Fog

af Daniel, Marc, Marcus, og Lasse

20/05-2020

Daniel Poulsen

Cph-Email: Cph-dp127@cphbusiness.dk

Github: Dane1998

Marc Ekström

Cph-Email: Cph-me206@cphbusiness.dk

Github: GGGE99

Marcus Jensen

Cph-Email: Cph-mj757@cphbusiness.dk

Github: MarcusJyl

Lasse Birk

Cph-Email: Cph-lb296@cphbusiness.dk

Github: LasseBirk96

Link til demovideo: https://youtu.be/kPG6DxvrG8g

Link til droplet: http://cph-projekter.me:8080/Fog-0/

Admin side: http://cph-projekter.me:8080/Fog-0/admin

Server login:

ip: 104.248.22.173

user: guest

Password: guest123/

Github repository: https://github.com/MarcusJyl/Fog

Sælger mail: cphprojekter@gmail.com

kode: 1123581321#

Link til Mockup:
https://xd.adobe.com/view/e26ce538-1dda-468a-7708-aa73e4d36852-945d/n

Indledning

Baggrund

Softwaren er lavet til det danske byggefirma Johannes Fog som har hovedsæde i Kongens Lyngby. Johannes Fog består af et Bolig & Designhus og ni Trælast & Byggecenter-butikker fordelt i hele Nordsjælland. Johannes Fogs krav til systemet grunder i at deres nuværende it system til at udvikle speciallavet carporte er outdated samt at de ønsker at deres system er mere brugervenligt og automatiseret. Kravene til systemet var at Johannes Fog kunne opdatere deres product backlog, i form af byggematerialer. Fog ønsker også en bedre visualisering overfor kunden og overordnet automatisering af deres it system, så de ikke behøver at skrive mål og materialer ind manuelt. Johannes Fog ønsker dog stadig at deres system er, som de kalder det, “lavpraktisk”, i form af at der en sælger der giver sender tilbuddet videre til kunden og har den fremadrettede kontakt. Dette er bibeholdt da dette fremmer den personlige tilgang til deres kunder.

Teknologi

Systemet er blevet kodet i Java via Intellij IDEA 2019 3.3 og benytter sig af SQL workbench og JDBC som database. Systemet benytter sig også af build-automatiserings værktøjet Maven. Hjemmesiderne er bygget i HTML på JSP sider, med JavaScript funktionaliteter.

Krav

Fog har et IT-system med en del mangler, der ønskes derfor et forbedret system der løser problemerne de har med det nuværende forældede system. Systemet er på mange fastlåste elementer således at værdierne ikke kan ændres. Det giver problemer
når vare og priser på vare skal udskiftes eller tilpasses. Der ønskes en let og simpel måde hvorpå der kan ændres i valgte værdier.
Som bruger skal man kunne vælge mål på sin ønskede carport, som systemet derefter generere en tegning ud fra. Tegningen skal indeholde mål på brugerens ønskede carport.
Fra brugerens valgte mål af carport evt. med skur, skal en komplet materialeliste genereres til sælger med en samlet pris.

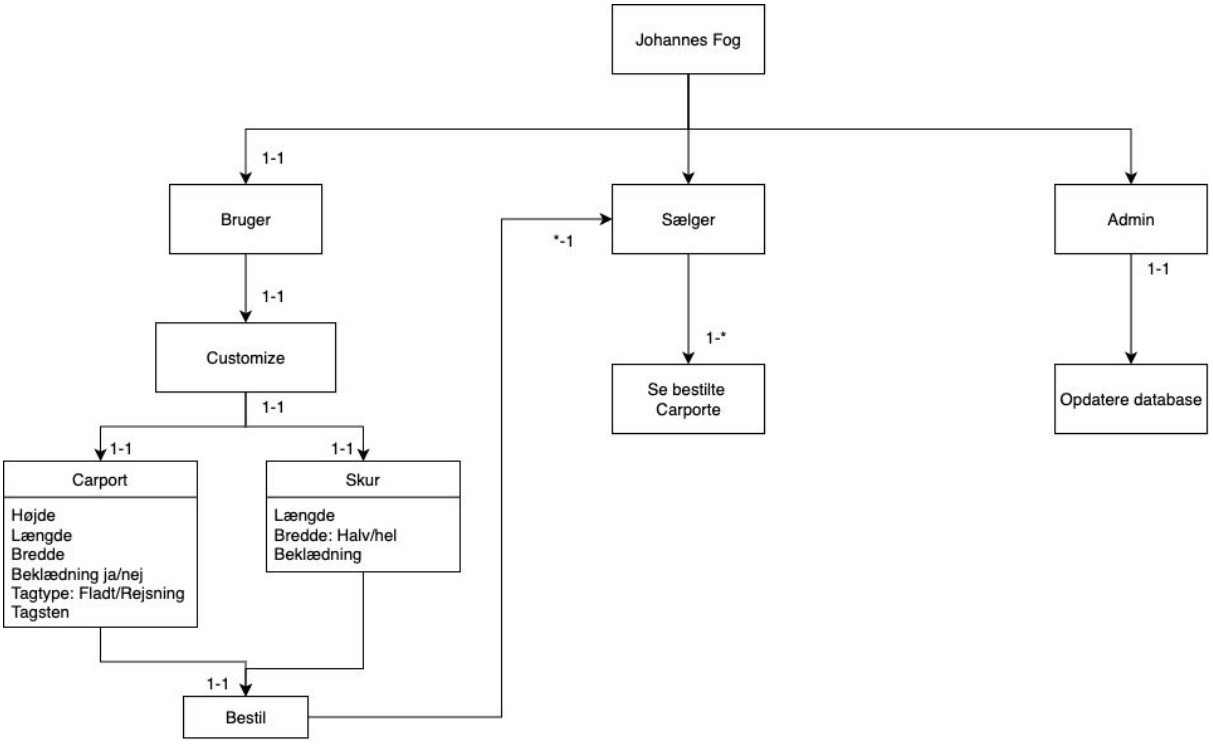
Implementeringen af det nye system hos Fog letter en række arbejdsopgaver som de havde svært ved at fuldføre. Systemet har førhen forhindret Fog i at tilpasse systemet til deres eget varelager, det har de nu let muligheden for.
Med opdateret og nøjagtig materialeliste bliver arbejdsprocessen lettere når materialer til bestilt carport skal samles. Færre fejl vil blive lavet da mere præcise lister vil blive genereret.

For kunden bliver deres personlig tilpasset carport genereret i en tegning, hvilket giver kunden forbedret visualisering af hvordan det endelige resultat kommer til at se ud. Dette letter beslutningsprocessen for kunden og flere carporte vil således blive solgt.

