ANGULAR

ANGULAR

¿QUE ES ANGULAR?

Es un framework de diseño de aplicaciones y plataforma de desarrollo para crear aplicaciones de una sola página eficientes y sofisticadas

ANGULAR

REQUERIMIENTOS:

Requiere conocimientos en HTML, CSS, JAVASCRIPT y TYPESCRIPT

Requiere tener instalada la versión LTS de NODE y NPM

Requiere conocimentos básicos de la terminal/bash

ANGULAR

¿QUE ES MVC?

Es un patrón de diseño de software ampliamente utilizado en la industria del desarrollo de aplicaciones. Su objetivo principal es separar las preocupaciones dentro de una aplicación en tres componentes distintos para mejorar la modularidad, la mantenibilidad y la escalabilidad del código

ANGULAR

MODELO:

Representa principalmente los datos y la lógica de negocio de la aplicación. Los modelos se definen generalmente como clases TypeScript y pueden incluir propiedades y métodos que describen cómo se acceden y se manipulan los datos.

ANGULAR

VISTA:

La vista es la capa de presentación de la aplicación. Se encarga de mostrar los datos al usuario y de presentar la interfaz de usuario de manera adecuada. La vista recibe datos del modelo y muestra la información de una manera que sea comprensible y atractiva para el usuario. En muchas aplicaciones web, la vista suele estar implementada en HTML y CSS.

ANGULAR

CONTROLADOR

El controlador actúa como intermediario entre el modelo y la vista. Se encarga de manejar las interacciones del usuario, como clics de botón o eventos de entrada, y coordina las acciones necesarias para que el modelo y la vista se mantengan sincronizados. En otras palabras, el controlador controla el flujo de datos entre el modelo y la vista.
Generalmente en Typescript

ANGULAR

CLI DE ANGULAR:

Angular Command Line Interface, es una herramienta de línea de comandos que se utiliza para crear, desarrollar y administrar aplicaciones web en Angular de manera eficiente. Angular es un popular framework de desarrollo de aplicaciones web y el CLI proporciona una serie de utilidades y comandos que simplifican muchas tareas comunes en el proceso de desarrollo.

ANGULAR

CLI DE ANGULAR:

Instalar el CLI de Angular:

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

ANGULAR

CLI DE ANGULAR:

