# Módulo 1 - Lectura: Uso de módulos e inyección de dependencias

Para registrar una dependencia, tenemos que vincularla a algo que identifique esa dependencia. Esta identificación se llama el token de dependencia. Por ejemplo, si queremos registrar la URL de un API, podemos usar la cadena API_URL como el token. Del mismo modo, si estamos registrando una clase, podemos utilizar la clase en sí misma como su token, como veremos a continuación.

La inyección de dependencia en Angular consta de tres piezas:

- el Proveedor (también conocido como enlace) mapea un token (que puede ser una cadena o una clase) a una lista de dependencias. Le dice a Angular cómo crear un objeto, dado un token.
- el inyector que contiene un conjunto de enlaces y es responsable de resolver las dependencias e inyectándolos al crear objetos
- La dependencia que es lo que se inyecta.

**Proporcionando Dependencias con NgModule**

Si bien es interesante ver cómo se crea directamente un inyector, esa no es la forma típica que usaríamos inyecciones. En cambio, lo que normalmente haríamos es:

• usar NgModule para registrar lo que inyectaremos, se llaman proveedores y
• usar decoradores (generalmente en un constructor) para especificar qué estamos inyectando

Al realizar estos dos pasos, Angular gestionará la creación del inyector y la resolución de las dependencias.

Convirtamos un objeto; partamos de un UserService para ser inyectable como un singleton en nuestra aplicación. Primero, vamos a añadirlo a la clave de proveedores de nuestro NgModule:

```
@NgModule ({
providers: [
UserService // <- añadido aquí
],
})
```

Ahora podemos inyectar UserService en nuestro componente simplemente agregándolo en nuestro constructor como un argumento de entrada.

Lo bueno de esto es que Angular está administrando cuándo crear no solo el componente, sino que también se ocupa de crear sus dependencias, y no tenemos que preocuparnos por hacerlo nosotros mismos.

Cada clase que inyecta el UserService, recibirá el mismo objeto, esto se conoce como el patrón de diseño singleton.

## Buenas prácticas para la definición de Módulos

Cada aplicación Angular tiene al menos un módulo, el módulo raíz.
Usted arranca ese módulo para iniciar la aplicación.
Según los diseñadores de Angular existen diversas buenas prácticas para definir módulos y separar responsabilidades en nuestras aplicaciones.
A continuación, veremos diversas situaciones o patrones de definición de módulos.

**Módulos de funciones/funcionalidades o features**

Los módulos de funciones (o features) son NgModules con el propósito de organizar el código.
Existen categorías en las que agrupamos a los Módulos de funciones, solo con el fin de establecer una mejor comunicación entre nosotros como programadores.
Estas categorías son las recomendadas por el equipo desarrollador de Angular, en Google, y que nos servirán para organizar mejor el código.

**NgModule para Features de Dominio**

Los módulos de features de dominio ofrecen una experiencia de usuario dedicada a un dominio de aplicación en particular, como editar un cliente o realizar un pedido.

Por lo general, tienen un componente superior que actúa como raíz de la característica, y los subcomponentes de soporte descienden de él.

Los módulos de features de dominio consisten principalmente en declaraciones. Solo se exporta el componente superior.

Rara vez tienen proveedores. Cuando lo hacen, la vida útil de los servicios proporcionados debe ser la misma que la vida útil del módulo.

Los módulos de features de dominio se suelen importar exactamente una vez por un módulo de funciones más grande, es decir, un módulo superior.

Pueden ser importados por el módulo de aplicación raíz de una aplicación pequeña que carece de enrutamiento.

**NgModule para Enrutamiento**

Un módulo de enrutamiento proporciona la configuración de enrutamiento para otro módulo y separa las preocupaciones de enrutamiento de su módulo complementario.

Un módulo de enrutamiento normalmente hace lo siguiente:

1. Define las rutas.

2. Agrega la configuración del enrutador a las importaciones del módulo.
3. Agrega proveedores de Guards y proveedores de servicio del módulo.
4. El nombre del módulo de enrutamiento debe ser paralelo al nombre de su módulo complementario, utilizando el sufijo "Enrutamiento" o “Routing”. Por ejemplo, FooModule en foo.module.ts tiene un módulo de enrutamiento llamado FooRoutingModule en foo-routing.module.ts. Si el módulo complementario es el módulo de aplicación raíz, AppRoutingModule agrega la configuración del enrutador a sus importaciones con RouterModule.forRoot (rutas). Todos los demás módulos de enrutamiento son hijos que importan RouterModule.forChild (rutas).
5. Un módulo de enrutamiento reexporta el RouterModule como una convención para que los componentes del módulo complementario tengan acceso a las directivas del enrutador, como RouterLink y RouterOutlet.
6. Un módulo de enrutamiento no tiene sus propias declaraciones. Los componentes, las directivas y los pipes son responsabilidad del módulo de features, no del módulo de enrutamiento.

Un módulo de enrutamiento solo debe ser importado por su módulo complementario.

**NgModule para Servicios (código de dominio)**

Los módulos de servicio proporcionan servicios de utilidad tales como acceso a datos y mensajería. Idealmente, consisten enteramente de proveedores y no tienen declaraciones.
El HttpClientModule de Angular es un buen ejemplo de un módulo de servicio.

El AppModule raíz es el único módulo que debe importar los módulos de servicio.

**NgModule para Widgets**

Un módulo de widgets hace que los componentes, directivas y pipes estén disponibles para los módulos externos. Muchas bibliotecas de componentes de interfaz de usuario de terceros son módulos de widgets.

Un módulo de widget debe consistir completamente en declaraciones, la mayoría de ellas exportadas.

Un módulo de widgets rara vez debería tener proveedores.

Importe módulos de widgets en cualquier módulo cuyas plantillas de componentes necesiten los widgets.

**NgModule para enrutamiento dinámico**

Los módulos de enrutamiento son módulos de funciones de dominio cuyos componentes principales son los objetivos de las rutas de navegación del enrutador.

Los módulos de funciones enrutadas no exportan nada porque sus componentes nunca aparecen en la plantilla de un componente externo.

Un módulo de características enrutadas cargado de forma perezosa no debe ser importado por ningún módulo. Hacerlo provocaría una carga voraz, anulando el propósito de la carga perezosa. Eso significa que no los verá mencionados entre las importaciones de AppModule. Otro módulo debe importar un módulo de funciones enrutadas dinámco para que el compilador conozca sus componentes.

Los módulos de funciones enrutadas rara vez tienen proveedores por los motivos explicados anteriormente. Cuando lo hacen, la vida útil de los servicios proporcionados debe ser la misma que la vida útil del módulo. No proporcione servicios singleton para toda la aplicación en un módulo de funciones enrutadas o en un módulo que importe el módulo enrutado.