**POLITICAS DE NOMENCLATURA**

**INTRODUCCION**

En el desarrollo de software, la política de nomenclatura juega un papel fundamental en la legibilidad, la mantenibilidad y la comprensión del código. Un conjunto de reglas y recomendaciones bien definidas para la creación de nombres de elementos (como variables, funciones, clases, bases de datos, etc.) puede marcar la diferencia en la calidad y la eficiencia del desarrollo.

Las políticas de nomenclatura son un conjunto de pautas y mejores prácticas diseñadas para:

* Establecer una base sólida para el desarrollo: La nomenclatura consistente facilita la comprensión del código para todos los miembros del equipo, lo que a su vez reduce el tiempo de aprendizaje y aumenta la eficiencia.
* Mejorar la legibilidad: Los nombres descriptivos y bien estructurados hacen que el código sea más fácil de leer y comprender, incluso para aquellos que no lo han escrito.
* Fomentar la mantenibilidad: Un código con una nomenclatura clara y organizada facilita la identificación de errores y la realización de cambios en el futuro.
* Promover la colaboración: La estandarización de la nomenclatura permite una mejor comunicación entre los desarrolladores, lo que facilita el trabajo en equipo y la colaboración en proyectos.

En este documento, se presenta una política de nomenclatura detallada para el equipo de desarrollo de la compañía. La política abarca diferentes aspectos como:

* Nomenclatura de bases de datos
* Nomenclatura de variables
* Nomenclatura de clases y funciones
* Nomenclatura en Git
* Nomenclatura de servicios API
* Nomenclatura de nombres de archivos y directorios

Se espera que la aplicación de esta política ayude a mejorar la calidad del código desarrollado por la compañía y facilite la colaboración entre los miembros del equipo.

**OBJETIVO**

Incluir prácticas de programación comprobada que conduzcan un código seguro, confiable, comprobable y mantenible.

**NOMENCLATURA NOMBRE DE ARCHIVOS Y DIRECTORIOS**

La nomenclatura de archivos y carpetas juega un papel crucial en la organización y legibilidad de un proyecto de software. Una estructura clara y consistente facilita la búsqueda y comprensión del código, mejorando la eficiencia y la colaboración entre los desarrolladores.

* **Nombres**

Los nombres de los archivos y carpetas deben ser claros y concisos, utilizando palabras en ingles que describan con precisión lo que almacenan.

* + Utilice minúsculas en los nombres de carpetas para evitar posibles problemas al cambiar entre servidores que distinguen mayúsculas de minúsculas. Ejemplo: auth
  + Los nombres de los archivos deben mantenerse lo más cortos posible y al mismo tiempo ser significativos. Ejemplo: authService.js
* **Delimitando por varias palabras**

Consiste en separar palabras en un nombre de archivos sin el uso de espacios en blanco. Los espacios en blanco suelen ser difíciles de interpretar para los lenguajes de programación. La convención que se utilizará es:

* + CamelCase.: El camelCase son palabras que están delimitadas por letras mayúsculas, excepto la palabra inicial y se utiliza para nombrar archivos, Ejemplo: authController

**Recomendaciones**:

* Agrupar los archivos en directorios según su funcionalidad y responsabilidad
* Seguir las convenciones de nomenclatura específicas del framework o biblioteca que estés utilizando
* El nombre del archivo del backend referencia a la base de datos se concatenará con la palabra "BD". Ejemplo: gestorVideoBD.

**NOMENCLATURA BASES DE DATOS**

Las políticas de nomenclatura de bases de datos son importantes por dos razones: facilitar a los desarrolladores la comprensión de la base de datos y facilitar su lectura mediante herramientas automatizadas.

* **Nombres**

Los nombres deben ser claros y concisos, utilizando palabras en ingles que describan con precisión lo que almacenan.

