EKSAMENSPROJEKT - EG

PBA I WEBUDVIKLING 2. SEMESTER 2022

GRUPPE 2: DENNIS, FREJA, MORTEN OG RENÈ

SILVAN

Indholdsfortegnelse

Indledning

EG A/S er en virksomhed, som udvikler brancheløsninger i form af software til blandt andet trælaster og byggemarkeder. EG A/S er i gang med at udvikle en mobil app, som skal bruges som selvbetjening til forbrugere, når de handler i byggemarkeder. I den anledning har de fokus på en feature, som skal huske en kunde på at købe alle de ting med som han/hun skal bruge til sit projekt.
Hertil har de brug for nogle refinements af et par emner som fx ER-diagrammer og frontend design.
I den anledning har vi valgt at arbejde med følgende problemformulering:

Problemformulering:

Hvordan kan vi skabe en webapplikation, som hjælper kunder i byggemarkeder med at huske produkter, som de skal bruge til deres projekt.

Metodeovervejelser

Tidsplan
Vi har valgt at lave en tidsplan, for at danne et overblik over projektet. Her har vi delt de forskellige opgaver, som vi skal have lavet, ud på de uger, som vi har til at lave projektet.

Postman
Postman er et software der tillader enkel og fleksibel testning af HTTP methods hvor CRUD indgår som en standardiseret (*Postman API Platform | Sign Up for Free* n.d.).

Node.js
Som beskrevet i projektbeskrivelsen bruger vi Node.js til udvikling af vores produkt. Vi gør også brug af moduler i form af Express framework med Ejs som view engine.

MySQL
MySQL bruger vi til at skabe forbindelse samt håndtering af vores relationsdatabase. Derudover er MySQL en database management system til behandling af vores data.

ER-diagram
Vi vil udarbejde et ER-diagram for at danne et overblik over de elementer vi skal have med i vores database. Dette gælder både hvilke tabeller og tilhørende kolonner, som vi skal have oprettet, som vi skal bruge senere til projektet. Dette er en nødvendighed også i forhold til arbejdet med relationsdatabaser.

Wireframe/prototype
Vi anvender wireframes til at skitsere, hvordan vi vil opbygge vores frontend-design. Herefter vil vi ud fra vores wireframes udarbejde en prototype, hvor vi får farver og billeder på, så vi kan se, hvordan det endelige design skal se ud.

Passport
Vores loginsystem er bygget op med Passport (local strategy), som er et middleware til brug i Node.js til håndtering af autentifikation (Hanson 2022).

Research

Design

Til at udarbejde vores design, har vi lavet noget research på, hvad andre virksomheders apps indeholder. Her har vi blandt andet fundet inspiration på følgende sider:

- Bilka hjemmeside
- Bauhaus hjemmeside
- Bodylab hjemmeside
- Silvans hjemmeside

Idet vi skal udvikle nogle forslag til en app til et byggemarked, har vi især lavet noget research på disse typers hjemmesider.
Da vi ikke har fået at vide, hvilket byggemarked vi specifikt skal designe til, har vi valgt at tage udgangspunkt i Silvan. Derfor har vi været inde på Silvans hjemmeside, for at finde frem til, hvilke farver og typografier de bruger, da vi vil anvende præcis det samme. For at finde de nøjagtige typografier var det nødvendigt at bruge browserens værktøjer til at inspicere disse.

Grunden til at vi valgte Silvan, er fordi, vi også har lavet noget research i en af deres fysiske butikker. Her var vi nede for at spørge omkring deres varelokationer. Her fandt vi frem til, at deres forskellige lokationer har et lokationsnummer, som svarer til den hylde, som et bestemt antal varer er placeret. Dertil har hver vare et sekvensnummer, som svarer til den placering de har på selve lokationen. Det er også nødvendigt at pointere at disse lokationer er forskellige fra butik til butik.
Disse informationer har givet os et bedre indblik i, hvordan de håndterer deres varer. Dette kan vi bruge til at udarbejde/forbedre vores ER-diagram.

Programmering

NPM

For at udvikle den bedste mulige løsning til vores applikation, benytter vi os af diverse NPM-moduler (*Npm* n.d.) til at understøtte vores arbejde, med diverse funktionaliteter. For at bruge disse er det nødvendigt at bruge npm install for at installere disse. Alle moduler ligger i vores projekt under filen package-lock.json hvor versionen også bliver opgivet.

