Vanilla JavaScript -> ES6 -> Object Orienteret

Indhold

Der arbejdes med følgende opsætning:

- Node.js
- npm
- Webpack
- Es6 (Javascript)
- Sass

Node.js:

Node.js er et open-source JavaScript run-time miljø, som bruges til udvikling af webapplikationer. Det anbefales at man installerer node.js og npm via nvm (node version manager)
https://nodejs.org/en/download/

Nvm:

Håndtering og installation af Node versioner.

- install nvm (node version manager)
- https://dev.to/skaytech/how-to-install-node-version-manager-nvm-for-windows-10-4nbi
- nvm install 13.13.0 / 16.13.2
- nvm use 16.13.2

nvm list (se installerede versioner af node)

Npm:

pakkehåndtering
https://en.wikipedia.org/wiki/Npm_(software)
https://www.npmjs.com/

Webpack:

Javascript modul bundler.

Deres forside siger alt.
https://webpack.js.org/

installering af webpack. Se: https://webpack.js.org/guides/installation/

Super guide til opsætning af en webpack boilerplate: https://www.valentinog.com/blog/webpack/

ES6:

ECMAScript er et script-programmeringssprog, standardiseret af Ecma International i ECMA-262 specifikationen. Sproget bliver brugt i stor udstrækning på nettet og ofte refereret til som JavaScript.

TypeScript er et andet meget populært sprog og som spiller fint sammen med både Node.js, npm og Webpack.

Sass:
Udvidet css håndtering.
https://sass-lang.com/documentation

Her er en spændende video omhandlende CSS frameworks vs. ”almindelig” CSS:
https://www.youtube.com/watch?v=LwYZTKxj-do

Citat (Scott Tolinsky**):**
You probably don’t need a CSS framework. While CSS Frameworks are wonderful and certainly have their place in the development landscape, they are often overused in place of a custom solution."

In this talk, Scott will show you ways to avoid the additional performance cost of using a CSS framework, and how to "quickly build a robust design system of components tuned to your needs using CSS variables."

Du kan tjekke din node version med cmd promt:

```
node -v
```

og du kan tjekke om du har npm installeret ved at køre denne i cmd promt

```
npm -v
```

Der kan hentes en BoilerPlate **på github:**

https://github.com/lagr-web/boilerplate

Når du har hentet vores boilderplate ned, skal du åbne en cmd promt på din boilderplate, som er din projekt mappe fremover, mappe og skrive følgende. Den nemmeste måde at gøre det på er at højreklikke på mappen, samtidig med at du holder shift tasten ned, så får du mulighed for at vælge at åbne et PowerShell vindue på mappen:

```
npm i
```

Dette opretter en node_modules mappe med alle de node pakker, som der er angivet i package.json. Kan ikke garantere at du skal bruge alle pakkerne 😊

Så kører du en cmd med følgende, som pakker dit projekt og sikrer, at når du har hentet det første gang opretter en dist mappe med en pakket udvikler (mode: development) version.

```
npm run webpack
```

Start dit projekt op ved at køre cmd på mappen

```
npm run start
```

Så åbner browseren og du er klar til at gå i gang.

Mappe struktur:

- Index.html
- dist ->bundle.js
- node_modules
- assets
- css
- src
- app.js

Mappen "dist" og "bundle.js" bliver oprettet første gang du kører cmd > npm run webpack på dit projekt. Hvis du åbner på "webpack.config", kan du på linie 79 se hvilken mappe den laver et output til.

I index.html refereres der til "dist/bundle.js": <script src="dist/bundle.js"></script>

- Mappen "src" bruges til alle dine js filer, der kan sagtens laves undermapper i "src" mappen.
- Mappen "css" indeholder en SASS fil.
- Mappen "assets" indeholder lige nu to undermapper "images" og "sound".

gennemgang af:

1. package.json:
2. webpack.config
3. prod.config

1. package.json: Henvisning til alle hentede npm pakker, som installeres i "node_modules" mappen. Når du starter en nyt projekt op, starter du som regel med at køre en "npm init", som automatisk opretter en "package.json" fil. Derefter kan du hente alle de pakker, som du nu skal bruge i projektet. Det er en proces du kan gøre løbende, efterhånden, som du får brug for pakker. Du kan oprette npm script til at eksekvere kommandoer, så du ikke skal skrive dem hvergang i din terminal(cmd promt).
2. Webpack.config: Opsætning af regler for export af kode og bundler til development brug. Se: https://webpack.js.org/.
3. Prod.config: Opsætning af regler for export af kode og bundler og pakker til produktion.

Her kan du få et hurtigt overblik over webpack: https://webpack.js.org/guides/getting-started/

Webpack har sit eget cli, så du kan bruge en terminal (cmd promt) til at oprette en webpack konfiguration. Du kan læse meget mere om webpack konfiguration på https://webpack.js.org/configuration/ og om hvordan du arbejder med webpacks cli. Husk at det er også helt lovligt at genbruge sin boilerpalte igen. Man behøver ikke oprette en ny fra bunden af, hver gang. En super god ide er at have sin boilderplate

liggende på Github, så kan du have ”version control” på din boilerpalte. Husk ikke smide din ”node_modules” mappe med (sæt regler i dit github projekt ved hjælp af en ”gitignore” fil), da den fylder rigtig meget. Når du opretter et nyt projekt med din boilderpalte, sørger din package.json fil for at installere det nødvendige, når du via terminalen(cmd promt) kører en npm install eller bare npm i .

Der er i ”package.json” lavet små npm scripts, som kan kaldes og gøre bruge af webpack filerne:

```
"scripts": {
    "webpack": "webpack --watch",
    "build": "webpack --config prod.config.js",
    "start": "http-server -p 8080 -o http://localhost:8080/index.html"
  }
```

Link til opskrift på at sætte sin egen boilderplate op (der er mange af dem derude):

https://medium.com/javascript-in-plain-english/the-basics-of-setting-up-an-es6-development-environment-with-webpack-and-babel-153d1bc3b4a5

Men start på Webpack egen site:
https://webpack.js.org/

Alternativer til Webpack:
https://gulpjs.com/ (som også kan bruges sammen med Webpack)

Her er et bud på top 20 alternativer:
https://www.slant.co/options/11602/alternatives/~webpack-alternatives

Hvis du vil lave din egen boilerplate kan du meget simpelt starte med, når du har installeret node, at køre en kommando i en cmd promt på den mappe hvor du har dit projekt:

```
npm init --y
```

--y gør at den laver nogle default indstillinger til dig. Når den kommando er kørt får du package.json fil, som holder styr på dine pakker(npm pakker), så skal der oprettes en webpack konfiguration fil der kan sørge for en masse ting i forhold til hvordan og hvorfra din boilerpalte skal køre og hvordan den skal pakkes.

Når du har hentet vores boilerplate ned:

Først kører du en cmd med følgende:

```
npm run webpack
```

Hvis alt kører som det skal skulle du gerne, når du kører en cmd:

```
npm run start
```

Skulle du gerne få en besked: ”alt ok”

Den console.log kommer fra js filen initialize.js filen.

Åbn projektet i Vs code:

I roden af projektet kan du åbne ”app.js” filen. Denne funktion kaldes en
IIFE: Immediately Invoked Function Expression og er en måde at udføre funktioner med det samme, så snart de oprettes. IIFE'er er meget nyttige, fordi de ikke forurener det globale objekt, og de er en enkel måde at isolere erklæringer om variabler på. Jeg bruger den altid, som mit første kald.

```
import Initialize from './Initialize';
(function () {
    console.log('starting the app');
    let init = new Initialize();
})();
```

Der importeres en klasse ”Initialize”, som kommer fra Initialize.js

Vi laver en instance af Initalize klassen og den bliver eksekveret.

Det vil sige at alt hvad vi laver i Initialize klassen bliver eksekveret ”igennem” App.js

Lad os prøve at oprette klassen Initialize, som bliver kaldt fra App.js

Opret en ny fil i src mappen og navngiv den ”Initilalize.js” og opret en klasse med tilhørende konstruktør:

```
export default class Initialize {
   constructor() {
  }
}
```

Som udgangspunkt, skal alt hvad der skal importeres, eksporteres.

Så en ES6 klasse skal have en export default. En klasse har som default en ”constructor”, som automatisk bliver kaldt, når en klasse initialiseres.

Klassen kan bruges på flere måder. Du kan også oprette din Initialize klasse på følgende måde:

```
class Initialize {
   constructor() {


  }
}
export default Initialize;
```

Det er helt op til dig selv, hvilken model der passer dig bedst. Lad os arbejde lidt videre med Initialize klasse.

Opret en console.log (”alt ok”) i Initialize klassens konstruktøren.

```
class Initialize {
   constructor() {
    console.log(“alt ok”);
  }
}
export default Initialize;
```

For at se ændringerne skal du køre en cmd på dit projekt.

```
npm run webpack
```

Når du har kørt den en gang, vil webpack selv sørge for at opdatere dit projekt, når du gemmer.

