

DIT KØLESKAB

Informatik B - Eksamensprojekt



MAY 3, 2021
KØBENHAVNS MEDIEGYMNASIUM
Jack T. Oulund

Indholdsfortegnelse

Kravsspecifikation	2
Design	2
Præsentationslag	2
Logiklag	6
Data lag	7
Realtime database	7
Authentication data	9
Implementation	10
App og Login	11
Display af vare.....	13
Vise varene.....	13
Test	17
Konklusion	17
References.....	18

Denne rapport omhandler mit eksamensprojekt i informatik B, hvor jeg har valgt at lave en web app med tilhørende database, som skal hjælpe brugeren med at holde styr på sit køleskab. Det kunne være når personen har brug for et overblik over hvad køleskabet er fyldt med, hvis man for eksempel er ude at handle eller måske bare er usikker over hvad der er i det. Derfor har jeg valgt at lave "Dit Køleskab." I rapporten vil du først komme til at læse om kravene, der er blevet stillet til projektet, derefter vil der komme et afsnit som handler om designet af hele produktet. Det er alt fra database strukturen og forskellige ting til at lave koden til valg af udsénet af UI. Efter design kommer der et afsnit om hvordan jeg har lavet web-appen og implementeret de forskellige funktioner. Næst sidste afsnit handler om hvordan min fremgangs måde i udviklingen og hvordan jeg har lavet forskellige test. Til sidst kommer min konklusion, hvor jeg konkludere hvordan det er lykkedes med at udvikle web-appen og hva der kunne have været gjort bedre og fremtidige muligheder for produktet.

Kravsspecifikation

En dynamisk web app der skal opføre sig som brugerens køleskab online ved at:

- Have et system der kan autentisere brugerne, så de kan få et individuelt køleskab og en login og logud knap
- Vise varene i køleskabet i en tabel med et billede af varen, et navn, en udløbsdato for den ældste vare og en visning af antallet af den vare
- Tilføje flere vare til køleskabet
- Have en database med en tabel til alle varerne man kan vælge, og en til alle brugerne med oplysningerne om dem og alle deres personlige varer.+
- Have et simpelt og stilrent design.

Design

Her i design afsnittet har jeg delt det op i tre dele efter de tre lag af webudvikling, som er Præsentationslag, logik lag og datalag. (Damhus, Buch, & Husum, 2021)

Præsentationslag

Målet for præsentationslaget var at det skulle være et simpelt og stilrent design. Derfor lavede jeg nogle mockups af det i Adobe XD, som er beregnet til at lave mockups af hjemmesider, mobil apps og mere.

Jeg vil forklare de forskellige dele af siden ved hjælp af nogle af de forskellige design love, regler og den slags.