Crear una nueva APP basada en Angular:

```
ng new my-app
```

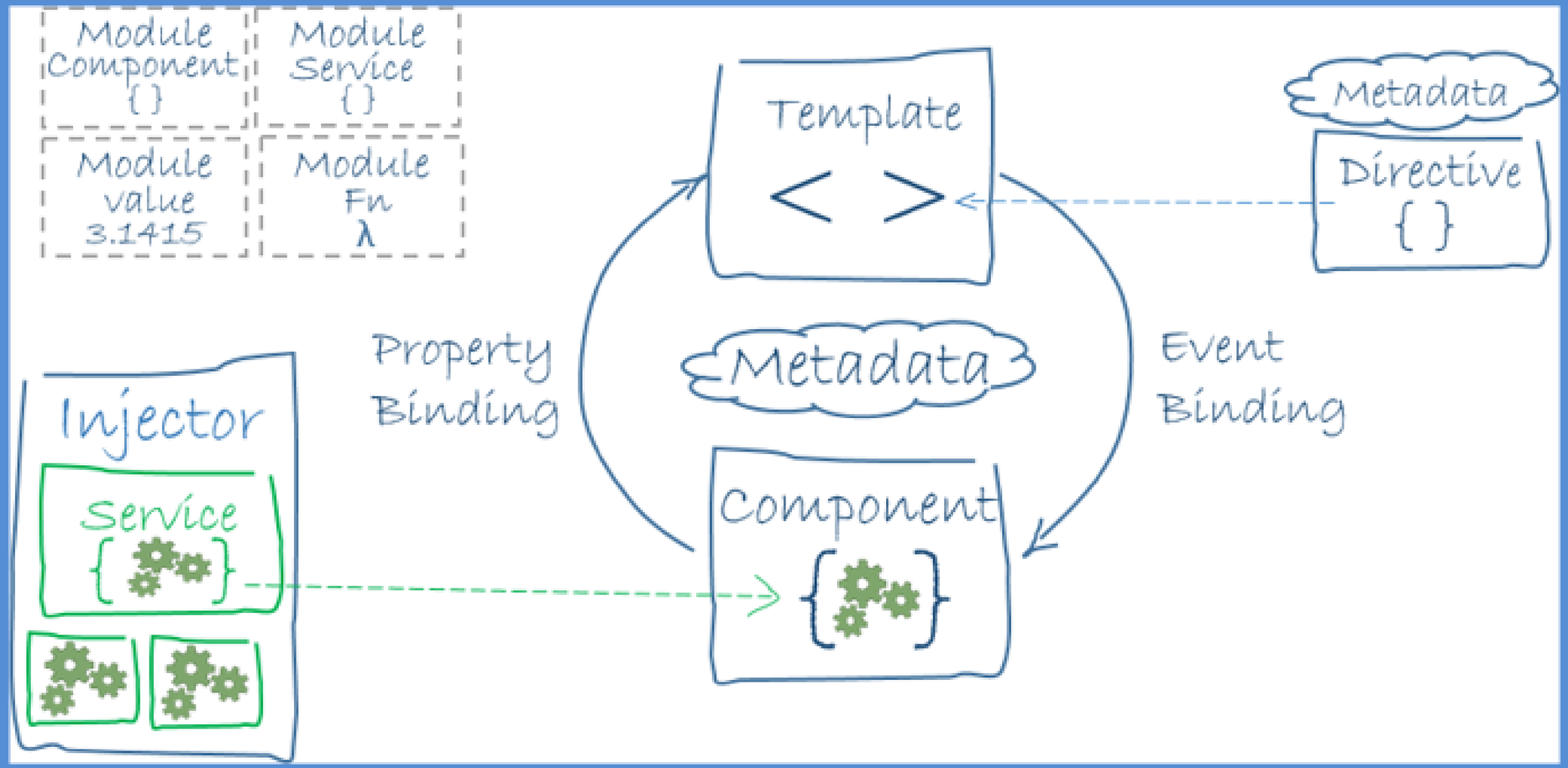
ANGULAR

CONCEPTOS BÁSICOS:

- Módulos
- Componentes
- Servicios
- Dependencias

- Plantillas
- Directivas
- Enrutamiento
- Metadata

ANGULAR


Module
Component
{}
Module
Service
{}
Module
value
3.1415
Module
Fn
λ
Template
< >
Metadata
Directive
{ }
Property
Binding
Metadata
Event
Binding
Injector
Service
Component

ANGULAR

MODULOS:

Son una parte fundamental de la arquitectura de la aplicación. Un módulo es un mecanismo de organización y encapsulación que se utiliza para agrupar componentes, directivas, pipes (filtros), servicios y otros elementos relacionados en una unidad funcional coherente. Los módulos ayudan a dividir una aplicación en partes más pequeñas y manejables, lo que facilita el desarrollo, la mantenibilidad y la escalabilidad.

ANGULAR

MODULOS:

- Unidades organizativas.
- Encapsulan funcionalidades.
- Dividen la aplicación.
- Importan/exportan elementos.
- Registran proveedores.
- Evitan conflictos de nombres.

