Evidencias de desempeño: definir estándares de codificación de acuerdo a plataforma de desarrollo elegida.

GA7-220501096-AA1-EV02

Introducción

En el ámbito del desarrollo de software, la definición y adherencia a estándares de codificación es fundamental para garantizar la calidad, mantenibilidad y eficiencia del código. Los estándares de codificación establecen una serie de prácticas y convenciones que deben seguirse en un proyecto de software, facilitando la colaboración entre desarrolladores y asegurando la consistencia del código a lo largo del tiempo.

Este documento presenta los estándares de codificación recomendados para diferentes plataformas de desarrollo, abarcando tanto aplicaciones web como móviles. Se incluyen los lenguajes y tecnologías más comúnmente utilizados: JavaScript, HTML, CSS para desarrollo web; Java y Kotlin para aplicaciones Android; Swift para iOS; y prácticas generales aplicables a cualquier entorno de programación.

Objetivos

Fomentar la Consistencia del Código: Establecer un conjunto de reglas que promuevan la uniformidad en el estilo y la estructura del código, facilitando su lectura y mantenimiento.

Mejorar la Calidad del Software: Asegurar que el código cumpla con los estándares de calidad necesarios para minimizar errores y optimizar el rendimiento.

Facilitar la Colaboración: Proporcionar un marco común para que los desarrolladores puedan trabajar juntos de manera efectiva y entender el código de otros miembros del equipo.

Asegurar la Mantenibilidad: Permitir que el código sea fácilmente modificable y expansible, reduciendo los costos de mantenimiento a largo plazo.

Promover las Buenas Prácticas de Programación: Inculcar hábitos de codificación que sean acordes a las mejores prácticas de la industria.

Estándares de Codificación por Plataforma

1. Desarrollo Web (HTML, CSS, JavaScript)

HTML:

Indentación y Espaciado: Utilizar dos o cuatro espacios para la indentación. Mantener la consistencia en todo el documento.

Nombres de Etiquetas: Utilizar etiquetas en minúsculas (<div>, <span>, <a>).

Atributos de Etiquetas: Incluir siempre comillas dobles alrededor de los valores de los atributos (class="header").

Estructura del Documento: Organizar el HTML en secciones claras (header, main, footer) y usar comentarios para separar áreas funcionales del código.

CSS:

Nombres de Clases y IDs: Utilizar nombres significativos y en minúsculas, con guiones para separar palabras (.menu-bar, #main-content).

Indentación: Utilizar dos espacios para la indentación de reglas CSS.

Uso de Selectores: Prefiere selectores de clases sobre IDs para mejorar la reutilización y la especificidad controlada.

Separación de Preocupaciones: Mantener el CSS separado del HTML. Evitar estilos en línea siempre que sea posible.

Orden de las Propiedades: Ordenar las propiedades CSS de manera lógica, preferentemente alfabéticamente o agrupadas por tipo (layout, color, tipografía).

JavaScript:

Uso de Variables: Utilizar let y const en lugar de var para declarar variables. Prefiere const para valores que no cambian.

Nombres de Variables y Funciones: Utilizar el estilo camelCase para nombres de variables y funciones (userName, fetchData).

Formateo de Código: Adherirse a un estilo de formateo consistente, preferiblemente usando herramientas como Prettier o ESLint.

Funciones Flecha: Prefiere las funciones flecha (() => {}) sobre las funciones tradicionales, especialmente para callbacks y funciones cortas.

Manejo de Errores: Utilizar try/catch para manejar errores en el código asíncrono y proporcionar mensajes de error claros.

1. Desarrollo de Aplicaciones Android (Java, Kotlin)

Java:

Nombres de Clases: Utilizar CamelCase para nombres de clases (MainActivity, UserProfile).

Métodos y Variables: Usar camelCase para métodos y variables (getUserName, userAge).

Indentación: Utilizar cuatro espacios para la indentación.

Comentarios: Utilizar comentarios Javadoc para documentar clases, métodos y propiedades (/\*\* ... \*/).

