# Projekt: Web App

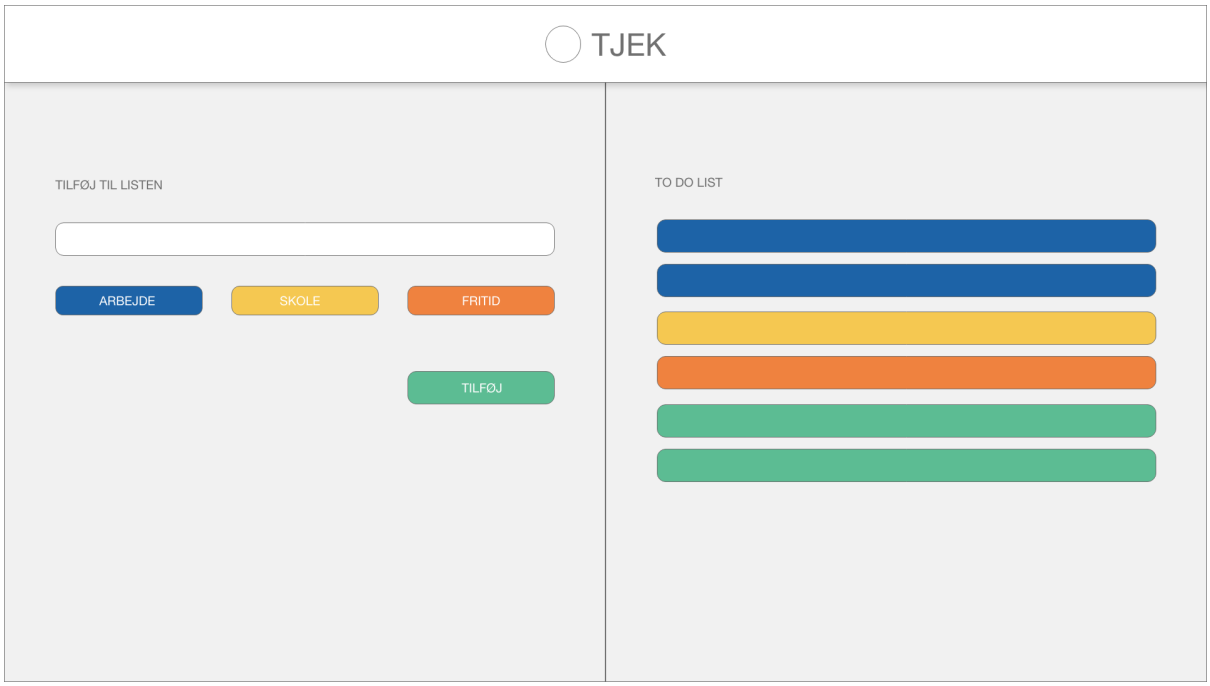
## Redegørelse

Vi har valgt at lave en To Do liste hvor brugerne selv kan skrive flere opgaver til listen. Der er tre kategorier hvor opgaverne kan inddeles i (fritid, arbejde, skole). Hver kategori har sin egen farve for at gøre det mere overskueligt for brugeren at se hvilken opgave hører til hvilken kategori. Når de har udført deres opgave som der står på listen kan de trykke på den og den bliver overstreget og der kommer et Tjek tegn (derfor navnet "Tjek") ved siden af opgaven. De kan også fjerne opgaven helt fra listen ved at trykke på krydset i tilfælde af at listen bliver for lang og uoverskuelig.

## Beslutningsfasen

Vi startede med en anden idé som vi blev enige om var for kompleks at arbejde med. Derfor fandt vi på  idéen om at lave en To do liste som vi syntes var mere håndterbar. For dog stadig at udfordre os selv valgte vi at lave flere forskellige funktionaliteter.

Efter at have lave et par skitseringer af hvordan Web App'en skulle se ud lavede vi en mark-up.



Da vi havde færdiglagt idéen til vores Web App, satte vi os nogle læringsmål for hvad vi ville have ud af projektet. Det overordnede mål var at blive mere komfortable med Javascript. Herudfra lavede vi følgende læringsmål:

- Hvordan tilføjer og sletter man en opgave til en liste via en brugergrænseflade?
- Hvordan tilføjer man forskellige kategorier til en opgave via en brugergrænseflade?
- Hvordan filtrere man opgaver/liste-elementer?

Derudover havde vi også nogle andre mere overordnede mål:

- Blive komfortable med at bruge semantic HTML language
- Forstå og anvende brugen af grid og flexbox
- Anvende Separations Of Concern

Da læringsmålene var færdigdefineret, prøvede vi at finde ud af, hvilke kilder og ressourcer vi skulle bruge, for at opnå disse.