ANGULAR

MODULOS CON CLI:

```
ng generate module nombre-del-modulo
```

```
ng g m nombre-del-modulo
```

ANGULAR

MODULOS:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MiComponente } from './mi-componente.component';

@NgModule({
  declarations: [MiComponente],
  imports: [CommonModule],
  exports: [MiComponente]
})
export class MiModulo { }
```

ANGULAR

COMPONENTES:

Es un bloque fundamental de construcción para la creación de interfaces de usuario (UI) en una aplicación web. Los componentes son responsables de definir cómo se ve y se comporta una parte específica de la interfaz de usuario de la aplicación. Cada componente representa un elemento visual o funcional de la página web, como un encabezado, un pie de página, un formulario, una lista de elementos, etc.

ANGULAR

COMPONENTES:

- Bloques de UI.
- Definen vistas y lógica.
- Usan clases TypeScript.
- Tienen plantillas HTML.
- Son reactivos.
- Altamente reutilizables.
- Se encapsulan.
- Jerarquía en la UI.

ANGULAR

COMPONENTS CON CLI:

```
ng generate component nombre-del-componente
```

```
ng g c nombre-del-componente
```

ANGULAR

COMPONENTS CON CLI:

Se crearán 4 archivos:

- **el archivo del componente (**nombre-del-componente.component.ts**)**
- **el archivo HTML de la plantilla (**nombre-del-componente.component.html**)**
- **el archivo de estilos CSS (**nombre-del-componente.component.css**)**
- **un archivo de prueba (**nombre-del-componente.component.spec.ts**)**

ANGULAR

COMPONENTE: controller

```ts
// nombre-del-componente.component.ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-nombre-del-componente',
  templateUrl: './nombre-del-componente.component.html',
  styleUrls: ['./nombre-del-componente.component.css']
})
export class NombreDelComponenteComponent implements OnInit {

  // Propiedades del componente
  titulo: string = 'Mi Componente';

  // Constructor
  constructor() { }

  // Método de inicialización
  ngOnInit(): void {
  }

  // Otros métodos y lógica del componente
}
```

ANGULAR

COMPONENTE: plantilla HTML

```html
<!-- nombre-del-componente.component.html -->
<div>
  <h1>{{ titulo }}</h1>
  <p>Este es el contenido de mi componente.</p>
</div>
```

ANGULAR

COMPONENTE: estilos CSS

```css
/* nombre-del-componente.component.css */
div {
  background-color: #f0f0f0;
  padding: 20px;
  border: 1px solid #ccc;
}

h1 {
  color: #007bff;
}
```

ANGULAR

COMPONENTE: pruebas SPEC

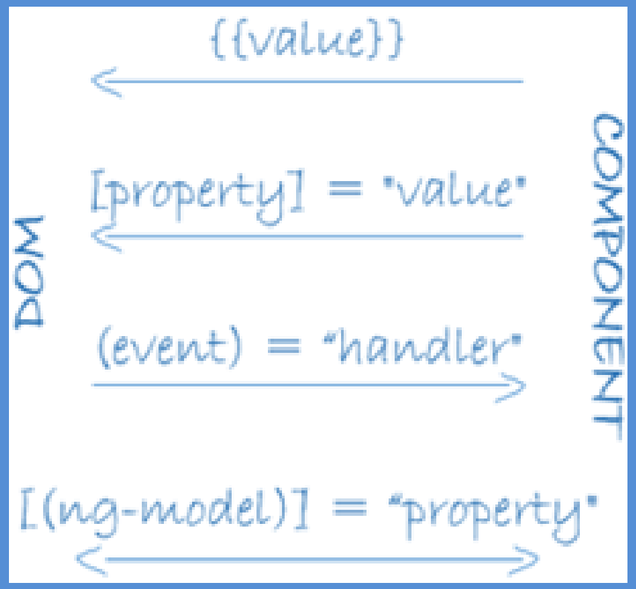
```
// nombre-del-componente.component.spec.ts
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { NombreDelComponenteComponent } from './nombre-del-componente.component';

describe('NombreDelComponenteComponent', () => {
  let component: NombreDelComponenteComponent;
  let fixture: ComponentFixture<NombreDelComponenteComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [NombreDelComponenteComponent]
    })
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(NombreDelComponenteComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

