Forside

Navn	**E-mail**	**ID**	**Klasse**
August Manniche Møller	cph-am382@cphbuiness.dk	manniche1234	C-klassen
Søren Aagaard Bendtsen	cph-sb442@cphbusiness.dk	saabendtsen	C-klassen
Younes Karim	cph-yk31@cphbusiness.dk	younesk31	C-klassen

Github:	https://github.com/saabendtsen/Carport2Sem
Deployment:	http://shitlord.me:8080/carport/ Sælger navn: admin Sælger kode: 1
Demovideo:	https://youtu.be/-mLg3y41IEk

Rapporten er færdig skrevet d. 27/05/2021.

Indholdsfortegnelse

Indledning

Johannesfog er et større byggecenter og trælast. De sælger blandt andet carporte i faste størrelser samt kunde defineret størrelser. Carporten leveres med en stykliste samt en brugsanvisning til kunden, der selv samler og opsætter carporten.

Vores kunde Fog trælast har bedt om et system til at beregne carport styklister, og lave skitser ud fra en kundes selvvalgte dimensioner. Ifølge vores kontakt hos Fog skal en kunde have mulighed for at indtaste selvvalgte mål for dimensioner på en carport samt et eventuelt redskabsskur.

Systemets opgave er derefter at beregne en stykliste, og tegne en skitse af den valgte carport. Dernæst skal bestillingen blive præsenteret for en sælger, der kontakter kunden og rådgiver kunden inden bestillingen godkendes. Når bestilling godkendes af sælger, skal kunden blive præsenteret for sin endelige størrelse, stykliste, pris og tegning.

Systemet skal fungere gennem en webserver, der kan implementeres på Fogs egen hjemmeside efter produktet er færdigudviklet. Fog har den dag i dag et ældre system, der kan beregne styklister, men kræver meget arbejde fra sælgers side samt mangler vigtige funktioner. Vores nye system er tiltænkt at erstatte Fogs nuværende system.
Fog ønsker at et tilbud skal godkendes af en sælger før bestillingen afsluttes, da det bibeholder kundekontakt. Systemet skal altså ikke være fuldt automatiseret.

Teknologivalg

Der er gjort brug af IntelliJ v.2021.1.1, MySQL v.8.0.22, Datagrip v 21.1.1 samt apache tomcat server v.9.0.44

JDBC er brugt til at oprette forbindelse mellem java og vores MySQL database, som bliver etableret under vores frontController.java fil. Denne har diverse instanser af datamapper(er), som hver har sin funktionalitet/metoder til at hive relevant data ud af databasen og indsætte dette i Html via servlet attributter i tomcat.

Krav

US1 - Small

Som køber Skal jeg kunne trykke på en knap med “egne mål” Så køberen kan skræddersy sin egen carport.

Givet at vores website har en form, når en kunde udfylder formen, så sendes data til databasen.

US2-a - Medium

Som køber skal jeg kunne tilføje/definere et redskabsskur til min carport, så køber kan få et redskabsskur af specifik mål.

Givet at vores køber tilvælger et redskabsskur, når man laver en bestilling. Så skal det være muligt at indtaste mål til dette,

US2-b - Small

Som køber Skal jeg kunne vælge forskellige slags beklædning og størrelse til mit redskabsskur Så køberen kan få et redskabsskur der passer til hans/hendes smag.

Givet at køber ønsker redskabsskur, når kunde tilføjer til ordren. Så skal det være muligt at tilkøbe personlige præferencer i form af beklædning

US3 - Large

Som køber skal jeg kunne hente en pdf Så jeg kan se tegningen af min carport

Givet at køber har udfyldt ønsker til sin carport, når køber skal godkende sin bestilling. Så skal det være muligt at hente en pdf med tegning.

US4 - Medium

Som sælger, skal jeg kunne godkende køberens ordre før han køber det, så sælger kan få kundekontakt og give bedst mulige rådgivning. (Når sælger godkender modtager køber en kvittering, for lille til en US?)

Givet en køber har ekstra behov for vejledning, når køber har lagt en bestilling, så skal sælger kunne se bestillingen og kunne kontakte køber.

US5 - Large

Som sælger skal jeg kunne se brugeren krav, så programmet kan komme med en stykliste.

Givet at sælger kan se købers krav, når køber har lagt en bestilling. så skal køber får en stykliste udregnet af programmet.

US5.a - Medium

Som sælger skal en købers stkliste være synlig når køber har lagt en ordrer, så den godkendte stk liste kan sendes til køber efterfølgende.

Givet at køber har lagt en ordre, skal stkliste være synlig for sælger. Når sælger godkender ordre, så skal stkliste være synlig for køber.

US6 - Small

Som sælger Skal jeg selv kunne ændre i hvor meget carporten skal koste Så jeg kan give rabat.

Givet at sælger ønsker at give en rabat, når en ny ordre er under behandling. Så kan sælger kunne ændre salgsprisen

US7 - Small

Som sælger skal jeg kunne se hvad dækningsbidraget er, så man ikke går for langt ned i pris.

Givet at sælger skal give et tilbud med rabat, når en køber ønsker et tilbud, så skal dækningsgraden være synlig så sælger ikke giver for høj rabat.

US8 - Medium

Som admin skal jeg kunne ændre på varens kostpris, udsalgspris, beskrivelse, så admin kan opdatere systemet.

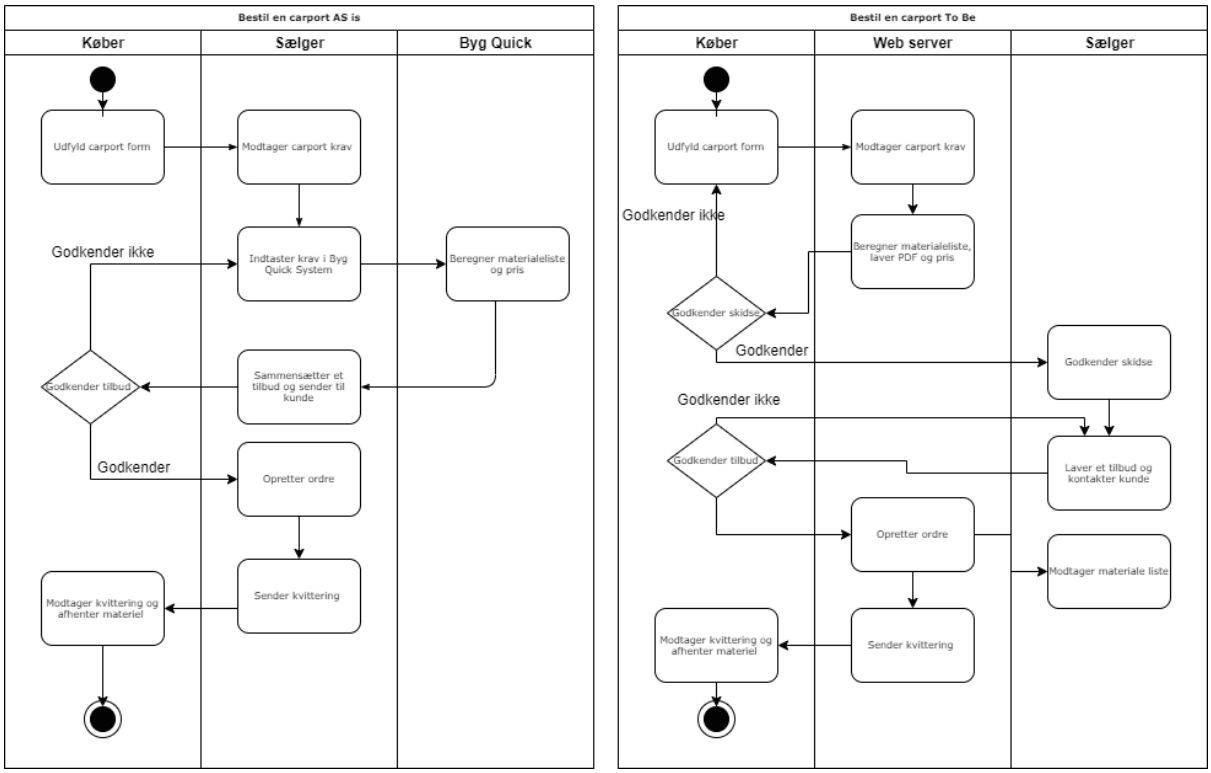
Givet at man er logget på som admin når der er en prisændring så kan admin kunne ændre kostpris, udsalgspris og beskrivelse i systemet.

US9 - Medium (Large)

Som admin skal jeg kunne tilføje og fjerne ting fra systemet, så jeg kan opdatere sortimentet.)

Vi har fravalgt at implementere User Story 8 og 9.
Grundet den smalle tidsramme, valgte vi at lægge fokus på rapportskrivning og debriefing af projektforløbet. Selve implementeringen af disse to user stories ser vi som forholdsvis enkle, og ikke noget nyt stof vi ikke har lavet før. Begge kræver en administratorrolle, der skal have sin egen landingpage hvor det er muligt at lave diverse databasekald ud fra brugerinput.

Arbejdsgange der skal IT-støttes



Workflow as is:
Køber indtaster den ønskede carport størrelse. Sælger indtaster derefter ønskerne i det nuværende beregningssystem "Byg Quick". Byg Quick returnerer en stykliste samt en pris, hvorefter sælger sammensætter tilbuddet. Da "byg Quick" ikke har nye materialer i systemet, indebærer dette, at sælger manuelt beregner fra gamle til nye materialer. Sælger fremsender til sidst det færdige tilbud til kunden.

Kunden vælger derefter at acceptere eller afvise tilbuddet. Accepteres tilbuddet opretter sælger en ordre og sender en kvittering til kunden, der kan afhente materialerne i henhold til styklisten.

Workflow to be:

Køber indtaster ønsker til carport størrelse. Web Serveren modtager de indsendte ønsker, og beregner en materialeliste samt skitse af carporten ud fra dette. Køber modtager skitse og kan vælge at godkende eller at afvise tilbuddet. Ved godkendelse sendes skitse og stykliste videre til sælger, der kontakter kunden for videre rådgivning og laver et endeligt tilbud. Hvis kunden godkender tilbuddet, oprettes ordren i databasen, hvorefter Webserveren laver en kvittering der gør stykliste, endelige pris og skitse synlig for kunden.

Scrum user stories & værktøjet Taiga.io

Vi har valgt at bruge "Taiga" som projektstyringsværktøj, der kan give et visuelt overblik og skabe en bedre struktur for selv opgavens udformning. Vi anså det som en fordel at have fået erfaring og arbejdet med værktøjet, da det er noget, som vi kommer til at skulle anvende på de fremtidige semestre. Dette projekt bærer præg af, at det er første gang at programmet "Taiga" anvendes. Det betyder, at der grundet den manglende erfaring kan være utilstrækkelige elementer, da programmets funktion ikke kendes til fulde. Vi gjorde brug af den overordnet struktur af User Stories fordelt på sprints. Dette hjalp med at skabe fokus på de underliggende tasks en problemstilling ad gangen. Da vi har siddet sammen under hele projektets forløb, har den primære projektstyring dog foregået mundtligt. Derfor har nogle user stories også langt flere underliggende tasks end andre. Et enkelt medlem af gruppen gør desuden brug af papir og blyant til at holde styr på manglende små opgaver. Derfor er tasks, der er opstået efter sprint planning meeting, desværre ikke alle repræsenteret i taiga.
Vi var usikre på hvordan story less task skulle bruges. Vi brugte dem primært i sprint 3, hvor en udvikler gennemgik hele vores site, og testede alle funktioner. Udvikler 2 og 3 skrev løbende alle små fejl og rettelser som enkle task. Derved opsnappede vi mange mindre design fejl på kort tid, og kunne derefter fordele de enkelte opgaver.

Det kan være enormt svært at finde frem til alle de underliggende tasks per user story under sprint planning meeting. Derfor vil der opstå nye "undertask" når en udvikler tager hånd om en task. Da vi arbejder tæt sammen i gruppen, ville processen af nye opstået task foregå som følgende:

Udvikler nr. 1 identificere manglede opgaver inden én task kan gennemføres. Udvikler nr. 2 griber tjansen og der kommunikeres løbende omkring metode, navne mm. Hvis en anden udvikler ikke har tid, skrives opgaverne ned i nogle tilfælde på et stykke papir. Dette skal optimalt set dokumenteres i taiga task board, da det er vigtigt for product owner at kunne følge udviklingen tæt.

Generelt synes vi, at det var lidt vanskeligt at bruge Taiga og taskboard under et lille projekt, især når gruppen arbejder tæt sammen.

Den største fordel for gruppen har været til daily scrum meeting om morgenen, samt debrief ved dagens afslutning.

User Stories uddybet:

User Story 1:
"Som køber Skal jeg kunne trykke på en knap med "egne mål" Så køberen kan skræddersy sin egen carport.

Acceptance criteria:
"Givet at vores website har en form, når en kunde udfylder formen, så sendes data til

databasen."

Når en bruger succesfuldt har registreret en ny profil, eller logget ind med en eksisterende profil bliver bruger redirected til sælger/bestillingssiden. Der kan også trykkes på "bestil carport". Her vil bruger blive præsenteret for en form, hvor ønsker til carport kan udfyldes. Når felter er udfyldt korrekt, trykkes på "bestil tilbud". Derefter er det muligt at se bestillingen i databasen.

Følgende task har vi placeret under user story 1:
#24 Design orderpage.JSP
#25 Implementer at der hentes beklædning af redskabsskur fra DB til dropdown menu.
#26 Implementer registrer User.
#27 implementer login
#28 Send form data til frontcontroller og datamapper gennem command.
#30 Tilføj form data til database.
#31 Lav en kvitterings page
#32 Lave "se gamle bestillinger" page.

I retrospektive møder kan vi konstatere at flere af disse tasks, ikke hører under denne User story. Blandt andet task der omhandler login, registrering af profil mm. Disse tasks vil være mere korrekt at finde under en "Storyless task" eller en user story for sig.

Vi har vurderet størrelsen af user story 1 til small da flere af de underliggende tasks allerede delvist er implementeret i vores udleveret skabelon. Dette har vi efterfølgende vurderet til at være korrekt.

User Story 4:

"Som sælger, skal jeg kunne godkende køberens ordre før han køber det, så sælger kan få kundekontakt og give bedst mulige rådgivning. Når sælger godkender modtager, køber en kvittering."

Acceptance criteria:

"Givet en køber har ekstra behov for vejledning, når køber har lagt en bestilling, så skal sælger kunne se bestillingen og kunne kontakte køber."

Når der logges ind som sælger, klikkes på "se aktive ordre". Her vil bruger blive præsenteret for alle ordre, der ikke er færdiggjort. Ved at klikke på "se denne ordre" vil man kunne se ordrens detaljer herunder kontaktinformation på kunden.
Her kan sælger vælge at tage kontakt til kunden og give videre rådgivning. Derefter kan sælger vælge at give rabat og derefter godkende bestillingen. Når en ordre er godkendt, kan kunden, der har lagt ordren, logge ind og se sin fulde bestilling og endelige pris.

Følgende task har vi placeret under user story 4:
#36: "Oprette en sælger landingpage, der viser oversigt over aktive ordre"
#38: "Give sælger adgang til kundens orderpage"
#39: "Skal lave en knap der ændre status på en ordre"
#46: "Opdatere "see orders page"

Vi har vurderet user story 4 til størrelse medium. Vi har efterfølgende konstateret at vores estimat passede til opgavens omfang.

User story 5

“Som sælger skal jeg kunne se brugerens krav, så programmet kan komme med en stykliste.”

Acceptance criteria:

“Givet at sælger kan se købers krav, når køber har lagt en bestilling, så skal køber få en stykliste udregnet af programmet.”

Sælger skal logge ind for at finde aktive ordre og klikker på “se denne ordre”. Her vil ordrens stykliste fremgå for sælger.

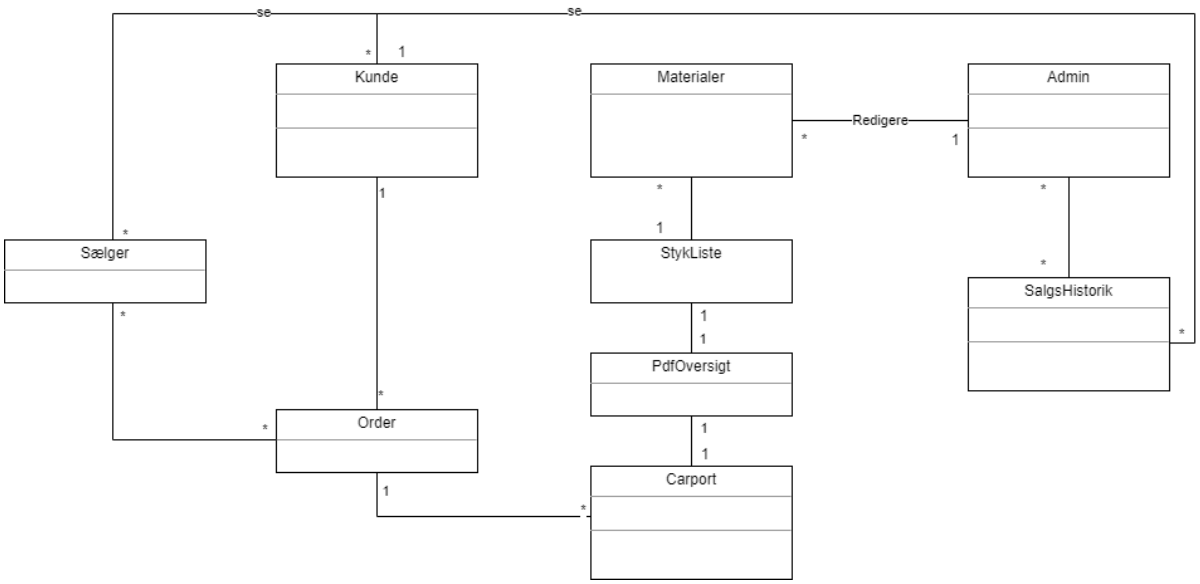
Følgende task har vi placeret under user story 5:
#50 Rette tag calculator
#48 beregn total kostpris for materialer og total salgspris.
#43 lave calculator til at lave stykliste til skur og carport ud fra kunden specifikationer”

Vi har vurderet user story 5 til størrelse Large. Det passede fint i forhold til arbejdsbyrden.

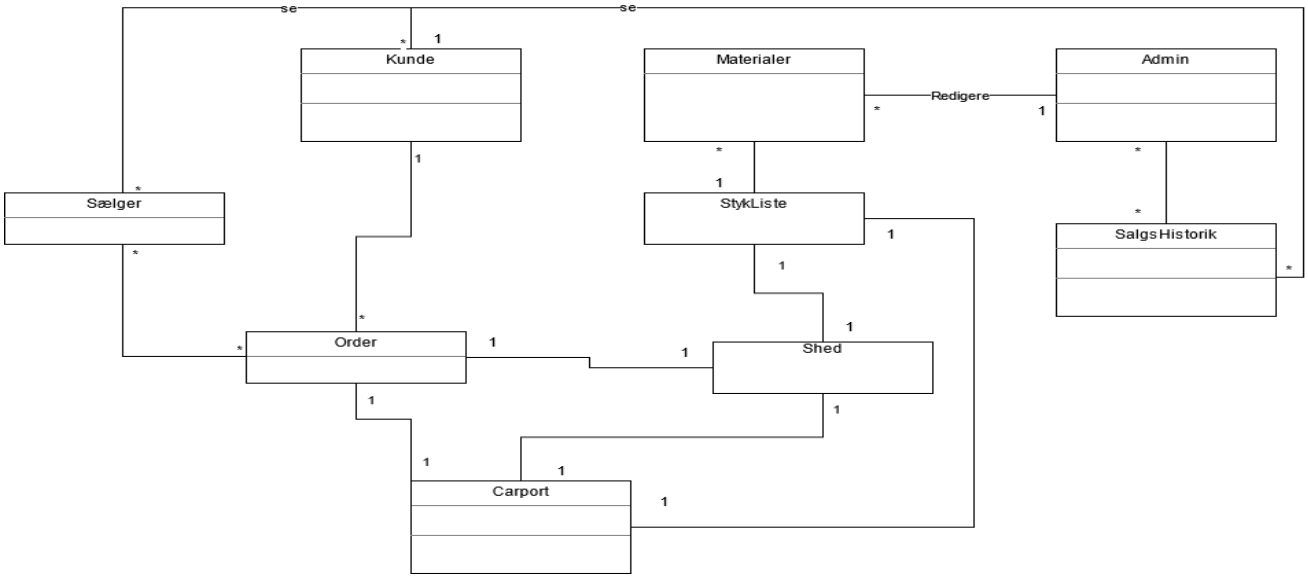
Efterfølgende har vi talt om at især task #43 burde være delt bedre op i f.eks. “lave beregning af spær”, “Lave beregning af remme”, men da vi udviklede beregneren sammen i gruppen, brugte vi kommentar i intellij samt papir og blyant for at følge implementering af denne user story.

Domæne model og ER diagram

Domænemodel



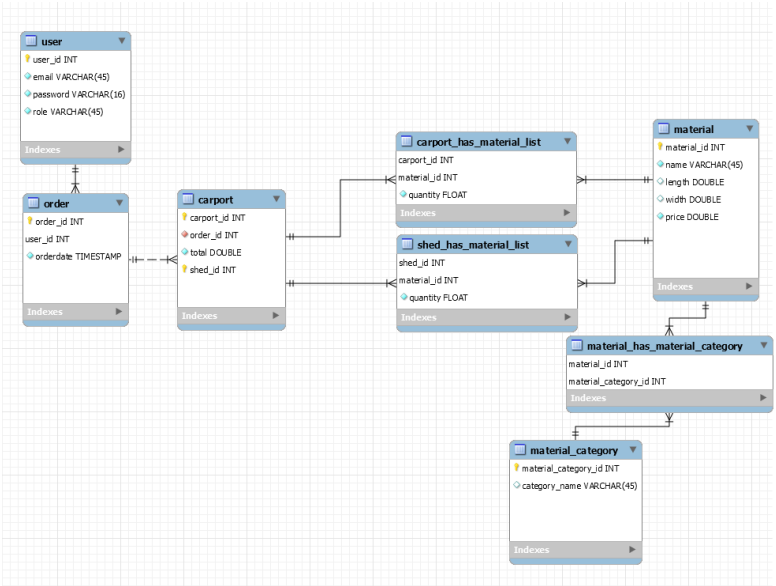
Den table der hedder “pdfOversigt” er ikke blevet implementeret i denne opgave, da den ikke var et krav for selve projektets udførelse. Vi har lavet en model, der kunne muliggøre det at en kunde kunne have op til flere sælgere. Dette ville være en god service for kunden, hvis der f.eks. skulle opstå sygdom blandt en af sælgerne. Hermed ville det være en god gevinst for både firma og kunden, at en anden sælger nemt kan stå til rådighed, så kunden ikke behøver at vente på at bestillingen bliver gennemført.

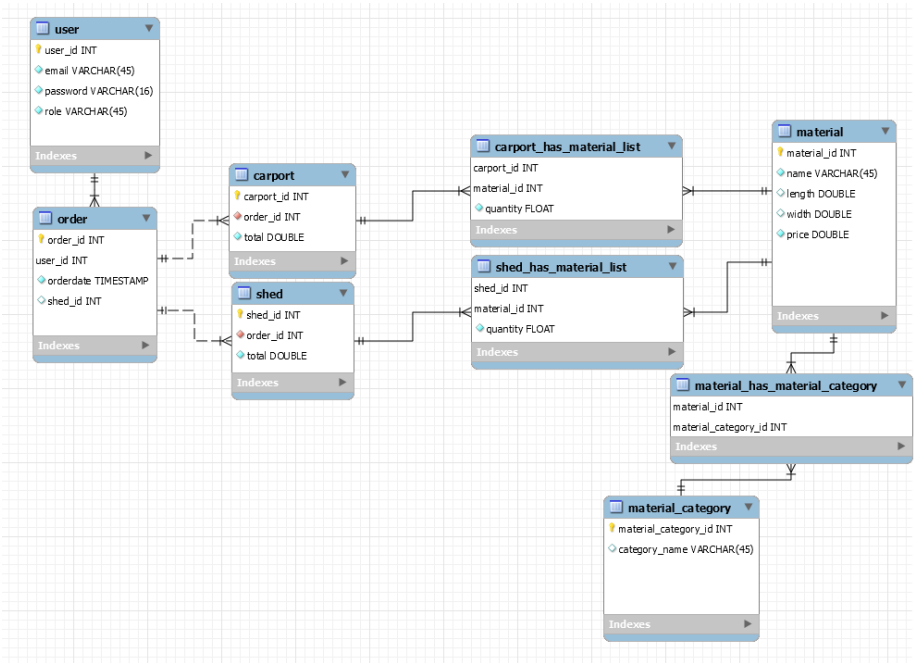


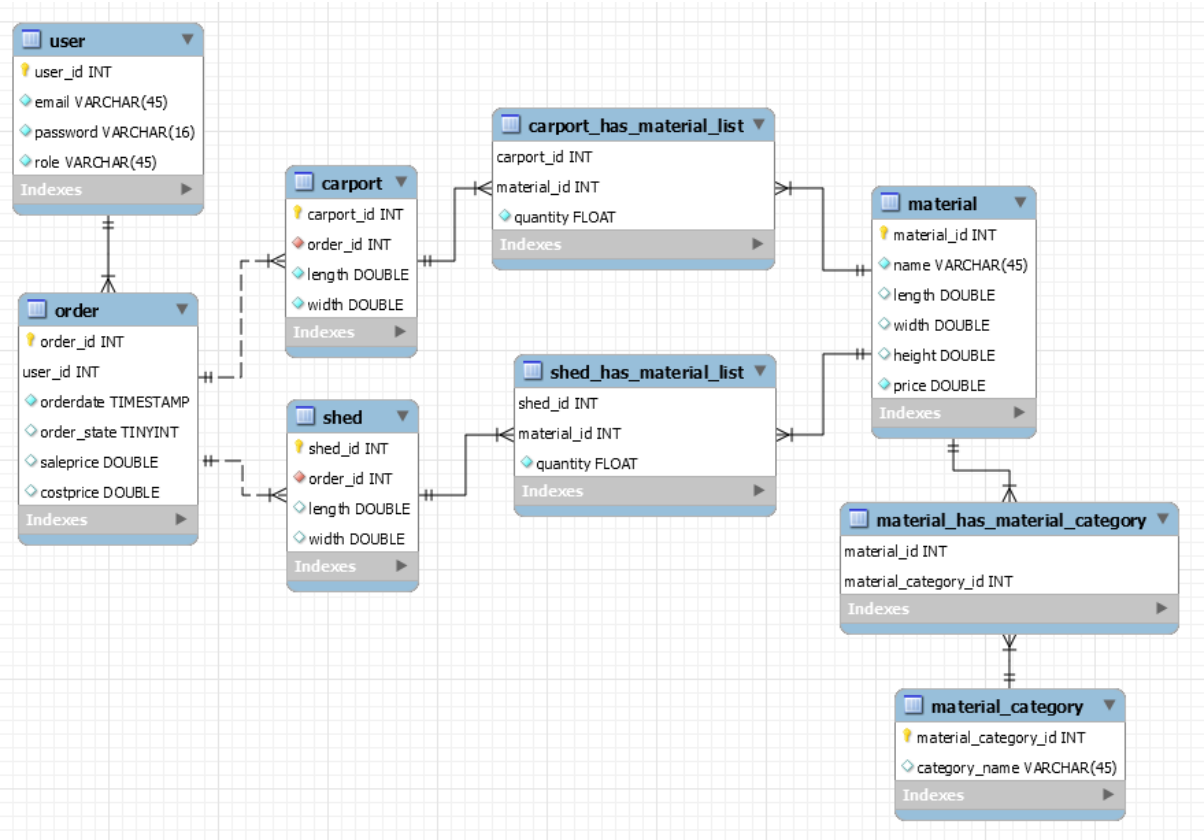
Vi har opdateret vores Domæne model og har tilføjet “shed”. Vores forhold mellem ordre til carport er også ændret fra ‘1 til *’ til ‘1 til 1’. Dette skyldes at vi da projektet startede, havde gjort os den tanke at man godt kunne købe flere carporte pr. ordre. Det kunne være at man f.eks. ville købe en både til sig selv og til naboen. Det viste

sig dog at være problematisk, hvis den ene af den købende part ønskede et redskabsskur, men den anden ikke ønskede det i forlængelse af carportene. Det blev derfor uklart om hvilken af de to carporte redskabsskuret hørte til.

EER-Diagram

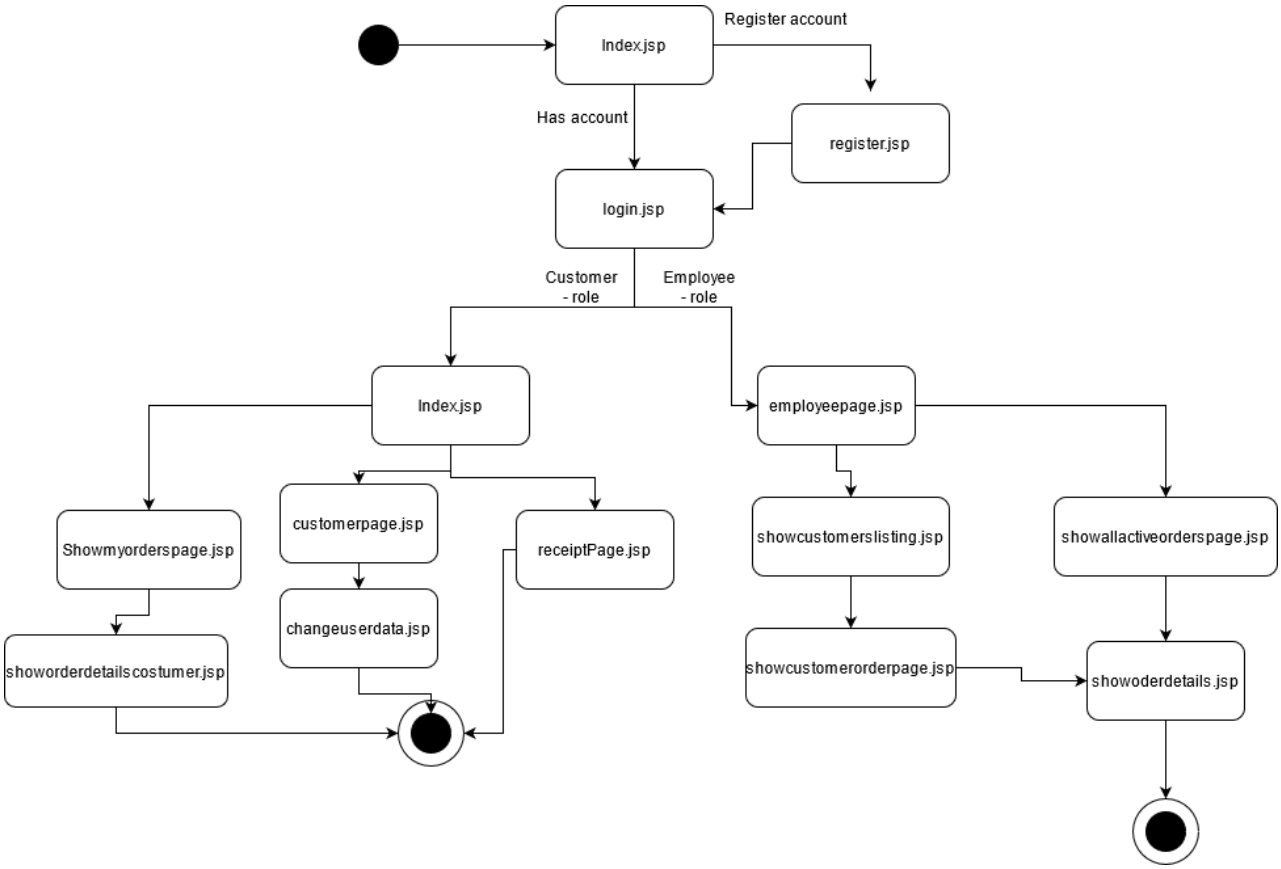






I starten havde vi den ide at shed_has_matrial_list skulle linkes på vores carport table. Men det er styklisten af materialerne til redskabsskur, og før vi kan lave styklisten, er vi nødt til at oprette et redskabsskur. Selve styklisten til redskabsskuret består kun af beklædning og intet andet. Man burde have lavet styklisten opdelt så redskabsskuret havde sine egne materialer, stolper, remme og spær, i stedet for at smide alt ind i styklisten til carporten. Især fordi vi har gjort det sådan at man altid kan få et redskabsskur, i den størrelse man gerne vil. Det kan give to stolper lige ved siden af hinanden. Der vil selvfølgelig komme en sælger ind over, og vejlede, men hvis køber vil have det på den måde, så kan han/hun godt få det. Den tanke ramte os først lidt sent, og der var vi færdige med at lave vores stykliste calculator. Vi valgte at fokusere på noget andet, i stedet for at skulle gå tilbage og lave vores kode om for at tilpasse den.
Vores user table burde have haft flere parametre som f.eks. adresse, by, postnummer, telefonnummer, men så skal man huske de 3. normal former, og lave en ekstra table som består af by og postnummer. Det er meget hurtigt og nemt lige at tilføje men vi valgte det fra, så man ikke skulle til at skrive 6 forskellige ting, hver gang man skulle lave en ny bruger.

Navigationsdiagram & Mockups

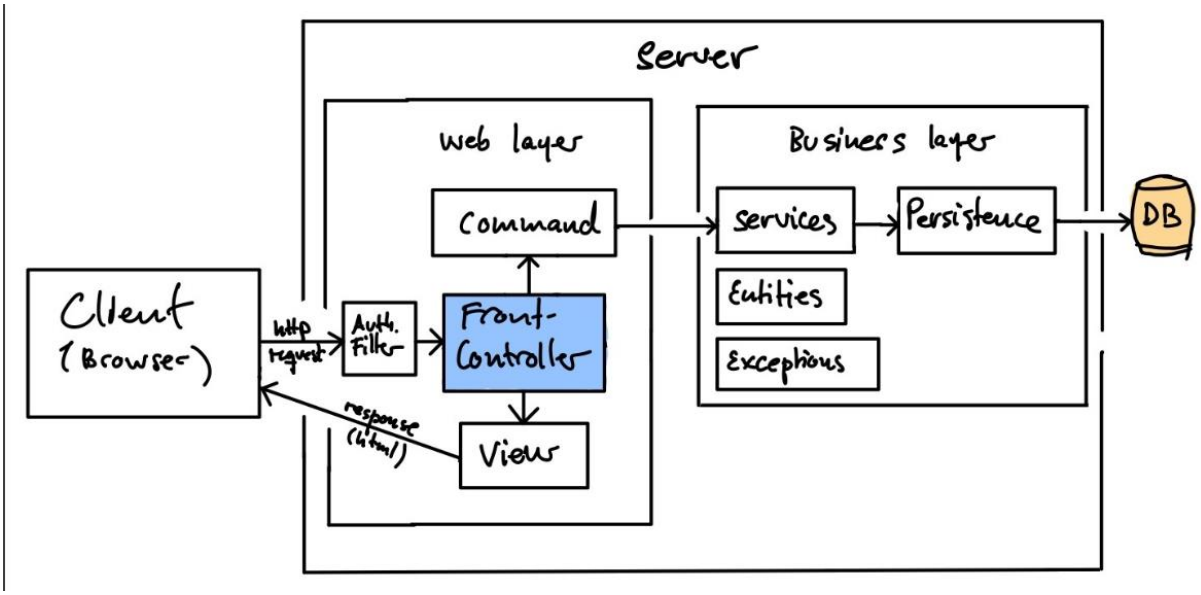


I toppen af hjemmesiden er der en navigation bar, der skifter, alt efter hvilken rolle du er logget ind med. Som bruger vil der være et felt som ses som “Bestil carport” der fører brugeren videre til index.jsp. Under “Min side” vil man ende ved customerpage.jsp. Feltet “Mine ordre” som viderefører kunden til showmyorderspage.jsp. Under Employees navigations bar er første punkt “Admin side” der fører til employeepage.jsp. Vælger man her “Se brugere” vil man blive viderestillet til showcustomerslisting.jsp. Vælges “Se Aktive Ordre” vil man ende ved showallactiveorderspage.jsp

Da opgaven har til formål, at den skal implementeres til fogs egen hjemmeside, har vi valgt at lave vores mockup efter fogs designer valg.

Valg af arkitektur

Vi har fået udleveret en skabelon til brug af projektet. Skabelonen er bygget op i et weblag og et businesslag på serversiden. Weblaget er opbygget omkring en Frontcontroller.
Som vist på billedet herunder, laves en httprequest fra klienten. Den bliver håndteret i frontcontrolleren, der executer en command specificeret gennem et command pattern.
Commanden taler til vores business lag, der eventuelt kan hente data fra databasen. Data bliver sendt retur gennem business laget, til command, videre til frontcontrolleren og efterfølgende sendt i et http response view til clienten.



Fordelen ved en frontcontroller er, at vores servlet kald bliver samlet ned i et url kald, i stedet for at man skal lave den kode til hver enkelt command, jf. eksempel til koden nedenunder. Det betyder samtidig at vores login til databasen kun skal stå et sted, i stedet for i hver command. At redirect til den næste side bliver også håndteret i frontcontrolleren.

Vores command pattern er centreret omkring den abstrakte klasse *Command*, som bliver extended ned i *CommandProtectedPage* og *CommandUnprotectedPage*. Disse klasser styrer henholdsvist om den JSP side, der redirecter til behøver et login eller ej.
Vores egne command fx *MakeOrderCommand* extender, f.eks. CommandProtectedPage, da brugeren skal være logget ind for at lægge en bestilling. Hver command overskriver metoden *execute*. Det er her vi lægger funktionaliteten ind, der taler med business laget, og lægger data i scope til JSP.
Vores commands er indsat som et hashmap i Command. Hvert element i dette hashmap består af en key value, der består af navnet som kaldes fra vores action på

en given jsp side. Hver key er derefter koblet sammen med en command inklusive parametrene *pageToShow* og *role*.
Sådan kan det lykkedes at kalde forskellige commands gennem vores frontcontroller som beskrevet herunder.

Når man f.eks. fra index siden submitter en form, så bliver der lavet et url kald til vores frontcontroller. Den første del af vores ekspression tekst, angiver projektets egne sti i windows. Det gør, at vores action kan være ens på alle i gruppens lokale maskiner.

```
<form id="form" method="post" action="${pageContext.request.contextPath}/fc/orderPage">
```

I frontcontroller klassen kan vi øverst se URL kaldet til denne servlet

```
@WebServlet(name = "FrontController", urlPatterns = {"/fc/*"})
```

I formen på jsp er der defineret, om *requestet* er et doPost eller *doGet* kald, hvor det i vores eksempel er et post. Begge metoder kalder *processRequest* i frontcontrolleren. Denne metode instansere en ny command. Den given Request streng deles ved /fc/, og vi får derved et navn på vores command. Denne command findes igennem command designet, som vi har beskrevet ovenover. Derefter bliver den pågældende command klasse egne *execute* metode kaldet. Denne execute metode returnerer en *pageToShow* string. Til sidst forwarde metoden *processRequest* brugeren til den givne .jsp som defineret via *pageToShow* igennem *getRequestDispathcer*().*forward*()

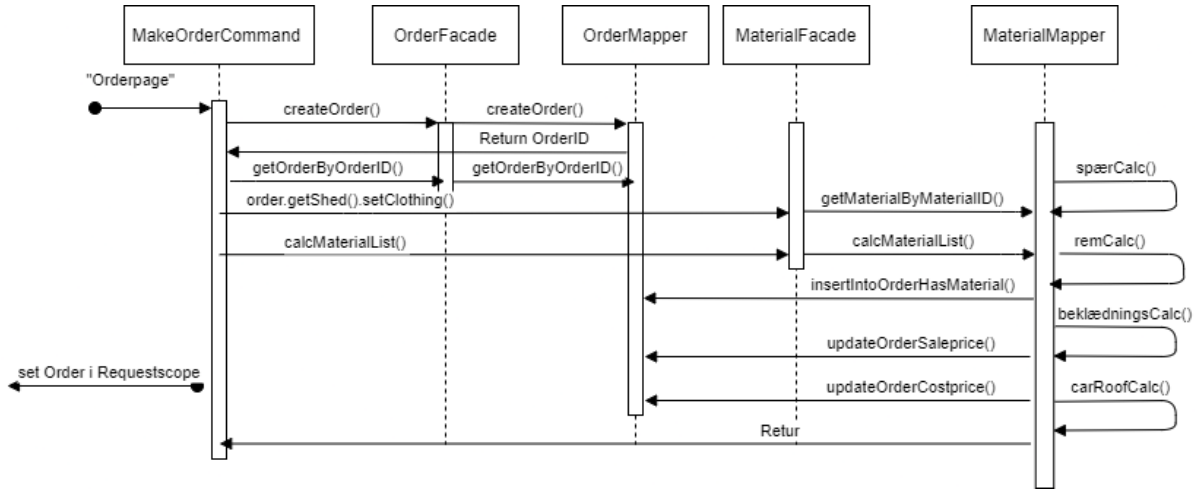
Eksempel på en command *ShowOrdersCommand*:

```
@Override
public String execute(HttpServletRequest request, HttpServletResponse response) throws UserException {

    try {
        User user = (User) request.getSession().getAttribute( s: "user");

        request.setAttribute( s: "ordersList", orderFacade.getOrderByUserId(user.getUser_id()));

    } catch (NumberFormatException e) {
        request.setAttribute( s: "error",  o: "Du mangler at udfylde nogle felter!");
    }
    return  pageToShow;
```

Her skal metoden vise alle ordre for den kunde, der er logget ind.
Først instantieres et User objekt, der hentes fra session scopet. Dette bliver sat ved login.
Derefter kaldes *orderFacade.getOrderByUserId*() som returnerer en liste af ordre. Denne sættes i request Scopet, da dataen kun skal bruges på den kommende jsp siden.
Til sidst redirecter til den jsp side som er defineret i vores command hashMap.

Facade-lag: Det smarte ved et facade-lag er, at man kan samle nogle af sine metodekald i en metode.
I vores projekt har vi ikke brugt facade laget på den korrekte måde, men er derimod blevet til redundant kode. Da vi kalder en metode i facaden, hvis eneste funktion er at kalde en metode i mapper'en. I teorien kunne hele facadelaget fjernes ud af koden, hvor den stadig ville virke. Dette blev først opdaget imens vi skrev rapporten, eftersom vi på nuværende tidspunkt ikke har fået den fornødne undervisning omkring brugen af facade-lag endnu. Et godt sted hvor facade-laget kunne komme i karakter i opgaven, ville være når ordre objektet skal instantieres. I stedet for at kalde en masse forskellige metoder under execute metoden, så ville facaden give mulighed for at lave et samlet metodekald.

Sekvens diagrammer



Det ovenstående sekvensdiagram beskriver processen, hvor en køber lægger en ordre fra Index siden, kalder “Orderpage” commanden gennem frontcontrolleren og til slut bliver præsenteret for den færdige ordre med tilhørende stykliste.

Order Page command instantiere vores MakeOrderCommand. Her bliver form data først hentet fra request scopet, input valideret og derefter videresendt via metoden createOrder til OrderFacaden og OrderMapper. Her bliver form data indsat i Databasen, og returnere det automatiske genereret Order Id fra databasen. Derefter kaldes getOrderbyID, der returnerer den nye ordrer som et java objekt. Den valgte beklædning til redskabsskuret indsættes i Order objektet gennem order.getShed().setClothing().getMaterialByMaterialID(), som taler til vores MaterialMapper.
Den nødvendige data er nu samlet i Order objektet. Det sendes til MaterialMapper gennem calcMaterialList(). I vores MaterialMapper beregnes materialisten enkeltvis via stolpeCalc(), spærCalc(), remCalc(), beklædningsCalc() og carRoofCalc(). De udregnede værdier indsættes i en liste af materialer objekter og indsættes i Order objektet.
Den færdige ordrer indsættes i vores database i OrderMapper gennem metoden insertIntoOrderHasMaterial(). Til sidst bliver ordren sendt retur til MakeOrderCommand, der sætter ordren i requestscopet til at præsentere på kvittering siden.

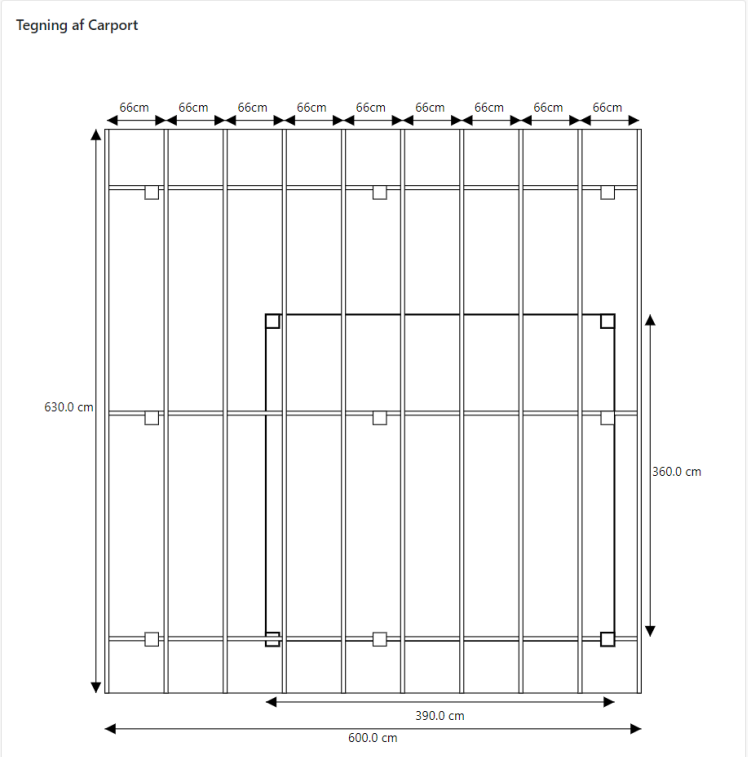
Denne process er delvist beskrevet i detaljer under afsnittet kodeeksempler.

Særlige forhold

Vi valgte at lægge vores database kald til beklædning af redskabsskuret i request scopet frem for session scope. Da det kun skal bruges på index.jsp, og altid vil være den nyeste data fra databasen. Ændringer der undervejs sker af materialer attributter vil altid være opdateret i det nyeste request scopet.

Antagelser vedrørende redskabsskur.

Ved bestilling af en carport, og der ønskes redskabsskur, vil kunden modtage en fejl, hvis der bestilles et redskabsskur, som er for stort, dvs. hvis det er under 15 cm fra carportens kant. Vi har antaget at kunden skal have mulighed for at vælge præcis den størrelse redskabsskur personen ønsker, og ikke fastlåst til halv størrelse af carport. Dog vil redskabsskuret altid blive tegnet i højre nedre hjørne. Derfor vil carporten altid have egne stolper til at montere remme, og redskabsskuret vil have sine egne stolper, med undtagelse af den nederste til højre.
Dette vil i nogle specielle tilfælde give nogle lidt spøjse tegninger, jf. figur under dette afsnit. Hvis f.eks. kunden ønsker en lille carport til cykler, med et stort redskabsskur, vil dette give en stolpe midt i skuret. Hertil er det tiltænkt at sælger tager dialog med kunden for enten ændre lidt på tegningen eller opbygningen.



Antagelser vedrørende stykliste beregner

For at kunne lave vores beregner har det været nødvendigt at have nogle antagelser til vores regneregler. Disse er fundet frem til gennem størrelser på faste carporte solgt hos Fog:
Rem: Der skal minimum være 2 remme, er carporten over 5 meter i bredden skal der være 3 remme.
Spær: Der skal være minimum 55 cm mellem hvert spær. Der skal altid være et spær forrest og bagest på rem.
Stolper: Vi antager der altid er 4 stolper i et redskabsskur, et til hvert hjørne, hvor den ene stolpe er delt mellem carport og redskabsrum. Der må maks. være 3 meters mellemrum mellem hver stolpe på en rem.

Vi har valgt at have alle materialer samlet i et samlet objekt, da materialerne har samme attributter. Et navn, længde, bredde, højde, salgspris, kostpris, antal og en kategori.
Gennem kategori attribut kan vi sortere i materialerne. Følgende kategorier er defineret af os:

Kategori oversigt:
1: Redskabs beklædning.
2: Carport beklædning
3: spær
4: rem
5: stolper

Udvalgte kodeeksempler

I dette kodeeksempel viser vi hvordan man tager en bestilling og gemmer den i databasen.

```
double carportLength = Double.parseDouble(request.getParameter( name: "carportLength"));
double carportWidth = Double.parseDouble(request.getParameter( name: "carportWidth"));
double shedLength = Double.parseDouble(request.getParameter( name: "shedLength"));
double shedWidth = Double.parseDouble(request.getParameter( name: "shedWidth"));
int carportRoof_materialID = Integer.parseInt(request.getParameter( name: "carportRoof"));
int shedClothing_materialID = Integer.parseInt(request.getParameter( name: "shedClothing"));
```

I MakeOrderCommand.java henter vi data fra requestScopet og konvertere det til de datatyper, som vi skal bruge for at kunne lave fejlhåndteringen forneden.

```
double totalLength = (carportLength-45) - shedLength;
double totalwidth = (carportWidth*0.9) - shedWidth;

if (totalLength < 30 || totalwidth < 30) {
    request.setAttribute( name: "error",  o: "Redskabsrums er større end carporten, vælg en mindre størrelse");
    Command command = new NavigateToIndexCommand( pageToShow: "index",  role: "customer");
    return command.execute(request, response);
}

if (shedLength > 0 && shedWidth == 0 && shedClothing_materialID == 0 || shedLength == 0 && shedWidth > 0 && shedClothing_materialID == 0) {
    request.setAttribute( name: "error",  o: "Du mangler at udfylde et felt");
    Command command = new NavigateToIndexCommand( pageToShow: "index",  role: "customer");
    return command.execute(request, response);
}
```

Denne del af koden sørger for den fejlhåndtering der er ift. redskabsskuret og manglende data.

```
User user = (User) request.getSession().getAttribute( name: "user");
int user_id = user.getUser_id();
int orderid = orderFacade.createOrder(user_id, carportLength, carportWidth, shedLength, shedWidth);
```

Vi henter vores user objekt fra requestScopet, som vi gemmer i et java objekt, der indeholder en specifik kundes id som bruges til at oprette en ordre.

```
public int createOrder(int user_id, double carportLength, double carportWidth, double shedLength, double shedWidth) throws UserException {
    return orderMapper.createOrder(user_id, carportLength, carportWidth, shedLength, shedWidth);
}
```

Vi bruger en instans af *orderFacade*, som har et metodekald på *orderMapper*, og her er det vigtigt at de korrekte data er med.

```
public int createOrder(int user_id, double carportLength, double carportWidth, double shedLength, double shedWidth) throws UserException {
    try (Connection connection = database.connect()) {
        String sql = "INSERT INTO `order` (user_id) VALUES (?)";
        try (PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            ps.setInt( parameterIndex: 1, user_id);
            ps.executeUpdate();

            ResultSet order_id = ps.getGeneratedKeys();
            order_id.next();
            int id = order_id.getInt( columnIndex: 1);
            insertIntoCarport(carportLength, carportWidth, id);

            //check if the value of shed is 0, if so dont add a shed to the order
            if (shedLength != 0 && shedWidth != 0) {
                insertIntoShed(shedLength, shedWidth, id);
            }

            return id;
```

```java
public void insertIntoCarport(double carportLength, double carportWidth, int order_id) throws UserException {
    try (Connection connection = database.connect()) {
        String sql = "INSERT INTO `carport` (order_id, length, width) VALUES (?, ?, ?)";

        try (PreparedStatement ps = connection.prepareStatement(sql)) {
            ps.setInt( parameterIndex: 1, order_id);
            ps.setDouble( parameterIndex: 2, carportLength);
            ps.setDouble( parameterIndex: 3, carportWidth);
            ps.executeUpdate();
```

Med vores tabelstruktur skal der indsættes i 3 tabeller, hvis man har et redskabsskur og i 2 tabeller, hvis ikke det indgår i ordren.

Det er antaget at en ny ordre vil blive gemt i databasen, og vil fremgå således:

```
1 •  SELECT * FROM carport.order;
```

order_id	user_id	orderdate	order_state	saleprice	costprice
1	3	2021-05-21 13:07:14	0	8285.22	5799.65

```
1 •  SELECT * FROM carport.carport;
```

carport_id	order_id	length	width
1	1	780	750

```
1 •  SELECT * FROM carport.shed;
```

shed_id	order_id	length	width
1	1	180	240

Det skal pointeres at denne ordre er bestilt med redskabsskur.

```java
request.setAttribute( name: "order", order);
```

Nu har vi en carport med redskabsrum baseret på kundens ønsker og det kan gemmes i requestscopet som et helt ordreobjekt.

Status på implementering

Der er i dette projekt ikke lavet en implementering af User story 8 og 9 grundet tidsmangel. User story 8 indebærer en administrator kan logge ind og ændre på priser på enkle materialer. User story 9 skal give administrator mulighed for at slette eller oprette nye materialer i systemet.
Valideringen af mål for nye materialer ville skulle være præcise, da fejlindtastet materiale mål ville ødelægge beregningerne samt tegninger.

Vi har ikke fået refaktoreret vores metoder blandt andet i klassen “orderMapper” hvilket giver nogle klasser en rodet struktur.

Vi har valgt kun at medtage de basale materialer i vores stykliste regner. Yderligere materialer vil kun bestå af basale matematiske beregninger, hvor vi har valgt at prioritere at bruge tiden blandt andet på at genoverveje diverse diagrammer, og lave review af vores arbejdsproces for at forbedre vores samarbejde til fremtiden.

Salgsprisen på vores materialer er taget direkte på Fogs hjemmeside. Vores kostpris på materialer er udregnet ved at tage 70% af salgsprisen. Dette bør være implementeret som et datafelt i databasen.

I vores svg tegning, har vi fundet en fejl, som vi ikke har nået at rette endnu, den kommer kun når man også har et redskabsskur. Vi har en mistanke omkring, at der i vores svg-mapper under stolper laver et for-loop, hvor man går “i < stolper.getquantity / rem.getquantity” igennem. Ved at tilbyde et skur i præcise længder og bredder efter eget ønske, ændrede vi vores calculator således at der ved tilvalg af et redskabsskur bliver vores count af stolper plusset med tre. Hvilket vil sige at så snart vi har et redskabsskur på vores carport, så vil der blive tegnet en ekstra sæt stolper uden for tegningen, som man ikke kan se, men kun i kildekoden. Dette vil dog nemt blive opdaget hvis man havde lavet fungerende Unit test.

Test

Vi har valgt at teste gennem udskrivninger i terminalen, skriv til databasen og “bug runs” i intellij, hvor vi kan følge koden trinvist. Det er naturligvis ikke den mest korrekte måde at gøre det på, men da vi havde svært ved at implementere meningsfulde unit test og integrationstest i den udleveret skabelon valgte vi ovenstående. Da vi modtog undervisning i hvordan Unit test og integrationstest i skabelonen var vi næsten færdig med at udviklingen og projektforløbet.

Det ville havde været oplagt at implementere unit testing i vores stykliste beregner. Der kunne f.eks. testes op mod faste carporte solgt af Fog. En succesfuld test ville give samme stykliste som den vedlagte af Fog, hvis der bestilles i samme dimensioner.

En anden fordel ved at benytte Unit test omkring vores svg-tegning, for at se hvor mange stolper, remme, og spær den tegner og om det passer i forhold til vores stykliste calculator. Der ville man skulle tage styklisten med for at holde den op imod tegningen. Så ville man have koden der tegner det hele med, hvor man laver en count af hver enkelt ting, og så tjekker man stolpeCounten op i mod stolpeQuantity, i styklisten og remCount med remQuantity osv.

Dernæst kunne det være godt at have lavet en integrationstest for vores database kald. Specielt når en færdig udregnet stykliste lægges i databasen.
Method createOrder() findes i OrderMapper. Den kalder metoderne InsertIntoCarport() og laver derefter en test for at se om ordren indeholder et redskabsskur, hvis ja kaldes inserIntoShed(). Disse tre metoder indeholder alle databasekald hvor det ville havde været en hjælp med enkeltvise integrationstest.

Methoden calcMaterialList() i MaterialeMapperen kalder metoder til udregninger af materiale listen. Derefter laves tre databasekald, InsertIntoOrderHasMaterial der indsætter styklisten. updateOrderSale() opdatere samlet salgspris for carport, updateOrderCost() opdatere samlet kostpris for carporten. Alle disse metoder havde været gode at have integrationstest på.

Proces

Arbejdsprocessen faktuelt

Vi har brugt rigtig meget tid sammen som gruppe på at diskutere hvordan vi skulle håndtere dette projekt. Vi har haft scrum møder hver dag, der har ligget først på dagen, hvor dagens agenda gennemgås i plenum. Vi har prøvet at have en scrum master, og har haft retrospective møder sidst på ugen. I vores tilfælde har det at have en scrum master, oplevet som en anelse redundant, eftersom vi hver dag har siddet og arbejdet på projektet sammen. Det har af den årsag, været muligt for de enkelte gruppemedlemmer at tale åbent omkring idéer, tanker og eventuelle opdateringer i løbet af dagen. Vi har fokuseret på at lave vores diagrammer og løbende opdatere dem. Alt i alt har vi arbejdet med 3 sprints hvor PO har været inden over, som har fungeret godt. Til mange af møderne fik vi også bekræftet at det kan betale sig med alt vores forarbejde.

Arbejdsprocessen reflekteret

En stor del af tiden er valgt at blive prioriteret på sys delen i dette projekt, eftersom dette er et område vi generelt oplever en manglende erfaring. Som gruppe har vi oplevet en tendens til at ende i den samme problematik med, at man hver især har sin egen måde at tænke og kode på. Dette kan resultere i nogle faglige diskussioner, hvor man må argumentere for sit standpunkt. Når der opstår episoder som disse, er det tydeligt hvilken fordel scrum møderne har, herunder forebyggelse og forventningsafstemning på et struktureret grundlag.
Ved at lave sit forarbejde og grundig forberedelse, har det skabt ekstra tid på at fokusere og gå i dybden med samarbejdet i gruppen. Det er et stykke arbejde som alle hver især har budt ind med, og som har gavnet projektet. Fremadrettet vil vi ved næste projekt tage denne erfaring med os videre.
Til trods for godt forarbejde, har der alligevel været enkelte områder, som vi ikke havde forberedt os på, herunder opdeling af styklisten. Her burde der internt i gruppen have været grundigere med kortlægning af opbygningen inden vi startede op. Dette medførte misforståelser for løsning af problemer undervejs i processen. Undervejs blev der forsøgt at rette op dette, da den mangelfulde opdeling af styklisten blev opdaget, ved at lave et forsimplet klassediagram, så gruppen kunne komme tilbage på et fælles spor igen.

Selve Scrum master rollen, kan vi se har potentiale, men ingen af os har nogen erfaringer med den endnu. Det kunne havde været en god hjælp, og en spændende læring at have haft en erfaren scrum master med på forløbet, fx en tutor. Det ville have givet os et godt billede af hvordan rollen præcis skulle faciliteres.

Overordnet set, har vi været gode til at estimere størrelsen af de forskellige user stories. Vores user story 5 omkring at lave en calculator, var vi nødt at nedbryde og dele op i mindre bidder, da den var for stor til en user story. Det betød også at vi ikke ramte vores estimering af denne user story.
Vores generelle oplevelse af møderne med vejledning og PO, har været at det primært har været os som gruppe der skulle tage initiativet og fortælle. Det har derfor føltes mere som en fremlæggelse end en egentlig vejledning, da vi har manglet feedback og råd. Til trods for den gode ros og anerkendelse ift. til projektets udfoldelse, ville det for en ambitiøs gruppe have været givende med at blive udfordret, ved at have haft nogle enkelte pejlemærker, vi kunne have været endnu bedre til. F.eks. brugte vi meget tid på at lave vores EER-diagram og kunne godt havde brugt vejledning og sparring i den forbindelse.
Samlet set har det været en god og lærerig proces, og har styrket vores samarbejde som projektgruppe.