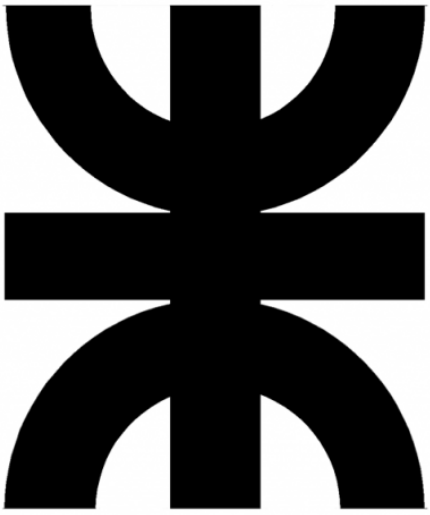

# UNIVERSIDAD TECNOLÓGICA NACIONAL

# FACULTAD REGIONAL CÓRDOBA

**Cátedra: Ingeniería y Calidad de Software - 4K3**

## TRABAJO PRÁCTICO N°6

## Implementación de user stories


**Docentes:**

**Adjunto:** Ing. Covaro, Laura

**Auxiliares de Trabajos Prácticos:**

- Ing. Massano, Cecilia
- Ávila, Pilar

**Integrantes Grupo N° 7:**

- Cañete Julio, Federico - Legajo: 83.184
- Cargnelutti, Lautaro - Legajo: 82.376
- Farace, Florencia Candelaria - Legajo: 82.043
- Luque, Mariano Nicolás - Legajo: 84.777
- Romero, Julieta - Legajo: 86.398


**Fecha de presentación:** 13/09/2023

# Documento de estilo de código

A continuación listamos una serie de buenas prácticas que seguimos para trabajar con Angular

## Single responsibility

Aplica el principio de responsabilidad única (SRP) a todos los componentes, servicios y otros símbolos. Esto ayuda a que la aplicación sea más limpia, más fácil de leer y mantener, y más comprobable.

### 1. Regla de uno

Defina una cosa, como un servicio o componente, por archivo. Considere limitar los archivos a 400 líneas de código.
**¿Por qué?** Un componente por archivo hace que sea mucho más fácil de leer, mantener y evitar colisiones con los equipos de control de código fuente. Evita errores que a menudo surgen al combinar componentes en un archivo donde pueden compartir variables, crear cierres no deseados o acoplamientos no deseados con dependencias.

### 2. Pequeñas funciones

Defina funciones pequeñas, considerando limitarlas a no más de 75 líneas.
**¿Por qué?** Las funciones pequeñas son más fáciles de probar, especialmente cuando hacen una cosa y tienen un propósito. Además, las funciones cortas promueven la reutilización de código, son más fáciles de leer y son más fáciles de mantener.

## Directrices generales de la nomenclatura

Las convenciones de nomenclatura son muy importantes para la mantenibilidad y la legibilidad. Esta guía recomienda convenciones de nomenclatura para el nombre del archivo y el nombre del símbolo.

### 1. Pautas generales

Utilice nombres coherentes para todos los símbolos. Siga un patrón que describa la característica del símbolo y luego su tipo. El patrón recomendado es, por ejemplo: `feature.type.ts`
**¿Por qué?** Las convenciones de nomenclatura ayudan a proporcionar una forma coherente de encontrar contenido de un vistazo. Es decir, deberían

ayudar a encontrar el código deseado más rápidamente y hacerlo más fácil de entender.
La coherencia dentro del proyecto es vital. La coherencia con un equipo es importante. La coherencia en toda la empresa proporciona una enorme eficiencia.
Los nombres de carpetas y archivos deben transmitir claramente su intención.

## 2.Nombres de archivos y símbolos

Utilice nombres coherentes para todos los assets que lleven el nombre de lo que representan. Utilice camelcase para los nombres de clases. Hacer coincidir el nombre del símbolo con el nombre del archivo.

Agregue el nombre del símbolo con el sufijo convencional (como Component, Directive, Module, Pipeo Service) para algo de ese tipo.
Ejemplo: `@Component({ … }) export class PedidoComponent { }`

Asigne al nombre del archivo el sufijo convencional (como .component.ts, .directive.ts, .module.ts, .pipe.tso .service.ts) para un archivo de ese tipo.
Ejemplo: `pedido.component.ts`

## 3.Arranque

Coloque el arranque y la lógica de la plataforma para la aplicación en un archivo llamado `main.ts` . Incluya el manejo de errores en la lógica de arranque. Evite poner la lógica de la aplicación en main.ts. En su lugar, considere colocarlo en un componente o servicio.
**¿Por qué?** Sigue una convención coherente para la lógica de inicio de una aplicación. Sigue una convención familiar de otras plataformas tecnológicas.

