Programmering og udvikling af små systemer samt databaser

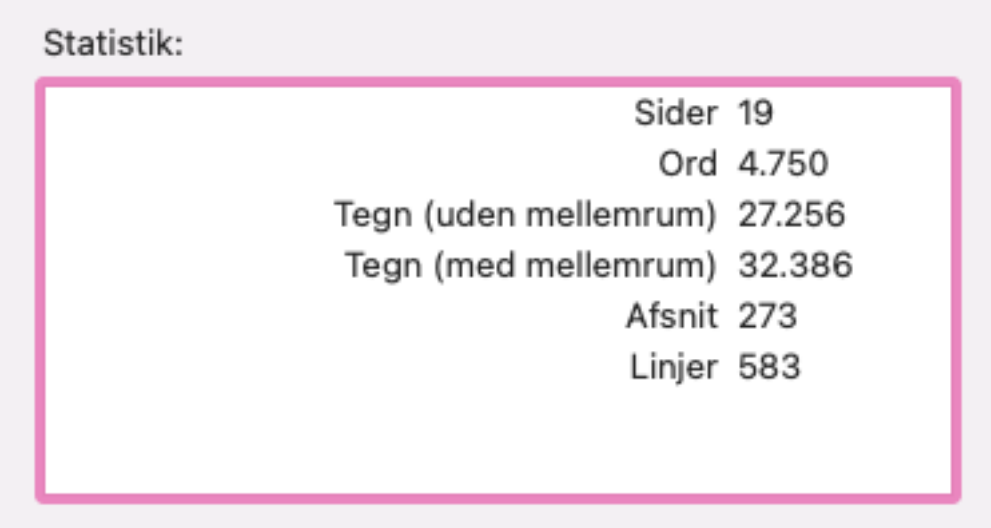
Skriftligt produkt

HA (it.) - 1. semester

Afleveres d. 12. december 2022 senest kl. 12:00

Eksamensnummer: S162183

Antal anslag og sider:



Indholdsfortegnelse

Kravsspecifikationer	
1. login	
Ja	a) App'en skal tillade brugeren at logge ind
Ja	b) App'en skal gøre det muligt for en bruger at logge ud
2. Profil	
Ja	a) App'en skal tillade en bruger at oprette en profil
Nej	b) App'en skal tillade en bruger at slette sin egen profil
N**ej**	c) App'en skal tillade en bruger at opdatere sin egen profil
N**ej**	d) App'en skal tillade at en bruger kan sætte sine yndlingsnyhedskategorier
N**ej**	e) App'en skal tillade at en bruger kan tilføje en eller flere nyhedsartikler til favoritter.
N**ej**	f) App'en skal tracke hvilke nyhedsartikler en bruger har læst. Dette gemmes ved hjælp af localstorage.
3. Nyheder	
Ja	a) App'en skal vise en liste af de seneste nyheder til brugeren.
Ja	b) App'en skal kunne vise en enkelt nyhed med links og billede.
N**ej**	c) App'en skal gøre det muligt for en bruger at søge på mindst to forskellige parametre for nyheder. F.eks. Dato, tid, sprog, land, osv… (få inspiration her).
4. Vejret	
Ja	a) App'en skal vise den nuværende tid og dato.
Ja	b) App'en skal hente og vise vejrudsigten for de næste 7 dage.
Ja	c) App'en skal vise den nuværende temperatur.
Ja	d) App'en skal vise solopgang og solnedgangs tidspunkt.
Ja	e) App'en skal bruge vejrdata for lokationen København.

Indledning

I denne rapport beskrives implementerings- samt udviklingsforløbet for en nyhedsapplikation. Målet med denne applikation er en udvikling af en client-facing front-end applikation i Node.js. Nyhedsapplikation skal have en funktionalitet, hvor det skal være muligt at oprette en profil med tilhørende metadata. Derudover skal det være muligt at læse nyheder, der jævnligt bliver opdateret gennem et API. I forlængelse med dette, skal det være muligt at gemme, hvilke nyhedsartikler, der bliver læst, gennem localStorage. Endvidere, er et af kravene at brugeren skal blive opdateret med information om vejret, gennem et vejr-API.

For at kunne udføre disse krav, udvikles klienten samt Storage. Klienten udvikles gennem JavaScript, HTML samt CSS. Storage er en dataløsning, som kan gemme en brugers information i en JSON-fil.

