

# Fog Carport Projekt



Ingrid Karen Svendsen	Ingridksv	Cph-is142@cphbusiness.dk
Malou Vich Lavrentiew	MalouVich	Cph-ml786@cphbusiness.dk
André Samuelsen	AndyTheDragon	Cph-as760@cphbusiness.dk
Frederik Michael Franck	MojoOno	Cph-ff72@cphbusiness.dk
Mathias Kroghave Falcham	Falcho	Cph-mf398@cphbusiness.dk

18. november 2024 – 19. December 2024

Links .....	3
Indledning .....	4
Baggrund .....	4
Virksomheden/Forretningsforståelse .....	5
Teknologi valg .....	9
Krav .....	10
Arbejdsgange der skal IT-støttes.....	11
User stories .....	14
Domæne model og ER-diagram .....	16
Navigationsdiagram & Mockups 9 .....	20
Valg af arkitektur 10.....	27
Særlige forhold 11.....	31
Status på implementering 12.....	37
Automatiserede tests 13.....	39
User Acceptance tests 14.....	40
Proces 15.....	44
Bilag.....	54

## Links

- Link til GitHub repository - <https://github.com/AndyTheDragon/Fog-projektet>
- Link til demovideo -  
<https://cphbusiness.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=bba4ac2c-4fa5-4e6d-ba4a-b24b008050ed>
- Link til hjemmeside - <https://carport.wyrmings.dk/>  
Email: [mortens@fog.dk](mailto:mortens@fog.dk)  
Password: fog2024\$1

## Indledning

Denne rapport er rettet mod dig, som medstuderende på datamatikeruddannelsens 2. semester. Formålet er at give et klart og spændende indblik i vores projekt – fra de teknologiske valg til udviklingsprocessen bag løsningen. Du vil blive introduceret til projektets krav, de værktøjer og teknologier, vi har anvendt, samt de metoder, vi brugte til at udvikle et program, der opfylder de behov, som forhandleren Fog har stillet.

Forestil dig at kunne designe din egen carport med få klik – skræddersyet efter dine mål og behov. En intuitiv, brugervenlig webapplikation, hvor processen er enkel, overskuelig og direkte forbundet med forhandleren Fog. Det har været kernen i vores projekt at skabe en løsning, der gør det nemt og bekvemt for kunderne at designe og bestille carporte på specialmål og samtidig sikre en smidig kommunikation med Fogs sælgere.

## Baggrund

Johannes FOG A/S er en dansk virksomhed grundlagt i 1920, med base i Nordsjælland. Virksomheden er kendt for at levere kvalitetsprodukter samt faglig rådgivning til både professionelle håndværkere og private. Deres sortiment tilbyder alt fra interiør til carporte, hvilket gør dem til en alsidig aktør i bygge- og boligbranchen. Forretningen bygger på et fundament af samfundssind og medmenneskelighed, og de har fokus på især godt håndværk, innovation og miljø.

*Kunden må på ingen måde se styklisten, før de har betalt for den ønskede carport, da dette er hele forretningsmodellen for FOG.*

Vi har fra kunden modtaget en video beskrivende deres nuværende udfordringer med systemet de bruger, og ud fra denne er der identificeret ét krav.

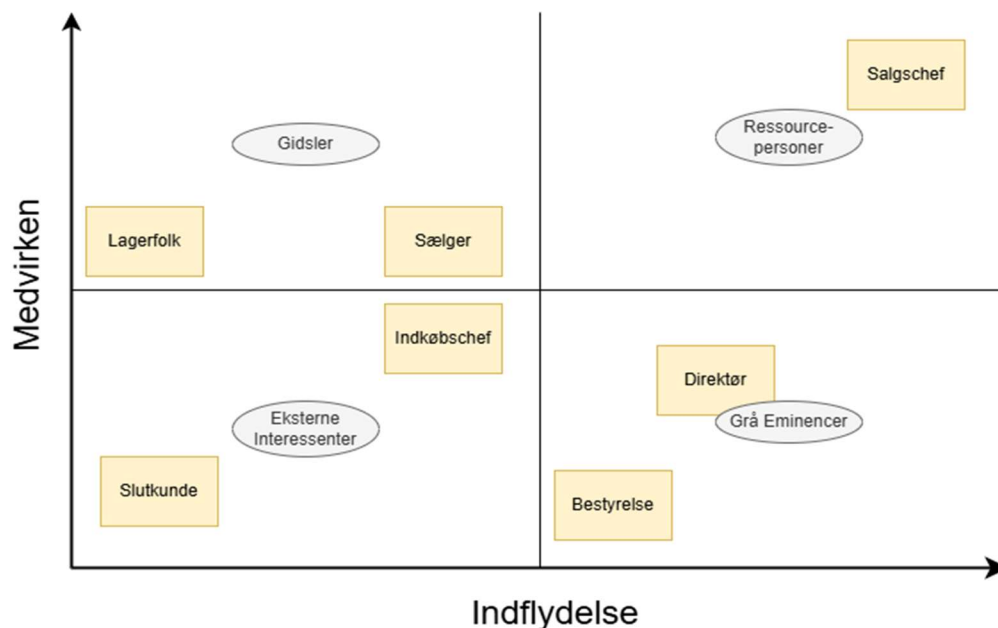
Udfordringerne stillet i videoen har vi forsøgt at optimere på bedst vis, samtidig med vi har overholdt det ene krav Martin stillede, nemlig at *Kunden må under ingen omstændigheder modtage en stykliste, før ordren er betalt*. Dette er nemlig essensen af forretningsmodellen, og derfor essentielt for at produktet kan blive godkendt af kunden.

## Virksomheden/Forretningsforståelse

Interessentanalysen, som vi har udarbejdet herunder i *Figur 3.1*, har givet os et klart overblik over de forskellige interessenter, som har haft en indflydelse på projektet. Analysen har hjulpet med at identificere de vigtigste nøglepersoner og gjort det tydeligt, hvilke interessenter der skulle inddrages aktivt, og hvilke der blot skulle holdes opdateret i løbet af projektet.

Dette overblik var med til at sikre en effektiv kommunikation med de relevante parter og skabe en fælles forståelse af projektets mål og fremdrift. Ved at have en klar plan for interessenthåndtering, som det fremgår af *Tabel 3.2*, kunne vi sikre, at kundens behov og ønsker blev imødekommet, og at alle involverede interessenter var i samme øjenhøjde. Samtidig blev det lettere at reagere hurtigt, hvis der opstod udfordringer som forsinkelser, eller hvis beslutninger skulle træffes.

Analysen viste os også, hvem vi skulle henvende os til, når specifikke spørgsmål skulle afklares, eller der var behov for ressourcer og input. For eksempel blev salgsschefen udpeget som en central beslutningstager, mens direktøren og bestyrelsen primært skulle holdes informeret med opdateringer og sikre opbakning ved sprintsne.



Figur 3.1

Interessentanalyse - indflydelse/medvirken-matrix

<b>Interessentanalyse Fog Carport</b>				
Interessent	Interessenten kan opleve følgende FORDELE ved projektet	Interessenten kan opleve følgende ULEMPER ved projektet	Samlet vurdering af interessentens bidrag/position	Håndtering af interessenten
Salgschef	Øget salg og bedre produktudbud.	Ressourcepres og ansvar for resultater.	Høj indflydelse, høj medvirken.	Salgschefen er ind over hele projektet, fra start til slut, bruges som beslutningstager.
Direktør	Strategisk vækst og forbedret bundlinje.	Øget fokus på projektets risiko og økonomi.	Mellem - høj indflydelse, lavere medvirken.	Direktøren holdes informeret angående projektstatus.
Bestyrelse	Forbedret virksomhedsstrategi og investeringer.	Usikkerhed ved implementering og langsigtede gevinster.	Lav medvirken, middel indflydelse.	Bestyrelsen holdes opdateret ved sprints og der vises projektets værdi.
Sælger	Mulighed for mersalg og provision, hurtigere gennemarbejd af ordre samt et samlet overblik over alle ordre og kunder.	Ny implementering af system, muligvis ændringer i rutiner. Senere kontakt til kunden i forhold til nye ordreoprettelser.	Mellem indflydelse, mellem medvirken.	Skab dialog og involver sælgere i planlægning igennem salgschefen for et bredere accept.
Indkøbschef	Bedre produktløsninger og indkøbsoptimering.	Muligvis øget ansvar, hvis der kommer nye leverancer og partnere.	Mellem indflydelse, lavt til mellem medvirken.	Orienteres inden slutsprint.
Lagerfolk	Effektiv lagerstyring og bedre arbejdsgang.	Potentielle ændringer i arbejdsgange og pres på kapacitet/ arbejde, ved mersalg.	Lav indflydelse, høj medvirken.	Involveres i processen og høres ved bekymringer.
Slutkunde	Bedre og mere overskuelig købsoplevelse, serviceres muligvis hurtigere da sælgere kan arbejde mere effektiv.	Senere kontakt til sælger i forhold til oprettelse af ordre.	Lav indflydelse, lav medvirken.	Kommunikér fordele klart og tilbyd kundesupport.

Tabel 3.2 Interessentanalyse

Risikoanalysen som ses i *tabel 3.3* herunder, udarbejdede vi i starten af projektet, for at have en oversigt over de potentielle risici, der kunne påvirke projektets gennemførelse. Hver risiko har et **risikotal (K x S)** som er produktet af **Konsekvens (K)** og **Sandsynlighed (S)**, som hvert er vurderet på en skala fra 1 til 5, hvor 1 er lav og 5 er høj, samt en række forebyggende og afbødende handlinger, for minimering af sandsynligheden af risiciene, eller for at reducere konsekvenserne, hvis de skulle opstå.

Et risikotal på 10 eller derover har vi vurderet til kritisk og kræver ekstra forebyggelse og planlægning, heraf har vi én, som er en menneskelig faktor, hvortil vi ud fra bedst mulige evner har prioriteret kommunikation og planlægning, for at reducere konsekvenserne, hvis den skulle opstå.

Risik o ID	Hvad kan gå galt?	K	S	K x S	Forebyggende handlinger	Afbødende handling
1	Over 50% sygdom i udviklingsteamet	3	4	12	Planlægning med Kanban, daglige stand up meetings (SUM) samt logføring af daglige arbejde, så alle har overblik over opgaver og teammedlemmers arbejdsstatus.	Prioritere opgaverne efter hvor vigtige de er for projektet, for at sikre fremdrift. Dem der er til stede, har det endelige ord.
2	FOG går konkurs	5	1	5	Risikovurdering af FOG's økonomi. Udarbejd klare kontrakter og samarbejdsaftaler med betalingsbetingelser.	Sikre backup af alle nødvendige data og dokumentation, så projektet kan fortsætte uden større forsinkelser ved en evt. ny overdragelse af FOG.
3	Sygdom hos ressourcepersoner	4	2	8	Dokumenter og del al viden for at mindske afhængighed med daglige SUM. Planlægning af de efterfølgende dage.	Omfordel opgaver midlertidigt i teamet, så vi ikke er afhængige af enkelte personer.
4	Udviklingsteam går konkurs	5	1	5	Dokumenter arbejdet løbende og sørg for omfattende videns overdragelse, så projektet nemt kan videreføres af andre i tilfælde af konkurs.	Indgå samarbejdsaftaler med eksterne partnere eller konsulenter, der kan overtage projektet midlertidigt, hvis udviklingsteamet går konkurs.
5	Skifte i system B (Info - m3)	4	1	4	Hold tæt kommunikation med system B-udbyderen og få en overblik over	Planlæg implementering af systemskifte udenfor kritiske projektsprint.

					opdateringsplaner. Lav en testplan.	
6	Skifter hjemmeside (fog)	1	2	2	Koordiner med IT-afdelingen hos FOG og få tidlig indsigt i ændringer. Test funktionalitet løbende.	Sørg for backup af eksisterende hjemmeside, og identificer løsninger til hurtig udbedring af fejl.
7	Dødsfald	5	1	5	Sørg for dokumentation og deling af viden gennem hele projektet.	Prioriter de vigtigste opgaver i en nødsituation.
8	Sygdom hos Tech-lead	5	1	5	Sørg for deling af viden gennem dokumentation og vidensdeling på tværs af teamet.	Omfordel ansvar til de andre teammedlemmer, dem der er til stede, har det endelige ord.
9	Udviklings system bliver hacket	2	3	6	Implementer sikkerhedsforanstaltninger som firewall, kodebackup og adgangsbegrænsning.	Genopret fra backup-systemer. Gennemgå sikkerhedsbruddet og implementer forbedringer for at forhindre gentagelse.

Tabel 3.3 Risikoanalyse



## Teknologi valg

Projektet er udviklet med følgende teknologier:

- **Java 22:** Hovedsprog for applikationens backend.
- **Javalin (version 6.3.0):** Webframework til at håndtere HTTP-forespørgsler og responses.
- **Thymeleaf (version 3.0.15):** Template engine til at generere dynamiske HTML-sider.
- **PostgreSQL (version 14):** Database, der håndterer lagring af data om kunder, ordrer og produkter.
- **JDBC:** Anvendes til at etablere forbindelse mellem Java-applikationen og PostgreSQL-databasen.
- **HTML5 og CSS3:** Bruges til at designe og strukturere webapplikationens brugergrænseflade.
- **Bootstrap (version 5.3.3):** Responsivt CSS (og JavaScript) framework, som giver en grundlæggende styling og funktionalitet der kan forventes af et moderne website.
- **IntelliJ IDEA 2024.2.4 (Ultimate):** IDE til udvikling og debugging af projektet.

Disse teknologier understøtter udviklingen af en pålidelig og dynamisk webapplikation, som opfylder kundens behov for tilpasset bestilling og administrativt overblik.

## Krav

Vi har ud fra den video vi modtog fra kunden (Fog), der beskriver hvad de ønsker sig deres nye program kan, og hvilke nye features de tænker kunne være en forbedring, udviklet en masse user stories som vi tog udgangspunkt i for at udvikle det. Vi har efterfølgende sorteret i vigtigheden af dem for at komme frem til hvilke der var absolut nødvendige at have med for at få en funktionel applikation, og hvilke der ville være "quality of life" forbedringer.

Som nævnt før er det ene krav som kunden havde:

*Kunden må på ingen måde se styklisten, før de har betalt for den ønskede carport, da dette er hele forretningsmodellen for FOG*

Vi har udover det ene krav som kunden havde, sat nogle krav til vores egen løsning af opgaven:

*Når kunden trykker send på sin custom carport formular, bliver ordren sat direkte ind i databasen, så programmet kan udarbejde en ordreoversigt til sælger*

*En sælger skal kunne tilgå en side som indeholder carport ordre tilsendt fra slut kunden. Fra denne side skal sælger kunne tildele sig selv en ordre for at danne overblik over dimensioner, tegning og stykliste for ultimativt at kunne tilsende et tilbud til slut kunden*

Vi har forsøgt at lave flere mindre user stories så de var mere overkommelige at skulle arbejde på, hvilket er grunden til at vi har 25, og vi har senere opdelt dem i sprint efter vigtighed og er derfor ikke nødvendigvis i prioriteret rækkefølge.