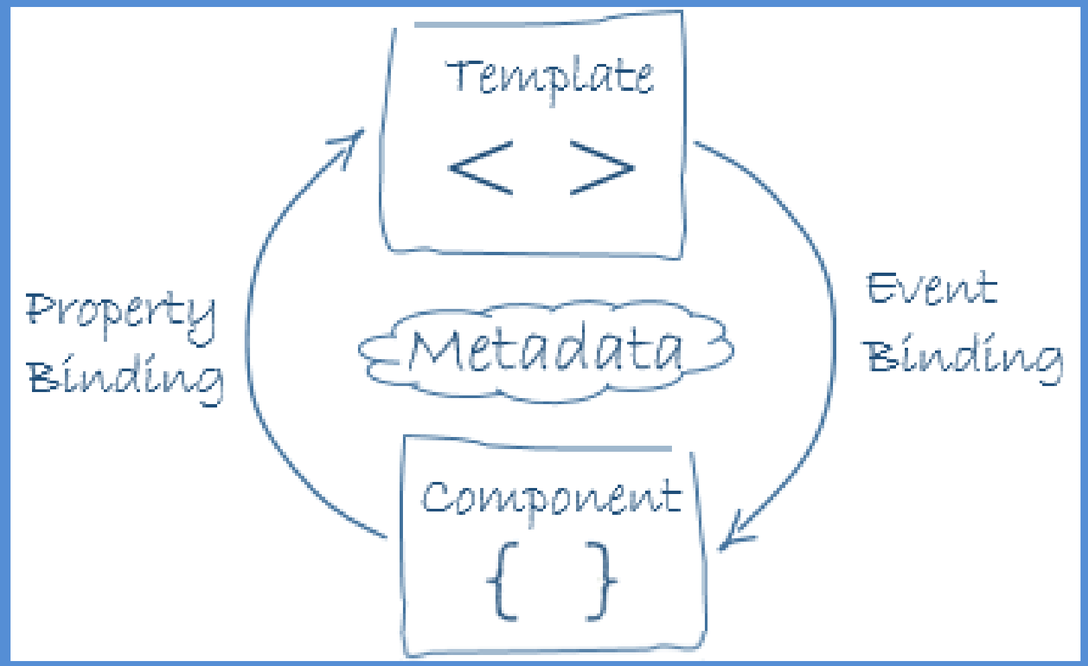
ANGULAR

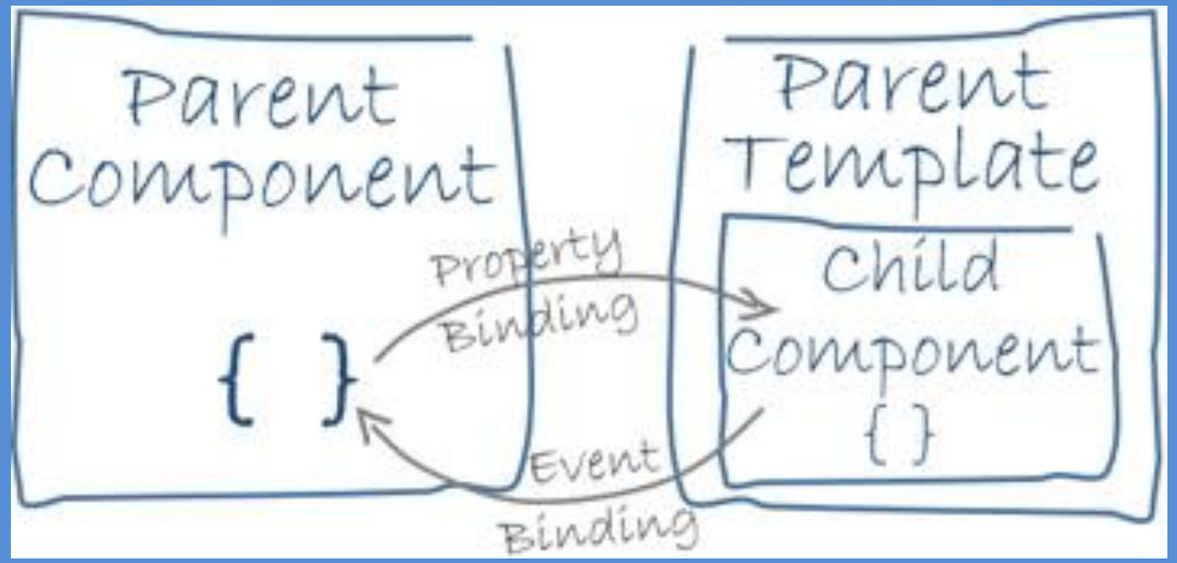
ENLACES DE DATOS:

