

Fog Projekt Rapport

Gruppe: CRMD

MICHAEL DUE P., CPH-MP447@CPHBUSINESS.DK, MICH561D
J. CHRISTIAN RYGE, CPH-JR221@CPHBUSINESS.DK, RANGERRYGE
GITHUB LINK: [HTTPS://GITHUB.COM/MICH561D/PROJECTFOG](https://github.com/MICH561D/PROJECTFOG)
WEBLINK: [HTTP://104.248.29.81/PROJECTFOG-1.0-SNAPSHOT/](http://104.248.29.81/PROJECTFOG-1.0-SNAPSHOT/)

Indholdsfortegnelse

Indledning	3
Baggrund	3
Valg af teknologi.....	4
Krav	4
Overordnet beskrivelse af virksomheden	4
Arbejdsgange der skal IT-støttes	5
AS-IS	5
TO-BE	7
Scrum userstories	9
Modeller og diagrammer	16
Domæne model.....	16
Forklaring til domæne model	17
Begrundelse for vores valg af 1-1 relation mellem order og carport.....	17
Begrundelse for vores valg af 1-1 relation mellem customer og address.....	18
Begrundelse for vores valg af 1..*-1..* relation mellem storage/shop og employee	18
Konsistentitet.....	19
EER diagram.....	19
Normalformer	20
Autogenereret id	21
Fremmednøgler og begrænsninger	21
Forbedringer	22
Klasse diagram.....	23
Navigationsdiagram.....	25
Sekvens diagram.....	27
Særlige forhold.....	28
Session	28
Brugerinput validering.....	30
Sikkerhed	31
Prepared Statement	31
Hashing	31
Salt	32

Brugertyper	32
SVG	32
Stykliste	33
Arkitektur.....	33
Exceptions	34
Logging.....	35
Udvalgte kodeeksempler	35
Hash password	35
Util	37
Stykliste	37
Carport udregning	38
Status på implementation.....	39
Test.....	40
Master branch	41
Proces	42
Arbejdsprocessen faktisk	42
Sprints og userstories	42
SCRUM Master.....	42
PO møder	42
Daglige SCRUM-møder	42
Retrospektives	43
Arbejdsprocessen reflekteret.....	43
SCRUM Master reflekteret	43
Retrospektive møder	43
Userstories	43
Estimeringer.....	43
Vejledning og PO møder	43
Produktivitet	44
SCRUM generelt.....	44
Bilag.....	45
Bilag 1 – Klasse diagram: Webpages	45
Bilag 2 – Klasse diagram: Præsentations-lag.....	45

Bilag 3 – Klasse diagram: Carport udregnings system	46
Bilag 4 – Klasse diagram: Database lag	47
Bilag 5 – Nuværende stykliste	48
Bilag 6 – Sprints	48

Indledning

Fog Træhandel har siden 1990'erne brugt et IT-system udviklet af en tidligere medarbejder til at beregne mål på carporte og styklister hertil. Systemet har tidligere været frakoblet deres hjemmeside og kørt på en gammel maskine, og på grund af dette kan prissatte produkter ikke ændres. Dette har medført at den totale pris har været ubrugelig i mange år, hvilket gør at medarbejderen skal udføre ekstra arbejde ved at finde de korrekte produkter.

Fog har derfor brug for et nyt IT-system, der er bedre integreret på deres nuværende webside og som er mere fleksibelt i forhold til prisstigninger og ordrestyring.

I indeværende projekt har vi udarbejdet en hjemmeside for Johannes Fog, hvorpå der kan bestilles en konfigurerbar carport efter en brugers specifikke valg samt ønsker til materialer og udformning. Før at kunne bestille skal brugeren oprettes i databasen, hvorfor vi har implementeret mulighed for registrering på siden. Efter oprettelse kan der logges ind på siden, hvorefter brugeren kan se tidligere ordre og ændre deres oplysninger.

Vi har også oprettet en administrator side, hvori der bl.a. kan ændres i produkters data, pris, beskrivelse. Ydermere skal en administrator kunne oprette nye administratorer og behandle brugers ordre.

Baggrund

Johannes Fog ønsker et system der kan udregne materialer til en carport efter deres kunders specifikke og unikke ønsker. I systemet skal de kunne vælge længde, bredde og højde, samt hvilken type tag de ønsker og til slut om de ønsker et redskabsskur integreret i deres nye carport. Her skal systemet kunne generere en stykliste, hvorved kunden hurtigt får præsenteret en samlet pris, og der nemt og hurtigt kan pakkes fra lageret. Johannes Fog ønsker at deres kunder kan logge ind på en personlig side, hvor kunden har overblik over tidligere køb og der kan ændres i kundens

oplysninger. Der skal også være en administrator-del, hvorfra Johannes Fogs ansatte kan opdatere produkter inkl. priser, størrelser, producenter m.v., samt tilføje og slette produkter. En administrator skal også kunne behandle en kundes ordre.

Valg af teknologi

Vi har valgt at skrive programmet i Java. Derfor faldt valget på Netbeans, idet programmet effektivt håndterer de valgte programmeringssprog og understøtter Apache Tomcat, hvilket er et godt serverværktøj. Til opbygningen af databasen har vi valgt MySQL, da deres Workbench er et godt udviklet SQL-program.

Programmer:

- Netbeans 8.2
- MySQL Workbench 8.0
- Java & Java SE (JDBC) 181
- Apache Tomcat 8.0.27.0

Sprog:

- Java
- HTML
- CSS
- MySQL 14.14

Vi har også brugt front-end framework: Bootstrap 4.1.0. Derudover brugte vi Code Coverage (TikiOne JaCoCoverage v. 1.5.3.2).

Krav

Johannes Fog har fremsat følgende krav vedrørende websiden:

- Kunden skal kunne bestille en carport med angivne mål og kontaktoplysninger.
- Det skal være muligt at tilpasse priser på produkterne.
- En kunde skal først modtage stykliste når carporten er købt og betalt, da dette er en del af Fogs servicedesign.
- Det skal være muligt for en medarbejder at ekspedere en ordre.
- Det skal være muligt for medarbejderne at tilføje flere produkter.
- En ordre skal inkludere en plantegning.

Overordnet beskrivelse af virksomheden

Johannes Fog er et byggemarked der også har en bolig- og designafdeling. Johannes Fog dækker Sjælland med deres ni byggemarkeder, hvoraf det ene befinder sig i Vordingborg og de øvrige i Nordsjælland. Virksomheden blev grundlagt i 1920, og indtil 1970 var det en enkeltmandsejet

virksomhed. I dag er Fog udelukkende ejet af Johannes Fogs Fond. Virksomheden beskæftiger sig primært med træ og tilbehør hertil, men de har også stor ekspertise i salg af jern, stål og andre metaller. De kan levere til alle slags opgaver og fokuserer på et bredt udvalg i størrelser og kvalitet. Samtidigt leverer de også værktøj til alt slags arbejde. Virksomheden beskæftigede i 2017 ca. 459 fuldtidsansatte, og havde en omsætning på lidt over DKK 1.2 mia.

Fog benytter deres eget IT-system til varestyring. Dog har de også behov for andre IT-løsninger til f.eks. at håndtere specifikke ordre som deres nuværende system ikke understøtter. Med hensyn til bestilling af carporte fungerer virksomheden således, at en salgsmedarbejder hjælper kunden gennem ordreprocessen. Dette sker ved en fagperson, såsom Martin Kristensen, i mellemtiden kontakter kunden vedrørende de tekniske detaljer i forbindelse med ordren.

Fog ser ikke blot deres produkter, såsom brædder og skruer til byg af carporten, som værende den samlede service de sælger til kunden. De mener også, at den vejledning kunden modtager, både via ordreprocessen og vejledningen, er en stor del af den ydede service. Kunden betaler dermed ikke blot for materialer, men også for den service Fog leverer under handlen. Grundet dette er det vigtigt for Fog at deres nye IT-system, som håndterer disse handler, understøtter og eventuelt forbedrer denne proces.

Arbejdsgange der skal IT-støttes

I det følgende afsnit vil vi gennemgå den "gamle" metode hvorigennem ordre behandles. Herefter redegøres der for, hvordan vores nyudviklede system vil strømline og effektivisere denne proces. Dertil har vi udarbejdet to aktivitetsdiagrammer som vil blive benævnt AS-IS og TO-BE.

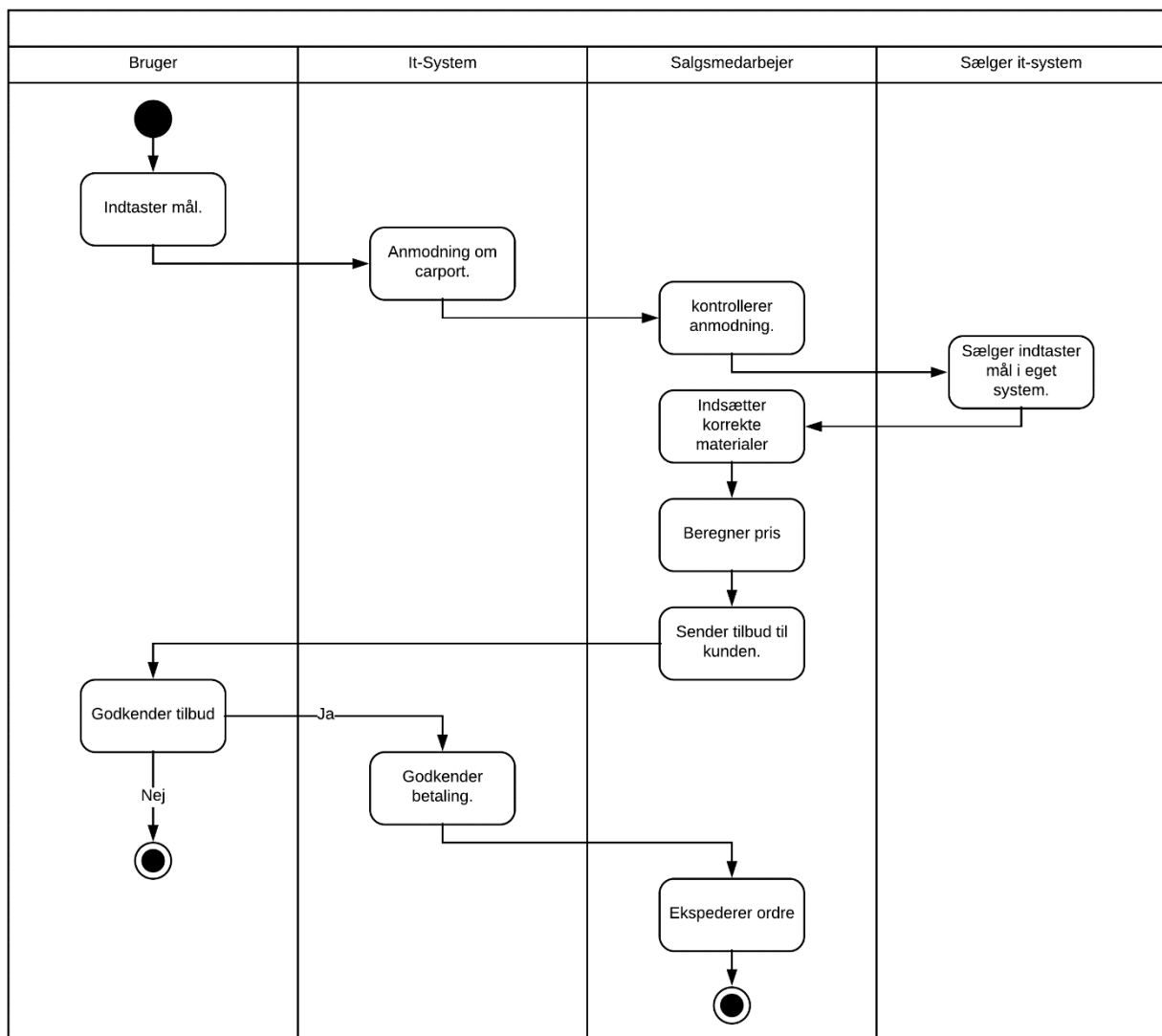
AS-IS

Når en kunde navigerer ind på Fogs carport-hjemmeside, skal de først tage stilling til hvorvidt de ønsker en carport med et fladt tag eller et tag med rejsning. Derefter kan de vælge deres ønskede mål ud fra dropdown menuer, hvilket består af faste værdier som medfører at en kunde ikke umiddelbart kan få en carport på unikke værdier.

