



# Rapport for Fog Carport - Gruppe E

Dato 19.12.2024

## Deltagere

### Navn

Casper Alexander Gervig

Christoffer Aaby Leisted

Idris Azmi Ata Isci

Rikke Fruelund Walther Mariussen

### Github

Gervig

Crispy212

Mridrisisci

RikkeMariussen

### Email

[cph-cg201@cphbusiness.dk](mailto:cph-cg201@cphbusiness.dk)

[cph-cl446@cphbusiness.dk](mailto:cph-cl446@cphbusiness.dk)

[cph-ii38@cphbusiness.dk](mailto:cph-ii38@cphbusiness.dk)

[cph-rm225@cphbusiness.dk](mailto:cph-rm225@cphbusiness.dk)

# Indholdsfortegnelse

<b>Links .....</b>	<b>1</b>
<b>Indledning.....</b>	<b>2</b>
<b>Baggrund.....</b>	<b>2</b>
<b>Forretningsforståelse .....</b>	<b>2</b>
Interessentanalyse .....	3
Risikoanalyse .....	6
<b>Teknologivalg.....</b>	<b>7</b>
<b>Krav.....</b>	<b>8</b>
<b>User Stories.....</b>	<b>8</b>
<b>Mockups .....</b>	<b>11</b>
<b>Arkitektur.....</b>	<b>12</b>
<b>Navigationsdiagram.....</b>	<b>14</b>
<b>Aktivitetsdiagram.....</b>	<b>15</b>
<b>Domænemodel .....</b>	<b>18</b>
<b>Entity Relationship Diagram (ERD).....</b>	<b>19</b>
<b>Klassediagram .....</b>	<b>20</b>
<b>Udvalgte kodeeksempler .....</b>	<b>22</b>
Eksempel på vores beregningsmetoder.....	22
Eksempel på håndtering af emails til slutkunden .....	25
Eksempel på hentning af data fra databasen.....	26
Eksempel på håndtering af data fra frontenden .....	27
Eksempel på håndtering af exceptions og fejl.....	28
Eksempel på håndtering af input validering .....	28
Eksempel på oprettelse af data i databasen .....	29
<b>Tests (kvalitetssikring) .....</b>	<b>29</b>
User acceptance test.....	30
<b>Særlige forhold .....</b>	<b>30</b>
<b>Status på implementation.....</b>	<b>31</b>
<b>Den faktuelle arbejdsproces .....</b>	<b>33</b>
<b>Den reflekteret arbejdsproces.....</b>	<b>35</b>
<b>Bilag.....</b>	<b>39</b>

## Links

Hjemmeside: <https://carport.nordaire.dk/>

Github repository: <https://github.com/mridrisisci/fog-projekt>

Demo video: [https://youtu.be/U24B\\_mwYqeE](https://youtu.be/U24B_mwYqeE)

Link til figma: <https://www.figma.com/design/v373noJfyxBovDUaipKc6V/fog-projekt?node-id=0-1&t=xGTIs3MDFiL8R3PW-1>

User acceptance test af User Story 1: <https://youtu.be/uUOBBRG8sPQ>

User acceptance test af User Story 2 og 7: <https://youtu.be/8tXL2KHI5jE>

User acceptance test af User Story 4 og 10: <https://youtu.be/Fpw09sXUsjs>

Admin login	Sælger login
<a href="mailto:admin@cph.dk">admin@cph.dk</a> Kodeord: 1234	<a href="mailto:sales.person.fog@gmail.com">sales.person.fog@gmail.com</a> Kodeord: 1234

## Indledning

Når man arbejder på en bestemt ting, kan det være både besværligt og frustrerende, hvis man skal benytte flere forskellige stykker software. Denne udfordring står Johannes Fog med, når der skal laves tilbud til deres kunder på carporte med specialmål. De har derfor taget kontakt til os, studerende på andet semester på Datamatikeruddannelsen, og sendt os en video hvor salgschefen for Johannes Fog, Martin, bliver interviewet vedrørende deres nuværende systemer og mangler dertil. Her forklarer han blandt andet at de førhen har brugt penge på et nyt system, som de endte med ikke at bruge, da det ikke kunne samarbejde med deres lagersystem. Endvidere benytter de et adskilt system til at sende e-mails til deres kunder vedrørende tilbud. Dette er ikke optimalt, da det gør at sælgerne skal hoppe frem og tilbage i programmerne.

Denne rapport tager derfor udgangspunkt i deres behov for at samle beregningssystemerne for modtagelse af kundens mål og beregning af styklister samt dækningsgraden. Dertil automatiseres udsendelsen af tilbud til kunden. Den er tilegnet andre studerende på samme niveau af Datamatikeruddannelsen, samt virksomheden Johannes Fog's (herefter Fog) salgschef. Hen over denne projektperiode er vi kommet frem til en potentiel løsning for Fog i form af en hjemmeside som samler beregningssystemerne, styklister og andre funktioner ud fra de udarbejdede krav vi kom frem til i videoen fra Fog.

## Baggrund

Johannes Fog har i forbindelse med salg af carporte med specialmål været nødsaget til at benytte to beregningssystemer til modtagelse af kundens mål, udregning af styklister og dækningsgrad, da det ene system ikke kunne opdateres med de nye materialer og deres mål. Johannes Fog vil derfor gerne have samlet alle funktioner i et system, både for at gøre transaktioner nemmere at håndtere, men også for at skabe en bedre kundeoplevelse. At forbedre kundeoplevelsen ville være muligt, da Fogs sælgere ville have nemmere ved at behandle kundernes bestillinger, og derved forkorte behandlingens ventetid.

## Forretningsforståelse

Når der arbejdes sammen med et firma, er det oftest relevant at udarbejde en interessent- og risikoanalyse.

Interessentanalysen kan bruges til at identificere de relevante nøglepersoner som kan have indflydelse og påvirkning på projektet.

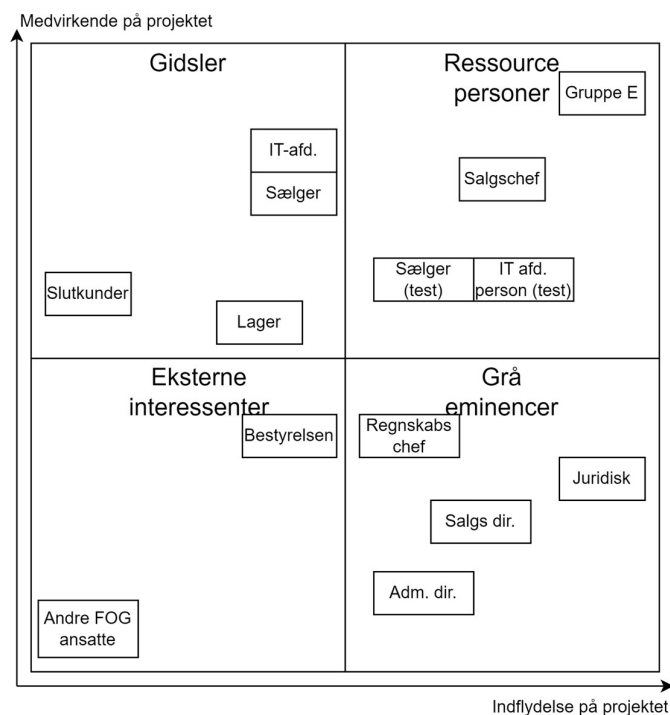
Da Fog havde problemer med det program de tidligere havde fået udviklet i forhold til at være kompatibelt med andre systemer, vil det være relevant at medtage IT-afdelingen, for at sikre

at alle systemer er kompatible med hinanden. Det havde vi dog ikke mulighed for, da dette er et skoleprojekt og vi havde ikke kontakt til de ansatte hos Fog.

Vi har nedenfor udviklet en interessentanalyse, baseret på hvis man som udviklere skulle udvikle et program for Fog, med kontakt til de ansatte.

En risikoanalyse kan bruges til at identificere visse risici og svagheder, der kan opstå ved et projekt. Dette er særligt vigtigt, hvis man ansættes af et andet firma, da det kan være med til at fastsætte rettigheder og rammer for kontrakten der laves mellem parterne.

## Interessentanalyse



Figur 1 - Interessentanalyse

### Gidsler

**Slutkunderne** er nødvendige for salg af carporte, men har ingen indflydelse på selve projektet. Hvorfor de er placeret som et gidsel med næsten ingen indflydelse.

**Lagermedarbejdere** (skrevet som "Lager") er gidsler med mindre medvirken og lidt mere indflydelse på projektet end slutkunderne, da de skal kunne modtage styklisten/pluklisten.

**Sælger** er dernæst også placeret som et gidsel, da den almene sælger ikke har meget indflydelse på projektet men har meget medvirken da de skal benytte sig af programmet når det er færdigt.

Det anbefales at have minimum én sælger med i projektets udvikling som ressourceperson til at teste systemet og konsultere om deres krav af nødvendigheder til programmet.

**IT-medarbejdere** (skrevet som IT-afd.) er til sidst også placeret som et gidsel, da den almene it-medarbejder ikke har meget indflydelse på projektet men har meget medvirken da de skal vedligeholde programmet når det er færdigt, og sørge for at det bliver ved med at køre.

Det anbefales at have minimum én it-medarbejder med i projektets udvikling som ressourceperson til at teste systemet og konsultere om deres teknologiske valg, så man sikrer sig at de har den korrekte software til at benytte, og så projektet bliver udviklet i versioner som de kan benytte.

## Eksterne interessenter

**Andre Fog ansatte** kan have en begrænset interesse for dette, da det handler om deres arbejdsplads, men ikke deres arbejdsgang. I Fog's CSR rapport fra 2022 skriver de at alle oplysninger vedrørende nye procedurer og lignende bliver lagt op på deres interne hjemmeside, så alle ansatte kan læse hvad der sker i virksomheden. Derfor kan det argumenteres at andre ansatte i Fog er eksterne interessenter, da de ikke har indflydelse på projektet, og teknisk set heller ikke har nogen medvirken, men kan stadig følge med i udviklingen, derfor har de en smule medvirken.

**Bestyrelsen for Fog** har en økonomisk interesse i projektet, da dette kan streamline sælgernes arbejdsgang og dermed effektivisere processen. De har lav medvirken, da de ikke har direkte kontakt med salgsafdelingen eller udviklerne.

## Ressourcepersoner

Vi antager i forbindelse med denne rapport at der er en testperson fra salgsteamet samt IT-afdelingen, da dette anbefales af udviklingsteamet således at man har nogle der sidder med programmet og problemstillingen til dagligt.

**Sælger** (skrevet som "Sælger(test)") er med til at udarbejde hvilke krav og behov der er relevante for salgsafdelingen for at dette projekt bliver et som firmaet vil benytte sig af. Det anbefales at de bliver inddraget i processen af flere omgange, evt. som udviklingen sker, men også i starten, så man ved at systemet er brugervenligt, og opfylder deres krav.

**IT-medarbejder** (skrevet som IT afd. Person (test)) er med til at udarbejde hvilke krav og behov der er relevante for it-afdelingen for at dette projekt bliver et som firmaet vil benytte sig af. Det anbefales at de bliver inddraget i processen af flere omgange, evt. Som udviklingen sker, men også i starten, så man ved at deres systemer er kompatibelt med programmet.

**Salgschefen (Martin)** har en høj medvirken og indflydelse på programmet, da han er med til at udvikle kravene (user stories), og er personen der har rekvireret et nyt program. Han bør opdateres ugentligt og ved ændringer i flows.

**Gruppe E** er udviklerne som skal udarbejde programmet og har derfor viden om hvorvidt ønskede funktioner er mulige. Udviklerne sidder med projektet hver dag og skal derfor opdateres med det samme, hvis der sker ændringer i krav eller lignende.

## Grå eminencer

**Regnskabschefen** har en økonomisk interesse i projektet, og kan vælge at nedlægge projektet hvis det ikke anses for værende profitabelt. Da vedkommende har mulighed for at nedlægge projektet, har vedkommende en høj indflydelse på projektet, men ikke megen anden medvirken.

**Salgsdirektøren** er salgschefens chef, og har derfor lagt ansvaret for projektet hen til salgschefen, som står for projektet. Den pågældende person kan dog vælge at stoppe projektet, hvis det vurderes til ikke at være nødvendigt, profitabelt eller hvis udsigten om effektivisering af arbejdsgangene bliver ændret. Vedkommende har derfor en høj indflydelse på projektet, men lav medvirken, vedkommende burde stadig blive opdateret undervejs, dog ikke lige ofte som salgschefen.

**Den administrerende direktør** er salgsdirektørens chef, og har derfor lagt ansvaret for projektet ned til salgschefen, som står for projektet. Den administrerende direktør kan dog nedlægge projektet, på samme baggrund som salgsdirektøren. Vedkommende har derfor en høj indflydelse på projektet, men lav medvirken, de burde stadig blive opdateret undervejs, dog ikke lige ofte som salgschefen, eventuelt kun ved start og slut, og ved eventuelle større ændringer.

**Juridisk afdeling** har høj indflydelse på projektet i forbindelse med byggekrav og GDPR. Dette vedrører, om oplysningerne der opbevares bliver opbevaret korrekt og om der er belæg for det. Endvidere er det også den juridiske afdeling, der kan bekræfte, om materialerne, som fremgår i projektets database, opretholder eventuelle byggekrav. De har således høj indflydelse på visse dele af projektet, men ikke på om det bliver nedlagt. Deres medvirken vedrører således også kun disse dele.

## Risikoanalyse

Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau
1	FOG går konkurs	Uacceptabelt	Usandsynlig	Høj
2	Sygdom i Gruppe E	Acceptabel	Sandsynligt	Medium
3	Teknologi går i stykker hos et medlem	Uønsket	Muligt	Medium
4	Ustruktureret arbejde	Uønsket	Muligt	Medium
5	Ressource personer er utilgængelig	Uønsket	Muligt	Høj

Figur 2 - Udklip af Risikoanalyse (se bilag 2)

Risiko-analysen er udarbejdet med henblik på hvis man var et udvikler-team som blev lønnet af Johannes Fog med en ressourceperson tilknyttet, men realiteten er, at det er et eksamensprojekt, hvor vi ikke bliver aflønnet eller har mulighed for kontakt med ressourcepersoner fra Johannes Fog.

