

# Servicios en Angular 5

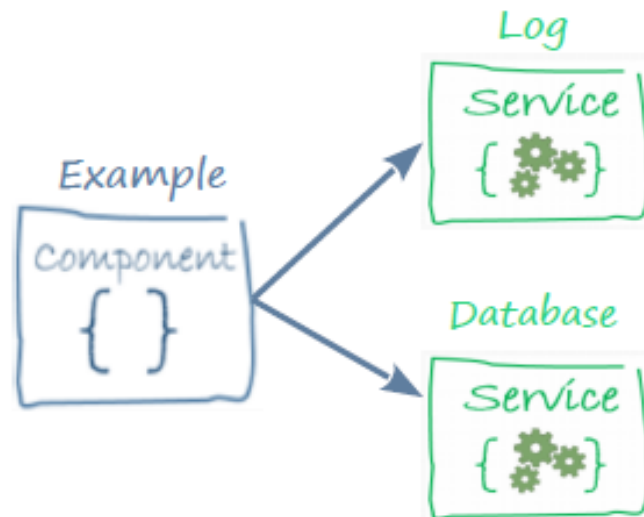


# Servicios en Angular

¿Qué es?



- ▶ Encapsula funcionalidad común a varios componentes
- ▶ Un componente se centra en la presentación de datos, cualquier otra cosa: servicio
- ▶ Los componentes son consumidores de servicios



# Servicios en Angular

- Los **servicios** representan:
  - ❑ La **LÓGICA** de **NEGOCIO**
  - ❑ El acceso a los **DATOS (MODELO)**
- Los **componentes** sólo se encargan de la interacción con el usuario y **llamar a SERVICIOS cuando necesitan acceder a datos.**
- Los servicios **son clases TypeScript**

# Servicios en Angular

- Los **servicios** permiten **compartir información entre clases**:
  - ❑ *Sin que sepan unas de otras*
  - ❑ **Creando una única instancia compartida (patrón SINGLETON)** del servicio.
  - ❑ Mediante **inyección de dependencias**
  - ❑ Cada clase que lo quiera usar, lo **INYECTA en su constructor**.

¿Cómo se crea un  
servicio?



# ¿Cómo se crea un servicio?

---

## Crear un proveedor de servicios

- ▶ Clase decorada con `@Injectable`
- ▶ Puede ofrecer cualquier interfaz

log.service.ts

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

@Injectable()
export class LogService {
  log(msg: any) { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any) { console.warn(msg); }
}
```

- ▶ Con Angular CLI

```
> ng generate service log
```

# ¿Cómo se usa un servicio?

## Inyección de dependencias

- Un componente consume servicios (dependencias)
- Un componente lista sus dependencias como parámetros del constructor
- Angular automáticamente crea los servicios y los inyecta

log.service.ts

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

```
@Injectable()  
export class LogService {  
  log(msg: any) {  
    console.log(msg);  
  }  
}
```

example.component.ts

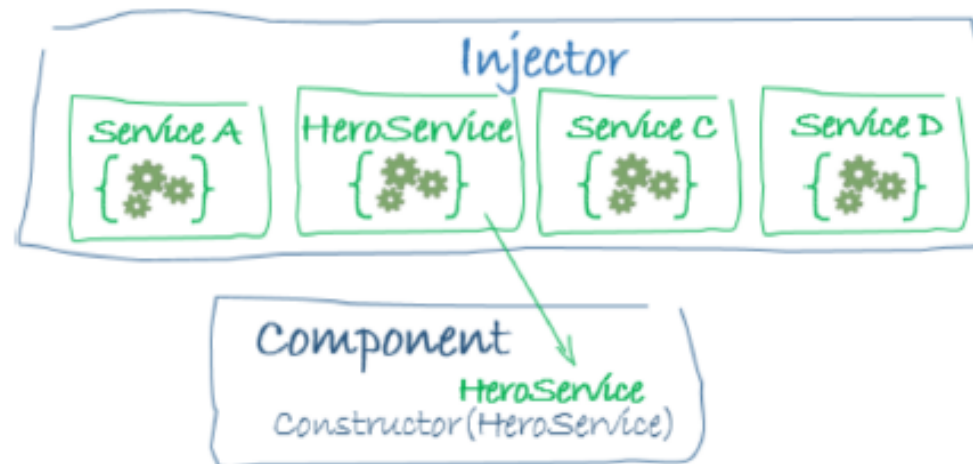
```
import { LogService } from 'log.service';
```

```
@Component({ ... })  
export class ExampleComponent {  
  constructor(private logger: LogService) {  
    this.logger.log('Hello');  
  }  
}
```

# ¿Cómo se usa un servicio?

## Inyector de dependencias

- ▶ Para poder usar un servicio, primero es necesario registrar su proveedor en un inyector
- ▶ Un inyector es un contenedor de servicios, responsable de crearlos, inyectarlos y mantenerlos
- ▶ Una vez creado, el servicio existe hasta que el inyector se destruye





# ¿Cómo se usa un servicio?

## Inyector de dependencias

- ▶ Dos inyectores principales: `@NgModule` y `@Component`
- ▶ Registramos un proveedor de servicios con el metadato `providers`

*Ejemplo con @NgModule*

### app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { LogService } from './log.service';
@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule ],
  providers: [ LogService ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

### log.service.ts

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

@Injectable()
export class LogService {
  log(msg: any) { console.log(msg); }
}
```

Se puede hacer directamente al crear el servicio con:

```
ng generate service hero --module=app
```

# ¿Cómo se usa un servicio?

## Inyector de dependencias

- ▶ Dos inyectores principales: `@NgModule` y `@Component`
- ▶ Registramos un proveedor de servicios con el metadato `providers`

*Ejemplo con @Component*

example.component.ts

```
import { Component } from '@angular/core';
import { LogService } from '../log.service';
@Component({
  selector: 'example',
  templateUrl: './example.component.html',
  providers: [LogService]
})
export class ExampleComponent {
  constructor(private logger: LogService) {
    this.logger.log('Hello world!');
  }
}
```

log.service.ts

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

@Injectable()
export class LogService {
  log(msg: any) {
    console.log(msg);
  }
}
```

# ¿Cómo se usa un servicio?

---

## Inyector de dependencias

- ▶ El inyector `@NgModule` se crea al cargar el módulo (cuando arranca la aplicación): sus servicios no se destruyen nunca
- ▶ El inyector `@Component` se crea al instanciar un componente: sus servicios se destruyen al destruir el componente
- ▶ `@NgModule` vs `@Component`: servicio compartido por múltiples componentes vs servicio privativo de un componente (e hijos)

# ¿Cómo se usa un servicio?

## Inyector de dependencias

- Un servicio puede tener sus propias dependencias

### app.module.ts

```
import { LogService }
  from './log.service';
@NgModule({
  ...
  providers: [LogService]
})
export class AppModule { }
```

### db.service.ts

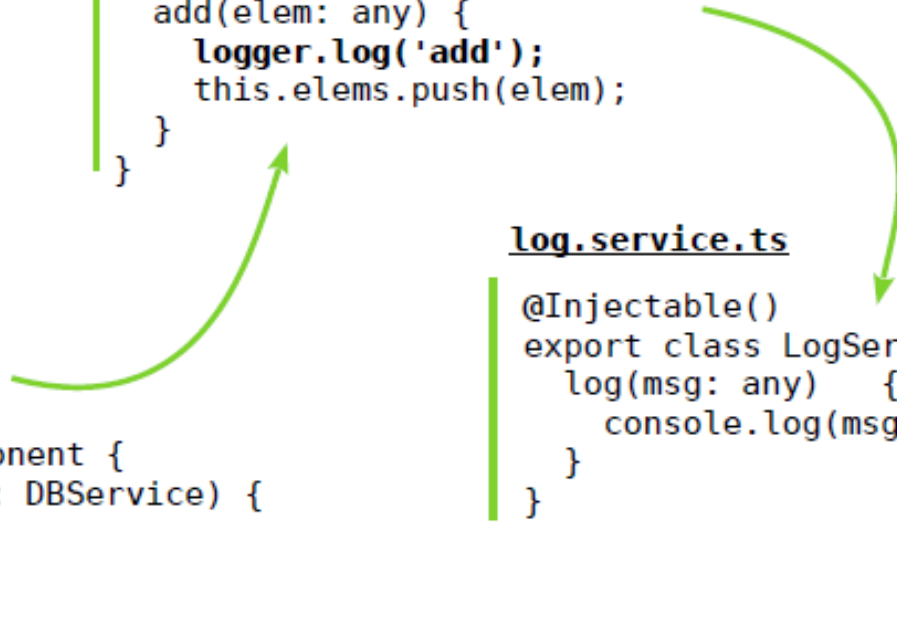
```
@Injectable()
export class DBService {
  private elems: number[] = [];
  constructor(private logger: LogService) { }
  add(elem: any) {
    logger.log('add');
    this.elems.push(elem);
  }
}
```

### example.component.ts

```
import { DBService }
  from './db.service';
@Component({
  providers: [DBService]
})
export class ExampleComponent {
  constructor(private db: DBService) {
    this.db.add('Hello');
  }
}
```

### log.service.ts

```
@Injectable()
export class LogService {
  log(msg: any) {
    console.log(msg);
  }
}
```



# Servicios

## ➤ IMPORTANTE:

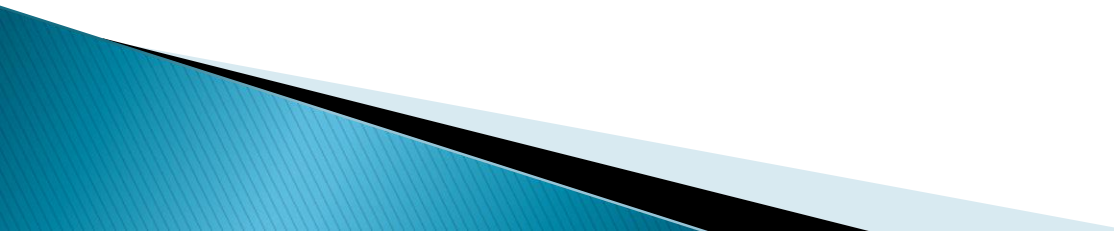
- ❑ *Un servicio puede ser inyectado dentro de otro servicio.*
- ❑ Ver el ejemplo del tutorial HEROES de angular:

[ENLACE](#)

# Principales usos de los servicios



# Principales usos de los servicios

- Servicio de logging
  - Servicio de datos
  - Bus de mensajes
  - Cálculo de impuestos
  - Configuración de la app
  - Hacer peticiones a un servidor
- 

# Servicios Síncronos y Asíncronos





# Servicios Síncronos vs Asíncronos

- Los servicios tal como los hemos visto son **SÍNCRONOS**.
  - Hasta que no obtienen todos los datos, que solicitan, la aplicación queda parada.
- Eso **NO ES MUY PRÁCTICO**.
- Normalmente los **servicios** serán **ASÍNCRONOS**:
  - Lanzas una petición y no se bloquea la aplicación.
  - Cuando la petición obtiene los datos, avisa, y continúa el programa donde le corresponda.
  - Para gestionar este tipo de servicios se usa:
    1. **PROMESAS** (promises) → en Angular 4 mejor NO
    2. **OBSERVABLES** → aconsejable.

Promesas vs Observables

# Servicios Síncronos vs Asíncronos

- Más adelante veremos los OBSERVABLES.
- Por ahora, usaremos servicios SÍNCRONOS