Så er vi i gang og kan nu fokusere på Javascript (ES6).

- https://medium.com/javascript-in-plain-english/javascript-es6-tutorial-a-complete-crash-course-on-modern-js-a09294bffdb7
- https://www.freecodecamp.org/news/make-your-code-cleaner-shorter-and-easier-to-read-es6-tips-and-tricks-afd4ce25977c/

Hvad er OOP: https://da.wikipedia.org/wiki/Objektorienteret_programmering

Variabler:

Vi arbejder i dette projekt med 3 former for variabler:

- Let

- Const
- this.myVariables (tilknyttet en klasse)

I ES6 er variabler typeløse. Det vil sige at i forhold til mange andre sprog definerer man ikke hvilken type det er: Number, String osv.

Når du erklærer en variabel f.eks.:

let myNumber=50; så arbejder vi med en numerisk værdi.

Når vi arbejder i et miljø med klasser kan vi angive en variabel med en et "this keyword":

this.counter = 0;

Så lever variablen i klassen og kan kaldes af f.eks. af en funktion uden for klassens konstruktør.

Du kan selvfølgelig stadigvæk bruge let og const variabler i din klasse.

Husk at variabler i ES6 er case sensitive.

Mere om regler for navngivelse af blandt andet variabler:
https://www.robinwieruch.de/javascript-naming-conventions

```
export default class Initialize {
   constructor() {
    console.log("alt ok");
     this.counter = 0;
     const myName = "Ollerik Wandbjerg";
     let age = 46;
  }
}
```

Funktioner:

Lad os oprette en funktion i vores initialize klasse og kalde den fra initalize konstruktøren:

En funktion i en ES6 klasse vil ligge inden for klasse erklæringen, men uden for konstruktøren, som i sig er en funktion.

```
export default class Initialize {
   constructor() {
```

```
    console.log("alt ok");

     this.counter = 0;
     const myName = "Ollerik Wandbjerg";
     let age = 46;
     this.myFunc();
  }

     myFunc(){
     console.log(this.counter);
   }
}
```

Opgave 1: Prøv at console.log de andre variabler i din konstruktør og se hvad der sker. Kan man løse den problematik der opstår i forbindelse med et funcktionskald. **Hint:** argumenter.

Vi skal have oprettet et object i vores initialize klasse:

```
export default class Initialize {
   constructor() {
    console.log("alt ok");

     this.counter = 0;
     const myName = "Ollerik Wandbjerg";
     let age = 46;

     this.weather = {
       author: "The Weatherman",
       month: ["januar", "februar", "marts", "april", "maj", "juni"],
       averageTemperature: [-10, -10, -10, 12, 15, 20, 25],
       iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot"]
        }

     this.myFunc();
```

```
  }

     myFunc(){
     console.log(this.counter);
   }
}
```

Opgave 2. Kald objektet this.weather i din function og console.log author

Du kan også vælge at **returnere** værdien af dit kald. I stedet for at udskrive værdien i funktionen.

```
export default class Initialize {

    constructor() {

        console.log('alt ok');

        this.counter = 0;
        const myName = "Ollerik Wandbjerg";
        let age = 46;

        let firstNumber = 10;
        let secondNmmber = 20;

        this.weather = {
            author: "The Weatherman",
            month: ["januar", "februar", "marts", "april", "maj", "juni"],
            averageTemperature: [-10, -10, -10, 12, 15, 20, 25],
            iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot"]
        }

        console.log(this.myFunc(myName));

    } // END constructor
```

```
    myFunc(mn) {
        return mn;
    }

}
```

Opgave 3. Send firstNumber og secondNumber til din funktion og returner en samlet beregning på de to numeriske variabler.

Løkker**(loops)**

Vi har oprettet objektet this.weather, som består både af String elementer og Array elementer. Lad os tage fat i month arrayet i this.weather objektet og loope igennem det.

```
export default class Initialize {

    constructor() {

        this.weather = {
            author: "The Weathermann",
            month: ["januar", "februar", "marts", "april", "maj", "juni"],
            averageTemperature: [-10, -10, -10, 12, 15, 20, 25],
            iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot"]
        }

//eksempel med forEach
        this.weather.month.forEach(item => {
           console.log(item);
        })

//eksempel med for loop
        for(let i =0; i< this.weather.month.length; i++){
          console.log(this.weather.month[i]);
```

```
        }

    } // END constructor
}
```

Der er mange måder at loope igennem et array i ES6 og alt efter hvordan data skal håndteres. Der bruges arrow funktioner, som gør vores indhold nemmere at håndtere, specielt når vi arbejde med this nøgleordet(keyword) i javascript klasser. Læs mere om arrow funktioner her:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

https://www.w3schools.com/js/js_arrow_function.asp

Hvis vi nu skulle bruge det sammen med HTML kan man med fordel bruge array.map.

Vi skal have oprettet en html > div med Javascript. Ikke at det er et krav, men det giver måske en bedre logik i ens kode? Derefter laver vi et loop, som kører igennem vores data objekt this.weather og som tager fat i month arrayet. Som vi looper igennem og nester det i this.container div elementet. I et array map() metode kan man få adgang til selve objektet og indekseringen af objektet, og det bruger vi i den div struktur vi bygger op. Vi vender tilbage til document.createElement senere.

```
import "@babel/polyfill";
//import 'bootstrap/dist/css/bootstrap.min.css'
import "../css/style.scss";

export default class Initialize {

    constructor() {

        this.container = document.createElement('div');
        this.container.id = "container";
        this.container.className = "container";
        document.body.appendChild(this.container);

        this.container.innerHTML = '' + this.weather.month.map((item, index) => {
            return `
                <div class = '${"maps" + index}' id='block'>${item}</div>
            `
```

```
        }).join('') + '';

    } // END constructor
}
```

I dette eksempel bliver der brugt template literals, som giver mulighed for at inkludere expressions, blandt andet variabler:

```
let name = ”The Weathermann”;
```

Kan med template literals bruges sammen med html:

```
let name= ”The Weathermann”;
Const markup = `<div class=”person”>${name}</div>`;
document.body.innerHTML = markup;
```

Opgave 4: Sørg for at jeres HTML struktur vises korrekt uden fejl, da vi skal bruge det videre. Det i skal have nu er følgende:

```
import "@babel/polyfill";
import "../css/style.scss";

export default class Initialize {
    constructor() {

   this.weather = {
        author: "The Weathermann",
        month: ["januar", "februar", "marts", "april", "maj", "juni" , "juli"],
        averageTemperature: [-10, -10, -10, 12, 15, 20, 25],
        iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot"]
     }

    this.container = document.createElement('div');
    this.container.id = "container";
    this.container.className = "container";
```

```
    document.body.appendChild(this.container);


    this.container.innerHTML = '' + this.weather.month.map((item, index) => {
         return `
              <div class = '${"maps" + index}' id='block'>${item}</div>
             `
     }).join('') + '';

  } // END constructor
}
```

Og skulle gerne se sådan her ud visuelt (StepOne.js).

Sass:

Sass er en udvidelse til CSS3, som tilføjer indlejrede regler, som variabler, mixins og nedarvinger til selektorer og meget mere. Sass generer god og korrekt formateret CSS og gør dit stylesheet meget nemmere at organisere og vedligeholde.

https://sass-lang.com/

CSS til denne opgave:

Der er tilknyttet et stylesheet, som er opbygget i Sass og som sikrer at vores div struktur, både er responsiv og placerer de enkelte div elementer horisontalt. I dette tilfælde er der gjort brug af CSS Grid:

https://css-tricks.com/snippets/css/complete-guide-grid/

Der er bygget columns omkring en hovedcontainer, som er centreret, der samtidig er sat til, at alt hvad der er i div hovedcontaineren, har et flow, der gør at alle nestede div bliver til en column. Det gælder både div til:

- visuel temperatur visning
- reelle temperatur
- månedsvisning.

Overstående er ikke bygget endnu, det kommer vi til senere.

Hovedcontaineren er sat til at have en position:absolute, da den gerne skulle ”klister” til bunden af browser vinduet.

Hver af de nestede div er hovedsaligt sat til at have en margin og padding for at kontrollerer afstand. Der er gjort brug beregning via calc, så vi kan skalerer efter viewporten settings.

Se style.scss *for at få et overblik.*

Vi skal bygge videre på siden, og til det skal vi bruge en mere avanceret datastruktur.

