



# Introducción a Angular

UD2: Desarrollo de aplicaciones basado en  
componentes con Angular





# Objetivos de aprendizaje

- Aprender a crear una nueva aplicación Angular
- Comprender las principales partes que tenemos en una aplicación Angular
- Organizar una aplicación y crear componentes personalizados
- Entender las distintas formas de comunicación entre componentes y plantillas y las directivas básicas



# Índice

- Introducción a Angular
- Arranque de una App Angular
- Componentes en Angular
- Data binding
- Directivas





# Introducción a Angular

UD2: Desarrollo de aplicaciones basado  
en componentes con Angular

[Indice](#)

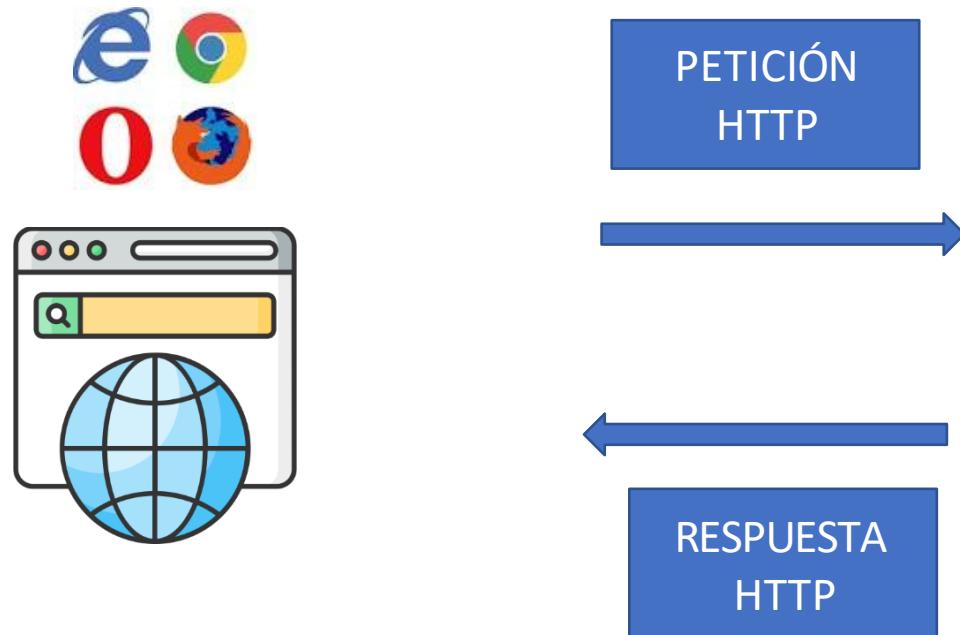


# Recordemos

- Al desarrollar en Web dividimos nuestro código en dos partes:
  - Parte servidora
    - Código que se ejecuta en un equipo remoto (servidor).
    - Funciones principales es el manejo de datos (acceso a bases de datos), autentificación, seguridad...
    - El usuario no tiene acceso al código de esta parte
  - Parte cliente
    - Código que se ejecuta en mi propio dispositivo (navegador Web)
- El diálogo entre cliente y servidor se establece a través del protocolo HTTP



# Modelo Web



©DESIGNALIKIE

# Tecnologías Web

Lado Cliente	 
Lado Servidor	 

# Frameworks

- Un framework es un conjunto de utilidades probadas y validadas que me permiten solucionar problemas que me encuentro de manera común al realizar una aplicación.
  - Organización de archivos, separación de funcionalidad, manejo de rutas, seguridad, acceso a datos, manejo de usuarios....
- Cada plataforma de desarrollo suele tener su propio framework asociado.





