**What is a Configuration?**

A configuration is a set of rules, options, or settings that tells a system, tool, or application how to behave.

In programming, configurations help:

* Control how code is compiled or executed.
* Customize environment-specific behavior.
* Manage preferences without changing the actual code.

**In TypeScript: What is Configuration?**

In **TypeScript**, configuration is defined in the file tsconfig.json.

This tells the TypeScript compiler:

* **Which files to compile**
* **How strictly to check types**
* **Where to output compiled JS**
* **Which JavaScript version to use**, etc.

**✅ 1. "strict": true**

Enables **all strict type-checking options**:

* noImplicitAny
* strictNullChecks
* strictFunctionTypes
* strictBindCallApply
* alwaysStrict
* Others like strictPropertyInitialization, strictSaveCallChecks, etc.

**✅ Effect:**

If you enable strict: true, it turns on **all the options below automatically**.

**2. "noImplicitAny": true**

**Disallows implicit any types.** You must **explicitly annotate** types if they aren’t inferred.

**❌ Without it:2**

function greet(name) {

console.log("Hello " + name); // name is implicitly 'any'

}

**✅ With it:**

function greet(name: string) {

console.log("Hello " + name);

}

**3. "strictNullChecks": true**

**Disallows assigning null or undefined to any variable unless explicitly declared.**

**❌ Without it:**

let message: string = null; // Allowed

**✅ With it:**

let message: string = null; // ❌ Error

let safeMessage: string | null = null; // ✅ OK

Also prevents calling methods on values that might be null or undefined.

function printName(name: string | null) {

console.log(name.toUpperCase()); // ❌ Error

}

**4. "strictFunctionTypes": true**

Ensures that function types are only assignable if **parameters and return types are compatible**.

**❌ Without it:**

type A = (value: string) => void;

type B = (value: any) => void;

let a: A = (value) => console.log(value);

let b: B = a; // Allowed, even though 'any' is too loose

**✅ With it:**

let b: B = a; // ❌ Error: type mismatch in function parameter

**5. "strictBindCallApply": true**

**Type-checks arguments** passed to bind, call, and apply.

**❌ Without it:**

function sum(a: number, b: number) {

return a + b;

}

sum.call(null, "1", "2"); // No error

**✅ With it:**

sum.call(null, "1", "2"); // ❌ Error: Arguments must be numbers

sum.call(null, 1, 2); // ✅ OK

**6. "alwaysStrict": true**

Emits "use strict"; in every JavaScript output file and parses the code in **strict mode**.

**✅ Benefit:**

Strict mode:

* Prevents accidental global variables
* Throws errors for unsafe assignments

**Example Output:**

If you write:

let a = 5;

Compiled JS with "alwaysStrict": true:

"use strict";

var a = 5;

This helps ensure your JS runs in **strict mode**, which catches more bugs at runtime.