

Les 4 — Lesopdracht

TypeScript Escaperoom

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

Welkom in de TypeScript Escaperoom! Je bent opgesloten in 8 kamers vol met type-puzzels. Je enige kans op vrijheid? Het juiste type op de juiste plaats zetten. Ga je het redden?

Wat ga je doen?

Je gaat een interactieve "escaperoom" doorlopen in TypeScript. Er zijn **8 kamers**, elk met TypeScript puzzels die steeds moeilijker worden. Je taak: **fix alle type errors zodat de code compileert zonder fouten**.

Als je alle kamers hebt opgelost, voert je project automatisch uit en ontvang je de **escape code** — het bewijs dat je echt bent ontsnapt!

Dit is geen standaard oefening — dit is **gamification** zoals het hoort te zijn. Je gaat TypeScript types niet zomaar "leren", je gaat ze **meesteren** door ze in actie toe te passen.

Waarom dit format?

- **Gamified:** Elke kamer is een mini-challenge met increasing difficulty
- **Doel-gericht:** Je weet EXACT wanneer je klaar bent (de escape code!)
- **Hands-on:** Je leert door té DOEN, niet door té LEZEN
- **Cursor-friendly:** Perfect voor het gebruik van Cursor's AI-features

Setup (5 minuten)

Volg deze stappen voordat je begint:

Stap 1: Download het project

- Ga naar de **Teams channel** voor Les 4
- Download het bestand: `les4-typescript-escaperoom.zip`
- Pak het uit in een nieuwe folder

Stap 2: Open in Cursor

- Open de folder in **Cursor** (of VS Code)
- Zorg dat je het root-level bestand ziet: `package.json`

Stap 3: Installeer dependencies

```
npm install
```

Dit installeert TypeScript en alle tools die je nodig hebt.

Stap 4: Maak je ready

```
npm run check
```

Je ziet nu een overzicht van alle kamers en welke errors nog opgelost moeten worden. Dit commando voer je regelmatig uit om je voortgang te tracken.

Stap 5: Begin!

Lees de rest van dit document en start bij **Kamer 1**.

De 8 Kamers

Elke kamer bevat een TypeScript-bestand met fouten en taken. De difficulty loopt op van ■ (makkelijk) naar ■■■■ (moeilijk).

■ Kamer 1: Basic Types

Moeilijkheid: ■ Makkelijk **Bestand:** `src/kamer1-basics.ts`

Doel

Fix de **type annotations** op variabelen. TypeScript geeft je rode squiggles — jouw taak is die weg te krijgen door juiste types toe te voegen.

Wat je gaat leren

- Basis TypeScript types: `string`, `number`, `boolean`
- Arrays typen: `string[]` en `number[]`
- Type annotations schrijven: `let x: string = "hello"`

De Challenge

Het bestand bevat variabelen zonder types OF met foute types. Bijvoorbeeld:

```
let name = "Alice";           // ■ Mis je type hier?
let age: string = 25;         // ■ Verkeerde type!
let hobbies = ["chess"];      // ■ Welk array-type?
```

Jouw taak: Maak alle types **expliciet en correct**.

Hints

- Hover over rode errors in Cursor — de error message vertelt je precies wat fout is
- Probeer eerst jezelf — dit is de makkelijkste kamer, je lukt het!
- `let x: Type = value` is de syntax

■ Kamer 2: Type Inference

Moeilijkheid: ■ Makkelijk **Bestand:** `src/kamer2-inference.ts`

Doel

Dit is een "**omgekeerde puzzle**". In plaats van types ÁF te schrijven, verwijder je onnodige types. TypeScript kan namelijk veel types **zelf afleiden** (inferren).

Wat je gaat leren

- Wanneer TypeScript types automatisch kan inferren
- Dat expliciete types niet altijd nodig zijn
- Schone code** = minimale redundantie

De Challenge

Het bestand zit vol met **overexpliciet getypte** code:

```
let count: number = 42;           // ■ Overbodig, TypeScript ziet het al
const message: string = "Hello";  // ■ Overbodig
let items: boolean[] = [true, false]; // ■ Overbodig
```

Maar sommige types **MOETEN** blijven:

```
function calculate(x: number, y: number): number { } // ■ Nodig!
```

Jouw taak: Verwijder onnodige type annotations, houd de essentiële.

Hints

- Rule of thumb: functie parameters en return types altijd typen, lokale variabelen zelden
- `npm run check` zal je zeggen wat je mist
- Dit is nog steeds makkelijk — je bent warm up!

