

Fog Quick-Byg Carport

15/11/24 - 19/12/24



DAT2F2024 - Gruppe C

Agnethe Christensen, cph-ac331@cphbusiness.dk, Agnetechr

Rolf Lundsgaard Josephsen, cph-rj17@cphbusiness.dk, MrPrimeBeef

Peter Bollhorn, cph-pb274@cphbusiness.dk, pbollhorn

Lasse Jensen, cph-lj481@cphbusiness.dk, Lorsayden

Indholdsfortegnelse

Indledning.....	3
Baggrund.....	3
Virksomheden & Forretningsforståelse.....	4
Teknologivalg.....	6
Krav.....	7
Arbejdsgange der skal IT-støttes.....	8
User stories.....	10
Prioritet P1 – Vigtige funktioner.....	10
Prioritet P2 – Sekundære funktioner.....	11
Domænemodel og ER diagram.....	12
Domænemodel.....	12
Kunde, sælger og admin.....	13
Alle varer i Fogs sortiment.....	13
Ordrelinjer og ordrer.....	13
ER diagram.....	14
Sammenligning med domænemodellen.....	15
3. normal form.....	15
Primærnøgler.....	15
Fremmednøgler.....	15
Unique constraint.....	16
Tabel som vi burde have haft.....	16
Tidlig version af ER diagram.....	16
Navigationsdiagram & Mockups.....	17
Valg af arkitektur.....	23
Packages og deres funktioner.....	23
Særlige forhold & Udvalgte kodeeksempler.....	25
Session Management i Webapplikationen.....	25
Exception handling og Logging.....	25
DTO (Data Transfer Objects).....	26
Guard Condition.....	27
Password generator.....	28
Beregningsmotor.....	29
SVG motor (tegningsmotor).....	31
Automatiserede tests.....	32
Status på implementering.....	33
Kvalitetssikring (test).....	36
Automatiserede tests.....	36
User Acceptance tests.....	38
Proces.....	40
Arbejdsprocessen faktuelt.....	42

Arbejdsprocessen reflekteret.....	43
Bilag.....	44
Bilag A - User stories.....	44
Bilag B - Udsnit af logbog.....	49
Bilag C - Code of conduct.....	50
Bilag D - Kodestandard.....	51
Bilag E - Tidlig version af ER diagram.....	52

Link til repository: <https://github.com/mrPrimeBeef/FogCarport>

Link til demo video: https://youtu.be/kcolhAZ_xQo

Link til deployment: <https://carport.jcoder.dk/>

Login til en kundekonto: a@a.a kode: 1234

Login til en sælgerkonto: b@b.b kode: 1234

Indledning

Dette projekt er udviklet som en erstatning til Johannes Fogs "Quick Byg" del af deres eksisterende system. Deres nuværende system er forældet og vanskeligt at redigere, hvilket gør det ineffektivt for deres behov. Formålet med projektet var at udvikle en web applikation, der kan tage input fra brugeren, beregne en stykliste(materialeliste), og generere arbejdstegninger til en carport baseret på kundens specifikationer.

Web applikationen skulle inkludere en "beregningsmotor", der estimerer de nødvendige materialer baseret på kundens ønsker, samt en "tegningsmotor", der visualiserer designet. Projektet skulle modernisere processen og gøre det lettere for Johannes Fogs sælgere at tilbyde kundetilpassede løsninger, samtidig med at det forenkler arbejdsgangen. En vigtig præmis var, at kunden kun modtager materialelisten, når produktet er købt.

Projektet blev indledt med en video fra Martin, salgschef hos Johannes Fog, der gennemgik fordelene og udfordringerne ved deres nuværende quick-byg system. En væsentlig pointe var ønsket om at forbedre kundens købsoplevelse uden at gå på kompromis med sælgerens involvering.

Baggrund

Johannes Fog er en velkendt trælast og byggecenter, der tilbyder en bred vifte af løsninger inden for byggeri, herunder specialdesignede carporte. Virksomheden ønskede et nyt system, der kunne samle hele processen omkring carport design og -beregning i én integreret løsning, så de kunne undgå brug af flere adskilte systemer.

De vigtigste krav til det nye system var:

- Kunden skal kunne indtaste deres ønskede mål for carporten, herunder højde, bredde og længde, inden for specificerede minimum- og maksimumværdier. - jævnfør samtal med "Martin"¹
- Systemet skal bevare kontaktfladen mellem sælger og kunde, da den personlige service er en central del af Johannes Fogs forretningsmodel.

¹ Bilag B - Logbog - Fredag 29. nov. 2024

- Systemet skal give kunden en nem og overskuelig måde at designe en carport på, samtidig med at sælgeren bevarer kontrollen over udleveringen af materialelisten, som først må deles, når produktet er købt.

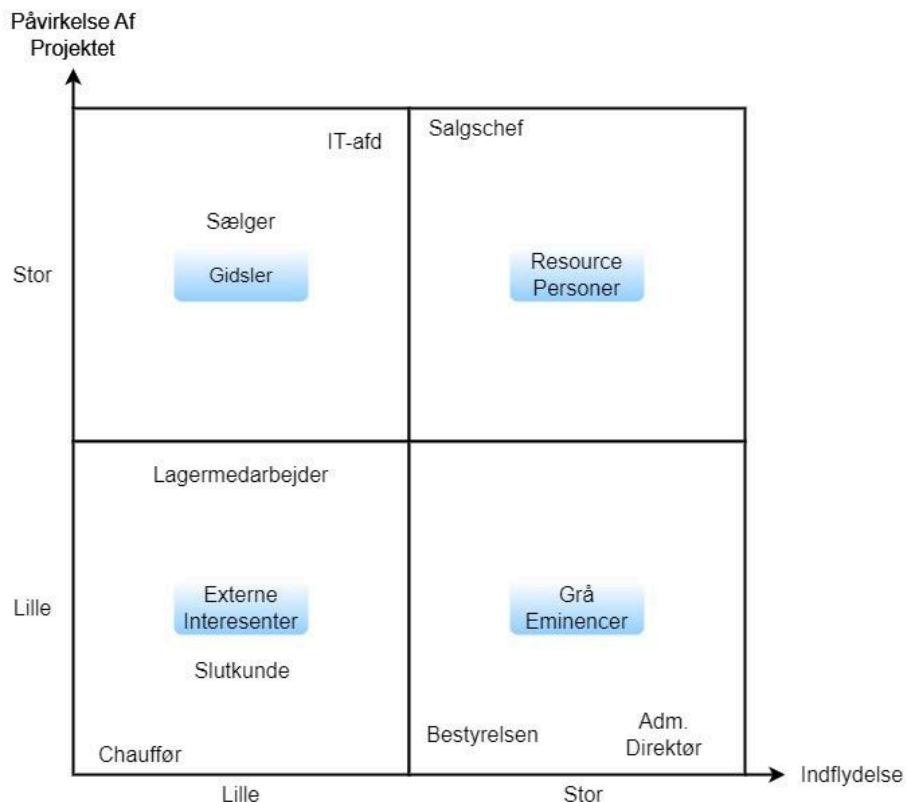
Ved at imødekomme disse krav ønskede Johannes Fog at skabe en mere moderne og effektiv løsning, der kunne forbedre både sælgernes arbejdsproces og kundernes oplevelse.

Virksomheden & Forretningsforståelse

For at sikre, at projektet blev udført med høj præcision af kundens behov, har vi gennemført to analyser: en interessentanalyse (figur 1) og en risikovurdering (figur 2).

Interessentanalysen har til formål at identificere de personer eller grupper, der kan påvirke projektet eller blive påvirket af det. Målet er at kortlægge interesserternes behov og indflydelse, så projektet kan styres succesfuldt.

I vores tilfælde når det er et skoleprojekt - er der ikke andre end "kunden", men analysen er foretaget som det var et ægte projekt, vi har identificeret flere væsentlige interesserter men for dette projekt er det kun "Martin". Martin spiller en afgørende rolle, da han både har indflydelse på projektets forløb og er afhængig af dets resultater. Hans input danner grundlag for kravspecifikationen, og hans aktive deltagelse er essentiel for at sikre projektets succes.



Figur 1 - Interessentanalyse

Risikoanalysen (figur 2) blev udført for at identificere potentielle risici, der kunne forsinke eller afspore projektet. Derefter vurderede vi risikoen ved at analysere sandsynligheden og alvoren af hver risiko og udviklede planer for forebyggelse eller afværgning.

Til risikoanalysen benyttede vi skabelonen fra automledelse.dk, men vi tilføjede en ekstra kategori under sandsynlighed, "meget usandsynligt", for at skabe en mere retvisende vurdering af visse risici.

Forebyggelse						
Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau	Plan for forebyggelse/afværgning	
1	Sygdom i udviklingsteamet	Acceptabel	Muligt	Lav	God kommunikation, og fornuftigt projektstruktur	
2	Fog går konkurs	Uacceptabelt	Meget usandsynlig	Medium	Forbyggelse: hold øje med regnskab, Afværgning: betaling I bider	
3	Tab af kodearbejde	Uacceptabelt	Muligt	Ekstrem	Vi bruger github og laver mange små commits og pushe	
4	Misforståelse Fogs behov	Uønsket	Muligt	Høj	Aftaler på skrift(logbog) løbende kommunikation med Fog - feedback loops	
5	Dårlig planlægning af projektets struktur	Uønsket	Muligt	Høj	Fornuftige estimater og priotering af opgaver. Se iørigt punkt 4	
6	Resource personer langtidssyge/fyret	Uønsket	Usandsynlig	Medium	Vi anbefaler grupper istedet for enkelte personer fra fog	
7	Teknologi døder	Uønsket	Meget usandsynlig	Lav	Vælge almene brugt teknologier samt være opmærksom på EOL/EOS	
8	Uønskede kode - hvis dependencies bliver inficeret eller vi bliver hacket	Uacceptabelt	Meget usandsynlig	Medium	Slut verificere projektet	
9	Forkerte beregninger på strukturelle behov eller forkerte styklister grundet vores program	Uacceptabelt	Muligt	Ekstrem	Uddybende Unit test på beregningsdelen	
10	Bytter rundt på Fogodata (fence post problem)	Uacceptabelt	Meget usandsynlig	Medium	Integrationstests	
11	Fogs kunder kan se private data grundet programmeringsfejl	Uacceptabelt	Usandsynlig	Høj	laver tests	
12	Hvis en kunde sletter en email og en ny kunde lave samme email, så er der data problemer	Uønsket	Meget usandsynlig	Lav		

Figur 2 - Risikoanalyse

Vi har udarbejdet forebyggelses- og afværgnings planer for næsten alle risici, men vores primære fokus ligger på de risici, der vurderes at have et niveau over "medium". For eksempel arbejder vi aktivt med forebyggelse eller afværgning af punkterne 3, 4, 5, 9 og 11.

En af de mest fremtrædende risici, vi identificerer, er risikoen for, at en ressourceperson bliver langtidssyg, fyret eller selv søger nyt arbejde. Vi vurderer denne risiko som "medium". Som forebyggelse og afværgning foreslår vi, at Fog nedsætter en styregruppe, som kan fungere som en ressourceperson og minimere afhængigheden af enkeltpersoner.

Teknologivalg

Teknologi	Funktion
Udviklingsværktøjer	
IntelliJ IDEA (version 2024.1)	IDE (Integreret udviklingsmiljø)
Git og Github	Version Control
Docker Desktop (version 4.34.1)	Containerization Software
pgAdmin 4 8.3	Database Administration Tool
JUnit (version 5.10.2)	Unit tests og integrationstest
Design	
Figma	Mockup design
Server til deployment	
DigitalOcean	Cloud Service Provider
Ubuntu (version 22.04.5)	Styresystem
Docker Engine (version 27.4.0)	Containerization Software
Caddy	Web server
Backend	
Java (version 17)	Programmeringssprog
Javalin (version 6.1.3)	Java Web Framework
PostgreSQL (version 16)	Database
HikariCP (version 5.1.0)	JDBC connection pool
Frontend	
Thymeleaf (version 3.1.2)	Java Template Engine Framework
HTML5	Markup-sprog til strukturering af webindhold.
Bootstrap 5	Framework til styling og responsivt design
CCS 3	Muliggør mere unik og avanceret styling

Krav

De krav der er stillet til vores løsning, vores løsning omhandler at man som kunde kan få et tilbud på "Quick-byg carport", har vi skulle udlede ud fra et interview lavet med Fog "Salgschef" Martin. Der gennemgår han forløbet, som kunden kommer igennem og beskriver de gode og dårlige ting ved deres nuværende situation.

Det som de godt kunne tænke sig bliver forbedret er:

- De kunne ikke opdatere deres materialedatabase på quick byg
- Bedre visualisering
- Et system, i stedet for flere
- Mindre manuelt arbejde

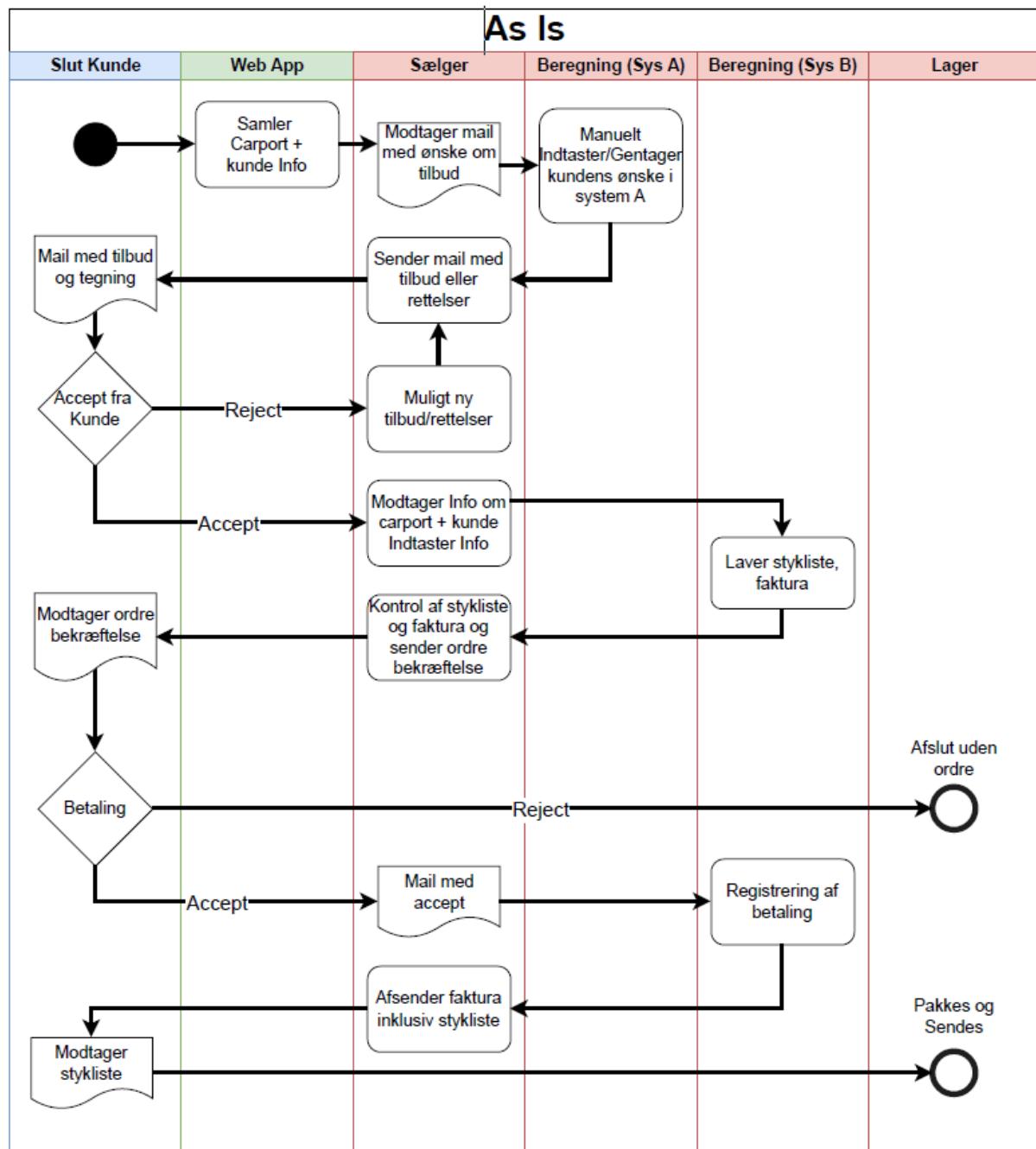
Skal som fungere godt og som minimum skal gøres efter:

- Quick byg kan lave en stykliste (dog kan de ikke ændre materialer).
- Styklisten som bliver lavet skal først blive givet til kunden efter køb
- Når en kunde køber en carport skal der en sælger indover for at give dem en bedre oplevelse og sikre at de ikke laver fodfejl.
- Nuværende program regner pris og giver mulighed for at ændre i dækningsbidraget.

