Fog Carport

Hold A Gruppe 3

Projektstart:	03 / 05 / 2021
Projektslut:	28 / 05 / 2021

Informationer

Navn
Frederik Bilgrav Andersen

Mail
cph-fa116@cphbusiness.dk

Github
github.com/FrederikBA

Hold
Hold A

Navn
Janus Stivang Rasmussen

Mail
cph-jr270@cphbusiness.dk

Github
github.com/Janussr

Hold
Hold A

Links

Github Repository

https://github.com/FrederikBA/FogProject

Demovideo af hjemmesidens funktionaliteter

https://www.youtube.com/watch?v=7viStqiaaQM

Vores hjemmeside deployed på droplet:

http://139.59.139.32:8080/FogProjekt-1.0-SNAPSHOT/

Administrator login til hjemmeside:

Brugernavn: f@mail.dk

Password: 123

Taiga backlog

https://tree.taiga.io/project/janussr-fog/backlog

Sprint 1

https://tree.taiga.io/project/janussr-fog/taskboard/sprint-1-18884

Sprint 2

https://tree.taiga.io/project/janussr-fog/taskboard/sprint-2-10696

Sprint 3

https://tree.taiga.io/project/janussr-fog/taskboard/spring-3-48

Indledning

Vi har i dette semesterprojekt fået som opgave at udvikle et it-system, mere specifikt en hjemmeside, der som formål skal fungere som webshop for kundens produkt samt løse diverse administrative opgaver for kunden.
Vores projekt er derfor omstændigt indenfor både frontend og backend med baggrund i den undervisning vi har haft på andet semester.
Projektet er et Java webprojekt baseret på Java servlets. Det er udviklet ved hjælp af en udleveret startkode der gør brug af en række forskellige design patterns.

Baggrund

Vi har fået som opgave at udarbejde et mere *up-to-date* it-system til virksomheden Johannes Fog der specialiserer sig indenfor byggematerialer, trælast, design og lignende. Fog har ønsket at få revideret og opdateret deres it-løsning der benyttes til salg af carporte. Produktet der ønskes, er et system der kan tilgås både som kunde men også af personalet selv til at løse diverse administrative opgaver. Systemet skal være et website hvor en kunde hos Fog, kan skræddersy og bestille sin egen carport efter behov. Dette skal gøres ved hjælp af en formular hvor kunden afgiver diverse informationer såsom carportens ønskede længde og bredde, valg af skur og designet af carportens tag.

Når en ordrer er blevet lagt af en kunde, skal it-systemet kunne udarbejde en stykliste med de materialer der skal bruges til at sammensætte den færdige carport, så en medarbejder kan gå på lageret og klargøre ordren. It-systemet skal også kunne udarbejde to tegninger af carporten med mål. Den første tegning skal være en tegning af carporten set oppefra uden tag. Her skal det være synligt hvordan materialer såsom rem, spær og stolper er placeret. Den anden tegning er af carporten set fra siden, så man kan danne sig et overblik over hvordan carporten vil komme til at se ud.

Personalet hos Fog skal også kunne tilgå hjemmesiden ved hjælp af et personale login. Her skal det være muligt for, for eksempel en sælger, at få et overblik over de ordrer der er blevet lagt, så sælgeren kan behandle den ordre som kunden har bestilt. Der skal også være en materiale side der viser de materialer og produkter som Johannes Fog har på deres lager. Her

vil det være muligt for medarbejderen at opdatere, tilføje og slette produkter så lageret og dets tilhørende materialer kan justeres.

Teknologivalg

Vores projekt er et java baseret webprojekt og vi har taget brug af følgende teknologier og programmer til at udarbejde det:

Editor/IDE

IntelliJ IDEA (version 2021.1)

Udviklingsværktøjer / Sprog

Java (*Java Development Kit 11.0.4*)
Maven
MySQL Workbench (version 8.0)
HTML5
CSS
Bootstrap (version 5.0)

Droplet til deployment

Digital Ocean

Web server:

Apache Tomcat (version 9.0.46)

Krav og Scrum user stories

Ud fra samtalerne med product owneren, blev der stillet nogle krav, for hvilke funktioner hjemmesiden skulle have, disse krav blev lavet om til userstories.

US-1: Som kunde kan jeg aflægge en forespørgsel til en skræddersyet carport efter egne mål, materialevalg og tag, så jeg kan få designet en carport efter mine behov.

US-2: Som kunde kan jeg modtage et tilbud på en tilsendt forespørgsel ved en skræddersyet carport, så jeg kan bestille min ønskede carport.

US-3: Som kunde skal jeg kunne indtaste mine informationer ved bestilling vha. et login-system, sådan at sælgeren kan komme i kontakt med mig.

US-4: Som kunde skal jeg kunne se en tegning på den sammensatte carport, så jeg kan få et visuelt billede over min carport.

US-5: Som lagermedarbejder kan jeg modtage en e-mail når en kunde har lagt en bestilling, så jeg kan behandle kundens ordre ud fra en tilsendt stykliste.

US-6: Som administrator skal jeg kunne se en kundeoversigt, så jeg kan få et overblik over vores kunder.

US-7: Som administrator skal jeg kunne se kundens ordrehistorik på en kundeoversigt sådan at de rigtige varer bliver sendt ud til kunden.

US-8: Som administrator kan jeg tilføje og fjerne produkter samt produktbeskrivelser, sådan at kunden altid kan se de nyeste produkter vi tilbyder.

US-9: Som administrator skal jeg kunne fjerne bestillinger der ikke længere er gældende, så systemet ikke kommer til at indeholde ugyldige ordrer.

US-10: Som bruger kan jeg benytte mig af hjemmesiden på både mobil, tablet og pc på en brugervenlig måde, så jeg har god mulighed for at tilgå hjemmesiden.

US-11: Som Kunde skal jeg indtaste en formular med adresse og andre informationer sådan at fog kan komme i kontakt med mig.

US-12: Som kunde kan jeg se en oversigt over mine tidligere ordrer, så jeg kan se de informationer jeg skal bruge til min bestilte carport.

Estimater

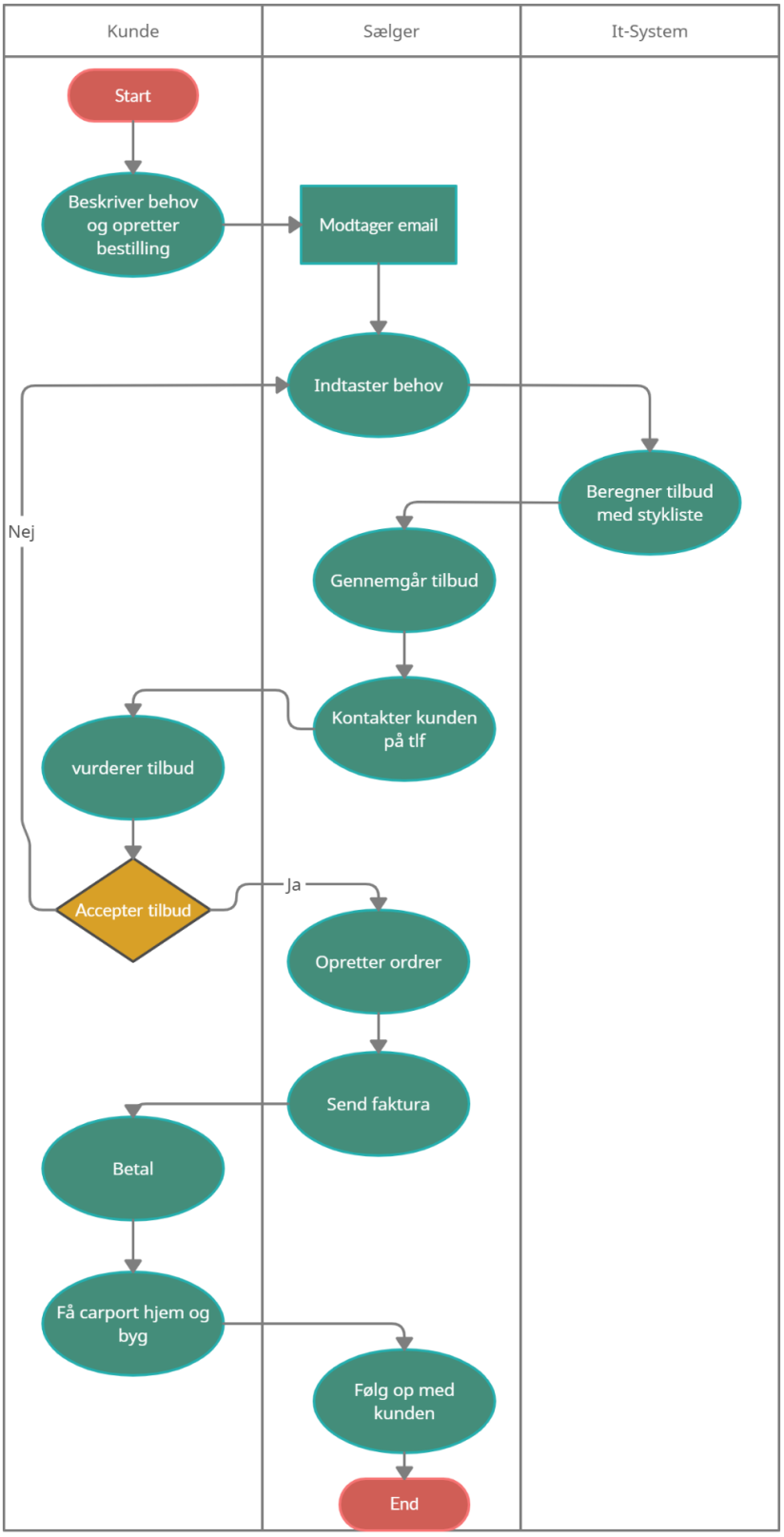
Hver af disse user stories er blevet givet et estimat afhængig af hvor lang tid vi mente den enkelte user story ville tage. Estimatet beregnes efter "t-shirt størrelser" og vi har brugt skalaen fra small til large. Vores estimater lyder således:

#	Estimate
US-1	L
US-2	L
US-3	S
US-4	L
US-5	M
US-6	M
US-7	S
US-8	M
US-9	M
US-10	L
US-11	S
US-12	S

Vores user stories er ydermere brudt ned i tasks på hvert sprint. Dette vil kunne ses på vores Taiga backlog på dette link: https://tree.taiga.io/project/janussr-fog/backlog

Arbejdsgange der skal IT-støttes

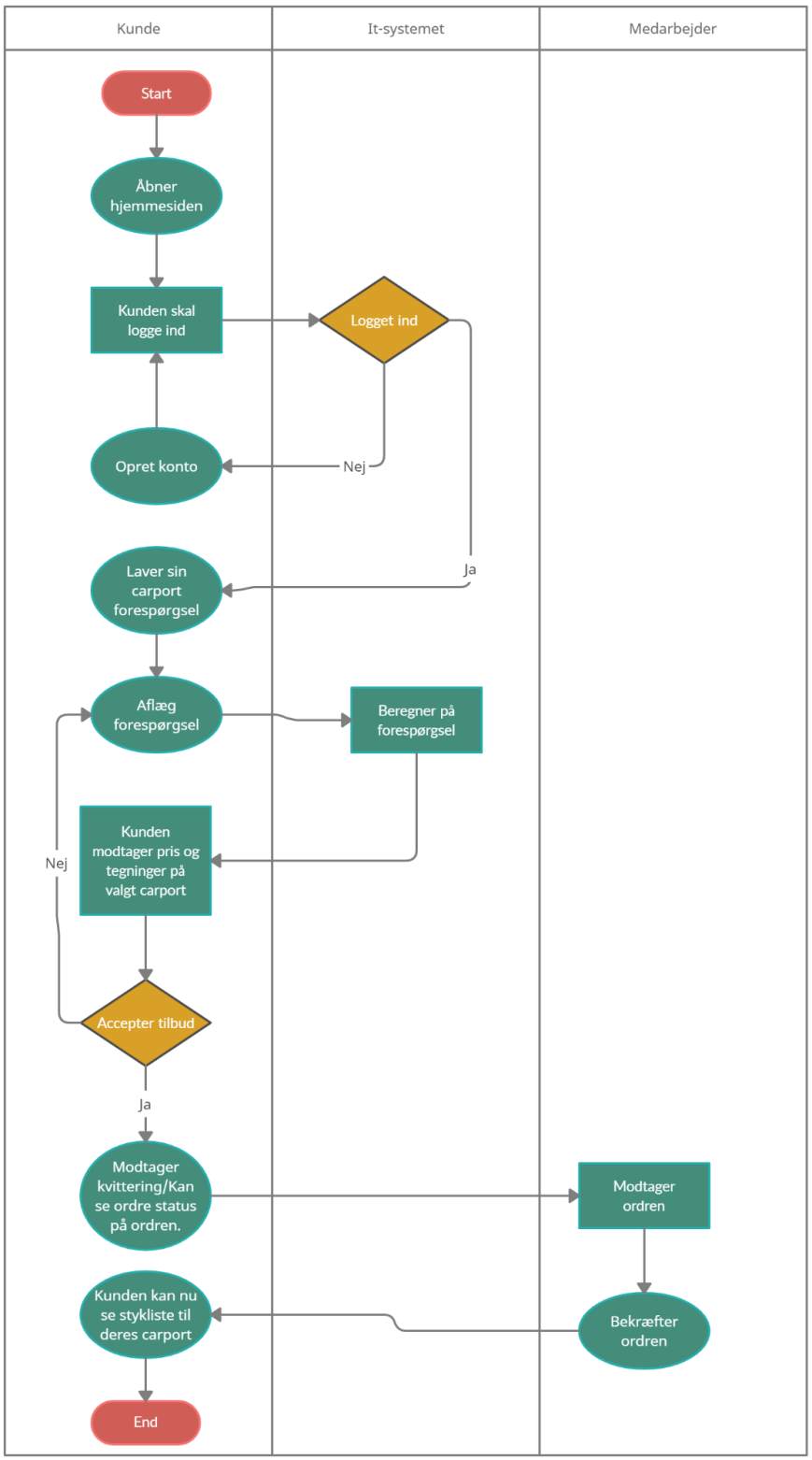
As-Is Aktivitetsdiagram



Vores as-is aktivitetsdiagram beskriver fremgangsmåden i en bestilling af en carport fra Johannes Fog nuværende system, som lederen Martin har beskrevet det.
Som diagrammet viser, starter kunden med at oprette en bestilling ud fra ønsket carport størrelse, derefter modtager sælgeren en email med kundens bestilling.
Sælgeren indtaster så informationerne om carporten i deres it-system kaldet *quickbyg*, som så hjælper med at udregne en pris til både kunden, og fortjeneste til Johannes Fog.
Sælgeren kigger så tilbuddet igennem, og kan vælge at ændre på pris, i tilfælde af at kunden skal have rabat eller lignende.
Sælgeren kontakter kunden på tlf, og kunden har så mulighed for at vurdere om de er interesseret, hvis nej er der mulighed for at gå tilbage og ændre i carport, for at få en anden, eller finde et andet firma at købe fra.

Hvis de derimod er interesseret og accepterer tilbuddet, opretter sælgeren ordren og sender en faktura til kunden.
Kunden betaler så for sit produkt, og får deres carport. Til sidst er der mulighed for at sælgeren kontakter kunden for at følge op på ordren, og sikre sig at alt er som ønsket.

To-Be Aktivitetsdiagram

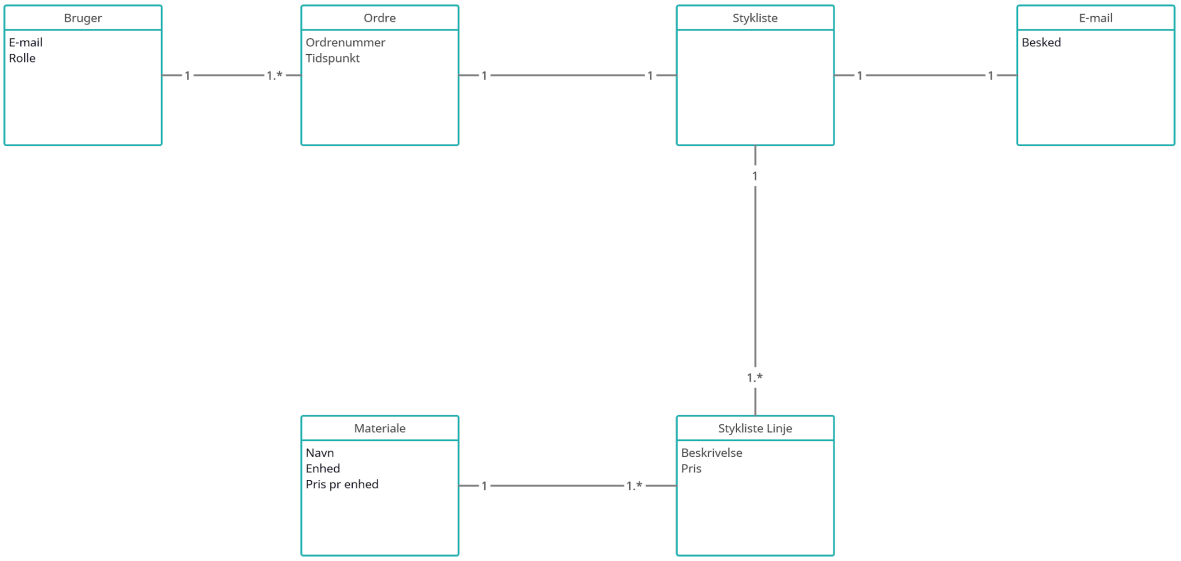


I vores to-be aktivitetsdiagram, starter kunden på hjemmesiden, og skal derefter logge ind eller oprette en konto, før de kan lave nogen handlinger. Efter de er logget ind, har de mulighed for at skræddersy deres carport, som de godt kunne tænke sig den, de aflægger så en forespørgsel, som it-systemet beregner, og sender derefter et tilbud tilbage til kunden, hvor de kan se pris og tegninger. De kan vælge enten at accepterer tilbuddet, eller gå tilbage, og ændre i deres bestilling. Vælger de at accepterer tilbuddet, modtager de en kvittering, og har mulighed for at se status på deres ordrer. Medarbejderen modtager ordren, og kan bekræfte den, så kunden kan komme ind og se sin stykliste til deres nye carport.

Så de implementationer vi har opgraderet, er at der ikke behøver at være en sælger inde over, som skal skrive ordren ind manuelt for at se prisen, for det gør vores system automatisk. Vi har også samlet det hele på én hjemmeside, så der ikke er behov for flere forskellige programmer. Medarbejderen, kan også tilføje materialer, opdatere i prisen, og fjerne materialer, som heller ikke var muligt i deres tidligere program.

UML

Domæne Model



Beskrivelse

Vores domænemodel viser de domæne- eller entitetsklasser vi i starten af projektet havde forestillet os vi skulle have med i programmet, samt deres relationer. Herunder: *Bruger, Ordre, Stykliste, Email, Stykliste Linje, Materiale*.

Relationer beskrevet

En bruger kan have mange ordre, men en ordre vil kun have en bruger tilknyttet. Der vil være en stykliste tilknyttet hver ordre. Den stykliste er tilknyttet én mail der bliver sendt til lageret på Fog. Styklisten har mange linjer som hver er tilknyttet den samme stykliste. En stykliste linje har kun ét materiale, men et materiale kan godt forekomme på flere stykliste linjer.

ER Diagram

Beskrivelse

Ovenfor ses vores ER diagram, der viser hvordan vi har valgt at opsætte vores database. Hver tabel har en *primary key* som kan ses på de attributter der er markeret med en gul nøgle. Vi forbinder vores tabeller med *foreign keys* som refererer til et ID tilhørende den tabel der er forbundet en relation til. Disse er markeret med rødt.

Vores database opfylder på nuværende tidspunkt ikke den tredje normalform. Vi valgte i starten af projektet at simplificere vores database, således at den var let tilgængelig at arbejde med. Og eftersom at vi stadig mangler diverse funktionaliteter i programmet, har vi ikke prioriteret at yderligere normalisere vores database.

For at databasen skulle opfyldes på tredje normalform, vil det have krævet en række flere tabeller. Dette kunne for eksempel være at adskille *role* fra user tabellen og oprette en *role* tabel med relation til user tabellen.

User

Hver række i denne tabel præsenterer en bruger i programmet. Dette er gældende både for en kunde og for en medarbejder.

I tabellen har vi følgende attributter udover den primære nøgle:

- En email der bruges til at logge ind på hjemmesiden.
- Et password der bruges til at logge ind på hjemmesiden.
- En rolle der definerer hvilken bruger der bruger hjemmesiden (kunde eller medarbejder).

Orders

Hver række i denne tabel repræsenterer en ordre eller forespørgsel på en carport foretaget af en kunde.

I tabellen har vi følgende attributter udover den primære nøgle:

- Et bruger ID så man ved hvilken bruger der har lagt ordren.
- En pris der beskriver total prisen af hele ordren, altså hvad carporten i sidste ende kommer til at koste.
- En status der beskriver om ordren er henholdsvis en forespørgsel, under behandling eller bekræftet.
- Carportens længde
- Carportens højde
- Et tidspunkt for hvornår ordren er foretaget.

Bom_items (Stykliste Linje)

Hver række i denne tabel repræsenterer en enkelt linje på en stykliste.

I tabellen har vi følgende attributter udover den primære nøgle:

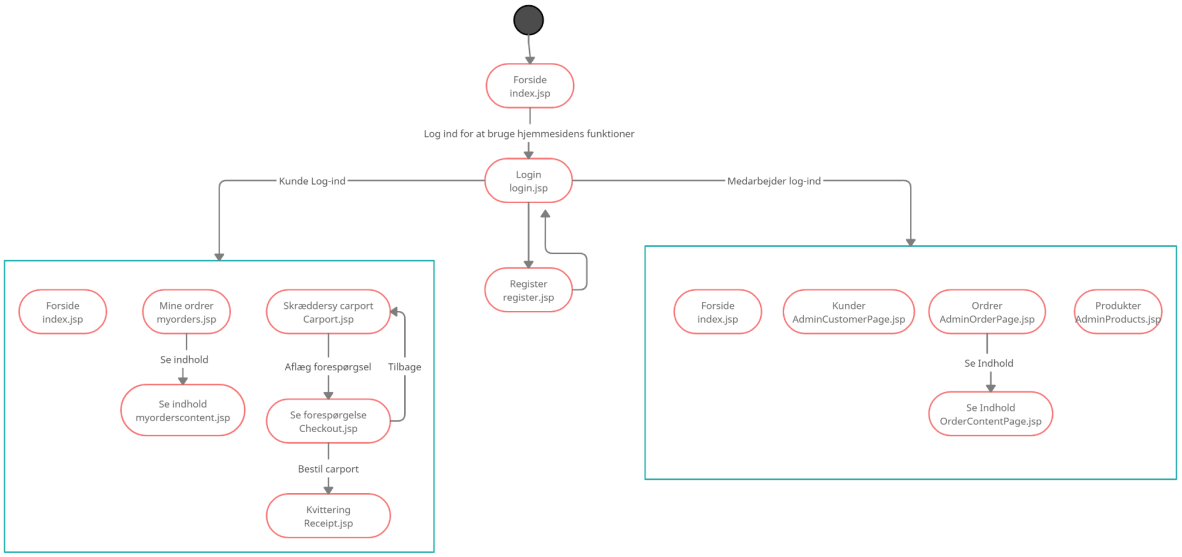
- Et ordre ID så man ved hvilken ordre styklisten tilhører.
- Et materiale ID så man ved hvilket materiale der skal bruges til stykliste linjen.
- Navnet på det materiale der bruges.
- Antallet af materialer der skal bruges.
- Længden af det materiale der skal bruges.
- En enhed på det materiale der bliver taget i brug. For eksempel “Stk”.
- En kort beskrivelse hvad materialet skal bruges til.
- En pris der beskriver materialeprisen ganget med antallet.

Material

Hver række i denne tabel repræsenterer et materiale eller produkt som Fog har til rådighed på deres lager.
I tabellen har vi følgende attributter udover den primære nøgle:

- En kort beskrivelse eller navn på materialet.
- Hvilken enhed materialet bliver leveret i. For eksempel “Stk”.
- En pris pr enhed der beskriver hvad, for eksempel et enkelt stk træ koster.
- En type der beskriver hvilken type materialet er. Eksempelvis om det er træ eller skruer osv.

Navigationsdiagram



På vores hjemmeside, bruger vi en fælles navigations bar, som bliver brugt på alle siderne. Navigationsbaren indeholder forskellige funktioner alt efter, om man er logget ind som medarbejder, eller kunde.
Som det kan ses på diagrammet, starter man på *index.jsp*. Man kan ikke bruge nogle af funktionerne med mindre man logger på, så det er selvfølgelig næste trin når man vil bestille en carport.

Når man logger på som kunde, kan man gå på *index.jsp*, og kigge på egne ordre på *myorders.jsp*. Den indeholder ikke noget, før man har købt en carport, og medarbejderen har bekræftet ordren.
Når man på forsiden vælger at klikke på skræddersy carport bliver man omdirigeret til *carport.jsp*, der kan man så vælge størrelsen på den ønskede carport. Derefter kan man klikke

aflæg forespørgsel, så man bliver omdirigeret til *checkout.jsp,* hvor man kan se sine valgte størrelser, pris og tegninger på carporten. Hvis man ikke er tilfreds, kan man gå tilbage, og ændre størrelsen. Men ellers klikker man på bestil carport. Hvor man så får sin kvittering vist på *receipt.jsp*.
Hvis kunden vil se sin ordrer på *myorders.jsp* siden, kræver det at medarbejderen har bekræftet kundens ordre, så indtil da skal kunden vente, på bekræftelse.

Når man logger ind som medarbejder, har man tre funktioner der står i navigationsbaren. Hvis man klikker på kunder, bliver man omdirigeret til *AdminCustomerPage.jsp*. Der kan man se en liste over oprettede kunder.
Man kan se alle materialer, der bliver brugt til en carport hvis man klikker på produkter og bliver omdirigeret til *adminproducts.jsp*. Så kan man til sidst gå under ordrer siden *AdminOrderPage.jsp*, og se alle kundernes ordrer, hvor man har mulighed for at klikke på se indhold for at få vist den enkelte kundes ordre på *OrderContentPage.jsp*. Ellers er der mulighed for at bekræfte kundens ordre, på ordrer siden, så kunden kan få sin stykliste vist.

Vi har valgt at opdele det således, at kunden har muligheden for at bestille en carport, og se sin ordre status efter køb, og styklisten, når medarbejderen har bekræftet ordren.
Medarbejderen har så mulighed for at bekræfte kundens bestilling, se en oversigt over de oprettede kunder, se kundernes ordrer, og se en liste over firmaets materialer.

Mockup

Før vi begyndte med at kode, lavede vi et mockup på hvordan, i grove træk, vi havde tænkt vores hjemmeside skulle se ud. Vores mockup er designet i Adobe XD som er et program til at sammensætte og designe UI/UX til for eksempel en hjemmeside. Vi har vedhæftet vores mockup på github under mappen **Mockup**.

Valg af arkitektur

Vores projekt tager brug af en udleveret startkode. Det vil sige at vores kodes arkitektur afspejler sig i den startkode og er mere eller mindre opbygget rundt om den.

Startkoden kommer med en række funktionaliteter der er implementeret på forhånd. Heriblandt diverse sider som for eksempel en forside, header, footer, customer/employee sider med mere. Startkoden kommer også med et login system samt logikken bag det.
For at forstå startkoden, kræver det en forståelse af den arkitektur og de design patterns der er blevet brugt. Disse *design patterns* er som følgende:

- Model View Controller (MVC pattern)
- Front Controller pattern
- Command pattern
- Singleton pattern
- Facade pattern
- Dependency injection

Model View Controller

Model View Controller eller MVC er et design pattern der bruges til at adskille forskellige lag af kode i et program så der på bedste vis kan opnås en løsere kobling i koden. Det drejer sig om tre lag, henholdsvis model, view og controller.

Model laget behandler programmets logik og data, i vores program kunne det for eksempel være vores data mappers og vores service-klasser.

View laget er enhver form for præsentation af information eller data. I et webprojekt ville dette i de fleste tilfælde være selveste hjemmesiden, altså vores jsp sider og den html der bruges til at skabe dem.

Controller laget behandler en forespørgsel fra en klient og fortæller serveren hvad der skal ske med den forespørgsel. Den fungerer oftest som en mellemmand mellem de andre lag, model og view, da controlleren modtager data fra model laget så den senere kan behandle den og give den videre til view laget. I projektet vil en repræsentation af controller laget være front controller klassen samt vores command-klasser.

Startkoden er systematisk inddelt i tre lag der på bedste vis efterligner MVC konceptet. Vi har business laget (model) hvor vores logik er. Web laget (controller) hvor vores front controller og commands er. Og vores klient som er webbrowseren der tilgår hjemmesiden via http (view).

Front Controller pattern

Startkoden er i stor grad afhængig af front controller klassen da det er den som står for hele flowet i programmet. Ligesom controller laget i MVC, står front controlleren for at behandle forespørgsler eller *requests* sendt mellem klient og server, både den ene og den anden vej. Det er front-controlleren der står for den rutning mellem jsp siderne når en command eksekveres.

Command pattern

Command pattern er et design pattern der som funktion står for at indkapsle en request sendt fra klienten som et objekt for senere, at gemme eller eksekvere requesten. I startkoden er command klassen en abstrakt klasse man kan extende, hvor alle commands har samme metode til fælles, nemlig *execute()*. De to underklasser man kan extende er henholdsvis *CommandProtectedPage* og *CommandUnprotectedPage*. Hvis siden commanden der skal vises kun skal kunne tilgås af en bestemt defineret bruger, skal *CommandProtectedPage* bruges som tager to parametre (*pageToShow* og *role*). Hvis siden skal kunne tilgås af alle brugere på hjemmesiden, også dem der ikke er logget ind, kan *CommandUnprotectedPage* bruges der blot tager en parameter (*pageToShow*).

Singleton pattern

Singleton pattern er et design pattern hvis funktion er at sikre sig, at en klasse altid kun vil have én instans af et objekt tilhørende klassen. Eksempler på singleton i vores projekt er for eksempel ved instantiering af vores stykliste (*Bom*) hvor der kun må laves et nyt objekt af styklisten hvis den er lig med *null.* Eksempelvis:

```
if (bom == null) {
bom = new Bom();
}
```

Facade pattern

Facade pattern er et design pattern der som formål har at gøre kode mere læselig og gemme store dele af kode væk, bag en *"facade"*. Facaden vil herefter virke som et mellempunkt mellem controller laget. Facade pattern bliver ofte brugt når kode bliver meget komplekst og svært at navigere, facaden gemmer alt det ulæselige væk og viser kun et metode navn og hvad den returnerer i vores tilfælde.

Dependency injection

Dependency injection er en teknik hvor et givet objekt modtager andre objekter som den er afhængig af. Et eksempel på dette i startkoden er i vores data mappere samt facadeklasser som alle indeholder et database objekt. I dette database objekt ligger JDBC connect strengene som så vilkårligt kan ændres hvis nødvendigt. Dette giver en super løs kobling da der kun skal ændres i disse strenge for at forbindelsen til databasen skal opnås i data mapperne og diverse service klasser.

Særlige forhold

Brug af scopes

Java servlets indeholder en række forskellige scopes der bruges til at opbevare forskellig data i. Det drejer sig om tre scopes: *RequestScope, SesssionScope* og *ApplicationScope*. Hvert scope har en bestemt levetid. Request Scopet vil kun hentes i det siden genindlæses og lever indtil request objektet ophører, dette scope har den korteste levetid. Session Scopet lever i en såkaldt *"session"* som bliver ved med at køre så længe klienten interagerer med siden. Standard levetiden på en session er 30 minutter, så hvis klienten er inaktiv i hele perioden vil scopet lukke. Application Scopet bliver startet fra det tidspunkt hjemmesiden bliver deployet og lever så lang tid som hjemmesiden er online.
I dette projekt har vi kun benyttet os af Request og Session scopet.
Request Scopet har vi brugt til fremvisning af alt information og data. De steder vi viser informationer i opgaven skal siden oftest kun tilgås en enkelt gang og der har derfor ikke været behov for, at gemme informationerne i et session scope. Det er informationer som lister, priser og diverse beskeder vi sender til brugeren.

Session Scopet har vi kun benyttet en enkelt gang til at opbevare en bruger. Det gør så der opnås adgang til brugerens informationer på alle sider så længe sessionen stadig kører.

Fejlhåndtering

For at hjælpe brugeren godt på vej på hjemmesiden er der et behov for fejlhåndtering på de fejl en bruger kunne begå, når man navigere en hjemmeside. Startkoden har i forvejen håndteret en række fejl der kunne forekomme ved login-systemet. For eksempel hvis brugeren indtaster et ugyldigt brugernavn eller password, sendes en fejlmeddelelse til brugeren for at gøre dem opmærksom på dette.
Vi har også selv tilføjet fejlhåndtering i programmet.
For en kunde, vil der vises en fejl hvis kunden ikke er logget ind på hjemmesiden ved klik på *"skræddersy din carport"*. På den måde sikre vi os, at en bruger altid er logget ind når en bestilling eller forespørgsel af en carport påbegyndes, så vi kan få de nødvendige brugerinformationer med på ordren. For at gøre dette skal vi lave en condition, der tjekker både om knappen er blevet trykket på, men også at brugeren er logget ind. Eksempelvis:

```
if (request.getParameter("begin") != null &&
session.getAttribute("user") == null) {
   throw new UserException("Du skal være logged ind for at
skræddersy en carport.");
}
```

Vi kaster altså en custom exception *UserException* som er en del af den udleverede startkode. UserException indeholder en constructor med en String *msg* som er den besked brugeren vil få vist, hvis vores condition er opfyldt.

Vi har derudover også lavet fejlhåndtering på vores materialeoversigt ved selve *delete* funktionen. Det har vi tilføjet fordi det er vigtigt at en medarbejder ikke sletter et materiale, som er taget i brug til sammensætningen af en carport i vores *BomService*. Denne fejlhåndtering er dog udarbejdet anderledes end den forhenværende. I stedet for at kaste en exception, har vi skræddersyet vores sql statement på delete mapperen, så den kun kan slette et materiale, hvis id'et på dette materiale ikke befinder sig på en stykliste i databasen. Vores delete mapper metode returnerer derefter en integer: *rowsAffected* som er lig med de antal rækker mapperen har eksekveret en opdatering på. Hvis *rowsAffected* er større end 0, altså at

en eller flere rækker blev opdateret, så vil metoden eksekveres. Hvis ikke sendes der en fejlmeddelelse til brugeren. Eksempelvis:

```
if (materialFacade.deleteMaterial(Integer.parseInt(deleteId)) > 0)
{
   //Update lists
   materials = materialFacade.getAllMaterials();
   wood = materialFacade.getAllWood();
   accesories = materialFacade.getAllAccesories();
} else {
   request.setAttribute("error", "Dette materiale kan ikke fjernes
da det er i brug.");
}
```

SVG Tegning og tilhørende materiale beregninger

I projektet havde vi som opgave at fremstille en tegning af den skræddersyede carport som kunden havde bestilt. Til det benyttede vi *SVG (Scaleable Vector Graphics)* SVG er et XML baseret billedformat der gør det muligt at fremstille en tegning ved brug af forskellige XML tags. Disse tags kan man så med fordel indsætte i et HTML dokument hvor tegningen så vil blive renderet og vist på siden.

Tegningen skulle være dynamisk efter de mål kunden indtastede ved bestillingen. Til det oprettede vi en java klasse *SVG* hvis opgave, var at bygge en streng som indeholdte de nødvendige XML tags, der skulle bruges til at fremstille en komplet SVG tegning. I klassen oprettede vi en slags template metoder til for eksempel at tegne en firkant eller en simpel linje. Vi kunne så med fordel kalde disse templates i vores *DrawingService* klasse hvor alle tegningerne blev lavet ud fra de korrekte beregninger gemt i variabler.

For at lave tegningen skulle vi beslutte hvordan og hvorledes vores beregninger skulle laves i forhold til de materialer der skulle tegnes. De materialer er henholdsvis: Spær, Rem, Stolper og Hulbånd.

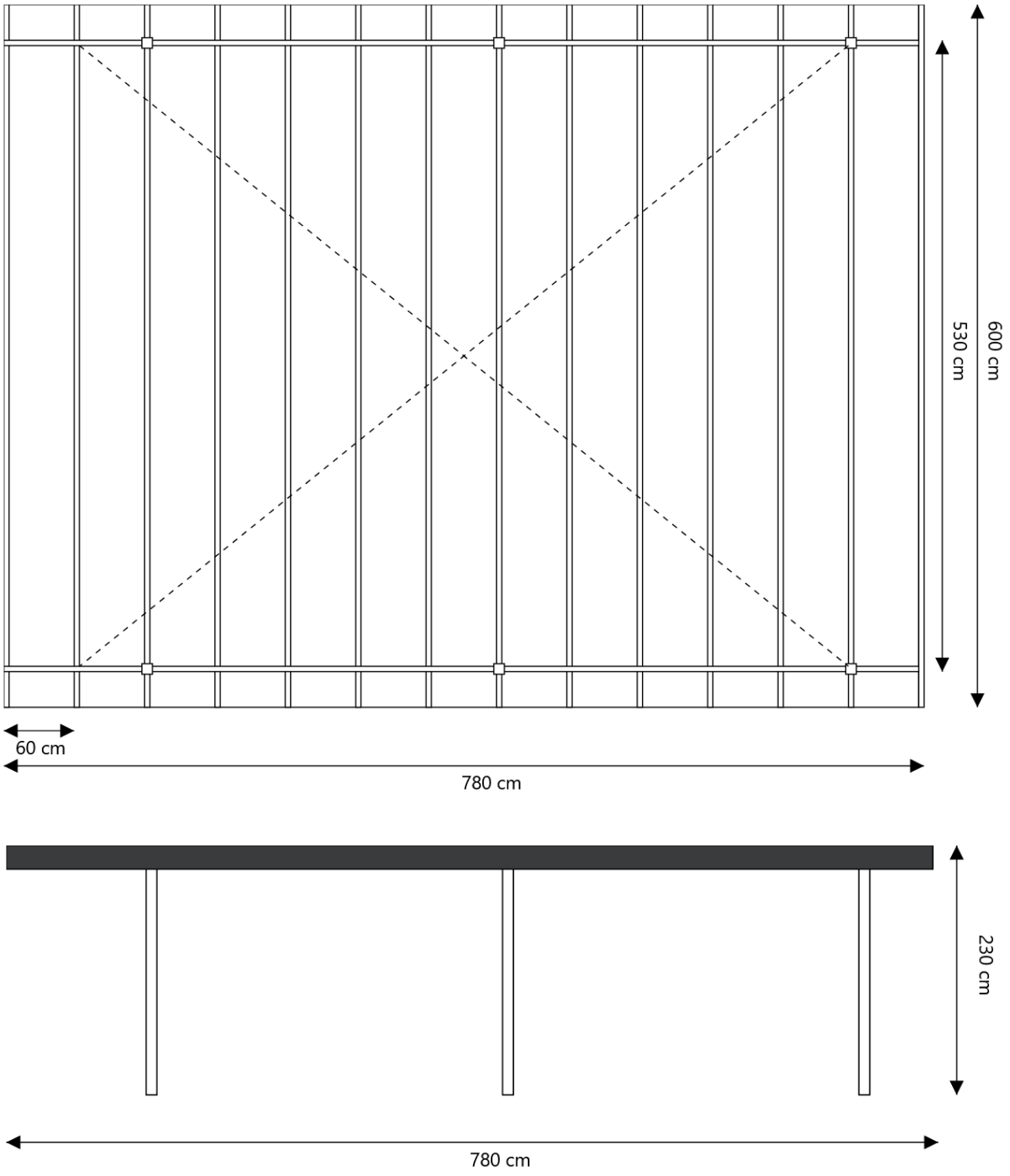
I forhold til vores spær, har vi som udgangspunkt besluttet at der maksimalt må være 60 cm mellem hvert spær på tegningen. Dette var også dokumenteret som maks distancen i den udleverede vejledning fra Fog vedrørende en carport med målene 6.0 gange 7.8 meter. Hvis 60 ikke går op i carportens længde, vil distancen mellem spærene være mindre og kræve flere spær.

Placering på vores rem er beregnet ud fra distancen i procent fra carportens bredde og bredden mellem hver rem på reference tegningen i vejledningen.

Stolpernes placering er afhængig af, hvor mange stolper der skal bruges til carporten. Ud fra vores beregninger, vil carporten, afhængig af de mål der bliver indsat, have fire eller seks stolper.
Hvis carporten har fire stolper, har vi blot placeret en stolpe på hver side af carporten, på nummer to spær samt det andet sidste spær.
Hvis carporten derimod har seks stolper, har vi placeret den første stolpe på det tredje spær. De resterende placeres så med et mellemrum på fire spær.

Med hulbåndet skulle vi blot finde start og slut koordinatet på stregen. Hulbåndet går fra nummer to spær til det andet sidste og går fra rem til rem på tværs af carporten.
Vi har udarbejdet to tegninger, en set oppefra og en set fra siden. Begge tegninger er iført mål der er nyttige at vide når man skal bygge carporten.

Vi har valgt at fokusere på de stykliste beregninger der skulle bruges til tegningen. Det har betydet at de resterende beregninger i nogle tilfælde vil forekomme som statiske og altså ikke er dynamiske efter carportens mål.
Et eksempel på vores SVG tegninger ud fra målene 6.0 gange 7.8 meter:



Udvalgte kodeeksempler

Eksempel på en beregningsmetode

```
public BomLine calculateSpærFromMeasurements(int width, int
length) throws UserException {
   Material material = materialFacade.getMaterialById(5);
```

```
    int materialId = material.getMaterialId();
    String name = material.getDescription();
    double dlength = length;
    double n = Math.ceil(dlength / 60);
    double dquantity = n + 1;
    int quantity = (int) dquantity;
    int materialLength = width;
    String unit = material.getUnit();
    String description = "Spær, monteres på rem";
    double price = 0;

    for (int i = 0; i < quantity; i++) {
        price += material.getPricePerUnit();
    }


    BomLine bomLine = new BomLine(materialId, name, quantity,
materialLength, unit, description, price);

    return bomLine;
}
```

Eksempel på en tegning i DrawingService

```
public SVG drawCarportTop(double width, double length, int
orderId) throws UserException {
    List<BomLine> billOfMaterials =
bomFacade.getBomByOrderId(orderId);
    String viewBox = "0, 0, " + 855 + ", " + 855;
    SVG svg = new SVG(0, 0, viewBox, 100, 100);

    //Draw Frame
    svg.addRect(0, 0, width, length);

    //Draw Spær
    BomLine spær = billOfMaterials.get(1);
    double dquantity = spær.getQuantity();
    double distance = length / (dquantity - 1);
    for (int x = 0; x < spær.getQuantity(); x++) {
        svg.addRect(distance * x, 0, width, 4.5);
```

```
    }

    //Draw Rem
    double remDistance = width / 100 * 5.83;
    svg.addRect(0, remDistance - 4.5, 4.5, length);
    svg.addRect(0, width - remDistance, 4.5, length);

    //Draw Stolper
    BomLine stolpe = billOfMaterials.get(0);
    double firstDistance = distance * 2;
    double distanceBetween = distance * 5;
    double secondDistance = distance * spær.getQuantity() - (2 *
distance);
    if (length > 630) {
        for (int x = 0; x < stolpe.getQuantity(); x++) {
            svg.addRect(firstDistance + distanceBetween * x - 2.25,
width - remDistance - 2.25, 9, 9);
            svg.addRect(firstDistance + distanceBetween * x - 2.25,
remDistance - 4.5 - 2.25, 9, 9);
        }
    } else {
        firstDistance = distance * 1;
        svg.addRect(firstDistance - 2.25, width - remDistance -
2.25, 9, 9);
        svg.addRect(secondDistance - 2.25, width - remDistance -
2.25, 9, 9);
        svg.addRect(firstDistance - 2.25, remDistance - 4.5 - 2.25,
9, 9);
        svg.addRect(secondDistance - 2.25, remDistance - 4.5 -
2.25, 9, 9);
    }

    //Draw Hulbånd
    double firstDistanceHulbånd = distance * 1;
    svg.addLine(firstDistanceHulbånd + 4.5, remDistance,
secondDistance, width - remDistance);
    svg.addLine(secondDistance, remDistance, firstDistanceHulbånd +
4.5, width - remDistance);

    return svg;
}
```

Eksempel på vores createOrder metode fra OrderMapper.

```
public void createOrder(int userId, int length, int width,
List<BomLine> bomLines) throws UserException {
   double orderPrice = 0;
   int orderId = 0;
   String status = "Forespørgsel";

   for (BomLine bl : bomLines) {
       orderPrice += bl.getPrice();
   }
   try (Connection connection = database.connect()) {
       String sql = "INSERT INTO orders
(user_id,price,status,length,width) VALUES (?,?,?,?,?)";

       try (PreparedStatement ps =
connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS))
{
           ps.setInt(1, userId);
           ps.setDouble(2, orderPrice);
           ps.setString(3, status);
           ps.setInt(4, length);
           ps.setInt(5, width);


           ps.executeUpdate();
           ResultSet ids = ps.getGeneratedKeys();
           ids.next();
           orderId = ids.getInt(1);
       } catch (SQLException ex) {
           throw new UserException(ex.getMessage());
       }

       for (BomLine bomLine : bomLines) {
           insertIntoBomItems(orderId, bomLine);
       }

   } catch (SQLException | UserException ex) {
       throw new UserException(ex.getMessage());
   }
}
```

Status på Implementering

Vi har i alt haft tolv user stories, som skulle implementeres i vores projekt, ud af de tolv nåede vi syv. De fem user stories vi ikke nåede var af lavere prioritering, og er derfor ikke blevet implementeret, da der var andre funktionaliteter der var vigtigere for os at have fuldendt før deadline.
Vi havde fokus på det mest essentielle i programmet der skulle til, for at hjemmesiden virkede som planlagt. De fem user stories vi ikke nåede lyder således.

User Stories vi ikke fik lavet

US-5: Som lagermedarbejder kan jeg modtage en e-mail når en kunde har lagt en bestilling, så jeg kan behandle kundens ordre ud fra en tilsendt stykliste.
US-7: Som administrator skal jeg kunne se kundens ordrehistorik på en kundeoversigt sådan at de rigtige varer bliver sendt ud til kunden.
US-10: Som bruger kan jeg benytte mig af hjemmesiden på både mobil, tablet og pc på en brugervenlig måde, så jeg har god mulighed for at tilgå hjemmesiden.
US-11: Som Kunde skal jeg indtaste en formular med adresse og andre informationer sådan at fog kan komme i kontakt med mig.
US-12: Som kunde kan jeg se en oversigt over mine tidligere ordrer, så jeg kan se de informationer jeg skal bruge til min bestilte carport.

Styling

I forhold til styling af vores projekt, er alle sider stylet, men det er ikke alt der er responsivt. For eksempel, er vores tables ikke responsive, hvis der kommer for mange kolonner. Det kunne vi have løst med bootstrap ved brug af en bootstrap class *"table-responsive"*. Det gav dog nogle andre komplikationer og krævede noget mere styling for at gøre det pænt, så det droppede vi.
Vi har valgt at nedprioritere styling og diverse HTML dummy tekstelementer i denne omgang, og kun fokuseret på de elementer der gør vores logik brugbar.

Bugs vi ikke fik rettet

På styk- og materiale listen fremgår hver en pris som double. Vi ville gerne have formateret denne pris til et ekstra decimal ved brug af DecimalFormat objektet i Java. Dette har vi tidligere gjort på vores *totalPrice* variabel uden problemer. Men da prisen på listerne er en del af et *material* eller *bomItem* objekt var der komplikationer med DecimalFormat, fordi den ændrede vores double til en streng og smed en fejl da vi derefter castede den tilbage til en double.

Test

Vi har i projektet udført både unit-tests ved brug af JUnit 5 på vores beregningsmetoder samt integrationstests på vores data mappere.
I vores unit tests har vi taget udgangspunkt i diverse beregningsmetoder på forskellige materialer der skal bruges til at samle en carport. Disse befinder sig i klassen *BomService*. Vi har taget udgangspunkt i de materialer der skal bruges til vores SVG tegning, da vi mente at det netop var disse metoder der skulle testes ordentligt igennem, så vi vil få det forventede resultat og en præcis tegning. Det er også disse metoder vi har lagt vægt i at lave dynamisk i forhold til carportens længde og højde.
Der er tilføjet ækvivalensklasse partitionering, hvor vi tester hvor grænsen går mellem, de to forventede resultater på hver test.

Vi har udført integrationstests på flere af vores data mappere. I vores OrderMapper har vi testet en enkelt metode *createOrder*, hvor vi bruger *assertEquals* til at se forskellen på, om der er kommet en ekstra ordrer ind i vores database.
Det samme gør vi når vi tester på vores MaterialMapper klasse. Der har vi tilføjet de fire forskellige *create, read, update, delete* sql statements.
I vores create metode, foregår det på samme måde som da vi testede *createOrder* fra den anden test klasse. Så har vi *getAllMaterials*, som er vores *read*, der læser hvor mange materialer der er på vores database, og ved hjælp af *assertEquals*, sammenligner om der er det rigtige antal materialer i databasen.
Vores *updateMaterialById* tester *update* i metoden, hvor vi har sat en startpris, på et stykke materiale, og derefter en ny pris som den skal opdatere med, i dette tilfælde er startprisen på

100, og skifter derefter til 500, og til sidst 250, og med *assertEquals* sammenligner vi prisen, med den nuværende.

Til sidst har vi testet *delete,* med vores *deleteMaterial*, hvor vi i starten har indsat tre forskellige materialer, og i delete metoden sletter vi den ud fra id 1, hvor vi bagefter bruger *assertEquals,* til at sammenligne, at der er to materialer tilbage og ikke de tre vi startede med at indsætte.

I den skabelon vi har brugt til vores projekt, var der allerede lavet tests på user mapperen.

Når vi ikke testede ved hjælp af j-unit, gjorde vi således, at inden vi skubbede vores ændringer op til master branch, sørgede vi for at funktionerne virkede som ønsket. Det blev oftest gjort ved at starte projektet, også teste resultatet via hjemmesiden.

Vi har lavet dækningsgrad af vores tests for de valgte klasser og metoder, som ses på tabellen nedenunder. Tabellen viser, også procentdele af klassen der er blevet tests, som ses i højre kolonne.

Klasse	Metode der er testet	Metode %
BomService	• calculateStolperFromMeasurements • calculateSpærFromMeasurements • calculateRemFromMeasurements • calculateHulbåndFromMeasurements	88%(22/25)
MaterialMapper	• addMaterials • getAllMaterials • updateMaterialById • deleteMaterial	75% (6/8)
OrderMapper	• createOrder	33% (4/12)
UserMapper	• createUser • login • getAllUsers	75%(3/4)

Proces

Arbejdsprocessen faktuelt

I dette forløb har vi for første gang taget hul på et nyt begreb indenfor Scrum. Vi blev introduceret til begrebet *sprint*. Formålet med et sprint er at bryde et projekt ned i mindre bidder. Et sprint vil have en tidsbegrænsning og en deadline på, hvornår for eksempel en funktionalitet i et program skal være færdigt. I vores tilfælde var længden på et sprint syv dage og var inddelt i henholdsvis sprint et, to og tre.
Vi udarbejde disse user stories i hvert sprint:
I sprint 1:

- **US-1:** Som kunde kan jeg aflægge en forespørgsel til en skræddersyet carport efter egne mål, materialevalg og tag, så jeg kan få designet en carport efter mine behov.
- **US-2:** Som kunde kan jeg modtage et tilbud på en tilsendt forespørgsel ved en skræddersyet carport, så jeg kan bestille min ønskede carport.
- **US-3:** Som kunde skal jeg kunne indtaste mine informationer ved bestilling via. et login-system, sådan at sælgeren kan komme i kontakt med mig.

I sprint 2:

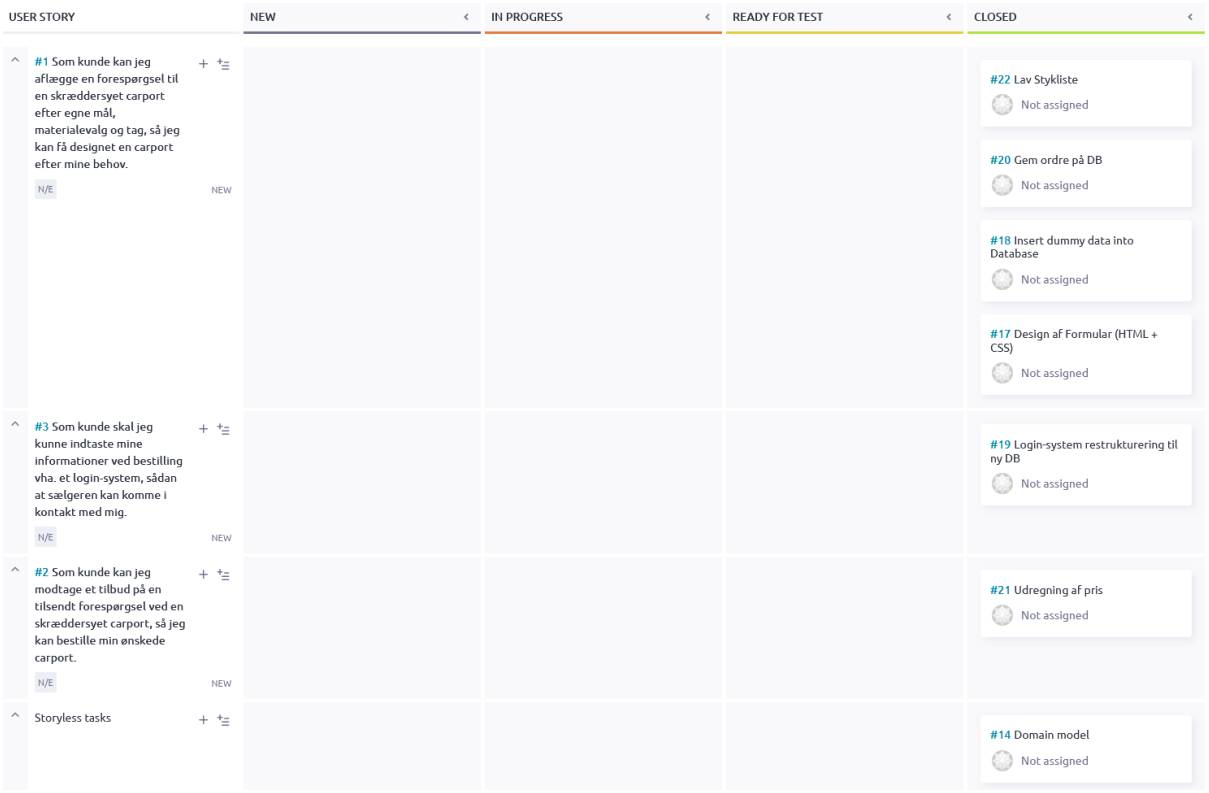
- **US-4**: Som kunde skal jeg kunne se en tegning på den sammensatte carport, så jeg kan få et visuelt billede over min carport.
- Derefter arbejdede på at færdiggøre de ikke-afsluttede user stories fra sprint 1 som var en blanding af user story 1 og 2.

I sprint 3:

- **US-6:** Som administrator skal jeg kunne se en kundeoversigt, så jeg kan få et overblik over vores kunder.
- **US-8:** Som administrator kan jeg tilføje og fjerne produkter samt produktbeskrivelser, sådan at kunden altid kan se de nyeste produkter vi tilbyder.
- **US-9:** Som administrator skal jeg kunne fjerne bestillinger der ikke længere er gældende, så systemet ikke kommer til at indeholde ugyldige ordrer.

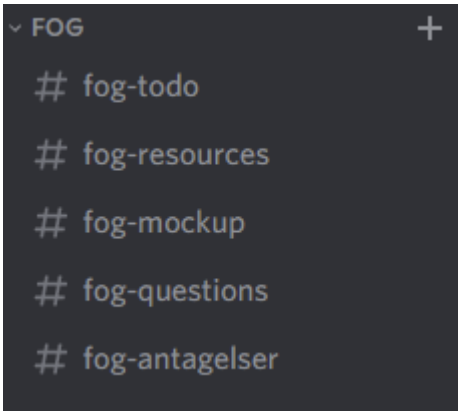
Vi har benyttet os af værktøjet Taiga til vores Scrum forløb. Vores board består af en backlog der indeholder alle de definerede user stories vi med product owneren fik aftalt før projektstart.

I begyndelsen af hvert sprint oprettede vi en sektion på boardet til det tilhørende sprint og aftalte med vores product owner, hvilke user stories der skulle arbejdes på den kommende uge. Når mødet var overstået fortsatte vi i gruppen et fælles møde og aftalte, hvilke tasks hver user story kunne nedbrydes til, her snakkede vi også om hvordan processen er gået set i retrospektiv. Taiga board for sprint 1:



Udover Taiga, har vi benyttet os af programmet Discord. Discord er et kommunikationsprogram med adgang til både tale- og tekst kanaler. Det var primært over Discord vi holdte vores møder og snakkede sammen over voicechat til de gange vi har haft behov for par programmering. Men udover dette brugte vi også Discord som en del af Scrum processen.

Vi oprettede forskellige tekst kanaler på discord som disse set på billedet:



Et eksempel på en af disse er *"todo"* kanalen. Todo fungerede som en form for eksternt scrum-board hvor vi skrev små tasks eller bugs op som vi stødte på når vi testede programmet. Hvis en task var blevet færdiggjort kunne vi så reagere på kommentaren med et flueben for at markere at denne task er fikset. Eksempel på dette:

Udover todo havde vi:

En ressource kanal hvis funktion er til deling af materiale som SQL dumps, UML-diagrammer og lignende.

En mock-up kanal hvor vores mock-up lå gemt så vi hurtigt kunne referer til det.

En questions kanal som vi brugte til at forberede spørgsmål til mødet med vores product owner inden hver sprint og til vores vejleder ved de tekniske reviews.
Og til slut vores antagelser som var der vi dokumenterede for de beregninger vi fandt frem til gennem forløbet.

Vores arbejdsdag begyndte altid med et aftalt morgenmøde mellem 09:00 og 10:00. Her dannede vi os et overblik og uddelegerede opgaver. Vi brugte også dette morgenmøde på review af features som vi hver især har fået implementeret. Når reviewet er overstået bruger vi tid på at pushe op til github og samtidig pulle hinandens arbejde ned lokalt. Vi løser også diverse merge conflicts der kunne forekomme hvis vi har arbejdet i samme klasse. Nogle tasks har vi løst ved par programmering fra morgenstunden af, men de fleste tasks har vi uddelegeret, så man selv får lov til at sidde med en opgave og fordybe sig i stoffet.

Arbejdsprocessen reflekteret

I og med at vores gruppe kun bestod af to personer, valgte vi ikke at uddelegere rollen som Scrum master. Vi mente ikke det ville gavne det store, givet gruppens størrelse.
Det vi gjorde i stedet var, at vi fælles traf de beslutninger der var nødvendige for at holde flowet i gang.
Vi fandt dog ud af, at produktet af dette resulterede i at vi brugte mindre tid på vores scrum-board på Taiga, efter vores tasks var blevet lavet. Fordi at der ikke var mange personer at holde styr på, så gav Scrum mindre mening for os. Men af samme årsag betød det også at selvom vores Scrum proces måske ikke var top-notch så kørte flowet stadig rigtig fint. Vi arbejdede primært ud fra verbale aftaler til vores daglige morgenmøder på discord og vores discord tekst kanaler. Set i bakspejlet vil vi dog stadig have fokuseret mere på vores Scrum board, da det er god træning til fremtidige projekter. Til næste projekt, for at sikre et bedre brug af vores taiga board, har vi snakket om, hvad vi kunne have gjort bedre, for at inkorporere taiga boardet noget mere i processen. En idé vi kom frem til var, at gøre det til at dagligt ritual ved morgenmødet at kigge taiga boardet igennem og flytte rundt på de tasks der skal closes og begyndes på.

Selve arbejdsprocessen i Fog kontra Cupcake projektet var en smule anderledes på grund af sprints og projektets længde. I cupcake var vi meget opsat på at komme rigtig godt fra start

og derfor kørte vi bare derudaf. I Fog projektet havde vi nok en mere afslappet tilgang og fokuserede mere eller mindre kun på det sprint og de tasks tilknyttet. Dette gjorde vi måske ikke nåede nogle ting som kunne have været gode at have med som vi havde nået hvis vi havde samme arbejdsmåde som i Cupcake.
Til gengæld betød det at der var flere fridage, hvilket for os har været tiltrængt efter intensiv arbejde både fra cupcake og fra hvert sprint i Fog.