Bases Angular

Centro público integrado de formación profesional

Nuevo (desglose IES Campanillas)

TS – Temas puntuales de la sección

¿Qué veremos en esta sección?

Este es un breve listado de los temas fundamentales:

1. Crear proyectos de Angular (para versiones diferentes)
2. Explicar cada archivo y directorio de un proyecto
3. Componentes
4. Módulos
5. One way data binding
6. Uso del AngularCLI - Angular Command Line Interface
7. Directivas creadas por Angular
8. ngIf y ngIf-else

TS – Nuevo apartado

Introducción

TS – Introducción

- **Consideraciones iniciales**

Lo primero, la cantidad tan grande, tan enorme de **archivos**.

No es necesario memorizar que hace cada uno, solo tener una idea de su utilidad.

Ejemplo: angular.json

Solo tenemos que saber **dónde está y su propósito**

En Angular todo, absolutamente todo son **Clases** (Class) y estas tienen sus **Decoradores** que nos permitirán transformarlas a Componentes, Servicio, Módulo

Directiva nglf, nos permite mostrar/ocultar elementos HTML, en realidad, literalmente lo destruye.

Directiva ngFor, nos permite duplicar elementos HTML.

TS – Introducción

- **Consideraciones iniciales**

Un componente de angular es una archivo.ts que cuenta con tres partes.

El principal es el **.ts**, que puede tener

Un **.**html, o bien in-line, dentro del .ts, o bien a parte .html (¿cuándo lo sacamos fuera?) cuando tengo más de **4 líneas**, esto es, queremos el **código disociado**.

Luego están los **.css** (1 o más, tantos como se necesiten) están **encapsulados** y solo se aplican al .ts

Y por último los **.spec**, todo lo relacionado con la **comprobación y testing** de componentes.

TS – Nuevo apartado

Exposición de Angular

TS – Exposición de Angular

- **Consideraciones iniciales**

Todo comienzo tiene un origen, este es: **angular.io**

Angular es multiplataforma. Podemos desarrollar para: web, móvil, escritorio (Windows, Linux, Mac…)…

- Web, con Angular o Angular Universal (Server-Side Rendering)

Toda la carga de proceso está en el servidor. El cliente solo muestra el resultado.

- Movil, NativeScript que compila directamente para Swift o Java, o también Movil Web, haciendo uso del código de Angular o de código de terceros (Ionic).
- Escritorio, con Electron.

Angular es un Framework (**marco de trabajo estandarizado**).

Viene con todo lo que necesitas para trabajar.

Es modular.

Google es quien le da mantenimiento a Angular.

TS – Exposición de Angular

- **Consideraciones iniciales**

Angular tiene **5 componentes** fundamentales:

- **Componentes**

HTML + (Clase TypeScript, que tiene un decorador)

- **Servicios**, lugares para centralizar la información

Componente -> Botón -> Servicios

- **Directivas**, tres tipos:
  - Componentes, igual que un componente pero con cógido HTML reutilizable y funcional
  - Estructurales, modifican el DOM
  - Atributos, cambian la apariencia de un elemento, componente o directiva
- **Rutas**, muestran diferentes componentes basados en la URL
- **Módulos**, perminen agrupar todo lo anterior y además módulos.

TS – Exposición de Angular

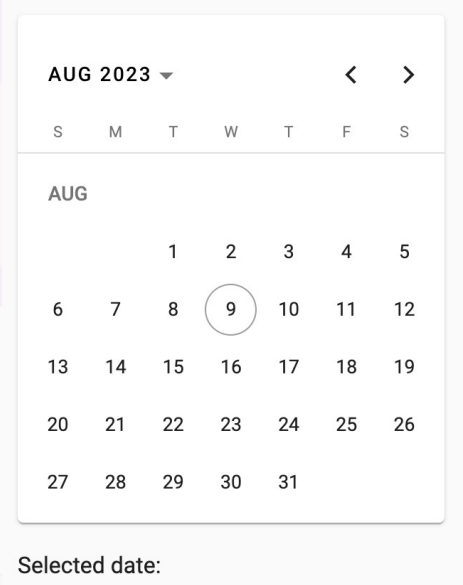
- **Consideraciones iniciales**

Ejemplo **Calendario**.

No hay necesidad de crear un Calendario ya que hay componentes que ya están hechos, solo hay que descargarlos e integrarlos en nuestra aplicación.

Ejemplos de Librerías de componentes (comparamos con el mismo componente 'DataPicker'):

- Angular Material



TS – Exposición de Angular

- **Consideraciones iniciales**

- NG-Bootstrap

Vacations

from **Aug 9, 2023** to **Aug 24, 2023**

Aug 2023

August 2023

	Mo	Tu	We	Th	Fr	Sa	Su
31		1	2	3	4	5	6
32	7	8	9	10	11	12	13
33	14	15	16	17	18	19	20
34	21	22	23	24	25	26	27
35	28	29	30	31			
36							

September 2023

	Mo	Tu	We	Th	Fr	Sa	Su
35					1	2	3
36	4	5	6	7	8	9	10
37	11	12	13	14	15	16	17
38	18	19	20	21	22	23	24
39	25	26	27	28	29	30	
40							

- Clarity (de pago?)

02/13/2018

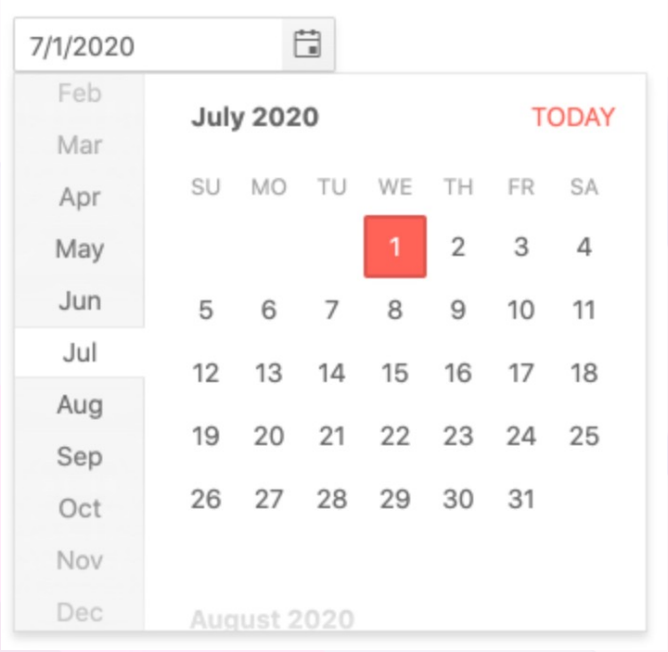
Feb 2018

S	M	T	W	T	F	S
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	1	2	3
4	5	6	7	8	9	10