# Frameworks JS

- Actualmente está muy extendido el uso de 3 frameworks JS
- Son similares pero cada uno tiene sus particularidades y curva de aprendizaje





# ¿Qué es una SPA?

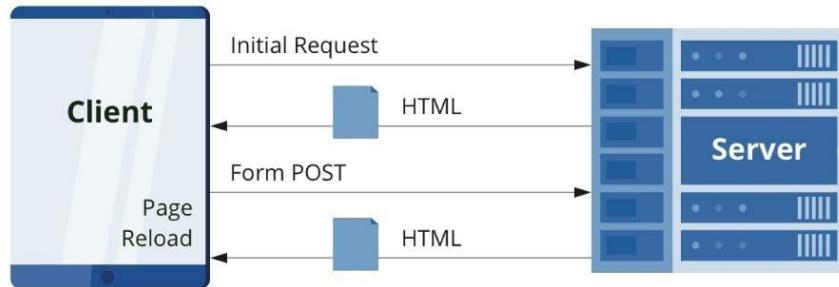
- SPA = Single Page Application
- Aplicación Web donde todo el contenido se carga en la primera petición
  - Primera carga más lenta pero mejora la experiencia de usuario durante la navegación



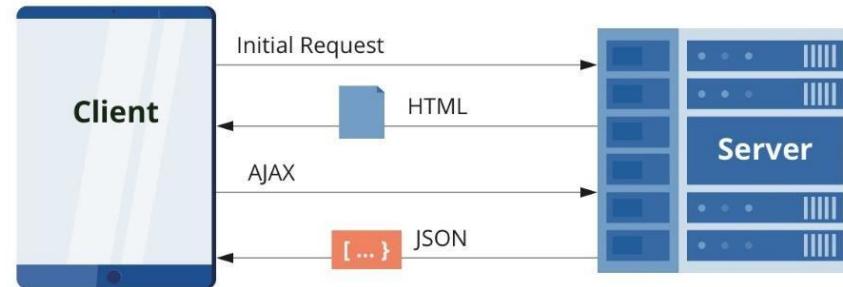


# SPA vs MPA

## Traditional Page Lifecycle



## SPA Lifecycle





# Angular

- Framework para aplicaciones Web SPA
- Código abierto
- Mantenido por Google 
- Lanzado en 2016 (Angular 2)
- Última versión estable: 20.3.12 (noviembre 2025)

EN ESTA PARTE DEL CURSO VAMOS A USAR LA VERSIÓN 20





# Últimas versiones

	Angular 19	Angular 20
Fecha de lanzamiento	Noviembre 2024	Mayo 2025
Principales novedades	<ul style="list-style-type: none"><li>• Standalone components/directives/pipes por defecto</li><li>• Hydratación incremental (SSR)</li><li>• Nuevas APIs de señales (“linkedSignal”, etc)</li></ul>	<ul style="list-style-type: none"><li>• API de Signals ahora estable</li><li>• Modo “zoneless” de detección de cambios en preview</li><li>• Mejoras de rendimiento, bundle más pequeños, SSR mejorado</li></ul>





# ¿Por qué la versión 20?

- Introduce el modelo moderno basado en *standalone components*.
- Simplifica la estructura del proyecto
  - Elimina la necesidad de módulos
  - Permite un desarrollo más rápido, limpio y fácil de mantener.
- Versión recomendada por el propio equipo de Angular



# Características de Angular

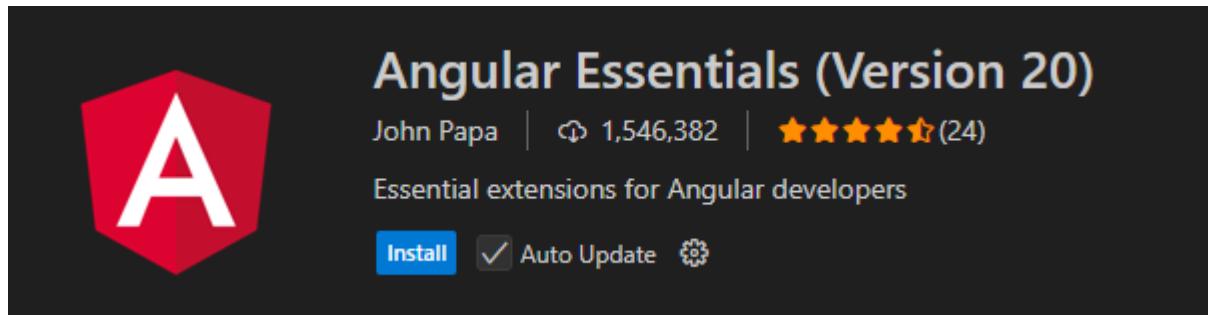
- Velocidad y rendimiento
  - Optimiza el código
  - Multiplataforma
  - Carga solo lo que necesitamos
- Productividad
  - Creamos elementos de manera rápida
  - Angular CLI
  - Integración con la mayoría de editores de texto





