

Fog carports

Gruppe H

Natasja Vitoft, cph-nn194@cpbbusiness.dk, github: NatasjaVitoft

Oliver Ravnkilde, cph-op82@cphbusiness.dk , github: Ravnkilde1995

Start: 28/11/2022

Aflevering: 04/01/2023

Links	4
Indledning	4
Baggrund	4
User stories	5
Process	7
Første iteration:	7
Anden iteration:	8
Tredje iteration:	9
Fjerde iteration:	9
Femte iteration:	10
Git	11
Kanban	11
Kommunikation	12
Virksomheden	12
SWOT analyse	12
VPC analyse	13
Virksomheden Johannes Fog	13
Bæredygtighed	14
Refleksion	15
System	16
Kravspecifikationer	16
Usecases	17
Usecase kunde/medarbejder	17
Usecase system	17
Figma mockup	19
Arkitektur	21
MVC arkitektur	21
Page controller	21
Data: mapper & facade	21
Test	22
Modeller & diagrammer	24
Domæne model	24
Navigationsdiagram	25
Database	26
Normalisering	26
Beskrivelse af databasen	26
Generelt	27
ERR Diagram	27
Særlige forhold	28
Exceptions	28
Sessions	29
Application Scope:	29
Session Scope:	29

Request Scope:	29
SVG tegning	29
Opbygning af stykliste	32
Calculator og calculatorShed	32
CalculatorList	33
Sikkerhed og validering af brugerinput	33
Errors	33
SQL injection	36
Implementering	37
Userstories	37
CRUD	38
Generelt	38
SVG tegningen	39

Links

- Link til Github projekt : [Github](#)
- Link til Figma mockup : [Figma](#)
- Link til Trello: [Trello](#)
- Link til video pitch: [Pitch](#)
- Link til demo video: [Demo](#)

Indledning

Som opgave skal vi udvikle en dynamisk web applikation for Fog trælast- og byggecenter. Vi skal udvikle, og til dels teste dele af et system, som imødekommer Fog's krav. De overordnede krav for opgaven er opsummeret således:

- Det som skal udvikles (produktet)
- Hvordan vi skal samarbejde (processen)
- Hvad der skal afleveres eller demonstreres (dokumentation)

Vi benytter os af Java til udviklingen af backend systemet, samt Java servlets til at kommunikere fra frontend til backend. I frontend gør vi brug af HTML 5, CSS og bootstrap til at designe og gøre hjemmesiden responsiv.

Vi vil i rapporten starte ud med en kort baggrund om Fog, og deres system hvor kunderne skal kunne sammensætte deres egen carport. Herefter vil vi opridse vores userstories, og komme nærmere ind på, hvordan vi har valgt at arbejde med opgaven under proces afsnittet. Efterfølgende vil vi fokusere på Fog som virksomhed og vores egen gruppe som en virksomhed under forretnings afsnittet. Her vil vi komme ind på Fogs forretningsmodel, og hvordan de kan optimere deres system. I anden del af opgaven under system afsnittet, vil vi gå i detaljer med hvordan vi har valgt at kode opgaven, forklare hvad vi har gjort, og demonstrerer dele af koden.

Baggrund

Vi har fået en opgave fra virksomheden Johannes Fog, som ønsker at fornye deres system, hvor det er muligt for deres kunder at sammensætte deres egen carport ud fra ønskede mål. Johannes Fog er en virksomhed, som har byggecentre mange steder i Danmark. Fog sælger primært materialer til forskellige slags byggeprojekter. En service som Fog udbyder på nuværende tidspunkt, er bl.a. at deres kunder, skal kunne sammensætte deres egen carport og herefter modtage en stykliste, skitse og

vejledning til at sætte carporten op. Fog vil dog gerne have optimeret denne del af deres system, så det fungerer bedre. Det skal være muligt for kunden, at indtaste mål på carporten og vælge om der eventuelt skal et skur med. Det skal også være muligt for kunden at se en tegning over carporten, hvilket kunden både skal have mulighed for før og efter køb af carporten.

Som systemet er nu, kan kunden indtaste mål på carporten og et eventuelt skur. Herefter bliver informationen sendt videre til en medarbejder hos Fog, som skal udarbejde et tilbud til kunden gennem et system, som kun medarbejderne har adgang til. Når en medarbejder har godkendt dette, bliver en stykliste og skitse af carporten sendt videre til kunden. Systemet er fra 1999, så det er efterhånden ved at være ret gammelt. Det er bl.a. ikke muligt for medarbejderne hos Fog at tilføje nye materiale eller ændre i de gamle materialer i databasen til systemet. Det er muligt for medarbejderne at ændre prisen på materialerne, hvis der benyttes et bestemt login. På nuværende tidspunkt har kunden også kun mulighed for at vælge mellem to slags tage, selvom der i dag er flere valgmuligheder.

User stories

	Userstories	Acceptkriterier
1	Som kunde ønsker jeg at kunne oprette mig som bruger og logge ind, for at kunne bestille en carport.	Når jeg klikker på opret eller login i navigationsbaren, skal jeg blive sendt videre til enten en formular, hvor jeg kan oprette mig som bruger, eller en formular hvor jeg kan indtaste mit brugernavn og password for at logge ind.
2	Som kunde ønsker jeg at kunne sammensætte min egen carport med specifikke mål.	Når jeg klikker på carport i navigationsbaren, skal jeg blive sendt videre til en formular, hvor jeg kan indtaste specifikke mål for en carport med eller uden skur.
3	Som kunde ønsker jeg at kunne bestille min sammensatte carport og få muligheden for at se tegningerne inden køb.	Når jeg har indtastet de ønskede mål inde på formularen, skal jeg kunne trykke på knappen "bestil" og blive sendt videre til en ny formular, hvor

		det er muligt at klikke på en knap, hvor jeg kan se tegningerne til carporten.
4	Som kunde ønsker jeg at kunne tilgå en stykliste efter køb af carporten.	Når jeg har købt min carport ved at trykke på “køb” knappen, skal jeg blive sendt videre til “min side”, hvor det skal være muligt se en oversigt over mine ordre. Herefter kan jeg klikke på knappen “stykliste”, som fører mig videre til en ny side, hvor styklisten bliver vist.
5	Som kunde ønsker jeg en “min side” hvor jeg kan se alle mine bestillinger.	Som kunde ønsker jeg altid at kunne tilgå mine ordre, når jeg er logget ind, ved at trykke på “min side” i navigationsbaren og hermed få vist en oversigt over alle mine ordre.
6	Som kunde skal jeg kunne ændre i min personlige oplysninger, så der f.eks. ikke står den forkerte adresse, hvis delene til carporten skal fragtes.	Som kunde skal jeg kunne ændre i mine personlige oplysninger ved at klikke på “min side” i navigationsbaren, når jeg er logget ind.
7	Som administrator kan jeg se alle ordre i systemet med tilhørende stykliste og tegninger.	Som administrator skal jeg kunne logge ind med et bestemt login og tilgå en side, hvor alle ordre i systemet er printet ud.
8	Som administrator kan jeg slette ordre i systemet, for at holde styr på gamle og nye ordre.	Som administrator skal jeg kunne se en oversigt over alle ordre i systemet, og have muligheden for at trykke på “slet” knappen som er ud fra hver ordre.

Process

Vi har valgt at dele processen op i fire afsnit, som vi kalder iterationer. I afsnittet nedenfor vil vi gennemgå de forskellige iterationer. Vi har valgt at dele processen op i fire iterationer, da det også var sådan vi arbejdede undervejs igennem projektet. Vi har opdelt det således, at det var nemmere for os at sætte os nogle mål for hvornår forskellige dele af opgaven skulle være færdig og samtidig opsummere hvor langt vi var efter hver iteration. Strukturen er sat op således, at vi sætter en hypotese op om hvad vi kan forvente at nå i første del af processen, samt en periode for hvornår vi forventer at iterationen er færdig. Når vi når de slutdatoen af hver iteration gennemgår vi hvad vi faktisk har nået at få lavet og hvilke komplikationer der eventuelt opstod undervejs.

Første iteration:

Dato: 28/11 - 4/12

Forventet arbejdstimer på denne uge: 30 timer

- SWOT model
- VPC model
- Usecases
- Domæne model
- User stories
- Opsætning af trello kanban
- Video pitch

Inden vi starter med at kode skal vi have lave vores forretningsmodeller og danne os et overblik over hvilke klasser vi eventuelt skal bruge til projektet ved at lave en domænemodel. I vores forretnings del har vi valgt at tage udgangspunkt i SWOT modellen for at beskrive vores gruppes strategiske styrke/svagheder mm, så det bliver nemmere at facilitere et godt gruppearbejde. Vi tager ydermere også udgangspunkt i Value Proposition Canvas (VPC), hvor vi identificere Fogs styrker og svagheder, for at finde frem til hvordan deres system kan forberedes. De beskrevet modeller vil vi bruge til at forme vores userstories.

Afrunding første iteration:

Vi har fået skabt et fint overblik over opgaven samt fog som forretning. Vi har opsat et trello kanban, hvor vi kan administrere gruppens arbejdsopgaver over de næste par uger. I vores trello kanban har vi opsat labels med hvor mange timer vi forventer de forskellige arbejdsopgaver kommer til at tage. De

forskellige labels er opdelt i farver, så vi hurtigt kan danne os et overblik over hvor tidskrævende opgaven bliver. Vi havde lidt problemer med vores VPC model da den var lidt for vag ift. at fokusere på Fog som forretning og ikke bare fokusere på carport delen. Efter en samtale med vores vejleder har vi herefter fået rettet til, så den giver et bedre overblik over fog som virksomhed. Vi har aftalt at udskyde video pitch til et senere tidspunkt, da modellerne nok bliver opdateret undervejs.

Anden iteration:

5/12-11/12

Forventede arbejdstimer: 35 timer

- ERR Diagram
- Opdatere vores kanban med nye arbejdsopgaver
- Opsætning af database
- Opsætning af startkode
- Opsætning af brugerside
- Opsætning af et login system
- Opsætning af en opret bruger funktion
- Opsætte en algoritme der kan beregne carport ud fra brugerinput

I denne uge vil vi beskæftige os med kode. Vores fokuspunkt kommer til at ligge på at lave en algoritme, som kan beregne en carport ud fra brugerinputs. Det skal muligt for brugeren at vælge specifikke mål til sin carport. Dertil skal vi have en database der er tilsvarende vores ERR diagram som vi har lavet med relevante relationer. Herudover skal vi have lavet et login system, som brugeren kan logge på. Det skal også være muligt for brugeren at oprette sig i systemet. Her benytter vi os af CRUD operationerne til at komme de oprettede objekter ned i database, hvor de herefter gemmes i databaselaget.

Afrunding anden iteration:

Under anden iteration havde vi lidt problemer med at forstå den udleveret plantegning fra Fog, så vi havde lidt svært ved at lave beregningerne, da vi ikke rigtig kunne finde ud af hvad vi egentlig skulle beregne på. Så vi besluttede at lade den ligge til vi har fået os et bedre overblik over styklisten. Vi fik sat en fungerende startkode op med en database der er tilsvarende vores ERR diagram med relevante relationer. Det lykkedes os også at lave et velfungerende login system og opret funktion med tilhørende mappers og styling til jsp siderne. De færdiggjorte arbejdsopgaver er blevet opdateret i vores kanban

Tredje iteration:

12/12-18/12

Forventede arbejdstimer: 40 timer

- Odatere vores kanban med nye arbejdsopgaver
- Opstart på frontend
- Kalkulationer
- Test
- SVG tegning

I denne uge vil vi fortsat beskæftige os med at forstå tegningerne og styklisten fra Fog af carporten. Vi vil bruge længere tid på at undersøge tegningen og styklisten, så vi er sikre på at vi får de rigtige materiale mål med, så vi kan begynde på vores beregningsmotor. Når vi har fået dannet os et overblik over beregningerne, kan vi bruge JUnit til at teste vores resultater, for at se om de stemmer overens med den udleverede stykliste. Vi skal i denne uge også kigge på hvordan SVG fungerer ved at se vejledningerne på moodle igennem. Vi er også blevet enige om i gruppen, at det ville være rart at få en del frontend på plads, så kan vi altid style de sider vi laver senere.