Se refiere a la capacidad de conectar y sincronizar automáticamente los datos entre el modelo (que representa el estado de la aplicación) y la vista (la interfaz de usuario que muestra esos datos). El enlace de datos es una característica fundamental que permite que los cambios en el modelo se reflejen automáticamente en la vista y viceversa, sin necesidad de intervención manual.

ANGULAR


{{value}}
[property] = "value"
(event) = "handler"
[(ng-model)] = "property"
DOM
COMPONENT


Template
< >
Property Binding
Metadata
Event Binding
Component
{ }

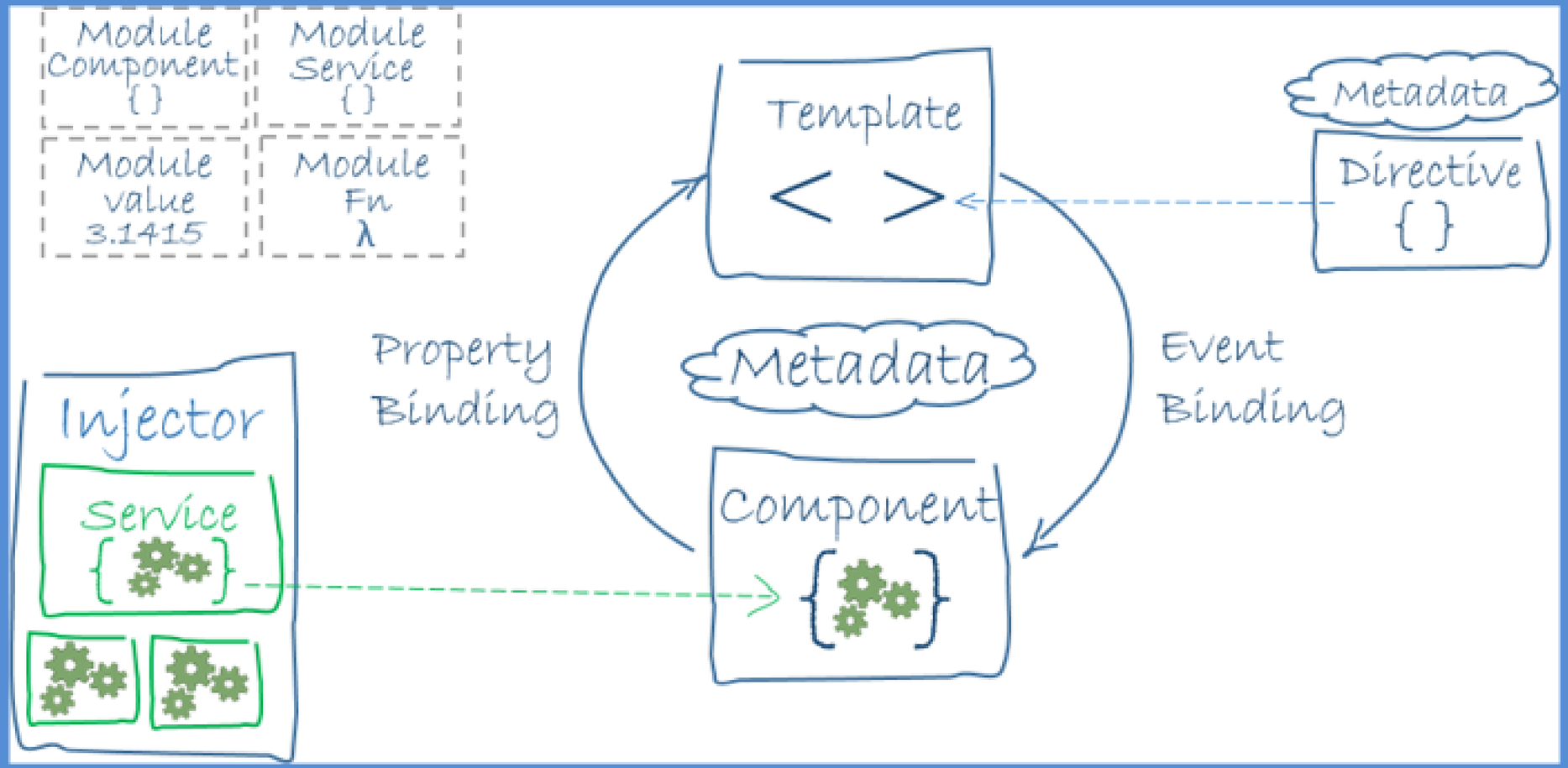

Parent Component
{ }
Property Binding
Event Binding
Parent Template
Child Component
{ }

ANGULAR

ENLACES DE DATOS:

- **Conexión Automática: Sincroniza datos entre el modelo y la vista.**
- **Unidireccional: Los cambios en el modelo se reflejan en la vista.**
- **Bidireccional: Cambios en la vista actualizan el modelo (por ejemplo, formularios).**
- **Reactivo: Utiliza Observables para actualizaciones en tiempo real.**
- **Simplifica Interacción: Facilita la creación de aplicaciones interactivas.**
- **Automatiza Actualizaciones: Cambios se reflejan sin intervención manual.**

ANGULAR


Module
Component
{}
Module
Service
{}
Module
value
3.1415
Module
Fn
λ
Template
< >
Metadata
Directive
{ }
Property
Binding
Metadata
Event
Binding
Injector
Service
Component

ANGULAR

ENLACE DE DATOS:

```ts
// contador.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-contador',
  templateUrl: './contador.component.html',
  styleUrls: ['./contador.component.css']
})
export class ContadorComponent {
  valorContador: number = 0;

  incrementar() {
    this.valorContador++;
  }

  decrementar() {
    this.valorContador--;
  }
}
```

ANGULAR

ENLACE DE DATOS:

En la plantilla HTML:

```
<!-- contador.component.html -->
<h1>Contador: {{ valorContador }}</h1>
<button (click)="incrementar()">Incrementar</button>
<button (click)="decrementar()">Decrementar</button>
```

ANGULAR

METADATA:

Se refiere a la información adicional que se proporciona mediante decoradores en las clases que definen componentes, módulos, servicios y otras partes de una aplicación. La metadata se utiliza para configurar y personalizar el comportamiento de estas partes de la aplicación. Los decoradores, como @Component, @NgModule, @Injectable, entre otros, se utilizan para adjuntar esta metadata a las clases.

ANGULAR

METADATA:

- **Configuración: Define cómo se comportan las partes de la aplicación.**
- **Decoradores: Se utiliza con decoradores como @Component, @NgModule, @Injectable.**
- **Personalización: Ajusta el comportamiento con propiedades clave.**
- **Componente: Metadata para componentes, incluye plantilla y estilos.**
- **Módulo: Configuración de módulos, como declaraciones e importaciones.**
- **Servicio: Metadata para servicios, define su alcance y proveedores.**
- **Directiva: Define metadata de directivas personalizadas, como selectores.**

ANGULAR

SERVICIOS:

Es una clase TypeScript que se utiliza para organizar y compartir lógica, datos o funcionalidades comunes entre diferentes componentes de una aplicación. Los servicios son una parte fundamental de la arquitectura de Angular y proporcionan una forma de centralizar y reutilizar la lógica que no está relacionada directamente con la interfaz de usuario.