# Preparando el entorno

- Conjunto de Extensiones “Angular Essentials” de VSCode





# Preparando el entorno

- Otras extensiones:

- [Angular Language Service](#)
- [Angular Snippets](#)
- [Angular Schematics](#)
- [Angular 2 Inline](#)
- [Auto Close Tag](#)
- [Auto import](#)
- [Auto Rename Tag](#)
- [Error Lens](#)
- [Paste JSON as Code](#)
- [TypeScript Importer - optional](#)
- [Editor Config for VSCode](#)
- [Better Comments](#)
- [Tailwind CSS IntelliSense](#)
- [Pretty TypeScript Errors](#)





# Preparando el entorno

- Extensión Angular Dev Tools de Google Chrome

Descubrir    **Extensiones**    Temas

The screenshot shows the Angular DevTools extension page on the Chrome Web Store. At the top, there are three tabs: 'Descubrir', 'Extensiones' (which is underlined), and 'Temas'. Below the tabs, the extension icon (a purple and pink angular symbol) is displayed next to the text 'Angular DevTools'. Underneath the icon, the rating '3,7 ★ (187 valoraciones)' and a 'Compartir' button are visible. At the bottom of the screenshot, there are three buttons: 'Extensión', 'DevTools', and '400.000 usuarios'.



# Angular CLI

- Es una herramienta en línea de comandos para trabajar con Angular
- [Documentación](#)
- Instalación (global):



```
npm install -g @angular/cli@20
```

- Comprobación:



```
ng version
```

# Creación de un proyecto Angular



```
ng new NombreProyecto
```

- Instala paquetes y dependencias necesarios para nuestro proyecto
- Se nos hacen algunas preguntas para comenzar

Pregunta	Escogemos
Tipo de estilo que queremos añadir	<u>SCSS</u>
(SSR) (SSG/Prerendering)	<u>No</u>
zoneless	<u>No</u>
IA	<u>No</u>



# Inspeccionando el código

Archivo/carpeta	Explicación
package.json	Gestión de las dependencias de proyecto. Debería incluir las librerías de Angular y otras dependencias
node_modules	Carpeta con las dependencias
public/	Carpeta para <b>archivos estáticos</b> accesibles directamente por el navegador. (Imágenes, Iconos, pdf...)
src	La carpeta <b>más importante del proyecto</b> . Contiene <b>todo el código fuente</b> que programas tú.
main.ts	punto de arranque (con bootstrapApplication)
editconfig	Configura reglas comunes para todos los editores: <ul style="list-style-type: none"><li>Indentación</li><li>Uso de espacios/tabuladores</li><li>Codificación UTF-8</li></ul>



angular.json	Configuración de la aplicación Angular
tsconfig.json tsconfig.app.json tsconfig.spec.json	Configuración de TypeScript





# Inspeccionando el código (II)

Archivo/carpeta	Explicación
src	Carpeta donde se encuentra el código de nuestra aplicación
src/index.html	Base de nuestra aplicación
<u>src/main.ts</u>	Arranque de nuestra aplicación (punto de salida)
styles.css	Estilos a nivel de aplicación
assets	Documentación, archivos estáticos, imágenes, archivos de JS...