Afrunding tredje iteration:

Vi er begyndt på beregningerne og testet de vigtigste af metoderne. Der er dog nogle af beregningerne som ikke er helt korrekte. Nu har vi dog en beregning vi kan arbejde videre med, så vi til sidst kan få genereret en stykliste til brugeren efter køb af en carport. Vi har også fået lavet en masse frontend arbejde heriblandt: Forside, Login, Opret, Produkt, Min side og Bestil. Vi har også brugt et par timer på at style siderne, således at brugeren får en bedre oplevelse når der navigeres rundt på siderne. Siderne er også responsive, så hvis brugeren ønsker at surfe på et mindre device, så er det også muligt. Vi er desværre løbet ind i lidt problemer med at generere en SVG tegning. Vi har lavet metoder som skulle generere henholdsvis; en streg, spær og pæle, men på nuværende tidspunkt virker det ikke. Vi har brug for mere tid til at fikse problemet, så vi tager det med i næste uge.

Fjerde iteration:

19/12-25/12

Forventede arbejdstimer: 30 timer

- Odatere arbejdsprocess på kanban
- Debugge SVG Tegningen
- Lave en stykliste der bliver gemt i session

- Debugge på de hardcoded værdier i databasen samt finde en reel pris til de forskellige materialer
- Opsætning af begrænsning hvis man ikke er logged in
- Opsætning af begrænsning så man ikke kan oprette en bruger med et brugernavn der allerede eksistere i databasen.

I denne uge skal det handle om at debugge og samle alle de løse tråde, så vi faktisk har en prototype der kan generere en carport ud fra brugerens inputs, Vi skal også sætte nogle kriterier op for hvad brugeren må kunne på siden, hvis brugeren fx ikke er logget ind. Det skal heller ikke være muligt for brugeren at samle en hvilken som helst carport, så vi bliver nødt til at give brugeren en dropdown menu med udvalgte værdier, som brugeren så herefter kan vælge mellem. Til sidst skal vi have styr på, hvorfor at vi ikke kan tegne carporten på jsp siden, når vi i backenden har kørt tests og kan se at den faktisk tegner det.

Afrundet fjerde iteration:

19/12-25/12

Vi har i denne periode kigget på debugging af SVG tegningen, og vi fandt ud af at vi manglede en knap i frontenden, der skulle aktivere tegningen. Så det er blevet sat på nu og vi får printet en pæn tegning ud fra brugerens mål.

Ydermere så har vi fået kigget på fejlhåndtering ift jsp siderne. En bruger modtager nu en fejlbesked i frontend laget, hvis brugeren fx opretter en bruger, med et allerede eksisterende username. Der er blevet sat hardcoded værdier ind i en dropdown menu på bestillingssiden, så brugeren har realistiske mål at vælge ud fra. På bestillingssiden bliver brugeren også redirected til loginsiden med en fejlbesked, når der trykkes “køb carport”, og brugeren ikke er logget ind.

I vores prototype er der plads til optimering på dele af systemet, men vi er desværre løbet tør for tid, og har været nødt til at prioritere. Vi har valgt at dedikere et afsnit i rapporten der skal dække, vores optimering af prototypen.

Femte iteration:

25/12-4/1

Forventede arbejdstimer: 45 timer

- Opdatere arbejdsprocess på kanban
- Rapportskrivning
- Demo video
- Pitch Video

Dette er den afsluttende fase af projektet. Vi skal lave en video demo af vores færdige prototype og en pitch video af vores forretningsmodel VPC, herudover skal vi også samle alle vores noter og skrive rapporten færdig. Så vi kan nå at få afleveret opgaven.

Afrundet femte iteration:

25/12-1/4

Vi har i denne periode primært fokuseret på at få skrevet vores rapport, og følge den vedlagte rapport skabelon, som vi fik udleveret af vores undervisere. Den måde vi har grebet rapport delen an på, var at vi havde en masse noter skrevet ned i et dokument, hvor vi har omskrevet de noter og fyldt mere på i rapporten. Vi har valgt ikke at følge rapport skabelonen helt til punkt og prikke, da vi mente, at vi så ville gentage os selv lidt for meget. Vi har valgt at prioritere afsnit som process, funktionalitet og forretning, da vi mener at det er vigtigt at forstå, hvordan vi har grebet opgaven an.

Ydermere har vi optaget en pitch, hvor vi fokusere på forretningsdelen af Fog som virksomhed. I videoen gennemgår vi nogle af de punkter, hvor Fog kan optimere deres forretningsmodel. Vi har bl.a. kigget på bæredygtighed og brugeroplevelsen.

I opgaven fik vi også et krav om at vi skulle optage en demo video af hjemmesiden. Denne video skulle indeholde en kort præsentation af hjemmesiden og dens funktionaliteter.

Git

Vi har i opgaven benyttet os af git, mest i terminalen. Den måde vi valgte at bruge git på var at benytte os af forskellige branches, heriblandt Test, styling, error, svg for at nævne et par stykker. Ved at bruge branches og arbejde i branches gør processen meget nemmere, når vi committer og pusher på forskellige tidspunkter, så vi undgår konflikter når der arbejdes på forskellige enheder (Computere).

I gruppen har vi aftalt at der bliver informeret når et push rammer skyen, så kan vi merge branchen sammen med main, og der kan laves et pull på main fra en anden enhed, så koden er up to date, herefter fortsættes processen.

Kanban

For at få et overblik over hvilke arbejdsprocesser der igangsat eller ikke er startet, har vi benyttet os af et Trello board. I vores Trello board har vi struktureret de forskellige iterationer og sat en tidsramme på hvad vi forventede blev færdigt. Vores Trello board har også et farvekodesystem, i form af labels der indikere en tidshorisont, for hvor tung/lette de forskellige opgaver er. I Trello boardet fremgår også arbejdsprocesser fra før vi begyndte at kode, heriblandt Design/Business fasen. Hvor vores primære fokus var på at lave modeller/diagrammer.

Vi har ikke valgt at have en Kanban master, da vi kun er en to mands gruppe. Men vi er mødtes dagligt til en sparring om eventuelle problemer der var opstået dagen forinde, og prøvede at løse dem i fællesskab. Det var også typisk her vi opdaterer vores kanban, og fik os et overblik over hvad dagens opgaver var.

Kommunikation

I gruppen har vi primært mødtes fysisk på skolen, for at sidde og arbejdet koncentreret sammen. Men vi har også været nødsaget til at mødes online i perioder, her har vi benyttet os af discord og messenger. Vi har primært brugt Discord til at snakke sammen i en tale-kanal, men vi brugte også Discord til at dele informationer, såsom links, kode osv. De dage hvor det ikke har været muligt at mødes på Discord eller fysisk, har vi benyttet os af Facebook Messenger, for at fortælle hvad vi arbejdede på, evt opståede problemer, som vi skal være opmærksomme på eller hvis vi har pushet til skyen.

Virksomheden

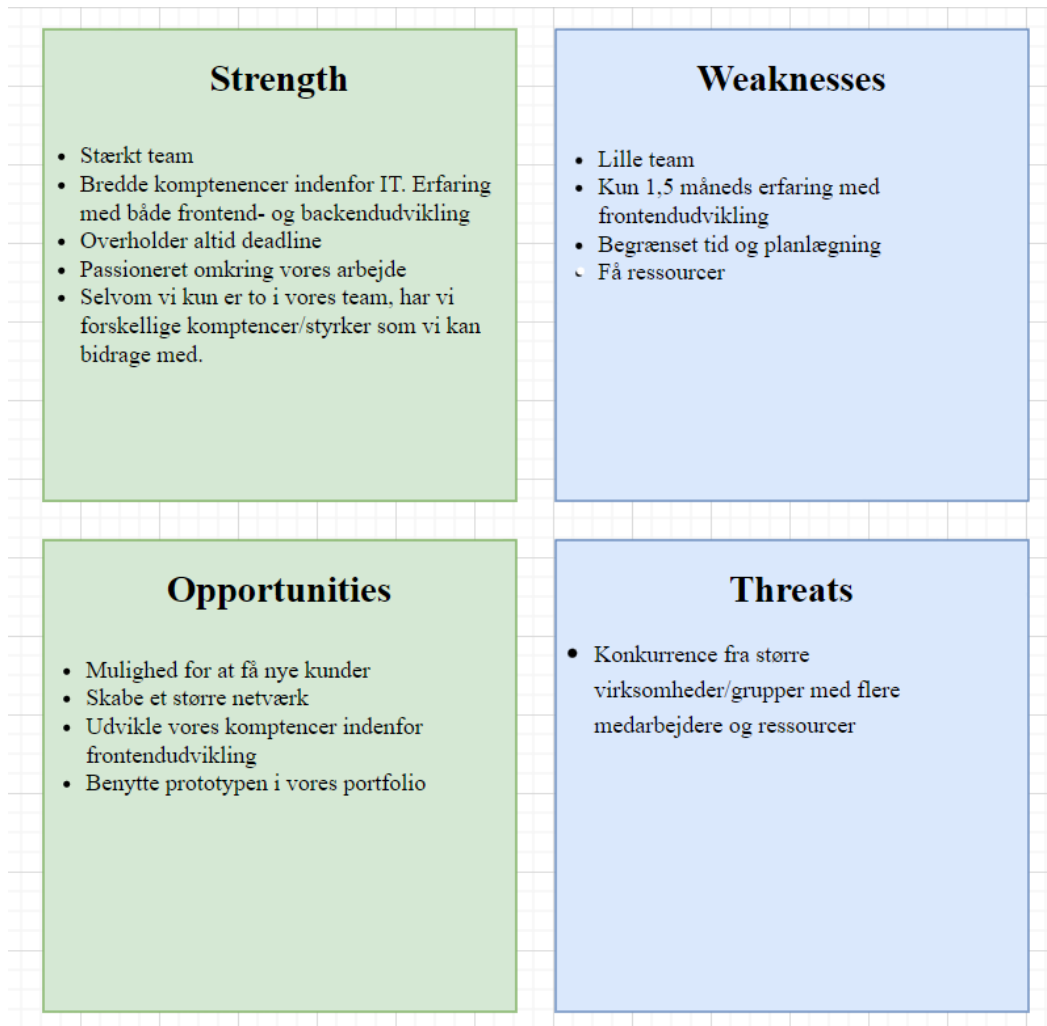
SWOT analyse

Vi har valgt at lave en SWOT analyse ud fra vores gruppe, for at danne os et overblik over vores styrker, svagheder og strategiske position. I analysen antager vi at vores gruppe, er en lille to mands virksomhed, som har fået en større opgave fra virksomheden Johannes Fog.

Under styrker har vi den fordel, at det er vores første større projekt, så derfor er vi ekstra passioneret omkring vores arbejde, da det er vigtigt for os, at vi fremover kan få flere kunder og nye opgaver. Vi har også en bred forståelse indenfor programmering, da vi både har erfaring med frontend – og backend udvikling. Vi har dog kun 1,5 måneds erfaring med frontend udvikling, hvilket kan blive en udfordring ift., hvor hurtigt vi arbejder. Dog har vi begge i gruppen hver vores styrker, som vi kan bidrage med. Dvs. at vi er gode til forskellige ting, hvor den ene måske er god til noget, den anden ikke er særlig god til.

Vores svaghed er, at vi er en mindre gruppe med få ressourcer, så derfor kan det blive en udfordring at nå alt det, som vi gerne vil med projektet. Ift. udefrakommende trusler kommer vi tilbage til det her med, at vi er en mindre gruppe. En reel trussel er netop, at der findes mange større virksomheder/grupper, som er i besiddelse af flere ressourcer, og potentielt kan udkonkurrere os. Mulighederne for os er dog, at hvis projektet bliver en succes, og at vores kunde er tilfreds, så kan vi

skabe et større netværk og muligvis få flere nye opgaver i fremtiden og udvide virksomheden/gruppen.



