Programmerings

Synopse

–

Køleskab

Indholdsfortegnelse

[Problemformulering 3](#_Toc102842583)

[Arbejdsproces 3](#_Toc102842584)

[Programmering og sprogvalg 3](#_Toc102842585)

[Forklaring af min kode 5](#_Toc102842586)

[Klienten 8](#_Toc102842587)

[Test af programmet 8](#_Toc102842588)

[Konklusion 9](#_Toc102842589)

[Bilag 9](#_Toc102842590)

Jeg har sammen med Thomas fået ideen om at lave et program til ens køleskab som kan holde styr på de varer man har i køleskabet. Det program kan man bruge til for eksempel at finde opskrifter på internettet automatisk baseret på hvad man har derhjemme, eller man kan gå ind på sin telefon og se om man havde fået købt mælk eller ej mens man er ude og handle.

# Problemformulering

Hvordan kan man lave et program der kan give overblik over de ting man har i sit køleskab og hjælpe en med at tage beslutninger i forhold til madlavning?

# Arbejdsproces

Det første jeg gør når jeg skal lave et nyt projekt, er at finde ud af hvad det generelle mål er og hvad strukturen ca. kommer til at være. Her fandt jeg ud af at jeg ville have en server som skal virke som en database til de køleskabe og brugere der har ting i deres køleskab, og så skulle der være to typer klienter. En type klient skulle være det der kørte på selve computeren i køleskabet, som kan registrere hver gang man sætter noget i køleskabet, og en klient som en bruger kunne logge ind med til at se hvilke varer der ligger i køleskabet.

Når jeg programmerer, bruger jeg på en måde ”stepwise improvement” metoden, hvor jeg starter med at lave en lille del af programmet og får det til at virke, før jeg går videre med at tilføje flere funktioner. Det jeg startede med, var at lave en server som man kunne tilgå fra en browser, for at få etableret det helt basale, en server og en klient. Derefter fik jeg klienten og serveren til at kommunikere sammen direkte så hjemmesiden kan være dynamisk og så det er lettere at have flere personer på samme tid. Jeg skal også have en database til serveren, sådan så den kan huske hvilke varer der ligger i forskellige køleskabe. Da jeg kun skulle bruge dette til en lille skala valgte jeg bare at opbevare data i en fil på min computer i stedet for at have en ordentlig database. Nu hvor jeg havde de basale ting til en server kunne jeg beslutte hvordan min datastruktur skulle være. Jeg lavede et overblik over hvilke data hver ting man kunne have i sit køleskab skulle have, jeg lavede en liste af endpoints som min server skulle svare på med data til klienten, og jeg lavede en liste af ting som klienten skal kunne modtage når serveren sender dem. Den liste har jeg lavet i filen structure.md.

# Programmering og sprogvalg

Jeg har valgt at bruge programmeringssproget typeScript, som er en udvidelse af sproget javaScript, til at lave dette projekt fordi det fungerer godt til webapplikationer. JavaScript er et Objekt orienteret sprog som de fleste store programmeringssprog er, og det betyder at det er baseret på objekter og klasser som er fleksible og kan indeholde store mængder forskellige data. Objekt orienteret er modsætningen til funktionel programmering som er mere fast og bruger funktioner til at behandle data. Det ses typisk brugt i meget optimerede systemer hvor det er vigtigt at alt går meget hurtigt, eller hvor man ikke har så mange ressourcer tilgængelige, som i microprocessors. JavaScript er også et high-level sprog som er forskellig fra for eksempel C som er mere low-level. Det betyder at JS er længere væk fra selve den processor den kører på, fordi alt det bliver styret af ECMA Script engine og man skal så ikke selv håndtere ting som memory management og cpu registers.

I JavaScript er der nogle forskellige data typer. Disse inkluderer

* Boolean
* Number
* String
* Object
* Function

Boolean datatypen er den mest simple datatype. Den repræsenterer en enkelt bit, som er en værdi af enten sandt eller falsk, også repræsenteret som 1 eller 0, og tændt eller slukket. En boolean værdi kan bruges direkte ved bare at skrive enten true eller false.

true //sandt

false //falskt

Man kan også bruge en boolean værdi ved forskellige conditions. Det kan man gøre ved at bruge forskellige operatorer som fx større end, mindre end, og lig. De bruges ved at man har en værdi på hver side af operatoren og baseret på de værdier bliver det resulterende enten sandt eller falskt

5 > 2 //true

8 < 6 //false

14069 == 4 //false

Number datatypen er et tal. I andre programmeringssprog har man typisk forskellige slags tal, fordi de er brugt så meget. De typiske datatyper er integer, long integer, float, og double som betyder heltal, decimaltal, og decimaltal med dobbeltpræcision. Men i JS består alle tal af doubles, hvilket betyder at et tal kan have en værdi på op til 9007199254740991 (2^53) hvorefter tallet ikke længere er præcist.