Herudover har vi haft møder med "Martin", som har givet udtryk for at kunden skal kunne bestille en carport med egne mål, helt frit fra de 30 cm intervaller som angivet på deres nuværende side.

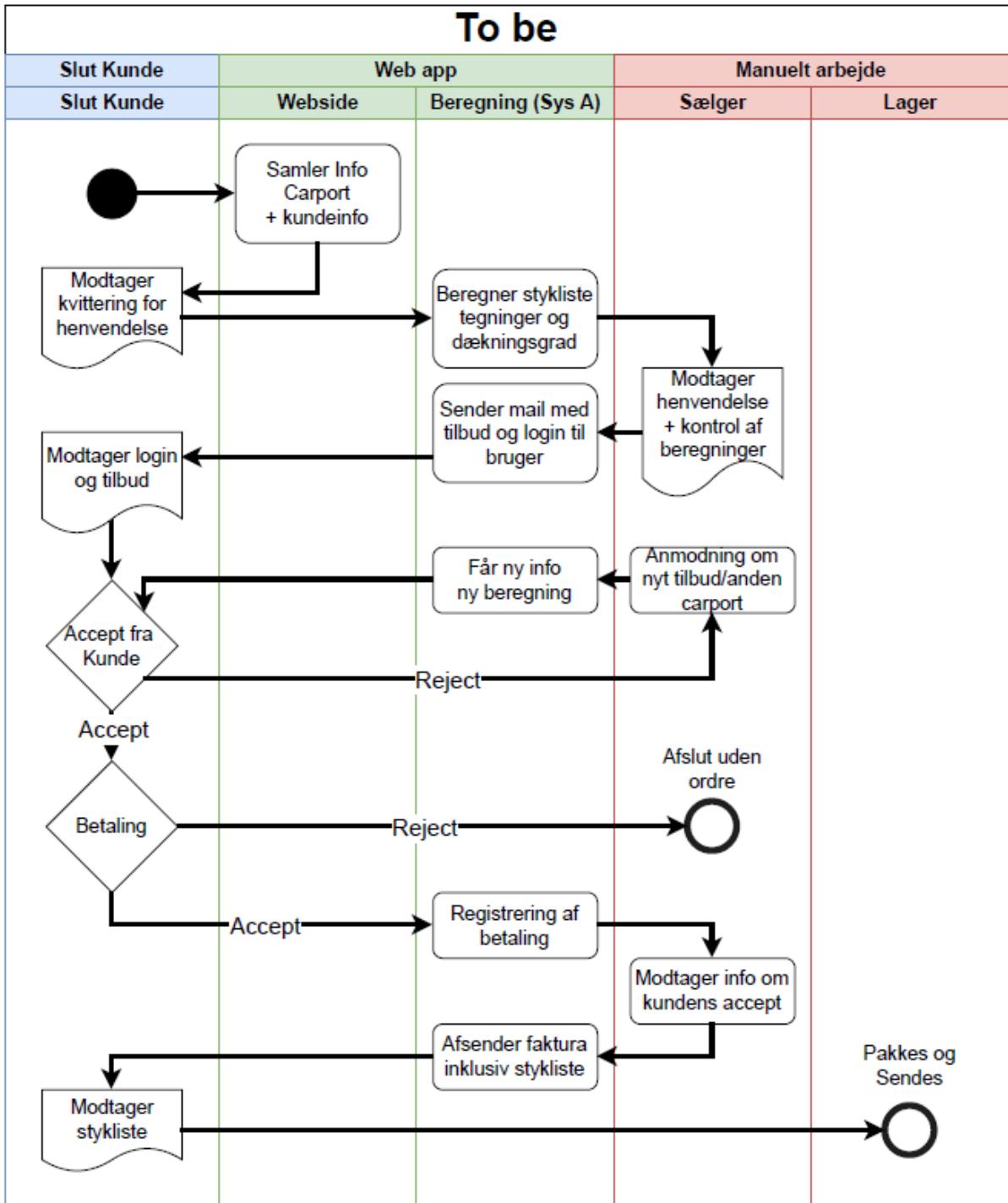
Arbejdsgange der skal IT-støttes

I både figur 3 og 4, vises der ikke at kunden faktisk når som helst i processen kan afbryde købet, hvis figurerne skulle vise det, ville de hurtigt blive uoverskuelige.



Figur 3 - As is, aktivitetsdiagram

Figur 3 viser hvordan Fog's system til carporte virker på nuværende tidspunkt. Det er værd at bemærke at der er to systemer som skal benyttes. Der er også meget manuelt arbejde ved dette system, hvilket der ikke burde være ved et system som dette.



Figur 4 - To be, aktivitetsdiagram

I figur 4 ses vores løsning på et carport beregnings system. Vi bruger ikke et helt traditionelt swimlane diagram, men har i stedet modificeret det til vores behov. I den måde vi viser et swimlane diagram på, gør det sig gældende at man kan "kategorisere" flere swimlanes under en overordnet swimlane. For eksempel er både "website" og "beregnings system" under kategorien "web app".

User stories

For at sikre en IT-støttet arbejdsgang, som afspejler kundens behov og de tidligere aktivitetsdiagrammer (Figur 3 og 4), har vi udarbejdet og prioriteret en række user stories. Hver user story er specificeret med **prioritet**, **estimeret størrelse**, og **acceptkriterier**, hvilket gør dem sporbare til aktivitetsdiagrammerne og nemme at bryde ned i mindre opgaver.

Nedenfor præsenteres et udvalg af de vigtigste user stories. Den samlede liste af user stories samt detaljerede tasks og estimater er vedlagt som bilag².

Prioritet P0 – Kritiske funktioner

1. US1 – Indtast ønsket carport og få tilbud (Large)

Som uregistreret kunde vil jeg kunne indtaste mål for min carport, så jeg kan sende dem til Fog for et tilbud.

Acceptkriterier:

- Givet at jeg er på carport-siden.
- Når jeg vælger mine mål.
- Så kan jegindsende informationerne til Fog.

2. US5 – Sælger klikker på henvendelse og ser beregninger (Large)

Som sælger vil jeg kunne se beregninger, stykliste og dækningsgrad for en henvendelse, så jeg kan træffe beslutning om videre rådgivning.

3. US6 – Sælger ændrer beregninger/tilbud (Large)

Som sælger vil jeg kunne ændre beregninger for at give kunden et opdateret tilbud.

Prioritet P1 – Vigtige funktioner

4. US3 – Sælger ser oversigt over carport-henvendelser (Medium)

Som sælger vil jeg kunne se en oversigt over henvendelser og klikke ind på hver enkelt.

5. US4 – Tak for din henvendelse (Medium)

Som kunde vil jeg modtage en kvittering for min henvendelse.

6. US7 – Bruger login (Small)

Som bruger vil jeg kunne logge ind og få adgang til mine funktioner afhængigt af rolle (sælger/kunde).

² Bilag A - User stories

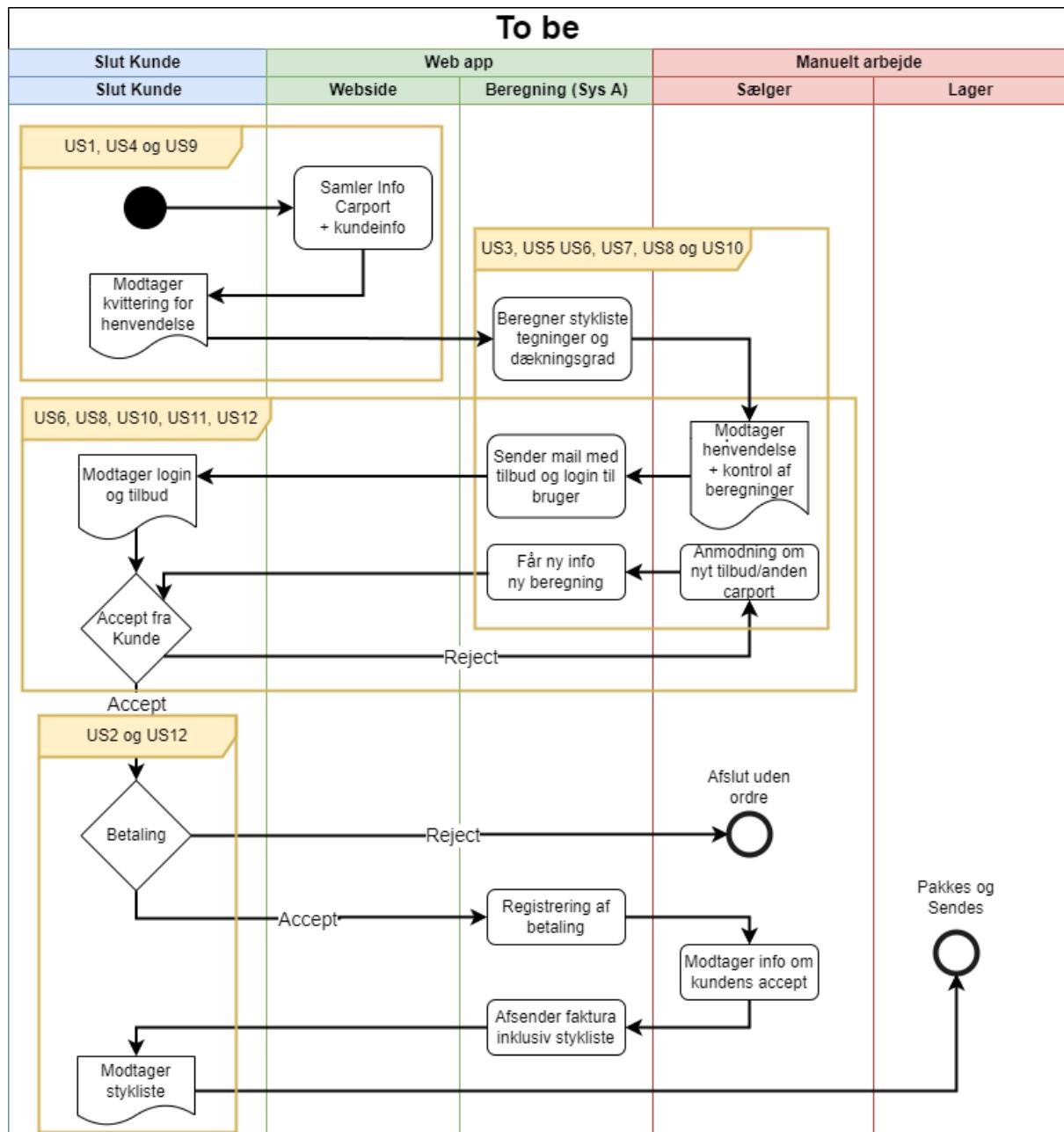
Prioritet P2 – Sekundære funktioner

7. US10 – Se alle kunder (Medium)

Som sælger vil jeg kunne se en oversigt over alle kunder.

8. US13 – Sælger markerer henvendelse som igangværende (Medium)

Som sælger vil jeg kunne markere, hvilke henvendelser jeg arbejder på, så andre kan se det.

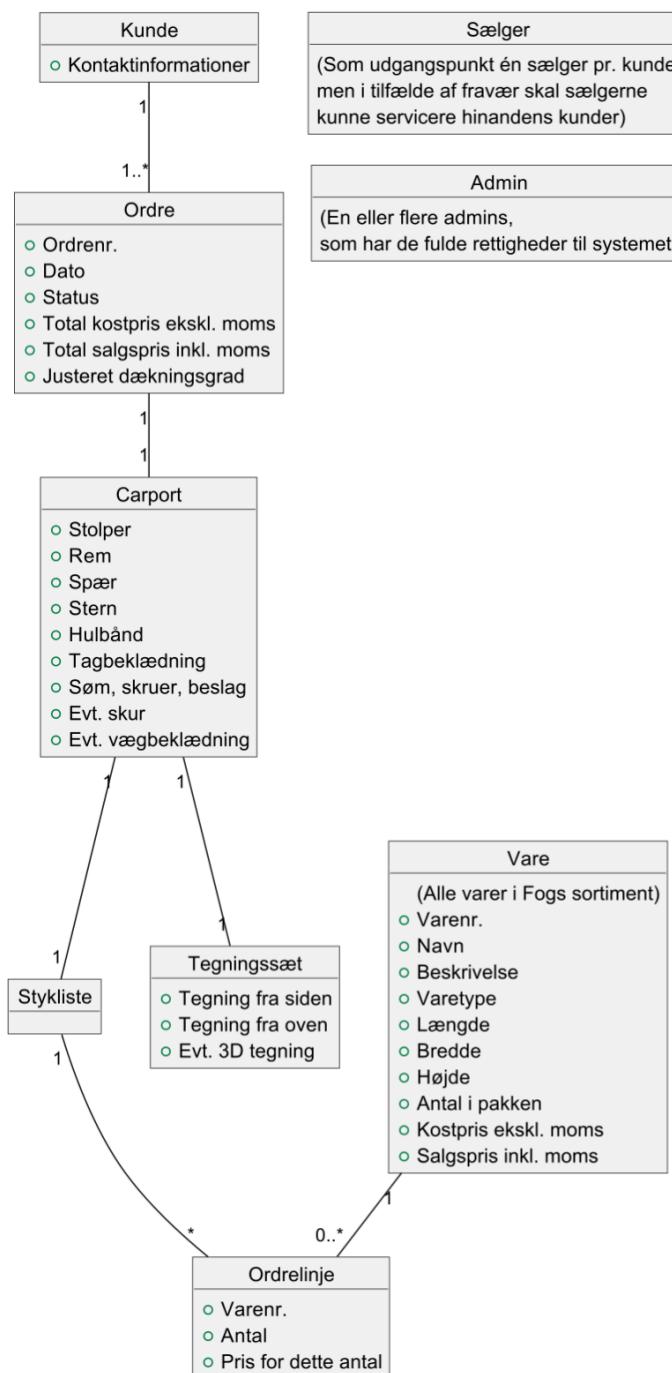


Figur 5 - To be, aktivitetsdiagram med User stories

Domænemodel og ER diagram

Domænemodel

For at være sikker på at vi har forstået Fogs behov til web applikation, har udfra "Interview"³ med Martin udviklet en domænemodel over deres custom carport forretning. Domænemodellen er vist på figur 6.



