

Fog-Carport 2. Semesterprojekt

Datamatiker - CPH Business Bornholm

Udarbejdet af:

- Patrick Nielsen - cph-pn157@cphbusiness.dk - github.com/patrickogn
- Mark Nielsen - cph-mn651@cphbusiness.dk - github.com/Markersej1234
- Marcus van der Weij - cph-mv222@cphbusiness.dk - github.com/Altant457
- Mikkel Molte Jensen - cph-mj931@cphbusiness.dk - github.com/mollerten
- Micki Koefoed - cph-mk378@cphbusiness.dk - github.com/CodingKof

Projektet blev påbegyndt d. 02/05-2022 og afleveret d. 31/05-2022

GitHub	https://github.com/Mollerten/Carport-Projekt
Demo	https://youtu.be/VgNjHR2YCLY
Pitch	https://youtu.be/7Gxka3sDFkw
Deployment	http://188.166.163.22:8080/carport

Indledning	4
Baggrund	4
Virksomheden/Forretningsforståelse	4
SWOT analyse:	4
Value proposition canvas:	5
Teknologivalg	6
Krav til projektet	6
Aktivitetsdiagram	7
As-is	7
To-be	8
User stories	9
Domæne model	10
EER diagram	11
User	11
Roof og wood_cladding	11
Request	11
Stock	12
Overvejelser	12
Navigationsdiagram	13
Admin adgang	13
Brugeradgang	13
Navigationsbar	13
Uden logon	14
Mock-ups	15
Valg af arkitektur	17
MVC-arkitektur	17
Front controller med command pattern	17
Facades	17
Brugeroplevelse	17
UI	17
UX	17
Særlige forhold	18
Exceptions	18
Data i session og request	19
Bruger inputs	19
Sikkerhed ved login	20
Status på implementation	20
CRUD	20

Bruger oplevelsen	20
Proces	21
Arbejdsgangen	21
Kanban	21
Scrum	22
Navigationsdiagram	22
Discord	23
Udvalgt kode	23
addCity	23
Opret request	24
Opret stock	25
Maps i Calculator	26
SVG tegning	27
Tests	28

Indledning

Carport projektet er et eksamensprojekt for 2. semester på datamatiker uddannelsen. Vi tager udgangspunkt i at læsere af denne rapport har som minimum en tilsvarende eller højere faglig baggrund. Projektet på ud på at lave en webløsning for et bolig- og designhus der forsøger at skille sig ud fra sine konkurrenter med et website der er i stand til at tage specifikke længde- og breddemål via brugerinputs for derefter at designe en unik carport der passer til kundens behov. programmet skal altså være i stand til at generere en stykliste over hvilke, samt hvor mange materialer der er behov for, for at bygge denne carport. Desuden skal programmet kunne generere en modeltegning over carporten for at give brugeren, samt admin en visuel forståelse over byggeprojektet.

Baggrund

Johannes Fog består af et bolig- og designhus og ni trælast- og byggecentre rundt omkring på sjælland. De leverer materialer til alle brofaste øer i Danmark og giver kvalitetsrådgivning om nødvendigt. én af deres services er at de tilbyder standard carporte. Dette gør det nemt for kunden at bestille en carport, selv hvis kunden ikke har megen erfaring inden for tømrerfaget. Men hvis kunden ønsker det, er der også mulighed for at gå ind på Fogs hjemmeside og designe sin egen carport. Det er denne service der er opgaven at udvikle i projektet.

Virksomheden/Forretningsforståelse

SWOT analyse:

Vi mener at forretningen kunne have gavn af at få udvidet firmaet ved at ansætte deres egne tømrer. Hvis FOG har deres eget team af tømrer der kommer og installerer carporten efter transaktionen. Dette vil gøre processen nemmere som kunde, da kunden ikke selv skal finde et tømrer firma. Ved at ansætte tømrer vil man også kunne skabe en større omsætning og på samme tid udvide firmaet.

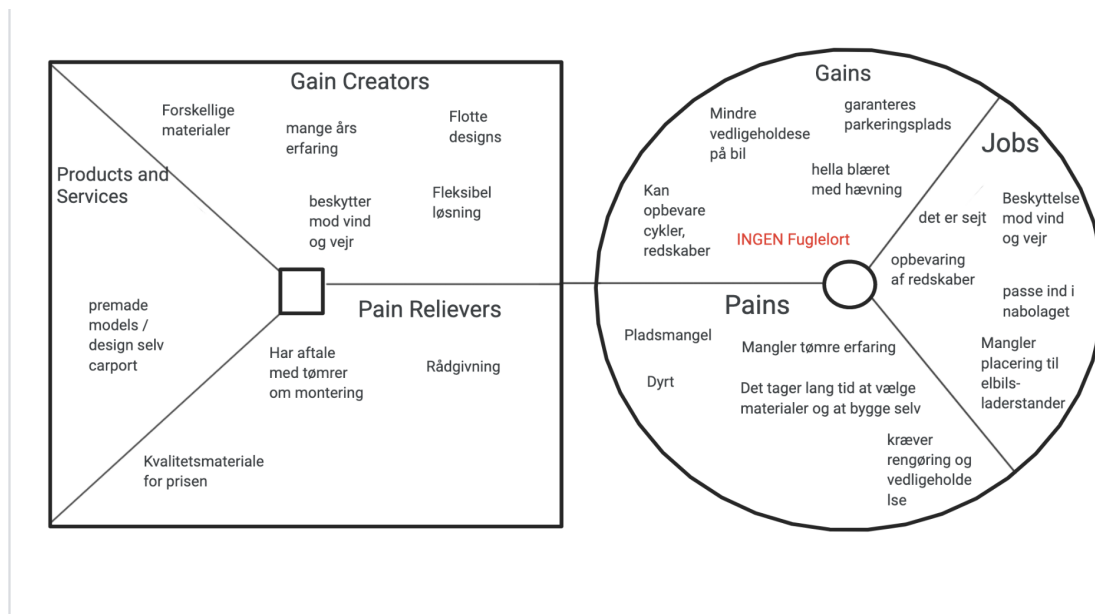
Vi har også et forslag der omhandler det tekniske aspekt, da man i videoen kan se at det program FOG bruger til at se carport-bestillingerne er meget forældet.

Hvis man opdaterede programmet til at ligne noget der var fra det her årtusinde.

Ved at gøre det kan man også tilføje muligheden for at kunne se 3d-modeller over den valgte carport. Hvilket man som køber vil give et ekstra lag af tiltro til at firmaet har styr på, hvad de laver. Det vil med sikkerhed også gøre det nemmere i bestillingsproceduren for medarbejderne og vil derfor spare medarbejderne i FOG en masse tid og unødigt arbejdskraft.

Vores sidste forslag er at lave en ordliste som kan oversætte fagudtrykne til en person der ikke er så erfaren. Da en ikke erfaren person måske havde fået den skøre tanke om at droppe tømrerne og selv at lave sin egen carport.

Value proposition canvas:



Man kan godt fornemme at FOG er et firma der har styr på, det de laver. Med deres produkt som er en carport designer kan de nemt finde materialerne frem og give dig en stykliste til en carport med kvalitetsmaterialer. De gains creators der er ved FOG er at firmaet har mange års erfaring og flotte carporte. Hvilket skaber gains som en carport der er attraktiv for ham der ikke har et sted at parkerer sin bil for vinterens udfordringer og gerne vil slippe for at skrabne sne og andre ting fra bilen.

Andre gains ved at have en carport er at man kan bruge den til at opbevare haveredskaber, cykler ovs. Hvis man vælger at få nogle tømrer til at lave carporten for en er der er ikke så mange pains da man egentligt kun selv skal holde carporten ren og nydelig. Vi kan også gå ud fra at FOG også har fået sit eget tømrer team fra vores SWOT analyse er der ikke så mange pains, da hele processen bare løber på skinner. Kunden bestiller en carport FOG modtager bestillingen og sender styklisten til tømrerne som bygger den hos kunden. Det kan ikke blive meget nemmere at bestille end carport som kunde end det.

For kunden er der også mange pain relievers da man ved at carporten vil være pengene værd og at man sagtens kan få rådgivning fra enten tømrer eller FOG selv.

Ved at købe en carport får kunden følelsesmæssigt ro i sjælen da bilen er i sikkerhed for dårligt vejr. Kunden får også en følelse af fællesskab, hvis der er andre i nabolaget der også ejer en carport.

Teknologivalg

- IntelliJ IDEA 2021.3.3 (Ultimate Edition)
- MySQL Workbench 8.0 CE
- MySQL 8.0.28
- Apache Tomcat 9.0.60
- Java 11
- Digital Ocean droplet 5.4.0
- Ubuntu 20.04
- Maven

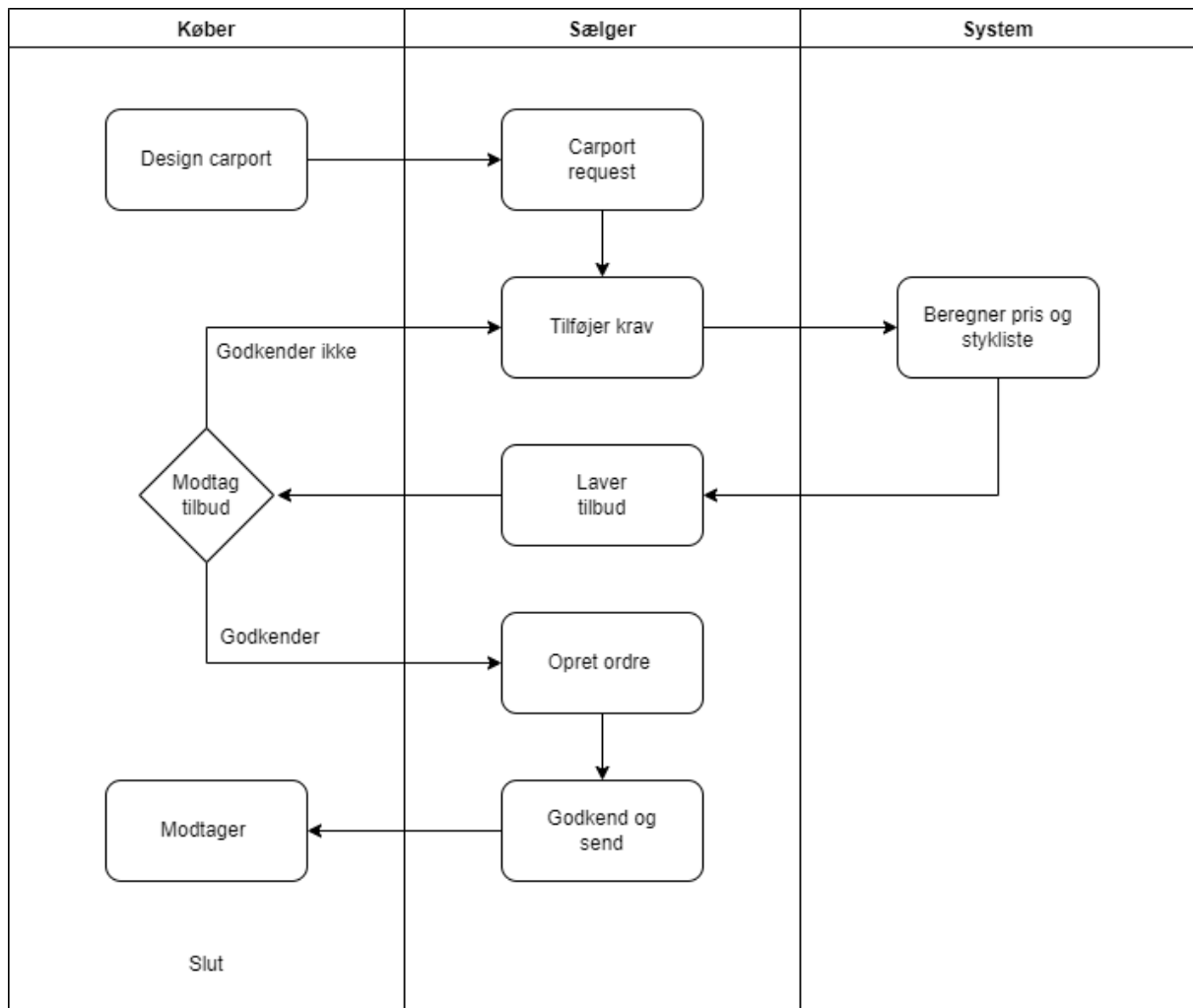
Krav til projektet

Som nævnt går projektet ud på at udvikle et website der gør det enkelt for Fogs kunder at designe en unik carport efter deres behov. Kravet til denne carport designer er som følger:

- Kunden skal kunne logge ind og designe sin carport med længde- og breddemål, diverse tag og beklædningsmaterialer, samt et eventuelt redskabsskur.
- Admin skal kunne se de forespørgsler der bliver sendt igennem systemet og generere en styklister ud fra disse forespørgsler
- Kundens informationer skal gemmes i en database.
- En modeltegning skal genereres ud fra kundens angivne mål.

