Buenas prácticas para el uso de tipos en TypeScript

Jerónimo Eliud Carrillo Lucio 26/08/2024 Nombre del profesor: [Digital NAO1

Comprobación estricta de datos.

Esta práctica consiste en detectar esos errores furtivos que pueden colarse en tu código incluso antes de que ocurran.

Características

1. Consiste en asegurarse de que los tipos de tus variables coinciden con los tipos que esperas que sean. Esto significa que si declaras que una variable es de tipo cadena, TypeScript se asegurará de que el valor asignado a esa variable sea efectivamente una cadena y no un número.
2. Activar la comprobación estricta de tipos es tan sencillo como añadir "strict": true a tu archivo tsconfig.json.
3. Con esto te aseguras de que tu código funciona según lo previsto, lo que te ahorrará tiempo y frustraciones a largo plazo.

Esta característica se puede aplicar en los formularios y catálogos donde el usuario captura información la cual no puede estar alineada con el tipo de datos que se espera.

Inferencia de tipos.

Es la capacidad del compilador de TypeScript para determinar automáticamente el tipo de una variable basándose en el valor que se le asigna.

Características

A. No tienes que especificar explícitamente el tipo de una variable cada vez que la declaras. En su lugar, el compilador mirará el valor e inferirá el tipo por ti.
B. Es especialmente útil cuando se trabaja con tipos complejos o cuando se inicializa una variable con un valor devuelto por una función.
C. A veces es mejor ser explícito con los tipos, especialmente cuando se trabaja con tipos complejos o cuando quieres asegurarte de que se utiliza un tipo específico.

Esta característica se puede aplicar en todo el proyecto y ayuda a ver más limpio tu código.

Linters.

Son herramientas que pueden ayudarte a escribir mejor código aplicando un conjunto de reglas y directrices. Pueden ayudarte a detectar errores potenciales y mejorar la calidad general de tu código.

Características

D. Hay varios linters disponibles para TypeScript, como TSLint y ESLint, que pueden ayudarte a imponer un estilo de código consistente y a detectar errores potenciales.
E. Se pueden configurarse para comprobar cosas como la falta de punto y coma, variables no utilizadas y otros problemas comunes.

Esta característica se puede aplicar en todo el proyecto y ayuda a ver más limpio tu código, robusto y optimizado.

Desestructuración en las propiedades.

La desestructuración, introducida en ECMAScript 6 (ES6), es una JavaScript función que permite extraer varios datos de una matriz u objeto y asignarlos a sus propias variables.

Ejemplo:

```
const object = {
   objname: "obj",
   scope: "this",
};
const oName = object.objname;
const oScop = object.scope;
const { objname, scope } = object;
```

Nomenclatura estándar.

La aplicación de una convención de nomenclatura mantiene la coherencia de la base de código y reduce la sobrecarga a la hora de pensar en cómo nombrar una variable. Le recomendamos lo siguiente:

- Se usa camelCase para nombres de variables y funciones.

- Se usa PascalCase para nombres de clases y nombres de interfaz.
- Se utiliza camelCase para los miembros de la interfaz.
- Se utiliza PascalCase para nombres de tipos y nombres de enumeración.
- Nombre los archivos con camelCase (por ejemplo, ebsVolumes.tsx ostorage.tsb)

No utilizar la palabra clave var.

La sentencia let ese utiliza para declarar una variable local en TypeScript.

Es similar a la var palabra clave, pero tiene algunas restricciones de alcance en comparación con la var palabra clave.

Una variable declarada en un bloque con let solo se puede utilizar en ese bloque.

La var palabra clave no puede tener un ámbito de bloques, lo que significa que se puede acceder a ella desde fuera de un bloque concreto (representado por{}), pero no fuera de la función en la que está definida.