Fog trælast (carport afd.)

Jesper Petersen cph-jp284@cphbusiness.dk

Claus Kramath cph-ck83@cphbusiness.dk

Github repo: https://github.com/CK2800/Fog

Udarbejdet 12.11.2018 – 19.12.2018

Indholdsfortegnelse

1 Indledning

Denne rapport handler om udviklingen og dokumentationen af software til Fog Trælast og byggecenter i Værebro ved Ølstykke.

Trælasten har siden midt-90erne brugt software til at beregne priser såvel som styklister til carporte i forskellige mål[1]. Softwaren er i sin tid udviklet af en medarbejder i virksomheden og fungerer i rimelig udstrækning stadig i dag.

Dog er der væsentlige mangler, som gør softwaren og brugen heraf sårbar, disse skal derfor afhjælpes med ny software, som dokumenteres i denne rapport.

1.1 Baggrund

Fog Trælast og byggecenter i Værebro er en del af Johannes Fog A/S. I trælasten kan man bl.a. købe standard byg-selv carporte. Disse carporte er designet efter nogle faste mål og kan eksempelvis bestilles direkte fra Fogs hjemmeside. Materialer og byggevejledning kan leveres direkte til kundens adresse.

Man kan også vælge at designe en carport med egne mål og øvrige krav. Disse kan ligeledes angives på Fogs hjemmeside. De specialdesignede carporte leveres på samme måde som standard carportene.

Hvis man designer sin egen carport, sendes dette design til Fog som en forespørgsel pr. e-mail. Hos Fog behandles den og man har oftest en telefonisk dialog med kunden, inden endelig bestilling.

Fælles for begge typer carporte er, at materialerne, der indgår, samt den endelige pris beregnes af software. Softwaren benyttes primært af medarbejdere i trælasten, og den er blevet demonstreret for os af Martin Kristensen, som er chef i trælasten.

Under demonstrationen fremkom en række umiddelbare mangler og krav, om end implicitte, til den nye software. F.eks. skal datagrundlaget kunne aktualiseres, så varenumre kan opdateres og brugerkonti nulstilles. Forretningsgangen ifm. specialdesignede carporte skal forenkles, så kunden kan visualisere designet omgående og hver afsendte forespørgsel registreres automatisk. Softwaren skal også give kunden mulighed for valg af tagbelægning og beklædning, dette krav var eksplicit.

Softwaren er formentlig udviklet til styresystemet Microsoft Windows 98 / XP, som ikke længere supporteres af Microsoft. Softwaren skønnes derfor sårbar, da den nok afvikles i kompatibilitetsmodus, hvilket måske vil fejle på et tidspunkt.

[1] Jf. videointerview: https://www.youtube.com/watch?v=OMatlvol_ns @13.25

2 Interessentanalyse

- **Martin (eller anden nøglemedarbejder i trælasten):**
  1. Har en stor viden om nuværende situation og ønsker om den fremtidige.
  2. Er – så vidt vides – den eneste, der har adgang til systemets administrative del.
  3. Har indgående kendskab til sædvanlige problemstillinger ifm. bestillinger af carporte, f.eks. skæve dimensioner, for store skure, for høj rejsning mv.
  4. Står for nuværende prisjustering og "mapning" mellem gl. varenumre og nye.
- **Alm. medarbejdere i trælasten:**
  1. Skal evt. kunne betjene systemet, så salg af carporte/træk af styklister, ændring af priser og pakning også kan ske, når Martin eller anden nøglemedarbejder ikke er på arbejde.
- **Kunder:**
  1. Jf. interview vil kunder få mulighed for valg af beklædningsmaterialer, tagbelægning mv.
- **Administration:**
  1. Evt. færre reklamationssager afledt af f.eks. misforstået telefonisk dialog.
  2. Bedre/korrekt overskudsgrad som flg. af korrekt beregning af f.eks. belægning/beklædning.
  3. Flere ordrer fordi systemet giver kunder mulighed for at gennemføre forespørgsler til ende før telefonisk kontakt, f.eks. valg af beklædning, belægning, dimensioner mv.
  4. Færre risici forbundet med opdatering / vedligehold af systemet (f.eks. v. sygdom/bortgang af nøglemedarbejder).
- **Direktion:**
  1. Skal beslutte om systemet har et fornuftigt ROI og dermed om det skal implementeres.
  2. Bedre bundlinje ved større salg/optimering af overskudsgrad.

▼Indflydelse	Medvirken ► HØJ	LAV
HØJ	• Martin og øvrige ngl. medarbejdere i trælasten.	• Direktion • Kunder*
LAV	• Administration • Alm. medarbejdere i trælasten	• Aktionærer

* = Kunder er vigtige aktører i det nuværende og fremtidige system. De har lav medvirken men høj, om end indirekte, indflydelse, idet de ikke aktivt inddrages i udviklingen. Deres feedback/henvendelser er dog medvirkende til at definere krav til den del af systemet, som de bruger (hjemmesiden).

3 SWOT-analyser

Fog ®	Positive	Negative
Interne	Strengths: • Klar over at nuværende situation er sårbar. • Ngl.medarbejder med indgående kendskab til situationen og problemområdet. • Carporte i bedre kvalitet end Silvan/Bauhaus og evt andre.	Weaknesses: • En eller få medarbejdere kan betjene systemet. • Systemet er uddateret og måske i fare for ikke at kunne fungere inden for overskuelig fremtid. • Systemet er i 3 fragmenter – frontend til kunder, backend (Quickbyg) og et eksternt system til oprettelse af styklister. • Kunden oplever det evt. besværligt og ufuldstændigt at lave forespørgsler.
Eksterne	Opportunities: • Større salg som flg. af større brugervenlighed. • Ny/bedre løsning i websitet kan tiltrække flere kunder. • Bedre backend mindsker evt. flaskehalse hvis ngl.medarbejder er fraværende. • COTS system som kun kræver licens eller engangssum.	Threats: • Konkurrenter udvikler tilsvarende eller bedre løsning hurtigere. • Kunder vender sig mod billige standardcarporte. • Budget byggemarkeder har flere butikker => mere tilgængelige. • COTS system viser sig at have begrænsninger ift. krav som måtte opstå senere.

Projektet	Positive	Negative
Interne	Strengths: • Dedikeret ngl.medarbejder tæt på problemområdet med masser af viden, både som it-bruger og håndværker/trælastmand. • Fog har tidligere fået udviklet et system, så de er klar over, at der er en gevinst ved nyt system.	Weaknesses: • Konflikt: Man kan godt lide enkeltheden i Quickbyg samtidig med at man vil have et system som potentielt kan give mere vedligehold af varer, priser mv. • Der er skitseret en række udfordringer, f.eks. opdatering af varenumre, priser og tilføjelser af flere varer, men kun ytret 2 konkrete krav. • Der er kun foretaget interview af 1 person, så det er uklart hvordan resten af organisationen forholder sig. • Et tidligere projekt er skrinlagt, er man urealistisk i sine mål? Man ved hvad man har, men ved man hvad man vil?
Eksterne	Opportunities: • COTS ERP-system kan evt. indfri krav hurtigt, ved få justeringer/tilføjelser?	Threats: • ERP system er måske over målet/for indviklet til at medarbejderne gider bruge det. • Modvillighed blandt medarbejdere kan skabe splid og skade virksomheden.

		• Markedet ændres i en sådan grad, at systemet ikke kan bruges. • Klimaforandringer medfører ændringer i byggeregulativer som fordrer konstruktionsændringer i carporte. • Har vi egne biler i fremtiden?

JC	Positive	Negative
Interne	Strengths: • Lille team med korte beslutningsveje. • Dedikerede og debatlystne teammedlemmer med en del erfaring. • Kan både levere backend, frontend og designe GUI. • Er klar over udfordringerne med at være få og er lykkedes fint med tidligere projekter.	Weaknesses: • Få teammedlemmer kan give unuanceret billede af problemområdet og dermed løsning. • Svært at gennemføre flertalsbeslutninger ved uenighed.
Eksterne	Opportunities: • Flere teammedlemmer ville bidrage til mere nuancerede diskussioner og større sikkerhed i gennemførsel af opgaver.	Threats: • Uenighed bliver så alvorlig at teamet splittes op. • Sygdom kan hurtigt forsinke projektet.

4 Formål

At give kunden bedre mulighed for at kunne visualisere sin carport.
Bedre integration af forespørgselsdata i tilbudsberegneren.
At sikre systemets drift på tværs af medarbejdere.
At defragmentere systemet så det er et samlet hele.

5 Mål

På alle forespørgsler er det muligt at vælge beklædning og tagbelægning.
På alle 'vis carport' sider vises 3D tegning med korrekte farver, belægninger og beklædninger.
Alle forespørgsler gemmes på en måde, så behovet for gentagne indtastninger fjernes.
Alle systemets brugerkonti sikres således, at de kan nulstilles.
Alle data vedr. carporte skal komme samme sted fra, så anomalier, f.eks. ift. varenumre, undgås.

6 Teknologivalg

- NetBeans IDE 8.2 (Build 201609300101) opdateret til Patch 2
- Java: 1.8.0_192
- Runtime: Java(TM) SE Runtime Environment 1.8.0_192-b12
- JDBC (bundle ver. 8.0.12)
- MySQL Workbench 8.0
- Maven (bundled w. NetBeans) v. 3.0.5
- Apache Tomcat 8.0.27.0
- MySQL RDBMS v 14.14 Distribution 5.7.24
- Bootstrap v. 4

Løsningens MySQL RDBMS er hostet på en linux maskine på Digital Ocean, www.digitalocean.com. Maskinen kører Ubuntu 16.04.1 baseret på Linux 4.4.0-138-generic GNU/Linux kernel.

7 Krav

Afledt af de observationer, som er nævnt i baggrunden for projektet, er visionen at få et system, som, udover at tilbyde de samme muligheder som det nuværende, også giver mulighed for aktualisering af datagrundlaget. Videre skal kunden have mulighed for bedre at kunne visualisere sit carportdesign samt vælge diverse beklædnings- og belægningstyper. Arbejdsgangen vedr. forespørgsler på carporte skal også forenkles.

Vi har således taget udgangspunkt i demonstrationen for at afdække brugsmønstre svarende til de nuværende funktionelle krav. Kravenes / brugsmønstrenes kompleksitet er samtidig vurderet.

7.1 Funktionelle krav

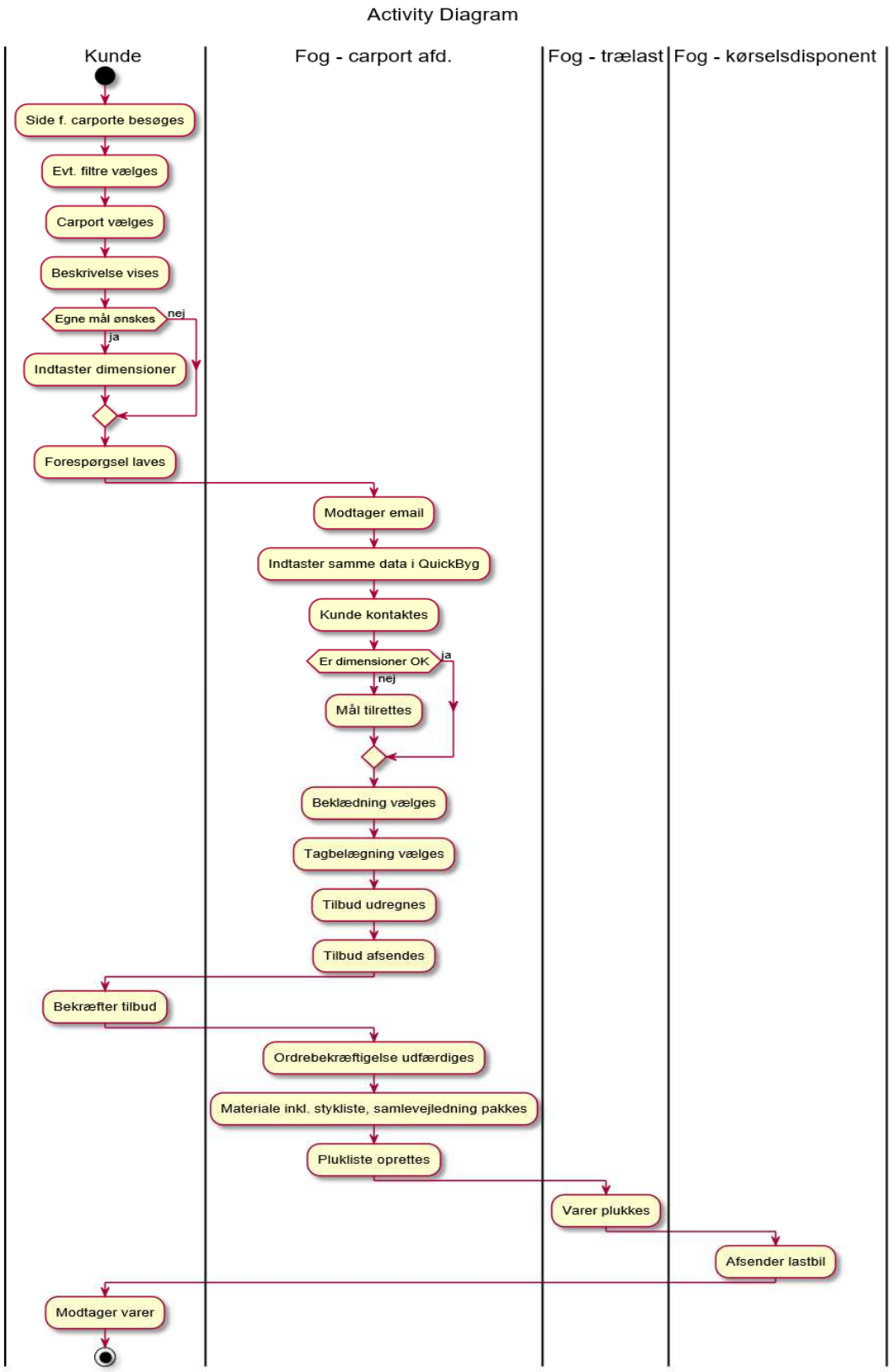
Navn	Kompleksitet	Type
Filtrer carport	Simpel	Aflæsning
Hent / vis tegning	Særdeles kompleks	Beregning
Vis carport (billede af)	Simpel	Aflæsning
Vis leveringspris	Særdeles kompleks	Beregning
Design egen carport	Simpel	Opdatering
Afsend carportforespørgsel	Simpel	Opdatering
Åbn carportforespørgsel	Simpel	Aflæsning
Besvar carportforespørgsel	Simpel	Opdatering
Konfigurer carport	Simpel	Opdatering
Administrer kunde	Simpel	Opdatering
Beregn stykliste	Særdeles kompleks	Beregning
Rediger dækningsgrad	Simpel	Opdatering
Rediger hjælpetekst	Simpel	Opdatering
Tilføj vare fra databasen	Simpel	Opdatering
Udskriv tegning	Simpel (hvis tegning er dannet)	
Se beskrivelse	Simpel	Aflæsning
Administrer varer	Middel	Opdatering
Nulstil brugerkonto	Middel	Opdatering
Administrer brugerkonti	Middel	Opdatering
Administrer samlevejledninger	Middel	Opdatering
Udskriv ordredokumenter	Middel	Aflæsning
Vis fejl i carportforespørgsel	Særdeles kompleks	Beregning
Se ordrestatus	Simpel	Aflæsning

Log ind/ud	Simpel	Aflæsning

Denne liste af nuværende krav kan analyseres lingvistisk efter navneord for at frembringe en liste med kandidatklasser, som danner grundlag for domænemodellen, se afsnittet 8 Domænemodel og ER diagram.

Af listen over funktionelle krav er især arbejdsgangen vedr. carportforespørgsler interessant, da den findes i 3 af de funktionelle krav. Af demonstrationen fremgår det, at en kunde laver en forespørgsel vedr. en carport og sender den til Fog. Fog modtager forespørgslen som en e-mail. Denne åbnes og forespørgslens data indtastes i Fogs eget system; Quick-byg, som bl.a. bruges til at beregne materialer og pris.