Efter en kunde har indtastet sine mål, sendes der en forespørgsel til Fog hvorefter en salgsmedarbejder skal gennemgå ordren. I tilfælde af, at kunden har indtastet nogle urealistiske mål, f.eks. en carport på mål længde: 240, bredde: 240 men samtidigt vælger et skur på størrelse længde: 400, bredde: 400, bliver denne ikke valideret førend salgsmedarbejderen bekræfter

manuelt og, i samarbejde med kunden, har kontrolleret og sikret ordren. Fog bruger på nuværende tidspunkt en forældet applikation som kan udregne hvilke og hvor mange materialer der skal bruges for at bygge kundens carport. Ulempen i dette ligger ved, at førnævnte applikation ikke kan kommunikere med Fogs varelager og derfor angiver produkter der ikke længere findes i Fogs sortiment. Dette medfører at salgsmedarbejderen manuelt skal indsætte opdaterede produkter og priser i tilbuddet.

Når salgsmedarbejderen har udregnet et færdigt tilbud med korrekte materialer, fremsendes dette til kunden. Herefter skal kunden godkende tilbuddet hvorefter der kan genereres et salg.



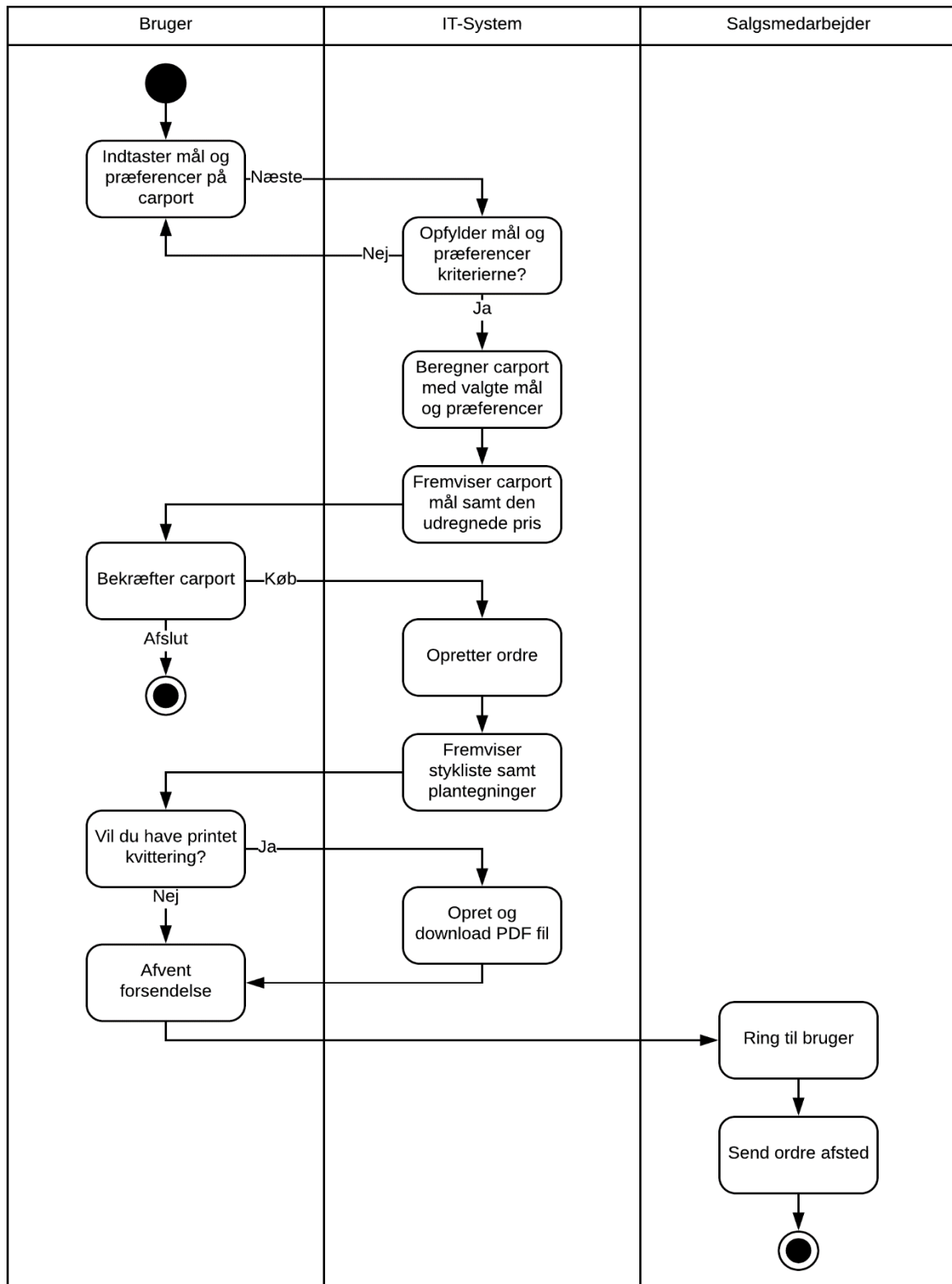
Figur 1: Aktivitetsdiagram

TO-BE

I vores system, vil processen "at købe en carport" blive strømlinet samt give kunden flere valg mht. materialer og mål. Vi vil sikre, at kunden kan vælge ukurante mål, dvs. mål der ikke stemmer med standardmålene på tømmer. Dette vil give større mulighed for at få præcis den carport kunden ønsker. Målene vil herefter blive valideret af it-systemet for at sikre at en kunde ikke kan bestille en carport, hvor der f.eks. ikke er plads til et skur. Herefter vil systemet gennemgå carporten for kunden, og kunden kan se sine valgte mål før bestillingen placeres.

Nu kan kunden afgive en ordre, og når denne er betalt, vil der blive genereret en stykliste samt vist SVG-tegninger af kundens fremtidige carport. Vores mål er dernæst at give kunden mulighed for at printe en kvittering som PDF-fil.

Først nu vil salgsmedarbejderen skulle træde til, f.eks. ved at ringe til kunden og kontrollere at alt stemmer overens samt sikre sig, at kunden ikke har glemt at tage højde for eventuelle byggetilladelser, størrelser på køretøj eller placering af carporten. Dette vil også kunne genere mersalg i form af træbeskyttelse. Efterfølgende kan salgsmedarbejderen ekspedere ordren.



Figur 2: Aktivitetsdiagram

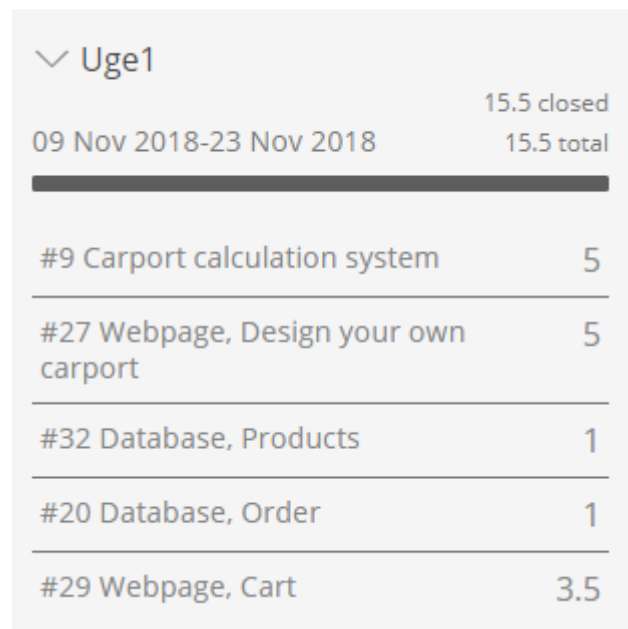
Scrum userstories

I dette følgende afsnit vil vi redegøre for hvordan vi aftalte *sprints* med Project Owner, senere forkortet PO. Afsnittet gennemgår også en håndfuld userstories fordelt over alle fire sprints. Vi brugte Taiga.io som taskboard til udformning af userstories og sprints. Taiga.io er et udmærket og gratis værktøj til SCRUM, idet det er overskueligt ved sin drag-and-drop funktion og har alle nødvendige værktøjer til at udføre SCRUM.

Første uge efter vi havde lært om SCRUM, oprettede vi en del userstories. Disse fik vi rettet til og sammen med PO kom vi frem til et sprint der ser ud som vist i figur 3 nedenfor.

PO lagde vægt på, at han ifølge egen fremgangsmåde og syn på projektet kunne genere et salg ud fra vores system allerede inden for første uge. Derfor lagde vi ud med at skrive en calculator der kunne udregne en basis carport, med stolper, remme, spær, vandbrædder, trapezplader til taget samt tilhørende skruer osv. Alt dette på mål en kunde selv kunne indtaste.

Da vi oprettede userstories fik de i Taiga.io tildelt et # nummer. Disse har ikke relevans for vores userstories men kan give en indikation på, hvordan vores userstories blev udformet og hvordan vi måtte slette mange, da de ikke kunne opfylde senere specificerede krav.



Figur 3: Sprint Uge 1

I det følgende gennemgås userstoryen for Carport Calculation System.

As a:	I want:	So that:	Done:	How to Demo:	Sub tasks.	Point
Sales employee	The system to calculate the price and parts	Customers gets their desired products	At vi kan modtage alle forskellige	Gå igennem processen med at designe en	Sizechart Materials.	5

	needed for the desired product without needing to use a 3rd party software.	faster and I can work with something else to be more productive.	parametre og give et korrekt resultat.	carport adskillige gange, med forskellige parametre for at sikre at vi regner rigtigt og ikke blot har hardcodede priser ind.	Stolpe beregner. Remme beregner. Spær beregner. Tag beregner.	
--	---	--	--	---	--	--

Som det kan ses i diagrammet ovenfor, har vi udarbejdet vores userstories ud fra kriterierne der er fremhævet i de grå felter. På denne led mener vi, at vi opfylder kravene til opbygningen af userstories.

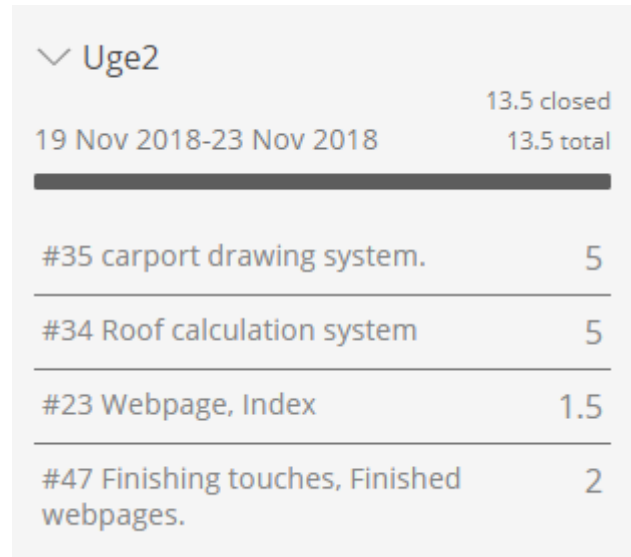
Hver userstory skal indeholde "As a", "I want" og et "So that" eller "As" punkt, hvor det bliver gjort tydeligt hvem der ønsker dette implementeret i systemet, samt hvilke behov der opfyldes. Vi har også beskrevet, hvordan vi anser en userstory som værende færdig samt hvordan vi kan teste om dette er tilfældet. Ydermere har vi også delt de fleste userstories op i mindre brudstykker, kaldet sub tasks. Sidste punkt i vores userstories er et point system, hvor vi estimerer den krævede tid samt kompleksitet af en userstory. Dette kan virke abstrakt, men for os gav det et godt overblik over en opgave. Vi har disse cirka estimater på point, som vi håber er fyldestgørende for forståelsen:

- 1 point = en meget simpel opgave som vil tage ca. 1 time at udføre.
- 5 point = en stor og krævende opgave der kan tage op til to hele arbejdsdage udføre.

Vi har udarbejdet alle userstories efter disse principper hvilket kan ses i *bilag 6*. Til vores andet møde med PO kunne vi fremvise et færdigt sprint med tilhørende tests på vores beregninger samt demonstrere en simpel side hvorpå en bruger kunne indtaste alle mål imellem 240cm til 780cm (længde), 240cm til 750cm (bredde) og 200cm og 250cm (højde). Herefter blev der genereret en stykliste og der blev udregnet præcis hvilke materialer der skulle bruges, ud fra regler

fundet hos Fog for hvordan en carport skulle udformes. F.eks. skal stolper graves minimum 90cm ned i jorden, og dette tages der også højde for i systemet.