Opgave 5: Udvid data objektet this.weater med følgende og sikre dig, at der ikke kommer fejlmeddelelse i konsollen:

```
this.weather = {
      author: "The Weatherman",
      month: ["januar", "februar", "marts", "april", "maj", "juni", "juli"],
      temperature: [
        {
          year: "2018",
          averageTemperature: [5, 4, 3, 11, 14, 21, 23]
        },

        {
            year: "2019",
          averageTemperature: [2, 5, 5, 12, 13, 22, 24]
        },

        {
            year: "2020",
          averageTemperature: [12, 8, 2, 14, 17, 29, 22]
        },
      ],
      iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot", "hot"]
    };
```

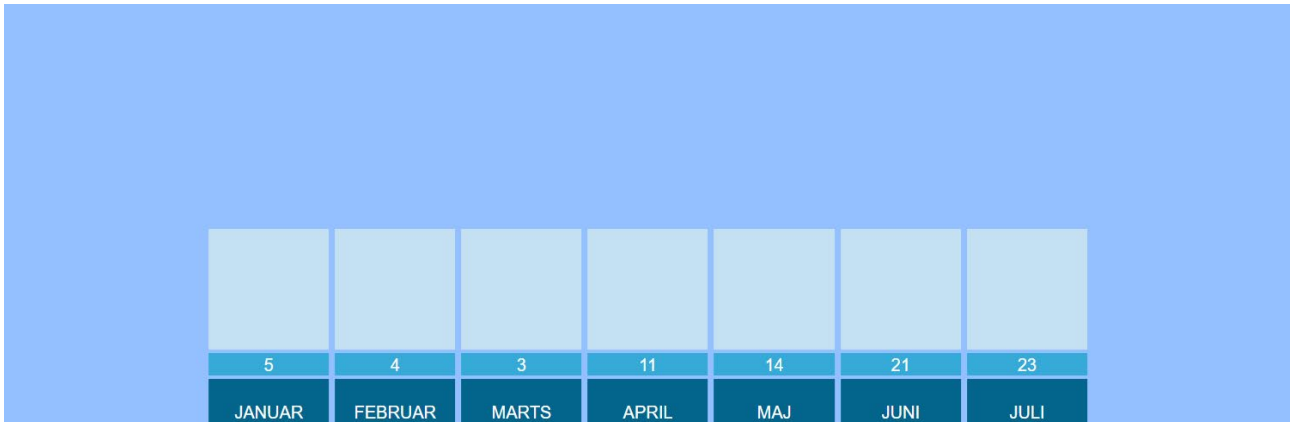
Når det er på plads skal der via vores array.map() metode udskrives yderligere to div elementer din innerHTML:

```
   this.container.innerHTML = "" +
      this.weather.month.map((item, index) => {


          return `
            <div id="childCon">
                 <div class = '${"maps" + index}' id='block'></div>
                 <div class="temperature">${this.weather.temperature[0].averageT
emperature[index]}</div>
                 <div class="month">${this.weather.month[index]}</div>
             </div>


                `; //END return
        })
        .join("") + "";
```

Der er oprette en div->childCon, som nester:

- div->block
- div->temperature
- div->month

Husk at tjekke din console hvis det ikke virker. Du skulle nu gerne have følgende visuelt (StepTwo.js):



… og koden skulle så gerne se sådan her ud:

```
import "@babel/polyfill";
import "../css/style.scss";

export default class initialize {
  constructor() {



    this.weather = {
      author: "The Weatherman",
      month: ["januar", "februar", "marts", "april", "maj", "juni", "juli"],

      temperature: [
        {
          year: "2018",
          averageTemperature: [5, 4, 3, 11, 14, 21, 23],
        },

        {
          year: "2019",
          averageTemperature: [2, 5, 5, 12, 13, 22, 24],
        },

        {
          year: "2020",
          averageTemperature: [12, 8, 2, 14, 17, 29, 22],
```

```
        },
      ],
      iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot", "hot"],
    };

    this.container = document.createElement("div");
    this.container.id = "container";
    this.container.className = "container";
    document.body.appendChild(this.container);

    this.container.innerHTML = "" +
      this.weather.month.map((item, index) => {
          return `

            <div id="childCon">
                 <div class = '${"maps" + index}' id='block'></div>
                   <div class = '${"temp" + index}' id="temperature" >${this.wea
ther.temperature[0].averageTemperature[index]
                 }</div>
                 <div class="month">${this.weather.month[index]}</div>
             </div>


                `; //END return
        })
        .join("") +
      "";
  } // END constructor
}
```

Husk at tjekke din console for eventuelle fejl 😊

GSAP Animationer

GreenSock Animation Platform (GSAP) er et populært sæt JavaScript-værktøjer til opbygning af animationer på nettet. Alt hvad du ser i din webbrowser kan animeres med GSAP.

https://greensock.com/gsap/

Som det første skal GSAP importeres ind i vores projekt. Som udgangspunkt skal det også hentes som en npm pakke, men som du kan se i package.json er det allerede gjort, så vi skal bare importere den ind i vores initialize klasse:

```
import { gsap } from "gsap";
```

Vi skal have visualiseret vores data og vi vil bruge div-> block til at gøre dette. Det vil sige helt overordnet vil vi med hjælp fra GSAP animere vores data.

Det vil sige at vi skal have fat i alle div -> block elementer og loope igennem dem, og give dem data input fra this.weather objektet. Det kommer til at ske igennem en funktion. Vi så kort på et funktions kald tidligere og vi så at vi placerede en funktion i klassen, altså uden for konstruktøren, da den i sig selv er en funktion:

```
calculate() {
console.log('calculate');
}
```

Vi kan animarer meget simpelt:

```
gsap.to(document.querySelector("#childCon"), {
        duration: 1,
        scaleY: 2,
        transformOrigin: "bottom",
        ease: "elastic.out(1, 0.3)"
```

Her animerer vi et element så den skalerer på Y aksen og vi sikrer, at det sker fra bunden og op med transformOrigin og vi smider en easing på, så vi i dette tilfælde får en elastisk animation over et sekund(duration)

Vi har dog mange elementer, så vi looper igennem dem og i dette tilfælde, vil vi animerer alle div ->block elementer, som har fået et unikt class navn:

```
<div class = '${"maps" + index}' id='block'></div>
```

Hvilket gør at vi kan loope igennem hver enkelt af div -> block elementerne. Så lad os bygge vores funktion færdig:

```
calculate(averagetemp) {

   let averagetempLength = averagetemp.length;

   for (let i = 0; i < averagetempLength; i++) {
     gsap.to(document.querySelectorAll("#childCon > .maps" + i), {
       duration: 1,
       scaleY: (averagetemp[i] / 10).toFixed(2),
       transformOrigin: "bottom",
       ease: "elastic.out(1, 0.3)"

     });

   }
 }
```

Vi tager et enkelt argument med over, så vi ved hvor mange gange vi skal loope

Loop igennem alle div > -block elementer og tilgå dem med:

```
document.querySelectorAll("#childCon > .maps" + i
```

med en henvisning til alle indlejerede i div -> childCon, som har et class navn: ”maps0”, ”maps1”… osv

Så er det bare at kalde funktionen i konstruktøren med et argument, så vi kan tælle hvor mange gange vi skal loope igennem:

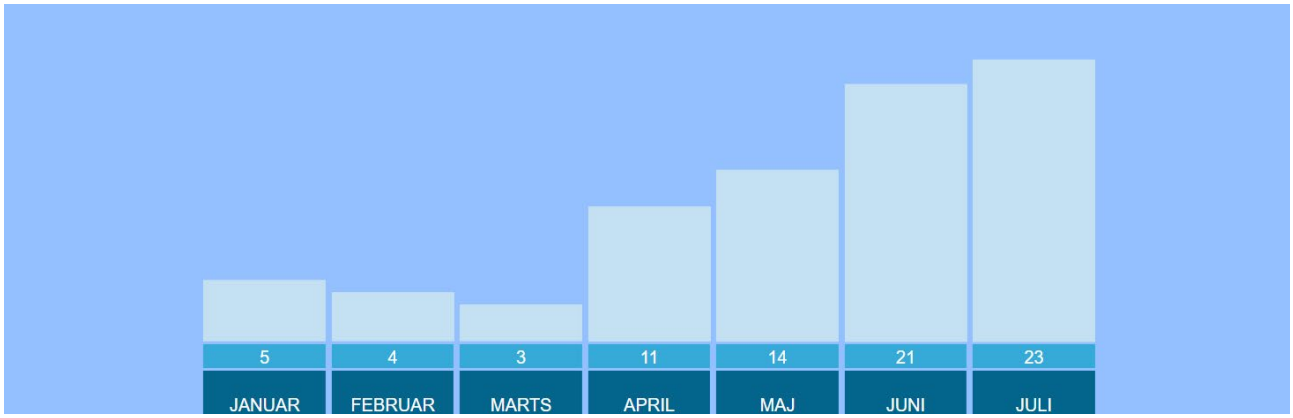
```
this.calculate(this.weather.temperature[0].averageTemperature);
```