* + Las tablas, vistas y otras relaciones que contienen datos deben tener nombres en singular, no en plural. Ejemplo: team\_member
  + Los campos de clave primaria de una sola columna deben tener el prefijo “id” seguido del nombre de la tabla: Ejemplo: id\_person
  + Los campos de llave foránea deben ser una combinación del nombre de la tabla a la que se hace referencia con el prefijo “fk”. Ejemplo: fk\_id\_actor
* **Delimitador**

Consiste en separar palabras en un nombre sin el uso de espacios en blanco. Los espacios en blanco suelen ser difíciles de interpretar para los lenguajes de programación. Las convenciones que se utilizará son:

* + CamelCase.: El camelCase son palabras que están delimitadas por letras mayúsculas, excepto la palabra inicial y se utiliza para nombrar la base de datos. Ejemplo: gestorVideoBD
  + Snakecase: El Snackecase son palabras que están delimitadas por un guion bajo. Se utiliza para nombrar tablas y campos de la respectiva tabla. Ejemplo: team\_member

**NOMENCLATURA DE VARIABLES**

La denominación de las variables es un aspecto importante para que el código sea legible. Nombrar las variables sigue una idea simple, en la cual es crear variables que describen la función y que siga un tema consistente en todo el código.

* **Creación de objetos**
  + Todas las variables deben declararse antes de usarse y utilice comas al final
  + Agregar declaraciones breves en una sola línea. Ejemplo:

let fruits = [‘apple’, ‘pineapple’];

* **Nombres de variables**

Los nombres de las variables deben ser claros y concisos, utilizando palabras en ingles que describan con precisión el valor que almacenan.

* + En lugar de utilizar nombres genéricos como "date", se recomienda usar nombres más específicos como "publicationDate" o "birthDate".
  + Para las variables booleanas, se recomienda utilizar un prefijo como "is” para indicar si el valor es verdadero. Esto ayuda a mejorar la legibilidad del código y facilita la comprensión del significado de la variable. Ejemplo: isActiveClass
  + Las constantes deben escribirse en mayúsculas porque son variables que no cambian. Ejemplo: TIME\_LIMIT
* **Delimitando por varias palabras**

Consiste en separar palabras en un nombre de variables sin el uso de espacios en blanco. Los espacios en blanco dentro de las variables suelen ser difíciles de interpretar para los lenguajes de programación. Las convenciones que se utilizará son:

* + CamelCase.: El camelCase son palabras que están delimitadas por letras mayúsculas, excepto la palabra inicial y se utiliza para nombrar variables. Ejemplo: sumAngules
  + Snakecase: El Snackecase son palabras que están delimitadas por un guion bajo y se utiliza para nombrar constantes, para diferenciarlas de las variables y facilitar su identificación. Ejemplo: MAX\_ATTEMPTS

De acuerdo a lo anterior se agrega los siguientes ejemplos:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Descripción** | **Tipo** |
| publicationDateVideo | Almacena la fecha de publicación de un video. | CamelCase |
| isActiveUser | Indica si un usuario está activo o no. | CamelCase, prefijo "is" |
| MAX\_ATTEMPTS | Número máximo de intentos permitidos. | SnakeCase, constante |

**Recomendaciones**:

* Utilizar nombres largos puede incluso ser más susceptible a errores tipográficos a la hora de escribirlo. La longitud recomendable es de 2 a 4 palabras o entre 8 y 20 caracteres.
* Los nombres deben empezar con una letra, nunca con un dígito.
* No pueden ser una palabra reservada del lenguaje
* Usar comentarios para dar una explicación detallada

**NOMENCLATURA DE CLASES Y FUNCIONES**

Al igual que las convenciones de nomenclatura de variables, las funciones y clases también deben seguir una estructura similar de títulos descriptivos delimitados con las convenciones descritas anteriormente. Un aspecto importante de la denominación es garantizar que las clases, funciones y variables se puedan distinguir entre sí.

* **Nombres de funciones y clases**

Los nombres de las clases y funciones deben ser claros y concisos, utilizando palabras en ingles que describan con precisión el valor que almacenan.