MySQL2

Til at skabe en forbindelse mellem applikationen og vores relationsdatabase, bruger vi MySQL2 (*Mysql2* 2021). I vores tilfælde er det ikke nok bare at "require(mysql2)" da et undlader essentielle funktionaliteter, som for eksempel brugen af async/await. Derfor requires mysql således: "require(mysql2/promise)". Brugen af async/await gør det muligt at strukturere asynkrone, ikke-blokerende funktioner, til brug i diverse queries.
Til MySQL2 modulet hører der også prepared statements, som også virker som et ekstra lag af sikkerhed, da et prepared statement er en genbrugelig SQL-forespørgsel, som tvinger os til at skrive SQL-kommandoen samt de brugerleverede data separat, hvilket forhindrer SQL injections.

Passport

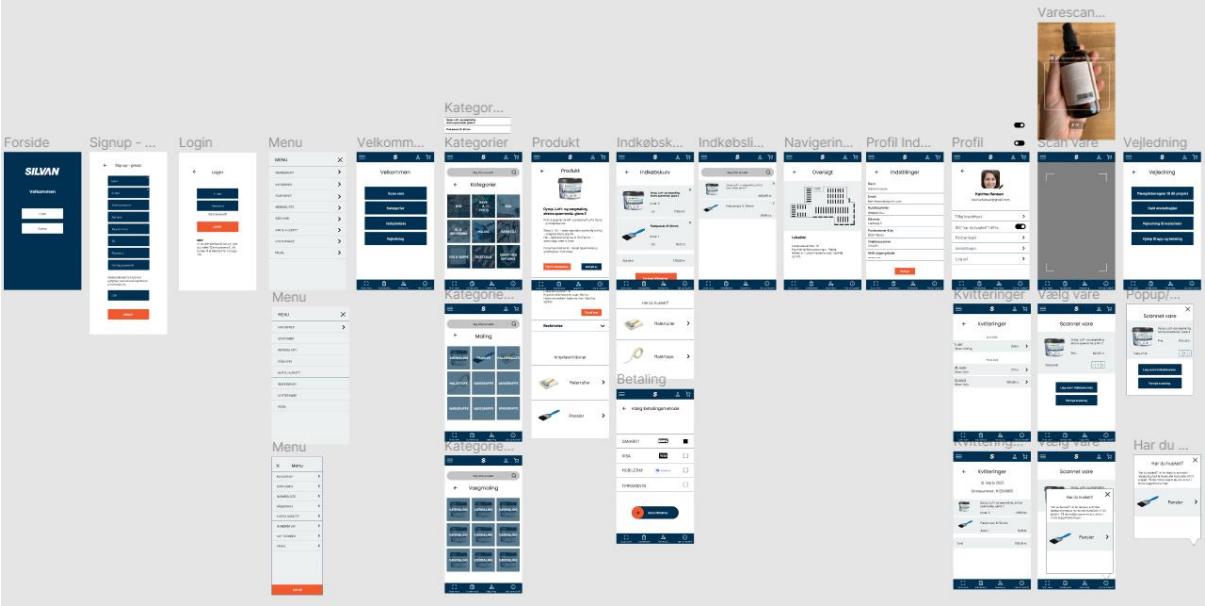
Når det kommer til autentifikation, er der forskellige måder det kan gøres på, for eksempel JSON-web tokens, OAuth eller sessions.
Sessionsbaseret autentificering er en af de ældste metoder, men bestemt ikke forældet.
Sessionsbaseret autentificering er "kernen" i passport-local strategy (Hanson 2022), som vi benytter os af. Denne metode er server-sided, hvilket betyder, at vores express-applikation og database arbejder sammen for at bevare den aktuelle autentificeringsstatus for hver bruger, der besøger vores applikation.

Analyse

Design

Prototype

Vi har valgt at lave en prototype i Figma (*Figma: The Collaborative Interface Design Tool.* n.d.) for at få et overblik over, hvordan vores design skulle se ud. På den måde havde vi noget at kigge efter, når vi skulle kode vores sider til applikationen.