100

75

90911901919

var a = 100;

a //100

hvis man gerne vil have tal der er større end 9007199254740991, kan man bruge klassen BigInt, som ikke har en størrelses grænse, men som kun kan have heltal. Den bruger også mere ram og støtter ikke alle operationer.

BigInt(100); //100n

BigInt(129048319048391048910348109348)+BigInt(1); //129048319048391048910348109349n

Datatypen string er en række af bogstaver, som fx et ord, en sætning eller anden kombination af bogstaver og tal. Næsten hver gang man skal have tekst på en skærm bruger man strings til at opbevare den tekst. Der er ikke nogen grænse for hvor store strings kan være. Men man skal selvfølgelig passe på at man ikke løber tør for ram. Et bogstav i JS fylder som standard 2 bytes, så en string kommer til at fylde længde\*2 bytes.

"bogstaver"

"string" + "string" //"stringstring"

var navn = "Nicklas";

var efternavn = "Østerberg";

navn + " " + efternavn //"Nicklas Østerberg"

hvis man vil være lidt mere fancy, kan man også bruge string substitution, som er hvor man har en string og man sætter en anden string ind i den.

`${navn} ${efternavn}` //"Nicklas Østerberg"

JavaScript objektet er nok en af de bredeste funktioner i sproget. Næsten alt man bruger i JS er på en eller anden måde et objekt. Et objekt i sin simpleste form er bare nogle key-value-pairs, altså man har værdier som kan være en af alle JS datatyper, eller med andre ord, kan være lige hvad man vil, og så har man keys som er navnene på de værdier og det man bruger til at referere til de værdier. Keys kan kun være af datatypen string.

var person = {navn: "Nicklas", efternavn: "Østerberg", alder: 18, sej: true, sjovJoke: () => {return "What do you call a fish with no eyes? Fsh";}};

Den sidste datatype er funktionen. Funktionen består ikke af et stykke information, men i stedet noget kode som kan udføre forskellige handlinger med de input parametre den tager ind. En funktion kan ændre på data i globale variabler, eller den kan returnere noget data som man kan bruge når man kalder funktionen.

Her er fx en funktion som tager noget data og kryptere det med en offentlig nøgle

function encrypt(data) {

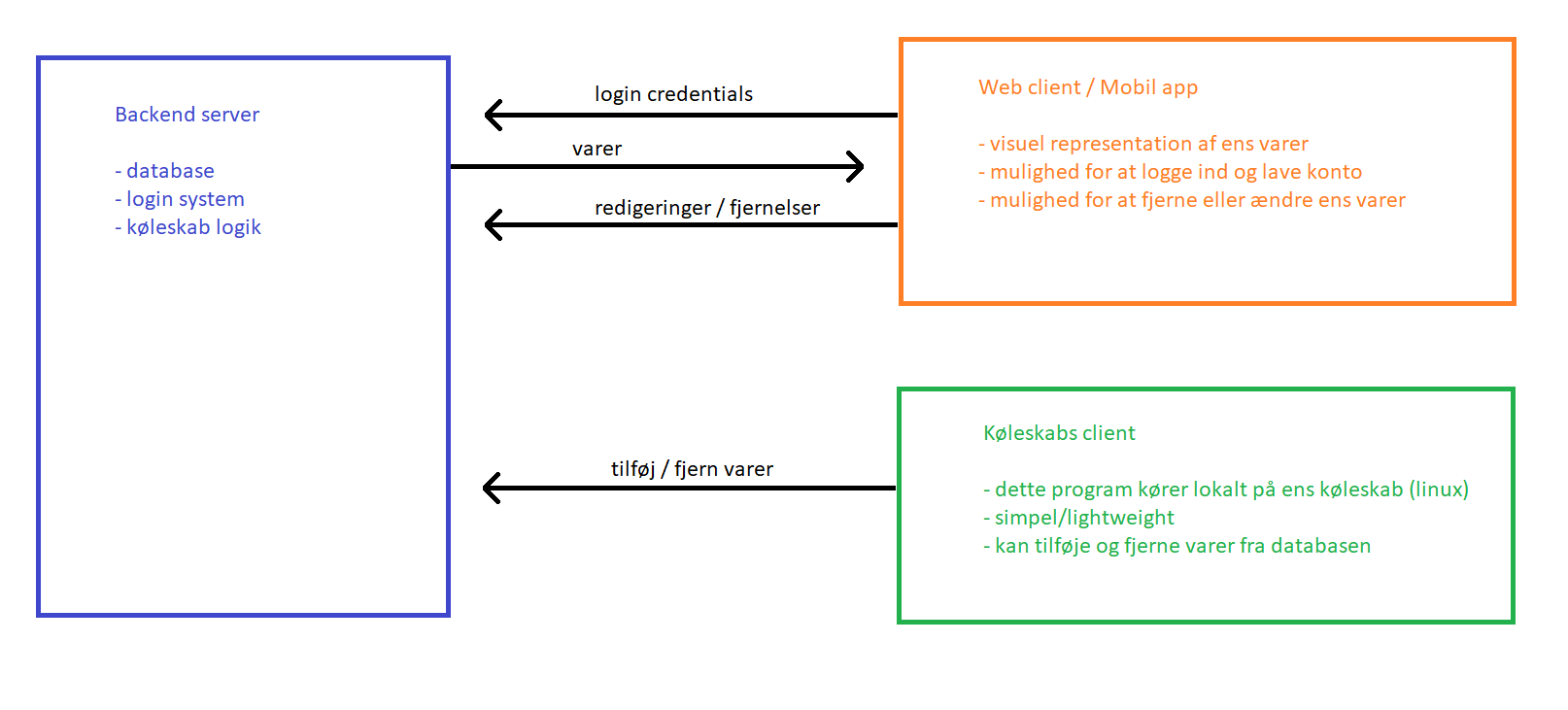
    const encrypted = crypto.publicEncrypt(publicKey, Buffer.from(data));