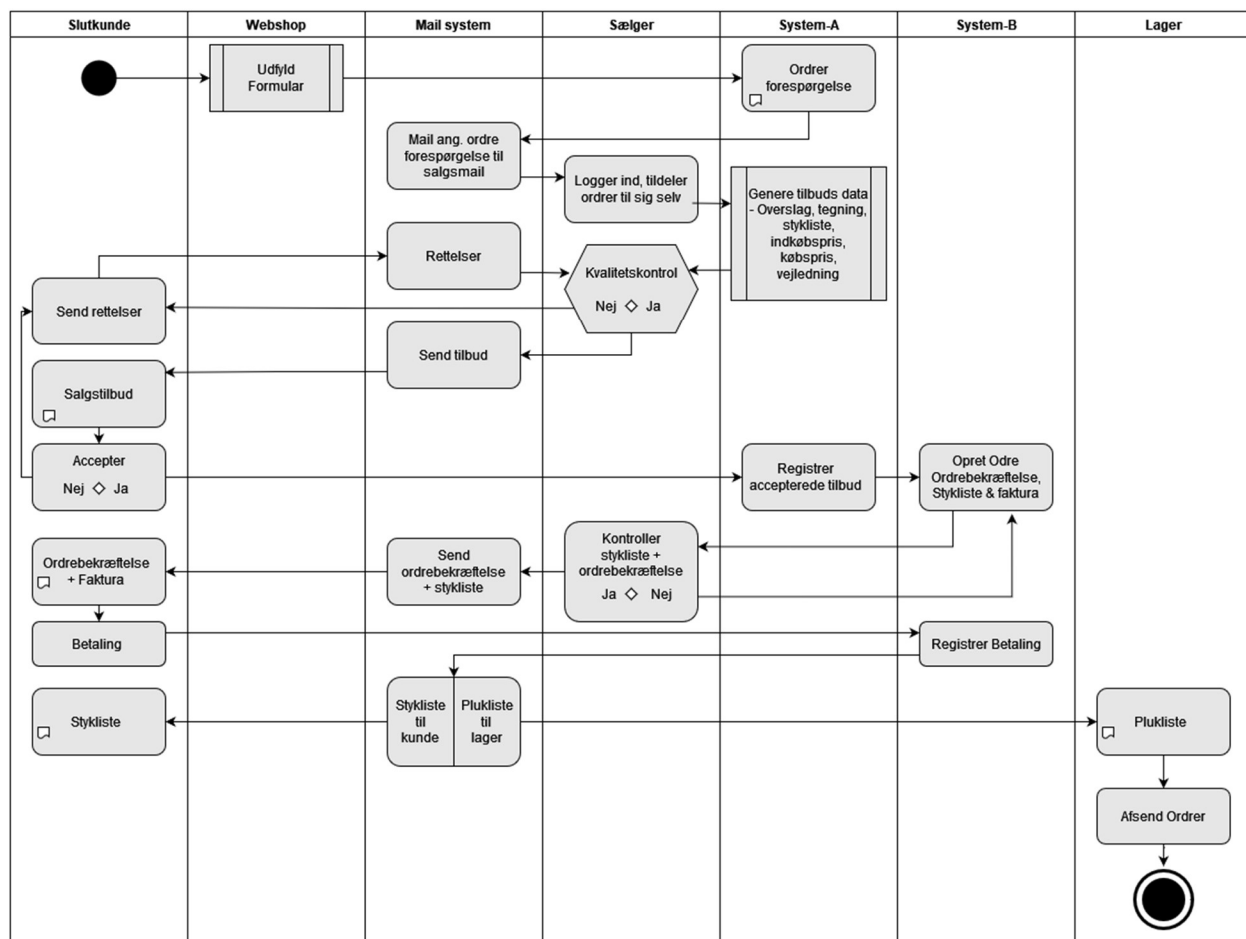
## Arbejdsgange der skal IT-støttes

Diagrammerne AS-IS og TO-BE beskriver begge processen fra, at en kunde udfylder en formular i webshoppen, til ordren er blevet behandlet og sendt fra lageret. Det viser hvordan forskellige aktører og systemer arbejder sammen, hvilke trin der er nødvendige, og hvilke beslutningspunkter der findes. Herunder ses aktivitetsdiagrammet AS-IS for Fogs eksisterende proces for bestilling af en carport. Diagrammerne er lavet ud fra Fog videoen med Martin.

Som det ses i AS-IS-diagrammet i figur 6.1 (se bilag) starter processen ved slutkunden, der udfylder en formular på Fogs webshop med data ang. en carport på special mål. Formular dataene bliver her sendt videre til sælgeren som foretager en kvalitetskontrol for at sikre at oplysningerne giver mening. Hvis det er nødvendigt, vil sælgeren bede kunden om eventuelle rettelser. Når dataene er korrekte, indtaster sælgeren dem i System A.

System A genererer et tilbud med tegning, stykliste, kostpris og salgspris. Tilbuddet vil blive sendt til sælgeren som foretager en ny kvalitetskontrol af tilbuddet og en mail vil blive sendt til kunden. Sælger vil modtage at tilbuddet er accepteret, hvis ikke dette er tilfældet skal der sendes nye rettelser til sælgeren.

Ved accept af tilbuddet indtaster sælgeren den relevante data ind i System B, som genererer en stykliste, ordrebekræftelse og faktura. Sælger vil igen kontrollere tilbuddet fra System B og en mail vil blive sendt til kunden med en ordrebekræftelse og faktura som skal betales. Når betalingen er modtaget og registeret bliver der sendt en kvittering med styklisten til kunden. Lageret vil modtage en plukliste og få afsendt ordren.



Figur 6.2: Aktivitetsdiagram TO-BE

I TO-BE diagrammet som ses i figur 6.2, starter processen også med slutkunden, der udfylder en formular i webshoppen. Når formularen er indsendt, videregives informationen til mailsystemet, som sender en mail til sælgeren. Her vil System A automatisk selv oprette en ordre på baggrund af den indsendte data. System A generer et tilbud med tegning, styklister, kostpris og salgspris. Sælgeren logger sig derefter ind i systemet og tildeler ordreforespørgslen til sig selv for videre behandling.

Kvalitetskontrol, rettelser samt accept af tilbuddet fra kunden fungerer på samme måde som i AS-IS-diagrammet. Når kunden har accepteret tilbuddet, vil System A selv registrere dette og sende det videre til System B som opretter ordren med ordrebekræftelse, styklister og faktura.

De to diagrammer viser den samme proces fra kundens ordreforespørgsel til afsendelse af ordren, men de adskiller sig ved detaljer og struktur.

AS-IS-diagrammet er mere detaljeret og opdeler processen i små trin, hvilket er med til at give en dybere forståelse af hver enkelt aktivitet og interaktionen mellem systemer og aktører. Diagrammet specificerer overførslen af data mellem de forskellige systemer (System A og System B) mere detaljeret, samt giver et større fokus på beslutningspunkterne, hvor sælgeren selv skal foretage kontroller.

TO-BE diagrammet fokuserer mere på det overordnede overblik. Processer, der i AS-IS-diagrammet er opdelt i flere trin, er blevet samlet under mere overordnede punkter. For eksempel er kvalitetskontrollen ikke vist i detaljer, men fremstår som én samlet aktivitet. Overførelse af data mellem systemerne er også mindre detaljeret. Beslutningspunkterne er færre og præsenteres mere direkte end i AS-IS-diagrammet.

De største forskelle på de to diagrammer ses hos sælgeren. I TO-BE diagrammet er flere af processerne blevet automatiseret. Sælgeren skal stadig foretage kvalitetskontrol af tilbuddet men skal ikke længere selv manuelt indtaste data fra carportformularen ind i systemerne.

## User stories

I forbindelse med udviklingen af programmet, har vi udarbejdet en række user-stories fra kundens ønsker. Disse user-stories fungerer som bro mellem kundens krav og de tekniske løsninger, der implementeres i projektet. User-stories beskriver funktionaliteten set fra brugerens perspektiv og opdeles i mindre tasks, så de kan implementeres trinvis. Samtidig er de blevet estimeret i forhold til den forventede arbejdsindsats.

Et eksempel på en user-story, der er blevet gennemarbejdet og fuldt ud implementeret i projektet, er følgende:

### **US9 - Beregningsmotor - Generering af stykliste**

Beskrivelse:

Som en sælger får jeg genereret en stykliste til carporten, så jeg ved, hvilke materialer der skal bruges.

Accept-kriterier:

- Given jeg har tilgået en kundes forespørgsel,
- When ordren er blevet oprettet i systemet,
- Then opretter systemet en liste over de nødvendige materialer.

Tasks:

- Implementer calcOptimalWood()-metoden, som optimerer brugen af materialer ved at minimere spild.
- Udvikl klassen ConstructionWood, så materialer kan sammenlignes på baggrund af type, længde og antal.
- Implementer metoder som calcFascia(), calcBeam(), calcPosts() og lignende, der hver især beregner specifikke materialetyper baseret på carportens dimensioner.
- Lav tests for hver beregningsmetode for at sikre korrekt funktionalitet.
- Skab en Carport-constructor, der samler alle beregninger og automatisk genererer en komplet stykliste ved oprettelse af et carport-objekt.

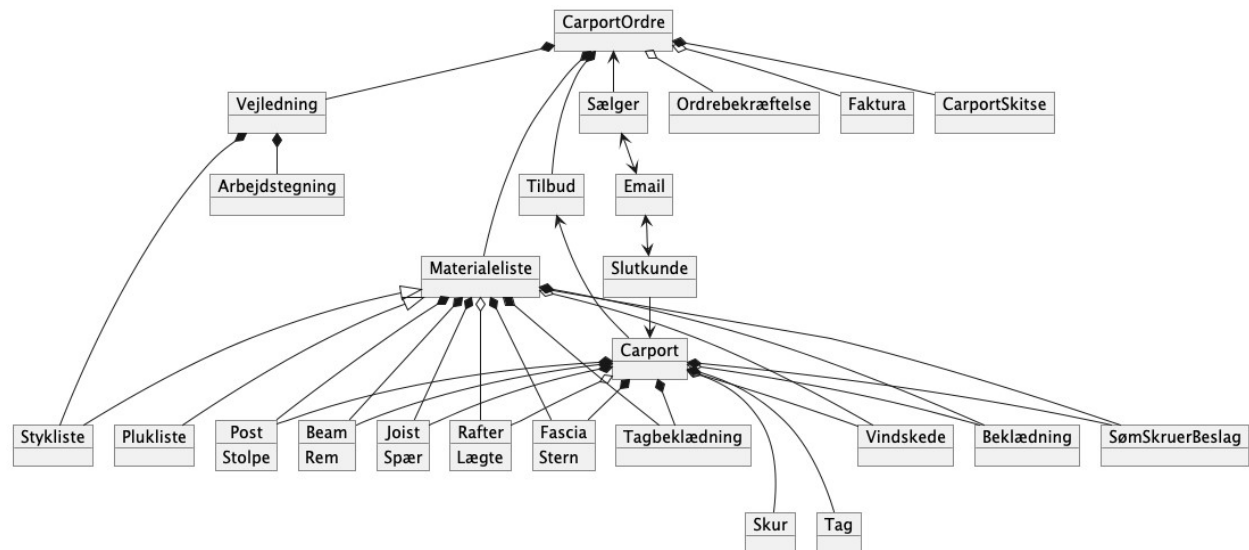
User-storyen er blevet estimeret som XL på grund af dens kompleksitet og omfang, da den involverer udviklingen af flere beregningsfunktioner samt en gennemgående test- og valideringsproces.

For at sikre, at user-storyen opfylder kundens behov, blev der udarbejdet klare accept-kriterier. Disse kriterier definerer, hvornår funktionen betragtes som færdig, og gør det muligt at validere løsningen. F.eks. skal systemet altid returnere en korrekt og optimeret stykliste baseret på kundens input.

Denne user-story er central for projektets hovedformål, da den udgør kernen i beregningsmotoren.

Dette eksempel demonstrerer, hvordan vi arbejder med user-stories i projektet, og hvordan de brydes ned i konkrete tasks, estimeres og valideres. Gennem denne proces sikrer vi, at kundens krav bliver opfyldt på en struktureret og effektiv måde.

## Domæne model og ER-diagram



Figur 8.1: Domæne model

Vores domæne model er udarbejdet samt tager udgangspunkt i videoen med Fog. I modellen bruges forskellige typer pile og symboler til at repræsentere relationer og afhængigheder mellem entiteterne:

Vi har gjort brug af en stærk relation (**composition**) kaldet den **sorte diamant** som markerer en **composition-relation**, hvor en del ikke kan eksistere uden helheden.

For eksempel:

- *Carport* har en stærk sammenhæng med materialelisten og de specifikke materialer såsom *Stolper*, *Rafte*, *Tagbeklædning*, og andre dele. Hvis en *Carport* ikke eksisterer vil disse materialer ikke give mening i systemet.

Der er brugt almindelige pile til at angive forbindelserne mellem entiteter.

For eksempel:

- *CarportOrdre* → *Vejledning*: En ordre kan inkludere en vejledning, der bruges som en arbejdstegning.

Vores domæne model har en hierarkisk opbygning, hvor visse entiteter samler og organiserer data

For eksempel:



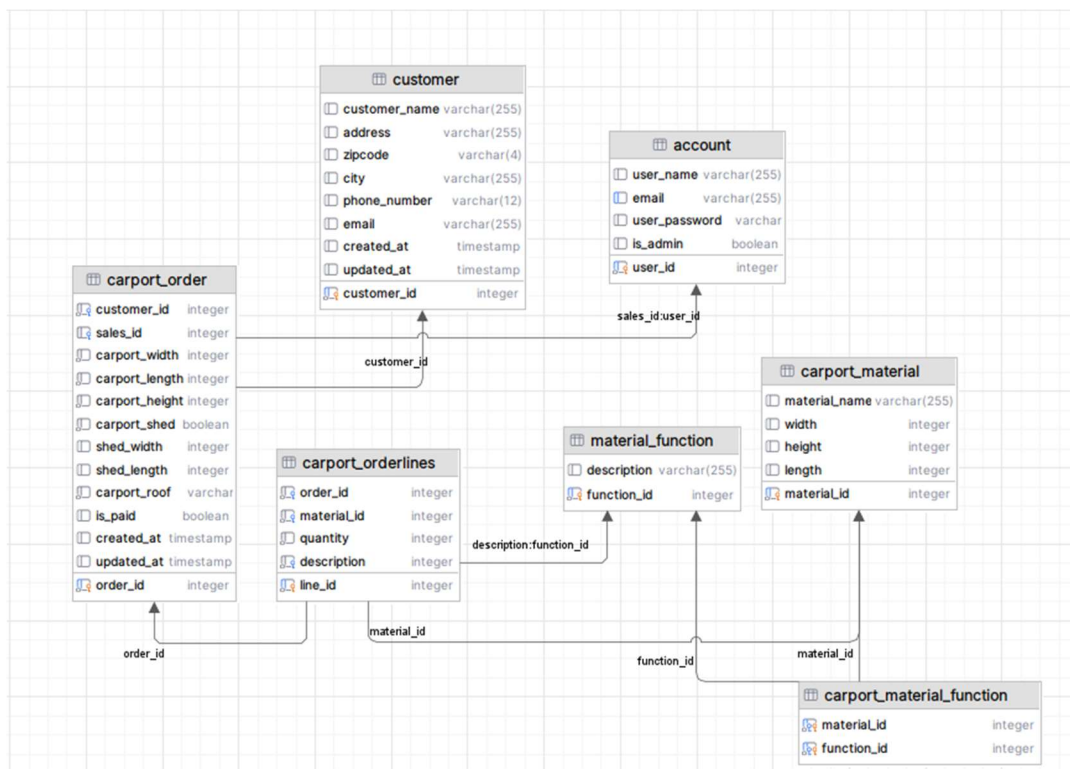
- *CarportOrdre* samler informationer fra slutkunden, tilbuddet og de nødvendige materialelister.
- *Carport* fungerer som en overordnet enhed, der forbinder alle nødvendige materialer og specifikationer (f.eks. *Tag*, *Skur*, *Beklædning* og *Vindskede*).

Vores domæne model viser, hvordan data overføres mellem entiteter og de forskellige dele af systemet.

For eksempel:

- Når slutkunden initierer en *CarportOrdre*, genereres der en *Materialeliste*, der indeholder de nødvendige materialer.
- En ordre fører til oprettelsen af *skitser*, *Vejledninger* og eventuelle arbejdstegninger.

Vores domæne model giver et klart samt struktureret overblik over systemets entiteter og deres reaktioner. Vi har designet modellen således at den stiller kravene til system og en god forudsætning for implementeringen.



Figur 8.2: ER-Diagram

## Sammenligning med domænemodel.

Entiteterne i domæne modellen går også igen som tabeller i vores database.

- **carport\_order** tabellen er i databasen oprettet sådan at den indeholder oplysninger om hver ordre med den specifikke kunde og sælger der er tilknyttet ordren.
- Databasen er opbygget således at den ikke kun har en materialeliste tabel. I vores database er "carport" og "materialeliste" opdelt i tre tabeller **carport\_material**, **material\_function** og **carport\_material\_function**. De tre tabeller fungerer sådan at **carport\_material** indeholder materialer med hver deres id, der skal bruges til at bygge carporten. Tabellen **material\_function** indeholder et function\_id med en tilhørende beskrivelse. Tabellen **carport\_material\_function** har så kun de to id'er der bruges til at koble materiale på baggrund af deres id sammen med hvilken funktion der tilhører det enkelte materiale.
- Vores database indeholder også en ENUM på attributten orderStatus i **carport\_order**. Her er der 10 forskellige statusser som en ordre kan have.

## Afvielser fra 3. Normal form

Vores database overholder 3. Normal form bortset fra hvad angår city og zipcode i **customer** tabellen. I tabellen afhænger city ikke af customer\_id men indirekte af zipcode og bryder 3. normal form da den ikke opfylder kravet til ikke at afhænge af andet end primærnøglen.

For at kunne have opfyldt 3. normal form skulle city og zipcode have ligget i deres egen tabel og derved havde city kun været afhængig af zipcode som den primære nøgle. Vi kunne derfor have beholdt zipcode i **Customer** tabellen og derved hente byen ud fra postnummeret.

## Primærnøgler

Vi bruger et automatisk genereret ID som primærnøgle i alle vores tabeller. Navnet på vores primærnøgler er lavet ved at tage tabelnavnet eller den sidste del af tabelnavnet og så sætte "\_id" bag på. Der er undtagelser på **account** og **carport\_order** tabellerne. I **account** tabellen har vi valgt at navngive primærnøglen som user\_id. Dette er gjort da vi har valgt at en sælger bliver oprettet som en user i programmet men da "user" og "order" er et reserverede ord i PostgreSQL har vi navngivet tabellerne **account** og **carport\_order** for at undgå konflikter. Primærnøglen i **carport\_order** tabellen er

navngivet som `order_id`, da vi har valgt at denne primærnøgle skulle tages med ud i frontend og vises til sælgeren i systemet.