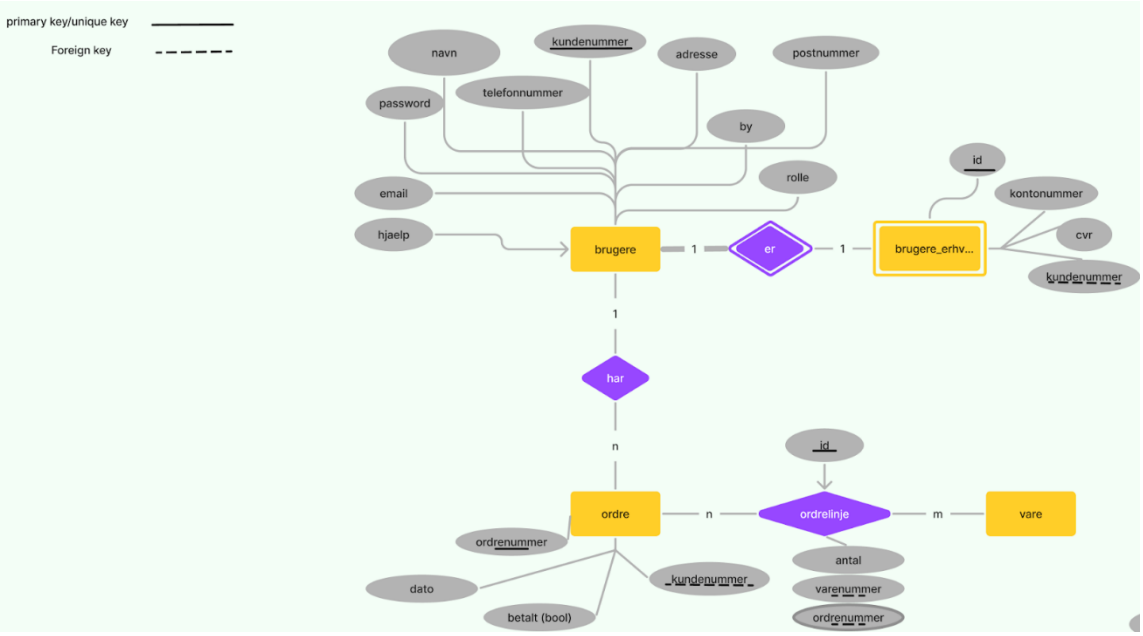
Programmering

Loginsystem

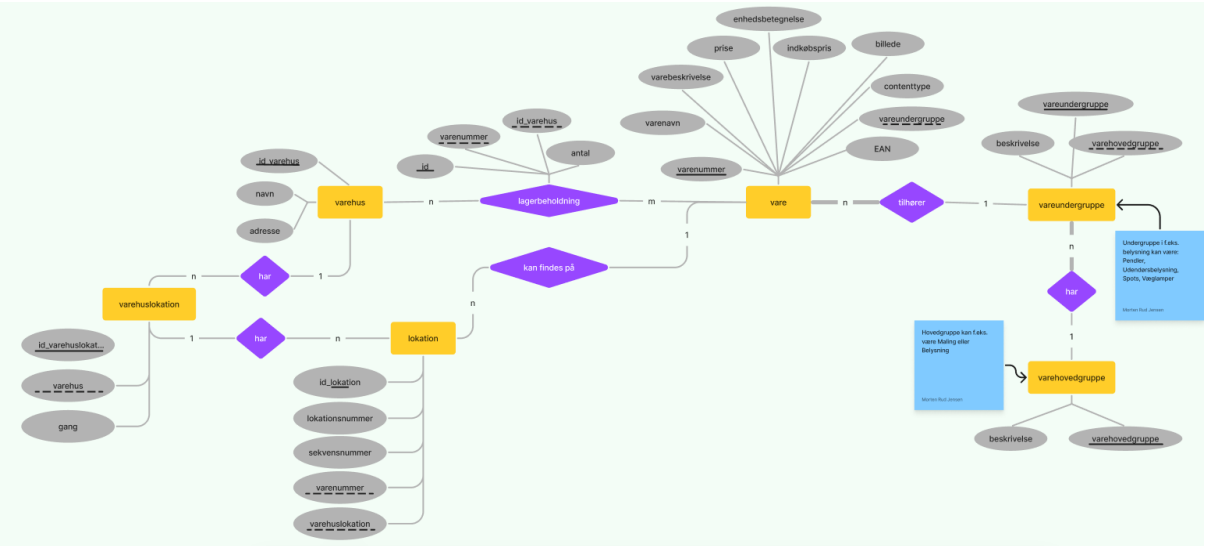
Vi har valgt at lave et loginsystem, da der kommer til at være nogle features i vores app, som alle ikke skal have adgang til, hvis ikke man har oprettet en bruger til appen. Derudover har vi med to forskellige kundesegmenter at gøre - nemlig "almindelige" kunder, som vi vælger at kalde de "private" kunder. Derudover har vi også de "professionelle", som vi vælger at kalde "erhverv" kunder.
Den primære forskel som skal være mellem en "privatkunde" og en "erhvervskunde" er mulighederne for betaling. Her skal det være muligt for en erhvervskunde at sætte et køb på en regning, som senere bliver sendt til den virksomhed, som er registreret. Denne mulighed skal en privatkunde ikke have, idet de skal betale med det samme.