I samarbejde med PO udformede vi sprintet for Uge 2 efter vores Demo af Uge 1. Dette sprint udmærkede sig ved at vi skulle kunne generere en plantegning i SVG, og tilføje muligheden for at beregne et tag med rejsning. Sprintet udformede sig som vist i *Figur 4*.



Figur 4: Sprint Uge 2

For PO var det vigtigt at kunne genere en plantegning til kunderne, netop for at synliggøre hvordan en given carport bør konstrueres. Da vi igennem førnævnte regler har taget højde for f.eks. afstand imellem stolper og spær ud fra max distancer, var det vigtigt for PO at dette blev illustreret for kunden. PO ønskede også at kunne sælge carporte med spidst tag, og derfor måtte vi også tilføje yderligere beregninger til vores calculator. Det var også vigtigt for PO at vi kunne byde kunder velkomne på en flot og indbydende webside hvorfra de kan navigere rundt i systemet. Til slut ønskede PO at vi gennemførte designet fra Index-siden ud over hele systemet – altså implementerede det på custom carport siden, samt indkøbskurven.

I nedenstående gennemgås Finishing touches Finished webpages.

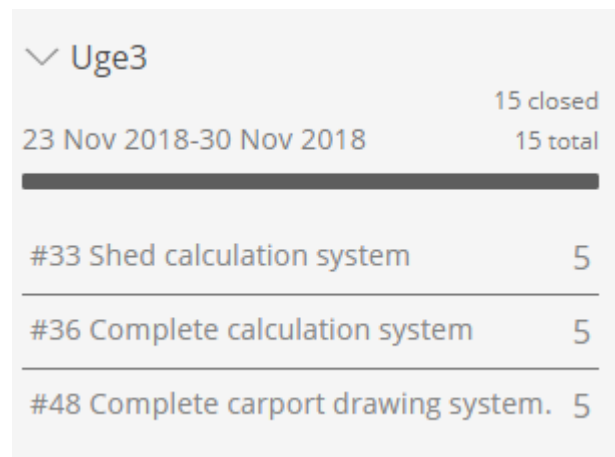
As a:	I want:	As it:	Done:	How to demo:	Sub tasks:	Point:
Customer.	I want the finished	it greatly increases	når siderne er færdige og ser	vi kan navigere	Finish design on	2

	webpages to look nice and crisp.	my trust in the company and increases my overall experience.	lækre ud, med alle funktioner implementeret.	rundt på de færdige sider.	webpages via. Bootstrap.	
--	---	--	--	----------------------------------	--------------------------------	--

Her begyndte vi at implementere bootstrap over alle undersider, efter vi havde lavet det overordnede design for siden. Dette var en mindre opgave som vi estimerede ville tage 3-4 timer.

Sprintet for Uge 2 nåede vi også uden at overskride tidsplanen.

I Uge 3-sprintet var fokus på at blive færdige med alle udregninger, og derfor vi implementerede en udregner for et skur, samt opdeling af alle calculator klasser og gøre det mere overskueligt hvilke klasser der gjorde hvad. Sprintet endte derfor med at være tre store userstories.



Dette sprint blev overskredet, da vi i denne uge også arbejdede meget med at teste systemet. Dette blev ikke til deciderede userstories. Da vi skulle genere tests for alle vores udregninger, blev alle userstories i dette sprint forsømt i en eller anden grad. Men vi vurderede sammen med PO at vi nåede 12 point for ugen. Dette fandt han acceptabelt, idet han forsøgte at udfordre os. Skulle test-delen laves til userstories ville denne uden tvivl også udgøre 5 point.

Den følgende userstory vi vil gennemgå, er Complete Calculation System. Denne userstory samler alle udregninger og krævedelgeledes en del oprydning i samtlige af vores udregningsklasser. Flere af klasserne var på flere hundrede linjer kode, og disse er i dag delt ud over mange klasser med tydelige metoder der vil gøre et eventuelt vedligehold relativt ubesværet i fremtiden. Samtidigt med at det vil være let for en fagfælle at sætte sig ind i, hvordan de fungerer.

Figur 5: Sprint Uge 3

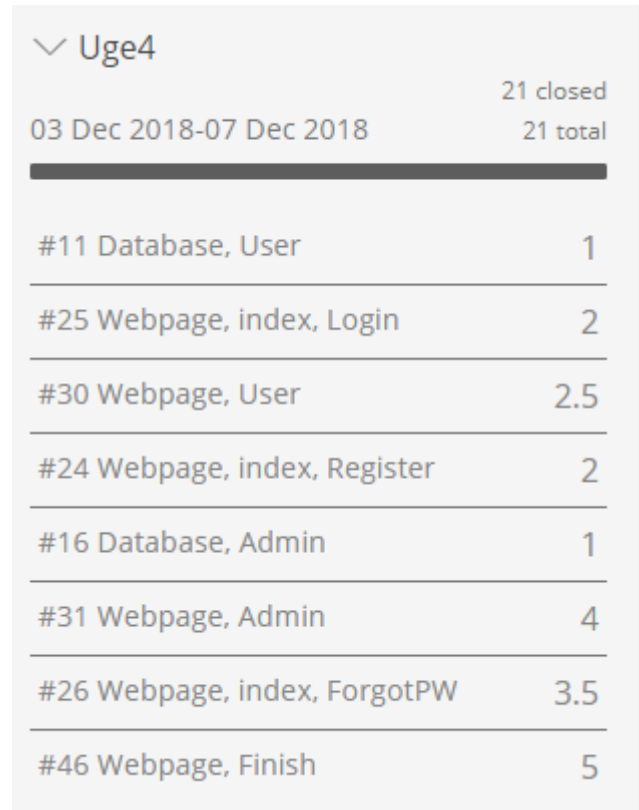
Userstoryen er delt op i to sub tasks: 1) Tie calculators together og 2) Fix calculators. Men denne burde og kunne deles yderligere op, da vi blandt andet fik oprettet en ren klasse for regler, for hvordan man bygger en carport i forhold til Fogs anbefalinger.

As A:	I want:	As it:	Done:	How to demo:	Sub tasks:	Point:
Customer	a complete calculation system, so I don't have to use three calculators then to figure out how they should be connected. one system, one set of data, one price, one action.	I get my desired product faster and with better user experience.	Vi kan samle alle vores udregninger og få den færdige, samlede løsning.	Vi kan vise at vi kan samle de øvrige calculators og vise den totale pris for en vilkårlig carport med eller uden et skur. Og med et fladt tag eller et tag med rejsning.	Tie Calculators together. Fix Calculators.	5

Da afleveringen var sat til d. 29. december i vores skema, i stedet for den officielle aflevering d. 19. december, havde vi oprindeligt forestillet os at vores projekt skulle strække sig over fem sprints. Dette er særlig tydeligt idet sidste sprint er fyldt med mange små opgaver, også i form af rester fra forrige sprint. Grundet dette blev den sidste uge presset tidsmæssigt.

Sprintet for Uge 4 endte ud som vist til højre. PO mente ikke, der var behov for at skære i antallet af userstories. Dette resulterede i, at vi endte ud med dette enorme sprint på 21 point samtidigt med, vi havde en rest fra den forrige uge.

Det var for PO vigtigt, vi fik færdiggjort databasen med brugere og administratorer. Disse skulle også have deres respektive rettigheder implementeret i systemet. Derudover skulle der oprettes en registrering for nye brugere, samt gøres mulighed for at logge ind. Derfor kan man ikke længere tilgå calculator eller andre sider af systemet før man er logget ind.



Uge4

03 Dec 2018-07 Dec 2018

21 closed
21 total

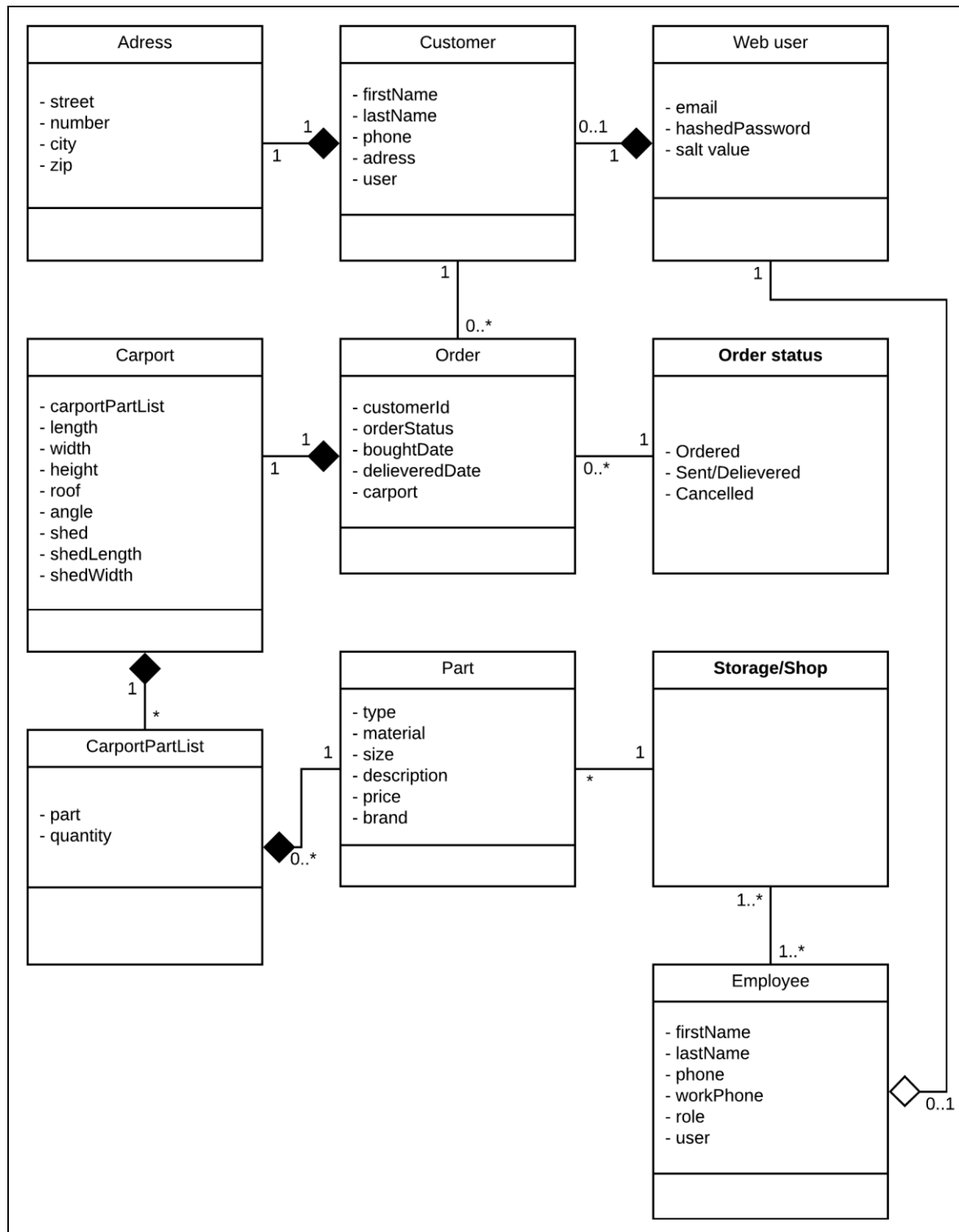
#11 Database, User	1
#25 Webpage, index, Login	2
#30 Webpage, User	2.5
#24 Webpage, index, Register	2
#16 Database, Admin	1
#31 Webpage, Admin	4
#26 Webpage, index, ForgotPW	3.5
#46 Webpage, Finish	5

Figur 6: Sprint Uge 4

For en administrator gælder det, at disse kun kan oprettes af en eksisterende administrator. En administrator kan også, igennem systemet, opdatere databasen med priser, nye produkter osv. Samtidigt skal en administrator også behandle ordre i systemet. Da vi endte ud med et gevaldigt tidspres, har vi ikke implementeret muligheden for at rekvirere et glemt password, og dette sidste sprint blev derfor ikke færdigt til tiden.

Modeller og diagrammer

Domæne model



Forklaring til domæne model

Domænemodellen er centreret omkring ordren, som er opdelt i carport og ordre. Customer er opbygget ud fra vores valg om at separere brugeren fra ordren. Vi er opmærksomme på, at en customer højst sandsynligt ikke vil købe op til flere carporte, men at det er enkelt gangs investeringer, og derfor kunne vi have valgt at slå customer og ordre sammen. Dette ville resultere i, at customer ikke vil have log-in muligheder, og at customer manuelt skal skrive sine oplysninger ved køb af ordren.

