Eksamensopgave efterår 2023

Hjemmeopgave i Programmering og udvikling af små systemet samt databaser

Erhvervsøkonomi & informationsteknologi – Copenhagen Business school

Aflevering den 21/12/2023 kl. 12:00

Vejleder: Ole Torp Lassen

Studie nr: 168795

Sider	21
Ord	3.459
Tegn (uden mellemrum)	21.050
Tegn (med mellemrum)	24.362
Afsnit	216
Linjer	456

Indholdsfortegnelse

Kravspecifikationer		
Meal Creator		
a)	Det skal være muligt for en bruger at oprette et eller flere måltider og væsker som kan bestå af 1 eller flere ingredienser (fødevarer).	Opfyldt
b)	For hvert måltid eller væske, skal der udregnes samlet ernærings indhold med minimum følgende parametre (Energi, Protein, Fedt og Fiber) baseret på alle de fødevarer som bliver brugt i retten.	Opfyldt
c)	Skal vise hvor mange ingredienser som er brugt i opskriften	Opfyldt
Meal Tracker		
a)	Det skal være muligt i App'en at registre ens næringsindtag som vægt af en given opskrift, som er skabt i meal creator, samt hvornår man har indtaget dette.	Opfyldt
b)	Det skal være muligt i App'en at registre ens indtag af væske i løbet af dagen, samt tidspunkt for indtaget.	Opfyldt
c)	Der skal være en visning af ens indtaget registreringer som inkluderer hvornår det er indtaget, vægten og Ernæringsindholdet.	Opfyldt
d)	Det skal være muligt at redigere ens registreringer.	Opfyldt
e)	Det skal være muligt at slette ens registreringer.	Opfyldt
Nutrition Dashboard		
a)	Skal vise hvor mange måltider som den aktive bruger har indtaget i løbet af dagen.	Opfyldt
b)	Det skal indeholde en summering af indtaget af energi i Kcal ved måltiderne i løbet af dagen.	Opfyldt
c)	Det skal indeholde en summering af væskeindtaget i løbet af dagen.	Opfyldt
d)	Det skal indeholde en summering af totalt indtag af protein i løbet af dagen.	Opfyldt
Nutrition Report		
a)	Skal der være et view som viser en detaljeret liste over samlet indtag af væske og antal måltider i løbet af de sidste min. 7 dage. Denne liste skal også indeholde en 4 mere detaljeret liste over indtaget næring (energi, Protein, Fedt og fibre).	Opfyldt
Food Item Inspector		
a)	Et sted i appen skal det være muligt at finde information omkring hver enkelt fødevare i datasættet samt dets ernæringsindhold.	Opfyldt
Data		
a)	Det skal være muligt at komme til at se et detaljeret overblik over hver fødevarer, som er brugt i en opskrift.	Opfyldt
b)	Alt data til brug i applikationen skal tages fra Api'et.	Opfyldt

c) Data som hentes i appen, skal hentes med ens studienummer som API-key.	Opfyldt

Indledning

Eksamensprojektet i dette år handler, om at skabe en interaktiv platform. Denne platform skal muliggøre, at brugeren let kan sammensætte måltider, overvåge deres ernæringsindtag, spore kalorier og samtidig opnå en omfattende oversigt over deres ugentlige indtag

I denne rapport, vil i blive præsenteret overfor en trinvis gennemgang, af de funktionelle aspekter, samt de krav, som er blevet opgivet til at udvikle webapplikationen ”NutriTracker”. Derudover inkluderer rapporten en procesevaluering, der reflekterer over udviklingsprocessen og de færdigheder, der er blevet forbedret gennem projektet. Endelig udforsker rapporten perspektiverne for fremtidig udvidelse og forbedring af NutriTracker og konkluderer med en opsummering af projektets resultater og bidrag.

Med udviklingen af NutriTracker har jeg skabt en platform, der ikke kun opfylder de oprindelige krav, men også har potentiale til at udvikle sig yderligere som et pålideligt værktøj til brugere, der ønsker at styre deres kost og ernæring. Projektet repræsenterer ikke blot teknisk dygtighed, men også en forståelse af, hvordan man skaber struktureret samt funktionel kode i større webapplikationer.

Layout

Før jeg begyndte at udvikle mit JavaScript, begyndte jeg at udvikle den overordnet struktur samt layout for platformen NutriTracker. Dette gjorde jeg via HTML samt CSS. For at skabe layoutet til min hjemmeside anvendte jeg designforslaget, der blev angivet i opgaven, som udgangspunkt. Jeg udformede fire forskellige HTML filer, samt en CSS-fil. De fire HTML filer, er til for at skabe god struktur til koden.
Til de forskellige ikoner/knapper, har jeg benyttet mig af ”google fonts[1]”. Linket til disse ikoner er refereret til i starten af mine forskellige HTML filer.