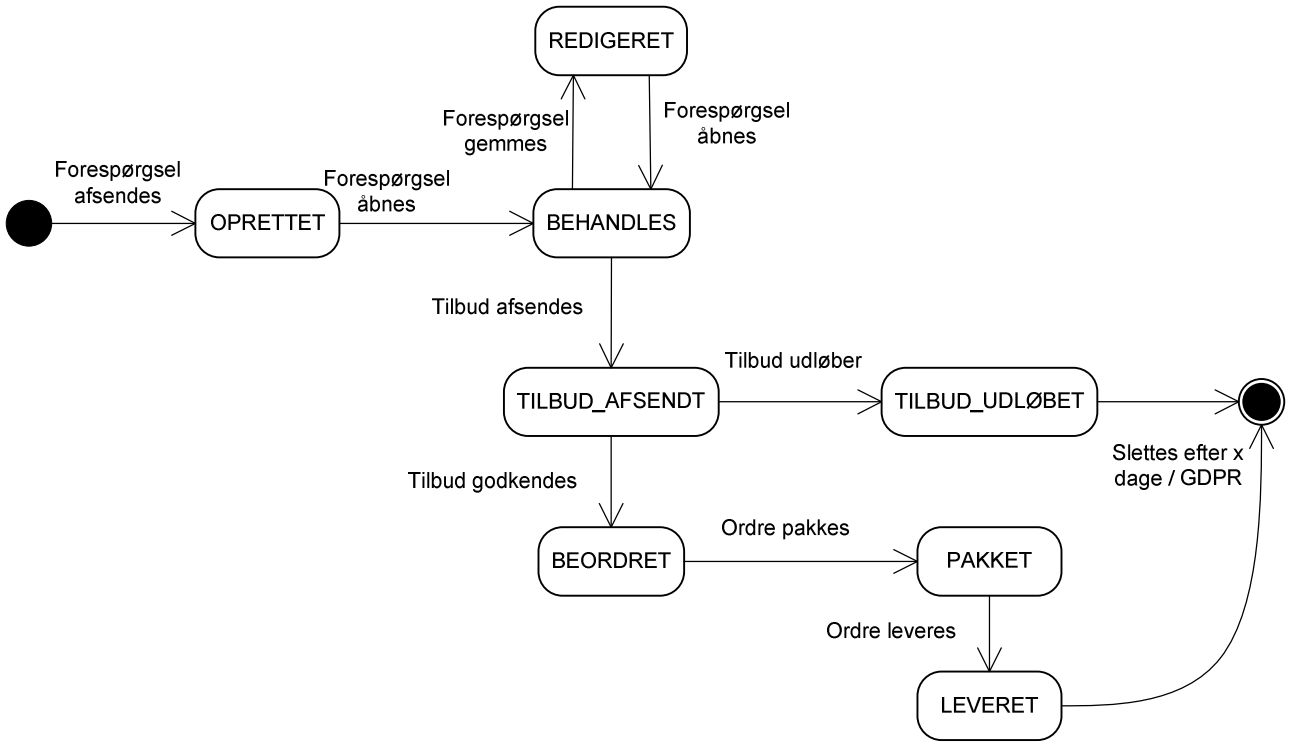
Arbejdsgangen kan illustreres vha. flg. aktivitetsdiagram:



Figur 1 – Aktivitetsdiagram for Fog Carporte AS-IS.

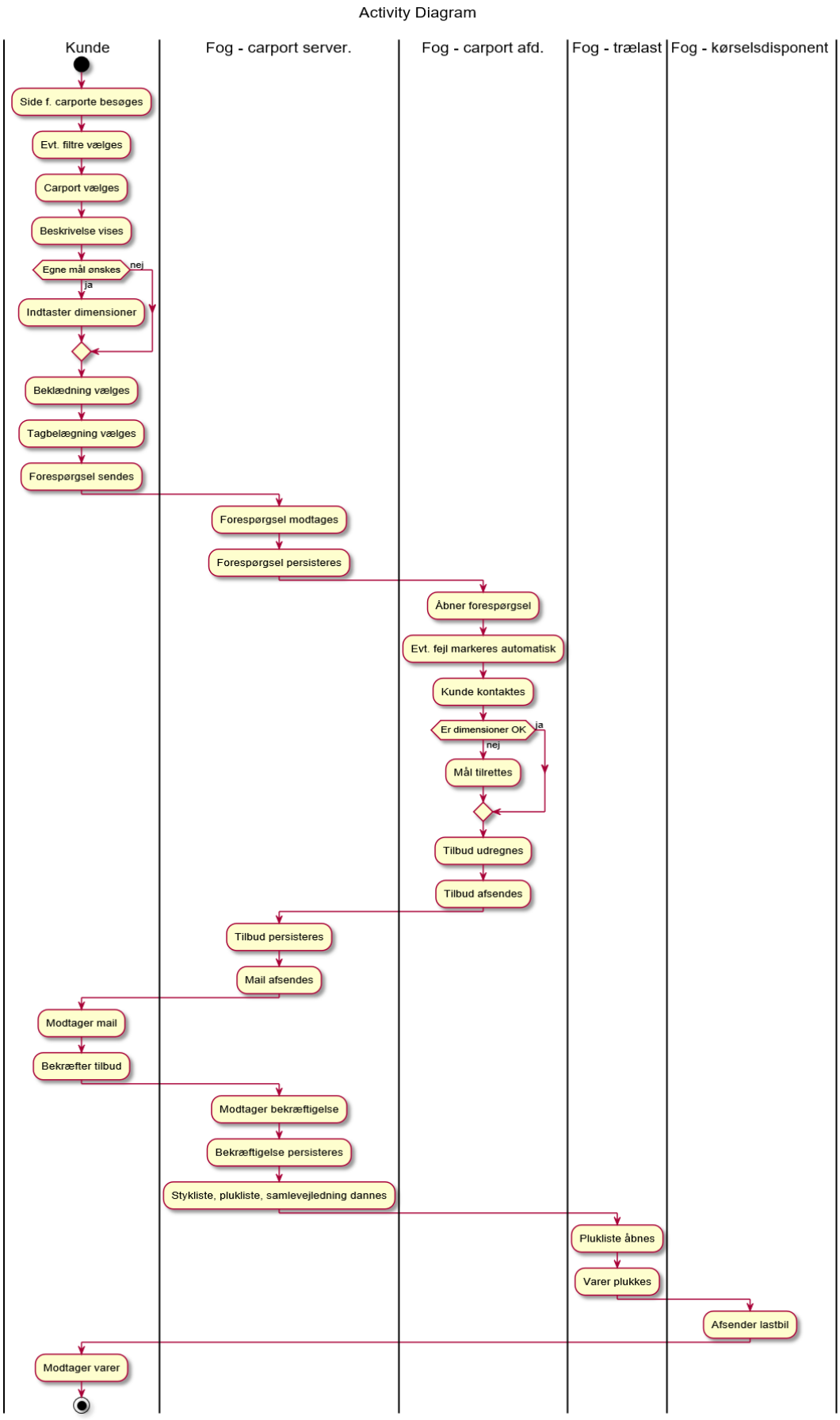
Medarbejderen i trælasten skal foretage mange skridt, før et tilbud kan afsendes. Nogle af disse skal kunden fremover selv foretage, inden han/hun afsender en forespørgsel, nemlig valg af tagbelægning og beklædning. Andre skridt kan understøttes af it.

Det er endvidere vanskeligt at afgøre, hvordan medarbejderen holder styr på, hvor langt en forespørgsel er kommet i forløbet, da kommunikationen tilsyneladende udelukkende foregår pr. e-mail og telefon. Måske er en forespørgsel kun en forespørgsel indtil et tilbud oprettes, tilbuddet bliver senere evt. til en ordrebekræftelse. Denne uklarhed skal det fremtidige system eliminere. Vi har derfor opstillet flg. tilstandsdiagram for en forespørgsel for at skabe overblik:



Figur 2 – Tilstandsdiagram for forespørgsel.

De tilstande, vi ønsker at gemme data for, må persisteres i databasen, f.eks. som en attribut på en forespørgselstabel. Det kunne eksempelvis være, hvornår tilbuddet blev afsendt eller hvilken dato, ordren blev afgivet. Man kunne også anskue disse tilstande som entiteter, hvilket ville medføre en tabel for tilbud og en tabel for ordre i databasen. Uanset hvordan forespørgslen og dens tilstandsskift implementeres, skal systemet resultere i flg. aktivitetsdiagram. Læg mærke til, hvor mange opgaver det nu påhviler serveren at håndtere og hvor forenklet arbejdsgangen for medarbejderen i carport afd. er.



Figur 3 – Aktivitetsdiagram for Fog Carporte TO-BE.

De 3 funktionelle krav/brugsmønstre, som tilsammen udgør arbejdsgang vedr. carportforespørgsel, er her beskrevet som Scrum User Stories:

ID	Navn	Imp	Est	How to demo	Notes	Derfor!
US-6	Afsend carport forespørgsel	50	2	Unit test med data for en forespørgsel oprettes, data gemmes i databasen, data fra senest oprettede forespørgsel hentes fra databasen og højde, bredde og længde sammenlignes, disse skal være ens.	Kræver DB, unit test.	I: Ellers tabes mange salg. E: Create
T-29	Opret JUnit som tester funktionaliteten.					
T-30	Opret SQL til indsættelse af forespørgsel i db.					
T-31	Opret DAO som udfører indsættelsen.					
T-32	Opret tabellerne kunde, postnr, tag, skur, forespørgsel i databasen.					
T-33	Opret DTO til forespørgsel.					
T-34	Connector klasse til databasen.					
US-7	Åbn carport-forespørgsel	50	2	Åbn webside med liste over forespørgsler, en forespørgsel klikkes og webside med formular med forespørgslens indhold vises	Kræver DB.	I: Ellers er der ingen kunder. E: Read
T-42	FrontController opsættes.					
T-43	Command pattern implementeres.					
T-44	Side med liste over forespørgsler oprettes.					
T-45	Side som viser indhold af en forespørgsel.					
T-46	Div. tilpasninger i ForesporgselDAO.					
T-47	UnknownCommand, ShowSingleRequestCommand og ShowRequestsCommand laves.					
T-48	Implementering af eget Exception system.					
T-49	Filer med konstanter til Commands og Pages.					
T-50	Facade i data lag.					
T-51	Try catch i commands.					
T-52	Links fra siden med listen til siden med enkelt forespørgsel.					
T-53	Link fra siden med enkelt forespørgsel tilbage til siden med listen.					
US-16	Besvare carport forespørgsel	45	1	Åbn webside med carporte, klik på linket 'Carport med egne mål', tag vælges og carportens mål indtastes. Indtast mail adresse der er adgang til under test. Knappen 'Send forespørgsel' klikkes. Log ind, åbn personlig webside. Link til webside med liste over forespørgsler vælges, den netop oprettede forespørgsel klikkes og webside med formular med forespørgslens indhold vises. Svartekst indtastet i svarfeltet og systemet danner e-mail og vedhæfter automatisk relevante dokumenter. Tjek mailbox for svarmail og vedhæftninger, tjek at forespørgsel skifter status til 'besvaret' i listen over forespørgsler.		I: En obligatorisk del af salgsprocessen. E: Simpel funktion
Ingen tasks – da denne User Story ikke er planlagt.						

Den fulde backlog kan ses i bilagene.

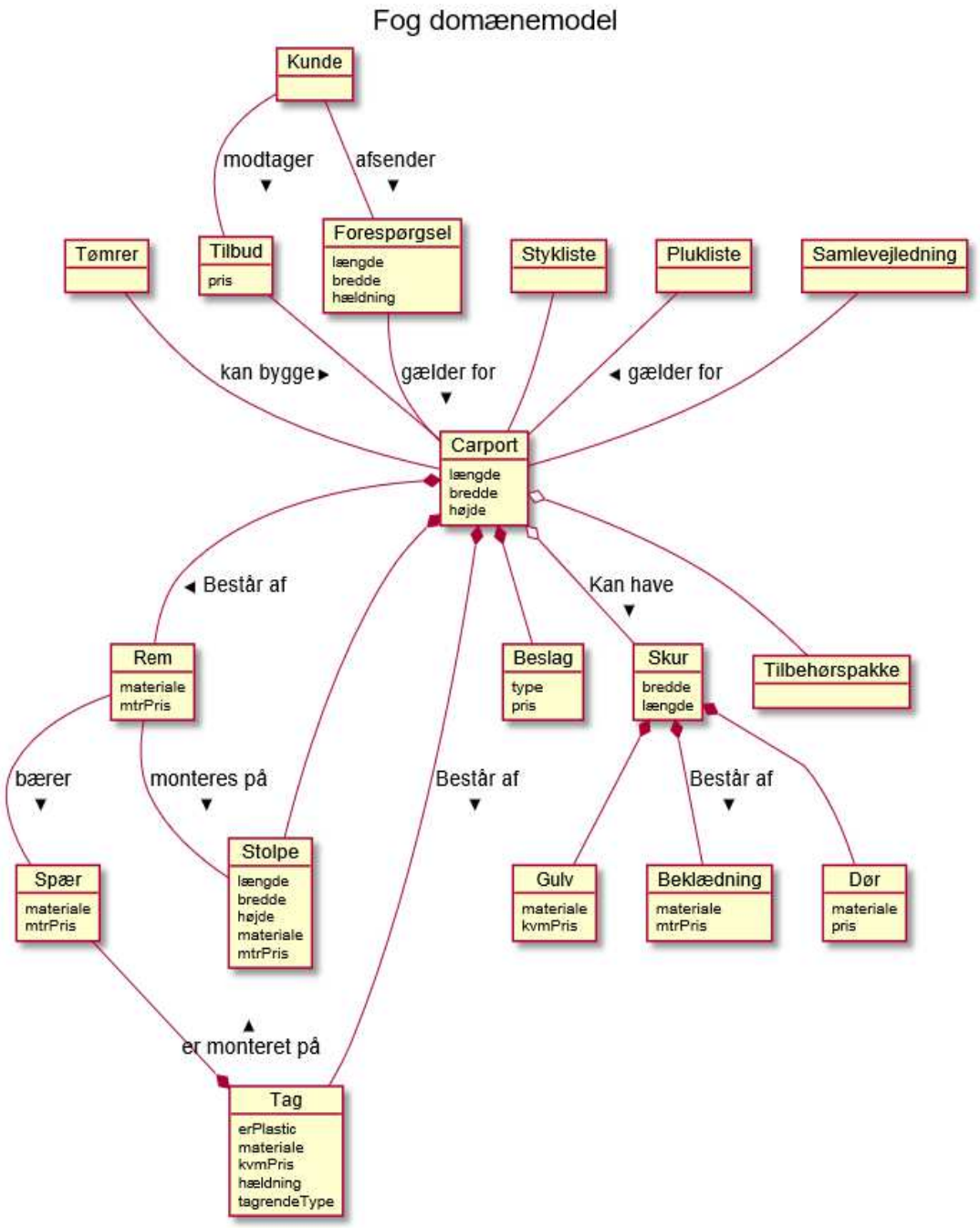
8 Domænemodel og ER diagram

Som nævnt i afsnittet 7 Krav, har vi afledt en række kandidatklasser fra de funktionelle krav. Blandt disse krav findes også en række beregningsfunktioner med høj kompleksitet. Disse har vi forsøgt at deltaljere vha. pseudokode, se afsnittet 14.9 Beregning af materialer., ligesom vi har gransket interviewet nøjere for at opnå en dybere forståelse af problemområdet med fokus på carporte og deres bestanddele.

Således er vi kommet frem til flg. kandidatklasser:

Carport	*Tegning**	*Pris **	*Design**	*Forespørgsel*
Kunde	*Stykliste*	*Dækningsgrad**	*Hjælpetekst**	*Vare**
*Beskrivelse**	*Brugerkonto**	*Samlevejledning*	*Faktura**	*Plukliste*
*Fejl**	*Ordrestatus**	*Ordre**	*Rem*	*Stolpe*
Spær	*Tag*	*Skur*	*Beslag*	*Gulv*
Beklædning	*Dør*	*Tilbehørspakke*	*Tømrer*	*Tilbud*

De fleste af disse kandidater er vist på domænemodellen nedenfor. Visse (markeret med *) er udeladt, dels fordi de er beregninger og dels fordi de tænkes at være attributter på andre kandidatklasser, f.eks. Ordrestatus, som snarere bliver til en status attribut på Ordre klassen og Design, som i stedet er en række attributter på Forespørgsel-klassen.

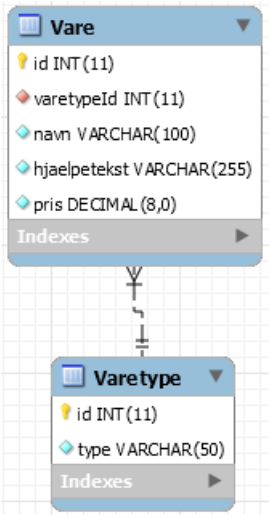


Figur 4 – Domænemodel over problemområdet.

Domænemodellen skal skabe et overblik over problemområdet og skal ikke være et udtryk for endelige designklasser. Det er meningen, at den skal illustrere den forståede opfattelse af det område, softwaren skal løse et problem i. Således er den et udgangspunkt for videre diskussion og forfinelse mod det første design-klasse diagram og databasen. Begge disse forfines i iterationer gennem hele udviklingsprocessen.

Scrum processen har bl.a. til formål, sammen med kunden, at identificere de user stories, der giver mest værdi for kunden, således at disse kan udvikles først. Dette vil også medføre nye krav efterhånden som del-systemer bliver færdige. Således skal man ikke forsøge at afdække alle behov fra starten (vandfaldsmodellen), da disse er foranderlige.