Vi mener dog, at det førstnævnte er en optimeret og bedre løsning for Johannes Fog. Vores begrundelse for dette er, at Johannes Fog langt nemmere vil kunne ekspandere og integrere sin forretning ind i systemet i fremtiden. Ved valg af denne løsning har brugeren også mulighed for at logge ind og navigere rundt på den personlige profil, hvor brugeren kan følge med i ordre- og afsendelsesstatus.

Medarbejderne har ikke en direkte forbindelse til ordren, men det er medarbejderne, som med en kritisk tilgang skal vurdere brugerens indtastede data inden, at ordren bliver sendt afsted. I tilfælde af at en ordre ser ukorrekt ud kan medarbejderne således kontakte brugeren gennem deres opgivne kontaktoplysninger og sikre ordrens indhold. Et andet ønske fra Johannes Fog var, at medarbejderne har adgang og mulighed for at opdatere, tilføje og fjerne materialer/dele gennem websystemet, hvilket kommer til syne ved at medarbejderne har en reference til butikken.

Vi vil til sidst kort bemærke, at Storage/Shop er en reference til fysiske lagre og butikker, hvor materialer/dele bliver opbevaret, og hvor det er medarbejderne, som har kundekontakt eller interagerer i websystemet.

Begrundelse for vores valg af 1-1 relation mellem order og carport

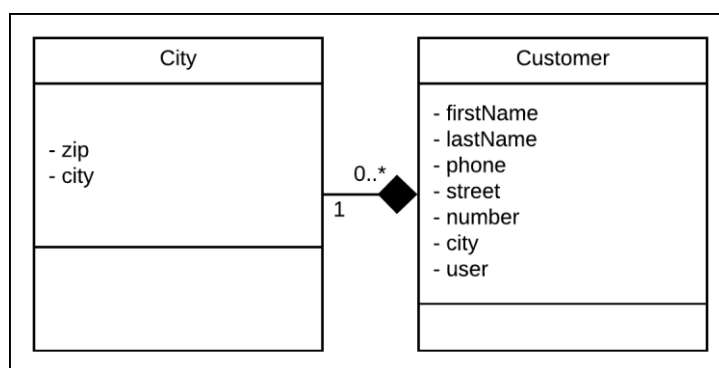
Når websystemet bruger de to dele af ordren, bruges de sommetider forskelligt, og derfor giver bedst mening i vores øjne at dele dem op. Ordren indeholder de konkrete informationer fra selve ordren, der vises når brugeren følger sin tidligere ordre, og endvidere når en medarbejderne skal vurdere ordren før afsendelse. Carporten indeholder data, om hvordan den skal bygges, hvordan styklisten generes og hvordan plantegningerne tegnes. Der havde ikke været en væsentlig forskel, hvis vi havde slået dem sammen, men vores tanke bag dette, er igen, Johannes Fogs ønske omkring

at kunne integrere og ekspandere sin forretning ind i systemet. I tilfælde af at han i fremtiden integrerer en større del af hans forretning i systemet, vil det betyde, at 1-1 relationen mellem ordre og carport vil blive 1-1..* relation mellem ordre og produkt, hvor produkt eksempelvis kan indeholde en carport, værktøj og boliginterior.

Begrundelse for vores valg af 1-1 relation mellem customer og address

Implementationen af address burde ændres, da man med fordel kunne have delt address op i to dele, hvor af den ene indeholder postnummer samt by, og den anden indeholder vej og vejnummer. Ud fra dette skal vej og vejnummer sættes sammen med customer, hvor customer således har en reference via postnummer til postnummer og by.

1-1 relationen mellem customer og address er ikke nødvendig, og derfor burde customer og address slås sammen. Separation af by og vej vil resultere i en 1-0..* relation, hvor et postnummer vil referere til et ubestemt antal brugere, og hvor en bruger vil referere til et bestemt postnummer.



Grunden til at det ovenstående ikke er blevet ændret i projektet, er fordi, at vi først blev opmærksomme på det efterfølgende, og det er således ikke blevet implementeret på grund af tidsmangel.

Begrundelse for vores valg af 1..*-1..* relation mellem storage/shop og employee

En medarbejder kan være tilknyttet til flere lagersteder og butikker, da man for eksempel kan have en leder- eller vikariat stilling, eller hvis man bliver omroket i de forskellige butikker. Modsat skal et lager/butik også kunne have flere medarbejdere til en sådan stor virksomhed som Johannes Fog.

Konsistentitet

Ved første øjekast kan man fejlagtigt tro at domæne modellen går i ring, men dette er dog ikke muligt, da user kun har reference til enten employee eller customer, og derfor er der ingen forbindelse mellem employee og customer.

Som tidligere pointeret er storage/shop en visuel forklaring på, hvordan systemet hænger sammen, og skal derfor ikke forveksles med en del af websystemet.

EER diagram



Normalformer

For at sikre vores database mod redundante data samt gøre vedligeholdelsen og ekspandering af databasen nemmere, har vi gjort brug af de tre normalformer.

Første normalform

De krav der skal opfyldes for at gøre brug af første normalform, er:

- Unikke kolonnenavne
- Same type data i kolonnen
- Ingen 'Multivalues'

Alle tabeller i databasen har unikke kolonnenavne. Kolonnerne indeholder også kun samme type data, og der er desuden ingen 'multivalues' i dataen.

Anden normalform

De krav der skal opfyldes for at gøre brug af anden normalform, er:

- Først normalform
- Ingen partielle afhængigheder

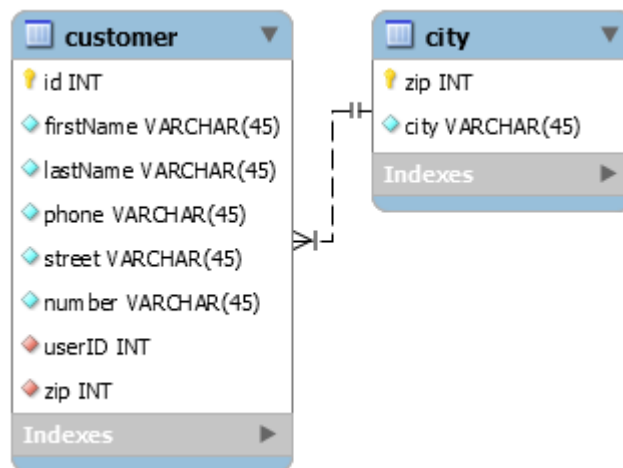
Da kravene til første normalform allerede er opfyldt, og vores database ikke har nogen partielle afhængigheder, er kravene til anden normalform dermed også opfyldt.

Tredje normalform

De krav der skal opfyldes for at gøre brug af tredje normalform, er:

- Anden normalform
- Ingen transitive afhængigheder

Som tidligere nævnt i afsnittet omkring domæne modellen, så er vores address tabel ikke i tredje normalform, da der hurtigt vil opstå en stor mængde af den samme data. Dette er fordi, at man har mange brugere, som vil have bopæl i samme by. Ved at ændre dette, vil address blive til city og city vil således være i tredje normalform og vores andre tabeller opfylder kravene til tredje normalform.



Autogenereret id

Til alle vores tabeller bruger vi autogenereret id. Hvis vi skulle lave den tidligere nævnte ændring hvor address vil blive ændret til city, så ville city ikke have et autogenereret id. Dette er fordi, at city primary key og unique identifier vil være zip, som skal indeholde de rigtige postnumre, udviklerne skal derfor manuelt skrive by og postnumre ind i systemet. Ved opstart vil udviklerne skulle køre et 'city data'-SQL script som indeholder alle postnummre i Danmark, så man nemt kan indsætte dataen i databasen i stedet for en længere omvej.

Fremmednøgler og begrænsninger

Vi har flere steder brugt fremmednøgler for at kunne skabe en bedre database. Som tidligere nævnt har vi valgt at holde brugeren og ordren særskilt, da vi har Johannes Fogs mulige fremtidsplaner i mente. Vi har derfor valgt en fremmednøgle i ordren som refererer til en bestemt bruger, som er brugeren der har købt produktet. Vi har valgt at bruge begrænsningen 'cascade' på 'on delete', hvilket medfører, at når en bruger bliver slettet fra systemet, så fjerner vi også alle ordrer forbundet med brugeren. Dette mener vi, er en god idé, da vi på denne måde kan rydde alt urelevant og dermed unødvendigt fyldende data fra databasen.

Foreign Key Name	Referenced Table	Column	Referenced Column	Foreign Key Options
OrdersProduct	'ProjectFogDatabase'. 'carport'	<input type="checkbox"/> id		On Update: NO ACTION
OrdersCustomer	'ProjectFogDatabase'. 'customer'	<input type="checkbox"/> orderStatus		On Delete: CASCADE
		<input type="checkbox"/> boughtDate		
		<input type="checkbox"/> deliveredDate		
		<input type="checkbox"/> productID		<input type="checkbox"/> Skip in SQL generation
		<input checked="" type="checkbox"/> customerID	id	

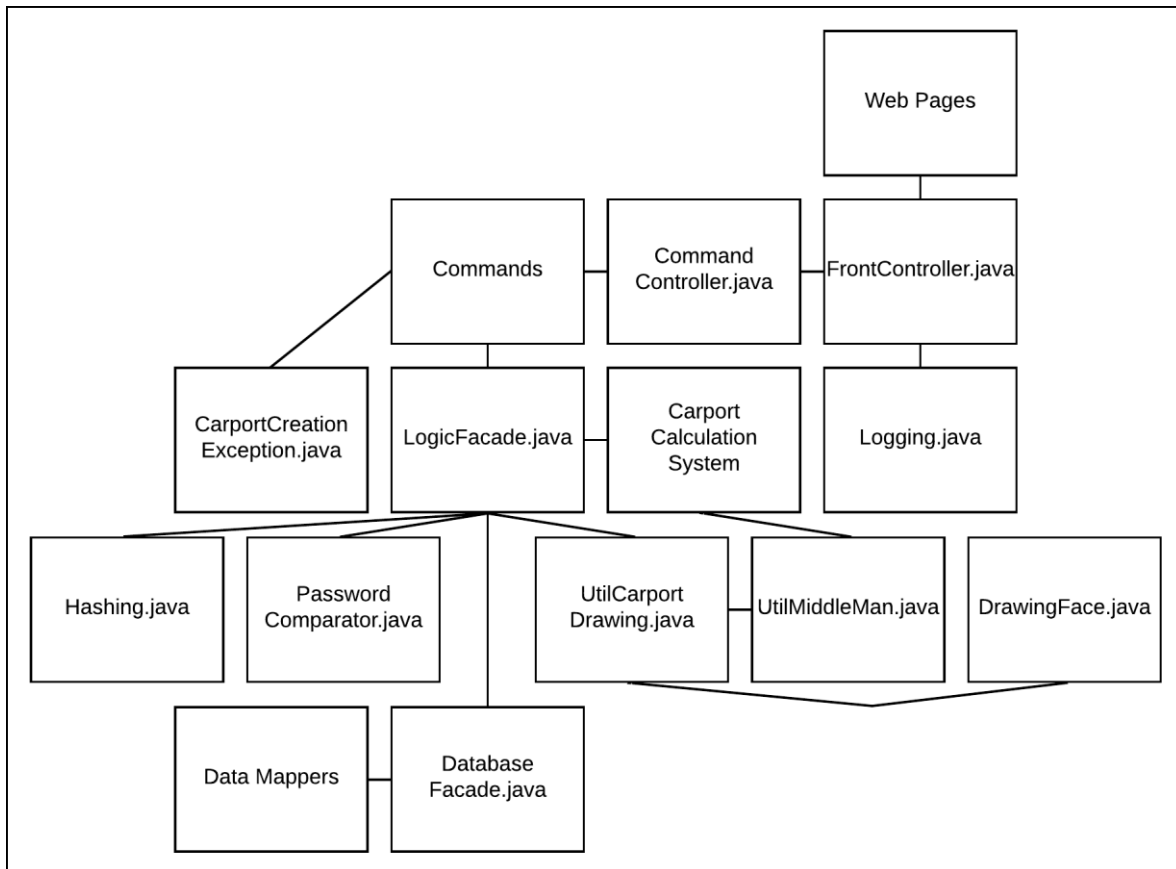
På ovenstående user tabel kan det ses, at der er to yderligere tabeller, som har en fremmednøgle, der refererer til user tabellen. Dette betyder, at en user enten er forbundet med en customer, eller en employee. Her gør vi igen brug af 'cascade' på 'on delete', da vi vil fjerne dataen fra en ikke længere eksisterende bruger eller en medarbejder.