* + Los nombres de las funciones deben ser especifico y contener verbos o frases verbales que expliquen lo que hace la función. Ejemplo: getBooks(url);

Además, los argumentos de la función deben ser incluidos en el nombre de la función si son relevantes para su comprensión. Ejemplo: getUserByEmail(email)

* + Los nombres de las clases deben ser sustantivos que representen el tipo de objeto que define la clase. Ejemplo: User
* **Delimitando por varias palabras**

Consiste en separar palabras en un nombre sin el uso de espacios en blanco. Los espacios en blanco suelen ser difíciles de interpretar para los lenguajes de programación. Las convenciones que se utilizará son:

* + CamelCase.: El camelCase son palabras que están delimitadas por letras mayúsculas, excepto la palabra inicial y se utiliza para nombrar funciones. Ejemplo: usuarioAdmin
  + Pascalcase: El Pascalcase son palabras que están delimitadas por letras mayúsculas y se utiliza para nombrar clases, para diferenciarlas de las variables y facilitar su identificación. Ejemplo: BankAccount()

De acuerdo a lo anterior se agrega los siguientes ejemplos:

|  |  |  |  |
| --- | --- | --- | --- |
| **Clase** | **Descripción** | **Función** | **Descripción** |
| Person | Representa a una persona. | createPerson(name, lastName) | Crear una nueva persona |
| Book | Representa un libro. | getBooks(author, genre) | Obtener una lista de libro por autor y nombre |
| Calculator | Realiza operaciones matemáticas. | add(number1, number2) | Suma de dos números |

**Recomendaciones**:

* Elegir nombres relevantes al contexto del código
* Los nombres deben empezar con una letra, nunca con un dígito.
* Utilizar nombres en sustantivo y singular
* Usar palabras completas, evitar acrónimos y abreviaturas en caso de no ser representativos, por ejemplo, URL.
* Usar comentarios para dar una explicación detallada indicando lo que hace la función y que parámetros recibe y retorna

**NOMENCLATURA DE SERVICIOS API**

La nomenclatura en el diseño de APIs desempeña un papel crucial en la claridad, coherencia y comprensión tanto para los desarrolladores que crean las APIs como para aquellos que las consumen. Al establecer pautas y convenciones sólidas, se facilita la comunicación entre diferentes partes del sistema y se mejora la mantenibilidad del código. En este contexto, la nomenclatura abarca desde los nombres de servicios y rutas hasta la estructura de las respuestas.

* **Nombres**

Los nombres de los servicios y URI deben ser claros y concisos, utilizando palabras en ingles que describan con precisión el valor que almacenan

* **Operaciones api y URI**

El protocolo HTTP define una serie de métodos que asignan significado semántico a una solicitud. Los métodos HTTP comunes utilizados por la mayoría de las API web RESTful son:

* + GET recupera una representación del recurso en el URI especificado. El cuerpo del mensaje de respuesta contiene los detalles del recurso solicitado.
  + POST crea un nuevo recurso en el URI especificado. El cuerpo del mensaje de solicitud proporciona los detalles del nuevo recurso. Tenga en cuenta que POST también se puede utilizar para desencadenar operaciones que en realidad no crean recursos.
  + PUT crea o reemplaza el recurso en el URI especificado. El cuerpo del mensaje de solicitud especifica el recurso que se creará o actualizará.
  + PATCH realiza una actualización parcial de un recurso. El cuerpo de la solicitud especifica el conjunto de cambios que se aplicarán al recurso.
  + DELETE elimina el recurso en el URI especificado.

El efecto de una solicitud específica debería depender de si el recurso es una colección o un elemento individual.

Céntrese en las entidades comerciales que expone la API web. Por ejemplo, en un sistema de comercio electrónico, las entidades principales pueden ser los clientes y los pedidos. Se puede crear un pedido enviando una solicitud HTTP POST que contenga la información del pedido. La respuesta HTTP indica si el pedido se realizó correctamente o no. Cuando sea posible, los URI de recursos deben basarse en sustantivos (el recurso) y no en verbos (las operaciones en el recurso).