Aktivitetsdiagram

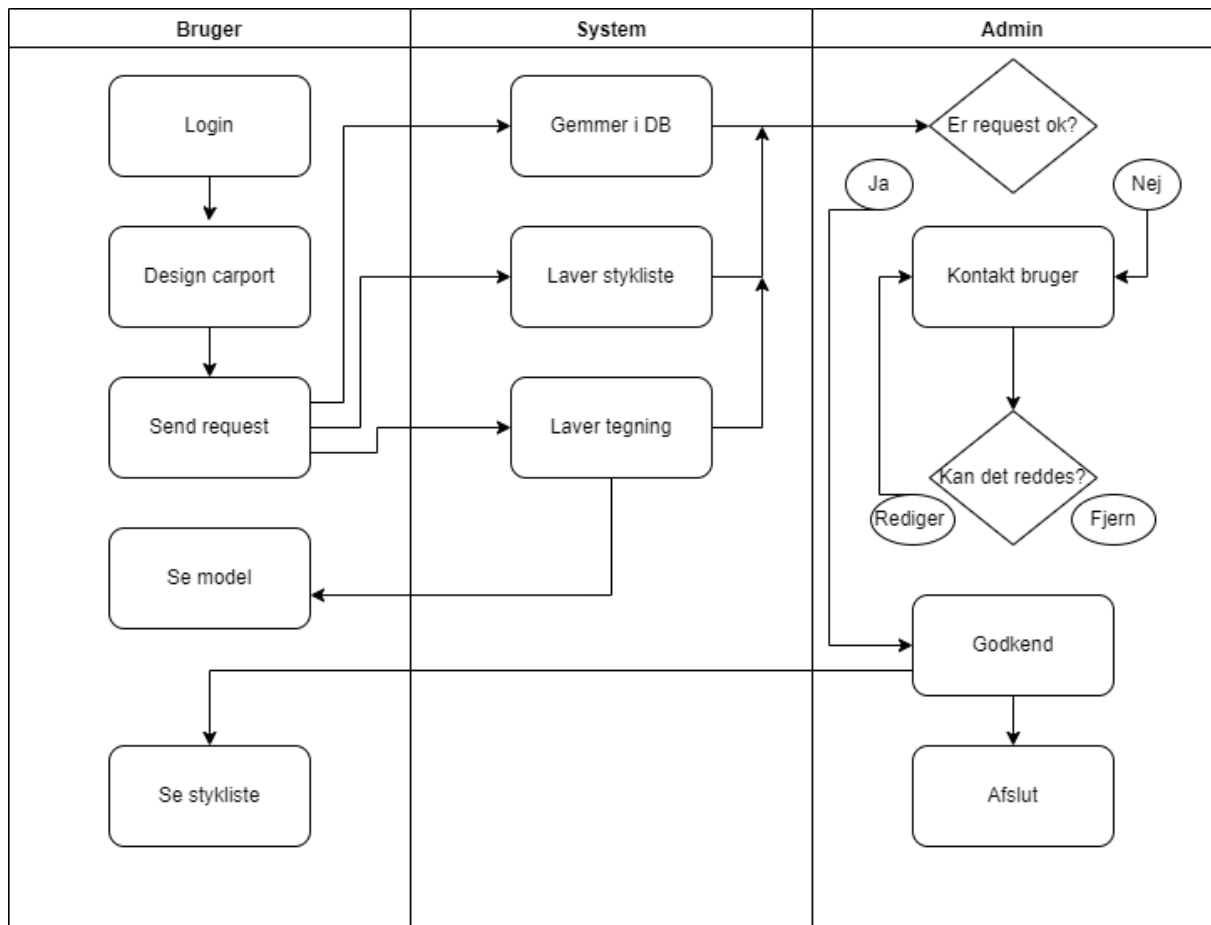
As-is



Kunden indtaster den ønskede carport størrelse. Sælger indtaster derefter ønskerne i det nuværende beregningssystem. Systemet returnerer en stykliste samt en pris, hvorefter sælger sammensætter tilbuddet. Deres nuværende system kan ikke blive opdateret med nye materialer, derfor indebærer dette, at sælger manuelt beregner fra gamle til nye materialer. Sælger fremsender til sidst det færdige tilbud til kunden.

Kunden vælger derefter at acceptere eller afvise tilbuddet. Accepteres tilbuddet opretter sælger en ordre og sender en kvittering til kunden, der kan afhente materialerne i henhold til styklisten.

To-be



// styklisten bliver først genereret når admin godkender

I billedet over kan man se forløbet af hvordan man bestiller en carport i vores system. Kunden logger ind og laver et request med de ønskede mål til en carport. Systemet behandler det, regner en stykliste ud og generere en tegning, hvorefter den sender tegningen til kunden så de kan få et lille preview. Admin modtager styklisten hvorefter vedkommende skal godkende eller redigere bestillingen. Såfremt bestillingen er godkendt af admin har kunden lov til at se styklisten.

User stories

Us-1: (medium)

Når brugeren logger ind, bliver han bedt om brugernavn og kodeord, derefter klikker brugeren på carport designer, brugeren kan herefter gå i gang med at vælge relevante oplysninger til carporten.

Us-2: (small)

Når brugeren klikker på log ind, bliver jeg bedt om at taste brugernavn og kodeord, jeg kan herefter logge ind som admin og gå til admin siden og se stock, request og opdater oplysninger.

Us-3: (medium)

Når admin har logget ind skal admin klikke på request, her kan jeg se alle forespørgslerne, jeg kan derefter redigere, godkende, slette og se en model over carporten.

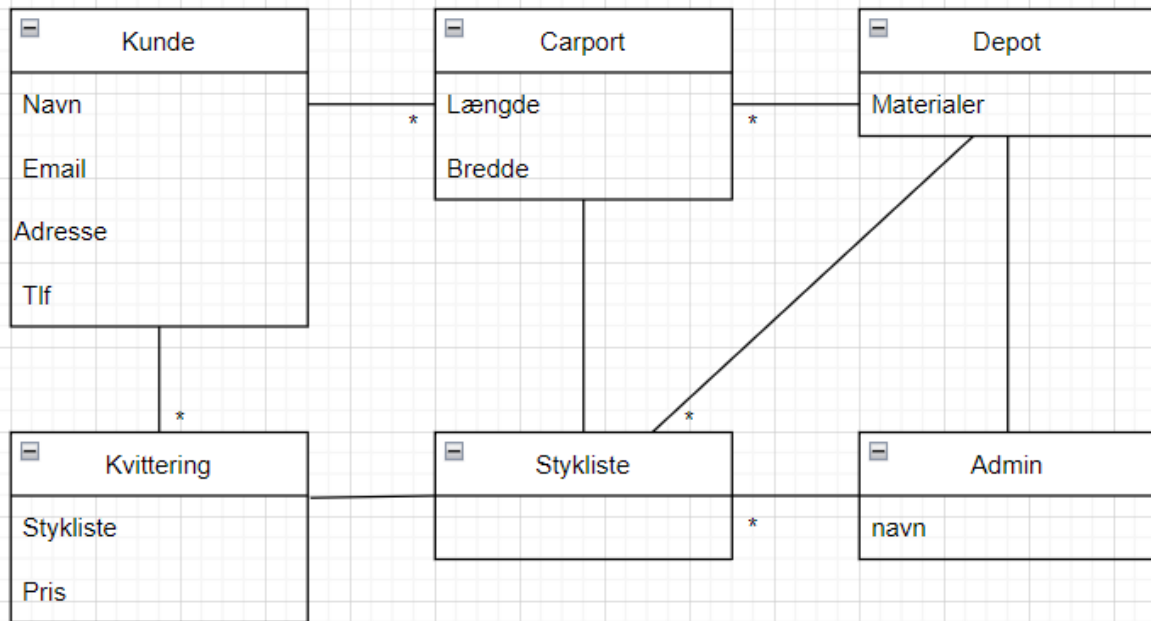
Us-4: (medium)

Når admin har logget ind skal admin klikke på stock, her kan admin se styklisten med priser, admin kan derefter oprette på listen.

Us-5: (large)

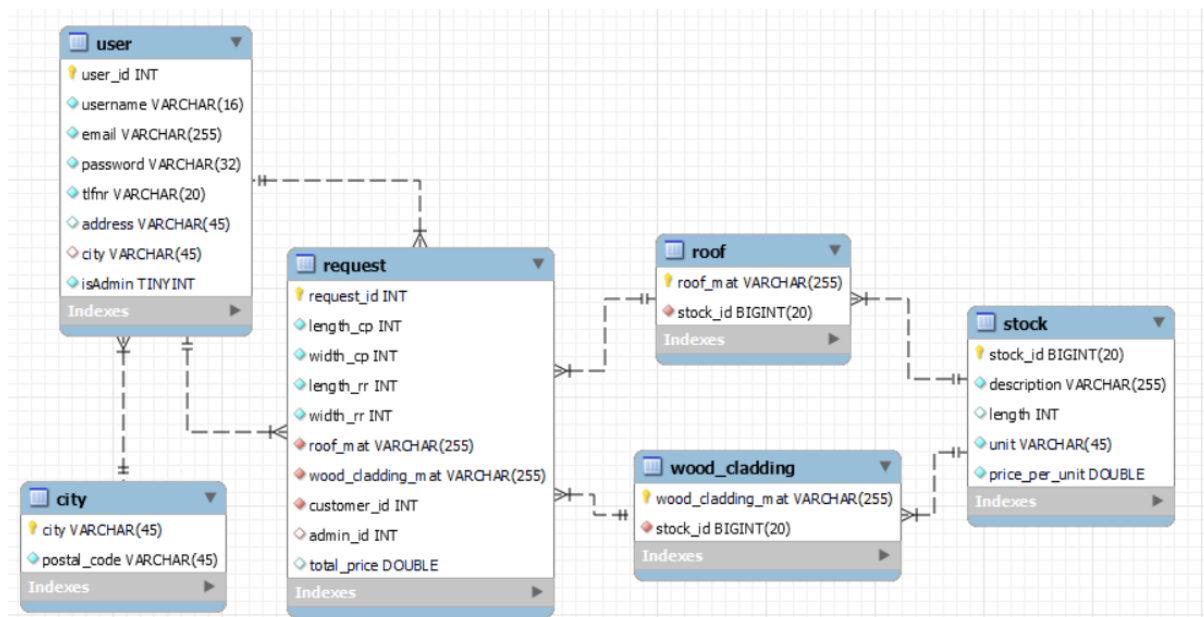
Når admin har logget ind skal admin klikke på stock, her kan admin se styklisten med priser, admin kan derefter fjerne, redigere og oprette på listen.