Vi har brugt fremmednøgle flere steder og sat begrænsninger på dem alle, det vil sige, at de alle virker på samme måde, og har alle samme begrænsning. Hvis vi skulle implementere den yderligere nævnte forbedring omhandlende address, ville der være en anden begrænsning, da vi ikke ønsker at slette en city, og derfor ville vælge 'no action' på 'on delete'.

Forbedringer

Først og fremmest ville vi gerne have optimeret address tabellen, udover dette ville vi gerne have defineret data allokeringerne for de enkelte datatyper. Dette ville vi gerne have optimeret for at spare serverplads på længere sigt, for jo længere systemet er i brug - jo mere data vil der blive lagret i databasen. I tilfælde af at Johannes Fog vil integrerer resten af sin virksomhed og måske samtidig vokser og tiltrækker flere kunder, vil data allokeringer være en god idé at få implementeret. Vi har kun brugt standard allokeringer bortset fra ét sted, som er under part tabellen, hvor beskrivelsen er blevet udvidet for at kunne indeholde en fyldestgørende beskrivelse.

Klasse diagram



Den ovenstående illustration er et klasse diagram, som er lavet for, at kunne give en nemt og hurtigt overblik, over hvordan klasserne snakker sammen.

Vi har valgt at bruge tre-lags arkitektur i diagrammet:

1. Præsentations lag
2. Logik lag
3. Data lag

Vi vil nu konkretisere brugen af tre-lags arkitekturen i diagrammet. Øverst findes præsentations laget, hvilket er strategisk placeret så, at det stemmer overens med metoden tre-lags arkitektur er opbygget efter. Derefter har vi placeret logik laget i midten og data laget i bunden af samme grund som nævnt tidligere.

Vi har valgt at opbygge vores system ud fra dette princip, da vi mener, at det gør systemet nemmere at vedligeholde og videreudvikle. Argumentationen for vores valg af tre-lags arkitekturen, vil blive uddybet under afsnittet *særlige forhold*.

På diagrammet kan det også ses, at vi gør brug af facader mellem vores lag, som er af samme grund for vores valg af tre-lags arkitektur nemlig, og gøre systemet nemmere at vedligeholde samt videreudvikle på.

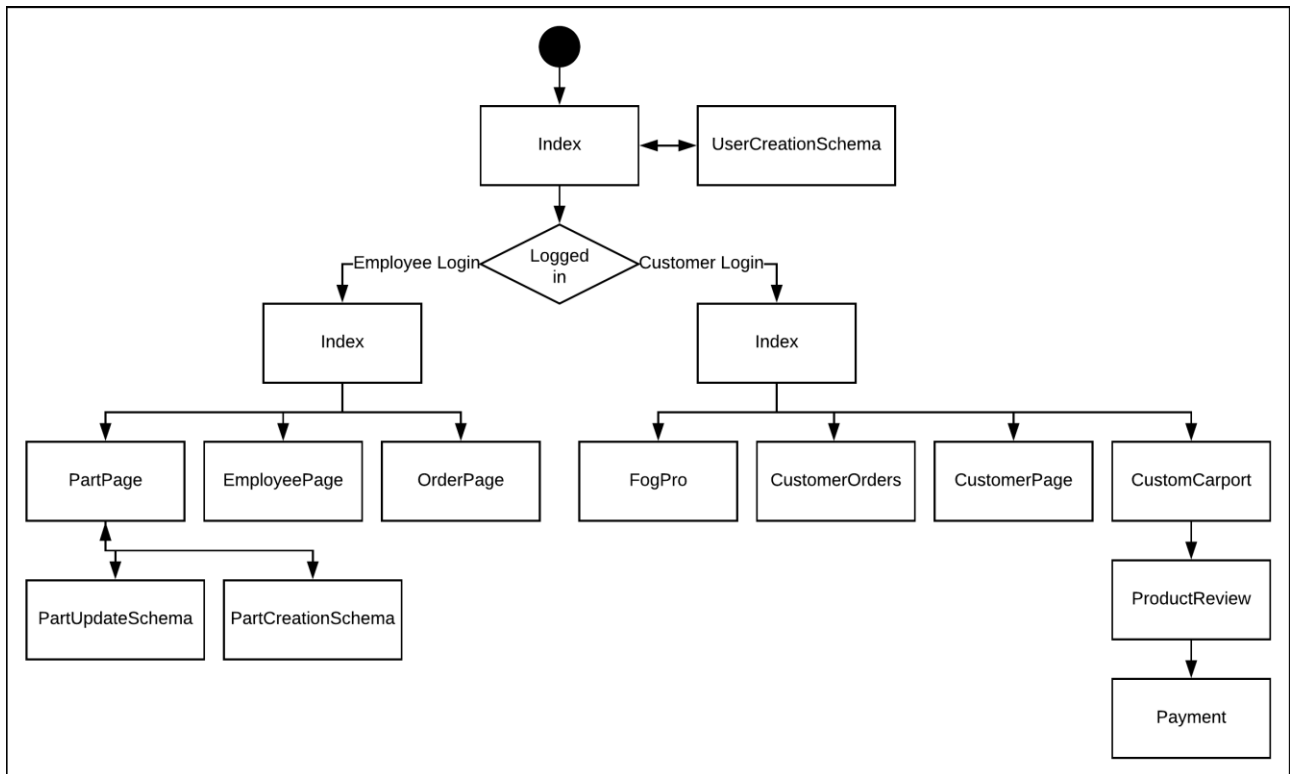
Det kan endvidere ses på diagrammet, at vi også gør brug af et command pattern, som er måden hvorpå vi skifter fra side til side på, og som sørger for, at den information vi har brug for, bliver stillet til rådighed. Dette kommer vil også blive uddybet i afsnittet *særlige forhold*.

Systemet er blevet forholdsvis stort, derfor har vi valgt at lave denne forsimplet version af klasse diagrammet. Vi har dog også lavet en del andre klasse diagrammer, som uddyber de vigtige dele af det overordnede klasse diagram.

Det overordnede klasse diagram indeholder java klasser, men refererer også til de tidligere nævnte klasse diagrammer, som kan findes under følgende bilag:

- *Bilag 1 - Webpage*
- *Bilag 2 - Commands*
- *Bilag 3 - Carport Calculation System*
- *Bilag 4 - Data Mappers*

Navigationsdiagram



På dette navigationsdiagram, kan det ses hvordan, at en person vil starte på index siden og derfra har to muligheder, for at komme videre. Enten kan vedkommende oprette en bruger og efterfølgende logge ind som bruger, eller også kan vedkommende logge ind fra start, hvis personen allerede er oprettet i systemet, og derfor ikke behøver at registrere sig igen.

Videre på diagrammet kan det nu ses, at de mulige funktionaliteter afhænger af hvilken person, det er, som logger ind. Dette skyldes, at systemet vil vise forskellige ting, som er illustreret ved hjælp af diamanten(logged in). Index siden bruger nemlig en navigation bar, som ændrer sig alt efter, om det er en bruger eller en medarbejder, som logger ind. Dette kan ses efter personen har logget ind, da brugeren eller medarbejderen bliver viderestillet til hver sin version af index siden.

Hvis det er en bruger, som logger ind vil brugerens navigations bar være fyldt op af flere funktioner. Brugeren har for det første mulighed for at navigere videre til Johannes Fogs FogPro, eller gå videre til sin egen brugerprofil med personlige oplysninger, som kan redigeres. Brugeren kan også navigere hen til sin ordre side, hvor status på både nuværende og tidligere ordre kan ses. Brugeren har desuden altid mulighed for at vende tilbage til index siden ved et enkelt klik på Fogs logo i navigationsbaren.

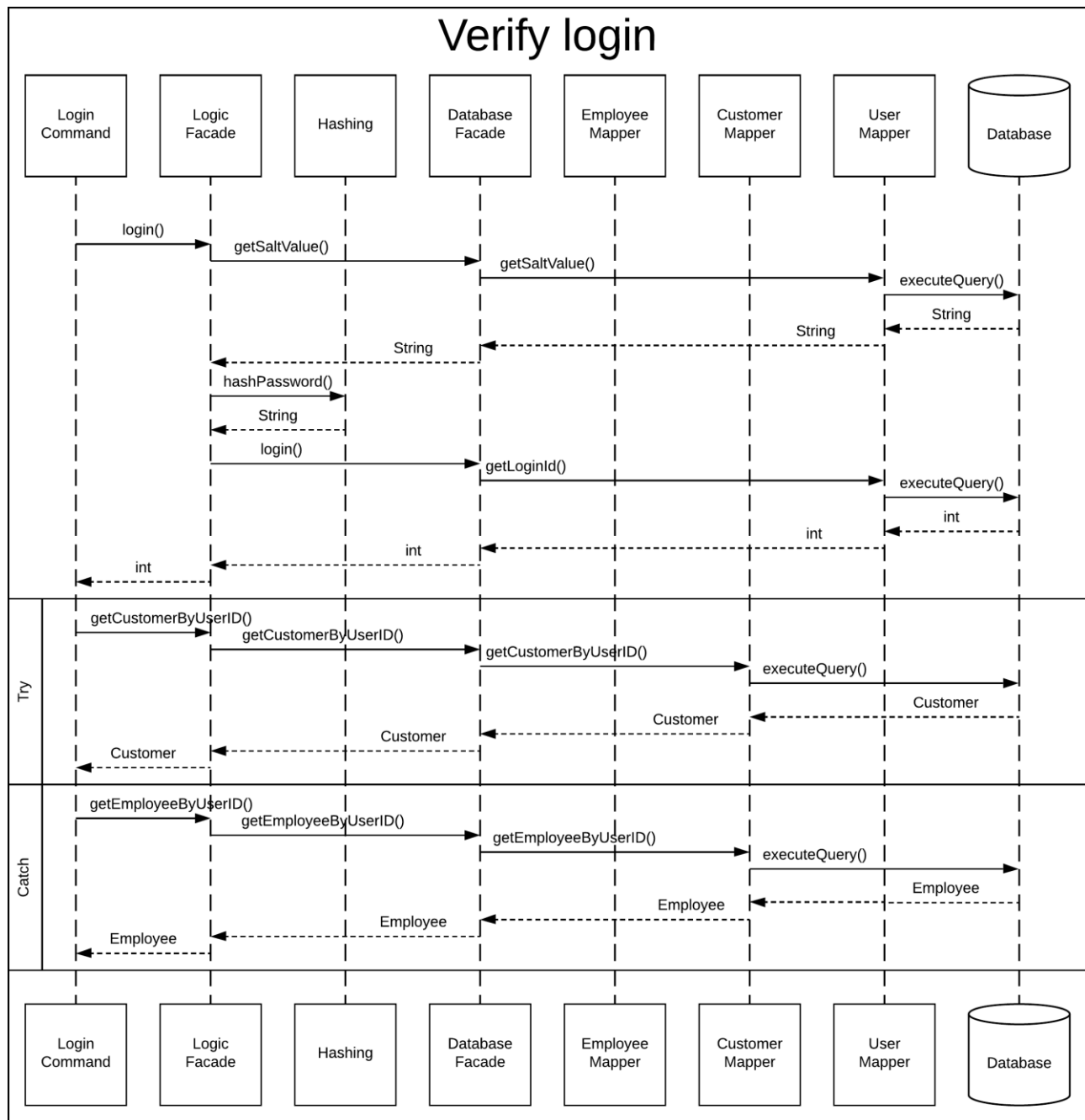
På index siden kan brugeren kan brugeren klikke på det interaktive billede for at blive navigeret videre til siden, hvor det ønskede carport design nu kan iværksættes. Først og fremmest skal brugeren opgive sine krav og præferencer som eks. mål, rejsning eller tagmateriale, derefter vil blive viderestillet til en side, hvor det er muligt at se disse valgte præferencer fulgt af en estimeret pris på den udregnede carport. Hvis kunden er tilfreds og bekræfter dette, vil der oprettes en ordre og kunden bliver nu viderestillet til en side bestående af stykliste og plantegninger over carporten.