Opgavebesvarelsen afspejles i applikationens, samtlige 10 ud af 16 krav. Grundet tidspres, har det ikke været muligt at leve op til alle krav eller styling. - I rapporten uddybes der, hvilke løsninger der er etableret for at imødekomme kravspecifikationerne for eksamensopgaven. For at imødekomme kravspecifikationerne bedst, er der prøvet at blive taget højde for god kodeskik. - Det vil sige, at der er taget et bevidst valg om at tilknytte kommentarer i koden, for at hjælpe læser med en bedre grundforståelse for koden. Derudover har løsningsovervejelserne til dels ikke levet op til en god kode skik. - I opgaven har jeg derfor søgt på nettet, for at finde et bedre alternativ løsning samt kodeskik til min løsningsovervejelse

Programmerings- samt udviklings værktøjer

Applikationen er udviklet i Node,js, ud fra et objektorienteret system, der følger MVC-modellen. MVC-modellen er en tilgang, hvor man opdeler en applikation i tre dele, herunder: Model, View og Controllers (Sep 21, 2022, by MDN contributors). Endvidere har jeg brugt *Visual Studio code*, som kildekode-editor, til at implementere applikationen.

Login

Krav 1: Appen skal tillade brugeren af logge ind

På hjemmesiden, skal brugeren have mulighed for at logge ind, her har jeg brugt filerne logIn.HTML samt signUp.js.

For at nå dette krav, er det nødvendigt at have en fil, for at kunne tjekke om vedkommende er registreret. Måden jeg har løst dette på, er ved at gemme brugeren i *localStorage*. Når en bruger forsøger at logge ind, kontrolleres der, om de indtastede oplysninger er identiske med et bruger-objekt i *localStorage*. Hvis dette er tilfældet, vil brugeren blive ført hen til index.html, som består af min nyhedsside. Koden består af, at der oprettes en funktion ”checkLogin”, der inkluderer, *e.preventDefault()*, som er en indbygget funktion, der gør at siden ikke genindlæser, eller sletter indholdet på siden. Herefter definerer jeg to *variabler* ”a” og ”b”, som tager fat i HTML-element ”email4” samt ”password4*”* værdi, disse elementer er id’erne på hver input-boks.

I forlængelse med dette, opretter jeg nye variabler, som inkluderer, ”user”, der indeholder *localStorgae.getItem*-metoden. Dette betyder, at serveren går ind og tager en allerede eksisterende *item*. For at JavaScript skal kunne forstå en JSON-fil, opretter jeg en ny *variable* ”data”, og sætter den lig med *JSON.parse* metoden, som ændrer et JSON-objekt til en JavaScript *value* (Nov 10, 2022, by MDN contributors). Endvidere oprettes der et *if-else-statement*, for at se om brugerens oplysninger, stemmer overens med *localStorage*. Dette *if-else-statement* indebærer; når funktionen kører, kontrollerer den om ”user” findes, hvis ”user” ikke findes, så console.log (”Wrong email”). Hvis den findes, kører funktionen videre, her kontrollerer den om data.email og datapassword er *true*. Hvis ja, vil den returnere til index.html.

```
function checkLogin(){
  event.preventDefault();

  var a = document.getElementById("email4").value;
  var b = document.getElementById("password4").value;

  var user = localStorage.getItem(a);
  var data = JSON.parse(user);

  if(user==null){
    console.log("Wrong email")
  } else if(a == data.email && b == data.password){
    window.location.href = "http://127.0.0.1:5500/index.html"
  } else {
    console.log("Wrong password")
  }
```

```
  }
```

Krav 1: Appen skal gøre det muligt for en bruger at logge ud

Kravsspecifikationerne om at appen skal gøre det muligt for en bruger at logge ud, er opfyldt. Dette er opfyldt i index.html samt index.js, idet log ud knappen, er knyttet til index.html. For at kunne få dette krav opfyldt, har jeg lavet en knap i HTML. - Denne knap er inkluderet i funktionen *logUdbutton*, når brugeren trykker på knappen, sætter den funktionen i gang. I forlængelse med dette bruger jeg localStorage.removeItem(email), som gør, at når jeg trykker på knappen *logUdButton*, går den ind i localStorage og fjerner key'et e-mail, og herved returnerer til index.html, hvilket er metoden *window.location.href*.

```
function logUdButton(){
    let logUdButton = document.getElementById("logUdButton");
    localStorage.removeItem(email);
    window.location.href = "http://127.0.0.1:5500/login/login.html"
};
```