# Lanzando el proyecto



```
ng serve
```

- Hace una precompilación de nuestro proyecto
- Lo ejecuta en un servidor local bajo Node.js
- Se indica la dirección por defecto donde está levantado
- Opciones:



```
ng serve --port puerto
```

Puerto a medida



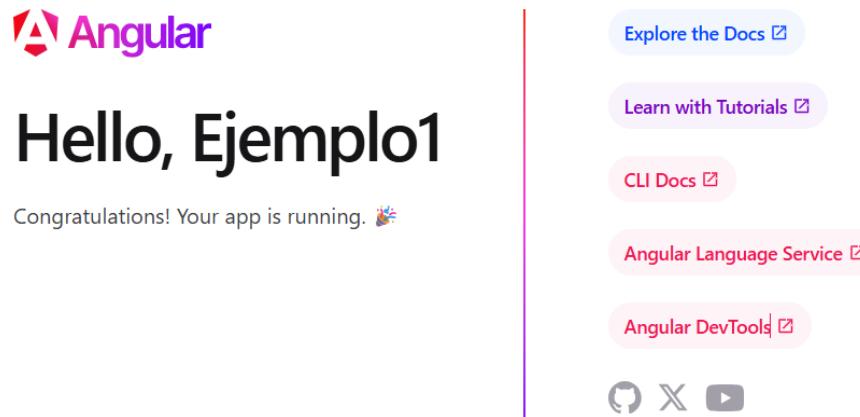
```
ng serve --open
```

Abre en el navegador



# Lanzando el proyecto

- Página predeterminada en <http://localhost:4200>



# npm y GitHub



- Nuestro repositorio git debe tener añadida la carpeta `node_modules` en `.gitignore`
- Cuando subamos nuestro proyecto al repositorio de GitHub, se subirá el archivo `package.json` que indica las dependencias pero no las propias dependencias
- Cuando recuperemos el proyecto desde GitHub, tenemos que descargar de nuevo las dependencias con el comando

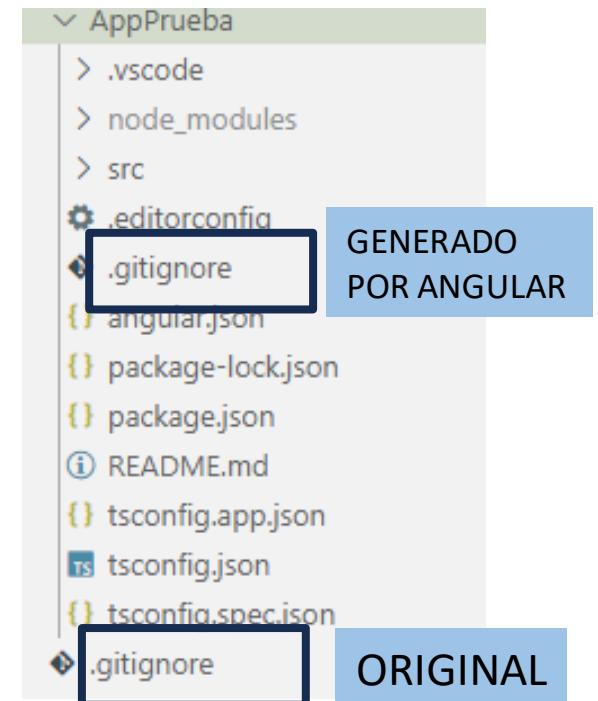


`npm install`



# Angular y Git

- Cuando creamos un proyecto Angular, si no estamos dentro de un repositorio Git se pre-inicializa uno.
- Se genera un archivo `.gitignore` con las rutas que se deberían ignorar cuando hacemos commit
- Teóricamente deberíamos tener un solo `.gitignore` pero git admite varios anidados





# Contenido de .gitignore

```
# Compiled output
/dist
/tmp
/out-tsc
/bazel-out

# Node
/node_modules
npm-debug.log
yarn-error.log

# IDEs and editors
.idea/
.project
.classpath
.c9/
*.launch
.settings/
*.sublime-workspace
```

```
# Visual Studio Code
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json
!.vscode/extensions.json
.history/*

# Miscellaneous
/.angular/cache
.sass-cache/
/connect.lock
/coverage
/libpeerconnection.log
testem.log
/typings

# System files
.DS_Store
Thumbs.db
```

# Angular y Git (II)

- También podemos copiar el contenido a nuestro `.gitignore`, adaptando las rutas y luego borrar el original

ORIGINAL

Subcarpeta (1 nivel)  
node\_modules

```
...  
# Node  
/node_modules  
npm-debug.log  
yarn-error.log
```

MODIFICADO

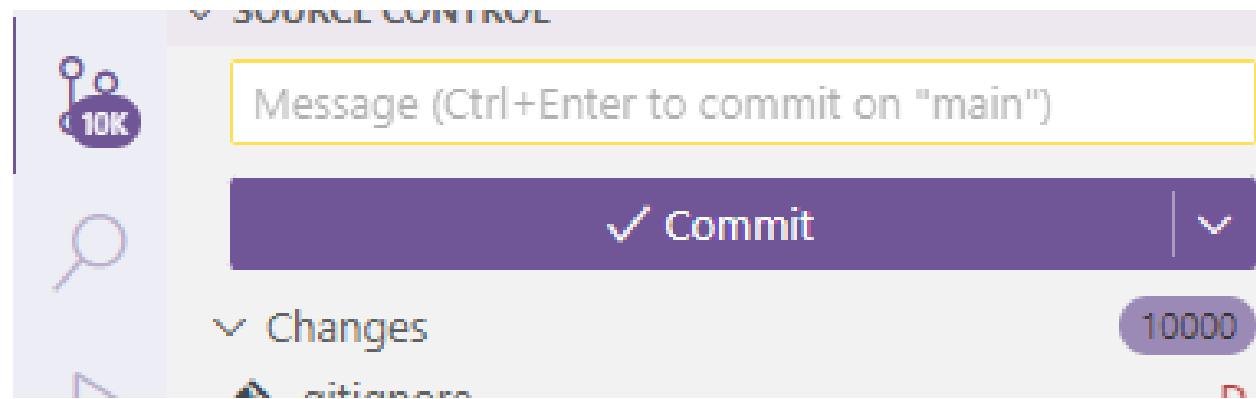
Cualquier carpeta  
node\_modules

```
...  
# Node  
node_modules  
npm-debug.log  
yarn-error.log
```

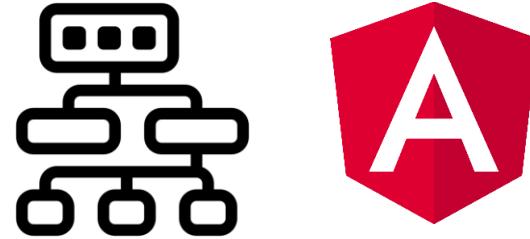




# Angular y Git



INDICATIVO DE QUE ESTOY HACIENDO ALGO MAL



# Arranque de una app Angular

UD2: Desarrollo de aplicaciones basado  
en componentes con Angular

[Indice](#)





# Estructura básica de un proyecto Angular

- Angular 20 la estructura se organiza principalmente mediante **componentes standalone**.
- Cada **componente standalone**:
  - Representa una unidad funcional de la aplicación.
  - Puede importar directamente otros componentes, servicios, directivas, pipes, formularios, etc.
- La aplicación ya **no necesita un módulo raíz**.
  - En su lugar, se inicia con `bootstrapApplication()` cargando el componente principal de la app.



# Bloques fundamentales



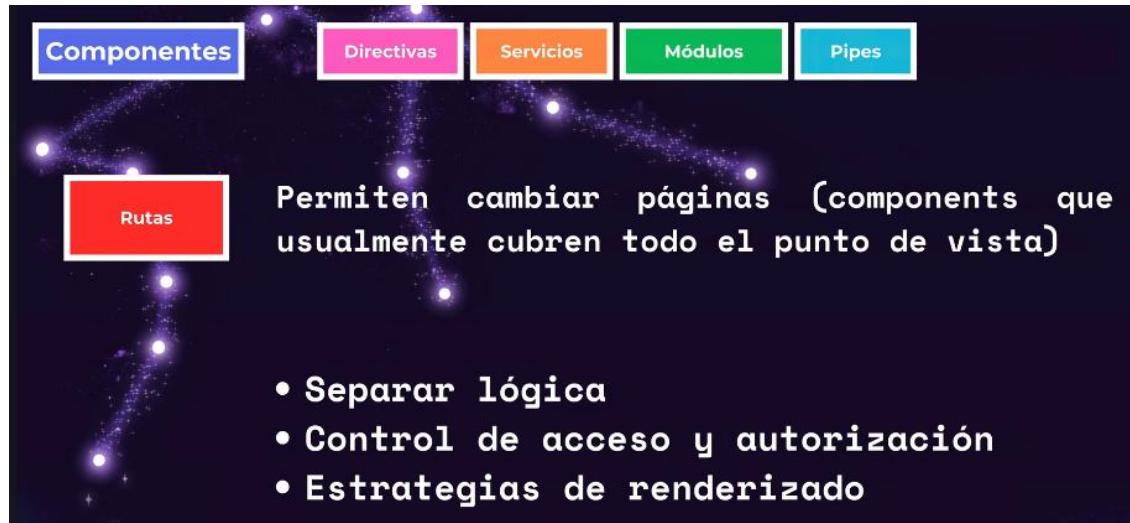
# Bloques fundamentales. Componentes

The diagram features a dark blue background with a purple starry nebula effect. At the top, five colored boxes represent Angular components: Rutas (red), Directivas (pink), Servicios (orange), Módulos (green), and Pipes (blue). Below them, a large blue box labeled "Componentes" contains a bulleted list. The first bullet point defines a component as a user interface piece. The second bullet point lists the three parts of a component: logic (TypeScript), styles (SASS/CSS), and a template (HTML), noting that styles are optional.

- Es una pieza que representa una parte de la interfaz de usuario.
- Lógica (TS)
- Estilos - (SASS, CSS, etc) - opcional
- Plantilla HTML



# Bloques fundamentales. Rutas



Componentes      Directivas      Servicios      Módulos      Pipes

Rutas

Permiten cambiar páginas (components que usualmente cubren todo el punto de vista)

- Separar lógica
- Control de acceso y autorización
- Estrategias de renderizado



# Bloques fundamentales. Directivas



- Relacionados con atributos, cambios de clases, cambios de estilo...



# Bloques fundamentales. Servicios



# Bloques fundamentales. Módulos

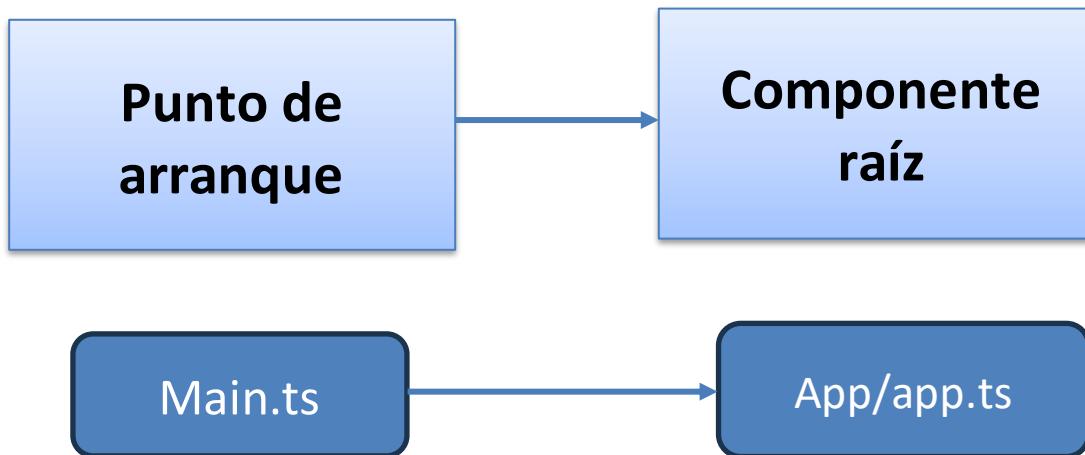


# Bloques fundamentales. Módulos



- Por ejemplo si tienes un número y quieres que aparezca en moneda, pues le pasas un pipe.

# Arranque de nuestra aplicación



# Arranque de nuestra aplicación

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { App } from './app/app';

bootstrapApplication(App, appConfig)
  .catch((err) => console.error(err));
```





# Main.ts

- Este es un archivo que nos va a servir para decirle a Angular cómo queremos que corra la aplicación.
- El componente raíz es App y tiene un appConfig, que es un archivo de configuración y nos permite definir cómo corre la aplicación.



# App.ts

- No es más que una clase con un decorador que transforma mi clase a un componente.

```
import { Component, signal } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.html'
})
export class App {
```

Decorador



# Decoradores

- Si tenemos un decorador aplicado sobre una clase le estamos dando una funcionalidad distinta
- Uso:
  - Antes de definir la clase mencionamos el decorador con @
  - Dicho decorador puede incluir parámetros

```
@DecoradorClase  
class miClase{
```

```
@DecoradorClase(parámetros)  
class miClase{
```

Utilizaremos decoradores ya definidos



## Tarea 1:

- Cambia el nombre mostrado y pon el tuyo
- Muestra tu apellido en otra variable
- Cambia el logo



# Ejercicio 1a. Primer componente desde cero

- Creamos un proyecto “Contador”
- Borrar pruebas y el css
- Creamos carpeta pages
- Dentro de pages creamos fichero /counter/counter.ts
- Creamos la clase Counter
- Establecemos el decorador que la cataloga como un componente
- Definimos una ruta para ese componente (por ejemplo, la raíz).
- Creamos una variable en la clase counter inicializada a 10
- Creamos un método que incremente en un valor que se le pasa por parámetro el contador.
- Creamos dentro de la clase una template y mostramos el contador en html y creamos 3 botones: sumar 1, restar 1 y resetear que devuelva el contador a su valor inicial.
- Una vez que se muestran los campos extraer el código a un fichero html.



# Señales

- Un señal es una variable reactiva que guarda un valor y avisa automáticamente a Angular para que actualice la vista cuando ese valor cambia.
- Lectura y escritura sencilla: se lee como una función (signal()) y se actualiza con .set() o .update().
- Renderizado automático: cuando cambia su valor, Angular vuelve a pintar solo lo necesario sin usar zone.js ni ChangeDetectorRef.

```
export class Counter {  
  counter = 10;  
  counterSignal = signal(10);
```



## Ejercicio 1b. Practicar lo aprendido hasta el momento

- Descarga el código y enunciado de Moodle y realiza el ejercicio propuesto.

# Señales computadas. Readonly signals

- Señal cuyo valor se calcula automáticamente a partir de otras señales, sin que tú tengas que actualizarla manualmente (no se puede hacer).

ts

```
name = signal("Ironman");
age = signal(45);

heroDescription = computed(()=> `${this.name()} - ${this.age()}`);
```

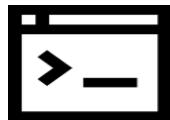
html

```
<td>Método:</td>
<dd>{{ heroDescription() }}</dd>
```

## Resumen:

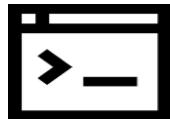
- Carpeta app contiene todo lo que se usa en la aplicación
- Pages para componentes que ocupan toda la pantalla.
- Las páginas se referencian en routes.ts
- App.html contiene el router-outlet, que se encarga de mostrar un componente particular en función de la ruta en la que nos encontramos.
- Index.html, toda nuestra aplicación está montada en app-root
- Señales es una forma de cambiar el estado de la aplicación.

# Crear componentes



```
ng generate component navbar
```

II



```
ng g c navbar
```

# RouterLink

Es una directiva del router de Angular que sirve para navegar entre rutas de la aplicación desde el HTML sin recargar la página.

Funciona como un enlace (href) pero gestionado por Angular.