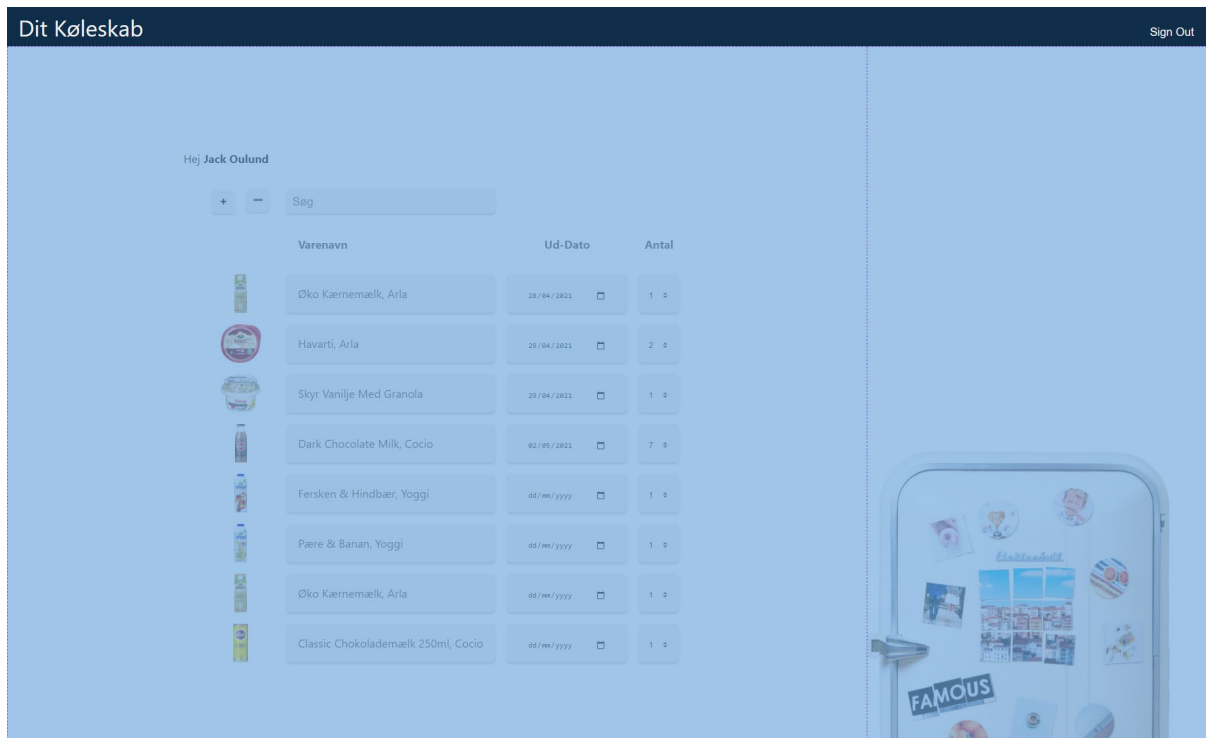
Åben dit køleskab, som var du derhjemme

Det her er dit online køleskab, så du altid vil kunne se hvad du har i det, selvom du er på farten.



Her over ses index(hoved siden) af præsentationslaget. Det er den første side man får at se når man kommer ind på siden. Den består af 3 komponenter, en navigationsbar, et velkomstområde, som også er hovedområdet og et billede, som går igen på hele websitet og også bliver brugt som icon/logo for siden.

Det generelle layout for siden, er som vist på billedet nedenunder, hvor navigationsbaren er i toppen og de to sektioner nedenunder er opdelt af ved den lyserøde streg.



Allerede her er der en af gestalt lovene der træder i kraft. Nemlig loven om lukkethed, som viser sig ved at navigationsbaren har en mørk baggrund, som gør at den bliver set som et lukket element og er noget for sig selv imodsætning til det med hvid baggrund.

Tilbage til de 3 hoved elementer, kan det også siges at det følger KISS(Keep It Simple Stupid) og FTF(First Thing First), da hele designet er simplificeret ved at dele siden op i de 3 elementer, som også har gjort det nemmere at få FTF ind i det, da hele layout følger den naturlige retning vi læser i (oppe fra og ned, fra venstre til højre), som gør at man først ser nav-baren, derefter hoveddelen og til sidst billedet af et køleskab.

Den næste side der kommer efter man har trykket på login, det er ens profil side.








Dit Køleskab


Log Ud

Dine varer

+ -

Søg

Varenavn	Ud-Dato	Antal
 Skummetmælk	Dato	1▼
 Skummetmælk	Dato	1▼
 Skummetmælk	Dato	1▼
 Skummetmælk	Dato	1▼
 Skummetmælk	Dato	1▼
 Skummetmælk	Dato	1▼
 Skummetmælk	Dato	1▼



Her er forskellen på hovedsiden at man kan se sit personlige online køleskab.

Hovedområdet på den her side gør sig meget brug af loven for nærhed og loven for lighed.

Loven for nærhed kommer til syne ved at alle de komponenter der er, er tættere på

hinanden alt afhængig af hvor meget de hænger sammen med hinanden. Det kan man se ved at rækkerne i tabellen ligger tættere på hinanden end de ligger på søgeområdet.

Derudover ligger scrollbaren i mockup'et også tæt på tabellen for at man kan se at den hører

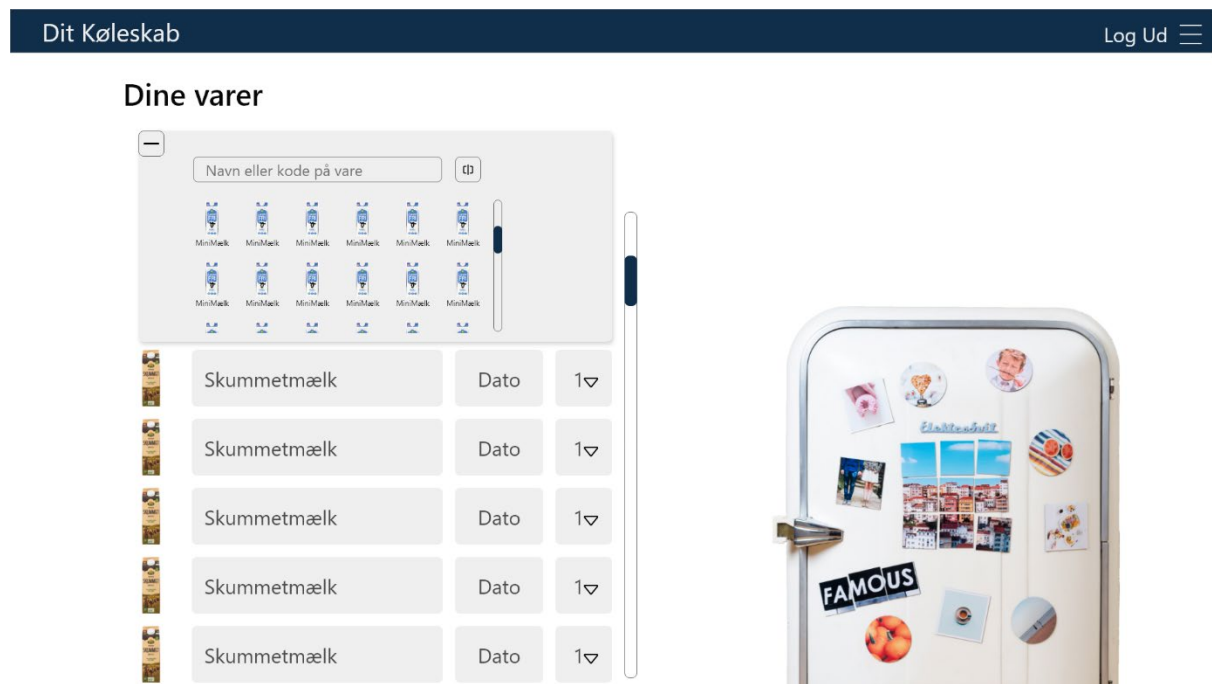
til tabellen. Lighed kommer til udtryk ved at alle rækkerne ligner hinanden. De har alle

sammen de samme felter, som derudover også har samme størrelse. Det eneste der er til

forskel ved dem er indholdet i felterne. Derfor bliver felterne også vist med en baggrund,

som gør at selvom om de har et andet indhold end de andre, har de stadigvæk en lighed til de andre.

Det næste mockup er når man skal tilføje en vare til sit køleskab, som kommer frem når man trykker på plusset ved siden af søgefeltet.



Her kommer loven for lukkethed virkelig til udtryk, da hele feltet man kan vælge en ny vare i er lukket inde i en boks. Alle Varene bliver også vist som en masse billeder med et navn tilhørende. Her kan man se forskel på hver vare pga. teksten og varen ligger tæt på hinanden og at de andre vare har det samme format at blive vist på. Derfor er der også en del lighed i menuen.

Da det er en hjemmeside, bliver userinterfacet bygget op af html og css, hvor at html koden bliver generet i logiklaget af React.

Logiklag

I logiklaget har jeg valgt at bruge JavaScript biblioteket React til at styre logikken. React er godt til at lave dynamiske og brugervenlige userinterfaces. Måden det er bygget op på er ved hjælp af forskellige komponenter, som man laver, hvor det er rigtig nemt at lave logikken for hver komponent. Hvert React komponent kan også have forskellige states, som er med til lave dynamikken på siden, for hver gang et state ændres på et enkelt komponent, behøver logiklaget kun at re-render det enkelte komponent, frem for at skulle reload hele siden hver gang, der sker noget. Det kan give en meget bedre og mere "smooth" "user experience."

React's komponenter bygger også "user interfacet" op ved hjælp af JSX. JSX er en nemmere måde at lave dynamisk html kode i JavaScript, dog skal det kompileres før det er brugeren får det på client side.

Det er nogle af grundene til at jeg har valgt at bruge React.js til at bygge webappens logiklag. Det gør også at jeg nemmere kan dele koden op i de forskellige komponenter der er i designlaget. React(logiklaget) kommer også til at køre som min back end, som også gør at jeg kan bruge firebase's JavaScript bibliotek på serveren i stedet for på client side, som også er mere sikkert når man skal have kontakt med data laget(databasen), som er lavet med firebase.

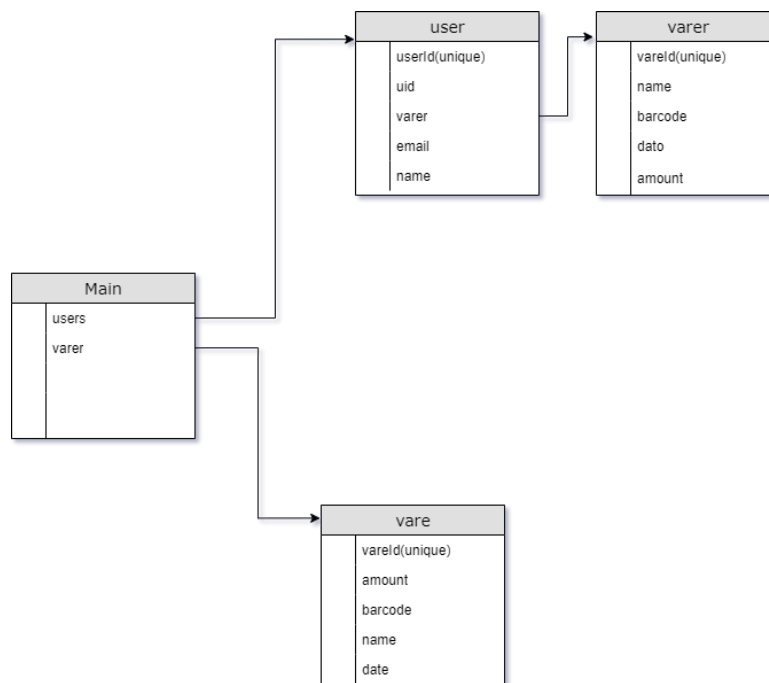
Data lag

Mit datalag består af en firebase realtime database og firebase' authentication system, som opbevare brugerdata.

Realtime database

Realtime databasen er en database, som mange andre er, men den er smart på det punkt, at den opdaterer i real tid. Derfor får der sker nogen ændringer i databasen, bliver de opdateret med det samme, og de clienter eller servere der lytter/henter data fra den, bliver automatisk opdateret. Det passer også rigtig godt sammen med react, da man fx kan få webappen til at rerender komponentet's state lige så snart at der sker en opdatering i databasen, sådan at klienten hele tiden har den nyeste data.

Herunder er et er-diagram, som beskriver databasens struktur opbygning.

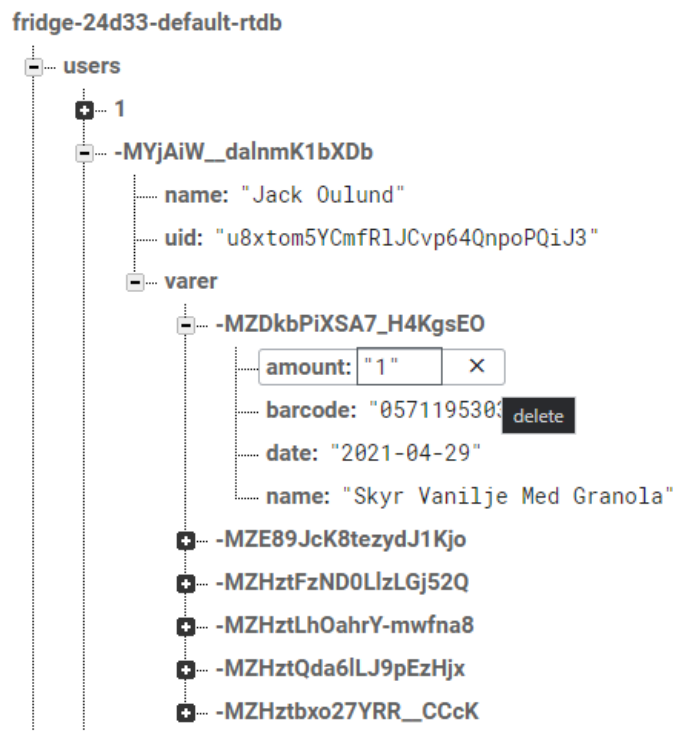


Den består hovedsageligt af 2 tabeller. En users tabel, som indeholder alle brugernes data, og en varer tabel, som indeholder alle data om de forskellige vare.

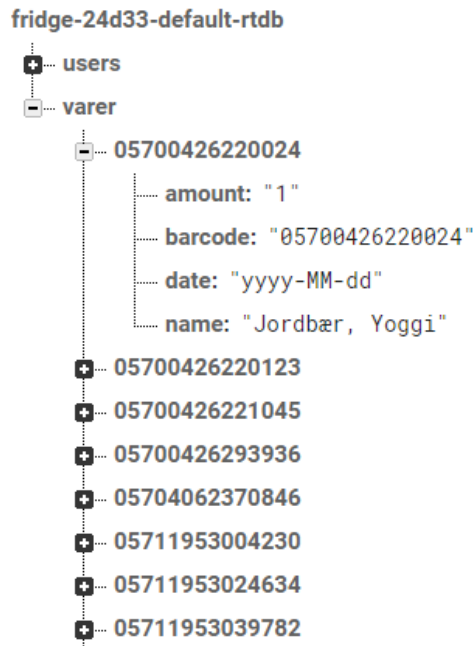
I users tabellen har hvert element nogle forskellige data. Hver user har et userid, der er en unik nøgle, et uid, som kommer fra firebase' authentication, som også er unikt, men det bliver ikke brugt til at sortere brugerne med. Derudover har brugerne en email og et navn fra firebase auth. Det sidste felt hver bruger har er et felt med alle brugerens vare. Vare er en undertabel til hver bruger, som kan indeholde en masse elementer. Det er et vareld, som er unikt, et navn, en barcode, en dato og amount.

Vare tabellens elementer indeholder de samme felter som brugerens varetabels elementer gør, da det er meningen at elementerne i den overordnede tabel bare fungerer som skabeloner. Det vil sige at når man så tilføjer en vare til brugeren kopierer man varen fra den overordnede tabel og udskifter de placeholders der er (amount og date), så den dermed kommer til at passe til brugerens fysiske vare.

Herunder er et eksempel på et brugerelement i databasen



Herunder er et eksempel på en vare i varer tabellen



Authentication data

Authentication

Users








Sign-in method

Templates

Usage

Search by email address, phone number, or user UID

Add user

Identifier	Providers	Created	Signed In	User UID ↑
ablat666@gmail.com		Apr 20, 2021	Apr 23, 2021	9Dax2akDfVkuPdUE3H9uV3DMu172
madenroost@gmail.com		Apr 20, 2021	Apr 20, 2021	5aGedn7u...51aPAMagocym...
silvest16@gmail.com		Apr 20, 2021	Apr 20, 2021	4DpG45...175BhR9u1000...V2
felixyn00@gmail.com		Apr 20, 2021	Apr 22, 2021	6CendgRy4Q8poc6a9ndHqm7...
tohus1@gmail.com		Mar 3, 2021	Apr 26, 2021	6a...5W3...JBU6...16mpcP3u3
ydmmareus@gmail.com		Apr 20, 2021	Apr 20, 2021	UNWEP...5U...6...5...
...@gmail.com		Apr 20, 2021	Apr 20, 2021	...71TbP82anZcB...

Rows per page: 50

1 - 7 of 7

Authentication bruger databasen er en function, som firebase også tilbyder til at holde styr på ens brugere. Det jeg bruger den til mest er at holde styr på hvilken bruger det er, som er logget ind på siden. Der får react appen et uid her fra, hver gang en bruger logger ind. Det kan så bruges til at finde en bruger i realtime databasen. Det kommer der mere om i implementationen af loginsystemet.

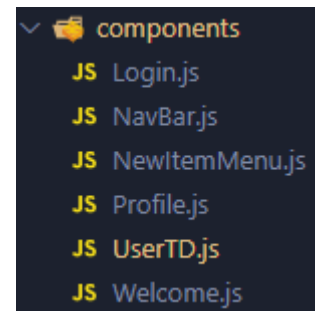
Implementation

Hele webappen er lavet ud fra en skabelon som bliver lavet med et bibliotek, create-react-app, som er installeret med node.js. Det man gør for at bruge det er at man kalder "node create-react-app *navnet på dit projekt*" og så laver den automatisk en ny mappe som er en template af en react app i den mappe ved navn *navnet på dit projekt.* Det skaber en meget simpel react app, hvor det meste af template koden bare er blevet fjernet her, da det ikke skulle bruges.

Derefter har jeg lavet en mappe inde i "src" mappen, som hedder components, hvor jeg holder alle mine komponenter er i udover App.js filen, da den er hovedfilen for webappen udover index.js. Create-react-app har derudover også selv lavet en css fil der hedder App.css og importeret den til App.js, så det skal man heller ikke tænke på. Herefter er bunden lavet.

I components mappen er der 6 komponent filer. De forskellige komponenter er delt op i separate filer, for at man kan få en pænere kode og man nemmere kan genbruge komponenterne i andre komponenter eller filer. Man kan for eksempel se i App.js hvor der i toppen er importeret 3 af dem.

```
import {Profile} from './components/Profile';  
import {Welcome} from './components/Welcome';  
import {NavBar} from './components/NavBar';
```



De tre komponenter der bliver importeret er med til at skabe 3 af sidens hovedkomponenter, som der er blevet beskrevet i design. NavBar komponentet er det komponent der laver navigationsbaren i toppen af siden, hvor Profile og Welcome bliver vist, henholdsvis i forhold til om brugeren er logget ind eller ej, i sidens hoved del.

App og Login

```
21  const database = firebase.database();
22  const auth = firebase.auth();
23
24
25  export function App() {
26
27    const [user] = useAuthState(auth);
28
29
30    return(<div id="app">
31
32      <NavBar user={user} auth={auth} />
33      <div id="main">
34
35        {
36          user
37        }
38        <Profile user={user} database={database} />
39        :
40        <Welcome/>
41      }
42
43      <div id="fridge-image-container">
44        <img id="fridge-image" src={"/images/Fridge-sticker-final.png"} alt="Fridge Logo" />
45      </div>
46
47    </div>
48  </div> );
49 }
```

Herover er koden for hoved delen af App.js. Det er den del der kommer efter alle import statements'ne.

Til at starte med bliver de to firebase funktioner instantieret ved hjælp af en config JavaScript fil, som man får fra sit projekt på firebase's konsol hjemmeside, der er importeret til App.js. Der kan man kalde to funktioner til "firebase," som er den variabel jeg har kalder min firebase app fra. Ved hjælp af firebase variabelen laver jeg to variable, database og auth, med en funktion der henholdsvis hedder det samme som variabel navnet. De to variable (som er konstanter, så man ikke kan ændre i selve variabelen, men kan kalde funktioner til den) er den måde man tilgår firebase databasen og firebase authentication.

App komponentet her er lavet som et funktions komponent, som er et af de 2 typer komponenter der er i react. I forhold til den anden type komponent, som der kommer noget om senere i rapporten, er at man bruger hooks til at lave forskellige ting, som kunne være at lave en state variabel, hvor man skal bruge useState() hook'en. Til Firebase's login system har de lavet deres egne react hooks, hvor useAuthState() bliver brugt på linje 27.

UseAuthState tager et parameter, som er vores auth variabel, som gør at "user" variabelen hele tiden er lig med den bruger der kommer fra firebase authentication. Det vil sige at hvis

brugeren ikke er logget ind er user lig med "null", men hvis den er logget ind, er den lig med et bruger objekt.

Efter user variablen er blevet lavet, kommer return funktionen af App komponentet, som er den der returnere JSX'en der skal kompileres. Et JSX element skal altid bestå af et ydre tag, som oftest er en div, det er det også her med div'en, som har id'et app. Inde i app div'en bliver der først kaldt NavBar komponentet, og den får attributterne user={user} og auth={auth}. Krølle paranteserne er JSX' måde at integrere variable og andre ting ind i html kode. Her er det user og auth variablerne der bliver givet. Derefter er der div'en med id'et main, som er der hoved delen af siden fra design afsnittet, hvor køleskabsbilledet også er inde i, da man så ved hjælp af css grid kan dele main op i de to områder, som planlagt i designet. De forskellige dele af html koden får id'er så det er nemmere at målrette css'en til de specifikke elementer, det vil også blive brugt en del i de andre komponenter, udover klasser som også bliver brugt.

Udover billedet, som der er i main div'en, er der også en krølleparantespar. Den har dog ikke bare en variabel inde i sig. Det der er inde i krølleparanteserne her er et form for if-statement, som man gør sig meget brug af i react komponenter. Formen på det er således: x ? y : z. Det betyder at hvis x er true, så gør den y som er efter spørgsmålstegnet, ellers, som er det samme som kolonnet, gør den z. Her tjekker den derfor om der er en bruger. Hvis der er det, returnere den Profile komponentet, ellers returnere den Welcome komponentet, som bare er en div med en velkomst besked. Måden user variablen bliver opdateret er fra login og signout komponentet der ligger i NavBar komponentet.

```
1  import firebase from '../firebase.js'
2
3  export function Login(props) {
4      const signInWithGoogle = () => {
5          const provider = new firebase.auth.GoogleAuthProvider();
6          props.auth.signInWithPopup(provider);
7      }
8      return(
9          <button className="sign-in" onClick={signInWithGoogle}>Login</button>
10     )
11 }
12
13
14 export function SignOut(props) {
15     return props.auth.currentUser && (
16         <li><button className="sign-out" onClick={() => props.auth.signOut()} >Sign Out</button></li>
17     )
18 }
```

Login og SignOut er begge to komponenter der ligner hinanden. De returnere begge en knap, som har en onClick funktion der enten logge personen ind eller ud. Signout har en rimelig simpel funktion hvor den bare tager den attribut den har fået fra props der hedder

auth, som er vores firebase authentication, og kalder `signOut()`, som logger personen ud og ændrer `useAuthState`'et. `SignOut` `onClick` laver en arrow funktion inde i krølleparanteserne.

I `Login`, laves der i modsætning til `SignOut` en separat funktion inde i komponentet som hedder `signInWithGoogle`. Det den gør, er at den laver en ny google auth provider ved hjælp af firebase auth, som der skal bruges for at man kan logge ind med sin google account. Derefter laver den så et pop op vindue på næste linje med funktionen `signInWithPopup()`, som tager en provider, der er vores google provider. Det pop op vindue klarer resten af login processen og ændrer `useAuthState`'et også.

Hver gang `AuthState`'et bliver ændret bliver den her eventhandler kaldt som er vist herunder.

Det den gør er at den laver en reference til `users` tabellen i databasen, hvor den derefter looper igennem alle bruger objekterne i tabellen med et `forEach` loop. Det den så derefter tjekker efter, er om `user.uid`, altså brugerens uid er det samme som et af uid'erne til brugerne i databasen. Hvis det så er i databasen er funktionen færdig efter loopet er kørt igennem. Ellers kører den ind i if-statementet for hvis den er falsk. Det push så et nyt objekt til `users` tabellen og sætter det samtidig til at være lig med de forskellige felter et bruger objekt skal have fra `user` objektet og et tomt objekt til `varene`.

```
59   auth.onAuthStateChanged(user => {  
60     if (user) {  
61       if (user !== null) {  
62         //path for user in db  
63         const usersRef = database.ref('users/');  
64         let inDb = false;  
65         //gets users  
66         usersRef.on('value', (snapshot) => {  
67           const users = snapshot.val();  
68           Object.values(users).forEach(checkingUser => {  
69             //checks for uid in db  
70             if (user.uid === checkingUser.uid) {  
71               inDb = true;  
72             }  
73           });  
74           //if user isn't in db, push user  
75           if (inDb === false) {  
76             usersRef.push().set({  
77               "uid": user.uid,  
78               "name": user.displayName,  
79               "email": user.email,  
80               "vare": {}  
81             });  
82             console.log('user added');  
83           }  
84         })  
85       }  
86     });  
87   });
```

Display af vare

Inde i profilen retuneres et komponent, som hedder `UserStorage`, som er det komponent der laver en html tabel, der viser alle brugerens vare, tilføj og fjern knapperne og søgefeltet.

Vis varene

Et react komponent har en livcyklus, hvor vi skal bruge det sted i den hvor at komponentet "mounter" til UI. Når det event sker bliver en funktion der hedder `componentDidMount()`

kaldt. Den defineres til at gøre en masse forskellige ting, en af de ting er at kalde en funktion der sætter en eventlister på databasen, der loader brugerens varene ind i en state variabel kaldt varer.

```
62   getUserKey() {
63     const usersRef = this.props.database.ref('users/');
64     usersRef.on('value', (snapshot) => {
65       const users = snapshot.val();
66       if(users) {
67         let i = 0;
68         Object.values(users).forEach(thisUser=>{
69           const key = Object.keys(users)[i];
70           if (thisUser.uid === this.props.user.uid) {
71             this.setState({userKey: key});
72           }
73           const userItemsRef = this.props.database.ref('users/' + key + '/varer');
74           userItemsRef.on('value', (snapshot) => {
75             const items = snapshot.val();
76             if(items) {
77               let itemsFormatted = [];
78               let i = 0;
79               Object.values(items).forEach(item=>{
80                 const key = Object.keys(items)[i];
81                 let itemFormatted = {
82                   key: key,
83                   date: item.date,
84                   name: item.name,
85                   amount: item.amount,
86                   picpath: "/images/" + item.barcode + ".jpg"
87                 };
88                 i++;
89                 if(itemFormatted.name.toLowerCase().includes(String(this.state.searchinput))) {
90                   itemsFormatted.push(itemFormatted);
91                 }
92               });
93               this.setState({varer: itemsFormatted});
94             });
95           }
96           i++;
97         });
98       }
99     });
100  }
```

Her over er den funktion og det er getUserKey(). Den laver igen en reference i databasen til users tabellen. Det gør den da vi først skal have fundet en det unikke id, som brugeren har i databasen, som her er kaldt key. Derfor loopes der igen igennem brugerne for at finde det. Hvis den så finder en bruger, hvor uid'erne passer sammen, går den videre og laver en ny reference. Det er ved hjælp af den key, som der lige er blevet fundet at den så laver en reference til brugerens vare på linje 73 hvor stien er 'users/*key*/varer', som er den specifikke brugers vare. Derefter laves der en eventlister på den tabel, der lytter efter en værdi. Hver gang det event sker, bliver der så retuneret et snapshot til komponentet. Det snapshot formateres så til en masse nye objekter af varene, så det er nemmere at omdanne dem til et JSX element senere. Det gør den ved at loope igennem alle objekterne i snapshot's værdi og derefter lave et nyt array af de nye objekter. Men hvert vare objekt

bliver kun tilføjet til itemsFormatted array'et, hvis navnet indeholder det der står i søgefeltet på siden, som bliver tjekket i if-statementet på linje 89 og 90. Derefter bruges der setState() til at updatere UserStorage komponentets state af varer til at være lig med itemsFormatted.

Nu når alle varene så er hentet sat ind i en state variabel, kan de så blive lavet til et JSX element.

```
225     {/* Rows of items */}
226     { this.state.varer.map(item =>
227       <tr key={item.key}>
228         <td className="image-display" >
229           <img src={item.picpath} alt={item.picpath}/>
230         </td>
231         <td>{item.name}</td>
232         <td className="ud-dato" ><ChangeDate onChange={this.changeDateHandler} item={item} /></td>
233         <td className="amount" ><ChangeAmount onChange={this.changeAmountHandler} item={item} /></td>
234       </tr>
235     ) }
```

Det sker på billedet herover og det hele foregår inde i krølleparanteser i tabellens tbody tag. Det der sker, er at der bruges map funktionen på komponentets state varer, som fungerer ligesom et forEach loop. Map giver hvert item som en parameter til en arrow funktion, der laver en ny tr(tabelrow) i tabellen. Det er så her hvor de forskellige ting fra et vare objekt bliver vist frem. Der er fx picpath, som blev lavet af varens barcode, som giver stigen til varens billede, så er der varens navn og den udløbsdato og mængden. Hvilket giver en række som vist på billedet under.



Skyr Vanilje Med Granola

29 / 04 / 2021



1 ↕

Noget andet der er hver at er at udløbsdatoen og mængden faktisk er to separate komponenter. Hvert komponent for komponentets eventhandler og varen, loopet er noget til, som atribut. Grunden til at de to felter er lavet som komponenter, er fordi man skal kunne ændre datoen og mængde ved at ændre inputet i det html input felt det er. Vist herunder.

```
241     const ChangeDate = props => {
242
243       const handleClick = (e) => {
244         if (props.onChange) {
245           props.onChange(props.item, e.target);
246         }
247       }
248
249       return (
250         <input id={props.item.key + "dato"} className="input-date" type="date" value={props.item.date} onChange={handleClick}></input>
251       )
252     }
```

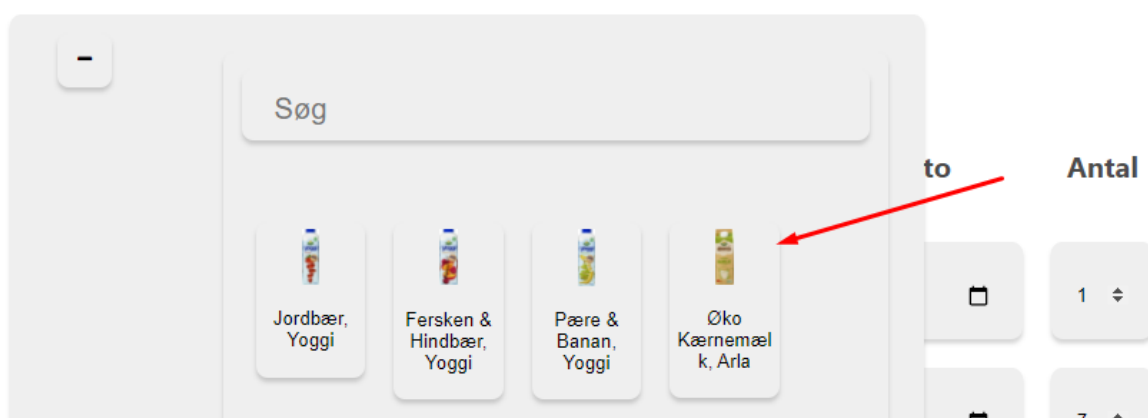

Her kan det ses at der retuneres et inputfelt i return(). Men da eventlisteners ikke kan give andet end eventet som parameter til eventhandleren, bliver man nødt til at lave et trick, som react heldigvis lader os gøre rimelig nemt. Der man gør er at som det kan ses på linje 243, laver man en funktion som man bruger til et slags mellem led i mellem eventet og den rigtige eventhandler. Den funktion checker om der er en eventhandler, her onChange, og derefter kalder den fra props.onChange med props.item og e.target. Det giver os muligheden for at skrive funktionen på billedet herunder.

```
102 //change the date of the item eventhandler
103 changeDateHandler = (item, element) => {
104   const userVarer = this.props.database.ref('users/' + this.state.userKey + '/varer/' + item.key)
105
106   userVarer.update({date: element.value});
107 }
```

Det er så eventhandleren. Den laver en reference til brugerens vare tabel og videre ud i den specifikke vare, ved hjælp af den key fra item parametret. Derefter bruger den en funktion til firebase databasen, update(), som opdaterer date feltet for den vare.

Det princip her, med at lave et nyt element for at kunne give nogle flere parametre, bliver brugt flere steder i appen. Det er fx i searchbaren for at kunne give value videre og ændre statet i komponentet, eller i når man skal tilføje en ny vare fra menuen, og trykker på den knap der viser varen, så skal man også have selve vare objektet og brugeren med ind i eventhandleren, for at man kan tilføje de varen til den bruger. Vist herunder, hvor hvis man fx trykker på den vare, som er en knap, tilføjer den den vare til brugeren.

Hej **Jack Oulund**



Test

Måden jeg har udviklet webappen på har været i stil med en iterativ udviklings metode. Hvor den iterative imodsætning til vandfaldsmodellen lader en fx udvikle en funktion, derefter implementere den, og så teste den, hvor hvis den så ikke fungerer, kan man gå tilbage i processen. Derfor er den iterative metode, som jeg har brugt, mere 'trial-n-error' udvikling end vandfaldsmodellen.

Derudover for at få nogle lidt andre syn på hjemmesiden, har jeg også ladet nogle testpersoner prøve den, og hørt nogle af deres meninger. Dog var de fleste af de test dog mens der stadig var en del af funktionerne, der ikke helt fungerede endnu, men det vidste jeg godt, men det var mest sådan nogle ting der var kritik på, ellers var de fleste rigtig overvældede over hvor godt det hele hang sammen.

Konklusion

Alt i alt har er webappen mundet rigtig godt ud og over al forventning. Den har formået at opfylde alle de krav der var i kravspecifikationen, da den viser varene for hver individuelle bruger der er logget ind, hvor personen også kan tilføje nye varer, som var hoved målet. Derudover kan man også ændre udløbsdatoen og mængden af hver vare og alt dataen ligger i enten firebase's realtime database eller deres authentication system. Designet er også end med at blive rigtig simpelt og flot, uden at der er alt muligt der forstyrrer og trækker opmærksomheden væk fra de vigtigste ting.

Dog er der nogle ting man kunne have implementeret, som ville kunne have tilføjet mere værdi til hjemmesiden. Der er i hvert fald 2 ting, som er en barcode scanner til søgefeltet både i hovedvisningen af varene og i menúen hvor man vælger nye vare, men også en funktion så brugerne selv har mulighed for at inputte nye vare til hoved tabellen for varene i databasen. Det ville gøre det meget nemmere, hvis man i fremtiden ville opscalere mængden af vare, som brugerne kan vælge i mellem, for hvis det mangler den vare, som de har, kan de bare tilføje den.

References

(2021, 5 3). Retrieved from Madværkstedet: <https://xn--madvrkstedet-9cb.dk/>

Damhus, M., Buch, J., & Husum, E. (3. 5 2021). *Systime*. Hentet fra Informatik: <https://informatik.systime.dk/index.php?id=1124&L=0>

Firebase Tutorial. (2021, 5 3). Retrieved from tutorialspoint: <https://www.tutorialspoint.com/firebase/index.htm>

React. (2021, 5 3). Retrieved from <https://reactjs.org/>

WIERUCH, R. (2021, 5 3). *How to use Firebase Realtime Database in React*. Retrieved from robinwieruch: <https://www.robinwieruch.de/react-firebase-realtime-database>