Domæne model



Vi forsøgte på første dag at opnå en forståelse af hvordan projektet skulle se ved at lave en domænemodel. Vi fandt frem til at en kunde skulle være i stand til ikke blot at designe en enkelt carport, men mange i tilfælde af at kunden skulle opgradere sin gamle, eller bestille for en tredjepart. Desuden skulle både depotet og admin kunne håndtere at arbejde med flere styklister, som bliver lavet ud fra den enkelte carport. I takt med at en bruger kan designe flere carporte skal kunden naturligvis også kunne modtage flere kvitteringer.

EER diagram



User

Vores user tabel var oprindeligt opdelt i 2. en customer tabel og en admin tabel men vi besluttede os for at slå dem sammen og skelne mellem de to med en boolean isAdmin. Herefter fandt vi det nødvendigt at lave en separat tabel til city tabellen med henblik på at opnå 3. normalform.

Roof og wood_cladding

Vi har lavet 2 link-tabeller: roof og wood_cladding. Dette blev gjort med henblik på at kunne hente data fra stock tabellen uden at kende det unikke id der er forbundet med materialerne. Vi kan gennem disse tabeller hente materialer ud fra stock tabellen ved kun at kende navnet. Det gør det lettere for brugeren at benytte carport designeren da de blot skal vælge hvilket materiale de ønsker gennem en drop-down menu. roof- og wood_cladding tabellen indeholder dog kun én værdi: not_yet_implemented. De er ganske enkelt ikke blevet implementeret endnu, men vi har bibeholdt dem i databasen for lettere at kunne udvide programmets funktionalitet i fremtiden.

Request

Request tabellen er den mest centrale del af databasen. Det er den tabel der bliver oprettet når brugeren designer sin carport. Vi har valgt at koble 2 user_id på request tabellen. Dette gøres for først at koble brugeren på det

request de opretter når de designer deres carport, og senere koble et admin_id på requested når en admin har set requested og godkendt det. Desuden bliver styklisten også genereret ud fra denne tabel. Dette sker ud fra length_cp og width_cp der bliver brugt til at beregne nøjagtigt hvor mange materialer der skal bruges. Det er disse mål bliver også brugt når vi laver en SVG tegning over carporten.

total_price i request tabellen bliver først sat når admin tilføjer sit user_id til tabellen. Der er 2 årsager til dette. For det første kan admin se requested og gennemgå det for potentielle fejl og mangler inden en pris bliver tilbudt kunden, derved opnår Fog en bedre user-experience. Den anden årsag er at vi "fastfryser" prisen på godkendelses tidspunktet. Admin kan altså gå tilbage til tidligere requests og se hvilket pris der blev genereret, selv hvis stock tabellens priser skulle hæves eller sænkes på et senere tidspunkt. Det er altså en måde at hjælpe Fog med at holde regnskab på deres salg.

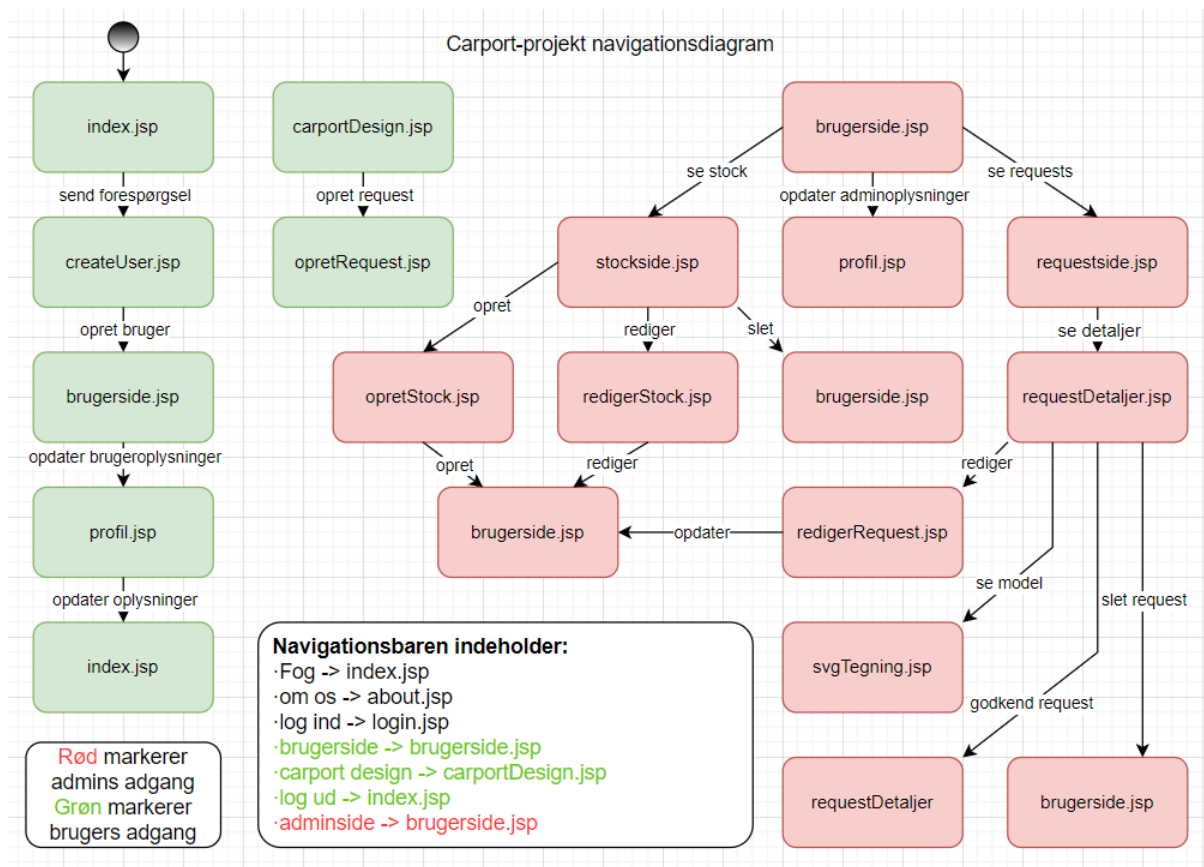
Stock

De unikke Id'er vi tilføjer til stock tabellen er ikke autogenereret. Dette skyldes at Fog naturligvis har deres egen database med hvad end de har af materialer på lageret. For at gøre det lettere for Fog at opdatere denne database har vi indskrevet deres egne materiale-id'er på vores materialer.

Overvejelser

Vi havde tidligere i forløbet overvejet at gemme styklisten som sin egen tabel i databasen. Planen var at kalde denne tabel partsList og vi ville gemme hele styklisten som et objekt, for at koble dette til request tabellen. Idéen blev senere til at vi ville have en ekstra kolonne på vores request tabel der indeholdt et objekt, men igen blev idéen skrottet grundet manglende tid til rådighed. Som det står nu bliver styklisten generet som et objekt i java når det skal bruges i diverse beregninger.

Navigationsdiagram



Admin adgang

Ovenstående navigationsdiagram er websitet umiddelbare navigationsmuligheder. De røde felter er de sider admin har rådighed over. Ved login bliver admin præsenteret for en brugerside, der bliver brugt som en central for alt hvad admin har rådighed over. Vi fandt under udviklingen at admins funktioner var mere fagligt interessante, hvorfor vi har prioriteret disse funktioner over brugerens.

Brugeradgang

De grønne felter er de felter brugeren, eller kunden har adgang til. Det er tydeligt at brugeren har færre muligheder end admin, hvilket desværre skyldes manglende tid til rådighed. Der mangler enkelte kernefunktioner som eksempelvis at kunden skal kunne se den SVG tegning der bliver genereret i forbindelse med den forespørgsel de sender.

Navigationsbar

Vi har valgt at implementere en navigationsbar, der tilpasses den bruger der er aktiv. Hvis en kunde er logget ind vil de have adgang til caportDesign.jsp, hvilket admin

ikke har adgang til umiddelbart. Ellers indeholder baren et logo der sender brugeren til index.jsp, et about.jsp side der indeholder en smule information om Johannes Fog som virksomhed og så naturligvis en log-ud knap. Hvis ingen bruger er logget ind vil log-ud knappen forekomme som en log-ind knap.