Hvis det er medarbejderen, som er logger ind, vil navigations baren indeholde andre funktioner end brugerens navigations bar. Medarbejderens navigations bar giver adgang til siden, hvor alle brugernes ordre er arkiveret, og hvor det er muligt at sende disse ordre afsted. Medarbejderen har også adgang til siden, hvor der kan oprettes nye medarbejdere i systemet. Desuden har man som medarbejder også adgang til siden, hvor der kan redigeres, tilføjes og slettes materialer i databasen.

Både medarbejderen og brugeren kan logge ud, det vil sige, at de vender tilbage index, men med den konsekvens, at både brugeren og medarbejderen mister deres udvidet navigations bar. Det er desuden værd at bemærke, at medarbejderen ikke har adgang, til de sider som tilhører brugerens navigations bar, dette gør sig selvfølgelig også gældende omvendt.

Vi vil afslutningsvist bemærke at alt navigering på hjemmesiden, som vist på navigationsdiagrammet, foregår via vores 'frontController'-servlet.

Sekvens diagram



Dette sekvens diagram viser forløbet af en person, der indtaster email og password i login felterne på startside. Efter at personen har indtastet email og password, vil systemet kontrollere, at de indtastede oplysninger findes i systemet.

Dette vil foregå således, at systemet starter i præsentations-laget hvor *login command* vil starte med at hente emailens salt værdi. For at kunne hente værdien, kalder systemet på *logic facade*, som så kalder på *database facade*, og fører videre til *user mapper*, der indeholder et SQL-statement, som

henter salt værdien fra *database*. Når systemet har hentet værdien, vil den blive returneret til *logic facade* medmindre, at emailen ikke findes i systemet, så vil der nemlig blive kastet en 'LoginException' fra *user mapper*. Når *logic facade* har indhentet salt værdien, vil den derefter gå igang med, at hashe det givne password, sammen med salt værdien. Dette sker i et kald til *hashing*, som både tager password og salt værdien, og returnerer det hashede password. Systemet har nu de nødvendige oplysninger til, at kunne tjekke, om personen har indtastet det korrekte password. *Logic facade* vil nu sende bud efter *database facade*, som derefter kalder på *user mapper*, som så vil tjekke om emailen og det hashede password er et match i *database*. I tilfælde af de passer sammen vil id'et blive hentet fra *database* og sendt retur til *login command*, men hvis de derimod ikke passer sammen, vil der blive kastet en 'LoginException'. Hvis systemet henter en valid bruger, skal systemet dernæst identificere, om det er en bruger eller en medarbejder, der har logget ind.

Hvis det hentede id ikke matcher en bruger, vil der blive kastet en 'LoginException'. Ved hjælp af en 'try-catch block' kan systemet identificere, om det er en bruger eller en medarbejder, som skal hentes. Det vil sige, at hvis systemet fejler i at hente en bruger, er det fordi, at en medarbejder forsøger at logge ind. Dette foregår således at, *login command* går i *database* gennem *logic facade* og videre til *database facade* som derefter gør brug af *customer mapper*, der så tjekker i *database*, om der findes en bruger med det hentede 'userID'. Hvis dette er tilfældet, vil denne bruger (customer) blive returneret til *login command*, men hvis det ikke er tilfældet, skal systemet indhente en medarbejder fra *database*. Systemet indhenter medarbejderen ved, at *login command* kalder til *logic facade*, som dernæst kalder til *database facade*, der til forskel fra indhentningen af bruger, gøre brug af *employee mapper*. Her tjekkes der i *database* for at finde netop den medarbejder, som har det korrekte 'userID', derefter returneres medarbejderen (employee) til *login command*. Når *login command* har fået fat i den korrekte bruger eller medarbejder, vil dataen blive gemt i sessionen, og personen vil nu være logget ind i systemet.

Særlige forhold

Session

Vi bruger sessionen til at gemme informationer, som de andre sider skal bruge til enten fremvisninger, valideringer eller udregninger. Vi vil nu komme med et eksempel på noget af det første, vi gemmer i sessionen:

Når en person skal oprette sig i systemet, vil de indtaste deres email samt et gyldigt password, som herefter vil blive gemt i sessionen. Det vil sige, at systemet får givet en reference via sessionen til disse input data, til når systemet skal oprette brugere i databasen. Efter at brugeren eller medarbejderen har logget ind vil deres id samt navn desuden også blive gemt i sessionen.

Vi har følgende oplistet hvilke data tilknyttet til brugere og medarbejder, som vil blive gemt i sessionen:

- | | |
|---------------|--------------|
| • RegEmail | • User |
| • RegPassword | • CustomerID |
| • Customer | • EmployeeID |
| • Address | • Name |

Vi har også flere forskellige bannere, som kan blive aktiveret på to måder, men begge måder udspringer fra den gemte data i sessionen.

Den ene måde at bannerne kan blive aktiveret på, er ved, at der bliver gemt en besked i sessionen, som vil udløse en fremvisning af et af bannerne. Den anden måde de kan aktiveres på, er ved, at der bliver gemt et boolsk udtryk i sessionen, som vil få den betydning, at et allerede eksisterende banner vil blive fremvist, hvis udtrykket er sandt.

Vi har følgende oplistet banner beskeder og boolske værdier, som skal aktivere bannerne:

- | | |
|-----------|----------------|
| • Msg | • ErrorMessage |
| • Updated | • Error |

Ved viderestilling til siden hvor man har mulighed for at indtaste sine præferencer til carporten, sørger systemet for, at en liste af tegl vil blive gemt i sessionen, som er det brugeren kan se og vælge imellem. Hvis brugeren har indtastet sine præferencer, vil disse også blive gemt i sessionen.

Vi har følgende oplistet de indtastede carport værdier:

- | | |
|-----------------|-----------------|
| • CarportLength | • Roofing |
| • CarportWidth | • Shed(boolean) |

- CarportHeight
- AngledRoof
- Angle
- ShedLength
- ShedWidth
- Flooring

Dataen omkring produkter, materialer og ordre som bliver brugt til fremvisning på andre sider, bliver også gemt. De kan ses nedenstående:

- Product (data klassen carport med tilhørende mål og præferencer)
- ProductPrice (pris for den udregnede carport)
- ProductList (styklister for den udregnede carport)
- SVG (plantegninger af den udregnede carport)
- Orders (en liste af ordrer)
- Products (en liste af carporte)
- Part (en liste af materialer)
- Part (til redigering af materiale)

Brugerinput validering

Vi har tænkt meget over, hvordan vi skulle håndtere valideringen af brugerinput, da der nemt kan forekomme tastefejl samt misforståelser. I vores HTML kode har vi på input felterne specificeret hvilken type data, der er mulig at indtaste. Dette er for at brugerne forhåbentlig laver færre fejl.

Der bliver desuden tjekket flere steder i systemet for, om det input systemet får, er valid input. Systemet kan nemlig smide fejl, hvis dette input er ukorrekt. Dette er et eksempel på en hurtig validering, som skal sikre systemet mod tom data:

```
private boolean needsChange(String input) {  
    if (input == null) {  
        return false;  
    } else if (input.isEmpty()) {  
        return false;  
    }  
    return true;  
}
```

Den ovenstående metode bruges, når brugeren skal ændre sine personlige oplysninger, idet personen har mulighed for at ændre alle oplysninger på engang, men også kan vælge at ændre på kun én oplysning.

Ved indtastning af præferencer til carporten, har brugeren også mulighed for at tilvælge et skur og/eller et tag med rejsning, men hvis brugeren fravælger dette, kunne dette have været et potentielt problem for systemet. Dette har vi dog taget højde for ved først at tjekke, om de vil have skur og/eller et tag med rejsning, før vi henter dataen, der er tilknyttet dertil:

```
String roofOnOrNot = request.getParameter("angledRoof");
boolean angledRoof = false;
if ("on".equals(roofOnOrNot)) {
    angledRoof = true;
}
```

Dette er et tjek, der sørger for, at systemet vil hive den ønsket vinklen ud, hvis brugeren har tilvalgt tag, men hvis brugeren derimod har fravalgt tag, vil systemet ikke gøre dette.

Sikkerhed

Prepared Statement

Vi bruger Prepared Statements til en del af sikkerheden i vores system, da der tit forekommer brugerinput i systemets SQL statements. Det vil sige at uden Prepared Statements, vil der kunne forekomme SQL injection og Cross-site scripting, som ville kunne resultere i et stort brud på sikkerheden og ydermere skabe ravage i hele systemet.

Hashing

For yderligere at øge sikkerheden i systemet, har vi valgt at bruge hashing, da dette øger sikkerheden omkring brugernes og medarbejdernes login oplysninger. Idet der bliver registeret eller oprettet en ny bruger eller medarbejder i systemet, vil passwordet være som en ren tekst, der derefter bliver sendt til vores logik lag for at blive hashet.

Måden hvorpå et password bliver hashet, er ved brug af sha-1. Sha-1 hasher inputet om til 160 bit eller til et array af 20 byte. Når sha-1 har hashet inputet, som i dette tilfælde er et password, vil systemet konvertere det hashede password til heximal tal, som derefter sendes til data laget, så det kan blive gemt i databasen.

På trods af denne foranstaltning, er der dog stadig mulighed for et potentelt brud på sikkerheden, da en hacker vil kunne udnytte svagheden mellem præsentations- og logik laget, hvor det er muligt at se passwordet i ren tekst, før det bliver konverteret.

Salt

For at beskytter brugerne og medarbejderne, har vi gjort brug af salt. Salt er en metode, der forhindrer hackere i at penetrere databasen, og dermed have mulighed for at se, om der er flere brugere, som har det samme hashede password. Hvis to brugere har samme password, vil det nemlig blive hashet på præcis samme måde, og i sidste ende vil det se ens ud, men dette kan salt gøre noget ved. Salt er en tilfældig genereret tekststreng, som bliver sat sammen med brugerens password, før det bliver hashet, da dette sørger for at alle passwords bliver unikke. Der er dog stadig en minimal chance for, at to brugere har den samme autogenerede salt værdi, og bruger det samme password. Dette kan dog ændres, hvis Johannes Fog mener, at det er vigtigt. Måden hvorpå dette kan ændres, er ved at tjekke om den nye autogenereret salt, allerede findes i databasen.

Salt beskytter brugerne og medarbejderne imod et hacker begreb kaldet 'rainbow tables', som er en samling af knækkede hashede passwords. Dette kan udnyttes og bruges som et opslagsværk til at tjekke om en bruger, har et af de hashede password i et rainbow table.

Brugertyper

Dette IT-system har to brugertyper. Den første type er Johannes Fog's kunder, som er brugerne, dem refererer vi til som en bruger eller customer. Den anden type er Johannes Fog's ansatte, som er medarbejderne, dem refererer vi til som en medarbejder eller employee.

SVG

Vi bruger SVG (Scalable Vector Graphics) til at tegne carportens tre forskellige plantegninger. Da de designede carporte har mange forskellige former, størrelser og udseende, vil det være kompliceret og alt for meget arbejde, der skal til for at gemme alle plantegningerne. Derfor har vi udviklet en util klasse, som ud fra de givne mål og præferencer kan tegne plantegningen over carportene fra flere forskellige vinkler. I afsnittet *kodeeksempler* vil util klassens funktionalitet og vores brug af den blive uddybet.

Stykliste

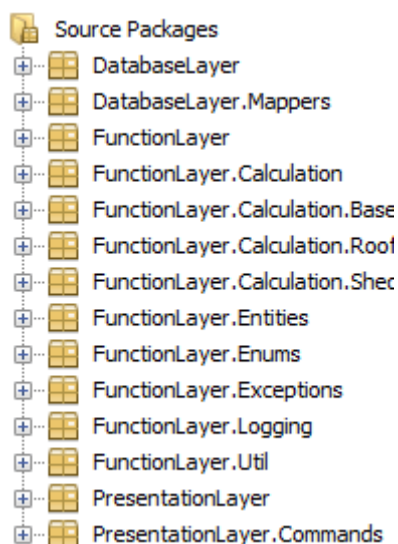
Vi begyndte allerførst med at udarbejde styklisten ved hjælp af en 'ArrayList', men vi fandt os hurtigt nødsaget til at ændre dette valg. Den viste nemlig hver materiale/del, så det blev pludselig en meget lang liste af materiale, hvilket hverken er pænt eller brugervenligt.

Vi prøvede på adskillige måder at få arraylisten til at fungere som planlagt, men efter at have brugt omfattende tid på dette problem, måtte vi gå på kompromis.