Figur 6 - Domænemodel over Fogs custom carport forretning.

³ Link til video

Relationerne i domænemodellen er:

- Én kunde kan have én til mange ordrer. For man kan først blive oprettet som kunde i forbindelse med at man laver sin første ordre.
- Én ordre kan have én custom carport, og ikke andre ting. For custom carporte bliver solgt uden om kurv-systemet på Fogs webshop.
- Én carport har ét sæt tegninger (Fra siden, fra oven, evt. 3D)
- Én carport har én stykliste
- Én stykliste indeholder mange ordrelinjer
- Én vare kan indgå i nul til mange ordrelinjer.

Kunde, sælger og admin

Fog har allerede en eksisterende database med kundekontoer. Køb af carporte skal registreres i disse kontoers ordrehistorik. Kunder som ikke har en kundekonto, får automatisk oprettet en kundekonto, som en del af købsprocessen.

Sælgers opgaver er at:

- Køre carport beregningen, som genererer tegningerne og styklisten.
- Have dialog med kunden, f.eks. hvis der er noget der ikke giver mening.
- Justere dækningsgraden, så der f.eks. kan gives en god salgspris til en loyal kunde.

Admin skal have de fulde rettigheder til systemet, dvs. kunne:

- Oprette og slette sælgere.
- Ændre password og andre kontooplysninger for sælgere.
- Tilføje, opdatere og slette varer.
- Ændrer på minimum dækningsgrad som sælgerne må sælge carportene til.

Alle varer i Fogs sortiment

Den nye web applikation skal integreres med Fogs eksisterende varedatabase, således at styklisten bliver genereret ud fra de varer som Fog rent faktisk sælger. Fogs varedatabase har alt fra brædder og beslag til havemøbler og cykeludstyr. Varerne har derfor en individuel kostpris og salgspris uafhængig af hinanden. For kostprisen er den pris som Fog indkøber varen til, og salgsprisen er det de sælger den til, hvilket de justerer for hver enkelt vare for at kunne være konkurrencedygtig på markedet.

Ordrelinjer og ordrer

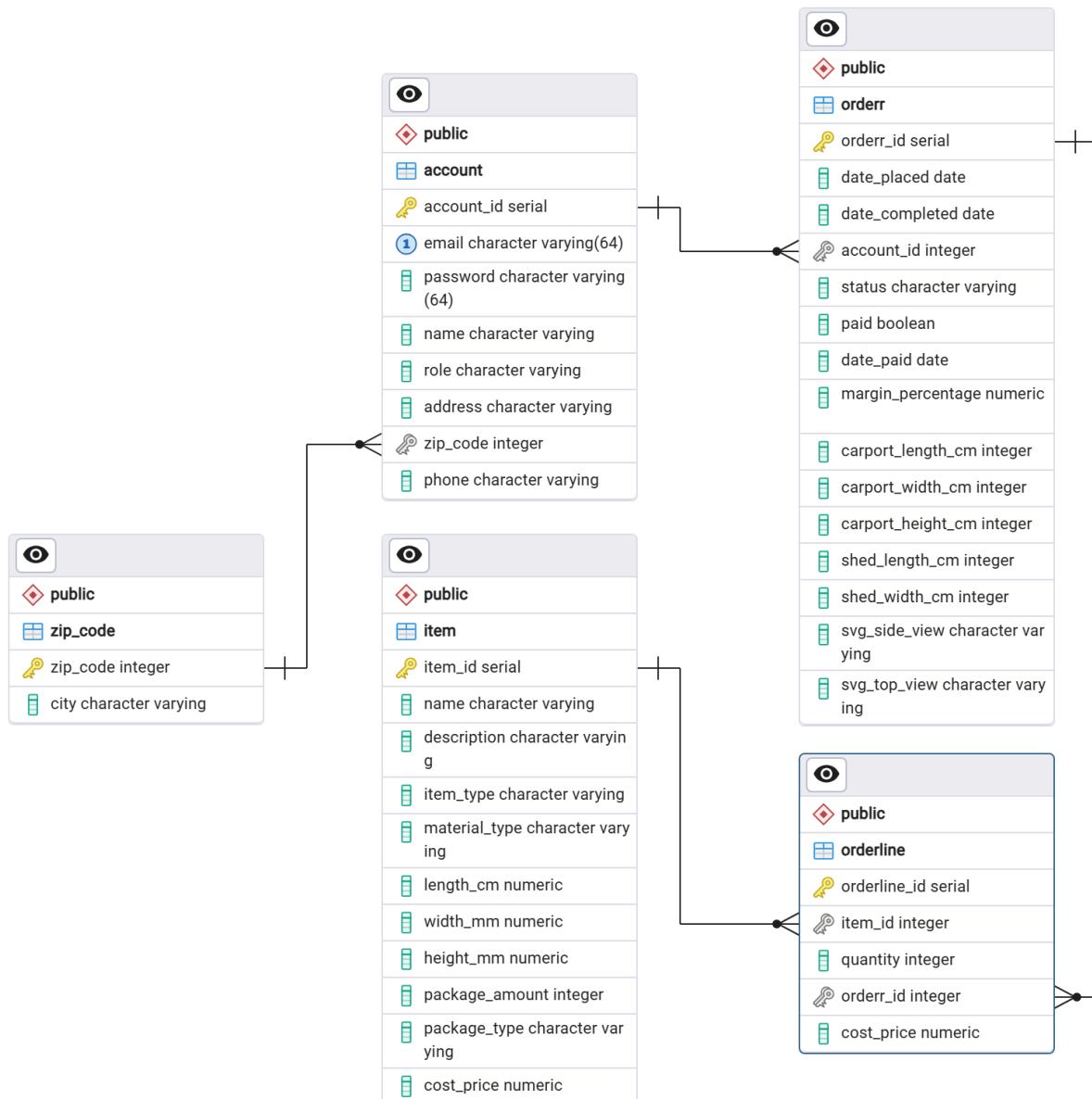
Det er et ønske fra Fog, at faktura for carportene skal udspecifiseres med pris⁴ for hver enkelt linje i styklisten. Linjerne i styklisten svarer derfor til ordrelinjer, som tilsammen udgør en ordre.

⁴ Bilag B - Logbog: Fredag d. 22 nov. 2024

Ordrer har en total kostpris og total salgspris, der skal beregnes ud fra kostpris og salgspris på de enkelte varer. Dog skal sælger have mulighed for at kunne justere dækningsgraden, f.eks. for at kunne sænke salgsprisen til en loyal kunde.

ER diagram

Efter lavet domænemodellen gik vi i gang med at udvikle et database design med udgangspunkt i domænemodellen. Vores database design er vist på figur 7. Bemærk at vores ordretabel er navngivet **orderr** med et ekstra r på. Dette er for at undgå konflikt med PostgreSQL's keyword "order".



Figur 7 - ER diagram for vores database.

Sammenligning med domænemodellen

Vi har designet vores database så den kan rumme domænemodellen, dog med nogle simplificeringer for at reducere vores scope:

- **account** tabellen indeholder kunder og sælgere, men ikke admin, da vi har valgt ikke at tage admin funktionaliteter med i vores scope.
- **orderr** tabellen indeholder generel ordreinfo (dato, status, osv.) og info specifik for carport ordre, såsom carport længde, bredde, højde og svg tegninger. I et rigtigt system ville det være forståeligt hvis Fog ønskede ikke at få carport specifik info ind i deres ordretabel. I så fald kunne man lave en separat carport tabel med en 1:1 relation til ordretabellen.
- **orderline** tabellen repræsenterer både ordrelinjerne og styklisten.
- **item** tabellen repræsenterer varerne i Fogs sortiment, dvs. deres eksisterende varedatabase. Vi har simplificeret ved kun at tage kostpris (uden moms) med og udelade salgspris.

3. normal form

Vores database overholder 3. normal form bortset fra hvad angår kostpris, som vi har med i både **item** tabellen og **orderline** tabellen.

Der er en god grund til at vi har kostpris med i begge tabeller: Fog skal løbende kunne justere kostpriserne i **item** pga. inflation og andre årsager. Fordi vi også har kostpris med i **orderline**, forbliver priserne i ordrehistorikken til den pris de er solgt til, uanset hvordan kostpriserne i **item** ændres.

Det er med vilje vi har valgt ikke at have kostpris med i **orderr** tabellen. Det ville være et brud på 3. normal form, og det ville betyde, at vi skulle sørge for at den kostpris altid er synkroniseret med summen af kostpriserne i **orderline**. I stedet har vi margin_percentage (dvs. dækningsgrad) med i **orderr** tabellen. Ud fra summen af kostpriserne i **orderline** og dækningsgraden beregner vores Java backend den totale salgspris.

Primærnøgler

Vi bruger automatisk genereret ID som primærnøgle i alle vores tabeller pånær **zip_code** tabellen. I denne tabel er primærnøglen zip_code (dvs. postnummer) hvilket er en naturlig nøgle.

Fremmednøgler

Relationerne imellem vores tabeller har vi etableret vha. fremmednøgler med foreign key constraints på. Foreign key constraints er smart, for det betyder, at når vores Java backend forsøger at indsætte en række i f.eks. **orderline** tabellen, så giver PostgreSQL kun lov til det hvis orderr_id rent faktisk eksisterer i **orderr** tabellen. Dette er med til at sikre, at vi har konsistente data i vores database.

Unique constraint

Vi har brugt en unique constraint på email attributten i account tabellen. For i vores web applikation skal brugere logge på med email og password, og dermed er det vigtigt at email unikt definerer en bruger.

Tabel som vi burde have haft

I vores web applikation har vi status strings hardcoded mange steder rundt omkring i forskellige filer. I bagklogskabens lys burde vores database have haft en **status** tabel, som vi burde have programmet vores løsning op omkring:

status_id	status_name
1	Henvendelse
2	Igangværende
3	Leveret
4	Afsluttet
5	Annuleret

Tidlig version af ER diagram

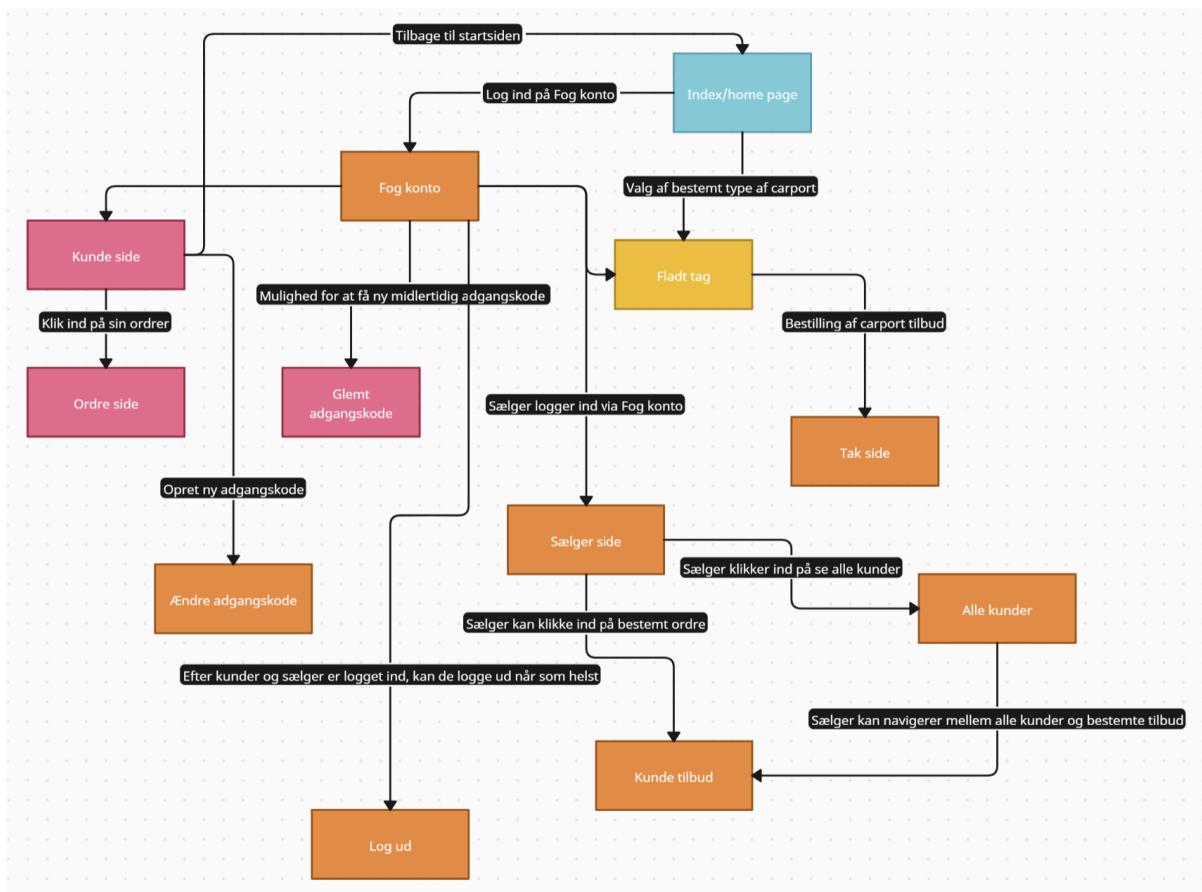
I bilag E kan man se en tidlig version af vores database design. Der er en del forskelle i forhold til vores endelige design på figur 7:

- Vi var ikke sikker på om **orderr** og **carport** skulle være to separate tabeller. Vi endte med at samle dem i tabellen **orderr**.
- Vi havde ikke tænkt over at SVG tegningerne skulle lagres i databasen. Vi endte med at lagre dem som "svg_side_view" og "svg_top_view" i **orderr** tabellen.
- Vi havde kostpris og salgspris med i både **item** og **orderline**, og slet ikke noget med pris i **orderr**. Vi endte med at have kostpris med i **item** og **orderline**, og så have "margin_percentage" (dvs. dækningsgrad) med i **orderr**.
- Endvidere fik vi tilføjet en del ekstra parametre til **item** tabellen, for bedre at kunne beskrive de forskellige varer.

Navigationsdiagram & Mockups

Diagrammet (figur 8) for det samlede navigationsdiagram er et samlet overblik over vores web applikation. Diagrammet viser det flow, vi har skabt i vores program for både kundeoplevelsen og sælger oplevelsen. Vi har en kunde header som viser Fog logoet og har nogle links hvorpå to af dem henviser til andre html sider. Vi har valgt at tilføje "dummy links" for at vise at vores web applikation kan integreres i Fogs allerede eksisterende hjemmeside.

Det samlede navigationsdiagram viser hvordan kunden kan navigere sig rundt i vores web applikation og hvilke muligheder kunden har i forhold til valg af custom carporte og hvad kunden har at se på sin egen kundekonto side. Diagrammet viser også hvordan vores web app er bygget op for sælger, herunder hvad sælgeren kan se på sælgersiden og hvordan sælger nemt kan navigere sig rundt mellem siderne og klikke ind på en bestemt kunde ordre og redigere det efter behov. Herunder kan sælger sende kundens log ind informationer som gør det muligt for en ny kunde, selv at logge ind på deres konto og se deres henvendelse som de har oprettet.