# Estructura LIFT

Estructure la aplicación de modo que pueda Localizar el código rápidamente, Identificar el código de un vistazo, Mantener la estructura más plana que pueda e Intentando estar seco*. Defina la estructura para seguir estas cuatro pautas básicas, enumeradas en orden de importancia.
¿Por qué? LIFT proporciona una estructura consistente que escala bien, es modular y facilita el aumento de la eficiencia del desarrollador al encontrar el código rápidamente. Para confirmar su intuición sobre una estructura en

particular, pregunte: ¿puedo abrir rápidamente y comenzar a trabajar en todos los archivos relacionados para esta característica?

## Localizar(located)

Haga que la localización del código sea intuitiva, simple y rápida.
**¿Por qué?** Para trabajar de manera eficiente, debe poder encontrar archivos con rapidez, especialmente cuando no sabe (o no recuerda) los nombres de los archivos. Mantener los archivos relacionados cerca unos de otros en una ubicación intuitiva ahorra tiempo. Una estructura de carpetas descriptiva marca una gran diferencia para  usted y las personas que vienen después de usted.

## Identificar(identify)

Nombra el archivo de manera que sepas al instante lo que contiene y representa. Sea descriptivo con los nombres de archivo y mantenga el contenido del archivo en exactamente un componente. Evite archivos con múltiples componentes, múltiples servicios o una mezcla.
**¿Por qué?** Pase menos tiempo buscando y picoteando el código, y sea más eficiente. Los nombres de archivo más largos son mucho mejores que los nombres abreviados cortos pero oscuros.

## Plano(flat)

Mantenga una estructura de carpetas plana el mayor tiempo posible. Considere la posibilidad de crear subcarpetas cuando una carpeta llegue a siete o más archivos. Además, considere la posibilidad de configurar el IDE para ocultar archivos irrelevantes que distraigan, como archivos generados .jsy .js.map.
**¿Por qué?**

- Nadie quiere buscar un archivo a través de siete niveles de carpetas.
- Una estructura plana es fácil de escanear.
- Por otro lado, los psicólogos creen que los humanos comienzan a tener problemas cuando el número de cosas interesantes adyacentes supera las nueve. Entonces, cuando una carpeta tiene diez o más archivos, puede ser el momento de crear subcarpetas.

Base su decisión en su nivel de comodidad. Use una estructura más plana hasta que haya un valor obvio para
crear una nueva carpeta

## Try to be dry

Aplicar patrón de DRY (No se repita). Evite aplicar en exceso el patrón DRY, para no comprometer la legibilidad.
**¿Por qué?** No repetirse es importante, pero no crucial si sacrifica los otros elementos de LIFT (por eso es tratar de estar seco). Por ejemplo, es redundante nombrar una plantilla hero-view.component.html porque con la extensión html, obviamente es una vista. Pero si algo no es obvio o se aparta de una convención, entonces explíquelo.

## Estructura de carpetas por características

Cree carpetas con el nombre del área característica que representan.
**¿Por qué?**

- Un desarrollador puede localizar el código e identificar lo que representa cada archivo de un vistazo. La estructura es lo más plana posible y no hay nombres repetitivos o redundantes.
- Todas las pautas de LIFT están cubiertas
- Ayuda a evitar que la aplicación se abarrote organizando el contenido y manteniéndolo alineado con las pautas de LIFT.
- Cuando hay muchos archivos, por ejemplo más de 10, localizarlos es más fácil con una estructura de carpetas coherente y más difícil con una estructura plana.

Cree un Ng Module para cada área de funciones ya que este facilita la carga diferida de funciones enrutables, el aislamiento, la prueba y la reutilización de funciones.

## Secuencia de miembros

Coloque las propiedades en la parte superior seguidas de los métodos. Coloque los miembros privados después de los miembros públicos, en orden alfabético.
**¿Por qué?** La colocación de los miembros en una secuencia coherente facilita la lectura y ayuda a identificar instantáneamente qué miembros del componente sirven para qué propósito

**Bibliografía**

https://angular.io/guide/styleguide