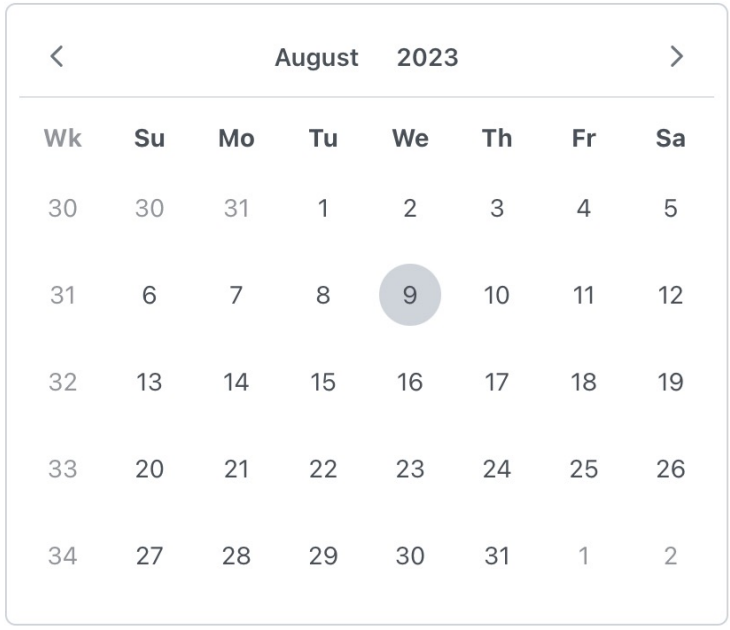
TS – Exposición de Angular

- **Consideraciones iniciales**

- Kendo UI (de pago?)



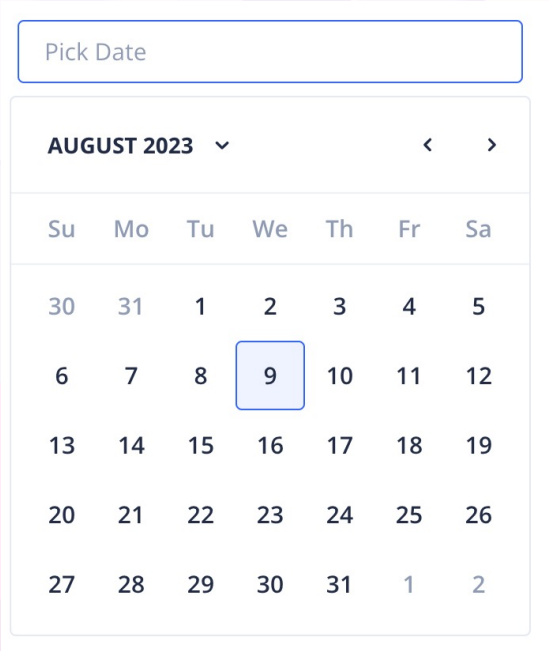
- PrimeNG



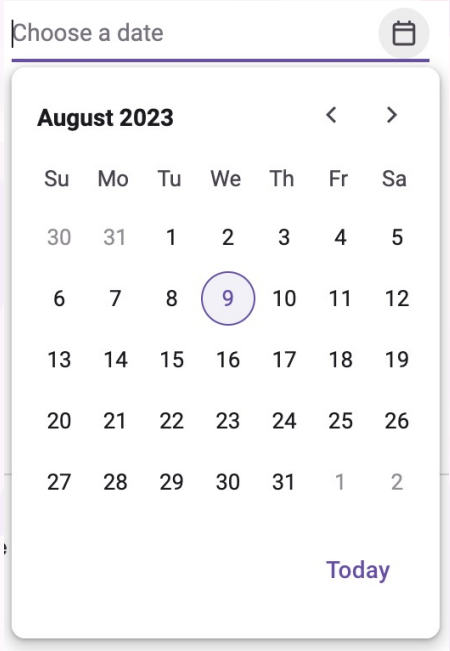
TS – Exposición de Angular

- **Consideraciones iniciales**

- Nebular



- Syncfusion



TS – Nuevo apartado

Nuestro primer proyecto en Angular

TS – Nuestro primer proyecto en Angular

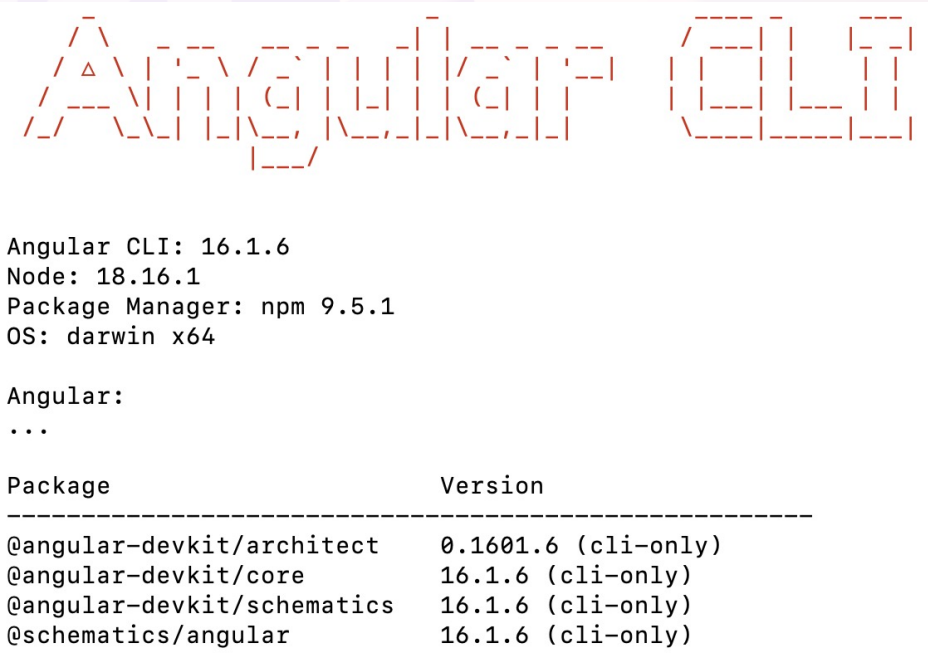
- **¿Cómo se crea un proyecto desde cero?**

Paso 1 Creamos una carpeta donde guardar nuestro proyectos.

mkdir 11-ANGULAR (cd 11-ANGULAR)

Paso 2 Comprobamos que tenemos instalado AngularCLI

ng version



```
Angular CLI: 16.1.6
Node: 18.16.1
Package Manager: npm 9.5.1
OS: darwin x64

Angular:
...

Package                         Version
---------------------------------------------------------
@angular-devkit/architect       0.1601.6 (cli-only)
@angular-devkit/core            16.1.6 (cli-only)
@angular-devkit/schematics      16.1.6 (cli-only)
@schematics/angular             16.1.6 (cli-only)
```

TS – Nuestro primer proyecto en Angular

- **¿Cómo se crea un proyecto desde cero?**

Sino tienes Angular instalado. Dos opciones:

Opción 1

Instalamos la última versión y programamos para la versión 19.