Så nu skulle din kode gerne se sådan her ud:

```
import "@babel/polyfill";
import "../css/style.scss";
import { gsap } from "gsap";

export default class Initialize {
  constructor() {

    this.weather = {
      author: "The Weatherman",
      month: ["januar", "februar", "marts", "april", "maj", "juni", "juli"],
      temperature: [
        {
          year: "2018",
          averageTemperature: [5, 4, 3, 11, 14, 21, 23]
        },
        {
            year: "2019",
          averageTemperature: [2, 5, 5, 12, 13, 22, 24]
        },
        {
            year: "2020",
          averageTemperature: [12, 8, 2, 14, 17, 29, 22]
        },
      ],
      iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot", "hot"]
    };
    this.container = document.createElement("div");
    this.container.id = "container";
    this.container.className = "container";
    document.body.appendChild(this.container);

    this.container.innerHTML = "" +
```

```
      this.weather.month.map((item, index) => {

          return `

            <div id="childCon">
                 <div class = '${"maps" + index}' id='block'></div>
                 <div class="temperature">${this.weather.temperature[0].averageT
emperature[index]}</div>
                 <div class="month">${this.weather.month[index]}</div>
             </div>

                `; //END return
        })
        .join("") + "";

    this.calculate(this.weather.temperature[0].averageTemperature);
  } // END constructor

  calculate(averagetemp) {

    let averagetempLength = averagetemp.length;

    for (let i = 0; i < averagetempLength; i++) {
      gsap.to(document.querySelectorAll("#childCon > .maps" + i), {
        duration: 1,
        scaleY: (averagetemp[i] / 10).toFixed(2),
        transformOrigin: "bottom",
        ease: "elastic.out(1, 0.3)"

      });

    }
  }
}
```

Så rent visuelt skulle det hele nu gerne se sådan her ud og du skulle gerne få en animation på alle dine div -> block elementer når siden starter op:



Conditional Statements (betingelser):

Betingede udsagn bruges til at udføre forskellige handlinger baseret på forskellige input og forhold.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else

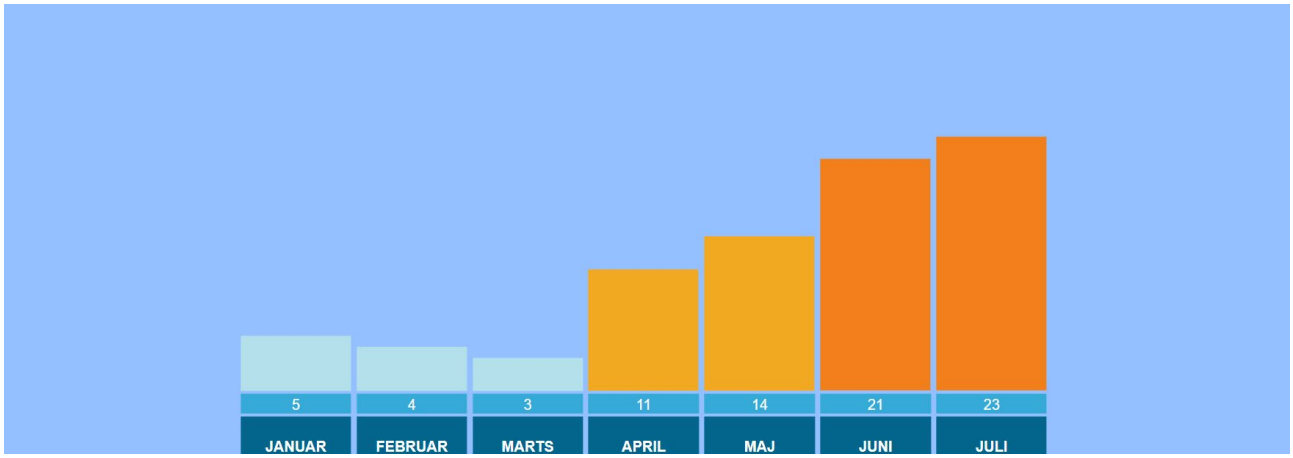
- Hvis temperaturen er større end 10 men mindre end 20, sættes farven på div -> block til en farve
- Hvis temperaturen er større end 20 sættes div -> block til en anden farve.
- Ellers skal div -> block have en default farve.

Så opdaterer vi calculate funktionen således:

```
calculate(averagetemp) {
  let averagetempLength = averagetemp.length;

  for (let i = 0; i < averagetempLength; i++) {
    gsap.to(document.querySelectorAll("#childCon > .maps" + i), {
      duration: 1,
      scaleY: (averagetemp[i] / 10).toFixed(2),
      transformOrigin: "bottom",
      ease: "elastic.out(1, 0.3)",
    });
```

```
  if (averagetemp[i] > 10 && averagetemp[i] < 20) {
      document.querySelector("#childCon > .maps" + i).style.backgroundColor =
        "#F2A922";
    } else if (averagetemp[i] > 20) {
      document.querySelector("#childCon > .maps" + i).style.backgroundColor =
        "#F27F1B";
    }else{
      document.querySelector("#childCon > .maps" + i).style.backgroundColor =
      "#b4e0eb";
    }
  }
}
```

Det skulle gerne vise følgende (StepThree.js):



Opgave 6: Ryd op i calculate funktionen, så der kun bruges en tilgang til div -> block klassen.

Videre med animationer

For at gøre det lidt ekstra lækkert kunne vi placere en animation i baggrunden af vores temperaturvisning. VI vil i denne omgang bruge en css animation:

https://css-tricks.com/almanac/properties/a/animation/

Først om fremmest skal der oprettes en div. Vi bruger igen Javascript til det, når vi nu er i gang:

```
    this.sun = document.createElement("div");
    this.sun.id = "sun";
    this.sun.className = "sun";
    document.body.appendChild(this.sun);
    this.sun.innerHTML = `<img src  = ${mSun} />`;
```

Der er en reference til grafikken der skal bruges: **${mSun}**
Det er en reference der kommer fra en import:

```
import mSun from "../assets/images/sun-big-white.png";
```

Det er hvad der skal til. Resten styrer vi fra CSS.

Der er en del der skal til i vores css for at få det til at køre:

I body skal vi sikre os at animationen ikke går ud over scroll området:

```
body {
  background-color: $body-color;
  min-height: 100vh;
  overflow: hidden;
  margin: 0;
}
```

Så positionerer vi vores div -> sun: Se sun klassen i style.scss, hvor også animationen er implementeret.
Og til sidst opretter vi animationen:

```
.sun {
  animation: rotation 30s infinite linear;
}

@keyframes rotation {
  from {
```

```
    transform: rotate(0deg);
  }
  to {
    transform: rotate(359deg);
  }
}
```

Vi opretter en animation til vores sun klasse navngiver rotation, så opretter vi keyframes rotation med en transform med en from to. Vi har desuden sikret os at den kører altid i form af infinite.

Vi kunne også have brugt GSAP her.

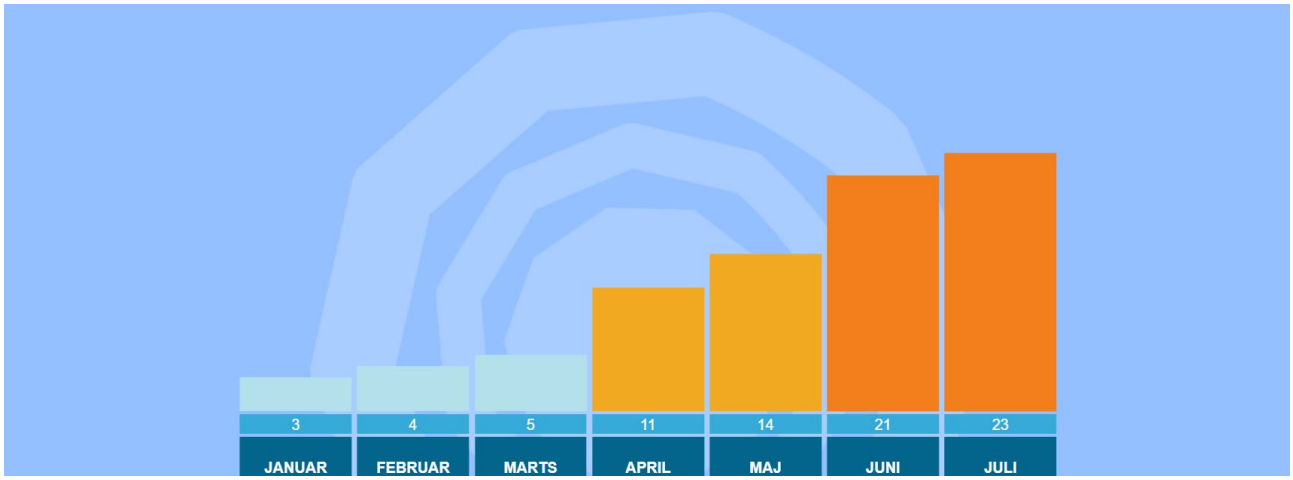
```
gsap.to("#sun", {
      duration:30,
       rotation: 360,
       transformOrigin: "center",
repeat:-1
    });
```

Som ville give det samme resultat.
En anden animations model kunne være at vi bruger web animations api direkte via Javascript:

https://developer.mozilla.org/en-US/docs/Web/API/Web_Animations_API/Using_the_Web_Animations_API

```
let cubeRotating = [
      {transform: 'rotate(0deg)'},
      {transform: 'rotate(360deg)'}

    ]

    let cubeTiming = {
      duration: 30000,
      iterations: Infinity
    }

    document.querySelector('#sun').animate(
      cubeRotating,
      cubeTiming
  )
```

Med GSAP, som jeg har valgt at bruge i dette tilfælde, skulle det hele gerne se sådan her ud:



Flere klasser

Inden vi går videre med projektet, skulle vi måske få ryddet lidt op 😊 Indtil videre har vi kun en brugt en klasse og et enkelt funktionskald. Der begynder sig at tegne et manglende overblik over indholdet.

https://www.geeksforgeeks.org/es6-classes/

Vi kunne sagtens flytte calculate funktionen ind i sin egen klasse og så importerer den i initialize klasse.