## Fremmednøgler

Relationerne imellem vores tabeller er etableret vha. fremmednøgler. Vi har gjort brug af fremmednøgler i tabellerne **carport\_order**, **carport\_material\_function** og **carport\_orderlines**. Alle vores fremmednøgler er deklareret i PostgreSQL med foreign key constraints. Dette er gjort med henblik på en fordel, fordi PostgreSQL kun tillader, at vores Java-backend indsætter en række i f.eks. **orderline**-tabellen, hvis de refererede nøgler faktisk eksisterer som primærnøgler i de tilknyttede tabeller. Dette sikrer dataintegritet og forhindrer referencer til ikke-eksisterende data."

## Andre constraints

Vi har valgt at bruge en *unique* constraint på email attributten i **account** tabellen. Dette har vi valgt da hver sælger skal have deres eget login med email og password til webappen for at kunne tildele en ordre til sig selv i systemmet via sælgerens id. Det er derfor vigtigt at email er unikt for kun at definere en sælger med den email og id.

## Navigationsdiagram & Mockups

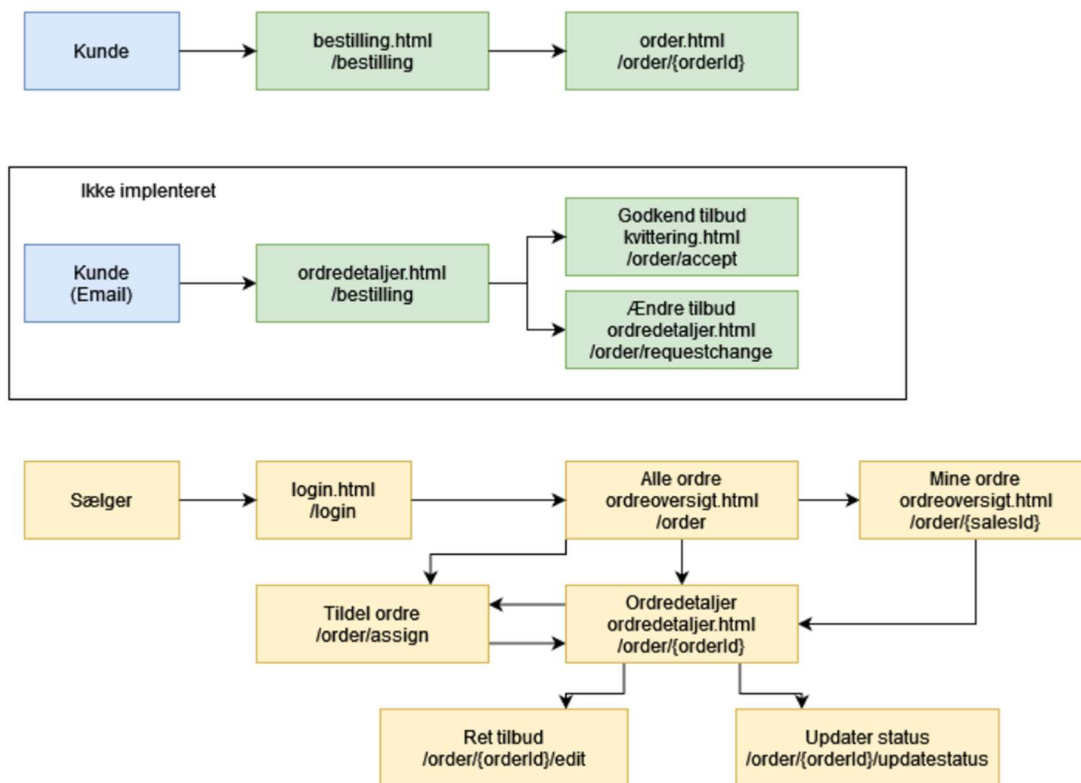
Dette afsnit giver et indblik i systemets funktioner og arbejdsprocesser gennem en række illustrative billeder. Systemet er udviklet med fokus på brugervenlighed og effektivitet, hvor både kunder og sælgere nemt kan oprette, administrere og behandle ordrer. De kommende billeder demonstrerer nøje, hvordan navigationsstrukturen, bestillingsformularen og redigeringsmulighederne spiller sammen for at sikre en smidig og overskuelig ordreproces. Målet er at skabe et klart overblik over de værktøjer, der understøtter en effektiv interaktion mellem kunde og sælger.

Mockups kan ses i bilag (figur 9.1.1-9.1.8).

### Navigationsdiagram

Billedet viser et navigationsdiagram for hele systemet. Diagrammet giver et visuelt overblik over de routes vores sider bruger for at komme rundt i systemet

Se bilag for mockups



Figur 9.1: Navigationsdiagram

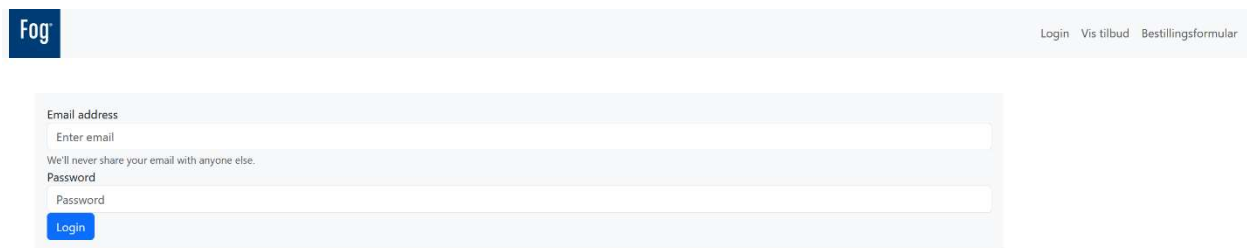
## Login siden

Dette billede viser login-siden som er den første side vi møder.

Her kan sælger logge ind med brugernavn og kodeord.

Siden fungerer som adgangskontrol og sikrer, at kun autoriserede brugere kan tilgå ordresystemet.

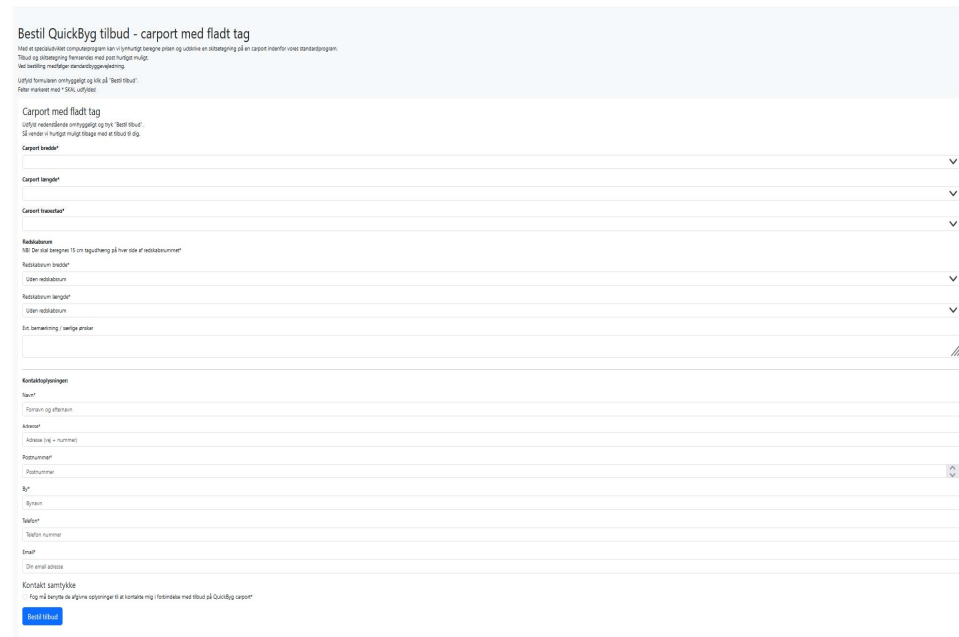
I navigationslinjen har vi bestillingsformularen som er den kunden ville skulle kunne se når de skal bestille en ny carport



Figur 9.2: Sælgers login side

## Bestillingsformular

Dette billede viser den formular, som kunden kan udfylde for at oprette en ordre. Formularen inkluderer Dimensioner og kontakthinformation på kunden



Figur 9.3: Bestillingsformular

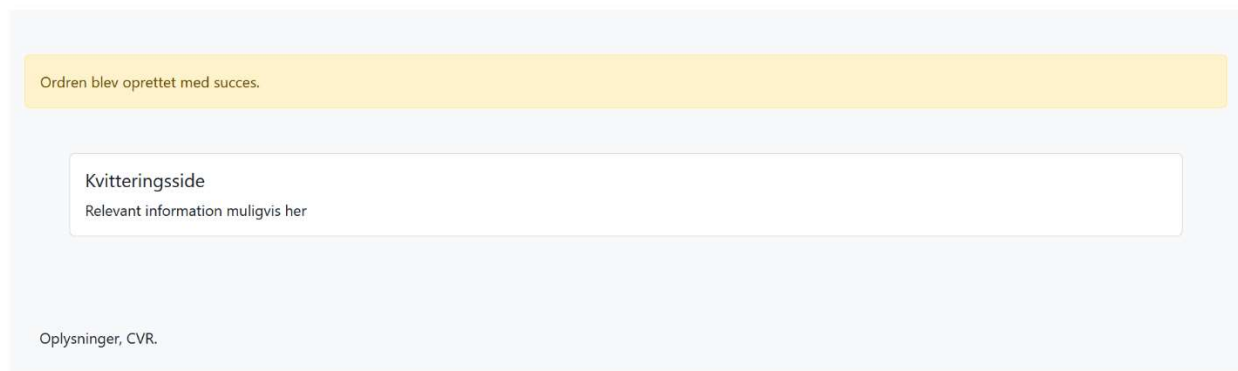
## Kvittering til kunden

Efter bestillingen vises en kvitteringsside. Her kan kunden se:

En bekræftelse på, at ordren er modtaget.

Ordrenummer og betalingsstatus.

En oversigt over de valgte dimensioner og materialer.



Figur 9.4: Kvittering

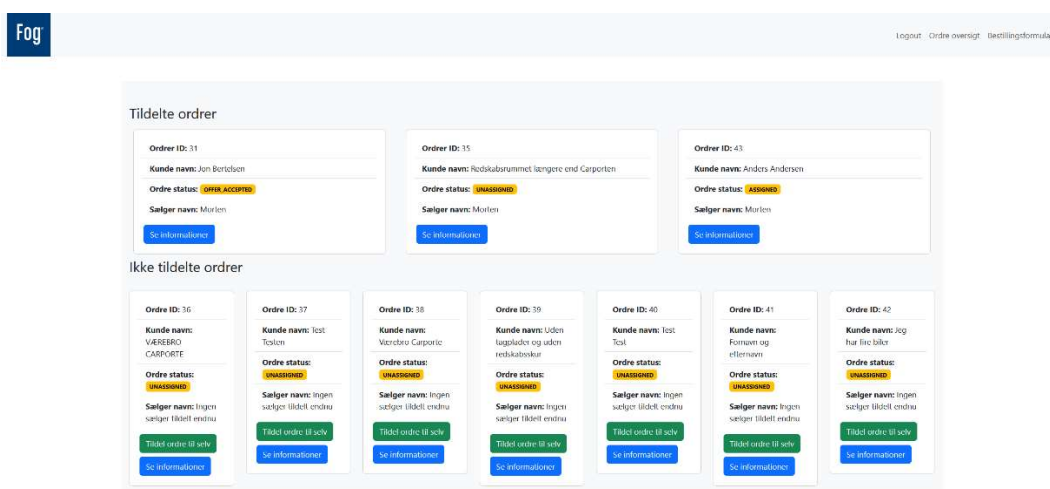
## Ordreoversigt

Når sælger logger ind, bliver de mødt af ordreoversigten. Denne oversigt viser en liste over alle ordrer med:

Ordre ID, Kundens navn, Ordre status (f.eks. "OFFER\_ACCEPTED").

Handlinger som at redigere, slette eller opdatere en ordre.

Ordreoversigten giver et hurtigt overblik over alle aktive og tidligere ordrer



Figur 9.5: Ordreoversigt

## Ordredetaljer

Her ses de fulde ordredetaljer for en enkelt ordre:

Dimensioner af carport og skur.

Tagtype og beklædning.

En oversigt over kundeinformation: navn, adresse, telefonnummer og e-mail.

**Ordredetaljer**

**Kundens ønsker**

Kunden har sendt følgende informationer ind via hjemmesiden:

Carport længde: 780 cm
Carport bredde: 600 cm
Skur længde: 0 cm
Skur bredde: 0 cm
Skur beklædning: forskudt
Tag type: FLAT
Tag beklædning: Trapez

**Kundens kontaktoplysninger**

Kunden har sendt følgende informationer ind via hjemmesiden:

Navn: Jon Bertelsen
E-mail: camelCase@sout.dk
Tlf: 69696969
Adresse: Datamatiker vej 1337
Postnr.: 2100
By: København Ø

**Produkt og sælger information**

Ordre ID: 31
Sælger: Morten
<b>Ordre status:</b> OFFER ACCEPTED
Ordren er betalt: true

Figur 9.6: Del 1/4 af Ordredetaljer

De næste billeder dykker ned i materialelisten. Hver række i tabellen inkluderer:

Materiale navn (f.eks. "25x200 mm trykimpr. bræt")

Længde og antal af materialer

Enhed (stk, pakke, rulle)

Beskrivelse af materialets anvendelse (f.eks. "monteres på spær").

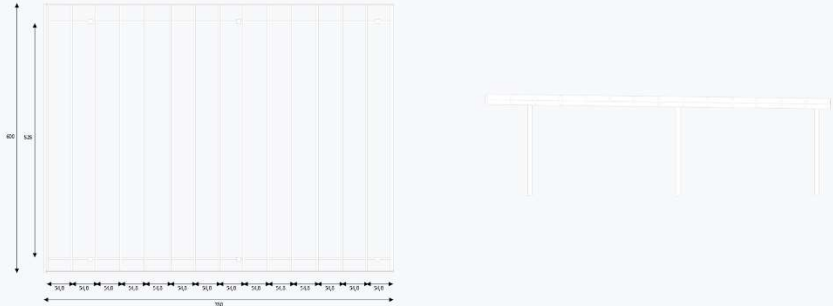
Sælger kan justere antal materialer ved at ændre værdier i inputfelterne. Dette sikrer, at materialelisten altid er opdateret.

Materialeliste og priser

Ordre status: OFFER ACCEPTED

Ordren er betalt: true

Seneste opdatering: 2024-12-13T10:17:41.776593



Materiale navn	Længde	Antal	Enhed	Beskrivelse
Træ og tagplader				
25x200 mm. trykimp. bræt	360	7	stk	understernbrædder
25x200 mm. trykimp. bræt	540	1	stk	understernbrædder
25x125 mm. trykimp. bræt	360	7	stk	oversternbrædder
25x125 mm. trykimp. bræt	540	1	stk	oversternbrædder
45x195 mm. spærtræ ubh.	600	1	stk	Remme i sider, sadles ned i stolper
45x195 mm. spærtræ ubh.	540	2	stk	Remme i sider, sadles ned i stolper
97x97 mm. trykimp. stolpe	300	6	stk	Stolper nedgraves 90 cm. i jord
45x195 mm. spærtræ ubh.	300	15	stk	Spær, monteres på rem
Plastmo Ecolite blåtonet	600	6	stk	tagplader monteres på spær
Plastmo Ecolite blåtonet	360	6	stk	tagplader monteres på spær
Skruer og beslag				
plastmo bundskruer 200 stk	0	3	pakke	Skruer til tagplader
universal 190 mm højre	0	15	stk	Til montering af spær på rem
universal 190 mm højre	0	15	stk	Til montering af spær på rem
4,0 x 50 mm beslagskruer 250 stk.	0	2	pakke	Til montering af universalbeslag + hulbånd
4,5 x 60 mm skruer 200 stk.	0	1	pakke	Til montering af stern&vandbrædt
hulbånd 1x20 mm. 10 mtr.	0	1	rulle	Til vindkryds på spær
bræddebolt 10 x 120 mm	0	16	stk	Til montering af rem på stolper
bræddebolt 10 x 120 mm	0	16	stk	Til montering af rem på stolper

Carport indkøbspris: 5481.0 kr

Carport foreslået salgspris: 8221.5 kr

Avance procent 1.5

Tildel

Send tilbud

Ret tilbud

Oplysninger, CVR.

Figur 9.7: Del 2/2 af Ordreoversigt





## Materialeliste og handlinger

Formularen inkluderer også den fulde materialeliste og nødvendige skruer og beslag. Sælger kan:

Justere antal materialer via dropdown-menuer.

Genberegne ordren ved hjælp af knappen "Genberegner".

Opdatere ordrestatus i en dropdown (f.eks. "OFFER\_ACCEPTED").

Handlingerne afsluttes med:

Gem ændringer via den grønne knap.

Annuller via den røde knap.

25x200 mm. trykimp. bræt	540	<input type="text" value="1"/>	stk	understernbrædder
25x125 mm. trykimp. bræt	360	<input type="text" value="7"/>	stk	oversternbrædder
25x125 mm. trykimp. bræt	540	<input type="text" value="1"/>	stk	oversternbrædder
45x195 mm. spærtræ ubh.	600	<input type="text" value="1"/>	stk	Remme i sider, sadles ned i stolper
45x195 mm. spærtræ ubh.	540	<input type="text" value="2"/>	stk	Remme i sider, sadles ned i stolper
97x97 mm. trykimp. stolpe	300	<input type="text" value="6"/>	stk	Stolper nedgraves 90 cm. i jord
45x195 mm. spærtræ ubh.	300	<input type="text" value="15"/>	stk	Spær, monteres på rem
Plastmo Ecolite blåtonet	600	<input type="text" value="6"/>	stk	tagplader monteres på spær
Plastmo Ecolite blåtonet	360	<input type="text" value="6"/>	stk	tagplader monteres på spær
<b>Skruer og beslag</b>				
plastmo bundskruer 200 stk	0	<input type="text" value="3"/>	pakke	Skruer til tagplader
universal 190 mm højre	0	<input type="text" value="15"/>	stk	Til montering af spær på rem
universal 190 mm højre	0	<input type="text" value="15"/>	stk	Til montering af spær på rem
4,0 x 50 mm beslagskruer 250 stk.	0	<input type="text" value="2"/>	pakke	Til montering af universalbeslag + hulbånd
universal 190 mm højre	0	<input type="text" value="15"/>	stk	Til montering af spær på rem
4,0 x 50 mm beslagskruer 250 stk.	0	<input type="text" value="2"/>	pakke	Til montering af universalbeslag + hulbånd
4,5 x 60 mm skruer 200 stk.	0	<input type="text" value="1"/>	pakke	Til montering af stern&vandbrædt
hulbånd 1x20 mm. 10 mtr.	0	<input type="text" value="1"/>	rulle	Til vindkryds på spær
brædebolt 10 x 120 mm	0	<input type="text" value="16"/>	stk	Til montering af rem på stolper
brædebolt 10 x 120 mm	0	<input type="text" value="16"/>	stk	Til montering af rem på stolper

Figur 9.9: Del 2/2 af redigering af tilbud

## Valg af arkitektur

### Brug af MVC design pattern

Et af kravene til opgaven var at vi skulle udvikle en multipage webapplikation med brug af Javalin web framework, Thymeleaf og en Hikari connection pool der snakker med en PostgreSQL database. Vi har valgt en arkitektur der inspireret af MVC design mønstret. MVC-mønstret består af følgende dele

**Model:** Repræsenterer forretningslogikken og dataene i systemet. Det er modellen der har ansvaret for at hive ting ind og ud af databasen, og repræsentere data fra databasen som objekter. Vi har delt modellen op i tre packages:

- persistence**, der indeholder alle klasser som snakker med databasen.

- entities**, der indeholder de klasser som modellerer forretningslogikken, og

- services**, der er nogle hjælpeklasser som entity klasserne bruger.

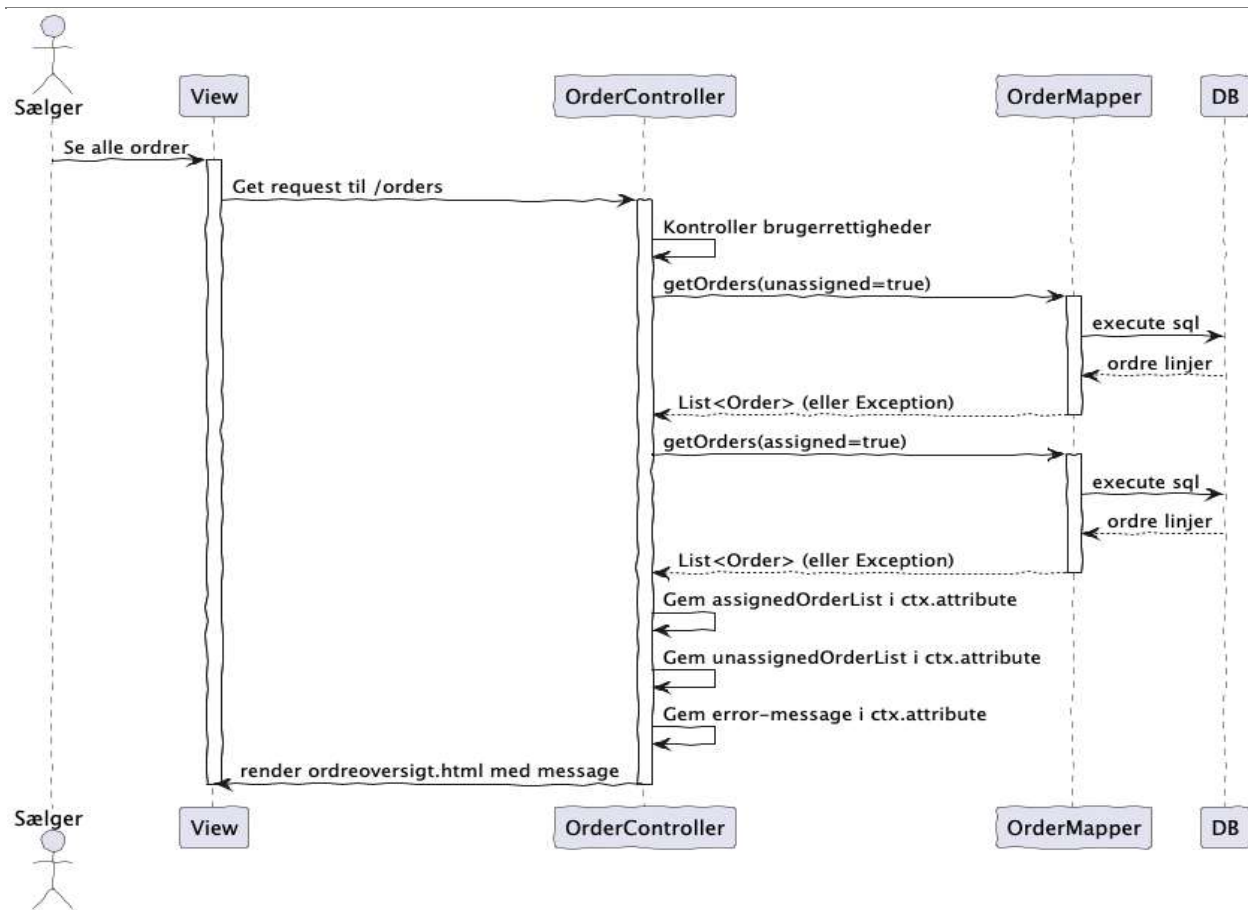
**View:** Har ansvaret for at vise data fra Modellen til brugeren, i vores tilfælde i form af dynamiske HTML sider. Vi bruger Thymeleaf, der kan indsætte data fra vores Model i de HTML templates vi har lavet. I vores HTML bruger vi Bootstrap, et frontend CSS og JavaScript framework, der er designet til nemt at lave mobilvenlige hjemmesider.

**Controller:** Fungerer som en mellemmand mellem View og Model. Controlleren modtager bruger input i form af HTTP forespørgsler, interagerer med modellen, og sender relevante data fra Modellen videre til View. Javalin står for at abstrahere den del med at modtage HTTP forespørgsler væk, så vi bare skal koncentrere os om at definere hvilke routes/end points der skal være tilgængelige for brugeren, samt at skrive logikken der styrer interaktionen med Modellen. Denne kode har vi lagt i en package kaldet **controllers**.

Vi har desuden en **config** package, som Javalin og Thymeleaf skal bruge for at virke, og en **exceptions** package med to Exception klasser, så vi kan generere bedre fejlmeddelelser.

## Oversigt over packages i projektet

Package	Beskrivelse
<b>config</b>	Boilerplate fra Javalin og Thymeleaf
<b>controllers</b>	En OrderController og en UserController
<b>entities</b>	Del af Model
<b>exceptions</b>	For bedre fejlmeddelelser
<b>persistence</b>	Den del af Model som snakker med databasen
<b>services</b>	Hjælpeklasser til Model



Figur 10.2.1: Sekvensdiagram over MVC-arkitektur

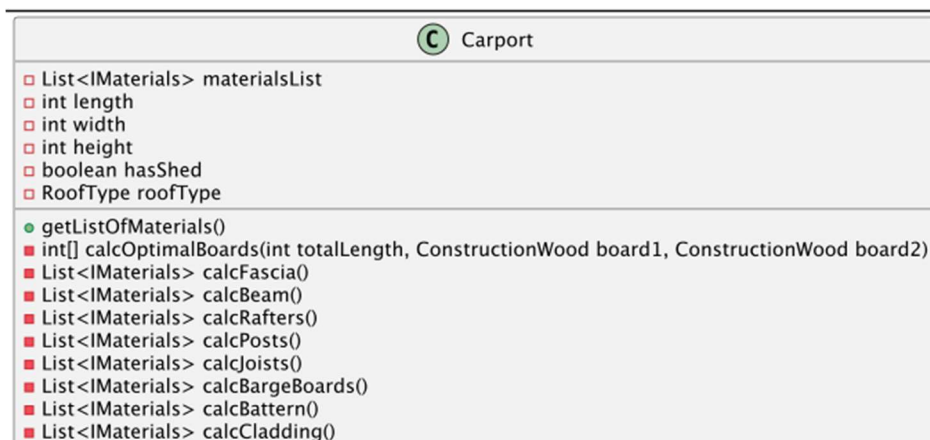
Ovenstående sekvensdiagram illustrerer hvordan vi bruger MVC mønsteret. Brugeren interagerer med Viewet, ved at have trykket på et link, så der sendes et get request til serveren. OrderControllerens showOrders() metode har ansvaret for at håndtere de forespørgsler, som sendes til "/orders". I dette tilfælde kontrollerer programmet om der er et User objekt gemt i session attributten "currentUser".

Hvis det er tilfældet, så kalder den på OrderMapperens metode `getOrders()`, som returnerer to lister af Order objekter en med tildelte ordrer og en med ikke-tildelte ordrer. OrderMapperen hører til Modellen og det gør Order klassen også. Controlleren gemmer de to lister i hver sin attribute, og eventuelle Exceptions gribes og laves om til tekst-fejlmeddelelser, der gemmes i en message attribute. Thymeleafs template motor (Viewet) har adgang til disse attributes, og bruger dem til at generere HTML og et HTTP response som sendes til brugeren (eller rettere brugerens browser).

## Brug af Strategy Design Pattern

Hen imod slutningen af projektperioden blev det tydeligt for os at vi havde brug for mere end én måde at beregne en carport på. Derfor valgte vi at begynde at omskrive koden sådan at vi kunne isolere metoderne vi bruger til at beregne stykliste og andre vigtige parametre der skal bruges til at tegne carporten, og lægge dem ud i sin egen klasse. Herefter skabte vi et interface med netop disse metoder. På den måde er det muligt at lave en alternativ implementering af interfacet, og have mulighed for, under runtime, at skifte mellem forskellige beregningsmotorer.

I figur 10.3.1 kan vi se en Carport klasse som indeholder en masse private metoder til at beregne de forskellige materialer, som skal tilføjes til materialelisten.



Figur 10.3.1: Carport klassen før refaktoring.

I figur 10.3.2 kan vi se hvordan beregningsmetoderne er flyttet ud i et interface, og to mulige implementeringer af det interface. Carport klassen indeholder nu en `CarportCalculator`, hvilket gør at vi kan skifte beregningsmetode undervejs, mens programmet kører.



## Særlige forhold

### Beregningsmotoren version et

Som det fremgår af figur 10.3.1 forestillede vi os til at starte med at Carport klassen kunne indeholde alle de nødvendige metoder til at beregne styklisten og alle andre informationer som måtte være relevante at beregne når man skal producere arbejdstegninger til at bygge en carport. Vi gik derfor i gang med at prøve at gennemskue hvordan den udleverede stykliste kunne beregnes ud fra de informationer om længde, bredde og skur, som kunden har leveret. Et af de problemer vi stødte på, var hvordan man havde besluttet sig for at der skal bruges 4 stk. 360 cm lange brædder til understern i for og bagende, samt 4 stk. 540 cm lange brædder til understern i siderne, når målene på carporten var 600 cm i bredden (for og bagende), og 780 cm i længden (siderne). Da der på de to sider tilsammen skal bruges 1560 cm brædder, så er det tilstrækkeligt at købe 3 stk. 540 cm brædder, idet  $3 \cdot 540 = 1620$  . Vi besluttede derfor at lave en metode, der kunne tage en ønsket total længde og en liste af bræddelængder som input, og returnere et array af antal brædder i en de givne længder, som ville give mindst muligt spild.

I første version af beregningsmotoren blev alle materialerne instantieret som objekter, og tilføjet til materialelisten inde i carportobjektet. Dette vidste vi godt, at vi gerne ville lave om på et senere tidspunkt. Vi ville nemlig gerne hente materialerne ud af databasen gennem en metode i vores MaterialMapper. Dette var en af grundene til at vi senere lavede beregningsmotoren om.

Figur 11.1 viser kildekoden til en version af metoden der beregner de optimale antal brædder der skal bruges. Vi lærte undervejs, at det var en god idé at have en spildfaktor med, sådan at tømreren har noget træ at miste af når der saves i brædderne. Men hvis, der ikke skal saves i brædderne behøves vi heller ikke have noget spild. Derfor undersøger vi om der er en bræddelængde som går op i den ønskede totallængde, og i så fald sætter vi spildprocenten til 0. Ellers sætter vi spildprocenten til 5%. Det ville være rart at spildprocenten kunne justeres, men det nåede vi ikke at implementere. Den første version af metoden kunne kun håndtere to længder brædder, men det lykkedes os at lave denne version, som kan håndtere et vilkårligt antal brædder. Det er vigtigt at huske, når man bruger metoden, at den foretrækker løsninger som ligger først i listen, dvs. Hvis to længder brædder giver samme mængde spild, så foretrækker metoden det bræt som den har beregnet først, altså det som ligger først i listen.

```

1. public int[] calcOptimalWood(int totalLength, List<IMaterials> conWoodList) {
2.     double wastePercentage = 1.05;
3.     boolean exactMatchFound = conWoodList.stream().anyMatch(board -> totalLength %
board.getLength() == 0);
4.     if (exactMatchFound) {
5.         wastePercentage = 1.0;
6.     }
7.     double totalLengthInclWaste = totalLength * wastePercentage;
8.     int maxPieces = 20;
9.
10.    int[] bestResult = new int[conWoodList.size()];
11.    double bestWaste = Double.MAX_VALUE;
12.
13.    int[] indices = new int[conWoodList.size()];
14.
15.    while (true) {
16.        double curentLength = 0;
17.        for (int i = 0; i < conWoodList.size(); i++) {
18.            curentLength += indices[i] * conWoodList.get(i).getLength();
19.        }
20.        if (curentLength >= totalLengthInclWaste) {
21.            double waste = curentLength - totalLengthInclWaste;
22.            if (waste < bestWaste) {
23.                bestWaste = waste;
24.                System.arraycopy(indices, 0, bestResult, 0, indices.length);
25.            }
26.        }
27.        int index = 0;
28.        while (index < conWoodList.size()) {
29.            if (indices[index] < maxPieces) {
30.                indices[index]++;
31.                break;
32.            } else {
33.                indices[index] = 0;
34.                index++;
35.            }
36.        }
37.        if (index == conWoodList.size()) {
38.            break;
39.        }
40.    }
41.    return bestResult;
42. }