Uden logon

Uden et gyldigt log on er navigationen naturligvis begrænset. En bruger der ikke er logget ind har mulighed for at se about.jsp og derved læse om Fog. Index.jsp siden der der guider brugeren mod at oprette en bruger, og så selvfølgelig createUser.jsp og login.jsp hvor brugeren kan oprette sig eller logge ind hvis allerede oprettet i systemet.

Mock-ups

FOGs fabuløse carport designer

Logud

Carport
længde

480 cm

▼

Carport
bredde

180 cm

▼

Redskabs-
skur længde

180 cm

▼

Redskabs-
skur bredde

180 cm

▼

Vælg tag

Eternit-tag

▼

Vælg
beklædning

Egetræ

▼

Se
carport

Send
forespørgsel

Under vores foranalyse af projektet lavede vi et mockup af carportdesigneren. På ovenstående billede ses første del af vores mockup og viser hvordan vi havde forestillet os at brugeren skulle indtaste mål på carporten. Vi valgte bevidst at indføre drop-down menuer for at begrænse brugeren i sine valg. Virkeligheden blev meget lig dette mockup som set på nedenstående billede.

Carport Designer:

Carport Længde:
Vælg

▼

Carport Bredde:
Vælg

▼

Skur Længde:
Vælg

▼

Skur Bredde:
Vælg

▼

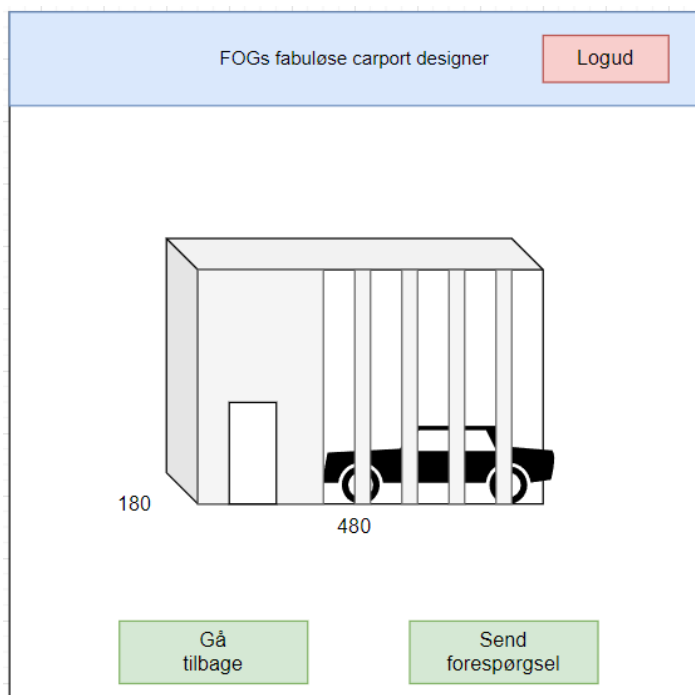
Tag Materiale:
Vælg

▼

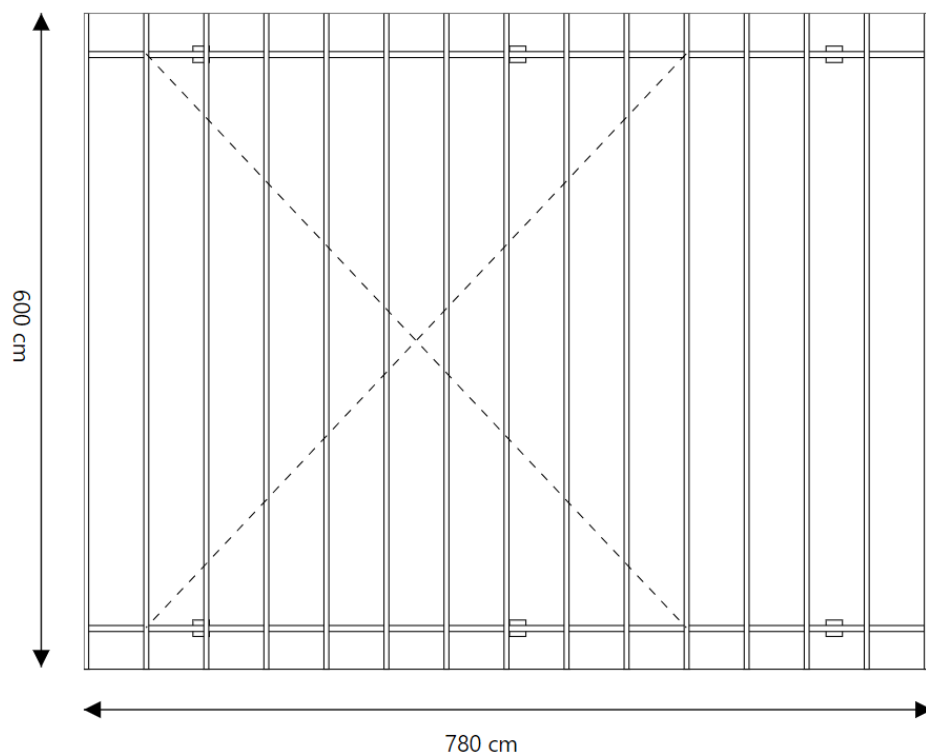
Træ Beklædning Materiale:
Vælg

▼

Opret Request



På ovenstående billede ses anden del af vores mockup. Ingen på teamet havde på forhånd erfaring med SVG så vi var i tvivl om hvordan den endelige model ville se ud, men vi vidste at vi ville have en model med specifikke mål, der giver kunden en ide om hvordan carporten vil se ud i sidste ende, så kunden føler sig tryk ved handlen. På nedenstående billede ses et eksempel af hvordan en model ser ud i det endelige projekt.



Valg af arkitektur

MVC-arkitektur

Som udgangspunkt stod vi med to muligheder inden for arkitektur. Vi kunne enten bruge en MVC-arkitektur, hvilket havde været i fordel på kort sigt. Da vi alle har erfaring med MVC-arkitekturen havde vi været i stand til at arbejde på de centrale features i løbet af et par dage. Men da dette trods alt er et læringsforløb frem for alt besluttede vi os for i stedet at arbejde med vores anden mulighed; en frontcontroller med commandpattern.

Front controller med command pattern

Vi besluttede i sidste ende at arbejde med front controller arkitekturen. På grund af dette måtte vi afsætte den første tid til at forstå hvordan denne front controller egentlig fungerer. Til gengæld fandt vi at det var, når ordentligt implementeret, forholdsvis nemt at tilføje nye sider og funktioner til vores program.

Facades

Vi benytter os af flere facader gennem projektet. I første omgang fandt vi at brugen af disse facader over komplicerede programmet, men efter udvidelse af programmet fandt vi at facaderne gav et overblik over funktionaliteter vi ellers ikke ville have og når først facaden var oprettet, sænkede det ikke vores arbejds effektivitet synderligt.

Brugeroplevelse

UI

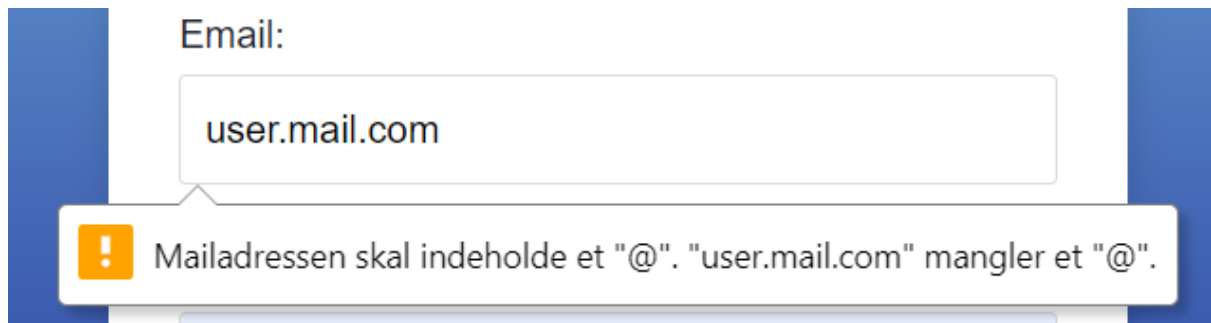
Vi har haft fokus på både user interface og user experience når vi har designet vores website. I forhold til user interfacet har vi lavet styling på samtlige sider som brugeren kan tilgå. Her har vi benyttet os af både inline, internal og external CSS samt en del bootstrap. De sider brugeren tilgår er der linket et stylesheet som giver en skyblue/navy farve. Dette tema er gennemgående for hele projektet og giver et professionelt udseende. Derudover har vi implementeret en navigationsbar som hjælper til at optimere brugerens mobilitet og navigation på siden. Farvemæssigt og designmæssigt har vi lænet os så meget som realistisk muligt op ad Fogs egen hjemmeside, og på forsiden, når brugeren først tilgår websitet afspilles Fogs carport reklame. Formålet med denne baggrunds video er at gøre sitet mere præsentabelt samt at øge kundens engagement.