Opret en fil i src mappen og kald den calculate.js, og opret en klasse:

```
export default class Calculate {

 constructor() {

  console.log("Calculate");
 }

}
```

Lad os teste klassen ved at importere den i initialize.js og initialisere den. Bare lige for at sikre at den kører.

```
import "@babel/polyfill";
import "../css/style.scss";
import { gsap } from "gsap";
import mSun from "../assets/images/sun-big-white.png";
```

```
import Caulculate from "./Calculate"

export default class Initialize {
  constructor() {

      this.cal = new Caulculate();

        //… rest of the code
   }
}
```

Hvis du ser en console udskrivning ”Calculate” er alt godt.

Vi bruger new operatøren til at oprette en forekomst af et brugerdefineret objekt. I dette tilfælde er det Calculate. Så vi kalder vores importerede klasse.

Så nu kan vi prøve at tage calculate funktionen i initialize klassen og flytte den til Calculate klassens konstruktør, med nogle få ændringer.
Vi skal bruge GSAP så den importer vi ind i Calculate klassen.
Desuden rydder vi lige lidt op i kaldet til objektet vi vil animere. Ved at lave en henvisning via en let variabel, så vi ikke skal kalde objektet hver gang vi bruger det.

```
import {gsap} from 'gsap';

export default class Calculate {
  constructor( averagetemp ) {
      let averagetempLength = averagetemp.length;

    for (let i = 0; i < averagetempLength; i++) {

      let obj = document.querySelector('.maps' + i);

/*** START:  GSAP ***/
      gsap.to(obj, {
        duration: 1,
        scaleY: (averagetemp[i] / 10).toFixed(2),
        transformOrigin: 'bottom',
        ease: 'elastic.out(1, 0.3)',
      });
/*** END:  GSAP ***/

/*** START:  condition ***/
      if (averagetemp[i] > 10 && averagetemp[i] < 20) {
```

```
        obj.style.backgroundColor = '#F2A922';
      } else if (averagetemp[i] > 20) {
        obj.style.backgroundColor = '#F27F1B';
      } else {
        obj.style.backgroundColor = '#b4e0eb';
        }
/*** END:  condition ***/

    } //END loop

  }// END constructor

} // End class
```

Og her er kaldet fra initialize klassen

```
this.cal = new Caulculate( this.weather.temperature[0].averageTemperatu
re );
```

Vi kunne også rydde op i vores weather objekt, eller vi kunne flytte vores weather objekt ud i en selvstændig json fil. Det kræver en del ændringer i det kald vi gør til weather objektet.

Der ligger allerede en temperature.json fil i src mappen. Den skal importeres ind i Initialize klassen:

```
import data from "./temperature.json";
```

I din Initialize konstruktør kan du prøve at lave en console.log på data. Vi skal nu erstatte de data vi får fra this.weather objektet til at hente dem fra temperature.json data objektet.

Det er helt ok at hente data på den måde, når vi snakker om lokal data, hvis vi eksempelvis skulle hente data fra en ekstern kilde (et api), f.eks. DMI er det ikke nok at importer data, så er vi nødt til at sikrer os at de data vi skal bruge, er klar. I kommer til at arbejde med API kald senere på uddannelsen: Men et lille hint: Fetch API, som gør brug af Promises læs mere her: https://www.javascripttutorial.net/javascript-fetch-api/

Der er flere steder vi skal skifte ud:

```
this.container.innerHTML = "" +
      this.weather.month.map((item, index) => {

          return `
```

```
            <div id="childCon">
                 <div class = '${"maps" + index}' id='block' data-
index=${index} ></div>
                 <div class="temperature" id="temperature" >${this.weat
her.temperature[0].averageTemperature[index]}</div>
                 <div class="month">${this.weather.month[index]}</div>
             </div>

                `; //END return
        })
        .join("") + "";
```

Vi opretter en:

```
this.data = data;
```

Og så er det bare at skift alle this.weather til this.data og så skulle det gerne virke som før. Vores importeret klasse Caulculate sender også et argument, som bruger this.weather. Ret også dette til. Så kan du slette dit this.weather objekt.

Så skulle alt gerne se sådan her ud (StepFour.js):

```
import '@babel/polyfill';
import '../css/style.scss';
import {gsap} from 'gsap';
import data from './temperature.json';
import mSun from '../assets/images/sun-big-white.png';
import Caulculate from './Calculate';

export default class Initialize {

  constructor() {

    this.mInterval;
    this.data = data;

    this.sun = document.createElement('div');
```

```
    this.sun.id = 'sun';
    this.sun.className = 'sun';
    document.body.appendChild(this.sun);
    this.sun.innerHTML = `<img src  = ${mSun} />`;

    gsap.to('#sun', {
      duration: 30,
      rotation: 359,
      transformOrigin: 'center',
      repeat: -1,
    });

    this.container = document.createElement('div');
    this.container.id = 'container';
    this.container.className = 'container';
    document.body.appendChild(this.container);



   this.container.innerHTML ='' +
      this.data.month.map((item, index) => {

          return `

            <div id="childCon">
                 <div class = '${
                   'maps' + index
                 }' id='block' data-index=${index} ></div>
                 <div class="temperature" id="temperature" >${
                   this.data.temperature[0].averageTemperature[index]
                 }</div>
                 <div class="month">${this.data.month[index]}</div>
             </div>

                `; //END return
        })
        .join('') + '';

    this.cal = new Caulculate(this.data.temperature[0].averageTemperature);
  } // END constructor
}
```

Events(stepFive.js):

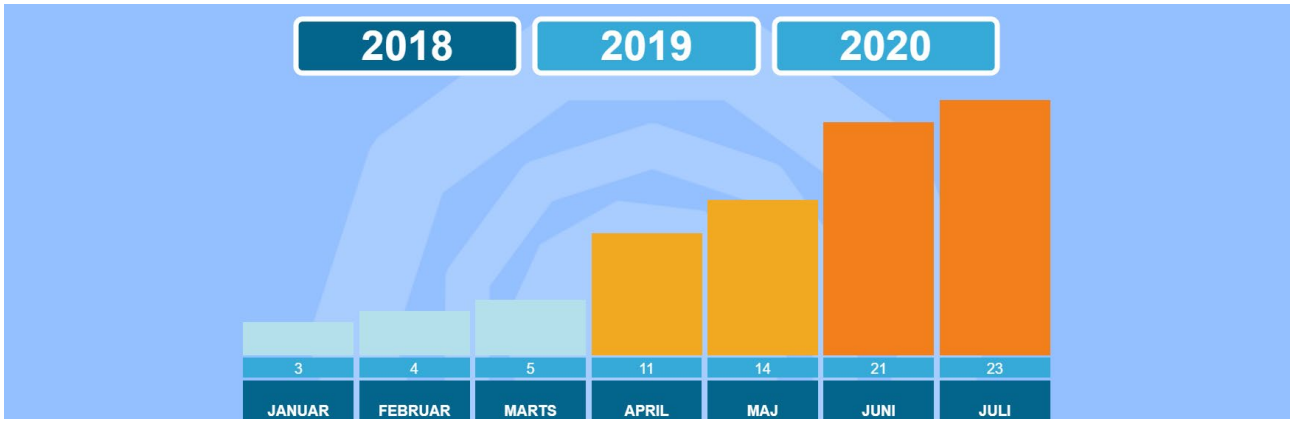
Det næste vi skal er at oprette en ”menu”, der giver mulighed for at navigere igennem årstal. Til dette vil vi bruge mouseEvents med addEventListener. Men først skal vi have menuen på plads.

