

PROGRAMADO EL FRONTEND CON ANGULAR



CARACTERÍSTICAS DE ANGULAR

- **Framework para la creación de aplicaciones Web de lado cliente**
- **Basado en la creación de plantillas HTML gestionadas por componentes**
- **Simplifica la interacción con el usuario**
- **Vinculación a datos**
- **Implementación de código mediante TypeScript (superconjunto JavaScript orientado a objetos)**
- **Aplicaciones basadas en patrón MVC**

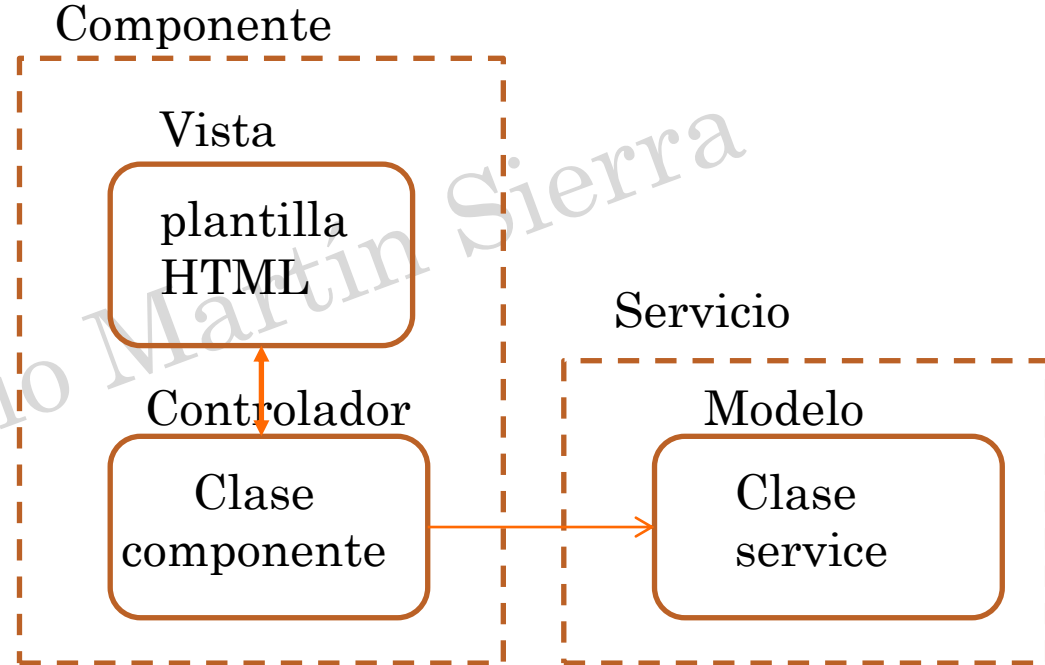


PATRÓN MVC

➤ **Modelo: lógica de negocio**

➤ **Controlador: gestión de eventos de usuario**

➤ **Vista: generación de respuestas**



CONFIGURACIÓN

➤ **Instalación de Node.js. Servidor para publicación de aplicaciones Angular. Incluye npm, una herramienta para gestión de dependencias. Descargable en:**

`https://nodejs.org/es/download/`

➤ **Angular cli. Herramienta para crear proyectos angular. Se instala con npm:**

`> npm install -g @angular/cli`

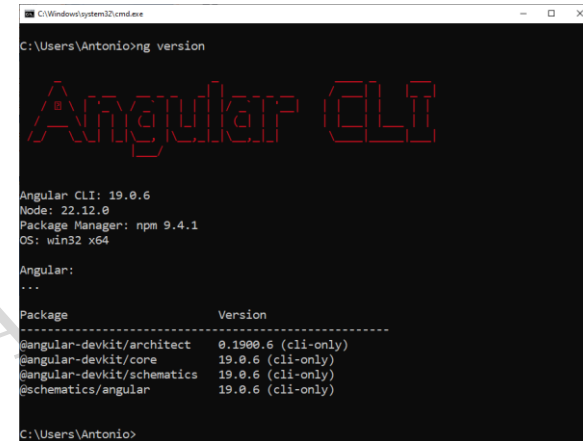
➤ **Visual Studio Code. Editor de proyectos angular para facilitar la codificación.**



VERSIONES

➤ Se puede comprobar las versiones de herramientas mediante el comando:

>ng version



```
C:\Users\Antonio>ng version

Angular CLI
Angular CLI: 19.0.6
Node: 22.12.0
Package Manager: npm 9.4.1
OS: win32 x64

Angular:
...
Package      Version
-----
@angular-devkit/architect 0.1900.6 (cli-only)
@angular-devkit/core      19.0.6 (cli-only)
@angular-devkit/schematics 19.0.6 (cli-only)
@schematics/angular       19.0.6 (cli-only)

C:\Users\Antonio>
```

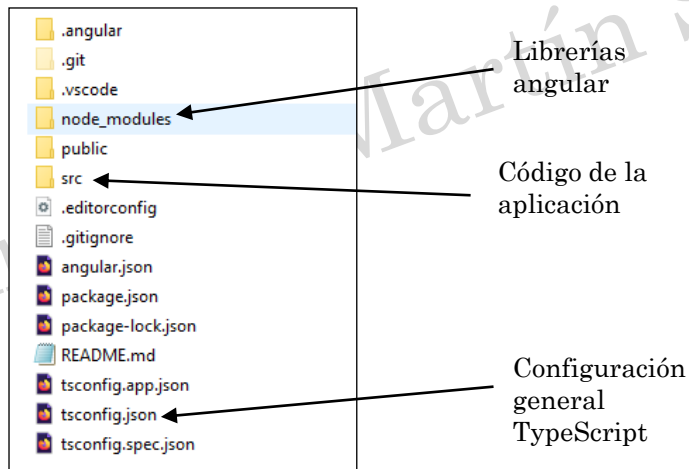


CREACIÓN DE UNA APLICACIÓN

➤ Se crea desde línea de comandos utilizando:

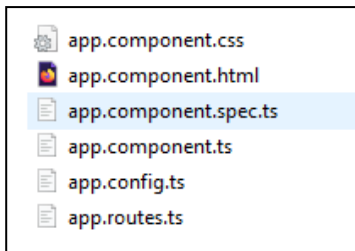
```
>ng new nombreApp
```

➤ Se crea una estructura similar a esta:



COMPONENTES

➤ Al crear un proyecto se genera automáticamente un componente dentro de la carpeta src:



➤ Un componente define el aspecto de una página de la aplicación y controla su comportamiento

➤ Consta de dos elementos:

- Plantilla. Archivo HTML que genera la vista de la página
- Componente. Clase TypeScript donde se implementa la funcionalidad de la página



PLANTILLA

- **Bloque HTML que forma un componente y establece el aspecto de la página.**
- **Puede incluir vínculos hacia la clase del componente para generar dinámicamente contenido y suministrar datos de usuario a dicha clase**

```
<div class="formulario">
  <div>
    <label for="">Introduce código: </label>
    <input type="text" [(ngModel)]="codigo">

    <br>
    <p><a>{{producto.name}}</a><br>
      <a>{{producto.precio}}</a>
    </p>
  </div>
</div>
```

Hacia la clase
componente

Volcado desde el
componente hacia
la página



CLASE DEL COMPONENTE (CONTROLADOR)

➤ Define el comportamiento de la página, recogiendo datos de esta, respondiendo a evento y generando resultados

Etiqueta para referirse al componente desde la página principal

Módulos externos requeridos

Plantilla HTML del componente

Código del componente

```
@Component({
  selector: 'app-buscador',
  imports: [FormsModule, CommonModule],
  templateUrl: './buscador.component.html',
  styleUrls: ['./buscador.component.css']
})
export class BuscadorComponent {
  producto: Producto
  constructor(private buscadorService: BuscadorService){
    producto = new Producto();
  }
  codigo: string;
}
```

LA PÁGINA INDEX.HTML

- Es la página que se carga en el navegador al ejecutar una aplicación Angular
- Contiene referencias a componentes mediante la etiqueta de selector

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Aplicación Angular</title>
  <base href="/">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-buscador></app-buscador>
</body>
</html>
```

Referencia al
componente que será
procesado al solicitar
la página

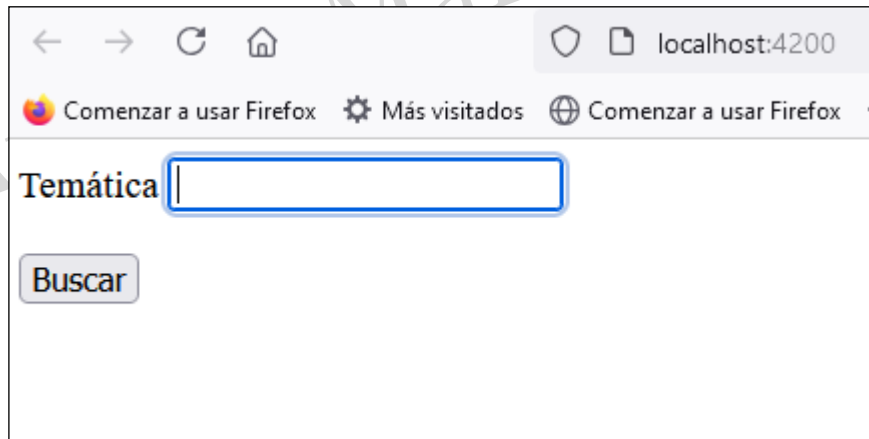


EJECUCIÓN DE LA APLICACIÓN

➤ Para ejecutar una aplicación angular utilizaremos el comando:

```
>ng serve -o
```

➤ La aplicación se ejecutará automáticamente en un servidor node.js, se abrirá un navegador y se lanzará una solicitud de index.html



VINCULACIÓN A DATOS

➤ A través de directivas asociamos el contenido de controles HTML de la platilla a propiedades del componente

```
<input type="text" [(ngModel)]="nombre">
```

Se debe importar el módulo *FormsModule* en el componente

➤ Mediante los interpoladores se vuelcan propiedades del componente en la página

```
<h2>{{texto}}</h2>
```

```
export class DatosComponent {  
  nombre:string;  
  texto:string;  
  :  
}
```



PASO DE PARÁMETROS AL COMPONENTE

➤ Se pueden pasar parámetros al componente desde la página index mediante atributos de la etiqueta asociada, utilizando *property binding*

```
<app-root [level]="5"></app-root>
```

```
export class DatosComponent {  
  level:number;  
  :  
}
```

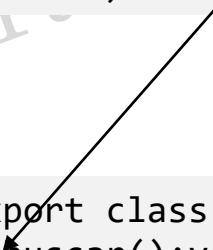


EVENTOS

- Una de las funcionalidades de la capa front es capturar eventos o acciones de usuario.
- En angular los eventos se manejan a través de funciones de respuesta definidas en el componente

```
<input type="button" value="Buscar" (click)="buscar()">
```

```
export class BuscadorComponent {  
  buscar():void{  
    :  
  }  
}
```



DIRECTIVA NGIF

➤ Puede ser incluida en cualquier etiqueta HTML para que dicha etiqueta sea o no procesada en función de una condición.

➤ Su formato:

```
<etiqueta *ngIf="expresion">...</etiqueta>
```

➤ Si *expresion* es evaluada como falso, la etiqueta será eliminada del árbol de objeto DOM.

➤ Para poder utilizar esta etiqueta es necesario importar el **CommonModule** en el componente:

```
@Component({  
  selector: 'app-buscador',  
  imports: [CommonModule],  
  :
```



DIRECTIVA NGFOR

➤ Se incluye en una etiqueta para que esta aparecerá tantas veces como se indique en la expresión de iteración asignada a la directiva.

➤ Su formato:

```
<etiqueta *ngFor="let variable of array">...</etiqueta>
```

➤ Ejemplo:

Genera tantas filas <tr> como elementos haya en la colección o array "agenda"

```
<tr *ngFor="let c of agenda">  
  <td>{{c.nombre}}</td><td>{{c.edad}}</td><td>{{c.telefono}}</td>  
</tr>
```

➤ El uso de esta directiva también requiere la importación de CommonModule

DIRECTIVA NGSWITCH

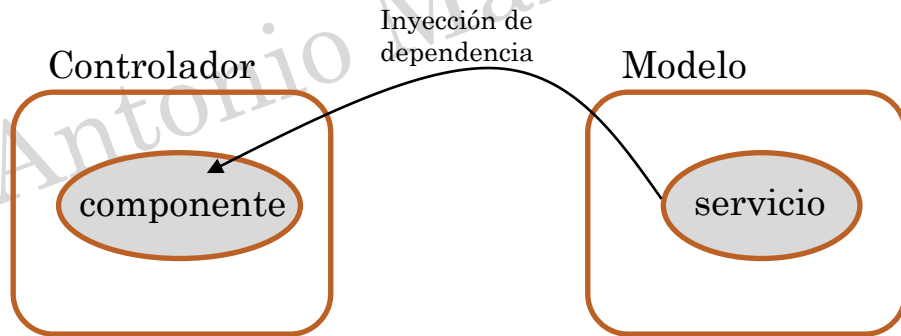
- Utilizamos esta directiva para mostrar diferentes elementos de un conjunto de ellos, en función del resultado de una expresión.
- Procesa la etiqueta cuyo valor @ngSwitchWhen coincide con el resultado de la expresión:

```
<div [ngSwitch]="seleccionado.nombre">  
  <div *ngSwitchWhen="Ana">Vive cerca</div>  
  <div *ngSwitchWhen="Belén">Acaba de mudarse</div>  
  <div *ngSwitchWhen="Marcos">Vive ahí desde siempre</div>  
  <div *ngSwitchDefault>desconocidos</div>  
</div>
```



SERVICIOS

- Encapsulan la lógica de negocio de la aplicación (Modelo).
- Exponen su funcionalidad al componente controlador a través de métodos.
- El servicio es inyectado en el componente para que pueda hacer uso del mismo



IMPLEMENTACIÓN DE UN SERVICIO

➤ Se implementa en una clase estándar anotada con **@Injectable**:

```
@Injectable({  
  providedIn: 'root'  
})  
export class BuscadorService {  
  buscar(tematica:string):String []{  
    :  
  }  
}
```

➤ En el componente se inyecta a través del constructor:

```
export class BuscadorComponent {  
  constructor(private buscadorService:BuscadorService){  
  
  }  
}
```

Inyección de servicio en un
atributo de la clase



PETICIONES HTTP

➤ Se realizan a través del componente `HttpClient` incluido en la librería `http`.

➤ Este componente deberá ser utilizado desde un servicio. Puede ser inyectado a través del constructor:

```
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class BuscadorService {
  constructor(private http: HttpClient){
  }
}
```

➤ El proveedor del módulo debe ser declarado en `app.config`:

```
import { provideHttpClient } from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [provideHttpClient(),...]
};
```



OBSERVABLES

- Una petición HTTP (get, post, ...) devuelve un Observable
- A un Observable se suscribe el componente controlador para procesar el resultado cuando esté disponible:

componente

```
export class PaisComponent {  
  paises:Pais[];  
  constructor(private paisesService:PaisesService){  
  }  
  cargarPaises():void{  
    this.paisesService.obtenerPaises().subscribe(data=>this.paises=data);  
  }  
}
```

suscripción al
observable

servicio

```
export class PaisesService{  
  constructor(private http:HttpClient){  
  }  
  public obtenerPaises(): Observable<Pais[]> {  
    return this.http.get<Pais[]>(this.url); //Observable  
  }  
}
```

ENVÍO DE DATOS EN PETICIONES HTTP

➤ Desde un cliente se pueden enviar datos a un recurso externo de la siguiente manera:

- Path variables. Los datos se envían como parte de la dirección
- QueryString. Se envían parámetros en la URL en parejas clave=valor
- Form-urlencoded. Se envían parámetros en el cuerpo de la petición en parejas clave=valor
- JSON. Se pueden enviar datos como un documento JSON en el cuerpo de la petición



PATH VARIABLES Y QUERYSTRING

➤ Path variable:

```
export class BuscadorService{  
    find(cod:number):Observable<Item> {  
        return this.http.get<Item>("http://localhost:8000/buscador/${cod}");  
    }  
}
```

➤ Envío como QueryString:

```
export class BooksService{  
    listBooks(isbn: string):Observable<Book[]>{  
        return this.http.get<Book[]>("http://localhost:9000/books",{  
            params:{"isbn":isbn}  
        });  
    }  
}
```

FORM-URLENCODED

➤ **Utilizado habitualmente en peticiones post cuando el backend espera recibir un formulario de datos:**

```
export class EmpleadosService{
  save(cod:number,name:string,age:number):Observable<void>{
    let params=new HttpParams();
    let heads=new HttpHeaders();
    //los parámetros se definen en un objeto params
    params=params.set("codigo",cod);
    params=params.set("nombre",name);
    params=params.set("edad",age);
    //se debe establecer un encabezado con el tipo de contenido
    heads=heads.set("Content-Type","application/x-www-form-urlencoded");
    return this.http.post<void>(url,params,{"headers":heads});
  }
}
```


JSON

➤ **Forma de envío habitual para enviar un grupo de datos a un servicio REST en el cuerpo de la petición**

```
export class ClientesService{  
  save(cliente:Cliente):Observable<void>{  
    let heads=new HttpHeaders();  
    //se debe establecer un encabezado con el tipo de contenido  
    heads=heads.set("Content-Type","application/json");  
    return this.http.post<void>(url,cliente,{"headers":heads});  
  }  
}
```

Amo



CABECERAS DE RESPUESTA

➤ Si queremos tener acceso a las cabeceras de respuesta, se debe incluir el parámetro `observe:"response"` en la lista de parámetros opcionales de la petición:

```
buscar(tematica:string):Observable<any>{  
    return this.http.get(this.urlBase+"buscar",  
        {params:{tematica:tematica},observe:"response"});  
}
```

➤ En el componente:

```
buscar():void{  
    this.buscadorService.buscar(this.tematica)  
        .subscribe(data=>{  
            console.log(data.headers)); //cabeceras  
            console.log(data.body);    //cuerpo  
        })  
}
```

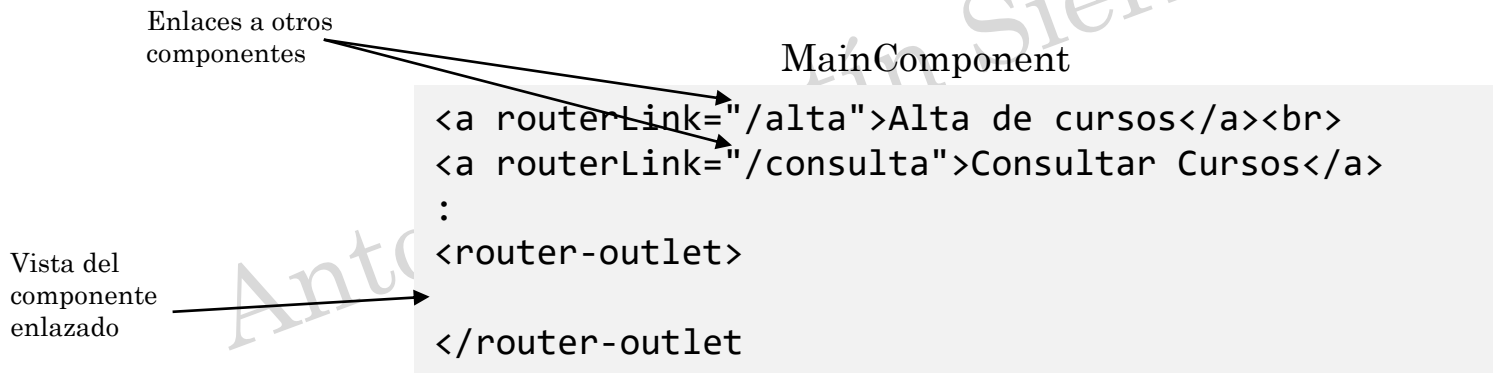
ROUTING

- Es la capacidad para navegar entre componentes
- Enlaces en la vista de un componente provocan la carga de otros componentes
- La navegación puede realizarse también desde código
- El routing es habilitado por defecto al crear la aplicación.
- Las reglas de navegación se definen en el archivo `app-routing.module.ts`.



CREAR RUTAS

- Los enlaces a otros componentes se generan en la vista del componente principal mediante el atributo *routerlink*.
- En esta vista se incluirá una etiqueta `<router-outlet>` donde se mostrará la vista del componente enlazado:



- La navegación se puede realizar desde código mediante:

```
this.router.navigate(["/eliminar"])
```

MAPEADO DE RUTAS A COMPONENTES

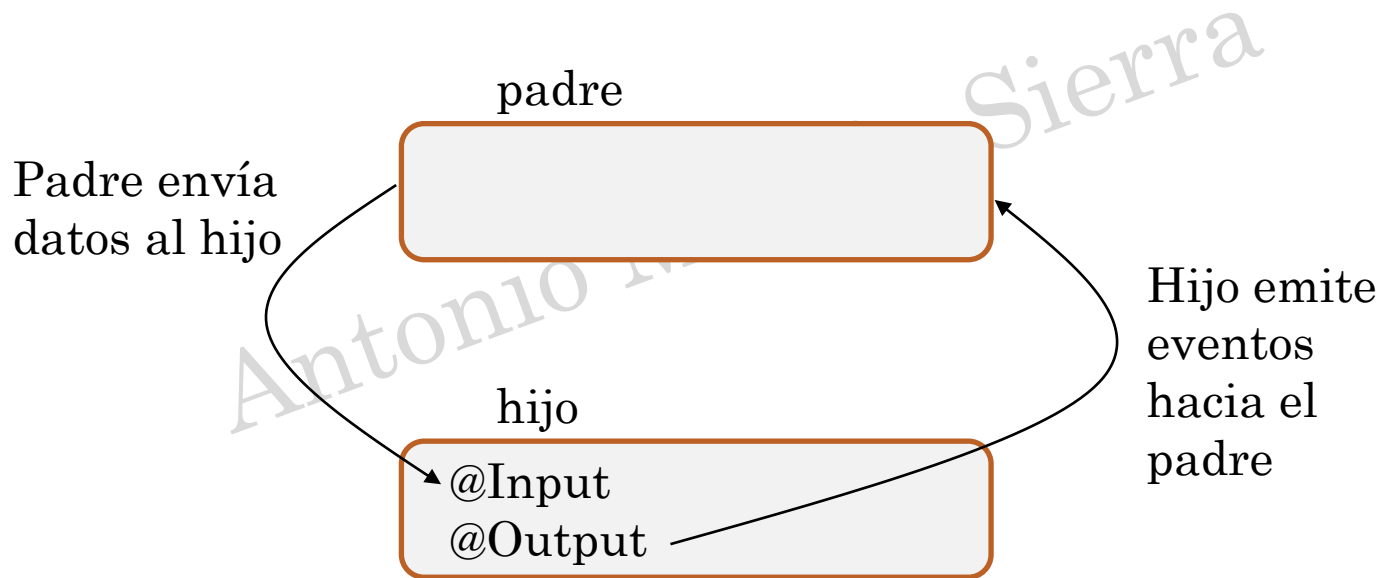
➤ La asociación de rutas a componentes se realiza en el archivo `app-routing.module.ts` :

```
const routes: Routes = [{  
  path:"alta",  
  component:AltaComponent  
},  
{  
  path:"consulta",  
  component: ConsultaComponent  
},  
{  
  path:"eliminar",  
  component: EliminarComponent  
}  
];
```



COMPONENTES PADRE-HIJO

- Cuando un componente va a ser reutilizado en distintas partes de la aplicación.
- Facilita la modularización de la aplicación



DISTRIBUIR UNA APLICACIÓN ANGULAR

➤ Se debe traducir todo el código TypeScript+librerías angular a JavaScript, que es lo que entienden los navegadores.

➤ Para ello se utiliza el comando:

```
>ng build --base-href=/nombre_app/
```

➤ Se genera un index.html y dos archivos .js en la carpeta dist del proyecto o workspace. Esos archivos se suben a un servidor Web



TESTING EN ANGULAR

- La librería **jazmin** permite realizar pruebas unitarias en Angular
- Se incorpora por defecto con la instalación de Angular cli
- Los métodos de test se incluyen en los archivos **spec.ts** asociados a **service** y **controllers**
- Las pruebas se basan en resultados esperados en las llamadas a los métodos



EJEMPLO TESTING DE UN SERVICE

Para ser ejecutado
antes de cada test

```
describe('CalculadoraService', () => {  
  let service: CalculadoraService;  
  beforeEach(() => {  
    TestBed.configureTestingModule({});  
    service = TestBed.inject(CalculadoraService);  
  });  
  it("suma de 5 y 6",()=>{  
    expect(service.sumar(5,6)).toBe(11);  
  })  
  it("multiplicacion de 7 y 4",()=>{  
    expect(service.multiplicar(7,4)).toBe(28);  
  })  
  it("factorial de 5",()=>{  
    expect(service.factorial(5)).toBe(120);  
  })  
  it("factorial de -3",()=>{  
    expect(service.factorial(-3)).toBe(0);  
  })  
});
```

➤ Para ejecutar las pruebas:

>ng test

>ng test nombre_app

En aplicaciones
dentro de un
workspace



CREACIÓN DE UN PIPE PERSONALIZADO

➤ Los pipes aplican transformaciones a los datos cuando se van a presentar en la página

➤ Ejemplo:

nombre del pipe

```
@Pipe({  
  name: 'poblacion'  
})  
export class PoblacionPipe implements PipeTransform {  
  transform(value: number): string {  
    return value.toLocaleString();  
  }  
}
```

➤ Uso:

```
<td>{{p.population|poblacion}}</td>
```

