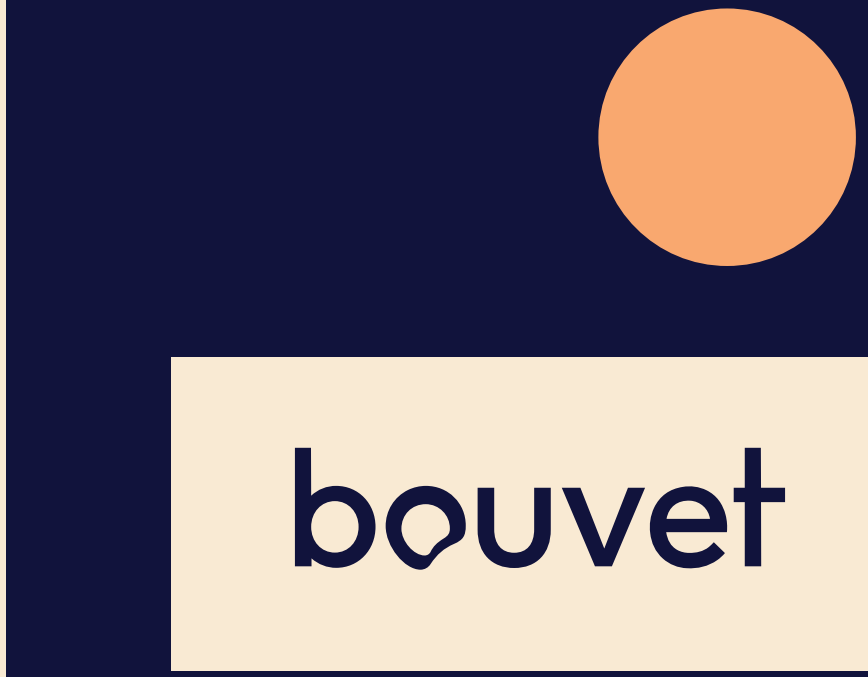
TYPESCRIPT



NTNU - 26 september 2022

Christian, Oscar, Vigdis, Ebba og Julie Adele

Hva er TypeScript?

«JavaScript with syntax for types»

- Et sterkt typet programmeringsspråk som bygger på JavaScript
- Et super-set av JavaScript
- Utvikles av Microsoft
- TypeScript kan kompileres til JavaScript
- Det er JavaScript som kjører run-time, ikke TypeScript

bouvet

Fordeler og ulemper

Fordeler:

+ Fange enkle bugs allerede under utvikling
+ Bedre kodekvalitet
+ Intellisense/Autocomplete
+ Gjør det enklere å forstå hva koden er ment å gjøre

Ulemper:

- Bratt læringskurve
- Tar litt lengre tid å skrive
- Kan bli vanskelig å lese hvis det er kompleks typing

Når er det lurt å bruke TypeScript?

- Store prosjekter
- Virksomhetskritiske systemer
- Et prosjekt med fullstack-utviklere som allerede er vant til typesikre språk (f.eks. C# eller Java)
- Kvalitet og stabilitet er viktigere enn raske endringer og fleksibilitet
  - Virksomhetskritiske systemer vs. En POC eller hackathon 😄

Komme i gang
(ikke i dag)

- `npm i typescript -D` eller `npx tsc`
  - `tsc` er CLI-kommandoen for å kjøre TypeScript-compileren
- Du kan velge om du vil kjøre transpilering via TypeScript-compileren eller via en plugin i babel, MEN:
  - Babel gir deg ikke feilmeldinger på type errors - "Oversetter" bare TS til vanlig JS.
  - Du må bruke egne plugins for features som bare er proposals i ES, men standard i TS (dette var f.eks. et problem med optional chaining `?` og nullish coalescing `??`)
- Vite.js (create-vite) har flere templates for å bootstrappe TS prosjekter:
  - `npm create vite@latest my-ts-app --template vanilla-ts`

tsconfig

Konfigurer settings for TypeScript

- Kan enten være **`tsconfig.json`** eller **`jsconfig.json`**
- Ligger typisk i rot-mappen (er der TS default leter etter den)

```
{
  "compilerOptions": {},
  "files" : ["custom.d.ts"]
  "include": ["src/**/*", "tests/**/*"],
  "exclude": ["*ignore.ts"],
  "extends": "../some-other-project-/ts-config",
  "references": [
       { ”short-path-name": "../deeply/nested/file/you/want/short/imports/for" }
  ]
}
```

bouvet

tsconfig

compilerOptions (noen av de)

- `outDir` – Hvor outputfilene (JS) skal emittes
- `noImplicitAny`
- `skipLibCheck`
- `noEmit` – Hvorvidt TS skal emitte JS filer eller ikke
- `jsx` - Innebygd støtte for å håndtere jsx
- `module` – Hvilken module resolution skal brukes (for import/export)
- `target` – Hvilken ES versjon skal TS kompileres til?
- `allowJs`
- `noImplicitReturns - Må returnere noe`
- `esModuleInterop` – Gjør det mulig å importere CommonJS moduler som ES moduler

Typer

Primitive typer

- undefined
- null
- number
- bigint
- boolean
- string

Hva med?