Derfor var den største risiko for projektet sygdom og at teknologi gik i stykker.

I forbindelse med arbejdet var der en del sygdom, derfor kan der også ses en skævvridning i mængden commits på vores Github.

Planlægningsmæssigt har der også været udfordringer, da sygdom typisk opstår pludseligt, og kan udsætte færdiggørelsen af den syges arbejde. En af vores planer for afværgning af sygdom har været at videregive arbejdet, dette har ikke hver gang været muligt, da andre gruppemedlemmer også har haft deres egne opgaver, hvorfor eventuelle deadlines, aftalt internt i gruppen, har været skubbet.

Dog har der ikke været nogle væsentlige it-problemer der har skabt problemer.



## Teknologivalg

<b>Teknologi</b>	<b>Funktion</b>
<b>IDE</b>	IntelliJ IDEA 2023.3.4
<b>Containerization</b>	Docker Desktop 4.34.3
<b>Database Management</b>	PostgreSQL 42.7.2, PGAdmin 4 (version 8.3)
<b>JDK</b>	Amazon Corretto 22 (version 22.0.2)
<b>SDK-language:</b>	SDK default
<b>Framework</b>	Javalin 6.1.3
<b>Template Engine</b>	Thymeleaf 3.1.2 (Release), Thymeleaf-extras 3.0.4 (Release)
<b>Frontend</b>	CSS3, HTML5, (Bootstrap version 5.3.3)
<b>Build Tool</b>	Maven (compiler version 3.13.0)
<b>Encoding</b>	UTF-8
<b>JDBC Driver</b>	JDBC 4.2
<b>Interface Design Tool:</b>	Figma Desktop App Version 124.5.5
<b>JUnit:</b>	JUnit 5.10.2
<b>SVG</b>	SVG 1.1
<b>SendGrid</b>	Version 4.10.1

Figur 3 - Teknologivalg

## Krav

I interviewet forklarer Martin om deres nuværende systemer og om nogle af deres krav til det nye system. Han gennemgår hvordan deres nuværende systemer fungerer, i forbindelse med ordrehåndtering og bestilling af en carport.

Et af hovedpunkterne som Martin kommer med i interviewet er ønsket om at det bliver til et system, i stedet for to. Endvidere er det meget vigtigt for dem at styklisten ikke bliver sendt til slutkunden før de har betalt.

Vores fokus har således været på at det hele fungerer via én hjemmeside, hvor sælgeren kan logge ind og se alle bestillinger og sende tilbud, samtidig med at tilrette prisen. Samt at en admin kan ændre i materialelisten.

Normalt udvikler man nogle user stories i samarbejde med kunden (Fog), så man kan lave nogle problemstillinger og acceptkriterier. I forbindelse med dette program har vi ikke haft kontakt til kunden, og har kun haft oplysningerne givet i videoen, hvorfra vi selv har udarbejdet nogle user stories.

## User Stories

I dette afsnit kan man læse om de user stories vi lavede og byggede projektet ud fra. Under bilag 3 (se [bilag 3](#)) kan man se vores fulde samling af user stories med tasks og estimat.

User story	Accept-kriterier
1. Som kunde skal jeg kunne udfylde en formular for en custom carport, så jeg kan bestille et tilbud	Givet at jeg har udfyldt alle nødvendige felter, når jeg klikker på knappen "Bestil tilbud", så bliver min forespørgsel sendt.
2. Som sælger, kan jeg vælge en kundes forespørgsel vedr. custom carport, og sende et pristilbud	Givet at jeg er sælger. Når jeg kan vælge en kundes forespørgsel på custom carport. Så jeg kan gennemse detaljerne på carporten og se om det er muligt. Og jeg kan derefter sende et pristilbud til kunden. + en svg-tegning af carporten.
3. Som sælger, kan jeg generere en stykliste så jeg kan sende den til kunden og lageret.	Givet jeg er sælger, når jeg er logget ind og har set at kunden har betalt for sit pristilbud kan jeg generere en stykliste og sende det til kunden + lageret.

4. Som admin, kan jeg logge ind på beregningssystemet- siden så jeg kan tilføje nyt materiale (f.eks. ny tagtype).	Givet jeg er admin, når jeg logger ind på beregningssystemet- siden og kigger på tilgængeligt materiale fra inventaret, kan jeg tilføje nyt materiale.
5. Som sælger kan jeg bruge/oprette en mail-skabelon og indtaste kundens info, så jeg kan sende kunden et pristilbud / en kvittering	Givet at jeg er sælger, Når jeg vælger en kunde og en ordre og klikker på "Opret email skabelon", Så genereres en email skabelon med kundens information, ordreinformation og tegning af carport. Og jeg kan indtaste en ekstra besked i emailen. Og et link til at betale for ens order hvis man accepterer. Og der er en tilbage-knap, som tillader mig at gå tilbage og rette eventuelle fejl.
6. Som kunde, kan jeg klikke på linket i mailen - fra sælgeren så jeg kan afvise mit pristilbud.	Givet jeg er kunde, når jeg klikker på linket i mailen og ender på "betalingssiden" kan jeg trykke på "afvis"-knappen og afvise mit pristilbud.
7. Som sælger skal jeg kunne redigere i tilbudsprisen, inden den sendes til kunden.	Givet jeg er sælger og står på en given tilbuds-formularside , når jeg indtaster en ny tilbudspris kan jeg trykke på "ret pris"-knappen, kan jeg sende tilbuddet til kunden med en anden tilbudspris og dækningsgrad.
8. Som kunde, kan jeg klikke på linket i mailen - fra sælgeren så jeg kan betale for min ønskede forespørgsel (custom carport).	Givet jeg er kunde, når jeg klikker på linket i mailen og ender på "betalingssiden" kan jeg trykke på "betal"-knappen og betale med min balance?
9. Som sælger, kan jeg modtage en email-notifikation så jeg kan se, når en kunde har oprettet en forespørgsel og behandle den.	Givet jeg er sælger, når en kunde opretter sin forespørgsel, kan jeg modtage en mail og vælge at behandle deres forespørgsel.
10. Som admin kan jeg logge ind på beregningsappen- siden så jeg kan fjerne materiale jeg ikke ønsker værende tilgængeligt længere (f.eks. dørhåndtag).	Givet jeg er admin, når jeg er logger ind på beregningsappen- siden og kigger på tilgængeligt materiale kan jeg klikke på et givent materiale og fjerne det.

11. Som admin, kan jeg kan opdatere indholdet i mail-skabelon, som sælgeren sender til kunder.	Givet jeg er admin, Når jeg vælger at opdatere email-skabelonen kan jeg opdatere email-indholdet som sælgerne bruger.
12. Som sælger ønsker jeg at kunne redigere al information i styklisten, så jeg kan opdatere navn, beskrivelser, materialer og mængder efter behov.	Når jeg navigerer til styklisten, skal jeg kunne se en redigeringsknap ved siden af hvert element. Når jeg klikker på redigeringsknappen, skal jeg kunne ændre navn, beskrivelse, materialer og mængder. Når jeg har foretaget ændringerne og klikker på "Gem", skal ændringerne gemmes og vises korrekt i styklisten. Hvis jeg forsøger at gemme uden at udfylde nødvendige felter, skal jeg få en fejlmeddelelse.

Figur 4 - Udklip af User Stories tabel

## Mockups

Når man skal udvikle et produkt med grafik, der skal benyttes af andre, er det oftest aktuelt at udvikle en prototype af det grafiske design. Vi har benyttet Figma, som er en software udviklet til at lave mockups af digitale produkter med et visuelt udtryk.

Vi har brugt vores mockups som en hjælp til udviklingen af vores css. Til vores endelige design af hjemmesiden har vi benyttet bootstrap.

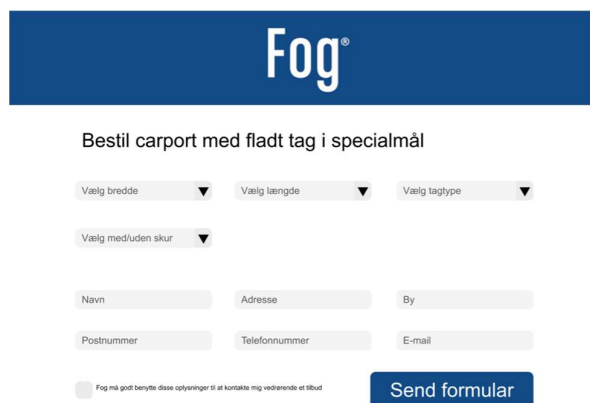
Når man har en kunde, kan et mockup være et godt værktøj til at hjælpe kunden med at visualisere hvordan produktet ender med at se ud. Endvidere kan det gøre så udviklingsteamet og kunden bliver enige om hvordan hjemmesiden skal se ud.

Disse mockups repræsenterer, hvordan vi tænkte på at lave hjemmesiden, med afsæt i Fogs eksisterende logo. Vi har dog lavet det mere simpelt. Mockuppen er udviklet ud fra nogle af gestaltlovene, som arbejder med hvordan man gør noget brugervenligt.

Hele vores mockup hvor man kan se de forskellige processer vi var igennem, findes under links.



Figur 5 - Billede af vores forside



Figur 6 - Billede af vores side hvor kunden udfylder information

# Arkitektur

## Vores valg

Vi har valgt at bruge en MVC (Model-View-Controller) struktur. Dette har vi valgt at gøre, fordi det gør det nemt for os at adskille forretningslogikken, datadomænet og views.

## Begrundelse for valget

Vores designvalg gjorde det nemmere at kvalitetssikre applikationen gennem unit tests, skalere appen efter behov og sørge for ordentlig vedligeholdelse.

Da vi også har at gøre med input-, brugergrænseflade- og forretningslogik, synes vi, det gav en god fordel at anvende denne struktur. Det skabte et godt og struktureret overblik over hele appen og sørgede for, at alle komponenter nemt kunne kommunikere med hinanden. Dertil gør det appen mere robust, og MVC-modellen synes vi derfor er meget attraktiv at bruge til udvikling af større webapps/systemer.

## Særligt ved modellen

Særligt ved vores arkitektur er vores model. Vi har i vores 'entities' pakke et "order" objekt der indeholder alle detaljer om en kundes custom carport samt svg-tegning og alle relevante detaljer til at håndtere forretnings-, backend- og frontend-logikken. Har man behov for flere oplysninger om en ordre fx. Angående materialer, er det nødvendigt at anvende nogle DML-queries (Data Manipulation Language) i form af joins.

## Controller

Vi gør brug af en OrderController, der håndterer primært alt der har at gøre med en ordre. Hertil, vil controlleren også håndtere input/output-logik fra frontenden og kaste relevante værdier videre til mapperen med henblik på at oprette/hente data i databasen.

Vores AccountController sørger for at håndtere data vedrørende en konto, som både kan være en sælger eller en kundekonto. AccountController vil kommunikere med en Mapper, og ved dens hjælp oprette/hente data i tabellen.

MaterialControlleren håndterer opgaver relateret til materialer. MaterialControlleren håndterer primært forretningslogik i forbindelse med at hente materialer ned fra tabellen, som skal vises i frontend eller at tilføje nye materialer i tabellen. Disse nye materialer vil også blive defineret i frontend.

## Mapper

OrderMapper håndterer primært alle opgaver, der har med en ordre at gøre. OrderMapperens opgave er at oprette data i tabellerne vedrørende en ordre eller at hente data fra/om en given ordre, og kaste det videre til en controller, der behandler dataen efterfølgende.

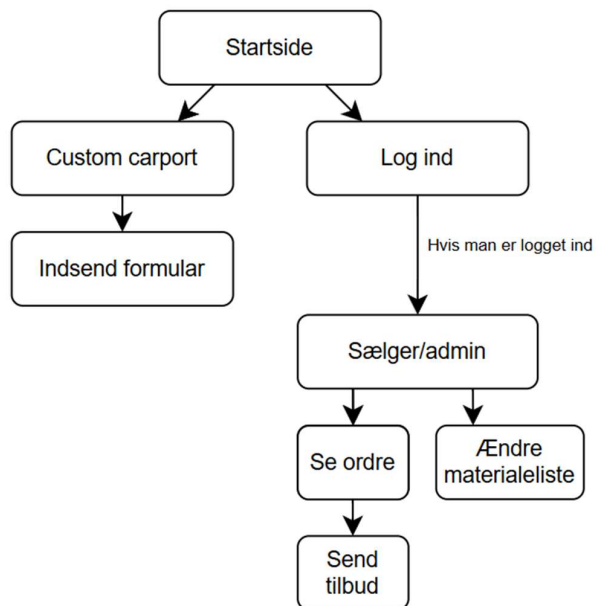
AccountMapper håndterer primært alle opgaver, der har med en konto at gøre. Vi har valgt at kalde det en AccountMapper, da vi skelner mellem en Bruger og en Konto. Vi har tiltænkt, at en bruger forudsætter at en kunde kan oprette en konto og agere som en bruger på platformen. Dette designvalg traf vi tidligt i processen og besluttede at vi ville blot implementere en "kundekonto" som kunden ikke kan logge ind på, men er en konto, der vil blive oprettet i tabellen, som man som ansat hos Fog kan bruge til at tilknytte til en ordre og sende et pristilbud og stykliste.

MaterialMapper håndterer primært alle opgaver, der har med styklisten og materialer at gøre. Vores mapper sørger for at fylde tabellerne ud med data om alle materialer for en given ordre, beregne antallet af materialer der er behov for til en given carport ud fra en given ordre. Ligeledes, er mapperen også ansvarlig for at hente data ned vedrørende materialer), hente og indsætte en stykliste.

## **View**

Vi gjorde brug af Bootstrap i kombination med Thymeleaf, da det gjorde, at vi hurtigt kunne skabe nogle semi-flotte hjemmesider. Bootstrap-teknologien gav os adgang til mange prædefinerede grænseflader af forskellige farver, udseende og layout. Det har gjort at vores app ligner mere en reel app og hvad man kan forvente fra en moderne app/hjemmeside og styrker brugeroplevelsen.