Da vi vendte tilbage til arbejdet angående styklisten, var vi i mellemtiden kommet på en god idé nemlig at lave et 'HashMap' med strenge som 'key' og en arrayliste som 'value'. Dette viste sig at være en god og holdbar løsning, da den viste en mere overskuelig og pæn stykliste. Et eksempel på en stykliste fra en test carport med den nuværende kode, kan ses på *bilag 5*. Vi vil senere uddybe kodningen af styklisten under afsnittet *kodeeksempler*.

Arkitektur

Til opbygning af systemet, har vi som tidligere nævnt, gjort brug af nogle arkitektoniske metoder. Dette resulterer i at systemet nemt kan opdateres, videreudvikles og gør det desuden lettere for nytilkomne udviklere at forstå programmet, og navigerer rundt i det. Vi har først og fremmest brugt 'tre lags arkitektur' som bliver tydeliggjort, hvis man åbner koden i NetBeans. Følgende illustration er et eksempel fra NetBeans:



På dette eksempel kan det ses, at pakkerne er navngivet ud fra deres lag, som er efterfulgt af information omkring indeholdt i mappen.

For at forbedre arkitekturen har vi valgt at benytte os af 'facade pattern', som fungerer godt sammen med tre-lags arkitekturen. Facade pattern hjælper systemet med at få en lav kobling og en høj binding. Når vi skal fra presentations laget og videre ned, går vi igennem 'logic facade', som derfra kalder de nødvendige metoder. Hvis vi får brug for, at skulle ned i data laget, går vi igennem 'database facade', for at komme ud til de korrekte mappers.

Præsentations laget vil vi gerne gøre overskueligt og nemt viderudvikle på, derfor har vi gjort brug af 'frontcontroller pattern', hvilket er en enkel servlet, som tager imod alle forespørgelser. Dertil bruger vi 'command pattern', som vi mener, fungerer optimalt med frontcontroller pattern, da man nemt kan oprette nye kommandoer med funktionalitet, og som derefter ved hjælp af frontcontrolleren, kan viderestille brugeren eller medarbejderen til den nye side.

Exceptions

For at beholde den lave kobling, har vi valgt at lave vores egne exceptions, som kan blive kastet flere steder i systemet, men alle vil blive fanget i vores frontcontroller, hvor der vil blive gemt en fejlbesked i session som nævnt tidligere, derefter vil systemet logge fejlen til en fil, dette kommer vi mere ind på under afsnittet *Logging*. Til sidst vil vi blive viderestillet til den side som bruger fejlen til noget konstruktivt, for eksempel at vise bruger hvad der gik galt. For at gøre det nemt for at selv, lavede vi fire egne exceptions:

- RegisterException
- LoginException
- CarportCreationException
- FogException

Vi har navngivet vores exceptions så, at man nemt kan få øje for, hvad de bruges til.

RegisterException bliver smidt, hvis der sker skulle forekomme fejl ved oprettelse af brugere eller medarbejdere.

LoginException omhandler fejl ved login. Det kunne eksempelvis være, hvis brugeren eller medarbejderen skriver et forkert password, eller hvis vedkommende ikke eksisterer i systemet.

CarportCreationException bruges til at oplyse brugeren omkring ugyldigt brugerinput i designprocessen af carporten.

FogException håndterer resten af de fejl, som har med systemet at gøre.

For at optimere vores exceptions, når de skal logges, bliver de tildelt et 'level', som viser hvor fatal fejlen har været for systemet. Dette betyder, at man nemt kan frasortere de små fejl, når man som systemudvikler læser loggen.

Logging

Der kan forekomme fejl, som vi ikke kender til. For at rette disse fejl gør vi brug af konceptet logging, hvilket bidrager til tilkendegive og liste disse fejl ned i en fil, der dermed giver os som systemudviklere mulighed for at læse og forstå fejlene, og derfor kan teste på problemet og få løst fejlen. Følgende er et eksempel på en fejl i loggen:

```
1534 dec. 18, 2018 10:28:38 AM FunctionLayer.Logging.Logging write  
1535 INFO: User 'fdsa@fdsa.dd' do not exist!
```

På dette eksempel ses datoen og klokkeslæt hvorpå fejlen opstod. Det kan desuden ses, hvor fatal fejlen har været, i dette tilfælde var det 'info', hvilket er en mindre fejl. Til sidst i beskeden oplyses fejl beskeden. Denne fejl er forårsaget af en person, der har prøvet at logge ind som enten bruger eller medarbejder, men som ikke eksisterede i systemet.

Udvalgte kodeeksempler

Hash password

Vi har valgt følgende eksempel, da det bliver brugt i vores sekvens diagram. Dette er hele metoden, hvorpå et password bliver hashet. Vi har nedenstående delt eksemplet op i flere dele for at uddybe netop disse:

```
public static String hashPassword(String password) throws FogException {  
    try {  
        MessageDigest md = MessageDigest.getInstance("SHA-1");  
        md.update(password.getBytes());  
        byte[] bytes = md.digest();  
        String hashed = DatatypeConverter.printHexBinary(bytes);  
        return hashed;  
    } catch (NoSuchAlgorithmException ex) {  
        throw new FogException("An error occurred while trying to hash password!", Level.SEVERE);  
    }  
}
```

I denne forklaring vil vi tage udgangspunkt i parameteret:

```
public static String hashPassword(String password) throws FogException {
```

Dette er en statisk metode, der tager imod en streng med navnet password, som er det password, der skal hashes. Metoden returnerer en streng, hvilket er det hashede password i hexital. Hvis der opstår en fejl, vil der blive kastet en FogException i stedet for returneringen af strengen. Det første trin i metoden ses her:

```
MessageDigest md = MessageDigest.getInstance("SHA-1");
```

```
md.update(password.getBytes());
```

Her oprettes en MessageDigest, som er en instans af sha-1, hvilket vi skal bruge til at konvertere passwordet. Dernæst bliver passwordet konverteret til bytes, hvor vi processerer byte arrayet med MessageDigest.

```
byte[] bytes = md.digest();
```

Hvorefter det processerede byte array bliver gemt i en ny variabel.

```
String hashed = DatatypeConverter.printHexBinary(bytes);
```

Arrayet konverteres til hexadecimal tal, og gemmes i en ny variabel for efterfølgende at blive returneret. I tilfælde af der opstår en 'NoSuchAlgorithmException', vil exceptionen stamme fra den første linje i metoden. Vi har derfor lavet en catchblock, som kan tage hånd om denne fejl.

```
} catch (NoSuchAlgorithmException ex) {  
    throw new FogException("An error occurred while trying to hash password!", Level.SEVERE);  
}
```

I catchblokken bliver der en af vores exceptions kastet videre til præsentations laget, hvor den vil blive grebet.

Util

```
private static String drawSVG(int width, int height, DrawingFace face) {  
    String backgroundColor = "#f2f2f2"; // #f2f2f2 = lightgrey  
    int correctHeight = getCorrectHeight(face);  
    StringBuilder SB = new StringBuilder();  
    SB.append("<SVG width=\"600\" height=\"")  
        .append(correctHeight)  
        .append("\" viewBox=\"0 0 ")  
        .append(width)  
        .append(" ")  
        .append(height)  
        .append("\" style=\"background-color: ")  
        .append(backgroundColor)  
        .append(";\">");  
    return SB.toString();  
}
```

Dette er metoden, som laver den grå baggrund på plantegningen. Det ovenstående billede viser, hvordan vi bruger util klassen til at tegne varierende carporte. Metoden skal have oplyst, hvilken størrelse og side carporten skal tegnes i. Det kan ses på billedet, hvordan vi bruger en StringBuilder til at lave en lang html streng, der indeholder parameter variablerne, som i dette tilfælde vil tegne forskellige størrelse kvadrater.

I opstartsprocessen tegnede vi plantegninger manuelt, for at gøre det nemmere for os selv. Da designet var på plads, var det ligetil at implementere i util klassen.

Stykliste

```
public static HashMap<String, ArrayList<Part>> convertListToMap(ArrayList<Part> list) {  
    HashMap<String, ArrayList<Part>> map = new HashMap();  
    for (Part part : list) {  
        String key = part.getType(); // What about size difference???  
        ArrayList<Part> al;  
        if (map.containsKey(key)) {  
            al = (ArrayList<Part>) map.get(key);  
        } else {  
            al = new ArrayList();  
        }  
        al.add(part);  
        map.put(key, al);  
    }  
    return map;  
}
```

Denne metode skal ændre styklisten fra en arraylist til et hashmap, hvilket vi som tidligere nævnt, valgte at gøre for at generere en mere overskuelig stykliste.

Metoden tager imod arraylisten og returnerer hashmappet. Første trin sker ved oprettelse af en variabel og tildeling af et nyt hashmap som værdi, hvorefter arraylisten bliver loopet igennem. Hvis der allerede findes en kopi af denne type i hashmappet, vil den blive gemt under den key, ellers vil der blive oprettet en ny key, hvor den vil blive gemt.

På billedet ses der også en to-do kommentar, hvilket vil blive uddybet senere i afsnittet *status på implementation*.

Carport udregning

```
private void calcRoofing(double sideC, int length, String roofing) throws FogException {  
    Double area = ((sideC * length) * 2) + (RAFT_THICKNESS * length);  
    area += area * ROOFING_FELT_OVERLAP;  
    int squareMetersCount = (int) Math.ceil(area / 10000); // Square Cm to Square m  
    String type = roofing;  
    if ("Sort".equals(roofing.split(" ")[0])) {  
        type = "Sort højglas";  
    }  
    addPartToList(squareMetersCount, type, CalculatorHelper.getCorrectRoofing(roofing), "100x100cm");  
}
```

Denne metode udregner hvor meget tag, der skal være på et tag med rejsning. Metoden tager imod den skrå bredde (sideC), længden samt valget af tagmateriale. Dens output ligger i metodekaldet addPartToList, da denne metode modificerer en arraylist. Metoden kan kaste en FogException, som kommer fra metodekaldet addPartToList, denne opstår hvis materialet ikke findes i databasen. Det første, der sker på billedet er at arealet af taget bliver beregnet. Dette udregnes ved at addere de to skrå sider med midterstolpens bredde. Derefter beregnes det overlappede areal med i det samlede areal for at opfylde carportens byggereregler. Da denne udregning er i centimeter (cm) og ikke i meter (m), betyder det, at målene skal konverteres. Dette gør vi ved brug af math klassen, så vi nemt kan runde udregningen op, og så brugeren dermed ikke vil mangle materiale. Derefter er vores valg af løsning ikke optimal, da vi ikke har modificeret tomcat til at bruge special bogstaver, hvilket vil blive uddybet i *status på implementation*. Derefter kalder vi metoden addPartToList, som sørger for at materialet bliver tilføjet til arraylisten.

Status på implementation

Da dette projekt som tidligere nævnt har været planlagt til 5 sprints, kom vi i tidspres grundet gruppens størrelse på 2 personer. I dette afsnit lister vi de få mangler, vi har i vores system i forhold til vores egne målsætninger.

Special tegn

Da vores tastaturer (character encoding) i Norden kører efter UTF-8 kender vi flere tegn end de gør i US-ASCII, hvilket Apache Tomcat bruger. Derfor kan vores system ikke skrive Æ, Ø og Å. Vi har grundet prioriteringer ikke implementeret UTF-8 i Tomcat for vores system.

JSP

Da vores kompetencer ikke ligger i front-end og vores prioritering har været at lave et færdigt og funktionelt system, har vi forsømt en del på JSP-siderne. Dette ses i form af, at meget af vores kode er samlet med "lappeløsninger". Dette ville ikke være tilfældet, hvis vi havde været flere i gruppen, eller hvis vi havde fokuseret mindre på opbygningen af en rigtig carport med alle dertilhørende dele.

PDF

Grundet tidspres fik vi ikke implementeret en metode hvori en bruger kan udskrive en PDF, som viser stykliste samt SVG-tegninger. En alternativomkostning som konsekvens af valg af fokus.

Mailsystem

Vores mål har fra starten været at oprette et mailsystem som bl.a. kan bruges til at sende bekræftelser på køb til brugere, og skabe kontakt mellem brugere og salgsmedarbejdere. Ydermere skulle en bruger også kunne resette sit password hvis det er blevet glemt. Det ny-generede password sendes derefter gennem mailsystemet til brugeren.

Calculator

For at systemet skulle være helt færdigt, kunne vi have talt med en teknisk specialist i carporte. Da vi udelukkende har brugt internettet til at finde information omkring udformningen af en carport, mener vi, der kan være mangler i vores konstruktion.