UX

I forhold til user experience har vores fokus ligget på at lave et intuitivt og let tilgængeligt website, hvorfra kunder nemt kan oprette en bruger for derefter at lave en forespørgsel til en carport. Dette ses tydeligt på index.jsp siden hvor brugeren med det samme ser knappen "send forespørgsel". Ved tast på denne knap bliver brugeren guidet til createUser.jsp hvor brugeren hurtigt kan oprette en bruger. Efter oprettelse kan en bruger ved et enkelt tast på navigationsbaren komme direkte til carportDesign.jsp siden, hvor brugeren nemt og hurtigt kan designe sin carport efter specifikke mål.

Særlige forhold

Exceptions



The screenshot shows a web form with a label "Email:" and an input field containing the text "user.mail.com". Below the input field, there is a red error message box with a white exclamation mark icon. The message reads: "Mailadressen skal indeholde et '@'. 'user.mail.com' mangler et '@'." The form is set against a blue background.

Vi har forsøgt at imødekomme de mest gængse exceptions ved at begrænse brugeren således de aldrig opstår. Eksempelvis er det påkrævet at brugeren indtaster et "@" efterfulgt af en tekst for at kunne oprette sin mail i systemet. Dette ville ikke udløse en exception men er blot et eksempel på hvordan vi løser sådanne problematikker.

Hvis der opstår en fejl vi ikke har gennemskuet, har vi forsøgt at omdirigerer brugeren til en error.jsp side. Her genererer vi en kort fejlbesked der kan hjælpe brugeren til at forstå hvad der gik galt.

Denne side er ikke tilgængelig

Fejlkode: 404

Fejl! en fejl er sket i serveren

Wrong username or password

Data i session og request

I session gemmer vi ikke mange data. Vi gemmer et user objekt, som bruges til at genkende brugeren når de navigerer rundt på sitet. Samme objekt bliver brugt lige meget om det er en normal bruger eller en admin. Forskellen er deres rolle, som ligger på objektet. Hvis en side har nogle ting som brugeren ikke skal have adgang til, men en admin skal, så bliver dette gjort ved at vi kigger på rollen på objektet der ligger i session.

Bruger inputs

Såfremt brugeren ønsker at opdatere sine brugeroplysninger har vi valgt at brugeren skal skrive sin nye email to gange, det samme er gældende ved ændring af password. Dette gøres for at sikre at brugeren indtaster de korrekte oplysninger og altså ikke laver fejl i indtastningen. Desuden skal brugeren indtaste sit gamle password ved opdatering af kontooplysninger. Dette gøres for at sikre sig at det er den reelle bruger der ændrer oplysningerne. På nedenstående billede ses et udsnit af koden der håndterer brugerens input ved opdatering af kontooplysninger. Hvis brugeren taster forkert bliver brugeren returneret til samme side med en fejlbesked der informerer om hvad der gik galt.

```
// Hvis den nye kode ikke er udfyldt, eller de to indtastede koder ikke stemmer overens sættes koden til forrige.
if (newPassword.equals("")) { //
    newPassword = password;
    confirmNewPassword = password;
}

if (!newPassword.equals("") && !newPassword.equals(confirmNewPassword)) {
    request.setAttribute( name: "errorMsg", o: "You failed to confirm your new password.");
    return "updateUser";
}

// Hvis blanketten er udfyldt og de to nye emails stemmer overens skiftes emailen
if (newEmail.equals("")) {
    newEmail = email;
    confirmNewEmail = email;
}

if (!newEmail.equals("") && !newEmail.equals(confirmNewEmail)) {
    request.setAttribute( name: "errorMsg", o: "You failed to confirm your new email.");
    return "updateUser";
}
```

Når brugeren designer sin carport giver vi brugeren valgmuligheder frem for frit at skrive længde og bredde. Dette gøres for at brugerens input giver et brugbart output da vores udregninger og model er baseret på faste størrelser.

Carport Bredde:

Vælg

210 CM

240 CM

270 CM

300 CM

330 CM

360 CM

390 CM

420 CM

450 CM

480 CM

510 CM

540 CM

570 CM

600 CM

Vælg

Sikkerhed ved login

Når brugeren logger ind, søger vi i databasen efter en bruger med tilsvarende brugernavn og password. Såfremt et match ikke eksisterer i databasen bliver brugeren omdirigeret til errorpage hvor de får fejlbeskeden: "wrong username or password". Dette gør at brugeren ikke er i tvivl om hvor fejlen ligger, samt at der ikke bliver oprettet nogle forespørgsler medmindre brugeren er verificeret.

Status på implementation

CRUD

Vi har implementeret de fleste nødvendige CRUD operationer men grundet tidspres har vi ikke implementeret nogen delete funktion på user objektet. Dette gør at hverken bruger eller admin kan slette brugere fra databasen, der på lang sigt kan være problematisk hvis der opstår uventede problemer med en bruger, hvis en lagerplads nogensinde skulle blive et problem, samt hvis brugeren ikke længere at have sine oplysninger stående på databasen.

Bruger oplevelsen

Brugeren er i stand til at designe sin carport ud fra specifikke længde- og breddemål, men brugeren er ikke i stand til at tilvælge et redskabsskur. For at implementere dette skal vi opdatere SVG tegningen så den tager højde for at et skur kun er eventuelt, samt tilføje en række udregninger i vores Calculations klasse.

Brugeren kan ikke se den SVG modeltegning der bliver lavet som følge af at de designer carporten. For at implementere dette skal vi blot tilføje en link til brugerside.jsp. En mindre mangel, der hurtigt kan implementeres, men en mangel uanset.

Admin er i stand til at se styklisten efter brugeren har designet sin carport og derved oprettet et request i databasen. I forhold til vores use-cases bør brugeren ligeledes kunne se styklisten, men dette er ikke blevet implementeret. For at kunne implementere dette skal et link tilføjes til brugersiden der indeholder en if sætning for at kontrollere hvorvidt en admin har godkendt dette request.

Ved design af carport bør brugeren være i stand til at beklæde sin carport med en trætype. Dette kræver dog en udvidet database da vi ganske enkelt ikke har de data der kræves for at kunne implementere denne feature. For at kunne implementere denne feature skal der være et større udvalg af beklædningstyper, samt Calculations klassen skal udvides til at kunne håndtere disse data.

Ligeledes er brugeren ikke i stand til at tilvælge en tagtype. Vi mangler igen informationer til databasen for at dette kan implementeres, men også en udvidelse af SVG klassen samt Calculations klassen for at kunne håndtere dette ekstra element. Især hvis tagtypen er et tag med rejsning. Sidstnævnte er langt fra at kunne implementeres.