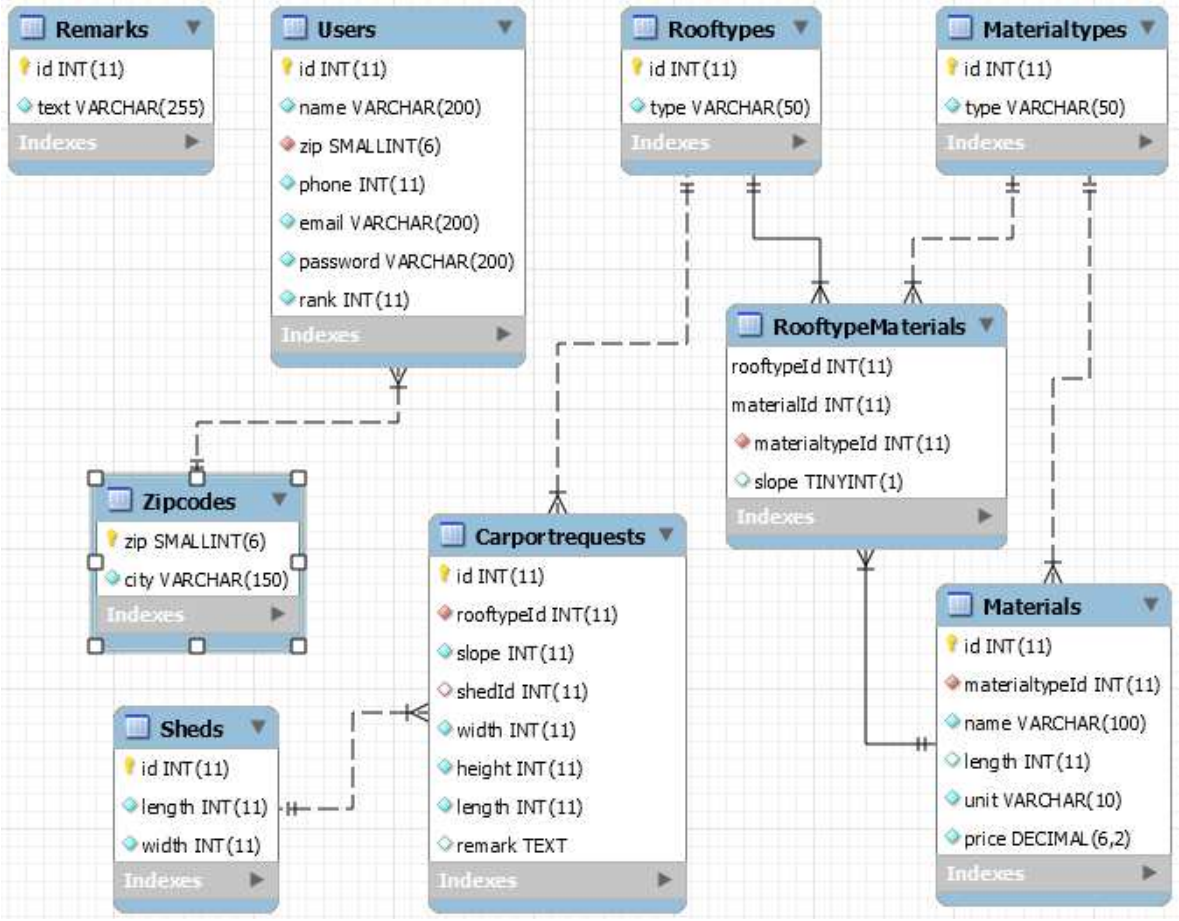
For at kunne udvikle de user stories, der giver mest værdi, bliver databasen også udviklet iterativt. Således havde vores første ER diagram blot 2 tabeller.



Figur 5 – Databasens første layout.

Af det beskedne design kan man udlede, at entiteterne rem, stolpe, skur, tag mv. er samlet i en ny entitet; Vare. For at skelne mellem diverse varer og deres brug, er der indført en relation til tabel med typer; Varetype.

Siden er vores database blevet udvidet, i takt med at flere user stories er lagt i sprints. Af tidsmæssige årsager er vi ikke blevet færdige med at udvikle på vores database. Der henstår bl.a. at binde kunden til en carport forespørgsel, idet en carport forespørgsel altid initieres af en kunde og slutteligt, formentlig, skal sendes til denne. Databasens layout er også skiftet fra dansk til engelsk, som led i en omfattende ny navngivning af vores klasser.



Figur 6 – Databasen er udviklet iterativt gennem Scrum Sprints, navne er skiftet fra dansk til engelsk.

Af det seneste ER diagram ses, at enkelte af domænemodellens klassekandidater har fundet vej til databasen. Andre er blevet til attributter på entiteterne, f.eks. 'Design', som reelt blot er længde, højde og hældning på carportforespørgslen samt valget af skur og tagtype.

8.1 Tabellerne.

Her følger en beskrivelse af de oprettede tabeller.

Navn	Beskrivelse
Users	Lagrer information om brugerne. Vi har her overvejet datatypen for zip (postnumre), idet der blot er behov for at kunne lagre værdier < 10000. For at lagre 9999 kræves 14 bits. Oprundet til 2 bytes kan MySql lagre dette som en smallint. Zip er fremmednøgle og refererer til primærnøglen i tabellen Zipcodes. Email er unik og kunne som sådan have været en primær nøgle. Vi har dog valgt en surrogat nøgle, fordi en nøgle skal identificere rækken og ikke ændres med tiden. En ændring af e-mail-adressen ville være et almindeligt forekommende brugsmønster.
Zipcodes	Lagrer danske postnumre. Da der ikke må være 2 ens, har vi ladet zip være fremmednøglen i stedet for et autogenereret id.
Materials	Lagrer varer. Fremmednøglen materialtypeId indikerer hvilken materialetype, materialet har, f.eks. rem, spær, tagrygbelægning, tagfladebelægning. Attributten price har datatypen DECIMAL(6,2), hvilket betyder, at prisen kan gå til 9999,99 – 6 betydende cifre, 2 efter kommaet.
Materialtypes	Bruges til at inddele materialerne i forskellige typer. Tabellen MaterialTypes er nødvendig for at softwaren eksempelvis ved beregning af materialer til tegltag, kan sondre mellem at beregne materialer til tagfladen hhv.

	tagryggen.
RooftypeMaterials	Lagrer hvilke materialer der indgår i forskellige tagtyper. Denne tabel løser n-m relationen mellem Rooftypes og Materials. Primærnøglen udgøres af de 2 fremmednøgler; rooftypeId og materialId. Således kan et materiale kun indgå i en tagtype 1 gang. De 2 fremmednøgler er identificerende relationer. Tabellen har også reference til MaterialType, se mere om denne relation i afsnittet 8.1.1.2 Tabellen RooftypeMaterials.
Rooftypes	Beskriver de tagtyper, der kan vælges, når carport designs.
Carportrequests	Lagrer de data der omhandler en forespørgsel på en carport. Valget af tagtype og skur resulterer i referencer til de 2 tabeller; Rooftypes og Sheds.
Sheds	Indtaster kunden dimensioner for et skur, lagres de her. Da et skur er valgfrit, er relationen 0..1 : 1 mellem Sheds og Carportrequests. Tabellen er oprettet, fordi alternativet havde været at lade disse attributter være i Carportrequests, med tomme felter hver gang en forespørgsel oprettes uden skur.
Remarks	Lagrer instruktioner for bygningsdele.

8.1.1 Begrænsninger.

I databasen er der en række begrænsninger, 'constraints', som sikrer, at dataintegriteten overholdes.

8.1.1.1 Tabellen Carportrequests.

```
CREATE TABLE Carportrequests(
    id int PRIMARY KEY AUTO_INCREMENT,
    rooftypeId int NOT NULL, -- id for tagtypen.
    /*tagId int NOT NULL, -- carport har altid et tag.*/
    slope int NOT NULL default 0, -- hældning er 0 hvis intet andet angives.
    -- shedId int, -- carport har ikke altid et skur.
    width int NOT NULL,
    height int NOT NULL,
    length int NOT NULL,
    remark text,
    /*CONSTRAINT fk_Carportrequests_Sheds
    FOREIGN KEY(shedId)
    REFERENCES Sheds(id)
    ON DELETE SET NULL, -- slettes skuret, skal referencen til det fjernes.*/
    CONSTRAINT fk_Carportrequests_Rooftypes
    FOREIGN KEY (rooftypeId)
    REFERENCES Rooftypes(id)
    ON DELETE NO ACTION -- en tagtype, som refereres af en carport forespørgsel, må ikke slettes.
);
```

Figur 7 – SQL sætning for oprettelse af tabellen Carportrequests.

Begrænsningen fk_Carportrequests_Rooftypes sikrer, at hvis en carportforespørgsel peger på en tagtype, så kan tagtypen, dvs. tuplen i Rooftypes, ikke slettes.

8.1.1.2 Tabellen RooftypeMaterials.

```
-- Info om hvilke materialer der indgår i en tagtype. F.eks. røde rygningssten og røde belægningssten til rødt tegltag.
-- Tabellen findes, fordi en tagtype kan have flere materialer, og disse kan være af forskellig materialetype. Koden må
-- afgøre hvordan materialeantallet beregnes. V. flade tage kan man se efter materialets dimension, v. tag m. rejsning
-- må man kigge efter materialets materialetype, f.eks. enten rygsten eller tagsten, og så beregne antallet.
CREATE TABLE RooftypeMaterials(
    rooftypeId int,
    materialId int,
    materialtypeId int NOT NULL,
    slope boolean,
    -- PRIMARY KEY(rooftypeId, materialTypeId), -- kombination af tagets type og materialets type sikrer, at der f.eks. kun
    -- er 1 belægningstype for hver tagtype. MEN: Duer ikke v. plasttag mv. pga flere dimensioner for samme materialetype.
    PRIMARY KEY(rooftypeId, materialId),
    CONSTRAINT fk_RooftypeMaterials_Rooftypes
    FOREIGN KEY(rooftypeId)
    REFERENCES Rooftypes(id)
    ON DELETE CASCADE, -- Hvis tagtype slettes, slettes information om tagtypens materialer også.
    CONSTRAINT fk_RooftypeMaterials_Materialtypes
    FOREIGN KEY(materialtypeId)
    REFERENCES Materialtypes(id)
    ON DELETE CASCADE, -- Hvis materialetypen slettes, skal information om tagtypens materiale også slettes.
    CONSTRAINT fk_RooftypeMaterials_Materials
    FOREIGN KEY(materialId)
    REFERENCES Materials(id)
    ON DELETE CASCADE -- Hvis materialet slettes, skal denne information også slettes.
);
```

Figur 8 - SQL sætning for oprettelse af tabellen RooftypeMaterials.

Begrænsningen fk_RooftypeMaterials_Rooftypes sørger for, at hvis en tagtype slettes, slettes også informationer om hvilke materialer, der indgik i den.

Begrænsningen fk_RooftypeMaterials_Materialtypes sørger for, at hvis en materialetype slettes, så slettes også informationen om, at materialetypen indgik i tagtypen. Denne relation er reelt overflødig, da en materialetype kan fås gennem relationen til Materials-tabellen. Tanken med denne relation var at implementere en begrænsning i databasen, så vi undgik at kunne lægge røde tegl på sorte tegltage ved at lade denne relation indgå i primærnøglen, se udkommenteret linje i figur 8. Men da flade tage har flere varer af samme type, f.eks. plastic tag i flere længder, duer denne kombination ikke. Begrænsningen må implementeres i softwaren.

8.1.1.3 Tabellen Users.

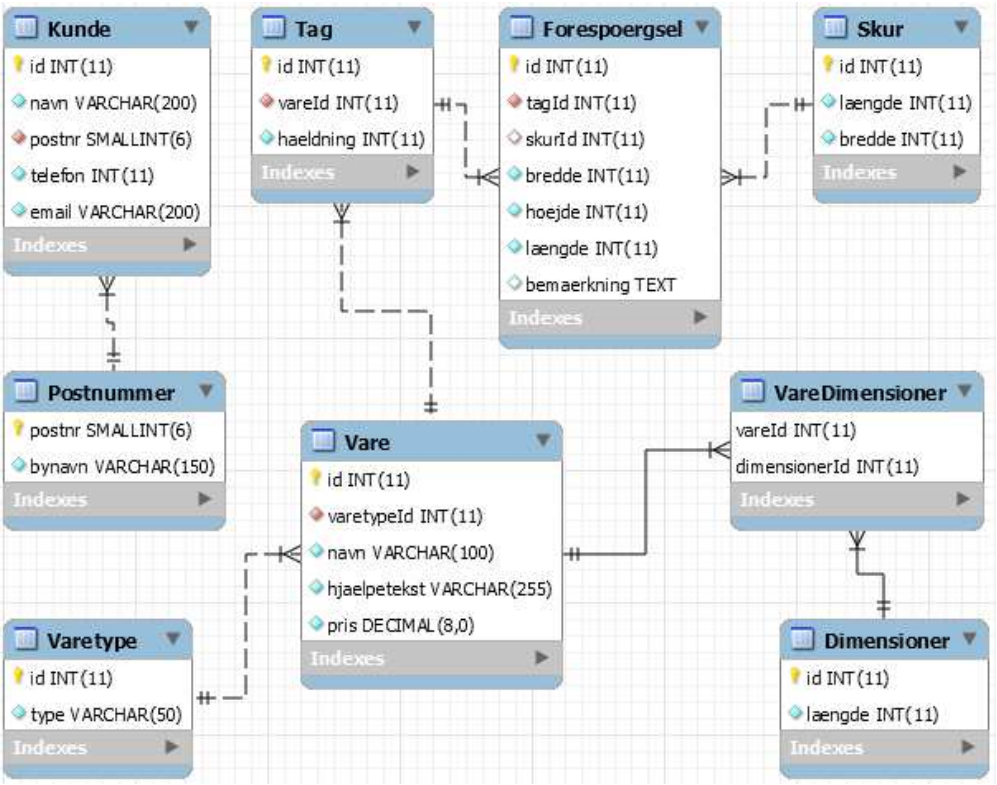
```
CREATE TABLE Users(
    id int primary key auto_increment,
    `name` varchar(200) not null,
    zip smallint not null, -- 14 bits til 9999 ==> 16 bits eller 2 bytes ==> smallint.
    phone int not null,
    email varchar(200) not null unique,
    `password` varchar(200) not null ,
    rank int not null DEFAULT 5,
    CONSTRAINT fk_Users_Zipcodes
    FOREIGN KEY (zip)
    REFERENCES Zipcodes(zip)
    ON DELETE NO ACTION -- postnumre i brug i denne tabel må ikke slettes.
);
```

Figur 9 - SQL sætning for oprettelse af tabellen Users

Begrænsningen fk_Users_Zipcodes sørger for, at postnumre i brug i Users ikke slettes. I virkeligheden er der sjældent brug for at slette postnumre.

8.2 Videre optimering.

Attributten length angiver varens/materialets længde. Da ikke alle varer har længder, kan det diskuteres, om denne tabel er normaliseret tilstrækkeligt, da mange varer blot har nulls i denne kolonne. Vi har tidligere haft en tabel i databasen for håndtering af netop dette, dog har vi, pga. manglende tid, tilbagerullet så der er en vare for hver længde den findes i.

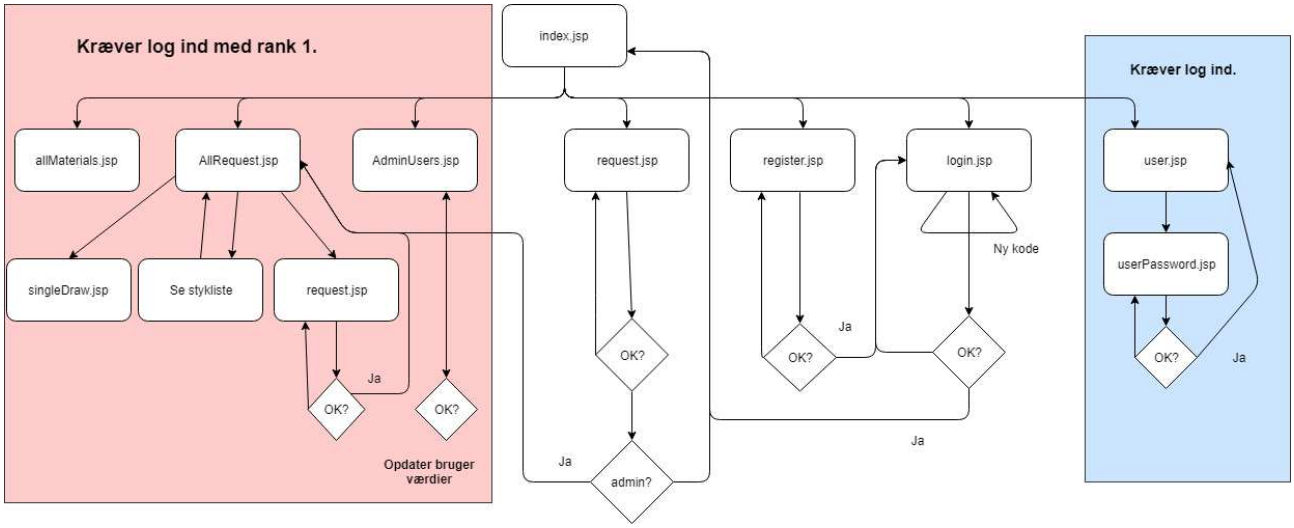


Figur 10 - Databasen med tabeller for varers dimensioner.

