Fog trælast (carport afd.)

# 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. 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.

## 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 sin 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 udviklet til styresystemet Microsoft Windows XP, som ikke længere supporteres af Microsoft. Softwaren skønnes derfor sårbar, da den kræver et Windows XP miljø til at kunne afvikles, og det kan med tiden blive stadig vanskeligere at finde.

# 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 ngl. medarbejder).

* **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).

# 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 indenfor 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 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? |

|  |  |  |
| --- | --- | --- |
| The binaries | 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. |

# 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.

# 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.

## Teknologivalg

Java, html, jsp, mysql og så fremdeles. Husk versionsnumre.

# 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.

### 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 for at frembringe en liste med kandidatklasser, som danner grundlag for domænemodellen, se afsnittet herom.

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.

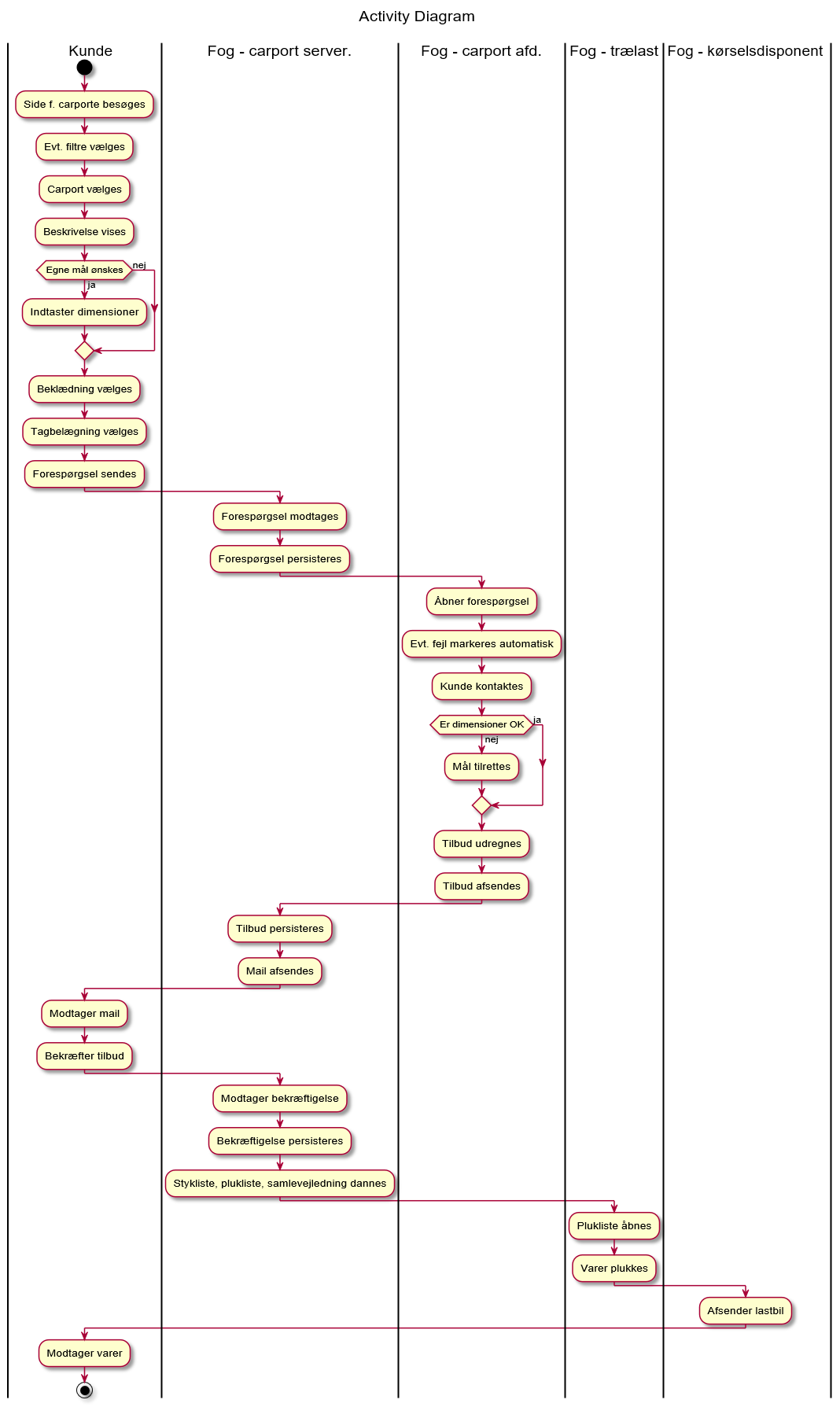
Medarbejderen i trælasten skal foretage mange skridt, før et tilbud kan afsendes. Nogle af disse skal kunden fremover selv foretage, inden han/hun afsender en forespørgsel, nemlig valg af tagbelægning og beklædning. Andre skridt kan håndteres elektronisk.

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 brugsmønstre, som tilsammen udgør den overordnede arbejdsgang, 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 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 |
| Ingen tasks – da denne User Story ikke er planlagt. | | | | | | |

…

Den fulde backlog kan ses i appendix …x

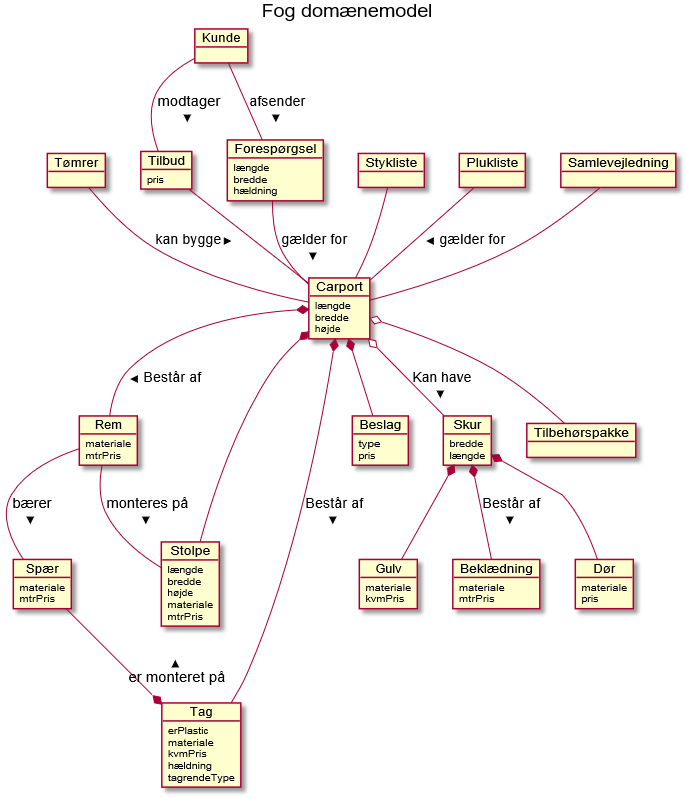
# Domænemodel og ER diagram

Som nævnt i afsnittet ’Krav’, har vi afledt en række kandidatklasser fra de funktionelle krav. I listen findes også en række beregningsfunktioner med høj kompleksitet. Disse har vi forsøgt at deltaljere vha. pseudokode, se afsnittet om kode, 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.

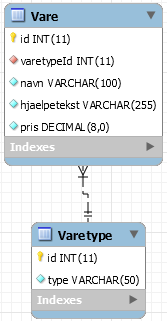


Figur 4 – Domænemodel over problemområdet.

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 (INVEST?) således at disse kan udvikles først. Det 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.

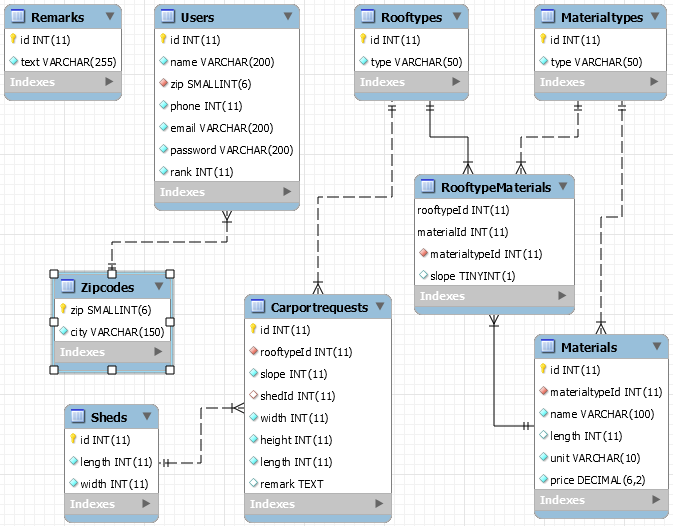
For at kunne udvikle de user stories, der giver mest værdi, bliver databasen også udviklet iterativt. Således havde vores første ER diagram blot 2 tabeller.



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 indført en relation til tabel med typer; Varetype.

Siden er vores database blevet udvidet i takt med 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 intieres 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.

## 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 email 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 ’begrænsninger – 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. |

### Begrænsninger.

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

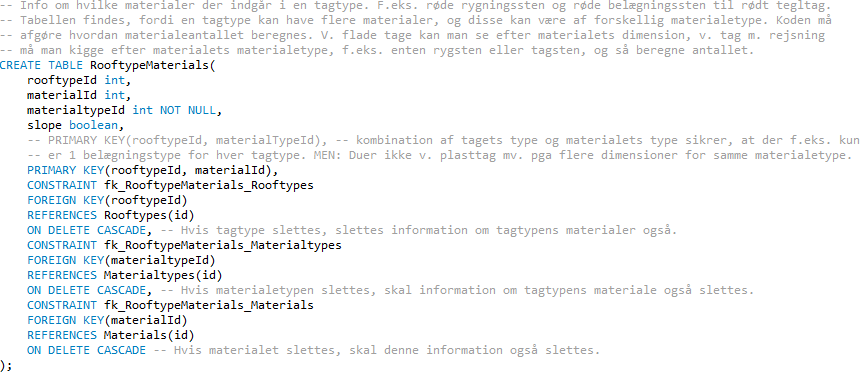
#### Tabellen Carportrequests.



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.

#### Tabellen RooftypeMaterials.

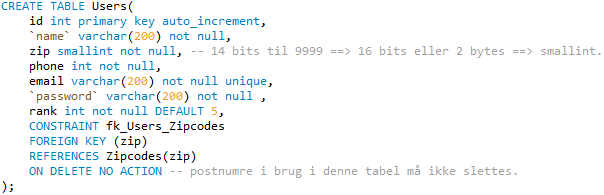


Figur 8 - SQL sætning for oprettelse af tabellen RooftypeMaterials.

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.

#### Tabellen Users.

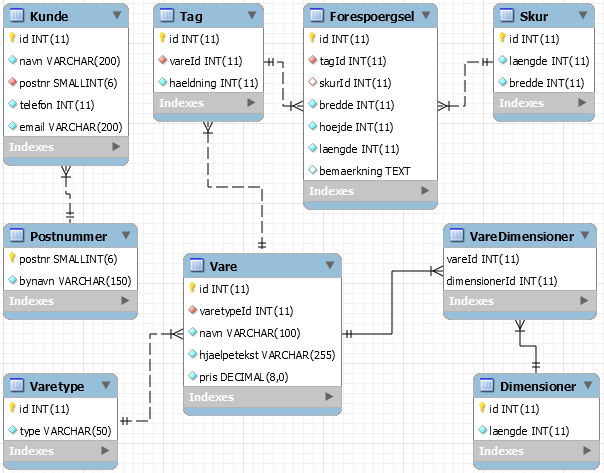


Figur 9 - SQL sætning for oprettelse af tabellen Users

Begrænsningen fk\_Users\_Zipcodes sørger for at postnumre i brug i Users ikke slettes. I virkeligheden er der sjældent brug for at slette postnumre.

SENERE Vedr MAterials tabellen:

Attributten length angiver varens 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.

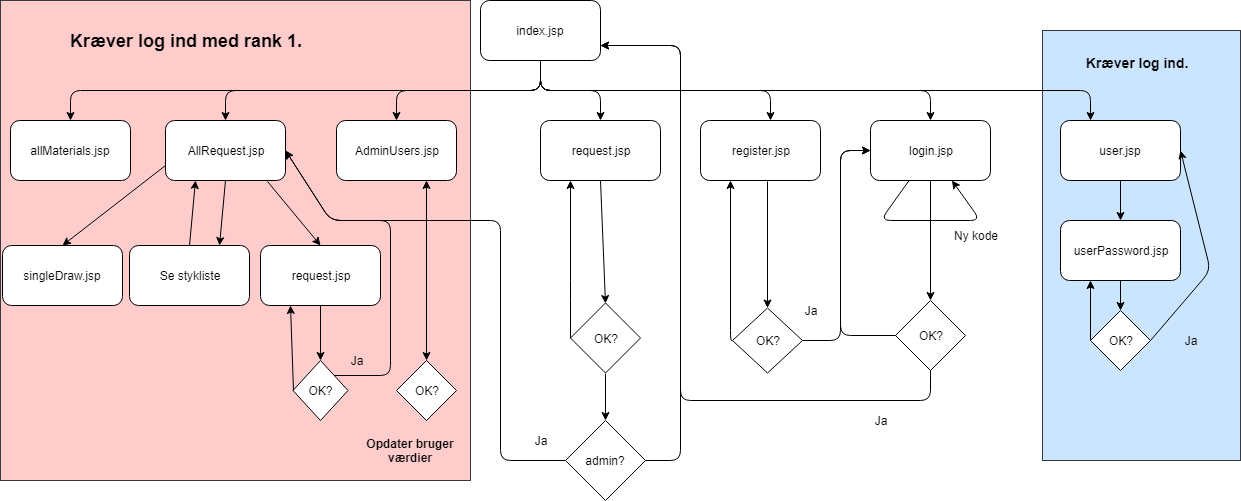


Figur 10 - Databasen med tabeller for varers dimensioner.