## Opbygning

For at gøre det lettere for os selv at opnå det ønskede resultat af projektet, blev det overordnede mål af Web App'ens funktionalitet brudt ned til små forskellige dele. For at synliggøre dette blev kommentare brugt til at opdele Javascript og CSS-filen, der forklare hvad det enkelte afsnit indeholder. Dette gjorde det også nemmere for os at vende tilbage til koden senere i processen.

### HTML

Eftersom vi ønskede at lave en dynamisk App med brugergenereret indhold, valgte vi at opbygge HTML-filen med fhv. lidt indhold og lade det meste af det blive genereret ved brug af metoden createElement i Javascript.

Vi forsøgte at opbygge vores HTML med semantisk mark-up uden det store brug af div og span, hvilket også var en af vores 5 læringsmål (se ovenstående).

## CSS

Vi startede med at anvende flexbox til at opstille kategori-knapperne og tilføj knappen. Men vi fandt hurtigt ud af, at grid fungerede bedre. Her anvendte vi en grid metode med autofill og minmax, der selv sørger for at opstille layoutet af knapperne i mobil- og tablet version.

Ud over det har vi fundet ud af, hvordan man skal bruge Javascript og CSS sammen når det kommer til at tilføje, toggle og fjerne klasse ved forskellige funktioner. I forlængelse af det, har vi prøvet at respektere princippet fra 'seperations of concern' ved ikke at bruge 'style' metoden i Javascript til f.eks. at fjerne/skjule et element.

For at gøre vores styling mere overskuelig har vi anvendt kommentarer til at opdele CSS-filen. Den er inddelt i 5 forskellige kategorier (header, body, venstre sektion, højre sektion og Javascript CSS classes).

## Javascript

Den overordnede funktionalitet, der gør det muligt for brugeren af skrive, tilføje og slette en opgave til og fra To-Do listen, samt at give den enkelte opgave en kategori, er samlet under én og samme klik-funktion kaldet "addItem". Dennes indhold er primært fundet og inspireret fra www.w3schools.com.

Funktionen genererer en 'li' i takt med at den om-genererer indholdet i input-feltet til en textNode, der bliver tillagt 'li' ved brug af appendChild. Hvis input-feltet ikke er tomt, bliver den genererede 'li' tillagt To-Do listen igen via appendChild.

```
function addItem() {
    let li = document.createElement("li");
    let inputValue = document.querySelector("#myInput").value;
    let t = document.createTextNode(inputValue);
    li.appendChild(t);

    //Hvis input feltet er tomt, do nothing
    if (inputValue === "") {
        console.log("skriv noget!")

        //Hvis ikke det er tomt, tillæg list-item til min liste
    } else {
        myToDoList.appendChild(li);
        console.log("virker det?");
    }

```

Vi ønskede at gøre det muligt for brugeren at slette en opgave og faldt over w3schools eksempel, der gør brug af en loop. Vi prøvede at se om det var muligt, at skabe samme funktionalitet uden brug af loop, da dette er et emne vi alle har været udfordret med at forstå. Vi måtte dog erkende til sidst, at det ikke var muligt uden, da vi anvender querySelecterAll til at fange alle 'li'. Vi endte derfor med at anvende eksemplet fra w3schools.com med nogle enkelte justeringer. Loopet kører igennem de genererede 'li' og tillægger dem hver især en eventListener og derefter en klik-funktion, der fjerner den pågældende 'li' ved hjælp af en CSS-klasse der indeholder 'display: none'.

```
    //Slet listItem når man klipper den tilhørerende 'luk' knap
    let close = document.querySelectorAll('.close');
    let i;
    for (i = 0; i < close.length; i++) {
        close[i].addEventListener('click', function () {
            console.log("something's working")
            this.parentElement.style.display = "none";
            console.log(this.parentElement);
        })
    }

```

Næste udfordring var at finde ud af, hvordan vi kunne kategoriserer opgaverne så de fik en bestemt farve. Spørgsmålet var hvordan vi fik fat i et lokalt-defineret 'li' element og bruge det i en global sammenhæng - altså bruge det i en anden klik-funktion. Dette blev løst via en if/else statement, der finder ud af, hvilken kategori brugeren har tildelt den pågældende opgave og derefter give den pågældende 'li' en bestemt CSS-klasse.

## Links

Link til GitHub repository: Web-App-Tjek
https://github.com/odjernes/Web-App-Tjek