```

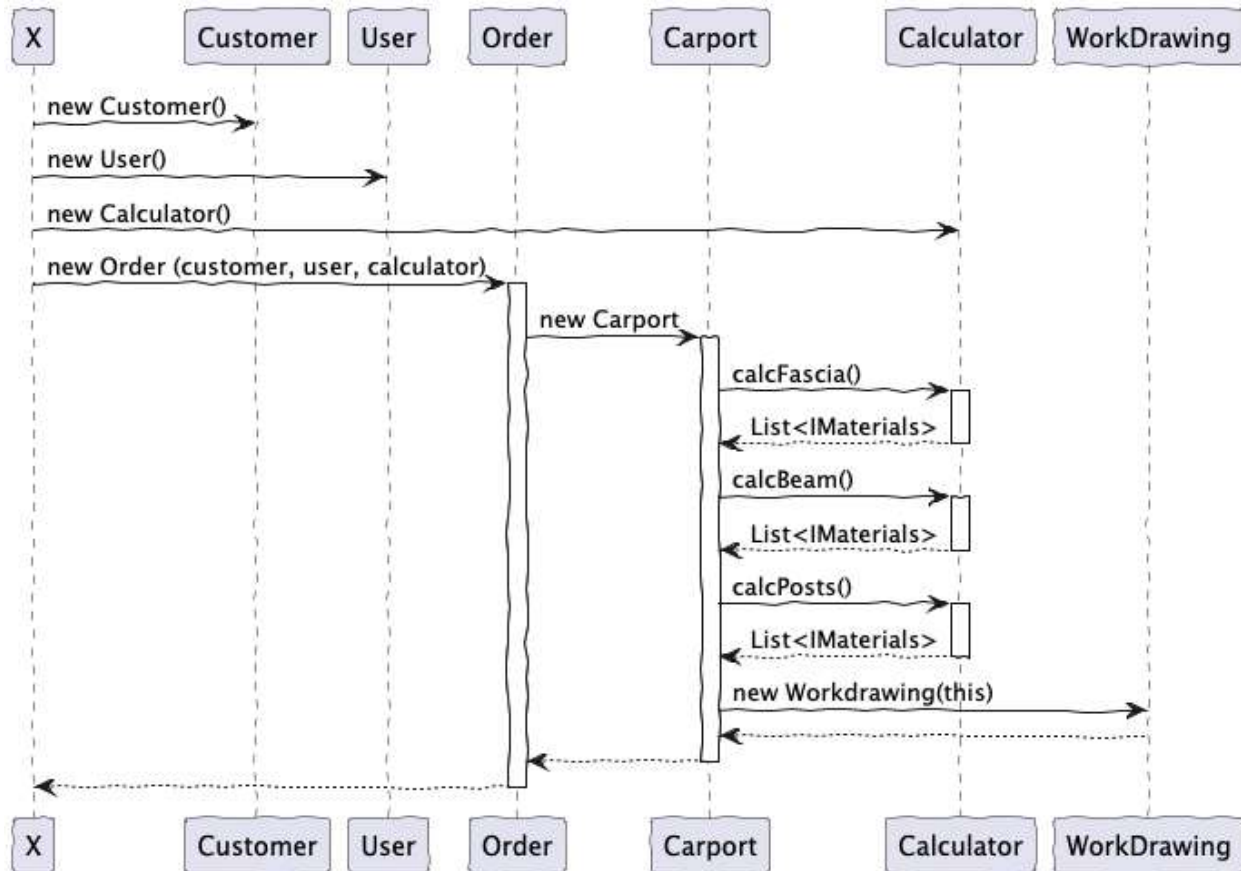
Figur 11.1 – Beregningsmotor

Da vi nåede uge 3 i vores kodning, gik det op for os at der var et problem med den måde vi laver vores carport på. Vi beregner altid den optimale carport. Der er ikke nogen mulighed for at sælgeren kan ændre i den carport vores beregningsmotor har lavet. Det er et kæmpe problem, fordi vi ved at Fog lægger meget vægt på at de kan bruge deres ekspertise til at vejlede kunden til den rigtige carport. Og vi tror ikke på at vores beregningsmotor altid laver den rigtige carport for kunden. Vi har derfor brug for en



alternativ beregningsmotor, som kan producere arbejdstegninger (og andre nødvendige informationer), ud fra en materialeliste som sælgeren har tilpasset.

## Samspillet mellem Order og Carport



Figur 11.2 - Sekvensdiagram af flow ved oprettelse af ny ordre

Vi ønskede at være i stand til at håndtere (mindst) to forskellige måder at beregne carporte på. Først en metode baseret på vores OptimalWood tankegang, og bagefter en metode baseret på en materialeliste som en rigtig ekspert (sælgeren fra Fog) har kontrolleret. Det fik os til at overveje et Strategy Design Pattern<sup>1</sup>, som går ud på at indkapsle en algoritme, eller en samling af beregninger, i en separat klasse, og gøre det muligt at udskifte den. I figur 11.2 har vi skitseret hvordan Order klassen og Carport klassen hænger sammen, efter vi har introduceret et Calculator interface. Vi har udeladt dimensionerne for overblikkets skyld.

<sup>1</sup> Som beskrevet i Freeman & Robsen: Head First Design Patterns

For at lave en ny ordre skal vi bruge dimensioner på en carport og eventuelt skur, en customer, en sælger (User) og en calculator. Når vi opretter ordren, laver vi et nyt carport objekt som bruger calculatoren til at udregne alle materialer, for at sætte dem ind i lister. Vi kalder herefter vores WorkDrawing klasse som laver tegningen af den ønskede carport. Den der laver Order objektet kan vælge hvilken slags Calculator, Carport objektet skal bruge.

## Beregningsmotor version to

Vores version 2 af styklister beregneren flyttede vi ned i vores service package, lavede et CarportCalculator interface og to calculator metoder.

OptimalWoodCalculator er en dynamisk calculator som beregner materialer ud fra de dimensioner den givne carport har.

FromListCalculator er en calculator som bruger en allerede eksisterende liste af materialer, til at beregne de nødvendige de ting som WorkDrawing skal bruge til at lave arbejdstegningerne.

Da OptimalWoodCalculator henter de tilgængelige materialer ud af databasen har den brug for en database forbindelse. Vi brød os ikke om at en klasse i entity pakken havde brug for en databaseforbindelse, og flyttede derfor alle Calculator klasserne ned i services.

## Exceptions håndteres i Controlleren

Vi har arbejdet med en intention om at gribe alle de Exceptions vi kan i Controlleren, og så vise en relevant fejlbesked for brugeren. Det er primært i Mapper klasserne at vi griber SQL Exceptions og laver dem om til DatabaseExceptions. Da vi gerne vil gribe Exceptions i Controlleren, og vores ene CarportCalculator taler med databasen (og derfor potentielt kan kaste DatabaseExceptions), mens den anden CarportCalculator ikke snakker med databasen (og derfor ikke kaster DatabaseExceptions), har vi valgt at vores CarportCalculator interface definerer at der skal kastes CalculatorExceptions. På den måde har vi også forsøgt at fremtidssikre vores program, forstået på den måde at fremtidige implementeringer af CarportCalculator interface nu selv kan gribe eventuelle Exceptions og lave dem om til CalculatorExceptions.

Figur 11.3 viser et udsnit af koden fra OptimalWoodCalculator klassen, hvor man kan se hvordan vi griber den DatabaseException som MaterialMapper.getMaterialOfTypeAndLength() kan kaste, og vælger at kaste den videre op i systemet som en CalculatorException.

```

1. public List<IMaterials> calcBeam() throws CalculatorException {
2.     try {
3.         List<IMaterials> beamList = new ArrayList<>();
4.         List<IMaterials> allBeamList = MaterialMapper.getMaterialOfTypeAndLength(
5.             "Remme i sider, sadles ned i stolper", (carportLength / 2), dbConnection
6.         );
7.         int totalLength = carportLength * 2;
8.
9.         int[] optimalWood = calcOptimalWood(totalLength, allBeamList);
10.
11.         for (int i = 0; i < optimalWood.length; i++) {
12.             if (optimalWood[i] > 0) {
13.                 IMaterials beam = (allBeamList.get(i).setAmount(optimalWood[i]));
14.                 beamList.add(beam);
15.             }
16.         }
17.         return beamList;
18.     } catch (DatabaseException e) {
19.         throw new CalculatorException(e.getMessage());
20.     }
21. }

```

Figur 11.3 – Metoder der griber en database exception, og kaster en ny calculator exception.

Vi har også en del steder hvor der er potentiale for NumberFormatExceptions, som gribes og håndteres. Vi har primært brugt IntelliJ til at se hvilke Exceptions de metodekald vi bruger, kan kaste.

Figur 11.4 viser OrderControllerens createOrder() metode, som et eksempel på at vi catcher diverse exceptions, for så at vise en fejlmeddelelse til brugeren. Det er dog at bemærke, grundet code freeze fik vi aldrig implementeret vores CalculatorException som set i linje 15-19 i figuren.

```

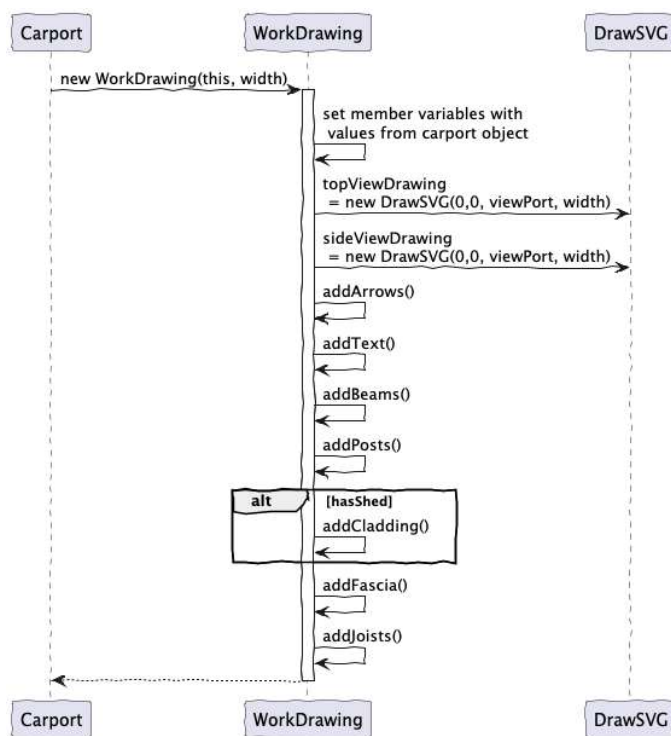
1. public static void createOrder(Context ctx, ConnectionPool dbConnection)
2. {
3.     try
4.     { ... }
5.     catch (NumberFormatException e)
6.     {
7.         ctx.attribute("message", "Ugyldige dimensioner");
8.         ctx.render("bestilling.html");
9.     }
10.    catch (DatabaseException e)
11.    {
12.        ctx.attribute("message", "Databasefejl: " + e.getMessage());
13.        ctx.render("bestilling.html");
14.    }
15.    catch (CalculatorException e)
16.    {
17.        throw new RuntimeException(e);
18.    }
19. }

```

Figur 11.4 – Udklip af en metode som catcher vores exceptions, og viser det som besked til brugeren

## Arbejdstegning

Vi valgte at bruge skitsen til den DrawSVG klasse, som vi havde fået udleveret<sup>i</sup> som en hjælp. Formålet med klassen er at kunne producere et korrekt formateret svg billede bestående af nogle "byggeklodser"; rette linjer, rektangler, pile og tekst. Klassens toString()-metode returnerer et korrekt formateret HTML <svg> element. Herefter har vi lavet en WorkDrawing klasse, hvis formål er at producere de to arbejdstegninger, topView og sideView. WorkDrawing klassen fungerer ved at instantiere to DrawSVG objekter, og har herefter en masse private metoder, som udregner position og tilføjer de forskellige elementer som arbejdstegningerne består af, stolper, remme, lægter osv.



Figur 11.2 - Sekvensdiagram af flow ved oprettelse af ny ordre

Vi startede med at udvikle en WorkDrawing klassen sådan at den kunne producere to model carporte, og havde planer om at vende tilbage til klassen senere i projektet, for at tilføje funktionalitet til at håndtere andre størrelser skur, samt placering af skur i højre og venstre side. Dette nåede vi desværre ikke.

## CurrentUser gemmes i Session

Når man logger ind på siden, gemmes et User objekt i en session variabel (currentUser).

## Status på implementering

Vi startede med at dele hele projektet op i 3 sprints, hvoraf sprint 1 var obligatoriske og højt prioriterede tasks, sprint 2 var lavt prioriterede tasks og sprint 3 var *“nice to have”* tasks. Grundet et måske for urealistisk mål for projektet som også er beskrevet længere nede, har vi måtte vælge at fokusere på at få gjort sprint 1 færdig. Det betyder også at der er flere user stories fra sprint 2 og 3 som ikke er påbegyndt endnu. Da vi nåede til code freeze var alle user stories med tilhørende tasks fra sprint 1 lavet med en undtagelse af user story 23 – redigering af stykliste. Denne userstory er begyndt men mangler den sidste funktionalitet for at kunne redigere og opdatere styklisten korrekt. Vi startede med 25 user stories, hvoraf vi har fået implementeret 12. Vi har 3 som er delvist implementeret men mangler den sidste funktionalitet og 10 som ikke er påbegyndt.

Vi har fået lavet alle vores websider til programmet som er stylet pænt og professionelt.

Vi ville gerne have nået at implementere et mailsystem sådan at sælger fik en mail ang. Nye ordre og kunden modtog en mail med tilbud. Vi har oprettet profilen på sendgrid samt opsat api nøglen, sender Authentication og lavet en dynamisk template for e-mail. Klassen SendGrid er også lavet i projektet men mangler stadig logikken til at kunne sende mails. Da mail systemet ikke er implementeret kan der ikke sendes en mail til kunden med tilbuddet og et link til hvor kunde enten kan acceptere eller afvise tilbuddet.

Som programmet er lige nu, har vi en fast avance procent som er hardcodet ind i vores kode. Vores mål var at sælger kunne tilpasse avanceprocenten til den enkelte kunde. Denne feature har vi ikke nået at implementere så den eneste måde for at ændre avanceprocenten er ved at ændre den i selve koden.

User story 23 som går ud på at redigere i styklisten er påbegyndt. Frontend delen er lavet og man kan opdatere hvilken status den enkelte ordrer skal have. Der mangler fortsat at få implementeret logikken bag ved at ændre i selve styklisten, sådan at den opdaterer og gemmer den nye stykliste korrekt.

User story 18 som håndtere om kunden accepterer eller afviser tilbuddet mangler funktionalitet på afvis knappen. Som kunde kan man acceptere tilbuddet men ikke afvise det. Vi kan som sælger godt gå ind på den enkelte ordre og opdatere status til tilbud afvist. Når man som kunde trykker på afvis bliver man ført over til den rigtig webseite hvor man kan give en forklaring for afvisningen. Fra denne side kan man ikke komme videre med at afvise tilbuddet, da logikken mangler.

Vi har fundet en lille bug i programmet som tillader at skuret f.eks. er længere end carporten. Vores generering af carportskiten kan derfor tegne skitsen kun på baggrund af de indtastede værdier fra kunden. Der mangler at blive valideret om hvorvidt målene passer sammen og kan lade sig gøre.

CRUD-konceptet er blevet anvendt som en retningslinje for implementeringen af databasemetoder i dette projekt. CRUD dækker over de fire grundlæggende operationer: **Create, Read, Update** og **Delete**, som tilsammen muliggør effektiv datahåndtering.

#### CREATE:

Vi har fået implementeret CREATE metoder til følgende tabeller i vores database. Carport\_order, carport\_orderlines og customer.

#### READ:

Tabellerne Carport\_order, customer, account, carport\_material\_funktion, material\_function og carport\_material har alle metoder til at hente og læse fra databasen.

#### UPDATE:

Det er kun tabellen carport\_order som har fået implementeret metoder til at kunne opdatere dataene i tabellen.

#### DELETE:

Vi har ikke nået at få lavet DELETE metoder til nogle af tabellerne.

Vi har formået at implementere størstedelen af de vigtige funktioner, som gør det muligt for kunder at indsende forespørgsler og for at sælgere kan behandle dem, tildele ordrer og generere tilbud. De delvist implementerede funktioner omfatter primært notifikationer og tilbudsafsendelse, som kræver yderligere arbejde. De ikke-implementerede user stories omfatter avancerede funktioner som materialehåndtering, prisopdateringer og administrative værktøjer for salgsschefen.

Det samlede system er funktionelt i sig selv og opfylder hovedformålet, men mangler visse funktioner for at være helt færdigt.

## Automatiserede tests

Vi startede projektet med en klar forventning om, at vi ville udvikle vores beregningsmotor ud fra Test-Driven Development (TDD). Vi begyndte derfor med at skrive tests til alle metoderne, inden vi implementerede dem i carportklassen.

Senere i udviklingsprocessen besluttede vi, at det ville være smartere at udskille beregningsmetoderne i deres egen klasse. Først flyttede vi metoderne til en ny klasse i samme package, men senere flyttede vi klassen til en anden package for bedre struktur. Desværre opdaterede vi ikke vores tests efter denne refaktoring. Dette betyder, at OptimalWoodCalculator, som nu indeholder beregningsmetoderne, ikke længere er testet.

Men inden vi refaktorerer OptimalWoodCalculator til vores service package, så vores dækning således ud, herunder er et direkte udtag af vores 'Code Coverage' fra IntelliJ

Klasse	Testede Metoder	Linjedækning (%)	Kommentar
OptimalWoodCalculator	92% (24/26)	97% (226/232)	Høj testdækning af beregningsmetoder.
Carport	68% (13/19)	88% (47/53)	Gammel beregningslogik er delvist dækket.
CarportCalculator	100% (0/0)	100% (0/0)	Interface, ingen implementeret kode.
BoltsScrewsBrackets	28% (2/7)	64% (9/14)	Lav dækning af logik.
ConstructionWood	14% (1/7)	60% (9/15)	Minimal testdækning.
RoofCovering	14% (1/7)	57% (8/14)	Minimal testdækning.

Figur 13.1: Tabel over code coverage

Vi var godt klar over at refaktoring uden at opdatere vores tests kunne føre til manglende testdækning. Dette er især vigtigt for OptimalWoodCalculator, som indeholder kritiske beregningsmetoder, der ikke længere blev testet. I fremtidige projekter vil vi prioritere at opdatere vores tests, når vi refaktorerer koden, for at sikre højere testdækning og stabilitet.