VPC analyse

Virksomheden Johannes Fog

Vi har valgt at lave en VPC analyse af Johannes Fog som virksomhed, for at identificere hvilken værdi deres produkter og service giver deres kunder, og hvor de eventuelt kan optimere deres forretningsmodel.

Som det ses i nedenstående model, er kundens primære job at købe Fogs produkter/Carporte, administrere deres økonomi og tænke mere bæredygtigt. Det kunden får ud af at handle hos Fog er, at det er muligt at få sammensat en carport med specielle mål, som passer perfekt til kundens behov og ønsker. En ulempe hos kunden er dog, at på nuværende tidspunkt virker systemet hvor man kan sammensætte sin egen carport ikke optimalt. Kunden får ikke særlig mange valgmuligheder, når der

f.eks. skal vælge et tag til sin carport, selvom der burde være det, da der er flere tage i Fog's sortiment. Desuden er det svært at navigere rundt på hjemmesiden, hvor det for kunden kan være tidskrævende og besværligt at finde frem til siden, hvor man sammensætter sin egen carport. På nuværende tidspunkt skal man klikke sig ind på en færdigdesignet carport, gå ned under dokumenter & links, og herefter klikke sig ind på formularen, hvor man indtaster målene på den carport man ønsker at bestille. En måde hvor Fogh gør det lidt nemmere for kunden, er ved at have en god kundeservice. Der er altid hjælp at hente i en af Fogs butikker, og desuden har Fogs hjemmeside en chatfunktion, hvor det er muligt at skrive til en medarbejder for at få hjælp, hvis man ikke er fysisk til stede i en butik. Denne del er en pain relief for kunden ift., at det er svært at navigere rundt på hjemmesiden. Det mest optimale havde dog været, hvis Fog gør hjemmesiden nemmere at navigere rundt i. Det ville både være en fordel for kunderne, som får en bedre oplevelse når de skal sammensætte deres egen carport, men det ville også være en fordel for Fog som virksomhed, da de ikke ville skulle bruge lige så mange ressourcer på at have medarbejdere klar til at hjælpe i chatfunktionen.

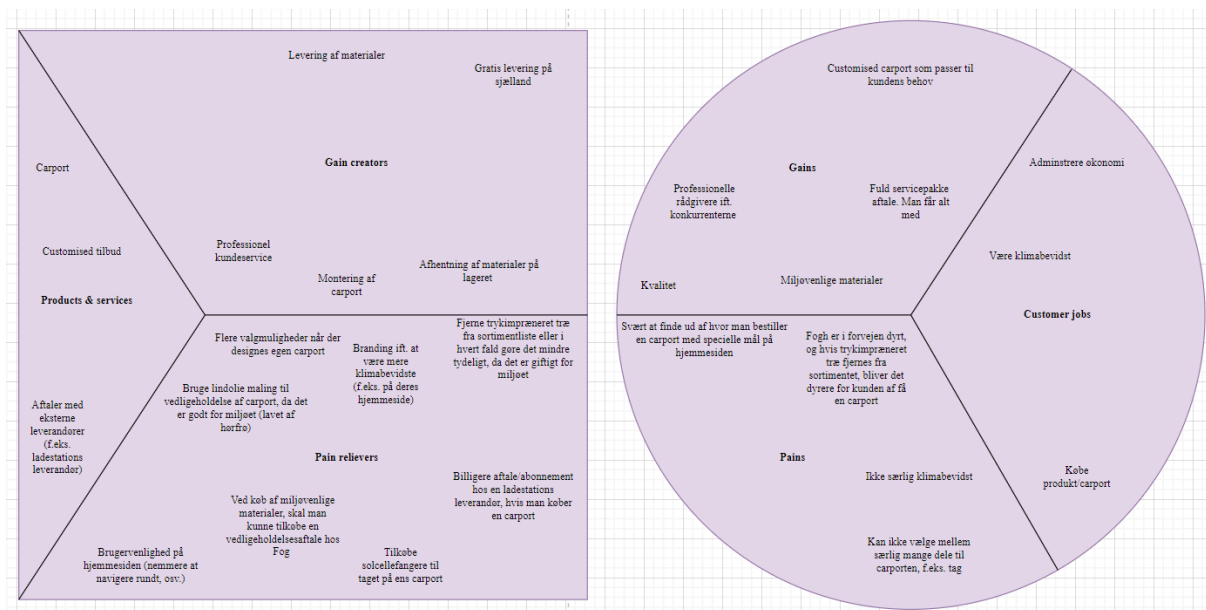
Under gain creators, som er det der hjælper produkterne med at skabe værdi for kunden, tilbyder Fogh både levering af de materialer som kunden bestiller, og derudover kan kunden også selv vælge at afhente det på et lager. Når kunden har bestilt en carport, vil medarbejderne i Fogh sørge for at alle dele kunden skal bruge for at bygge carporten, er samlet når delenene til carporten afhentes. Dette gør det nemt og overskueligt for kunden, som ikke skal bruge tid på selv at finde ud af hvilke dele, de skal bruge til deres customised carport. Desuden får kunden også gratis fragt og levering med, hvis varen leveres et sted på sjælland, så det ikke er en ekstra omkostning for kunden, men i stedet en ekstra service som Fogh udbyder.

Ydermere kan man sige, at det er en pain for kunderne at Fog er lidt dyrere end mange andre byggemarkederne. Dog kan man sige, at det er et gain for kunderne, at det er professionelle rådgivere i Fog som står til rådighed, og de har oftest en uddannelse eller grundig oplæring indenfor området ift. de ansatte i mange andre byggemarkeder.

Bæredygtighed

Vi har valgt i vores VPC analyse for Johannes Fog at sætte en smule fokus på bæredygtighed også, som en strategi for at optimere deres forretningsudvikling. Det er i dag blevet vigtigt for mange mennesker at tænke og handle mere bæredygtig, hvilket også er derfor, at vi har sat det ind som en del af customer jobs. En af Fogs pains er bl.a. at de ikke benytter særlig mange klimabevidste materialer (som f.eks. trykimp træ). Det som Fog bl.a. kan gøre for at være mere klimabevidste, er at benytte materialer som ikke er lige så miljøskadelige. Det kunne være en ide bl.a. at bruge linoliemaling til

vedligeholdelse af carportene. Derudover kunne Fog også begynde at samarbejde med andre virksomhederne om at kunne tilbyde kunderne at sætte solcelle fangere på carport tagene og købe en ladestation til Elbiler med, når der sammensættes egen carport. Det ville både vise at Fog er klimabevidste, men det ville også være nemmere for kunderne, da de kan få det hele samlet et sted, i stedet for at skulle gå ud og købe det forskellige steder.



Refleksion

Vi har valgt at inddrage en VPC analyse af Fog som virksomhed, for at gøre det nemmere for os at finde ud af hvad vi skulle kode, og hvor fokus skulle være. I vores VPC analyse kom vi frem til at Fog godt kunne gøre mere ud af brugervenligheden, når der bestilles en carport. Hjemmesiden som den er nu, kan godt virke forvirrende og uoverskuelig for kunden. Derfor har vi også valgt at have fokus på at hjemmesiden, skal være nemmere at navigere rundt i, i vores program. Vi har f.eks. valgt at have en knap i navigationsbaren på hjemmesiden, hvor man kan klikke sig direkte ind for at sammensætte carporten. I vores VPC analyse har vi også fokuseret en smule på bæredygtighed, hvilket har givet os nogle ideer til, hvordan man kunne videreudvikle på systemet, hvis vi havde tid. Man kunne f.eks. implementere en form for CO2 beregner, når der sammensættes en carport, så kunderne kan vælge den mest bæredygtige løsning. Ved at lave en SWOT analyse har det hjulpet os til at strukturere vores gruppearbejde og finde ud af hvad vi kunne forvente at nå at få implementeret. Vores svaghed i gruppen er, at vi ikke er så mange, så derfor har vi hele tiden været bevidst om, at vi nok ikke kunne nå alle de userstories, som vi gerne ville.

System

Kravspecifikationer

I projektet har vi brugt en række forskellige teknologier og værktøjer. Nogle var på forhånd bestemt af opgavestillerne og andre har vi valgt selv. Herunder er en oversigt, som er ment som hjælp til andre udviklere, hvis de eventuelt skal overtage projektet.

Editor/IDE:

- IntelliJ IDEA (Version 2020.2.4)

Sprog/Udviklingsværktøjer:

- Java
- Openjdk-17
- MYSQL Workbench (Version 8.0)
- MYSQL Server (Version 8.0)
- HTML 5
- CSS 3.0
- Twitter Bootstrap 5.0

Webserver:

- Apache Tomcat (Version 9.0.67)

Planlægning og rapportskrivning:

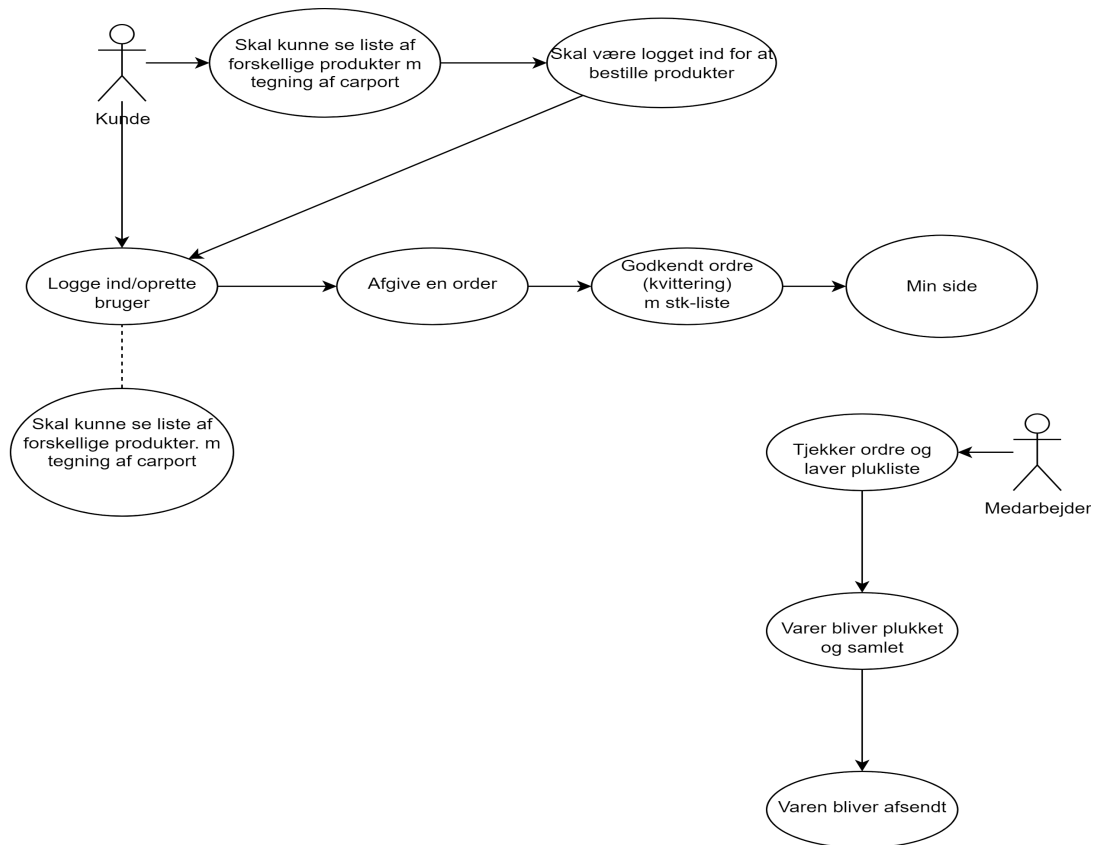
- Google Docs
- Trello - Til projektstyring
- Draw.io og PlantUML - Til diagrammer
- Github - Til versionering og deling af kode
- Figma - Til design af mockups

Usecases

Usercase kunde/medarbejder

Som kunde skal det være muligt at kigge på hjemmesiden med begrænsninger, man kan logge ind for at få fuld udnyttelse af hjemmesiden. Som kunde som ikke er oprettet i systemet kan vi kun se på

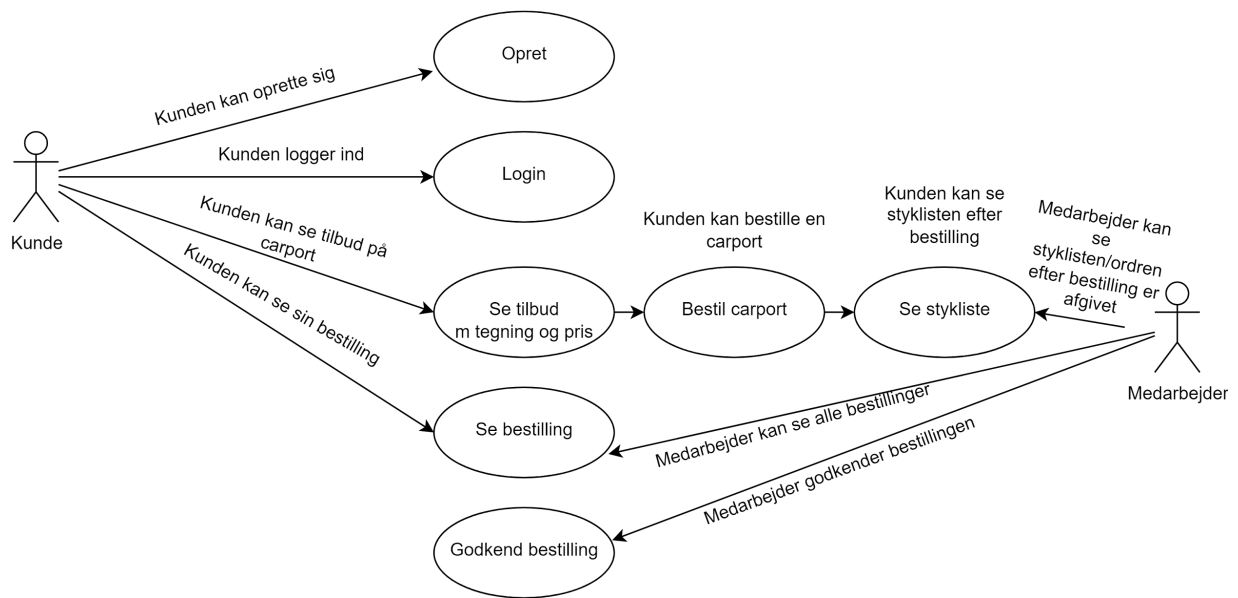
produkter og skitser. Når kunden har oprettet sig i systemet, får kunden adgang til at kunne bestille carporte - hertil følger en stykliste med til ordren så kunden kan se en oversigt over materialer. Denne liste skal kunne tilgås fra min side. Medarbejderen modtager ordren, og laver herefter en plukliste som bliver sendt til lageret. herefter plukkes de forskellige dele til ordren, hvor det hele samles og sendes til kunden.



Usecase system

System skal kunne administrere de forskellige brugere i systemet, samt sætte begrænsninger for hvad man kan se, hvis man f.eks. ikke er logget ind. Hvis man har oprettet en bruger/logget ind, får man mulighed for at bestille en carport, og modtage en stykliste for hvad der skal bruges af materialer for at sætte en carporten op.

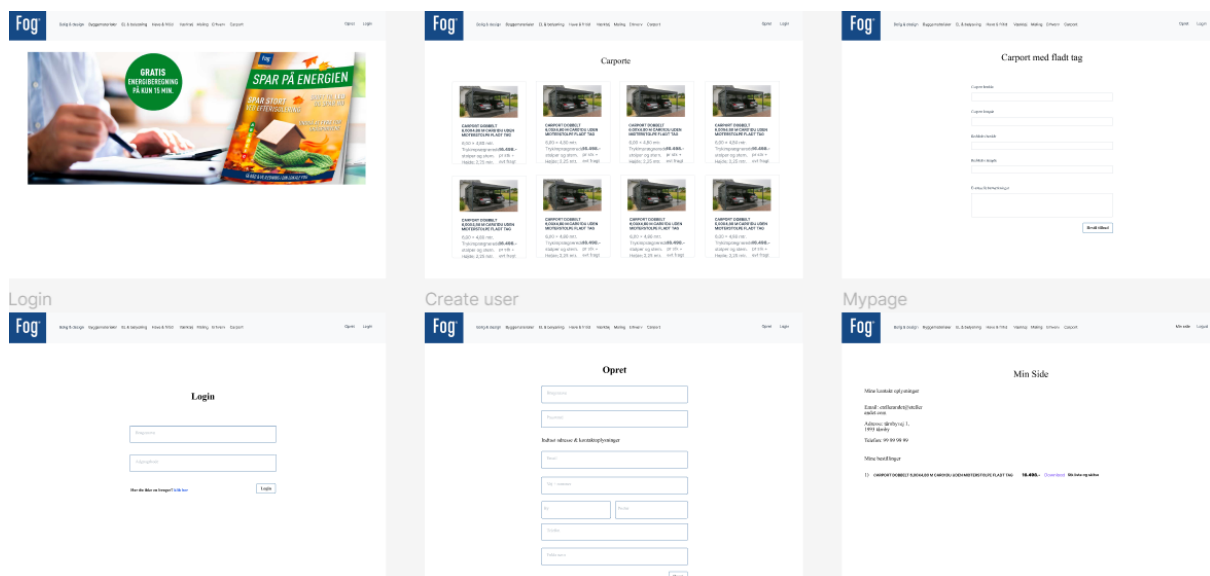
Systemet skal også kunne beregne hvor mange forskellige materialer der skal bruges, vha en algoritme, så kunden bare kan skrive værdier ind i et input felt via hjemmesiden, og så beregnes materiale listen når bestillingen er godkendt. Til sidst skal vi modtage en ordre, af alle informationerne i input felterne i en færdig stk liste, som kunden kan tilgå.



Ovenstående usecase diagram viser et billede af hvordan kunden og medarbejderen kan interagere med systemet. I diagrammet ses i alt ni use cases, som relateres til vores userstories. Første usecase er hvor kunden, kan oprette sig som bruger ved at tilgå en formular i navigationsbaren på hjemmeside. Anden usecase er hvor kunden kan logge ind med en eksisterende bruger, ved at klikke på login i navigationsbaren på hjemmesiden. Tredje usecase er hvor man som kunde, kan se et tilbud på en speciel carport sammensat af kunden. Dette gør man ved at klikke på “carport” i navigationsbaren. Fjerde usecase går videre fra tredje usecase. I fjerde usecase skal det være muligt for brugeren, at bestille/købe den sammensatte carport. Femte usecase går videre fra fjerde usecase. i femte usecase skal det være muligt for brugeren at se styklisten og en skitse efter køb af carporten under “min side”. Sjette usecase går videre til femte usecase, hvor det skal være muligt for administratoren at se alle kunders ordre og tilhørende stykliste og skitse. I syvende usecase skal kunden se alle sine bestillinger/ordre, ved at klikke på “min side” i navigationsbaren efter at være logget ind. Inde under min side vil kunden kunne se en liste af alle de ordre, som kunden har gennemført. På samme side kan kunden også se sin stykliste og SVG denne vej igennem.

Figma mockup

I første del af iterationen havde vi meget fokus på at danne os et overblik over projektet heriblandt at få os et overblik over hvordan vores hjemmeside skulle se ud. Det er klart at når vi kommer længere frem i processen, så kan der ske afvigelser fordi man har løst problemer anderledes end først antaget.



Ovenstående billede viser vores figma mockup. Se evt link i toppen for at få et tydeligere billede.

Da vi kom videre fra første iteration af processen og vores analyse VPC analyse, blev gruppen enige om at Fog's hjemmeside kunne virke uoverskuelig som bruger, da man får vist meget information på hjemmesiden på en gang. Dette ville vi gerne optimere ved at bruge en "karrusel" hvori information kan findes på forsiden. Det ville være nemmere for brugeren at overskue et slideshow af tilbud i karrusellen, og vi har selvfølgelig også gjort plads til statisk information, som kunne være vigtigt for brugeren. Det vi mente kunne være vigtig var at fortælle brugeren, at fog også havde en app, og hvis du som bruger downloadede appen, kunne du få en unik service.



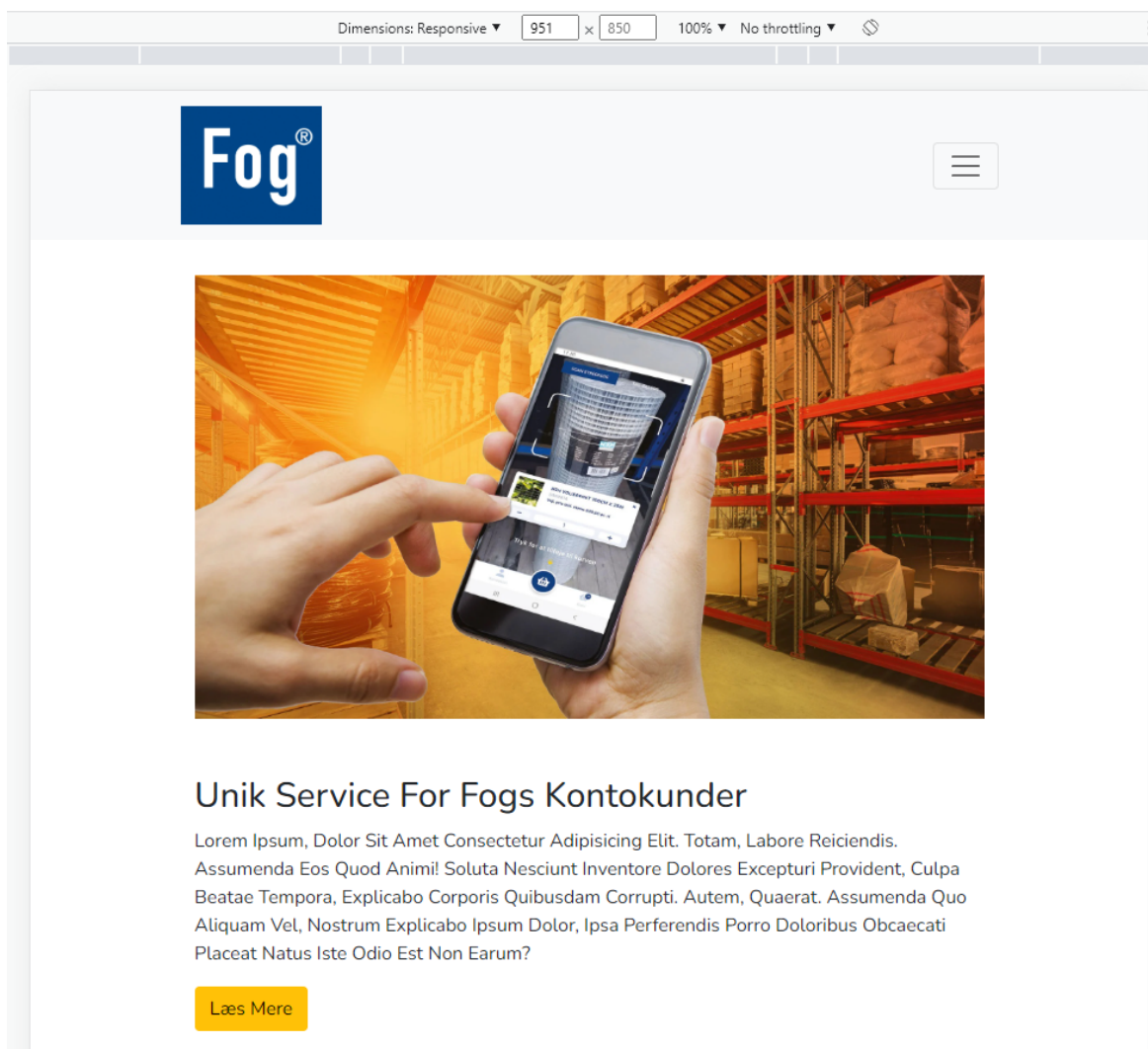
Unik Service For Fogs Kontokunder

Lorem Ipsum, Dolor Sit Amet Consectetur Adipiscing Elit. Totam, Labore Reiciendis. Assumenda Eos Quod Animi! Soluta Nesciunt Inventore Dolores Excepturi Provident, Culpa Beatae Tempora, Explicabo Corporis Quibusdam Corrupti. Autem, Quaeat. Assumenda Quo Aliquam Vel, Nostrum Explicabo Ipsum Dolor, Ipsa Perferendis Porro Doloribus Obcaecati Placeat Natus Iste Odio Est Non Earum?

Læs Mere

Ovenstående billede er fra forsiden desktop view.

I projektet har vi også taget højde for at en bruger kan bruge andre devices, som fx mobiltelefon eller tablet når hjemmesiden besøges. Så vi har arbejdet med responsive design, så tekst, billeder mm skalere efter forhold. billedet er taget fra forsiden. Eksempel på responsive design, mobil view.



Ovenstående viser et billed der er responsive i et skalerings forhold tilsvarende en mobil.

Arkitektur

MVC arkitektur

Vores projekt tager udgangspunkt i en udleverede start kode, som er bygget op efter MVC arkitekturen. MVC står for *model, view & control*. Inde under Java mappen har vi control og model mapperne. Inde under control mappen findes alle vores servlets, som vi har med i vores kode. Control mappen indeholder kun de klasser, som står for interaktionen mellem brugeren og systemet. F.eks. vil alle vores servlets, være der hvor anmodningerne fra brugeren bliver behandlet i systemet. Et eksempel på dette kan være, når en bruger skal oprette et login i systemet. Brugeren indtaster en

række af informationer, som f.eks. navn, på hjemmesiden, hvor det er i createUser servleten at denne information bliver behandlet.

Inde under model mappen har vi fem forskellige undermapper (config, entites, exceptions, persistence og services) som indeholder hele systemets logik og data. Inde i mappen services har vi primært klasser, som vi bruger til at udregne stykliste, tegne SVG og forskellige hjælpefunktioner. I mappen entites har vi forskellige klasser, som vi benytter, når der f.eks. skal gemmes informationer i databasen. F.eks. har vi klassen User inde under mappen entites. I User klassen findes forskellige constructors og getter & settere. Attributterne i User klassen stemmer overens med de felter, som vi har i vores user tabel i databasen. Vi benytter bl.a. denne klasse, når vi skal gemme en bruger i databasen ved at instantiere et objekt af User klassen.

Page controller

Page controlleren er et objekt, som håndterer et request fra en bestemt side eller action på en hjemmeside. En page controller kan være struktureret på flere forskellige måder, men i vores projekt er det i forbindelse med at vi benytter servlets, som nævnt i ovenstående afsnit. I vores projekt benytter vi altså vores servlets som controlleren. Alle JSP sider som har en bestemt funktion (f.eks. at logge ind, oprette en bruger, osv.), er tilknyttet en servlet. F.eks. har vi JSP siden "opret", som har en forbindelse til login servleten igennem HTML formen. Dertil har vi en submit knap inde på opret JSP siden, så når der klikkes på knappen vil koden i login servleten blive udført.

Data: mapper & facade

Inde i mappen persistence under model findes vores mapper og facade klasser. Mapper klasserne er der hvor, vi gemmer den information, vi skal bruge i en database, og i facade klasserne kalder vi på metoderne inde fra mapper klasserne, og returnerer disse. Mapper klasserne er her, vi skaber en forbindelse mellem vores database og koden. Det er også i mapper klasserne, at alle CRUD operationerne findes. Med andre ord vil metoderne i mapper klasserne kun være metoder, som har en direkte funktion ift. enten at læse information fra databasen eller indsætte information i databasen. I facade klasserne kalder vi på metoderne inde fra mapper klasserne. Hver mapper klasse har en facade klasse. F.eks. har vi UserMapper og UserFacade, som hænger sammen. Vi har facade klasserne med, da det skaber et bedre overblik over hvilke metoder, der er i mapper klasserne. Der kan hurtigt komme mange metoder inde i mapper klasserne, så derfor giver det et bedre overblik også have en facade klasse med til hver mapper klasse.

Man kan efterfølgende reflektere over, om man ikke kunne have nøjes med kun at have en samlet facade klasse ift. størrelsen på dette projekt. Som nævnt giver det god mening at have flere facade

klasser med, hvis der er tale om mapper klasser, som har mange metoder. Dog kan man sige at det gør opdelingen meget tydelig, at vi har en facade klasse med til hver mapper klasse, og det ville hellere ikke være et problem hvis systemet skal udvides, og der eventuelt ville komme flere metoder i mapper klasserne.

Test

Vi benytter os af JUnit til at teste om vores beregningsmetoder, til at udregne styklisten, er korrekte . JUnit test er en metode i en test klasse, som kun bruges til test. For at markere en metode som en testmetode, skal den anmærkes som `@Test`. Vi har i vores projekt udført tests på udregningerne, som vi bruger til at beregne antal af materialer, vi skal bruge til at sammensætte en carport ud fra dynamiske inputs. Nogle af testene er udført med 600 cm i bredde og 780 cm i længden uden skur, så vi kan få et bedre indblik i hvad vores forventede udfald ville være i forhold til det faktiske udfald. Enkelte tests er testet med 600 x 780 cm, og skur mål på 600 x 200 cm, for at se om vores metoder til der skal tage højde for ekstra materialer ved skur, også fungere. Vi har brugt de samme mål som vi fik udleveret i den færdige stykliste af fog, for at dobbelttjekke om vi kom frem til det samme resultat, som fremgik i deres stykliste. Deres mål for en standard carport er 600 x 780 cm.

```
56 // Calc of stern to front and back. 600x780 cm without shed
57 @Test
58 void calcUndersternFrontAndBack () throws DatabaseException {
59     int stern = Calculator.calcUnderSternFrontAndBack( ID: 0, width: 600, length: 780, connectionPool).getQuantity();
60     int expectedStern = 4;
61     assertEquals(expectedStern, stern);
62 }
```

På billedet ses at expected stern & actual stern er eksakte, derfor får vi et grønt flueben ude til venstre.

```
✓ Tests passed: 1 of 1 test - 599 ms
22:42:14.497 [main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
dec. 30, 2022 10:42:14 PM dat.backend.model.persistence.ConnectionPool getConnection
INFO: : get data connection
dec. 30, 2022 10:42:14 PM dat.backend.model.persistence.ConnectionPool getConnection
INFO: : get data connection
22:42:14.528 [HikariPool-1 connection adder] DEBUG com.zaxxer.hikari.pool.HikariPool - HikariP
Expected stern: 4
Actually stern: 4

Process finished with exit code 0

Tests passed: 1
```

Billed af "test passed" i console med eksakte værdier.

```

83 // calc under stern in front and back end of the garage.
84 // 5x200 mm. trykimp. Brædt 360 4 Stk understernbrædder til for & bag ende
85 @ public static BillOfMaterialLine calcUnderSternFrontAndBack(int ID, double width, double length, ConnectionPool connectionPool) throws DatabaseException {
86
87     int stern1 = (int) (width + 5);
88     int stern2 = (int) (width + 5);
89     int stern3 = stern1 + stern2;
90     int stern_result = (int) Math.ceil(stern3 / 360.0);
91
92     Item items = ItemFacade.getItemByID(ID, 1, connectionPool);
93     int price = stern_result * items.getPrice();
94
95     return new BillOfMaterialLine(items.getItem_id(), items.getItem_name(), items.getUnit(), items.getLength(), price, items.getItem_description(), stern_result, ID);
96 }

```

Billed af metoden vi tester på.

Nedenstående skema viser en oversigt over hvilke klasser og metoder, vi har valgt at lave test på.

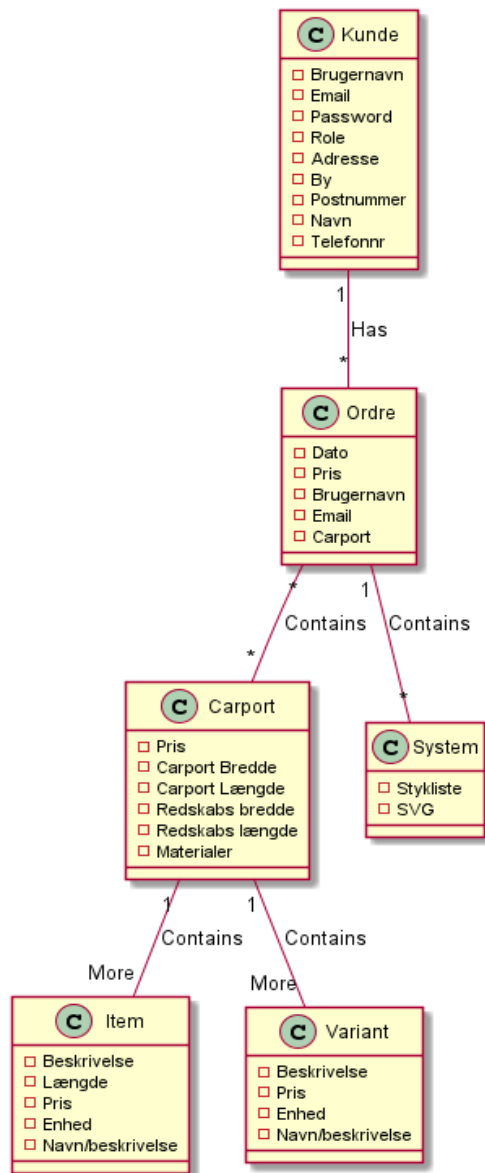
Klasse	Metode
Calculator	CalcPost()
Calculator	CalcStrap()
Calculator	CalcRafter()
Calculator	CalcUnderSternFrontAndBack()
Calculator	CalcUndersternSides()
Calculator	CalcUndersternSmall()
Calculator	CalcWeatherBoard1()
Calculator	CalcWeatherBoard2()
Calculator	CalcMeasureMentTape()
CalculatorShed	CalcStrapForShed()

Modeller & diagrammer

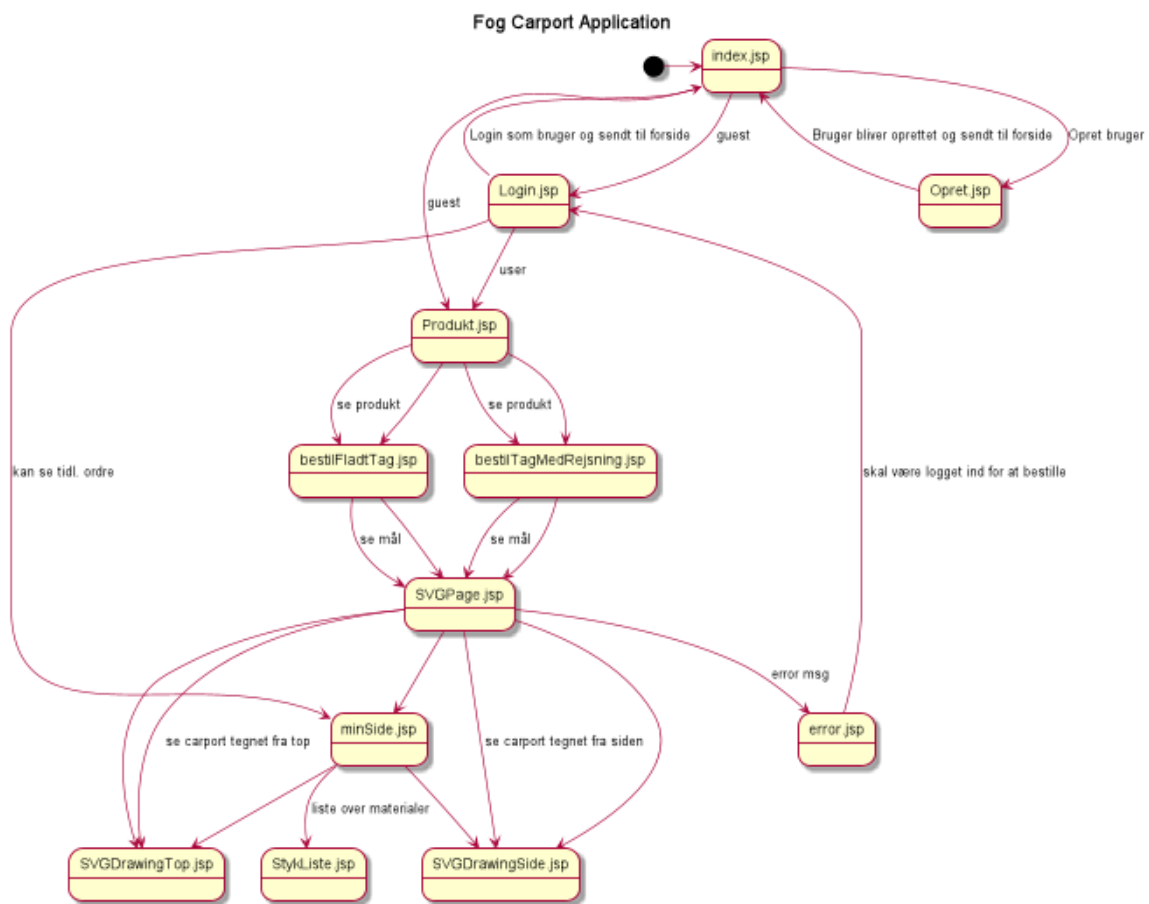
Domæne model

Vores domæne model viser en logisk-, og overskuelig struktur over hvordan den forsimplede process foregår. Vi har valgt at tage få attributter med i domænemodellen, da vi havde en ide om at disse attributter, kunne være relevante i forhold til projektet. Hvis vi starter med at have en oprettet kunde, så må vi antage at en kunde kan have ingen eller flere ordre. En ordre kan indeholde ingen eller flere carporte, samtidig kan en ordre også indeholde inputs fra systemet, hvor der altså kan være ingen eller flere tegninger. En carport indeholder flere items og en carport indeholder også flere varianter.

Her ses vores domæne model.



Navigationsdiagram



Ovenstående navigationsdiagram viser, hvordan man som bruger af hjemmesiden, kan navigere rundt på siden. Index JSP siden er den første side, som brugeren kommer ind på ved at tilgå hjemmesiden. Herefter er det muligt for brugeren at vælge mellem at oprette sig som bruger, logge ind med en allerede eksisterende bruger eller gå videre til at sammensætte sin egen carport. Ift. JSP siderne er det muligt at komme videre til opret, login og produkt JSP siderne fra index siden.

Hvis man som bruger af hjemmesiden ikke har et login, er det kun muligt at sammensætte en specifik carport og modtage tegningerne af den. Det er dog ikke muligt for brugeren at købe carporten, før brugeren har oprettet et login i systemet. Hvis brugeren alligevel forsøger at købe carporten, vil brugeren blive sendt videre til login JSP siden, med en besked om at man skal være logget ind, før man kan købe en carport.

Når brugeren har oprettet sig i systemet, er det muligt at logge ind med et brugernavn og password. Herefter er det også muligt at købe en carport ud fra specifikke mål. Når brugeren fra index siden trykker på carport i navigationsbaren, vil brugeren blive sendt videre til produkt JSP siden. Her er det

muligt at vælge mellem en carport med fladt tag eller en carport med høj rejsning. Når brugeren har valgt en af de to muligheder, vil brugeren blive sendt videre til enten `bestilCarportMedFladtTag` eller `CarportMedHøjRejsning`. Her vil brugeren indtaste de ønskede mål for den specifikke carport, hvor det også er muligt at indtaste målene på et skur, hvis dette ønskes. Herefter vil brugeren blive sendt videre for at acceptere købet af carporten. Når carporten er købt, bliver brugeren sendt videre til min side, hvor brugeren kan tilgå alle ordre og tilhørende stykliste og tegninger. Det er desuden muligt at logge ud fra alle JSP siderne, hvor at brugeren vil blive sendt tilbage til index siden.

Database

Normalisering

Nedenstående billede viser et EER diagram over vores database. Som det kan ses, har vi valgt at indsætte 6 tabeller i vores database. Vi har forsøgt at følge tredje normalform indenfor normalisering af databaser. For at opfylde kravene til tredje normalform, skal databasen leve op til forskellige krav. Det vi har gjort for at følge tredje normalform er bl.a., at vi i hver tabel har autogenereret et unikt ID til hver række i tabellen, for at det er nemt at identificere en bestemt række i en tabel. Ydermere vil felterne i vores tabeller kun indeholde en værdi (atomare værdier). Et eksempel på dette kan være, at vi i vores user tabel har valgt at opdele adressen i flere felter under navnene “adresse”, “city” og “postcode”, i stedet for at samle alle informationerne/værdierne om adressen i et felt. Her er det relevant at overveje, om vi også burde indsætte endnu et felt som f.eks. “husnummer” for at gøre opdelingen endnu mere tydelig. For at følge tredje normalform skal alle kolonner i en tabel indeholde data om kun en entitet. I de fleste tabeller har vi fulgt dette krav, men i bomvariant og bom tabellerne refererer vi dog til to entiteter. Bomvariant tabellen refererer både til et bestemt `ordreID` og `itemvariant ID`. Hvis man ser på felterne i bomvariant tabellen er der egentlig kun en sammenhæng mellem `itemvariant ID` og de forskellige felter, som indeholder information om de enkelte materialer. Vi har dog valgt at have `ordre ID` med, for at skabe en sammenhæng mellem ordren og styklisten.

Beskrivelse af databasen

`Itemvariant` og `item` tabellerne indeholder begge beskrivelserne af materialerne, som manuelt er blevet sat ind i tabellerne. `Itemvariant` tabellen er til skruer & beslag, og `item` tabellen er til træ & tagplader. `Itemvariant` tabellen har en en-til-mange relation til `bomvariant` tabellen, og `item` tabellen har en en-til-mange relation til `bom` tabellen.

`Bom` og `bomvariant` tabellerne tager informationerne fra `itemvariant` og `item` tabellerne, men har også et ekstra felt, som hedder `quantity`. Her kommer antal af materialerne ind efter beregningerne, og passer til en bestemt ordre, som også er et felt i `bom` og `bomvariant` tabellerne.

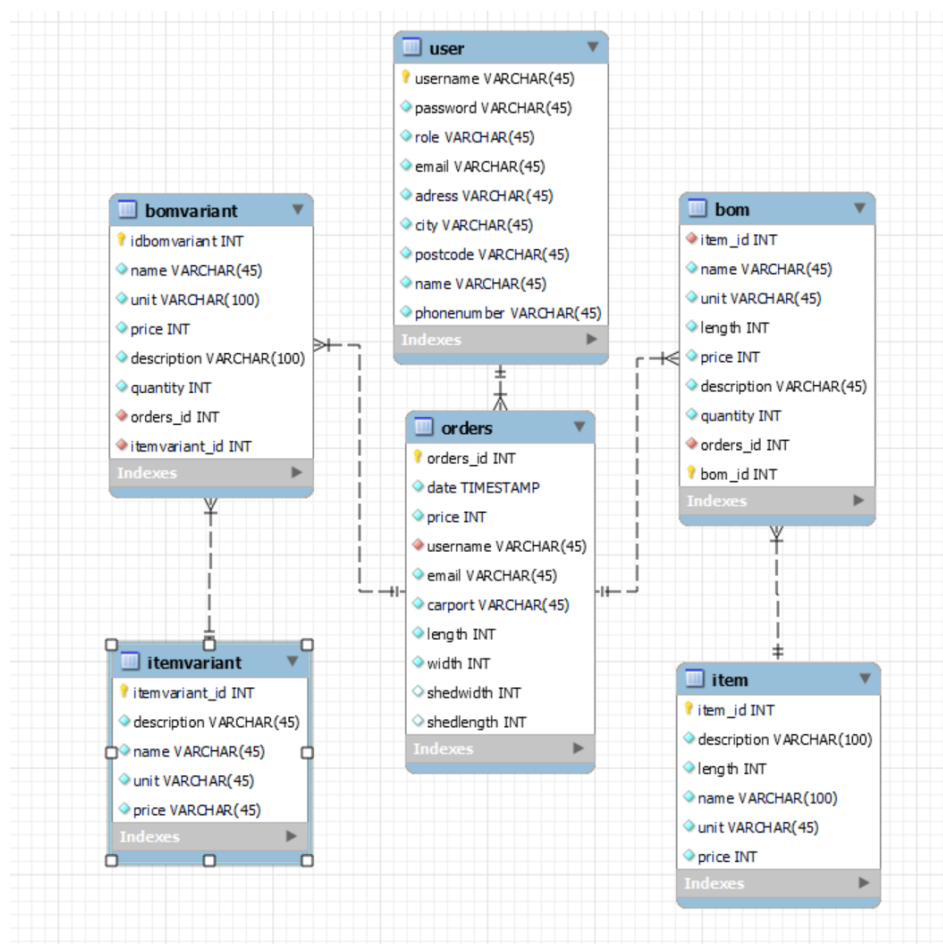
BomVariant har som nævnt en en til mange relation med ItemVariant. Den er sat op således, at der kan være mange BomVariants, men kun en ItemVariant. Der kan være mange forskellige BomVarianter, som tilhører forskellige ordre, men inde i ItemVariant tabellen findes kun de manuelt indsatte informationer om materialerne. Den samme relation gælder for Bom og Item tabellerne.

Bom og BomVariant tabellerne har desuden en en til mange relation til order tabellen, hvor Bom og BomVariant vil der være mange rækker af, men som kun tilhører en ordre. Orders tabellen har en en til mange relation til User tabellen. Der kan være mange ordre, men kun en user på den ordre.

Generelt

Man kunne have valgt at fjerne tabellerne item og itemvariant helt, da tabellerne kun indeholder manuelt indsatte værdier med informationer om materialerne. Vi valgte dog at tage disse tabeller med alligevel, for at det blev nemmere at ændre informationerne om materialerne, hvis dette skulle gøres, eller tilføje nye materialer til listen. F.eks. kan det være at prisen på materialerne ændres, og derfor ville det være nemmere at gå ind og ændre det i databasen for en medarbejder hos Fog, i stedet for at ændre det i koden.

ERR Diagram



Særlige forhold

Exceptions

I de fleste af mapper klasserne findes metoder, som har til hensigt at indsætte informationer i vores database. Det er kun `ItemMapper` og `ItemVariantMapper`, som kun har det formål at læse informationen fra databasen. I de klasser hvor der er en tilhørende metode, som har det formål at indsætte information i databasen, benytter vi os af Try-Catch og Throws keyword til at håndtere eventuelle fejl. Vi bruger Try til at teste vores del af koden, hvor vi prøver at indsætte information i databasen, og bruger catch til at kalde på et Throw statement, hvor vi selv formulerer hvad fejlbeskeden skal være. I `UserMapper` klassen i login metoden er fejlbeskeden "Forkert brugernavn eller kodeord". Når vi kalder på denne metode i `UserFacade`, sætter vi throws keyword på metode signaturen som vist nedenfor.

```
public static User login(String username, String password, ConnectionPool connectionPool) throws DatabaseException {  
    return UserMapper.Login(username, password, connectionPool);  
}
```

Ved at benytte throws keyword tager vi vores customised exception med. Vi kalder på metoden inde i vores login servlet, hvor vi pakker metode kaldet ind i en Try-Catch som vist nedenfor.

```
try {  
    User user = UserFacade.login(username, password, connectionPool);  
    session = request.getSession();  
    session.setAttribute("user", user);  
    session.setAttribute("username", username);  
    request.getRequestDispatcher("index.jsp").forward(request, response);  
} catch (DatabaseException e) {  
    request.setAttribute("errorMessage", e.getMessage());  
    request.getRequestDispatcher("error.jsp").forward(request, response);  
}
```

I Try delen kalder vi på login metoden inde fra vores `UserFacade`. Hvis der opstår en fejl (f.eks. at brugeren ikke har indtastet det rigtige brugernavn eller adgangskode) vil beskeden vi formulerede i `UserMapper` klassen, blive gemt i et request scope, så brugeren kan få beskeden vist på hjemmesiden. Denne fremgangsmåde bliver brugt hver gang, der skal indsættes informationer i databasen. Vi formulerer en customised fejlbesked inde i mapperne, og håndtere først fejlen inde i de forskellige servlets.

Sessions

I Johannes Fog carport applikationen gemmer vi værdier og objekter på forskellige scopes. Det vil sige at de ikke lever lige lang tid.

Application Scope:

I applikationen gemmer vi et Connection Pool objekt, som holder fem aktive forbindelser til databasen. Dette har betydning, da vi ikke behøver at bruge så mange ressourcer og tid på at oprette nye forbindelser. Dette objekt gemmes på tværs af applikationen.

Session Scope:

Når der er et session scope, så nulstilles sessionen ikke når den rammer klienten. Derimod lever sessionen videre på klienten, så længe sessionen ikke bliver stoppet eller timer ud. Et eksempel på hvor vi koden vi har valgt at benytte session scope, er at vi gemmer et User objekt på session scope. Det vil sige at når en bruger logger ind med korrekt brugernavn og adgangskode, så gemmes brugeren på sessionen, så længe brugeren er aktiv. Brugeren timer ud efter en 30 minutters session. En af grundene til at vi gør dette, er bl.a. fordi at brugeren, skal have vist sine egne ordre inde under "min side". Når en ordre skal printes, kan vi derfor tjekke, hvilket brugernavn som er gemt på sessionen, og dermed printe de korrekte ordre ud.

Request Scope:

Når data sendes fra en servlet til en jsp side, så hægtes en reference på et request objekt. Man kan sige at referencen gemmes i et hashmap, og nulstilles så snart et respons når klienten. Processen gentages, hver gang der laves et request til klienten. Vi bruger request scopes forskellige steder i koden. En af de steder vi bruger det, er i vores SVGAfterPurchase servlet. Her gemmer vi vores SVG tegning på et request scope, og kalder på det inde i vores SVGDrawingTop jsp side. Grunden til at vi gemmer på et request scope netop her frem for et session scope, er fordi, at vi ikke har behov for at den specifikke SVG tegning, skal leve videre, som vi har med informationen om brugeren.

SVG tegning

Vi har valgt at bygge SVG tegningen op således, at vi har tre klasser (SVGTop, Helpfunction & SVGDrawing) som bruges til at tegne carporten, så den kan ses på vores hjemmeside. SVGDrawing klassen bruges primært til templates for de forskellige rektangler, linjer osv., vi skal bruge til at tegne carporten. Helpfunction klassen er her vi gør tegningen dynamisk. I helpfunction klassen findes forskellige funktioner, som skal udregne hvor mange spær, stolper, osv. der skal til for at bygge en carport ud fra de mål, som kunden indtaster i formularen.

```

public static SVGDrawing addRafter(SVGDrawing svg, SVGDrawing svg2, int length, int width) {

    int rafter1 = (int) Math.ceil(length / 55);

    int rafter2 = length / rafter1;

    int rafter = 0;

    while (rafter <= length) {
        svg2.addRect(rafter, y: 0, width, width: 4.5);
        svg.addSVG(svg2);
        svg.addLine( x1: rafter + 2, y1: 0, x2: rafter + 2, y2: -30);
        svg.addLine( x1: -20, y1: 0, x2: -20, width);

        if(rafter < length - 55) {
            svg.addText( x: rafter + 20, y: -15);
        }
        rafter = rafter + rafter2;
    }
    return svg;
}

```

Ovenstående billede viser en metode i helpfunction klassen, som skal beregne antal af spær til tegningen, så det bliver dynamisk, alt efter hvilke mål carporten har. I metoden starter vi med at finde frem til hvor mange spær, der skal bruges til carporten ud fra længden. Herefter indsætter vi et while loop, som tilføjer et spær hver gang længden bliver 55 cm større, da dette er afstanden mellem hvert spær ifølge skitsen, som vi fik udleveret med opgavebeskrivelsen. While loopet slutter, hvis rafter variablen bliver større end længden. I loopet tilføjes også en linje for hver gang rafter bliver 55 cm større, for at kunne tyde målene på tegningen. Metoden returnerer til sidst et svg objekt. Metoden har desuden to forskellige svg objekter med i parametrene. Dette gør vi fordi at vi skal tegne en tegning ovenpå en anden tegning.

```

40     int ID = (int) session.getAttribute( s: "ID");
41
42     try {
43         currentOrder = OrderFacade.getOrderById(ID, connectionPool);
44     } catch (DatabaseException e) {
45         e.printStackTrace();
46     }
47
48     int length = currentOrder.getLength();
49     int width = currentOrder.getWidth();
50     int shedwidth = currentOrder.getShedwidth();
51     int shedlength = currentOrder.getShedlength();
52
53     SVGDrawing carport = HelpFunction.createNewSVG( x: 0, y: 0, height: 80, width: 60, viewBox: "0 -30 1500 1500");
54     SVGDrawing carport2 = HelpFunction.createNewSVG( x: 0, y: 0, height: 100, width: 100, viewBox: "0 -30 1500 1500");
55
56     HelpFunction.addShed(carport, length, width, shedwidth, shedlength);
57     HelpFunction.addRafter(carport, carport2, length, width);
58     HelpFunction.addStrap1(carport, carport2, length, width);
59     HelpFunction.addStrap2(carport, carport2, length, width);
60     HelpFunction.addPostToShed(carport, carport2, length, width, shedwidth, shedlength);
61     HelpFunction.addDashedLines(carport, length, width, shedwidth, shedlength);
62     HelpFunction.addPost(carport, carport2, length, width);
63
64     request.setAttribute( s: "svg", carport.toString());
65     request.getRequestDispatcher( s: "SVGDrawingTop.jsp").forward(request, response);
66 }

```

Sidst har vi klassen SVGAfterPurchase, som er en servlet. I SVGAfterPurchase klassen starter vi med at fat i et ordre ID på linje 40, som er gemt på et sessionscope. Herefter kalder vi på en metode fra vores OrderFacade klasse på linje 43, som får fat i alle de informationer om ordren, vi har brug for, for at lave tegningen til carporten. Vi har brug for at kende længden, bredden, skurlængden og skurbredden på carporten fra den gældende ordre, for at kunne tegne carporten. Disse informationer gemmer vi i fire forskellige variable på linje 48-51. På linje 53 og 54 instantierer vi tre nye SVG tegninger, som vi bruger når vi skal kalde på hjælpefunktionerne fra HelpFunction klassen. Til sidst på linje 64 gemmer vi carport på request scopet, så vi kan kalde på den inde i en JSP side, så brugeren kan se tegningen på hjemmesiden.

Ydermere har vi også servlet klassen “SVGBeforePurchase” med når der skal tegnes en SVG tegning før køb af carporten. Det skal være muligt for brugeren at se en tegning af carporten, før købet/ordren gennemføres.

```
21 HttpSession session = request.getSession();
22 request.getSession();
23
24 int length = (int) session.getAttribute( s: "length");
25 int width = (int) session.getAttribute( s: "width");
26 int shedlength = (int) session.getAttribute( s: "shedlength");
27 int shedwidth = (int) session.getAttribute( s: "shedwidth");
28
29
30 SVGDrawing carport = HelpFunction.createNewSVG( x: 0, y: 0, height: 80, width: 60, viewBox: "0 -30 1500 1500");
31 SVGDrawing carport2 = HelpFunction.createNewSVG( x: 0, y: 0, height: 100, width: 100, viewBox: "0 -30 1500 1500");
32 SVGDrawing carportText = HelpFunction.createNewSVG( x: 0, y: 0, height: 100, width: 100, viewBox: "0 -30 1500 1500");
33
34 HelpFunction.addShed(carport, length, width, shedwidth, shedlength);
35 HelpFunction.addRafter(carport, carport2, length, width);
36 HelpFunction.addStrap1(carport, carport2, length, width);
37 HelpFunction.addStrap2(carport, carport2, length, width);
38 HelpFunction.addPostToShed(carport, carport2, length, width, shedwidth, shedlength);
39 HelpFunction.addDashedLines(carport, length, width, shedwidth, shedlength);
40 HelpFunction.addPost(carport, carport2, length, width);
41
42 request.setAttribute( s: "svg", carport.toString());
43 request.getRequestDispatcher( s: "SVGDrawingTop.jsp").forward(request, response);
44 }
```

Ovenstående billede viser SVGBeforePurchase klassen. Denne servlet gør meget af det samme som SVGAfterPurchase klassen, men der hentes dog ikke et bestemt ordreID her, da carporten ikke er købt endnu. Derfor sætter vi længden, bredden, skurlængden og skurbredden til at være det som er gemt på vores sessionscope. Når kunden sammensætter målene på sin carport, vil de inputs blive gemt på sessionscopet inde under vores CreateOrder servlet.

Opbygning af stykliste

Calculator og calculatorShed

Vi har valgt at bygge styklisten op således, at vi har to klasser (calculator og calculatorShed), som bruges til at beregne hvert materiale i styklisten. Vi laver en metode/funktion til hvert materiale i styklisten, som efterfølgende returnerer en instans af BillofMaterialline klassen.

```
28 @
29
30 public static BillofMaterialLine calcRafter(ConnectionPool connectionPool, int ID, double width, double length) throws DatabaseException, SQLException {
31     int rafter = (int) Math.ceil(length / 55);
32     Item items = ItemFacade.getItemByID(ID, connectionPool);
33
34     int price = items.getPrice() * rafter;
35     return new BillofMaterialLine(items.getItem_id(), items.getItem_name(), items.getUnit(), items.getLength(), price, items.getItem_description(), rafter, ID);
36 }
```

Ovenstående billede viser metoden fra Calculator klassen, som bruges til at beregne antal af spær/rafter til carporten. Vi henter informationerne om hvert materiale fra databasen ud fra et unikt ID, og sætter quantity til at være det, som beregningerne finder frem til. På linje 30 beregner vi antal af spær ved at dividere længden på carporten med 55, som er afstanden mellem hvert spær. På linje 31 instantierer vi et nyt objekt af Item, og sætter det lig med at være en metode fra ItemFacade klassen. Metoden returnerer et bestemt item ud fra det ID, man sætter i parametrene. På linje 33 sætter vi prisen ved at gange prisen på det bestemte item med antallet af spær. Til sidst på linje 34 returnerer vi en ny instans af BillofMaterialLine. Vi bruger gettere til at få fat i informationerne om materialet (som f.eks. navn, beskrivelse, osv.) og sætter quantity til at være variablen rafter, som resultatet af beregningerne. Vi sætter OrderID til at være ID, som er det metoden har med i parameterlisten.

Calculator klassen har udelukkende metoder/funktioner som beregner en carport uden skur, hvor calculatorShed klassen har metoder/funktioner med, som beregner de materialer fra styklisten, som kun skal med, hvis carporten har et skur. Den måde vi tjekker om carporten har et skur, er ved at tjekke ved hjælp af en if sætning om bredden og længden på skuret er 0. Hvis bredden og længden er 0, må vi antage at kunden ikke ønsker et skur, og så vil metoderne hvor skur delene beregnes, ikke blive tilføjet til en liste.


```

42  @
43
44  public static BillOfMaterialLine calcStrap(int ID, double width, double length, int shedWidth, int shedLength, ConnectionPool connectionPool) throws DatabaseException {
45      if (shedWidth == 0 && shedLength == 0) {
46          int strap = (int) length * 2;
47          int strap_result = (int) Math.ceil(strap / 600.0);
48          Item items = ItemFacade.getItemByID(ID, 8, connectionPool);
49          int price = strap_result * items.getPrice();
50          return new BillOfMaterialLine(items.getItem_id(), items.getItem_name(), items.getUnit(), items.getLength(), price, items.getItem_description(), strap_result, ID);
51      } else {
52          double n_length = length - shedLength;
53          int strap = (int) (n_length * 2);
54          int strap_result = (int) Math.ceil(strap / 600.0);
55          Item items = ItemFacade.getItemByID(ID, 8, connectionPool);
56          int price = strap_result * items.getPrice();
57          return new BillOfMaterialLine(items.getItem_id(), items.getItem_name(), items.getUnit(), items.getLength(), price, items.getItem_description(), strap_result, ID);
58      }
59  }

```

Ovenstående billede viser metoden til at beregne remmen på carporten fra Calculator klassen. Opbygningen af metoden er lidt det samme som metoden til at beregne spær, men her tjekker vi også for om carporten har et skur. På linje 44 tjekker vi for om carporten har et skur, ved at lave et if statement som lyder på, at hvis skurbredden og skurlængden er 0, så skal remmen beregnes ved at gange længden med 2, da vi antager at remmen skal dække længden af skuret to gange (en på hver side). Hvis skurbredden og skurlængden ikke er 0, så beregnes remmen ved først at minus længden med skurlængden, og herefter gange resultatet med 2.

CalculatorList

Derudover har vi klassen calculatorList, som bruges til at sætte de returnerede instanser ind i en liste. Klassen indeholder to metoder/funktioner, hvor den første metode bruges til at sætte træ og tagplade materialerne ind i en liste, og den anden metode bruges til at sætte skruer og beslag materialerne ind i en liste. Vi har valgt at opdele det sådan, da skruer og beslag ikke har en længde, hvor træ og tagplader derimod har. Begge metoder i klassen returnerer instanser af billofmaterielline, som vi efterfølgende kalder på inde i vores addorder servlet.

Sikkerhed og validering af brugerinput

I dette afsnit vil vi gennemgå sikkerheden i vores projekt, samt hvordan vi håndtere og validere brugerens input.

Errors

Vi benytter os af en error messages rundt omkring i projektet for at sikre os at når brugeren indtaster de korrekte værdier eller følger den rigtige struktur på siden.

I login smider vi en error message til brugeren hvis der bliver indtastet forkerte login oplysninger.

```
34
35 // sets the user object = null
36 session.setAttribute(s: "user", o: null);
37 // gets the parameter username & password
38 String username = request.getParameter(s: "username");
39 String password = request.getParameter(s: "password");
40
41 try {
42 // method call in facade that checks if the inputs match
43 User user = UserFacade.login(username, password, connectionPool);
44 session = request.getSession();
45 //if the values matches, set user & username and redirect to index.jsp
46 session.setAttribute(s: "user", user);
47 session.setAttribute(s: "username", username);
48 request.getRequestDispatcher(s: "index.jsp").forward(request, response);
49 } catch (DatabaseException e) {
50 // if values doesn't match redirect to error.jsp with an error messages
51 request.setAttribute(s: "errorMessage", e.getMessage());
52 request.getRequestDispatcher(s: "error.jsp").forward(request, response);
53 }
54 }
```

Brugeren bliver bedt om at prøve at logge ind igen.

Her ses et udklip af vores login servlet.

Når brugeren rammer login servlet, sættes useren til nul. Dette skyldes at brugeren ikke er logget ind endnu. Når brugeren indtaster login informationerne, bliver de indtastede værdier “username” og “password” af brugeren hentet på henholdsvis linje 38 og 39.

På linje 43 kaldes der på UserFacade.Login metoden og her tages de indtastede værdier med i parametrene. I UserFacade.Login metoden bliver vi sendt videre til vores UserMapper, som tjekker om parametrene stemmer overens med objektet, som er oprettet i databasen.

Hvis værdierne stemmer overens med de inputs, som brugeren har indtastet i frontend, sættes en ny user og username attribut i session scope og brugeren bliver derpå sendt videre til forsiden og et korrekt login. Hvis Brugeren har indtastet forkerte oplysninger gribes compileren med en database Exception e. Linje 51 sættes en ny attributes med en string “errorMessage” og brugeren bliver sendt videre til error.jsp med fejlbeskeden.

An error has occurred. This is the best message we can come up with right now:

Forkert brugernavn eller kodeord.

Jump back to the [Frontpage](#), or try [logging](#) in again.

Her ses fejl beskeden som brugeren modtager hvis der er indtastet forkerte login oplysninger, brugerne er sendt til error.jsp og får muligheden for at logge ind igen.

I bestilling af en carport bruges samme systematik som forrige eksempel. Systemet tjekker om brugeren er logget ind før den opretter en ordre. Hvis det er tilfældet at brugeren ikke er logget ind, bliver brugeren mødt med en error message. Brugeren bliver sendt til login siden med beskeden om at logge ind før der kan bestilles en carport. Når så brugeren er logget ind kan brugeren få lov til at bestille sin carport, og tilgå ordren på min side.

Du skal være logget ind for at oprette en ordre

Login Form




☐ Remember me

Login

[Forgot Password?](#)

[Sign up](#)

Or Sign Up With



fejlkode på login side, hvis brugeren prøver at bestille en carport uden at være logget ind.

I Vores opret servlet kalder vi på metoden i userFacaden som hedder getUserByUsername, og da vores username i databasen er unikt, kan vi altså tjekke om der allerede findes et brugernavn i databasen med samme navn. Hvis der ikke findes en bruger med samme username som blev indtastet

af brugeren, så kaldes metoden `userFacade.createUser`, som referere videre til vores mapper metode `createUser`. I denne metode oprettes et nyt user objekt med de indtastede værdier af brugeren. Hvis der allerede eksisterer det indtastede username i databasen, så sendes en fejlmeddelelse til brugeren om at brugernavnet allerede er taget.



Brugernavnet findes allerede

Opret bruger

Username

Password

Her ses fejlkoden når brugeren prøver at oprette et brugernavn der allerede findes i databasen.

SQL injection

En SQL injection kan defineres ved at en hacker misbruger en svaghed i SQL query, som giver hackeren adgang til fortrolige data, såsom brugernavne og adgangskoder. Hackeren kan herefter modificere dataen når der er adgang. Der er forskellige former for SQL injections, men vi vil gerne vise et eksempel med SQL injection hvor hackeren bruger en boolsk værdi til at skaffe sig adgang til databasen. Hvis vi forestiller os at hackeren står på en login side på en hjemmeside der ikke har sikret sin SQL query, det kunne at der kun blev brugt string concatenation (sammenkædning af flere strings). Dette kunne fx se sådan her ud.

```
preparedStatement = "SELECT * FROM users WHERE name = '" + userName + "';";
```

Så ville hackeren kunne skaffe sig adgang ved at skrive `' or '1'='1` i login feltet. SQL ville genkende den streng der bliver sendt med og omformulere querying til dette:

```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

Da 1 lig med 1 er en boolsk værdi og altid vil være true, så ville hackeren altså få returneret alle objekter fra “users” tabellen i databasen.

Vi har gjort følgende for at forhindre en SQL injection: I vores SQL Streng sætter vi username og password lig med en ukendt værdi(Det er “?” i querrien), her bruger vi PreparedStatement ps, ps.setString til at sætte usernamet der er indtastet af brugeren på den ukendtes værdi plads i sql Strengen. Vores “SELECT” statement i strengen beder systemet om at hente alle objekter i databasen og matche med de værdier vi har fået af brugeren.

```
String sql = "SELECT * FROM user WHERE username = ? AND password = ?";
```

```
try (Connection connection = connectionPool.getConnection()) {  
    try (PreparedStatement ps = connection.prepareStatement(sql)) {  
        ps.setString( parameterIndex: 1, username);  
        ps.setString( parameterIndex: 2, password);  
    }  
}
```

Implementering

I dette afsnit vil vi reflektere over hvilke userstories, vi nåede at implementere, og andre dele af opgaven som . Vi har i alt otte userstories, hvor vi kun har implementeret fem af dem. Det er primært de userstories, som har med kunden at gøre, vi har fået implementeret. Vi har også to userstories set fra administratorens synspunkt, som vi ikke nåede at implementere.

Userstories

Som nævnt mangler vi at implementere administratorens userstories. Her er der tale om userstories syv og otte, som kan ses i tabellen under afsnittet userstories øverst i rapporten. Vi var under tidspress, så derfor nåede vi ikke at fokusere særlig meget på administratorens userstories. Derfor valgte vi at fokusere på at få implementeret alle de userstories, som havde noget med kunden/brugeren at gøre.

Det som vi mangler er bl.a. at bruge “role” under user tabellen til noget relevant. På nuværende tidspunkt skal brugeren indtaste nogle værdier under feltet “role”, når der oprettes en bruger. Meningen var, at brugeren slet ikke skulle indtaste noget i dette felt, men at det skulle bruges som en slags checker, når der logges ind. Måden vi kunne implementere denne del på, er ved manuelt at oprette en admin user i vores database, hvor der under role ville stå “admin” i stedet for “user”. Når brugeren så logger ind på hjemmesiden, kunne man tjekke op på, hvilken rolle brugeren har ved en if sætning. Hvis brugerens rolle er “admin”, skulle brugeren se nogle andre ting end en normal bruger af hjemmesiden (kunde).

Ift. syvende user story ville vi implementere det lidt på samme måde, som når en kunde får printet alle sine ordre ud under “min side”. Når en kunde får printet sine ordre ud under “min side”, tjekkes der hvilken bruger som er logget ind, og printer kun de ordre ud, hvor brugernavnet på ordren stemmer overens med brugeren, som er logget ind. Når administratoren logger ind, skal det være sådan, at det bare er alle ordre i databasen, som bliver printet ud.

Den sjette userstory mangler vi også at få implementeret. Her var meningen, at det skal være muligt for brugeren at opdatere sine personlige informationer, som f.eks. adressen, hvis nu brugeren er flyttet, og gerne vil ændre det i systemet. Den måde vi gerne ville have løst det på, var ved at tilføje en ekstra metode i UserMapper klassen, som skulle hedde noget lignende “updateUser()”. Ift. til frontend delen skulle det være muligt for brugeren at få vist en formular inde under “min side” hvor brugeren kan se sine informationer, og eventuelt kunne ændre det her.

CRUD

I starten overvejede vi hvordan vi kunne sørge for at inddrage alle CRUD operationer i vores kode og hvor det ville give mest mening. Vi har de fleste af CRUD operationerne med, men mangler dog stadig at have *delete* og *update* operationerne med. Det var tiltænkt at vores sjette userstory skulle sørge for update og ottende userstory skulle sørge for at have delete med.

Generelt

Generelt mangler vi også at rydde lidt op i koden. En af målene vi havde sat os var bl.a. at undgå, at der kom alt for meget kode i vores servlets. I de fleste af vores servlets har vi netop også gjort dette, men vi mangler at få en del kode væk fra vores AddOrder servlet/klasse. Det vi eventuelt kunne have gjort, var at sætte alt koden som skal udregne styklisten og loope igennem listen, ind i en metode i klassen CalculatorList. Herinde har vi allerede en hjælpemetode, som udregner den samlet pris for carporten, som vi så kalder på inde i CreateOrder servletten.

En anden ting vi også gerne ville have fokuseret mere på var, at klassenavnene og variabelnavnene gav mere mening. Nogle steder i koden bruger vi camelcase, og andre steder bruger vi en bundstreg til at separere ordene i et variabelnavn. Vi kunne godt have tænkt os at der var lidt mere konsistens i vores kode, så alt passede bedre sammen og gav bedre mening.

Ydermere mangler vi at man kan vælge mellem forskellige tage til carporten. Kunde kan på nuværende tidspunkt vælge mellem “carport med fladt tag” eller “carport med høj rejsning”. Uanset hvilken en mulighed kunden vælger, vil det være en carport med fladt tag.

SVG tegningen

Ydermere ville vi også gerne have haft mere fokus på SVG tegningen. Vi har formået at tegne en skitse af en carport, og den er også gjort dynamisk således, at der kan tegnes en carport ud fra forskellige mål. Dog mangler vi at få sat pile på tegningen og at hulbåndet er en stiplede linje i stedet for en almindelig linje. Som tidligere nævnt var vi under lidt tidspress, så vi nåede ikke at få indført alle de ting på tegningen, vi gerne ville.