ANGULAR

SERVICIOS:

- **Reutilización: Lógica compartida.**
- **Separación de preocupaciones: Divide lógica y UI.**
- **Inyección de dependencias: Instancias proporcionadas.**
- **Centralización de datos: Almacena y gestiona datos compartidos.**
- **Comunicación entre componentes: Facilita la comunicación.**
- **Lifecycle independiente: No vinculado a vistas.**
- **Testeabilidad: Fácil de probar.**

ANGULAR

SERVICIO CON CLI:

```
ng generate service nombre-del-servicio
```

```
ng g s nombre-del-servicio
```

ANGULAR

SERVICIO:

```typescript
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class MiServicioService {

  constructor() { }

  // Métodos y lógica del servicio
}
```

ANGULAR

DEPENDENCIAS:

Son los recursos externos y módulos de código que una aplicación necesita para funcionar correctamente. Estos recursos pueden incluir bibliotecas externas, módulos de Angular, servicios personalizados, componentes y otros elementos que se utilizan en la aplicación para realizar tareas específicas.

Las dependencias en Angular se gestionan principalmente a través del sistema de inyección de dependencias (DI), que es una característica clave del framework.

ANGULAR

DEPENDENCIAS:

- Recursos Externos: Bibliotecas y recursos de terceros.
- Módulos de Angular: Unidades organizativas con funcionalidades.
- Servicios Personalizados: Funcionalidad compartida.
- Inyección de Dependencias: Gestión automática de instancias.
- Inyección en Constructores: Dependencias pasadas a través de constructores.
- Gestión de Ciclo de Vida: Control de creación y destrucción.
- Testeabilidad: Facilita pruebas unitarias y de integración.

ANGULAR

INYECCION DE DEPENDENCIAS

```typescript
import { Component } from '@angular/core';
import { MiServicio } from './mi-servicio.service';

@Component({
  selector: 'app-mi-componente',
  templateUrl: './mi-componente.component.html'
})
export class MiComponente {
  constructor(private miServicio: MiServicio) {
    // ...
  }
}
```

ANGULAR

DIRECTIVAS:

Son instrucciones en el marcado HTML que proporcionan funcionalidad adicional a los elementos DOM existentes o personalizan su comportamiento. Las directivas son un componente clave en la construcción de aplicaciones web en Angular, ya que permiten extender y manipular el DOM de manera declarativa, lo que facilita la creación de interfaces de usuario dinámicas e interactivas. Angular proporciona varias directivas incorporadas y también permite la creación de directivas personalizadas.

ANGULAR

DIRECTIVAS:

- **Instrucciones HTML: Extienden o personalizan elementos HTML.**
- **Directivas Incorporadas: Ofrecen funcionalidad predefinida.**
- **Directivas Estructurales: Manipulan la estructura del DOM.**
- **Directivas de Atributos: Cambian atributos y propiedades.**
- **Directivas de Eventos: Capturan y responden a eventos del usuario.**
- **Directivas Personalizadas: Creadas para necesidades específicas.**
- **Inyección de Dependencias: Acceso a servicios y datos.**
- **Flexibilidad de Aplicación: Se pueden aplicar como atributos o elementos.**

ANGULAR

DIRECTIVA CON CLI:

```
ng generate directive nombre-de-la-directiva
```

```
ng g d nombre-de-la-directiva
```