## User Acceptance tests

I dette afsnit vil vi via en tabel danne et overblik over vores acceptance kriterier, samt hvorvidt de er godkendt eller ej.

Vi har arbejdet med formatet, **Given, When, Then** til udarbejdelse af disse acceptance kriterier. Disse kriterier er med for at der er en organisk afslutning på en user story. Hvis ikke vi har en instans af hvornår en user story er færdig, er det svært at komme videre fra én task til en anden.

ID	User stories	Acceptance criteria	Godkendt/ ikke godkendt
1.	Som en kunde kan jeg udfylde en formular på FOGS hjemmeside, så jeg kan sende en forespørgsel til FOG.	<b>Given</b> jeg besøger FOGS hjemmeside for at bestille en custom carport, <b>When</b> jeg udfylder og indsender formularen med mine oplysninger og forespørgsel, <b>Then</b> modtager systemet min forespørgsel og sender en bekræftelse på slutkundens skærm.	Godkendt
2.	Som en sælger kan jeg blive notificeret på mail om nye forespørgsler, så jeg hurtigt kan reagere og tildele mig selv en ordre.	<b>Given</b> en kunde har sendt en forespørgsel, <b>When</b> systemet registrerer forespørgslen, <b>Then</b> kan jeg som sælger modtage en notifikation via e-mail med de relevante detaljer.	Ikke godkendt, denne user story er ikke implementeret, da den er en del af sprint 2.
3.	Som en sælger kan jeg logge ind i programmet, så jeg kan tilgå nye og eksisterende ordrer.	<b>Given</b> jeg har adgang til systemets login-side, <b>When</b> jeg indtaster mit brugernavn og kodeord korrekt, <b>Then</b> bliver jeg logget ind og kan se oversigten over nye og eksisterende ordrer.	Godkendt
4.	Som en sælger kan jeg se alle unassigned forespørgsler, så jeg kan vælge at tildele mig selv en ordre.	<b>Given</b> der er nye forespørgsler i systemet, <b>When</b> jeg navigerer til oversigten over forespørgsler, <b>Then</b> kan jeg se en liste over alle forespørgsler markeret som "unassigned." <b>And</b> jeg kan se en liste over alle de forespørgsler som jeg arbejder på.	Semi - Godkendt når man unassigner sig, så står ordren stadigvæk som tildelt.



5.	Som en sælger kan jeg tildele mig selv en ordre, så jeg kan begynde at behandle den.	<b>Given</b> jeg har fundet en "unassigned" forespørgsel, <b>When</b> jeg vælger at tildele forespørgslen til mig selv, <b>Then</b> bliver forespørgslen markeret som "assigned" til mig i systemet.	Godkendt
6.	Som en sælger kan jeg tilgå andres tildelte ordrer (kun til visning) og ikke-tildelte ordrer i en oversigt, så jeg kan få overblik over åbne forespørgsler og tage over, i tilfælde af sygdom eller fravær.	<b>Given</b> en forespørgsel er tildelt en anden sælger, <b>When</b> jeg åbner forespørgslen i visningstilstand, <b>Then</b> kan jeg se alle detaljer uden at kunne ændre dem.	Godkendt
7.	Som en sælger kan jeg tilgå en forespørgsel og få genereret et overslag, indkøbspris, salgspris og avancen i procent, så jeg har et udgangspunkt for et tilbud	<b>Given</b> forespørgslen er oprettet i systemet, <b>When</b> jeg har tilgået en kundes forespørgsel, <b>Then</b> beregner systemet en indkøbspris, foreslået salgspris, og avancen i procent.	Godkendt  Med forbehold i, at man ikke kan ændre i avanceprocenten, denne er fast sat til 1,5 procent.
8.	Som en sælger får jeg genereret en tegning af carporten, så kunden kan se et visuelt design.	<b>Given</b> jeg har tilgået en kundes forespørgsel, <b>When</b> ordren er blevet oprettet i systemet, <b>Then</b> opretter systemet en visualisering af carporten.	Godkendt
9.	Som en sælger kan jeg generere en stykke til carporten, så jeg ved, hvilke materialer der skal bruges.	<b>Given</b> jeg har tilgået en kundes forespørgsel, <b>When</b> ordren er blevet oprettet i systemet, <b>Then</b> opretter systemet en liste over de nødvendige materialer.	Godkendt
10.	Som en sælger kan jeg generere en vejledning, så kunden kan forstå, hvordan carporten skal bygges.	<b>Given</b> jeg har tilgået en kundes forespørgsel, <b>When</b> ordren er blevet oprettet i systemet, <b>Then</b> opretter systemet en trinvis guide til konstruktionen af carporten.	Ikke godkendt, der bliver ikke sendt en vejledning med, dog vedhæfter sælgeren denne.
11.	Som salgsschef kan jeg tildele en ordre til en sælger (f.eks. i tilfælde af sygdom hvor en anden sælger skal tage over), så en anden kan overtage sagsbehandlingen af en ordre.	<b>Given</b> en ordre er allerede oprettet og tildelt en sælger, <b>When</b> jeg som salgsschef vælger at redigere ordretildelingen, <b>Then</b> kan jeg vælge en anden sælger fra listen og gemme ændringen, så den nye sælger får adgang til ordren	Godkendt. Alle ordre kan tages fra alle sælgere, selv de i forvejen tildelte ordre.
12.	Som en sælger kan jeg ændre avanceprocenten i forespørgslen, så jeg	<b>Given</b> jeg har tilgået en kundes forespørgsel, <b>When</b> jeg vælger at redigere	Ikke godkendt.

	kan tilpasse avancen i procenter efter hvem kunden er.	avanceprocenten, <b>Then</b> udregner beregningsmodellen en ny salgspris.	Denne del er ikke blevet implementeret, da den er en del af sprint 2.
13	Som salgschef kan jeg synkronisere priserne i System A med priserne i System B (dvs. hente priser fra system B og automatisk opdatere dem i vores system A), så den tilbudspris programmet udregner er korrekt.	<b>Given</b> System A og System B er integreret, <b>When</b> jeg vælger at synkronisere priserne i systemet, <b>Then</b> henter systemet de nyeste priser fra System B og opdaterer de tilsvarende materialer og produkter i System A.	Ikke godkendt. Denne del er ikke blevet implementeret, da den er en del af sprint 3.
14	Som en sælger kan jeg godkende en forespørgsel, så den bliver omdannet til et tilbud der kan redigeres.	<b>Given</b> jeg har kontrolleret en forespørgsel, <b>When</b> jeg vælger at godkende forespørgslen, <b>Then</b> bliver der oprettet et redigerbart tilbud til kunden	Ikke godkendt, man kan ændre i status, men dette påvirker ikke andre dele af programmet og derfor bliver der ikke oprettet et redigerbart tilbud til kunden.
15	Som en sælger kan jeg redigere et tilbud, så den passer til kundens ønsker.	<b>Given</b> kunden/sælgeren ønsker ændringer i tilbuddet, <b>When</b> sælgeren trykker på rediger tilbuddet, <b>Then</b> vises en ny formular.	Godkendt. Med forbehold, at tegningen ikke skal stemme overens med det faktiske carport/skur. (Skuret skal være ligeså bedt som carporten, for at tegningen passer.)
16	Som en sælger kan jeg sende et tilbud til kunden via systemet eller telefonisk, så kunden kan tage stilling til det.	<b>Given</b> jeg har genereret et tilbud, <b>When</b> jeg vælger at sende det via systemet eller kontakter kunden, <b>Then</b> modtager kunden tilbuddet og kan acceptere eller afvise det.	Ikke godkendt, denne user story er ikke implementeret, da den er en del af sprint 2.
17	Som en kunde kan jeg acceptere eller afvise et tilbud, så jeg kan vælge, om jeg vil gå videre med ordren.	<b>Given</b> jeg har modtaget en e-mail med et link og et tilbud fra sælgeren, <b>When</b> jeg trykker på det link og vælger at acceptere eller afvise det, <b>Then</b> opdateres status for forespørgslen i systemet.	Ikke godkendt, Kan ikke godkendes, da vi ikke sender et link til kunden, hvor kunden kan godkende tilbuddet. Men det er muligt med en test ordre.
18	Som en sælger kan jeg tage kontakt til kunden, hvis de afviser et tilbud, så jeg kan forstå og forsøge at finde en løsning.	<b>Given</b> kunden har afvist et tilbud, <b>When</b> jeg vælger at kontakte kunden via mailsystemet eller telefonisk, <b>Then</b> kan jeg sende en besked	Ikke godkendt, Da vi kun kan godkende/afvise en test ordre, og ikke har implementeret mailsystemet.

		eller ringe til kunden for at diskutere videre muligheder.	
19	Som en sælger kan jeg rette ordrebekræftelsen, hvis der er fejl eller mangler.	<b>Given</b> jeg har identificeret fejl i ordrebekræftelsen, <b>When</b> jeg vælger at redigere dokumentet, <b>Then</b> kan jeg rette og gemme de opdaterede oplysninger i systemet.	Godkendt
20	Som en sælger kan jeg rette stykliste, hvis der er fejl eller mangler.	<b>Given</b> der er fejl eller uoverensstemmelser i styklisten, <b>When</b> jeg vælger at redigere listen, <b>Then</b> kan jeg opdatere og gemme ændringerne i systemet.	Godkendt
21	Som en sælger kan jeg give grønt lys til at sende ordrebekræftelse og faktura til kunden, hvis alt stemmer overens.	<b>Given</b> jeg har gennemgået ordrebekræftelsen og fakturaen, <b>When</b> jeg bekræfter, at alt er korrekt, <b>Then</b> bliver ordrebekræftelsen og fakturaen sendt til kunden via systemet.	Ikke godkendt, denne user story er ikke implementeret, da den er en del af sprint 2.
22	Som en sælger kan jeg opdatere betalingsstatus, så ordren kan blive færdigbehandlet, når det er blevet betalt.	<b>Given</b> system b har registreret betaling fra kunden, <b>When</b> jeg trykker på opdater betaling knappen, <b>Then</b> opdaterer systemet ordren til "betalt," og processen fortsætter.	Godkendt
23	Som salgschef kan jeg tilføje og redigere materialer, så programmet altid er opdateret med de korrekte tilgængelige materialer og beskrivelser.	<b>Given</b> jeg har adgang til materialedatabasen i systemet, <b>When</b> jeg vælger at tilføje eller redigere et materiale, <b>Then</b> kan jeg indtaste eller ændre navnet, beskrivelsen og prisen på materialet, og ændringerne gemmes i systemet.	Ikke godkendt, denne user story er ikke implementeret, da den er en del af sprint 3.
24	Som salgschef kan jeg bestemme om mine sælgere skal have en notifikation via e-mail, når der er nye ordre i systemet, så jeg kan tilpasse mængden af e-mail systemet sender.	<b>Given</b> jeg har adgang til indstillinger for notifikationer, <b>When</b> jeg vælger at aktivere eller deaktivere e-mail-notifikationer for nye ordrer, <b>Then</b> opdaterer systemet indstillingerne, og sælgerne modtager eller stopper med at modtage disse notifikationer.	Ikke godkendt, denne user story er ikke implementeret, da den er en del af sprint 3.

25	Som en salgschef kan jeg vælge hvad vores standard avance skal være i systemet, så sælgerne har et godt udgangspunkt til at lave tilbud fra.	<b>Given</b> jeg har adgang til prisindstillingerne i systemet, <b>When</b> jeg indstiller en standard avanceprocent, <b>Then</b> gemmes denne avance i systemet og anvendes som standard, når sælgerne udarbejder nye tilbud.	Ikke godkendt, denne user story er ikke implementeret, da den er en del af sprint 3.
----	--	--	--

Tabel 14.1 User Acceptance tests

## Proces

### Introduktion

I ugen op til kickoff på projektet havde vi forretningsforståelse med Kim som gav os en masse værktøjer til at analysere produkt, virksomhed, udfordringer, samt kommunikative værktøjer til at vide hvem vi kontakter og hvordan vi kommunikerer.

**Uge 1** var vores første arbejdsuge, hvor vi fik lavet en masse standarder for hvordan vi ville arbejde, lavede analytisk arbejde samt fik sat vores Kanban og IntelliJ op så vi havde et overblik og en ensretning i vores arbejdsgang.

**Uge 2** var sprint 1 som indeholdt de mest essentielle samt tunge dele af projektet

**Uge 3** var sprint 2 som muligvis kunne blive overlappet med sprint 1, hvor vi ville færdig implementere elementerne i sprint 1, for så at påbegynde sprint 2. Dette sprint var defineret som elementer der var vigtige for programmet, men ikke essentielle for det færdige produkt.

**Uge 4**, som var vores sidste uge med at kode, var vores sprint 3 som indeholdt features som ikke var nødvendige for programmet, men som kunne fuldende det som et færdigt program.

Den sidste uge havde vi valgt at sætte af til at skrive rapport.

Vi definerer derfor 5 faser i vores projektperiode; *Før vi begyndte, Forberedelsen, Kodning, Code Freeze og Rapport*

Nedenfor vil de 5 fase blive beskrevet først faktuel og derefter en refleksion over projektet som helhed.

## Arbejdsprocessen faktuel

### *Før vi begyndte – Uge 0*

Vores proces startede da vi arbejdede med forretningsanalytiske værktøjer i form af, Interessentanalyse, Risiko analyse, Aktivitetsdiagram og User stories. Især de to sidstnævnte værktøjer fandt vi både vigtige samt essentielle for hvordan vi ville arbejde i gruppen. Aktivitetsdiagrammet, hvor vi lavede to instanser af disse, et AS-IS diagram og et TO-BE diagram – og 5 vigtige user stories. Aktivitetsdiagrammet kunne vi bruge til at fremme vores forståelse for hvordan det nuværende system hos FOG fungerede og hvordan vi kunne forestille os at optimere det. User stories til at transskribere hvad Martin sagde i videoen til relevante opgaver som vi kunne programmere.

Risiko analyse samt interessent analyse fandt vi mindre vigtige, hvilket skulle vise sig senere at blive mere eller mindre ironisk. Vi lavede dem dog selvfølgelig, fejede pænt interessentanalysen væk, samt at vi vurderede at risiko analyse højst sandsynligt ikke ville blive relevant. Og hvis den gjorde havde vi, i vores optik, lavet en god plan for hvad vi ville gøre i diverse scenarier. Ydermere blev der udarbejdet en gruppekontrakt som blev mundtlig aftalt fra samtlige medlemmer i gruppen. Der blev vedtaget at vi ville uddelegere nogle roller i gruppen, i tilfælde af uenigheder. Dette blev ikke gjort for at fremme nogens position, men udelukkende for at have en beslutningskyndig i tilfælde af vi sad fast, og skulle videre. Disse roller var, *Project lead*, *Tech lead* og *Rapport lead*. Disse roller blev udfyldt løbende og godkendt af samtlige medlemmer i gruppen. Der blev oprettet et Kanban board via *GithubProjects* som ville fungere som vores primære strukturelle værktøj. Udover dette brugte vi også discord til diverse planlægning og deling af relevant information i gruppen

Sidste dag i denne del af processen brugte vi på at definere samt lave noget struktur på projektet. Vi valgte at arbejde SCRUM baseret og inddelte derfor projektet op i tre sprints. Vi inddelte kun kodningsdelen af projektet op i sprints. Vi aftalte derfor også at have 2 daglige SCRUMs, som blev holdt som *stand up meetings*, et om morgenen og et inden vi tog hjem.

På trods af vi havde lavet struktur på projektet, så skulle det vise sig at denne ville ændre sig mange gange i løbet af projektet.

### *Forberedelsen – Uge 1*

Projektets officielt første uge havde vi på forhånd planlagt skulle gå udelukkende med analytisk arbejde for at have en grundig forståelse for projektet, produktet og arbejdsmæssig gang inden vi begyndte at

kode. Nogen i gruppen havde før arbejdet sammen og haft gode erfaringer med en lidt længere, men mere grundig forberedelse - målet var ultimativt at med bedre forståelse og forberedelse, så ville vi lave en mere korrekt og sammenhængende kode, og derfor i sidste ende spare os tid.

Vi fik konfigureret vores IDE'er sådan at vi overholdt de samme kodestandarder, vi satte dem op sådan at vi kunne arbejde på en database i skyen i stedet for lokalt, og påbegyndte derefter arbejdet med at udarbejde samt godkende projektets struktur. Vores plan var at lave en grundig udarbejdelse af vores TO-BE navigationsdiagram, blive enige om og godkende denne for så at lave user stories ud fra dette diagram. Vi valgte at vi hellere ville have mange små og præcise user stories frem for få store, for at kunne danne et bedre overblik over hvilke user stories der skulle tilhøre hvilke sprints. Da vi havde et første udkast til user stories, lavede vi acceptance kriterier til disse. Nu havde vi et første udkast til hvordan vi så vores projekt og produkt skulle være hvorfor vi lavede domæne model og klasse diagram v.1.

