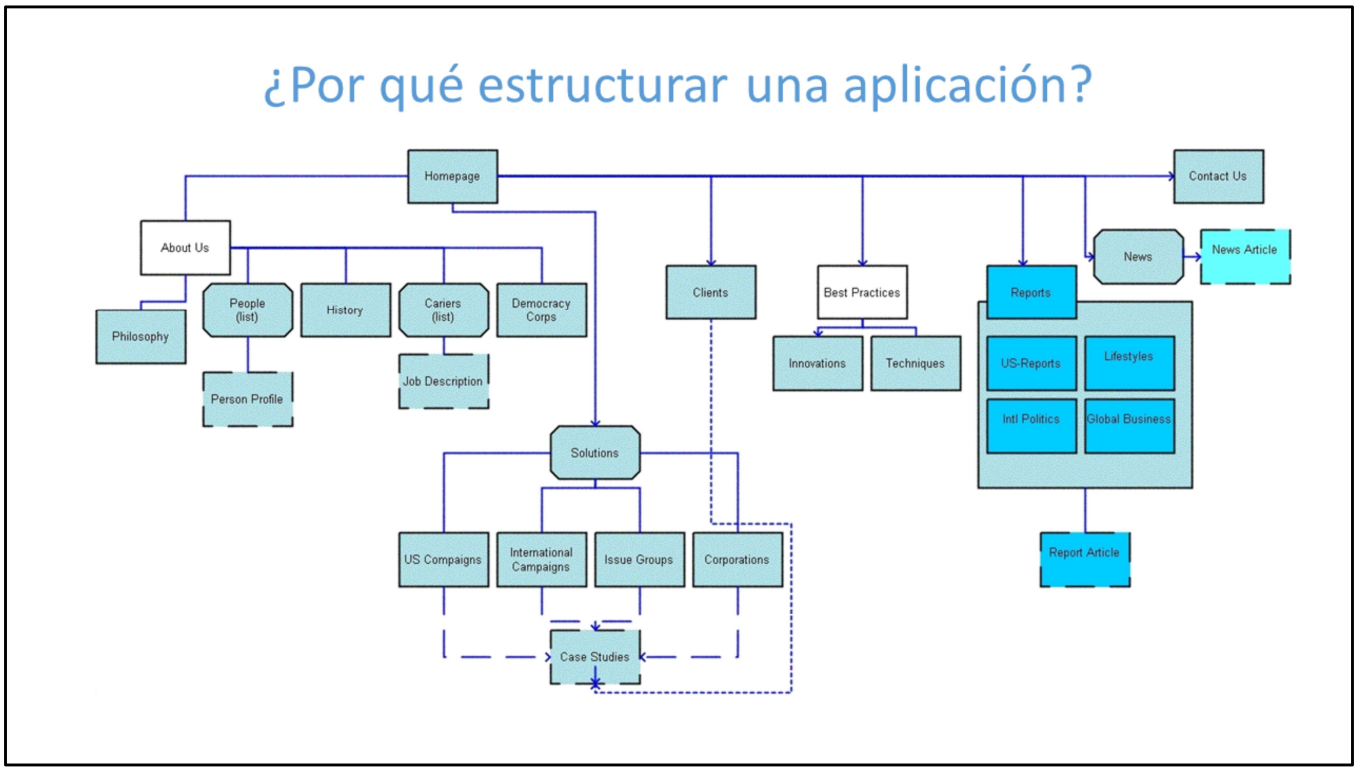
Tema 3 – Fundamentos de Angular

- Qué son los módulos y componentes en Angular
- Estructura y creación de un módulo
- Estructura y creación de un componente
- Qué son las plantillas HTML
- Qué son los servicios y para qué sirven
- *Prácticas-1: Aplicación Datos Deportivos*

¿Qué son los módulos?

- Los módulos estructuran la aplicación
- Son declarativos, no contienen código:
  - Declaran componentes que implementan la funcionalidad
  - Importan módulos compartidos o de Node.js que necesita para dar la funcionalidad
- Nueva aplicación:
  - Módulo AppModule (/app/app.module.ts)
  - Componente AppComponent (/app/app.component.ts)



Una aplicación grande puede tener decenas de componentes y clases, además de un árbol de navegación complejo.
Esto presenta varios problemas:

- La aplicación final es muy grande, y su descarga al cliente y arranque puede ser lento.
- Aunque el usuario vaya a usar solo una pequeña parte de la aplicación, debe descargársela entera, consumiendo más memoria de la que realmente le haría falta.

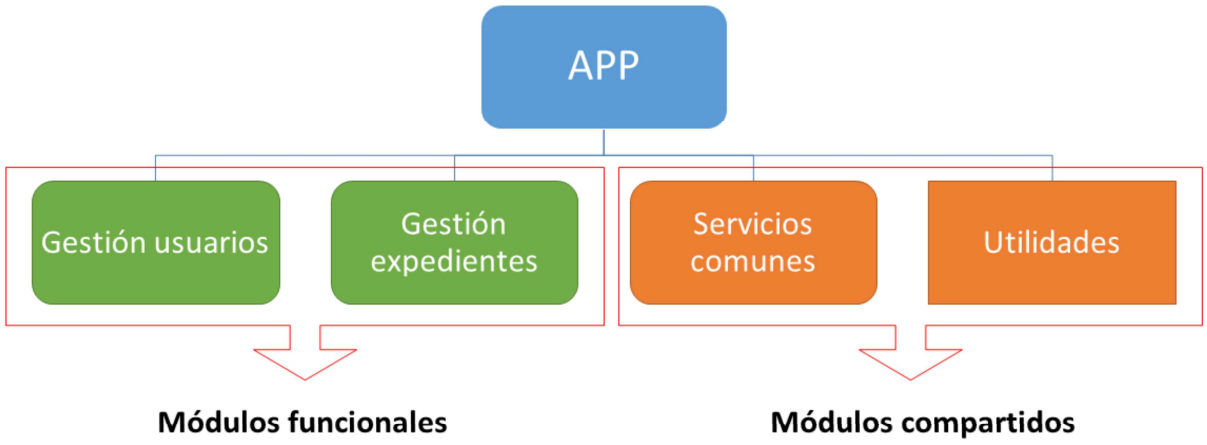
Ventajas de modularizar

- Cada módulo se “empaqueta” por separado (*bundles*)
  - Al arrancar la aplicación no se descargan todos los módulos, solo el inicial (AppModule).
  - Si el usuario necesita usar un componente de otro módulo, se descarga solo en ese momento
- Permite reutilizar componentes propios en varias aplicaciones:
  - Utilidades
  - Servicios comunes

La aplicación final tendrá un "paquete" por cada módulo.

- Al arrancar la aplicación no se descargan todos los módulos, solo el que yo quiero.
- Si el usuario necesita usar un componente de otro módulo, se descarga solo en ese momento.

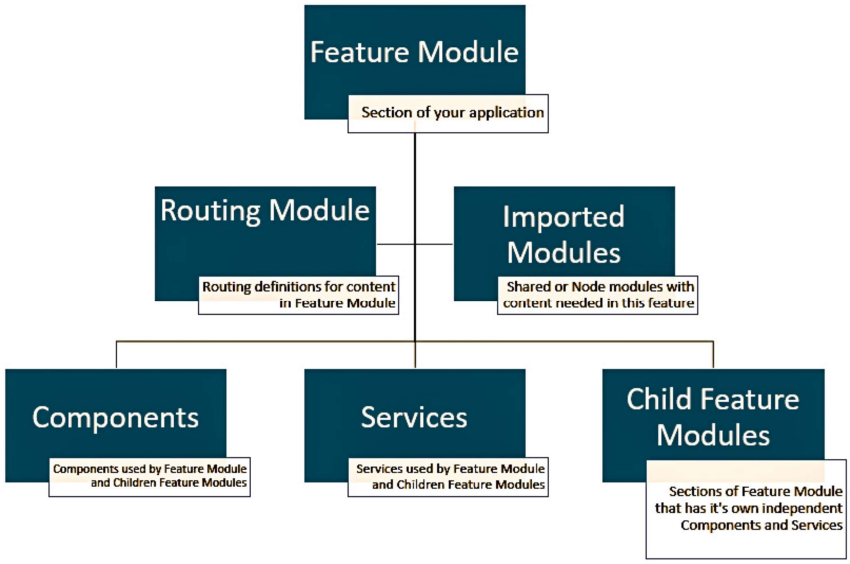
¿Cómo debo estructurar mi aplicación?



Podemos asimilar cada módulo funcional a un subsistema funcional de nuestra aplicación.

Si un componente va a ser usado en varias funcionalidades, es mejor declararlo en un módulo compartido independiente.

Estructura de un módulo funcional

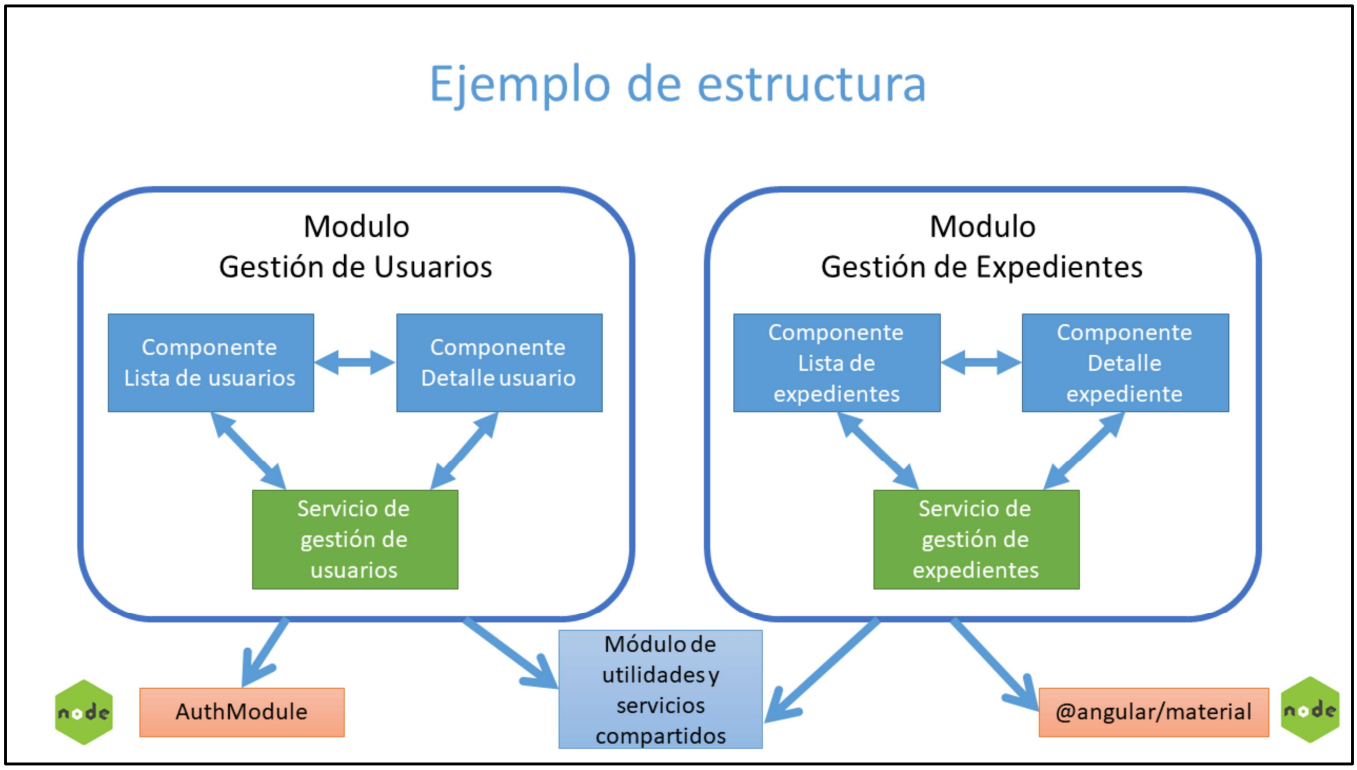


https://www.intertech.com/

Cada módulo funcional:

- Se relaciona con un módulo de navegación (routing), opcionalmente.
- Importa clases y componentes necesarios de otros módulos compartidos por mi aplicación, o de Node. Si no se declaran aquí, no se podrán usar en los componentes del módulo.
- Declaran los componentes del propio módulo, de forma que estén disponibles para el resto de los módulos.
- Declara clases que implementan servicios usados por los componentes propios y de otros módulos
- Declara a su vez módulos hijos, que estructuran la funcionalidad del módulo funcional, si este es muy complejo.

Ejemplo de estructura



Crear un nuevo módulo con Angular CLI

```
> ng generate module nombre-modulo
```

```
> ng g m nombre-modulo
```

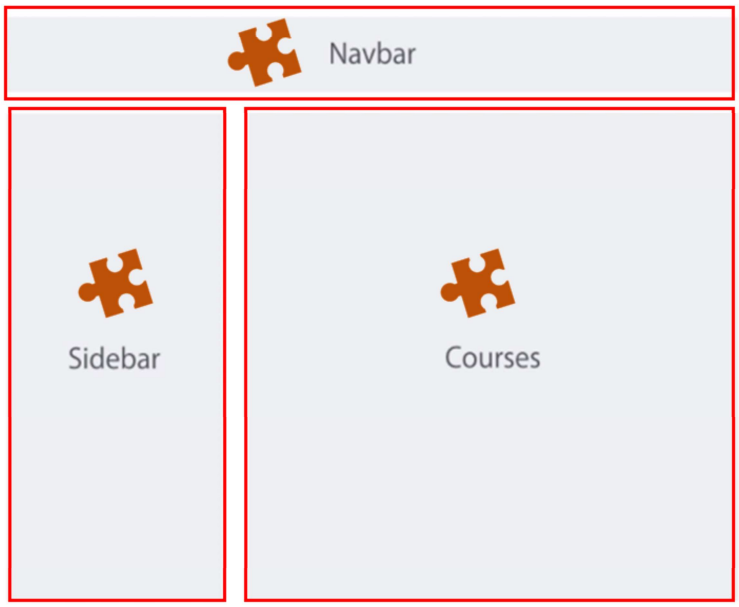
¿Qué son los componentes?

Modelo **V**ista **C**ontrolador

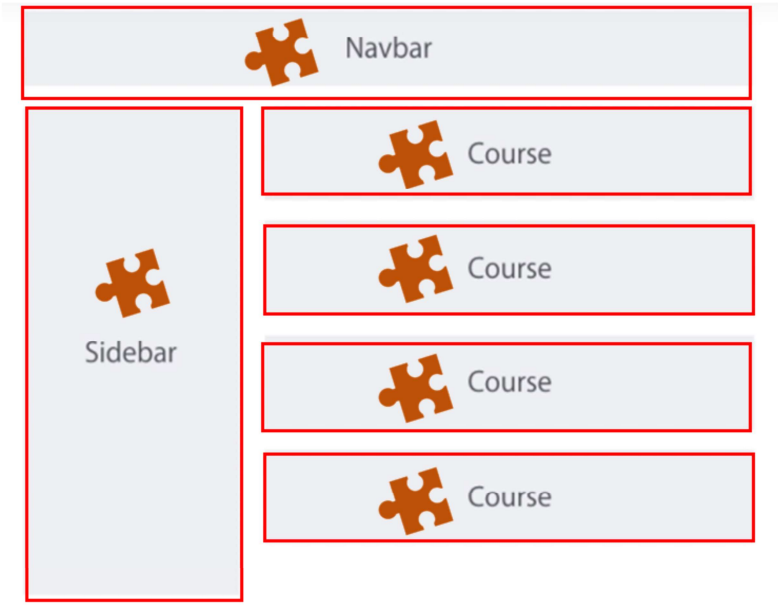
Componentes de una App

app

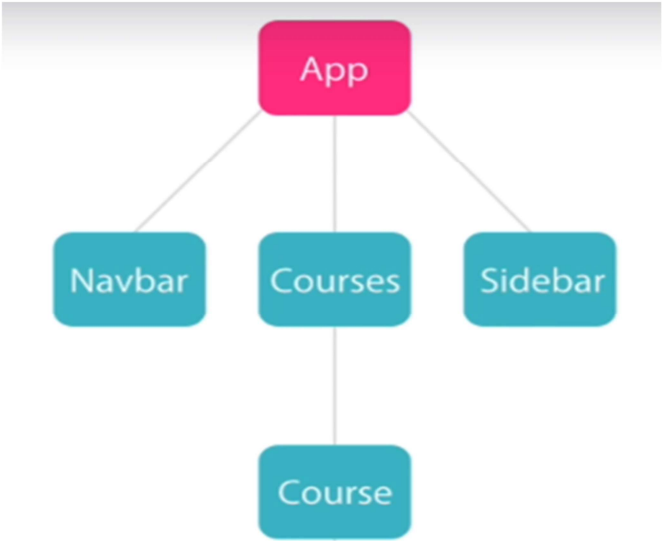
Componentes de una App



Componentes de una App



Arbol de componentes



Creación de componentes

Para crear un componente y poder usarlo, son necesarias 3 acciones:

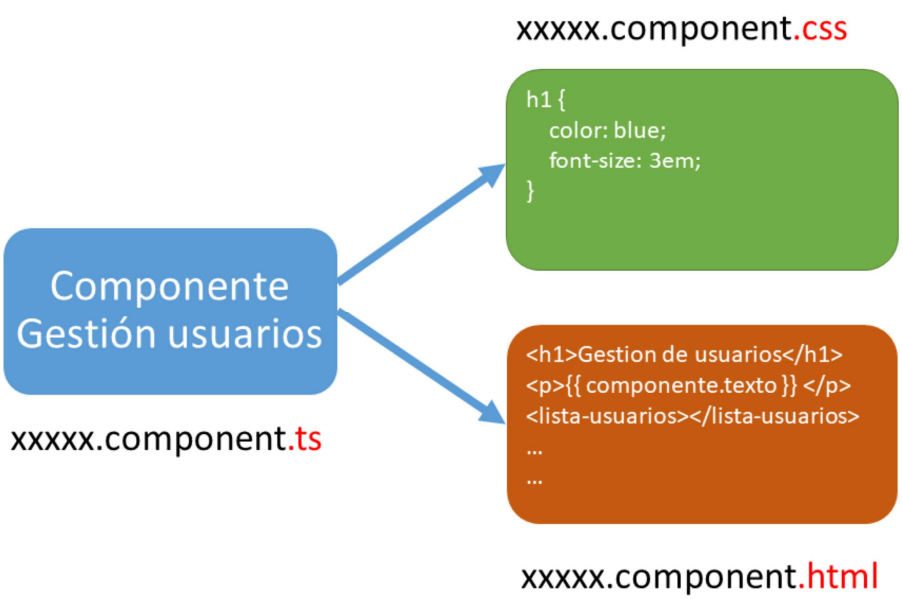
- Crear la clase componente
- Registrar el componente en un módulo
- Añadir el componente a la plantilla HTML

Creación de un componente con Angular CLI

```
> ng generate component nombre-componente
```

```
> ng g c nombre-componente
```

Plantillas HTML



Contenido dinámico en plantillas HTML

Interpolación

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

```
export class MiComponente {
    texto = 'Parrafo con texto'
}
```

Parrafo con texto

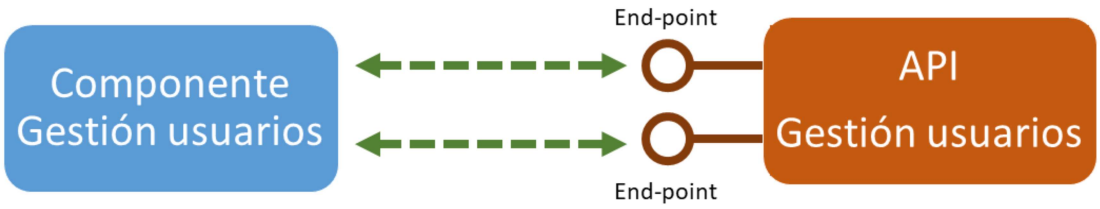
Directivas

```
<ul> <li
*ngFor='let usuario of usuarios'>
</li> </ul>
```

```
export class MiComponente {
    usuarios = [ 'Uno', 'Dos', 'Tres']
}
```

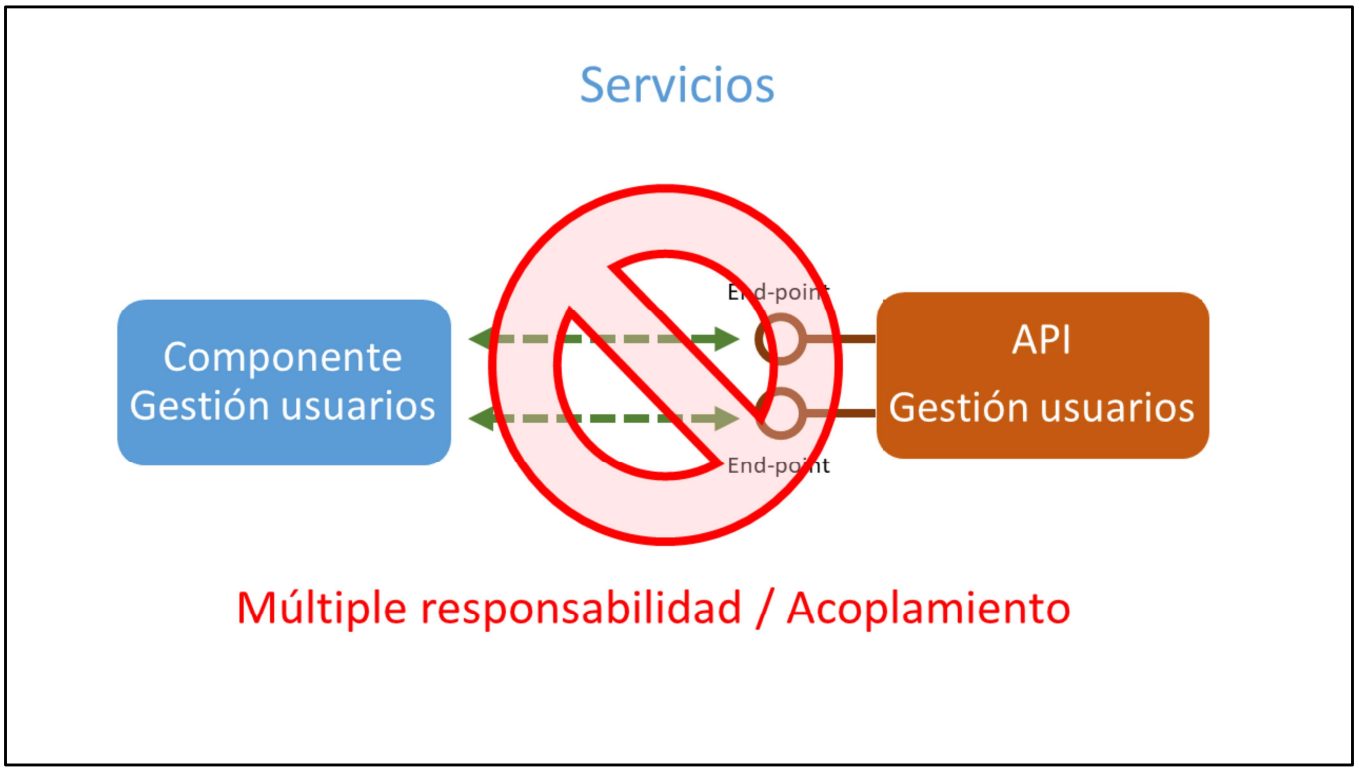
- Uno
- Dos
- Tres

Servicios

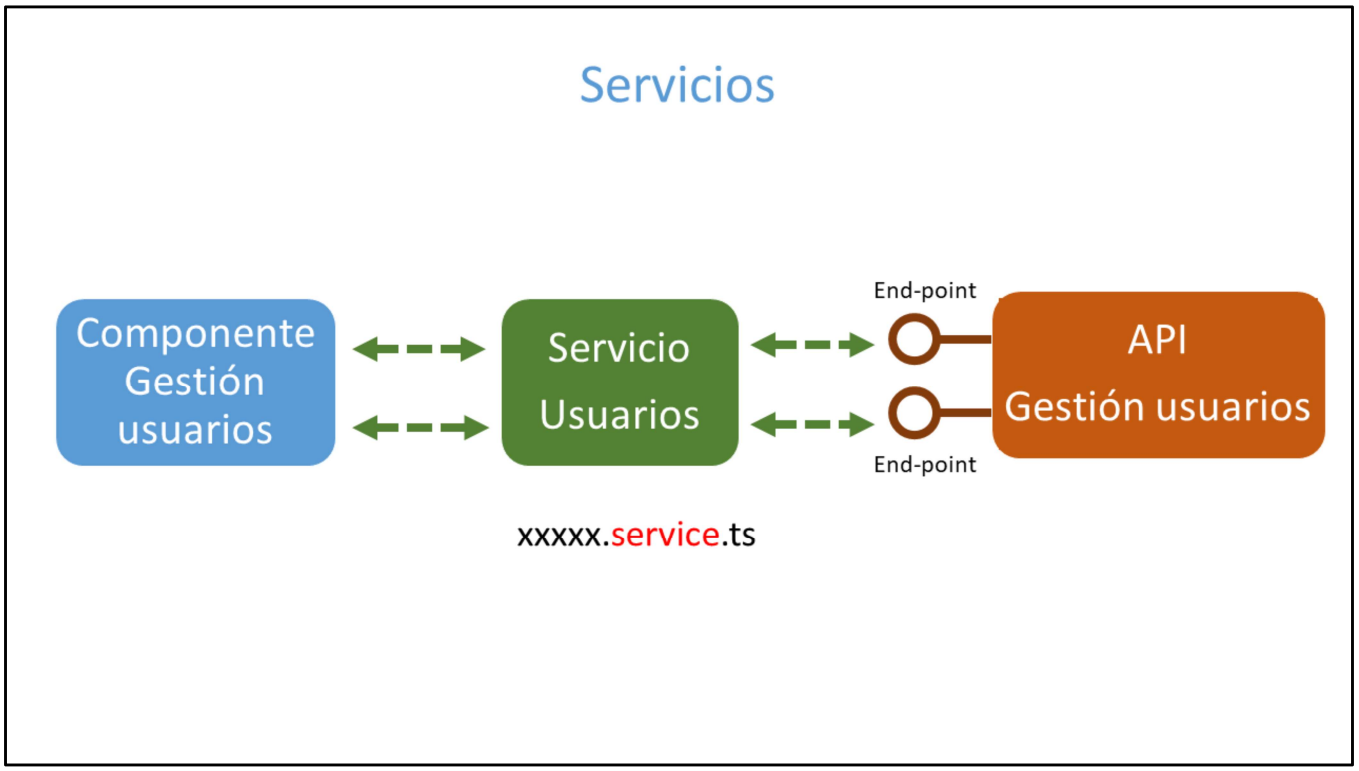


Lo normal será que los datos mostrados se obtengan a partir de servicios publicados por un servidor, mediante una API HTTP.

Angular tiene clases especiales, que veremos más adelante, para conectarse a una API remota y obtener o enviar datos.

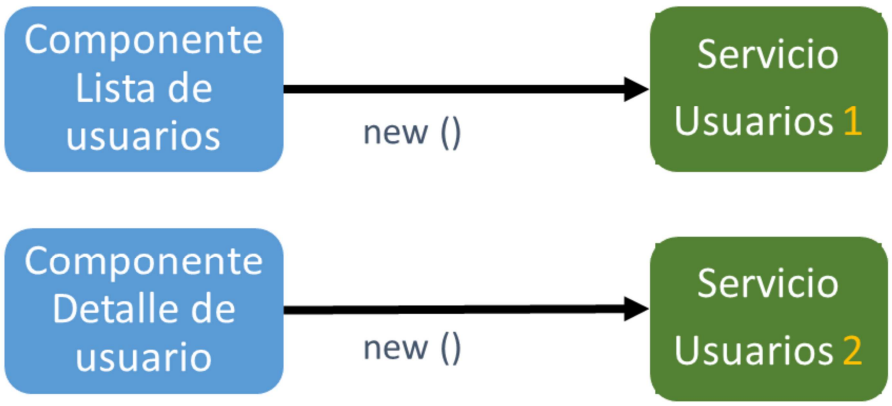


- Sin embargo, no es una buena práctica conectar directamente un componente a una API externa, por 2 motivos:
  - Separación de responsabilidades: En OO, cada elemento se ocupa solo de un aspecto. En Angular, el componente solo debe ocuparse de la interfaz de usuario, no de obtener datos de una API.
  - Provoca lo que se denomina "acoplamiento" entre el componente y la API externa. El acoplamiento no es bueno, ya que hace dependiente al componente de la API, y además dificulta la realización de test unitarios, ya que no hay forma de probar el componente si el servidor de API no está disponible.
  - Provoca duplicación del código de acceso al backend.

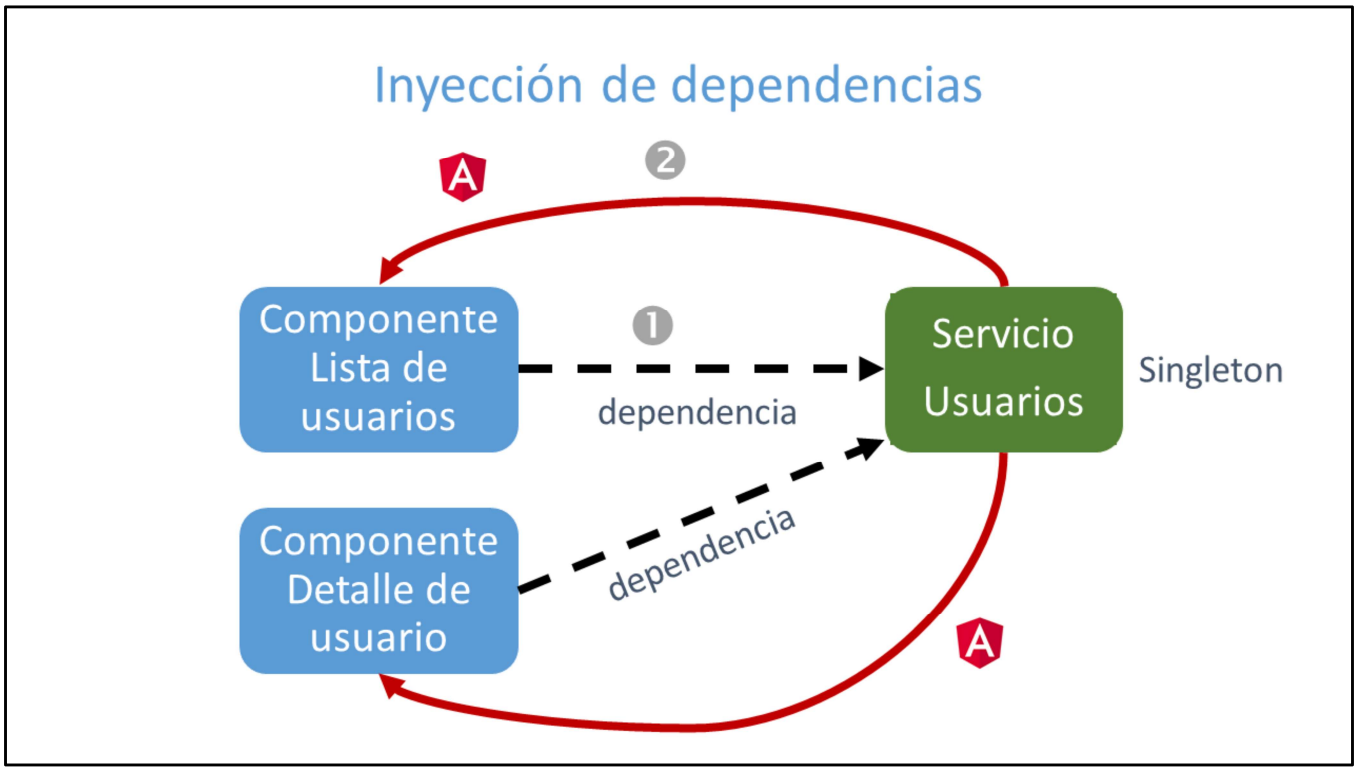


- Por ello debemos usar clases diferentes para conectarnos a APIs de datos o de negocio. Estas clases las denominamos "servicios".
- Por convención, los nombramos xxxxx.service.ts.

Dependencia de un componente con un servicio



- Los componentes usan los servicios, que al fin y al cabo son objetos de la clase que implementa el servicio.
- Si cada componente crea un objeto servicio (con el operador *new*), estaríamos creando varias instancias del mismo servicio, sin necesidad.



- Todos los componentes deberían poder compartir una misma instancia de servicio. Esto es lo que se conoce como patrón "Singleton".

- La gestión de los singletons de los servicios lo podemos hacer nosotros, a nuestro buen entender, o podemos dejar que Angular se encargue de ello. Lo único que debemos hacer es:
    - (1) Decirle a Angular que un componente depende de un servicio,
    - (2) Dejar que Angular obtenga y suministre un objeto de dicho tipo de servicio al componente, sin que este tenga que crearlo o destruirlo.

- Es lo que se denomina "Inyección de Dependencias".

Cómo declarar una dependencia de un componente a un servicio

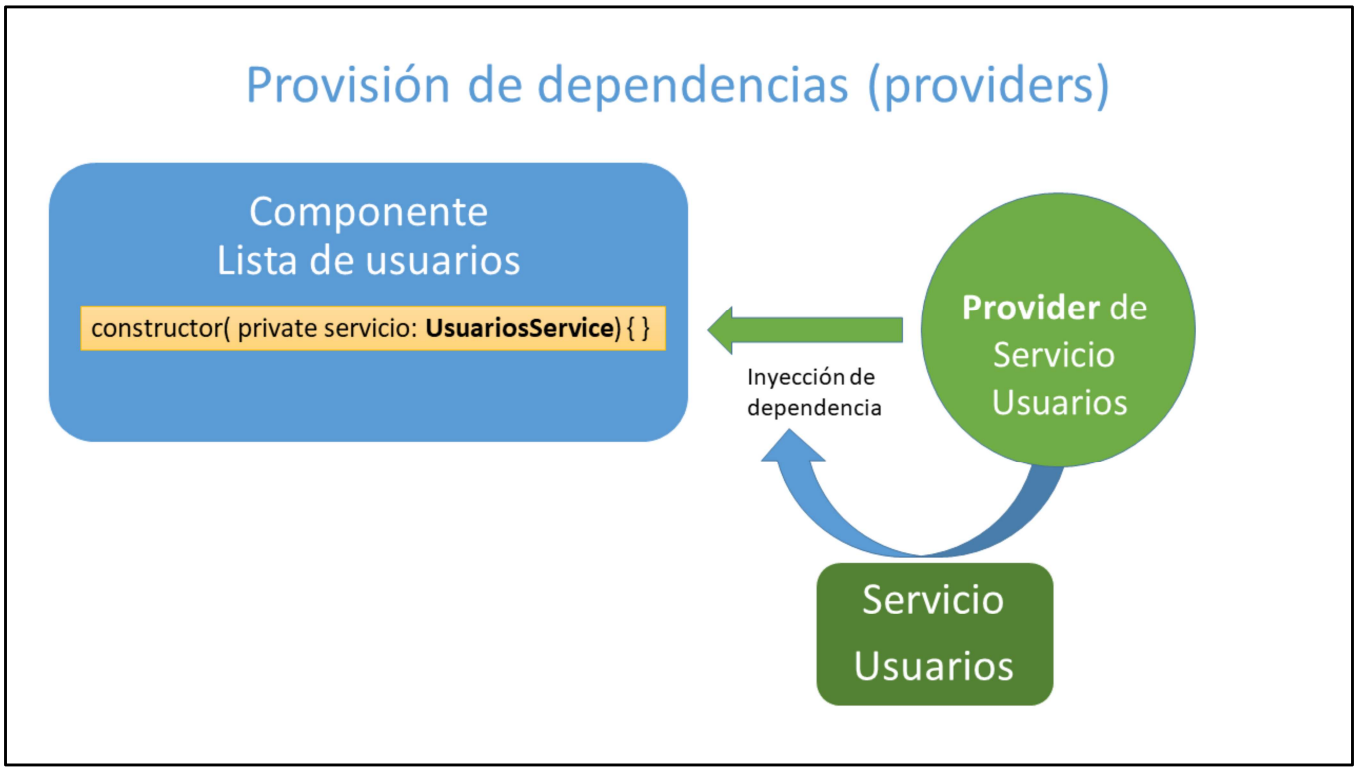
```
export class ListaUsuariosComponent {

   usuarios: string[];

   // Inyecta el servicio en el constructor
   constructor(private usuariosService: UsuariosService) {  }

   leerUsuarios(){
      this.usuarios = this.usuariosService.obtenerListaUsuarios();
   }
}
```

El componente debe disponer de un constructor, y especificar un parámetro del tipo del servicio del que se depende.



Para que Angular pueda "proveer" el servicio al componente, usa los denominados "providers".

El papel básico del provider es buscar un objeto del tipo solicitado e inyectarlo. Por tanto, si un servicio no dispone de un provider, no podrá ser inyectado en ningún otro componente o servicio.

¿Dónde se pone el “provider”?

app.module.ts

```
@NgModule({
  declarations: […],
  imports: […],
  providers: [
    { provider: UsuariosService, useClass: UsuariosService }
  ]
  // O simplemente:
  providers: [ UsuariosService ]
  bootstrap: […]
```

Los providers se declaran en los módulos, en concreto en el atributo “providers” de la anotación “NgModule”.

Lo más simple es declarar los providers para los servicios en el propio módulo AppModule.

Hay 2 sintaxis, la normal y la abreviada.

Puede parecer redundante declarar que si me piden un objeto de tipo UsuariosService, Angular provea un objeto de dicho tipo, pero hay casos en los que puede no ser así, y nos interese devolver un objeto de un tipo parecido, como por ejemplo para la realización de test. En ese caso podríamos devolver un objeto de tipo UsuariosTestService, que suministrara una lista fija de usuarios de prueba en lugar de la lista real.

Creación de un servicio con Angular CLI

```
> ng generate service nombre-servicio
```

```
> ng g s nombre-servicio
```

- Podemos crear un servicio con el comando "ng g s" y el nombre del servicio.
- Con esta operación, se crea la clase servicio y una clase para los test unitarios.
- El fichero ts generado tiene la nomenclatura nombre-servicio.service.ts. Es una convención de nombre similar a la usada para componentes y módulos.
- En la clase generada, vemos una clase normal y corriente. En este caso no vemos una anotación del tipo @Service, no es necesario en Angular.
- Hay una sola anotación @Injectable. Esta anotación le indica a Angular qué clases pueden ser “provistas”, y por tanto inyectadas como dependencias de otros.