Figur 8 - Samlet navigationsdiagram

Fra Fog Konto kan både kunde og sælge, logge ind og blive henvist til deres sider som er gældende for enten kunde (figur 9 & 10) eller sælger (figur 11). Logger man ind som sælger, har vi oprettet en anden header som giver sælger mulighed for at se alle kunder og alle ordrer i systemet.



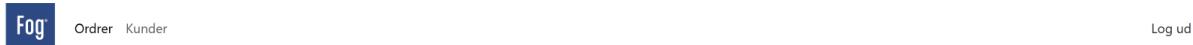
Figur 9 - Header til kunde

Vi har valgt at vise kundens mail og tilføje muligheden for at ændre adgangskode i headeren når kunden er logget ind på deres side, samt tilføje en log ud knap.



Figur 10 - Header til kunde efter login

Sælger har mulighed for at navigere sig rundt mellem ordre siden og alle kunder siden, samt klikke sig ind på en bestemt kundeordre, og herinde har mulighed for at redigere i tilbuddet.



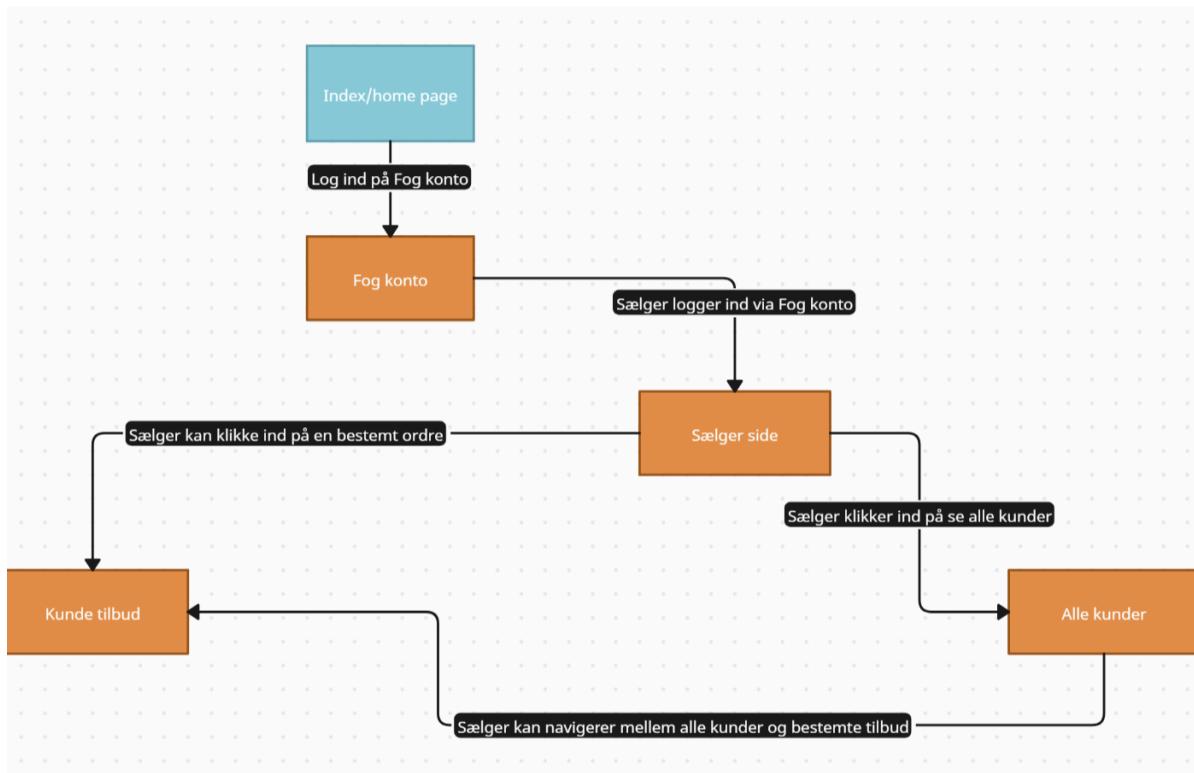
Figur 11 - Header til sælger

Vi har valgt at oprette en fælles footer som går igen på alle sider som kun viser at det er en skoleopgave og ikke den rigtige Fog hjemmeside.



Figur 12 - Footer

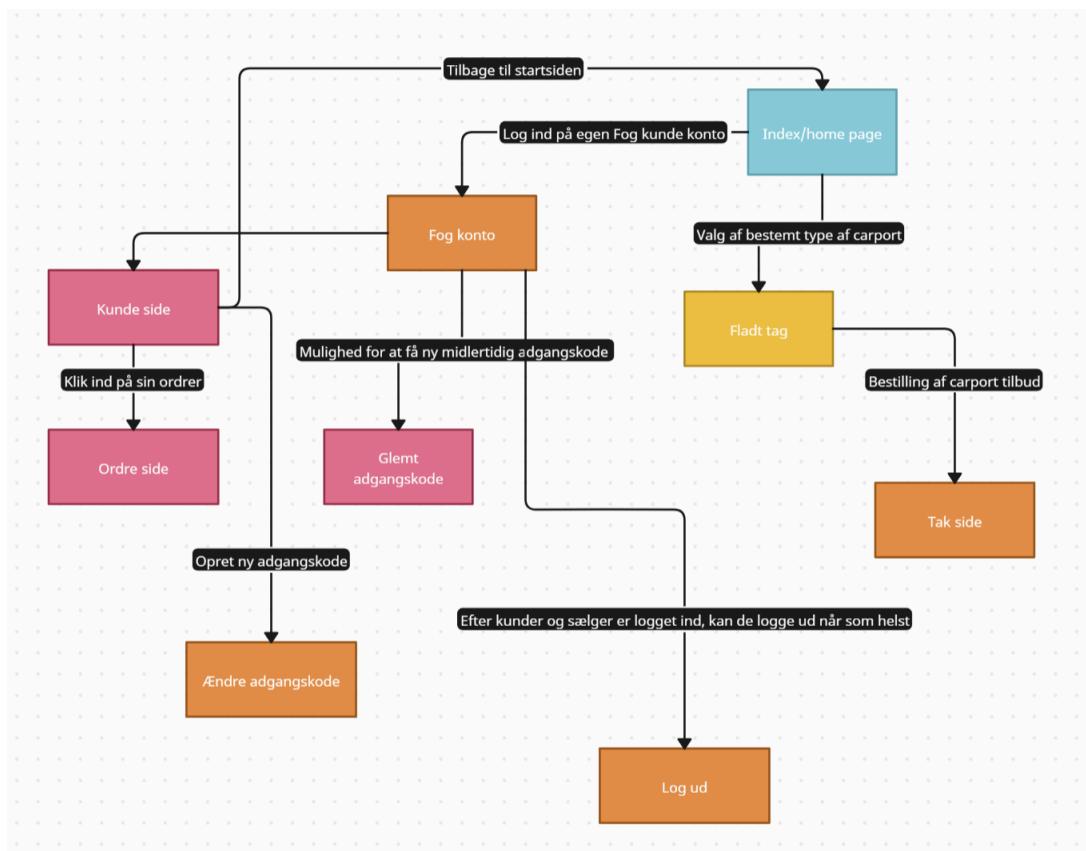
Diagrammet på figur 13 viser flowet igennem vores web applikation set ud fra sælgers synspunkt. Her vises der hvordan sælger, efter at sælger er logget ind i systemet, har to sider at nавigerer sig rundt imellem, samt hvordan inde på ordre siden har mulighed for at klikke ind på en bestemt kundens ordre og redigerer efter behov og sende tilbuddet til kunden, samt de nødvendige log ind informationer til kunden.



Figur 13 - Diagram over sælgers flow

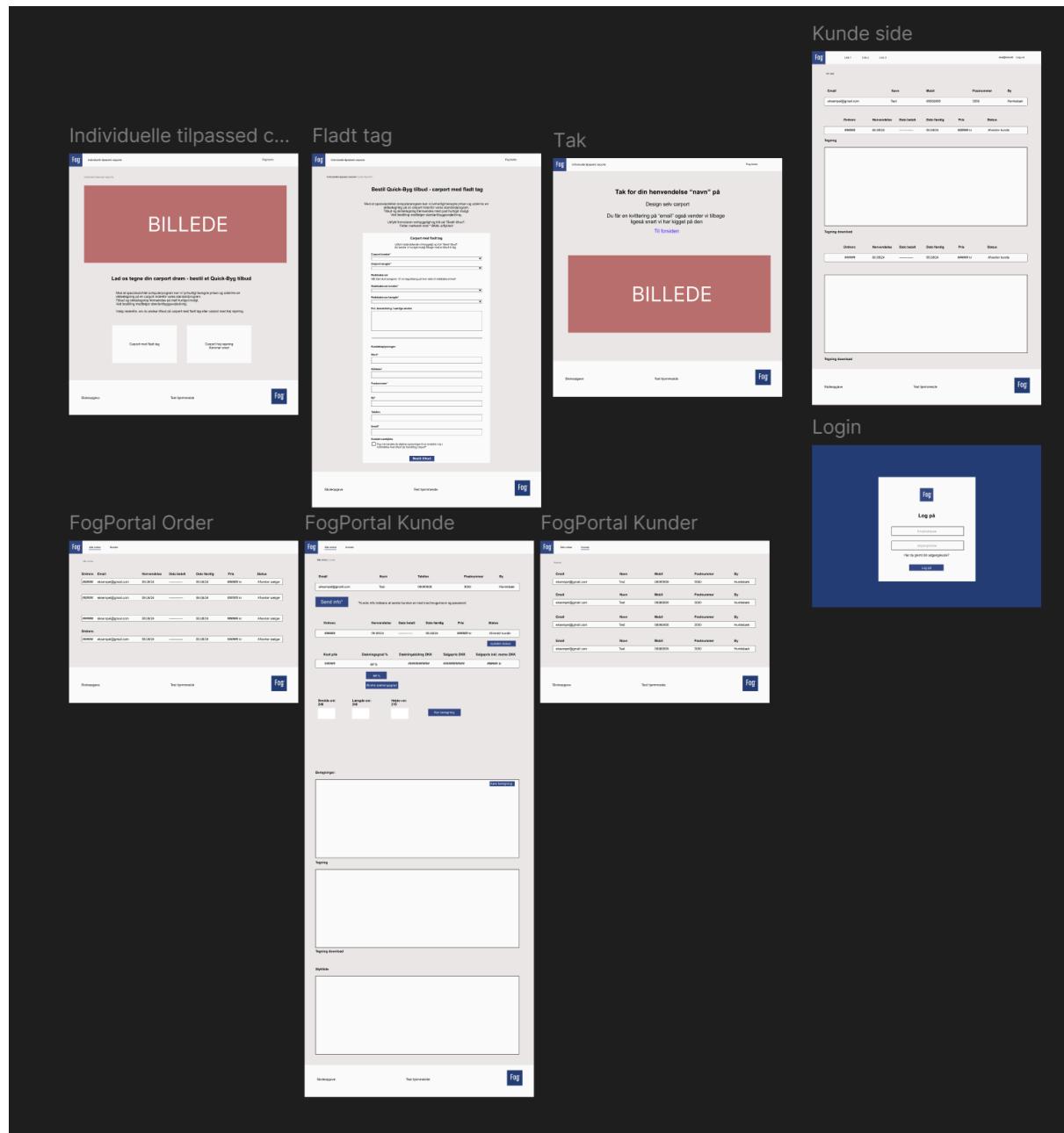
Diagrammet (figur 14) illustrerer kundeoplevelsen i vores webapplikation og viser kundens rejse gennem platformen. Når kunden har valgt deres custom carport, kan de logge ind på deres kundekonto for at se det tilbud, de har oprettet. Dette kræver dog, at sælgeren har sendt kundelogin til kunden.

Diagrammet viser også hvordan de har mulighed for at ændre adgangskode efter de er logget ind på deres kundeside. På deres kundeside vises alle deres ordrer, hvor kunden har mulighed for at klikke sig ind på netop den ordre, de ønsker at se. Her vises alle tegner over deres custom carport, samt efter betalingen er gået igennem, vises styklisten i bunden af deres ordreside.



Figur 14 - Diagram over kundens flow

På figur 15⁵ ses der samlet overblik over vores Figma mockups over nogle af vores html sider, som vi har arbejdet ud fra dets design til de gældende html sider, samt de html sider som vi ikke fik lavet mockups af. Vi har taget udgangspunkt i Fogs eksisterende design, da vi fra starten har tænkt, at vores design skal kunne integreres ind på Fogs hjemmeside. Til de html sider hvor vi ikke nået at oprette nye mockups til, har vi taget udgangspunkt i det design vi har lavet til de andre html sider. Herved har vi skabt et fælles design som går igennem hele vores web applikation.

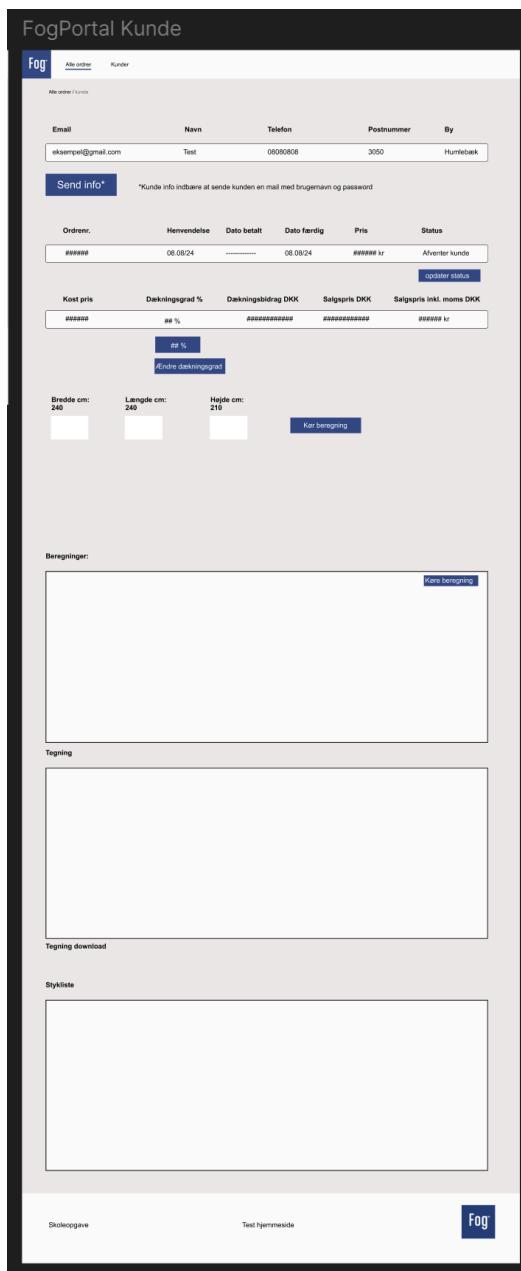


Figur 15 - Figma mockup af hele designet

⁵ Større billede kan ses på GitHub repository

På figur 16 ses et eksempel på Figma mockup, eksemplet viser en kundeordre som sælgeren kan se dem efter de har klikket ind på netop den ordre de vil arbejde med. Her viser vi hvordan sælger kan sende de nødvendige kundeoplysninger til kunden, så kunden har mulighed for at logge ind på deres egen side og se deres tilbud. Her viser vi også hvordan sælgeren har mulighed for at tage kontakt til kunden hvis der skal ændres i tilbuddet og sende de nye beregninger af sted til kunden. Kontakten er fortsat i form af telefon.

