Utility Types

📌 `Partial<T>` → Hace todas las propiedades opcionales

✅ Es útil cuando **no queremos requerir todos los datos** al modificar un objeto.

```
interface Tarea {
  titulo: string;
  descripcion: string;
  completado: boolean;
}

// Ahora todas las propiedades de `Tarea` son opcionales
const tareaParcial: Partial<Tarea> = { titulo: "Comprar leche" };

console.log(tareaParcial); // { titulo: 'Comprar leche' }
```

📌 `Required<T>` → Hace todas las propiedades obligatorias

✅ Se usa cuando queremos **asegurarnos de que todas las propiedades estén presentes**.

```
interface Usuario {
  nombre?: string;
  email?: string;
}

// Convertimos todas las propiedades en obligatorias
const usuarioCompleto: Required<Usuario> = {
  nombre: "Juan",
  email: "juan@example.com",
};

console.log(usuarioCompleto);
```

❌ **Si falta una propiedad, TypeScript marcará error.**

📌 `Readonly<T>` → Evita modificaciones en un objeto

✅ Útil para proteger datos de cambios accidentales.

```
interface Configuracion {
  apiUrl: string;
  version: number;
}

// Ahora `config` es inmutable
const config: Readonly<Configuracion> = {
  apiUrl: "https://miapi.com",
  version: 1,
};

// config.apiUrl = "https://otraapi.com"; ❌ Error: No se puede modificar una propiedad readonly
console.log(config);
```

📌 `Pick<T, K>` → Extrae solo algunas propiedades

✅ Ideal para **crear versiones reducidas de objetos** sin todas sus propiedades.

```
interface Usuario {
  id: number;
  nombre: string;
  email: string;
  edad: number;
  rol: string; // Admin, User
}

// Solo queremos "nombre" y "email"
type UsuarioPublico = Pick<Usuario, "nombre" | "email">;

type ActualizarPerfilDTO = Pick<Usuario, "nombre" | "edad">;

const usuario: UsuarioPublico = { nombre: "María", email: "maria@example.com" };

const actualizarPerfil = (usuario: ActualizarPerfilDTO) => {
```

```
}

console.log(usuario);
```

📌 `Omit<T, K>` → Excluye propiedades

✅ Lo opuesto a `Pick`, elimina propiedades no deseadas.

```
interface Producto {
  id: number;
  nombre: string;
  precio: number;
  stock: number;
}

// Eliminamos "stock" del tipo
type ProductoSinStock = Omit<Producto, "stock">;

const producto: ProductoSinStock = { id: 1, nombre: "Laptop", precio: 1200
};

console.log(producto);
```

📌 `Record<K, T>` → Crea un objeto con claves de un tipo y valores de otro

✅ Se usa cuando queremos **estructuras homogéneas**.

```
type Traducciones = Record<string, string>;

const mensajes: Traducciones = {
  es: "Hola",
  en: "Hello",
  fr: "Bonjour",
};

console.log(mensajes["en"]); // "Hello"
```

📝 **Otro ejemplo con un ENUM como clave:**

```ts
enum Rol {
  Admin = "admin",
  Usuario = "usuario",
}

const permisos: Record<Rol, string[]> = {
  admin: ["crear", "editar", "eliminar"],
  usuario: ["leer"],
};

console.log(permisos);
```

📌 `Exclude<T, U>` → Elimina tipos de una unión

✅ Se usa para **filtrar tipos de una unión**.

```ts
type Estado = "activo" | "inactivo" | "pendiente" | "eliminado";

// Eliminamos "eliminado" de la lista de estados válidos
type EstadoValido = Exclude<Estado, "eliminado">;

let estado: EstadoValido = "activo";
// estado = "eliminado"; ❌ Error, "eliminado" ya no es válido

console.log(estado);
```

📌 `Extract<T, U>` → Extrae solo los tipos que coinciden

✅ Lo opuesto a `Exclude`, mantiene solo los tipos deseados.

```ts
type Estado = "activo" | "inactivo" | "pendiente" | "eliminado";

// Extraemos solo "activo" y "pendiente"
type EstadoAprobado = Extract<Estado, "activo" | "pendiente">;

let estado: EstadoAprobado = "activo"; // ✅ Correcto
// estado = "eliminado"; ❌ Error, "eliminado" no es parte de
`EstadoAprobado`

console.log(estado);
```

Resumen

Utility Type	Descripción
`Partial<T>`	Hace todas las propiedades opcionales.
`Required<T>`	Convierte todas las propiedades en obligatorias.
`Readonly<T>`	Hace que las propiedades no puedan modificarse.
`Pick<T, K>`	Extrae solo algunas propiedades de un tipo.
`Omit<T, K>`	Excluye propiedades de un tipo.
`Record<K, T>`	Crea un objeto con claves de `K` y valores de `T`.
`Exclude<T, U>`	Elimina ciertos tipos de una unión.
`Extract<T, U>`	Mantiene solo los tipos que coinciden.