Convenciones de Código: Seguir las convenciones de codificación de Java (nombres de paquetes en minúsculas, etc.).

Kotlin:

Nombres de Clases y Objetos: Usar CamelCase para nombres de clases y objetos (MainActivity, UserProfile).

Variables Inmutables: Utilizar val para variables inmutables y var para variables mutables.

Funciones y Propiedades: Prefiere camelCase para nombres de funciones y propiedades (fetchData, userName).

Concisión: Aprovechar las características concisas de Kotlin (funciones de una sola línea, operadores de nulabilidad, etc.).

Uso de Null Safety: Utilizar el sistema de null safety de Kotlin para evitar errores NullPointerException.

1. Desarrollo de Aplicaciones iOS (Swift)

Swift:

Nombres de Clases y Métodos: Utilizar CamelCase para nombres de clases y métodos (ViewController, fetchUserData()).

Indentación: Utilizar cuatro espacios para la indentación.

Variables y Constantes: Prefiere let para constantes y var para variables que cambian.

Comentarios: Usar comentarios de estilo Swift (// Comentario de una línea, /\* Comentario de múltiples líneas \*/).

Uso de Optionals: Utilizar ? y ! para manejar opcionales de manera segura y clara.

Conclusiones

Implementar y adherirse a estándares de codificación específicos para cada plataforma de desarrollo es vital para el éxito de cualquier proyecto de software. Estos estándares ayudan a garantizar la calidad y la consistencia del código, facilitan la colaboración entre los desarrolladores y hacen que el código sea más fácil de mantener y escalar. Adoptar estas prácticas no solo mejora la productividad, sino que también contribuye a la creación de software robusto y eficiente.

En un entorno de desarrollo diverso, es crucial que cada equipo defina y documente claramente sus propios estándares de codificación, adaptándolos según las necesidades y tecnologías específicas de su proyecto. La continua revisión y actualización de estos estándares es fundamental para mantener la relevancia y la efectividad en un campo tan dinámico como el desarrollo de software.

**Referencias**

(*▷ Qué es un script, para qué sirve y cómo crear uno | InboundCycle*, 2024; *CSS | MDN*, 2024; *Diagrama entidad relación: ¿Qué es para qué sirve? Con ejemplos | Miro*, s. f.; *Diferencias entre DDL, DML y DCL - TodoPostgreSQL*, s. f.; *HTML*, 2023; *Las primeras microtiendas del mercado comienzan a abrir | Hoy*, s. f.; *¿Qué es la Factura Electrónica?*, s. f.; *Qué es una base de datos | Oracle Colombia*, s. f.)

*▷ Qué es un script, para qué sirve y cómo crear uno | InboundCycle*. (2024, enero 8). https://www.inboundcycle.com/diccionario-marketing-online/script

*CSS | MDN*. (2024, junio 6). https://developer.mozilla.org/es/docs/Web/CSS

*Diagrama entidad relación: ¿Qué es para qué sirve? Con ejemplos | Miro*. (s. f.). Recuperado 8 de junio de 2024, de https://miro.com/es/diagrama/que-es-diagrama-entidad-relacion/

*Diferencias entre DDL, DML y DCL - TodoPostgreSQL*. (s. f.). Recuperado 8 de junio de 2024, de https://www.todopostgresql.com/diferencias-entre-ddl-dml-y-dcl/

*HTML: Lenguaje de etiquetas de hipertexto | MDN*. (2023, julio 24). https://developer.mozilla.org/es/docs/Web/HTML

*Las primeras microtiendas del mercado comienzan a abrir | Hoy*. (s. f.). Recuperado 8 de junio de 2024, de https://www.hoy.es/20090819/don-benito/primeras-microtiendas-mercado-comienzan-20090819.html

*¿Qué es la Factura Electrónica?* (s. f.). Recuperado 8 de junio de 2024, de https://micrositios.dian.gov.co/sistema-de-facturacion-electronica/factura-electronica/

*Qué es una base de datos | Oracle Colombia*. (s. f.). Recuperado 8 de junio de 2024, de https://www.oracle.com/co/database/what-is-database/