ER-diagram

For at lave vores database startede vi med at lave et ER-diagram. Her startede vi med at lave brugerdelen idet vi vil lave et loginsystem. Her har vi angivet en "brugere" entity som indeholder alle de attributter som er fælles for privat- og erhvervskunder. Hertil har vi tilføjet et kundenummer på begge entities. Dette kundenummer skal vi bruge til at læse på flere af tabellerne på én gang, hvor vi kan finde frem til, om en kunde er privat eller erhverv.



Herefter lavede vi ER-diagrammet over resten, hvor "vare" fra ovenstående del er den samme entity som "vare" i delen nedenfor.



Da billeder kan være svære at se, har vi vedlagt ER-diagrammet i filen "ER-diagram.pdf"
De største udfordringer er i forhold til at angive en varerelation imellem de forskellige varer. Her vil vi tage udgangspunkt i kategorier, hvor vi bliver nødt til at dele varerne op i både hovedgrupper og undergrupper.

Database

Som man kan se i ER-diagrammet, så tager vi udgangspunkt i, at der er en samlet user-tabel, hvor alle de praktiske oplysninger kommer til at være. Hertil har vi en erhverv-tabel, som skal indeholde CVR og kontonummer, idet det kun er erhvervskunder som skal have disse oplysninger.
Vi har valgt at fokusere på, at man kun kan have adgang til én erhvervskonto. Dette betyder at man fx ikke kan tilknytte sig som privat til en erhvervskonto.
Når en bruger opretter sig, vil vi automatisk oprette et kundenummer til brugeren. Kundenummeret vil vi have til at auto incremente, men et kundenummer skal være 7 cifre, så derfor vil vi teste, om det er muligt at starte auto increment fra 1000000. Ved en manuel indsættelse af kundenummeret og herefter automatisk indsættelse bliver resultatet som forventet 1000001.

Konstruktion

Guide

Vi har gennem projektet formået at oprette et projekt med en relationsdatabasen bestående af et express framework. For at kunne bruge projektet optimalt er der lagt vægt på en brugermanual, som er med til at skabe lethed gennem hele processen. Denne brugermanual består af en fil i repoet "README.txt" fil som skridt for skridt vejleder brugen heraf.

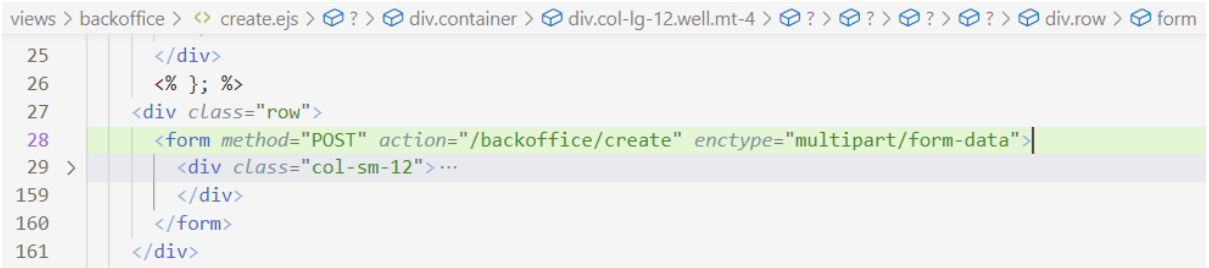
MVC

Vores produkt bliver bygget i et Model–view–controller (MVC) design pattern, som er med til at sikre opdeling af projektet. Målet er at kunne arbejde på tværs af de tre mønstre på en nem og effektiv måde. Modellen består af databaseforbindelser hvor controlleren holder på både forbindelsen mellem modellen samt view. I controlleren ligger routeren som er med til give en "rute" på baggrund af den HTTP request vi får fra view. Viewet består grundlæggende af HTML, CSS samt JavaScript.

Upload billede til database

For at få et billede eller en fil uploaded til serveren, har vi brugt et NPM-modul kaldet Multer (*Multer* 2021).
Det er en middleware, som ved submit af en form kan tilføje en eller flere filer til request-objektet inden det sendes videre i behandlingen af post-handlingen. For at bruge det er det nødvendigt at lave en storage variabel. Her har vi valgt blot at uploade til memory, i stedet for et directory på serveren.