# Navigationsdiagram

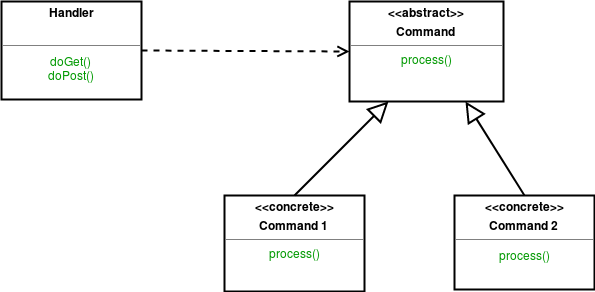
Webapplikationen består af en navigationsbar på alle sider. Fra den vil der være forskellige muligheder for navigation afhængig af om man er logget ind samt med hvilken rank.

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.

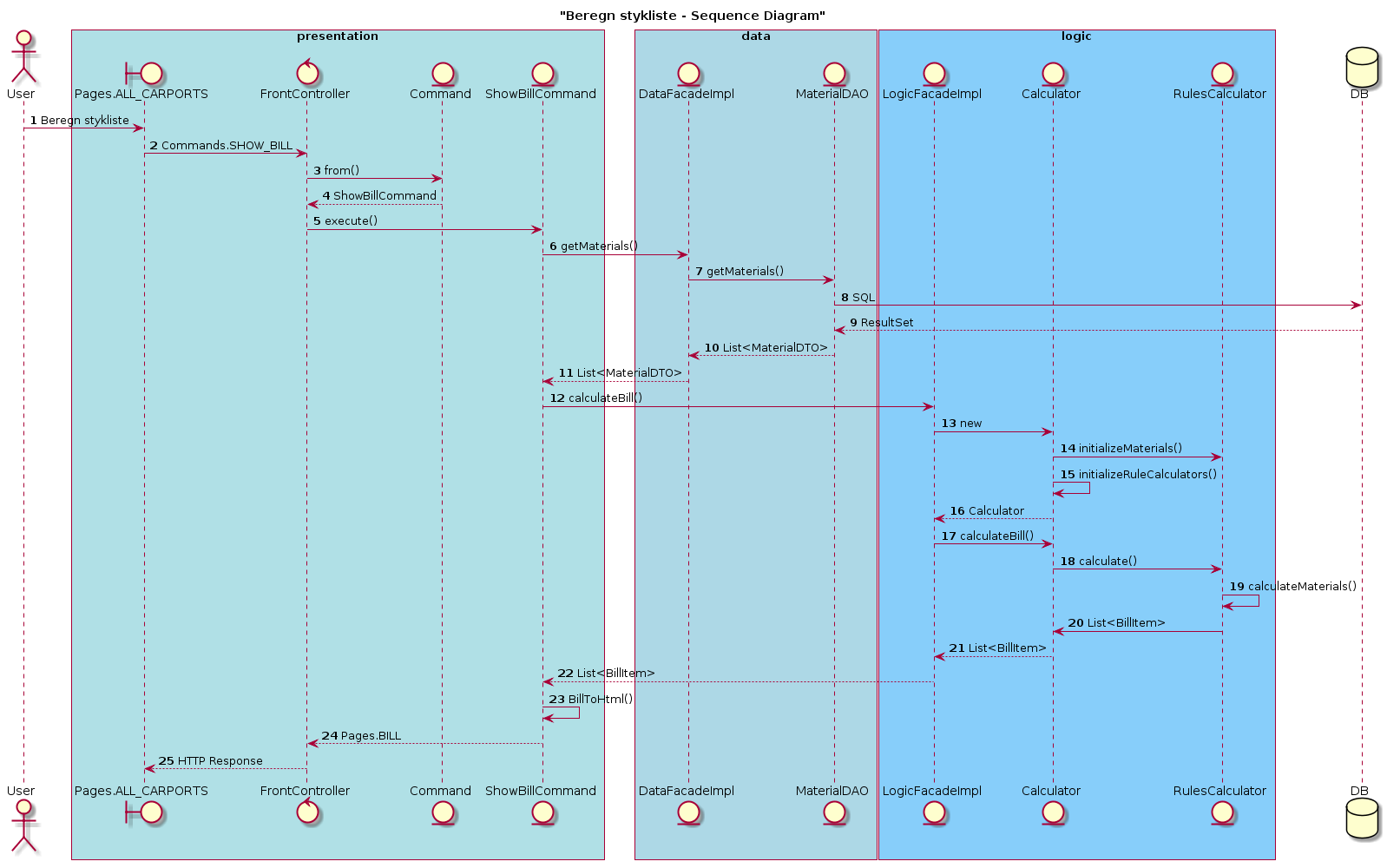


Figur 12 - FrontController pattern med Command objekter

For at forenkle udviklingen, har vi samlet en række konstanter for hver Command og hver resulterende side i klasserne Commands og Pages. Dette er også med til at sikre, at der er konsistens ift. HERTIL. .Se mere i afsnittet ’Udvalgte kodeeksempler’.