Vi deler os op i grupper af 2 og hvor én gruppe laver mock-ups den anden går i dybden med at færdiggøre user stories + acceptance kriterier. Der bliver i denne del af processen vendt en del tanker i forhold til hvordan flow i systemet skal være samt en masse design spørgsmål.

Vi havde arrangeret med en del af de andre grupper at hver gruppe holder 30 minutters oplæg omkring deres nuværende modeller for at brainstorme ideer samt at sammenligne flow og prioriteter. Da opgaven har været meget åben til fortolkning, fandt vi sikkerhed i at kunne se om folk var på vej i samme retning. Vi blev enige om, at inden vi begyndte at kode skulle de ovennævnte roller være fastlagte. Tech lead blev her defineret, sådan at denne kunne begynde at planlægge hvilken vej koden skulle gå. Dette satte gang i et omfattende møde omkring hvordan vores sprints skulle deles op. Vi skrev alle vores user stories op på post its, og så gennemgik vi dem ellers på en tavle og fik dem inddelt i vores tre sprints blev defineret som;

**Sprint 1** var user stories der omhandlede funktioner i programmet som var essentielle for hvad kunden havde bedt om.

**Sprint 2** var user stories som var vigtige, men som vi mente kunne være features som blev klaret efter deadline

**Sprint 3** var user stories som var "nice to have" som vi ville foreslå kunden.

Sidste dag i denne fase begyndte vi med et vejledermøde, som skulle simulere et møde med kunden. Vi havde her aftalt at fremlægge vores struktur, arbejdsgang samt user stories. Vi havde et godt møde som dog skabte en vis bekymring fra kunden om hvorvidt vi havde scopet for stort. Vi manede bekymringen til jorden ved at forsikre om at vores omfattende analyse, samt mange user stories var for at vi selv kunne identificere hvad der var essentielt, vigtigt og ikke vigtigt for at kunne levere et produkt til vores deadline. Dette blev accepteret og både udviklingsteam samt kunde gik tilfredse derfra.

Det sidste vi gjorde inden vi gik på weekend, var at tale logik om vores beregningsmotor, der blev pseudo kodet og der blev samlet op om eventuelle løse ender som omhandlede analysedelen af projektet. Vi gik på weekend med oprejt pande og følte os klar til at bygge projektet.

### *Kodning – Uge 2, 3 og 4*

Første dag startede med en revurdering af vores gruppekontrakt. Der blev nævnt at folks mødetidspunkter var for løse, og hvordan det kunne være. Vi havde en god dialog, og kom frem til at vi rykkede vores daglige SCRUM en halv time frem, da det passede bedre størstedelen af gruppen. Efter revurdering af kontrakt, valgte vi de resterende roller som vi havde defineret tidligere. Franck blev valgt som Project lead og Mathias blev valgt som Rapport lead. André var tidligere valgt som Tech lead. Vi blev ydermere enige om, at selvom man havde en "titel" så er man ikke højere rangeret i beslutningstagninger eller på anden måde "mere værd" end de resterende medlemmer. Det er udelukkende for, at der kommer noget struktur samt et sikkerhedsnet hvis skulle vi sidde fast. Vi havde også en kort refleksion af vores analyse uge, samt om vi burde starte på rapporten allerede nu. Der blev argumenteret for, at det ville være smart at skrive den del af rapporten som omhandler analysearbejdet, nu hvor det stadig var friskt. Vi var alle enige i, at det ville være en god idé, men på trods af det vurderede vi, at selve kode delen var vigtigere nu, da opgaven var omfattende. Arbejdsgangen i størstedelen af kode forløbet er enten pair-programming eller enkeltvis på sin user story. Tech lead arbejde med at lave tasks fyldestgørende, lave sekvensdiagrammer og få det ind i kanban så det er nemt og overskueligt for os andre at gå i gang med en user story. Dette resulterer så i, at de møder vi har omhandler mere hvor vi hver især er henne i vores individuelle user stories, og ikke så meget omkring en grundig gennemgang af hvordan vi ser flow og scope for projektet. Vi har alle en idé, men da vi mister André af personlige årsager, finder vi pludselig ud af, at vi måske har lagt lidt for mange æg i én kurv. Vi andre i gruppen bliver derfor nødt til at scope lidt om, og tager beslutninger, som vi tror tech lead har udtænkt det. Dette skaber en del usikkerhed og ustabilitet i arbejdsgangen.

Vi arbejder i uge 1 med User story 1, 4, 6, 18 og 9 hvor vi får færdiggjort user story 4 og 9. User story 9, som er en del af beregningsmotoren, kastede vi os ud i at lave TDD (test driven development), og ved at gøre det fik vi et godt indblik i hvordan vi ville have motoren til at fungere. Ved at lave TDD fandt vi fejl som vi ikke havde tænkt over, og vi var ret begejstrede over at det virkede så godt som det gjorde.

Uge 2 starter med at tech lead er tilbage igen, men vi bliver her ramt af en del sygdom. Det gør det igen meget svært at få alle med på samme bølgelængde, selvom vi fører logbog dagligt, kommunikerer på discord osv. Der arbejdes videre på user stories tilhørende sprint 1 da vi kan se vi er bagud her. På nuværende tidspunkt begynder vi stille og roligt at indse, at de features vi havde talt om vi skulle arbejde på i sprint tre, nok ikke bliver en realitet. Vi tager derfor en del af user stories fra sprint 1 med over i sprint 2. User stories der bliver arbejdet på i dette sprint er primært US 1, 6, 15, 18, 5 og 3. Det er også i uge 2 vi får vores database design op at køre. Vi valgte at have en database kørende gennem en droplet på digital ocean, så vi hele tiden havde en opdateret database at arbejde med. Vi rykker en del her i uge to, hvor vi mere eller mindre har et fuldt dummy projekt kørende, med html som viser os det vi gerne vil have den viser. Det giver blod på tanden, og en fornemmelse af at "nu er det jo bare at få det til at snakke sammen". Sidste dag i uge to bliver der lavet code review på det vi har, merget sammen med dev og der bliver aftalt at vi holder code freeze søndag ti dage senere.

Uge 3 starter ud med at vi fandt en design flaw i vores beregner, og den er derfor blevet refaktoreret ud i sin egen klasse. Vi skal til at have de sidste ting bundet sammen og user stories lavet færdig, så vi laver alle i gruppen estimerer på hvor langt vi er fra at være færdige med det vi har gang i, og tech lead træder til hvor der er brug for hjælp. Som ugen skrider frem, kan vi godt se at et code freeze søndag først er for risikabelt, og der bliver derfor besluttet at det er rykket tilbage til fredag i stedet for.

Beslutningsgrundlaget for dette er, at vi på de få dage ikke ville kunne nå det som var planen, og derfor vil vi hellere fokusere på at få projektet samlet, finpudset og deployet, for så at have de ekstra par dage til at skrive rapporten. Vi begynder at samle projektet ved at sidde alle om én skærm, og løse eventuelle merge konflikter imens vi samler branches sammen med dev branchen. Imens vi gør dette og tester programmet, får vi nogle error messages vi ikke er helt vilde med, og beslutter os for at vi vil lave vores egen exception handler, for at få et bedre overblik over hvad der sker hvor, når noget går galt. Det hele går pludselig ret hurtigt og som vi får samlet projektet ender vi ud med at have et produkt, som vi ikke nødvendigvis er helt tilfredse med, men noget vi føler os bekendte at kunne aflevere. Vi bruger sidste dag på at gå produktet igennem, for så at deploy det til skyen.



### *Code Freeze & rapport – uge 5*

Sidste uge inden vi skal aflevere har vi lagt koden til side for at fokusere på rapporten. Vi har sat nogle skrivestandarder op, så vi ikke skal rette dumme fejl. Vi lavede et nyt kanban board til rapporten, hvor vi satte samtlige afsnit ind, og uddelegerede derefter opgaver ud fra dette, så vi ikke kom til at lave dobbelt arbejde. Vi aftalte at skrive over weekenden + mandag, og holder første rapport review tirsdag morgen, hvor man kan stille spørgsmål, vise hvor langt man er og spørge om hjælp hvis nødvendigt. Så har vi hele tirsdag og det meste af onsdagen til at skrive, og så holder vi et sidste review onsdag morgen, så man kan nå at rette de sidste ting i løbet af onsdagen, for så at stoppe med at skrive onsdag kl. 16. Herefter skulle vi rette igennem, og nærlæse rapporten for fejl mangler, så vi ville være klar til at aflevere torsdag kl. 10.

### *Arbejdsprocessen reflekteret*

I dette afsnit vil vi reflektere over projektets gang, og vende store som små refleksioner. En refleksion oven på et projekt kan ofte være overvældende både for læser og for gruppen der skriver det. Derfor har vi valgt at dele det op i nogle mindre afsnit, for at danne et bedre overblik.

### *Sygdom/fravær*

Inden vi startede projektet, havde vi en uge med forretningsanalyse, hvor vi blev bedt om at lave en risikoanalyse. Der blev jokede lidt på kryds og tværs om hvor meget sølvpapirshat man skulle have på, men vi lavede den ud fra hvad vi synes var realistisk. Vores største risiko havde vi vurderet til at være sygdom i gruppen, hvilket vi også rystede lidt på hovedet ad. Det skulle vise sig at være, hvis ikke lige så alvorligt som vi havde vurderet, så mere alvorligt. Vi havde slækket lidt på hvad vi ville gøre i tilfælde af sygdom, hvor løsningen for os var at uddelegere. Dette skulle vi retrospektivt have defineret meget bedre, for vi havde sammenlagt 2-3 dage over hele projektet, hvor samtlige medlemmer var til rådighed i gruppen. Sygdom viste sig at være roden til rigtig meget af det som gav os mange udfordringer, under hele forløbet. Selvsagt er sygdom noget der kan sætte en kæp i hjulet for ethvert projekt, da det er utroligt svært at lave et præcist estimat på, hvor slem sygdom kan blive for en gruppe - især fordi dette er en helt naturlig og meget virkelighedsnær problemstilling. Alle bliver jo syge før eller siden. Men i dét omfang vi oplevede det, eksponerede dette mange andre ting, hvor vi pludselig så hvor sårbare vi var, ved ikke at have lavet gode nok planer skulle uheldet være ude. Kunne vi have gjort noget for at

mindske den risiko der er ved sygdom i gruppen? Ja, helt sikkert! Der er mange værktøjer, og et eksempel kunne være at arbejde efter en *T-formet* model, hvor alle i gruppen ved noget om det hele, men man har stadig folk som er mere specialiseret indenfor deres respektive område.

### *Roller i gruppen*

Meningen med at identificere nogle roller i gruppen, som havde diverse ansvarsområder, virkede rigtig godt, når alle var til stede. Men som nævnt ovenfor havde vi rigtig meget sygdom/fravær i gruppen, hvilket ultimativt eksponerede os for, at have for mange æg i samme kurv. Der blev til tider stolet for meget på, at tech lead var ansvarlig for udarbejdelse af tasks på kanban boardet, samt overblikket for koden. Som sagt fungerede det rigtig godt når vi var til stede, men ligeledes skabte det udfordringer når, især, tech lead var væk. Her kunne vi have haft gavn af at identificere arbejdsgangen som *T-formet* og derfor være bedre forberedt på sygdom/fravær. Så selvom vi blev "tvunget" ud i at arbejde t-formet, så havde vi ikke talt om det inden. Dette kunne muligvis have bidraget til en bedre og mere gennemtænkt overordnet forståelse for projektet samt dets gang.

Rollen som project lead, bestod primært af at styre det daglige scrum møde, føre logbog og skabe lidt arbejdsstruktur i gruppen. Da det at arbejde scrum baseret er nyt for os, er der helt klart nogle ting som kunne være forbedret. Det er rigtig smart at bruge kanban og inddele sit arbejde i sprints – men dette kan også skabe frustration, hvis ikke man estimerer godt og præcist hvad og hvor længe et sprint er. I vores tilfælde, synes vi konstant bagud og eftersom vi "bare" rykkede vores sprints rundt som det passede os, så slækkede vi automatisk på strukturen.

### *Kommunikation*

Kommunikation er en svær disciplin at mestre, og er et essentielt punkt i enhver gruppe. Vores tilfælde er ingen undtagelse. Vi har igennem gruppekontrakten aftalt, at det er vigtigt at kommunikere ud, lige meget hvor småt det end måtte være. For lidt kommunikation avler mere kommunikation. Vi har holdt flere daglige møder i analysefasen, og ét dagligt møde under kodningsfasen. Vi har diverse tråde på discord dedikeret til forskellige former for kommunikation. Vi har kanaler dedikeret til fravær, logbog, user stories og til generelle ting. Vi har lavet Code of Conduct, sådan at der er kommunikeret et regelsæt ud og derfor kan kommunikere passivt at kunne fordybe sig i sit arbejde. På trods af alle disse tiltag og aftaler vi har lavet indbyrdes, så er kommunikation stadig svært. Vi har haft tilfælde hvor folk har lavet det samme fordi man ikke har skrevet sig på kanban. Vi har været nødsaget til at revurdere vores

mødetidspunkter, fordi disse ikke blev overholdt. Alt sammen er bekræftende i at det er ekstremt vigtigt at blive enige om nogle basis kommunikative standarder, som alle i gruppen kan overholde. Måske der endda er nogen som får særlige forhold, fordi de ikke passer ind i den ramme, som de andre kan gøre. Vi talte om til et møde, om der overhovedet skulle være et mødetidspunkt, da nogen foretrak at arbejde fra kl. 07, andre fra kl. 10.

Det blev nævnt til et af vores vejleder møder, at vi godt nok havde mange møder. Det tog vi til os, og prøvede at minimere møderne under kodeforløbet til maks. 1 om dagen. Jeg tror vi har lært, at det måske er vigtigere at holde et mere omfattende møde, når vi starter på noget nyt, for så at holde små daily scrums. Hvis ikke der er en skarp dagsorden, kan møderne hurtigt løbe af sporet, hvilket helt sikkert er noget vi kan tage med os videre.

### *Arbejdsgang*

Når vi så taler arbejdsgang, så er der en stærk sammenhæng mellem kommunikation. I den første uge arbejdede vi meget kollektivt. Dette bevidst, selvom det blev en anelse omfattende til tider. Men vi havde identificeret analysefasen som ekstrem vigtig, og der hvor vi virkelig kunne blive enige om både design og flow i projektet.

Som vi gik over i kode fasen, blev alting uddelegeret i user stories og tasks, og arbejdet blev derfor noget mere individuelt. Der var til tider folk arbejde i par, men størstedelen af tiden sad man med sin egen ting. Når man deler op i at arbejde individuelt er det virkelig vigtigt at afstemme hvordan man arbejder og hvordan man spørger om hjælp. Her er det også vigtigt at vedkommende som er valgt som project lead er opmærksom, da det at spørge om hjælp kan være ufatteligt svært. I vores tilfælde oplevede vi nogen medlemmer i gruppen fløj igennem tasks, og andre sad noget længere tid med deres. Set tilbage skulle der have været en bedre kommunikation til vores daily scrum, og hvor langt man var. I værtfald i mere præcise omgang end vi gjorde. Så kunne vi have identificeret folk som sad fast, hurtigere og mere effektivt end vi gjorde.

### *User stories, tasks og aktivitetsdiagram*

Vores produkt og projekt startede ud med et aktivitetsdiagram som byggesten for det hele. Vores user stories blev lavet ud fra denne, og tasks lavet ud fra user stories. En proces som er helt essentiel, men meget omfattende. Vi talte om at det var mere overskueligt at arbejde ud fra aktivitetsdiagrammet end user stories, især nu hvor vi selv skal lave dem ud fra en video. Fremadrettet ville vi have mere gavn af at have bedre og mere fyldestgørende tasks at arbejde ud fra. Dette kombineret med at have kortere og

mere præcise sprints ville gøre en stor forskel i produktivitet og succes oplevelser. Hvor i vores tilfælde følte vi ofte at vi haltede bagud.

### *Code of Conduct*

Nogle i gruppen har arbejdet sammen siden dag et, hvor andre er hoppet til i dette projekt. Dem der havde arbejdet sammen før, havde tidligere en erfaring omkring hvor vigtig en god fælles forståelse for ens code of conduct er. Så vi brugte en del tid på at udarbejde denne, samt at sætte vores IDE'er op, sådan at vi koder på samme måde, med samme opsætning. Vi har også, en god forståelse for hvordan vi bruger GitHub, og har fra start haft en god standard for dette.

Vi har en Main branch som fungerer som vores fail safe. På et givent tidspunkt i processen hvor produktet når en vis kvalitet, fungerer vores main branch som "Sidst velfungerende produkt som virker". Så har vi en dev branch, som vi laver feature branches ud fra, for at minimere risikoen for at noget går virkelig galt. Som tiden blev mere presset, begyndte vi at slække lidt på vores standard om at bruge pull requests fra vores feature branches til dev branchen, da tech lead da kunne bruge alt sin tid på code review. Vi overgik derfor til at vi selv stod for at merge vores feature branches in i dev, og lave dertil hørende review. Dette gik overraskende godt, og vi oplevede ikke på noget tidspunkt kritiske merge konflikter. Det kunne dog have været rart at kunne overholde de standarder som vi aftalte fra start, men overordnet set har dette fungeret rigtig godt.