9 Navigationsdiagram

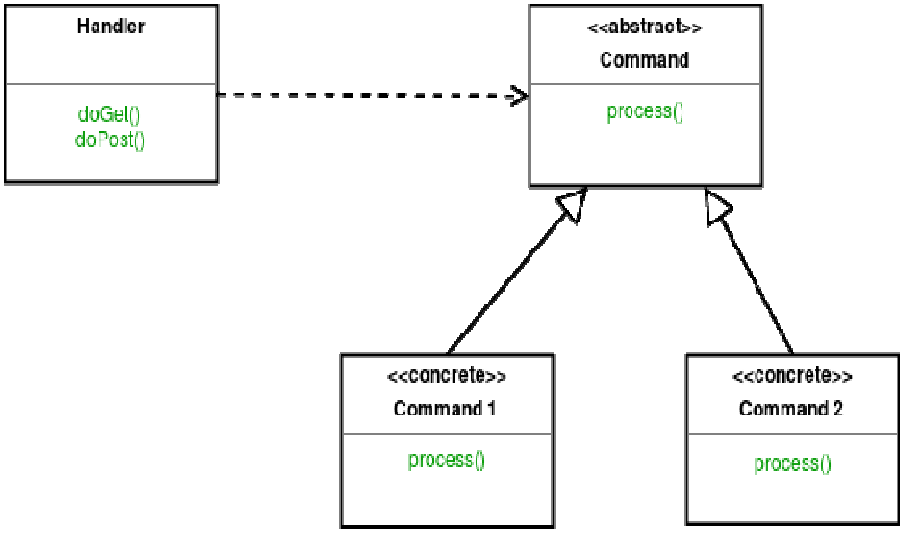
Webapplikationen består af en navigationsbar på alle sider. Fra den vil der være forskellige muligheder for navigation afhængig af, om man er logget ind samt med hvilken rank.

Der er således områder, der kun kan benyttes, hvis man er logget ind, se det blå felt i figur 11. Har man administrative rettigheder, kan man også nå siderne i det røde felt. Sider i det hvide område kan tilgås af alle.



Figur 11 - Navigationsdiagram med opdelte brugerområder.

Navigationen er implementeret med FrontController pattern, som en central servlet, der håndterer alle requests og, baseret værdien af parameteren 'command', kalder et korresponderende Command-objekt's execute()-metode. Command-objektet har ansvaret for at kalde de lavere liggende lag for at udføre sin opgave og derefter for at returnere den korrekte side til brugeren.

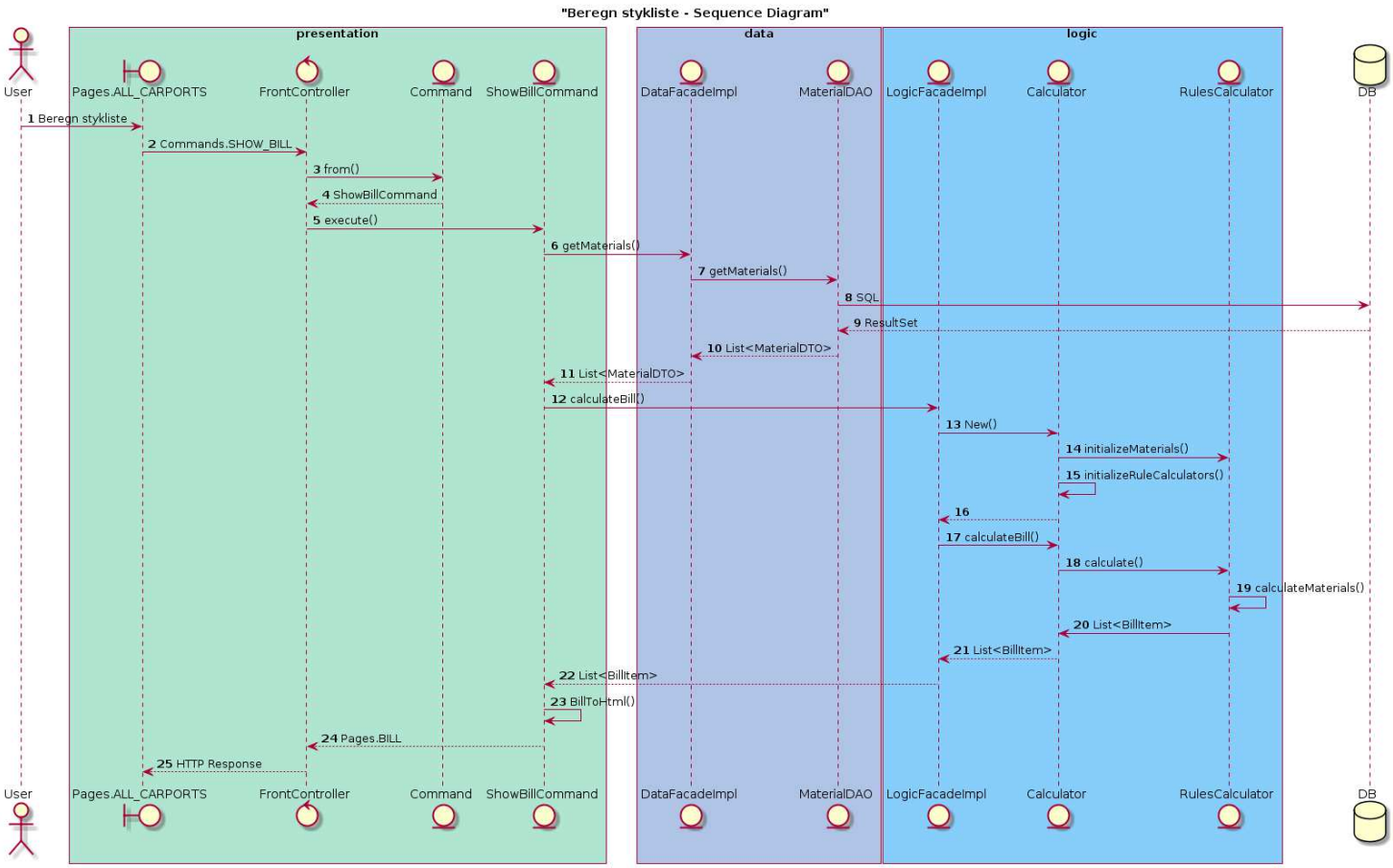


Figur 12 - FrontController pattern med Command objekter

For at forenkle udviklingen, har vi samlet en række konstanter for hvert Command og hver resulterende side i klasserne Commands og Pages. Dette er også med til at sikre, at der er konsistens mellem jsp-siderne og tilgængelige commands og omvendt og vi undgår stavefejl. Se f.eks. figur 13 pkt. 2 herunder.

10 Sekvensdiagrammer

10.1 Beregning af stykliste.



Figur 13 - Sekvensdiagram for beregning af stykliste.

I figur 13, beregning af stykliste, kan man tydeligt se den lagdelte arkitektur og kaldene imellem dem.

FrontController instansen får det rette Command-objekt, her ShowBillCommand, fra et HashMap i Command-klassen (3). Hvis det passende Command objekt ikke findes, returneres et UnknownCommand objekt som resulterer i en fejlside med passende fejltekst.

ShowBillCommand objektet bruger både DataFacade- og LogicFacade-objekter til udregning af styklisten. Datafacaden bruges til at hente materialer (6) og logicfacaden til udregning af antal (12).

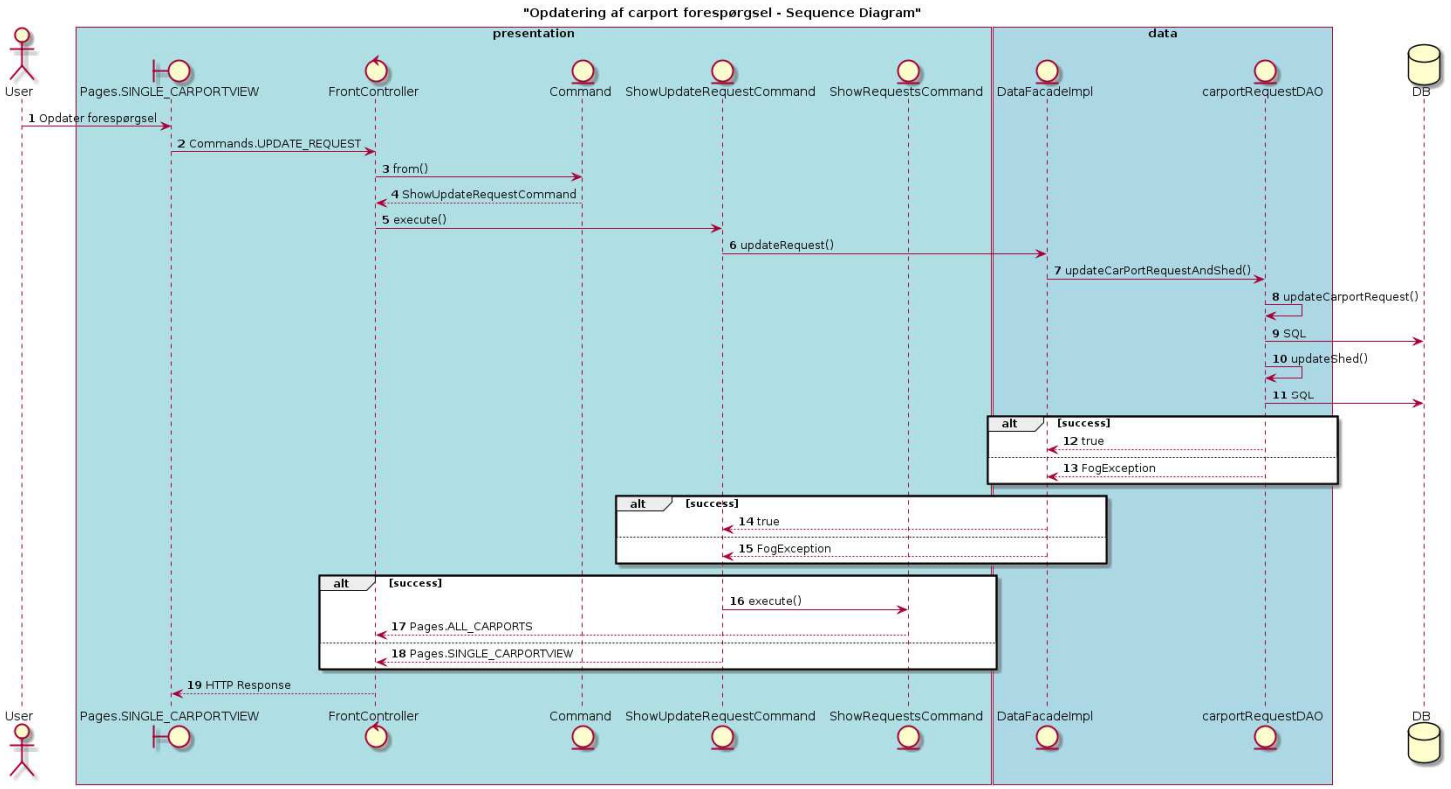
Logicfacaden bruger et Calculator objekt(13), som initialiserer et hashmap af materialer(14), til at beregne styklisten (17). Calculator objektet benytter objekter af RulesCalculator typen til at udregne hver del af carporten (18). Der er således flere RulesCalculators end der er illustreret på diagrammet.

I selve udregningen (19) skelnes der, i relevante RuleCalculators, mellem carporte med fladt tag og carporte med tag med rejsning. Dette gøres i if-sætninger, som evaluerer forespørgslens hældning. Baseret på denne værdi vælges f.eks. materialer enten af typen byg-selv spær eller spærtræ.

Disse valg har betydning for materialeforbruget, da byg-selv-spær, til tage med hældning, kan spænde over hele carportens vidde, mens spær til fladt tag evt. skal samles af flere stykker spærtræ. Dette har også betydning for antallet af stolper.

Når styklisten er udregnet, konverteres den til en HTML-repræsentation i ShowBillCommand objektet (23) inden den returneres med siden.

10.2 Opdatering af carportforespørgsel.



Figur 14 - opdatering af carport forespørgsel.

I figur 14 kan man se, at her er der tale om en ren opdateringsfunktion. Logic-laget er ikke med i denne funktion, da der ikke sker beregninger.

Da en carportforespørgsel også består af et skur, som har en separat tabel i databasen, har vi lavet en transaktion i CarportRequestDAO-klassen, så alle trinnene fra 8-11 skal udføres, ellers ingen. Dvs. både carportforespørgsel OG skur skal opdateres, ellers skal de delvise ændringer i databasen tilbagerulles. Se mere i afsnittet 14.5 Transaktion som opdaterer både skur og carport forespørgsel.

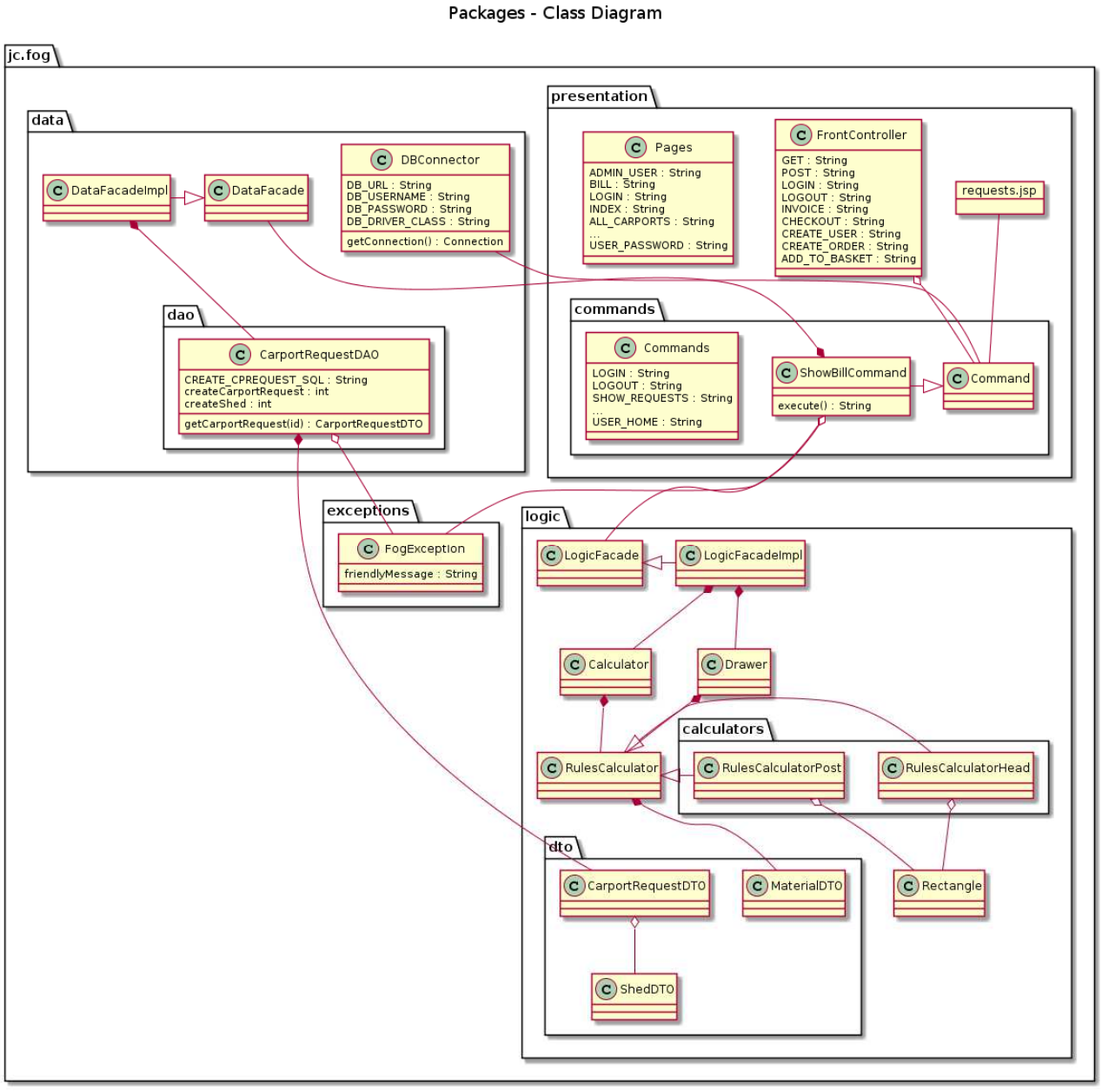
Hvis databaseopdateringerne fejler, kastes en FogException fra DAO'en, via Datafacaden til Command'et, som så viser formularen for carportforespørgslen igen. Hvis opdateringerne lykkes, returneres listen af carportforespørgsler via ShowRequestsCommand'et.

Da formularen også kan slette eller tilføje et skur til carportforespørgslen, må DAO'en afgøre, hvilke trin (8-11) der skal udføres. Her har vi lavet en if-sætning, som evaluerer flg:

1. Hvis skurets id er større end nul og fluebenet for skuret er valgt, skal skuret opdateres. Dernæst opdateres carporten.
2. Hvis skurets id er større end nul og fluebenet ikke er valgt, skal skuret slettes. Dernæst opdateres carporten.
3. Hvis skurets id er nul og fluebenet er valgt, opdateres carporten og skuret oprettes.
4. Ellers opdateres kun carporten.

11 Arkitektur

Applikationen er udviklet i en 3-lags arkitektur med facader mellem lagene, så den bagvedliggende implementation er skjult for klienten. På denne måde er det uden betydning for f.eks. et Command objekt, hvordan en liste med carport forespørgsler dannes. Dette er med til at gøre systemet nemmere at vedligeholde.



Figur 15 - Applikationens lagdelte arkitektur.

Bemærk at der kun er vist en del af klasser og associationer, for at skabe overblik.

12 Særlige forhold

12.1 Fejlhåndtering.

Softwaren skal naturligvis være brugervenlig. Dette sikrer vi bl.a. ved at implementere fejlhåndtering så brugeren ikke oplever, at applikationen går i stå eller giver svar, der er svære at forstå. Vi håndterer primært

fejl, dvs. Exceptions, i datalaget, hvor vi fanger fejl fra lavere liggende lag/pakker og konverterer dem til FogExceptions, som er mere brugervenlige. Disse FogExceptions sendes videre op gennem den lagdelte arkitektur, hvor Command-objekterne er endestationen. Her gemmes FogException's friendlyMessage på requestet og udskrives i jsp'en.

12.2 Validering af bruger-input.

For yderligere at sikre brugervenlighed, har vi implementeret input validering client-side. Der er altid den risiko, at man støder på en bruger, som har en browser, der ikke understøtter html5 validering (eller validering vha. JavaScript), derfor er validering server-side naturligvis et must. Det har vi desværre måttet nedprioritere pga. tidspres.

Client-side validering er implementeret på formularer til både carportforespørgsel og brugeroprettelse.

Vi har dog sikret os mod SQL injection ved at bruge PreparedStatement objekter, som udskifter wildcards i en sql sætning med modtagne værdier. Input trimmes ligeledes for leading og trailing whitespace.

12.3 Sessionsdata.

Når en bruger er logget ind, gemmes hans data i sessionen, så de kan aflæses på tværs af http-requests. Vi har valgt at gemme hele UsersDTO-entiteten for brugeren, så vi nemt kan udskrive f.eks. brugerens navn på siderne. Alternativet havde været at gå i databasen efter navnet hver gang og det belaster forbindelsen unødigt. Det skal nævnes, at password for den indloggede bruger ikke gemmes i sessionen.

12.4 Sikkerhed.

Brugerens kodeord er gemt som alm. tekst i databasen. Dette skal naturligvis ændres inden udrulning, hvor et kodeord hashes, f.eks. med SSHA-512, med et tilfældigt salt. Hash-værdien og salt-værdien gemmes i brugertabellen, hashing sker server-side. Vi havde forestillet os at bruge et bibliotek som giver en tilpas tilfældig salt, hver gang en bruger skal oprettes, som f.eks. java.security.SecureRandom. Implementationen er nedprioriteret pga. tidsmangel.

Har man glemt sit kodeord, er der implementeret mulighed for at nulstille denne. På sigt vil brugeren få en e-mail som leder til en login side, hvor man bliver bedt om at oprette en ny, dette er ikke implementeret endnu.

12.5 Brugertyper.

Systemet kan skelne mellem flere typer brugere vha. rank-værdier. Baseret på en brugers rank, gives adgang til forskellige administrative funktioner, og det er således muligt at have flere niveauer af administrative brugere. For Fog betyder det, at f.eks. prisjustering kan begrænses til at kunne udføres af et udsnit af de administrative brugere. Anonyme brugere, som afsender en forespørgsel, bliver ved afsendelsen oprettet som bruger med et automatisk genereret kodeord. E-mail skal herefter sendes til brugeren, denne del er dog faldet for tidsfristen.

Systemet håndterer, at sidste bruger med en given rank ikke kan nedgradere sig selv, således at systemet altid har mindst en bruger med en given rank. Vi har også overvejet blot at deaktivere en bruger i stedet for at slette, så evt. fremtidig logning af brugerhandlinger kunne ske uden database anomali ved senere sletning af bruger.

Når man skal oprette en administrativ bruger, skal man oprette en almindelig bruger og ophøje denne. Senere kunne man forestille sig, at systemet giver den administrative bruger mulighed for at sætte rank på en ny bruger ved oprettelsen.

12.6 Brugertyper i databasen.

Vi har blot implementeret en enkelt brugertype i databasen og denne har fulde rettigheder på alle tabeller i Fog-databasen.

12.7 Databaseforbindelse.

Da db.properties filen ikke findes på Github, skal forbindelsen konfigureres manuelt ved at indsætte flg. i jc.fog.data.DbConnector klassen:

URL: jdbc:mysql://142.93.174.238:3306/Fog
USERNAME: fogUser
PASSWORD: fogUser@2018

12.8 Dokumentation.

Systemet har Javadoc på github.

13 Kodemæssige forbedringer.

13.1 Claus

Navngivning af commands burde ske efter en vedtaget konvention, da det er svært at gennemskue, hvad de forskellige commands gør. Strengkonstanter i Commands-klassen bør netop indikere, hvad der skal ske og Command'et, der implementerer dette, bør have et lignende navn for overskuelighedens skyld.

Nogle commands kunne nok samles, således kunne også undgås, at de er indbyrdes afhængige, idet nogle commands returnerer svaret fra andre.

Alle commands skal fange de kastede FogExceptions og lagre en venlig besked på requestet, så brugeren ved hvad der er gået galt.

Diverse formularer ville jeg have implementeret direkte i den jsp, hvor de vises, i stedet for at danne dem i Command-objekter.

ShowCarPortCommand.carportRequestToBill() er overfyldt og udfører flere opgaver. En korrekt navngivning og afgrænset ansvar skal implementeres ved at opdele metoden i flere små.

Når metoder i DAO klasserne opretter tupler, bør en korresponderende DTO eller tuplens id returneres. Hvis oprettelse slår fejl, skal der fanges en FogException, ikke blot returneres true, false eller 0.

Bedre dokumentation (Javadoc) af metoder generelt.

13.2 Jesper

Jeg mener bestemt, at de kan blive gjort noget mere af filer. I forhold til, at der er mindre filer og nogen kan med sikkerhed godt bliv lagt sammen.

Der skal have lavet sådan, at klik vi på "tilføj" skur og man har fortalt, at der hældning skal være på 0 så for man lov til, at kun vælge fx noget bestemt tag type.

At der bliver arbejde med gjort noget mere ud af fejl handling af hvis der går noget galt.

Skabt et bedre overblik omkring Commands. Evt bedre navn til filer.

Som udgangspunkt har vi kigget på, at opret forespørgelse og updaterforespørgelse på en side vil giv mest mening i forhold til DRY.

Det er gjort for hvis der fx skulle komme nyt tiltale omkring noget til carporten så har vi valgt at gør det så let som overhovedet muligt.

```
// get request's id from request hvis den findes.
int id = 0;

// Get DataFacadeImpl.
DataFacadeImpldataFacade = newDataFacadeImpl(DbConnector.getConnection());
String requestForm = null;

HttpSession session = request.getSession();
UsersDTO user = (UsersDTO)session.getAttribute("user");

// Har vi et gyldigt id ?
try
{
// Findes id ikke på requestet, catcher vi exception
    id = Integer.parseInt(request.getParameter("id"));
}
catch(NumberFormatException n){
// NumberFormatException er forventet, hvis request ikke har id, som så vil være 0.
}

List<RooftypeDTO>Rooftypes = dataFacade.getRooftypes();

// Fandt vi et gyldigt id på requestet (dvs. > 0)
if(id > 0)
{
    // Har vi en user i session, er denne logget ind, gå til index side.
    if(user == null || user != null&&user.getRank() > 1)
            {
    returnPages.INDEX;
}

//Det her skal bliver vist hvis man skal updater indhold.
CarportRequestDTOcarportRequestDTO = dataFacade.getCarport(id);
// Create HTML form with request's data and set it on http request.
requestForm = carportRequestToBill(carportRequestDTO, user, Rooftypes);
        }
else
        {
// Ingen id i requestet, lav en tom formular til ny oprettelse af carportrequest

//Det her skal blive vist hvis man skal opret en forespørgelse.
requestForm = carportRequestToBill(null, user, Rooftypes);
        }

request.setAttribute("requestForm", requestForm);//Det er form til den enkelt som skal
bruges

returnPages.SINGLE_CARPORTVIEW;
```

14 Udvalgte kodeeksempler

14.1 Validering af brugerinput client-side.

I ShowLoginCommand.login() er brugt et regulært udtryk for at validere email.

```
stringbuilder.append("E-mail (Brugernavn):<br /><input type=\"email\" pattern=\"[a-zA-Z0-
9.-_]{1,}@[a-zA-Z.-]{2,}[.]{1}[a-zA-Z]{2,}\" name=\"email\" required class=\"form-
control\" placeholder=\"Din Email\" /><br />");
```

I ShowCarportCommand.carportRequestToBill() er der opstillet krav til input.

```
stringBuilder.append("Bredde:<br /><input type=\"number\" maxlength=\"3\" required
min=\"300\" max=\"900\" name=\"width\" class=\"form-control\" value=\"$carport2\"
placeholder=\"Bredde på carport\" /><br />");
```

14.2 Sessionsdata.

I ShowLoginCheckCommand.execute() sættes brugeren i sessionen.

```
UsersDTO user = dataFacade.login(email, password);

//tilføjer session på Rank og Id.
HttpSession session = request.getSession();
session.setAttribute("user", user);
```

14.3 Nulstilling af kodeord.

UserDAO.randomPassword() nulstiller brugerens kodeord.

```
public String randomPassword()
{
    Random rand = new Random();
    int max = rand.nextInt(10) + 5; //Hvis nextInt bliver "0" så vil den altid tilføj 5.

    String uniqueText = UUID.randomUUID().toString(); // 0f8fad5bd9cb-469f-a165-
70867728950e
    return uniqueText.replace("-", "").substring(0, max);
}
```

14.4 Lukning af ressourcer med try with ressources.

For at løse udfordringen med at lukke ressourcer korrekt, bruges *try with ressources* hvor det er muligt. For at kunne indsætte værdier i placeholders i en sql streng, har vi tidligere indlejret et par try-sætninger, som her illustreret fra UserDAO.getUser() :

```
try(PreparedStatement pstm = connection.prepareStatement(CREATE_USER_SQL,
Statement.RETURN_GENERATED_KEYS))
{
    pstm.setString(1, name);
    pstm.setInt(2, zipcode);
    pstm.setInt(3, phone);
    pstm.setString(4, email);
    pstm.setString(5, password);
    // udfør opdatering.
    pstm.executeUpdate();
    // Hent id for netop oprettet bruger.
    try (ResultSet rs = pstm.getGeneratedKeys())
    {
        if (rs.next())
```

```
            result = rs.getInt(1);
    }
}
```

Men da vi ikke synes, indlejrede try-sætninger er god skik, har vi i stedet implementeret en hjælpemetode der kan returnere et PreparedStatement komplet med de indsatte værdier på de rette pladser. Med denne metode, bliver ovenstående kode til:

```
Pair<Integer, Object> pair1 = new Pair<>(1, name);
Pair<Integer, Object> pair2 = new Pair<>(2, zipcode);
Pair<Integer, Object> pair3 = new Pair<>(3, phone);
Pair<Integer, Object> pair4 = new Pair<>(4, email);
Pair<Integer, Object> pair5 = new Pair<>(5, password);
try
(
    PreparedStatement pstm = createPreparedStatement(connection, CREATE_USER_SQL,
Statement.RETURN_GENERATED_KEYS, pair1, pair2, pair3, pair4, pair5);
    ResultSet rs = updateAndGetKeys(pstm);
)
{
    if (rs.next())
        result = rs.getInt(1);
}
```

Pair<K, V> key-value pair objekterne erklæres til at indeholde typerne Integer og Object. Object har vi brugt, fordi vi gerne vil kunne sende både int's, strings og andre typer til vores metode, i øvrigt i et vilkårligt antal. Derfor er metoden nedenfor implementeret med varargs og gør brug af PreparedStatement.SetObject() til at mappe værdierne til deres respektive typer og indsætte dem i sql'en:

```
protected PreparedStatement createPreparedStatement(Connection connection, String sql, int
autoGeneratedKeys, Pair<Integer, Object>... parameters) throws SQLException
{
    PreparedStatement pstm;
    // Hvis autoGeneratedKeys ikke har en gyldig værdi, oprettes 'alm.' PreparedStatement
objekt.
    if (autoGeneratedKeys != Statement.NO_GENERATED_KEYS && autoGeneratedKeys !=
Statement.RETURN_GENERATED_KEYS)
        pstm = connection.prepareStatement(sql);
    else
        pstm = connection.prepareStatement(sql, autoGeneratedKeys);

    // Sæt de evt. modtagne key-value par ind i sql sætningen.
    if (parameters != null)
        for (Pair<Integer, Object> pair : parameters)
        {
            pstm.setObject(pair.getKey(), pair.getValue());
        }

    return pstm;
}
```

Som det fremgår af *try with ressources*-kaldet, benyttes endnu en hjælpemetode til at køre executeUpdate() på et PreparedStatement objekt. Denne returnerer et ResultSet med nøgler, hvis det ønskes.

```
protected ResultSet updateAndGetKeys(PreparedStatement preparedStatement) throws
SQLException
{
    preparedStatement.executeUpdate();
    ResultSet rs = preparedStatement.getGeneratedKeys();
    return rs;
}
```

Da både PreparedStatement og ResultSet implementerer AutoCloseable interfacet, kan begge metoder dermed kaldes i *try with ressources*. Både createPreparedStatement() og executeUpdate() er implementeret i AbstractDAO, så alle DAO nedarvninger kan drage fordel af dem.

14.5 Transaktion som opdaterer både skur og carport forespørgsel.

Som tidligere beskrevet, kan en carportforespørgsel også bestå af et skur som er en entitet for sig. For at undgå at en carportforespørgsel oprettes, selvom oprettelsen af skuret fejler, har vi lavet en transaktion for at sikre, at begge dele oprettes korrekt. Her et udsnit fra CarportRequestDAO.createCarportRequestAndShed().

```
try
{
    // Start transaktion, da BÅDE carport OG skur (hvis valgt) skal oprettes.
    connection.setAutoCommit(false);

    // Først oprettes forespørgsel.
    int carportRequestId = createCarportRequest(rooftypeId, slope, width, height, length,
remark);
    // hvis skur dimensioner er angivet, oprettes det nu.
    if (shedLength > 0 && shedWidth > 0)
        createShed(carportRequestId, shedLength, shedWidth);

    connection.commit();
    // Reset autocommit på forbindelsen.
    connection.setAutoCommit(true);

    return true;
}
catch(SQLException s)
{
    try
    {
        connection.rollback();
    }
    catch(SQLException se)
    {
        // rollback failed, lad os lukke forbindelsen.
        DbConnector.closeConnection();
        throw new FogException("Forespørgslen blev ikke oprettet.", se.getMessage(), se);
    }

    throw new FogException("Forespørgslen blev ikke oprettet.", s.getMessage(), s);
}
```

I Catch-delen, rulles databaseændringerne tilbage og en FogException, med beskrivende tekst kastes videre i arkitekturen. Skulle rollback() fejle, lukkes forbindelsen og en FogException kastes.

14.6 Materialer til beregning inddeles efter type i HashMap.

Alle RulesCalculator-nedarvning deler et HashMap af materialer, som er opdelt efter materialetypen, så de hver især kan tilgå præcis den delmængde materialer, de har brug for. Opdelingen sker ved at gennemløbe en enum, Rules.Materialtype, som indeholder id's for materialetyper. For hver id filtreres den samlede liste af materialer, og delmængden lagres i HashMap'et.

```
public static void initializeMaterials(List<MaterialDTO> materialList) throws FogException
{
    try
    {
        // Opret ny hashmap.
        materials = new HashMap<>();
        // Gennemløb enum og opret HashMap key-value pair med subset af listen, hvor
        // hvert MaterialDTO objekts materialtypeId svarer til værdien i enum.
        for(Materialtype mt : Materialtype.values())
        {
            materials.put(mt.name(), filter(materialList, mt.getMaterialtypeId()));
        }
    }
    catch(Exception e)
```

```
    {
        throw new FogException("Materialer blev ikke initialiseret", e.getMessage(), e);
    }
}
```

Herunder ses hjælpemetoden som filtrerer materialelisten vha. et lambda udtryk.

```
private static List<MaterialDTO> filter(List<MaterialDTO> list, int typeId)
{
    Stream<MaterialDTO> stream = list.stream().filter(m -> m.getMaterialtypeDTO().getId()
== typeId);
    List<MaterialDTO> result = stream.collect(Collectors.toList());
    return result;
}
```

14.7 Ved beregninger promotes operander for at finde korrekt antal.

For at sikre, at bl.a. carportens tag ikke får for få spær, er vi nødt til at promote operanderne, når vi dividerer tagets længde med afstanden mellem spærene. Dette fordi vi må have floating point division for at beholde fraktionen. Således kan vi runde op, så vi, i eksemplet herunder, ved roofLength på 200 får 200/55 = 3,6 ~ 4 mellemrum mellem 5 spær. På samme måde udregnes den aktuelle afstand ml. spærene, her 50 cm.

```
// Find tagets længde og udregn antal spær, adskilt 55 cm fra hinanden.
raftersCount = (int)Math.ceil(roofLength / Rules.RAFTER_SPACING) + 1;// Et spær ekstra, da
vi har udregnet antal mellemrum.
// Gem aktuel afstand ml. spær.
raftersSpacing = (int)Math.ceil(roofLength / (float)(raftersCount-1));// Husk at fratrække
slut spær for at få nok mellemrum.
```

14.8 Dataforbindelse indlæses fra properties fil.

Databaseforbindelsens url, brugernavn og adgangskode skal ikke kunne aflæses direkte i kildekoden lagt på Github. Derfor har vi valgt at lave en konfigurationsfil til forbindelsen og ekskludere den fra Github. Den må derfor indlæses, når en forbindelse skal etableres:

```
public static Connection getConnection() throws FogException
{
    // Hvis connection er etableret, se om den er lukket.
    if (connection != null)
    {
        try
        {
            if (connection.isClosed())
                connection = null;
        }
        catch(SQLException s)
        {
            // Får vi exception ved isClosed, fjern reference så den kan garbage
collectes.
            connection = null;
        }
    }
    if ( connection == null)
    {
        try
        {
            Class.forName( DRIVER_CLASS );

            // Read the properties of the database connection from
target/classes/db.properties
            Properties dbProperties = new Properties();
            InputStream inputStream =
DbConnector.class.getResourceAsStream("/db.properties");
            dbProperties.load(inputStream);
```

```
            connection = DriverManager.getConnection(dbProperties.getProperty("URL"),
dbProperties.getProperty("USERNAME"),
dbProperties.getProperty("PASSWORD"));
        }
        catch(Exception e)
        {
            throw new FogException("Fejl v. etablering af db. forbindelse.",
e.getMessage(), e);
        }
    }
    return connection;
}
```

14.9 Beregning af materialer.

Dette er en kompleks beregning som foretages af flere delberegnere. Inden implementation, dannedes et overblik vha. pseudokode.

Beregn carport

-> beregn tag

Hvis fladt tag:

Tjek regel for afstand mellem strøer
Tjek regel for udhæng
Beregn antal strøer og deres længde
Beregn kvadratmeter for tag
Udregn

Hvis tag med rejsning:

Tjek regel for afstand mellem bjælker
Tjek regel for udhæng
Beregn antal spær og deres længde
Beregn kvadratmeter for tag
Beregn antal mtr. tagrende hvis tilvalgt
Beregn antal mtr. Nedløbsrør

-> beregn stolper

-> beregn skur

Her omsat til kode i Calculator-klassen:

```
private void initializeRulesCalculators()
{
    calculators = new ArrayList<RulesCalculator>();

    calculators.add(new RulesCalculatorHead()); // Udregner rem.
    calculators.add(new RulesCalculatorPost()); // Udregner stolper.
    calculators.add(new RulesCalculatorRafters()); // Udregner spær.
    calculators.add(new RulesCalculatorShed()); // Udregner skurets beklædning.
    calculators.add(new RulesCalculatorBattens()); // Udregner lægter.
    calculators.add(new RulesCalculatorRoof()); // Udregner tagbelægning.
}

protected List<BillItem> calculateBill(CarportRequestDTO carportRequest) throws
FogException
{
    // Opret tom stykliste.
    ArrayList<BillItem> bill = new ArrayList<>();

    // Gennemløb alle rule calculators.
    for(RulesCalculator calculator : calculators)
```

```
    {
        bill.addAll(calculator.calculate(carportRequest));
    }

    return bill;
}
```

15 Status på implementationen.

15.1 Funktionelle mangler.

Flg. user stories findes stadig i vores backlog.

Vis carport
Design egen carport
Se beskrivelse
Udskriv ordredokumenter
Filtrer carport
Vis fejl i carportforespørgsel
Tilføj vare fra databasen til styklisten
Administrer kunde
Besvare carportforespørgsel
Administrer samlevejledninger
Se ordrestatus
Vis leveringspris
Rediger dækningsgrad
Rediger hjælpetekst

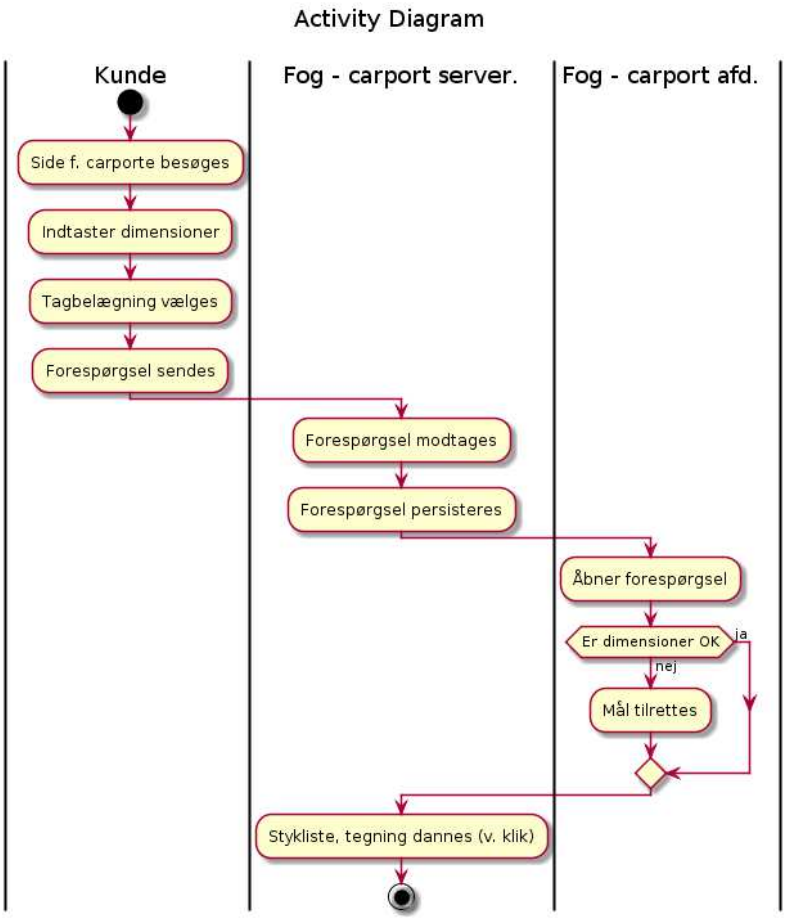
Listen er ikke udtømmende, da der findes user stories/tasks, der ikke er registreret i Backlog'en endnu. Flg. er ikke implementeret:

- Administration af materialer
- Kunde knyttes til forespørgsel
- Se status for forespørgsel

Systemet har flg. funktioner:

- Opret brugerkonto
- Log ind & log ud
- Opret forespørgsel
- Åbn forespørgsel
- Opdater forespørgsel
- Vis materialer
- Beregn stykliste
- Nulstil kodeord til brugerkonto
- Administrer brugerkonti
- Validering af brugerinput
- Vis tegning
- Se/udskriv tegning

Her ses et aktivitetsdiagram for det udførte software:



15.2 Øvrige mangler.

Oprydning i try-catch sætninger ønskelig.

Konsistente kommentarer, så man ikke skal helt ned i en DAO for at finde ud af, hvornår hvad returneres men i stedet se det i facaderne.

Visse tests fungerer kun under visse forudsætninger og skulle i stedet have været lavet, så de selv opfylder forudsætningerne. F.eks. CarportRequestDaoITest.getCarport() som forudsætter, at en carportforespørgsel med id=1 findes. I stedet skulle testen oprette en forespørgsel, se om denne kunne hentes, som i UserDaoITest.testGetAllUsers().

Funktioner, som opretter tupler i databasen, skal returnere oprettet id eller exception, ikke true/false. Ellers er det svært at teste metoder, der henter tupler i databasen, da disse altid fordrer et id. Hvis ikke vi kan oprette og få et id tilbage, kræver disse tests, at tuplen findes i databasen.

Ikke alle tabeller har CRUD, og dette har betydning for visse tests. F.eks. RooftypeDaoITest som kræver, at der findes et datagrundlag på forhånd. Ideelt skal testen selv kunne oprette det data, der skal testes på.

Tests af calculators skulle have været mere opdelt, så der var en UnitTest fil for hver RulesCalculator implementation. Dette har vi måttet nedprioritere pga. tidspres.

For at undgå at sende formular data flere gange ved genindlæsning af en jsp, skal FrontController redirecte i stedet for at forwarde requestet. Information om request eller redirect kunne evt. gemmes i en AbstractRedirectCommand.

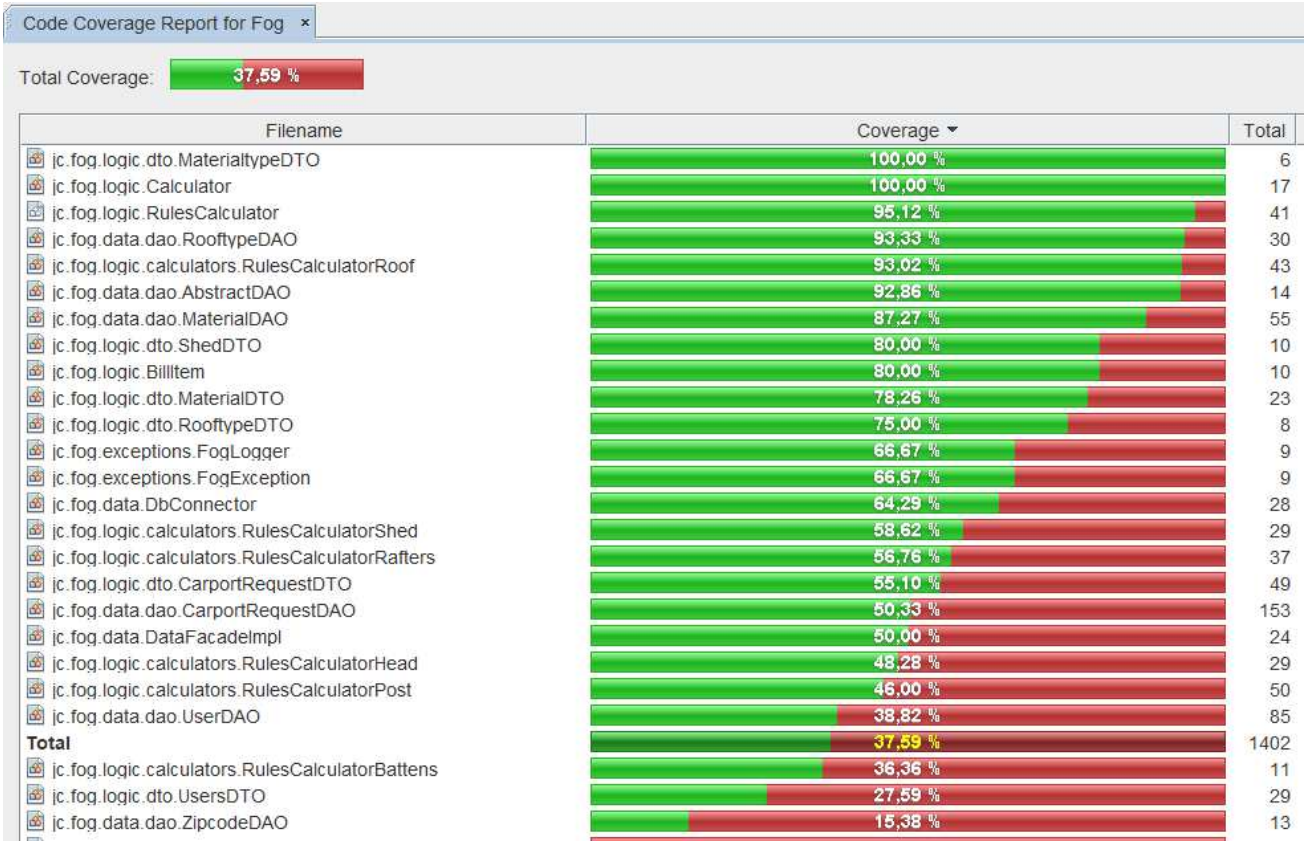
16 Tests

I vores system har vi både unit tests og integration tests. Unit tests har postfix UTest, integration test har prefix ITest.
Det skal nævnes, at vi havde en udfordring med at køre UserDaoITest fordi metoderne createUser og getAllUsers kørte parallelt. Da begge metoder opretter brugere med samme email, som er unik, fejlede den ene metode, indtil vi tvang NetBeans til at køre metoderne i rækkefølge vha. flg. anmærkning, erklæret over klassedeklarationen: `@FixMethodOrder(MethodSorters.NAME_ASCENDING)`.

Klasse	Metode	Kommentar:
CarportRequestDaoITest	getCarports	Henter carportforespørgsler i databasen. Testen går igennem datafacaden.
	createCarportRequestFails	Forsøger at oprette en carportforespørgsel, men fejler, da databaseforbindelsen lukkes. Vi tester at vi får den forventede Exception.
	getCarport	Henter en carportforespørgsel med angivet id gennem datafacaden.
	getCarportsFails	Forsøger at hente carporte fra databasen, men fejler, da forb. lukkes.
	getRequest	Henter en carportforespørgsel med angivet id direkte på DAO'en.
	getRequests	Henter carportforespørgsler i databasen. Testen arbejder direkte på DAO'en.
	createRequestWithShed	Opretter carportforespørgsel med skur.
	createRequestNoShed	Opretter carportforespørgsel uden skur.
	updateRequest	Opdaterer en carportforespørgsel.
MaterialDaoITest	createMaterial	Tester om det er muligt at oprette et materiale i databasen.
	testGetMaterials	Henter alle materialer fra databasen.
	testGetMaterialsFails	Fejler når materialer forsøges hentet, fordi forbindelsen til databasen lukkes inden. Vi tester om vi får korrekt exception.
	testGetSingleMaterial	Henter et bestemt materiale i databasen. Kræver dog at tuplen findes i databasen.
RooftypeDaoITest	getRooftypes	Henter systemets tagtyper fra databasen.
UserDaoITest	createUser	Tester oprettelse af bruger og nedlægger denne igen efter test.
	getAllUsers	Opretter selv en bruger inden hentning af systemets brugere, således at testen kan køre, selvom databasen måtte være uden brugere. Brugeren nedlægges efter brug.
CalculatorITest	testCalculators	Tester at RulesCalculators kan beregne samlet stykliste.

CalculatorUTest	calculateHypothenuse	Tester at udregningen af en given bredde med given hældning giver korrekt taglængde.
	calculateHypothenuseFailAngleTooBig	Tester at udregningen af taglængde fejler korrekt, hvis hældningen er 90 grader eller over.
	calculateHypothenuseFailAngleTooSmall	Tester at udregningen af taglængde fejler korrekt, hvis hældningen er 0 grader eller under.
	calculateMaterialFail	Tester at udregningen fejler korrekt, hvis der ingen materialer findes i RulesCalculator.materials.
	postCalculateMaterial	Tester at stolpeudregning er korrekt ift. tilgængelige materialer og carportens konfiguration.
	postCalculateMaterialFail	Tester at stolpeudregning kaster korrekt exception hvis udregning fejler. Fejl fremprovokeres ved at have spærtræ med længde 0.
	raftersCalculateMaterialFail	Tester at spærudregning kaster korrekt exception hvis udregning fejler. Fejl fremprovokeres ved at have spærtræ med negativ længde.
	raftersCalculateMaterialNoSlope	Tester korrekt udregning af spær til carport med fladt tag.
	raftersCalculateMaterialSlope	Tester korrekt udregning af spær til carport med tag med rejsning.
RulesCalculatorITest	testHeadCalculatorNoSlope	Tester at RulesCalculatorHead udregner korrekt v. fladt tag med materialer fra databasen via DAO.
	testPostCalculator	Tester RulesCalculatorPost med materialer fra databasen hentet med DAO.
	testRoofCalculatorNoSlope	Tester RulesCalculatorRoof materialer fra databasen til fladt tag hentet med DAO.
	testRoofCalculatorSloped	Som ovenstående, dog til tag med rejsning.
MaterialDtoUTest	materialComparison	Tester implementationen af Comparable interfacet på MaterialDTO, hvor sammenligning sker på materialTypeId.
	materialComparisonWithNegativeTypeId	Tester at implementationen af Comparable<MaterialDTO> ikke fejler v. negative værdier.
	materialComparisonWithNull	Tester at Comparable<MaterialDTO> ikke fejler ved null-værdier.

16.1 JaCoCo rapport for projektet.


Code Coverage Report for Fog ×

Total Coverage: 37,59 %

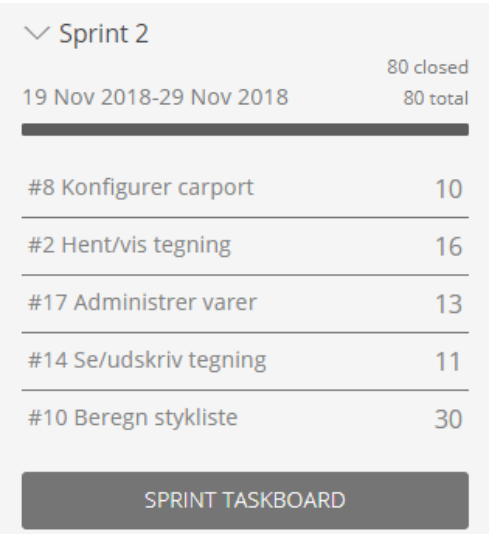
Filename	Coverage	Total
jc.fog.logic.dto.MaterialtypeDTO	100,00 %	6
jc.fog.logic.Calculator	100,00 %	17
jc.fog.logic.RulesCalculator	95,12 %	41
jc.fog.data.dao.RooftypeDAO	93,33 %	30
jc.fog.logic.calculators.RulesCalculatorRoof	93,02 %	43
jc.fog.data.dao.AbstractDAO	92,86 %	14
jc.fog.data.dao.MaterialDAO	87,27 %	55
jc.fog.logic.dto.ShedDTO	80,00 %	10
jc.fog.logic.BillItem	80,00 %	10
jc.fog.logic.dto.MaterialDTO	78,26 %	23
jc.fog.logic.dto.RooftypeDTO	75,00 %	8
jc.fog.exceptions.FogLogger	66,67 %	9
jc.fog.exceptions.FogException	66,67 %	9
jc.fog.data.DbConnector	64,29 %	28
jc.fog.logic.calculators.RulesCalculatorShed	58,62 %	29
jc.fog.logic.calculators.RulesCalculatorRafters	56,76 %	37
jc.fog.logic.dto.CarportRequestDTO	55,10 %	49
jc.fog.data.dao.CarportRequestDAO	50,33 %	153
jc.fog.data.DataFacadeImpl	50,00 %	24
jc.fog.logic.calculators.RulesCalculatorHead	48,28 %	29
jc.fog.logic.calculators.RulesCalculatorPost	46,00 %	50
jc.fog.data.dao.UserDAO	38,82 %	85
Total	37,59 %	1402
jc.fog.logic.calculators.RulesCalculatorBattens	36,36 %	11
jc.fog.logic.dto.UsersDTO	27,59 %	29
jc.fog.data.dao.ZipcodeDAO	15,38 %	13

17 Scrum

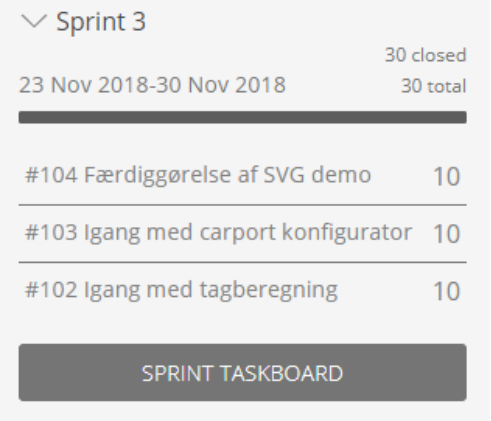
17.1 Arbejdsprocessen faktuelt

Vores arbejdsproces blev inddelt i sprints jf. scrum metoden. Vi har haft 4 sprints af en ca. 1 uges varighed.

Til første PO møde havde vi identificeret og estimeret user stories efter INVEST princippet, dog var det nødvendigt med justeringer for at få enkelte stories helt uafhængige. Vi havde især svært ved at gøre how-to-demo for en story helt uafhængig af en anden.
På PO-mødet inden første sprint blev der lagt vægt på at få leveret user stories med "business value". PO var enig i vores vurdering ift. vigtigheden af #6 og #7 og disse fandt derfor vej til dette sprint. I sprintet var oprindeligt også lagt #17-administrer varer og #10-beregn stykliste. Dog nåede vi ikke disse og de blev flyttet til 2. sprint.



På PO møde forud for sprint 2, gjorde man det klart, at man ville have mulighed for at kunne designe/konfigurere en carport, da det ville muliggøre salg. En tegning og stykliste dertil ville være en stor hjælp, så disse user stories kom med.



På PO mødet forud for dette sprint, var vi efterhånden så langt bagud, at vi blev enige om at lave blot en demo-tegning, en beregner til en del af carporten og at påbegynde carport designeren.

Sprint 4

292 closed
30 Nov 2018-11 Dec 2018 292 total

#18 Nulstil brugerkonto	12
#24 Log ind	15
#19 Administrer brugerkonti	18
#25 Log ud	8
#130 Validering af brugerinput	20
#73 Test system	20
#115 Admin sider skal beskyttes af login	23
#86 Exception håndtering og logging	40
#88 Færdiggøre stykliste	20
#91 Færdiggøre konfigurator til carport	26
#84 Oprydning af sprint 3	20
#87 Færdiggøre tests	30
#85 Tegning færdiggøres	40

Sidste PO-møde handlede mest om at få lukket de huller vi havde fået lavet gennem de seneste sprints. Så udviklerteamet besluttede at give dette sprint 3 dage ekstra til de mange opgaver.

17.2 Arbejdsprocessen reflekteret

Scrum har i nogen grad fungeret for os, men det har bestemt ikke været uden udfordringer. Inden projektstart blev vi blev advaret om, at scrum måske ikke gav meget mening for en 2-mands gruppe, om det er præcis derfor, vi har haft udfordringer, skal vi lade være usagt. Men ringe erfaring med scrum i det hele taget, har også bidraget væsentligt, formoder vi.

Til trods for dette, har vi dagligt afholdt stand-up møder, som ofte endte med at gå ud over de foreskrevne 15 minutters varighed. På disse møder har vi talt om, hvor langt vi nåede dagen i forvejen, hvad vi ville lave den pågældende dag og om der var problemer.

Havde vi problemer, syntes vi det gav bedst mening at adressere disse med det samme, så begge team medlemmer kunne komme videre derefter. Således undgik vi også ofte, at nogle opgavers færdiggørelse afventede den anden udvikler.

Vi har begge forsøgt os som scrum master og det er gået fint i forhold til daily standup. Til gengæld har det haltet alvorligt mht. at få backlog og sprints opdateret, fordi vi ret hurtigt kom bagud i sprints og, i stedet for at styre, havde fokus på at færdiggøre. Det tog indtil sprint 4, før vi fik defineret opgaver, som gjorde det muligt at få ryddet op. Her kom den manglende erfaring til udtryk.

Når et sprint gik godt, var det fordi vi tog os tid til at få delt user stories op i mindre tasks. Når det lykkedes, og man var omhyggelig med at få opdateret task-status, var taiga boardet en stor hjælp.

Fredag afholdt vi retrospectives, da vi mente mandagen ellers ville gå med for meget mødeaktivitet, her havde vi også PO møder. Samtidig var ugen i frisk erindring inden weekenden.

PO-møder bar ofte præg af vores forsinkelser ift. sprint arbejdsvolumen. Selvom det er givet, at man bliver bedre til at estimere ved at estimere forkert nogle gange, var der alligevel så få uger og dermed sprints, at de tidlige guesstimates var svære at få rettet ind, nok fordi vi samtidig var for dårlige til at afgrænse os.

Det giver ganske givet nogle bedre møder, hvis man mødes ansigt til ansigt. Vi har afholdt daily standup og retrospectives over telefonen, da vi ikke bor i nærheden af hinanden. Vi har på den måde sparet tid og har løst mange problemer via teamviewer, men måske er der et andet commitment når man mødes fysisk.

Men selvom det har været en udfordret proces, har viljen ikke fejlet noget, hvilket også kan ses af sprint 4, hvor der blev arbejdet igennem for at lukke åbne tasks.

Bilag

1 SCRUM user stories:

1. *Filtrer carport:*
- Som kunde vil jeg gerne kunne filtrere blandt carporte for at finde den rette.
2. *Hent/vis tegning:*
- Som kunde vil jeg gerne kunne hente tegninger af en valgt carport for at kunne visualisere den.
3. *Vis carport:*
- Som kunde vil jeg gerne kunne få vist en carport, så jeg kan danne mig et indtryk og læse mere om den.
4. *Vis leveringspris:*
- Som kunde ønsker jeg at kunne se leveringsprisen for bedre at kunne beslutte mig.
5. *Design egen carport:*
- Som kunde vil jeg gerne kunne designe carport efter egne mål, så jeg får en carport der passer bedre end standard carporte. (OBS: Hvad med at angive bilens bredde og højde…)
6. *Afsend carportforespørgsel:*
- Som kunde vil jeg gerne kunne forespørge på en carport jeg selv har designet.
7. *Åbn carportforespørgsel:*
- Som indlogget medarbejder i trælasten vil jeg kunne åbne en modtaget carportforespørgsel på en måde så jeg undgår at skulle indtaste kundens valg igen.
8. *Konfigurer carport:*
- Som indlogget medarbejder i trælasten vil jeg kunne konfigurere en carport således at fejl evt. undgås.
9. *Administrer kunde:*
- Som indlogget medarbejder i trælasten vil jeg kunne administrere kunder således at stamdata er korrekte til sikring af f.eks. korrekt leveringsadresse.
10. *Beregn stykliste:*
- Som indlogget medarbejder i trælasten vil jeg kunne beregne styklisten automatisk, således at fejl undgås, priser bliver korrekte og lagerbeholdning vedligeholdes korrekt.
11. *Rediger dækningsgrad:*
- Som indlogget medarbejder i trælasten vil jeg kunne redigere dækningsgraden så jeg kan lave et godt tilbud hvis der er behov.
12. *Rediger hjælpetekst:*
- Som indlogget medarbejder i trælasten vil jeg kunne redigere hjælpeteksten så kunden har bedre mulighed for at samle carporten korrekt.
13. *Tilføj vare fra databasen:*
- Som indlogget medarbejder i trælasten vil jeg kunne tilføje flere varer til ordren fra databasen, for at sikre at kunden får alle beslag og øvrige varer nødvendige for en korrekt samling af carporten. Evt. mersalg?
14. *Se/udskriv tegning:*
- Som indlogget medarbejder i trælasten vil jeg kunne se/udskrive tegningen af den konstruerede carport så jeg kan bedre kan visualisere den og evt. besvare kundespørgsmål.
15. *Se beskrivelse:*

- Som indlogget medarbejder i trælasten vil jeg kunne se carportens beskrivelse, så jeg kan svare på spørgsmål fra kunden mv.

16. *Besvare carportforespørgsel:*

- Som indlogget medarbejder i trælasten vil jeg kunne besvare forspørgsler vedr. carporte elektronisk, så kunden hurtigt får en tegning og oplever oprigtig interesse i kundens projekt.

17. *Administrer varer:*

- Som indlogget medarbejder i trælasten vil jeg kunne oprette og redigere varer, så jeg undgår at skulle håndtere forskellige varenumre, tekster mv. for den samme vare.

18. *Nulstil brugerkonto:*

- Som indlogget medarbejder i trælasten vil jeg kunne nulstille min brugerkonto, så jeg kan igen kan komme ind i systemet, selvom jeg har glemt mit kodeord.

19. *Administrer brugerkonti:*

- Som indlogget medarbejder i trælasten vil jeg kunne nedlægge eller oprette konti for andre medarbejdere, så de kan tilgå systemet og arbejde med carport forespørgsler.

20. *Administrer samlevejledninger:*

- Som indlogget medarbejder i trælasten vil jeg kunne redigere i eller tilføje samlevejledninger, så de altid er aktuelle.

21. *Udskriv ordredokumenter:*

- Som indlogget medarbejder i trælasten vil jeg kunne udskrive enkelte eller samtlige dokumenter for en given ordre på en gang, inkl. Styk- og pluklister, samlevejledning, faktura, ordrebekræftelse mv., så manglende dokumenter undgås.

22. *Vis fejl i carportforespørgsel:*

- Som indlogget medarbejder i trælasten vil jeg kunne se fejl i carportforespørgslen automatisk, så risikoen for at overse en fejl mindskes.

23. *Se ordrestatus:*

- Som indlogget medarbejder i trælasten vil jeg til enhver tid kunne se hvor langt i processen, en ordre er nået, så jeg kan svare kunden ved en henvendelse.

24. *Log ind:*

- Som medarbejder i trælasten vil jeg kunne logge ind i systemet for at udføre administrative opgaver.

25. *Log ud:*

- Som indlogget medarbejder i trælasten vil jeg kunne logge ud af systemet, så min konto ikke misbruges.

2 Product Backlog

ID	Navn	Imp	Est	How to demo	Notes	Derfor!
6	Afsend carport forespørgsel	50	2	Unit test med data for en forespørgsel oprettes, data gemmes i databasen, data fra senest oprettede forespørgsel hentes fra databasen og højde, bredde og længde sammenlignes, disse skal være ens.	Kræver DB, unit test.	I: Ellers tabes mange salg. E: Create

7	Åbn carport-forespørgsel	50	2	Log ind, åbn personlige webside. Link til webside med liste over forespørgsler vælges, en forespørgsel klikkes og webside med formular med forespørgslens indhold vises.	Kræver DB, Login.	I: Ellers er der ingen kunder. E: Read
17	Administrer varer	50	3	Log ind, åbn personlig webside. Link til webside med liste over varer vælges. Vare klikkes, varens redigeringsside åbnes, data indtastes og der trykkes 'gem'. Tjek at varens nye data fremgår af siden med liste over varer.		I: Elementært, for at undgå fejl i priser/beskrivelser mv. og for at kunne samkøre med lagersystem. E: CRUD
10	Beregn stykliste	49	5+	Unit test oprettes hvor forespørgsel hentes fra databasen. Baseret på forespørgslens data beregnes styklisten som udskrives i konsollen.	Kræver DB	I: Kan vi ikke gøre dette dynamisk, er det svært at ekspedere carporte i dynamiske str. E: Umiddelbar kompleks, fordrer mere analyse.
13	Tilføj vare fra databasen	45	2	Unit test som opretter tom stykliste og derefter lægger en vare fra databasen i styklisten. Styklisten gemmes i databasen, hentes ind igen og vareantallet sammenlignes med det tilføjede vareantal.		I: For at sikre en korrekt stykliste og mersalg. E: CRU
16	Besvare carport forespørgsel	45	1	Åbn webside med carporte, klik på linket 'Carport med egne mål', tag vælges og carportens mål indtastes. Indtast mail adresse der er adgang til under test. Knappen 'Send forespørgsel' klikkes. Log ind, åbn personlig webside. Link til webside med liste over forespørgsler vælges, den netop oprettede forespørgsel klikkes og webside med formular med forespørgslens indhold vises. Svartekst indtastet i svarfeltet og systemet danner email og vedhæfter automatisk relevante dokumenter. Tjek mailbox for svarmail og vedhæftninger, tjek at forespørgsel skifter status til 'besvaret' i listen over forespørgsler.		I: En obligatorisk del af salgsprocessen. E: Simpel funktion
19	Administrer brugerkonti	45	2	Log ind, åbn personlig webside. Link til webside med liste over		I: Flere brugere sikrer hurtigere

				brugere vælges, fra listen vælges bruger, redigeringsside for brugerkonto vises, data redigeres, knappen 'gem' klikkes, tjek at nye data fremgår af listen over brugere.		forretningsgang og tidl. Medarbejdere kan nedlægges. E: CRUD
2	Hent/vis tegning	40	2	Websiden med carporte åbnes i browseren, en carport vælges og websiden med carportens detaljer vises. Knappen 'vis tegning' klikkes og tegningen vises i browseren.		I: Vigtig for at kunne danne sig et indtryk. E: Ingen db, blot grafik.
3	Vis carport	40	1	Websiden med carporte åbnes i browseren, knappen 'vis carport' eller billedet af en carport klikkes og carportens detailside vises.		I: Vigtig for at kunne danne sig et indtryk. E: Simpelt db udtræk
5	Design egen carport	40	2	Websiden med carporte åbnes i browseren, linket 'carport i tilpassede mål' klikkes, side for valg af tagtype vises, tagtype vælges, side med formular for carportens længde, højde og bredde vises og mål indtastes.		I: Det der adskiller Fog fra andre trælaster. E: Formular skal laves.
8	Konfigurer carport	40	2	Unit test oprettes som henter en forespørgsel fra databasen. Data ændres og gemmes i databasen. Forespørgslen hentes fra databasen igen og der testes om nye data stemmer med den foretagede ændring.	Kræver DB, unit test.	I: Så der kan rettes fejl og kundens tilfredshed og succes sikres. E: RU.
15	Se beskrivelse	40	1	Log ind, åbn personlige webside. Link til webside med liste over forespørgsler vælges, en forespørgsel klikkes og webside med formular med forespørgslens indhold vises. Knappen 'vis beskrivelse' klikkes, beskrivelsen vises i browseren.		I: Så medarbejderen kan sikre salget ved hurtigt at kunne besvare spørgsmål. E: R
24	Log ind	40	1	Login websiden åbnes i browseren, data indtastes og brugerens personlige webside vises. Ved fejlet login vises fejlbesked på login websiden.	Kræver DB	I: For at sikre at vitale dele af systemet kun kan benyttes af godkendte brugere. E: RU
25	Log ud	40	1	Login websiden åbnes i browseren, data indtastes og brugerens personlige webside vises. Øverst i websiden klikkes på 'log ud'-knappen og login siden vises.	Kræver DB og #26.	I: For at sikre at konti ikke misbruges.. E: RU
18	Nulstil brugerkonto	38	2	Side med loginformular åbnes, email indtastes, knappen 'nulstil brugerkonto' klikkes. Tjek at mail med ny login fremsendes.		I: Så man kan komme på systemet og betjene kunder mv. E: RU og mail.

23	Se ordrestatus	38	1	Kør User Story 7, se forespørgslernes forskellige statusser.		I: For nemt at kunne følge fremdrift. E: U
14	Se/udskriv tegning	35	2	Log ind, åbn personlige webside. Link til webside med liste over forespørgsler vælges, en forespørgsel klikkes og webside med formular med forespørgslens indhold vises. Knappen 'vis tegning' klikkes og tegning af carporten åbnes i browseren.		I: God visuali-sering er vigtig for kunden og dermed salget. E: Ingen db, blot grafik.
21	Udskriv ordre-dokumenter	34	1	Kør User Story 7, fanen 'Dokumenter' vælges, ønskede dokumenter vælges, knappen 'udskriv' klikkes, tjek at dokumenterne udskrives.		I: En del af pakke-processen. E:
1	Filtrer carport	27	2	Websiden med carporte åbnes i browseren, filtre sættes og siden opdateres med carporte der matcher kriterierne eller søgeord.		I: For nemmere at kunne finde den ønskede carport. E: R
22	Vis fejl i carport forespørgsel	25	4	Kør User Story 7, forespørgsel vælges, carportens konfigurationsside vises, tjek at der fremgår en fejltekst i tekstfeltet.		I: Ikke vigtig nu, men rart senere. E: Kræver algoritme ift. regler. Yderligere analyse påkrævet.
4	Se leveringspris	20	1	Websiden med carporte åbnes i browseren, en carport vælges og carportens detailside vises. Knappen 'vis leveringspris' klikkes og siden opdateres med leveringspris.	Findes dog på kurven…	I: Godt at have med når man tager beslutningen. E: Nem beregning
9	Administrer kunde	20	1	Log ind, åbn personlig webside. Link til webside med liste over kunder vælges. En kunde klikkes og webside med formular med kundens data vises. Kundens data ændres og knappen 'gem' klikkes. Kundens nye data fremgår af formularen.	Kræver DB, unit test.	I: Vigtig for levering og vedligehold af stamdata. E: RUD
11	Rediger dækningsgrad	20	1	Unit test oprettes hvor forespørgslens data hentes fra databasen. Data for forespørgslens dækningsgrad ændres og gemmes i databasen. Forespørgslens data hentes igen og data for dækningsgraden sammenlignes med ændringen.		I: Har kun lille betydning for salget. E: RU
12	Rediger hjælpetekst	15	1	Unit test indhenter data for en vare fra databasen. Varens hjælpetekst ændres og data gemmes i databasen igen. Varens data hentes igen og data for	Samme som 19?	I: Har ingen betydning for salget. E: RU

				varens hjælpetekst sammenlignes med ændringen.	
20	Administrer samlevejledning	10	3	Log ind, åbn personlig webside. Link til webside med liste over samlevejledninger åbnes, samlevejledning vælges, redigeringsformular for samlevejledning vises, indhold ændres, knappen 'gem' klikkes, tjek at samlevejledningens indhold er ændret.	I: En sekundær nice-to-have funktion. Ikke vigtig i salgsproces. E: Kræver omstrukturering af samtlige dokumenter.

OBS: Vi havde en snak med Ronni 09/11-2018 inden PO møde, hvor han fortalte, at afhængigheder ikke altid kan undgås. F.eks. user stories som kræver login. Måden man kan gøre disse uafhængige er at hardcode en indlogget bruger og så udføre testen. Alternativt kan man undlade at skrive 'log in' i user storien, og så senere ændre de user stories, som flyttes "bag" en loginfunktion, f.eks. administrative user stories.

3 Tidlig featureliste

1. Brugeradministration med mulighed for oprettelse og redigering af brugere samt nulstilling af kodeord.
2. Ordreforespørgsler lagres i databasen så dobbeltindtastning elimineres.
3. Mulighed for redigering af ordreforspørgsler så evt. fejl kan rettes.
4. Mulighed for at redigere i regler bag materialeudregning aht. ændrede byggekrav.
5. 3D rendering af carport i frontend.
6. Mulighed for redigering af varer.
7. Mulighed for fuldstændig konfiguration af carport på frontend, dvs. mål, beklædning, belægning, evt. fliser, taghældning.

4 Daily Scrum meetings outcome and decisions:

4.1 13-11-2018:

Efter en famlende mandag, hvor vi fik taget hul på databasen sammen, blev vi enige om at tage hul på US#6 og vi formulerede sammen de identificerede tasks. Disse blev fordelt således:

Task nr.	Beskrivelse	Udvikler
29	Opret JUnit som tester funktionaliteten	Claus
30	Opret sql til indsættelse af forespørgsel i db	Jesper
31	Opret DAO som udfører indsættelsen	Jesper
32	Opret tabellerne kunde, postnr, tag, skur, forespørgsel i databasen	Claus
33	Opret DTO til forespørgsel	Jesper
34	Connector klasse til databasen	Claus

Efter fordeling blev vi enige om at snakke sammen telefonisk et par timer senere, idet vi arbejdede hjemmefra for at få ro. Vi har besluttet at medtage en række spørgsmål til afklaring ved technical review 14-11, inden vi går i gang med de næste user stories i sprint 1.

4.2 14-11-2018:

Taiga har vist sig at være godt til at holde sporet, så ikke der laves for meget andet. Der henstår lidt smårettelser i koden som vi skal se på sammen. Vi kom ikke helt i mål med dagens opgaver fordi vi undervurderede de fornødne, indledende opgaver som f.eks. dummy data i databasen, færdiggørelse af samme, opsætning af DbConnector, github mv.

Vi har besluttet at lave de ting færdige der mangler fra i går sammen, da der mangler lidt dialog om problemstillingerne. Vi forfatter nogle spørgsmål til teknisk review i dag, så vi kan få yderligere klarhed om f.eks. database, brug af branching i github mv.

Vi venter med at tildele hinanden flere tasks, da vi først må have færdiggjort opgaver fra i går.

4.3 15-11-2018:

Vi har fået lavet manglerne fra 13-11 færdige. Efter technical review 14-11 har vi besluttet at opdele US#10 i 3 mindre, realistiske delleverancer, således at udregninger implementeres i flg. rækkefølge: stolper i sprint 1, tag i sprint 2, skur i sprint 3.

Herudover har vi besluttet at US#17 vedr. administration af varer, må rykkes til et senere sprint.

Dagens indsats koncentreres om US#7 hvor vi åbner forespørgsler. De fornødne tasks har vi defineret og skal fordeles således:

Task nr.	Beskrivelse	Udvikler
42	FrontController oprettes	Claus
43	Command pattern implementeres	Claus
44	Side med liste over forespørgsler	Jesper
45	Side som viser indhold af en forespørgsel	Jesper
46	Div. tilpasninger i ForesporgselDAO	Jesper

4.4 16-11-2018:

Torsdagens tasks viste sig at blive til en del flere og derfor blev torsdagens opgaver fuldført sent. Der henstår fortsat lidt oprydning og optimering i disse tasks, idet vi har haft brug for at få klarhed over og forståelse af arkitekturens ideelle opbygning. Vi besluttede torsdag at tage et hurtigt møde ang. arkitekturen, så fremtidige udfordringer med den mindskes.

I dag, inden PO møde, færdiggøres de 2 views fra i går, dvs. visning af listen over forespørgsler og visningen af en enkelt forespørgsel, således at der linkes til / fra begge sider.

Claus påbegynder udvikling af styklisteberegneren.

Fordelingen af tasks ser derfor således ud:

Task nr.	Beskrivelse	Udvikler
52	Links fra siden med listen til siden med enkelt forespørgsel	Jesper
53	Link fra siden med enkelt forespørgsel tilbage til siden med listen	Jesper
36	Opret klasse i logic lag til udregning	Claus

4.5 16-11-2018 Retrospective af uge 1:

Den første uge var forvirrende og kompliceret, fordi teamet umiddelbart forsøgte at skabe den korrekte database fra starten. Men da teamet samtidig manglede overblik over resten af opgaven, afledte databaseudviklingen flere spørgsmål end svar, især ift. udregning af styklisten. I stedet for at fokusere på problemet, fokuserede vi på løsningen. Således blev uge 1 en utilfredsstillende oplevelse med manglende fremdrift.

4.6 19-11-2018:

Der henstår et par opgaver fra sidste uge, som vi må gøre færdige i denne. Jesper tager sig af vareadministration og Claus af styklisteberegneren:

Task nr.	Beskrivelse	Udvikler
56	SQL til hentning af alle varer	Jesper
57	SQL til at gemme vare	Jesper
58	SQL til hentning af enkelt vare	Jesper
59	Metode som henter alle varer	Jesper
60	Metode som henter en enkelt vare	Jesper
61	Metode som gemmer vare	Jesper
62	JUnit test af 59, 60, 61	Jesper
63	JSP til visning af alle varer på liste (som v. forespørgsler)	Jesper
64	JSP til visning af formular til enkelt vare	Jesper
36	Opret klasse i logic lag til udregning	Claus
39	Opret Unit test til demo af hver udregner	Claus

4.7 20-11-2018:

Jesper er færdig med sine opgaver fra i går (56-64), og tester i dag JSP siderne for funktionsfejl. Claus har lavet en udregner til rem, udregnere til spær og stolper udvikles i dag.

4.8 21-11-2018:

Jesper har rettet småfejl i sine views fra mandag (56-64) så de nu er klar til fremvisning på Technical Review i dag. Jesper ser på Task 70 for at igangsætte udvikling af tegningen. Claus har udviklet de første 4 beregnere til styklisten og laver en side så styklisten kan vises.

Task nr.	Beskrivelse	Udvikler
70	Demo tegning for at skabe overblik.	Jesper
71	Side til visning af stykliste for en forespørgsel.	Claus

Inden review i dag demoer vi for hinanden.

4.9 22-11-2018:

Der var udfordringer med at få demo-tegning til at virke i går, derfor går vi i dag sammen om at udvikle det grundlæggende for at kunne lave en demo-tegning færdig i dag. Reviewet i går affødte en del tanker vedr. optimering af vores styklisteberegner og navngivning generelt. Styklisteberegneren virker fornuftigt for en version 0.8, siden som viser resultatet blev også klar i går, om end den savner design.

Jesper kører videre i SVG-sporet, så vi kan demonstrere en tegning.

4.10 23-11-2018:

SVG har drillet i et par dage, og vi vil gerne have det klar til en demo og evt. koordinater fra styklisteberegneren i løbet af næste uge, derfor går vi sammen om at få en demo tegning klar i dag.

Udregning af tagmaterialer har givet anledning til udvidelse af database, idet tagtyper er forskellige aht. hældning og fordi tagtyper består af forskellige materialer. Til PO møde får vi kastet yderligere lys over dette. Rooftype og RooftypeMaterial tabel er oprettet for at håndtere udfordringen.

4.11 23-11-2018 Retrospective af uge 2:

En bedre uge hvor vi havde mere klarhed over opgaverne idet PO stillede klare krav til efterfølgende fredag (23-11), hvor bl.a. styklisteberegneren skulle være færdig. Technical Review og PO-møde gav også klarhed over forventningerne, så vi bedre kunne retfærdiggøre vores afgrænsninger i projektet. Styklisteberegneren blev således 80% færdig til i dag, men afledte også en ændring i databasen, som igen gav anledning til flere overvejelser. Med afgrænsningen i baghovedet kunne vi hurtigt begrænse omfanget af ændringen.

4.12 26-11-2018:

Ej afholdt pga. undervisning.

4.13 27-11-2018:

Vi har i dag aftalt at hjælpes ad med at få den sidste funktionalitet på plads ift. tegning med svg, så vi er klar til at modtage en række rektangler fra materiale udregningen. Jesper ser herudover på carport konfiguratoren, så Martin, kunden m.fl. kan indtaste og gemme hhv. beregne stykliste direkte. Claus går i gang med beregning af tag.

4.14 28-11-2018:

Forespørgselsområde blev ikke færdigt i går, Jesper arbejder videre hermed. Claus færdiggør tagberegning og igangsætter udregning af koordinater til tegning.

Vi har udfordringer med tidsplan, især ift. manglende analysearbejde, vi tager en snak til Technical Review i dag for at se på muligheder for at imødegå disse.

4.15 29-11-2018:

Forespørgselsområdet mangler lidt små justeringer og at kunne sende forespørgselsdata direkte til beregneren uden at gemme forespørgslen i databasen. Jesper arbejder videre hermed. Claus fik optimeret koden mht. at finde materiale i nødvendig længde, således at en sortering af materialeisten ikke er nødvendig længere. Således er koden optimeret i alle beregnere. I dag gøres tagberegning færdig.

Vi skal sammen have set på hvordan vi sikrer at både skur og forespørgsel oprettes/redigeres eller ingen af delene (commit / rollback).

4.16 30-11-2018:

Forespørgselsområdet mangler drop down for valg af tagtype til carport. Jesper ser på dette i dag. Tagberegneren blev færdig i går og grundlaget for tegningen er lagt, så der i dag kan oprettes forskellige del-tegnere. Dette ser Claus på.

4.17 30-11-2018 Retrospective af uge3:

En mere kaotisk uge fordi sprint 2 gik lidt ind i sprint 3 og fordi der ingen nye User Stories var i sprint 3. Således blev vi fanget i at fortsætte udviklingen af de i gangværende opgaver, og fremdriften led lidt. På den anden side blev de igangværende opgaver færdige, og huller således lukket.

4.18 3-12-2018:

Intet scrum pga. undervisning. Vi fik dog efterfølgende kigget på transaktioner i sql, da både skur og carportforspørgsel skal oprettes/opdateres eller ingen af delene.

4.19 4-12-2018:

Transaktion i SQL drillede, idet vi ikke var klar over at et ResultSet skal lukkes, inden næste sql udtryk køres. Derfor hang transaktionen. Jesper har løst dette i går aftes, samt div. opdateringer i layoutet. Claus fik de sidste udregninger klar vedr. stolper hvor rem brydes og rem hvor spær brydes. Disse skal bruges til korrekt tegning. Materialer har fået price-attribut i databasen samt i koden, pris udregnes og vises på styklisten.

Jesper ser på drop-down til tagtyper i forespørgselsområdet, Claus færdiggør tegning.

4.20 5-12-2018:

Forespørgselsområdet er færdigt, tagtyper kan vælges og forespørgsler kan oprettes og ændres. Skur kan til- og fravælges. Tegningen mangler blot skur.

Jesper går i gang med user story #19 – Administrer brugerkonti, først ændres Customers tabellen til også at indeholde password og brugertype/rank. Dernæst DTO og DAO.

Claus tegner skur og går i gang med unit tests.

4.21 6-12-2018:

I går fik Jesper ændret Customers tabellen til Users og tilføjet passwords mv. Data er lagt i postnumre og hentning af brugere fra databasen er påbegyndt i UsersDAO. Login-side og opret-bruger-side er påbegyndt.

Claus blev færdig med tegning af carporten: Stolper, spær, skur og rem tegnes. De første unit tests af div. udregnere er påbegyndt.

I dag fortsætter Jesper arbejdet med opret bruger-delen og login-delen, så det bliver færdigt. Unit tests laver Claus færdige i dag.

4.22 7-12-2018:

Jesper har færdiggjort login / log ud funktionalitet samt opret bruger delen. Claus har lavet unit tests og er i gang med integration tests.

I dag laver Jesper #115 vedr. admin sider der skal beskyttes af login, samt #18 hvor brugerkonto nulstilles.

Claus laver #73/#87 vedr. integration tests færdig og ser herefter på #86 exceptions og logging.

4.23 7-12-2018 Retrospective af uge 4:

En langt bedre uge hvor vi havde mange definerede opgaver opdelt ordentligt i tasks. Her fik vi endelig ordentlig fremdrift og kunne fornemme at opgaver blev lukket rigtigt i ugens løb.

4.24 10-12-2018:

Jesper fortsætter med administration af brugerkonti, så man kan ned/opgradere rank for brugere. Koden skal tage højde for at sidste administrator ikke nedgraderes. Nulstilling af brugerkonto implementeres også.

Claus laver interface over DataFacade og exception handling gøres færdig i dag.

4.25 11-12-2018:

Brugerkontoadministration er klar, så indlogget bruger kan ændre sit password. Indlogget administrator kan op/nedgradere rank, nulstille kodeord og slette brugere. Sidste administrator kan ikke nedgraderes og man kan ikke slette egen konto. Jesper ser i dag på validering af brugerinput client side, Claus når at gøre exception handling og logging færdig i dag.

4.26 12-12-2018:

Jesper færdiggjorde validering af brugerinput client side i går, Claus fik gjort logging og exception handling færdig. Logging er dog kun testet på lokal maskine da vi endnu ikke har udrullet til server.

I dag starter vi på rapportskrivning, Claus ser på Indledning, baggrund, krav mv. Jesper skriver om status på implementationen og om processen.

4.27 13-12-2018:

Claus: Indledning og baggrund er i gang fra i går, Krav er påbegyndt. Jesper er i gang med punkterne fra i går, vi afventer svar på mail fra Ronni vedr. status på implementationen, men vi har besluttet at liste både hvad vi nåede og hvad vi ikke nåede, ellers kan man ikke se hvor langt vi kom.

Vi har i dag fået lagt rapportens delelementer i scrum boardet, så vi bedre kan følge fremdriften.