Profil

Krav 2: Appen skal tillade brugeren at oprette en profil

På hjemmesiden, skal brugeren have mulighed for at oprette en bruger. Jeg startede med at oprette 4 inputfelter, samt 1 knap, som skal forestille sig at være en submit knap. Disse input, har fået et hver deres id, så jeg kan benytte mig af dem i mit JavaScript. I mit JavaScript oprettes en funktion resultSignup (), som indeholder min nøgle og resten af indholdet. Denne funktion inkluderer, *event.preventDefault(),* som er en metode en definition af navnet på en variabel, som er lig med mit HTML elements værdi. Efterfulgt af dette, opretter jeg et objekt, som inkluderer id'et på HTML input elements værdi, lig med de dannede variabler (Samad Kanton, 2021). Efterfølgende defineres en variabel *JSON*, som er lig med:

```
function resultSignup() {
  event.preventDefault();
  var fname = document.getElementById("firstName1").value;
  var lname = document.getElementById("lastName1").value;
  var email = document.getElementById("email1").value;
  var pw = document.getElementById("password1").value;
```

```
let json = JSON.stringify(userData);
```

Dette element er nødvendigt, for at dataet kan blive gemt i localStorage. Endvidere, konverterer *Stringify* en JavaScript fil til en JSON-fil (MDN, contributers OCT. 30, 2022).

Et alternativ for *opretButton*, ville være, at når man trykkede på knappen, ville de retunere til login siden, så brugeren ville have mulighed for at logge ind. Her ville jeg benytte mig af denne kode, som ikke inkluderer i den endelige kode, idet det ikke fungerer helt optimalt.

```
function opretButton () {
  let opretButton = document.getElementById("opretBrugerButton");
  window.location.href = "http://127.0.0.1:5500/login/login.html"
}
```

Krav 2: Appen skal tillade brugeren at slette sin egen profil

Kravet om at brugeren skal have mulighed for at slette sin egen profil, er ikke opnået grundet tidspres. Et løsningsforslag til dette krav er at oprette en funktion samt en knap, som har en funktionalitet med *eventListner,* som indebærer, at når man trykker på en knap, hvor der står "slet profil", vil den køre funktionen med metoden localStorage.removeItem(), og slette brugerens *values*. Dog vil de oprindelige *keys* stadig være der, idet de fremgår i den oprindelige singUp.js.

Krav 2: Appen skal tillade brugeren at opdatere sin egen profil

Appen skal tillade brugeren at opdatere sin egen profil. Dette krav er ikke opfyldt grundet tidspres. En alternativ løsning ville være at bruge PATCH-metoden, er en metode der kan ændre eksisterende objekter (Sep 15, 2022, by MDN contributors). Her oprette en funktion ”ændreOplysninger”, med en knap*(id=”*knap1*”),* med en tilhørende *eventListener*. Denne knaps funktionalitet er, at når man trykker på knappen, fører den brugeren hen til en html-side med deres oplysninger. Når brugeren kommer ind på hjemmesiden, vil brugeren se 3 inputfelter, med placeholdere, ”ForNavn”, ”Email”, ”password”. Når brugeren skriver i inputfelterne, og trykker enter;

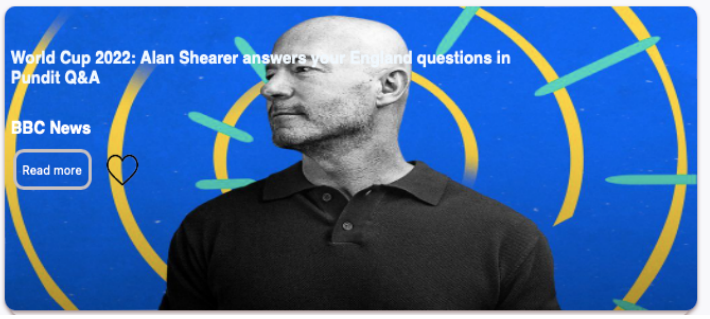
```
if (e.key === "Enter")
```

vil inputs felternes *value*, komme ind i *localStorage*.

Krav 2: Appen skal tillade en bruger at sætte sine nyhedskategorier

Kravet om at appen skal tillade en bruger at sætte sine nyhedskategorier, er ikke opfyldt. En løsningovervejelse vil være; bruge API’et, der fetcher nyhederne, her ville jeg oprette en funktion, der inkluderer et array, med en række nyhedskategorier, som brugeren kan indsatte. Når dette array er blevet *pushet,* vil disse nyhedskategorier blive gemt i *localStroage*, med et *key* ”Nyhedskategorier” samt et *value*, som vil indeholde det data, som brugeren kan vælge i mellem, ved at på at skrive sin nyhedskategori et inputfelt, og derefter trykke på en tilhørende knap ”Tilføj nyhedskategori”. Når brugeren trykker på denne knap, sender den en sætter funktionen i gang, og sætter det indtastede datas *value* ind i *localStorage*.

Krav 2: Appen skal tillade at en bruger kan tilføje en eller flere nyhedsartikler til favoritter.



Applikationen skal tillade, at en bruger kan tilføje en eller flere nyhedsartikler til favoritter. Der defineres en variabel ”tomthjerte”, som tager fat i HTML-elementet. Der tilføjes en *click-eventlistner* til dette HTML-element, som lytter til, at når der trykkes på hjertet, kalder den funktionen, som ændrer *img-elementet ”*tomHjerte.innerHTML”, til ”fyldtHjerte.png”. Når billedet af hjertet ændrer sig til et fyldt hjerte,

vil den gemme ”latestNews”, som er hele nyhedsboksen i localStorage, og derved er en nyhedsartikel gemt i localStorage.

JavaScript’en vil se således ud;

```
var favA = titleLatestNews.value;
  var tomtHjerte = document.getElementById("latestNews");
  tomtHjerte.addEventListener("click", like);

  function like(){
    tomtHjerte.innerHTML = "fyldthjerte.png";
    localStorage.setItem("favoritArtikler", favA);
  }
```

Ligeledes ser HTML’en således ud;

```
  <div id="latestNews"> <img id="tomtHjerte" src="tomthjerte.png" alt="tomtHjerte">
</div>
```

Krav 2: App’en skal tracke hvilke nyhedsartikler en bruger har læst. Dette gemmes ved hjælp af localstorage.

Applikationen skal kunne tracke, hvilke nyhedsartikler en bruger har læst fra brugeren. Dette krav er ikke opfyldt, men en alternativ løsning til dette krav, ville være at bruge den knappen ”ReadMore”, som fremgår på nyhedsartiklerne i index.html. Denne knap, ville få en *click-eventlistner,* således at når der trykkes på knappen, ændrer den dels knappens skrift fra ”Read” til ”Already read”. I forlængelse med dette, vil den gemme artiklens indhold i localStorage, ved at gøre brug af metoden *localStorage.setItem*.

Nyheder

Krav 3: Appen skal vise en liste af de seneste nyheder til brugeren.

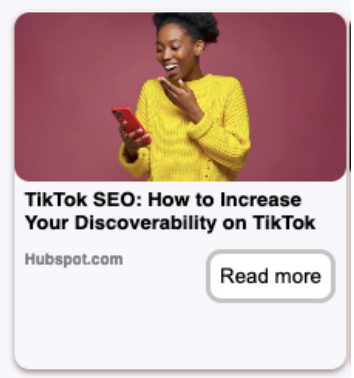
Det skal være muligt for brugeren at se en liste af de seneste nyheder på appen. For at opnå dette krav gjorde jeg mig en række overvejelser, som ikke fungerede på længere sigt. Den dybdegående forklaring på, hvorfor jeg valgte at oprette mine div i JavaScript, ses i afsnittet *Procesevaluering*. For at vise de seneste nyheder, som blev specificeret i CSS clinet-facing frontend, har jeg hentet nyhederne ned eksternt. Et API er et sæt af definitioner og protokoller, der interagerer med applikationssoftware. I denne opgave har jeg benyttet mig af et Representational State Transfer API, idet det er den mest anvendte, når man bruger JSON som et dataudviklingsformat. For at anvende dette API, kræver det også et API key, hvilket fungerer ligesom en adgangskode. Denne API key bruger man sammen med sit API, når man skal lave en forespørgsel. Det er meget vigtigt, at dette API-key bliver opbevaret et sikkert sted, da det ellers er let tilgængeligt for andre. Jeg fandt ingen løsning på at opbevare mit API et sikkert sted, hvorfor mit API er offentligt tilgængeligt i filen index.js. For at få mine nyheder frem, benyttede jeg mig af Fetch metoden. Denne metode er en JavaScript brugeroverflade, der gør det muligt at få adgang til dele protokoller ((MDN contributors Sep 9, 2022). Fetch sætter ligeledes en proces i gang, der henter en ressource fra et API-url, og til sidst returnerer et *promise*.

Jeg har brugt await metoden til at fetche, det gør at den starter at give en anmodning på mit API. Med dette, kan jeg nu anvende det returnerede data fra json-filen (Dmitri Pavlutin, Feb. 18, 2021).

```
const result = async () => {
  const småBox = document.getElementById('småBox');
  småBox.innerHTML = " ";
  let                    result                    =                    await
fetch('https://newsapi.org/v2/everything?q=us&apiKey=b57f6dc7b72c46d097591637d89ae19f')
  . then((r) => {return r.json()});
```

Min udarbejdelse af mine 6 små nyhedsbokse samt store nyhedsboks, er udarbejdet i JavaScript. De små nyhedsbokse, som fremgår i index.html, er lavet i et *for loop*, her looper det igennem 6 forskellige nyheder. Dette så jeg som en bedre kodeskik, idet det fylder mindre. Hvis de 6 små bokse er i et *for loop*, vil min kode være i større mængder, samt uoverskuelige at holde styr på. For at loop'ets ikke skal lave 6 nye bokse, oven i de eksisterende bokse, definerer jeg, at små bokse skal være i en tom *string*, hvilket gør at den starter forfra med at lave 6 bokse, når den kører funktionen *result.*

Krav 3: Appen skal vise en enkelt nyhed med links og billede



Ud over at appen skal vise de seneste nyheder, skal den også vise en enkelt nyhed med links og billede. I mit tilfælde, vises et *img*, *tittle*, *source* samt et *link* til nyheden, som er konstrueret gennem JavaScript, og formateret gennem CSS. For at få billedet frem, starter jeg med at bruge *createElement* metoden, til at lave et *img* element. Dette element inkluderer *src*, som er et strengobjekt der indeholder URL'en fra API'ets JSON. Dette sætter jeg lig med min konstant *result*, og refererer til *articles* og *urlToImage*, som er det der fremgår i JSON-filen. Til sidst appender jeg *photo6* til *senestNyt6*, som er id'et på boksens div. Denne metode gør, at billedet bliver indsat direkte i boksen, uden at tilpasse boksen.

```
let photo6 = document.createElement('img')
photo6.classList.add("photo6");
photo6.src = result.articles[index].urlToImage;
senestNyt6.append(photo6);
```

Et alternativ for at kunne formindske mængden af min CSS-styling ville være, at jeg i stedet for *src*, udskifter dette med;

```
photo6.style.backgroundImage = result.articles[index].urlToImage;
```

Dette ville gøre at billedet ville komme ind som et baggrundsbillede, således at dens bredte formaterer sig i forhold til boksens størrelse, og derfor kun skal justere højde samt border-radius i CSS.

Endvidere, har jeg lavet en *knap*, som har funktionaliteten, at når man trykker på knappen, vil den føre hen til den reelle artikel. Den kode jeg har skrevet, giver en funktionalitet for knappen;

```
readmore.action = result.articles[index].url; //lægger jeg linket i form
readmore.append(readMore6Button); /
author6.append(readmore);
```

Denne funktionalitet inkluderer, at når brugeren trykker på knappen, sendes anmodning til en *action,* som er en *property*, der sættes fra URL'en, som sendes videre i en *proces*, der sender brugeren hen til at oprindelige artikel.

Krav 3: Appen skal gøre det muligt for en bruger at søge på mindst to forskellige parametre for nyheder f.eks. Dato, tid, sprog, land

Appen skal gøre det muligt for en bruger at søge på mindst to forskellige parametre for nyheder. Dette krav er ikke blevet opfyldt i opgaven, da funktionen ikke fungerede. For at opfylde en lille del af dette krav, oprettes et inputfelt, med det tilhørende id *sFunktion,* for at kan designe den, så den kunne ligne

prototypen nogenlunde. Løsningsovervejelsen for dette krav inkluderer at denne funktion, vil starte en proces, hvor den kalder på et HTML-element, som er id'et på inputfeltet. I forlængelse med dette kunne man tilføje en *keypress eventlistner,* der detekter, når der trykkes på tastaturet. *Eventlistneren*, vil kun være gyldig, hvis det er tasten Enter, der bliver trykket på. Hvis udsagnet er korrekt, vil den tage en variabel, som indeholder inputtets værdi fra søgefeltet. Denne variabel, vil så blive indsat i API-linket, og ændre indholdet på nyhederne.

```
function searchFunction() {
    let sfunktion = document.getElementById("sfunktion ");
     searchBar.addEventListener("keypress", (e) => {
      if (e.key === "Enter") {
          let sfunktionValue = sfunktion.value;
          result = `https://newsapi.org/v2/everything?q=${sfunktionValue}&apiKey=b57f6dc7b72c46d097591637d89ae19f` }
    });
  }
  searchFunction();
```

Vejret

Krav 4: Appen skal vise nuværende tid og dato

På hjemmesiden skal brugeren have mulighed for at kunne se den nuværende tid og dato. For at nå dette krav, gjorde jeg mig en række overvejelser inden. Disse overvejelser inkluderede at der skal være en række tal, som skulle kunne opdatere sig hvert sekund. Jeg har benyttet en indbygget *contructor* funktion kaldet New Date, hvor jeg har gjort brug af metoderne; *getDate* og,. getSeconds etc., som logger den nuværende dag og tid (Nov 28, 2022, by MDN contributors). Jeg har defineret "clock" til at være lig med mit *span-tag* "clock" fra HTML. Efterfølgende skrives clock.textContent, for at få mine metoder ind på min HTML side. Dertil har jeg benyttet metoden slice (-2) til mine konstanter, således at de sidste to cifre i talværdien beholdes (Making live clock in JavaSript, 2016). ligeledes, tilføjet *en string* "0" til "hours", "minutes" *og "*seconds" variablerne. Så der står et 0 foran, hvis der kun fremgår et ciffer.

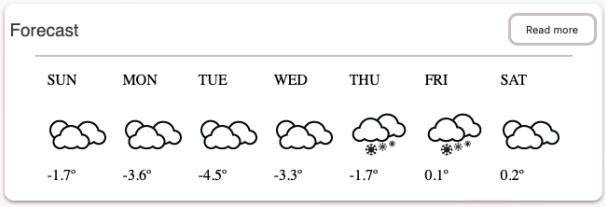
Se koden neden for;

```
let clock = document.getElementById('clock');

function time() {
  const date = new Date();
  const seconds = date.getSeconds();
  const minutes = date.getMinutes();
  const hours = date.getHours();
    clock.textContent =
      ("0" + hours).slice(-2) + ":" + ("0" + minutes).slice(-2) + ":" + ("0" + seconds).slice(-2);}
    setInterval(time, 1000)};
const today= new Date();
document.getElementById('date').innerHTML=today;
```

For at få datoen frem, har jeg igen brugt den indbyggede *constructor* funktion *New date*.- Definerer en konstant "day" til at være lig den indbyggede *constructor* funktion *new Date ().* Derefter tager det *div* elementet fra HTML lig konstanten, så det fremgår på siden.

Krav 4: Appen skal hente og vise vejrudsigten for de næste 7 dage.



På hjemmesiden skal appen hente vejrudsigten for de næste 7 dage. Jeg gjorde mig en række overvejelser inden, som inkluderede at jeg skulle lave et array, som ville indeholde ugens 7 dages navne. Endvidere, kunne jeg se en sammenhæng i, hvordan jeg *fetchede* min nyheder med, hvordan jeg ville *fetche* mit vejr-API:

```
const resultWeather = async () => {
let resultWeather = await fetch('https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/copenhagen?unitGroup=metric&key=V4UY9V2QSSU3B57FRSD9QPRW6&contentType=json')
    .then((r) => {return r.json()});

  console.log(resultWeather);
```

Derfor har jeg brugt den samme metode, hvor jeg har benyttet mig af et Representational State Transfer API, idet det er den mest anvendte, når man bruger JSON som et dataudviklingsformat. Efter vejr-API'et er *fetchet*, fremgår det i min konsol. I forlængelse med dette, bruger jeg *createElement* metoden til at oprette den store div til hele boksen, dette giver jeg en *class*: "tempAndDays1". Dertil opretter jeg et *for-loop*, idet jeg gerne vil have 7 -dage, temperaturer samt ikoner frem. I dette *for-loop* henter jeg mit data fra JSON-filen en ad gangen, og i hver deres *div*, samt hver deres *class*, med et navn:

```
tempraturWeek.textContent = resultWeather.days[index]. temp + "°";
```

Efter div'et *temperaturweek,* er blevet oprettet, henter jeg dataet ned på applikationen, ved at bruge *.textcontent*, hvilket betyder, at det returnerer hvert element i HTML-filen (Sep 9, 2022, by MDN contributors). I dette eksempel, har jeg derudover, tilføjet en *streng* med celsius tegnet til temperaturen, idet det ikke fremgår i den oprindelige JSON-fil.

For at få de korrekte dage ned, og få dem til at opdatere jævnligt, forhold til hvilken dag det er, har jeg dannet en konstant *daysArray,* som er et array, der indeholder elementer af 14 dage;

```
const daysArray = ['SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN','MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT'];
```

Jeg har taget en bevidst valg om, at arrayet, skal starte i "SUN", idet det betragtes som værende *index [0].* Endvidere er det bevidst, at det er 14 dage, idet det ellers ville arrayet printe "undefined*"* værdi på dagene, indtil den ville ramme *index[0]* igen.

Til ikonerne, har jeg oprettet en variable *iconDiv*, som bliver sat lig et *img* element, der har *className* "iconDiv", idet jeg skal style ikonet. Efterfulgt at dette, opretter jeg endnu en variable *myIcon*, som er lig mappen, med alle ikoner fra visualCrossing's hjemmeside, plus JSON-filens array *days*, som har et property *icons*, plus den tilhørende url. "Icondiv" *sættes til src,* herefter bliver det sat lig "myIcon"*,* som er alle ikonerne. Denne metode fungerer på en måde, hvor at ikonet ændrer sig efter API'et oplysninger, inden vejr-ikonet når ud til brugeren.

```
let iconDiv = document.createElement("img");
iconDiv.classList.add("iconDiv");
let myIcon = "icons/" + resultWeather.days[index]. icon + ".png";
iconDiv.src = myIcon;
```

Krav 4: Appen skal vise den nuværende temperatur.

Den nuværende temperatur, er blevet hentet ned fra vejr-API'et. Den benyttede metode, er den samme metode som den ovenstående. Dog har jeg sat arrayet til at starte *index [0],* så man giver den første temperatur i arrayet, hvilket svarer til i dag.

```
        let tempratur = document.createElement('div');
        tempratur.classList.add("tempratur");
        tempratur.textContent = resultWeather.days[0].temp + "º";
        venstre.append(tempratur);
```

Krav 4: Appen skal vise solopgang og solnedgangstidspunkt.

Ligesom det første krav, hentede jeg solopgang- samt solnedgang tidspunkt ned fra vejr-API'et fra visualcrossing til de respektive div'er, ved at bruge *append* metoden. Denne metode tilføjer div'et med API-dataet, til vejr-boksens div. Dette gør at API-dataet vil befinde sig i det div, der appendes fra. I forlængelse med dette, kalder jeg på ikonets funktion.

```
        let sunriseTime = document.createElement('div');
        sunriseTime.classList.add("sunriseTime")
        sunriseTime.textContent = resultWeather.days[0].sunrise.slice(0,5
        center.append(sunriseTime);
        feather.replace()
```

Krav 4: Appen skal bruge vejrdata for lokationen København.

Igen, har jeg brugt den samme metode, hvor jeg henter lokationen ned fra min vejr-API, ligeledes giver jeg den en klasse med id'et *location.* Det der adskiller denne metode fra de andre metoder er, at i JSON-filen er den oprindelige *key,* "ResolvedAddress", denne *key,* har den tilsvarende *value "*København, Hovedstaden, Danmark*"*. I og med at prototypen illustrer, at ordet 'hovedstaden' ikke er inkluderet i front-enden. Derfor har jeg brugt en *replace-metode*, som erstatter det jeg skriver i min *string*, som i dette tilfælde er "*København, Hovedstaden," "* " (en tom *string*). Til sidst benytter jeg mig af append metoden igen, hvor jeg appender respektive div til vejr-boksens div. Se koden nedenfor:

```
    let location = document.createElement('div');
    location.classList.add("location");
    location.textContent = resultWeather.address +
", "+resultWeather.resolvedAddress.replace ("København, Hovedstaden","," ");
```

```
venstre.append(location);
```

Data oversigt

I dette afsnit, vil der komme en uddybende forklaring på, hvordan jeg i denne opgave har valgt at strukturere mit data. For at kunne leve op til kravspecifikationerne, inkluderer denne opgave, at JSON filens objekter bliver gemt i localStorage.

Storage & JSON

Som nævnt i det ovenstående afsnit, har jeg i opgaven brugt localStorage, som fungerer således, at den opbevarer *key/values* i en browser. Key, er et hvilket som helst navn, man ønsker, skal stå i localStorage, hvor *values* er inputtet. I relation til opgavens kravspecifikationer, har jeg gemt brugerens oplysninger i localStorage. Når brugeren opretter en bruger, dannes der et *objekt* med forskellige *properties*, som er de input værdier, brugeren har givet. Dette ender med at bliver struktureret således i JSON;

```
▼0: {firstName: "Lars", lastName: "Larsen", email: "larsLarsen@gmail.com"
    email: "larsLarsen@gmail.com"
    firstName: "Lars"
    lastName: "Larsen"
    password: "ingenGætterMinKode"
```

Dette gør det muligt for brugeren at logge ind, efter de har oprettet en bruger i *signUp.html.* En vigtig pointe om *localStorage* er, at dataet ikke bliver slettet fra *localStorage*, med mindre brugeren bevidst går ind og gør det, eller at udvikleren gør det, gennem en kode, som;

```
localStorage.removeItem('key')
```

Løsnings overvejelser

Under processen, har jeg haft en del løsningsovervejelser, disse løsningsovervejelser kan læses under hver kravspecifikation.

Procesevaluering

Jeg vil i dette afsnit, uddybe denne eksamensopgaves proces. Ved udviklingen af denne nyhedsapplikation, er der blevet gennemgået løsningsovervejelser, der dels har ført til en løsning for

kravspecifikationerne, men også til, at nogle af kravspecifikationerne er blevet løst metodisk. Gennem hele processen, har jeg prøvet at opnå så mange krav som muligt, grundet tidspres, har det ikke været muligt at nå alle krav. I det næste afsnit, vil jeg forklare de mest essentielle problemer, jeg er stødt på gennem udviklingen af nyhedsapplikationen.

Front-end udvikling

I starten af udviklingsprocessen, blev der lagt markant fokus på clinet-facing frontend, altså HTML og CSS. Grunden til, at det var det, jeg fokuserede mest på, var dels fordi jeg følte det mere overskueligt og nemt, og dels for at få et visuelt billede af, hvordan min side ville se ud. Jeg fandt dog ud af, at det ville være mere praktisk at danne et fundament for JavaScript'en først.

Class, id & div

Gennem hele opgaven, har jeg dels benyttet mig af *class* samt *id*, for at klassificere en *div*. I denne opgave, har det været nødvendigt at inddele siden i flere *div'er*, idet kodningen mindskes i JavaScript samt CSS. I begyndelsen af processen, oprettede jeg alle *div'er* i HTML. Da jeg efterfølgende havde hentet mit newsAPI ned, kunne man hurtigt miste overblikket, i forhold til hvilke *div'er*, der skulle hentes data ned i. Endvidere, indså jeg også, at det ikke ville fungere, hvis jeg ville have de 6 små nyheder til at køre i et *for-loop.* Derfor valgte jeg at lave div'er i HTML, når det gav mening, for eksempel, har jeg inddelt siden i en højre og venstre div.

Mapper/struktur

Gennem hele opgavens proces, har jeg ikke haft i mente, at jeg ville inddele mine filer i mindre filer eller, samle filerne i forskellige mapper, for at gøre det lettere for læseren eller den næste udvikler. Dette er først blevet taget i betragtning på inden afleveringsfristen, hvilket har medført, at det har været sværere at linke mapper og filer i for eksempel index.html. I min næste udviklingsproces, vil dette være det første jeg opretter under *interface*, for at gøre det lettere at få et overblik over, hvilke filer man har.

Kodeskik

I mine koder, har jeg en tendens til at lave variabler på både engelsk og dansk, hvilket er dårlig kodeskik - det kan forvirre læseren/næste udvikler. Derfor vil jeg i min næste opgave overveje navnene grundigt, i forhold til hvilke variable, konstanter etc. samt holde mig til et sprog.

Konklusion

Det kan konkluderes at denne rapport beskriver implementerings- samt udviklingsforløbet for nyhedsapplikationen. Applikationen er blevet implementeret ved en udførelse af 10 ud af 16 kravsspecifikationer, som nævnt tidligere i rapporten, er disse krav hovedsageligt ikke opnået, grundet et tidspres, hvorfor knapperne på hjemmeskærmen ikke er stylet. I rapporten bliver der derfor beskrevet løsningsovervejelser til de ikke implementerede kravsspecifikationer. I forlængelse med dette, nævner rapporten proces evaluering, hvor der nævnes, hvilke problemer, der er blevet stødt på, under udviklingen af nyhedsapplikationen.

Endvidere bliver der fremvist forskellige metoder, herunder hvordan programmet bliver testet løbende i konsollen, ved brug af funktionen *console.log ()*. Dette er en metode der er blevet benyttet meget i denne opgave, idet det giver en forståelse for, om funktionen tager imod requestes, og hvor den ikke gør. Ligeledes beskriver rapporten, hvordan data'et fra nyhedsapplikation, herunder nyhedsdata samt brugerdata, bliver gemt eller slettet i localStorage, når man angiver *key* samt *value*. Afslutningsvis, er der ligeledes blevet redegjort for, enkelte kodestykker, som ses relevante for at få en forståelse af, hvorfor nyhedsartiklerne fremgår så præsentabelt, nyhedsapplikationen som helhed.

Litteraturliste

MDN contributors. (Sep 9, 2022) Using the Fetch API (*located November 21, 2022*)
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

MDN contributors (Sep 9, 2022) Node.textContent (*located November 21, 2022*)
https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent

MDN contributors (Nov 10, 2022) JSON.parse (*located december 7 2022*)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

MDN contributors (Sep 15, 2022) PATCH (*located december 10 2022*)
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH

MDN contributors (Sep 21, 2022) MVC (*located december 11 2022*)
https://developer.mozilla.org/en-US/docs/Glossary/MVC

NewsAPI (*located December 11, 2022*)
https://newsapi.org

Stackoverflow (2016) Making live clock in JavaScript (*located November 25, 2022*)
https://stackoverflow.com/questions/39418405/making-a-live-clock-in-javascript

Visual Crossing - Weather Query Builder (*located November 27, 2022*)
https://www.visualcrossing.com/weather/weather-data-services/København?v=api

Samad Kanton (2021) SignUp Form with LocalStorage (*located dec 7 2022*)
https://www.youtube.com/watch?v=oX7ko6M7YDc

Font Awsome (*located dec 7 2022*)
https://fontawesome.com/start

MDN contributors (Nov 28, 2022) Standard build-in objects (*located November 28, 2022*)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects