.component.spec.ts (686 bytes)
CREATE src/app/my-new-component/my-new-component.component.ts (313 bytes)
UPDATE src/app/app.module.ts (1095 bytes)
```

Componentes - Template (View)

- Creada en HTML define el diseño (*layout*) de la vista.
- Se pueden crear con una *template* externa, o bien, *inline*.
- Se le pueden añadir *directivas* y *bindings*.



Componentes - Clase

- Es el código, el lugar dónde se programa la lógica.
- Realmente, es una clase Typescript con **metadatos**.

contact.component.ts

```
1   import { Component, OnInit } from '@angular/core';
2
3   @Component({
4     selector: 'app-contact',
5     templateUrl: './contact.component.html',
6     styleUrls: ['./contact.component.css']
7   })
8   export class ContactComponent implements OnInit {
9
10    constructor() { }
11
12    ngOnInit(): void {
13    }
14
15  }
16
```

Componentes - Metadatos

- Información adicional acerca del componente.
- Se define mediante decoradores.
- Los decoradores se definen mediante **@decorator({ ...})**

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

@Component({
  selector: 'app-footer',
  template: `
    <p>
      footer works!
    </p>
  `,
  styles: [
  ]
})
export class FooterComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

Componentes - Registro en el módulo

- Para que un componente pueda ser utilizado otras “partes” de la aplicación necesita registrarse en el módulo *correspondiente* (app.module.ts)

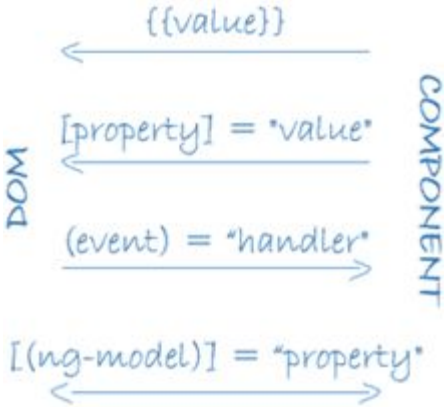
app.module.ts

You, a few seconds ago | 2 authors (Moisés Cid and others)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { ContactComponent } from './components/contact/contact.component';


You, a few seconds ago | 2 authors (Moisés Cid and others)
@NgModule({      Moisés Cid, 6 days ago • initial commit
  declarations: [
    AppComponent,
    ContactComponent
  ],
  imports: [
    BrowserModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Data Binding

- Los componentes deben ser dinámicos: permitir modificar datos, responder a interacciones de usuario y reaccionar ante eventos.
- El **data binding** una técnica en la que los datos permanecen sincronizados entre el componente y la vista.
    - Siempre que el usuario actualiza los datos en la vista, Angular actualiza el componente.
    - Cuando el componente obtiene nuevos datos, Angular actualiza la vista.



Data binding - *Interpolación*

- La información va del componente a la vista
- **{{ *templateExpression* }}**
- Angular evalúa la expresión, la convierte en cadena y reemplaza ***templateExpression*** por su valor.
- Cada vez que ***templateExpression*** se actualiza, Angular actualiza la cadena.

heroes.component.ts

```ts
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styles: [
  ]
})
export class HeroesComponent implements OnInit {

  hero: Hero = {
    id: 1,
    name: 'WindStorm'
  };

  constructor() { }

  ngOnInit(): void {
  }

}
```

heroes.component.html

```html
<h2>{{hero.name}} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div><span>name: </span>{{hero.name}}</div>
```

Data binding - *Property binding*

- Permite asociar una **propiedad de un elemento HTML** (class, href, src, etc.) con una **propiedad de un componente**.
- [binding-target]=”***templateExpression***”
- Cuando cambia el valor del componente, Angular actualiza la vista.