Der skal oprettes en div -> yearContainer. Den kan evt. startes efter div -> container

Der skal i div -> yearContainer oprettes en child div -> year som nedstående. Vi bruger igen array.map til at loope igennem vores dataset (this.data). Css til div - yearContainer er allerede skrevet i style.scss filen, hvor der blandt andet bliver brugt css grid til at stille menuen op horisontalt. Læg mærke til data-index = ${index}, som vi kommer til at bruge i vores eventlistener lige om lidt. Den holder øje med hvor vi er i vores indeksering.

```
    this.yearContainer = document.createElement("div");
    this.yearContainer.id = "yearContainer";
    this.yearContainer.className = "yearContainer";
    document.body.appendChild(this.yearContainer);

    this.yearContainer.innerHTML = "" +
      this.data.temperature.map((item, index) => {

          return `
  <div class = '${"year" + index}' id='year' data-
index=${index} >${this.data.temperature[index].year}</div>

          `; //END return
        })
        .join("") + "";
```

Det skulle så gerne give følgende output(StepFive.js):



Når vi klikker på hver af menupunkterne, skulle vi så gerne hente data fra vores dataset (temperature.json) og vise den korrekte data.
Først skal vi have tilgang til alle vores menupunkter, så vi kan loope igennem dem:

```
this.getchildYearCon = document.querySelectorAll("#yearContainer > #year");
```

Så looper vi igennem med en forEach og den vej kan vi med en addEventListener få fat i alle vores menupunkter. Dernæst siger vi at når man klikker på et menupunkt skifter det baggrundsfarve og med for loopet nulstiller vi alle baggrundsfarver. Event.currentTarget henviser til det objekt der er i focus.

```
this.getchildYearCon.forEach((item) => {
      item.addEventListener("click", (event) => {

        for (let i = 0; i < this.getchildYearCon.length; i++) {

              this.getchildYearCon[i].style.backgroundColor = "#36aad8"

            }


            event.currentTarget.style.backgroundColor = "#04658c";



          });
        });
```

Så mangler vi kun en ting. Vi skal have hentet de rigtige data alt efter hvad man vælger i menuen. Vi har jo vores Calculate klasse, som henter fra det første dataset i vores json fil:

```
this.cal = new Caulculate(this.data.temperature[0].averageTemperature);
```

Så vi skal egentligt bare have vores data indeksering med i vores eventlistener.
og videre til vores Calculate kald.

event.CurrentTarget er referencen til det aktuelle objekt.

Prøv evt. at console.log currentDataIndex. Det vil give dig et meget godt billede af hvad der sker. Vi går igennem de enkelte objekter i data.temperature arrayet i temperature.json.

```
this.getchildYearCon.forEach((item) => {
      item.addEventListener("click", (event) => {


        for (let i = 0; i < this.getchildYearCon.length; i++) {
```

```
              this.getchildYearCon[i].style.backgroundColor = "#36aad8"

            }

let currentDataIndex = event.currentTarget.dataset.index;
let getDataFromIndex= this.data.temperature[currentDataIndex].averageTemperature;

            this.cal = new Caulculate( getDataFromIndex );

);



            event.currentTarget.style.backgroundColor = "#04658c";



          });
        });
```

Vi skal også have ændret på den numeriske visning af det gennemsnitlige gradtal. Til det opretter vi en funktion mere til at hente de data ind og loope igennem de tal.

```
getAverageTemp(currentMenuItemIndex){

    let getchildTempCon = document.querySelectorAll("#childCon > #temperatu
re");

            for(let i=0;i<getchildTempCon.length;i++){

               document.querySelector(".temp" + i).innerHTML=this.data.temp
erature[currentMenuItemIndex].averageTemperature[i];

             }

  }
```

For at få det rigtige index tal fra menuen opretter vi en variabel, som vi kan tilgå i hele klassen. Opret den i konstruktøren.

```
this.getCurrentTargetIndex =0;
```

I vores menu click eventhandler giver vi så this.getCurrentTargetIndex den index værdi der passer til det valgte:

```
this.getCurrentTargetIndex = event.currentTarget.dataset.index;
```

Desuden skal vi lave vores div -> temperature class om til:

```
'${"temp" + index}'
```

Så er den noget nemmere at få fat i, når vi skal opdatere vores numeriske grad værdier i getAverageTemp funktionen. Desuden skal vi slette det indhold som div -> temperature viser nu(markeret med rød), da de data nu bliver dynamiske:

```
 <div class='${"temp" + index}' id="temperature" >${this.data.temperature[0].averageTemperature[index]}</div>
```

Så kalder vi funktionen fra konstruktøren:

```
this.getAverageTemp(this.getCurrentTargetIndex);
```

Og det samme gør vi fra menuens eventlistener:

```
this.getAverageTemp(this.getCurrentTargetIndex);
```

Så skulle det virke 😊

Opgave.7 Igen bliver vores kode noget rodet og det ville være naturligt at vi skiller menuen fra.
Hvis du syntes det virker uoverskueligt kan du tage en kig på StepSix.js

Det sidste vi skal igennem med dette projekt er, at vi gerne, når vi klikker på en blok, vil vise den daglige temperatur(StepSeven.js). Det er lidt fake, da vores temperature.json kun har et overordnet array til dette (dailyDegrees), men det er helt op til jer selv at rette det til. Hvis I vil. I dette step viser jeg hvordan man kan, visuelt henter dem ind i forhold til de antal dage der er i en måned.

Der bliver brugt et div overlay så vi kan se de nye data og giver den noget transparent.

Start med at oprette 3 div elementer:

```
this.modalContainer = document.createElement("div");
this.modalContainer.id = "modalContainer";
this.modalContainer.className = "modalContainer";
document.body.appendChild(this.modalContainer);
```

```
 this.closeModal = document.createElement("div");
 this.closeModal.id = "closeModal";
 this.closeModal.className = "closeModal";
 document.body.appendChild(this.closeModal);
  this.closeModal.innerHTML = `<svg id="Lag_2" data-
name="Lag 2" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 67.53 67.5
3"><defs><style>.cls-1{fill:#fff;}</style></defs><rect class="cls-
1" x="40.31" y="7.94" width="20.52" height="86.33" rx="10" transform="t
ranslate(-38 33.26) rotate(-45)"/><rect class="cls-
1" x="40.05" y="8.2" width="20.52" height="86.33" rx="10" transform="tr
anslate(34.38 -38) rotate(45)"/></svg>`;


 this.dailyWeatherContainer = document.createElement("div");
 this.dailyWeatherContainer.id = "dailyWeatherContainer";
 this.dailyWeatherContainer.className = "dailyWeatherContainer";
 document.body.appendChild(this.dailyWeatherContainer);
```

I skal ikke lade jer skræmme af <svg> elementet innerHTML af div -> closeModal. Der bliver smidt en direkte svg -> XML kode ind. Det kan gøres på mange måder. Det kommer vi ind på senere, når vores overlay er gjort færdig.

De 3 div elementer:

- Div -> modalContainer: det direkte overlay
- Div -> closeModal: Skal indeholde luk knappen til overlay
- Div -> dailyWeatherContainer: indeholder de data vi vil vise i overlayet

De bliver alle tilføjet til document.body

Som det første skal vi have vores overlay implementeret:

Vi skal have oprettet en eventListener på hver af div->block, da den henter data, som passer til de enkelte blokke alt efter hvilken en div->block man vælger. Det har vi gjort før med menuen.

div->modalContainer og div-> closeModal er i style.scss sat til at være:

display:none
opacity:0

getTargetClassName er en variabel, som laver en indeksering af de enkelte div-block class navne, som vi senere bruger til hente data.

Vi bruger så GSAP til at få en fadein af menuen. Så det ser ud som følgende:

```
let getchildConChild = document.querySelectorAll("#childCon > #block");

   getchildConChild.forEach((item) => {
```

```
      item.addEventListener("click", (event) => {
      let getTargetClassName = parseInt(event.target.className.substr(4, 4
));
        document.querySelector("#modalContainer").style.display = "block";
        document.querySelector("#dailyWeatherContainer").style.display =
"block";
        gsap.to("#modalContainer", {
          duration: 1,
          opacity: 0.8,
          onComplete: () => {
            document.querySelector("#closeModal").style.display = "block";
            gsap.to("#closeModal", { duration: 1, opacity: 1 });
          },
        });
      })
    })
```

Vi skal også kunne lukke overlayet. Så vi skal have en eventlistener på luk knappen. Den kan vi placere lige under vores div->closeModal:

```
const close = document.querySelector("#closeModal");

    close.addEventListener("click", (event) => {

      this.self = event.currentTarget;

      gsap.to([this.self, this.modalContainer], {
        duration: 1,
        opacity: 0,
        onComplete: () => {
          this.self.style.display = "none";
```

```
          this.modalContainer.style.display = "none";
        },
      })
    }); // END addEventListener
```

Logikken i luk overlay er følgende:

Når man klikker på luk knappen (div -> closeModal) fader både overlay (div - > modalContainer) og luk knappen ud med **GSAP**, og når den er færdig med at fade ud, sætter den overlay og lukkeknap til: display:none.

Vi starter med at lave en reference this.self til vores lukknap, da den ellers kan være svær at få fat på i onComplete funktionen, med mindre man bruger en direkte reference med querySelector.

Det sidste vi skal have løst, er at hente de data for den givne div-> block, når man trykker på. Vi skal oprette to div containere til at holde på henholdsvis visuel visning og den direkte tal visning og de skal tilføjes til div -> dailyWeatherContainer og de bliver begge centreret på siden (se style.scss). De skal oprettes i den eventlListener der åbner overlayet. Disse to div vil blive slettet, når man lukker overlayet. Det ser vi på senere:

```
  this.modalDailyWeather = document.createElement("div");
  this.modalDailyWeather.id = "modalDailyWeather";
  this.modalDailyWeather.className = "modalDailyWeather";
  document.querySelector("#dailyWeatherContainer").appendChild(this.modalD
ailyWeather);


  this.modalDailyWeatherDegrees = document.createElement("div");
  this.modalDailyWeatherDegrees.id = "modalDailyWeatherDegrees";
  this.modalDailyWeatherDegrees.className = "modalDailyWeatherDegrees";
  document.querySelector("#dailyWeatherContainer").appendChild(this.modalD
ailyWeatherDegrees);
```

Så skal der oprettes et loop, der smider data ind i de to oprettede divs:

Først opretter vi 1 variabel i vores konstruktør:

this.dailyWeatherLoopEnd er en Boolean, som vi sætter til falsk(false), som default og som vi senere sætter til sand(true) når vores loop er færdig, da vi gerne vil vide hvornår det sker.

```
this.dailyWeatherLoopEnd = false;
```

Så kører vi et loop igennem.

div -> modalDailyWeather får umiddelbart ingen data ind, da det skal være visuelle data input, så her gør vi klar til at bruge GSAP til at lave en visualiserung af vores data.

div -> modalDailyWeatherDegrees får data fra vores data objekt.

Vi tjekker at vores loop er færdig og sætter this.dailyWeatherLoopeEnd =true gør klar til at visualisere vores data

```
    for (let ii = 0;ii < this.data.dayesInMonth[getTargetClassName];ii++) {

          this.modalDailyWeather.innerHTML +=

          `

       <div class = '${"daily" + ii}' style="width:25px;background-
color:#fff"></div>

          `;


          this.modalDailyWeatherDegrees.innerHTML +=

          `

          <div class = '${"daily" + ii}' style="width:25px;background-
color:#36aad8;margin-top:10px" >${this.data.dailyDegrees[ii]

            }</div>

          `;


          if (ii == this.data.dayesInMonth[getTargetClassName] - 1) {

             this.dailyWeatherLoopEnd = true;

          }

        }
```

Vi vil visualisere vores data med GSAP og vi opretter en setInterval metode til at køre igennem alle de div elementer der skal visualiseres. Vi tjekker om this.dailyWeatherLoopEnd = true, da vi genre vil sikre at alle div elementer er klar før vi begynder at tage fat i dem.

Der skal oprettes to variabler i konstruktøren:

this.counter bruges til at opløfte til +1, hvergang den bliver kaldt og som vi bruger til at få fat i de enkelte div elementer der berøres.

this.mInterval er vores interval metode reference, så vi kan stoppe vores interval metode, når vi ikke skal bruge den mere.

```
this.counter = -1;

this.mInterval;
```

Og så kan vi starte vores interval.

Vi bruger vores this.counter, som vi opløfter og bruger som reference til vores div elementer.

Vi bruger vores this.mInterval til at initialisere vores interval metode

VI tjekker om this.counter er lig med det antal vi referer til og hvis det er stopper vi vores interval med clearInterval metoden, sætter this.counter tilbage til udgangspunkt og this.dailyWeatherLoopEnd til falsk

```
if (this.dailyWeatherLoopEnd) {

          this.mInterval = setInterval(() => {

            this.counter++;

  gsap.to(document.querySelectorAll("#modalDailyWeather > .daily" + this.co
unter),{

                duration: 1,
                scaleY: this.data.dailyDegrees[this.counter],
                alpha: 1,
                transformOrigin: "bottom",
                ease: "elastic.out",

              }

            );

    if (this.counter ==this.data.dayesInMonth[getTargetClassName] - 1) {

              clearInterval(this.mInterval);

              this.counter = -1;

              this.dailyWeatherLoopEnd = false;

            }

          }, 50); //END interval

        } //END if
```

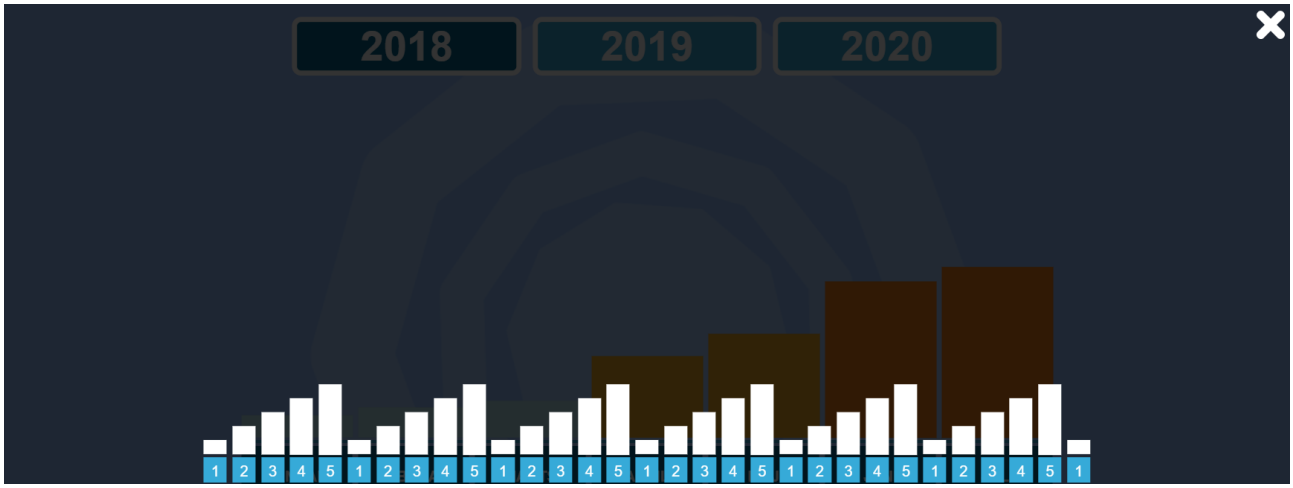
Når du prøver at lukke overlayet på luk knappen, kan du se at de data vi har hentet frem ikke forsvinder. Det skal vi lige have fikset. Det vil sige at vi nulstiller alle data når vi lukker overlayet.

Tilføj følgende til din eventListener for din luk knap:

```
const close = document.querySelector("#closeModal");


    close.addEventListener("click", (event) => {
      this.self = event.currentTarget;
      gsap.to([this.self, this.modalContainer], {
        duration: 1,
        opacity: 0,
        onComplete: () => {
          this.self.style.display = "none";
          this.modalContainer.style.display = "none";
        },
      })
    this.modalDailyWeather.remove();
    this.modalDailyWeatherDegrees.remove();
    clearInterval(this.mInterval);
    this.counter = -1;
    this.dailyWeatherLoopEnd = false;
    clearInterval(this.mInterval);


    });
```

Så skulle det gerne virke det hele. Husk at du tjekke op mod klassen StepSeven.js.

Sådan skulle det hele gerne se ud når du har åbnet et overlay:



SVG

Vi vender lige tilbage til den der svg luk knap. Der er flere måder at du kan hente en svg ind.

Her er nogle forskellige modeller, som kan bruges og de kan ses importeret i Initialize.js:

```
import svg from "../assets/images/close.svg";

import closeSVG from "./closeSVG";


 this.mclose=closeSVG();

  this.closeModal.innerHTML = `<img src = '${svg}' />`;

    this.closeModal.innerHTML = this.mclose;
```

De sidste ting vi har lavet med overlayet og de data vi hente ind fylder jo rigtigt meget, så det kunne være fedt at få ryddet lidt op.

Der er flere ting vi kunne gøre. f.eks.:

- Kunne vi lave overlayet i sin egen klasse, med den eventListener der åbner og lukker overlayet. hvilke problematikker løber vi ind i?
- De data der bliver vist i et overlay. Kunne de også have sin egen klasse og hvad kræver det?
- Hvad med data flowet i json filen. Hvis man skulle lave individuelle daglige grader. Hvad skulle man så gøre?
- Kunne man, i stedet for at angive det i json filen, finde antal dage der er i en given måned med javascript.