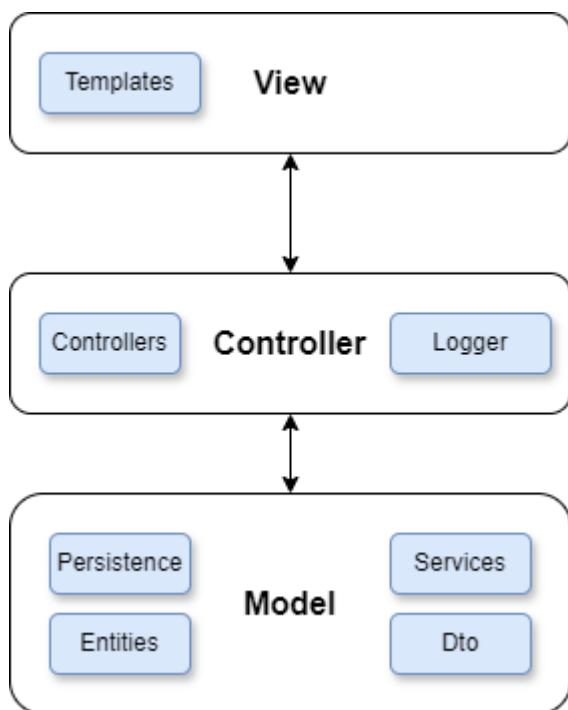
Til sidst viser vi tegningerne af carporten, set fra siden og set fra toppen, samt den samlede stykliste af carporten.



Figur 16 - Figma mockup af kundeordre

Valg af arkitektur

Til udviklingen af vores system har vi valgt en MVC-inspireret arkitektur (Model-View-Controller). Vi har valgt MVC, da det er en simpel arkitektur, der er tilstrækkelig til små og mellemstore projekter. Derudover har alle i teamet erfaring med denne arkitektur, hvilket er en fordel. MVC giver en klar adskillelse mellem logik, databehandling og præsentation, hvilket gør systemet lettere at vedligeholde. På figur 17 ses hvordan vores packages hører til i MVC strukturen.



Figur 17 - Diagram over hvor vores packages høre til i MVC arkitekturen

I vores implementering anvender vi desuden designmønsteret Singleton til vores logger for at sikre, at der kun eksisterer én instans af loggeren gennem hele systemet. Vi benytter serviceklasser til at håndtere specifikke opgaver som carport beregning og SVG generering. Dette hjælper med at opretholde en klar ansvarsfordeling mellem klasserne.

Packages og deres funktioner:

1. **app:** Indeholder main-klassen, som er indgangspunktet til applikationen. Den håndterer opstarten af Javalin-serveren og initialisering af connection pool, som sikrer effektiv databaseadgang. Denne package indeholder alle de nedenstående packages.
2. **controllers:** Indeholder controller-klasser, der håndterer HTTP-forespørgsler og kommunikerer mellem view og model.

3. **persistence:** Indeholder mapper-klasser der fungerer som dataadgangslag. De håndterer forespørgsler og opdateringer i databasen og sikrer adskillelse af applikationslogikken.
4. **services:** Indeholder serviceklasser som SvgEngine og Calculator, der håndterer specifikke opgaver såsom generering af SVG-diagrammer til visualisering af carporte og beregning af materialebehov.
5. **entities:** Indeholder datamodeller der repræsenterer de grundlæggende domæneobjekter i systemet. Disse klasser bruges som transportobjekter mellem mapper og controllers.
6. **dto:** Indeholder Data Transfer Objects, der bruges til at kombinere data fra flere tabeller og præsentere det i et samlet format til visning eller videre behandling.
7. **exceptions:** Indeholder brugerdefinerede undtagelser for at sikre en mere præcis fejlhåndtering og gøre fejlårsager nemmere at identificere.
8. **templates:** Indeholder vores HTML Thymeleaf templates.

Særlige forhold & Udvalgte kodeeksempler

Session Management i Webapplikationen

I vores webapplikation gemmer vi alt data direkte i databasen med det samme for at sikre konsistens og holdbarhed. Dog anvender vi **session scope** til at opbevare enkelte oplysninger midlertidigt, såsom brugerinformationen, som vi kalder "account". Her sender vi kun det absolut mest nødvendige data rundt, hvilket afspejler sig i antallet af konstruktører, vi har for de forskellige entiteter, et eksempel ses på figur 18.

```
if (role.equals("salesrep")) {  
    account = new Account(accountId, role);  
} else if (role.equals("Kunde")) {  
    account = new Account(accountId, email, name, role, address, city, phone);  
}
```

Figur 18 - Eksempel på at vi bruger to forskellige Account constructors

Derudover gemmer vi midlertidige attributter i sessionen for at kunne vise dem med Thymeleaf. Dette bruges eksempelvis til fejlbeskeder ved databasefejl eller lignende situationer.

Exception handling og Logging

I starten af vores proces havde vi en simpel exception handling-strategi, hvor alle fejl blev skrevet direkte til konsollen. Vi begyndte dog senere at implementere en mere struktureret tilgang ved at introducere logging.

Vi valgte at:

1. Logge engelske beskeder for at give programmører teknisk information.
2. Kaste danske beskeder videre til controllerne, så de kunne sendes til brugerne via vores errorpage (error.html).

I mapperne logger vi fejlen med en engelsk besked ved hjælp af **LOGGER.SEVERE()** (figur 19) og kaster en ny exception med den danske besked. I controllerne fanger vi denne exception og sørger for, at den danske besked bliver sendt videre til brugeren.

```

        return new DetailOrderAccountDto(orderId, accountId, email, name, phone, zip, city, da
    }
    LOGGER.severe( msg: "Error happen in showCustomerOrder()");
    throw new DatabaseException("Fejl ved hentning af ordrenr: " + orderId);
} catch (SQLException e) {
    LOGGER.severe( msg: "Error happen in showCustomerOrder() e.Message: " + e.getMessage());
    throw new DatabaseException("Fejl ved hentning af ordrenr: " + orderId);
}
}
}

```

Figur 19 - Eksempel på hvordan vi i en mapper logger og kaster exception

På figur 20 kan man se hvordan vi fanger exceptionen på dansk og sender den ud til vores errorpage.

```

private static void salesrepShowOrderPage(Context ctx, ConnectionPool connectionPool) { 1 usage  ± Peter Bolhorn +1
    Account activeAccount = ctx.sessionAttribute( key: "account");
    if (activeAccount == null || !"salesrep".equalsIgnoreCase(activeAccount.getRole())) {
        LOGGER.warning( msg: "Uautoriseret adgangsforsøg til ordresiden for sælgere. Rolle: " +
            (activeAccount != null ? activeAccount.getRole() : "Ingen konto"));
        ctx.attribute("errorMessage", "Kun adgang for sælgere.");
        ctx.render( filePath: "error.html");
        return;
    }

    int orderId = Integer.parseInt(ctx.queryParam( key: "ordrenr"));

    try {
        DetailOrderAccountDto detailOrderAccountDto = OrderMapper.getDetailOrderAccountDtoByOrderId(orderId, connectionPool);
        String role = activeAccount.getRole();
        ArrayList<Orderline> orderlines = OrderlineMapper.getOrderlinesForCustomerOrSalesrep(orderId, role, connectionPool);

        ctx.attribute("detailOrderAccountDto", detailOrderAccountDto);
        ctx.attribute("orderlines", orderlines);
        ctx.render( filePath: "sælgerordre.html");

    } catch (DatabaseException | OrderException e) {
        LOGGER.severe(e.getMessage());
        ctx.attribute("errorMessage", e.getMessage());
        ctx.render( filePath: "error.html");
    }
}

```

Figur 20 - Eksempel på hvordan vi fanger en exception og sender dens message ud til vores errorpage

Da logging blev implementeret sent i projektet, endte vi med en udfordring: Vi logger ofte både i mapper og controller. Vi overvejede at forenkle ved kun at logge i controllerne, da alle forespørgsler alligevel passerer gennem dem. Denne tilgang ville gøre at vi ville undgå dobbelt-logging.

DTO (Data Transfer Objects)

I eksemplet ovenover (figur 20) bruger vi DTO'er, fordi vores views i frontend skal vise specifikke data, som ofte er en kombination af forskellige entiteter. I stedet for at sende flere entiteter med unødvendig data frem og tilbage, laver vi en DTO, der indeholder kun præcis det data, vi har brug for. Dette sikrer at der er mindre datatrafik mellem backend og frontend og der er bedre struktur og lettere vedligeholdelse.

På figur 21 ses et eksempel på hvordan vi henter data fra forskellige data tabeller for at lave en DTO:

```
String sql = "SELECT orderr_id, account_id, email, name, phone, zip_code, city, date_placed, date_paid, date_completed, " +
    "margin_percentage, status, carport_length_cm, carport_width_cm, carport_height_cm, svg_side_view, svg_top_view, " +
    "(SELECT SUM(cost_price) FROM orderline WHERE orderline.order_id=orderr_id) " +
    "FROM orderr JOIN account USING(account_id) JOIN zip_code USING(zip_code) WHERE orderr_id = ?";
```

Figur 21 - SQL query som henter data fra 4 forskellige tabeller for at bygge vores DetailOrderAccountDto

Guard Condition

For at sikre, at brugerne kun har adgang til de data og funktioner, som deres rolle tillader, har vi implementeret guard conditions på alle endpoints (figur 22). Dette forhindrer uautoriseret adgang og sikrer en klar adskillelse af brugerrettigheder.

```
public static void showCustomerOrderPage(Context ctx, ConnectionPool connectionPool) { 1 usage  ± Peter Bolhorn +1
    Account activeAccount = ctx.sessionAttribute("key: "account");
    if (activeAccount == null || !"Kunde".equalsIgnoreCase(activeAccount.getRole())) {
        ctx.attribute("errorMessage", "Du er ikke logget ind");
        ctx.render("filePath: "error.html");
        return;
    }

    try {
        int orderId = Integer.parseInt(ctx.queryParam("key: "ordrenr"));

        DetailOrderAccountDto detailOrderAccountDto = OrderMapper.getDetailOrderAccountByOrderId(orderId, connectionPool);
        if (activeAccount.getAccountId() != detailOrderAccountDto.getAccountId()) {
            ctx.attribute("errorMessage", "Du har ikke adgang til at se denne ordre");
            ctx.render("filePath: "error.html");
            return;
        }
    }
}
```

Figur 22 - Metode med to guard conditions: Først checker vi om brugeren er logget ind som kunde. Dernæst checker vi om ordren tilhører denne kunde.

Vi arbejder med tre roller på hjemmesiden:

1. Kunde uden login
2. Kunde (logged in)
3. salesrep

Rollen bliver defineret ved login og afgør, hvilke adgangsrettigheder brugeren har:

- Kunde uden login: Har begrænset adgang, fx mulighed for at se og navigere hjemmesiden.
- Kunde: Kan tilgå og manipulere egne data, såsom ordreoversigt, profilinformation og opdateringer.
- salesrep: Har adgang til kundeinformation og ordredata, som er nødvendigt for deres arbejde.

Dette sikrer, at hver rolle kun kan tilgå og manipulere specifikke data, hvilket styrker både sikkerheden og dataintegriteten i systemet.

Password generator

I vores projekt har vi udviklet vores egen password generator, som bliver brugt til at autogenerere passwords til nyoprettede kunder.

Generatoren (figur 23) sikrer, at det oprettede password indeholder: Store bogstaver, små bogstaver , tal og specialtegn.

For at opnå dette genereres et password med minimum ét tegn fra hver kategori, hvorefter ekstra tegn tilføjes for at nå den ønskede længde. Afslutningsvis shuffles tegnene for at opnå et tilfældigt resultat.

Klassen er programmet, så det er nemt at tilpasse tegnsættet og længden, hvis der er behov for specifikke krav til sikkerheden.

```
public class PasswordGenerator { 6 usages ± Rolf
    private static final String UPPERCASE_CHARS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; 2 usages
    private static final String LOWERCASE_CHARS = "abcdefghijklmnopqrstuvwxyz"; 2 usages
    private static final String NUMBER_CHARS = "0123456789"; 2 usages
    private static final String SPECIAL_CHARS = "!@#$%^&*()_+=[]{}|;,.>?"; 2 usages

    private static final String ALL_CHARS = 1 usage
        UPPERCASE_CHARS + LOWERCASE_CHARS + NUMBER_CHARS + SPECIAL_CHARS;

    private static final SecureRandom random = new SecureRandom(); 2 usages

    public static String generatePassword() { 3 usages ± Rolf
        StringBuilder password = new StringBuilder();

        password.append(getRandomChar(UPPERCASE_CHARS));
        password.append(getRandomChar(LOWERCASE_CHARS));
        password.append(getRandomChar(NUMBER_CHARS));
        password.append(getRandomChar(SPECIAL_CHARS));

        // password length should be 6, so add 2 more characters
        for (int i = 0; i < 2; i++) {
            password.append(getRandomChar(ALL_CHARS));
        }

        return shufflechars(password.toString());
    }

    private static char getRandomChar(String charset) { return charset.charAt(random.nextInt(charset.length())); }

    private static String shufflechars(String input) { 1 usage ± Rolf
        char[] characters = input.toCharArray();
        for (int i = characters.length - 1; i > 0; i--) {
            int j = random.nextInt(bound: i + 1);
            char temp = characters[i];
            characters[i] = characters[j];
            characters[j] = temp;
        }
        return new String(characters);
    }
}
```

Figur 23 - Vores PasswordGenerator. Den er en serviceklasse og ligger derfor i pakken "services".

Beregningsmotor

Beregningsmotoren er designet sådan, at hver del af carporten beregnes separat i hovedmetoden calculateStructure() i klassen CarportCalculationStrategy. For eksempel beregnes spærne via metoden getRafterAndCalculate() (figur 24):

```
try {
    //***** Remme *****
    getBeamAndCalculate(carport);

    //***** Spær *****
    getRafterAndCalculate(carport);

    //***** Stolper *****
}
```

Figur 24 - Ruter ind til de individuelle beregnings dele for carporten

I getRafterAndCalculate() metoden bliver et builder-objekt brugt til at oprette et søgefiter, der indeholder parametre som et materiale kan have. Det kunne for eksempel være en **itemType** "spær", en materiale **length_cm** som kunne være lige med carport længden, eller en **cost_price** og meget mere. Dette filter sendes til vores ItemMapper klasse, som returnerer et materiale, der i dette tilfælde er et spær-materiale (rafterMaterial). Dette materiale bruges derefter i metoden calculateRafters() (figur 25).