# Sekvensdiagrammer

## Beregning af stykliste.



Figur 13 - Sekvensdiagram for beregning af stykliste.

I figur 11, 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).

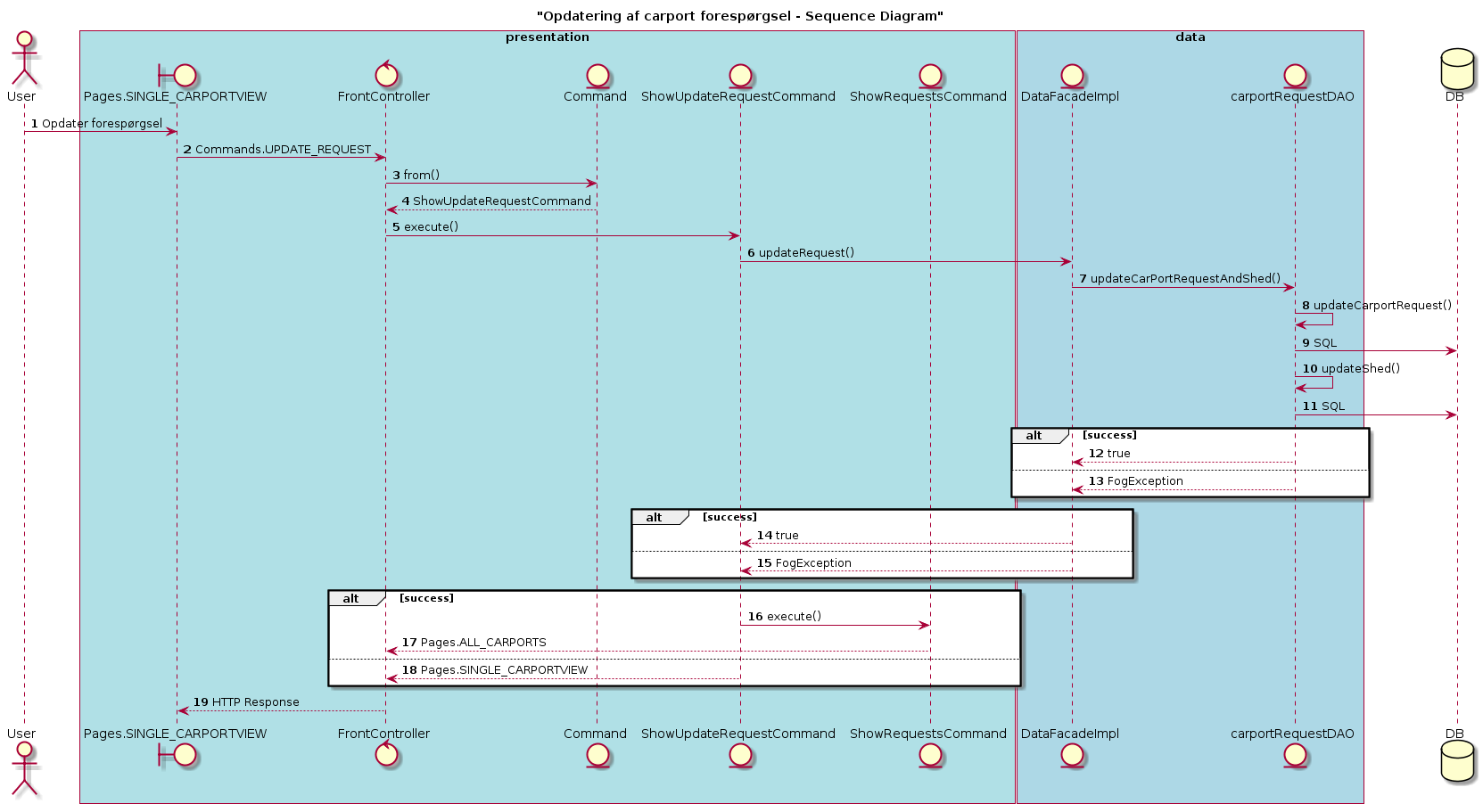
Logicfacaden bruger et Calculator objekt(13), som initialiserer et hashmap af materialer(14), til at beregne styklisten (17). Calculator objektet benytter objekter af RulesCalculator typen til at udregne hver del af carporten (18). Der er således flere RulesCalculators end der er illustreret på diagrammet.

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.

## Opdatering af carportforespørgsel.



Figur 14 - opdatering af carport forespørgsel.

I figur 12 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 om kode.

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 (se også afsnittet om kode) 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.

# Særlige forhold

## Validering af bruger-input.

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/servlet’en.

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.

## 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.

## Sikkerhed.

Brugerens kodeord er gemt som alm. tekst i databasen. Dette skal naturligvis ændres inden udrulning, hvor et kodeord md5-hashes 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 mail som leder til en login side, hvor man bliver bedt om at oprette en ny, dette er ikke implementeret endnu.

## 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. 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.

## Brugertyper i databasen.

Vi har blot implementeret en enkelt brugertype i databasen og denne har fulde rettigheder på alle tabeller. Man kunne have implementeret forskellige jdbc-brugere svarende til den brugertype, som logger ind på systemet. Således kunne … er det overhovedet smart?

## Kodemæssige forbedringer.

Claus mener…

Navngivning af commands efter en vedtaget konvention, det er svært at gennemskue, hvad de forskellige commands gør. Navnene bør være i tråd med deres konstanter fra Commands-klassen.

Når en bruger ikke kan logge ind, skal han have besked på at noget gik galt. Ikke blot ’no errors’.

Login formularen skal findes i login.jsp. Blot de 2 værdier for command parameteret; Commands.LOGIN\_CHECK og Commands.FORGET\_PASSWORD sendes med som variabler på requestet.

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

Nogle commands kunne nok samles.

Redirect i stedet for Forward i de commands, der opdaterer i databasen, så man ikke risikerer at poste 2 gange ved genindlæsning af siden.

Bedre dokumentation, JavaDoc, af metoder generelt.

### Jesper mener…

# Udvalgte kodeeksempler

## 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 />");



## 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);

## 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);  
}

## 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.

## 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.

## 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;   
}

## 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.

## 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;  
}

SENERE: Smart og Invest ?

KODE: Og vores pseudokode om beregninger.

# Status på implementationen.

Flg. userstories 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

## 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.

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

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

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.

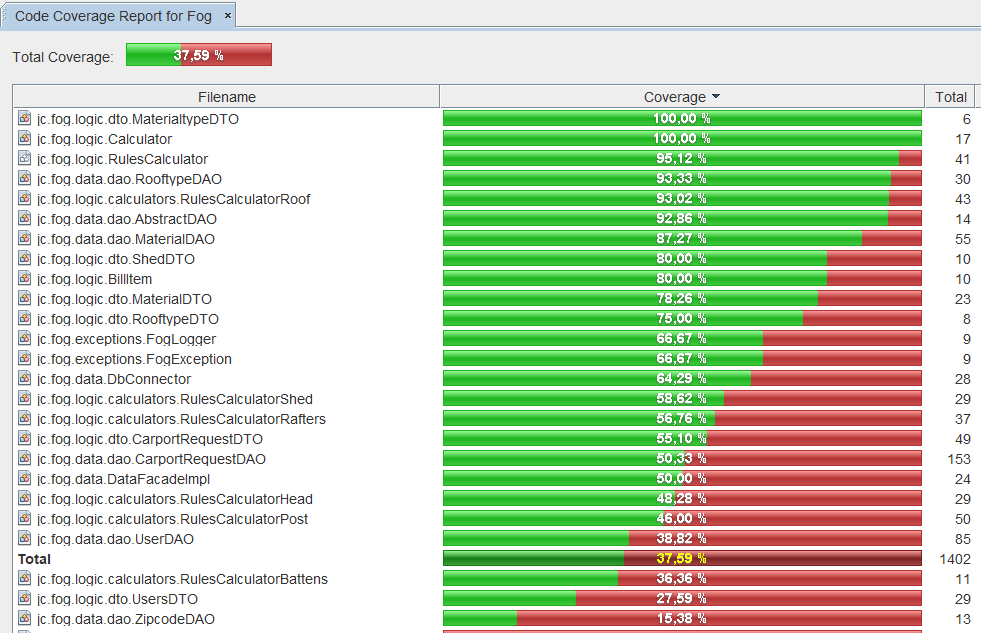
Her kan vi liste hvis vi finder noget når vi går koden igennem onsdag.

# 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. |

## Herunder ses en JaCoCo rapport for projektet.

















## SCRUM user stories:

1. *Filtrer carport:*

* Som kunde vil jeg gerne kunne filtrere blandt carporte for at finde den rette.

1. *Hent/vis tegning:*

* Som kunde vil jeg gerne kunne hente tegninger af en valgt carport for at kunne visualisere den.

1. *Vis carport:*

* Som kunde vil jeg gerne kunne få vist en carport, så jeg kan danne mig et indtryk og læse mere om den.

1. *Vis leveringspris:*

* Som kunde ønsker jeg at kunne se leveringsprisen for bedre at kunne beslutte mig.

1. *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…)

1. *Afsend carportforespørgsel:*

* Som kunde vil jeg gerne kunne forespørge på en carport jeg selv har designet.

1. *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.

1. *Konfigurer carport:*

* Som indlogget medarbejder i trælasten vil jeg kunne konfigurere en carport således at fejl evt. undgås.

1. *Administrer kunde:*

* Som indlogget medarbejder i trælasten vil jeg kunne administrere kunder således at stamdata er korrekte til sikring af f.eks. korrekt leveringsadresse.

1. *Beregn stykliste:*

* Som indlogget medarbejder i trælasten vil jeg kunne beregne styklisten automatisk, således at fejl undgås, priser bliver korrekte og lagerbeholdning vedligeholdes korrekt.

1. *Rediger dækningsgrad:*

* Som indlogget medarbejder i trælasten vil jeg kunne redigere dækningsgraden så jeg kan lave et godt tilbud hvis der er behov.

1. *Rediger hjælpetekst:*

* Som indlogget medarbejder i trælasten vil jeg kunne redigere hjælpeteksten så kunden har bedre mulighed for at samle carporten korrekt.

1. *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?

1. *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.

1. *Se beskrivelse:*

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

1. *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.

1. *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.

1. *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.

1. *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.

1. *Administrer samlevejledninger:*

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

1. *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.

1. *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.

1. *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.

1. *Log ind:*

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

1. *Log ud:*

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

# 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 brugere vælges, fra listen vælges bruger, redigeringsside for brugerkonto vises, data redigeres, knappen ’gem’ klikkes, tjek at nye data fremgår af listen over brugere. |  | I: Flere brugere sikrer hurtigere 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 varens hjælpetekst sammenlignes med ændringen. | Samme som 19? | I: Har ingen betydning for salget.  E: RU |
| 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.

# 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.

# Spørgsmål

Findes der andre der kan administrere systemet end Martin/ngl. medarbejdere?

Hvor meget ved de alm. medarbejdere om systemet ift. Martin?

Hvad sker der når Martin er fraværende og en kunde henvender sig / har lavet en forespørgsel?

Er der tiltag ift. hvis systemet en dag ikke kan køre pga. opdateringer i OS?

Er der taget endelig beslutning om hvorvidt man vil åbne op for valgmulighed for kunderne på bekostning af den nuværende ”enkelthed”?

Hvad er forskellen på stykliste og pakkeliste?

Ville en videreudvikling af det nu 7 år gamle system, der ligger ubrugt hen, kunne løse noget?

Ad ovenstående: Er Fog klar reelt klar over hvorfor det ikke virker? Er der adgang til kildekode mv.?

Hvor dynamisk er markedet ift. reguleringer, udvikling af nye produkter, udfasning af andre, f.eks. plastictage aht. miljø, stærkere beslag aht. flere og kraftigere storme mv.?

## Spørgsmål undervejs – kan måske inspirere til rapportindhold:

Må en dto indeholde en anden dto hvor det giver mening eller er det i strid med mønsteret? Ja naturligvis, hvis det giver konceptuel mening, derfor har vi ladet VareDTO indeholde samlinger af andre DTO’s.. (Er vi sikre på at det er VareDTO?)

En branch til hver udvikler eller til hver feature? Hvis vi arbejder på samme feature er det måske bedre med en branch til hver feature jf. Ronni (17-11-18)

Må commands gerne have funktionalitet til dannelse af html tabel? Ja, så længe commands er i presentation laget. Husk – og skriv – at presentation må gerne indeholde HTML, ingen logik og ingen sql. Logic laget har logik (business logic), ingen html og ingen sql. Data-laget har sql og ingen html.

Vi bør analysere lidt mere så vi har indhold ift. diagrammer mv. F.eks. tilstandsdiagram for forespørgsel, måden vi opretter kunden på (altså anonym forespørgsel med kundedata bliver til en forespørgsel og en kunde i databasen) osv.

Arkitekturen skal have et diagram og beskrives naturligvis.

Ifm. tests sender vi en CarportRequestDTO til styklisteberegneren. Måske skulle styklisteberegneren forvente simple datatyper som argumenter i stedet for et objekt. Så slipper vi for at erklære objekter heraf i unit tests. Men hvad er teorien for DTO’s / testbare metoder?

## Yderligere info til rapport:

Idet udregningerne er opdelte, er det let at tilføje yderligere regler til indholdet i styklisten. F.eks. udregner stolpeudregneren de krævede stolper i hjørnet + skur. Hvis man senere ville have en business rule implementeret om at afstand mellem stolper ikke må overskride 2 mtr, kan disse ekstra stolper nemt udregnes i en separat udregner…

Husk at skrive at logikken må styre at der ikke kommer f.eks. flere materialer af typen tagfladebelægning på samme tag, fordi sammensat primærngl af roofTypeId og materialTypeId ikke duer, fordi plasttag findes i 2 længder og derfor begge tilhører roofType med fladt tag og er af samme materialetype.

Statisk vs. Dynamisk: Husk et afsnit om udregnerne og deres design, idet: RuleCalculator har en statisk metode, initializeMaterials() som opretter et statisk HashMap som deles mellem alle RuleCalculator-implementationer. Oprindeligt stod Calculator-klassen for blot at oprette objekter af de forsk. RuleCalculator-implementationer og de stod så selv for at oprette det statiske HashMap på super klassen RuleCalculator. Men da hashmappet er statisk, oprettes der ikke et nyt, når en ny stribe RuleCalculator-implementationer blev oprettet af Calculator-klassen. Og det duer ikke hvis man gerne vil teste udregningerne med en ny stribe materialer. Derfor blev RuleCalculator.initializeMaterials() oprettet og den skal således kaldes inden udregninger kan ske. Dette muliggør også test af hver enkelt rulecalculator-implementation i et fejl-scenarie hvor materialelisten kan sættes til at være tom, hvilket medfører et tomt hashmap…

Mht. implementering af transaktioner i databasen lærte vi at et ResultSet SKAL lukkes inden næste statement udføres mod databasen, idet dette sidste statement ellers vil vente på at ResultSet lukkes indtil Lock timeout sker.

Vi kunne have ladet calculateMaterials på RuleCalculators være statiske, så kunne de have benyttet hinanden på tværs, f.eks. når rem/stolper skal tegnes på baggrund af hvor spær er brudt.

V. udregning af beklædningsbrædder, bør man evaluere på brættets dimensioner i stedet for blot en string, for at sikre, at materialet er korrekt.

Når vi eks. Nulstiller konto, returnerer vi resultatet af et andet command fra ForgetPassword commandet. Vi har overvejet at forwarde requestet i stedet, hvad er for og imod her? Kan vi skrive lidt om det?

Skriv om commit atomar.. Skriv om afvejning ift. calculators i drawers, vi beregner 2 gange kontra styklisten til drawers...

Scrum-delen:

Har scrum master fungeret?

Det er svært at sige om Scrum Master har fungeret. Det kræver at vi har en erfaring for hvad det vil sige. Eller sagt med andre ord - det kan have virket, uden vi ved det.

Vores fornemmelse er, at det har været fint med en Scrum Master - eller rettere, det holdepunkt, at der var et daily standup møde, hvor man skulle fortælle gårsdagen bragte og hvad man laver i dag. Det gav fornemmelsen af at være på et hold, at andre afhang af ens arbejde og at man kunne få hjælp hvis der var brug for det.

Om det var for Scrum Masteren eller blot 'os' at det var mest naturligt at afhjælpe evt. problemer direkte efter daily standup er svært at præcisere, men også det var en stor hjælp. Så snart en udfordring blev afhjulpet, kunne teammedlemmet komme videre og dermed holdet som helhed.

Hvilke problemer så i der og hvad gjorde i for at rette op på det

I takt med stigende erfaring i brug af Taiga og det faktum at nye opgaver/user stories skal lægges op så de kan følges, blev behovet for ordentlig tildeling af points til user stories større. Revidering af points på baggrund af erfaringer fra retrospectives var der også behov for. Dette blev der desværre ikke gjort ret meget væsen af, hvilket betyder at vores arbejdspres til tider var svært at fastsætte. Samtidig er der sprints, som ikke viser den rette arbejdsbyrde der er udført. Her skulle Scrum Master evt. have været mere omhyggelig, det fik vi dog aldrig rettet op på. Dette er formentlig et resultat af at vi ikke havde nogen eksplicit scrum master, som kunne have følt et ansvar for tildelingen. Måske fordi vi blot var 2, sammenholdt med at vi havde rigeligt med opgaver vi var i gang med.

# Daily Scrum meetings outcome and decisions:

## 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.

## 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.

## 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 |

## 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 |

## 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.

## 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 |

## 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.

## 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.

## 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.

## 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.

## 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.

## 26-11-2018:

Ej afholdt pga. undervisning.

## 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.

## 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.

## 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).

## 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.

## 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.

## 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.

## 4-12-2018:

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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 14-12-2018: