Jeg har valgt den nye teknologi - TypeScript, fordi det er et grundlag for at lære React senere. TypeScript er et moderne programmeringssprog, som er en overbygning på JavaScript.

I starten har jeg installeret TypeScript globalt på mit system ved at køre følgende kommando: *npm install -g typescript*.

Bagefter fik jeg Node.js's pakkehåndteringssystem med *npm*-kommando.

Det er praktisk at bruge en *tsconfig.json*-fil, som specificerer rodmappe og kompileringsindstillinger for projektet. Denne fil fik jeg nemt ved at køre: *tsc --init*.

TypeScript-kompileren (tsc) er det centrale værktøj, der omdanner TypeScript-kode til standard JavaScript, som jeg har brugt i min terminal med commandoen *tsc* for at kompilere index.ts-filen til index.js-filen, så mit projekt kan køre i browseren.

Formålet med TypeScript er at forbedre udviklingen af JavaScript-applikationer ved at tilføje avancerede funktioner. Her er nogle specifikke formål:

- **Fejlhåndtering**: Tidlig fejlopfangst under udvikling før koden udføres i produktion.
- **Forbedret kodekvalitet**: Tilføjelse af *interface* og *datatyper* i TypeScript hjælper med at definere klare og strukturerede typer for objekter i min applikation. Dette forbedrer ikke kun typesikkerheden, men forbedrer også kodelæsbarheden og vedligeholdelsesvenligheden, ligesom er der i mit projekt, f. eks.:

```
interface Task {
    id: number;
    title: string;
    completed: boolean;
}
```

```
const taskInput = document.getElementById('task') as HTMLInputElement;
const taskList = document.getElementById('taskList') as HTMLUListElement;
```

Disse ekstra typeangivelser hjælper med at sikre, at værdierne bruges konsekvent gennem min kode. Disse typecasts i TypeScript *HTMLInputElement*, *HTMLUListElement*, og *HTMLFormElement* giver følgende formål:

- **Sikre Typestyring**: Ved at tilføje specifikke typer til mine DOM-elementer (i dette tilfælde HTML-input, unorderlist og form), sikrer mig, at TypeScript ved præcis, hvilke typer objekter det arbejder med. Dette gør koden mere robust og minimerer risikoen for fejl.
- **Reducerer Fejl**: Typecasting hjælper med at reducere runtime-fejl ved at sikre, at vi kun får adgang til de egenskaber og metoder, der er tilgængelige for det specifikke element. For eksempel vil *taskInput.value* kun være tilgængelig, hvis *taskInput* er af typen *HTMLInputElement*.
- **Let integration**: Da TypeScript er tæt integreret med JavaScript, var det let at integreres i den eksisterende JavaScript-projekt.

I mit projekt har jeg anvendt TypeScript i følgende områder:

- ***Task interface(task):*** for at definerer strukturen af et objekt med properties for: **id** (number), **title** (string), **completed**(boolean).
- ***as HTMLInputElement***- til at få direkte adgang til *taskInput*.

- ***void -*** til at angive, at en funktion ikke returnerer nogen værdi. Funktionen ***updateTasks()*** er en sådan funktion. Den udfører en række handlinger (opdaterer listen over opgaver på skærmen), men den returnerer ikke noget resultat, som kan bruges af den kode, der kalder funktionen.
- inde i function ***toggleTasks()*** og i **deleteTasks()** har jeg includeret ***(taskId:number),*** som specificerer at property **taskId** skal være kun en numerisk værdi. Dette hjælper TypeScript-kompilatoren med at forstå, hvilken type data der skal arbejdes med, når funktionen kaldes.

Selvom mit projekt er lille, har jeg lært at anvende nogle af de nye funktioner i TypeScript. Jeg vil sige, at TypeScript er godt struktureret og hjælper med at opdage fejl hurtigt under udviklingen i terminalen. Men som ønske tænker jeg, at TypeScript kunne tilbyde flere tutorials og eksempler for en bedre forståelse af den nye teknologi.