## Navigationsdiagram



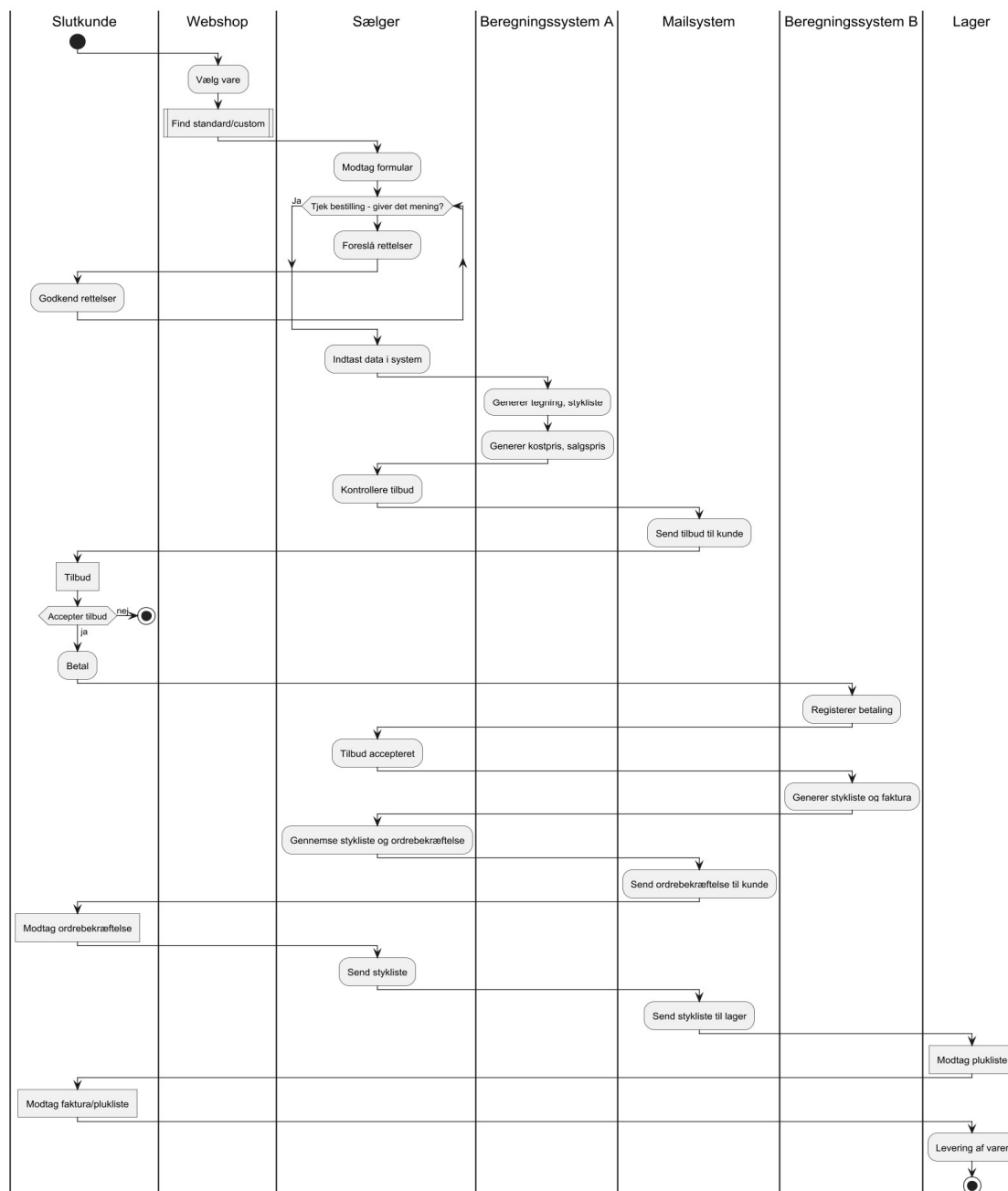
*Figur 7 - Navigationsdiagram*

Vores navigationsdiagram er tegnet i DrawIO, som er et tegneprogram lavet til at lave modeller og diagrammer. Vi synes at DrawIO er nemmere at arbejde med i forhold til navigationsdiagrammet end plantUML. På alle vores sider er der et billede som gør at man kommer til hovedsiden(startsiden). Admin og sælger siden kan kun nås hvis man er logget ind, som er indikeret på diagrammet. Endvidere har de en fælles navigationsbar, hvor man kan logge ud, oprette en ny bruger, se forespørgsler og se materialelisten.

Vi har på forhånd oprettet Martin som bruger og en admin bruger.



## Aktivitetsdiagram



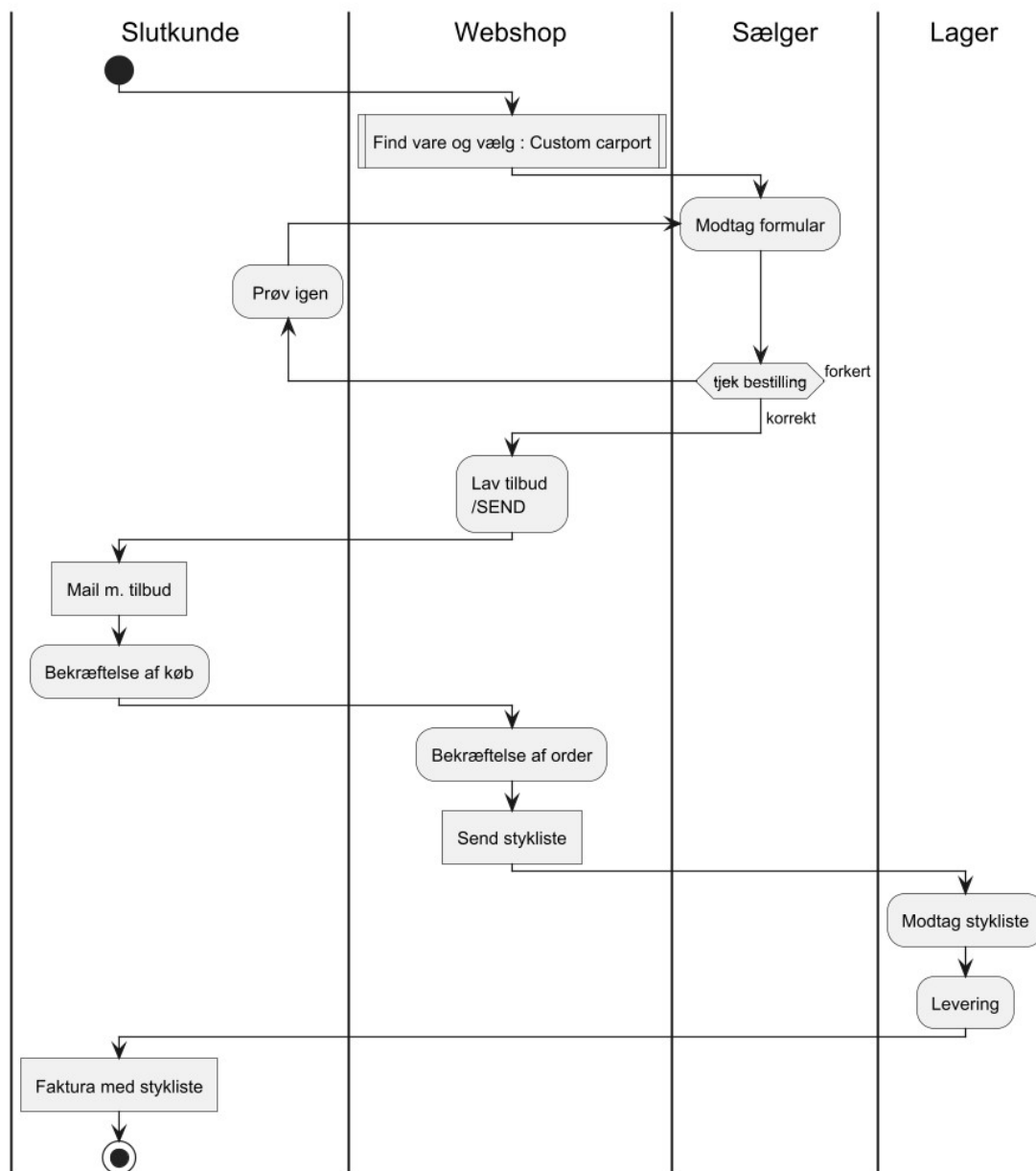
Figur 8 - AS-IS aktivitetsdiagram

### AS-IS aktivitetsdiagram

AS-IS diagrammet som set ovenfor viser den nuværende proces for Fogs salg af carporte. Den er opdelt i 7 svømmebaner som er bestående af: **Slutkunde**, **webshop**, **sælger**, **beregningssystem A og B**, **mailsystem** og **lager**.

Diagrammet viser, at købet af en carport er komplekst og indebærer flere trin. Slutkunden opretter en produktforespørgsel via webshoppen, hvorefter sælgeren tjekker målene, eventuelt justerer dem efter dialog med kunden, og sender forespørgslen videre til beregningssystem A. Her beregnes stykliste, kostpris og salgspris, som sælgeren kontrollerer, før han sender et tilbud til kunden via mailsystemet.

Hvis kunden accepterer og betaler, bekræftes købet gennem beregningssystem B, der genererer stykliste og faktura til sælgeren for kontrol. Herefter sender sælger ordrebekræftelse og faktura til kunden samt stykliste til lageret, som sørger for levering.



Figur 9 - TO-BE aktivitetsdiagram

## TO-BE aktivitetsdiagram

TO-BE-aktivitetsdiagrammet er en forenklet version af AS-IS-aktivitetsdiagrammet og er opdelt i fire svømmebaner: **Slutkunde**, **Webshop**, **Sælger** og **Lager**.

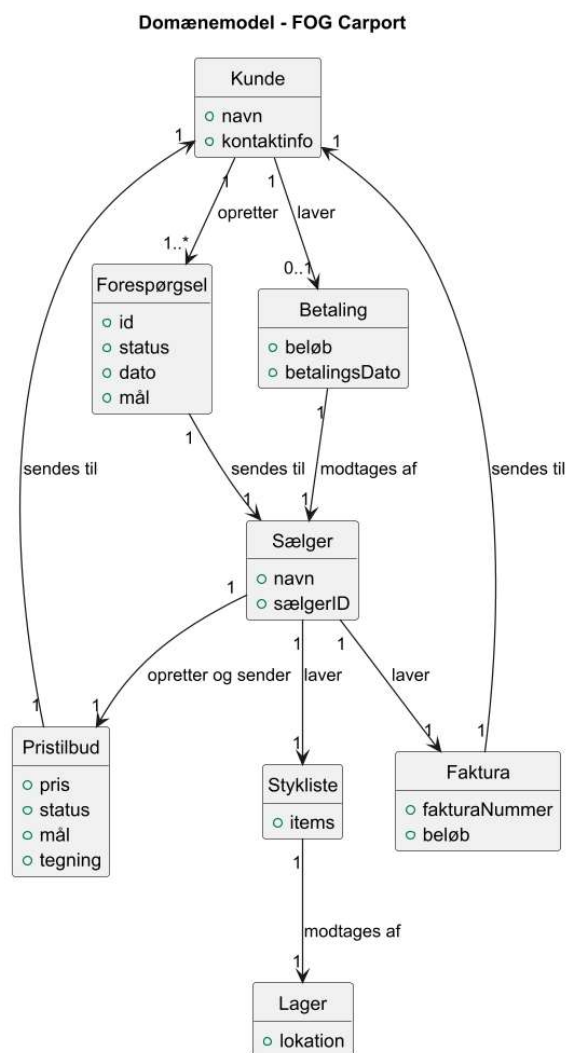
Processen begynder med, at slutkunden besøger webshoppen, vælger en carport og udfylder en formular med alle nødvendige oplysninger. Formularen modtages derefter af sælgeren, som kan tilgå den via en webside og gennemgå bestillingen. Sælgeren har mulighed for at

sende et tilbud til kunden; hvis det ikke sker, skal slutkunden udfylde en ny formular og foretage en ny bestilling.

Tilbuddet sendes til slutkunden via e-mail, der indeholder et link til webshoppen, hvor kunden kan bekræfte og acceptere ordren.

Når ordren er betalt, genereres der en stykliste, som sendes til lageret. Styklisten specificerer de materialer, der skal bruges til at fremstille carporten. Samtidig modtager slutkunden en faktura, som sendes til dem via e-mail. Denne email indeholder et link til fakturaen, som kan vises direkte på websiden.

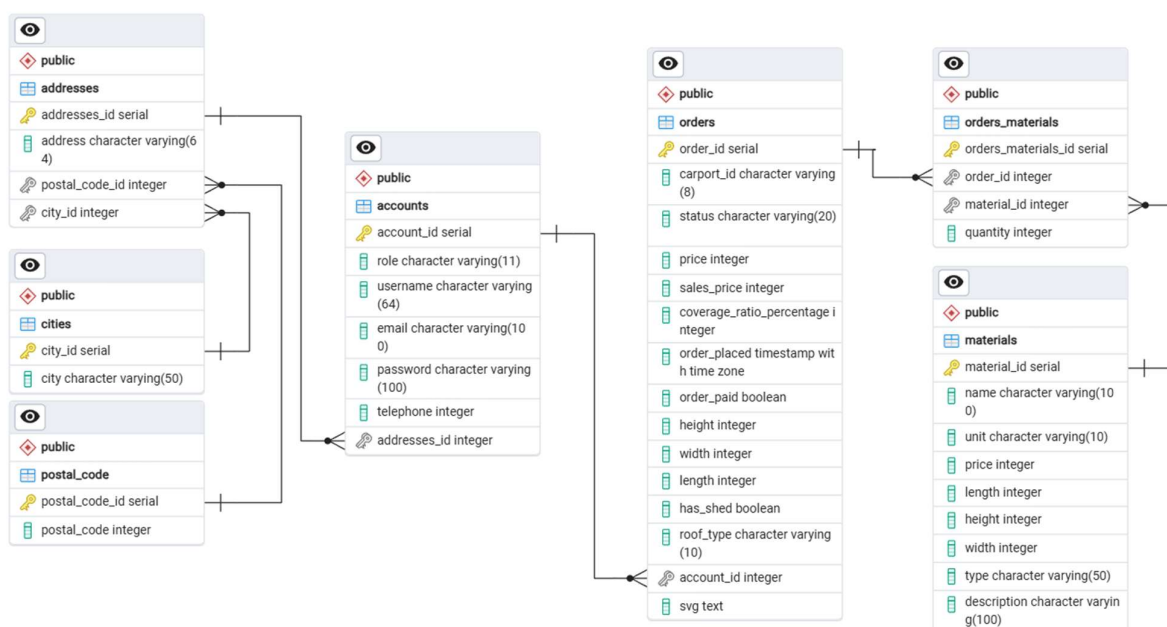
## Domænemodel



Figur 10 - Domænemodel version 1

På vores første udkast til en domænemodel har vi beskrevet hvordan vi mener at processen bag brugen af vores produkt skal fungere. Tanken er at en kunde vil oprette en forespørgsel på en carport som modtages af en sælger fra Fog. Sælger laver et pristilbud til kunden. (Venstre side) Kunden vil så kunne lave en betaling som sælger også modtager. Her vil der så blive oprettet en faktura som sendes til kunden (højre side) og en stykliste som laget får.