Builder-objektet er ganske enkelt til for at kunne lave en dynamisk søgning, i stedet for at bruge 10 overloaded metoder. Måden det sker på er at vi sætter nogle værdier som vi vil søge efter, for eksempel item_id, length_cm, material_type mm. Alle værdier er default null, med mindre de bliver sat. I build metoden tilføjer vi en key, som er database kolonnens navn, og en value, som vi lige har sat, til en hashmap, som er det vi vil søge efter. Den hashmap bliver så sendt til ItemMapper, som bruger de parametre til at søge efter et material.

```
private void getRafterAndCalculate(Carport carport) throws DatabaseException { 1 usage  Lasse Jensen

    ItemSearchBuilder builderRafter = new ItemSearchBuilder();
    Map<String, Object> filtersRafter = builderRafter
        .setItem("Spær")
        .setWidthMm(38)
        .setHeightMm(73)
        .setLengthCm(carport.getWidth())
        .build();
    Material rafterMaterial = ItemMapper.searchSingleItem(filtersRafter, pool);

    int amount = calculateRafters(carport, rafterMaterial);
    getRafterBracketsAndCalculate(carport, amount);
}
```

Figur 25 - Eksempel på metode som får et material fra mapperen, og sender til beregning

I calculateRafters() (Figur 26) bliver antallet af spær beregnet, og samtidig bestemmes deres positioner. Når positionen for et materiale er fastlagt, oprettes et PlacedMaterial-objekt. Et PlacedMaterial repræsenterer en material entitet, med en specifik placering i xyz-rummet. For at undgå problemer med at flere PlacedMaterial deler den samme material entitet, laves en klon af materialet for hvert nyt PlacedMaterial. På den måde undgår man, at ændringer til ét objekt påvirker alle andre.

```
private int calculateRafters(Carport carport, Material rafterMaterial) { 1 usage ▾ Lasse Jensen

    int amountOfRaftersX = carport.getLength() / rafterDistance;
    int totalAmount = 0;
    float totalRafterWidth = amountOfRaftersX * rafterMaterial.getWidthCm();
    float spacing = (carport.getLength() - totalRafterWidth) / (amountOfRaftersX - 1);

    for (int k = 0; k < amountOfRaftersX; k++) {
        float x = k * (rafterMaterial.getWidthCm() + spacing);

        Material clonedMaterial = rafterMaterial.cloneMaterial(rafterMaterial);
        clonedMaterial.setLengthCm(carport.getWidth()); // Adjust length to fit carport width
        PlacedMaterial placedRafter = new PlacedMaterial(clonedMaterial, x, y: 0, z: 0);

        rotateAroundZ(clonedMaterial);
        rotateAroundY(clonedMaterial);
        placedMaterialList.add(placedRafter);
        totalAmount++;
    }
    calculatePartsList(carport, rafterMaterial, totalAmount);
    return totalAmount;
}
```

Figur 26 - Eksempel fra selve beregningen af en del af carporten

Denne opdeling af opgaver sikrer, at hver del af carporten beregnes enkeltvis og placeres præcist, samtidig med at man undgår utilsigtede ændringer i de enkelte materialer gennem kloning. Alt dette beregning er kun til at beregne materialernes placering til SVG motoren. Måden beregnings motoren er lavet på, giver også mulighed for nemt at udvide til 3D tegninger på et senere tidspunkt. Ud over placering af materialerne, bliver materialerne også tilføjet til en stykliste via calculatePartsList. Dette er for at lave en klar opdeling mellem styklisten, som også indeholder beslag, skruer mm., og placeret materialer til SVG tegning, som ikke skal indeholde ting som beslag og skruer. Derfor fremgår spær både i listen over placerede materialer, og styklisten. Hvert carport indeholder sin egen stykliste, samt en liste over materialer som skal placeres i SVG tegningen.

Det var ikke alt i beregningsmotoren som vi nåede at lave, som for eksempel at

- Carport kunne indeholde et skur
- At kunne have tag med rejsning

- At beregningsmotoren blev bygget på en sådan måde at skuret senere kunne være standalone hvis det blev til et ønske

Grundet tidspress og fordi vi skulle fokusere mere på minimal viable product er disse ting ikke blevet færdiggjort, men er stadig i koden som skelet for strukturen generelt. Der blev indført mulighed for at skifte beregningsmodel under runtime, men dette er der ikke gjort brug af. Funktionaliteten ligger dog klar til brug.

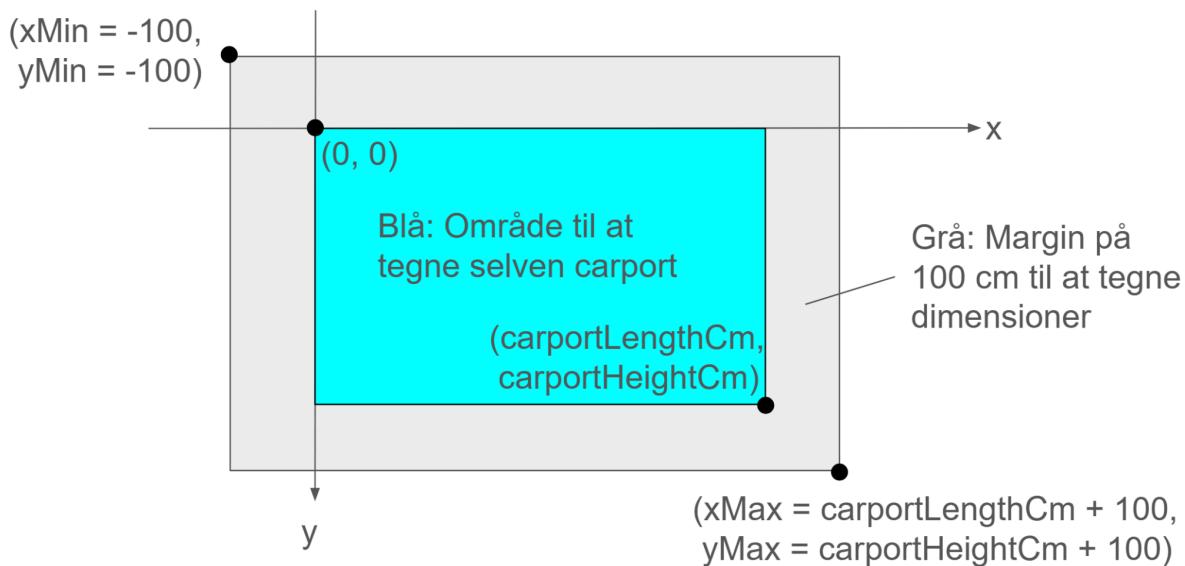
SVG motor (tegningsmotor)

SVG motoren er implementeret i pakken `svgEngine`. Det der skal fremhæves her er constructoren for `Svg` klassen, som har `xMin`, `yMin`, `xMax` og `yMax` som argumenter, hvilket definerer tegningsområdet (figur 27):

```
Svg svg = new Svg( xMin: -100, yMin: -100, xMax: carportLengthCm + 100, yMax: carportHeightCm + 100);
```

Figur 27 - Kald af `Svg` klassens constructor som definerer tegningsområdet.

For tegning af carporten set fra siden vælges `xMin`, `yMin`, `xMax` og `yMax` som vist på figur 27, hvilket giver tegningsområdet som vist på figur 28:



Figur 28 - Tegningsområde for SVG tegning af carport set fra siden

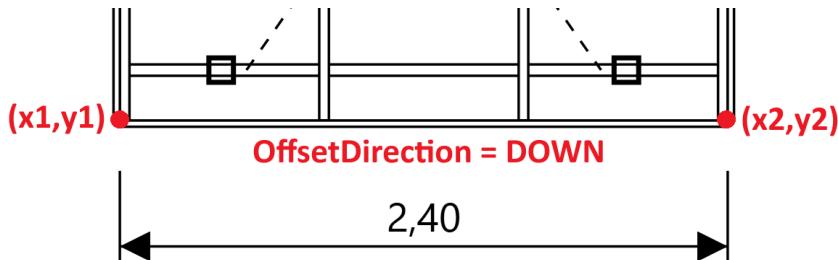
Det smarte ved at vi vælger dette tegningsområde er, at vi får det blå område til at tegne selve carporten i (har koordinater som er nemme at arbejde med) og det grå område til at tegne dimensioner i. Vel og mærke i en enkelt SVG tegning. Vi startede med at tegne en SVG i en SVG, men ulempen var at halvdelen af kanttykkelsen på de brædder der lå helt ude i kanten på den inderste SVG ikke blev vist.

En anden ting der skal fremhæves er Svg klassens addDimension metode (figur 29):

```
public void addDimension(double x1, double y1, double x2, double y2, OffsetDirection offsetDirection)
```

Figur 29 - Metodesignatur for Svg klassens addDimension metode.

Denne metode tager koordinaterne x1, y1, x2, y2 samt en OffsetDirection (UP, DOWN, LEFT, RIGHT) og sørger så for at tegne alt for dimensionen inklusiv extension lines og målet i meter (figur 30):



Figur 30 - Dimension tegnet vha. Svg klassens addDimension metode

Automatiserede tests

Det smarte ved vores automatiserede tests er at der er lavet en **abstract** klasse (figur 31) hvor de andre test mapper klasser kan nedarve fra, dvs. at vi kan holde os til DRY (Don't Repeat Yourself) principperne og kun skrive en metode til indsætning af data i test databasen, som kan kører inden hver testklasse.

```
private static final String USER =           1 usage
private static final String PASSWORD =        1 usage
private static final String URL = "jdbc:postgresql://localhost:5432/%s?currentSchema=test"; 1 usage
private static final String DB = "fog"; 1 usage

protected static final ConnectionPool connectionPool = ConnectionPool.getInstance(USER, PASSWORD, URL, DB); 20 usages

@BeforeEach @Rolf
public void setUpDatabase() {
    try (Connection connection = connectionPool.getConnection();
        Statement stmt = connection.createStatement()) {

        // Drop tables and sequences
        stmt.execute("DROP TABLE IF EXISTS orderline CASCADE");
        stmt.execute("DROP TABLE IF EXISTS orderr CASCADE");
        stmt.execute("DROP TABLE IF EXISTS account CASCADE");
        stmt.execute("DROP TABLE IF EXISTS item CASCADE");
        stmt.execute("DROP TABLE IF EXISTS zip_code CASCADE");

        // Create tables
    }
}
```

Figur 31 - Start på abstract test klasse med indsættelse af tabeller

Status på implementering

Vi har nået alle "Martins" krav. Vi har startet med at scope småt for at fokusere på de kerneydelser web applikationen skulle kunne. P0 er høj prioritet og P2 er laveste prioritet. Man kan se de fulde user stories i bilag B.

Grøn = fuldt implementeret

Orange = delvist implementeret

Rød = ikke implementeret.

User Story:	Acceptance:	Status:
US1 - Indtast ønsket carport og få tilbud (Prioritet P0)	Givet at jeg er inde på custom carporte siden Når jeg vælge mine mål Så kan jeg sende informationer ind til Fog.	Fuld implementeret, det eneste er dog at der ikke bliver sendt en rigtigt email, men den bliver printet ud til konsollen.
US2 - Sælger og Kunde side, så de kan ændre password (Prioritet P2)	Givet at jeg er logget ind i systemet som sælger/kunde Når jeg klikker på ændre adgangskode Så har jeg mulighed for at ændre adgangskoden.	Der mangler implementation af sælgers side hvor de ville kunne skiftet kodeord.
US3 - Sælger se carport henvendelser (Prioritet P1)	Givet at jeg logger ind som sælger Når jeg er logget ind Så kan jeg se oversigt over alle henvendelser.	Fuld implementeret, når sælgeren logger ind, kan personen, se alle henvendelse, uanset status.
US4 - Tak for din henvendelse (Prioritet P1)	Givet at man er en person som ikke er logget ind Når man har trykker send på sin henvendelse Så får man vist en Tak side og får en kvittering på "mail".	Fuld implementeret, når man indsender carporte informationer bliver man sendt til en tak side, hvor man får vist sit navn og mail, samt at der bliver "sendt" en kvittering "på mail", hvilket er at det bliver printet til konsollen.
US5 - Sælger, klik på henvendelse, og se beregninger, stykliste osv. (Prioritet P0)	Givet at jeg er logget ind i systemet som sælger Når jeg klikker på en henvendelse fra en kunde Så jeg kan se beregningerne, styklisten, tegninger samt se og ændre dækningsgraden.	Fuld implementeret, sælgeren kan ind på hver enkelt ordre og se alle detaljer omkring given ordre.
US6 - Sælger ændre beregninger/andet tilbud	Givet at jeg er logget ind som sælger	Fuld implementeret, sælgeren kan inde på hver enkelt ordre og ændre i dimensionerne for at få

User Story:	Acceptance:	Status:
(Prioritet P0)	Når jeg klikker på en henvendelse Så kan jeg ændre i tilbuddets pris og se opdateret dækningsbidrag.	lavet nye tegninger og dækningsgraden.
US7 - Bruger login (Prioritet P1)	Givet at jeg ikke er logget ind og er oprettet i systemet Når jeg logger ind Så får jeg adgang til mine passende funktioner(Sælger/kunde).	Fuld implementeret, både kunder og sælger kan logge ind og få tildelt deres rolle, så de har adgang til deres givne sider.
US8 - Sælger ændre status på ordre (Prioritet P1)	Givet jeg er logget ind i systemet som sælger Når jeg er under en henvendelse så ville jeg kunne ændre på statussen.	Fuld implementeret, sælgeren kan under en bestemt ordre ændre statussen på den ordre.
US9 - Se startside for custom carporte (Prioritet P1)	Givet at jeg som uregistreret kunder/kunde finder ind på startsiden Når jeg er der inde Så kan jeg se noget intro tekst omkring carporte og vælge faldt tag carport.	Fuld implementeret, det er i vores tilfælde startsiden og der kan man uanset rolle klikke sig ind videre.
US10 - Se alle kunder (Prioritet P2)	Givet jeg er logger ind i systemet som sælger Når jeg trykker log ind Så kan jeg klikke mig ind på en side hvor jeg kan se alle kunder.	Fuld implementeret, når man er logget ind kan man i menuen klikke sig ind og se alle registrerede kunder.
US11 - Send konto info til kunde (Prioritet P2)	Givet at jeg er logget ind i systemet som sælger Når jeg klikker ind på en henvendelse Så jeg kan trykke på en knap så kunden får tilsendt sine konto information	Fuld implementeret, når sælgeren er inde på en ordre, er der en knap som "sender" kundens info til personen, med en bestemt om tilbuddet er klar. Dog er det stadig ikke en rigtig mail, men det bliver udskrevet til konsollen.

User Story:	Acceptance:	Status:
US12 - Som kunde vil jeg kunne se min side (Prioritet P1)	Givet at som kunde har bestilt et tilbud og har snakket med sælger Når jeg får mine login informationer Så ville jeg kunne logge ind og kigge på mit tilbud.	Fuld implementeret, når sælger har været inde og "sende" kundens info, så kan kunden logge ind, men hvis kunden allerede er kunde, så ligge ordre på personens side med begrænset informationer.
US13 - Markere som sælger hvilken henvendelsen jeg arbejder med (Prioritet P2)	Givet at jeg er en sælger logget ind i systemet Når jeg trykke mig på en opgave Så kan de andre sælger se at jeg er på opgaven, så de ikke gør noget på den.	Ikke berørt.
US14 - Sælger kan sende rigtig emails til kunden (Prioritet P2) Nice to have	Givet at jeg er en sælger logget ind i systemet Når jeg trykker på send kundeinfo Så bliver en mail sendt til kunden med alle deres oplysninger.	Ikke berørt.
US15 - Custom carport med tagrejsning (Prioritet P2) Nice to have	Givet at jeg er inde på custom carporte siden Når jeg vælge mellem fladt eller rejsning på taget Så kan jeg sende informationerne ind til fog.	Ikke berørt.

