**Estándares de codificación:**

En HTML:

El HTML semántico favorece la accesibilidad y la usabilidad.HTML semántico significa marcar cada elemento según su función dentro del documento (valor semántico).Por ejemplo, no marcar un texto como si fuese un encabezado sólo para que coja esa apariencia, ni dejar de marcar un encabezado como tal para darle una apariencia distinta.La apariencia de los elementos debe cambiarse con CSS, para mantener separada la estructura (HTML) de la apariencia (CSS).Solo usar elementos para su propósito original, no cree su propia semántica. Si no puede encontrar uno adecuado, puede considerar elementos generales (como span o div) y usar la clase de atributo global para indicar su significado.

En SCSS:

Se usará como convención de nomenclatura la separación por guiones en el nombre de las clases.Al escribir atributos en el mismo selector, deben agruparse por función y escribirse en el siguiente orden:

Método de diseño y posición (posición / arriba / derecha / abajo / izquierda / flotante / pantalla / desbordamiento)

Modelo de caja (borde / margen / relleno / ancho / alto)

Relacionado con el texto (fuente / altura de línea / alineación de texto / ajuste de palabra)

Efectos visuales (fondo / color / transición / estilo de lista)

Estándares generales :

En blanco al final : Eliminar los espacios en blanco finales

codificación : Utilice UTF-8

Comentario : Alcance, propósito

Se mantendrá una sangría con 2 espacios en blanco:

.example {

color: blue;

}

Estándares para codificar en Ionic/Angular:

**Principio de codificación única :**

Los servicios y componentes están codificados en distintos ficheros

Cada fichero tiene un máximo de 400 líneas

El máximo de líneas por función es de 75 líneas

**Nombrado de ficheros:**

Se utilizan los nombres de manera homogénea.

Se emplea el patrón funcionalidad.tipo.ts en el nombrado de ficheros

Se emplean guiones “-” para separar palabras en nombres descriptivos

Se emplean puntos “.” para separar el nombre descriptivo del tipo

Se emplea el tipo “service” para identificar servicios

Se emplea el tipo “component” para identificar componentes

Se emplea el tipo “pipe” para identificar pipes

Se emplea el tipo “module” para identificar módulos

Se emplea el tipo “directive” para identificar directivas

Los nombres de las clases y otros símbolos están en formato camel case ('helloThereMister')

Se añade a los símbolos el sufijo en formato camel case con su tipo:

ejemplo de camelCase:

**Ejemplo**: EjemploDeUpperCamelCase. lowerCamelCase (o simplemente **camelCase**)

**Selectores de componentes:**

Se usa dashed-case o kebab-case para los nombres de los selectores de los

componentes

**Clases:** Se usa formato upper camel case para el nombrado de clases

**Constantes:** Si el valor de una variable no cambia se define como const Se usa el formato lower came case para su definición

**Interfaces :** Se usa el formato upper camel case para el nombrado de interfaces. No se emplea el prefijo I para su nombrado

**Propiedades y métodos**: Se usa el formato lowercamelcase para su definición

**Estructura de la aplicación y NG-Modules :**Todo el código fuente cuelga de la carpeta src Los ficheros de un mismo componente están agrupados en una única carpeta Se han creado carpetas por funcionalidad El módulo raíz está codificado en el fichero src/app/app.module.ts Se ha creado un NgModule para cada funcionalidad Cada módulo funcional está ubicado en una carpeta con su nombre La funcionalidad compartida se ubica en el fichero app/shared/shared.module.ts