```
<nav>
  <a routerLink="/">Contador</a>
  <a routerLink="/hero">Heroe</a>
</nav>
```



```
@Component({
  selector: 'app-navbar',
  imports: [RouterLink],
  templateUrl: './navbar.html',
  styleUrls: ['./navbar.scss'],
})
export class Navbar {
```

# Actividad

Crear un componente navbar con dos enlaces a la página contador y héroe.

**Nota 1:** Todos los componentes se colocarán en la ruta:

*app/components*

**Nota 2:** aplica este css:

```
nav {  
    display: flex;  
    justify-content: space-around;  
    background-color: #212121;  
    color: white;  
    padding: 10px;  
}  
  
nav a {  
    color: white;  
    text-decoration: none;  
}
```

# Rutas por defecto

Podemos establecer una ruta a la que redirigiremos el flujo de navegación en caso de no esté definida esa ruta:

```
{
  path: 'ruta1',
  component: MiComponente,
},
{
  path: '**',
  redirectTo: ''
}
```

app.routes.ts

# RouterLinkActive

- **routerLinkActive** es una directiva de Angular que te permite aplicar una clase CSS automáticamente a un elemento cuando la ruta asociada al routerLink está activa.
- Sirve para marcar visualmente **qué página está actualmente seleccionada** en un menú o barra de navegación.

```
.active {  
  font-weight: bold;  
  color: red;  
}
```

CSS

```
<a routerLink="/"   
 routerLinkActive="active">Contador</a>
```

html

- Si queremos que afecte solo al path exacto y no a los que contenga ese path :

```
<a routerLink="/"   
 routerLinkActive="active"   
 [routerLinkActiveOptions]="{exact:true}">  
 Contador</a>
```

html

# Tarea

- Crear componente dragonball
- Crear ruta para componente
- Agregarlo a navbar
- Crear un interface Personaje con campos id, name y power.



# Directivas

UD2: Desarrollo de aplicaciones basado en componentes con Angular

# Plantillas y vistas

- **Plantilla.** Es un archivo HTML donde mediante sintaxis Angular podemos alterar el DOM de la aplicación
  - Esto se hace mediante unas instrucciones especiales llamadas **directivas**
  - **Las plantillas mezclan HTML y directivas**
- **Vista.** Es lo que finalmente ve el navegador



# Algunas directivas comunes

Directiva	Descripción
for	Interacción con una lista
If	Muestra/oculta una etiqueta
(evento)	Definición de evento
{{propiedad}} [objecto]	Permiten enlazar con los datos a mostrar en el DOM
<app-usuario>	Selector de otro componente

DATA  
BINDING



# Directiva For

- Trabaja recorriendo un array, generando tantas etiquetas como elementos tiene el listado

```
public entradas:any[]=[  
    {titulo:"Entrada 1",resumen:"Contenido de la entrada 1"},  
    {titulo:"Entrada 2",resumen:"Contenido de la entrada 2"},  
    {titulo:"Entrada 3",resumen:"Contenido de la entrada 3"}  
];
```



COMPONENTE



```
<h2>Listado</h2>  
<p> Listado de usuarios</p>  
<ul>  
    @for (ingredient of ingredientList; track ingredient.name) {  
        <li>{{ ingredient.quantity }} - {{ ingredient.name }}</li>  
    }  
</ul>
```

PLANTILLA

# If

- Muestra/oculta en función del valor de un atributo

```
@if (isAdmin) {  
    <button>Borrar</button>  
}
```



El componente solo se muestra el botón si se cumple condición

# Tarea

- Seguimos con el componente dragonball
- Crear un interface Personaje con campos id, name y power.

```
interface Personaje {  
    id: number;  
    name: string;  
    power: number  
}
```



- Crear una señal personajes con un array de 3 personajes.
- Pintar en html los personajes usando un for
- Pintar solo los personajes que tengan un poder > 500 usando if

Ejercicio repaso:

Descargar ejercicio desde Moodle.