Dernæst skal man sørge for at submit-formen har en bestemt enctype property, da det ellers ikke vil fungere. Upload.single('billede') bestemmer hvor fra filen skal hentes fra formen. 'billede' refererer til <input type="file" name="billede">



```
models > JS products.js > <unknown> > createAProduct
91   // create a product.
92   async createAProduct(req, res) {
93     try {
94       let {
95         varenummer,
96         varenavn,
97         varebeskrivelse,
98         pris,
99         enhedsbetegnelse,
100        indkøbspris,
101        contenttype,
102        EAN,
103        vareundergruppe,
104      } = req.body;
105      let billede = req.file.buffer;
```

Til sidst er det simpelt at inkludere i sin model, når det skal indsættes i databasen. Hvor der skal indsættes en BLOB, defineres blot en variabel med indholdet af req.file.buffer. Denne variabel bruges i SQL querien som en del af vores prepared statement på de følgende linjer.
I tilfælde af at flere filer var uploaded på samme tid, ville det være req.files[i] der skulle bruges.
Selve eksekveringen af ovenstående var let, men først efter mange timers hovedbrud over hvordan filen skulle konverteres til binær. Meget hjælp blev søgt efter, for til sidst blot at prøve at sætte bufferen direkte ind i databasen. Ingen konvertering var nødvendig i sidste ende.

Display billede fra database

Da vi i vores database har en vare-tabel med en billede-kolonne, vil vi gerne have displayet dette billede, når en vare skal vises frem.
For at gør dette laver vi et rest-api i routeren, hvor man skal angive et varenummer i URL'en, for at se varen.

```
router.route("/produkt/:varenummer").get(indexController.produkt);
```

Inde i controlleren har vi en funktion, hvor vi kalder en funktion inde fra modellen.
Inde i getprodukt-funktionen i modellen, laver vi et prepared statement, hvor varenummeret bliver hentet fra URL'en via req.params og sendes med. Hertil henter vi med en SQL-kommando alle kolonner fra databasen, som har det angivne varenummer.
Resultatet bliver hentet i controller-funktionen, hvor vi sender et objekt med dataene med.

```
exports.produkt = async (req, res) => {
  try {
    let vare = await model.getproduct(req,res);
    res.render("produkt", {
      title: "Produktbeskrivelse",
      title_bar: "Produkt",
      arrrow_back: "href=" + "/vaegmaling",
      vare: vare[0][0],
    });
  } catch (e) {
    console.log(e);
  }
};
```

```
async getproduct(req, res) {
  try {
    const { varenummer } = req.params;
    let sql = `SELECT * FROM vare WHERE varenummer = ?`;
    let varenummerId = [varenummer];
    let row = await pool.query(sql, varenummerId);
    return row;
  } catch (e) {
    console.error(e.message);
  }
},
```

Objektet med dataene bruger vi i en view, hvor vi skal angive "source" til vores <img>-tag.
Hertil fandt vi frem til, at man skal først angive en type data (V 2019). Denne datatype angiver vi allerede i databasen, så vi har denne oplysning via objektet. Herefter angives "base64", fordi vi bruger det til næste step, hvor selve billedet skal angives, som vi henter fra objektet. Værdien til billedet er en "buffer", hvilket vil sige, at vi skal konvertere det, så det kan printes på skærmen. Dette gøres med "toString()"-funktionen, hvor vi angiver "base64" indeni. Base64 er en metode som understøttes af Node.js, som konverterer binary data til tekst (Pöhls 2021).

```
<img src="data:<%= vare.contenttype %>;base64,<%= vare.billede.toString('base64') %>" alt="malerbøtte">
```

Loginsystem med Passport

Passport har over 500 autentifikations strategier som kan bruges i node applikationer, for eksempel passsport-amazon, som lader en bruge autentificere via. deres amazon login oplysninger. Vi bruger passport-local, som autentificerer username/passwords "lokalt", hvilket betyder lokalt i forhold til appen (databasen), og ikke lokalt på brugerens side (computeren) (Hanson 2022).
Herunder beskrives hvordan passport fungerer i vores applikation:

1. En bruger besøger vores loginside og forsøger at logge ind. Når der trykkes "login", sendes der en POST request til vores /login route, som bruger passport.authenticate() middleware.

```
router
  .route("/login")
  .get(indexController.login)
  .post(
    // Passport settings
    passport.authenticate("local-login", {
      successRedirect: "/profile", // Redirect on success
      failureRedirect: "/login", // Redirect back to login if error
      failureFlash: true, // Allow flash messages
    }),
    indexController.loginSuccess
  );
```

2. Passport vil nu validere om de korrekte oplysninger er indtastet, alt efter udfaldet kan det ses på billedet hvad der vil ske.

3. Hvis brugeren valideres med success så bliver user tilknyttet den nuværende session -> passport.authenticate() returnere user objektet der blev valideret, derudover knytter den req.session.passport property til req.session objektet, serializer user med passport.serializeUser() og tilknytter så den "serialized" user (ID'et af useren) til req.session.passport.user property, til sidst tilknyttes det hele til req.user.

4. For hver ny request som user foretager sig, tilknyttes deres session cookie requesten, som efterfølgende bliver verificeret af passport. Hvis verifikationen er en success, responder serveren med med den requested route/data.

Man kan bruge req.user objektet på samtlige routes, så længe useren er autentificeret. I vores app bruger vi det til at trække data om en specifik user fra databasen e.g navn, kundenummer.

"Har du husket" funktion

En af de vigtige opgaver som vi skulle lave i projektet, var at lave en "har du husket"-funktion, som handler om at vise relaterede varer, som passer til de varer som en bruger skal bruge til deres projekt. Denne funktion har vi blandt andet anvendt på vores "produkt"-sider, hvor vi vil vise de produkter som er relateret til den angivne vare.
Inde i modellen har vi oprettet en funktion, som er delt op i tre dele:

```
async getAssociatedProducts(req, res) {
  try {
    //Først henter vi undergruppe fra produktet
    const { varenummer } = req.params;
    let sqlVareundergruppe = `SELECT vareundergruppe FROM vare WHERE varenummer = ?`;
    let varenummerId = [varenummer];
    let row = await pool.query(sqlVareundergruppe, varenummerId);

    //Dernæst bruger vi undergruppen til at finde hovedgruppen
    let sqlVarehovedgruppe = `SELECT vareundergruppe, varehovedgruppe
        FROM vareundergruppe
        WHERE vareundergruppe = ?`;
    let vareundergruppe = [row[0][0].vareundergruppe];
    row = await pool.query(sqlVarehovedgruppe, vareundergruppe);

      //Til sidst finder vi alle undergrupper relateret til hovedgruppen, minus den valgte undergruppe
      let sqlAnbefalet = `SELECT * FROM vareundergruppe WHERE varehovedgruppe = ? AND vareundergruppe != ? ORDER BY rand() LIMIT 2`;
      let vareParams = [row[0][0].varehovedgruppe, row[0][0].vareundergruppe];
      row = await pool.query(sqlAnbefalet, vareParams);

    console.log(row[0]);

    return row;
  } catch (e) {
    console.error(e.message);
  }
},
```

Første del:
Først henter vi "vareundergruppe" fra vare-tabellen, i et prepared statement, på det varenummer som er angivet i URL'en. Denne henter vi med req.params.

Anden del:
Anden del består i at hente både "vareundergruppe" og "varehovedgruppe" fra vores vareundergruppe-tabellen. Vi læser på den "vareundergruppe" som vi fik fra første del. På den måde kan vi finde frem til, hvilken hovedgruppe den angivne vare tilhører.

Tredje del:
Til sidst vælger vi alle vareundergrupper, som har samme varehovedgruppe, som vi fik fra anden del. Samtidig angiver vi, at den ikke skal vælge den vareundergruppe, som den angivne vare har. Dette gør vi for at "har du husket"-funktionen ikke skal vise den samme type vare, men kun de relaterede varer. Til slut angiver vi at den skal sortere random imellem vareundergrupperne, og at den kun skal vælge 2 ud fra resultatet.

Inde i vores view har vi sendt objektet, med de relaterede varer, med. Hertil angiver vi en "foreach", hvor vi looper igennem de relaterede varer. Til hver relateret vare henter vi billedet og beskrivelsen, som vi displayer.

```
<% anbefalet.forEach(function(anbefalet) { %>

    <% if(anbefalet.vareundergruppe == '3640') { %>
    <a href="/vaegmaling">
    <% } else { %>
    <a href="/maling">
    <% } %>

    <div class="harduhusket_item harduhusket_item1">
            <div class="harduhusket_item_img">
                <img src="data:<%= anbefalet.contenttype %>;base64,<%= anbefalet.billede.toString('base64') %>" alt="<%= anbefalet.beskrivelse %>">
            </div>
            <div class="harduhusket_item_text">
                <p><%= anbefalet.beskrivelse %></p>
            </div>
            <div class="harduhusket_item_arrow">
                <img src="../public/images/arrow_right2.png" alt="pil">
            </div>
        </div>
    </a>

<% }); %>
```

