**Engineer**

# TypeScript

JavaScript med typer som sjekkes før det kjører

## Hva det er

TypeScript, ofte forkortet TS, er JavaScript med typeannotasjoner, laget av Microsoft og levert sammen med en kompilator, `tsc`, som sjekker dem. Filene heter `.ts`, eller `.tsx` når de inneholder JSX-markup. All gyldig JavaScript-syntaks er også gyldig TypeScript, og derfor kalles språket et supersett av JavaScript. Kompilatoren er åpen kildekode under Apache License 2.0.

Typene lever bare så lenge sjekken varer. Vanlige typeannotasjoner viskes ut når TypeScript genererer JavaScript; de validerer ikke data under kjøring og krever ikke noe bibliotek for typekontroll under kjøring. Typeannotasjoner endrer ikke hvordan JavaScript-koden oppfører seg når den kjører. Det ene faktumet forklarer både nytten og grensene: sjekkeren resonnerer om typene som er beskrevet i kildekoden; den validerer ikke data som kommer inn mens programmet går.

Typesystemet er strukturelt. To typer er kompatible når formene deres stemmer, uansett hva de heter, og det passer vanen i JavaScript med å sende vanlige objekter rundt. C# og Java sammenligner i stedet de deklarerte navnene. Mye av typingen utledes av kompilatoren, så annotasjonene står mest på funksjonssignaturer og eksporterte verdier. Noen steder er systemet med vilje ikke vanntett, det engelske fagordet er *unsound*: `any` slår av sjekken for en verdi, og en typepåstand skrevet med `as` ber kompilatoren ta programmererens ord for typen. Begge forsvinner fra resultatet sammen med alt det andre.

Tre måter å kjøre det på er vanlige. `tsc` sjekker et prosjekt og skriver ut JavaScript. Buntere som esbuild fjerner typene uten å sjekke dem og overlater sjekken til en egen kjøring av `tsc --noEmit`. Node.js kjører siden v23.6.0 og v22.18.0 `.ts`-filer direkte ved å erstatte typesyntaksen med mellomrom, igjen

uten typesjekk, og godtar bare syntaks som kan fjernes på den måten. TypeScript 7.0, kunngjort i juli 2026, er kompilatoren portet til Go, og teamet oppgir at fulle bygg typisk går 8–12 ganger raskere.

Bruk begrepet i en beslutning

Bruk «Hva er TypeScript?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Grensesnitt som laster raskt, fungerer på mobilen og oppfører seg likt i alle nettlesere kundene dine måtte bruke.

**Se tjenestene våre** Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Styrte fullstack-utviklingen av GIS-kart — fullstack-kartutvikling, fra database til nettleser

**Styrte fullstack-utviklingen av GIS-kart** Ta kontakt →

## Hvorfor det er viktig

Et team som vurderer TypeScript, avgjør egentlig om et byggetrinn og en sjekker betaler for seg ved å fange feil før koden kjører. I en kodebase som flere personer endrer hver uke, gjør de som regel det. Gi et felt nytt navn, og hvert sted som brukte det, dukker opp. Editoren fullfører og forklarer kode ut fra typene, og signaturen til en modul blir en kontrakt som kallerne ikke kan bryte i det stille. Prisen er en sjekk i hver byggepipeline, en konfigurasjon med svært mange valg og typedefinisjoner som må holde tritt med bibliotekene de beskriver.

Strengheten er innstillingen som betyr mest. Flagget `strict` slår på en hel familie sjekker, bl.a. `strictNullChecks` og `noImplicitAny`, og håndboken beskriver resultatet som en dreieknapp heller enn en av/på-bryter. Siden TypeScript 6.0 er flagget på som standard, så et nytt prosjekt starter strengt. Et eldre prosjekt kan fortsatt sette det til false, og å slå det på senere betyr å arbeide seg gjennom hver fil det berører. To nyttige sjekker står utenfor familien og må slås på hver for seg: `noUncheckedIndexedAccess` og `exactOptionalPropertyTypes`.

Utviskingen setter grensen. Et program som leser JSON fra en nettverksforespørsel, en fil, en kø eller et webskjema, har bare programmererens ord for hvordan de dataene ser ut, fordi typene er borte når dataene kommer. Grensene trenger derfor validering under kjøring, helst generert fra det samme skjemaet som typene, slik at sjekken og typen ikke kan være uenige. TypeScript gjør innsiden av et program konsistent og overlater kantene til koden.

Sjekkeren mater også andre verktøy. En linter som typescript-eslint kan be kompilatoren om typeinformasjon, slik at en regel kan flagge et promise som ingen venter på med `await`, eller en verdi av typen `any` som sprer seg gjennom en modul. På den måten gjør et team sine egne konvensjoner om til feil i bygget i stedet for kommentarer i kodegjennomgangen.

## Slik ser det ut i praksis

I prosjektene på dette nettstedet holdes TypeScript ærlig av generert kode og av verktøy, ikke av gode forsetter. Tre sider viser hvordan.

På én plattform skrives TypeScript-klienten til frontenden aldri for hånd. Den genereres fra API-ets egen beskrivelse, som Huma lager ut fra Go-typene, som igjen følger databaseskjemaet og funksjonene i det. På det dokumenterte implementeringsstadiet var klienten på 44 076 linjer, og 61 mock-handlere som ble generert ved siden av, lot frontendens egne tester kjøre mot den virkelige kontrakten og ikke mot håndskrevne testdata. Hvert ledd har sin egen vakt. En kontraktskontroll i CI sammenligner det frontenden sender, med det API-et forventer, og får bygget til å feile ved avvik, med en skjemaprobe under som undersøker de faktiske formene. Den dekker valgfrie felt i forespørselskroppen, der «mangler» og «null» lett blandes sammen, og enumer i spørringsparametere. Et felt som endres på den ene siden, får nå et bygg til å feile minutter etter endringen, i stedet for å dukke opp som en 422 i nettleseren etter at avviket allerede er levert. Til gjengjeld når en endring i kontrakten bare frem til klientens typer ved at hele kjeden kjøres på nytt, og regenerering krever ekstraarbeid.

En annen side setter én standard for seks språk, TypeScript blant dem. Hver diagnostikk ESLint melder, er en feil, det finnes ikke noe «warn»-nivå å gjemme seg i, og inline-undertrykkelser er forbudt, så et funn må rettes og ikke bare dempes. Pre-commit- og pre-push-hooks kjører linterne og testene, og lengdegrenser for funksjoner og filer holder modulene lesbare.

Linterne som følger med et språk, stopper før prosjektets egne regler, og derfor holdt femten egne ESLint-regler repoets egne kodemønstre på plass i TypeScript på det dokumenterte implementeringsstadiet. Reglene for TypeScript var bare én del av en kontrollmotor der 268 av de 274 kontrollskriptene var registrert i commit-hookene, og kontrollene hadde 146 egne tester, fordi en ødelagt kontroll slipper alt gjennom uten at noen merker det. Én regel handler om tester

og ikke om typer: en commit med prefikset `fix:` må ha med en test som feiler uten rettelsen. Gjennomgangstiden flyttet seg fra mekanikk til design, og til gjengjeld ble det tregt å committe.

## Hvor det går galt

De fleste feil med TypeScript kommer av å stole på typene lenger enn kompilatoren kan se.

**Et felt typet som streng viser seg å være undefined under kjøring.** Dataene kom utenfra og fikk en type i stedet for en sjekk. DOM-biblioteket typer `res.json()` som et promise av `any`, så kompilatoren godtar hva enn koden gjør med resultatet, og en typepåstand gjør tilliten eksplisitt uten å gjøre den trygg:

```
const res = await fetch("/api/user");
const user = (await res.json()) as User; // stolt på, aldri sjekket
user.email.toLowerCase(); // TypeError hvis email mangler
```

Behandle data utenfra som `unknown`, valider dem ved grensen med et skjema som sjekkes under kjøring, og generer typen fra det skjemaet, så holder de to tritt. En typet lint-regel kan deretter flagge hver `any` som slipper forbi.

**Kode med typefeil havner i produksjon likevel.** `tsc` skriver fortsatt ut JavaScript når den melder feil, fordi `noEmitOnError` står på false som standard, og buntere og Node.js fjerner typene uten å sjekke dem i det hele tatt. Kjør `tsc --noEmit` som et obligatorisk steg i CI og i pre-commit-hooken, og la bygget feile på avslutningskoden.

**Et oppslag på indeks gir undefined der typen sa noe annet.** Å lese et array-element, eller en nøkkel gjennom en indekssignatur, gir en type som alltid er til stede, selv forbi slutten av arrayen. Slå på `noUncheckedIndexedAccess`, som legger `undefined` til slike lesinger og tvinger koden til å håndtere det. Kombiner det med `exactOptionalPropertyTypes`, som skiller en egenskap som mangler, fra en som er satt til `undefined`.

**En fil som kompilerer med tsc, feiler når Node.js kjører den.** Typestripping fjerner bare syntaks som er uten betydning under kjøring. Enumer og navnerom (*namespaces*) som inneholder verdier, får Node.js til å stoppe med

`ERR_UNSUPPORTED_TYPESCRIPT_SYNTAX`, og en type som importeres uten nøkkelordet `type`, behandles som en verdiimport som feiler under kjøring. Slå på `erasableSyntaxOnly` og `verbatimModuleSyntax`, så melder kompilatoren begge deler før Node.js gjør det. Skriv `import type` for typer, og erstatt enumer med objektliteraler merket `as const`.

## Les mer

- TypeScript Documentation — håndboken, referansesidene og versjonsnotatene, vedlikeholdt av TypeScript-teamet.
- The Basics — kapitlet i håndboken om statisk sjekking, utskrift tross feil, utviskede typer og strenghet.
- TSConfig Reference: strict — hvert kompilatorvalg med standardverdien, også sjekkene som `strict` slår på.
- Type Compatibility — strukturell subtyping og stedene der typesystemet med vilje har hull.
- Node.js: Modules: TypeScript — typestripping, syntaksen den avviser, og kompilatorinnstillingene Node.js anbefaler.
- Linting with Type Information — hvordan typescript-eslint gir lint-reglene tilgang til typesjekkeren.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

**Se hva det koster** Ta kontakt →

# Gjør begrepene praktiske

Spør hvordan «Hva er TypeScript?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/typescript/