Derudover har vi ikke nået at få lavet integrationstest til alle vores Mapper metoder og heller ikke nået at få lavet unit test til vores beregningsmotor.

Kvalitetssikring (test)

Vi har lavet automatiserede test i JUnit:

- Integrationstests af vores mappermetoder
- Unit tests af metoderne i vores serviceklasser

Derudover har vi lavet manuelle test af frontenden.

Vores tests anvender en noget som skulle være rigtigt **assertEquals**, og et andet resultat, som ikke burde generere et positivt resultat **assertNotEquals**. Vi har ikke anvendt TDD (Test Driven Development). Men vi har skrevet test løbende for trods alt at sikre at metodernes output forbliver det samme, også efter refactoring.

Automatiserede tests

Nedenstående tabel viser oversigt over skrevne tests

Package	Beskrivelse
config	Logger: testLogFileCreation() - Den tjekker om loggeren fungere og skriver den forventet besked
persistence	AccountMapper Test: getAllEmailsFromAccount() - Tjekker om at metoden henter all emails i databasen. getIdFromAccountEmail() - Tjekker om at metoden hente den rigtige account id. getAllCustomerAccounts() - Tjekker om at metoden alle kunder i databasen. createAccount() - Tjekker om det bliver skrevet en ny account ned i databasen. login() - Tjekker om man bliver logget rigtigt ind i forhold til data. getAccountByEmail() - Tjekker om man får hentet den rigtige account udfra. getPasswordAndEmail() - Tjekker om man får hentet det rigtige kodeord og email. updatePassword() - Tjekker man får opdateret givet brugers kodeord. OrderMapperTest: void createOrder() - Tjekker man får skrevet en ny ordre ned i databasen void getOverviewOrderAccountDtos() - Tjekker man får hentet dto'er fra databasen void getOrdersFromAccountId() - Tjekker man får hentet alle ordre fra en given bruger getOrder() - Tjekker man får hentet en ordre rigtigt getDetailOrderAccountDtoByOrderId()

	<ul style="list-style-type: none"> - <i>Tjekker man får hentet dto'er fra databasen.</i> <p><code>updateMarginPercentage()</code></p> <ul style="list-style-type: none"> - <i>Ikke skrevet</i> <p><code>updateCarport()</code></p> <ul style="list-style-type: none"> - <i>Ikke skrevet</i> <p><u>OrderlineMapperTest:</u></p> <p><code>getOrderlinesForCustomerOrSalesrep()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om man har fået hentet alle ordelinjer for udfra et orderid</i>
services	<p><u>PasswordGeneratorTest:</u></p> <p><code>generatePasswordContainsElements()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om de generet passwords indeholder alle de forskellige tegn</i> <p><u>SalePriceCalculatorTest:</u></p> <p><code>calculateSalePrice()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om beregning lever rigtige resultat - salgsprisen uden moms</i> <p><code>calculateSalePriceInclVAT()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om beregning lever rigtige resultat - salgsprisen med moms</i> <p><code>calculateMarginAmount()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om beregning lever rigtige resultat - dækningsbidrag</i> <p><u>SvgTest:</u></p> <p><code>openAndCloseTags()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om at svg tags bliver åbnet og lukket rigtigt</i> <p><code>addRectangle()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om givet svg bliver lavet rigtigt</i> <p><code>addLine()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om givet svg bliver lavet rigtigt</i> <p><code>dimensionText()</code></p> <ul style="list-style-type: none"> - <i>Tjekker om givet svg bliver lavet rigtigt</i>

På figur 32 kan der ses vores test coverage, hvilket ikke er særlig høj, og viser at der primært er lavet integration tests.

Element ^	Class, %	Method, %	Line, %	Branch, %
app	45% (17/37)	26% (67/256)	28% (419/1464)	14% (70/468)
config	66% (2/3)	75% (3/4)	72% (26/36)	66% (4/6)
CustomLogFormatter	100% (1/1)	100% (2/2)	90% (19/21)	50% (2/4)
LoggerConfig	100% (1/1)	100% (1/1)	77% (7/9)	100% (2/2)
ThymeleafConfig	0% (0/1)	0% (0/1)	0% (0/6)	100% (0/0)
controllers	0% (0/2)	0% (0/21)	0% (0/262)	0% (0/84)
dto	100% (2/2)	12% (4/31)	55% (33/60)	100% (0/0)
entities	75% (3/4)	33% (12/36)	37% (41/110)	100% (0/0)
exceptions	0% (0/3)	0% (0/3)	0% (0/6)	100% (0/0)
persistence	66% (4/6)	46% (23/49)	50% (252/504)	19% (61/316)
AccountMapper	100% (1/1)	81% (9/11)	67% (103/153)	25% (29/112)
ConnectionPool	100% (1/1)	83% (5/6)	75% (22/29)	50% (2/4)
ItemMapper	0% (0/1)	0% (0/5)	0% (0/48)	0% (0/38)
ItemSearchBuilder	0% (0/1)	0% (0/12)	0% (0/44)	0% (0/20)
OrderlineMapper	100% (1/1)	40% (2/5)	32% (20/62)	15% (7/46)
OrderMapper	100% (1/1)	70% (7/10)	63% (107/168)	23% (23/96)
services	37% (6/16)	22% (25/109)	14% (67/478)	8% (5/62)
StructureCalculationEngine	0% (0/9)	0% (0/81)	0% (0/330)	0% (0/38)
svgEngine	80% (4/5)	85% (18/21)	36% (47/128)	5% (1/20)
PasswordGenerator	100% (1/1)	100% (4/4)	100% (17/17)	100% (4/4)
SalePriceCalculator	100% (1/1)	100% (3/3)	100% (3/3)	100% (0/0)

Figur 32 - Vores test coverage

User Acceptance tests

Vi har ikke haft tid med kunden til at der kunne lave UAT (User Acceptance Tests) på de forskellige user stories, men derimod lavet dem selv, her ses resultaterne:

User Story	UAT
US1 - Indtast ønsket carport og få tilbud	godkendt
US2 - Sælger og Kunde side, så de kan ændre password	godkendt, på kundesiden.
US3 - Sælger se carport henvendelser	godkendt
US4 - Tak for din henvendelse	godkendt
US5 - Sælger, klik på henvendelse, og se beregninger, stykliste osv.	godkendt

User Story	UAT
US6 - Sælger ændre beregninger/andet tilbud	godkendt
US7 - Bruger login	godkendt
US8 - Sælger ændre status på ordre	godkendt
US9 - Se startside for custom carporte	godkendt
US10 - Se alle kunder	godkendt
US11 - Send konto info til kunde	godkendt
US12 - Som kunde vil jeg kunne se min side	godkendt

Proces

Vores proces var generelt baseret på at dele opgaver op i user stories og tasks via KanBan på GitHub. Herefter kunne gruppemedlemmer frit vælge en opgave at arbejde på. Hver task blev tildelt en størrelse fra XS til XL, svarende til et tidsestimat, der starter ved 1 time og stiger eksponentielt til 16+ timer (figur 33). Hver task eller user story blev desuden prioriteret fra P0 til P2, hvor P0 var højeste prioritet.

Størrelse	Tidsestimat [timer]	Prioritet	
XS	1	P0 - højeste prioritet	
S	2	P1 - medium prioritet	
M	4	P2 - lavest prioritet	
L	8		
XL	16+		

Figur 33 - Processbillede

Vi arbejdede med en struktureret branch model bestående af en fungerende main-branch, en dev-branch og flere feature branches. Når en task blev valgt, oprettede gruppemedlemmet en ny feature branch og flyttede tasken fra "TODO" til "In Progress" på KanBan-boardet. Når arbejdet med tasken var færdigt, blev dev-branchens nyeste version merget ind i feature branchen for at sikre kompatibilitet og løse eventuelle konflikter. Derefter blev der oprettet en pull request, som gennemgik et review for at blive merget ind i dev-branchen. Tasken blev på dette tidspunkt flyttet til "In Review" på KanBan-boardet. Når reviewet var godkendt og merged ind i main, blev tasken flyttet til "Done". Når dev-branchen var stabil og indeholdt nok funktioner, blev den merget ind i main-branchen. Main branchen efterstræbte vi at merge ind i en gang ugentlig.

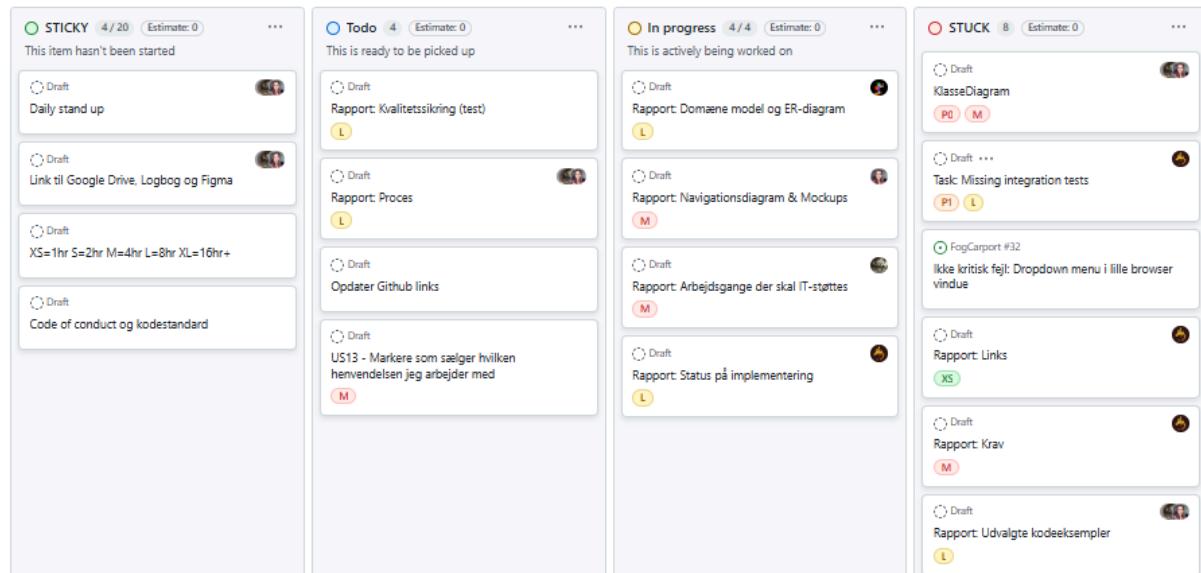
Vi anvendte KanBan-boardet på en lidt utraditionel måde (fuldt KanBan board på Figur 34 & 35) ved at tilføje ekstra kolonner. Vi havde fx en "Sticky"-kolonne til vores code of conduct⁶, links til rapporten, logbogen og andre ressourcer. Derudover havde vi en "Nice to have"-kolonne til funktioner, som ikke var kritiske, men som kunne prioriteres, hvis der blev tid. Denne tilgang gav os et godt overblik over både nødvendige og ønskede opgaver. Selvom dette ikke er den traditionelle måde at bruge et KanBan-board på, fungerede det effektivt for os. Dog vil denne tilgang sandsynligvis være uhensigtsmæssig i større projekter, hvor mængden af opgaver og information på KanBan-boardet kan gøre det svært at bevare et effektivt overblik.

Vi har gennem hele forløbet haft et code-of-conduct dokument⁷. Dette har medført at vi alle har været på samme side angående måden vi skulle skrive koden. Vi har også gjort brug af

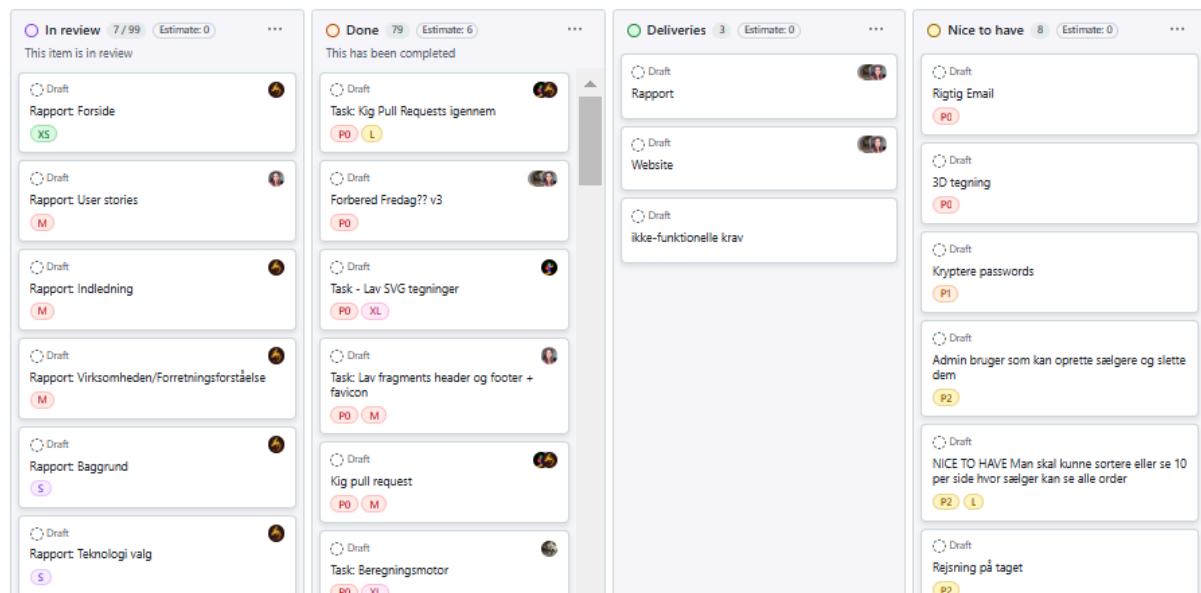
⁶ Bilag C - Code of conduct

⁷ Bilag C - Code of conduct

en aftalt kodelstandard⁸, hvilket har gjort vores formatering proces mere overskueligt. Vi har også gjort brug af daglig logbogsskrivning, som har kommet os til gode som reference til tidligere aftaler og samtaler, og har især kommet os til gode ved rapport skrivning. Vi har daglig afholdt standup møder, hvor vi i max 5 minutter forklare om hvad vi arbejdede på dagen før, hvad vi har tænkt os at arbejde på, på dagen, og om vi er stuck og har brug for hjælp. Efterfulgt af et planlægningsmøde for dagen.



Figur 34 - KanBan board - del 1



Figur 35 - KanBan board - del 2

⁸ Bilag D - Kodelstandard

Arbejdsprocessen faktuelt

- I første fase fokuserede vi på at afklare krav fra kunden og strukturere projektet. Dette inkluderede udarbejdelse af forskellige diagrammer (As-Is, To-Be, ERD osv.) samt planlægning af projektforløbet. Vi valgte også de teknologier, der skulle bruges til projektet. De første user stories blev forberedt til næste fase, og enkelte begyndte at kode mod slutningen af fasen.
- Anden fase var primært præget af programmering. Der blev også rettet i projektet og opdateret diagrammerne. Flere user stories blev afsluttet, mens arbejdet med beregningsmotoren fortsatte. På dette tidspunkt havde vi en ustylet hjemmeside med basal funktionalitet.
- I tredje fase blev beregningsmotoren færdiggjort. Hjemmesiden begyndte at tage form og fik både øget funktionalitet og styling, så den fremstod mere komplet.
- I fjerde fase blev de sidste dele af projektet programmeret. Der blev indført en code freeze midt i fasen, hvor kun fejlrettelser og deployment til Digital Ocean blev tilladt.

Vi havde en fra vores gruppe som stod for at oprette GitHub projekt samt KanBan, men derudover var der ikke nogen decideret KanBan Mester. Vi var alle med til at opdatere KanBan med de task der var, samt at indsætte nye ressourcer hvis det var nødvendigt. Vi kunne også selv vælge hvilken opgave vi ville have, men oftest gjorde man det i sammenråd med resten af gruppen for at vælge en opgave som passede godt ind med de andre opgaver og hvad status var for projektet. Det ville for eksempel ikke give mening at starte på en task som omhandlede generel styling af hjemmesiden, hvis der ikke allerede var en hjemmeside at style ud fra. Det var derfor meget frit, hvad man pådrog sig af opgaver.

Vi havde under et af vejledningsmøderne spurgt ind til forskellige ting som styklisten og hvor meget der skulle være med i den. Vi har også haft spurgt ind til diverse ting angående SVG og beregningsmotoren, og om hvad de skal kunne. Vi har på det næstsidste møde fået at vide af kunden, at carportene nu også skal kunne laves ikke kun i 30 cm increment, men i stedet alle tal fra 240 til 600 cm. Dette blev til en lille uforudset udfordring, men det løste sig heldigvis.

Vi mødtes som udgangspunkt på skolen, og derfor har kommunikationen generelt været god. De eneste dage, vi ikke mødtes som gruppe, var specifikt om torsdagen.

Der har været enkelte fejl i kommunikation derudover, mest af alt i misforståelser, eller fordi man har været så meget fordybet i sit eget, at det kunne være svært at forholde sig til andet.

Arbejdsprocessen reflekteret

Vi havde som sagt ikke nogen officiel KanBan master, og det synes at virke ret fornuftigt, især taget projektets størrelse i betragtning. Havde det været et større projekt, så skulle der nok have været en til at tage teten og stå for den overordnede struktur.

Det væsentligste emne var nok, at vi kun skulle lave minimal viable product. Overordnet set var der nok nogle enkelte dele som var i for stort scope, og burde have været mindre poleret. Der var også nogle problemer i hvordan vi havde udformet vores tasks, hvor nogle af tasks var meget store, og andre var ret små. Beregnings motoren blev der for eksempel arbejdet på over mange dage i en enkelt task. Optimalt skulle denne task have blevet brudt ned i mindre bidder, for at gøre det lettere, samt at skabe mere sikkerhed for resten af gruppen: Er man igang med en stor task som forløber sig over flere dage, og man midt i tasken bliver syg, så bliver det svært for andre at starte op hvor den anden slap. Ud over de helt store tasks, så var vi generelt fornuftige til at vurdere størrelsen af en task, samt at estimere tiden. Først da tasken blev større gav det problemer med tidsforbruget.

Da beregnings motoren og andre større tasks blev færdiggjort, tog projektet for alvor fart. Der var flere hænder ledige, som arbejdede på langt mindre tasks. Det hjalp med at man ikke blev udbrændt over en enkelt stor task som synes uoverkommelig. I fremtiden kunne det måske være fornuftigt at bryde selv større tasks op i mindre tasks. For eksempel kunne en task til beregnings motoren være at lave selve stolpe beregningen. Det ville også gøre det lettere for andre at tage over i tilfælde af sygdom, eller generelt at hjælpe med den overordnet task.

Hertil når vi ser tilbage på arbejdet kunne det have fungeret meget godt med noget mere trunk based development (TBD), især i start og slut fasen. Til det skulle vores User Stories og de tasks som ligger i dem, været delt op mindre og derved kunne vi få etableret TBD i arbejdsprocessen. Vi kom kun kort til at gøre brug af TBD i slutfasen da vi skulle merge fra dev til main. Vi kørte vores web applikation igennem og skrev vores rettelser ned, som vi så fik rettet og merged ind fra dev til main i hurtige sessioner.

Bilag

Bilag A - User stories

Priority P0:

US1 - Indtast ønsket carport og få tilbud (Large)

Som uregistreret kunde

Vil jeg kunne taste mål ind på min ønskede carport

Så jeg kan sende dem ind til fog for et tilbud

Acceptance Criteria:

Givet at jeg er inde på custom carporte siden

Når jeg vælge mine mål

Så kan jeg sende informationer ind til fog

Tasks:

Task 1: Lav HTML side og form (med Bootstrap)

Task 2: Lav en controller som fanger form

Task 3: Lav mapper som sætter ind i database

Task 4: Lav mock email – sout i konsollen

Task 5: integrations tests

US5 - Sælger, klik på henvendelse, og se beregninger, styklister osv. (Large)

Som sælger logget ind i systemet

Vil jeg kunne klikke ind på en enkelt carporte henvendelse

Så jeg kan se beregningerne, styklisten, tegninger samt se og ændre dækningsgraden for at træffe beslutning om videre rådgivning

Acceptance Criteria:

Givet at jeg er logget ind i systemet som sælger

Når jeg klikker på en henvendelse fra en kunde

Så jeg kan se beregningerne, styklisten, tegninger samt se og ændre dækningsgraden

Tasks:

Task 1: opdater Figma

Task 2 Lav html side

Task 3: Lav controller

Taks 4: Lav mapper

US6 - Sælger ændre beregninger/andet tilbud (Large)

Som sælger logget ind i systemet

Vil jeg kunne ændre i beregningerne

Så jeg kan give mine kunder et andet tilbud

Acceptance Criteria:

Givet at jeg er logget ind som sælger

Når jeg klikker på en henvendelse

Så kan jeg ændre i tilbuddet pris og se opdateret dækningsbidrag

Tasks:

Task 0: opdater Figma (tjek US 5, samme der skal opdateres)

Task 1: lav HTML side og form (med Bootstrap)

Task 2: lav en controller som fanger form

Task 3: Lav mapper som sætter ind i database

Priority P1:

US3 - Sælger se carport henvendelser (Medium)

Som sælger logget ind i systemet

Vil jeg kunne se en oversigt over custom carporte henvendelser i webappen

Så jeg kan klikke mig ind på hver enkelte henvendelse

Acceptance Criteria:

Givet at jeg logger ind som sælger

Når jeg er logget ind

Så kan jeg se oversigt over alle henvendelser

Tasks:

Task 1: lav HTML side og form (med Bootstrap)

Task 2: lav en controller som viser data

Task 3: Lav mapper

US4 - Tak for din henvendelse (Medium)

Som registreret kunde

Vil jeg kunne få en kvittering som tak for min henvendelse

Så jeg er sikker på min henvendelse er modtaget

Acceptance Criteria:

Givet at man er en person som ikke er logget ind

Når man har trykker send på sin henvendelse

Så får man vist en Tak side og får en kvittering på "mail"

Tasks:

Task 1: Lav Tak side html

Task 2: lav en controller

US7 - Bruger login (Small)

Som person som ikke er logget ind

Vil jeg kunne logge ind i systemet med en given rolle

Så jeg kan få adgang til det jeg skal (sælger/kunde)

Acceptance Criteria:

Givet at jeg ikke er logget ind og er oprettet i systemet

Når jeg logger ind

Så får jeg adgang til mine passende funktioner(Sælger / kunde)

Tasks:

Task 1: lav HTML side

Task 2: lav en controller

Task 3: Lav mapper henter fra database

US8 - Sælger ændre status på ordre (Medium)

Som sælger logget ind i systemet

Vil jeg kunne se og ændre status på igangværende ordre

Så at jeg kan ændre statussen på den pågældende ordre

Acceptance Criteria:

Givet jeg er logget ind i systemet som sælger

Når jeg er under en henvendelse

så ville jeg kunne ændre på statussen

Task:

Task 1: lav HTML side (tjek US 5 og 6)

Task 2: lav en controller

Task 3: Lav mapper som kan ændre i DB

US9 - Se startside for custom carporte (Medium)

Som uregistreret kunde

Vil jeg kunne se en startside med mine muligheder for custom carporte(flad tag, rejst tag)

Så man kan vælge hvilken type carport man ønsker

Acceptance Criteria:

Givet at jeg som uregistreret kunde/kunde finder ind på startsiden

Når jeg er der inde

Så kan jeg se noget intro tekst omkring carporte og vælge faldt tag carport

Tasks:

Task 1: opdater Figma Customcarport

Task 2: lav AC

Task 3: lav HTML side

US12 - Som kunde vil jeg kunne se min side (Large)

Som kunde der har bestilt et tilbud

Vil jeg kunne logge ind i fog systemet

Så jeg kan jeg kan se min ordre

Acceptance Criteria:

Givet at som kunde har bestilt et tilbud og har snakket med sælger

Når jeg får mine login informationer

Så ville jeg kunne logge ind og kigge på mit tilbud

Tasks:

Task 0: opdater Figma

Task 1: lav HTML side

Task 2: lav en kontrolle metode

Task 3: Lav mapper som henter

Priority P2:

US2 - Sælger og Kunde side, så de kan ændre password (Large)

Som sælger/kunde logget ind i systemet

Vil jeg kunne ændre min adgangskode

Så jeg kan ændre adgangskoden til et jeg ønsker

Acceptance Criteria:

Givet at jeg er logget ind i systemet som sælger/kunde

Når jeg klikker på ændre adgangskode

Så har jeg mulighed for at ændre adgangskoden.

Tasks:

Task 1: Lav Html side

Task 2: Lav en controller

Task 3: Lav en mapper metode

US10 - Se alle kunder (Medium)

Som sælger logget ind i systemet

Vil jeg kunne se en oversigt over alle kunder

Så jeg kan klikke mig en på en bestemt

Acceptance Criteria:

Givet jeg er logger ind i systemet som sælger

Når jeg trykker log ind

så kan jeg klikke mig ind på en siden hvor jeg kan se alle kunder

Tasks:

Task 0: lav AC

Task 1: lav HTML side og form (med Bootstrap)

Task 2: lav en controller

Task 3: Lav mapper henter fra DB

Task 4: Integration test og update classdiagram

US11 - Send konto info til kunde (Medium)

Som sælger logget ind i systemet og kigger på en henvendelse

Vil jeg kunne sende konto info til kunden

Så kunden kan logge ind og kigge deres tilbud

Acceptance Criteria:

Givet at jeg er logget ind i systemet som sælger

Når jeg klikker ind på en henvendelse

Så jeg kan trykke på en knap så kunden får tilsendt sine konto information

Tasks:

Task 1: lav HTML knap (se US5 for html side)

Task 2: tilføj controller metode som sender info

Task 3: Lav mapper som henter data

US13 - Markere som sælger hvilken henvendelsen jeg arbejder med (Medium)

Som sælger logget ind i systemet

Vil jeg kunne vælge at sætte mig på en henvendelse

Så de andre sælger ved at jeg arbejder med den

Acceptance Criteria:

Givet at jeg er en sælger logget ind i systemet

Når jeg trykke mig på en opgave

Så kan de andre sælger se at jeg er på opgaven, så de ikke gør noget på den

Nice to have:

US14 - Sælger kan sende rigtig emails til kunden.

Som sælger logget ind i systemet

Vil jeg kunne sende "rigtige" emails til kunden

Så al info til kunden bliver sendt på mail

Acceptance Criteria:

Givet at jeg er en sælger logget ind i systemet

Når jeg trykker på send kundeinfo

Så bliver en mail sendt til kunden med alle deres oplysninger

US15 - Custom carport med tagrejsning

Som uregistreret kunde

Vil jeg kunne vælge mellem fladt eller rejsning på taget

Så jeg kan vælge min ønsket carport

Acceptance Criteria:

Givet at jeg er inde på custom carporte siden

Når jeg vælge mellem fladt eller rejsning på taget

Så kan jeg sende informationerne ind til fog

Bilag B - Udsnit af logbog

Fredag 22. nov. 2024:

Vi startede dagen ud med daily stand up meeting og efter det kiggede vi videre på vores forberedelse til vores møde med Jesper/"Martin".

Noter til efter vores møde med Jesper/"Martin":

- Hvordan skal vi informere kunden om ændringer på carporten → skal vi sende "mail" om at der er sket en ændring på din henvendelse, du kan logge på din FOG konto og se dem.
- Skal vi tilføje et link til FOG's rigtige hjemmeside med standard carport på vores index html side?
- Stykliste prisen og en samlet pris skal vise til kunden når det skal betale (har betalt)?
- Vi kan starte med at kunden kun kan vælge tag, og hvis vi når langt, kan vi tilføje at de kan vælge uden tag.
- Vi kan godt tilføje til vores scorpe at kunden kan vælge rejsning på carport taget. (vi må se hvor langt vi kommer)
- **Noter til rapporten:** Kanban → vores måde at dele user stories ud på og hvordan vi holder styr på de forskellige tasks.
 - Vi sidder sammen på skolen.
 - Har daily stand up meeting
 - Skriver ind på de forskellige user stories i vores kanban.

Fredag 29. nov. 2024:

Møde med Jesper/"Martin".

Vores produkt indtil videre blev vist til "Martin" og han syntes det så flot ud.

Men han havde en vigtigt kommentar: Det er ikke godt nok at slutkunden kun kan vælge carport længde og bredde i inkrementer af 30 cm. Slutkunden skal frit kunne vælge carport længde og bredde (indenfor minimum og maximum).

Jesper syntes at vi meget snart skal sætte vores produkt sammen til et Minimum Viable Product. Inklusiv at vi får beregningsmotoren ind i vores Minimum Viable Product.

Bilag C - Code of conduct

1. Der skal bruges IntelliJs autoformatter, før der merges ind i fælles arbejde (for at undgå ligegyldig merge conflicts).
2. Vi laver pull requests når der merges ind i fællesarbejde (dvs. når feature branch skal ind i dev branch, og når dev branch skal ind i main).
3. Integrationstest / unit tests i det omfang det giver mening. Skal være skrevet før feature branch kommer ind i dev branch. (for det giver ikke mening at putte en feature ind som ikke er testet endnu).
4. Filer som ikke er med i GitHub projektet skal ind i Google Drive mappen. Giv filer et sigende navn, f.eks. "Interessentanalyse_v1.drawio". Gem løbende versioner (i det omfang det giver mening): "Interessentanalyse_v1.drawio", "Interessentanalyse_v2.drawio", "Interessentanalyse_v3.drawio".
5. Vi starter med at udvikle webappen uden styling.
6. I vores exception handling skriver vi metode navnet ind i developerMessage.
7. Vi smider brugerbeskederne nede fra mapperne og bruger dem i controllerne
8. Der laves PR fra dev til main ugentligt, såfremt dev branchen fungerer, ellers venter vi med PR til dev er oppe og fungere.

Bilag D - Kodestandard

- 1) Vi bruger sigende variabelnavne og metodenavne i alle vores kode sprog, Java, SQL, HTML og CSS.
- 2) Vi bruger IntelliJ's standardindstillinger for formatering (krøllet parentes på samme linje).
- 3) Vi bruger linjeskift til at lave en meningsfyldt inddeling af kode blokke (så kodelinjer som der hører sammen, står sammen inde i en metode).
- 4) Vi gør så der én tom linje efter hver metode og ingen tomme linje tilsidst.
- 5) Imports i rækkefølge: Java, "tom linje her", 3rd party, "tom linje her", vores egne.
- 6) HTML skabelon -> <head></head> <body> <header> <main> <footer> </body>
- 7) Der skal være en god grund til en kommentar, vi stræber efter selvforklarende kode.

Bilag E - Tidlig version af ER diagram