## Entity Relationship Diagram (ERD)



Figur 11 - ER diagram

Der er (jf. SQL-scriptet) automatisk to konti der oprettes hhv. En sælger konto og en admin konto. Dertil bliver der tilføjet alt det nødvendige materiale til material-tabellen.

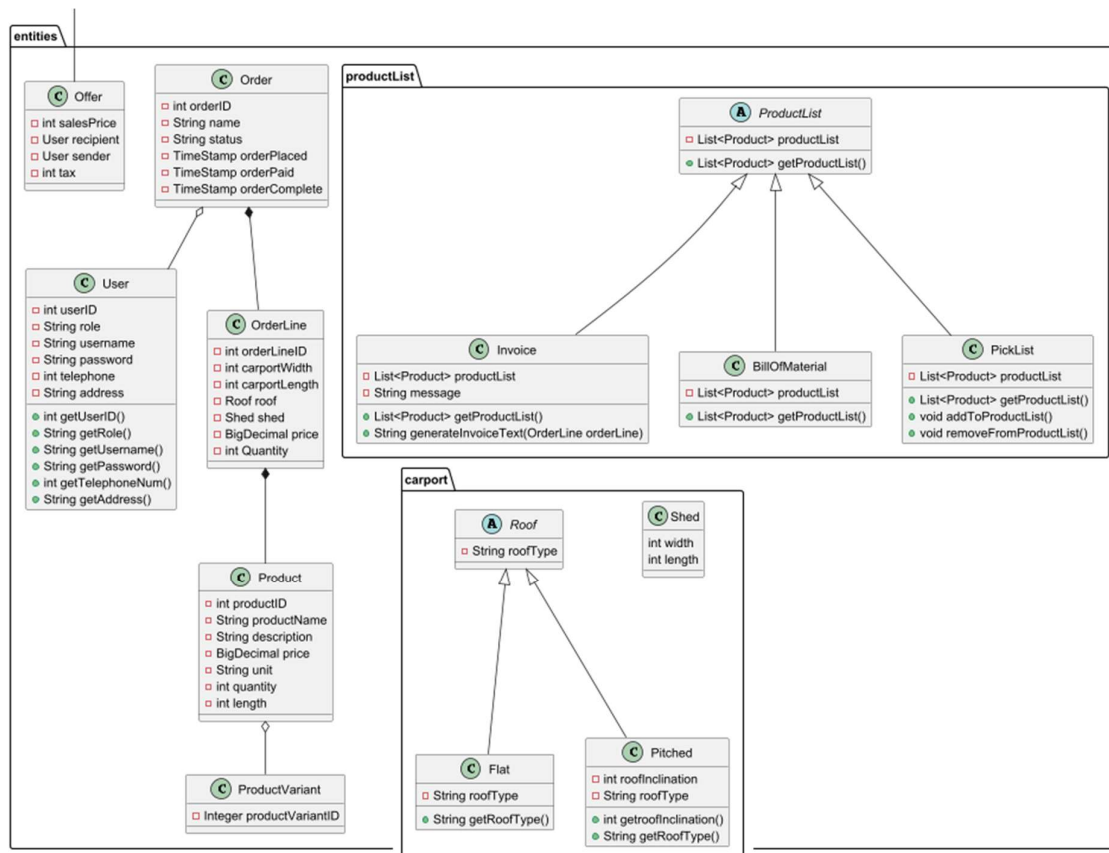
**Vi opfylder 3. Normalform**, da vi sørger for at opdele kolonner, der kan være indbyrdes afhængige af hinanden i deres egen tabel, derved undgår vi at skabe transitive afhængigheder. Samtidig sørger vi for at ingen kolonner indeholder værdier, der eksisterer i en anden kolonne, og at der er en fuld funktionel afhængighed af primærnøglen.

Vi bruger primært 1:M-relationer, da det giver mest mening mht. Fogs forretningslogik og for de forhold der er gældende i applikationsdomænet. Et konkret eksempel på dette kan være at en ordre har mange materialer

Vi har en orders\_materials tabel, den har 2 fremmednøgler, en til materials og orders tabellerne. Formålet med denne tabel er at vise antallet af et givent materiale, der skal bruges til en given ordre, så den repræsenterer vores stykliste til de forskellige ordrer.

Vi har valgt at navngive vores fremmednøgler (FK), det samme navn som tabellen vi refererer til, hedder ud for kolonnen, primærnøgle (PK).

## Klassediagram



Figur 12 - Udklip af klassediagram version 1

I Klassediagrammet (version 1) (se ovenfor) som var vores første udkast til vores programstruktur, kan man se de overordnede overvejelser vi havde gjort os i forhold til strukturen af vores Java backend. Vi brugte Model-view-controller (MVC) til design mønster i vores klassediagram, og plantUML til at skrive det. Vi opdelte vores klasser i forskellige packages. De packages har vi navngivet: **Controllers**, **Persistence**, **Entities**, **Utilities** og **Exceptions** (se [bilag 4](#)).

Vores entities package er der hvor funktionaliteten af vores kode er blevet lavet. Her i vores første udkast havde vi en **Order** og en **OrderLine** klasse. Vi valgte at tilføje disse klasser, da man i vores database også har disse tabeller og det ville gøre det muligt for en kunde at bestille flere carporte per ordre, skulle det ønskes.

Vi havde også to andre klasser i samme package kaldet Product og ProductVariant. De var tiltænkt at kunne håndtere alle slags materialer som skal bruges når der oprettes en carport og evt. nye materialer til databasen, hvis Fog skulle have brug for at opdatere deres liste af materialer.

I vores utilities package valgte vi at lave en Calculator klasse (se [bilag 7](#)), som skulle indeholde vores beregningssystem. Den skal bruges til at lave alle beregningerne når det kom til størrelsen, typen og antallet af byggematerialer, som diverse carporte skal bruge.

Calculator klassen hører tæt sammen med vores MaterialMapper klasse som i klassediagram version 3, MaterialMapper ligger i vores persistence package. MaterialMapper klassen bruger beregningerne fra Calculator klassen, når der oprettes lister af materialer til MaterialController klassen, som fremviser materialelister ved køb af en carport.

I version 2 havde vi en package kaldet productList. Denne package skulle indeholde en abstract klasse som hed det samme, nemlig ProductList. ProductList skulle indeholde faktura, plukliste og stykliste, da de har fælles træk, men også har unikke egenskaber. Dette valgte vi dog fra i version 3, da vi i stedet blot lavede metoder der lavede styklisten (getPickList) i MaterialMapper.

Vores exceptions package indeholder databaseException klassen som givet navnet, indeholder metoder for exceptions i databasen.

Controllers packages indeholder vores nævnte MaterialController som bruges til at sende informationer fra mappers til HTML-siderne, f.eks. når kunden har bestilt et tilbud, kommer der en SVG tegning op af carporten.

For en mere detaljeret forklaring af, hvad tanken bag de forskellige packages og strukturen af vores program, læs afsnittet om **Arkitektur**.

# Udvalgte kodeeksempler

## Eksempel på vores beregningsmetoder

### Udregning af remme

I metoden `calcBeams` udregner vi antallet af remme der skal bruges i en given carport. Vi giver et carport objekt med som en parameter, så vi ved hvor lang carporten er. Vi har 2 typer remme, en med længden 480 cm og en med længden 600 cm. Så har vi lavet nogle if og else if statements der tjekker længden på carporten, og sætter en længde på den type rem man skal bruge (`length`) og antallet man skal bruge af dem (`quantity`). Til sidst returneres et integer array med disse 2 tal.

```
public static int[] calcBeams(Carport carport)
{
    int[] beams = new int[2];
    int quantity = 0;
    int length = 0;

    if (carport.getLength() <= 300)
    {
        //Woodmaterial length = 600
        length = 600;
        quantity = 1;
    } else if (300 > carport.getLength() || carport.getLength() <= 480)
    {
        //Woodmaterial length = 480
        length = 480;
        quantity = 2;
    } else if (480 > carport.getLength() || carport.getLength() <= 600)
    {
        //Woodmaterial length = 600
        length = 600;
        quantity = 2;
    } else if (600 > carport.getLength() || carport.getLength() <= 780)
    {
        //Woodmaterial length = 480
        length = 480;
        quantity = 4;
    }

    beams[0] = quantity;
    beams[1] = length;

    return beams;
}
```

Figur 13 - Billede af metoden `calcBeams`, som udregner hvormange bjælker og hvilken længde der skal bruges



## Udregning af x og y positioner til spær

I metoden `calcRaftersXY` udregner vi x og y koordinater til spærerne som skal bruges til at tegne vores SVG tegning. Den har et `carport` objekt og en integer `matNum` som parameter. `Carport` objektet giver os længde og bredde på den carport vi skal tegne. `matNum` er nummeret på det spær vi skal sætte x og y koordinater til, vi kalder senere denne metode i et loop. Vi har sat afstanden mellem hvert spær til 60 cm jf. materialelisten, spærrenes koordinater bliver sat med et mellemrum på 60 fra hinanden. Hvis et spær har et `posX` der er større end længden på carporten, sætter vi x koordinatet til at være lig med længden af carporten, så vi ikke har spær der ligger udenfor carportens længde, men bliver placeret på enden.

```
public static int[] calcRaftersXY(Carport carport, int quantity, int matNum)
{
    int[] posXY = new int[4];
    // width is the y-axis
    int width = carport.getWidth();
    // length is the x-axis
    int length = carport.getLength();

    // afstanden mellem spærrene er taget fra materialelisten vi fik udleveret
    int spacing = 60;

    int posX = matNum * spacing;

    for (int i = 0; i <= matNum; i++)
    {
        posX = i * spacing;
        if (posX > length)
        {
            posX = length;
        }
    }

    int startPosY = 0;
    int endPosY = width;

    posXY[0] = posX;
    posXY[1] = posX;
    posXY[2] = startPosY;
    posXY[3] = endPosY;

    return posXY;
}
```

Figur 14 - Billede af metoden `calcRaftersXY`, som udregner X og Y positioner til spær

## Eksempel på generering af SVG

I vores SVGCreation klasse har vi f.eks. en metode generateRafters, der generer en String ud fra en liste af spær (rafters) som har fået sat x og y koordinater, fra den tidligere beregningsmetode (se [figur 14](#)). Metoden looper igennem alle spærerne i listen og får generet en string med en linje for hvert spær, hvor hvert x og y koordinat er indsat.

Det samme vil så ske for alle de andre materialer der skal bruges til SVG-tegningen, så vi til sidst kan render tegning på en html side.

```
public static String generateRafters(List<Material> raftersList)
{
    StringBuilder raftersBuilder = new StringBuilder();
    for (int i = 0; i < raftersList.size(); i++)
    {
        int startPosX = raftersList.get(i).getSvgStartPosX();
        int endPosX = raftersList.get(i).getSvgEndPosX();
        int startPosY = raftersList.get(i).getSvgStartPosY();
        int endPosY = raftersList.get(i).getSvgEndPosY();
        raftersBuilder.append(String.format(
            "<line id=\"rafter%d\" x1=\"%d\" x2=\"%d\" y1=\"%d\" y2=\"%d\" stroke=\"darkgray\" stroke-width=\"5\" />\n",
            i + 1, startPosX, endPosX, startPosY, endPosY
        ));
    }
    return raftersBuilder.toString();
}
```

Figur 15 - Billede af metoden generateRafters til brug ved SVG

## Eksempel på håndtering af emails til slutkunden

I vores OrderController har vi en metode kaldet acceptOrDeclineOffer. Der bruges en SendGrid API til at sende mails. Når sælgeren har valgt en ordre og sendt et pristilbud, er status på ordren 'tilbud\_sendt'. Hertil, når kunden har accepteret pristilbuddet, får vi en action værdi og et orderID fra html-skabelonen så vi kan hente den rigtige ordre fra databasen. Når styklisten er sendt, opdaterer vi status på tilbuddet til 'TILBUD\_GODKENDT' og ordrestatus til at ordren er betalt. Hvis kunden afviser tilbuddet, bliver ordren slettet fra systemet.

```
try
{
    String orderID = ctx.formParam( key: "offerid");
    Order order = OrderMapper.getOrderByID(Integer.parseInt(Objects.requireNonNull(orderID)), pool);
    if ("accept".equals(action)) // if customer pays for order, BOM is sent
    {
        SendGrid.sendBOM(email, subject: "Stykliste", order);
        OrderMapper.setPaymentStatusToPaid(Integer.parseInt(Objects.requireNonNull(orderID)), pool);
        OrderMapper.updateOrderStatusAfterPayment(Integer.parseInt(Objects.requireNonNull(orderID)), StatusType.TILBUD_GODKENDT, pool);
        ctx.attribute("message", "Tak for at have handlet hos Fog - byggemarked.");
        ctx.redirect( location: "/");
    } else if ("reject".equals(action)) // if customer declines order, customer data is deleted
    {
        OrderMapper.deleteOrderByID(Integer.parseInt(Objects.requireNonNull(orderID)), pool);
        ctx.attribute("message", "Din ordre er slettet. ");
        ctx.redirect( location: "/");
    }
} catch (DatabaseException e)
{
    ctx.attribute("message", e.getMessage());
    ctx.render( filePath: "/order/{id}/acceptoffer");
} catch (IOException e)
{
    ctx.attribute("message", e.getMessage());
    ctx.render( filePath: "/order/{id}/acceptoffer");
}
ctx.render( filePath: "acceptoffer.html");
```