Mere om klasser og metoder

Som de kan ses i Menu.js kan du sagtens oprette/tilføje metoder til din klasse, men det er også helt ok at bruge metoder fra en klasse i en anden klasse

```
export default class KlasseMetode {
  constructor () {
    console.log ('constructor');
  }
  myFunction () {
    console.log ('myFunction fra KlasseMetode');
  }
}
```

Og så kan du kalde den på følgende måde:

```
import KlasseMetode from "./KlasseMetode"
 let km = new KlasseMetode();

 km.myFunction(); //kalder metode i klassen KlasseMetode
```

Du kan også returnere fra en klasse via en metode:

```
export default class KlasseMetode {


  constructor () {
    console.log ('constructor');
    this.name = "Lakrids";
  }


  myFunction () {
    console.log ('myFunction fra KlasseMetode');
  }


  myReturnFunction(){
```

```
        return this.name;
    }


}
```

Og så kalde den(når den er importeret og initialiseret):

```
console.log(km.myReturnFunction());
```

Get og Set

Du kan også oprette egenskaber fra din klasse via get metoden (Person.js):

https://coryrylan.com/blog/javascript-es6-class-syntax

Opret en Person klasse:

```
export default class Person{

    constructor(){

        this._name="Lakrids";

    }


    get name(){

        return  this._name;

    }

}
```

Og så kalde den (når den er importeret):

```
let person  = new Person();

console.log(person.name);
```

Du kan selvfølgelig også bruge et objekt til at samle dine egenskaber for en klasse.

```
export default class Person{

    constructor(){

        this._data={

        name:"Lakrids",
        age:12

        };


        this._name="Lakrids";

    }


    get name(){

        return  this._name;


    }


    get data(){

        return this._data;

    }

}
```

Og så kalde den:

```
  let person  = new Person();
  console.log(person.data);
  console.log(person.data.name);
  console.log(person.data.age);
```

Du kan også bruge en getter til at gøre en setter klar, som kan ændre dine egenskaber:

```
export default class Person {
  constructor() {
    this._data = {
      name: 'Lakrids',
      age: 12,
    };

    this._name = 'Lakrids';
    this._age = 12;
  }

  get name() {
    return this._name;
  }
  get data() {
    return this._data;
  }
  get age() {
  console.log('ready get');
   return this._age;
  }
  set age(newAge) {
    console.log('set triggered!')
```

```
    this._age = newAge;
  }
}
```

Og så ændre din egenskab:

```
 let person  = new Person();
   console.log(person.age = 13);
```

Du kan også sætte det op sådan her, hvis du sender et argument med over til din PersonNext klasse (PersonNext.js):

```
Export default class PersonNext {
  constructor(name) {
     this._name = name;
  }
  get name() {
    return this._name;
  }
  set name(newName) {
    this._name = newName;
  }
}
```

Så kalder du:

```
let personnext = new PersonNext('Vermont');
document.body.innerHTML = personnext.name;
console.log(personnext.name);
```

Nedarvning (Inheritance):

```
import Person from "./Person"

export default class Inh extends Person {

  constructor() {
     super();

     console.log(this._name);//variable from Person
     console.log(this.age);//getter from Person
      console.log(this.age = 13); //setter from Person

    console.log(this.tastes());
  }

  tastes() {
    return `${this.name} smager skøøøøønt!`;
}

}
```

https://coryrylan.com/blog/javascript-es6-class-syntax

Vi kunne også vælge at nedarve vores klasse Person ind i en overordnet klasse og så bruge de egenskaber Person har i en anden klasse. Det kaldes nedarving (ihn.js).

Super-nøgleordet henviser til den overordnet klasse. Den bruges til at kalde konstruktøren til den overordnede klasse og få adgang til den nedarvede klasses egenskaber og metoder.

Lyd med Howler.js (stepEight.js)

https://howlerjs.com/

Vi skal se lidt på hvordan der kan arbejde med lyd i en webproduktion. Som udgangspunkt ser vi ikke brug af lyd på standard websider, men mange store medier, bruger interaktive moduler på deres websites, hvis der skal fortælles en historie eller for at underbygge en stemning i en artikel. Som udgangspunkt ville man bruge WebGL/canvas til dette, da der findes nogle rigtig gode JavaScript frameworks, der er specielt udviklet til at håndtere lyd og animation/spil.

Fede sider med masser af lyd 😊:
https://www.awwwards.com/websites/webgl/#google_vignette

Vi vil se lidt på et bibliotek der hedder howler.js. Howler i vores miljø er allerede installeret, så vi skal faktisk bare importere biblioteket og gå i gang.
I er velkommen til at bygge jeres egen eksempel, men jeg viser hvordan det kan bruges i den produktion vi allerede har lavet. Det virker ikke særlig logisk at bruge lyd i vejr projektet, men for forståelsens skyld vil vi lægge lidt lyd på når vi åbner en overlay.

Importer Howler.js:

```
import { Howl, Howler } from 'howler';
```

Howler har sin egen initialisering med en del egenskaber og indstillinger:

```
  let sound = new Howl({
          src: ['./assets/sound/open.mp3'],
           autoplay: true,
           loop: false,
           volume: 0.5,
            onend: () => {

                    //When sound end, something could happend

                     }
                });
```

I sin reneste form:

```
let sound = new Howl({

   src: ['./assets/sound/open.mp3']

    });

 sound.play();
```

Som udgangspunkt kan du bruge HTMLAudioElements, som er en standard. Fordelen ved at bruge Howler.js er at den sørger for meget, blandt andet tjekker den om webAudio kan bruges i den aktuelle browser, samtidig får du en masse ting foræret i Howler.js.

HTMLAudioElement-grænsefladen giver adgang til egenskaberne for <audio> elementet, som er HTML5 standard, samt metoder til at manipulere den.

```
var audio = new Audio("./assets/sound/open.mp3");

        document.onclick = function() {

          audio.play();

        }
```

https://developer.mozilla.org/en-US/docs/Web/API/HTMLAudioElement#Examples

https://www.w3docs.com/learn-html/html-audio-tag.html

Opgave 8: Hver gang man trykker på en div->block skal der afspilles en lyd (open.mp3). Desuden skal der afspilles en lyd(daily.mp3) når den daglige temperatur "ruller" ind i det tilhørende overlay. Lidt voldsomt, men sjovt 😊

Snippets:

Eventlistener med notelist:

https://developer.mozilla.org/en-US/docs/Web/API/NodeList

- Oprettelse af nodelist(getchildElements)
- Loop med forEach
- eventListener

HTML:

```
<div id=#mainContainer>
<div id="child"></div>
<div id="child"></div>
<div id="child"></div>
<div id="child"></div>
</div>
```

ES6:

```
let getchildElements = document.querySelectorAll("#mainContainer > #child");
```

```
    getchildElements .forEach((item) => {
      item.addEventListener("click", (event) => {

            console.log(item); // data fra nodelist(getchildElements)
            console.log(event.CurrentTarget); // objektet der bliver klikket på

            //eksempel på brug af aktuelle objekt
             event.CurrentTarget.style.backgroundColor = "#cccccc";
        })

})
```

Single eventlistener:

https://developer.mozilla.org/en-US/docs/Web/API/EventListener

HTML:

```
<div id="close">click me</div>
```

ES6:

```
 const close = document.querySelector("#close");
  close.eventListener("click", (event) => {
      this.self = event.currentTarget; // objektet der bliver klikket på
})
```

Brug af data:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects

Oprettelse af objekt med data:

```
 this.weather = {
        author: "The Weathermann",
        month: ["januar", "februar", "marts", "april", "maj", "juni" , "juli"],
        averageTemperature: [-10, -10, -10, 12, 15, 20, 25],
        iconType: ["cold", "cold", "cold", "notThatCold", "nice", "hot"]
    }
```

Oprettelse af html->div element med createElement:

```
    this.container = document.createElement('div');
    this.container.id = "container";
    this.container.className = "container";
    document.body.appendChild(this.container);
```

Loop igennem data med array.map() og returnere data med templates litteral og expressions:

```
    this.container.innerHTML = '' + this.weather.month.map((item, index) => {

         return `
              <div class = '${"maps" + index}' id='block'>${item}</div>
             `

     }).join('') + '';
```

-

Brug af data-* atributten til at oprette et dataset og hente data ud fra array index

https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

```
        this.yearContainer = document.createElement('div');
        this.yearContainer.id = 'yearContainer';
        this.yearContainer.className = 'yearContainer';
        document.body.appendChild(this.yearContainer);


        this.yearContainer.innerHTML = this.weather.month.map((item, index) => {

                return `
<div id='year' data-index=${index} >click me</div>

                `; //END return
            })
                .join('');


        this.getchildYearCon = document.querySelectorAll('#yearContainer >
#year');

        this.getchildYearCon.forEach((item) => {
            item.addEventListener('click', (event) => {

             this.getCurrentTargetIndex = event.currentTarget.dataset.index;


            });
        });
```

--

Tilgå dataset fra css

https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

```css
#year[data-index='1'] {
  width: 400px;
}

#year[data-index='3'] {
  width: 600px;
}
```