- Number
- Bigint
- Boolean
- String

bouvet

Typer

Litt om variabler

- Du *kan* deklarere en variabel på samme måte som i JavaScript:
  - `var`
  - `const`
  - `let`

```
let n = 0;
n++;

const n2 = n;
```

Typer

Eksplisitt typing med primitive typer

```
let n = 0; // Inferres som typen number
n = 1;
n = "text"; // Type 'string' is not assignable to type 'number'.ts(2322)

let n2: number;
n2 = 2;
n2 = "text"; // Type 'string' is not assignable to type 'number'.ts(2322)

let n3: number = 0;
n3 = 3;
n3 = "text"; // Type 'string' is not assignable to type 'number'.ts(2322)

const n4 = 0; // Inferres som typen 0
const n5 = 0 as number;
const n6: number = 0;
```

Operatorer

Innebygget i TS

- `as` (cast operator)
- `&` (intersection operator)
- `|` (union operator)
- `keyof` (lager en type basert på nøklene i en type)
- `typeof` (gir deg nøyakitg type for en verdi – en kraftfullere variant av typeof i JS)
- `-?` (remove optional)
- `+?` (add optional)
- `interface` (typer som beskriver objekter)
- `type` (beskrive typer, bade primitive og ikke primitive)
- `public`
- `private`
- `implements`
- `readonly`
- `namespace` (for å organisere og gruppere kode)

Typer

Funksjoner

```
const hi = (name: string) => console.log(`Hello ${name}!`);
hi("0");
hi(0); // Argument of type 'number' is not assignable to parameter of type 'string'.ts(2345)

const hi2 = (name: string): void => console.log(`Hello ${name}!`);

function hi3(name: string): void {
 console.log(`Hello ${name}!`);
}

const hi4 = function (name: string) {
 console.log(`Hello ${name}!`);
};

// Typen til alle disse funksjonene blir (name: string) => void
```

Typer

Funksjoner

```
// Du må som regel sette type til parameterne til en funksjon
const hi = (name: string) => console.log(`Hello ${name}!`);

// Med unntak av annonyme funksjoner som blir sendt som parameter
// der TS kjenner til typen som den parameteren (funksjonen) skal ha
[“Oscar”, ”Christian"].map((name) => ({ name }));

document.querySelectorAll("p").forEach((element) => {
    element.classList.add("text");
});
```

bouvet

Typer
Funksjoner

- Argumenter kan være optional

```
const hi = (firstName: string, lastName?: string) =>
  `Hello ${`${firstName} ${lastName ?? ''}.trim()`}!
```

- Argumentene kan også ha default verdier (som i JS):

```
const hi = (firstName: string, lastName = "") =>
  `Hello ${`${firstName} ${lastName}.trim()`}!
```

Typer

Objekter

```
const person = {
    name: “Oscar”
}; // TS inferrer typen: { name: string; }

interface Person {
    name: string;
};

type Person = {
    name: string;
};

let person: object = {
    name: “Oscar”
}; // Får den mer generiske typen object (sier ikke noe om { name: string })
```

Typer

Typer for objekter

- `interface`
  - Beskriver alltid objekter
  - Utvides med `extends`
- `type` – "Egendefinert type"
  - Kan definere både primitive og ikke-primitive typer (objekter)
  - Utvides med `&`
  - Kan lages som union: `type aOrb = "a" | "b";`
- Begge to hoistes til toppen og kan utvide (arve fra) hverandre

Typer
interfaces

- Beskriver kun objekter
- Beskriver hvilke properties et objekt skal ha
- Properties kan være optional:
  - optionalProperty?: string;
- Kan utivde andre, eksisterende interfaces (arv)
- Kan implementeres av klasser

Typer

interface

```
interface Pet {
    name: string;
    numberOfLegs?: number; // numberOfLegs er optional
}

interface FourLeggedPet extends Pet {
    numberOfLegs: number; // Vi overskriver numberOfLegs til å være required
}

interface Dog extends FourLeggedPet {
    bark: () => void;
    fetch: (item: string) => void;
}

const dog: Dog = {
    name: "Bajas",
    numberOfLegs: 4,
    bark: () => console.log("Voff voff!"),
    fetch: (item: string) => console.log(`Fetching ${item}`),
};
```

Typer

interface

```
interface Drivable {
    drive: () => void;
}

class Car implements Drivable {
    color: string;
    brand: string;
    constructor(color: string, brand: string) {
        this.color = color;
        this.brand = brand;
    }
    // Uten denne får vi: "Class 'Car' incorrectly implements interface 'Drivable’."
    public drive() {
            console.log("Driving!");
    }
}
```

Typer

type

```
type Pet = {
    name: string;
    numberOfLegs?: number;
}

type FourLeggedPet = Pet & {
    numberOfLegs: number;
}

type Dog = FourLeggedPet & {
    bark: () => void;
    fetch: (item: string) => void;
}

const dog: Dog = {
    name: "Bajas",
    numberOfLegs: 4,
    bark: () => console.log("Voff voff!"),
    fetch: (item: string) => console.log(`Fetching ${item}`),
};
```

Typer

type

```
type Drivable = {
    drive: () => void;
}

// Klasser kan også extende type som beskriver objekter
class Car implements Drivable {
    color: string;
    brand: string;
    constructor(color: string, brand: string) {
        this.color = color;
        this.brand = brand;
    }
    // Uten denne får vi: "Class 'Car' incorrectly implements interface 'Drivable’."
    public drive() {
            console.log("Driving!");
    }
}
```

Typer

type

```
type stringCopy = string; // Gir ikke mening, men dette er mulig

// Union types:
type stringOrNumber = string | number;
type trueOrFalse = true | false | "true" | "false" | 0 | 1;

type HasTrueOrFalse = {
    // Hvilken som helst nøkkel med typen string kan ha verdien til typen trueOrFalse
    [key: string]: trueOrFalse;
}

const hasTrueAndFalse : HasTrueOrFalse = { someKey: true, someOther: "false" };

// Union types er ikke begrenset til kun primitiver
type hasOrIsTrueOrFalse = HasTrueOrFalse | trueOrFalse;

console.log(hasTrueAndFalse["someKey"]); // true – mulig pga [key: string]: trueOrFalse;
```

Typer

Omit<Type, Keys>

```
interface Pet {
    name: string;
    owner: Person;
    numberOfLegs?: number;
}

type Snake = Omit<Pet, "numberOfLegs">;
type SnakeForSale = Omit<Pet, "numberOfLegs" | "owner" >;
// Alternativt:
type SnakeAlsoForSale = Omit<Snake, "owner">;

interface Fish extends Omit<Pet, "numberOfLegs"> {
    numberOfFins: number;
}
```

bouvet

Typer

Arrays

```
// Her må vi definere typen, hvis ikke inferres [] som typen never[]
let fruits : string[] = [];

fruits.push("Apple");

// Her må vi også definere typen, hvis ikke inferres [] som typen never[]
let altFruits : Array<string> = [];
altFruits.push("Apple");
```

Typer

Arrays

```
// Arrays kan være readonly
const readOnlyFruits : readonly string[] = ["Orange", "Apple"];
let readOnlyFruitsAlt : ReadonlyArray<string> = ["Orange", "Apple"];

// Property 'push' does not exist on type 'readonly string[]'.ts(2339):
readOnlyFruits.push("Pineapple");
readOnlyFruitsAlt.push("Pineapple");

// Index signature in type 'readonly string[]' only permits reading.ts(2542)
readOnlyFruits[0] = "Grape";
readOnlyFruitsAlt[0] = "Grape";

fruits2 = ["Tomato"]; // readOnlyFruitsAlt er en let der det er lov å re-assigne verdien

// Kan også lages som tupler - fast lengde med fast type på gitt indeks:
type customReactHookReturnType = [string, (newValue : string) => void];
```

Generiske parametere

“Variable” typer

- Defineres ofte som T, K eller annen stor forbokstav

```
const arrayIsEmpty = <T> (array: T[]) => array.length === 0;

interface Namable {
    name: string;
}

// Kan legge på constraints for objekter
const hi = <T extends Nameable> ({ name } : T) => `Hello ${name}!`;
hi({ name: "Christian", age: 27 }); // "Hello Christian!"
```

Keys

Nøkler for typer som beskriver objekter

- Operatoren **keyof** lager en union type av nøklene til et objekt

```
interface MyInterface {
    a: string;
    b: number
};
type myKeyType = keyof MyInterface; // Får typen "a" | "b”

// Kan definere nøkler basert på nøkkelverdiene til andre typer, kalles "mapped types":
type MyMappedType = {
    [key in keyof MyInterface]: string | number;
} // OBS! Dette er ikke lenger mulig i interface, så her må vi bruke type
// Her får vi typen { a: string | number, b: string | number }
```

typeof

Lag en egen type basert på en verdi

```
let str = "Some text";
type strTypeFromLet = typeof str; // Får typen string
const str = "Some text";
type strTypeFromConst = typeof str; // Får typen "Some text"

const myObject = {
     a: "A string",
     b: 22,
};
type MyType = typeof myObject; // Får typen { a: string; b: number }

// Kan kombineres med f.eks. keyof:
type myKeyType  = keyof typeof myObject; // Får typen "a" | "b”
```

Typer

Spesielle typer

- `any` – “Hva som helst”
  - Brukes typisk når du bruker en npm pakke som ikke har type definitions
  - Unngå helst, så langt det er mulig
- `unknown` – “Ukjent”
  - Brukes typisk når du ikke vet hva du får, f.eks. på en parameter til en util-funksjon som skal sjekke om “noe” er et tall.
  - Til forskjell fra any - du får ikke lov å gjøre noe med verdien uten å først spesifisere typen nærmere
- `never` – “Ingenting” / “Det som aldri skjer”
  - Representerer noe som ikke er mulig, går galt eller som aldri terminerer
  - *“The never type is a subtype of, and assignable to, every type; however, no type is a subtype of, or assignable to, never (except never itself). Even any isn’t assignable to never.”*

Spesielle typer

any vs unknown

```
// any
const isNumber = (num: any) => typeof num === "number";
isNumber(3); // true
isNumber("3"); // false

let anyValue: any = "any value"; // Vi kan assigne hva som helst til any
let str1: string = anyValue; // Any kan assignes til hva som helst

const anyInputIsValid = (input: any) => input && input.length;

// unknown - "Anything is assignable to unknown, but unknown isn't assignable to anything but itself"
const isString = (str: unknown) => typeof str === "string";
isString(3); // false
isString("3"); // true

let unknownValue: unknown = "unknown value"; // Vi kan assigne hva som helst til unknown
let str2: string = unknownValue; // Ikke lov: vi kan ikke assigne unknown til hva som helst
// Da får vi: Type 'unknown' is not assignable to type 'string'.ts(2322)

const unknownInputIsValid = (input: unknown) => input && (input as string).length;
```

Spesielle typer

never

```
// never - "Computer says no!"
const iIamNeverGoingToTerminate = () => {
    while (true) {
            console.log("Infinite loop in progress... please wait");
    }
}; // Får typen () => never

const iThrowErrorsAllTheTime = () => {
    throw new Error("An error occurred!");
}; // Får typen () => never

let emptyArr = []; // Får typen never[]

type impossible = string & number; // Får typen never
```

Typer

Spesielle typer

- **union types**

```
type trueOrFalse = "yes" | "no" | true | false;
const trueConst: trueOrFalse = "maybe";
    // Type "maybe" is not assignable to type 'trueOrFalse'.ts(2322)
type language = "no-nb" | "no-nn" | "en";
```

- conditional types

```
type StringOrNot<T> = T extends string ? string : never;

type isNever = StringOrNot<t>; // Får typen never

type isString = StringOrNot<string>; // Får typen string

type isAlsoString = StringOrNot<"hello">; // Får typen string
```

Eksterne pakker/dependencies

Hvordan bruke eksterne pakker i ditt TS prosjekt

- Built in types – types kommer med pakken 🥳
- DefinitelyTyped: https://github.com/DefinitelyTyped/DefinitelyTyped
- Worst case må du type importene som any 😬

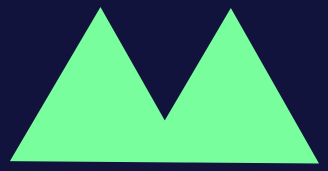
bouvet

Definere typer for data fra API

Noen tips

- Noen verktøy som f.eks. swagger har plugins for å autogenerere typer
- Hvis du jobber med backend-utvikleren må dere snakke sammen og bli enige om typene
- Det kan føles som mye jobb å spesifisere typer på noe du ikke selv har kontroll over, men det er vel verdt innsatsen hvis du først bruker TS

Spørsmål?




https://github.com/cbroko/viteTs