

Les 4 — Lesstof

TypeScript Fundamentals

Vak: Technical Foundations
Cursus: AI Developer
Volgende les: Les 5 — Next.js Basics

Vak: Technical Foundations **Vorige les:** Les 3 — Cursor Basics **Volgende les:** Les 5 — Next.js Basics

Inhoud

1. [Waarom TypeScript](#1-waarom-typescript)
2. [Setup — TS in een project](#2-setup--ts-in-een-project)
3. [Basic types](#3-basic-types)
4. [Object types + interfaces](#4-object-types--interfaces)
5. [Arrays + tuples](#5-arrays--tuples)
6. [Union + literal types](#6-union--literal-types)
7. [Functies typeren](#7-functies-typeren)
8. [Generics — herbruikbare types](#8-generics--herbruikbare-types)
9. [Type vs interface](#9-type-vs-interface)
10. [Type narrowing](#10-type-narrowing)
11. [Veelvoorkomende fouten](#11-veelvoorkomende-fouten)

1. Waarom TypeScript

JavaScript heeft geen type-checks. Code als `user . naem` (typo) draait, crasht pas in productie.

TypeScript = JavaScript + types. De compiler vangt veel fouten vóór runtime.

Voorbeelden van wat TS vangt

```
const user = { name: "Tim", age: 30 };
console.log(user.naem); // ■ Property 'naem' does not exist on type ...

function greet(name: string) { return `Hi ${name}`; }
greet(42); // ■ Argument of type 'number' is not assignable to parameter of type 'string'

const ids: number[] = [1, 2, 3];
ids.push("four"); // ■ Type 'string' is not assignable to type 'number'
```

Waarom in deze leerlijn

- Industrie-standaard sinds 2020
- Next.js + React + Supabase werken alle met TS
- AI-tools (Cursor, OpenCode) geven betere suggesties met types
- Zelf-documenterende code

Hobby projects mag JS. Productie = TS.

2. Setup — TS in een project

Nieuw project

```
pnpm create next-app # vraagt "TypeScript? Yes"
```

`tsconfig.json` wordt automatisch gemaakt.

Bestaand JS project

```
pnpm add -D typescript @types/node
npx tsc --init
```

Rename `.js` → `.ts` (of `.jsx` → `.tsx`).

Belangrijke tsconfig opties

```
{
  "compilerOptions": {
    "strict": true,           // Aanrader – vangt veel fouten
    "noImplicitAny": true,    // Geen any zonder expliciet maken
    "strictNullChecks": true, // null en undefined apart behandelen
    "target": "ES2022",
    "module": "ESNext",
    "moduleResolution": "Bundler"
  }
}
```

`strict: true` is **de** belangrijkste — zet 'm aan.

3. Basic types

Primitives

```
const name: string = "Tim";
const age: number = 30;
const isAdmin: boolean = true;
const nothing: null = null;
const notSet: undefined = undefined;
```

Inferentie

Vaak hoef je type niet te zetten:

```
const name = "Tim"; // TS weet: type is string
let count = 0;      // count is number
```

Regel: alleen type expliciet maken als TS het niet kan afleiden, of als signature documenteren.

`any` — vermijd

```
let data: any = ...; // ■ Alles mag, types check niet
```

`any` zet TS effectief uit voor die variabele. Gebruik alleen als laatste redmiddel.

Alternatief: `unknown` — moet je expliciet checken voor je hem gebruikt.

4. Object types + interfaces

Inline object type

```
const user: { name: string; age: number } = { name: "Tim", age: 30 };
```

Interface — herbruikbaar

```
interface User {
  name: string;
  age: number;
  email?: string; // optional
  readonly id: number; // alleen lezen
}

const tim: User = { id: 1, name: "Tim", age: 30 };
tim.id = 2; // ■ Cannot assign to 'id' because it is a read-only property
```

Nested objects

```
interface Post {
  title: string;
  author: {
    name: string;
    email: string;
  };
  tags: string[];
}
```

5. Arrays + tuples

Array

```
const numbers: number[] = [1, 2, 3];
const names: string[] = ["Tim", "Lisa"];

// Alternatieve syntax:
const items: Array<string> = ["a", "b"];
```

Tuple — vaste lengte, vaste types

```
const point: [number, number] = [10, 20];
const userRecord: [string, number, boolean] = ["Tim", 30, true];

// Useful voor React state:
const [count, setCount] = useState(0); // [number, (n: number) => void]
```

6. Union + literal types

Union — meerdere mogelijke types

```
let id: string | number = "abc-123";
id = 42; // ook OK

function format(value: string | number) {
  if (typeof value === "string") return value.toUpperCase();
  return value.toFixed(2);
}
```

Literal types — vaste waarden

```
type Status = "pending" | "active" | "archived";
let s: Status = "active";
s = "deleted"; // ■ Type '"deleted"' is not assignable to type 'Status'
```

Heel handig voor finite states.

Discriminated unions

```
type Result =
  | { ok: true; data: User }
  | { ok: false; error: string };

function handle(r: Result) {
  if (r.ok) {
    console.log(r.data.name); // TS weet: data bestaat hier
  } else {
    console.log(r.error);      // TS weet: error bestaat hier
  }
}
```

7. Functies typeren

Parameters + return type

```
function greet(name: string): string {
  return `Hi ${name}`;
}
```

Arrow function

```
const greet = (name: string): string => `Hi ${name}`;
```

Optional + default parameters

```
function log(msg: string, level: string = "info") { ... }
function find(id: number, includeDeleted?: boolean) { ... }
```

Function type as variable

```
type Handler = (event: string) => void;
const onClick: Handler = (e) => console.log(e);
```

Void vs undefined

- void = function returnt niets relevant (mag ook iets returnen, wordt genegeerd)
- undefined = explicit returnt undefined

Voor handlers: void. Voor functies die niets returnen: void of laat weg.

8. Generics — herbruikbare types

Generics = types met een "parameter" — flexibel maar nog steeds type-safe.

Voorbeeld

```
function first<T>(arr: T[]): T | undefined {
  return arr[0];
}

const num = first([1, 2, 3]); // type: number | undefined
const str = first(["a", "b"]); // type: string | undefined
```

is een type-parameter — TS infereert per call.

Use cases

Array helpers:

```
function map<T, U>(arr: T[], fn: (item: T) => U): U[] { ... }
```

API responses:

```
interface ApiResponse<T> {
  ok: boolean;
  data: T;
  error?: string;
}

const res: ApiResponse<User> = await fetch(...);
```

React components:

```
function List<T>({ items, render }: { items: T[]; render: (i: T) => ReactNode }) { ... }
```

In het begin verwarrend, na een week tweede natuur.

9. Type vs interface

Twee manieren om custom types te maken — wat is verschil?

```
type User = { name: string; age: number };
interface User { name: string; age: number; }
```

90% van de tijd: **uitwisselbaar**.

Verschillen

	Type	Interface
Unions	✓	✗
Intersection	A & B	extends
Declaration merging	✗	✓
Style	"modern"	"classic OOP"

Aanrader

- **Interface** voor objecten + classes (extends-pattern)
- **Type** voor unions, primitive aliases, complex types
- In team: kies één conventie, blijf consistent

10. Type narrowing

TS volgt je control-flow — types worden "smaller" binnen if-blocks.

typeof guard

```
function fmt(v: string | number) {
  if (typeof v === "string") {
    v.toUpperCase(); // OK
  } else {
    v.toFixed(2);    // OK – v is number hier
  }
}
```

in guard

```
type Cat = { meow: () => void };
type Dog = { bark: () => void };

function speak(animal: Cat | Dog) {
  if ("bark" in animal) animal.bark();
  else animal.meow();
}
```

Null check

```
const user: User | null = getUser();
if (user) {
  user.name; // OK – user is User hier
}
```

Type predicates

```
function isString(v: unknown): v is string {
  return typeof v === "string";
}

if (isString(input)) {
  input.toUpperCase(); // TS weet: input is string
}
```

11. Veelvoorkomende fouten

`any` overal


```
// ■  
function process(data: any) { ... }  
  
// ✓  
function process(data: unknown) { ... }  
function process(data: { id: number; name: string }) { ... }
```

Missing return type

```
// ■ Return type onduidelijk  
function fetchUser(id) { ... }  
  
// ✓  
function fetchUser(id: number): Promise<User> { ... }
```

Optional chaining vergeten

```
const name = user.profile.name; // ■ profile kan undefined zijn  
const name = user.profile?.name; // ✓
```

Non-null assertion misbruik

```
const el = document.getElementById("foo")!; // ■ "Ik beloof: niet null"  
// Als 't wel null is → runtime crash
```

Vermijd !. Liever: check expliciet.

`as any` om type-error te "fixen"

```
const data = response as any; // ■ Verbergt fout, vangt 'm niet
```

Liever: fix het type, of gebruik unknown + check.

Bronnen

- **TypeScript Handbook:** <https://www.typescriptlang.org/docs/handbook/intro.html>
- **TS Playground:** <https://www.typescriptlang.org/play>
- **Total TypeScript (Matt Pocock):** <https://www.totaltypescript.com>
- **Type Challenges:** <https://github.com/type-challenges/type-challenges>
- **React TypeScript Cheatsheet:** <https://react-typescript-cheatsheet.netlify.app>