

Les 4 — Huiswerk

TypeScript oefenen

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

In deze les heb je geleerd hoe TypeScript je code veiliger maakt met types. Nu ga je dat in de praktijk brengen! Je krijgt een JavaScript project met utility functies voor een e-commerce systeem. Jouw taak: **alles omzetten naar TypeScript zonder any types te gebruiken.**

Dit huiswerk bereidt je voor op Les 5, waar je TypeScript met React gaat combineren.

Setup

1. **Download** `les4-huiswerk-js-converter.zip` van Teams
2. **Pak uit** en open de folder in Cursor
3. **Installeer dependencies:** `npm install`
4. **Bekijk** de JavaScript bestanden in `src/`
5. **Begin omzetten!**

De folder ziet er zo uit:

```
les4-huiswerk-js-converter/  
├── src/  
│   ├── users.js  
│   ├── products.js  
│   ├── orders.js  
│   └── utils.js  
├── __tests__/  
│   ├── users.test.ts  
│   ├── products.test.ts  
│   ├── orders.test.ts  
│   └── utils.test.ts  
├── tsconfig.json  
├── package.json  
└── README.md
```

De Bestanden

Bestand 1: `src/users.ts`

Je moet `users.js` omzetten naar `users.ts`.

Functies die je zult tegenkomen:

- `createUser(name, email, age)` — maakt een nieuwe user aan
- `findUserByEmail(users, email)` — zoekt een user op e-mailadres
- `updateUser(user, updates)` — update user gegevens
- `filterActiveUsers(users)` — geeft alle actieve users terug
- `deactivateUser(user)` — zet een user op inactief

Wat je moet doen:

1. Maak een `User` interface met: `id`, `name`, `email`, `age`, `isActive`, `createdAt`
2. Zorg dat `createUser` het juiste type teruggeeft
3. `findUserByEmail` kan `User | null` teruggeven
4. Type alle parameters en return types volledig
5. Geen any!

Hint: Kijk in de tests hoe `User` wordt gebruikt — dat zegt je veel over de verwachte types.

Bestand 2: `src/products.ts`

Je moet `products.js` omzetten naar `products.ts`.

Functies die je zult tegenkomen:

- `createProduct(name, price, category, description)` — maakt een nieuw product aan
- `applyDiscount(product, percentage)` — past korting toe
- `getExpensiveProducts(products, minPrice)` — filtert dure producten
- `formatPrice(price, currency)` — formatteert prijs als string met valutasymbool
- `rateProduct(product, rating)` — geeft een product een beoordeling (1-5)

Er is ook een `CATEGORIES` constante: een array met geldige categorieën.

Wat je moet doen:

1. Maak een `Product` interface met: `id`, `name`, `price`, `category`, `description` (string of null), `inStock`, `rating` (number of null)
2. Maak een **union type** voor `category`: `"electronics" | "clothing" | "food" | "books" | "sports"`
3. Type alle functies compleet
4. De `applyDiscount` functie geeft een nieuw object terug (geen mutation)
5. Geen any!

Let op: `description` en `rating` kunnen null zijn. Gebruik `string | null` en `number | null`.

Bestand 3: `src/orders.ts`

Je moet `orders.js` omzetten naar `orders.ts`.

Functies die je zult tegenkomen:

- `createOrder(user, products)` — maakt een order aan
- `calculateTotal(products)` — berekent totale prijs van een productenlijst
- `updateOrderStatus(order, newStatus)` — wijzigt de status (met validatie van geldige transities)
- `getOrdersByUser(orders, userId)` — filtert orders per user
- `getOrdersByStatus(orders, status)` — filtert orders op status

Wat je moet doen:

1. Maak een `Order` interface met: `id`, `userId`, `products` (array van `{ productId: string; name: string; price: number }`), `total`, `status`, `createdAt`
2. Maak een **union type** voor `OrderStatus`: `"pending" | "processing" | "shipped" | "delivered" | "cancelled"`
3. Type alle functies compleet
4. Geen any!

Let op: De `products` in een `Order` zijn NIET dezelfde als `Product[]`. Het zijn vereenvoudigde objecten met alleen `productId`, `name` en `price`.