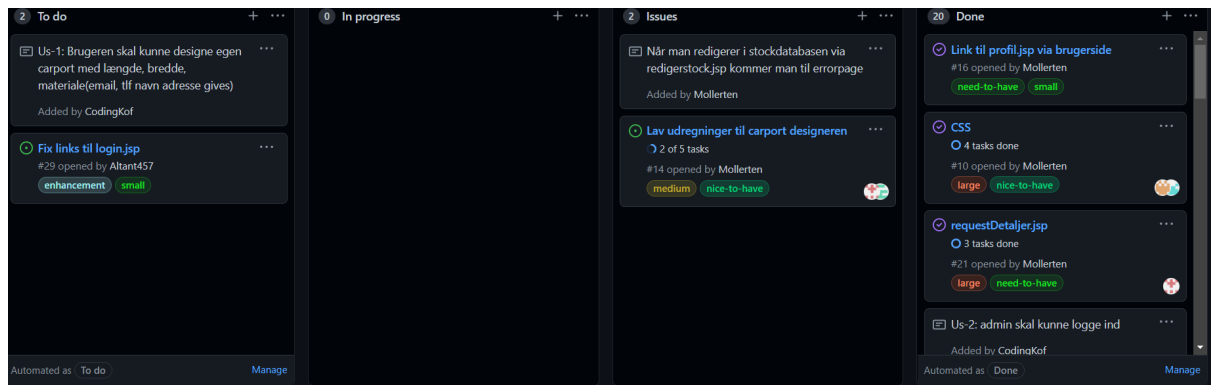
Proces

Arbejdsgangen

Vores arbejdsgang blev planlagt allerede ved projektets første dag. Vi besluttede os for at teamet ville arbejde på samme fysiske kontor for at have hurtigere adgang til hinandens kompetencer. Dette betød i praksis at vi oftest kunne arbejde os igennem problematikker på væsentligt kortere tid end hvis enkeltmand arbejde alene. Desuden besluttede vi os for at holde os konstant up-to-date med et online KanBan board. Dette gjorde det nemt at overskue hvilke features vi arbejdede på, samt hvor langt vi var i den samlede proces.

Kanban

KanBan er et online værktøj der hjælper et team med at danne et overblik over dens process. Her kan man blandt andet oprette stories, to-do's osv. alt efter hvad ens team har brug for.



På ovenstående billede ses et udsnit af vores KanBan board. Der er 3 primære kategorier som vi har anvendt til at sortere tasks, "To do", "In Progress" og "Done". Vi valgte at tilføje "Issues" så teamet kunne finde ud af hvem der havde problemer og hvorhenne.

Når man opretter en ny task kommer den til at ligge ind under "to do", hvorefter en KanBan masteren på teamet kan tildele sig selv eller andre den opgave og rykker den over i "In progress". "Issues" er hvis man kommer i problemer eller hvis man lige mangler den sidste finurlighed inden man flytter sin opgave over i "Done".

Gennem hele arbejdsprocessen udarbejdede vi en række issues med underpunkter vi kunne arbejde ud fra. Dette, sammen med at vi ofte arbejdede fysisk sammen hjalp os til at forstå hvor vi var hver især med vores opgaver.

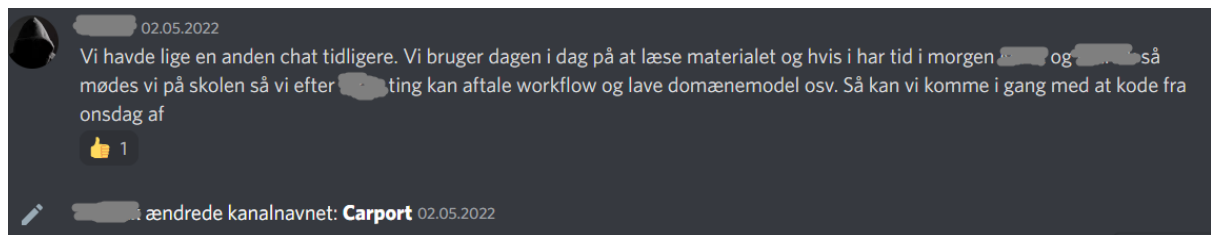
Scrum

Vi besluttede os for ikke at benytte os af scrum meetings. Vi fandt ganske simpelt at det var overflødigt for os at sætte et fast dagligt møde da vi i forvejen kunne holde ajour via KanBan. I bagklogskabens lys kunne det være en fordel for os at afholde disse møder alligevel da det kunne gøre det lettere for os at afsætte tid til at assistere hvor nødvendigt.

Navigationsdiagram

Vi fandt at det havde været fordelagtigt for os at udarbejde et groft navigationsdiagram tidligt i forløbet for at alle udviklere kunne have et ensartet mentalt billede af hvordan projektet skulle se ud i sidste ende. Dette gjorde vi desværre først i forløbets sidste dage, hvilket gjorde at der var uenighed om projektets navigationsmuligheder.

Discord



Vi brugte hyppigt Discord i arbejdsprocessen for at dele filer, aftale mødetider og meget mere. Det var en succes for os at have én central kommunikationslinje da ingen på noget tidspunkt var i tvivl om hvor man skulle gå hen for at holde sig up-to-date. Vi kunne have brugt zoom eller microsoft teams men vi fandt at discord var et medie vi alle havde erfaringer med på forhånd og desuden var vi ofte online på discord uden for den normale arbejdstid så der var også hjælp at hente når vi arbejdede skæve timer.

Udvalgt kode

I dette afsnit vil vi hver især indsætte et stykke kode fra projektet og forklare hvad koden gør samt hvorfor koden er interessant.

addCity

```
135 public City addCity(String city, String postalCode) throws DatabaseException
136 {
137     Logger.getLogger( name: "web").log(Level.INFO, msg: "");
138     City city1;
139     String sql = "SELECT * FROM city WHERE (city = ?)";
140     try (Connection connection = connectionPool.getConnection()) {
141         try (PreparedStatement ps = connection.prepareStatement(sql)) {
142             {
143                 ps.setString( parameterIndex: 1, city);
144                 ResultSet rs = ps.executeQuery();
145                 if (rs.next())
146                 {
147                     String cityName = rs.getString( columnLabel: "city");
148                     String postalCodeName = rs.getString( columnLabel: "postal_code");
149                     city1 = new City(cityname, postalCodeName);
150                 } else
151                 {
152                     String sql2 = "insert into city (city, postal_code) values (?,?)";
153                     try (Connection connection2 = connectionPool.getConnection()) {
154                         try (PreparedStatement ps2 = connection2.prepareStatement(sql2)) {
155                             ps2.setString( parameterIndex: 1, city);
156                             ps2.setString( parameterIndex: 2, postalCode);
157                             int rowsAffected = ps2.executeUpdate();
158                             if (rowsAffected == 1) {
159                                 city1 = new City(city, postalCode);
160                             } else {
161                                 throw new DatabaseException("The city with name: " + city + " could not be inserted into the database");
162                             }
163                         }
164                     }
165                 }
166             }
167         }
168     }
169 }
```

Hvad gør koden

Når brugeren opretter eller opdaterer sin bruger søger vi på linje 139 i databasen efter den indtastede by. Hvis denne by ikke eksisterer i databasen opretter vi på linje 152 et nyt row med brugerens indtastede by- og postnummer kombination. Herefter bliver byen indsat i user tabellen. Dette er dog ikke vist i eksemplet.

Hvorfor denne metode

Man kunne refaktorere koden således at sql sætninger blev opdelt i hver sin metode. én der søger i databasen efter identiske byer og én der opretter et nyt row såfremt byen ikke allerede eksisterer. Vi har dog valgt at bibeholde denne noget uoverskuelige kode da disse metoder udelukkende bliver brugt i sammenhæng med hinanden. Vi ser det ikke nødvendigt at opdele metoden i to, selvom det ville gøre det lettere at dokumentere og håndtere koden i fremtiden.