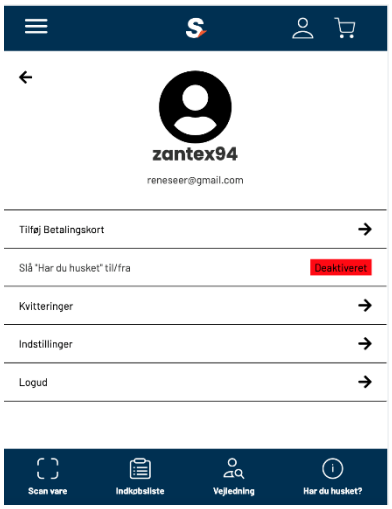
"Har du husket" Popup

I forlængelse af ovenstående har vi en popup, som automatisk kommer frem efter en timeout på x antal millisekunder, når man scanner en vare. Indholdet af denne er det samme som ovenstående liste, med det formål at minde brugeren om, at huske de tilhørende produkter. Denne popup kan lukkes igen via krydset, eller den knap, som popuppen kommer frem fra (i footer baren). Popuppen ligger som et view i footer baren, men er skjult, og funktionaliteten herunder viser den blot via en class toggle af hhv. "display: block" og "display: none".

```
10  function popup_timer_start() {
11    if (userHelp == 1) {
12      if (document.getElementsByClassName("kurv_item_1")[0]) {
13        timeout = setTimeout(open_popup, 3000);
14      }
15    }
16  }
17  function popup_timer_end() {
18    clearTimeout(timeout);
19    close_popup();
20  }
```

Slå "Har du husket" til/fra funktion

En vigtig del af projektet er at kunne tilbyde en brugere muligheden for at aktiver/deaktivere funktionen "Har du husket" i profil indstillinger.



Når man opretter sig som bruger har man mulighed for to ting: privat eller erhverv. Når en bruger opretter sig som erhverv bliver hjælpen automatisk slået fra, da vi her har at gøre med professionelle. Derimod hvis vi har en privat så er hjælpen automatisk det modsatte, da private ikke er professionelle, og dermed muligvis har brug for at få vist relaterede varer. Begge kan til enhver tid tænde/slukke for hjælpen, så professionelle har også mulighed for at gøre brug af funktionen.

I vores brugere tabel (se ER-diagram) har vi en attribut "hjaelp" som holder på en boolean. Denne værdi er en tænd/sluk mekanisme som bliver sendt med objektet user for alle sider i appen. Dette objekt bruger vi til at kontrollere om brugeren skal få vist hjælpen eller ikke:

```
<% if (user.hjaelp == 1) { %>

<%- include('anbefalet_liste.ejs', {anbefalet: anbefalet}); -%>
<% } %>
```

Illustration af produkt.ejs

Ovenfor er et eksempel hvor der genereres en anbefalet_liste.ejs hvor objektet "anbefalet" bliver sendt med ind. Før dette tjekker vi user-objektet for at redegøre for, om brugeren har behov for en anbefalet liste. I tilfælde af at funktionen er sandt får brugeren vist alle anbefalinger.

Alle sider bliver tjekket på bruger-objektet, og det er her nemt at tilgå disse.
Dog fandt vi ud af, at man nogen gange bliver nødt til at sende en hidden input med værdien ind såfremt denne skal bruges i forbindelse med en javascript fil. Herefter bruger vi DOM til at hente den givne værdi som vist:

```
<input id="user_help" value="<%= user.hjaelp %>" hidden>
```

illustration af hidden button i scanvare.ejs.

```
let scanArea = document.getElementsByClassName("scan-border")[0];
let modal = document.getElementById('myModal');
let closeModal = document.getElementsByClassName("close")[0];
let userHelp = document.getElementById("user_help").value;
let timeout;

function popup_timer_start() {
    if(userHelp == 1){
        if (document.getElementsByClassName("kurv_item_1")[0]) {
            timeout = setTimeout(open_popup, 3000);
        }
    }


}
```

illustration af scan.js javascript fil hvor vi henter værdien via dom fra objektet user. Bemærk i filen scanvare.ejs har vi en reference til javascript filen, så de kender hinanden

Selve funktion er tænkt for at hjælpe brugere med at have en form for selvkontrol. I tilfælde af de ønsker hjælp, har de stadig mulighed for at fjerne denne direkte hvis en given vare er utilpasset.