Figur 16 - Udklip af metoden acceptOrDeclineOffer, som håndterer afsendelse af emails til slutkunden

## Eksempel på hentning af data fra databasen

I vores MaterialMapper klasse har vi en metode kaldet getPickList. Her forbereder vi en liste til at holde på de materiale objekter vi vil lave og en String med en SQL query vi vil skrive til vores database. Metoden tager et orderId som parameter, så vi kun får materialer for en given ordre. SQL koden laver en JOIN på vores 2 tabeller, material og orders\_materials som har nøglen material\_id. Så kan vi få vist en tabel med alle de materialer der hører til en ordre, antallet af dem og andre informationer på materialerne. Oplysningerne bliver så brugt til at lave et materiale objekt, hvorefter det bliver tilføjet til listen pickList i et while loop for hver række i tabellen. Til sidst returnerer metoden listen pickList.

```
{
    List<Material> pickList = new ArrayList<>();

    String sql = "SELECT material.material_id, material.type, material.length, material.name, material.unit, material.price, material.description, " +
        "orders_materials.order_id, orders_materials.quantity\n" +
        "FROM orders_materials\n" +
        "INNER JOIN materials material ON orders_materials.material_id = material.material_id\n" +
        "WHERE order_id = ?";

    try (Connection connection = pool.getConnection();
        PreparedStatement ps = connection.prepareStatement(sql))
    {
        ps.setInt( parameterIndex: 1, orderId);
        ResultSet rs = ps.executeQuery();

        while (rs.next())
        {
            int materialID = rs.getInt( columnLabel: "material_id");
            String name = rs.getString( columnLabel: "name");
            String unit = rs.getString( columnLabel: "unit");
            int price = rs.getInt( columnLabel: "price");
            int length = rs.getInt( columnLabel: "length");
            String type = rs.getString( columnLabel: "type");
            String description = rs.getString( columnLabel: "description");
            int quantity = rs.getInt( columnLabel: "quantity");

            Material material = new Material(materialID, name, description, price, unit, quantity, length, type);
            pickList.add(material);
        }
    } catch (SQLException e)
    {
        throw new DatabaseException("Error fetching materials from the database", e.getMessage());
    }
    return pickList;
}
```

Figur 17 - Udklip af metode getPickList som henter data fra databasen

## Eksempel på håndtering af data fra frontenden

I `showOrderDetails` metoden i klassen `OrderController` gøres der brug af en try-catch hvor vi henter et id fra frontend i vores html-skabelon og sender det som en skjult værdi videre til controlleren. Vi bruger dette id til at fortælle mapperen hvilken ordre, der skal hentes fra databasen. Vi får et `order` + `account` objekt tilbage, der bliver adskilt i controlleren, og renderet på vores html-skabelon. På den måde kan vi vise en given ordre og den tilhørende kunde også.

```
try
{
    String orderID = ctx.pathParam( s: "id");
    int orderIDInt = Integer.parseInt(Objects.requireNonNull(orderID));
    List<Order> orderDetails;
    orderDetails = OrderMapper.getOrderDetails(orderIDInt, pool);

    // get Account object from orderdetails
    Order accountIndex = Objects.requireNonNull(orderDetails).getLast();
    Account account = accountIndex.getAccount();

    Order order = orderDetails.removeLast(); // removes Account object
    String orderSVG = OrderMapper.getSVGFromDatabase(orderIDInt, pool);
    order.setSvg(orderSVG);
    ctx.attribute("orderdetails", order);
    ctx.attribute("account", account);
    ctx.render( filePath: "/orderdetails.html");
} catch (DatabaseException e)
{
    ctx.attribute("message", e.getMessage());
    showOrderHistory(ctx, pool);
}
```

Figur 18 - Udklip af `showOrderDetails` hvor der håndteres data fra frontend

## Eksempel på håndtering af exceptions og fejl

Vi har i vores orderMapper en metode der hedder createQueryInOrders, der har en linje der kaster en SQL-exception, som vil blive grebet i vores controller metode og renderet ud på skærmen til brugeren/slutkunden. Vha. ctx.attribute i controlleren kan vi renderere hele fejlbeskeden, så det er nemmere at fejlsøge.

```
ResultSet rs = ps.getGeneratedKeys();
if (rs.next())
{
    return rs.getInt( columnIndex: 1);
} else
{
    throw new DatabaseException("kunne ikke hente autogenerated ID");
}

} catch (SQLException e)
{
    System.out.println(e.getMessage());
    throw new DatabaseException(e.getMessage());
}
```

Figur 19 - Udklip af createQueryInOrders der viser en exceptions håndtering

## Eksempel på håndtering af input validering

Vi håndterer brugerinput i vores html-formular, blandt andet HTML-siden createQuery. Det vil sige når en kunde bestiller et pristilbud, vil vores formular sørge for at relevante felter skal være udfyldt, før kunden kan komme videre. Dette sparer os for flere linjers kode, vi ellers vil have i controlleren - da vi bruger en 'required'-parameter.

```
←!— Bredde —→
<div class="col-md-4">
  <div class="form-floating">
    <select class="form-select" name="chooseLength" id="chooseLength" required>
      <option selected>Vælg længde</option>
      <option value="240">240 cm</option>
      <option value="270">270 cm</option>
      <option value="300">300 cm</option>
    </select>
  </div>
</div>
```

Figur 20 - Udklip af HTML-siden createQuery

## Eksempel på oprettelse af data i databasen

Vi gør brug af prepared statements. Det har vi valgt, fordi så undgår vi SQL injection angreb og gør koden mere sikker. Jf. vores database design bruger vi primærnøglen i orders-tabellen som en fremmednøgle i orders\_materials tabellen, og sikre derfor at vi får denne autogenerated værdi med fra postgres, når vi opretter en ny ordre og laver en stykliste.

```
public static int createQueryInOrders(String carportID, String status, Timestamp orderPlaced, boolean orderPaid, int length, int width, boolean hasShed, String roofType, int accountID) {
    String sql = "INSERT INTO orders (carport_id, status, " +
        "order_placed, order_paid, length, width, has_shed, roof_type, account_id) " +
        "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";

    try (Connection connection = pool.getConnection();
        PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
        ps.setString(1, carportID);
        ps.setString(2, status);
        ps.setTimestamp(3, orderPlaced);
        ps.setBoolean(4, orderPaid);
        ps.setInt(5, length);
        ps.setInt(6, width);
        ps.setBoolean(7, hasShed);
        ps.setString(8, roofType);
        ps.setInt(9, accountID);

        // Execute the query and retrieve the generated key
    }
```

Figur 21 - Udklip af createQueryInOrders som opretter en ordre i databasen

## Tests (kvalitetssikring)

Vi lavede **JUnit** test på vores **Calculator** klasse, så den har 100% codecoverage. Det er den klasse som står for hele vores beregningsmotor. Så den klasse valgte vi at få testet fuldt ud.

Vi lavede også en **Integrationstest**, hvor vi opsatte et test schema i databasen med vores data. Vi havde dog kun valgt at teste 2 funktionaliteter, at kunne oprette en ordre og få en stykliste med en pris.

Vi havde ikke valgt at lave test på andre metoder og klasser, da det ikke giver så meget mening, f.eks. vores controllers er det ikke muligt at opsætte JUnit test på.



## User acceptance test

I forbindelse med udviklingen af programmet har vi benyttet os af user stories. Når dette gøres, benytter man sig typisk også af user acceptance tests, hvilket er når kunden, i dette tilfælde ville det være Fog, afprøver programmet baseret på de forskellige user stories. Da vi ikke kan bede Martin fra Fog om at afprøve vores system, har vi i stedet bedt en udefrakommende, Keven, til at afprøve de forskellige funktioner i programmet. Han har derefter udfyldt et skema om (se [bilag 8](#)), hvorvidt de forskellige funktioner kan accepteres, baseret på user stories' acceptkriterier.

Vi har valgt at filme Keven imens han testede nogle af vores user stories. Disse videoer kan ses via de links der er tilføjet under afsnittet links.

Udover videoerne, har han udtalt at siderne er overskuelige og nem at finde rundt i.

## Særlige forhold

### Sikker login

Vi har i forbindelse med login for sælgere og admin, valgt at kryptere kodeordene med en Blowfish algoritme der bruger et salt. Hertil bruger vi BCrypt-biblioteket, der tilføjer et unikt salt til brugerens hashede kodeord og kører det gennem blowfish-algoritmen. Vi beskytter brugerens kodeord mod brute-force angreb og rainbow-table angreb.

### Brugerroller

Vi har defineret tre brugerroller, en admin, sælger og en kunde. Måden vi håndterer disse brugerroller afspejles i vores JDBC database. Her bruger vi en kolonne, der er navngivet "role". Hertil angiver vi, hvilken brugerrolle en given konto har.

### Navigationsbar

Vi bruger en fælles navigationsbar, så det er lettere for slutkunden og ansatte hos Fog at bruge hjemmesiden. Her vil man afhængigt af hvad ens brugerrolle i systemet er have adgang til ordrehistorik, ordredetaljer for en given ordre, materialelisten som er en samlet oversigt af materialer fra Fogs inventar, en opret bruger side og en login side.

### Begrænsede sider

Vi har en del sider der er begrænsede baseret på brugerrollen. Vi har en side hvor man kan opret en bruger, se tilgængelige materialer, se ordrehistorik og se ordredetaljer. Disse sider er kun tilgængelige for admin og sælger. Slutkunden vil kun have mulighed for at tilgå siden hvor man kan bestille et pristilbud eller login siden.



## Ruter & Thymeleaf-skabeloner

GET-ruter fra OrderControlleren:

- /createquery : Fører slutkunden over til "bestil pristilbud" siden
- /orderhistory : Fører sælger over til ordre historikken
- /order/acceptoffer/{orderId} : Fører slutkunden over til "accepter pristilbud side"
- /order/details/{id} : Fører sælgeren over til orde detaljer siden
- /order/billOfMaterials/{id} : Fører slutkunden over til sin stykliste

Vi har en orderhistory.html hvor vi henter data om alle bestilte pristilbud fra databasen og renderer det på en thymeleaf-skabelon.

Bestilte pristilbud							
Order #	Carport ID	Kundenavn	Status på behandling	Ordre oprettet	Betalingsstatus		
						Se	Se
1	CFU	<a href="#">idrees.isci</a>	TILBUD, GODKENDT	2024-12-13 12:16:12.679738	Ordren er betalt	<a href="#">ordre</a>	<a href="#">ordre</a>
2	CFU	<a href="#">Jennifer</a>	TILBUD, SENDT	2024-12-13 12:21:36.543176	Ordren afventer betaling	<a href="#">Se</a>	<a href="#">ordre</a>

Figur 22 - Screenshot af vores hjemmeside, på HTML-siden orderhistory

Vi bruger thymeleaf-skabeloner til næsten alle vores html-sider og de er navngivet som følger:

- Orderdetails.html
- Acceptoffer.html
- billOfMaterials.html
- listOfMaterials.html
- Paidqueries.html
- Receipt.html

## Status på implementation

På vores forside har vi to muligheder når man skal bestille en carport, man kan klikke på en carport med fladt tag og en med rejsning. Teksten til disse bokse mangler at blive opdateret. Der er ikke blevet implementeret funktionalitet til at bestille carport med høj rejsning, hvis man klikker på den, bliver man sendt til forsiden, der sker altså ikke noget.

Vi har også nogle route-fejl på vores orderhistory html når man har klikket sig ind på Se alle forespørgsler, hvis man klikker på skraldespands-ikonet og brugernavnet. Det samme sker hvis man er klikket ind på en ordre, f.eks. order/details/1 og så prøver at trykke på Materialeliste, så får man også en route-fejl.

Vi mangler at implementere unique constrain på vores database for oprettelse af bruger på database niveauet. Så man kan godt oprette den samme kunde flere gange i tabellen, men de får stadig en unique primary key.

Vi har også valgt ikke at implementere funktionalitet for carporte med skur, da scopet ville blive for stort.

Login funktionen virker til både admin og sælger, men lige nu har admin ikke nogen funktioner. Det var tænkt at admin kunne ændre materialelisten, men det er lige nu kun sælger der kan det.

Der mangler også en validering af telefonnumre i bestillingsformularen. Hvis man indtaster noget som ikke er tal, vil man få en server fejl.

Som sælger kan man fjerne og tilføje materialer til databasen, det skulle dog have været admin der havde den funktion.

På vores materialeliste kan man slette materialer fra databasen, måden man gør det på er ved at indtaste ID, navn, højde, længde og bredde. Men på listen er rækkefølgen, længde, bredde og højde. Rækkefølgen til at indsætte materiale i materialelisten er gjort på en tredje måde: længde, højde og bredde. Det skulle have være ensformigt for at være implementeret optimalt.

Vi mangler nogle html sider til når en kunde accepterer eller afviser en betaling. Lige nu bliver man sendt tilbage til forsiden uden nogen besked.

Vi har en html side der hedder paidqueries som vi ikke nåede at få brugt, samt en SVG.xml som blev brugt under test af SVG som ikke blev fjernet igen.

Ellers har vi fået implementeret alle de sider vi har i vores navigationsdiagram (se [figur 7](#)), samt CRUD-funktioner til at kommunikere med vores database.

Styling er også fuldt implementeret med brug af bootstrap. Dette valgte vi fordi bootstrap gav os mulighed for at vores app kan imødekomme en generel standard til hvordan en moderne webapp ser ud.

På vores billOfMaterials side, når kunden har modtaget deres stykliste, så får vi ikke vist antallet af et materiale og alle højder og bredder bliver sat til 0.

Vores SVG mangler et vindkryds som er lavet af hulbånd, vindkrydset skulle gå på tværs af carporten i hver ende. Vores sternbrædder (fasciaboards) er også blevet hardcoded og bruger ikke sternbrædder fra styklisten, så det er ikke dynamisk på den måde. Der mangler også illustrationer af længde og bredde i form af tal til de forskellige af materialer på tegningen.

Generelt mangler der også noget oprydning i koden; der er kommentarer og TODOs som burde fjernes, metoder som tager parametre der ikke bliver brugt og constructors som er

overloaded mange gange. Vi har haft ryddet op, men det var ikke alt der blev fundet før vi havde codefreeze.