```typescript
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styles: [
  ]
})
export class HeroesComponent implements OnInit {

  hero: Hero = {
    id: 1,
    name: 'WindStorm',
    lastUpdate: new Date(),
    isDisabled: true
  };

  title = 'List of Heroes';
  constructor() {
  }

  ngOnInit(): void {

  }

```

```html
<div *ngIf="hero.isDisabled">

  <h1 [innerText]="title"></h1>

<div>
```

Data binding - *Event binding*

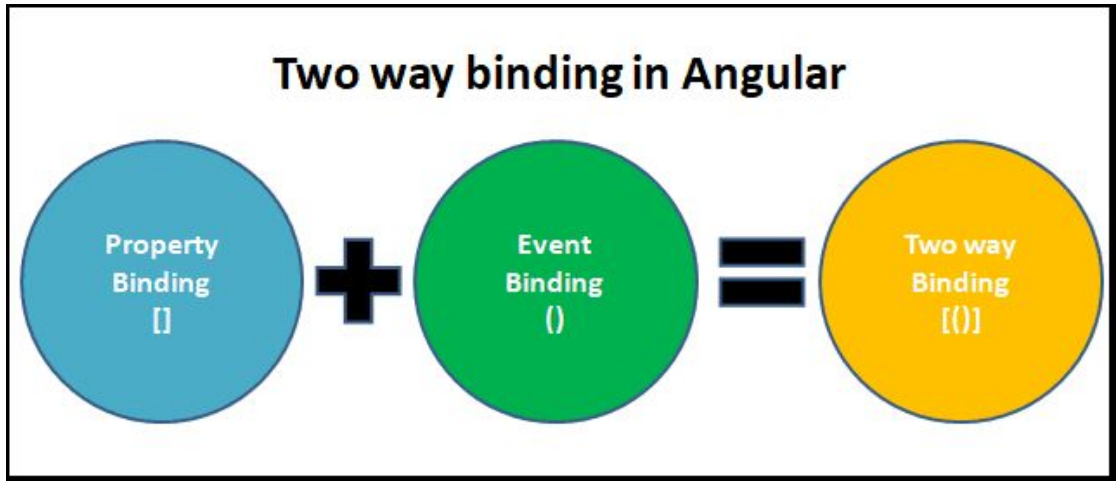
- Permite asociar **eventos del DOM** (clic, keystroke, hover, etc.) con un **método de un componente**.
- Por ejemplo, cuando el usuario cambia el valor de un input de texto se podría cambiar el modelo en el componente, ejecutar algún tipo de validación, etc.

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

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styles: [
  ]
})
export class HeaderComponent implements OnInit {

  prop1 = 'Propiedad 1';
  prop2 = 5;

  constructor() { }

  ngOnInit(): void {
  }

  changeProp1Value() {
    this.prop1 = 'Se ha cambiado el valor';
  }

}
```

```
<h1>Showing data from logic at HEADER</h1>
<p><strong>prop1</strong>: {{ prop1 }}</p>
<p><strong>prop2</strong>: {{ prop2 }}</p>



<h1>Executing logic when a template event occurs</h1>
<button (click)="changeProp1Value()">Cambiar valor</button>
```

Data binding - *Two Way data binding*

- Los cambios hechos en el componente se propagan a la vista y los cambios hechos en la vista se actualizan en el componente
- Se usa, fundamentalmente, en formularios



Data binding - *Two Way data binding*

- Los cambios hechos en el componente se propagan a la vista y los cambios hechos en la vista se actualizan en el componente
- Se usa, fundamentalmente, en formularios
- Requiere importar *FormsModule*, ya que no es parte de *Angular Core*

```
prop1 = 'Propiedad 1';
prop2 = 5;
value = '';
value1 = '';
prop3 = '';

constructor() { }

ngOnInit(): void {
}


clearProp3() {
  this.prop3 = '';
}
```

```
<h2>Two way binding</h2>
<input type="text" name="prop3" [(ngModel)]="prop3">
<p> Valor introducido: {{ prop3 }}</p>
<button (click)="clearProp3()">Borrar valor de prop3</button>
```

Directivas

- Sirven para cambiar la apariencia y el comportamiento de los elementos del DOM.
- Existen 3 tipos de directivas:
  - De componente: Son directivas con *template*
  - Estructurales: Cambian el **DOM** añadiendo o quitando elementos al mismo
  - De atributo: Cambian el comportamiento o la apariencia de un elemento, componente u otra directiva. Actúan sobre las propiedades del elemento.