La siguiente tabla muestra un ejemplo de las políticas que se debe adoptar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Recurso** | **Post** | **Get** | **Put** | **Delete** |
| /customers | Crear un nuevo cliente | Recuperar todos los clientes | Actualización masiva de clientes | Actualización masiva de clientes |
| /customers/:id | Error | Recuperar los detalles del cliente 1 | Actualizar los datos del cliente 1 si existe | Eliminar cliente 1 |
| /customers/:id/orders | Crear un nuevo pedido para el cliente 1 | Recuperar todos los pedidos del cliente 1 | Actualización masiva de pedidos para el cliente 1 | Eliminar todos los pedidos del cliente 1 |

* **Cuerpo de petición** 
  + El servicio de API debe devolver mensajes que contengan el código de estado HTTP correcto para permitir al cliente determinar cómo manejar el resultado, los encabezados HTTP apropiados para que el cliente comprenda la naturaleza del resultado y un cuerpo con el formato adecuado para permitir al cliente analizar el resultado.

Por ejemplo, una operación POST debería devolver el código de estado 201 (Creado) y el mensaje de respuesta debería incluir el URI del recurso recién creado en el encabezado Ubicación del mensaje de respuesta.

* + El servicio API proporcionará respuestas en formato JSON, que contendrán dos campos principales: msg (mensaje de la respuesta) y body. A continuación, se muestra un ejemplo de cómo se estructurará la respuesta:

{

"msg": "Operación exitosa",

"body": {

"orderId": 1,

"orderValue": 99.90,

"productId": 1,

"quantity": 1

}

}

* **Delimitando por varias palabras**

Consiste en separar palabras en un nombre sin el uso de espacios en blanco. Los espacios en blanco suelen ser difíciles de interpretar para los lenguajes de programación. Las convenciones que se utilizará son:

* + CamelCase.: El camelCase son palabras que están delimitadas por letras mayúsculas, excepto la palabra inicial y se utiliza para nombrar parámetros y cuerpo de la petición. Ejemplo: usuarioAdmin

**Recomendaciones:**

* Crear la Api para poder llamarla de manera independencia de la plataforma sin afectar la funcionalidad. Cualquier cliente debería poder llamar a la API, independientemente de cómo se implemente la API internamente. Esto requiere utilizar protocolos estándar y tener un mecanismo mediante el cual el cliente y el servicio web puedan acordar el formato de los datos a intercambiar.
* La API web debería poder evolucionar y agregar funcionalidades independientemente de las aplicaciones cliente. A medida que la API evoluciona, las aplicaciones cliente existentes deberían seguir funcionando sin modificaciones. Todas las funciones deben ser detectables para que las aplicaciones cliente puedan utilizarlas por completo.
* Intentar mantener los URI relativamente simples. Una vez que una aplicación tiene una referencia a un recurso, debería ser posible utilizar esta referencia para buscar elementos relacionados con ese recurso.

**NOMENCLATURA DE GIT**

* **Nombres**

Los nombres del repositorio y de las ramas deben ser claros y concisos, utilizando palabras en ingles que describan con precisión del funcionamiento.

* + Utilizar minúsculas para los nombres de las ramas
  + No utilizar guiones continuos o guiones finales
  + Evitar la puntuación, los espacios, los guiones bajos o cualquier carácter no alfanumérico.
* **Delimitando por varias palabras**

Consiste en separar palabras en un nombre sin el uso de espacios en blanco. Los espacios en blanco suelen ser difíciles de interpretar, por lo tanto, se utilizará guiones. Por ejemplo: feature/new-login

* **Branch**

En las ramas auxiliares se debe tener la rama Feature, la rama Release y la Rama Hotfix, que puede instanciarse todas las veces que se consideren necesarias:

* La rama Feature, para nuevas características, nuevos requisitos o nuevas historias de usuario.
* La rama Release, para estandarizar o cortar una serie de código que ha estado desarrollándose en la rama Develop, se saca una rama de este tipo, se mergea y ahí se depura.
* La rama Hotfix, que habitualmente se utiliza para código para depurar el código que venga de producción, por haberse detectado un defecto crítico en producción que deba resolverse, al que se le va a hacer una Release puntual para corregirlo.
* La rama docs que habitualmente se utiliza para escribir, actualizar o corregir documentación

Por lo anterior para nombrar los nombres de las sucursales con el uso de prefijos feature, release,hotfix o docs según sea el caso.

Ejemplo:

* feature/login-system
* hotfix/critical-security-issue
* release/v1.0.1
* docs/api-endpoints