■ Kamer 3: Interfaces

Moeilijkheid: ■■ Medium **Bestand:** `src/kamer3-interfaces.ts`

Doel

Nu ga je zelf **interfaces schrijven** voor objecten. Interfaces beschrijven de "vorm" van een object — welke properties het moet hebben en welke types die hebben.

Wat je gaat leren

- Interfaces definiëren: `interface User { ... }`
- Eigenschappen annoteren: `name: string`
- Nested properties en objecten
- Objects aan interfaces binden

De Challenge

Je krijgt objecten zonder types. Jij schrijft de interface:

```
// Gegeven:
const user = {
  name: "Alice",
  age: 30,
  email: "alice@example.com"
};

// Jouw taak: Schrijf interface User { ... }
// En maak: const user: User = { ... }
```

Sommige objecten zijn **genest** (objects in objects):

```
const person = {
  name: "Bob",
  address: {
    street: "Main St",
    city: "Amsterdam"
  }
};
// Dit vereist nested interface definitions!
```

Hints

- ■ Interfaces beginnen met een HOOFDLETTER: `interface User`, niet `interface user`
- ■ Voor nested objects maak je nested interfaces
- ■ Syntax: `interface Name { prop: Type; otherProp: Type; }`

■ Kamer 4: Optional Properties

Moeilijkheid: ■■ Medium Bestand: `src/kamer4-optionals.ts`

Doel

Niet alle properties zijn altijd aanwezig. Je leert **optionele properties** (?) te gebruiken — als iets "misschien" aanwezig is.

Wat je gaat leren

- Optionele properties: `property?: Type`
- Het verschil tussen `undefined` en "niet aanwezig"
- Praktische use cases (formulieren, API responses)

De Challenge

Je interfaces hebben properties die soms aanwezig zijn, soms niet:

```
interface User {
  name: string;           // ■ ALTIJD nodig
  nickname: ?;            // ■ SOMS nodig – use ?
  phone?: string;         // ■ Dit is correct!
  address?: {
    street: string;
    city: string;
  };
}
```

Zeg voor elke property: "is dit ALWAYS nodig, of SOMETIMES?" Met ? zeg je "sometimes".

Hints

- ■ `property?: Type` = "optioneel"
- ■ Zorg dat je objects niet meer/minder properties hebben dan je interface zegt
- ■ Optionele properties kunnen `undefined` zijn

■ Kamer 5: Union Types

Moeilijkheid: ■■ Medium **Bestand:** `src/kamer5-unions.ts`

Doel

Soms kan een variable **meerdere types** hebben. Dat is wat "union types" zijn: `Type1 | Type2`. Je leert ze gebruiken voor flexibele, veilige code.

Wat je gaat leren

- Union types: `string | number`
- Literal types: `"success" | "error"`
- Basis type narrowing (checken welk type het werkelijk is)

De Challenge

Je krijgt code met dingen die meerdere types kunnen zijn:

```
// Een ID kan string OF number zijn
let userId: ?; // = string | number

// Een status is altijd één van deze waarden
let status: ?; // = "pending" | "complete" | "failed"

// Een waarde kan string, number, OF boolean zijn
function format(value: ?): string {
  // Jouw taak: zeg wat de mogelijke types zijn
}
```

Je moet ook **type narrowing** gebruiken — checken wat het type werkelijk is:

```
if (typeof userId === "string") {
  // In dit blok WEET TypeScript dat userId een string is
}
```

Hints

- ■ Syntax: Type1 | Type2 | Type3 (met pipes)
- ■ Literal types zijn strings/numbers in quotes: "success", not success
- ■ Use typeof checks om te zien wat je hebt

■ Kamer 6: Type Aliases

Moeilijkheid: ■■ Medium **Bestand:** src/kamer6-aliases.ts

Doel

Wanneer je dezelfde type-definitie veel gebruikt, maken type aliases je code **schoner en herbruikbaar**. Dit is als een "naam geven" aan je types.

Wat je gaat leren

- Type aliases: type Name = Type
- Wanneer aliassen beter zijn dan inline types
- Type aliases voor complexe types

De Challenge

Je code herhaalt dezelfde type-definitie overal:

```
// ■ Overall schreef je dit:
function getUser(id: string | number): { name: string; age: number } { }
function updateUser(id: string | number, data: { name: string; age: number }): void { }
function deleteUser(id: string | number): { name: string; age: number } { }

// ■ Beter: maak type aliases!
type UserId = string | number;
type UserData = { name: string; age: number };
// En hergebruik ze:
function getUser(id: UserId): UserData { }
function updateUser(id: UserId, data: UserData): void { }
function deleteUser(id: UserId): UserData { }
```

Jouw taak: Identificeer repetitieve type-definitions en maak aliassen.

Hints

- ■ Syntax: `type NameOfType = ActualType;`
- ■ Aliassen beginnen meestal met een HOOFDLETTER
- ■ Dit voorkomt "copy-paste type-fouten"

■ Kamer 7: Functies Typen

Moeilijkheid: ■■■ Moeilijk **Bestand:** `src/kamer7-functies.ts`

Doel

Dit is waar het serieus wordt. Je gaat **functie parameters en return types** volledig typen. Dit is essentieel voor grote codebases.

Wat je gaat leren

- Function parameter types: `function foo(x: Type) { }`
- Return types: `function foo(): ReturnType { }`
- Callback types: functies die functies ontvangen
- Arrow function syntax typen

De Challenge

Je krijgt niet-getypte functies:

```
// ■ Geen types!
function add(a, b) {
  return a + b;
}

function filterNumbers(items, condition) {
  return items.filter(condition);
}

const greet = (name) => {
  return `Hello, ${name}!`;
};

// ■ Jouw taak: Maak ze VOLLEDIG getypt!
function add(a: number, b: number): number {
  return a + b;
}

function filterNumbers(items: number[], condition: (n: number) => boolean): number[] {
  return items.filter(condition);
}

const greet = (name: string): string => {
  return `Hello, ${name}!`;
};
```

Sommige functies hebben **callbacks** (functies als parameters):

```
function runTwice(fn: ?): void {
  fn();
  fn();
}

// Wat is het type van `fn`? Een functie die geen parameters neemt!
```

Hints

- Altijd annotate: parameters EN return type
- Arrow functions: (param: Type): ReturnType => { }
- Callback types: (param: Type) => ReturnType
- Geen parameters? () => ReturnType

■ Kamer 8: Boss Room ■

Moeilijkheid: ■■■ Moeilijk **Bestand:** src/kamer8-boss.ts

Doel

Dit is het **finale level**. Je combineert ALLES wat je hebt geleerd in één groot project. Dit is een echte, praktische TypeScript-scenario.

Wat je gaat leren

-
- Integratie van alle vorige concepts
 - Realistische type-design
 - Denken in termen van "veilige types"

De Challenge

Je krijgt een onvolledig systeem, bijv. een "Blogplatform" of "Takenmanager". Je taak:

1. **Schrijf alle interfaces** voor de data structures
2. **Type alle functies** volledig
3. **Gebruik union types, aliases, optionals** waar nodig
4. **Zorg dat alles samenwerkt**

Voorbeeld:

```
// ■ Gegeven (ongetypt):
class BlogManager {
    posts = [];

    addPost(title, content, author) {
        // ...
    }

    getPostsByAuthor(author) {
        // ...
    }

    deletePost(postId) {
        // ...
    }
}

// ■ Jouw taak: Maak het volledig getypt
interface Post {
    id: string;
    title: string;
    content: string;
    author: string;
    createdAt: Date;
}

type PostId = string;

class BlogManager {
    posts: Post[] = [];

    addPost(title: string, content: string, author: string): Post {
        // ...
    }

    getPostsByAuthor(author: string): Post[] {
        // ...
    }

    deletePost(postId: PostId): boolean {
        // ...
    }
}
```

Dit is realistisch werk. Je ziet hoe types een compleet systeem kunnen gidsen.

Hints

- ■ Begin met de interfaces voor je data
- ■ Type dan je functies
- ■ Controleer constant met `npm run check`
- ■ Dit is moeilijk, maar jij gaat het redden!

Hoe Track Je Je Voortgang?

Command 1: Check je status

```
npm run check
```

Dit toont je ALLE kamers en hun status:

```
■ Kamer 1: Basic Types
■ Kamer 2: Type Inference
■ Kamer 3: Interfaces      [3 errors]
■ Kamer 4: Optional Props  [1 error]
■ Kamer 5+: Nog niet gestart
```

Run dit regelmatig — het is je "scorebord"!

Command 2: Ontsnapt!

```
npm run escape
```

Runnen dit als ALLES ■ is. Je project compileert en geeft je de **escape code**:

```
■ ALLE KAMERS ONTSNAPT! ■
Je escape code is: TS2024-TYPES-MASTER-42
Gefeliciteerd! Je bent een TypeScript type-master!
```

Tips & Tricks

1■■ Gebruik Cursor's AI (gratis!)

- **Ctrl+K** (of **Cmd+K** op Mac): Open Cursor's code editor
- **Ctrl+L**: Highlight code en vraag om hulp
- **Ctrl+I**: Inline code generation

Voorbeeld:

```
"Ik heb een interface User met name, email, en optioneel phone.
Schrijf een functie getUser(id: string): User"
```

2■■ Hover voor Error Messages

Wanneer je rode squiggles ziet, **hover** erover. TypeScript zegt je exact wat verkeerd is.

3■■■ Bekijk Type Definitie

Ctrl+Click (of Cmd+Click) op een type om te zien hoe het gedefinieerd is.

4■■■ Probeer EERST Zelf

- Lees de comment-hints in het bestand
- Denk na: "Wat is het juiste type?"
- Pas het aan
- Check met `npm run check`

Pas dan naar Cursor Chat als je stuck bent.

5■■■ Lees de Comments!

Elk bestand bevat **hints in comments**:

```
// TODO: Type deze parameter als string | number
function process(value) { }

// TIP: Denk: moet dit ALTIJD aanwezig zijn?
interface Config {
  debug: boolean;
}
```

Wat Als Je Vastzit?

Stap 1: Lees de Error

```
error TS2322: Type 'string' is not assignable to type 'number'
```

Dit zegt: je probeert een string in een number-veld te stoppen. Fout type!

Stap 2: Check de Hints

- Lees comments in je bestand
- Hover over errors
- Kijk naar omringende code

Stap 3: Gebruik Cursor Chat

- Open **Cursor Chat** (Cmd+L)
- Vraag: "Wat is het juiste type voor deze functie?"
- Cursor helpt je (geen spoilers, maar guidance)

Stap 4: Documentatie

- TypeScript Handbook (<https://www.typescriptlang.org/docs/>) (Engels)
- Kijk op Canvas voor Nederlands uitleg

Bonus Challenges (Extra Sterren! ■)

Wanneer je alle 8 kamers hebt opgelost (en de escape code hebt!), kun je nog meer doen:

Bonus Kamer 9: Custom Domain

Verzin een eigen domein dat je interessant vindt (bijv. Recipes, Music Playlist, Game Inventory) en schrijf:

- 3-5 interfaces
- 2-3 getypte functies
- Minstens 1 union type of type alias

Bonus Kamer 10: Generics Preview

Generics zijn "types voor types". Dit is Les 5, maar je kunt vooruitlopen:

```
// Generic functie: werkt met elk array-type
function getFirst<T>(items: T[]): T {
  return items[0];
}

const firstString = getFirst(["a", "b"]); // string
const firstNumber = getFirst([1, 2, 3]); // number
```

Schrijf 2-3 generische functies en test ze.

Deliverables

Wanneer je klaar bent:

1. Screenshot van Escape Code

```
npm run escape
```

Maak een screenshot van je output (inclusief de escape code!).

2. Post in Teams

- Ga naar het **Les 4 channel** op Teams
- Post je screenshot
- Typ je escape code in de chat
- Voeg een kort comment toe: wat was het moeilijkste? (1-2 zinnen)

3. Bonus: Show & Tell

Heb je de bonus challenges gedaan? Post die ook! (screenshot of GitHub link)

Wat Je Hebt Geleerd

Na deze escaperoom begrijp je:

■ **Basic Types**: string, number, boolean, arrays ■ **Type Inference**: wanneer TypeScript types zelf kan afleiden ■ **Interfaces**: objecten typen ■ **Optionals**: properties die misschien aanwezig zijn ■ **Union Types**: "dit ÓF dat" ■ **Type Aliases**: types hergebruiken ■ **Function Types**: parameters en returns volledig typen ■ **Integration**: alles samen in real code

Dit zijn de **essentiële fundamenten** van TypeScript. Hiermee kun je veilige, schone code schrijven die zichzelf documenteert.

FAQ

V: Mag ik ChatGPT gebruiken? A: Ja, maar probeer EERST zelf. ChatGPT voert je door, Cursor Chat leert je.

V: Wat als ik een kamer echt niet snap? A: (1) Kijk de video-uitleg voor Les 4 opnieuw, (2) Vraag de docent, (3) Werk met een classmate samen.

V: Hoe lang duurt dit? A: 2-3 uur voor de basis kamers. Bonus challenges: +1 uur.

V: Is dit onderdeel van mijn cijfer? A: Ja! De escape code bewijst dat je het geleerd hebt. Post hem voor volle punten.

V: Kan ik offline werken? A: Ja! Download, install npm `install`, en werk offline. Controleer later je escape code.

Je Bent Klaar voor Les 5! ■

Als je alle kamers hebt ontsnapt, ben je klaar voor:

- **Generics** (Les 5)
- **Classes & Advanced Types** (Les 6)
- **Real TypeScript Projects** (Les 7+)

Veel succes, en geniet van de escaperoom! ■