Directivas Estructurales - *ngIf*

- Añada/elimina elementos HTML en base a la evaluación de una expresión

```
<div *ngIf="mostrar" class="card text-white bg-primary mb-3" style="width:100%">
  <div class="card-body">
    <h5 class="card-title"> {{ hero.name }} </h5>
    <p class="card-text">{{ hero.lastUpdate }} </p>
  </div>
</div>

<button (click)="mostrar = !mostrar" type="button" class="btn" [ngClass]="{'btn-danger': mostrar, 'btn-success': !mostrar}">Mostrar/Ocultar</button>
```

Directivas Estructurales - *ngFor*

- Repite una porción del HTML de una plantilla por cada elemento iterable de una colección

```
<div *ngFor="let item of heroes; let i= index;" class="card text-white bg-primary mb-3" style="width:100%">
  <div class="card-body">
    <h5 class="card-title">{{ i + '-' + item.name }}</h5>
    <p class="card-text">{{ item.lastUpdate }}</p>
  </div>
</div>
```

Directivas de Atributo - *ngClass*

- Añade/elimina clases CSS a un elemento HTML

```
<div *ngIf="mostrar" class="card text-white bg-primary mb-3" style="width:100%">
  <div class="card-body">
    <h5 class="card-title"> {{ hero.name }} </h5>
    <p class="card-text">{{ hero.lastUpdate }} </p>
  </div>
</div>

<button (click)="mostrar = !mostrar" type="button" class="btn" [ngClass]="{'btn-danger': mostrar, 'btn-success': !mostrar}">Mostrar/Ocultar</button>
```

```
<some-element [ngClass]="'first second'">...</some-element>

<some-element [ngClass]="['first', 'second']">...</some-element>

<some-element [ngClass]="{'first': true, 'second': true, 'third': false}">...</some-element>

<some-element [ngClass]="stringExp|arrayExp|objExp">...</some-element>

<some-element [ngClass]="{'class1 class2 class3' : true}">...</some-element>
```

https://angular.io/api/common/NgClass

Directivas de Atributo - *ngStyle*

- Añade o elimina un conjunto de estilos HTML
- Se utiliza para cambiar múltiples propiedades de estilo de los elementos HTML.
- Es posible *bindear* esas propiedades con valores que pueden ser actualizados en los componentes

Set the font of the containing element to the result of an expression.

```
<some-element [ngStyle]="{'font-style': styleExp}">...</some-element>
```

Set the width of the containing element to a pixel value returned by an expression.

```
<some-element [ngStyle]="{'max-width.px': widthExp}">...</some-element>
```

Set a collection of style values using an expression that returns key-value pairs.

```
<some-element [ngStyle]="objExp">...</some-element>
```

https://angular.io/api/common/NgStyle

Directivas de Atributo - *ngStyle*

- Sin ngStyle

```
<div [style.font-size]="isSpecial ? 'x-large' : 'smaller'">
  This div is x-large or smaller.
</div>
```

- Con ngStyle

```
currentStyles: {};
/* . . . */
  setCurrentStyles() {
    // CSS styles: set per current state of component propertie
    this.currentStyles = {
      'font-style':  this.canSave      ? 'italic' : 'normal',
      'font-weight': !this.isUnchanged ? 'bold'   : 'normal',
      'font-size':   this.isSpecial    ? '24px'   : '12px'
    };
  }
```

```
<div [ngStyle]="currentStyles">
  This div is initially italic, normal weight, and extra large (24px).
</div>
```

Directivas de Atributo - *ngModel*

- Añade el *two-way data binding* a un elemento de un formulario HTML

Bootstrap

- Bootstrap es un framework basado en HTML, CSS y JavaScript para crear webs responsives y mobile first.
- Existen diferentes alternativas para instalar Bootstrap en Angular:
  - https://getbootstrap.com/docs/4.5/getting-started/introduction/
  - https://www.techiediaries.com/angular-bootstrap/