Bestand 4: `src/utils.ts`

Je moet `utils.js` omzetten naar `utils.ts`. Dit bestand heeft wat moeilijkere type challenges.

Functies die je zult tegenkomen:

- `formatDate(date)` — formatteert een `Date` naar `"DD-MM-YYYY"`
- `generateId()` — genereert een willekeurige string ID
- `validateEmail(email)` — checkt of email geldig is (return boolean)
- `sortBy(items, key, direction)` — sorteert een array op een eigenschap, optioneel ascending/descending
- `groupBy(items, key)` — groepeert items per key-waarde in een object

Wat je moet doen:

1. `formatDate` ontvangt een `Date` en geeft een string terug
 2. `generateId` geeft altijd een string terug
 3. `validateEmail` geeft een boolean terug
 4. `sortBy` is het lastigste! Dit is waar **generics** om de hoek komen kijken
- `sortBy` moet werken met `*elk*` soort array
 - De functie signature zou iets als `function sortBy(items: T[], key: keyof T, direction?: string): T[]` moeten zijn

1. `groupBy` gebruikt ook generics: `function groupBy(items: T[], key: keyof T): Record`

Geen any!

Vereisten

Zorg dat je aan alle punten voldoet:

- ■ **Geen any types** — serieus, zelfs niet voor "snel omzetten"
- ■ **Geen `// @ts-ignore` comments** — verstop de problemen niet
- ■ **Alle functies volledig getypt** — parameters EN return types
- ■ **Interfaces voor alle objecten** — User, Product, Order
- ■ **Union types waar logisch** — `ProductCategory` en `OrderStatus`
- ■ **Nullable types waar nodig** — `string | null` en `number | null`
- ■ `npm run check` **moet 0 errors geven** — TypeScript compiler akkoord
- ■ `npm test` **moet groen zijn** — alle 26 tests slagen

Verificatie

Als je klaar bent, run je deze commands:

```
# Controleer TypeScript errors (moet 0 tonen)
npm run check

# Run de tests (alle tests moeten slagen)
npm test
```

De tests zijn al geschreven in TypeScript! Ze checken of jouw types correct zijn. Als tests mislukken, kijk dan naar de foutmelding — die zal je helpen.

Tips & Tricks

1. **Begin met interfaces** Definieer eerst de shapes van je data. Dan schrijf je veel makkelijker de functies.

```
interface User {
  id: string;
  name: string;
  email: string;
  age: number;
  isActive: boolean;
  createdAt: Date;
}
```

2. Kijk naar hoe functies GEBRUIKT worden Open de test files in `__tests__`/ — die laten zien hoe je types moeten werken. Tests zijn je "spec"!

3. Gebruik Cursor voor hints Cursor geeft je hover-tips. Hover over een functie en zie wat het verwacht. Maar probeer eerst zelf!

4. Union types zijn duidelijker dan strings Dit:

```
type OrderStatus = "pending" | "processing" | "shipped" | "delivered" | "cancelled";
```

Is veel beter dan:

```
status: string;
```

Waarom? TypeScript geeft je autocomplete en warnt je als je een verkeerde status invult.

5. Nullable types met `| null` Sommige velden kunnen null zijn:

```
interface Product {  
  id: string;  
  name: string;  
  price: number;  
  description: string | null;  
  rating: number | null;  
}
```

6. Ga stap voor stap

- Zet eerst alles om naar `.ts` files
- Type dan één bestand af
- Zorg dat `npm run check` nul errors toont
- Ga dan naar volgende bestand

Inleveren

1. **Push naar GitHub** — je repository waar je al aan werkt
2. **Branch:** `feature/les4-typescript-converter`
3. **Commit message:** `feat: convert JavaScript utilities to TypeScript`
4. **Deadline:** Voor het begin van Les 5

Verwachte tijd

Dit huiswerk duurt ongeveer **1,5 tot 2 uur** afhankelijk van je snelheid. Niet rennen — begrijpen is belangrijker!

Vragen?

- Stuck op een error? Lees het TypeScript error message goed
- Weet je de syntax van een type niet? Kijk in de lesson slides Les 4
- Denk je dat de tests fout zijn? Neem contact op!

Veel succes!