## Den faktuelle arbejdsproces

### Faser

1. Fase bestod af analysearbejde. Her snakkede vi om krav til produktet, udarbejdede diagrammer, og påbegyndte rapportskrivningen, hvor vi forklarede vores valg af diagrammer.
2. Fase bestod af vores første sprintuge. Her havde vi fokus på at tage hul på beregningsmotoren og gøre det muligt for slutkunden at kunne bestille et pristilbud.
3. Fase bestod af vores anden sprintuge. Her gik vi videre med beregningsmotoren, og at gøre det muligt at systemet kunne generere en stykliste ud fra en given ordre. Derudover fik vi opsat et mailsystem og testet det af og generelt var fokus på flere thymeleaf-skabeloner, så sælgeren kunne behandle kundeordrer.
4. Fase bestod af vores 3. Sprintuge, hvor vi gik i gang med at teste beregningsmotoren og lave en integrationstest. Dertil, gjorde vi mailsystemet klar til at blive deployet, og generelt var fokus på at produktet opfyldte vores user stories. I denne fase blev SVG afsluttet og implementeret.
5. Fase bestod af at vi gjorde produktet klar til at deploye og dernæst satte det op i skyen samt at gå videre med vores rapportskrivning.

### User Stories

Vi udarbejdede 12 user stories. Undervejs i processen var vi nødt til at ændre funktionaliteten på user story 3, da dens proces var overkompliceret. Vi tog derfor udgangspunkt i, at vi under en dialog med kunden ville have drøftet de tekniske aspekter og krav til user story 3. På baggrund af denne tilgang valgte vi at justere designet for funktionaliteten. Ændringen blev gennemført, da vi vurderede, at det nye design ville forbedre appen og gøre den mere brugervenlig for slutbrugeren.

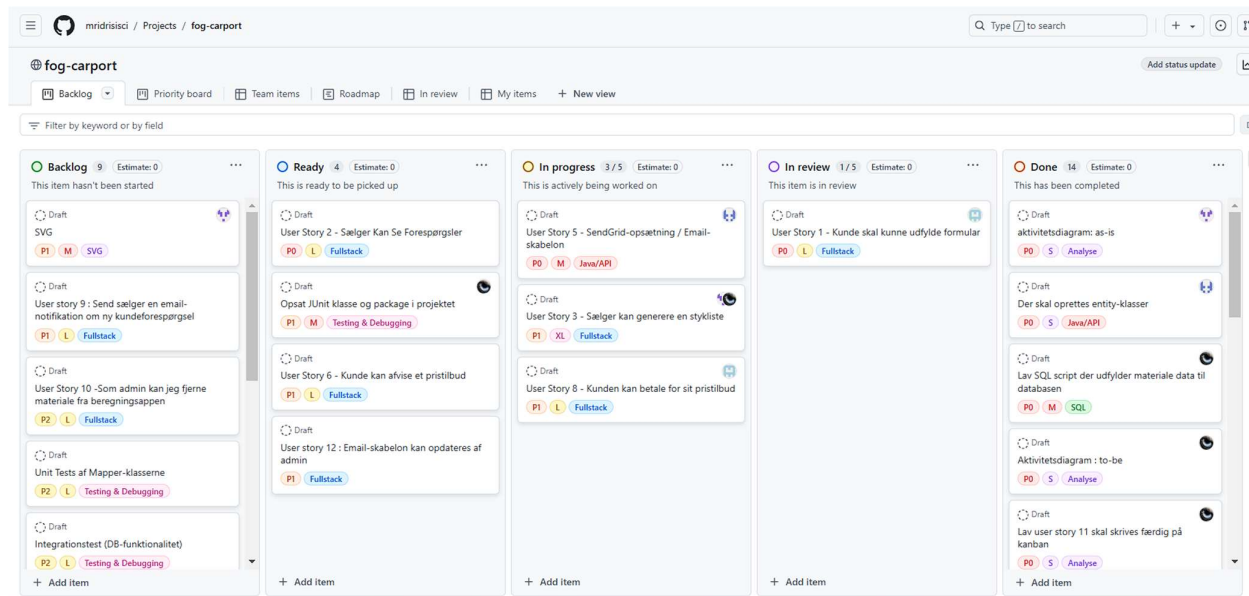
### Git

Vi valgte at Idris skulle være ansvarlig for at reviewe pull requests og merge til dev/main-branchen via GitHub. Vi brugte feature branches, når man skulle lave en ny funktionalitet. Vi endte med hver især nogle gange at arbejde på mere end 1 branch ad gangen. Vi stødte på et git-problem med vores repository, som vi fik løst.

### Kanban

Vi arbejdede efter Kanban-principperne. Vi startede med at fylde Kanban ud med user stories og andre relevante tasks. Derefter specificerede vi de forskellige kort med et estimat på størrelsen for opgaven, hvilken teknologi opgaven tilhører (backend/frontend) og et estimat på hvor høj prioritering kortet har.

Vi havde følgende titler på vores Kanban: Backlog, Ready, In-progress, In-review og Done. Den enkelte programmør var ansvarlig for at tildele sig selv på et kort og rykke kortet over til in-progress og derefter til in-review, når man var færdig med sin user story. Til sidst, var det Idris' ansvar at rykke kortet over til Done, såfremt acceptkriterierne var mødt og user storyen var gennemført. Det var også hans ansvar at rykke kortene fra 'backlog' over til 'ready'.



Figur 23 - Screenshot af vores Kanban-board

## Kommunikation

Vi aftalte i gruppen at vi ville arbejde på skolen og melde ud med de udfordringer man fik i løbet af projektet f.eks. arbejde, sygdom eller andet. Kommunikationen fra start til slut var god både på skolen og over chat. Det var muligt at komme i kontakt med gruppemedlemmerne og få svar på diverse spørgsmål, også uden for arbejdstiden.

## Analysearbejde

Den første uge af projektet brugte vi på analyse og diagrammer. Der blev diskuteret hvordan vi ville tilgå projektet. Vi arbejdede på at opsætte databasen og lavede klassediagram, sekvensdiagram, domænemodel, navigationsdiagram og aktivitetsdiagrammer.

I den første fase af analysearbejdet udviklede vi fælles AS-IS, mens vi gennemgik videoen fra Fog. Efter AS-IS, diskuterede vi hvordan vi selv ville lave vores program og kom frem til et TO-BE diagram.

Efter aktivitetsdiagrammerne delte vi arbejdet op til de resterende diagrammer og gennemgik dem i plenum.

## Pair-programming

I løbet af projektet har vi både kodet individuelt, og sammen, også kaldet pair-programming. Dette fungerede rigtig godt, især ved de større user stories. Da pair-programming gør, at man

er nødsaget til at kommunikerer og formulerer sig om koden, hvilket skaber en større forståelse. Samtidig med at det gør at man kan diskutere, hvilken fremgangsmåde man skal angribe en stor opgave på.

## Den reflekteret arbejdsproces

I arbejdsprocessen har Idris været vores tech-lead og stået for opsamling af opgaver, udarbejdelse af opgaver og pull requests. Dette har fungeret godt, da resten af gruppen har haft en person de kunne henvende sig til med pull requests, og ellers bare fokusere på deres user stories og tasks. Idris har været god til at merge tingene sammen og holde vores dev-branch opdateret.

### User story 3

Undervejs valgte vi at ændre på designet af funktionaliteten til user story 3, da vi syntes, at det ville være mere effektivt, at sælgeren skulle undlade at manuelt oprette og sende styklisten, efter at en kunde har accepteret sit pristilbud og betalt. Især da styklistens indhold lægger grund til kostprisen på carporten. Dette syntes vi da det kunne betyde, at kunden skulle vente i lang tid på at modtage sin stykliste, frem for at modtage sin stykliste med det samme.

### Kanban-board

Vi udarbejdede 12 user stories. Vores user stories endte med at være alt for store og de burde have været brudt ned i mindre user stories. Det syntes vi ville have skabt et bedre overblik, gjort det nemmere at hjælpe hinanden på kryds og tværs.

Størrelsen på vores user stories var alt for store, og det kunne man se på estimeringen, da størstedelen blev til large estimates. Det påvirkede vores proces, da det gjorde at hver person sad på samme opgave i lang tid og at flowet i Kanban-boardet var langsomt.

På den anden side syntes vi at det var godt, da der altid var noget at tage fat i og arbejde med. Dette var tilfældet da det tog meget lang tid at blive færdig med en given user story. Man kunne altid bidrage til andre user stories hvis de var optaget af en anden programmør, som havde brug for hjælp.

### User Stories

I starten af projektet lavede vi en masse user stories, som vi arbejdede for at opfylde. Undervejs i processen indså vi at nogle af vores user stories ikke kunne forblive som de oprindeligt var tiltænkt, da produktet ændrede sig i løbet af udviklingsprocessen. Det er stort set umuligt at lave en plan, fra starten af, som ikke kommer til at ændre sig når det er en så stor opgave. Det vi kan tage med os videre, er størrelsen på user stories, og at man skal være opmærksom på at man ikke har for mange store tasks.

Eksempelvis var user story 3, som vi havde regnet med, ville tage lang tid, men ikke var for uoverskueligt. Den endte med at være den største user story som vi sad med i flere uger, her ville vi tidligere i forløbet have delt den op i mindre bidder. Vi var enige om at vi gerne ville have lidt tungere user stories og så have mindre af dem, men når vi ender med at side fast på dem i lang tid, skal vi være bedre til at dele dem op. Vi glemte også at lave SVG tegningerne

om til en user story, som det står nu, er SVG bare et punkt på Kanban. Der var også andre ting som vi overså eksempelvis da glemte at skrive user story 7 ned på Kanban og user story 11's indhold, var at blive skrevet færdig på Kanban. Det er disse små fejl som gør at det senere i processen bliver svært at forholde os til hvad vi egentlig havde lagt af planer, vi skal derfor blive bedre til at tjekke vores Kanban igennem for fejl inden vi begynder på kodning.

### **Analysearbejdet**

Da vi selv skulle analysere os frem til kriterier og funktionalitetskrav, var det lidt svært at navigere ift. hvor højt skal man sætte barren og hvad sker der, hvis man ændrer på sin oprindelige plan, men alt stadig virker, som kunden ønsker det?

Vores analysearbejde gik overordnet set godt. Der var nogle diagrammer vi diskuterede meget, f.eks. vores ER diagram til databasen. Der var nogle kommunikationsudfordringer med at forklare hvad man mente og blive forstået. Så vi endte med at bruge meget tid på at snakke i ring om hvordan vi ville lave det.

Vores første version af klassediagrammet var også forholdsvis simpelt, da vi ikke havde det store overblik over hvad vi havde brug for i programmet. Men samtidig havde vi sat vores scope ret stort. Vi har arbejdet på at sætte scopet ned i fremtidige versioner.

Vores første klassediagram (version 1) (se [figur 12](#) og [bilag 4](#)) virkede godt som udgangspunkt til begyndelsen på projektet. Men som vi hurtigt fandt ud af, så havde vi meget store ambitioner for hvad vi kunne nå at lave til projektet. Derfor tog vi valget, at down scope vores projekt og fokusere på at lave et produkt som fungerer med lidt færre funktioner end at have et produkt hvor vi havde mange halvfærdige funktioner. Derfor har vi valgt at skære tag med rejsning fra til at starte med. Det har vi valgt, da det er den mest komplicerede af de to tagtyper og vi har vurderet at det ville tage væsentligt mere tid at lave og implementere end fladt tag. Det betyder dog ikke at vi ikke vil have det med, men at det blot er blevet nedprioriteret, til fordel for andre ting i projektet.

Med tilbageblik på det første klassediagram har vi også indset at OrderLine ikke er det mest optimale. Det kommer af at en kunde nok aldrig ville bestille mere end én custom carport ad gangen og derfor fjerner det nødvendigheden for at kunne bestille mere end én ad gangen.

I Klassediagrammet (version 2) (se [bilag 5](#)) er vi gået bort fra OrderLine klassen, i stedet havde vi en Carport klasse som ville indeholde alle variabler der tilhører den fysiske carport. Vi havde også valgt at lægge vores beregningsmetoder her. Vi gik bort fra dette i version 2, da en kunde kunne bestille mere end 1 carport på en ordre af gangen. Så hvis en kunde vil bestille mere end 1 carport, ville de være nødt til at lave en ny bestilling.

Vores endelige klassediagram (version 3) (se [bilag 6](#)) er udfyldt med alle relevante metoder, parametre og relationer mellem de forskellige klasser og packages. Vi endte også med at lave Calculator-klasse, som ligger i utilities, der har statiske metoder til at beregne de forskellige udregninger af materialer. Den bliver brugt af forskellige klasser, særligt MaterialMapper som henter hvilke materialer der skal bruges fra databasen og laver en stykliste.

### **Pair-programming**

I forbindelse med programmering af især de større opgaver på Kanban-boardet såsom user story 3 og SVG-opgaven har vi benyttet os af pair-programming især til udregningsmetoderne. Dette åbnede op for flere muligheder for at tale om koden og hvordan tingene skulle programmeres. Dette gjorde også at dem der sad med koden, fik en bedre forståelse for det enkelte stykke kode, da den blev diskuteret i dybden, imens man kodede. Dog kom det også med visse udfordringer, da der var risiko for at ens gruppemedlem overtog ens computer i noget tid, og man derfor sad uden mulighed for at arbejde videre. Dertil kunne vi have benyttet sig af at partneren kunne lave en feature branch og arbejde derfra i stedet, så man kun sad ved sin egen computer og skrev.

### **Problemløsning / kode udfordringer**

Gruppen var rigtig gode til at tage udfordringer op med hinanden og tale om løsninger, det hænger sammen med vores gruppe kommunikation. Vi var også gode til at hjælpe hinanden hvis nogen sad fast med et problem og f.eks. bruge debuggeren til hurtigt at finde frem til fejlen og løse det. Selvom vi tit havde nogle udfordringer med at koden ikke ville virke eller forvirrende fejlbeskeder, var vi i gruppen gode til at arbejde os ud af det. Vi var rigtig gode til at spørge hinanden om hjælp og ved hjælp af debugging kunne vi hurtigt komme videre i processen.

### **Kommunikation**

Gruppen var fra starten af enige om at vi gerne ville mødes inde på skolen på bestemte tidspunkter og arbejde sammen for bedre at kunne kommunikere med hinanden og også for at kunne spørge om hjælp i forhold til kode men også til forståelse for projektet og hvad status var, kort sagt var der god kommunikation, sparring og gruppedynamik. Det har gjort at projektet kørte godt, og selv hvis en person sad med ét emne, havde alle mulighed for at spørge ind til det og komme med input. I forhold til sygdom har der været god kommunikation fra de syge, så alle vidste hvad der blev arbejdet på selv når gruppemedlemmer ikke har haft mulighed for at møde op på skolen. Selv når vi er taget hjem, har der også været kommunikation mellem gruppemedlemmer, hvis man har arbejdet videre derhjemme eller hvis man lige er kommet på et spørgsmål man gerne ville have svar på. Alt kommunikation som har foregået online er sket gennem Discord-appen. Men selv om vi har mødtes i person, har der stadig været problemer i form af misforståelser mellem gruppemedlemmerne.

Eksempelvis blev der brugt rigtig mange timer over flere dage på at diskutere klassediagram og ER diagram i analyseugen mellem to personer i gruppen og det skyldes miskommunikation. Her skulle andre medlemmer i gruppen have trådt til tidligere i forløbet og hjulpet problemet og sige at vi skulle til at komme videre i stedet for at snakke forbi hinanden. Så der skulle have været taget bedre hånd om diskussioner for at forhindre og selv at bruge for lang tid på det samme.

### **Git**

Vi synes det gik godt at have en teach-lead, så vi undgik merge konflikter, og at vi var sikre på at vores Code of Conduct blev overholdt. Det fungerede også godt at have en person ansvarlig for at review pull requests og merge til dev/main-branchen og sikre at vores acceptkriterier blev overholdt samt user stories.



Vi oplevede at have mange aktive branches, da man nogle gange sad på mere end en ad gangen. Dette betød også at vi ikke kunne rydde op i alle branches særlig tit. Vi fik en "git origin history" fejl med vores repository, og var bange for at skulle lave et nyt repo. Heldigvis kunne vi løse problemet og beholde vores repository.

### **Implementering af SVG**

Udviklingen af SVG tegninger i projektet har ikke været så nemt som vi først havde antaget. Vi havde taget højde for at det nok ville tage noget tid at udvikle og implementere ind i produktet, dog viste det sig at være mere omfattende end forventet. Vi gav os selv det mål at begynde på det tidligt i processen, men på grund af sygdom i gruppen blev det udskudt til senere. Da vi endelig begyndte at skrive koden til SVG, gik det op for os at vores beregningsmotor manglede flere metoder som var nødvendige for at kunne lave vores SVG. Så på grund af nødvendigheden for at have en færdig beregningsmotor før vi kunne lave SVG endte vi med at have meget mindre tid til at lave det og slutproduktet endte med at have mangler og den ser meget "primitiv" ud. Til et fremtidigt projekt ville det være bedre at sætte sig mere ind i nye emner i den første uge hvor vi alligevel laver analysearbejde for at få en bedre forståelse for hvad og hvor stort et emne er for et produkt.

### **Sygdom**

I starten af projektet lavede vi en risikoanalyse, hvori vi tog udgangspunkt i at vi et projekt for en virksomhed. Da dette er et eksamensprojekt, vurderede vi derfor at sygdom og hardwareproblemer ville have den største risiko for at skade projektet, og andre dele af analysen ikke var relevante, såsom hvis Fog gik konkurs. Vi nedprioriterede hvad risikoanalysen faktisk betød for os da vi ikke regnede med at det ville have den store indflydelse på os, men ganske rigtigt så blev alle gruppemedlemmer ramt af sygdom i løbet af projektet, og det nedsatte produktiviteten for den person som var syg på det gældende tidspunkt. Selv med god kommunikation fra de sygemeldte har gruppen stadig været nødsaget til at arbejde mere for at dække for de syge. Selv om vi næsten nåede i mål med det vi havde sat os for, gjorde sygdom at vi ikke havde ekstra tid på hånden til at kunne arbejde med ekstra funktioner som vi gerne ville have implementeret i produktet, eksempelvis tag med hældning. Selv om det gik godt og vi endte med et tilfredsstillende produkt, skulle vi have taget risikoanalyse mere seriøst og brugt lidt mere tid på at sætte et system op i tilfælde af sygdom, så man ikke kommer ud for samme udfordringer.



## Bilag

### Bilag 1. Interessent-analyse

Interessent-analyse				
Interessent	Fordele	Ulemper	Vurdering af position	Håndtering
Slutkunde	De kan opnå en hurtigere betjening. Dette kan dertil give øget kundetilfredshed	Kan være der forekommer problemer i indkøringsperioden	Placeret i øverste kvadrant til venstre, da de har lav indflydelse men middelhøj interesse	
Sælgere	Nemmere arbejdsgang og dertil øget effektivitet.	De skal lære et nyt system, dette kan være svært.	Der er middelhøj interesse, hvorfor de er i øverste kvadrant, men er på kanten til høj indflydelse, da den enkelte sælger ikke har indflydelse på projektet.	Afdelingen bliver orienteret inden implementeringen af systemet af deres salgschef. Det anbefales at have en testperson med fra sælger-teamet til at give feedback under udviklingen
IT-ansatte	Da der benyttes et beregningsprogram i stedet for to, kan det give nemmere arbejdsgang	De skal lære at vedligeholde et nyt system. Der kan være problemer med migrationen, som de skal stå for.	Der er middelhøj interesse, hvorfor de er i øverste kvadrant, men er på kanten til høj indflydelse, da den enkelte sælger ikke har indflydelse på projektet.	Afdelingen bliver orienteret inden implementeringen af systemet af deres nærmeste leder. Det anbefales at have en testperson med fra IT-afdelingen til at give feedback under udviklingen.
Lager-ansatte	Umiddelbart er de ikke direkte påvirket, med mindre styklisterne/pluklisterne ændres.	Der kan ske kommunikationsfejl i systemerne under implementeringen af det nye system, som kan forsinke dem.	Ligger i bunden af den øverste kvadrant til venstre, da de har relativt lave indflydelse og middel-høj interesse	De skal opdateres undervejs, således de ved hvornår implementeringen går i gang.

Andre Fog-ansatte	-	-	Lav indflydelse og lav medvirken, er i nederste kvadrant til venstre.	De opdateres ved et opslag på Fogs dointerne hjemmeside
Bestyrelsen i Fog	Kan øge profit vedrørende special carporte over tid	Der er en periode, hvor der bliver brugt tid og penge på oplæring. Samt omkostningen for udviklingen af systemet	Lav indflydelse og lav medvirken, hvorfor de er i nederste kvadrant til venstre. Dette er da det antages at projektet er godkendt af adm. Direktøren.	Opdatering på mail med status, efter møder med salgschef.
IT-testperson	Da der benyttes et beregningsprogram i stedet for to, kan det give nemmere arbejdsgange. Samt da vedkommende er med som testperson, kan vedkommende komme med input til krav vedr. vedligeholdelse.	De skal lære at vedligeholde et nyt system. Der kan være problemer med migrationen, som de skal stå for.		Informeres og interviews vedrørende projektet inden start og undervejs.
Sælger-testperson	Nemmere arbejdsgang og dertil øget effektivitet. Samt da vedkommende er med som testperson, kan vedkommende komme med input til krav vedr. brugervenligheden	De skal lære et nyt system, dette kan være svært.		Informeres og interviews vedrørende projektet inden start og undervejs. Bruges til at teste systemet inden implementering.

Salgschef	Skaber nemmere arbejdsgang for sælgere, kan derfor øge effektiviteten og dernæst kundetilfredshed . Øget kundetilfredshed kan give øget salg. Endvidere gør programmet det nemmere at opdatere priserne på lageret.	Sælgerne skal lære et nyt system, det koster tid og penge. Der kan ske problemer i forbindelse med oplæringen, hvis sælgerne har svært ved programmet.	Vedkommende har høj interesse og høj medvirken i projektet og ligger derfor i øverste kvadrant til højre.	Informeres om projektets fremgang og involveres i større beslutninger der vedrører brugervenligheden. Gøres ved møder med projektlederen.
Adm. direktør	Kan øge effektiviteten over tid, samt øget profit.	Der er en periode, hvor der bliver brugt tid og penge på oplæring.	Vedkommende har høj indflydelse men relativ lav medvirken, da det antages at Salgschefen har fået tilladelse til agere på vedkommendes vegne i forbindelse med projektet.	Opdatering på mail med status, efter møder med salgschef.
Regnskabschef	Kan øge effektiviteten over tid, samt øget profit.	Der er en periode, hvor der bliver brugt tid og penge på oplæring. Samt omkostningen for udviklingen af systemet	Vedkommende har høj indflydelse men relativ lav medvirken, da det antages at Salgschefen har fået tilladelse til agere på vedkommendes vegne i forbindelse med projektet.	Opdatering på mail med status, efter møder med salgschef.
Salgsdirektør	Kan øge effektiviteten over tid, samt øget profit.	Der er en periode, hvor der bliver brugt tid og penge på oplæring. Samt omkostningen for udviklingen af systemet	Vedkommende har høj indflydelse men relativ lav medvirken, da det antages at Salgschefen har fået tilladelse til at agere på vedkommendes vegne i forbindelse med projektet.	Opdatering på mail med status, efter møder med salgschef.

Juridisk afdeling	-	-	De har indflydelse i forbindelse med GDPR regler og overholdelse af regler i forbindelse med krav for carporte, da det er deres ansvarsområde, endvidere har de derfor også en smule medvirken, og ligger derfor i nederste kvadrant til højre.	Deres involvering vedrører udelukkende om dataen der bliver inputtet i databasen (nuværende data fra tidligere system) stadig er "up to date", og input vedrørende hvilke data vi må opbevare af slutkunden.
Udviklere/Gruppe E	Bliver betalt af Fog	Skal bruge tid på projektet	Høj medvirken og høj indflydelse, da det er dem der står bag projektets udvikling. Det er dem der kan vurdere om en hvis funktion er mulig.	Dagligt møde om fremgang i gruppen, så alle er "up to date" og kan få hjælp når der er brug for det

## Bilag 2. Risiko-analyse

### Risikoanalyse

Bestilling- og beregningssystem til carporte for Johannes Fog

#### Forebyggelse

Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau	Plan for forebyggelse	Plan for afværgning
1	FOG går konkurs	Uacceptabelt	Usandsynlig	Høj	Gennemgå årsregnskab	Betaling i rater/melde krav til konkursbo
2	Sygdom i Gruppe E	Acceptabel	Sandsynligt	Medium	Arbejde hjemmefra hvis medlem er sløj	Arbejde hjemmefra hvis frisk nok - og videregive opgave
3	Teknologi går i stykker hos et medlem	Uønsket	Muligt	Medium	Sørge for opdatering af software, og benytte GitHub til mange commits og pull requests	
4	Ustruktureret arbejde	Uønsket	Muligt	Medium	Udarbejde modeller og diagrammer, samt benytte os af tasks i User Stories	Ugentlig opsamling på arbejdet og næste uges opgaver
5	Ressource personer er utilgængelig	Uønsket	Muligt	Høj	Sørge for at man har kontakt oplysninger på deres stedfortræder inden arbejdet påbegyndes	

## Bilag 3. User stories

User story	Accept-kriterie	Tasks	Estimat og Status
1. Som kunde skal jeg kunne udfylde en formular for en custom carport, så jeg kan bestille et tilbud	Givet at jeg har udfyldt alle nødvendige felter, når jeg klikker på knappen "Bestil tilbud", så bliver min forespørgsel sendt	Opret HTML. Opret en rute i addRoutes(). Lav metoder i controller der kan tage oplysningerne fra kundens formular. Lav metoder i mapperen som tager kundens oplysninger og opretter en query	Forventet: Stor  Reelt:  Status: Implementeret
2. Som sælger, kan jeg vælge en kundes forespørgsel vedr. custom carport, og sende et pristilbud	Givet at jeg er sælger. Når jeg kan vælge en kundes forespørgsel på custom carport. Så jeg kan gennemse detaljerne på carporten og se om det er muligt. Og jeg kan derefter sende et pristilbud til kunden. + en svg-tegning af carporten	Opret HTML. Opret en rute i addRoutes(). Lav metoder i controller der kan tage i mod mapperens metoder. Lav metoder i mapperen som tager alle ordre/queries	Forventet: Stor  Reelt:  Status: Implementeret
3. Som sælger, kan jeg generere en stykliste så jeg kan sende den til kunden og lageret	Givet jeg er sælger, når jeg er logget ind og har set at kunden har betalt for sit pristilbud kan jeg generere en stykliste og sende det til kunden + lageret	Opret HTML. Opret en rute i addRoutes(). Lav metoder i controller der kan tage i mod mapperens metoder. Lav en klasse med calculator der udregner hvor meget af hvert materiale der skal bruges. Lav metoder i mapperen som tager alle materialerne og laver en stykliste.	Forventet: Ekstra Stor  Reelt: Ekstra Stor  Status: Implementeret - men tilrettet

4. Som admin, kan jeg logge ind på beregningssystemet-siden så jeg kan tilføje nyt materiale (f.eks. ny tagtype)	Givet jeg er admin, når jeg logger ind på beregningssystemet-siden og kigger på tilgængeligt materiale fra inventaret ... kan jeg tilføje nyt materiale	Opret HTML-side med inputfelter til materialedata Valider admin-rolle og session med Thymeleaf Tilføj GET-request i controlleren for at vise siden POST-request for at sende materialedata til mapperen Mapperen indsætter materialedata i databasen.	Forventet: Stor  Reelt:  Status: Delvist Implementeret
5. Som sælger kan jeg bruge/oprette en mail-skabelon og indtaste kundens info, så jeg kan sende kunden et pristilbud / en kvittering	Givet at jeg er sælger, Når jeg vælger en kunde og en ordre og klikker på "Opret email skabelon", Så genereres en email skabelon med kundens information, ordreinformation og tegning af carport. Og et link til at betale for ens order hvis man accepterer.	Opsæt SendGrid og opret email-skabelon med kundens navn, email, ordreinfo og carport-tegning.  Tilføj tekstfelt til ekstra besked og link til betalingsside.  Implementer tilbage-knap til rettelser.  Design brugervenlig grænseflade og test funktionalitet med enheds- og integrationstests.	Forventet: Medium  Reelt:  Status: Implementeret
6. Som kunde, kan jeg klikke på linket i mailen - fra sælgeren så jeg kan afvise mit pristilbud	Givet jeg er kunde, når jeg klikker på linket i mailen og ender på "betalingssiden" kan jeg trykke på "afvis"-knappen og afvise mit pristilbud	Opret en html-side, der rendes for kunden, når de klikker på 'afvis'-knappen  Vis en generisk besked angående afvisningen af deres tilbud.	Forventet: Medium  Reelt:  Status: Implementeret

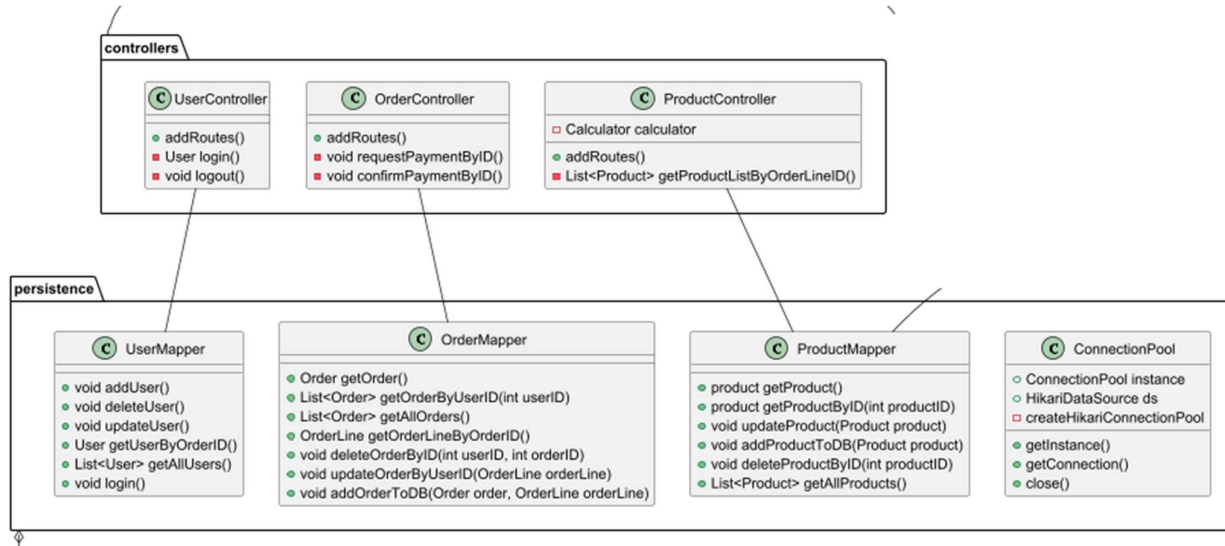
<p>7. Som sælger skal jeg kunne redigere i tilbudsprisen, inden den sendes til kunden</p>	<p>Givet jeg er sælger og står på en given tilbuds-formularside, når jeg indtaster en ny tilbudspris kan jeg trykke på "ret pris"-knappen, kan jeg sende tilbuddet til kunden med en anden tilbudspris og dækningsgrad</p>	<p>Tilføj inputfelt til redigering af tilbudspris på tilbudsformular siden.</p> <p>Implementer "ret pris"-knap, der opdaterer tilbudsprisen.</p> <p>Sørg for, at den nye pris og dækningsgrad gemmes og sendes til kunden.</p> <p>Test funktionaliteten for korrekt opdatering og afsendelse af tilbud.</p>	<p>Forventet: Medium</p> <p>Reelt:</p> <p>Status: Implementeret</p>
<p>8. Som kunde, kan jeg klikke på linket i mailen - fra sælgeren så jeg kan betale for min ønskede forespørgsel (custom carport)</p>	<p>Givet jeg er kunde, når jeg klikker på linket i mailen og ender på "betalingssiden" kan jeg trykke på "betal"-knappen og betale med min balance?</p>	<p>Lav en HTML-side med "betal"- og "afvis tilbud"- knapper.</p> <p>Opret rute og controller-metode til "betal"-knappen, der opdaterer status til 'paid' i databasen via Mapperen.</p> <p>Controlleren skal hente kundeinfo, sende til Mapperen og håndtere SQL-undtagelser.</p> <p>Mapperen skal hente kundens ordre fra databasen og opdatere status ved betaling.</p>	<p>Forventet: Stor</p> <p>Reelt:</p> <p>Status: Implementeret</p>



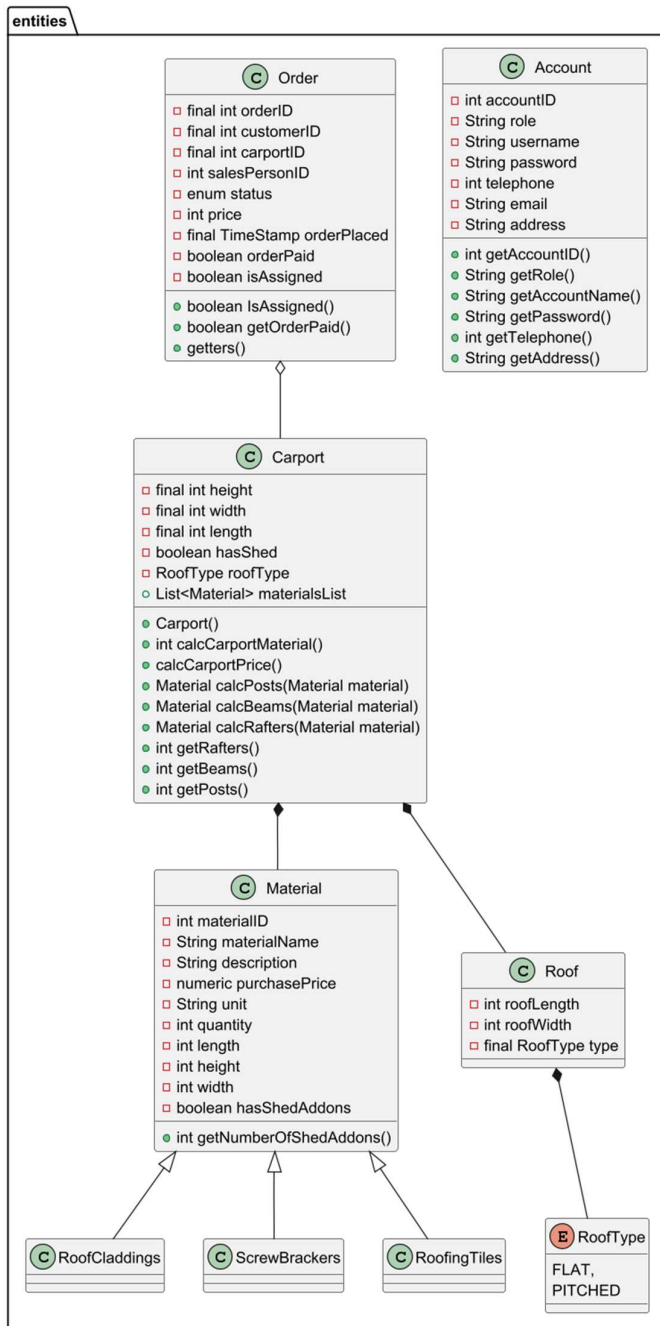
9. Som sælger, kan jeg modtage en email-notifikation så jeg kan se, når en kunde har oprettet en forespørgsel og behandle den	Givet jeg er sælger, når en kunde opretter sin forespørgsel, kan jeg modtage en mail og vælge at behandle deres forespørgsel	Brug den oprettede API-nøgle fra US-8 (opsætningen) til at sende mail  Opret en metode der henter relevante form-parametre om kundeordren  brug sælgerens mail i metoden.	Forventet: Stor  Reelt:  Status: Implementeret
10. Som admin kan jeg logge ind på beregningsappen-siden så jeg kan fjerne materiale jeg ikke ønsker værende tilgængeligt længere (f.eks. dørhåndtag)	Givet jeg er admin, når jeg er logger ind på beregningsappen-siden og kigger på tilgængeligt materiale kan jeg klikke på et givent materiale og fjerne det	Opret en HTML-side kun tilgængelig for admin med Thymeleaf og Bootstrap.  Implementer session-validering for at sikre, at brugeren er admin.  Tilføj et for-each loop i Thymeleaf til at vise alle materialer fra lageret.  Controlleren skal hente en liste fra Mapperen og sende den til HTML-siden via en Thymeleaf-rute.  Mapperen skal oprette en metode til at hente alle materialer fra materialetabellen og returnere en liste til Controlleren, med håndtering af SQL-undtagelser.	Forventet: Stor  Reelt:  Delvist implementeret

11. Som admin, kan jeg kan opdatere indholdet i mail-skabelon, som sælgeren sender til kunder.	Givet jeg er admin, Når jeg vælger at opdatere email-skabelonen kan jeg opdatere email-indholdet som sælgerne bruger	Opret redigerings knap for admin som giver adgang til email-skabelonen Tilføj redigeringsfunktion til skabelonen og opdater ændringerne når der trykkes "Gem ændringer"	Forventet: Stor  Reelt:  Status: Ikke implementeret
12. Som sælger ønsker jeg at kunne redigere al information i styklisten, så jeg kan opdatere navn, beskrivelse, materialer og mængder efter behov.	Når jeg navigerer til styklisten, skal jeg kunne se en redigeringsknap ved siden af hvert element. Når jeg klikker på redigeringsknappen, skal jeg kunne ændre navn, beskrivelse, materialer og mængder. Når jeg har foretaget ændringerne og klikker på "Gem", skal ændringerne gemmes og vises korrekt i styklisten. Hvis jeg forsøger at gemme uden at udfylde nødvendige felter, skal jeg få en fejlmeddelelse.	Opret og tilføj redigerings knap ved siden af hvert element i styklisten. Implementer en redigeringsfunktion, der gør det muligt at ændre navn, beskrivelse, materialer og mængder, når redigerings knappen klikkes. Sørg for, at ændringerne gemmes korrekt i databasen, når "Gem" knappen klikkes. Implementer fejlmeddelelser, hvis nødvendige felter ikke er udfyldt. Vis ændringer korrekt for alle brugere med adgang til styklisten.	Forventet: Stor  Reelt:  Status: Ikke implementeret

## Bilag 4. Udklip af Klassediagram version 1

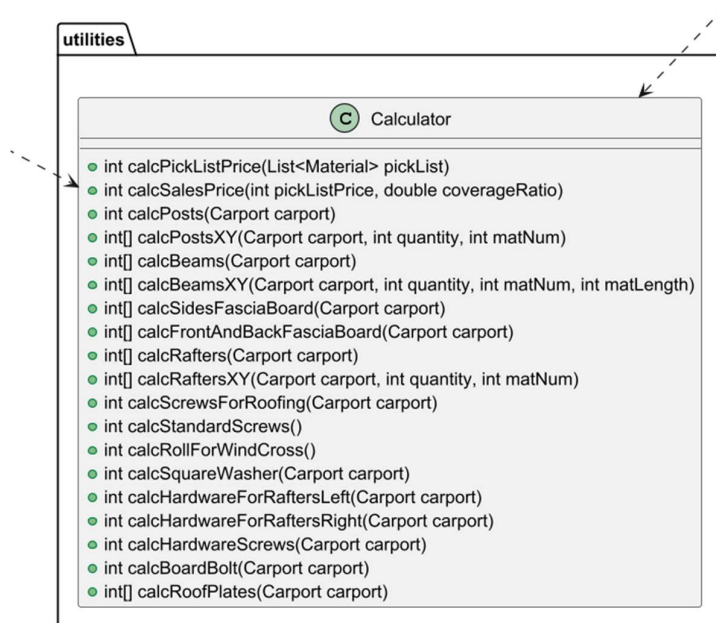


## Bilag 5. Udklip af Klassediagram version 2





## Bilag 7. Calculatorklassen fra klassesdiagram version 3



## Bilag 8. User acceptance test

<b>User acceptance test</b>	
<b>User Story</b>	<b>Godkendt/Ikke godkendt</b>
1. Som kunde skal jeg kunne udfylde en formular for en custom carport, så jeg kan bestille et tilbud	Godkendt
2. Som sælger, kan jeg vælge en kundes forespørgsel vedr. custom carport, og sende et pristilbud	Godkendt
3. Som sælger, kan jeg generere en stykliste så jeg kan sende den til kunden og lageret	Godkendt - delvist da styklisten genereres automatisk
4. Som admin, kan jeg logge ind på beregningssystemet-siden så jeg kan tilføje nyt materiale (f.eks. ny tagtype)	Godkendt - delvist da man kan gøre det som sælger
5. Som sælger, kan jeg bruge/oprette en mail-skabelon og indtaste kundens info, så jeg kan sende kunden et pristilbud / ordrebekræftelse og faktura	Godkendt
6. Som kunde, kan jeg klikke på linket i mailen - fra sælgeren så jeg kan afvise mit pristilbud	Godkendt
7. Som sælger skal jeg kunne redigere i tilbudsprisen, inden den sendes til kunden	Godkendt
8. Som kunde, kan jeg klikke på linket i mailen - fra sælgeren så jeg kan betale for min ønskede forespørgsel (custom carport)	Godkendt
9. Som sælger, kan jeg modtage en email-notifikation så jeg kan se, når en kunde har oprettet en forespørgsel og behandle den.	Delvist godkendt - indhold i e-mailen er ikke specificeret til tilbuddene
10. Som admin kan jeg logge ind på beregningsappen-siden så jeg kan fjerne materiale jeg ikke ønsker værende tilgængeligt længere (f.eks. dørhåndtag)	Godkendt - delvist da man kan gøre det som sælger

11. Som admin, kan jeg kan opdatere indholdet i mail-skabelon, som sælgeren sender til kunder	Ikke godkendt
12. Som sælger ønsker jeg at kunne redigere al information i styklisten, så jeg kan opdatere navn, beskrivelse, materialer og mængder efter behov	Ikke godkendt