ANGULAR

DIRECTIVA:

```ts
// nombre-de-la-directiva.directive.ts
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appMiDirectiva]'
})
export class MiDirectivaDirective {
  constructor(private el: ElementRef) {
    // Accede al elemento del DOM en el que se aplica la directiva (this.el.nativeElement)
    this.el.nativeElement.style.backgroundColor = 'yellow';
  }
}
```

ANGULAR

LLAMAR DIRECTIVA:

Desde la plantilla del HTML:

```html
<div appMiDirectiva>
  Este es un elemento con mi directiva personalizada.
</div>
```

ANGULAR

PIPES:

Son una característica que permite formatear y transformar datos en la vista de una aplicación web de manera sencilla y legible. Los pipes son funciones que toman un valor de entrada (como una cadena de texto, un número o un objeto) y lo procesan para proporcionar una representación modificada o formateada en la interfaz de usuario.

Los pipes se utilizan en las plantillas HTML de Angular y se aplican utilizando el símbolo de barra vertical |. Algunos ejemplos comunes de uso de pipes incluyen el formateo de fechas, números, monedas, texto en mayúsculas o minúsculas, entre otros.

Angular proporciona una serie de pipes integrados, como DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, DecimalPipe, PercentPipe, entre otros. Además, también puedes crear tus propios pipes personalizados cuando necesites realizar transformaciones específicas.

ANGULAR

PIPES:

- **Formateo de Datos: Transforma datos para presentarlos.**
- **Sintaxis de Pipe: Se aplica en plantillas con |.**
- **Pipes Integrados: Angular proporciona pipes predefinidos.**
- **Pipes Encadenados: Se pueden combinar varios pipes.**
- **Pipes Personalizados: Creados para necesidades específicas.**
- **Parámetros de Pipe: Aceptan configuración adicional.**
- **Inmutabilidad: No alteran los datos originales.**

ANGULAR

PIPES CON CLI:

```
ng generate pipe nombre-del-pipe
```

```
ng g p nombre-del-pipe
```

ANGULAR

PIPE:

```ts
// nombre-del-pipe.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'miPipe'
})
export class MiPipe implements PipeTransform {

  transform(valor: any): any {
    // Implementa la lógica de transformación aquí
    return valor.toUpperCase();
  }
}
```

ANGULAR

PIPE:

Pipe llamado en la plantilla HTML:

```
<p>{{ texto | miPipe }}</p>
```

ANGULAR

ENRUTAMIENTO:

Se refiere a la capacidad de dirigir el flujo de navegación de una aplicación web hacia diferentes vistas o páginas en función de la URL que el usuario solicita. Angular proporciona un módulo llamado RouterModule que permite la configuración y gestión del enrutamiento en una aplicación Angular. El enrutamiento es fundamental para crear aplicaciones de una sola página (SPA, por sus siglas en inglés) y para proporcionar una experiencia de usuario fluida al permitir la navegación entre diferentes secciones de la aplicación sin necesidad de recargar la página completa.

ANGULAR

ENRUTAMIENTO:

- Navegación basada en URL: Controla las vistas mediante URLs.
- Router Outlet: Punto de carga dinámica de componentes.
- Configuración de Rutas: Asocia URLs con componentes.
- Rutas Anidadas: Crea jerarquías de vistas.
- Parámetros de Ruta: Personaliza vistas con datos dinámicos.
- Guardias de Ruta: Control de acceso y autorización.
- Rutas con Nombre: Facilita la navegación programática.
- Lazy-Loading: Carga diferida de módulos y rutas.
- Navegación Programática: Cambios de ruta mediante código.

ANGULAR

ENRUTAMIENTO CON CLI:

```
ng new nombre-de-tu-proyecto --routing
```

ANGULAR

ENRUTAMIENTO:

```ts
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

ANGULAR

ENRUTAMIENTO:

```html
<!-- app.component.html -->
<nav>
  <a routerLink="/">Inicio</a>
  <a routerLink="/about">Acerca de</a>
</nav>

<router-outlet></router-outlet>
```