Opret request

```
@Override
public Request opretRequest(Request request) throws DatabaseException
{
    Logger.getLogger("web").log(Level.INFO, "");
    boolean result = false;
    int newId = 0;
    String sql = "insert into request (length_cp, width_cp, length_rr, width_rr, roof_mat, wood_cladding_mat, customer_id) values (?, ?, ?, ?, ?, ?, ?)";
    try (Connection connection = connectionPool.getConnection())
    {
        try (PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS))
        {
            ps.setInt(1, request.getLengthcp());
            ps.setInt(2, request.getWidthcp());
            ps.setInt(3, request.getLengthrr());
            ps.setInt(4, request.getWidthrr());
            ps.setString(5, request.getRoofmat());
            ps.setString(6, request.getWoodcladding());
            ps.setInt(7, request.getCustomerid());
            int rowsAffected = ps.executeUpdate();
            if (rowsAffected == 1)
            {
                result = true;
            }
            else
            {
                throw new DatabaseException("request med request id = " + request.getRequestid() + " kunne ikke oprettes i databasen");
            }
            ResultSet idResultSet = ps.getGeneratedKeys();
            if (idResultSet.next())
            {
                newId = idResultSet.getInt(1);
                request.setRequestid(newId);
            }
            else
            {
                request = null;
            }
        }
    }
    catch (SQLException ex)
    {
        throw new DatabaseException(ex, "Kunne ikke indsætte request i databasen");
    }
    return request;
}
```

Kode:

Når brugeren laver en request ved hjælp af drop down-boxene bliver der sendt en request afsted med følgende parametre på linje 280 i adminmapper:

length_cp,width_cp,length_rr,width_rr,roof_mat,wood_cladding_m

at, customer_id

Disse data bliver sendt ind i vores database ved hjælp af SQL-sætningen "insert into request", så en admin senere kan se dem under request og lave CRUD operationer. Det er vigtigt for en admin at kunne lave CRUD operationer som Read i request, Update i request og Delete i request. Det giver ikke mening for en admin at kunne Create da en medarbejder ikke skal designe en carport.

Hvis der af en eller anden grund går noget galt med request id'et, altså det ikke findes vil den slå fejl på linje 298 i adminmapper og dermed ikke blive oprettet.

Valg af kode:

Denne del er helt essentiel for opgaven da en bruger skal kunne designe sin egen carport med sine egne valgte mål. Uden dette ville en admin aldrig kunne se, hvilke bestillinger der er samt hvilke mål carporten har og vil derfor aldrig kunne godkende ordren.

Opret stock

Admin har mulighed for at tilføje materialer til stock, ved at indsætte værdierne for stockID, beskrivelse, antal, stock unit og stock per pris. Når admin trykker opret stock bliver parametrene bliver afsendt, og behandlet i OpretStock.java, her kaldes metoden opretNystock fra Adminmapperen som kommunikerer med databasen.

Her ses et skærmbillede af metoden opretNyStock i adminmapperen:

```
@Override
public Stock opretNyStock(Stock stock) throws DatabaseException {
    Logger.getLogger( name: "web").log(Level.INFO, msg: "");
    boolean result = false;
    int newId = 0;
    String sql = "insert into stock (stock_id, description, length, unit, price_per_unit) values (?, ?, ?, ?, ?)";
    try (Connection connection = connectionPool.getConnection()) {
        try (PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {

            ps.setLong( parameterIndex: 1, stock.getStockid());
            ps.setString( parameterIndex: 2, stock.getDescription());
            ps.setInt( parameterIndex: 3, stock.getAmount());
            ps.setString( parameterIndex: 4, stock.getUnit());
            ps.setDouble( parameterIndex: 5, stock.getPrice_per_unit());
            int rowsAffected = ps.executeUpdate();
            if (rowsAffected == 1) {
                result = true;
            } else {
                throw new DatabaseException("stock med beskrivelse = " + stock.getDescription() + " kunne ikke opret
            }
        }
    } catch (SQLException ex) {
        throw new DatabaseException(ex, "Kunne ikke indsætte stock i databasen");
    }
    return stock;
}
```

Det er vigtigt for admin at kunne tilføje materialer til stock, hvis der kommer nyt materiale på lager. Vores tables roof og wood_cladding er derudover også afhængige af et stockID i stock tabellen, før at disse kan indtastes i databasen.

Maps i Calculator

```
protected static Map<Integer, Integer> calcRafters(int length, int width)
{
    Map<Integer, Integer> rafters = new HashMap<>();
    int rafterAmount;
    int rafterLength;

    rafterLength = width;
    rafterAmount = (int) Math.floor((length-10) / 55.0 + 1);

    rafters.put(rafterLength, rafterAmount);
    return rafters;
}
```

Da vi først satte os for at lave en calculator besluttede vi os for at lave en udregning per materiale i styklisten. I ovenstående udsnit ser vi metoden calcRafters, det er altså her vi udregner antal spær i byggeprojektet. Hvis et materiale har en længde som i ovenstående eksempel, opretter vi et map der indeholder både mængde og længden.

```
Map<Integer, Integer> rafterNumbers = calcRafters(length, width);
int rafterLength = rafterNumbers.entrySet().stream().findFirst().get().getKey();
int rafterAmount = rafterNumbers.get(rafterLength);
partsList.setRafterCount(rafterAmount);
int tempRafterLength = rafterLength;
while (tempRafterLength % 60 != 0) tempRafterLength += 15;
Material rafters = new Material( description: "45x195 mm. spærtræ ubh.",
    rafterAmount,
    tempRafterLength,
    unit: "stk",
    helpText: "Spær, monteres på rem");
partsList.addMaterial(rafters);
```

At lægge materialerne i maps viste sig dog at være en knap så god beslutning. For at trække længden ud af mappet kalder vi:

```
entrySet().stream().findFirst().get().getKey();
```

En noget langhåret og overskuelig måde at trække en enkelt værdi ud af et map men brugbar uanset. Vi fandt desuden at vores udregner ikke tog højde for at brugeren altid vil sende længde og bredde på carporten i inkremitter af 30cm. For at imødekomme dette har vi brugt modulus % til at teste for om matematikken går op.

SVG tegning

For at kunne lave en tegning baseret på den carport som en kunde bestiller, skulle der laves en del kode der kunne lave en dynamisk tegning. Her er et udsnit af den kode.

```
56      // Add the poles
57      carport.addRect(100, 30, 15, 15);
58      carport.addRect(100, carportWidth-35-10, 15, 15);
59      carport.addRect(carportLength-100, 30, 15, 15);
60      carport.addRect(carportLength-100, carportWidth-35-10, 15, 15);
61      if (carportLength-200 > 310) {
62          float middlePoleX = carportLength / 2;
63          carport.addRect(middlePoleX, 30, 15, 15);
64          carport.addRect(middlePoleX, carportWidth-35-10, 15, 15);
65      }
```

På billedet her kan man se hvordan koden indsætter de rektangler som skal forestille stolperne på carporten. På linje 57 sættes den stolpe der, på tegningen, sidder oppe i øverste venstre hjørne. Denne stolpe sættes altid samme sted. På linje 58 og frem bliver stolpernes placering mere dynamiske. På linje 61 tjekker koden om der skal sættes en stolpe i midten af hver side.

Tests

Vi har lavet to test klasser, CalculatorTest og UserMapperTest, som direkte tester Calculator og UserMapper respektivt.

Klasse	Metode, %	Linje, %
City	25% (2/8)	53% (7/13)
User	23% (5/21)	58% (23/39)
DatabaseException	50% (1/2)	50% (2/4)
ConnectionPool	60% (3/5)	76% (20/26)
UserMapper	80% (4/5)	69% (72/103)
Calculator	83% (15/18)	43% (94/216)
UserFacade	75% (3/4)	75% (6/8)

Tabellens data er genereret af IntelliJ efter alle tests blev kørt ved at vælge "Run with Coverage".

Klasser der ikke står nævnt i tabellen, er ikke testet.

Som man kan se i tabellen, så er det ikke kun de klasser vi direkte tester der bliver testet. De bliver testet indirekte ved at de bliver kaldt fra de andre klasser.

Vi brugte ikke tests lige så meget som vi måske burde have gjort, set i bagklogskabens lys. CalculatorTest blev først lavet efter klassen som den tester var færdig. Der hjalp den til gengæld med at finde nogle fejl som vi ellers ikke havde opdaget.