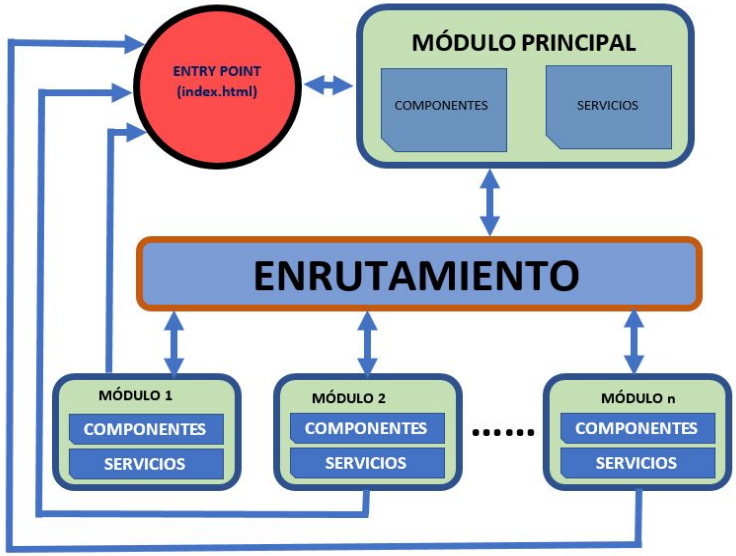
Ejercicio

- Crear una aplicación que disponga de lo siguiente:
  - Bootstrap
  - 3 componentes: encabezado, pie, cursos, inicio
  - El componente inicio será la presentación de la web
  - En el componente cursos definir un array que almacene, al menos, 2 cursos. Cada curso será un objeto con las siguientes propiedades: id, nombre, duración, descripción.
  - Haciendo usos de las directivas estructurales se deberán mostrar por pantalla los diferentes cursos en forma de *Cards* y habrá un párrafo HTML que indique el nº de cursos existentes.
  - Existirá un botón cuya funcionalidad será añadir un nuevo curso al array
  - Existirá otro botón que servirá para mostrar/ocultar todas las *Cards* y que cambiará su estilo en función de si las cursos se están mostrando o no
  - El componente encabezado mostrará la barra de navegación que permitirá navegar entre los componentes de inicio y cursos
  - Forzar a que tarde al menos 2 segundos en mostrarse los cursos. Mientras no se carga mostrar un aviso.

Módulo 3: Enrutamiento y navegación

RouterModule

- El **RouterModule** es el módulo que gestiona el enrutamiento y la navegación en Angular.
- Las aplicaciones Angular son SPA, es decir, sólo existe una página, pero múltiples vistas.
- El Router se encargará de procesar las rutas (URL's del navegador) y determinar cuál será la vista que deba mostrar en cada dirección.
- Gestiona la navegación **de una vista a otra vista** mediante la interpretación de la URL.
- Además, permite:
  - Pasar parámetros a la vista
  - Proteger determinadas rutas frente a usuarios no autorizados mediante *Guards*.



https://httpmasters.es/2018/05/02/estructura-de-una-aplicacion-angular-5/

RouterModule - Componentes

- Router
- Route
- Routes
- RouterOutlet
- RouterLink
- RouterLinkActive
- ActivatedRoute
- RouterLink Parameters

Router

- Objeto que habilita la navegación de un componente a otro cada vez que se hace clic en un link o escribe una URL en el navegador. Sus métodos de navegación principales son: `navigate()` y `navigateByURL()`

Route

- Es un objeto que consta, como mínimo, de un path y el componente asociado. Representa una ruta de navegación.
- Indica al **Router** que vista hay que mostrar cuando el usuario hace clic en un link o escribe una URL en el navegador.
- El **Router** parsea y construye la URL final usando la ruta, el objeto **Route**.
- Puede contener datos adicionales.
- Ejemplo de ruta:
  - `{ path: 'home', component: HomeComponent }`

Routes

- Array de las rutas soportadas por la aplicación. Es decir es un array de **Route**.
- Se definen en el fichero de rutas (**app-routing.module.ts**) que debe estar importado en el módulo principal.

```
const routes: Routes = [

  { path: '', component: HomeComponent },
  { path: 'home', component: HomeComponent, data: { titulo: 'Página de Inicio' } },
  { path: 'productos', component: ProductsComponent, data: { titulo: 'Página de Productos' } },
  { path: 'producto/:pk', component: ProductComponent, data: { titulo: 'Página de detalle de producto' } },
  { path: 'producto/search/:textToSearch', component: BuscadorComponent, data: { titulo: 'Página de búsqueda' } },
  { path: 'acerca-de', component: AboutComponent, data: { titulo: 'Página de Acerca de' } },
  { path: '404', component: NotFoundComponent },
  { path: '**', pathMatch: 'full', redirectTo: '404' }

];
```

RouterOutlet

- Es una directiva que actúa como contenedor en las que se mostrarán las vistas
- Se especifica con `<router-outlet></router-outlet>`
- Sin esta directiva, la aplicación solo mostraría los componentes indicados en `app.component.html`

RouterLink

- Es una directiva que enlaza los elementos HTML (`<a>`, `<button>`, ...) con una ruta (**Route**) para que cuando se haga clic se inicie la navegación a la ruta
- Se podría pensar que es el “href” del elemento HTML `<a>`
- Pueden contener parámetros

```
<ul class="navbar-nav mr-auto">
  <li class="nav-item" routerLinkActive="active">
    <a class="nav-link" [routerLink]="['home']">Inicio </a>
  </li>
  <li class="nav-item" routerLinkActive="active">
    <a class="nav-link" [routerLink]="['productos']">Productos </a>
  </li>
  <li class="nav-item" routerLinkActive="active">
    <a class="nav-link" [routerLink]="['acerca-de']">Acerca de</a>
  </li>
</ul>
```

RouterLinkActive

- Es una directiva añade/elimina clases CSS a un elemento HTML que está vinculado a un RouterLink

```
<ul class="navbar-nav mr-auto">
  <li class="nav-item" routerLinkActive="active">
    <a class="nav-link" [routerLink]="['home']">Inicio </a>
  </li>
  <li class="nav-item" routerLinkActive="active">
    <a class="nav-link" [routerLink]="['productos']">Productos </a>
  </li>
  <li class="nav-item" routerLinkActive="active">
    <a class="nav-link" [routerLink]="['acerca-de']">Acerca de</a>
  </li>
</ul>
```

ActivatedRoute

- Objeto que representa la ruta (Route) que está actualmente activada en el componente que se está cargando.
- Permite funcionalidades como obtener los parámetros que han sido enviados a través del **RouterLink**.

```
constructor(private activatedRoute: ActivatedRoute) {

  this.activatedRoute.params.subscribe(params => {
    const idProducto = Number(params.pk);
    this.product = this.findProduct(idProducto);

  });
}
```

RouterLink Parameters

- Es un array con los argumentos/parámetros de la ruta (**Route**)
- Este array se puede especificar tanto en la directiva RouterLink como en el método `Router.navigate()`

RouterModule - Fundamental

1. Establecer el `<base href>` en el index.html
2. Crear el módulo de enrutamiento (**RouterModule**)
    - Definir las rutas teniendo en cuenta su orden
    - Exportar el módulo
3. Importar el módulo de enrutamiento en el módulo principal (App.module.ts)
4. Asociar los eventos click de los elementos HTML con las rutas
5. Establecer dónde mostrar se mostrarán las vistas con la directiva **RouterOutlet**

Módulo 4: Servicios

Servicios

- Permiten crear código reutilizable para utilizar en cada componente que lo necesite.
- Son clases Typescript
- La gestión de los datos (obtener los datos del servidor, gestionar los posibles errores, etc) debería recaer siempre en los **servicios** y no en el propio componente: **principio de responsabilidad única** (SOLID).
- Los componentes deben centrarse en **presentar los datos** al usuario.

Servicios

- Se usan los servicios para:
  - Programar funcionalidades que son independientes de los componentes
  - Compartir lógica entre componentes
- Ventajas de implementar servicios
  - Son fáciles de testear
  - Son fáciles de depurar
  - Son altamente reutilizables

https://angular.io/api?query=pipe

Servicios - Inyección de dependencias

- Los servicios pueden ser instanciados directamente mediante `new MyService()` en el constructor, pero no es lo recomendable debido a:
  - Acoplamiento: Si el servicio cambia la definición de la clase se necesita actualizar todo el código en el que se use dicho servicio.
  - Test: Dificulta la realización de las pruebas automáticas.
- Los servicios pueden ser inyectados en los componentes (y en otros servicios) usando la **inyección de dependencias (DI)**.
- **DI** es una técnica que proporciona una instancia de un objeto a otro objeto que depende de él.

Servicios - Ejercicio

Módulo 5: Pipes

Pipes

- Son funciones simples que se usan para transformar datos.
- Aceptan un valor de entrada y devuelven un valor transformado (sin modificar el original).
- Pipes integradas:
  - DatePipe
  - UpperCasePipe
  - LowerCasePipe
  - CurrencyPipe
  - DecimalPipe
  - PercentPipe
  - SlicePipe
  - AsyncPipe
  - JsonPipe

https://angular.io/api?query=pipe

Pipes

- Sintasis:
  - expressionToTransform | pipeOperator [:pipeArguments]
- Ejemplo:

```
4   <h1 [innerText]="title"></h1>
5   <h2>{{hero.name}} Details</h2>
6   <div><span>id: </span>{{hero.id}}</div>
7   <div><span>name: </span>{{hero.name}}</div>
8   <div><span>last update: </span>{{ hero.lastUpdate | date : 'dd/MM/yyyy' }}</div>
```

Pipes - DatePipe

- Transforma una fecha en una cadena según el formato indicado
- Sintasis:
  - `dateToTransform | date [:format]`

```
<tr>
  <td> {{ fecha }} </td>
  <td> fecha </td>
  <td> {{ fecha | date: 'dd/MM/yyyy'  }} </td>
</tr>
<tr>
  <td> {{ fecha }} </td>
  <td> fecha MMMM-dd</td>
  <td> {{ fecha | date: 'MMMM - dd' : '' : 'es' }} </td>
</tr>
```

Pipes - Uppercase

- Transforma una cadena a mayúsculas
- Sintasis:
  - `stringToTransform | uppercase`

```
<tr>
  <td> {{ nombre }} </td>
  <td> uppercase </td>
  <td> {{ nombre | uppercase  }} </td>
</tr>
<tr>
  <td> {{ nombre }} </td>
  <td> lowercase </td>
  <td> {{ nombre | lowercase  }} </td>
</tr>
```

Pipes - Lowercase

- Transforma una cadena a minúsculas
- Sintasis:
  - `stringToTransform | lowercase`

```
<tr>
  <td> {{ nombre }} </td>
  <td> uppercase </td>
  <td> {{ nombre | uppercase  }} </td>
</tr>
<tr>
  <td> {{ nombre }} </td>
  <td> lowercase </td>
  <td> {{ nombre | lowercase  }} </td>
</tr>
```

Pipes - CurrencyPipe

- Transforma un número en una cadena con formato de moneda
- Sintasis:
  - `numberToTransform | currency[:currencyCode[:symbolDisplay[:digitInfo]]]`
    - currencyCode: Código ISO 4217 (‘US’ para USD, ‘EUR’ para euro, etc)
    - symbolDiplay: Booleando para indicar si mostrar o no el símbolo
    - digitInfo: *{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}*

```
14
15      <div><span>Precio: </span>{{ precio | currency: 'EUR' : true : 4.2-4 }}</div>
16
```

Pipes - DecimalPipe

- Transforma un número en una cadena según el formato indicado
- Sintasis:
  - `numberToTransform | number [:digitInfo]`
    - digitInfo: *{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}*

```
<tr>
  <td> {{ PI }} </td>
  <td> number </td>      You, a few seconds ago • Uncommitted
  <td> {{ PI | number:'.0-4' }} </td>
</tr>
```

Pipes - PercentPipe

- Formatea un número como un porcentaje
- Sintasis:
  - `numberToTransform | percent [:digitInfo]`
    - digitInfo: *{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}*

```
<tr>
  <td> {{ descuento }} </td>
  <td> porcentaje </td>
  <td> {{ descuento | percent }} </td>
</tr>
<tr>
  <td> {{ descuento }} </td>
  <td> porcentaje </td>
  <td> {{ descuento | percent:'2.2-2' }} </td>
</tr>
```

Pipes - SlicePipe

- Crea un nuevo array o cadena que contendrá un subconjunto de los elemento del original
- Sintasis:
  - `arrayOrStringToTransform | slice: start [:end]`

```
<tr>
  <td> {{ textoLargo }} </td>
  <td> slice </td>
  <td> {{ textoLargo | slice: 0 : 9 }} </td>
</tr>
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/slice

Pipes - AsyncPipe

- Permite que nos suscribamos a un Observable o a un Promesa desde la plantilla y devuelve el valor que emiten.
- Se realiza el **suscribe** cuando se carga el componente y se realiza el **unsubscribe** cuando el componente se destruye.

```
miPromesa = new Promise((resolve) => {
  setTimeout(() => {
    resolve('Datos obtenidos de la API');
  }, 2000);
});
```

```
<tr>
  <td> {{ miPromesa | json }} </td>
  <td> async </td>
  <td>{{ miPromesa | async }}</td>
</tr>
```

Pipes - JsonPipe

- Convierte un objeto en su representación en JSON.

```
<tr>
  <td> {{ myObject }} </td>
  <td> json </td>
  <td>
    <pre>
      {{ myObject | json }}
    </pre>
  </td>
</tr>
```