[1] https://fonts.google.com/

Analyse

Meal Creator

1.a

For at gøre det muligt at oprette et eller flere måltider samt væsker, lavede jeg en statisk [2] asynkron-funktion, som blev kaldt for fecthFoodItem[3]. Denne funktion bruges til at hente fødevaredata fra den givne API (dette forklares yderligere i *Data* afsnittet). Den udfører en GET-anmodning til URL'en "*https://nutrimonapi.azurewebsites.net/index.html*".

I anmodningen er der specificeret headers, herunder Accept og X-API-Key.

```
static async fetchFoodItem() {
  try {
    const response = await fetch('https://nutrimonapi.azurewebsites.net/api/FoodItems', {
      method: 'GET',
      headers: {
        'Accept': 'text/plain',
        'X-API-Key': 168795
      }
    });

    if (!response.ok) {
      throw new Error('Network response was not ok');
    }

    const data = await response.text();
    localStorage.setItem("fooditems", data);

  } catch (error) {
    console.error('Error during fetch:', error);
  }
}
}
```

Figur 1

Efter at have modtaget responsen tjekker funktionen, om anmodningen var vellykket. Hvis responsen er vellykket, konverteres kroppen af responsen til tekst ved hjælp af response.text(). Den resulterende tekst gemmes i localStorage under nøglen "fooditems" ved hjælp af localStorage.setItem. Hvis der opstår en fejl under fetch-anmodningen eller behandlingen af responsen, logges en fejlmeddelelse til konsollen.

Derudover har jeg indført en form af "sikkerhedsnet," ved at tilføje try-catch[4], hvilket betyder, at hvis noget går galt, under hentning af data fra serveren, kan man nemt finde fejlen, samt håndtere den mere effektivt.

[2] https://www.w3schools.com/js/js_class_static.asp
[3] Øvelse uge 46
[4] https://www.w3schools.com/js/js_errors.asp

1.b

Derefter har jeg oprettet en asynkron funktion for at løse dette krav. Den asynkrone funktion ”addFood”, tilføjer en fødevare til måltidet ved at oprette en ny instans af Food og hente ernæringsoplysninger fra en API ved hjælp af getfoodData-funktionen getfoodData()-funktionen kaldes gentagne gange for at indhente og beregne ernæringsmæssige data for den valgte fødevare, idet der anvendes forskellige sorteringsnøgler som "1030" for energi, "1110" for protein osv.

```
f = new Food(fid,fname,fqty,0,0,0,0);
await getfoodData(fid, "1030", "energy",fqty, f);
await getfoodData(fid, "1110", "protein",fqty, f);
await getfoodData(fid, "1310", "fat",fqty, f);
await getfoodData(fid, "1240", "fiber",fqty, f);
```

Figur 2

Disse beregnede værdier bliver gemt i Food-instansen, og derefter tilføjes de til arrayet ind, som senere bliver gemt i localStorage. Det ernæringsmæssige indhold for hver fødevare hentes asynkront ved at kalde getfoodData for forskellige næringsstoffer som energi, protein, fedt og fiber. Denne asynkrone tilgang sikrer, at data er tilgængelig, før de gemmes i localStorage.

1.c og 6.a

For at vise hvor mange ingredienser, der er brugt i opskriften, oprettede jeg en funktion kaldet for showind(i). Funktionen formål er at vise ingredienserne i et bestemt måltid ved hjælp af Bootstrap-modalen.

```
function showind(i)
```

Figur 3

Først initialiseres en Bootstrap modal[5] ved hjælp af det givne modal-element med ID'et 'indmodel', og derefter aktiveres modalen ved at kalde show()-metoden.

```
let myModal = new bootstrap.Modal(document.getElementById('indmodel'));
myModal.show();
```

Figur 4

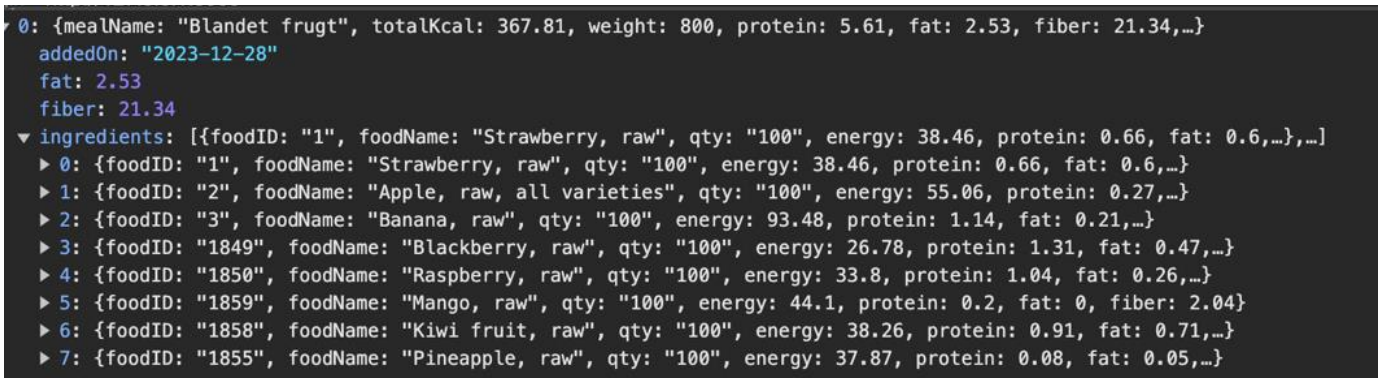
Her oprettes en Bootstrap-modal ved hjælp af modal-elementet med ID'et 'indmodel', og modalen aktiveres straks ved at kalde show()-metoden. Dette sikrer, at modalen er synlig, når funktionen kaldes.
Derefter hentes måltidsdata fra localStorage under nøglen "meal" og konverteres til et JavaScript-objekt:

[5] https://www.w3schools.com/bootstrap/bootstrap_modal.asp

```
let ml = JSON.parse(localStorage.getItem("meal"));
```

Figur 6

Ved hjælp af JSON.parse() [6] konverteres strengen til et JavaScript-objekt, der nu kan bruges i koden. For eksempel, hvis datastrukturen i lokal lagring ser ud som dette:



Figur 7

Så nu vil ”ml” indeholde arrayet af måltider, hvor man kan få adgang til et specifikt måltid ved at bruge indekset i.

Herefter genereres dynamisk tabeldata baseret på ingredienserne i det valgte måltid:

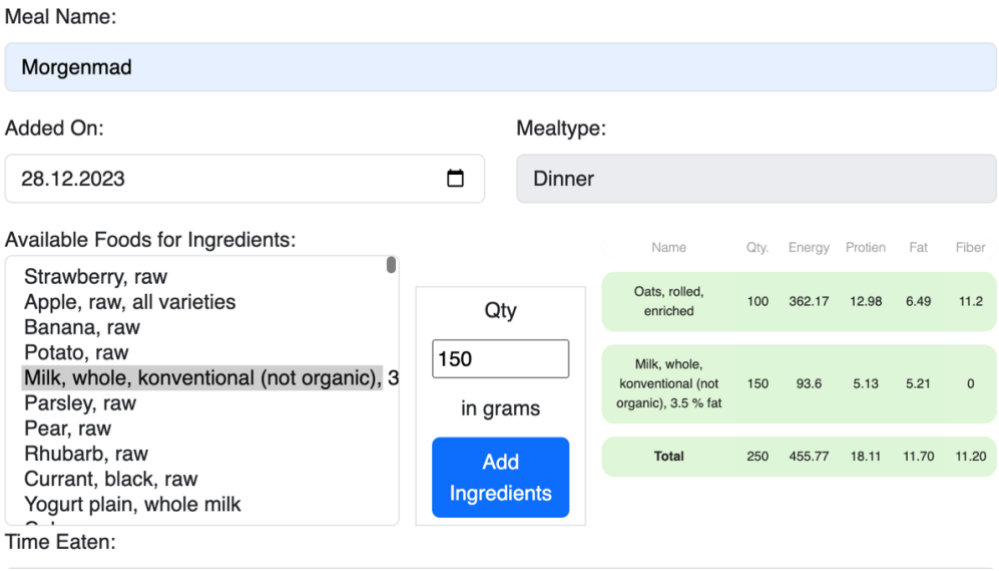
```
for (const e of ml[i].ingredients) {
```

Figur 8

For hver ingrediens i det valgte måltid oprettes en tabelrække, der indeholder oplysninger som fødevarenavn, mængde, energi, protein, fedt og fiber. Disse rækker tilføjes til en dynamisk opbygget streng kaldet tdata. Til sidst opdateres modalens overskrift med måltidets navn efterfulgt af "Ingredienser:", og modalens indhold opdaterer tabellen.

Det vil se således ud i applikationen:

[6] https://www.w3schools.com/js/js_json_parse.asp



Figur 9

Meal Tracker

2.a og 2.b

For at opfylde kravene 2.a og 2.b har jeg udviklet en klasse kaldet "MealConsumption". I konstruktøren af denne klasse gemmes vigtig information om måltider, herunder indeks, navn, tilføjelsesdato, forbrugstid og forbrugt vægt.

Yderligere har jeg implementeret to statiske metoder i klassen. Den første metode, "addMealtracker", har til formål at åbne et modalvindue, hente måltider af en specifik type fra localStorage samt at opdatere dropdown-liste med disse måltider. Dette muliggør, at brugeren kan vælge fra en liste af måltider, der er skabt i meal creator. Den anden statiske metode, "addmealconsumption", sikrer, at det valgte måltid tilføjes til localStorage som en instans af "MealConsumption". Den gemmer information som indeks, navn, tilføjelsesdato, forbrugstid og forbrugt vægt. Efter tilføjelsen nulstilles formularen, og en metode kaldet "showmealtracker" anvendes til at vise de opdaterede måltider. Dette opfylder kravet om at registrere næringsindtag som vægt af en given opskrift, skabt i meal creator, og også hvornår dette måltid eller væske er blevet indtaget.