User Stories

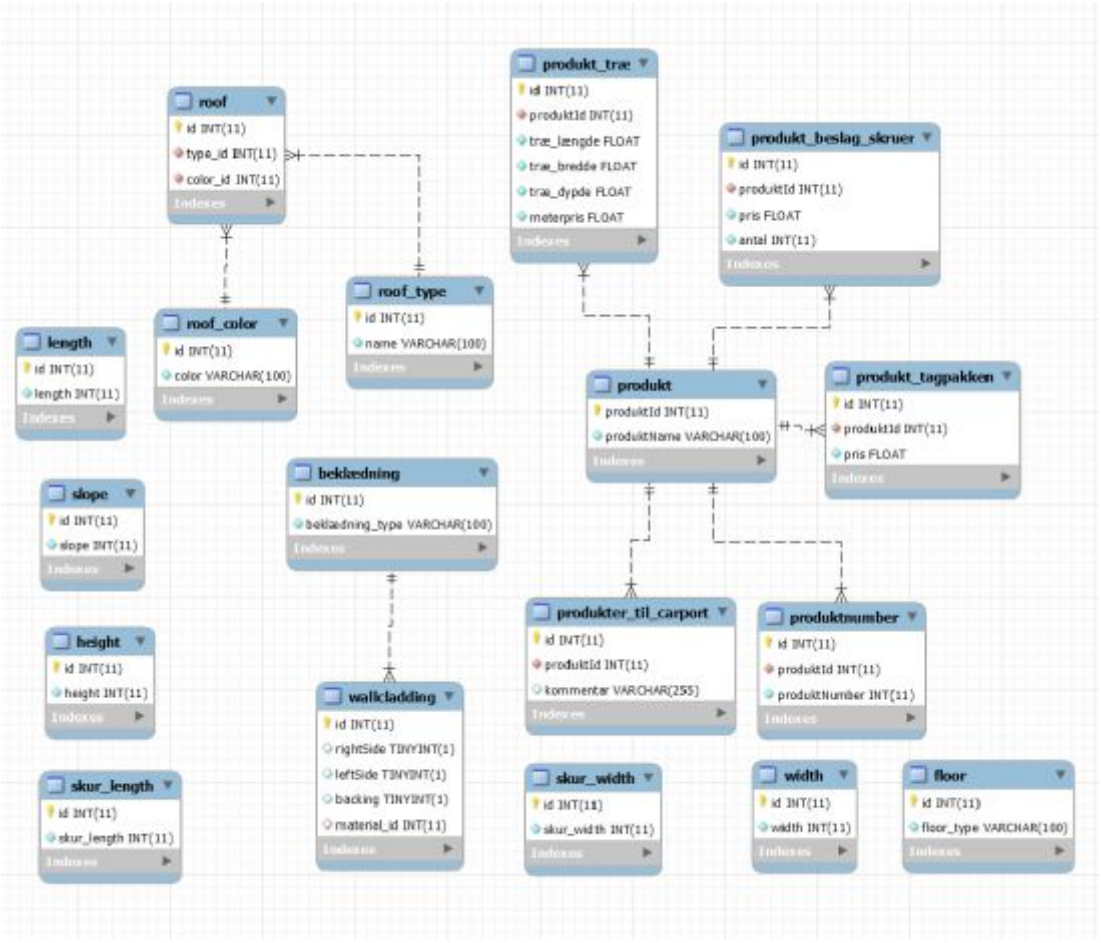
User Story	Beskrivelse
US-1	Admin kan ændre i alle databaserne fra en admin side(S) ● Lav dropdown med tabeller ● Opdaterer tabeller ● Fjerne værdier ● Tilføj nye værdier ● Test funktionalitet ● Marcus er en hipster
US-2	Som sælger, vil jeg kunne hente en materialeliste(L) ● Opsætte classer for forskellige dele af carporten ● Kombinere disse klasser for at få en materialeliste ● Kan sende en mail til sælger med materialeliste
US-3	kunden kan se 2D Plantegning af skuret som opdateres når man ændrer på variablerne. (L) ● Opret SVG-template ● Hente valgte værdier fra dropdown ● Indsæt valgte værdier i drawing class
US-4	Som kunde vil jeg gerne have en bedre oplevelse når jeg besøger hjemmesiden.(L) ● Lave mockup ● HTML ● JSP

Domænemodel



Domænemodellen over Johannes Fog carport bygger, har tre aktører: bruger, sælger og administrator. Brugere kan på siden selv lave deres egen carport som er er delt op i to dele, carport og skur. Under carport skal brugeren tage stilling til nogle valg til sin carport: Højde, bredde, længde, hvorvidt der ønskes beklædning, type af tag: fladt eller med rejsning, hvis kunden ønsker tag med rejsning: hældning og tagsten. Når kunden laver sin carport kan kunden også tilvælge skur til carporten. Herunder kan kunden vælge længde, om skuret skal have en hel eller halv bredde af carportens bredde, beklædning og gulvtype. Hertil kan kunden bestille sin carport. Denne bestilling bliver herefter sendt videre til en sælger som kan verificere carporten samt lave ændringer i pris hvis der skal gives et tilbud. Den sidste aktør, administratoren kan ændre i alle databasens forhold, herunder ændre på trævarers meterpriser og opdatere lagerbeholdning eller ændre i materialer hvis der kommer nye på lager.

ER-Diagram

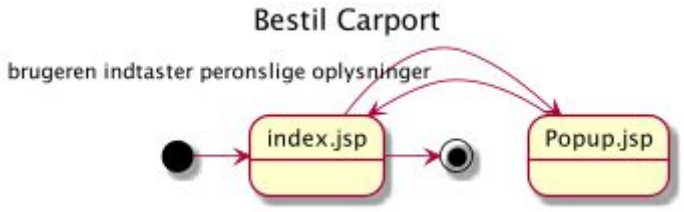


Databasen er oprettet efter det 3. normaliseringsprincip idet alle tabeller har relationer til hinanden i form af primary og foreign keys. Databasens opbygning er blevet bygget på baggrund af product owners ønsker. Derfor er der ingen tabeller til at holde på færdige carporte, da product owner ikke ønsker at kunne gemme carportene i databasen. På grund af disse ønsker fra product owner er der mange tabeller som ikke har relationer til hinanden, som: length, slope, height, skur-length, width, skur_width og floor. Alle disse tabeller bruges som variabler på index siden.
I forhold til relationer har tabellerne roof og wallcladding 1-1 relation. I tabellen roof er relationen lavet da en enkelt tagsten kan kun være af en type eller farve. I wallcaldding er det samme, der kan kun være en type af beklædning på en carport.
I tabellen produkter_til_carport har alle tabellerne en 1-* relation. Dette er gjort fordi tabellen indeholder de oplysninger, som skal sendes med videre til at genere en

materialeliste til den færdigbyggede carport. Som bekendt kan en carport indeholde flere brædder som har den samme længde, bredde og dybde derfor relationen. Det samme gælder for produkt_beslag_skruer og produkt_tagpakken. Alle disse tabeller som har med produkterne at gøre er gjort således for, igen, at bibeholde den 3. noramaliseringsform.

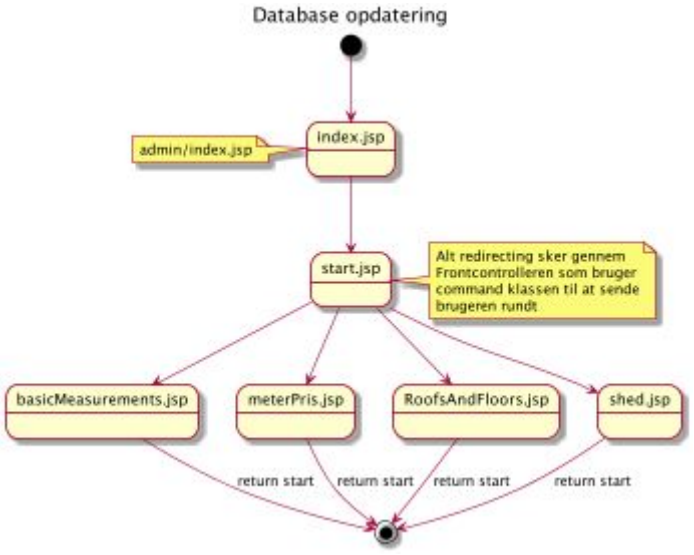
Navigationsdiagram

Quickbygger siden



Quickbygger siden er meget enkel. Den er bygget op omkring en enkelt side som skal implementeres i Johannes Fogs nuværende system. Derfor er der kun 2 jsp sider på quickbyggeren. Indexsiden sender ved hjælp af Frontcontrolleren, brugere videre til en popup side, når brugeren klikker på submit knappen for at bestille en carport. Når brugeren har indtastet personlige oplysninger på poopupvinduet, sendes brugeren tilbage til index siden.

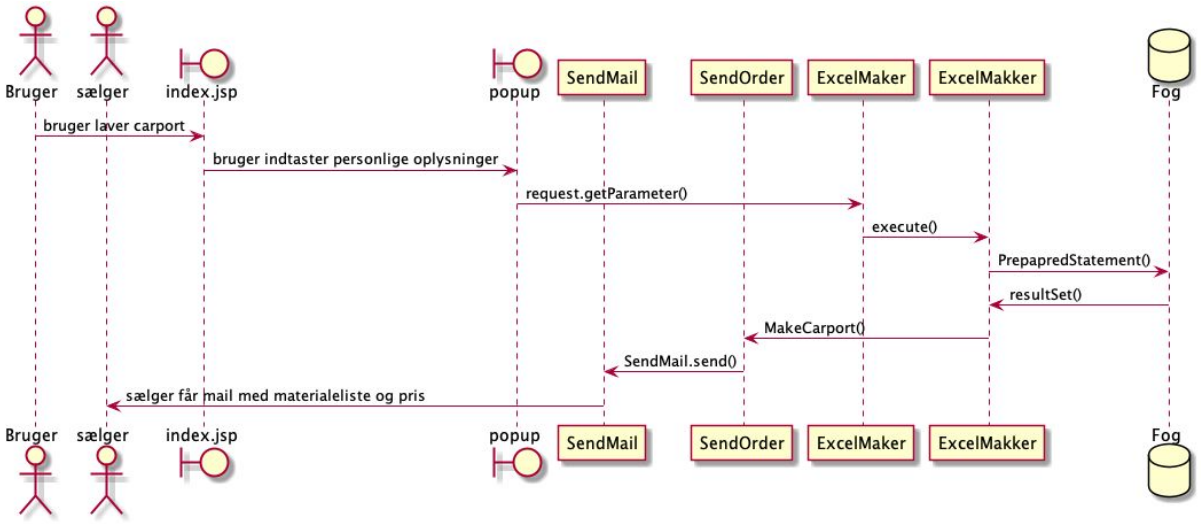
Administrator side



Administrator siden har flere sider. Administrator siden kan ligger under Fog/admin i sin egen undermappe med index side. Siden sender automatisk brugeren videre til start.jsp ved hjælp af Frontcontrolleren med commands adminSide. På start.jsp bliver brugeren præsenteret for en række valg til at ændre forskellige tabeller i databasen. Hertil kan brugeren gå videre til en af de forskellige undersider ved brug af submit knapperne som alle har en target value fra command klassen som sender brugeren videre til de respektive undersider.

Sekvensdiagrammer

Materialeliste og mail



Brugeren indtaster indtaster deres mål og tilvalg på deres custom carport. Når kunden trykker på send knappen i bunden, sender indexsiden en pop up frem på indexsiden ved hjælp af af en if sætning:

```
<c:if test="${sessionScope.send.equals('send')}">
```

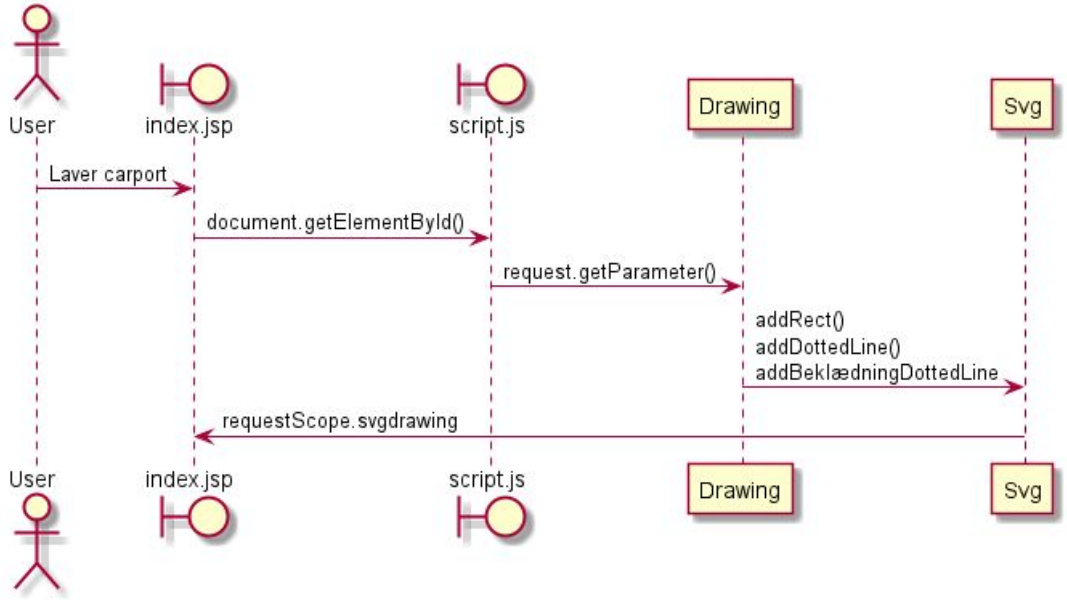
if sætning bliver aktiveret når kunden trykker på send, hvor kunden bliver bedt om at indtaste deres personlige oplysninger. Herefter går systemet ind i FrontExcelMaker klassen, hvor den laver en `request.getParameter()` for at få trukket variablerne ud fra index siden og popup vinduet. Klassen kalder hertil klassen ExcelMaker og laver en carport. Denne carport bliver sendt videre til ExcelMakker klassen hvor materialelisten bliver genereret ved hjælp af java Collection

ExcelMaker. I denne klasse bliver der brugt if statements til at sikre at der ikke bliver brugt varenumre som ikke skal bruges.

```
WoodFromDB wood = MaterialsListFunc.getAllWoodInfo((int)
entry.getKey());
     if ((int) entry.getValue() != 0) {
          Row row = sheet.createRow(i);
```

Denne blok sikrer at der ikke bliver indsat træ materialer som ikke skal bruges. Alle materialer fra tabellerne i Databasen bliver trukket ud og sat som entries i et HashMap hvor value er mængden der skal bruges og key er varenummeret. Linjen går igennem HashMappet og tjekker alle entries og fjerne alle de entries som har en value af 0, hvilket betyder at det ikke bliver brugt.
Hertil laver den så en row i excel arket med alle de materialer som bliver ligger i HashMappet. Alle rows bliver herefter lagt i en celle i excel arket med `Row.createCell` med de korresponderende i værdier. Den færdige materialeliste bliver sendt videre til SendMail klassen som sender en mail med den færdige carport til en sælger mail.

SVG-tegning



På indexsiden bliver der tegnet en SVG som automatisk bliver opdateret i takt med at kunden ændrer på de værdier som ligger i menuen. Sekvensdiagrammet beskriver hvordan brugeren input forplanter sig gennem projektet når de vælger en carport med skur. Brugeren starter med at vælge sin carport. Imens i Scrips.js trækker funktionen `draw`, variablerne ud fra indexsiden med denne funktion og sætter dem som variabler

```
var widthElement = document.getElementById("width");
var width = widthElement.options[widthElement.selectedIndex].text;
document.getElementById('senderWidth').value = width;
```

Denne snippet sætter variablen width, dog er der også funktioner som sætter `length, height, og shedlength` samt variabler til om skuret er halv eller hel længde. Disse variabler bliver gemt i session og sendt videre til Drawing klassen og gemt i variabler:

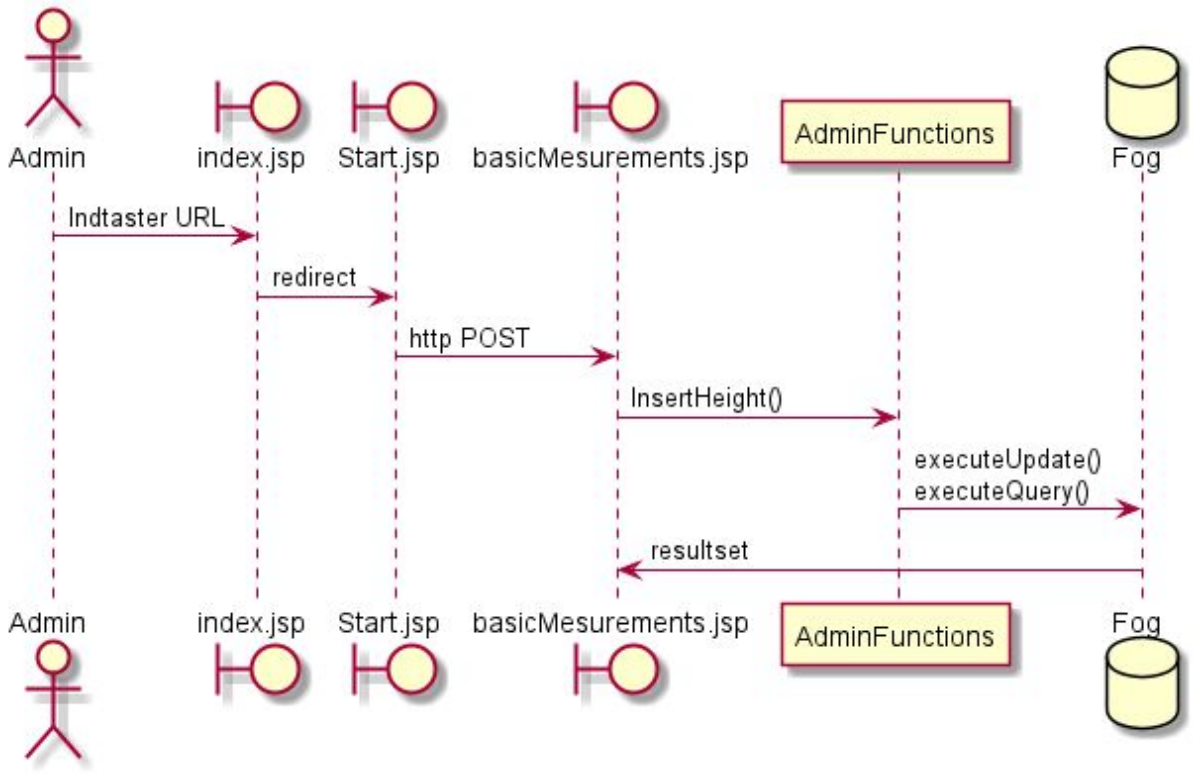
`int width = Integer.parseInt(request.getParameter("senderWidth"));` og ligesom før bliver alle andre variabler også gemt med `request.getParameter()`. Klassen kalder hertil `addRect(), addDottedLine()` og `addBeklædningDottedLine()` fra SVG klassen. Hertil tjekker klassen hvilken type carport der skal tegnes med en if sætning.

```
     if (shedCheck){
          if (shedCheckHalf) {
               //skur med halv bredde
} else if (shedCheckWhole) {
               //skur med hel bredde
} else {
               //carport uden skur
```

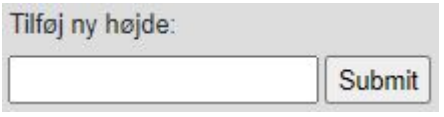
Her tjekkes først om der er valgt at der skal være skur og om skuret skal være af halv bredde. Hvis disse begger er true, tegner den en tegning med halv bredde. Hvis shedCheckHalf er false og shedCheckWhole er true så tegnes der en SVG med hel bredde og til sidst hvis shedCheck er false så tegnes der SVG uden skur. Herefter sættes tegning som attribut i session med `request.setAttribute("svgdrawing", svg.toString());`

og sendt til index siden og printet med `${requestScope.svgdrawing}`.

Admin



Admin indtaster URL til admin siden, og bliver redirected til Start.jsp hvor admin herfra kan vælge hvor i databasen der skal redigeres. Admin vælger den rette kategori som i dette tilfælde er de basale længder, og via FrontControlleren bliver ført videre til basicMeasurements. basicMeasurements giver nu muligheden for at tilføje eller fjerne værdier fra databasen. I dette tilfælde er der valgt at tilføje en ny højde. Højden der skal tilføjes bliver indtastet, submit kalder metoden InsertHeight() i AdminFunctions klassen og dette statement:



String SQL = **"INSERT INTO height (height) VALUES(?)"**; bliver bliver kørt i databasen med executeUpdate(). Admin kan med det samme tjekke om den nye værdi er korrekt tilføjet.

Særlige Forhold

Vi har på siden valgt at bruge localstorage til at gemme de valg som brugeren vælger i carport byggeren. Vi har valgt at gemme i localstorage da dette gør det muligt for brugeren at lukke siden ned uden at miste de informationer de har inde tastet. Localstorage har ikke en udløbsdato det vil sige at det lige meget om brugeren kommer tilbage om 1 dag, en uge eller 1 år alt information vil stadig være gemt. Dette gælder dog kun så længe det er i samme browser og selvfølig på samme computer som der hvor brugere har indtastet det. Vi bruger javaScript til at gemme og læse dataen. Da det er nemt at sætte html elements værdier igennem javaScript. For at gemme dataen har vi methoden vist nedenfor.

```
function saveToStorage(name, val) {
    localStorage.setItem(name, val);
}
```

Her gives der bare et navn på det der skal gemmes og den værdi der skal gemmes og så gemmer den, det element. Eksemplet herunder viser hvordan dataen hentes ind igen.

```
function load() {
    document.getElementById("height").selectedIndex =
localStorage.getItem("height");
```

Dette er kun den første linje af funktionen load, men de næste linjer gør det samme som vist her blot for andre værdier og elementer. Det denne linje gør er at hente en værdig ned fra localstorage med det navn som er givet i parameteren. Denne bliver så givet til det passende html element.

Som det kan ses i sekvensdiagrammet for plantegninger kan det ses at de bliver tegnet ved SVG tegninger. Dette afsnit vil gå dybere i hvordan det fungerer. SVG bliver lavet som punkter og streger i et omvendt koordinatsystem, derfor har alle linjer x og y koordinater. i Svg klassen bliver der lavet templates til de forskellige typer af linjer og firkanter der skal genereres på siden.

```
private final String headerTemplate = "<svg version=\"1.1\"
xmlns=\"http://www.w3.org/2000/svg\"
xmlns:xlink=\"http://www.w3.org/1999/xlink\" height=\"%s\"
width=\"%s\" viewBox=\"%s\"
preserveAspectRatio=\"xMinYMin\">";
```

Den første header der bliver lavet er templaten hvori tegningen skal tegnes, derfor har den kun en height og width. SVG tegninger er en XML fil type og skal refereres med den xml link der ses ovenfor.

```
private final String rectTemplate = "<rect x=\"%f\" y=\"%f\" height=\"%f\"
width=\"%f\" style=\"stroke:#000000; fill: #ffffff\" />";
```

Denne template bliver brugt til at tegne firkanter i SVG tegningen, derfor har den x og y værdi.

```
private final String lineTemplate = "<line x1=\"%d\" y1=\"%d\"
x2=\"%d\" y2=\"%d\" style=\"stroke:#000000; fill: #ffffff\"
/>";
```

Linjen over for bliver brugt til at lave linjer, derfor har de to x og y værdier, en til en startposition og slutposition. Der er også blevet lavet linjer til at bygge spær og beklædning af skur. Disse to linjer bruger begge samme template som `lineTemplate` med har under style henholdsvis `stroke-dasharray` og `stroke-width`. I Svg klassen laves der 3 metoder som alle tage udgangspunkt i svg objektet. De tre metoder:

```
public void addDottedLine(int x1, int y1, int x2, int y2){
   svg.append(String.format(Locale.US, dottedLineTemplate, x1, y1, x2, y2));

public void addRect(double x, double y, double width, double height){
   svg.append(String.format(Locale.US, rectTemplate, x, y, width, height));

public void addBeklædningDottedLine(int x1, double y1, double width1, double
height1){
svg.append(String.format(Locale.US, beklædningLineTemplate, x1, y1, width1,
height1));
```

som det ses ovenfor bliver metoderne lavet med parametrene fra svg objektet og i linjen nedenfor bliver `StringBuilder` klassen kaldt og svg tegningen bliver tegnet

med `svg.append`, dette er fordi SVG tegninger bliver tegnet i String format. Tegningerne bliver tegnet i Drawing klassen, hvor der bliver givet x og y koordinater hvor de skal tegnes.

Som nævnt i det tidligere afsnit så skal der bruges et objekt af type carport for at der kan laves et excel ark. Det excelMaker klassen gør er at den laver dette carport objekt om til et læseligt format i form af et excel ark. Dette bliver gjort ved brug af et java bibliotek kaldt poi. Poi er et bibliotek der gør det muligt at lave filer i nogle forskellige formater her i blandt xlsx også kendt som en excel fil. Det første vi skal gøre med dette bibliotek er at lave et ark vi kan arbejde det i det gøres således.

```
Sheet sheet = workbook.createSheet("Matrialer");
```

Dette laver et ark i excel kaldt "`Matrialer`" som kan refereres til i java ved navnet sheet. Det første der gøres med dette ark er at sætte bredden på alle kolonner. Dette gøres med dette statement.

```
sheet.setColumnWidth(0, 256 * 24);
```

Denne metode bliver kaldt på det ark vi havde før kaldt sheet. Den første parameter vi giver med er hvilken kolonne vi ønsker at ændre bredden på. Det andet parameter er hvor bred kolonnen skal være. Her svarer et bogstav i normal størrelse til 256 og derfor siges der 256 * 24 for at lave en kolonne der 24 teng bred. Dette har vi gjort for kolonne fra 0 - 4.

Det næste vi gør er at lave titlen på vores kolonner også kaldt header. For at lave headerne skal vi først bestem hvordan de skal se ud. Det gøres med disse statements.

```
Font headerFont = workbook.createFont();
headerFont.setBold(true);
headerFont.setFontHeightInPoints((short) 20);
```

Først laver vi et objekt af typen Font som vi kalder headerFont. Med dette objekt skal vi kun definer de ting vi ønsker at ændre fra default fonten. Det Første vi ændre er at vi sætter teksten til at være fed. Dette gøres ved brug af metoden `setBold()` hvor dens parameter sættes til true. Det næste vi gør er at sætte størrelsen på teksten. Dette gøres med metoden `setFontHeightInPoints()` her sættes størrelsen af

skriften ud fra den højden man angiver. Når fonten er lavet skal skal der laves en CellStyle ud fra den.

```
CellStyle headerCellStyle = workbook.createCellStyle();
headerCellStyle.setFont(headerFont);
```

Dette fungere på samme måde som det gjorde at lave fonten. Først laves objektet og derefter sættes fonten til den font vi lige har lavet. Når celle stilen er blevet lavet kan den gives til en celle. Det gøre som vist her.

```
Row headerRow = sheet.createRow(0);
Cell name = headerRow.createCell(0);
name.setCellValue("Produkt");
name.setCellStyle(headerCellStyle);
```

Det første der gøres her er at der laves her er et objekt af typen Row. Dette objekt definere en række og her bliver det defineret som række 0 altså den øverste. Når der er blevet lavet en række kan der laves en celle i denne række. Dette ses på anden linje ovenfor her laves der en Cell ud fra rækken vi lige har lavet. Denne celle gives der en parameter der angiver hvilken kolonne denne celle skal være i. Så denne celle vil være cell (0,0) altså øverst i venstre hjørne. Næste linje sætter værdien af cellen eller i dette tilfælde texten af celle. Til dette bruges metoden `setCellValue()` hvor den parameter vi giver med er værdien af cellen. Tilsidst bruges metoden `setCellStyle()` til at sætte cellens stil til at være den som vi lavede tidligere altså headerCellStyle. For alle de andre overskriver gentages det samme der bruges dog samme Row og samme CellStyle .

Når alle overskrifterne er lavet skal alt dataen om carporten indsættes. Dette bliver gjort en række af gangen. Alt træet bliver sat ind i excel ark ved brug af nedenstående kode.

```
int i = 1;
for (Wood w : carport.getWoods()) {
   HashMap<Integer, Integer> woods = Match.wood(w);
   for (Map.Entry entry : woods.entrySet()) {
       WoodFromDB wood = MaterialsListFunc.getAllWoodInfo((int) entry.getKey());
       if ((int) entry.getValue() != 0) {
           Row row = sheet.createRow(i);
           row.createCell(0).setCellValue(wood.getName());
```

```
            row.createCell(1).setCellValue((int) entry.getValue());
            row.createCell(2).setCellValue(wood.getLængde());
            row.createCell(3).setCellValue(MaterialsListFunc.
              getDescription(MaterialsListFunc.getAllWoodInfo(
              (int)entry.getKey()).getProduktId()));
            row.createCell(4).setCellValue((int) entry.getKey());
            row.createCell(5).setCellValue(wood.getMeterpris() *
              w.getAmount().getAmount() * (w.getAmount().getLength() / 100));
            i++;
        }
    }
} i++;
```

I det første for loop gennemgås alt træ der er i carport. Det andet for loop går i gennem alle varenummer der passer til det stykke træ. Hvis der bliver brugt noget med det varenummer så skrives varen ind i excel arket. Det bliver skrevet ind på samme måde som overskrifterne. Den eneste forskel er at her skrives der i flere kolonner i sammen række og det bliver ikke brugt nogle stil. Der er 3 nestede loops som dette, det ene er det der er vist ovenfor. Det andet gør det samme bare for tagpakken og den sidste er for beslag og skruer. I forhold til excel arket gør de alle 3 det samme men de kladde nogle forskellige databaser og brugere nogle forskellige klasser til at holde deres variabler.

Vi har lavet en celle der viser den samlet pris af alle delene det er lavet som vist her.

```
Row row = sheet.createRow(i);
Cell cell = row.createCell(2);
cell.setCellFormula("SUM(F2:F100000)");
row.createCell(1).setCellValue("Samlet pris:");
```

Her laves en celle på samme måde som de andre, men i stedet for at sætte værdien af cellen så sættes cellen til at vis resultatet af den formular der er givet som parameter. Til sidst laves filens navn og med dette sættes der hvor filen skal gemmes. Efter dette skrives filen og fil skriveren lukkes. Koden til dette kan ses nedenfor.

```
String fileName = carport.getHeigth() + "," + carport.getWidth() +
"," + carport.getLength() + ".xlsx";
```

```
FileOutputStream fileOut = new FileOutputStream(fileName);
workbook.write(fileOut);
fileOut.close();
Efter filen er gemt skal den sendes til en sælger så her bruger vi
metoden nedenfor.
SendMail.send(fileName, Subject, text);
```

SendMail er en klasse vi har lavet hvor der er en metode der hedder send. Denne metode tager 3 parameter det første er placeringen af filen der skal sendes. Den anden er overskrift der skal være på mailen og den sidste er hvad der skal stå i den. Til at lave denne klasse har vi brugt java biblioteket javaMail, dette biblioteket for simpler at sende en mail fra java.

```
String to = "cphprojekter@gmail.com";
String from = "cphprojekter@gmail.com";
String host = "smtp.gmail.com";

Properties properties = System.getProperties();
properties.put("mail.smtp.host", host);
properties.put("mail.smtp.port", "465");
properties.put("mail.smtp.ssl.enable", "true");
properties.put("mail.smtp.auth", "true");
```

Det første der gøres i klassen er at vi definerer hvilke mail der sender, hvilken der modtager og hvordan den sendes. Dette er det der bliver gjort i de 3 første linjer hvor det bliver gemt i strings. Derefter laves der et objekt af typen Properties som kalds properties. Her i ligges de værdier der skal bruges for at opsætte en mail server. Herefter laves der en session som ses nedenfor.

```
Session session = Session.getInstance(properties, new
javax.mail.Authenticator() {
   protected PasswordAuthentication getPasswordAuthentication() {
       return new PasswordAuthentication(from, "1123581321#");
   }
});
```

Session bliver lavet ud fra properties variable og en klasse kald javax.mail.Authenticator, denne klasse står for at valider kodeordet og mail. Nu kommer den del over vi opbygger mailen, koden til dette kan ses nedenfor.

```
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO, new
InternetAddress(to));
message.setSubject(Subject);
Multipart multipart = new MimeMultipart();
MimeBodyPart attachPart = new MimeBodyPart();

String attachFile = fileName;
DataSource source = new FileDataSource(attachFile);
attachPart.setDataHandler(new DataHandler(source));
attachPart.setFileName(new File(attachFile).getName());

multipart.addBodyPart(attachPart);
MimeBodyPart body = new MimeBodyPart();
body.setText(text);
multipart.addBodyPart(body);
message.setContent(multipart);
Transport.send(message);
```

Først laver vi en besked i denne besked sætter vi hvem den er fra, hvem det er til og hvad overskriften skal være. Efter dette laves der noget kaldt en Multipart, dette er indholdet af vores mail. Først lægges excel filen ind i denne multipart også texten der skal stå i mailen. Denne multipart bliver så tilføjet til beskeden også bliver beskeden sendt.

Status på implementering

Admin Side

På admin siden er der mulighed for at se alt træ i databasen og her kan man også opdatere pris, slette vare og indsætte nyt træ. Det var meningen at der skulle være samme muligheder for de andre varetyper altså for skruer, beslag og tag dele. På grund af tidsmangel er disse ikke alle blevet lavet, men de skulle fungere på samme måde som den der er lavet for træ. Det var også meningen at man fra admin siden skulle kunne ændre i tabellen kaldt “produkter_til_carport”. Her skulle man have været i stand til at ændre kommentarerne til vare og ændre hvilke produkter der skal bruges til de forskellige dele af carporten.

Frontend

På indexsiden bliver der genereret en SVG tegning af carportens plantegning set fra oven. Plantegningen er misvisende i den forstand at den ikke giver en fuldendt plantegning af carporten og skur. Plantegningen mangler præcision i forhold til stolper og spær. På grund af uligheder i tegningen ville den endelige byggede carport ikke være strukturelt ustabil. Dog giver plantegningen er godt billede af hvordan carporten cirka kan se ud. Samtidig viser plantegningen ikke mål, dette er ikke blevet implementeret på grund af tidsmangel.

Samtidig var planen at indexsiden skulle vise en plantegning af carporten set fra siden, men på grund af tidsmangel er denne del ikke blevet lavet.

Materialeliste

Programmet kan lave materialelister i Excel som den kan sende til en sælger. Materialelisten fungerer bedst ved valg af tag med rejsning men kan også lave for carport med fladt tag. Ingen af materiale listerne er perfekte da vi ikke har den tilstrækkelig byggefaglighed til at kunne lave regler til materialelisten helt korrekt. Derudover er der valgmuligheder på siden som materialelisten ikke tager højde for.

Dette er om der skal beklædning på carporten og hvilke tagtype der er valgt. Hvis der vælges fladt tag vil materialisten bruge tagplader og hvis det med rejsning vil den vælge tagsten. Den tager heller ikke højde for hvis man har valgt en anden farve tag.

Mail

Programmet er istand til at send en email til en sælgere. I denne e-mail vedhæftes der et excel ark med materialelisten i og her står også de informationer som brugere har indtastet om sige selv. Her kunne det godt være opsat pænere og på en måde der giver bedre mening. Derudover tager det ret lang tid for programmet at send mail og på den måde som det er opsat på følges det som om hjemmesiden hakker. Det kunne have været gjort på en måde hvor brugeren ikke oplevet denne forsinkelse da den alligevel er på serversiden. Der skal hertil siges at vor program ikke er i stand til at sende en email fra digitalocean server da de har blokeret port 25 som er en af dem der bruges til at sende mails. Dette er gjort af digital oceanien for at forhindre spam bots på deres server. Vi har desværre først opdaget dette lige før afleveringsfristen så det er ikke blevet fikset.

Mangler

Der er nogle ting som burde have været lavet, men som vi aldrig kom til. Den største af disse er nok sikkerhed, alle der ved at der er en admin side kan komme ind på den. Der skulle nok have været en login side så alle og enhver der gener url ikke kan komme ind og ændre i databasen. Det næste der skulle have været lavet er exception. Dette ville have gjort det nemmer at se hvilke fejl der sker når programmet er kommet ud på server. Så længe vi køre i intellij kan vi se fejlene i console men når det ligger på server er der ingen nem måde at se hvilke fejl der sker. Det er hovedsageligt i vores database kald hvor der er ting der kan gå galt og derfor også der vi skulle have haft exception. Der skulle også have været input validering på brugerinput, men dette er heller ikke lavet så brugere kan indtaste værdigere som programmet ikke kan håndterer.

Test

Materialeliste

Der er blevet benytte unit test til de klasser i materialelisten som er mere avanceret. Dette er gjort da det gjorde det muligt hele tiden at kunne test om klassen eller metoden giver det som det meningen den skal gøre. Derudover går det også at det er muligt at tjekke resultatet uden at skulle benytte klassen andre steder i koden. For materialelisten betyder det at det ikke var nødvendigt at kunne lave et excel ark før alle klasser der skulle bruges for at lave en carport var lavet.

Excel/ email

For klassen der laver excel arket og den der sender e-mail er der også brugt unit test, men her er det brugt på en lidt anden måde. Her bruges assertEquals methoden ikke af ander grunden end at det gøre så fejl der sker i testen vil blive printet i konsollen. Hovedårsagen til at disse klasser er i tests er dog at det gjorde det muligt at generere et excel ark eller sende en email uden at det skulle integreres i jsp siden. Dette gjorder at det var nemmer og hurtiger at tjekke om klasserne virkede og gjorde så man ved at klassen virke inden man integrerer det i jsp. Hvis man prøver at lave klassen og indsætte den i jsp på samme tid kan det være langt værre at vide hvor fejlen sker og der forhøjere den tid der skulle bruges på at lave det.

Arbejdsprocess faktuelt

Projektet blev lavet via udviklingsmetoden “Scrum”. Ved hjælp af scrum blev der afholdt review og planning møder i løbet af projektets arbejdsperiode, disse møder blev primært afholdt to gange om ugen, et møde til hvert formål.

Review-møderne foregik således at der blev fremvist nye elementer af projektet, som der var blevet aftalt skulle nås til det review-møde. Hvis en user-story ikke var

blevet nået, så ville den blive trukket tilbage til en liste af ikke gennemførte user-stories. Hvis user-storien derimod var blev nået, så ville den blive rykket over i en liste med andre færdige elementer af projektet.

Hvert planning-møde foregik således at der ville blive præsenteret en række user-stories til project owneren, som så ville fortælle hvilke user-stories som der var relevante at arbejde på frem imod næste review-møde. Her var der specielt fokus på at man hverken valgte for mange eller for få user-stories at arbejde med.

Det meste af arbejdet blev lavet ved diverse daglige møder som primært blev afholdt om formiddagen. Her ville der blive opdateret om status på projektet, samt hvad diverse medlemmer af gruppen arbejdede på så der ikke blev arbejdet på det samme. Hvis et element af systemet var besværligt, så ville flere medlemmer af gruppen sidde sammen og lave det, men ellers blev arbejdet primært lavet hver for sig.

Under selve projektet var der ikke nogle retrospective møder. Der blev ikke set nogen årsag til at kigge bagud fremfor fremad, så fokuset forblev på projektets mangler.

Det første scrum-møde blev afholdt den 16. april for gennemgå hvad der skulle være passende user-stories fremadrettet. 17. april var der møde med project owner Tue Hellstern og der blev der aftalt hvilke user-stories som var relevante at arbejde på til det næste review-møde. Alle user-stories fik enten et "S", et "M", eller et "L" tildelt ud fra hvor tidskrævende man antog den user-story var. Disse bogstaver står for "small", "medium", eller "large". I nogle tilfælde kunne en user-story også få et "X", som står for "extra", bagpå deres originale bogstav for at vise at denne user-story enten ville være markant nemmere eller sværere end først antaget. Diverse user-stories blev brudt ned til tasks, men blev primært klaret uden et decideret fokus på at løse målet ved hjælp af de tasks.

De første user-stories som der blev arbejdet på var følgende:

User Story	**Beskrivelse**

US-1	Som kunde vil jeg gerne kunne vælge mine carport værdier på en html side(M)
US-2	Jeg som kunde vil gerne kunne vælge højde, længde og bredde på min carport. (L)
US-3	Som kunde vil jeg gerne kunne vælge om taget på min carport skal være med fladt tag eller med rejsning. (S)
US-4	Som kunde vil jeg gerne kunne vælge om hvorvidt jeg ønsker skur med til min carport. (S)
US-5	Som kunde vil jeg gerne kunne vælge længde og bredde på mit skur, såfremt jeg ønsker skur til min carport. (S)
US-6	Som kunde skal jeg ikke kunne vælge et skur med en bredde der er større end carporten, da dette ikke kan produceres.(M)

Logikken bag valget af disse user-stories var at man antog efter en samtale med project owneren, at de ikke ville kræve for meget tid at lave, men dog ville være essentielle for projektet fremadrettet, da det ville være smart at have et fundament før man begyndte at tilføje mere kompliceret ting til projektet. I slutningen af denne sprint var forsiden til vores hjemmeside skabt, og der var også blevet lavet basale funktioner til en potentiel kunde.

Det næste review-møde fandt sted d. 23 april. Her blev der gennemgået om alle user-stories var blev nået og om man havde forstået meningen med scrum. Alle de aftalte user-stories var blevet nået, og den næste sprint kunne blive aftalt. D. 24 april blev der afholdt planning-møde.

Med fokus på mødet med project owneren og med det som der allerede var blevet bygget, blev der valgt følgende user-stories:

US-7	Som kunde vil jeg gerne kunne vælge beklædning på mit skur, såfremt jeg ønsker skur(S)

US-8	Som kunde vil jeg gerne kunne vælge gulvtype på mit skur, såfremt jeg ønsker skur(S)
US-9	Som kunde vil jeg gerne kunne vælge hvilken farve og materiale mit tag med rejsning skal være. (L)
US-10	Som kunde vil jeg gerne kunne gemme mine valg, så jeg kan se mine valg igen.(M)
US-11	Som administrator skal jeg kunne ændre på beklædningstyperne og gulvtyper.(M)
US-12	Som kunde vil jeg gerne kunne vælge beklædning til carport, hvis det ønskes(S)
US-13	Som kunde vil jeg gerne have en bedre oplevelse når jeg besøger hjemmesiden.(L)
US-14	Lav en side til admin, så han/hun kan redigere i udvalget af værdier(S)
US-15	Admin kan ændre i alle databaserne fra en admin side(S)

Efter at have skabt fundamentet for projektet i den første sprint, bragte denne sprint den administrative del af projektet og gav også den potentielle kunde flere optioner at vælge imellem angående deres carport og skur.

Det næste review møde blev afholdt d. 30 april. Der skete ikke synderligt meget ved dette møde, dog blev der snakket om status på projektet. Alle user-stories var blevet nået.

Dagen efter, d. 1 maj, blev der afholdt planning-møde. Ved mødet der stod det klart at det ikke ville være muligt at nå de resterende user-stories på denne sprint, så der blev planlagt at man i den sidste tid under projektet ville arbejde ihærdigt på de sidste user-stories, og så have statusopdatering hver uge. Efter mødet var følgende user-stories udvalgt til denne arbejdsprocess:

US-16	Som sælger vil jeg gerne have at administrator siden er meget overskuelig og pæn(M)
US-17	Som administrator vil jeg gerne kunne ændre på meterpriser på tømmer der

	ligger i databasen. (M)
US-19	Som administrator vil jeg gerne kunne ændre på meterpriser på tømmer der ligger i databasen. (M)
US-20	Som sælger, vil jeg kunne hente en materialeliste til en nybygget carport(XL)
US-21	Kunden kan se 2D Plantegning af skuret som opdateres når man ændrer på variablerne. (XL)

Der var yderligere to møder planlagt, et d. 7 maj og et d. 18 maj, men ingen af disse møder blev afholdt på den traditionelle måde. Mødet d. 7 maj var et meget kort og fungerende blot som en statusopdatering på de udvalgte user-stories.
Der blev kort snakket om tekniske problemer angående "SVG", men derudover bragte mødet ikke meget nyt til arbejdsprocessen.

Det sidste møde foregik d. 18 maj. Dette møde var et review-møde som skulle klargøre om projektet var færdigt eller ej, og hvilke småfejl som skulle rettes.
Alle user-stories var blevet nået, men visse ting skulle stadig justeres. Det var primært 2D plantegningen og materiale listen som skabte de største problemer.

Arbejdsprocessen Reflekteret

Igennem forløbet var der ikke en konkret scrummaster i hvert sprint, hvis ikke gruppen selv tog fat i en opgave blev opgavefordelingen pålagt, ikke nødvendigvis af den samme person hver gang. Dette fungerede fint men en mere struktureret arbejdsfordeling havde muligvis resulteret i endnu mere produktivitet.

Der har ikke som sådan været problemer med at få brudt user stories ned i tasks, men dels var der forskel på de userstories der skulle brydes ned. De længere og mere komplicerede user stories fik en lang række tasks, som har givet bedre overblik over hvordan den skulle løses. De kortere om mere simple user stories kunne ikke brydes ned i særlig mange tasks og det gav derfor ikke mening at gøre det.

Estimering på hver user story har ikke været spot on, da det på forhånd har været svært at forudse hvad det krævede at komme igennem de enkelte user story. De

fleste user stories blev fornuftigt estimeret men enkelte blev ramt en smule ved siden af. I disse tilfælde blev der når ugen var omme revurderet et tidsestimat ud fra hvor langt man var kommet med arbejdet.

Det var generelt svært at finde en god rytme under forløbet.
Grundet manglen på undervisning og dermed en struktureret hverdag, var det svært i perioder at engagere sig ordentligt i projektet.
Originalt var det planen at mødes hver dag om formiddagen for at arbejde på projektet, men der var en generel mangel på motivation til at få lavet projektet ordentligt, så i stedet for så blev det mere til meget korte møder hvor man internt ville fortælle hvad man havde tænkt sig at arbejde på den dag, for så at gå hvert til sit efterfølgende.

Da man så så opgaveformuleringen for første gang var der med det samme en generel enighed om at den mindede meget om en tidligere opgave, hvor i man skulle lave en hjemmeside som solgte cupcakes, og ikke carporte.
Dette gjorde at man havde svært ved at tage opgaven seriøst, da man i starten tænkte at man næsten bare kunne kigge på hvad man havde gjort i den første opgave, og så næsten kopiere det direkte over.
Dermed så kom der en generel følelse af at hvis man bare gjort det absolut minimum hver dag, at så ville man nå slutningen med et færdigt projekt på eller andet tidspunkt.
Dog fandt man ud af som forløbet skred frem at dette ikke var tilfældet.
Manglen på den strukturerede hverdag var stadig markant, men det fremkom lige så stille at der var elementer i denne opgaver så ikke var ens med den forrige cupcake-opgave.
Specielt kreationen af materialelisten og 2D-plantegningerne endte med at tage meget længere tid end først planlagt, da der var mangel på erfaringhed indefor SVG samt hvad der kræver at bygge en carport.
Man kunne klart mærke under forløbet at man tit følte sig uvidende om hvad som var den bedste og mest effektive løsning.

Her er dermed snakke om mangel på erfaring i forhold til hvordan man opbygger et projekt uden en klar motivation.

Der har generelt været et savn for ordentlig vejledning under projektet.

Planning og review møderne skulle imitere hvordan et rigtig forløb havde været med en fiktiv product owner, men dette bragte ikke den hjælp som der ellers var brug for under forløbet.

På trods af der ikke altid var den største motivation til at arbejde på projektet, og på trods af det nogle gange stod uklart hvor man var henne i projektet, så var det altid meget klart hvad man i teorien skulle gøre for at løse et problem.
Hele gruppen formåede også at have en god attitude under hele projektet.
Der var mange ting som var frustrerende og ikke virkede som planlagt, men der var altid en følelse af at man stod sammen i gruppen om hvad end problemet kunne være.

Alt i alt var det dermed svært at arbejde på projektet under dette forløb, da der var en mangel på en normal hverdag, og dermed ikke meget motivation til at lave arbejdet. Der var en klar mangel på erfaring om at arbejde under de forhold og det kunne mærkes.
På trods af diverse forhindringer, så blev opgaven stadig klaret, takket været teamwork og tålmodighed.
Til næste projekt vil der være mere fokus på lade være med at prokrastinere og på at blive bedre til at arbejde sammen på diverse opgaver fremfor hver for sig.