Documentación oficial de Angular CLI

Ejecutar el siguiente comando como **administrador**

sudo npm install -g @angular/cli

TS – Nuestro primer proyecto en Angular

- **¿Cómo se crea un proyecto desde cero?**

Paso 3 Creamos el proyecto desde 0. Dos formas:

- # npm init @angular 02-angular-bases
- # **ng new 02-angular-bases**

El proceso tardará un tiempo ya que está creando todo el esqueleto del proyecto.

Paso 4 Abrimos VSC y vamos a la paleta de comandos:

- Menú -> Ver -> Paleta de comandos…
- shift+cmd+p, **code** -> comando Shell: instalar el **comando code** en PATH

Conseguimos abrir VSC desde la carpeta del proyecto escribiendo: **code .**

Paso 5 Levantamos el servidor de Angular

Dentro de la carpeta del proyecto que acabamos de crear usamos el comando:

ng serve (estará sirviendo en: localhost:4200)

TS – Nuestro primer proyecto en Angular

- **¿Cómo se crea un proyecto desde cero?**

Opción 2

Aprendemos a instalar diferentes versiones: 16, 17, 18…

Paso 3a Creamos una carpeta para proyectos con versión Angular 16

mkdir ANGULAR16 (cd ANGULAR16)

Dentro de la esta carpeta crea los proyectos de angular16 tal y como se indica:

npm install @angular/cli@16.2.14

npx ng new 11a-angular-bases (cd 11a-angular-bases)

ng serve

Paso 4 Abrimos VSC y vamos a la paleta de comandos:

- Menú -> Ver -> Paleta de comandos…
- shift+cmd+p, **code** -> comando Shell: instalar el **comando code** en PATH

Conseguimos abrir VSC desde la carpeta del proyecto escribiendo: **code .**

TS – Nuestro primer proyecto en Angular

- **¿Cómo se crea un proyecto desde cero?**

Paso 3b Creamos una carpeta para proyectos con versión Angular 17

mkdir ANGULAR16 (cd ANGULAR16)

Dentro de la esta carpeta crea los proyectos de angular16 tal y como se indica:

npm install @angular/cli@17.3.8

npx ng new 11a-angular-bases

ng serve

Paso 3c Creamos una carpeta para proyectos con versión Angular 18

mkdir ANGULAR16 (cd ANGULAR16)

Dentro de la esta carpeta crea los proyectos de angular16 tal y como se indica:

npm install @angular/cli@18.0.4

npx ng new 11a-angular-bases

ng serve

TS – Nuestro primer proyecto en Angular

- **¿Cómo se crea un proyecto desde cero?**

NVM (Node Version Manager)

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash

wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash

```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
```

Fuente: https://github.com/nvm-sh/nvm

TS – Nuestro primer proyecto en Angular

- **¿Cómo se crea un proyecto desde cero?**

Nota

Resolviendo conflictos entre las versiones de **Angular y Node.js**

Angular	Node.js	TypeScript	RxJS
16.1.x || 16.2.x	^16.14.0 || ^18.10.0	>=4.9.3 <5.2.0	^6.5.3 || ^7.4.0
19.0.x	^18.19.1 || ^20.11.1 || ^22.0.0	>=5.5.0 <5.7.0	^6.5.3 || ^7.4.0
18.1.x || 18.2.x	^18.19.1 || ^20.11.1 || ^22.0.0	>=5.4.0 <5.6.0	^6.5.3 || ^7.4.0
18.0.x	^18.19.1 || ^20.11.1 || ^22.0.0	>=5.4.0 <5.5.0	^6.5.3 || ^7.4.0
17.3.x	^18.13.0 || ^20.9.0	>=5.2.0 <5.5.0	^6.5.3 || ^7.4.0
17.1.x || 17.2.x	^18.13.0 || ^20.9.0	>=5.2.0 <5.4.0	^6.5.3 || ^7.4.0
17.0.x	^18.13.0 || ^20.9.0	>=5.2.0 <5.3.0	^6.5.3 || ^7.4.0

TS – Nuevo apartado

Archivos y directorios del proyecto

TS – Archivos y directorios del proyecto

- Conjunto de **archivos**

angular.json, configuración de Angular para la ejecución del proyecto.

Ejemplo: archivos; index.html, main.ts (bundle), styles.css, etc…

Lo normal es hacer pequeños cambios, sin proceden, y no tocarlo más.

package-lock.json, como construir los módulos de Node

package.json, propio del proyecto Node, que es sobre el que se asienta Angular

Ejemplo: definimos comandos para arrancar el proyecto, dependencias, etc…

Las **dependencias de Angular** serán de producción, paquetes necesarios para que 'ng build' compile el proyecto tras hacer 'tree shaking'

Las dependencias de desarrollo son necesarias para programar nuestra aplicación.

Ejemplo: devkit, cli, compiler, jasmine (testing), etc…

TS – Archivos y directorios del proyecto

- Conjunto de **archivos**

README.md, dónde explicamos como funciona nuestro proyecto (Markdown)

shift+cmd+p, **markdown** -> Markdown: Refresh Preview

tsconfig.app.json, archivos que tenemos que revisar, archivo de salida, etc…

tsconfig.json, recomendaciones para trabajar dentro de Angular. Ej. Strict

Todos los desarrolladores de Angular siguen las directrices de este archivo.

tsconfig.spec,json, solo se enfoca en el testing

Opcionales:

.editorconfig, relacionado con la extensión, 'EditorConfig for VS Code'.

Son configuraciones del editor, no afectan al proyecto Angular.

.gitignore, archivo propio de Git.

Los archivos que esten en este archivo, serán ignorados al subir el proyecto.

TS – Archivos y directorios del proyecto

- Conjunto de **directorios**

.angular, no hay que tocarlo (de hecho está oculto)

Ayuda a Angular a que los cambios en la aplicación se apliquen.

.vscode, es algo propio de VSCode y no se sube a github.

node_modules, tampoco se sube a github

Esta carpeta se creo cuando lanzamos el comando: #npm install

src, nuestro proyecto:

- esta es nuestra aplicación,
- esto es lo que debemos conservar,
- esto es lo que nosotros programamos…

Mas info: Workspace and project file structure

TS – Archivos y directorios del proyecto

- **Directorio 'src'**

favicon.ico, el icono que se mostrará en la pestaña del navegador

index.html, un archivo simple con la referencia al favicon y con:

> **<app-root></app-root>**, que es el componente que hace referencia a nuestro proyecto en Angular

main.ts, punto de entrada a nuestra aplicación en Angular.

Usamos platformBrowserDynamic()… porque nuestra aplicación es web.

Podemos desarrollar para otras plataformas: Angular Universal, Ionic, Electron, etc…

Usamos …bootstrapModule(AppModule) como módulo principal

style.css, estilos globales.

Recordemos que cada componente tendrá su propio style.css

TS – Archivos y directorios del proyecto

- **Directorio ‘src’**

app, dónde empezaremos a construir toda la lógica: componentes, servicios, etc…
Mención especial para: ‘**app.component.ts**’ en el cuál encontramos: **’app-root**’
Este es el componente principal, primer componente del cual depende el resto.
app.module.ts, módulo principal, es muy importante y cuidado al usarlo.
Un cambio haría que la Aplicación dejara de funcionar.
Tiende a crecer a lo largo del desarrollo.
assets, recursos del proyecto
.gitkeep, dónde tengamos este archivo, git lo considerará

En resumen, a las empresas les encanta Angular:
Funcionalidad, Consistencia, Escalable, Fiable…

TS – Nuevo apartado

App Component

TS – App Component

- **¿Qué necesitamos?**
- Nos vamos a la carpeta: **11a-angular-bases**
- Abrimos el proyecto en VSC: **# code .**
- Lazamos el comando: **# ng server**
- En VSC abrimos el archivo: '**app.component.html**'
  Path: 11a-angular-bases -> src -> app
- También, abrimos el archivo: '**app.component.ts**'
  Este archivo es el importante, de el dependen el resto: .html, .css, etc…

TS – App Component

- **¿Qué necesitamos y que observamos?**

app.component.html Lo borramos todo y escribimos el siguiente código.

```
<h1>Hola Mundo</h1>
<hr>
<p>Bienvenidos a Angular</p>
<hr>
```

Todo ocurre

app.component.ts Cambiamos la línea de '**title**'.

```
export class AppComponent {
  public title: string = 'Mi primera app de Angular';
}
```

No ocurre nada, pero lo hemos definido acorde a los principios de TS y…

TS – App Component

- **¿Qué necesitamos y que observamos?**

app.component.html Añadimos las líneas:

<h1>{{**title**}}</h1> <!-- ctrl+(space), aparece 'title' -->

<hr>

Hay un bundle (relación), entre .html y .ts

Las propiedades de la clase AppComponent son accesibles en el .html

Este bundle se da solo en el mismo componente.

templateUrl: './app.component.html', Solo es posible **un .html**

styleUrls: ['./app.component.css'] Son posibles **varios estilos**

TS – App Component

- **¿Qué necesitamos y que observamos?**

app.component.ts Añadimos la línea:

```
export class AppComponent {
  …
  public counter: number = 10;
}
```

app.component.html Añadimos la línea:

```
<h3>Counter: {{counter}}</h3>
<hr>
```

Por último, comprobamos que tenemos acceso a las Angular DevTools en el navegador de Chrome.

TS – Nuevo apartado

Contador

TS – Contador

- **¿Qué necesitamos y que observamos?**

Vamos a crear un botón para incrementa el valor del contador en 1.

app.component.html Añadimos un método a la clase AppComponent

```
increaseBy(): void {
  this.counter += 1;
}
```

app.component.html Creamos el botón y el evento ‘click’:

```
<button (click)="increaseBy()">+1</button>
<hr>
```

TS – Contador

- **TAREA**

Crea un botón para decrementar el valor del contador en 1.

app.component.html Añadimos un método a la clase AppComponent

…

app.component.html Creamos el botón y el evento ‘click’:

…

TS – Contador

- **SOLUCIÓN**

app.component.html Añadimos un método a la clase AppComponent

```
decreaseBy(): void {
  this.counter -= 1;
}
```

app.component.html Creamos el botón y el evento ‘click’:

```
<button (click)=”decreaseBy()">-1</button>
<hr>
```

TS – Contador

- **TAREA**

Añadir la posibilidad de incrementar o decrementar según un valor indicado.

app.component.html Modificamos el método de la clase AppComponent

…

app.component.html Modificamos la llamada del evento ‘click’

…

TS – Contador

- **SOLUCIÓN**

app.component.html Modificamos el método de la clase AppComponent

```
increaseBy(value: number): void {
  this.counter += value;
}
decreaseBy(value: number): void {
  this.counter -= value;
}
```

app.component.html Modificamos la llamada del evento ‘click’

```
<button (click)="increaseBy(1)">+1</button>
<button (click)="decreaseBy(1)">-1</button>
```

TS – Contador

- **TAREA**

Crea un botón para reiniciar el valor del contador a 10.

app.component.html Añadimos un método a la clase AppComponent

...

app.component.html Creamos el botón:

...

TS – Contador

- **SOLUCIÓN**

app.component.html Añadimos un método a la clase AppComponent

```
resetCounter():void {
  this.counter = 10;
}
```

app.component.html Creamos el botón:

```
<button (click)="resetCounter()">Reset</button>
```

TS – Nuevo apartado

Contador Component

TS – Contador Component

- **¿Qué necesitamos?**
- Dentro de: **02-angular-bases/src/app**, creamos la carpeta: '**counter**'
- Creamos el archivo: '**counter.component.ts**'

Escribimos el siguiente código:

siguiente diapositiva…

TS – Contador Component

- **¿Qué necesitamos?**

counter.component.ts

```
import { Component } from "@angular/core";
@Component({
  selector: 'app-counter’,
  template: '<h5>Hola Counter</h5>'
})
export class CounterComponent {

}
```

app.component.html

```
<app-counter></app-counter>
```

TS – Contador Component

- **¿Qué observamos?**

Creamos la clase '**CounterComponent**'

Esta clase se transformará en un componente al usar el decorador: **@Component**

Este **decorador** lo importaremos desde '**Component**'

Definimos el '**selector**' (como voy a referenciar al componente desde el .html) y el '**template**' (el contenido del componente) del decorador:

selector: 'app-counter',

template: '<h5>Hola Counter</h5>'

Si intentamos usar el componente 'app-counter' en el .html nos da un error.

El error nos indica que el componente todavía no está integrado dentro de ningún módulo.