2.c

Via showmealtracker-metoden opdaterer jeg brugergrænsefladen ved at generere HTML-koden for måltidsregistreringer og præsentere dem i en tabel. Samtidig anvender vi mtcompareDates-funktionen til at sortere disse registreringer efter dato.

Først henter jeg to vigtige arrays fra localStorage, nemlig mt (måltidsregistreringer) og ml (måltidsdata). Dernæst sorteres mt-arrayet ved hjælp af mtcompareDates, så måltidsregistreringerne bliver præsenteret i stigende rækkefølge baseret på dato.

```
static showmealtracker() {
  let mt = JSON.parse(localStorage.getItem("mealtracker"));
  let ml = JSON.parse(localStorage.getItem("meal"));
  mt.sort(mtcompareDates);
```

Figur 10

Efter sorteringen anvender vi en forEach-løkke til at gennemgå hver måltidsregistrering og generere HTML-kode, der indeholder information som måltidsnavn, vægt, kalorier, osv. En tæller ”i ” bruges til at identificere rækkenummeret i tabellen.

```
let i = 0;
let tdata = "";
mt.forEach(element => {
  tdata += `
    <tr>
```

Figur 11

For hver måltidsregistrering opdateres tdata-strengen med den relevante HTML-kode. Til sidst opdateres localStorage med de sortererede måltidsregistreringer, og HTML-indholdet i tabellen opdateres med den genererede kode.

Hvad angår mtcompareDates-funktionen, så spiller den en nøglerolle ved at sikre, at måltidsregistreringerne er sorteret korrekt efter dato. Funktionen konverterer datoegenskaberne for to registreringer til JavaScript Date-objekter og sammenligner dem ved at trække dem fra hinanden.

```
function mtcompareDates(a, b) {
  let dateA = new Date(a.addedOn);
  let dateB = new Date(b.addedOn);
  return dateA - dateB;
```

Figur 12

2.d

Jeg har skrevet to JavaScript-funktioner i min MealConsumption-klasse for at kunne gøre det muligt at redigere måltiderne. Den første funktion, edittracker(i), tager en parameter ”i”, som repræsenterer indekset for måltidet i Meal Tracker. I funktionen henter jeg Meal Tracker dataene fra localStorage og beregner rækkenummeret i

```
static edittracker(i) {
  let mt = JSON.parse(localStorage.getItem("mealtracker"));
  let rowNumber = i + 1;
  let cel = mttable.rows[rowNumber].cells[3];
  cel.innerHTML = `<input id=uweight type="number" value=${mt[i].cweight}>`;
  cel = mttable.rows[rowNumber].cells[4];
  cel.inneHTML = `<input id=udate type="date" value=${mt[i].addedOn}><br><input id=utime type="time" value=${mt[i].ctime}><br><button onclick=MealConsumption.edittrackerupdate(${i})>Save</button>`;
}
```

tabellen baseret på indekset. Derefter opdaterer jeg de relevante celler i tabellen med inputfelter, der tillader redigering af målevægt, måledato og -tid. Derudover tilføjer jeg en "Gem"-knap i samme celle, som ved klik udløser MealConsumption.edittrackerupdate(i).

Figur 13

I den anden funktion, edittrackerupdate(i), opdaterer jeg faktisk Meal Tracker dataene, når brugeren klikker på "Gem"-knappen. Jeg henter først Meal Tracker dataene fra localStorage og opdaterer derefter de nødvendige egenskaber for det specifikke måltid ved hjælp af værdierne fra de tilsvarende inputfelter. Efter opdatering gemmer jeg de ændrede Meal Tracker dataene tilbage i localStorage. Derefter kaldes MealConsumption.showmealtracker() for at opdatere og vise den opdaterede Meal Tracker.

```
static edittrackerupdate(i) {
  let mt = JSON.parse(localStorage.getItem("mealtracker"));
  mt[i].addedOn = udate.value;
  mt[i].ctime = utime.value;
  mt[i].cweight = uweight.value;
  localStorage.setItem("mealtracker", JSON.stringify(mt));
  MealConsumption.showmealtracker();
}
```

Figur 14

2.e

For at kunne slette et måltid fra Meal Tracker, trækker jeg dataene for måltider localStorage ved at bruge localStorage.getItem("mealtracker").

```
let ml = JSON.parse(localStorage.getItem("mealtracker"));
```

Figur 15

Dernæst anvender jeg splice-metoden til at fjerne det ønskede måltid fra listen ved at angive indekset, hvor måltidet befinder sig, samt antallet af elementer, der skal fjernes. Efter sletningen gemmer jeg de opdaterede måltidsdata tilbage i localSorage ved hjælp af localStorage.setItem("mealtracker", JSON.stringify(ml)).

```
ml.splice(indexToRemove, 1);
localStorage.setItem("mealtracker", JSON.stringify(ml));
MealConsumption.showmealtracker();
```

Figur 16

Til sidst opdaterer jeg visningen af Meal Tracker ved at kalde funktionen showmealtracker(), som er ansvarlig for at opdatere brugergrænsefladen med de nyeste måltidsoplysninger. indexToRemove angiver det præcise indeks for måltidet, som jeg ønsker at slette fra listen, som man kan se på figur X.

Nurition Dashboard

For at opfylde de angivne krav (a, b, c, og d) har jeg oprettet en funktion ved navn dashboard. Denne funktion bruger data gemt i localStorage, nærmere sagt de gemte måltider samt måltidsdata, som er repræsenteret via variablerne mt og ml.

```
function dashboard() {
  let mt = JSON.parse(localStorage.getItem("mealtracker"));
  let ml = JSON.parse(localStorage.getItem("meal"));
```

Figur 17 1

For at løse krav a har jeg filtreret måltiderne baseret på den aktuelle dato og viser antallet af måltider ved at opdatere et dashboard-element (todaym.innerHTML) med længden af det filtrerede mtData-array.

```
let mtData = mt.filter(item => item.addedOn === today);
```

Figur 18

Kravet b opfyldes ved at beregne og opdatere variabelen tenergy, der repræsenterer den samlede energiindtagelse for dagen. Denne værdi beregnes ved at tage højde for kalorie-indholdet i hvert måltid, vægten af måltidet og opdateres på dashboardet (todaye.innerHTML).

```
tenergy += ml[e.mealIndex].totalKcal / ml[e.mealIndex].weight * e.cweight;
```

Figur 19

Kravet c opfyldes ved at beregne samt opdatere variabelen twater, der repræsenterer den samlede væskeindtagelse for dagen. Funktionen tjekker, om måltidstypen er "liquid". Den opdaterede værdi præsenteres på dashboardet (todayw.innerHTML).

```
if (ml[e.mealIndex].mealtype == "liquid") twater += e.cweight;
```

Figur 20

Kravet d opfyldes ved at beregne og opdatere en variabel (tp), der repræsenterer den samlede proteinindtagelse for dagen. Denne værdi beregnes ved at tage højde for proteinindholdet i hvert måltid, vægten af måltidet og opdateres på dashboardet (todayp.innerHTML).

```
tp += ml[e.mealIndex].protein / ml[e.mealIndex].weight * e.cweight;
```

Figur 21

Her er der en oversigt over koden for innerHTML:

```
todaym.innerHTML = mtData.length;
todaye.innerHTML = tenergy.toFixed(2);
todayp.innerHTML = tp.toFixed(2);
todayw.innerHTML = (twater / 1000).toFixed(2);
```

Figur 22

Nurition Report

Jeg har oprettet en funktion ved navn shownutrireport, som genererer som viser ernæringsrapporent baseret på data gemt i browserens localStorage. Rapporten opbygges som en HTML-tabel og opdateres i et specifikt element med ID'et "nutri".

Det aller første, som jeg foretog mig var at hente to sæt data fra localStorage ved hjælp af nøglerne "mealtracker" og "meal". Disse dataer er gemt som JSON-strenge og parses til JavaScript-objekterne mt og ml.

```
function shownutrireport() { // Fetch data from localStorage
  let mt = JSON.parse(localStorage.getItem("mealtracker"));
  let ml = JSON.parse(localStorage.getItem("meal"));
```

Figur 23

Dernæst udtrækkes datoer fra egenskaben "addedOn" i elementerne i mt, og de sorteres ved hjælp af en brugerdefineret funktion kaldet compareDates.

```
let uniqueDates = [...new Set(mt.map(item => item.addedOn))];
uniqueDates.sort(compareDates);
```

Figur 24

Der initialiseres derefter en række variabler til at opbevare kumulative værdier for næringsstoffer og anden information. Herefter foretages iteration over hver dato, hvor der udføres beregninger for de samlede værdier af energi, protein, fedt, fiber samt vandindtag for den pågældende dato. Beregningerne for de samlede næringsværdier udføres på baggrund af måltider registreret på den pågældende dato[7].

```
uniqueDates.forEach(dt => {
  tenergy = 0;
  tp = 0;
  tfat = 0;
  tfiber = 0;
  twater = 0;
```

Figur 25

HTML-tabelrækker opbygges med de beregnede værdier for hver dato, og en samlet HTML-streng (tdata) opdateres gradvist. Til sidst opdateres HTML-indholdet i det angivne element med ID'et "nutri" ved at tildele den opbyggede HTML-streng (tdata) til innerHTML. Dette resulterer i visningen af den genererede ernæringsrapport i en tabel på webstedet.