    return encrypted.toString("base64");

}

# Forklaring af min kode

Et diagram af mit program ser sådan ud:



Vi har et felt til hver af de 3 store komponenter, køleskab, server, og klient, samt nogle punkter om hvad de hver iser består af.

Serveren er den der styrer databasen i projektet. Databasen er i dette tilfælde bare en JSON fil der bliver styrret af et npm library ved navn ”node-json-db”. Dette er ikke en perfekt løsning da det både kræver at hele datasættet ligger i ram mens programmet er aktivt. Det gør også at det er langsomt at gemme data da det skal skrives til en fil hele tiden. Hvis dette projekt skulle udvides, skulle man bruge et bedre alternativ som en sql server eller andet database system.

import {JsonDB as jsondb} from "node-json-db";

var database = new jsondb("database.json", true, true);

De første to linjer importerer koden og laver en ny instance af jsondb klassen, som referer til den fil hvor alt dataen ligger på

Dette er et udsnit af forbindelsen mellem clienten og serveren.

Dette stykke kodes funktion er at vente på at en client vil tilføje en vare og derefter tilføje varen

socket.on("add", async (data: string, res: (res: boolean|object) => void) => {

    if (socket.user?.loggedIn) {

Først tjekker vi om brugeren faktisk er logget ind.

Denne variabel ligger på serveren, så den er beskyttet fra at brugeren kan ændre på den uden at have den rigtige kode

        console.log(data);

        var recieved: item = JSON.parse(data);

dette er den vare brugeren ønsker at tilføje.

På grund af sikkerhed tager vi alle de værdier vi skal bruge en ad gangen og laver et nyt objekt, så vi undgår at brugeren kan tilføje farlige attributter

        var item: item = {

            item: recieved.item,

            size: recieved.size,

            image: await getImage(recieved.item),

            date: recieved.date,

            id: crypto.randomUUID(),

            uses: 0,

            barcode: recieved.barcode || "",

            expirationPeriod: recieved.expirationPeriod || 0

        }

Derefter tilføjer vi varen til databasen, under brugerens konto.

        database.push("/accounts/" + socket.user.name + "/items[]", item);

        res(item);

og vi svarer også brugeren så den ved at dens anmodning er modtaget.

    } else {

        res(false);

hvis brugeren ikke er logget ind sender vi et svar tilbage uden data, så brugeren ved at der er sket en fejl.

    }

});

Denne funktion kommer fra webklienten og er den der sender dataen om at vi vil tilføje en ny vare til serveren. Først tilføjer vi funktionen til den handling der sker når man trykker på knappen for at tilføje en vare.

addItem.addEventListener("click", newItem);

function newItem() {

    var name = document.getElementById("item-name");

    var size = document.getElementById("item-size");

    var list = document.getElementById("list");

disse laver variabler med de forskellige html elementer vi skal bruge i funktionen.

    if (item.value !== "" && size.value !== "") {

        var item = {

            item: name,

            size: size

        }

Her sender vi dataen til serveren på endpointet “add” som er det samme som vi lytter til på serveren. Vi tilføjer også en function mere som bliver kaldt når serveren har behandlet anmodningen.

        socket.emit("add", JSON.stringify({item: item.name, size: size.value, date}), (res) => {

Hvis svaret fra serveren ikke er false, så kan vi lave et html element og tilføje det til listen af elementer.

