# TypeScript Style Guide

## Core Principles

- Safety and Clarity: Prioritize code that is easy to read and minimizes runtime errors.
- Strictness: Enable `strict: true` in `tsconfig.json`.
- Modernity: Use the latest stable ECMAScript features.

## Naming Conventions

- Classes, Interfaces, Enums, Type aliases: `UpperCamelCase`.
- Variables, Functions, Methods, Properties: `lowerCamelCase`.
- Constants: `UPPER_SNAKE_CASE`.
- Filenames: `kebab-case.ts`.
- Avoid prefixes like `I` for interfaces (e.g., use `User` instead of `IUser`).

## Essential Types

- `any`: DO NOT USE. Use `unknown` or a specific type instead.
- `unknown`: Use when the type is truly unknown, requiring a type check before use.
- `never`: Use for functions that never return or for exhaustive switch checks.
- Arrays: Prefer `T[]` (e.g., `string[]`) over `Array<T>`.

## Interfaces vs Type Aliases

- Interfaces: Use for defining the shape of objects, especially when they will be implemented by classes or extended.
- Type Aliases: Use for unions, intersections, primitives, or complex combined types.
- General Rule: Prefer `interface` where possible for better performance and error messages in older TS versions.

## Classes and Constructors

- Use `private`, `protected`, and `public` keywords explicitly (or use `#` for true runtime privacy).
- Parameter Properties: Use them for simple initialization.

```
constructor(private readonly name:
string) {}
```

- No Logic in Constructors: Keep constructors simple; use initialization methods for complex setup.

## Functions

- Optional Parameters: Use `?` and place them at the end.
- Default Values: Prefer over optional parameters where a sensible default exists.
- Return Types: Always explicitly define the return type for public functions and methods.
- Arity: Functions should generally take no more than 3-4 arguments. Use an options object for more.

## Enums and Alternatives

- Enums: Use sparingly. Prefer `const enum` or Union of String Literals for better tree-shaking and runtime performance. `type Status = 'active' | 'inactive';`

## Modern Practices

- Nullish Coalescing: Use `??` instead of `||` for default values to avoid bugs with `0` or `false`.
- Optional Chaining: Use `?.` to safely access deep properties.
- Type Assertions: Avoid `as T` where possible. Prefer type guards or narrow the type with logic.

## Best Practices

### Avoid Non-null Assertions

Minimize use of `!`. Use `if` checks or optional chaining to handle potentially null values safely.

### Descriptive Types

Instead of `number`, use `type Timestamp = number` to provide domain context.

### Immutability

Use `readonly` for properties and `ReadonlyArray<T>` to prevent accidental mutations.