Food Item Inspector

[7] https://www.freecodecamp.org/news/javascript-date-comparison-how-to-compare-dates-in-js/

For at opfylde dette krav, har jeg oprette tre asynkrone funktioner designet til at hente ernæringsoplysninger om specifikke fødevarer fra en API'et. Funktionen fdata(foodID, flag) tager en fødevare-ID og en flag-parameter for at specificere hvilken type information der ønskes.

```
async function fdata(foodID, flag) {
```

Figur 26

Den sender en GET-anmodning til API'en med den angivne ID, bruger en API-nøglen (mit studienummer) til godkendelse, og behandler derefter svaret som JSON. Funktionen returnerer den ønskede information baseret på det angivne flag.

```
method: 'GET',
headers: {
  'Accept': 'application/json',
  'X-API-Key': 168795
```

Figur 27

Funktionen fodoinsp(foodID, sortkey) bruges til at hente detaljeret ernæringsinformation om en specifik fødevare og en given sorteringsnøgle.

```
async function fodoinsp(foodID, sortkey) {
```

Figur 28

Den bruger også en GET-anmodning til API'en, inklusive både fødevare-ID og sorteringsnøgle i URL'en. Svaret behandles som JSON, og ernæringsværdierne hentes og konverteres til numeriske værdier med to decimaler.

```
return (Number(((data[0].resVal).replace(/,/g, '.')))).toFixed(2);
```

Figur 29

Funktionen foodinspector(), styrer processen med at inspicere og vise ernæringsoplysninger for en valgt fødevare.

```
async function foodinspector() {
```

Figur 30

Den viser en indlæsningsmeddelelse, henter fødevare-ID'en fra HTML-elementet, og kalder derefter de to tidligere nævnte funktioner for at hente forskellige ernæringsoplysninger. Disse oplysninger opdateres derefter i relevante HTML-elementer, herunder fødevaregruppe, taksonomisk navn, energi, kalorier, protein, fiber, fedt, kulhydrater, vandindhold og tørstofindhold. Det hele er integreret i en brugergrænseflade, hvor brugeren kan vælge en bestemt fødevare, og systemet vil hente og vise ernæringsoplysningerne ved hjælp af API-kaldene.

```
foodgroup.innerHTML = await fdata(fid, "fødevareGruppe");
tisk.innerHTML = await fdata(fid, "taxonomicName");
kj.innerHTML = await fodoinsp(fid, "1010");
kcal.innerHTML = await fodoinsp(fid, "1030");
protien.innerHTML = await fodoinsp(fid, "1110");
fiber.innerHTML = await fodoinsp(fid, "1240");
fedt.innerHTML = await fodoinsp(fid, "1310");
kulhydrat.innerHTML = await fodoinsp(fid, "1210");
vand.innerHTML = await fodoinsp(fid, "1620");
torstof.innerHTML = await fodoinsp(fid, "1610");
```

Figur 31

Data

I webapplikationen NutriTracker benytter vi localStorage til at gemme data i browseren i JSON-format, hvilket sikrer en problemfri, sikker og vedvarende datalagring.

Food inspector og Meal Creator anvender begge API 'er. Meal Creator adskiller sig dog fra Food Inspector. Deres fælles træk, er at de begge for deres data fra API'et, men Meal Creators data, bliver hent i browserens localStorage i JSON-format under nøglen "fooditems"[8].


```
▼ [{mealName: "Aftensmad", totalKcal: 310.58, weight: 240, protein: 32.5, fat: 12.649999999999999,…}]
  ▼ 0: {mealName: "Aftensmad", totalKcal: 310.58, weight: 240, protein: 32.5, fat: 12.649999999999999,…}
      addedOn: "2023-12-18"
      fat: 12.649999999999999
      fiber: 1.96
    ▼ ingredients: [,…]
      ▶ 0: {foodID: "754", foodName: "Chicken, breast, boiled, sliced", qty: "150", energy: 168.96, protein: 30.9,…}
      ▶ 1: {foodID: "1846", foodName: "Steak fries (thick fries with skin), frozen", qty: "80", energy: 95.44,…}
      ▶ 2: {foodID: "207", foodName: "Mayonnaise, low fat", qty: "10", energy: 46.18, protein: 0, fat: 5.13,…}
      mealName: "Aftensmad"
      mealtype: "Dinner"
      protein: 32.5
      timeEaten: "1"
      totalKcal: 310.58
      weight: 240
```


Figur 32

I Meal Tracker gemmes information om brugerens måltids- og vandindtag som individuelle elementer i localStorage. For hvert måltid gemmes detaljer såsom tidspunktet for indtag samt ernæringsindholdet. Meal Tracker opretholder en adskillelse mellem disse dataelementer for at bevare og organisere brugerens registreringer på en struktureret måde:

[8] https://developer.chrome.com/docs/devtools/storage/localstorage

Figur 33

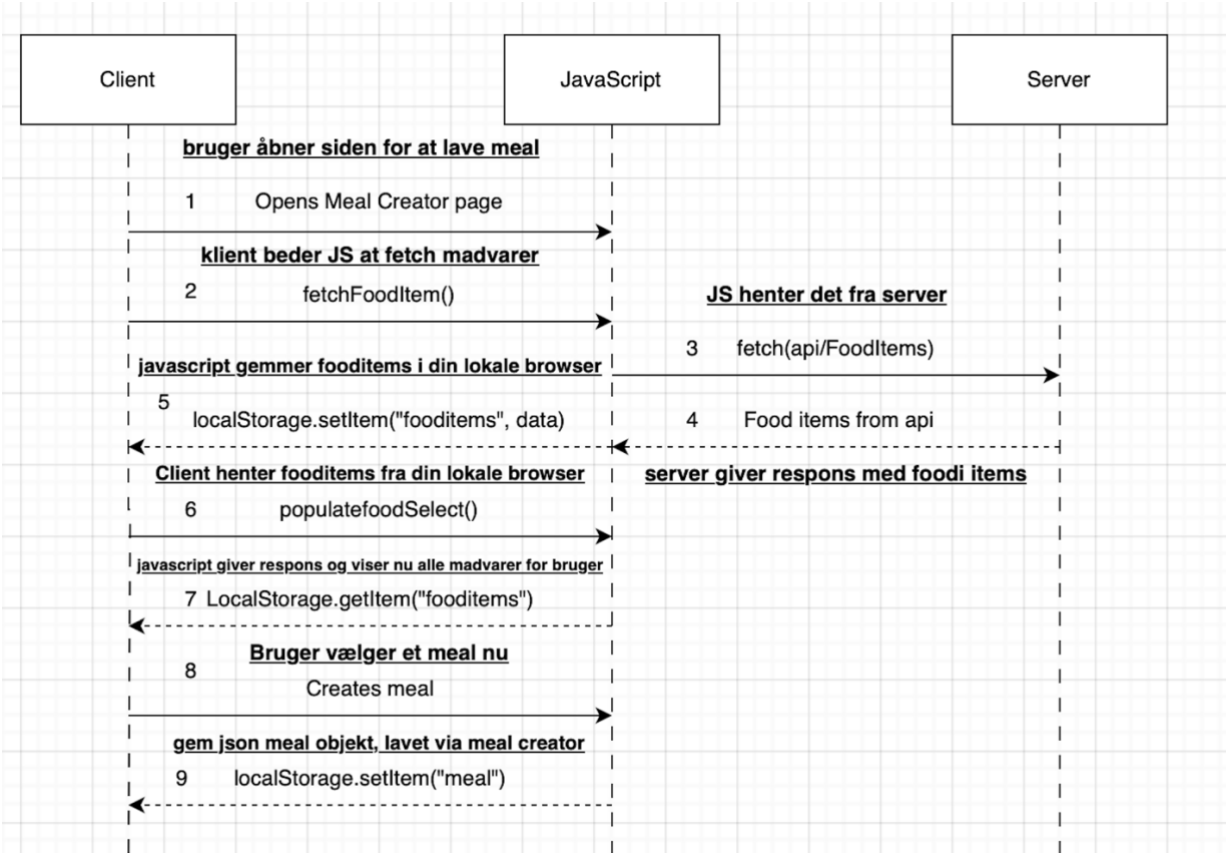
Både Dashboardet og Nurition Report henter data fra localStorage. Dashborat genererer et overblik over daglige måltider og vandindtag, mens Nurition Report får et overblik for de seneste syv dage. Informationen om måltiderne og måltidsregistreringerne er gemt i localStorage, hvorefter de bliver brugt til at opdatere brugergrænsefladen med aktuelle oplysninger om kost og vandforbrug. localStorage gør det muligt at bevare brugerens kosthistorik samt lettere at beregne samlede næringsværdier baseret på tidligere tilføjede måltider. Denne tilgang giver en mere sammenhængende og personlig oplevelse ved at holde styr på brugerens daglige vaner over tid.

Når din klient eller browser åbner meal creator-formularen, initiates en anmodning om at hente alle måltider fra en API. Disse måltider bliver derefter vist i formularen ved hjælp af funktionen populateFoodSelect samt localStorage.getItem. Når du opretter et nyt måltid, bliver dette nye måltid gemt.

Forsættes ved næste side

Sekvens diagram

Jeg har valgt at illustrere min kode med et UML-diagram, kaldet et sekvensdiagram[9]. Dette skyldes, at det viser interaktionen mellem objekter eller komponenter i et system og fokuserer på den kronologiske rækkefølge af beskeder, der udveksles mellem objekter.



Figur 34

Når klienten eller brugeren af programmet åbner meal creator-formularen, initieres fetchFoodItem()-funktionen. Dette trin omfatter hentning af alle nødvendige madvarer fra det eksterne API. Fra JavaScript til serveren etableres der en kommunikationspil, da madvarerne er placeret på en server og hentes via fetch-anmodningen i JavaScript-koden. Serveren svarer med en respons til JavaScript (det angivne sted som "food items from API").

Herefter genererer JavaScript-koden en respons til klienten, nærmere bestemt ved brug af localStorage.setItem-metoden. Dette gør det muligt at gemme madvarerne i localStorage på klientens computer i browseren.

[9] https://www.lucidchart.com/pages/uml-sequence-diagram

Når denne proces er fuldført, udføres populateFoodSelect-funktionen fra klientens side. JavaScript svarer ved at hente madvarerne fra localStorage vha. LocalStorage.getItem("fooditems") og viser dem derefter for brugeren. Efterfølgende nævnes trinnet "creates meal", idet brugeren nu har mulighed for at vælge et måltid. JavaScript reagerer ved at gemme det valgte måltidsobjekt i localStorage ved hjælp af localStorage.setItem("meal").
De pile, som går fra venstre mod højre, repræsenterer interaktionen mellem klienten og JavaScript. Piler, der går fra højre til venstre, såsom "food items from API" (fra serveren til JavaScript), indikerer responsfasen. Mellemrummene i responspilerne er markeret med stiplet linje.

Procesevaluering og perspektivering

At udvikle koden og skrive rapporten var virkelig lærerigt for mig. Det har hjulpet mig med at forbedre mine kodningsfærdigheder og grundlæggende forståelse af emnet. Kravene til opgaven passede godt sammen med det, vi lærte i løbet af semesteret, og det gjorde det nemt for mig at forstå sammenhængen med vores pensum. Arbejdet med et større projekt har givet mig mulighed for at anvende det, jeg har lært, og øve mig i at blive bedre til at kode.

Min forståelse af kodning er blevet markant bedre, især fordi jeg selv har skullet finde løsninger på udfordrende problemer. Det tog ofte tid og krævede grundig undersøgelse for at forstå, hvor i min kode fejlen var. En vigtig begivenhed, der påvirkede min måde at strukturere koden på, var opbygningen af selve koden. At skabe en tydelig struktur i mine filer var afgørende for at bevare overblikket over opgaven. I starten havde jeg alt mit JavaScript, i en fil, som skabte en del forvirring, samt rodet struktur i koden. Dette fik jeg løst, men til næste gang, vil jeg foretrække, at jeg deler koderne op i forskellige mapper, så alt HTML koden, vil være i en mappe, og alt JavaScript koden i en anden mappe. Dette vil bidrage til at skabe en bedre kodestruktur.

En anden ting, som jeg også har overvejet at implementere til fremtidige projekter, er løbende dokumentation. Dette vil hjælpe mig med at sikre, at alle mine tanker og beslutninger bliver klart dokumenteret.

Generelt set opstod den bedste forståelse af min kode, da jeg skulle skrive rapporten. At formulere mine tanker om koden og beskrive min tilgang testede både min viden og mine evner til at formidle komplekse ideer. Evnen til at koble teorien præcist til de enkelte funktioner samt elementer i koden har styrket min grundlæggende forståelse af de forskellige elementer, jeg har arbejdet med, både i opgaven og i løbet af semesterets undervisning.

Konklusion

Eksamensprojektet, NutriTracker, har resulteret i udviklingen af en effektiv webapplikation, der opfylder de specificerede krav, der blev identificeret i starten af udviklingsprocessen. Ved at kombinere JavaScript, HTML og CSS har NutriTracker skabt en robust og effektiv platform, der gør det let for brugere at spore og analysere deres ernæringsindtag. I løbet af projektet har jeg udforsket forskellige aspekter af webudvikling, herunder interaktionen med eksterne API'er. Samtidig har jeg sikret pålidelig datahåndtering ved hjælp af localStorage i JSON-format. Alle funktioner i NutriTracker, fra Meal Creator til Nutrition Report, er omhyggeligt designet for at sikre, at brugerne har et pålideligt værktøj til at støtte deres sundheds- og ernæringsmål. NutriTracker er ikke blot opfyldelsen af de oprindelige krav, men udgør også en platform med betydeligt potentiale for fremtidig udvidelse og forbedring. Dette projekt har ikke alene styrket min tekniske kompetence, men har også bidraget til min forståelse af, hvordan jeg i fremtidige projekter kan opretholde en mere struktureret kode.

Video

Link til Loom video: https://www.loom.com/share/7033b0e38c74404aa52ea95dda44db20?sid=4cf62e1b-97f7-40d3-a48b-91407f54ffb1

Litteraturliste:

- Google fonts, sidst besøgt 11/12 – 2023
  https://fonts.google.com/
- W3shcools, class static, sidst besøgt 15/12 – 2023
  https://www.w3schools.com/js/js_class_static.asp
- W3Schools, bootsrap modal, sidst besøgt 15/12 – 2023
  https://www.w3schools.com/bootstrap/bootstrap_modal.asp
- W3schools, JSON parse, sidst besøgt 15/12 – 2023
  https://www.w3schools.com/js/js_json_parse.asp
- Olawanle Joel, FreeCodeCamp, JavaScript Date Comparison, sidst besøgt 16/12 – 2023
  https://www.freecodecamp.org/news/javascript-date-comparison-how-to-compare-dates-in-js
- Basques og Emelianova, View and edit local storage, sidst besøgt 15/12 – 2023
  https://developer.chrome.com/docs/devtools/storage/localstorage
  HA(IT.) øvelsesvejleder, uge 46, sidst besøgt 15/12 – 2023
- https://cbscanvas.instructure.com/courses/33500/files/1185564?module_item_id=838276
- Lucidchart, UML Sequence Diagram Tutorial, sidst besøgt 20/12 – 2023
  https://www.lucidchart.com/pages/uml-sequence-diagram