            if (res) {

                list.insertBefore(itemElement(res), list.firstChild);

            } else {

Eller viser vi at der er sket en fejl

                alert("Fejl");

            }

Og vi tømmer også skrive felterne hvor vi har skrevet navn og størrelse på varen.

            item.value = "";

            size.value = "";

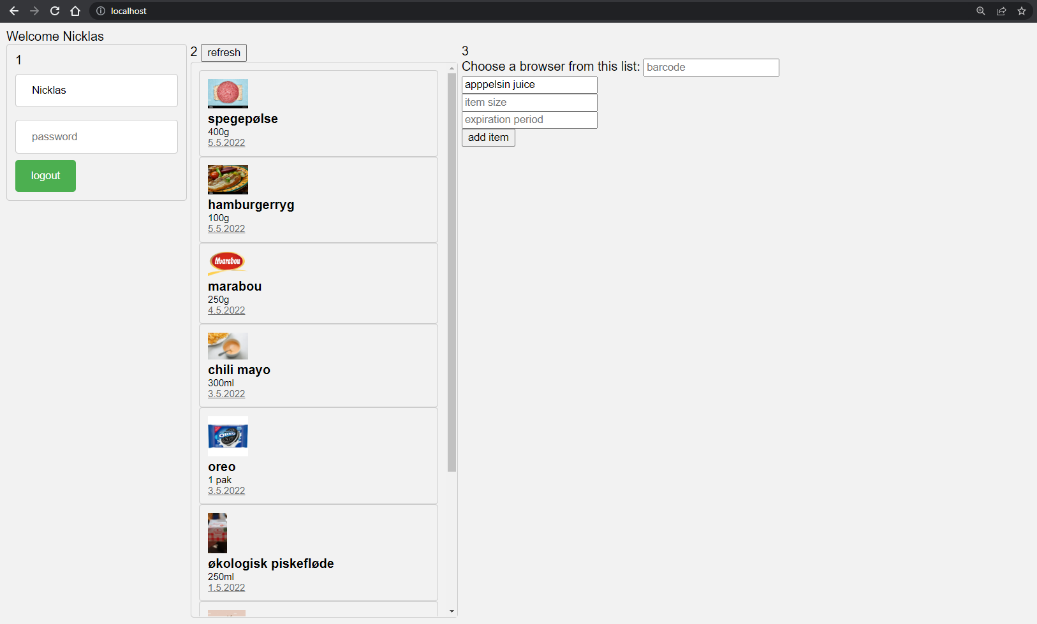
        });

    }

}

# Klienten

Da mig og Thomas ville lave selve projektet sammen, valgte vi at dele opgaven op så jeg stod for at lave det tekniske med kode og database, og hvor Thomas ville stå for designet af klienten som brugeren ville interagere med. Men Thomas har siden da ikke været meget i skole og har derfor ikke lavet sin del af opgaven, så fokusset for mig vil være hvordan den teknisk fungere og ikke hvordan dens udseende er fordi det tydeligvis ser meget dårligt ud.

Det brugeren kommer til at se når de kommer ind på hjemmesiden er et loginfelt, med alt hvad man regner med til sådan et, en liste af de ting man har i sit køleskab (når man er logget ind), og et sted hvor man kan tilføje vare til sit køleskab uden at bruge appen på køleskabet.

*Bilag 2*

# Test af programmet

Programmet har nogle få mangler som jeg ikke fik implementeret. Fx ville jeg gerne have tilføjet at man kunne se hvornår ens vare overskred udløbsdatoen, eller hvor mange gange man har brugt varen i køleskabet så man har en ide om hvor meget man har tilbage af hver vare. Men generelt set virker programmet ret godt og der ligger en video i mappen her, hvor man kan se mig logge ind på en konto og se mine varer og tilføje en ny vare til mit køleskab.

# Konklusion

Jeg har fået lavet et program som kan gøre det lettere for folk at huske hvad de har i deres køkken når de fx er ude at handle. Jeg har også lavet et program som let vil kunne udvides til at kunne finde opskrifter online som hovedsageligt benytter det man har i sit køkken. Det jeg ikke har fået nået, er at få designet hjemmesiden så den ser pen og indbydende ud, og jeg har heller ikke lavet en fysisk computer til et køleskab som kan læse de varer man sætter ind i køleskabet. Men projektet er alligevel nået en ret stor skala med mere end 1000 linjer kode i alt og har nået de fleste af de mål jeg har sat for det.

# Bilag

Der ligger bilag i den mappe hvor dette dokument også lå.

Der ligger et billede af diagrammet over min kode, en video af mig der viser programmet og alt koden til mit program.

Der ligger også en readme fil til hvordan man kører koden.