

En este ejemplo vamos a crear un componente nuevo '**Calc**' que sume dos números insertados en dos **inputs** y muestre el resultado en un contenedor. Todo ello sobre el directorio principal donde sólo existe el módulo raíz

Creamos nuestra aplicación

- El primer paso es, usando el CLI, crear nuestra aplicación nueva. Ejecutamos:

```
ng new calc --no-standalone
```

 y aceptamos por defecto.

- Entramos en el directorio creado `cd calc`

Creamos el nuevo componente 'calc'

- Creamos el nuevo componente `ng g component calc` , al crear el componente podemos comprobar que se ha creado una nueva carpeta '**calc**' que contiene los archivos que se generan con cada componente nuevo.
- También, y dado que no hemos creado ningún módulo nuevo para contener este componente (**CalcComponent**), se ha importado automáticamente al módulo que existe por defecto, y el cual es el principal '**app.module.ts**' (ver contenido)
- Entramos en el archivo de plantilla del componente **calc.component.html**, y vemos que contiene elementos generados por defecto.
- Echamos un vistazo al contenido del archivo **calc.component.ts** , el cual contiene como para cada componente nuevo creado, las siguientes propiedades que da nombre a la etiqueta que vamos a usar (**<app-calc>**), y la asociación de plantilla HTML y CSS.

```
selector: 'app-calc',
standalone: false,
templateUrl: './calc.component.html',
styleUrls: ['./calc.component.css']
```

- Hacemos uso del componente **calc** para hacer una pequeña prueba, para ello, sustituimos todo el contenido del archivo **app.component.html** (componente raíz) por :

```
<div style="text-align:center">
  <app-calc></app-calc>
</div>
```

Ejecutamos `ng serve -o` y veremos en el navegador (<http://localhost:4200/>) la plantilla por defecto, **calc.component.html**.

Agregamos código funcional a la calculadora

Sustituimos el código por defecto de **calc.component.html** , por :

```
<div class="container">
  <div class="header">
    <h2>
      Componente Calculadora
    </h2>
  </div>

  <div class="grid">
    <div class="row">
      <div class="col-6">
        <div class="operation">
          <div class="row">
            <div class="col-12">
```

```
        <input type="number" name=""
placeholder="number">
    </div>
</div>
<div class="row">
    <div class="col-12">
        <input type="number" name=""
placeholder="number">
    </div>
</div>
<div>
    <div class="col-12">
        <button class="button">
            Suma
        </button>
    </div>
</div>
</div>
<div class="col-6">
    <div class="result">
        <span>
            Resultado:
        </span>
    </div>
</div>
</div>
</div>
```

Agregamos los siguientes estilos a **calc.component.css** :

```
.grid{
    width: 100%
}

.row{
    width: 100%;
    display: flex;
}

.col-1 {
    width: 8.33%;
}

.col-2 {
    width: 16.66%;
}

.col-3 {
    width: 25%;
}

.col-4 {
    width: 33.33%;
}

.col-5 {
    width: 41.66%;
}

.col-6 {
    width: 50%;
}

.col-7 {
    width: 58.33%;
}
```

```
}

.col-8 {
    width: 66.66%;
}

.col-9 {
    width: 75%;
}

.col-10 {
    width: 83.33%;
}

.col-11 {
    width: 91.66%;
}

.col-12 {
    width: 100%;
}

.header{
    width: 100%;
    background-color: #003A60;
    height: 100px;
}

.header h2{
    line-height: 100px;
    color: #fff;
}

.button {
    background-color: #4CAF50; /* Green */
    border: none;
    color: white;
```

```
padding: 15px 32px;

text-align: center;

text-decoration: none;

display: inline-block;

font-size: 16px;

margin: 4px 2px;

cursor: pointer;
}

input{

border: none;

border-bottom: 1px solid grey;

width: 80%;

margin: 0% 10%;

padding: 5%;

}

.result{

background-color: #ddffff;

width: 80%;

margin: 20px 10px 10px 10px;

height: 100px;

border-left: 3px solid #2196F3;

}

.result span{

line-height: 100px;

}
```

Guardamos y comprobamos cambios en el navegador.

A continuación vamos a enlazar los datos de entrada mediante enlace bidireccional con **ngModel**, para que cualquier cambio que se dé en las entradas (input) del DOM, se propaguen a las variables (propiedades/atributos) en la clase y viceversa. Para ello, añadimos las propiedades o atributos **number1** y **number2** , en la clase **CalcComponent** en el archivo **calc.component.ts** y las inicializamos a **0**.

```
export class CalcComponent {  
  public number1 : number=0;  
  public number2 : number=0;  
  constructor() { }  
  ngOnInit(): void {  
  }  
}
```

Para hacer una prueba y comprobar que efectivamente cambian los valores de los **input** enlazados con los atributos definidos en la clase y poder usar el enlace **ngModel** en la plantilla del componente **calc**, tenemos que darlo a conocer realizando una importación del módulo **FormsModule** añadiéndolo al módulo principal **app.module.ts**:

```
import { FormsModule } from '@angular/forms';
```

y en el apartado de **imports** :

```
imports: [  
  BrowserModule,  
  FormsModule,  
],
```

Nota: En las últimas versiones de Angular al añadirlo en '**imports**', se importa automáticamente en la cabecera.

A continuación modificamos la plantilla **calc.component.html** , modificando los **input** con :

```
<input type="number" [(ngModel)]="number1" name="" placeholder="number">
```

```
<input type="number" [(ngModel)]="number2" name="" placeholder="number">
```

y en la sección del contenedor del resultado (provisionalmente) por:

```
<div class="result">

  <span>

    Number 1 : {{number1}}

    Number 2 : {{number2}}

  </span>

</div>
```

Guardamos y comprobamos que al cambiar los datos, es reflejado en las variables/propiedades interpoladas.

Añadimos el método suma ()

Modificamos **calc.component.ts** , añadiendo el atributo **resultado** y el método **suma()** , quedaría :

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

@Component({
  selector: 'app-calc',
  standalone: false,
  templateUrl: './calc.component.html',
  styleUrls: ['./calc.component.css']
})
```



```
export class CalcComponent{  
  public number1 : number=0;  
  public number2 : number=0;  
  public resultado : number=0;  
  constructor() { }  
  ngOnInit(): void {  
  }  
  public suma(){  
    this.resultado = this.number1 + this.number2  
  }  
}
```

Finalmente modificamos de nuevo **calc.component.html** , para crear el controlador de evento en el botón para que ejecute nuestro método, y para interpolar el resultado:

```
<button (click)="suma()" class="button">  
  Suma  
</button>
```

```
<div class="result">  
  <span>  
    Resultado : {{resultado}}  
  </span>  
</div>
```

Una vez que hemos comprobado que funciona generamos los archivos para producción, lo desplegamos en el servidor y entramos en la URL.

Finalmente añadimos el módulo SSR/SSG para mejorar la carga de los archivos en producción `ng add @angular/ssr` y volvemos a lanzar **build**.

Comprobamos que se genera un nuevo directorio **server** (SSR) en **dist/..**, además del directorio **browser** (SSG). Dicho directorio **server** contiene los archivos para producción con un servidor **node**, (podemos probarlo lanzando `node server.mjs`). Por otro lado, analizar el archivo **index.html** en **browser** (está generado principalmente con código HTML estático , sistema SSG de prerenderizado, se podrá ver mejor cuando existan varias páginas enrutadas)

Componente Calculadora

2

3

Suma

Resultado : 5