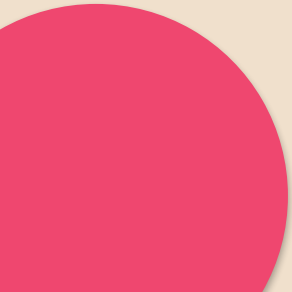




Les 4

TypeScript Fundamentals

De kracht van types - bugs vangen voor runtime



Terugblik

Les 3 - Cursor

Tab	Predictive completion
Inline Edit	Cmd+K, korte edits
Chat	Cmd+L, vragen + uitleg
Composer	Cmd+I, multi-file edits
@-mentions	Context kiezen voor AI
.cursorrules	Project-conventies

Vandaag: TypeScript.

AI-tools geven veel betere suggesties wanneer types in spel zijn.

Industrie-standaard. Productie = TypeScript.

Planning

Vandaag - 180 minuten

Welkom + terugblik 10 min

Het probleem + waarom TS 15 min

Basic types + interfaces 20 min

Unions, narrowing, guards 20 min

Pauze 15 min

Generics + type vs interface 15 min

Hands-on: TypeScript Escaperoom 70 min

Reflectie + huiswerk + afsluiting 15 min

Theorie 1

Waarom TypeScript?

JavaScript heeft geen type-checks. Typo's crashen pas in productie.

```
const user = { name: "Tim", age: 30 };  
console.log(user.naem);  
// JS: undefined - silent bug  
// TS: Property 'naem' does not exist on type {...}  
  
function greet(name: string) { return `Hi ${name}`; }  
greet(42); // TS: Argument of type 'number' not assignable
```

Voordelen:

- v Compiler vangt bugs voor runtime
- v Autocomplete + jump-to-def in editor
- v Zelf-documenterende code (types = docs)
- v AI-tools (Cursor) geven veel betere suggesties

Setup

TS in een project

Nieuw project (Next.js):

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

Bestaand JS project:

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

tsconfig.json - de belangrijke opties:

```
{  
  "compilerOptions": {  
    "strict": true,           // DE belangrijkste - zet 'm aan  
    "noImplicitAny": true,  
    "strictNullChecks": true  
  }  
}
```

Theorie 2

Basic types + interfaces

```
const name: string = "Tim";  
const age: number = 30;  
const isAdmin: boolean = true;  
  
// Inferentie:  
const x = "foo"; // string
```

```
interface User {  
  name: string;  
  age: number;  
  email?: string;  
  readonly id: number;  
}
```

Arrays + tuples:

```
const numbers: number[] = [1, 2, 3];  
const items: Array<string> = ["a", "b"];  
  
// Tuple - vaste lengte + types:  
const point: [number, number] = [10, 20];
```

Regel: alleen type expliciet als TS niet kan afleiden. Vermijd 'any'.

Theorie 3

Union + literal types

Union - meerdere mogelijke types met |:

```
let id: string | number = "abc-123";  
id = 42; // ook OK  
  
function format(value: string | number) { ... }
```

Literal types - vaste waarden:

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

Discriminated unions - veiligste pattern:

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

Theorie 4

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 - v is string hier
  } else {
    v.toFixed(2);    // OK - v is number hier
  }
}
```

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

```
// in guard:
type Cat = { meow: () => void };
type Dog = { bark: () => void };
if ("bark" in animal) animal.bark(); else animal.meow();
```




Pauze

15 minuten

Theorie 5

Functies typeren

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

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

Optional + default:

```
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 = returnt niets relevant. Voor handlers: void.

Theorie 6

Generics - herbruikbare types

Types met een parameter. Flexibel maar nog steeds type-safe.

```
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
```

Use case - API responses:

```
interface ApiResponse<T> {  
  ok: boolean;  
  data: T;  
  error?: string;  
}  
  
const res: ApiResponse<User> = await fetch(...).then(r => r.json());
```

In het begin verwarrend - na een week tweede natuur.

Theorie 7

Type vs interface

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

90% van de tijd: uitwisselbaar. Verschillen:

	Unions (A B)	type: ja	interface: nee
	Intersection	A & B	extends
	Declaration merging	nee	ja
	Style	modern	classic OOP

Aanrader:

Interface voor objecten + classes. Type voor unions, aliases, complex types.

Theorie 8

Veelvoorkomende fouten



any overall

Gebruik 'unknown' + check, of typ expliciet



Missing return type

Documenteer signature voor caller



Optional chaining vergeten

Gebruik ?. bij mogelijk-undefined



Non-null assertion (!)

Vermijd - check expliciet



as any om error te 'fixen'

Verbergt fout, fix het type

LESOPDRACHT

TypeScript Escaperoom (~70 min)

10 kamers, progressieve moeilijkheid:

Kamer 1-3

Basics: primitives, arrays, interfaces

Kamer 4-6

Unions, narrowing, type guards

Kamer 7-9

Generics, intersection, readonly

Kamer 10

Boss-level: alles combineren

Aanpak:

- Unzip les4-typescript-escaperoom.zip
- Open in Cursor - laat type-errors je gids zijn
- Per kamer: lees error -> fix -> verder
- AI is OK, maar probeer eerst zelf (leren door fouten)

Huiswerk

Voor Les 5 (Next.js)

1. JS -> TS converter klusje

Zie zip: les4-huiswerk-js-converter. Maak 'm typed.

2. Escaperoom afmaken

Niet alle kamers gehaald in de les? Thuis afmaken.

3. Reflectie 200 woorden

Welke type-feature vond je verrassend? Wat fout deed je vaak?

4. Inleveren via Teams

Voor volgende les

Tip: zet strict: true aan en lees elke error rustig. TS leert je veel.

Volgende les

Les 5 - Next.js Basics

TypeScript in actie binnen een framework:

- Next.js App Router (file-based routing)
- Server Components vs Client Components
- Data fetching patterns
- Hands-on: QuickPoll Part 1 - klassikaal bouwen

Vorbereiden:

Cursor open, pnpm geïnstalleerd, GitHub + Vercel accounts werkend.

Vragen?

Tot volgende week!