Fog-Carport 2 Semester Projekt

Flow 5: Hold C, Gruppe 3

I perioden

03/05/2021 - 28/05/2021

Peter Lønquist Thomasen

cph-pt123@cphbusiness.dk

Thomas Schwencke Overgaard

cph-to83@cphbusiness.dk

-

Demovideo:

https://youtu.be/Wmm9TC-O81I

Github:

github@Schwencke

github@PeterDaniel3D

https://github.com/Schwencke/fogCarport

Droplet:

(Peter) http://46.101.212.228:8080/Carport/

(Thomas) http://schwencke.dk

Login:

Bruger: Kunde, Email: q@q.dk, Password: q

Bruger: Salgsperson, Email: a@a.dk, Password: a

Indholdsfortegnelse

Indledning

I dette semester projekt beskæftiger vi os med udviklingen af en webbaseret applikation med udgangspunkt i semesterets undervisning. Udover har vi tilegnet os kendskab til javascript for, at implementere små funktioner som vi følte gav en mere dynamisk oplevelse.

Baggrund

Vi er blevet bedt om, at udvikle et beregningssystem til virksomheden Johannes Fog. Beregningssystemet skal benyttes af Johannes Fog's afdeling, der beskæftiger sig med salg af carporte på specialmål.

Systemet skal erstatte deres gamle system som er forældet og ufleksibelt. Det nye system skal både have gamle og nye funktioner.

Johannes Fog ønsker, med systemet, at kunne modtage og gemme forespørgsler fra kunder, redigere den enkelte forespørgsel i samråd med kunden og derefter generere et tilbud til kunden.

Samtidig skal der genereres en tegning, samt en stykliste på carporten, som efter betaling kan ses af kunden fra deres personlige side.

Det er også et ønske, at den enkelte medarbejder i afdelingen har mulighed for at logge på systemet og få et overblik over alle ordre og forespørgsler.

Kunden skal kunne logge ind på systemet og for at kunne se deres forespørgsler, ordre, gennemføre en betaling, og efter betaling se stykliste samt tegning af carporten.

Kravspecifikationer om webshoppens funktionalitet er udspecificeret i afsnittet om krav.

Teknologivalg

Applikationen er en Java baseret webapplikation, med tilknyttet MySQL database. Der er udviklet en frontend ved hjælp af HTML, JSTL og JavaScript. Applikationen er designet til, at blive afviklet på en Tomcat webcontainer. Styling af websitetet er foretaget med Twitters Bootstrap og egen CSS.

Applikationen tager udgangspunkt i denne startkode:
https://github.com/jonbertelsen/sem2-startcode

Udviklingsværktøjer

- IntelliJ 2020.3 - 2021.1
- Mysql Workbench - version 8.0

Teknologier

- Linux Ubuntu 20.04 x64 - Droplet server
- MySQL Ver 8.0.25-0ubuntu0.20.04.1 for Linux on x86_64 - Droplet Server
- Java - version 8
- JavaEE Web API - version 7.0
- Maven - version 1.8
- JSTL- version 1.2
- Apache Tomcat version 9.0

Dependencies og plugins

- JDBC MySQL connector - version 8.0.19
- Junit - version 5.7.1
- Hamcrest - version 1.3

Frameworks

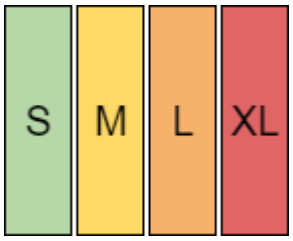
- Bootstrap - version 5.0

Krav

Johannes Fog ønsker, at der udvikles en ny smart og fleksibel carport beregner baseret på det eksisterende system. Formålet er, at skabe et mere brugervenlig og intuiativt miljø, som kan skabe forøget værdi for virksomheden og ikke mindst deres kunder, som også skal benytte sig af systemet. Den nye løsning skal forbindes med virksomhedens varekatalog, så priser og mål kan opdateres løbende uden, at skulle ændre på selve applikationen.

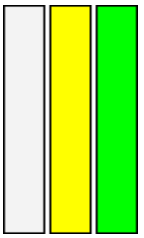
I samarbejde med Fog er der udarbejdet følgende user stories og acceptkriterier.

De enkelte stories er kategoriseret følgende:



Estimater i T-shirt størrelser

- S: Under 1 dag
- M: 1-2 dage
- L: 3-5 dage
- XL: 5-7 dage



Status indikering

- Grå: Ikke påbegyndt
- Gul: Under konstruktion
- Grøn: Afsluttet

User Stories: Kunde

Title:	#		Register	
User Story:		As	a customer	S
		I want to	register a new user	
		so that	I can place a request on my tailored carport	
Acceptance Criteria:	1	Given	that the account doesn't exists	
		When	the customer create a new user	
		Then	create a new user	

Title:	#		Login	
User Story:		As	a customer	S
		I want to	login to the webshop	
		so that	I can review my offers AND/OR orders	
Acceptance Criteria:	1	Given	that the user exists	
		When	the customer login	
		Then	log the customer onto the webshop	

Title:	#		Logout	
User Story:		As	a customer	S
		I want to	logout of the webshop	
		so that	another user can login	
Acceptance Criteria:	1	Given	that the user exists AND is logged on	
		When	the customer request a logout	
		Then	logoff the customer from the webshop	

Title:	#		Carport, Request	
User Story:		As	a customer	L
		I want to	request a custom build carport	
		so that	I can get an offer from the salesman	
Acceptance Criteria:	1	Given	that the customer has filled out the order form	
		When	the customer sends the request	
		Then	send a new carport request to the salesperson	
Acceptance Criteria:	2	Given	that the customer is logged on	
		When	the customer sends the request	
		Then	send a new carport request to the salesperson	

Title:	#		Offer, Accept	
User Story:		As	a customer	M
		I want to	accept the carport offer	
		so that	I can recieve my carport order	
Acceptance Criteria:	1	Given	that the offer exists	
		When	the customer has accepted the offer	
		Then	send an offer accepted status to the salesman	

Title:	#		Offer, Decline	
User Story:		As	a customer	M
		I want to	decline the carport offer	
		so that	I can get a new carport offer	
Acceptance Criteria:	1	Given	that the offer exists	
		When	the customer declines the offer	
		Then	send a declined offer status to the salesman	

Title:	#		Payment / Checkput	
User Story:		As	a customer	M
		I want to	pay for my order	
		so that	I can get my new carport	
Acceptance Criteria:	1	Given	that the customer has efficient funds	
		When	the customer make the payment	
		Then	change status of the order to paid	
Acceptance Criteria:	2	Given	that the customer has an order	
		When	the customer make the payment	
		Then	change status of the order to paid	

Title:	#		Bill Of Materials	
User Story:		As	a customer	M
		I want to	see my bill of materials	
		so that	I can check all the materials needed for my carport	
Acceptance Criteria:	1	Given	that the customer order is paid for	
		When	the customer review their order	
		Then	display the Bill Of Materials	

Title:	#		Carport, Draw		
User Story:		As	a customer	L	
		I want to	have access to a drawing off the my paid carport		
		so that	I can see how it looks		
Acceptance Criteria:	1	Given	that the customer order is paid for		
		When	the customer review their order		
		Then	draw the requested carport		

User Stories: Salgsperson

Title:	#		Login		
User Story:		As	a salesperson	S	
		I want to	login to the webshop		
		so that	I can see all offers AND orders		
Acceptance Criteria:	1	Given	that the user exists		
		When	the salesman login		
		Then	log the salesman onto the webshop		

Title:	#		Logout		
User Story:		As	a salesperson	S	
		I want to	logout of the webshop		
		so that	another user can login		
Acceptance Criteria:	1	Given	that the user exists AND is logged on		
		When	the salesman request a logout		
		Then	logoff the salesman from the webshop		

Title:	#		Offer, Create		
User Story:		As	a salesperson	XL	
		I want to	create a new offer for my customers		
		so that	the customer can review and accept my offer		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	create a new offer and calculate the costs		

Title:	#		Offer, Price: Carport Wood		
User Story:		As	a salesperson	M	
		I want to	calculate the amount of needed carport wood		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed wood and given costs		

Title:	#		Offer, Price: Carport Roofing		
User Story:		As	a salesperson	S	
		I want to	calculate the amount of needed roofing		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed roofing and given costs		

Title:	#		Offer, Price: Shed Wood		
User Story:		As	a salesperson	M	
		I want to	calc. needed shed wood without cladding		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed wood and given costs		

Title:	#		Offer, Price: Shed Cladding		
User Story:		As	a salesperson	S	
		I want to	calculate the amount of needed shed cladding		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed wood and given costs		

Title:	#		Offer, Price: Attachment		
User Story:		As	a salesperson	L	
		I want to	calc. the amount of needed screws, angle fittings etc.		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed attachments and given costs		

Title:	#		Offer, Modify		
User Story:		As	a salesperson	M	
		I want to	modify the offers for my customers		
		so that	I can help give them the best service		
Acceptance Criteria:	1	Given	that a customer offer exists		
		When	the salesman wants to modify the offer		
		Then	modify the offer		

Title:	#		Order, Create		
User Story:		As	a salesperson	M	
		I want to	create a new order		
		so that	I can send the order to the customer		
Acceptance Criteria:	1	Given	that the offer was accepted from the customer		
		When	the salesman create a new order		
		Then	send the new order to the customer		

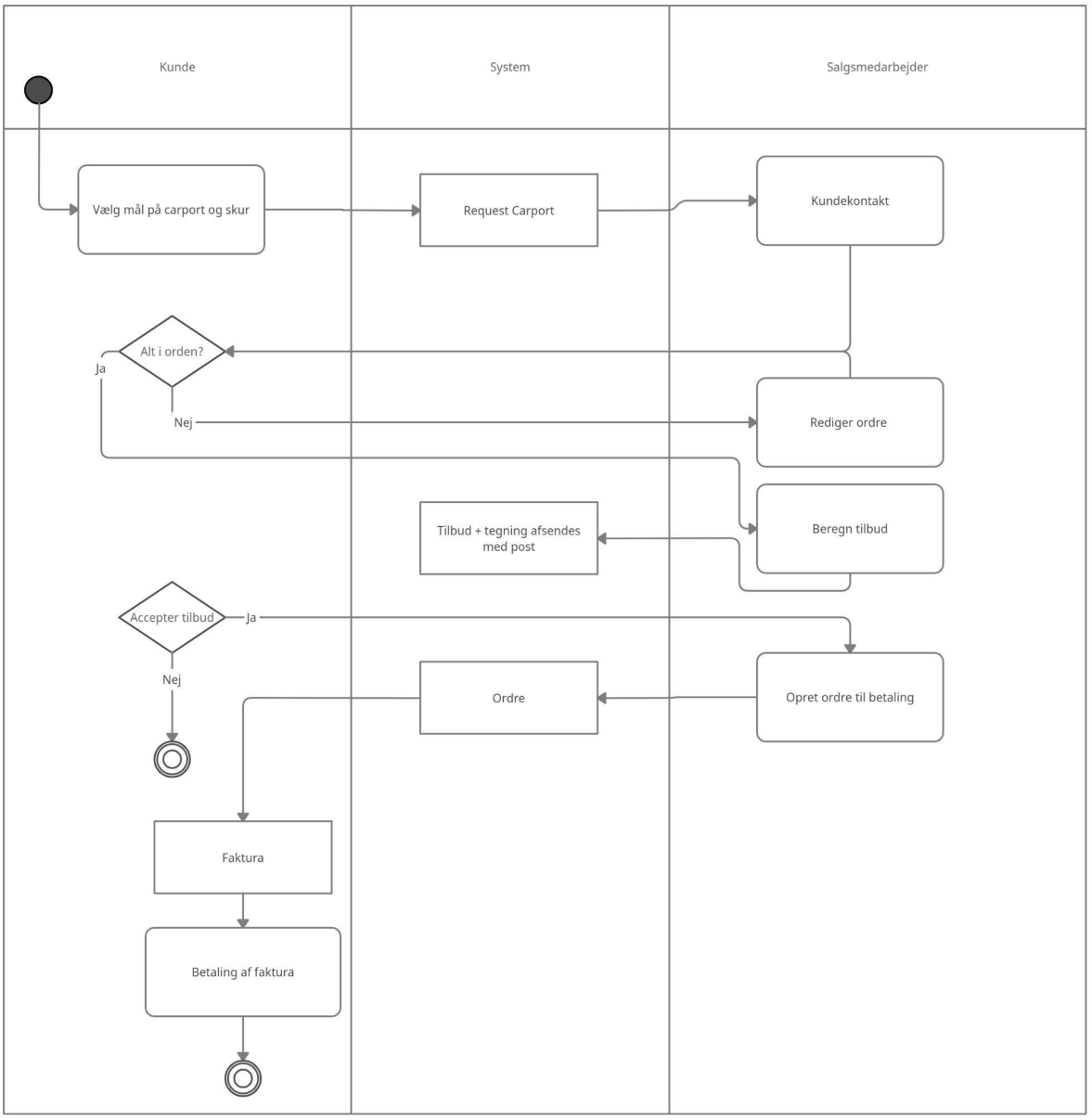
Title:	#		Update, Materials	
User Story:		As	a salesperson	M
		I want to	update the materials	
		so that	the materials are up to date	
Acceptance Criteria:	1	Given	the material exists	
		When	the salesman make an update	
		Then	update the material	

Title:	#		Bill Of Materials, Create	
User Story:		As	a salesperson	L
		I want to	create a bill of materials	
		so that	I can get the total price of the carport	
Acceptance Criteria:	1	Given	that a customer request exists	
		When	the salesman review the customers order	
		Then	calculate the needed materials and price	

Title:	#		Carport, Draw	
User Story:		As	a salesperson	L
		I want to	have access to a drawing off the carport	
		so that	I can see that everything is in order	
Acceptance Criteria:	1	Given	that a customer request exists	
		When	the salesman review the offer	
		Then	draw the requested carport	

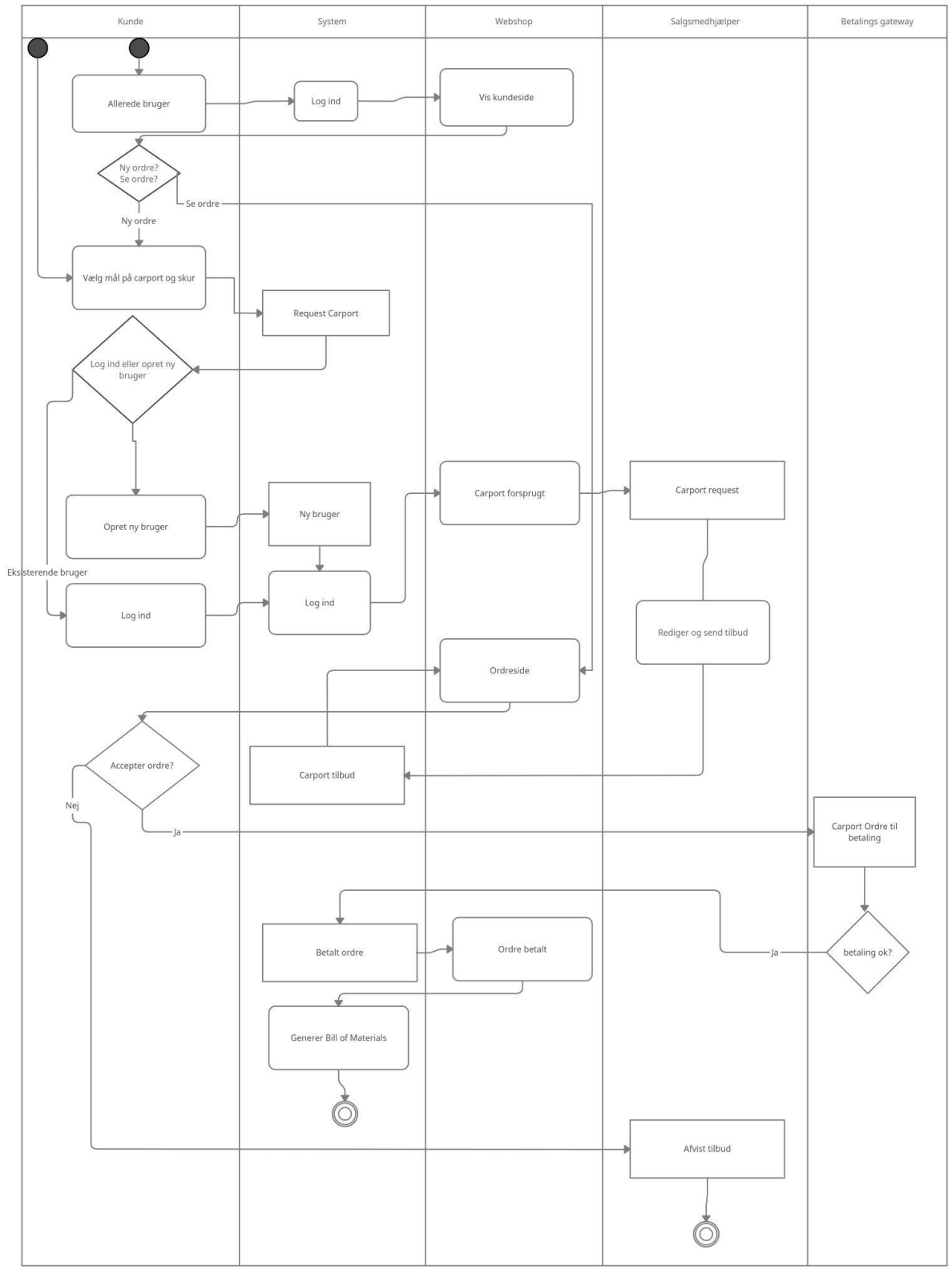
Arbejdsgange

as-is



Diagrammet beskriver arbejdsgangen på det gamle system for både virksomheds medarbejdere og kunder.

to-be



Diagrammet beskriver arbejdsgangen for både virksomheds medarbejdere og kunder, efter implementering af det nye system.

Scrum user stories

Der har løbende været afholdt møder med product-owner. Afsnittet her beskriver de aftalte user stories som product-owneren har ønsket fokus på imellem de enkelte sprints, samt gruppens antagelser af de enkelte forløb.

Forklaring på user story estimater og status kan findes under afsnittet om krav. Vi har valgt ikke at bruge estimater på enkelte opgaver, men i stedet løbende gjort status ud fra de overordnede estimeringer.

Oprettelse af tilbud

Title:	#		Offer, Create	
User Story:		**As**	a salesperson	XL
		I want to	create a new offer for my customers	
		so that	the customer can review and accept my offer	
Acceptance Criteria:	1	**Given**	that a customer request exists	
		When	the salesman creates a new offer	
		Then	create a new offer and calculate the costs	

Denne user story beskrives følgende: Som salgsperson skal man ud fra en forespørgsel, oprette et tilbud til den specifikke kunde. Ved oprettelsen af tilbuddet er prisen beregnet ud fra de ønskede mål.

Casen er en stor kernedel af Fogs kravspecifikationer, hvorfor gruppen ved 2. møde med product-owner blev bedt om, at opdele denne user story. Formålet var, at gøre de underliggende opgaver mere håndtérbare. Casen blev i første omgang opdelt i 3 dele: træ, beklædning og befæstigelse. Efterfølgende i 4 dele med tilføjelse af tag.

Efter at have påbegyndt kodning af beregningsmetoderne blev det hurtigt besluttet i gruppen, at ændre opdelingen yderligere for, at forenkle og overskueliggøre opgaverne endnu mere fra de ellers nyoprettede user stories.

Opgaver for Offer, Create:

- JSP side
- Layout
- Beregning af tilbuds pris
- Tegning af carport
- Visning af materialeliste
- Gemme tegning når tilbud er oprettet
- Gemme materialeliste når tilbud er oprettet

De seneste fordelte user stories fra Offer, Create er opdelt følgende: carport, tag, redskabsrum, beklædning og befæstigelse.

Title:	#		Offer, Price: Carport Wood		
User Story:		As	a salesperson	M	
		I want to	calculate the amount of needed carport wood		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed wood and given costs		

Title:	#		Offer, Price: Carport Roofing		
User Story:		As	a salesperson	S	
		I want to	calculate the amount of needed roofing		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed roofing and given costs		

Title:	#		Offer, Price: Shed Wood		
User Story:		As	a salesperson	M	
		I want to	calc. needed shed wood without cladding		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed wood and given costs		

Title:	#		Offer, Price: Shed Cladding		
User Story:		As	a salesperson	S	
		I want to	calculate the amount of needed shed cladding		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed wood and given costs		

Title:	#		Offer, Price: Attachment		
User Story:		As	a salesperson	L	
		I want to	calc. the amount of needed screws, angle fittings etc.		
		so that	I can calculate the price for the materials		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman creates a new offer		
		Then	calculate the needed attachments and given costs		

Gruppen har undervejs taget beslutning om ikke, at påbegynde beregningsmetoderne for redskabsrummet og befæstigelse. I stedet skullen tiden prioriteres på forbedre beregningsmetoderne for både carport og tag. Dette er med bagtanke om, at beregningerne, i tilfælde af overskydende tid, hurtigt ville kunne genbruges og tilpasses af den eksisterende kode på de manglede user stories: redskabsrum, beklædning og befæstigelse.

Opgaver for Offer, Price: Carport Wood / Roofing:

- Beregning af styk og længde på; stolper, remme, spær, sternbrædder, tag

SVG, Dækningsgrad, Materialeliste...

Ved 3. møde med product-owner blev der til sidste sprint aftalt en prioritering af opgaver: SVG, dækningsgrad, materialeliste, ændringer af lagerførte vare, beregning af skur.

Title:	#		Carport, Draw	
User Story:		As	a salesperson	L
		I want to	have access to a drawing off the carport	
		so that	I can see that everything is in order	
Acceptance Criteria:	1	Given	that a customer request exists	
		When	the salesman review the offer	
		Then	draw the requested carport	

Den implementerede user story fungerer således, at når en salgsperson sidder med en forespørgsel, har vedkommende mulighed for at se en tegning ud fra carportens mål.

Opgaver for Carport, Draw:

- Beregningsmetoder til tegning af: stolper, remme, spær, tag

Title:	#		Offer, Modify	
User Story:		As	a salesperson	M
		I want to	modify the offers for my customers	
		so that	I can help give them the best service	
Acceptance Criteria:	1	Given	that a customer offer exists	
		When	the salesman wants to modify the offer	
		Then	modify the offer	

Den user story beskriver, at man som salgsperson skal have mulighed for at ændre på en forespørgsel. Det indebærer bl.a. målene på carporten, men ikke mindst dækningsgrad af tilbuddet, som var en af de førnævnte prioriteringer fra product-owner.

Opgaver for Offer, Modify:

- Ændre og gemme carport mål
- Ændre dækningsgraden

Title:	#		Bill Of Materials, Create		
User Story:		As	a salesperson	L	
		I want to	create a bill of materials		
		so that	I can get the total price of the carport		
Acceptance Criteria:	1	Given	that a customer request exists		
		When	the salesman review the customers order		
		Then	calculate the needed materials and price		

Materialelisten oprettes automatisk for salg personen, så snart en forespørgsel foreligger. Materialelisten gemmes først i systemet, når tilbuddet oprettes og sendes tilbage til kunden.

Opgaver for Bill Of Materials, Create:

- Visning af materialeliste

Som kunde er det gældende, at man først får adgang til carportens tegning og materialeliste når der foretaget betaling af sin ordre. Det er for undgå, at man kan gå til anden leverandør for at købe de samme materialer til billigere penge.

Title:	#		Carport, Draw		
User Story:		As	a customer	L	
		I want to	have access to a drawing off the my paid carport		
		so that	I can see how it looks		
Acceptance Criteria:	1	Given	that the customer order is paid for		
		When	the customer review their order		
		Then	draw the requested carport		

Opgaver for Carport, Draw:

- Gemme SVG
- Indlæse SVG

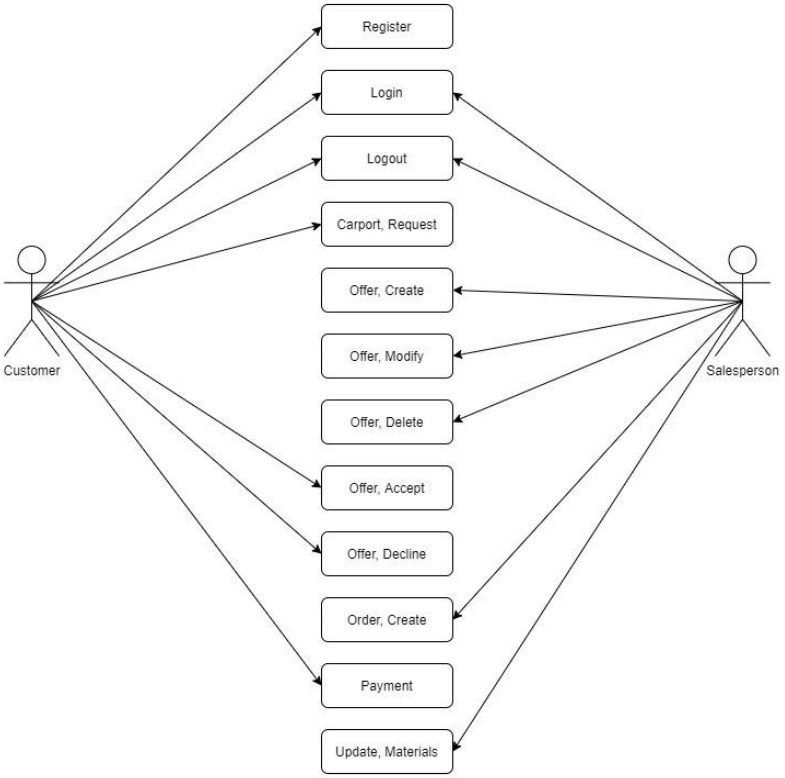
Title:	#		Bill Of Materials		
User Story:		As	a customer	M	
		I want to	see my bill of materials		
		so that	I can check all the materials needed for my carport		
Acceptance Criteria:	1	Given	that the customer order is paid for		
		When	the customer review their order		
		Then	display the Bill Of Materials		

Opgaver for Bill Of Materials:

- Gemme BoM
- Indlæse BoM

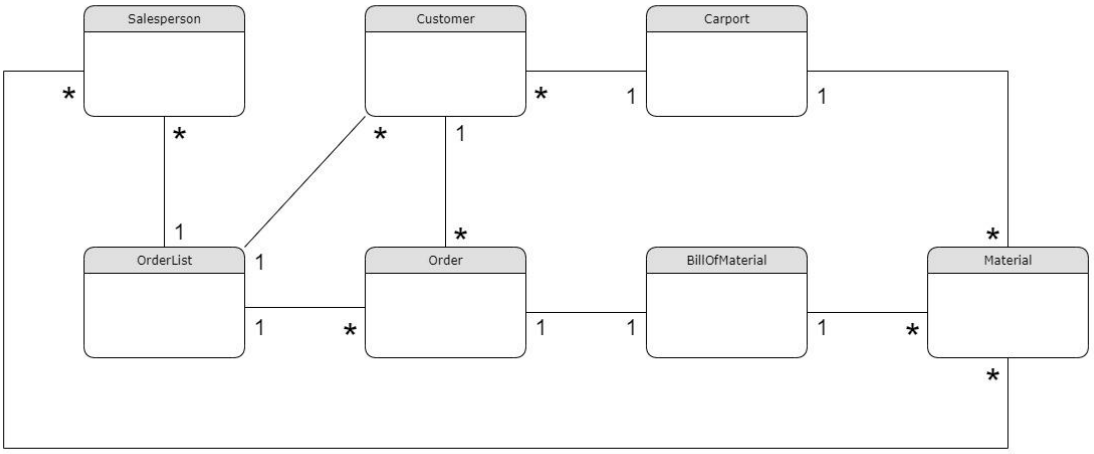
Diagrammer

Use Case



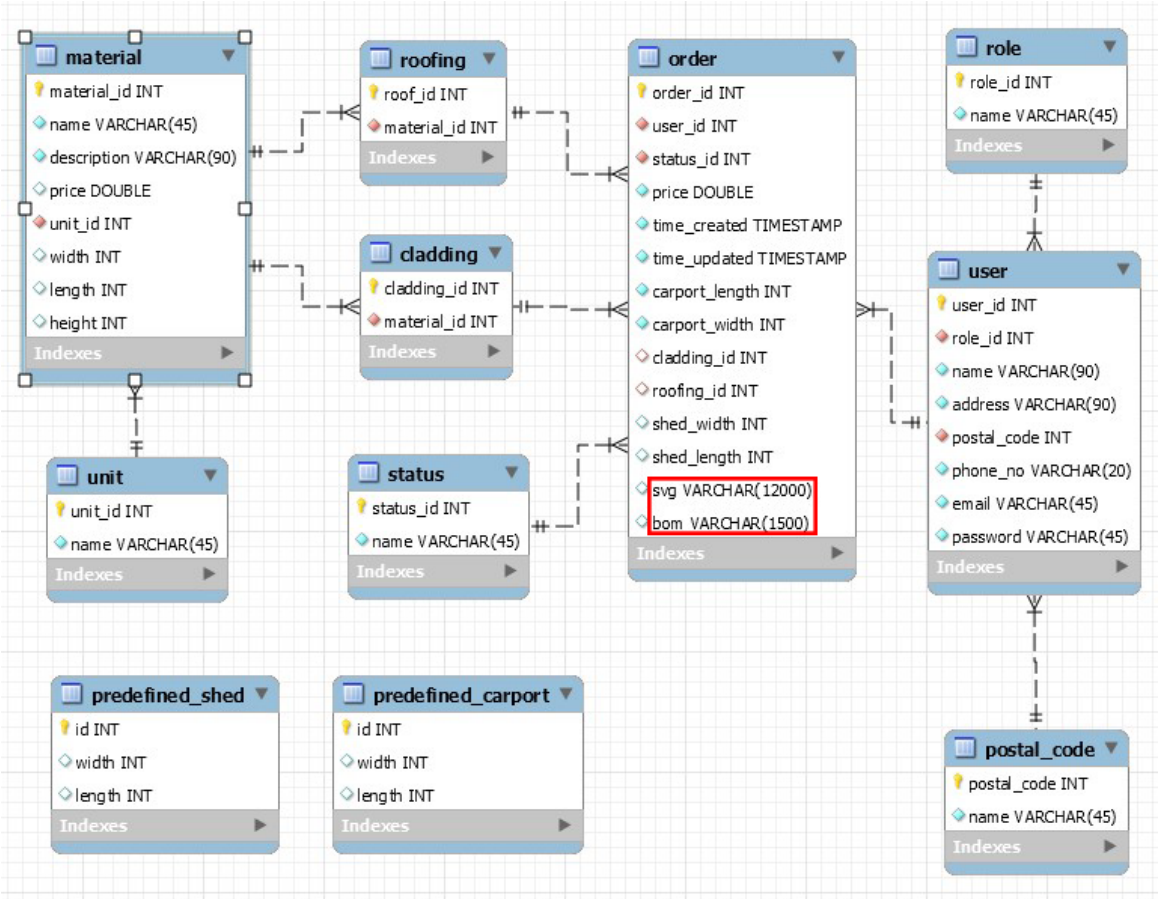
Use Case diagrammet og dens brug har været til formål at give Fog et overblik af, hvordan virksomheden og deres kunder tilgår systemet. Den er også brugt til udarbejdelsen af user stories.

Domæne model



Domæne modellen er lavet med udgangspunkt i kravspecifikationerne. Der er brugt god tid på diskussion om sammensætning af klasserne og deres relationer.

ER diagram



Databasen er sat op med tanke på enkelthed, genanvendelse, samt brugen af 3. normalform.

Primær Nøglerne på role, status og unit er ikke autogenereret nøgler, da det antages at kunden allerede har et POS-system (Point-of-Sale), hvor disse værdier indgår.

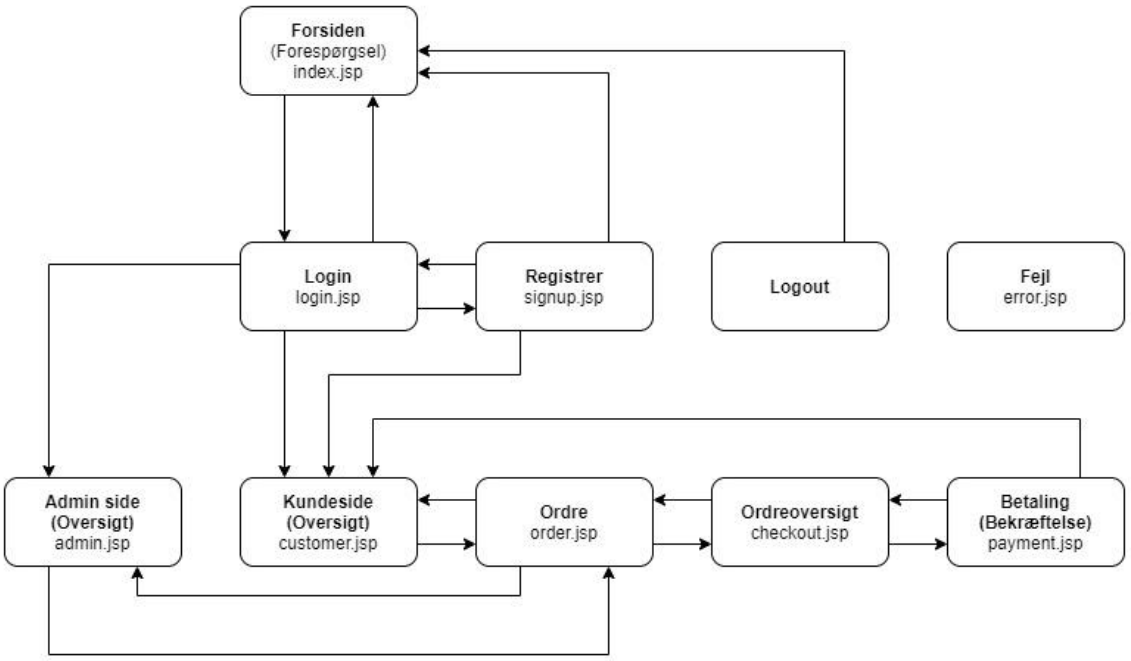
I den røde rubrik på ER diagrammet er svg og bom fremhævet. Disse variabler er tilføjet som eneste ændring ud fra vores oprindelige ER diagram. De bruges til at indeholde kundens carport tegning og materialeliste.

Tabellerne: roofing og cladding er opsatte link tabeller, som bruges til hhv. visning for valg af tag og beklædning til carporten. Idéen er, at ved udvidelse af Fogs varekatalog, så vil nye typer tag og beklædning kunne linkes således, at de kan bruges og vises automatisk i bestillingsformularen på Fogs hjemmeside.

predefinded_carport og predefined_shed er lavet som individuelle tabeller og indeholder mål for bredde og længde på hhv. carport og redskabsrum. Tabellerne indlæses og bruges til bestillingsformularen. Idéen er, at virksomheden vil have mulighed for, via hjemmesiden, selv at

kunne ændre målene efter behov. Dette ville ellers ikke være muligt, hvis målene var “hardcoded” i kildekoden.

Navigationsdiagram



Forside (index.jsp):

- Som kunde kan man bruge bestillingsformularen og sende en forespørgsel.
- Som kunde kan man anvende sit login for, at få adgang til systemet.
- Som salgsperson kan man anvende sit login for, at få adgang til systemet.

Login (login.jsp):

- Fra denne side kan man anvende sit login til, at få adgang til beskyttede sider. (Bemærk at man kun rammer denne side i tilfælde af, at man prøver at skaffe sig adgang til beskyttede sider, hvor man enten ikke er logget på eller som man ikke har rettigheder til at se.)
- Som kunde henvises man altid til forsiden.
- Som salgsperson henvises man til admin side.

Registrer (signup.jsp):

- På denne side kan man oprette sig som kunde hos Fog.

Logout:

- Log ud. Findes kun i headeren, når brugeren er logget på.

Fejl (error.jsp):

- Man henvises til denne side i tilfælde af opståede fejl, som f.eks. at der ikke kunne oprettes forbindelse til databasen.

Admin side (admin.jsp):

- På denne side kan man som salgsperson se en oversigt af tidligere og igangværende ordre.
- Fra denne side kan man tilgå en specifik ordre.

Kunde side (customer.jsp):

- På denne side kan man som kunde se en oversigt af tidligere og igangværende bestillinger.
- Fra denne side kan man tilgå en specifik bestilling.

Ordre (order.jsp):

- Som kunde kan man se de tilknyttede kundeoplysninger.
- Som kunde kan man se specifikationerne på den valgte bestilling.
- Som kunde kan finde sin tilbudspris når et tilbud er modtaget.
- Som kunde har man, afhængigt af status, mulighed for:
    - at acceptere et tilbud
    - at annullere et tilbud. (Dette fjerner bestillingen)
    - at gå til betalingssiden efter et accepteret tilbud.
- Som kunde har man mulighed for, efter betaling, at se en tegning af sin carport, samt en materialeliste.
- Som salgsperson kan man se de tilknyttede kundeoplysninger.
- Som salgsperson kan man se specifikationerne på den valgte ordre.
- Som salgsperson kan man se priser u. moms, dækningsbidrag, tilbudspris u./m. moms.
- Som salgsperson kan man se en komplet materialeliste med styk- og totalpriser.
- Som salgsperson kan man se en tegning af carporten.
- Som salgsperson har man afhængigt af status, mulighed for:
    - at ændre mål på carporten.
    - at ændre mål på redskabsrummet.
    - at ændre dækningsgraden.
    - at sende tilbud til kunden.
    - at slette ordren.

Ordreoversigt (checkout.jsp)

- Denne side viser en komplet ordreoversigt på den ønskede carport.
- Herfra kan kunden gennemføre sin betaling.

Betaling (payment.jsp)

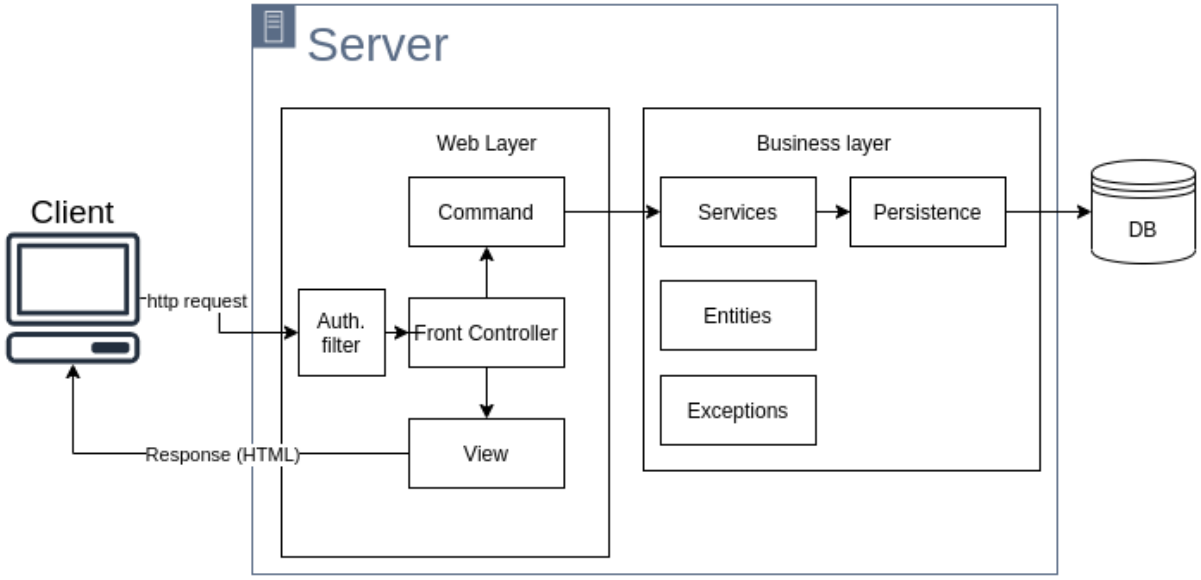
- Denne side vises som bekræftelse på, at betalingen er blevet gennemført for bestillingen.

Valg af arkitektur

Projektet tager udgangspunkt i denne startkode: https://github.com/jonbertelsen/sem2-startcode

Der benyttes blandt andet:

- Front Controller pattern
- Command pattern
- Facade pattern
- Dependency injection.



Der benyttes også MVC design metoden, Model-View-Controller, som illustreret her over.

Klientens forespørgsel løber til weblaget igennem et *authentication filter*, der er har til opgave at kontrollere, at klienten har de fornødne tilladelser til at se de resurser der forespørges. *Front Controlleren* organiserer hele applikationens flow.

Weblaget

Alt i weblaget drejer sig om at modtage en forespørgsel, udføre underliggende arbejde i businesslaget og derefter returnere resultaterne tilbage til klienten.

Authentication filter

Har til opgave at kontrollere at klienten, der sender forespørgslen, har de fornødne tilladelser til at se den forespurgte resurse. Der er tale om et helt basalt sikkerhedstjek der foretages for alle de beskyttede sider og commands.

Der tjekkes for:

1. at brugeren er logget ind.
2. at brugeren har den fornødne rolle til at tilgå siden.

Hvis sikkerhedstjekket går igennem, sendes forespørgslen videre til *Front Controlleren*.

Front Controller

Det centrale knudepunkt, der håndterer alle forespørgsler ind- og ud af webserveren.

Front Controlleren modtager og behandler forespørgslen fra klienten, der indeholder en command og et request objekt med tilhørende parameter. Commanden adskilles fra forespørgslen og slås op i command klassens tilhørende *hashmap*. Når commanden er eksekveret, omdirigeres forespørgslen til den tilhørende jsp side.

Command

Klassen implementere et Command Pattern som en abstrakt klasse. De forskellige commands holdes i et *hashmap*, hvor navnet på commanden tilskrives som nøgle og en instantieret subklasse af command klassen, som værdi, alt efter hvordan den givne command bruges.

De to subklasser er:

CommandProtectedPage benyttes på beskyttede side, hvor brugere skal være logget ind og have den rigtige rolle. Den tager to parametre *pageToShow* og *role*. pageToShow er navnet på den side som front controlleren skal omdirigere til, efter at commanden er blevet eksekveret. *role* er den rolle klienten skal matche for, at få adgang. I startkoden er det kun muligt, at tilskrive én rolle til hver side, men dette kan udvides. Vi har dog i denne opgave ikke haft behov for, at udvide funktionaliteten til at holde flere end én rolle pr. command.

CommandUnprotecedPage benyttes på de sider, hvor klienten ikke behøves at være logget ind. Derfor tager *constructoren* også kun én parameter, *pageToShow.*

Der findes en tredje subklasse, *UnknownCommand*. Denne bruges, hvis der ved gennemløb af *hash mappet* ikke findes en tilsvarende command. Der returneres her en 404, siden er ikke fundet.

View

Er jsp siderne i webapp mappen. I denne udgave af startkoden holdes siderne i WEB_INF folderen. Filer i WEB_INF mappen kan ikke tilgås, medmindre de tilgås i form af en forespørgsel, der sendes igennem en servlet. Vi har i gruppen valgt, at fortsætte denne tilgang for at siderne, som udgangspunkt, har en form for basis beskyttelse imod utilsigtet tilgang.

Businesslaget

Alt i businesslaget er tæt knyttet til kernefunktionerne i applikationen.

Entities

I entities holdes basis klasserne. Det er klasser der ofte indeholder et id nummer, som typisk bruges som nøgle i en eventuel database. I vores tilfælde: *User*, *Order*, *Material* og *BoM*.

Services

Her holdes de klasser der stykker funktionaliteten i applikationen sammen. Ofte beregningsklasser og hjælpemetoder.

Persistence

Her holdes de klasser der bruges til at persistere data. Gemme og hente data fra en ekstern kilde, ofte databaser. I denne startkode og i vores projekt persisterer vi vores data ved hjælp af en SQL database. Derfor er der klasser, datamappere, som henter og gemmer data på databasen. For at skabe forbindelsen til databasen er der i *Database.java* benyttet JDBC, Java Database Connectivity.

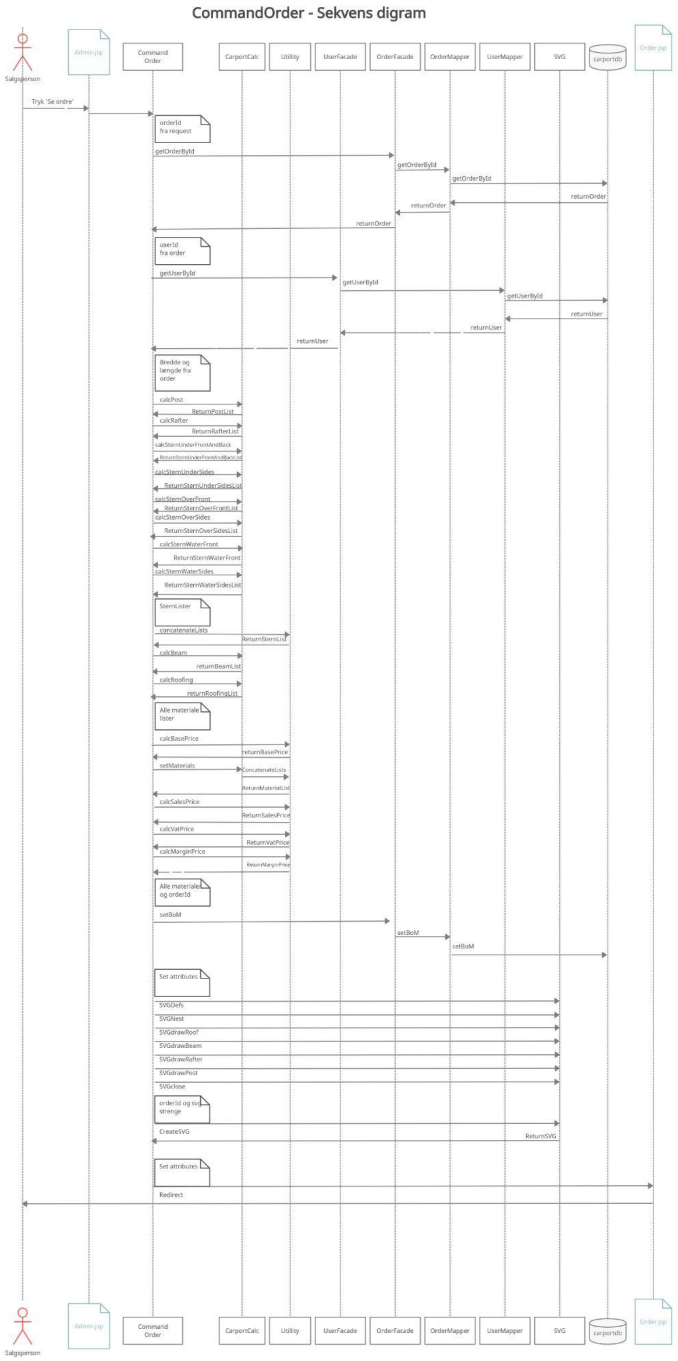
Login informationerne til databasen er injiceret i constructoren ved, at benytte *dependency injection* der gør det muligt, at skifte login informationerne, blandt andet når der testes.

Sekvensdiagrammer

Formålet med et sekvensdiagram er, at dokumentere og skabe overblik over vigtige dele af programmet. De følgende diagrammer viser et typisk flow.

CommandOrder

Fuld version, se bilag



Gennemgang af CommandOrder

Det antages, at salgspersonen er på oversigtssiden ved diagram start: *admin.jsp*.

Diagrammet viser flowet efter at der er blevet trykket på knappen, *se ordre*, ud for en specifik ordre til, at brugeren ender på ordre siden: *order.jsp*.

Kort gennemgang af de forskellige klasser der er i brug i denne sekvens:

Order

Order er en klasse der indeholder flere af de kritiske værdier for efterfølgende beregninger. Identifikation på kunden, samlede pris for ordren, højde og bredde på carport, såvel som redskabsrum og andre informative værdier.

Utillity

Utillity er en klasse der indeholder hjælpemetoder som bruges flere steder i systemet.

Material

Material er en klasse der kan tilskriver informationer om de enkelte materialer således, at der kan genereres en *Bill of Materials*.

Identifikationsnummer på materialet, beskrivelse, mål og pris. Pris samt mål benyttes til at beregne forbruget.

CarportCalc

CarportCalc er en klasse der indeholder flere beregningsmetoder til de forskellige materialer der benyttes til styklisten (BoM) og til tegning af SVG.

User

User er en klasse der bruges til at holde informationer om en given registreret bruger. Id, roleid og relevante kontaktinformationer.

BoM

BoM er en klasse der indeholder en array liste med materialer, totalprisen samt indkøbsrabat.

SVG

SVG er en klasse der benyttes til at generere tegning af en given carport. Der benyttes en *StringBuilder()* for, at sammensætte én lang *<svg>... </svg>*. SVG trækker blandt andet på værdierne fra Ordre klassen.

Commandens funktion

Commanden *CommandOrder* er klassen der kaldes når brugeren trykker på knappen: *se order*. Herefter sker der følgende:

1. *orderId* som er sendt med fra *admin.jsp* hentes fra *Requestlayer.*
2. Ordren hentes vha. *orderId* igennem *Ordefacade*
3. User hentes vha. *userId* igennem *Userfacade (*userId er holdt i orderen*)*
4. Der udregnes og hentes liste med stolper via *calcPost* metoden i CarportCalc
5. Der udregnes og hentes liste med Lægter via *calcRafter* metoden i CarportCalc
6. Der udregnes og hentes lister med de forskellige sternbrædder vha. metoder i CarportCalc
7. De mange Sternlister sættes sammen til én ved hjælp af *concatenateLists* i Utillity
8. Der udregnes og hentes liste med tagplader via calcRoof metoden i CarportCalc
9. En komplet *arrayliste* af materialer sammensættes vha. *concatenateLists* metoden i Utillity
10. Der udregnes indkøbspris ud fra materialelisten vha. *calcBasePrice*
11. Der udregnes salgsprisen ex. moms, ud fra materialelisten vha. *calcSalesPrice*
12. Der udregnes salgsprisen incl. moms, ud fra salgsprisen ex. moms *vha . calcVatPrice*
13. Dækningsbidraget udregnes vha. *calcMarginPrice*
14. Alle værdierne sættes som attributter og der genereres en BoM hvor disse tilskrives.
15. Der beregnes en SVG med værdier fra sessionslagets '*order*'. Metoder gør følgende:
    - 15.1. SVGDefs tilskriver de to definerede pile 'defArrowBegin' og 'defArrowEnd'.
    - 15.2. SVGNest bruges til nesting af SVG headeren.
    - 15.3. drawRoof beregner tegning af taget vha. værdier fra sessionslaget 'rooflist'.
    - 15.4. drawBeam beregner tegning af remme vha. værdier fra sessionslaget 'beamlist'.
    - 15.5. drawRafter beregner tegning af spær vha. gemte værdier fra sessionslaget 'rafterlist'.
    - 15.6. drawPost beregner tegning af stolper vha. gemte værdier fra sessionslaget 'postlist'.
    - 15.7. SVGClose tilskriver footeren og lukker den nestede SVG.
16. Den generede SVG streng skrives til databasen via orderfacaden
17. Til sidst returneres der til FrontControlleren som videresender til 'order.jsp'

Commanden benyttes kun af Salgspersonale/admin, men *order.jsp* håndterer data på flere måder, alt efter hvem der tilgår siden. dvs. kaldes order.jsp af en kunde så er det en anden Command der tilgås.

OrderFacadens funktion

OrderFacaden indhenter de relevante metoder og klassen fra de underliggende lag.

For at kunne håndtere ordermapperens funktioner instantieres denne med en database forbindelse.

I denne sammenhæng benyttes orderfacadens metode *getOrderById (int orderId)*

getOrderByID (int orderId) kalder ordermapperens *getOrderById* metode og sender et orderId med, metoden returnere derefter en instans af klassen *order* tilhørende det medsendte id.

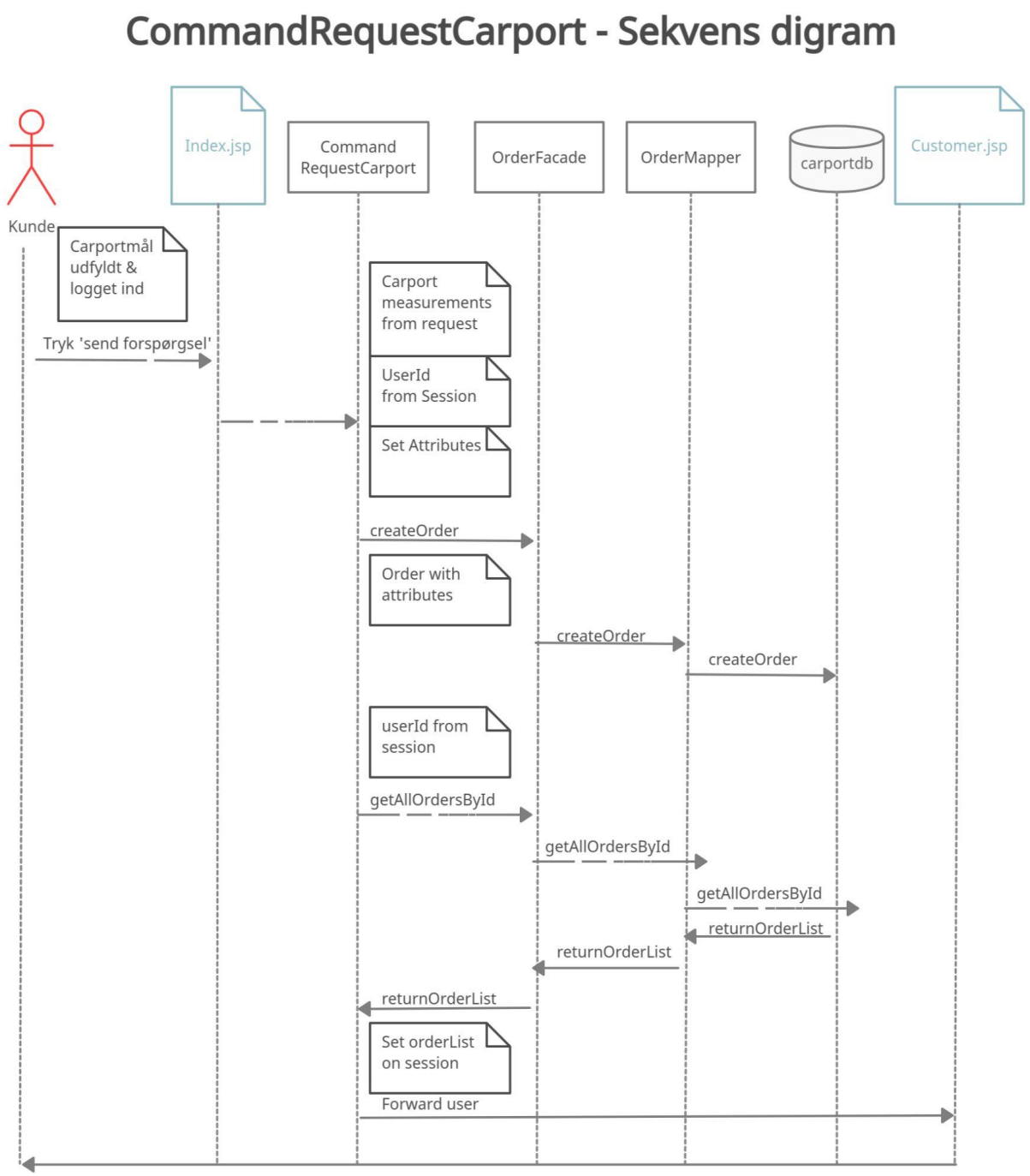
UserFacaden funktion

Userfacaden indhenter de relavante metoder og klasser fra de underliggende lag.
for at kunne håndtere usermapperens funktioner instantieres denne med en database forbindelse.

I denne sammenhæng benyttes userfacadens metode getUserById (int userId)
getUserById (int userId) kalder usermapperens getUserByID metode og sender et userid med, metoden returnerer derefter en instans af klassen *user* tilhørende det medsendte id.

CommandCarportRequest

Fuld version, se bilag



Gennemgang af CommandRequestCarport

Det antages at kunden er på forsiden ved diagram start. 'index.jsp', og er logget på systemet.

Diagrammet viser flowet efter der er trykket på knappen 'send forespørgsel', efter at have udfyldt de ønskede mål på carporten, til brugeren ender på kundesiden 'customer.jsp'.

Kort gennemgang af de forskellige klasser der er i brug i denne sekvens:

Order

Order er en klasse der indeholder flere af de kritiske værdier for efterfølgende beregninger. Identifikation på kunden, samlede pris for ordren, højde og bredde på carport såvel som redskabsrum, og andre informative værdier.

User

User er en klasse der bruges til at holde informationer om en given registreret bruger. Id, rolleid og relevante kontaktinformationer.

Commandens funktion

Commanden *CommandRequestCarport* er klassen der kaldes når brugeren trykker på knappen 'send forespørgsel' på index.jsp

Herefter sker der følgende.

1. *User* hentes fra *sessionslaget (sat da kunden loggede ind)*
2. Mål fra input formen, index.jsp, hentes fra requestlaget
3. En ny ordre med værdierne fra input formen samt *user objektet,* genereres.
4. Ordren sættes gennem orderfacaden til databasen.
5. En *arrayliste* kaldet 'orderlist' opdateres gennem ordrefacaden.
6. Til sidst returneres der til FrontControlleren som videresender til 'customer.jsp'

   Commanden benyttes kun af kunden, men order.jsp håndtere data på flere måder, alt efter hvem der tilgår siden. dvs. kaldes order.jsp af en salgsperson/admin så er det en anden Command der tilgås.

OrderFacadens funktion

OrderFacaden indhenter de relevante metoder og klassen fra de underliggende lag.

For at kunne håndtere ordermapperens funktioner instantieres denne med en database forbindelse.

I denne sammenhæng benyttes orderfacadens metoder *CreateOrder(Order order)* samt *getOrderById (int orderId)*

CreateOrder(Order order) kalder ordermapperens *createOrder* metode og sender en instans af klassen *order med. Metoden returnerer ikke noget.*

getOrderByID (int orderId) kalder ordermapperens *getOrderById* metode og sender et orderId med, metoden returnere derefter en instans af klassen *order* tilhørende det medsendte id.

OrderMapperens funktion

Mapperen står for at hente de data der persisteres i databasen. Mapperen indeholder de metoder der sørger for at kommunikere SQL-forespørgsler og oprette instanser af *Order* klassen.

Særlige forhold

Applikationslag

Når applikationen instantieres og front controllerens init() metode kaldes, initialiseres globale datastrukturer som sættes på applikationslaget. Disse kan derfor senere tilgås uden først at skulle tilgå de persisterede data fra databasen. det giver den fordel at man ikke hele tiden skal hente de samme værdier igen og igen. Det benyttes til data og værdier som må antages, ikke ændre sig i løbet af sessionen.

Vi henter og sætter arraylister over foruddefinerede mål på carport og skur til at populere dropdown-menuer på index.jsp.

til at parre et id med tilhørende værdi, sættes der HashMaps<integer, string> med værdier fra roller, order status, postnumre, enheder (stk. pk. rl. etc..).

Der sættes ligeledes HashMaps<integer, string> for beklædning og tagplader.

Således kan der altid trækkes et navn eller en beskrivelse ud fra et givent id, uden også at skulle i databasen efter dette, hver gang der refereres.

Når en bruger logger ind og bliver valideret instantieres der et User objekt som tilskrives værdierne på den enkelte bruge, samt dennes rolle id der benyttes gennem hele siden til at navigere de forskellige commands. Objektet skrives til sessionen. Således kan værdierne nemt invalideres ved log ud.

Sessionslag

I sessionslaget sætter vi de værdier der skal tilgås af sessionen flere steder. ved en eventuel refaktorering kunne man højst sandsynligt være lidt mere kritisk om hvad der ligger på sessionen og hvad man kunne nøjes med køre ind over *requestlaget*.

Når der beregnes en ordre/carport ud fra kundens mål, bliver der beregnet en masse *arraylister* med de forskellige materialer, også de beregnede priser sættes på sessionen.

Andet

Der er lavet validering på brugerens input flere steder.

Ét eksempel herunder, er valideringen på indtastning af email når der oprettes en bruger.

hvor vi sikre os, ud fra et *regex pattern* med karaktere, at brugeren indtaster en valid email.

For ikke at opfinde den dybe tallerken igen, er regex strengen kopieret fra en anden validator fundet på nettet. Der benyttes tilhørende Pattern og Matcher for at kontrollere at emailen kan valideres op imod vores regex streng.

```java
public static boolean validateEmailAddress(String email) {
    String regex = "^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@((\\[[0-9]{1,3}\\.[0-9]{1,3
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(email);
    return matcher.matches();
}
```

Der valideres også på input form til bestilling af carport hvor der er krav til at begge mål skal være udfyldt, og i input form til oprettelse af bruger. Her skal alle felter udfyldes. Ved oprettelse af brugeren kontrolleres der i databasen om mailen eksistere i forvejen, det har vi gjort simpelt, ved at sætte emailadressen til at være unik i databasen.

Der håndteres exceptions på de fleste metoder der kaster en fejl. der benyttes *UserException* som er med startkoden. Vi logger ikke fejlene, men der returneres til samme side som laver forespørgslen og brugeren modtager en fejlmeddelelse på siden, så det er tydeligt hvad fejlen består af.

Ekstraordinært er der benyttet en smule JavaScript til projektet, selvom dette ikke er en del af pensum. Der var et par funktioner vi syntes manglede, og som kunne lade sig gøre ved at implementere JavaScript. Når man vælger mål på forsiden, og bagefter logger ind, ville man normalt skulle vælge sine mål igen, men her benytter clientsiden til at opbevare de valgte data. Så slipper vi at persistere data i applikationen.

Ved login gemmes data ved at kalde *storedata* funktionen

```javascript
function storedata() {
    if (typeof (Storage) !== "undefined") {
        localStorage.setItem("carportwidth", document.getElementById( elementId: "carportwidth").value);
        localStorage.setItem("carportlength", document.getElementById( elementId: "carportlength").value);
        localStorage.setItem("shedwidth", document.getElementById( elementId: "shedWidth").value);
        localStorage.setItem("shedlength", document.getElementById( elementId: "shedLength").value);
    }
}
```

Ved indlæsning af siden kaldes denne funktion

```javascript
window.onload = function () {
    if ((localStorage.getItem( key: "carportwidth")) !== null) {
        document.getElementById( elementId: "carportwidth").value = localStorage.getItem( key: "carportwidth");
        document.getElementById( elementId: "carportlength").value = localStorage.getItem( key: "carportlength");
        document.getElementById( elementId: "shedWidth").value = localStorage.getItem( key: "shedwidth");
        document.getElementById( elementId: "shedLength").value = localStorage.getItem( key: "shedlength");
    }
}
```

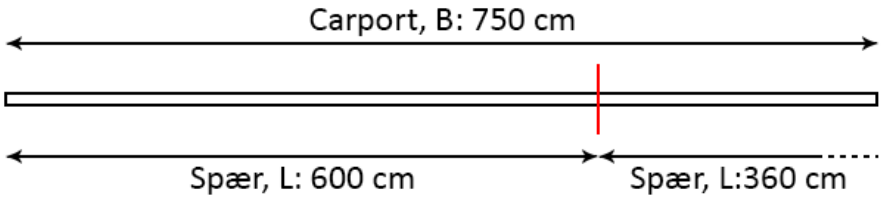
Når brugeren bekræfter og indsender formen, kaldes den indbyggede *localStorages.clear()* funktion og klientens localStorage tømmes.

Udvalgte kodeeksempler

Beregning af spær

Udvælgelsen for at fremhæve netop denne metode bygger på, at den har været byggesten for calcStern og calcRoof metoderne. Selvom metoden er den simpleste af varianterne, så giver den et fint indtryk af, hvordan de andre metoder fungerer.

Kort fortalt indlæses, fra databasen, det ønskede materiales fundne længder. De bruges til at fastsætte de rette længder, der skal bruges baseret på carportens mål. Metoden tager højde for, i tilfælde at en carport er bredere end længden på det længste materiale, at tilføje den rest der måtte være tilbage med den mindste mulige materiale længde.



Illustrationen her viser den størst- og mindst mulige spær længde på hhv. 600 cm og 360 cm, som samlet vil skulle bruges for at imødekommende carportens mål.

```
95
96      // Spær
97      public List<Material> calcRafter(int carportWidth, int carportLength) throws UserException {
98
99          // Get materials from database
100         String description = "Spær, monteres på rem";
101         List<Material> materialList = materialFacade.getMaterialByDescription(description);
102
103         // Create list with available lengths
104         List<Integer> availableLengths = new ArrayList<>();
105         for (Material material : materialList) {
106             availableLengths.add(material.getLength());
107         }
108
109         // Calculate
110         int maxWidth = 550;
111         int quantity = (int) ceil((double) carportLength / (double) maxWidth);
112
113         // Add the right lengths
114         List<Material> result = new ArrayList<>();
115         int length = carportWidth;
116         for (int i = availableLengths.size() - 1; i > 0; i--) {
117             if ((length) >= availableLengths.get(i)) {
118                 result.add(newItem(quantity, materialList.get(i).getMaterialID(), materialList.get(i)));
119                 length -= availableLengths.get(i);
120             }
121         }
122
123         // Minimum length
124         if (length > 0) {
125             result.add(newItem(quantity, materialList.get(0).getMaterialID(), materialList.get(0)));
126         }
127         return result;
128     }
129
```

Spær beregneren, calcRafter, kaldes med målene for carportens bredde og længde. Metoden vil returnere en liste af de beregnede materialer og inkludere et antal for hver fundne længde.

Linie 100-101: Der instantieres en liste med materialer, via facade/mapper, som matcher beskrivelsen fra den deklarerede streng; description.

Linie 101-167: Der instantieres endnu en liste til at indeholde længderne på det fundne materiale. Vha. ForEach-løkken gemmes længderne i denne liste.

Linie 110-110: Der sættes en max bredde mellem spærene, og antallet af spær beregnes.

Linie 114-115: Ny liste instantieres og skal bruges til at indeholde de værdier der skal returneres for metoden. Carportens bredde gemmes i en variabel, som skal bruges til at trække spær længderne fra når der itereres i for-løkken.

Linie 116-121: For-løkken løber antallet af fundne spær længder igennem, startende med den største længde. Hvis spær længden er mindre end carportens mål, tilskrives antal, materialets id og materialet listen:result, og målet fratrækkes variablen indeholdende carportens oprindelige bredde mål. Dette trin gentages indtil listen med fundne længder er løbet igennem.

Linie 124-126: I tilfælde af, at længderne ikke går op i for-løkken, vil der her blive tilskrevet den mindste mulige spær længde til listen.

Linie: 127: De beregnede værdier returneres her indeholdende de gemte data: antal, materiales ID og selve materiale.

Når der skal tilskrives til listen:result, bruger vi hjælpe metoden; newItem. Metoden blev til ved refaktorering, og visningen af den her i afsnittet har til formål at understøtte forklaringen af spær beregneren.

```
18
19      //<editor-fold desc="New Material">
20  @   private Material newItem(int quantity, int materialID, Material material) {
21          return new Material(materialID,
22                  material.getName(),
23                  material.getDescription(),
24                  material.getPrice(),
25                  material.getUnitId(),
26                  material.getWidth(),
27                  material.getLength(),
28                  material.getHeight(),
29                  quantity);
30      }
31      //</editor-fold>
32
```

Kort fortalt instantierer metoden et nyt materiale. Vi tilfører som parameter et givent antal af den beregnede materiale mængde, selve materiale ID'et, samt det valgte materiale fra listen.

Den oprindelige udgave af spær beregneren tilskrev kun antallet af materialer, samt materialets id. Gruppen besluttede senere også at tilskrive selve materialet til listen, da vi alligevel havde fat i databasen og på den måde kunne bruge resultat direkte i metoderne i CommandOrder-klassen.

Beslutningen er bl.a. baseret på den tilbageværende tid, prioritering af opgaver, antal databasekald, tilføjelse af facade/mapper.

Ved eftertanke burde man have sat beregneren op, så metoden kunne lave en mere ligeligt fordelt længde på spærene. Det ville gøres ved at sige, at når bredden på carporten var større end det længste spær, så del længden på spæret for carportens fulde bredde i to, og match dem ud fra de tilgængelig spær længder. Det ville også fungere ift. konstruktionen af carporten fordi, når bredden overstiger en vis størrelser, så indregnes der et ekstra sæt stolper og remme.

Sammenkædning af lister

```
42
43      @SafeVarargs
44  @   public static List<Material> concatenateLists(List<Material>... materials) {
45          List<Material> result = new ArrayList<>();
46          Stream.of(materials).forEach(result::addAll);
47          return result;
48      }
49
```

Med metoden, *concatenateLists*, kan du tage en række lister og sammenkæde dem i én enkelt liste.

Det er i sig selv ikke banebrydende. Det vi finder rigtig interessant er måden Stream fungerer på. Kort fortalt er det et hurtigt alternativ for, at skulle skrive sin egen multithread løsning over et array, der umiddelbart uden threading, vil skabe flaskehals ved håndtering af et større antal lister. Man vil også spare en masse linier kode, da hver enkelt ForEach løkke er inkluderet i kaldet når Stream afvikles.

Linie 45: Der instantieres en tom liste.

Linie 46: For hvert element Stream har modtaget, pakkes det givne indhold ud og tilskrives den nye liste.

Linie: 47: Her returneres den færdige liste med den samlede data af de tilskrevne lister.

Vi finder det også sjovt, for måden man angiver antallet af lister for værende ukendt på vha. tre punktummer.

Annotationen @safeVarargs bruges kun til at fjerne den advarsel, hvor man gøres opmærksom på, at "forkert indhold" fra de tilførte lister kan forårsage utilsigtet resultater. Der står sågar i dokumentationen for denne annotation "@safeVarargs // Not actually safe!".

Status på implementering

Selvom de fleste kernefunktioner virker, har det ikke været muligt at implementere alle kravene inden for projektets tidsramme.

Opnået systemfunktionalitet

Følgende krav er opfyldt:

- Ansatte kan logge ind og ud
- Ansatte kan beregne et tilbud (delvis)

  Denne story er opdelt i 5 punkter, den del der har med carport beregningen at gøre er gennemført, så der kan oprettes en ordre og afgives tilbud hvor carportens mål indgår i beregning af træ, tag og priser herpå.

- Ansatte kan modificere et tilbud
- Ansatte kan slette et tilbud
- Ansatte kan oprette en ordre ud fra et tilbud
- Ansatte kan generere en tegning af carporten ud fra mål
- Ansatte kan generere en stykliste
- Kunder kan logge ind og ud
- Kunder kan oprette en konto
- Kunder kan sende en forespørgsel på en carport efter mål
- Kunder kan modtage og acceptere et tilbud
- Kunder kan afvise og annullere et tilbud
- Kunder kan betale en ordre
- Kunder kan se en stykliste
- Kunder kan se en tegning af carporten

Kundens brug af systemet

Forsiden af vores webapplikation indeholder bestillingsformularen, som kunden udfylder for at sende sin forespørgsel. Kunden kan vælge at oprette en ny konto eller logge ind i en eksisterende for at sende en forespørgsel. Når en kunde er logget ind kan de se deres igangværende ordre, findes ingen ordrer bliver der gjort opmærksom på dette og henvist til forsiden. På ordre oversigten kan der trykkes ind på de enkelte ordre hvor informationer om ordren kan ses, det er her pris og accept (eller annullering) af ordren foregår. Der benyttes samme side til at vise forskellig information og knapper til henholdsvis accept, annullering og betaling. Til sidst kan tegning af carport samt stykliste ses af kunden.

Medarbejderes brug af systemet

når en medarbejder er logget ind i systemet har de adgang til at se alle ikke-afsluttede ordre, og klikke ind på denne. på ordre siden er der mulighed for at beregne et tilbud og ændre i mål og samlede tilbudspris. Hvis ordren er en forespørgsel beregnes tilbud, er ordren et tilbud og denne er accepteret. kan der ikke længere ændres i attributterne for ordren. Som kunden har medarbejderen også adgang til en stykliste samt en tegning af carporten, men allerede fra forespørgslen.

Ikke-opnået systemfunktionalitet

Følgende krav er ikke opfyldt:

- Ansatte kan beregne et tilbud (delvis)
    - Denne story er opdelt i 5 punkter, den del der har med redskabsskuret at gøre, er sprunget over, da det blev vurderet af det ville presse tiden.
    - ligeledes er befæstigelse heller ikke indbygget

- Ansatte kan opdatere materialerne i databasen
    - Denne story blev nedprioriteret både af PO til planlægningsmøder og internt. Da allerede eksisterende funktionalitet illustrere fremgangsmåden for en sådan funktion.

Kundens brug af systemet

- alle dele af systemet er implementeret.

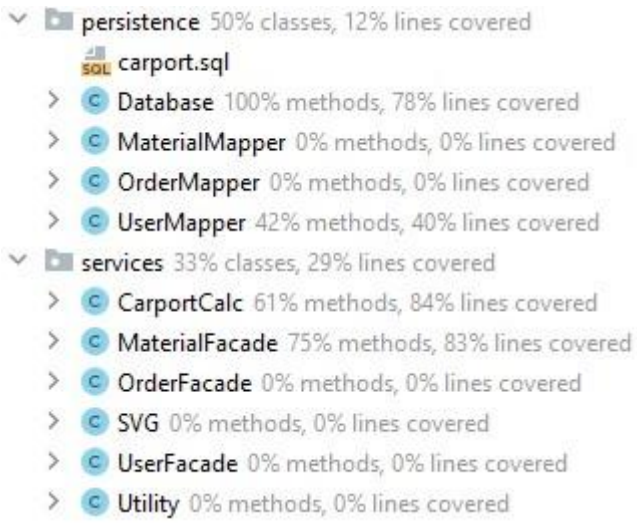
Medarbejderes brug af systemet

Det er ikke muligt for medarbejderen at medtage redskabsskurets mål i beregningen. Der sker derfor ingen ændring, også selvom der er mål på en evt redskabsskur.

Det er ikke muligt for medarbejderen at beregne beslag og befæstigelse.

Test

Unit- og integrationstest har kun været brugt i begrænset omfang ved udarbejdelsen af dette projekt. Der har som sådan ikke været en primær årsag til fravalget. Det har nok mest handlet om, at vi i vores planlægning af forløbet, ikke har haft tænkt på at inkludere den nødvendige tid til opsætning, samt brug, fordi det endnu ikke er blevet en fast rutine i egne workflows.



Som det ses på billedet er coverage ikke fyldestgørende, og det er noget vi som gruppe vil tage til efterretning.

Efter den indledende projekt uge modtog vi en kort introduktion til, hvordan vi med hjælp fra unittest kunne opbygge og teste vores metoder. Det lod os dog inspirere lidt til, at afprøve denne fremgangsmåde til beregningsmetoderne i CarportCalc klassen. Der er opstået et par problemstillinger undervejs, og det er forløbet af disse vi kort vil beskrive om lidt.

Det skal nævnes, at vi har fået udleveret en byggevejledning på en af Fog's byg selv carporte, og den er brugt som facit for, hvordan vi har kunne sikre os at finde de rigtige antal og længder af de forskellige materialer.

Det skal yderligere nævnes, at der ved 2. møde med product-owner blev stillet krav til, at vores oprindelig user story; Offer, Create, som indirekte indeholdt beregner, skulle opdeles. Vi endte med lave opdelingen i 5 mindre user stories; carport (træ), tag, redskabsrum (træ), beklædning og befæstigelse, hvoraf det kun er de to førstnævnte der er lavet beregninger på. De er herefter brudt ned i mindre opgaver, således at der skulle være en beregner metode for; stolper, remme, spær, stern, samt tag. Dvs. beregninger nok til, at kunne generere en tegning, samt materialeliste.

Da beregningsmetoderne i CarportCalc er varianter af hinanden, vil vi her fokusere på beregneren for spær, da den færdige metode kan refereres til i afsnittet, udvalgte kodeeksempler.

Undervisningen var som sagt inspirerende. Tilgangen til unittest blev introduceret omvendt ift. hvad vi på nuværende har dannet vane for. Det skal forstås på den måde, at man først skriver en simpel testmetode som giver det ønskede resultat, og herefter opbyggede så sin metode, i parallel med testen.

```
23
24      @Test
25      void calcRafter() {
26          assertEquals( expected: 15, carportCalc.calcRafter( carportLength: 780));
27      }
28
29      @Test
30      void calcRoofing() {
31          List<Integer> arrayList = new ArrayList<>();
32          arrayList.add(6);
33          arrayList.add(600);
34          arrayList.add(360);
35          assertEquals(arrayList, carportCalc.calcRoofing( carportWidth: 600,  carportLength: 780));
36      }
37
```

Her ses et par af de første testmetoder. Bemærk at, der på spær beregneren, calcRafter, kun testes på antallet af spær og endnu ikke på længden.

Indtil det tidspunkt, hvor der blev dannet grundlag for, hvordan længden skulle beregnes ud af en liste af mulige længder, havde der endnu ikke været nogle nævneværdige problemer. Det havde faktisk været en stor hjælp at arbejde i parallel, mellem metoden og unittesten.

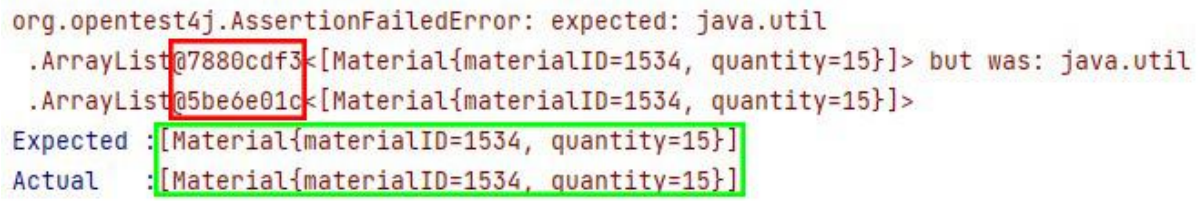
Da databasen skulle integreres for, at erstatte den fast kodede liste over materialets længder, endte det med at danne flaskehals for dagens igangværende arbejde.

Problemet skulle vise sig kun at gøre sig gældende for test-klassen på grund af en manglende instantiering af database i denne. Og kombineret med at have lænet os tæt op af en ellers fint fungerende unittest, fik vi set os blindt på, hvornår og hvordan problemet var opstået. Yderligere skal det nævnes, at forvirringen kun blev større, fordi den integrerede metode viste sig at fungere efter hensigten, netop fordi database i CarportCalc var korrekt instantieret.

Her ses seneste bud på unittesten. Problemet her, som stadig gør sig gældende er, at vi sammenligner to lister. Forklaring følger...

```
49
50      @Test
51      void calcRafter() throws UserException {
52          Material material = new Material();
53          List<Material> result = new ArrayList<>();
54          result.add(carportCalc.newItem( quantity: 15,  materialID: 1534, material));
55          assertEquals(result, carportCalc.calcRafter( carportWidth: 6000,  carportLength: 7800));
56      }
57
```

På næste billede kan vi se sammenligningen af selve testen. Resultatet er ens, men vi får stadig en fejl, da det er listens referencer som sammenlignes. Det giver fint mening, da vi ved at lister gemmes med unikke referencer til hukommelsen. Selvom vi ikke har fundet en løsning på denne fejl, har testen dog stadig været brugbar, netop fordi at resultatet er korrekt.



En mulig løsning vi har gjort os overvejelser om, kunne være at gemme listerne ned i hver deres streng variabel, og så sammenligne de to strenge i stedet. Vi har valgt for nu, at lade det ligge pga. anden prioritering.

Proces

Arbejdsprocessen faktuelt

Som del af SCRUM, er processen opdelt i sprint, som i dette projektforløb var sat til ca. 1 uges varighed. Hvilket under normale forhold ikke vil være hensigtsmæssigt, men i denne situation fungerede det udemærket.

Vi har i gruppen valgt ikke benytte os af en SCRUM Master, grundet gruppens størrelse. Vi har et godt samarbejde og mødes i fællesskab hver dag om arbejdet. Vi havde ligeledes rigtig fine P.O. møder, før hvert møde med P.O., afholdte vi i gruppen et internt møde for at afstemme og diskutere hvilke stories der evt. skulle fremhæves for P.O.

Grundet vores helt tætte samarbejde og gode forståelse for hinanden har det været en mulighed, ikke at afholde skemalagte Scrum møder, det er kommet helt naturligt ved dagens begyndelse og slutning. Efter hvert sprint er der også reflekteret over processen og diskuteret om der skulle ændres på arbejdsmetoder og former.

Sprint 1:

Til det første møde med PO blev det aftalt at der skulle prioriteres user story *Carport, Request.* Desuden skulle der arbejdes på at beregne estimater på de enkelte stories, så det blev lettere for P.O. at overskue opgaverne i forhold til hvad hans ønske var.

Fra vores backlog journal:

04/05/2021	1. møde med PO (Kim) om US/AC. Kort; Prioritér "Tailor Carport Form". Husk estimater <- Meget vigtigt. Revidering af US/AC. Påbegyndt UseCaseDiagram & DomainModel. Gruppen har talt om hvorvidt der skulle udarbejdes et skema for navngivning af variabler o. lign. - Et såkaldt nameconvention sheet. Beslutning var at tilrettelægge løbende for at tilpasse den eksisterende startkode.
05/05/2021	Opsætning af database ud fra konstrueret ER Diagram. Vi har snakket om, om vi på nuværende skulle inkl. tid til design, men venter indtil vi har en solid backend. Mockup laves i AdobeXD.
06/05/2021	1. møde med Nikolaj vedr. ER diagram og domænemodel. Det blev foreslået at fokusere på aktivitetsdiagram, samt navigationsdiagram. Vi har arbejdet videre med dokumentationen inkl. de foreslåede diagrammer. Der er arbejdet med at tilpasse "Startkoden" til projektet. Afsluttede US; Login, Logout, Signup
07/05/2021	Der er lagt en plan for dagen med henblik på at nå ugens sprint. Thomas har fokuseret på InputForm, samt orderMappe. Peter har refaktoreret det meste af den foreløbige kode på baggrund af tidligere erfaring med startkoden. Her har der været fokus på at bruge hjælpemetoder, hvor muligt, samt skift fra List til Map/Hashmap. Databasen er opdateret med diverse smårettelser; manglende AI og default values, 3NF etc.
08/05/2021	Thomas har lagt sidste hånd på påbegyndt arbejde vedr. inputform, nu kan der gennemføres en ordre fra inputform til database.
09/05/2021	Der er holdt kort møde vedr. problemstilling om ordrehåndtering og login. Inputform refaktoreret for at reflektere beslutninger vedr. orderhåndtering og login. Data indtastet i form før login,

	gemmes lokalt i clienten og hentets efter fuldendt login. (JS)
10/05/2021	Vi har tilføjet fejlhåndtering til inputform for at forbigå problemer ift. manglende input valg. Der skrives til brugeren ved både fejl og succes på siden. Funktion til at liste alle ordrer for både kunde og salgsmedarbejder tilføjet. Funktion til at liste valgt ordre for både kunder og salgsmedarbejder tilføjet. Tilføjelse af opgaver til kanban.

Sprint 2:

I det andet sprint blev det besluttet at fokusere på at kunne beregne et tilbud, stories "Offer, Create".

Som det kan læses ud fra journalen herunder, så blev det diskuteret om casen skulle opdeles for at gøre den lidt mere håndgribelig. Vi endte med at opdele den i 3 dele. (senere 5)

Fra vores backlog journal:

11/05/2021	2. møde PO (Kim). Det blev besluttet at omskrive 'Offer, Create' for at gøre casen mere håndterbart. Umiddelbart er overvejelsen at opdele casen i 3 dele baseret på beregninger udfra de enkelte typer; træ, beklædning/tag, befæstigelse. Ugens sprint er at få udregningerne for carporten implementeret. Der blev undervist i startkodens arkitektur, samt unittest af Nikolaj.
12/05/2021	Der blev introduceret vektortegning af Jon. De tilhørende SVG opgaver blev påbegyndt. Opdelt userstory; Offer, Create i 4 dele; Træ, Beklædning, Tag, Befæstigelse
13/05/2021	Dagen er brugt på SVG opgaveløsning, samt gennemgang af Jons video om udregninger. Tilføjet to nye User Stories: Carport, Draw; Tegning af carport ud fra valgte mål på forespørgsel. Funktion til udregning af Træ (pæle, remme, spær), samt Tag er tilføjet koden. Der er også tilføjet unittest til metoderne.
14/05/2021	Gruppen har holdt møde vedr. ugens sprint. For at imødekomme PO, har vi besluttet at ændre/tilføje på User Stories, så beregneren i stedet opdeles på carport, tag, skur, beklædning, befæstigelse. De tre sidstnævnte tages ud for nu, for at bruge tiden på at få de sidste beregningsmetoder helt på plads, inkl. håndtering af databasen-delen. Funktion til udregning af Træ (stern) tilføjet koden. Der er også tilføjet unittest til metoden.
17/05/2021	Dagen er startet med kort gruppemøde, hvor vi diskuterede dagens agenda, samt sidste uges forløb. Databasen er blevet opdateret med flere varianter af materialerne, samt priser for at imødekomme afviklingen af beregneren mm. Der har netop været stor fokus på beregneren (ugens sprint), som har voldt en del problemer; Bl.a. manglende/korrekt instantiering af database, hvilket også inkl. en logisk fejl unittest (nullpointer/database). Der blev oprettet ny mapper og klasse; MaterialMapper og Material, samt ny unit mapper til håndtering af enheder fra databasen (3NF). Vi har igennem dagen tilpasset beregneren løbende efter behov. Der er tilføjet/påbegyndt adminstrator panel med funktioner; beregn, rediger, slet. I processen er der lavet mappere, facade og commands.

Sprint 3:

Til vores tredje møde med PO gennemgik vi ugens opgaver og diskuterede de problemstillinger der var opstået. Der blev aftalt at arbejde med vores SVG beregning og tegning, herefter dækningsgrad på tilbud og justering af denne, Bill of Materials, ændringer til varekataloget, beregning af skur, fejlhåndtering mm.

Fra vores backlog journal:

18/05/2021	3. møde PO (Kim). På mødet gennemgik vi ugens sprint med UserStories og de antagelser/problemstillinger, der har været for beregneren. Forkert implementering af database/unittest havde været en stor flaskehals indtil løsningen var fundet. Vi snakkede også om refaktorering for at sikre en hurtigere håndtering af nye beregninsmetoder til bl.a. skur og befæstigelse. Vi gennemgik websitet med PO om den nuværende funktionelitet. Stor ros fra PO, om at vi havde delt Order, Create op i 3 dele som aftalt. Ved løbende antagelser opdelte vi samme userstories ydeligere til 5 dele for få et bedre overblik, men også bedre eksekvering af opgaverne. Ved planning meeting blev ugens sprint aftalt med følgende prioritering; SVG, Dækningsgrad, BOM, ændringer af lagerførte vare, beregning af skur. Fejlhåndtering af Login/Signup og rettelse af tekster mm. Gennemgang af koden med hensigt på at rette slåfejl o.lign. (QoL=Quality of Life). Påbegyndt SVG generering, opsætning af beregningsmetoder til materialerne ud fra carport mål.
19/05/2021	Videre arbejde med SVG generering og beregningsmetoder. En del besværligheder undervejs, så der gennemgås fremgangs- og beregningsmetoder. Der kan på nuværende genereres tegninger indenfor standart mål. Positionering af stolper, spær, remme. Der er tilføjet nyt layout for ordre specifikation til kunder, samt skrevet funktioner ind til at ændre status på en given ordre. Status tabellen i databasen blev ændret med ny tilføjelse af status, da vi manglede et "led". Dele af SVG beregningerne er løbende omskrevet for at simplificere processen. Vi har brugt alt for meget tid på SVG...
20/05/2021	Fortsat arbejde med SVG. De implementerede udregninger tegner og skalerer nu rigtigt. Tegningen er også dynamisk ift. antal material og deres størrelser. En antagelse ved SVG forløbet er at den dynamiske del godt kunne simplificeres, da udgangspunktets mål blev sat i procenter. Det har vist sig at besværliggøre udregningerne. Til næste gang vil vi prøve at starte med en målfast tegning, og dernæst "neste" den i endnu en SVG header, hvor denne er sat i procenter. Ved en simple test, viste det sig nemlig at det var en funktionsdygtig mulighed. Tilrettelse af layout for admin oversigten så den matcher kunde oversigten. Der er tilføjet ny mapper til at hente brugere baseret på role_id. Det er for at udelukke at blande medarbejdere og brugere sammen. Mapperen er skrevet til admin oversigten til at hente brugernes navne, telefon mm. ud fra de listede ordre.

21/05/2021	Der er tilføjet funktion for at gå til betaling når kundens ordrestatus er sat til Faktura. Der er opsat layout til checkout siden, samt tilføjet den nødvendige funktionalitet for at kunne skifte ordrestatus ved betaling og samtidig blive videresendt til betalingsbekræftigelsessiden. Der er tilføjet SVG beregninger til at inkl. tegnede linier med. De bruges til at flugte med fra pilene til de forskellige materialer. SVG er stadig tungt at danse med. Man skal holde tungen lige i munden. Der er opsat funktioner til salgspersonen, så der kan laves genereres et tilbud og hæftes en salgspris til ordren. Der beregnes samtidig nettopris og dækningsgrad. indkøbspris er beregnet ud fra 80% avance.
23/05/2021	Omskrivning af ordreberegning for henholdsvis kunde og salesperson. BoM og SVG tilskrives databasen og hentet direkte derfra når kunden har muligheden. der laves ikke længere beregninger når kunden tilgår en ordre. SVG gemmes som string der splittes ved indhentning. skal sættes ordentligt op.
25/05/2021	Fejrettet kode så tegning opdaterer korrekt når mål på carport ordre ændres. Vi har startet på rapportskrivning.
26/05/2021	Fortsat rapportskrivning. Vi har holdt statusmøde vedr. rapportskrivning. Refaktorering af calcPost.
27/05/2021	Sidste mæde med PO. Vi gennemgik backlog, og gav en demo af hjemmesiden. Det blev fremhævet at ændre ,- da det er forkert brug af komma og streg, og så er det oveni købet ulovligt. De eksisterende unittest / integrationstest er gennemgået og tilrettet. Dette er gjort med tanke på rapporten. Fjerne komma streg fra pris på order.jsp. Kim siger det er ulovligt. :)

Arbejdsprocessen reflekteret

Vi har i gruppen valgt ikke benytte os af en SCRUM Master, grundet gruppens størrelse. Dette har virket rigtig fint, da vi har haft en god arbejdsmoral og har været rigtig gode til hele tiden at forventningsafstemme vores proces, samt give plads til at reflektere over dagens/ugens arbejde.

Vores retrospektive møder lagde for det meste op til diskussion om fremgangsmåderne og måden arbejdet blev struktureret, og det er vigtigt, også selvom man føler man har godt fat i en god process.

Som tidligere, hvor gruppen har samarbejdet, gives der plads til meninger og holdninger. Vi vender og drejer situationer og fremgangsmåder for både kode og den agile process.

Da gruppen har samarbejdserfaring fra tidligere projekter, har vi et yderst effektivt flow og vi føler selv at vi er i stand til at holde produktivitetsniveauet oppe i hele perioden.

Vi gør meget ud af være undersøgende og nysgerrige på egen såvel som modpartens kode, og giver plads til forslag til ændring eller gennemgang af nye og spændende muligheder.

Et åbent miljø hvor en spade bliver kaldt for en spade giver i sidste ende også, et ærligt produkt.

Noget af det vi diskuterede mest, var opdelingen af user storien “offer create”. Efter praktisk erfaring og herefter yderligere diskussion, blev den delt fra 3 til 5 dele.

Opdelingen var vigtig for processen. Det gjorde os istand til nemmere at overskue opdelingen af arbejdsopgaver og vigtigst af alt, estimeringen. Dette gjorde at vi nedprioriterede den del af beregningen der havde med redskabsskuret at gøre.

Til gengæld blev den del af koden gennemarbejdet i en sådan grad, at det efterfølgende havde været overskueligt at implementere en beregning af redskabsskuret. Samtidig føler vi også at vores kode tydeligt giver udtryk for at vi er stand til at udarbejde og udvikle en beregningsalgoritme.