Backoffice

Vores backoffice er tiltænkt som et uafhængigt system (ikke implementeret i appen) hvor administratorer kan udføre vedligeholdelse af både brugere, varer osv. Dog har vi afgrænset os til kun at fokusere på vare delen til demonstration af diverse CRUD-operationer.

I princippet kunne man lave et helt nyt node-projekt ved siden af, til at udvikle backoffice, men vi har valgt at gøre det i en mappe for sig (backoffice), inde i vores views. Derudover har vi lagt alt logikken ind i backofficeController.js. I forhold til routes kunne man også inde i app.js oprette det således:

```
const backofficeRouter = require("./routes/backoffice");
app.use("/backoffice", backofficeRouter);
```

men har forholdt os til routes/index.js.

Den visuelle del af backoffice er alt sammen lavet med bootstrap v5.0 (*Introduction · Bootstrap v5.0* n.d.). Det sparede en masse tid at kunne importere diverse elementer som tables og buttons direkte, dertil kommer også javascript der bruges i popupbeskeder, som for eksempel vises efter man opretter en ny vare.

Vores backoffice er selvfølgelig dynamisk, i og med alt data der vises, er trukket fra databasen. På forsiden er en oversigt over samtlige produkter fra databasen. For hvert produkt har man mulighed for enten at slette eller opdatere det. For at udføre disse opgaver læser vi varens id (varenummer), hvorefter man kan manipulere med dem.

I toppen er der en navigationsbar som indeholder et søgefelt, der med en post request søger efter noget der indeholder det man skriver i søgefeltet. Til sidst kan man oprette en vare med et billede.

```
search = req.body.search;
let sql = "SELECT * FROM vare WHERE varenavn LIKE ?";
let row = await pool.query(sql, ["%" + search + "%"]);
return row;
```

Evaluering af proces

Gennem projektet har der været en rigtig god atmosfære på tværs af alle gruppemedlemmer. Vi har formået at iværksætte en prototype hvor alle medlemmer har været med ind over. Derudover har der været klarhed om, hvordan projektet skulle udvikles. Dertil skabte vi en tidsplan som gav os struktur over hvad der skulle laves i forhold til de givne krav.
Vi har også diskuteret en del om, hvilke designprincipper der skulle bruges til projektet, samt hvilke programmer til gennemførelse/opbygning af en prototype.
Vi har også brugt hinanden til sparring når dette var nødvendigt, og der har været en god dialog hele vejen igennem.
Gennem projektet har vi været inde over flere emner, som vi ikke har arbejdet med i undervisningen, og dermed tilegnet os ny viden.

Konklusion

Gennem projektet har vi med hjælp af Node.js og Express fået udviklet en webapplikation, som er tænkt til at skal benyttes af kunder i et byggemarked. Hertil kan vi konkludere at vi har lavet en "har du husket"-funktion, hvor brugerne får en liste over de produkter, som er relateret til den eller de varer, som brugere interagerer med. Dette er med til at hjælpe kunden med at huske alle de ting han/hun skal bruge til sit projekt.
Derudover kan vi konkludere at vi har implementeret en "backoffice" hvor det er muligt for administratoren fra et byggemarked at oprette, opdatere og slette varer fra databasen.

Referencer

Figma: The Collaborative Interface Design Tool. (n.d.) available from <https://www.figma.com/> [16 December 2021]

Hanson, J. (2022) *Passport* [online] available from <https://www.npmjs.com/package/passport> [18 May 2022]

Introduction · Bootstrap v5.0 (n.d.) available from <https://getbootstrap.com/docs/5.0/getting-started/introduction/> [24 May 2022]

Multer (2021) available from <https://www.npmjs.com/package/multer> [25 May 2022]

Mysql2 (2021) available from <https://www.npmjs.com/package/mysql2> [18 May 2022]

Npm (n.d.) available from <https://www.npmjs.com/> [25 May 2022]

Pöhls, M. (2021) *How to Base64 Encode/Decode a Value in Node.Js* [online] available from <https://futurestud.io/tutorials/how-to-base64-encode-decode-a-value-in-node-js> [18 May 2022]

Postman API Platform | Sign Up for Free (n.d.) available from <https://www.postman.com/> [25 May 2022]

V, A. (2019) 'How to Convert Buffer Object to Image in Javascript?' [20 December 2019] *Stack Overflow* [online] available from <https://stackoverflow.com/q/59430269> [18 May 2022]