TS – Contador Component

- **¿Qué observamos?**

Para usar el componente tenemos que integrarlo/importarlo en el módulo: '**app.module.ts**'

```
import { CounterComponent } from './counter/counter.component';
@NgModule({
  declarations: [
    AppComponent,
    CounterComponent
  ],
```

Nada más escribimos el nombre del componente, 'CounterComponent' se introduce automáticamente la línea del '**import…**'

TS – Nuevo apartado

Funcionalidad del contador (Angular Snippets)

TS – Funcionalidad del contador. Snippets

- **¿Qué necesitamos?**
- Comprobamos que tenemos instalada la extensión de VSC: '**Angular Snippets**'
- Recordemos todo el trabajo anterior para crear nuestro componente.
- Dentro de: **02-angular-bases/src/app**, creamos la carpeta: '**counterSnippet**'
- Creamos el archivo: '**counterSnippet.component.ts**'

Dentro del archivo escribimos el siguiente código:

siguiente diapositiva…

TS – Funcionalidad del contador. Snippets

- **¿Qué necesitamos?**

counterSnippet.component.ts

Escribimos: **a-**… y vemos como aparecen todos los componentes disponibles.

Elejimos: **a-component** y automáticamente nos escribe:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'selector-name',
  templateUrl: 'name.component.html'
})
export class NameComponent implements OnInit {
  constructor() { }
  ngOnInit() { }
}
```

TS – Funcionalidad del contador. Snippets

- **¿Qué necesitamos?**

counterSnippet.component.ts

Modificamos lo siguiente:

```
import { Component } from '@angular/core';
@Component({
  selector: app-counterSnippet’,
  templateUrl: '<h5>Hola Counter</h5>'
})
export class CounterSnippetComponent{
}
```

app.component.html

```
<app-counterSnippet></app-counterSnippet>
```

Ahora sí, podemos hacer uso del nuevo componente

TS – Nuevo apartado

Funcionalidad del contador (Standalone)

TS – Funcionalidad del contador. Standalone

- **¿Qué necesitamos?**
- Dentro de: **02-angular-bases/src/app**, creamos la carpeta: '**counterStandalone**'
- Creamos el archivo: '**counterStandalone.component.ts**'

Con todo lo aprendido hasta ahora, ¿sabrías creas el componente?

No te preocupes, es esta Tarea te proporcionaré código y las pistas para que sepas como construirlo.

Dentro del archivo: '**counterStandalone.component.ts**' escribimos el código:

siguiente diapositiva…

TS – Funcionalidad del contador. Standalone

- **TAREA** Crea el componente a partir del siguiente **código**:

counterStandalone.component.ts

Escribimos: **a-**… y vemos como aparecen todos los componentes disponibles.

Elejimos: **a-component** y modificamos el esqueleto que nos propone **Angular Snippets**:

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

@Component({
  selector: 'app-counter-standalone',
  template:
  … ¿Qué deberíamos incluir? ¿quizás el código HTML que ya tenemos en el .html?
})
export class CounterStandaloneComponent {
 … ¿Qué deberíamos incluir? ¿quizás el código TS que ya tenemos en el app.component.ts?
}
```

TS – Funcionalidad del contador. Standalone

- **TAREA**

app.module.ts

```
import { … } from '…';

@Component({
  declarations: [
  …
],
```

¿Qué deberíamos incluir? Observa el código de los componentes anteriores.

app.component.html

Haz uso del componente.

TS – Funcionalidad del contador. Standalone

- **SOLUCIÓN**

counterStandalone.component.ts

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

@Component({
  selector: 'app-counter-standalone',
  template:`
      <h5>Counter: {{counter}}</h5>
      <button (click)="increaseBy(1)">+1</button>
      <button (click)="resetCounter()">Reset</button>
      <button (click)="decreaseBy(1)">-1</button>
      `
})
export class CounterStandaloneComponent {
  public title: string = 'Mi primera app de Angular';
  public counter: number = 10;
  increaseBy(value: number): void {
    this.counter += value;
  }
  decreaseBy(value: number): void {
    this.counter -= value;
  }
  resetCounter(): void {
    this.counter = 10;
  }
}
```

app.module.ts

```
import { CounterStandaloneComponent } from './03-counterStandalone/counterStandalone.component';

@Component({
  declarations: [ …
        CounterStandaloneComponent
        …],
```

app.module.ts

```
<app-counter-standalone></app-counter-standalone>
```

TS – Nuevo apartado

Componente Hero y directorios

TS – Componente Hero y directorios

- **¿Qué necesitamos?**
- Creamos el proyecto: **02-angular-bases-heroes**
  - # ng new 02-angular-bases-héroes
- Lanzamos el **servidor**
  - Si tenemos algún servidor activo, o bien lo cerramos, o bien cambiamos el puerto.
- Dentro de **02-angular-bases/src/app**, creamos la carpeta '**heroes**' y dentro, las carpetas: '**hero**' y '**list**'
- **Creamos** el **componente** desde la terminal de VSC:
  - Menú -> Terminal (estamos dentro de la carpeta del proyecto)
  - # **ng g c heroes/hero**
- Con esto tenemos toda la **estructura de un componente** dentro de '**hero**':
  - .css (estilos encapsulados), .html, el selector de .ts
- **Cambiamos** el nombre del **selector** en '**hero.component.ts**':

  @Component({

  selector: **'app-heroes-list**', …
- En **app.component.html** comprobamos que funciona:
  - <app-heroes-hero></app-heroes-hero>

TS – Componente Hero y directorios

- **TAREA**

Creamos el **componente** '**list**'

Cambiamos el nombre del **selector** por: '**app-heroes-list**'

Comprobamos que funciona haciendo **uso** del mismo en '**app.component.html**'

TS – Componente Hero y directorios

- **SOLUCIÓN**

Creamos el componente desde la terminal de VSC:

Menú -> Terminal (estamos dentro de la carpeta del proyecto)

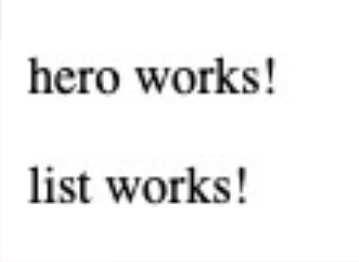
 # ng g c heroes/list

Cambiamos el nombre del **selector** en '**list.component.ts**':

```
@Component({
  selector: 'app-heroes-list', …
```

En **app.component.html** comprobamos que funciona:

```
<app-heroes-list></app-heroes-list>
```



TS – Nuevo apartado

Interpolación, estructura HTML y estilos

TS – Interpolación, estructura HTML y estilos

- **¿Qué necesitamos?**

Vamos a trabajar sobre el componente '**hero**'

Vamos a añadir código al: .html (aplicaremos **interpolación**), .css (haremos uso de **bootstrap**), .ts (modificaremos la clase **HeroComponent**)

Dentro de los archivos:

- hero.component.html
- hero.component.css
- hero.component.ts

escribimos el siguiente código:

siguiente diapositiva…

TS – Interpolación, estructura HTML y estilos

- **¿Qué necesitamos?**

hero.component.html	
Spiderman <dl> <td>Nombre:</td> <dd>nombre</dd> <td>Edad:</td> <dd>edad</dd> <td>Descripción:</td> <dd>descripción</dd> <td>Capitalizado:</td> <dd>nombre capitalizado</dd> </dl>	<button class="btn btn-primary mx-2"> Cambiar nombre </button> <button class="btn btn-primary"> Cambiar edad </button>

TS – Interpolación, estructura HTML y estilos

- **¿Qué necesitamos?**

hero.component.css	hero.component.ts

h1 {
  font-size: 50px;
}
 	
dd {
  font-size: 2rem;
  font-weight: bold;
}
	public name: string = 'ironman';
	public age: number = 45;

TS – Interpolación, estructura HTML y estilos

- **TAREA**

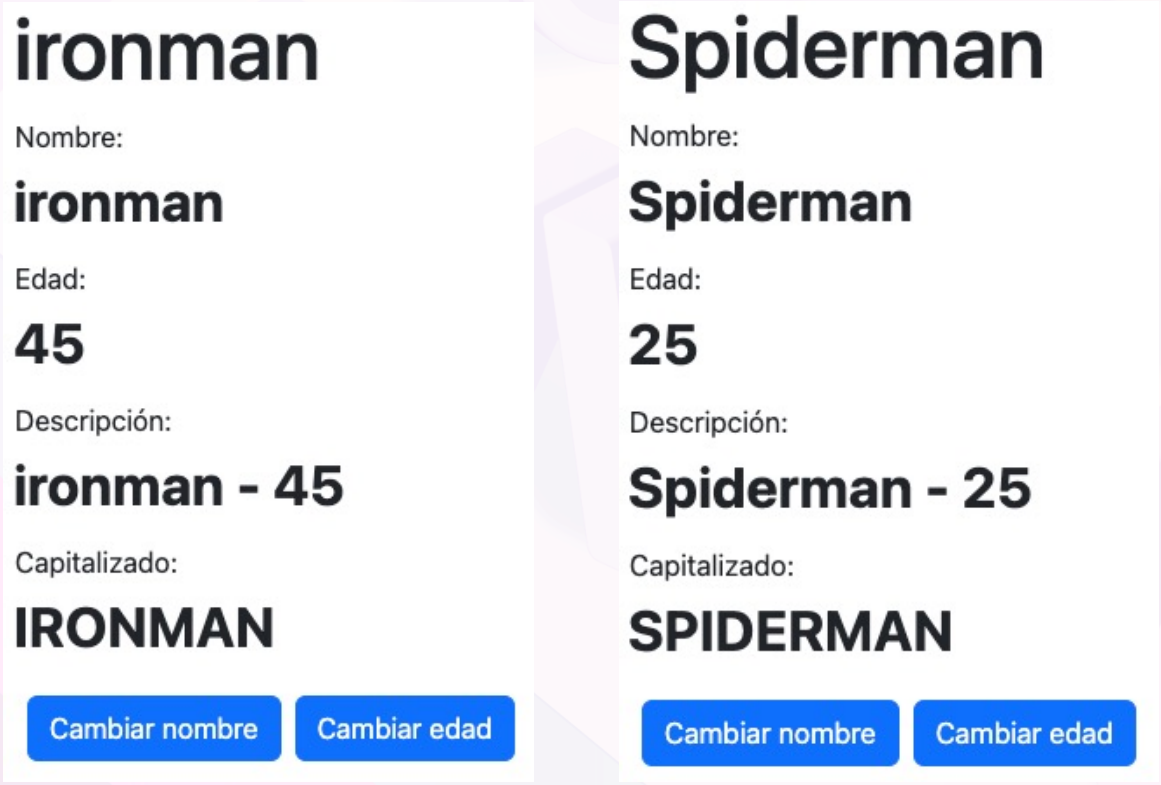
Modificar el contenido de '**hero.component.html**' aplicando **interpolación** para que:

- Se **muestre** el valor de **name** en la etiqueta '**h1**'
- Se **muestren** los valores de **name** y **age**, en las etiquetas 'dd' **nombre** y 'dd' **edad** respectivamente.
- Crear un **método** que muestre la **descripción** del héroe: '**name – age**'
- Crear un **método getter** que **capitalice** el nombre el héroe: **ABCDEF**
- Crear **dos métodos**, uno para el **nombre** y otro para la **edad**, de tal modo que cuando se pulsen los **botones**, los **valores** de nombre y edad **cambien**. Ejemplo: 'Spiderman', 25

La única diferencia entre este ejercicio y los que hicimos con contadores es que el código HTML se encuentra en un archivo aparte, por lo demás todo es igual.

TS – Interpolación, estructura HTML y estilos

- **VISTA PREVIA**



TS – Interpolación, estructura HTML y estilos

- **SOLUCIÓN**

hero.component.html	
{{name}} <dl> <td>Nombre:</td> <dd>{{name}}</dd> <td>Edad:</td> <dd>{{age}}</dd> <td>Método:</td> <dd>algún método</dd> <td>Capitalizado:</td> <dd>nombre capitalizado</dd> </dl>	<button class="btn btn-primary mx-2" (click)="cambiarNombre()"> Cambiar nombre </button> <button class="btn btn-primary" (click)="cambiarEdad()"> Cambiar edad </button>

TS – Interpolación, estructura HTML y estilos

- **SOLUCIÓN**

hero.component.ts	

public **name**: string = 'ironman';
public **age**: number = 45;
 	
cambiarNombre(): void {
this.name = "Jota";
}
 	
cambiarEdad(): void {
this.age = 47;
}
	getHeroDescription(): string {
	return `${this.name} - ${this.age}`;
	}
	get **capitalizedName**(): string {
	return this.name.toUpperCase();
	}

TS – Nuevo apartado

ngIf

TS – nglf

- **¿Qué necesitamos?**

HTML es un **lenguaje**… de **marcas**, pero a pesar de ser un lenguaje, no es capaz de tener sentencias condicionales que nos permita hacer una cosa u otra.

La **directiva nglf**, es un **atributo** que podemos agregar a los elementos HTML para **condicionar** (If) si dichas **marcas** deben **agregarse** a la página HTML.

Vamos a **eliminar el botón**, que cambia el valor del nombre, que se creó en el apartado anterior, una vez que sea usado.

Dentro del archivo '**hero.component.html**' escribimos el siguiente código:

siguiente diapositiva…

TS – ngIf

- **¿Qué necesitamos?**

En la etiqueta '**button**' para cambiar el valor del '**nombre**', añadimos la directiva 'ngIF' y expresamos la condición que se debe de cumplir para que el botón (marca) no se considere en el DOM

hero.component.html

```
<button
  *ngIf="name !== 'Spiderman'"
  (click)="changeName()"
  class="btn btn-primary mx-2">
  Cambiar nombre
</button>
```

Si todo está correcto debería de desaparecer el botón 'Cambiar nombre'

TS – ngIf

- **TAREA**

Aplica la directiva ‘ngIF’ en la etiqueta ‘**button**’ para cambiar el valor de la ‘**edad**’

Además:

- crea un tercer botón: ‘Resetear’
- crea un método: ‘resetForm()’
- usa el método en el botón para restablecer el formulario.

Aunque pueda parecer lo contrario, todo lo que hemos visto hasta ahora te permitirá realizar la tarea.

¡Ánimo!

TS – nglf

- **SOLUCIÓN**

hero.component.html		hero.component.ts

<button
***nglf="age !== 25"**
(click)="changeAge()"
class="btn btn-primary">
Cambiar edad
</button>
	<button
	(click)="**resetForm()**"
	class="btn btn-primary mx-2">
	Resetear
	</button>
		resetForm(): void {
		this.name = 'ironman';
		this.age = 45;
		}

TS – Nuevo apartado

ngFor

TS – ngFor

- **¿Qué necesitamos?**

HTML es un **lenguaje**… de **marcas**, pero a pesar de ser un lenguaje, no es capaz de tener sentencias iterativas que nos permitan repetir elementos.

La **directiva ngFor**, es un **atributo** que podemos agregar a los elementos HTML para **iterar** (For) dichas **marcas** y **agregarlas** a la página HTML.

Vamos a crear los componentes de una lista de héroes.

Dentro del archivo '**list.component.html**' escribimos el siguiente código:

siguiente diapositiva…

TS – ngFor

- **¿Qué necesitamos?**

list.component.html

```
<h3>Listado de Héroes</h3>
<button
  (click)="removeLastHero()"
  class="btn btn-outline-danger">
  Borrar último héroe
</button>
<ul class="mt-2 list-group">
  <li *ngFor="let name of heroNames"
  class="list-group-item">{{name}}</li>
</ul>
<h3>Héroe borrado: <small class="text danger">...</small></h3>
<h5>No ha borrado nada.</h5>
```

list.component.ts

```
public heroNames: string[] = [
  'Spiderman',
  'Spiderwoman',
  'Hulk',
  'She Hulk',
  'Thor',
  'Wonderwoman'
];
```

TS – ngFor

- **TAREA**

Creamos un botón: ‘Restablecer Lista de Héroes‘

Ayuda:

- Crea el botón en el archivo .html
- Crea un método en la clase, en el archivo .ts
- Referencia la clase en el botón creado

TS – ngFor

- **SOLUCIÓN**

list.component.html	list.component.ts

…(a continuación del botón anterior)
<button
(click)="**resetHeroNames()**"
class="btn btn-outline-danger mx-2">
Restablecer Lista de Héroes
</button>
…
	resetHeroNames(): void {
	this.heroNames = ['Spiderman', 'Spiderwoman', 'Hulk', 'She Hulk', 'Thor', 'Wonderwoman'];
	}

TS – Nuevo apartado

ngTemplate y ngIf-else

TS – ngTemplate y nglf-else

- **¿Qué necesitamos?**

La **directiva nglf**, es un **atributo** que podemos agregar a los elementos HTML para **condicionar** (If) si dichas **marcas** deben **agregarse** a la página HTML, pero… ¿**existe** la posibilidad de un (**else**)?

Para eso tenemos 'ngTemplate'.

Controlar cuando **mostrar** el **heroe borrado**

Controlar cuando **mostrar** el **botón borrar**

Controlar cuando **mostrar** el **mensaje** '**No se ha borrado nada**'

Dentro del archivo '**list.component.html**' escribimos el siguiente código:

siguiente diapositiva…

TS – ngTemplate y nglf-else

- **¿Qué necesitamos?**

list.component.ts	list.component.ts

<button (botón para borrar)
***ngIf="heroNames.length > 0"**
…
</button>
<div *ngIf="**deletedHero**; else **nothingDeleted**">
<h3>Héroe borrado: <small class="text-danger">{{deletedHero}}</small></h3>
 	
<**ng-template #nothingDeleted**>
<h5>No ha borrado nada.</h5>
</ng-template>
	public deletedHero?: string;
	
	resetHeroNames(): void {
	…
	this.deletedHero = '';
	}

TS – ngTemplate y ngIf-else

- **¿Qué observamos?**

Manipulamos el DOM (creamos, borramos, … elementos) sin mayor problema.

Angular hace transparente toda la manipulación del DOM

HTML se vuelve más dinámico con las directivas ngIf-else y ngFor.

Por ahora, la creación y gestión de un componente se reduce a la relación entres los archivos:

- .html
- .css
- .ts

TS – Nuevo apartado

Módulos en Angular

TS – Módulos en Angular

- **¿Qué necesitamos?**

En el archivo '**app.module.ts**' vemos que un **módulo** es:

- una **clase**, con
- un **decorador**.

Los módulos **agrupan/encapsulan** funcionalidades.

Modularicemos nuestra aplicación empezando por '**counter**'

1.- **Eliminemos** las **dependencias e importaciones** en: '**app.module.ts**'

2.- Dentro de la carpeta '**counter**' creamos

Consultar la hoja de Atajos: Definición de Módulos.

TS – Módulos en Angular

- **¿Qué necesitamos?**

PASOS

1.- **Eliminemos** las **dependencias e importaciones** en: '**app.module.ts**'

~~import { CounterComponent } from './counter/counter.component';~~

```
@NgModule({
declarations: [
AppComponent,
~~CounterComponent,~~
HeroComponent,
ListComponent
],
```

TS – Módulos en Angular

- **¿Qué necesitamos?**

PASOS

2.- Dentro de la carpeta '**counter**' creamos la carpeta '**components**' y dentro de esta '**counter**'

Así, todos los componentes relacionados con '**counter**' están en el mismo lugar.

3.- Ahora copiamos los archivos del componente '**counter**':

- .css,
- .html,
- spec.ts, y
- ts

dentro de: components -> counter

NOTA: Si nos pregunta si queremos **actualizar** los archivos de Angular, decimos que **NO**.
Así lo hacemos **nosotros manualmente**.

TS – Módulos en Angular

- **¿Qué necesitamos?**

PASOS

4.- Dentro de la carpeta '**components**' creamos el archivo: '**counter.module.ts**'

La estructura de un Módulo es :

- @Decorador, y
  - Declaración de componentes y importaciones/exportaciones de los mismos, etc…
- Clase del módulo

Dentro del archivo '**counter.module.ts**' escribimos el siguiente código:

siguiente diapositiva…

TS – Módulos en Angular

- **¿Qué necesitamos?**

```
import { NgModule } from "@angular/core";
import { CounterComponent } from "./counter/counter.component";

@NgModule({
  declarations: [
    CounterComponent
  ],
  exports: [
    CounterComponent
  ]
})
export class CounterModule {
}
```

Escribe '**NgModule**' y pulsa '**Tab**' (tabulador), creará el '**import**'

Lo mismo ocurre al escribir '**CounterComponent**'

TS – Módulos en Angular

- **¿Qué necesitamos?**

5.- Ahora, integramos el módulo en Angular a través de '**app.module.ts**'

```
...
imports: [
  BrowserModule,
  CounterModule
],
...
```

Eso es todo y aunque no lo parezca, hemos ido un paso más allá en la estructuración de nuestro código.

TS – Módulos en Angular

- **TAREA**

Crear otro módulo llamado: '**heroes.modulo.ts**', que contenga los dos componentes:

- hero, y
- list

Espera que te ayudo

1.- Creamos la carpeta '**components**' dentro de '**heroes**'

2.- Movemos: '**hero**' y '**list**', dentro de la carpeta '**components**'

3.- Creamos el archivo '**heroes.module.ts**' dentro de '**components**'

Nos ayudamos del ya creado para '**counter.module.ts**'

4.- Importamos CommonModule, para usar las directivas: ngIf y ngFor

TS – Nuevo apartado

Respaldo en Github

TS – Respaldo en Github

- **¿Qué necesitamos?**

Guardamos nuestro proyecto en GitHub.

1.- ¿Tenemos cuenta en **github.com**?

- **Si** la tienes, **recuerda** el usuario y contraseña de tu cuenta.
- **Si no** la tienes, crea una cuenta en github.com, recuerda tu usuario y contraseña.

2.- En la terminal (dentro del directorio del proyecto) **'02-angular-bases-héroes-modulos**'):

- # git init
- # git add .
- # git commit –m 'Fin de la sección 4'

3.- Accedemos a github.com

4.- Creamos el repositorio: '02-angular-bases'

TS – Respaldo en Github

- **¿Qué necesitamos?**

5.- Copiamos las **instrucciones** para:

'push an existing repository from the command line'

6.- Volvemos a la terminal y aplicamos las **instrucciones anteriores**:

- # git remote add origin https://github.com/jgarmay674/02-angular-bases.git
- # git branch -M main
- # git push -u origin main

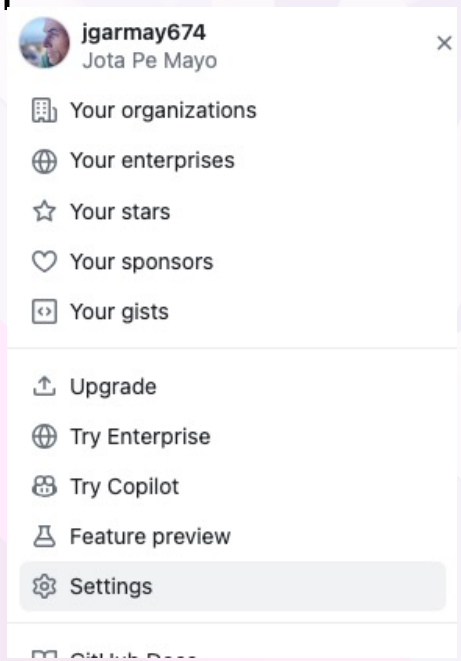
Error: remote: Support for password authentication was removed on August 13, 2021. remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication. fatal: Authentication failed for 'https://github.com/jgarmay674/02-angular-bases.git'

TS – Respaldo en Github

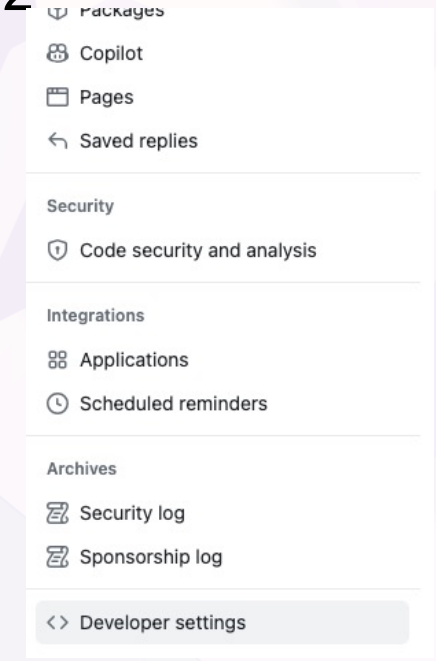
- **¿Qué necesitamos?**

7.- Creamos un token, Personal Access Token (PAT)

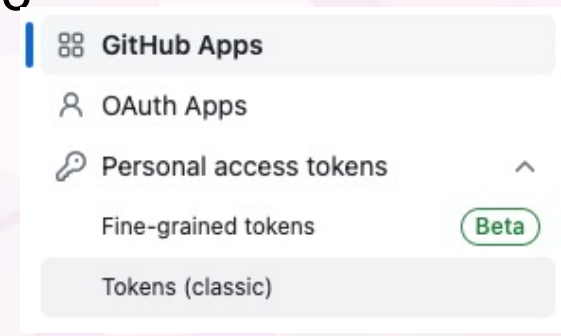
1



2



3



- Genera un nuevo token (classic)
- Indica el uso del token y ‘No expiration’
- Pulsa el botón ‘Generar token’

TS – Respaldo en Github

- **¿Qué necesitamos?**

8.- Ahora sí podemos ejecutar:

```
# git push -u origin main
```

Solicitará la siguiente información:

- Username (de github.com)
- Password (**pegamos el token**)

Guardar las **credenciales**:

Creamos el archivo '**.netrc**' en el raíz. Contenido:

machine **https://github.com/jg...74/02...es.git/**
login **jg…74**
password **ghp_NL…7c**