MT SOFTECH

SOLUCIONES EN INFORMÁTICA

¿Cómo mejorar legibilidad del código?

Autor: Diego Hernández

Introducción

Cuando nos dedicamos a desarrollar software nos detendremos la mayoría del tiempo a leer líneas de código, cuando leemos código analizamos el comportamiento de ejecución del programa o detectamos errores y anomalías, entre otras cosas más.
Leer el código se hará más ***tardado*** y/o complicado sobre todo si es ajeno, pues tendremos que ***adaptarnos*** a lógica impuesta y la situación puede empeorar si el código no sigue algunas pautas o aún más si utiliza diferentes estándares en su escritura.
Veremos algunas mejoras que podemos aplicar en nuestro código para hacerlo más legible de cara a un equipo de desarrollo, pues un buen enfoque para la escritura es pensar que lo leeremos nosotros mismos en un futuro o que alguien más lo tendrá que leer en nuestro lugar.

Uso de convenciones del lenguaje

Al escribir nuestros programas debemos tener en cuenta las convenciones recomendadas para el lenguaje que estamos utilizando, por ejemplo, en el lenguaje java se sigue la convención ***Camel Case*** para nombres de clase, identificador de variables, interfaces, etc.
Otro ejemplo son las definiciones de funciones, es recomendable indicar las funciones como acciones (verbos) y con sustantivos (sujetos) en caso de ser necesario el último caso.
Cada lenguaje de programación tendrá sus propias convenciones por aprender, pero algunas de ellas son compartidas, las anteriores se pueden encontrar tanto en java como JavaScript e incluso Python, pero con variaciones.
Usando convenciones nos permitiremos identificar más rápidamente fragmentos de código como las variables, métodos, paquetes, rutas, etc. Además de presentar un código cada vez más limpio.

Variables y métodos auto explicativos

Si bien podemos seguir convenciones como ***Camel Case*** puede que no le saquemos potencial si hacemos algo como lo siguiente:

```
function calculate( numberA, numberB){
    let totalValue = (numberA - numberB) *1.16;

    return totalValue;
}
```

En este ejemplo podemos deducir que es el cálculo de un impuesto con 2 valores de entrada que pudieran ser el total de un producto menos un descuento y el resultado se le agrega un impuesto por 16% del total obtenido, aunque podemos concluir que es así, no tendremos certeza hasta ejecutar el código si es que se trata de un programa muy extenso.
Podemos mejorar este código simplemente cambiando nombres e identificadores de la siguiente manera.

```
function calculateTaxProduct( price, discount){
    let totalAmount = (price - discount) *1.16;

    return totalAmount;
}
```

Con este ejemplo podemos tener mayor seguridad de lo que significa cada cosa que se está utilizando y no nos veremos en la necesidad de someterlo a dudas o análisis para asegurarnos.

Uso de comentarios

Aunque parezca poco relevante, los comentarios nos pueden salvar de perder mucho tiempo de lectura en algo que ya hayamos trabajado, los comentarios son un apoyo para los desarrolladores y entre más específicos sean son mejores, esto no quiere decir que debamos inundar en comentarios nuestro código, pues abusar de ellos puede hacer más difícil o molesta la lectura.

Un buen comentario nos permite identificar el por qué están siendo utilizadas ciertas pautas, métodos o variables en el código, pero si agregamos, además, código auto descriptivo, nos podemos ahorrar algunas líneas de comentarios y mantenerlo legible aún. Veamos un ejemplo:

```
/**
 * Calculates tax from a product and a discount
 *  aplied
 * @param {float} price product´s price
 * @param {float} discount product´s discount
 * @returns the calculated amount from the parameters and 16% on tax
 */
function calculateTaxProduct( price, discount){
    //get first the exact amount from price and discount, then
    //multiply by 1.16 to add tax
    let totalAmount = (price - discount) *1.16;
    return totalAmount;
}
```

Si bien en el ejemplo generamos más líneas de código, tenemos la explicación sobre su comportamiento, descripción de parámetros y variables y especificaciones del resultado esperado.
Si necesitásemos escalar este código y usáramos hipotéticamente la variable de entrada "**price"** 15 veces más, ya no tendríamos que especificar a que pertenece, pues su uso y origen se especificaron desde la generación de la función y si tenemos que retomar la funcionalidad 6 meses después, ya no tendremos que preocuparnos por recordar lo que hacía.