### *Åben sparring med klassen*

Vi har arrangeret sparring på kryds og tværs af de udviklingsteams der har været i klassen, for at sammenligne og være nysgerrige på hinandens arbejdsgange. Dette fungerer vildt godt, og er noget vi har taget med os fra tidligere erfaringer og tager med os videre.

### *Vejledning*

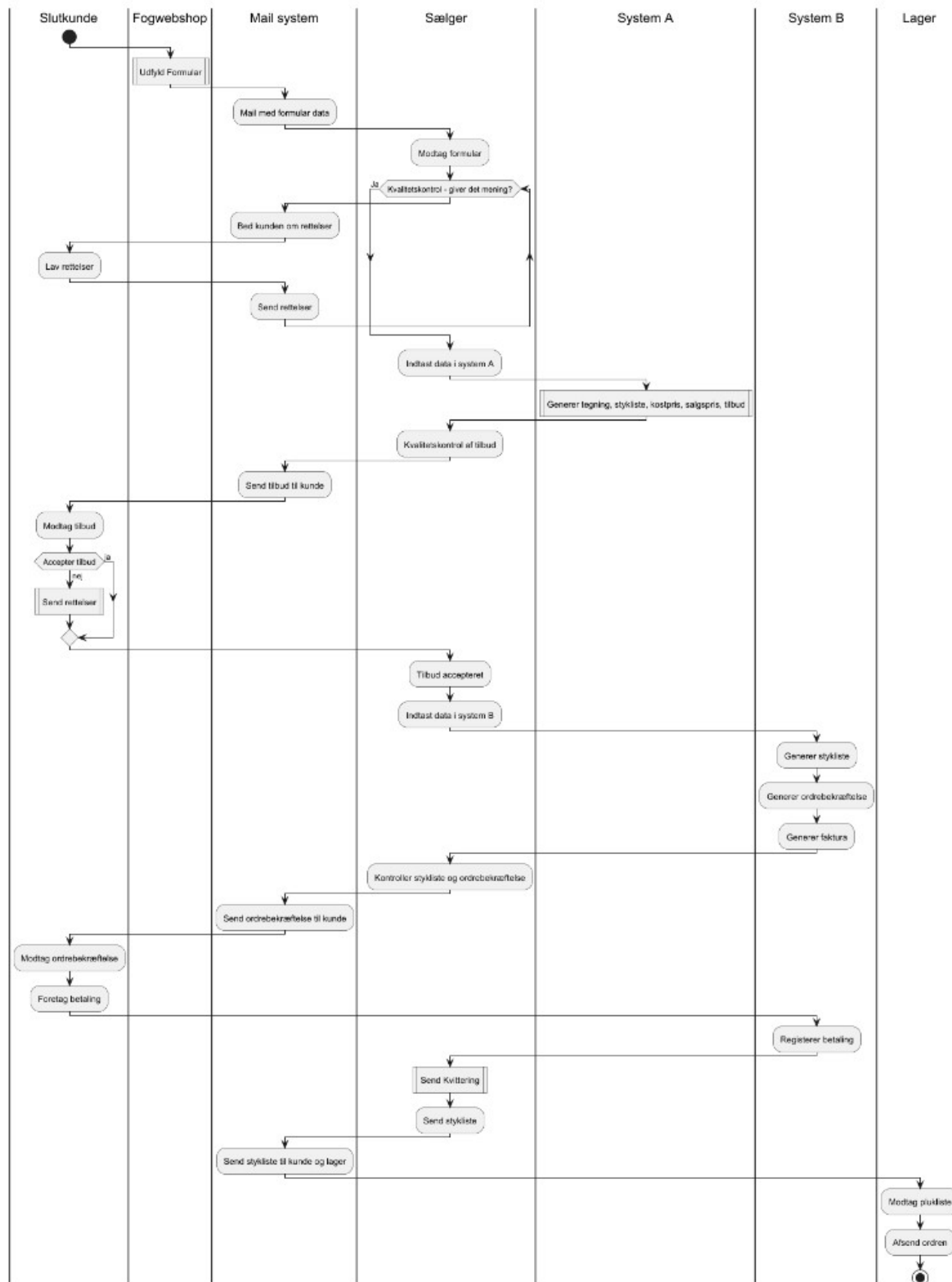
Vi skal være bedre til at bruge vores vejleder, og have en struktur for vores møde. Vores vejleder møder blev derfor mere en status på hvordan det går, og en snak om dette, istedet for en konstruktiv sparring med elementer fra vores produkt. Dette hænger sammen med kommunikationsdelen, og igen er det tydeligt at denne del ofte kan være bedre.

### *Opsummering*

Overordnet set, så har vores projekt været defineret som kaotisk. Når man rammer ind i de udfordringer, i det omfang som vi har, så er det en hård måde at lære på - mens nøgleordet her er lære. For vi har lært rigtig meget i vores konstante påmindelse omkring kommunikation, planlægning og at nogle gange er det bare sådan det er. Man kan ikke forberede sig på alt, men man kan prøve. Og her ville det for os have været smart at definere en arbejdsform, som for eksempel *T-modellen* sådan at vi havde været bedre dækket ind som gruppe, når et individ var sat ud af spil. Vi er nået i mål med et produkt, som vi kan se "potentiale" i, og vores estimering af hvad vi ville nå, har ramt en anelse ved siden af. Vi burde måske have været mere realistiske og estimeret et minimums krav som succes kriterie, istedet for en mere idealistisk tilgang, hvor vi sætter målet højt, for så ultimativt at stå med en følelse af skuffelse over ikke at nå i mål. Vi er ihvertfald i gruppen enige om, at vi stadig er i en proces hvor man lige skal lære at produktet ikke er ét og alt, hvilket kan være nemt at forstå, men svært at acceptere.

---

## Bilag

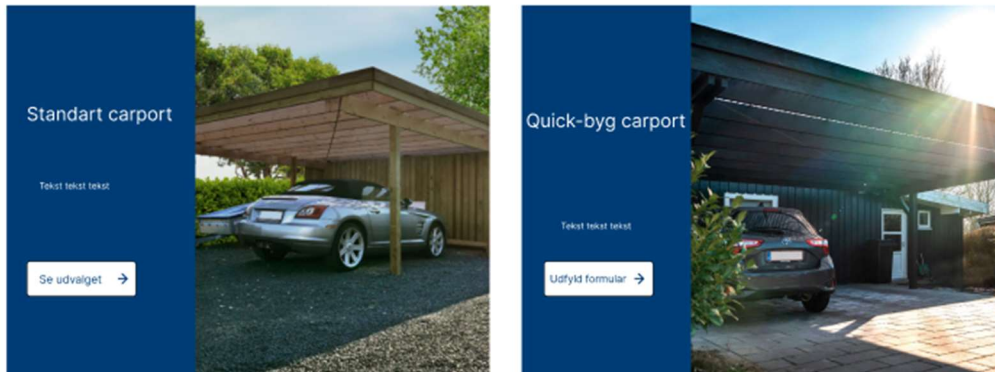


Figur 6.1: Aktivitetsdiagram AS-IS udarbejdet ud fra video fra kunden

# Mockups

Forside

Fog



Oplysninger, CVR.

Figur 9.1.1: Mockup Forside

## Bestil Quick-Byg tilbud - carport med fladt tag

Med et specialudviklet computerprogram kan vi lynhurtigt beregne prisen og udskrive en skitsetegning på en carport indenfor vores standardprogram.

Tilbud og skitsetegning fremsendes med post hurtigst muligt.

Ved bestilling medfølger standardbyggevejledning.

Udfyld formularen omhyggeligt og klik på "Bestil tilbud".

Felter markeret med \* SKAL udfyldes!

### Carport med fladt tag

Udfyld nedenstående omhyggeligt og tryk "Bestil tilbud".

Så vender vi hurtigst muligt tilbage med et tilbud til dig.

Carport bredde\*

Carport længde\*

Carport trapeztag\*

Redskabsrum bredde\*

NB! Der skal beregnes 15 cm tagudhang på hver side af redskabsrummet\*

Redskabsrum længde\*

Redskabsrum længde\*

Evt. bemærkning / særlige ønsker

Kontaktoplysninger:

Navn\*

Adresse\*

Postnummer\*

By\*

Telefon\*

E-mail\*

Kontakt samtykke

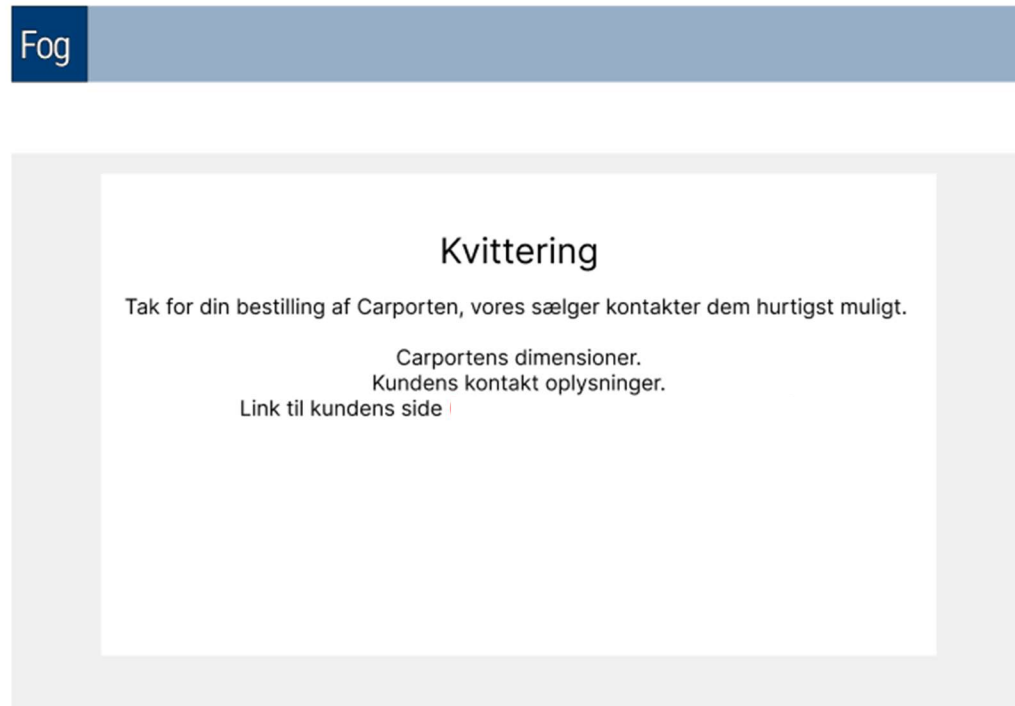
☒ Fog må benytte de afgivne oplysninger til at kontakte mig i forbindelse med tilbud på QuickByg carport\*

Bestil tilbud

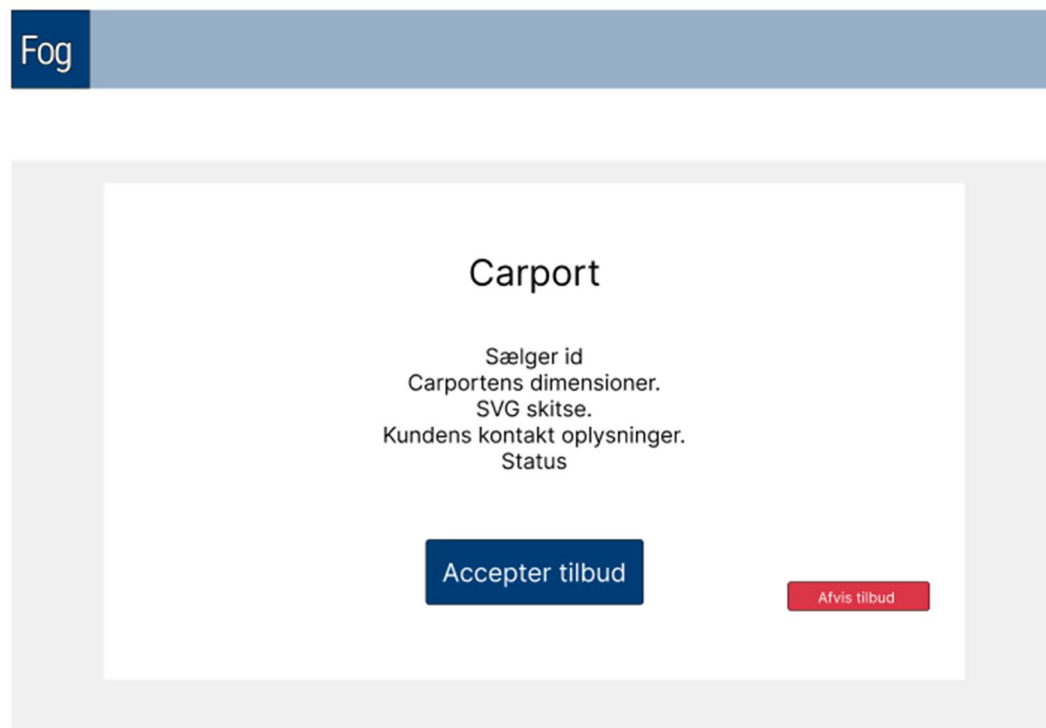
Oplysninger, CVR.

Figur 9.1.2: Mockup bestillingsformular





Figur 9.1.3: Mockup kvitteringsside



Figur 9.1.4: Mockup Kundens acceptering af tilbud side

## Login

Sælger@mail.com

Password

Login

Figur 9.1.5: Mockup sælger login

## Mine ordre:

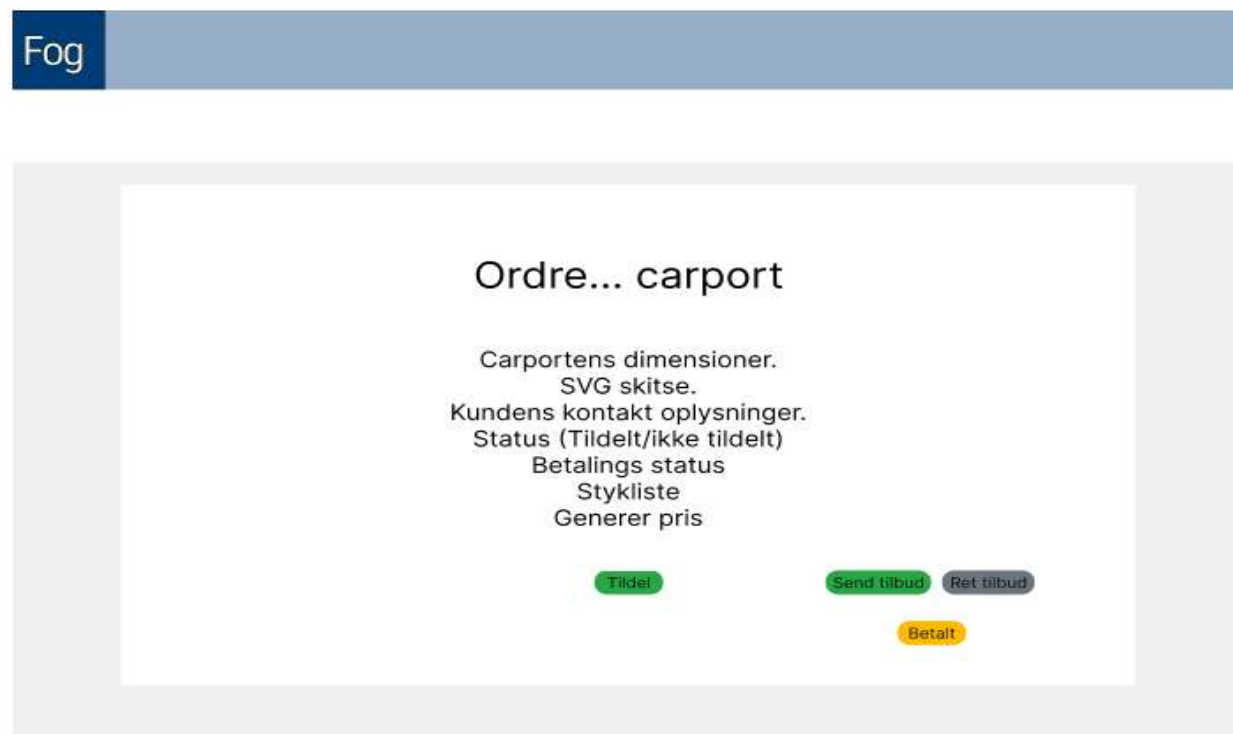
Eksempel 1 Ordre	Ret
Eksempel 2 Ordre	Ret
Eksempel 3 Ordre	Ret

Figur 9.1.6: Mockup sælger ordre side



Figur 9.1.7: Mockup sælger alle ordre side

Ordreoversigt:



Figur 9.1.8: Mockup ordredetaljer

---

<https://github.com/dat2Cph/svg>