Universidad Autónoma de Aguascalientes

Maestría en Ciencias con opciones a la Computación

Tópicos Selectos 1

Semestre Agosto-Diciembre 2023

ESTÁNDAR DE CODIFICACIÓN

Proyecto: Aula Virtual para personas sin habilidades informáticas

Integrantes:

Luis David Jiménez Martínez

Daniel Enrique Colunga Galván

Docente: Dr. Francisco Javier Álvarez

Aguascalientes, Ags a 22 de Septiembre de 2023

Contenido

ESTANDARES DE CODIFICACIÓN

El Estándar de Codificación es una herramienta fundamental en cualquier proyecto de desarrollo de software, ya que establece las directrices y reglas que se beben guiar para que la escritura de código sea de manera coherente y consistente. Proporciona un conjunto de normas que los desarrolladores deben seguir para asegurar que el código sea legible, mantenible y compatible con los objetivos del proyecto.

En este documento, presentaremos las principales pautas y convenciones que deben seguirse en aspectos clave como nomenclatura y estilo de codificación, para brindar una base sólida en el desarrollo del proyecto.

1 Nomenclatura

Para el nombramiento de archivos, componentes, interfaces, clases, métodos, variables y constantes debe ser en ingles al igual que todas las sentencias especiales del código. Lo único que va a ser en español son las descripciones de los comentarios y también el contenido que va a ser visible para el usuario final en la interfaz gráfica, incluyendo los valores de las variables cuando sea el caso de un valor string.

1.1 Componentes

Los nombres de los componentes deben ser descriptivos y en caso de tener mas de dos palabra, deben estar en formato camelCase.

Se deben evitar nombres genéricos o abreviaturas poco claras.

Los nombres de los archivos de los componentes deben coincidir con el nombre del componente y seguir la convención **.component.ts**, **.component.html**, **.component.css**.

Ejemplo: **nombreComponente.component.ts**

Los componentes deben estar organizados en archivos separados para mantener un código limpio y mantenible.

Se deben utilizar comentarios descriptivos para explicar la funcionalidad de los componentes y las partes importantes del código.

Se debe seguir una estructura de código coherente dentro de los componentes, como el orden de las propiedades, métodos y ciclos de vida del componente.

1.2 Nombres de interfaces

Los nombres de las interfaces deben ser descriptivos y en formato CamelCase.

Se deben evitar nombres genéricos o ambigüos. Los nombres deben reflejar claramente el propósito y la funcionalidad de la interfaz.

Los nombres de las interfaces deben ser significativos y representar el tipo de datos o entidad que se describe.

Evitar nombres cortos o abreviaturas poco claras. Es preferible una mayor longitud si mejora la comprensión del propósito de la interfaz.

En caso de dividir las interfaces en módulos separados, el nombre del módulo debe formar parte del nombre de la interfaz.

Ejemplo **Usuario.ts, Login.ts, ProveedorDeDatos.ts, etc**

1.3 Nombres de clases

Los nombres de las clases deben ser descriptivos y en formato CamelCase.

Utiliza nombres sustantivos en lugar de verbos para las clases. Las clases deben representar entidades, objetos o componentes, y no acciones.

Los nombres de las clases deben comenzar con una letra mayúscula.

Los nombres de las clases deben ser significativos y representar claramente la funcionalidad o entidad que la clase representa.

Evita nombres cortos o abreviaturas poco claras. Es preferible una mayor longitud si mejora la comprensión del propósito de la clase.

Utiliza nombres compuestos si es necesario para una mejor representación del objeto. Por ejemplo, `ControladorDeUsuarios` en lugar de `UsuarioCtrl.`

Evita el uso de caracteres especiales, números o espacios en los nombres de las clases.

1.5 Métodos

Todos los métodos deben estar acompañados de comentarios de documentación.

Los métodos deben devolver un valor significativo. Evitar métodos que no retornan ningún valor (**void**) a menos que sea necesario.

Tamaño y Complejidad: Los métodos deben ser de longitud moderada, generalmente no más de 20 líneas de código.

Cohesión: Cada método debe tener una única responsabilidad. Si un método realiza tareas múltiples, debe ser dividido en métodos más pequeños.

Acoplamiento: Minimiza la dependencia entre métodos. Los métodos deben ser independientes y no deben depender en exceso de otros métodos o clases.

Nombres de Parámetros: Los nombres de los parámetros deben ser descriptivos y seguir la convención camelCase. Ejemplo:

```
enviarCorreo(direccionDeCorreo: string): void {
    // Implementación del método aquí
}
```

1.6 Variables y Constantes

Se deben declarar variables locales utilizando la palabra clave **let**. Para constantes, se utiliza **const** en lugar de **let** para garantizar que no se modifiquen accidentalmente. Evitar el uso de **var.**

Se deben utilizar nombres descriptivos y significativos para las variables. Evitar nombres genéricos o abreviados que dificulten la comprensión.

Se sigue la convención camelCase al nombrar variables. Comenzarán con una letra minúscula y se utilizarán mayúsculas para palabras subsiguientes. Ejemplo:

```
let contador: number = 0;
const maxIntentos: number = 3;
```

Se debe mantener el alcance de las variables lo más limitado posible. Se declararán en el ámbito más cercano donde sean necesarias para evitar posibles conflictos y mejorar la legibilidad.

```
function calcularImpuesto(precio: number): number {
    let tasaImpuesto: number = 0.1; // Declaración local
    return precio * tasaImpuesto;
}
```

Siempre se deben inicializar las variables al declararlas, proporcionando un valor predeterminado cuando sea necesario, ejemplo `let mensaje: string = '';`

2 Estilo de codificación

Se debe hacer uso de tabulaciones para la identación del código y espacios para separar operadores y valores. Cada nivel de sangría se representa mediante una tabulación, y los espacios se utilizan para delimitar operadores y valores.

2.1 Comentarios

Comentarios Explicativos: Se deben usar comentarios de bloque para explicar partes del código que puedan no ser evidentes de inmediato. Esto incluye describir el propósito de una función, el significado de una variable o la lógica detrás de un bloque de código complejo.

Ejemplo:

```
/*
 * Esta función calcula el precio de los productos del carrito.
 * Recibe una lista de productos y devuelve el precio total.
 */
```

Comentarios Concisos: Los comentarios de línea deben ser concisos y proporcionar aclaraciones breves cuando sea necesario. Deben ser claros y no redundantes.

Ejemplo: `// Incrementar el contador en 1`

Cada archivo debe incluir un encabezado que describa su propósito. Esto facilita la identificación de la funcionalidad del archivo.

2.2 Sentencias

Las sentencias se refieren a cómo estructurar y formatear las instrucciones y estructuras de control de flujo en el código.

Sentencias de Control de Flujo: Se deben utilizar sentencias de control de flujo como **if**, **else**, **switch**, y **for** de manera clara y organizada. Utilizar llaves **{}** incluso para sentencias de una sola línea para mayor claridad. Ejemplo:

```
if (usuarioAutenticado) {
    // Código si el usuario está autenticado
} else {
    // Código si el usuario no está autenticado
}
for (let i = 0; i < listaElementos.length; i++) {
    // Código para procesar cada elemento de la lista
}
```

Sentencias de Asignación: Utilizar los operadores de asignación **=**, **+=**, **-=** y otros de manera clara y coherente. Separar los operadores de asignación de los valores con espacios para mejorar la legibilidad. Ejemplo:

```
let total = 0;
total += precioProducto;
```

Uso de Promesas y Observables: Al trabajar con operaciones asincrónicas, como solicitudes HTTP, utilizar promesas y observables de manera adecuada y manejar los casos de éxito y error de manera apropiada. Ejemplo:

```
this.http.get('https://api.com/data')
```

```
.toPromise()
.then((data) => {
    // Manejo de los datos exitosos
})
.catch((error) => {
    // Manejo de errores
});
```