Manglende CRUD-metoder

I følgende mapper har vi mangler i vores CRUD: 1) AddressMapper, hvor vi har ingen separat delete-del, 2) CustomerMapper, hvor vi har ingen separat delete-del, og 3) EmployeeMapper, hvor vi hverken har separat update eller delete-del. Disse mangler bliver dog hentet i vores UserMapper ved en cascade delete. 4) OrderMapper, som har ingen delete del, 5) ProductMapper, som har hverken update eller delete del. Disse mangler skyldes udelukkende tidspres.

Vi har dog komplet CRUD i både PartMapper og UserMapper.

SVG

Vi har desværre ikke nået at færdiggøre SVG-tegningerne, på grund af den uventede mængde af arbejde fra sprint 3. Derfor tegner vi kun én tegning, men denne genereres dog ud fra den nævnte Util klasse. Vores mål har også været at genere en tegning fra siden og forfra, begge for at give et bedre overblik.

Styklisten

Vores generede stykliste har en fejl, eksempelvis i form af, at et bræt på 300cm og et bræt på 350cm vil blive lagt oven i hinanden, da den ikke kender forskel på længderne grundet den måde vi henter dem på databasen. Den udregner den korrekte pris, men vi har ikke fået løst dette problem.

Test

Filename	Coverage	Total	Not Executed
DatabaseLayer.DatabaseFacade	2,56 %	39	38
FunctionLayer.Calculation.CalculatorHelper	45,00 %	100	55
FunctionLayer.Util.UtilMiddleMan	60,00 %	10	4
FunctionLayer.Exceptions.FogException	60,00 %	5	2
FunctionLayer.Entities.Part	68,75 %	16	5
DatabaseLayer.Mappers.PartMapper	85,71 %	42	6
FunctionLayer.Hashing	86,96 %	23	3
DatabaseLayer.DatabaseConnector	87,50 %	8	1
FunctionLayer.ListToMap	90,91 %	11	1
FunctionLayer.PasswordComparator	92,86 %	14	1
FunctionLayer.Exceptions.RegisterException	100,00 %	5	0
FunctionLayer.Calculation.Shed.ShedPolesCalculator	100,00 %	19	0
FunctionLayer.Calculation.Shed.ShedDoorCalculator	100,00 %	35	0
FunctionLayer.Calculation.Shed.ShedFloorCalculator	100,00 %	11	0
FunctionLayer.Calculation.Shed.CarportShedCalculator	100,00 %	12	0
FunctionLayer.Calculation.Shed.ShedCladdingCalculator	100,00 %	41	0
FunctionLayer.Calculation.Roof.RoofAngledCalculator	100,00 %	55	0
FunctionLayer.Calculation.Roof.RoofFlatCalculator	100,00 %	12	0
FunctionLayer.Calculation.Roof.RoofFlatCalculator	100,00 %	51	0
FunctionLayer.Calculation.Base.BaseRaftCalculator	100,00 %	34	0
FunctionLayer.Calculation.Base.BasePoleCalculator	100,00 %	32	0
FunctionLayer.Calculation.Base.BaseCalculator	100,00 %	9	0
FunctionLayer.Calculation.CarportCalculator	100,00 %	42	0
Total	33,91 %	1504	994

Det overstående billede viser hvilke klasser, og hvor stor en procentdel der bliver testet. Vi har brugt værktøjet Code Coverage til at fremvise dette billede. Billedet viser alt, vi tester på samt den samlede procentdel af hele systemet.

Til et af vores scrum møder med vores PO, fik vi afvide, at vi ikke skulle teste alt, men prioritere det vigtigste som beregninger og udførelse af integrations test.

Vi har testet hele carportens udregning, som igen kan ses på ovenstående billedet. Samtlige værdier er blevet testet, både nogen der ligger under grænseværdien, på grænseværdien og over grænseværdien for at sikre at systemet, virker som det skal. Vi sørger altså for at teste på de forskellige ækvivalensklasser igennem hele udregningen af vores carport. På billedet kan det også ses, at calculatorHelper ikke har hundrede procent, dette skyldes, at de manglende procent, ligger inden for de samme ækvivalensklasse. Det vil derfor ikke give nogen mening at skrive test, som tester det samme, selvom de har forskelligt input. Det væsentlige er nemlig, at teste på de forskellige ækvivalensklasser. Dette havde vi dog testet på, før vores PO anmærkede det.

Vi har også lagt vægt på at teste andre kerne funktioner igennem såsom PasswordComparator, Hashing og ListToMap.

Vi har også formået at teste på materialerne, om der forekommer korrekte exceptions samt henter alle de korrekte materialer. Disse test er lavet for at tjekke data laget i de forskellige ækvivalensklasser før vores integrations test.

Vores integrations test går fra logik laget til data laget. Testen udregner en hel carport, og tjekker, at den samlede pris samt tilhørende stykliste passer. Og de forskellige ækvivalensklasser testes på igen.

Master branch

Ved projektets opstart fik vi oplyst, at master branchen altid skal virke, og dermed ikke må smide nogen som helst fejl. Dette har vi sikret ved, at master branch kun skal have ét commit efter hvert sprint, da alle dele var færdiggjort og sammensat samt testet igennem med succes. Grundet nogle problemer med github opstod der nogle fejl, hvor alle commits blev kopieret i stedet for kun det sidste. Vi nåede ikke at blive færdige i nogle af de senere sprints, så derfor har vi ikke lagt ændringerne over i master branchen.

Proces

Da vi under udarbejdelsen af dette projekt arbejdede ud fra SCRUM metoden, vil vi her gennemgå vores proces i arbejdet.

Arbejdsprocessen faktuel

Sprints og userstories

Vi havde i dette projekt 4 sprints, og disse er vedlagt som bilag og kan også ses i afsnittet *userstories*.

SCRUM Master

Vi har brugt SCRUM Master i den første uge af forløbet, og vi havde planlagt at skiftes hver uge. I den første uge oprettede SCRUM Master Taiga.io gruppe samt uddelegerede opgaver. Vi fandt dog hurtigt ud af, at SCRUM ikke fungerer når man er to, idet det krævede flere brugere til at udfylde de nødvendige roller.

PO møder

Hver uge har vi haft et møde med PO. Her har der ikke været fokus på det tekniske, men derimod hvilke krav Fog havde til os som udviklere.

Vi vil i det følgende gennemgå vores første PO møde. Da vi allerede i løbet af første sprint havde skrevet et program der kunne udregne en custom carport, med fladt tag, samt lavet JSP-sider hvorpå man kunne tilgå dette, gik mødet med at demonstrere disse features. PO virkede begejstret, da vi allerede var nået langt med projektet. Efter demonstrationen havde PO nogle få ønsker til rettelser og forbedringer, bl.a. farvevalg og designændringer.

Dernæst gik vi i gang med at aftale næste uges sprint, hvori der skulle udvikles mere på udregneren til carporten samt generes plantegninger. Vi var selv tilfredse med vores indsats og oplevede, at mødet gik over al forventning. PO afslørede at han ikke havde forventet, vi var kommet i mål med hele sprintet.

Daglige SCRUM-møder

Hver dag startede med en gennemgang af, hvor langt vi var kommet med vores respektive opgaver. Vi snakkede også om hvilke nye tasks vi skulle opstarte i løbet af dagen.

Retrospektives

Vi begyndte hver mandag med at gennemgang vores forrige sprint. Her diskuterede vi vores tidsplaner, hvad der var gået godt og hvad der var gået mindre godt i forrige sprint. Blandt andet fandt vi på det andet retrospektive møde ud af, at vi undervurderede hvor lang tid det tager at lave front-end.

Arbejdsprocessen reflekteret

SCRUM Master reflekteret

Vi forsøgte os i første uge med en SCRUM-master. Men dette viste sig ikke at fungere, da vi fandt det overflødigt, da vi kun var to i gruppen. Dette ville dog ikke være tilfældet, havde vi været tre eller fire. Den største faktor for, det ikke fungerede har været, at vi begge tog meget ansvar under hele projektet. Eksempelvis rettede vi begge til i Taigo.io, og vi aftalte i fællesskab hvilke stykker arbejde der skulle udføres og af hvem.

Retrospektive møder

I vores retrospektive møder talte vi primært om, hvad der fungerede godt og skidt i det forrige sprint. Men da vi kun var to, var de fleste af disse emner dækket i løbet af ugen og udbedret.

Userstories

Vi fandt det forholdsvis let at nedbryde userstories i mindre tasks, men om disse opfylder alle SCRUM-kriterier er der tvivl omkring. Vi fandt dog, at vores system virkede og at vores userstories var håndgribelige og lige at gå til.

Estimeringer

Selv syntes vi, at vores estimer holdt stik igennem hele projektet, med undtagelse af Front-end. Dette skyldtes hovedsageligt, at ingen af os har vores kernekompetence inden for front-end.

Men vi har begge indset vigtigheden af at mestre front-end, hvorfor vi har sat os for at lære mere om dette i fremtiden.

Vejledning og PO møder

For vores vedkommende, fungerede møderne rigtig godt. Vi værdsatte virkelig den hjælp vi blev tilbudt. Samtidigt med, at vi ikke fik ”kørt ud ad en tangent” og lavet noget kunden ikke ønskede. Vi føler, at dette var en rigtig god introduktion til hvordan det fungerer i erhvervslivet.

Produktivitet

Hvad angår SCRUM kom vi aldrig ind i en rytme der passede dertil, igen skyldtes dette gruppens størrelse. Arbejdsmæssigt havde vi en rigtig god dynamik og arbejdsfordeling. Da vi har arbejdet sammen siden første semester, kendte vi til hinandens styrker og svagheder. Dette medførte, at vi allerede fra dag ét fik udført utroligt meget. Dette kan også ses i grundigheden af vores projekt. I forhold til gruppens størrelse mener vi, at vores projekt er meget omfattende.

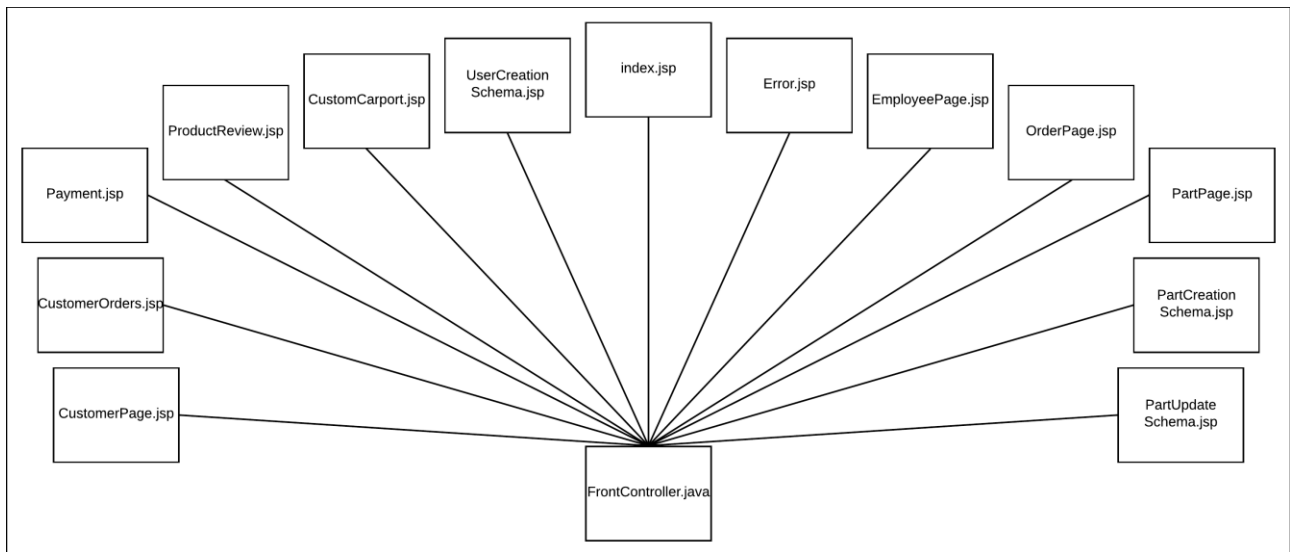
SCRUM generelt

Overordnet har vi indset vigtigheden af at arbejde i SCRUM og kan tydeligt se hvor vigtigt det er i forbindelse med projekter. Derfor vil vi i fremtiden forsøge at arbejde efter SCRUMS principper i alle projekter. At oprette userstories og have klare mål for hvornår og hvordan de blev gjort færdige, hjalp os utroligt meget i processen. Det lettede arbejdet i forhold til tidligere projekter at vi havde en væsentligt bedre og stærkere struktur.

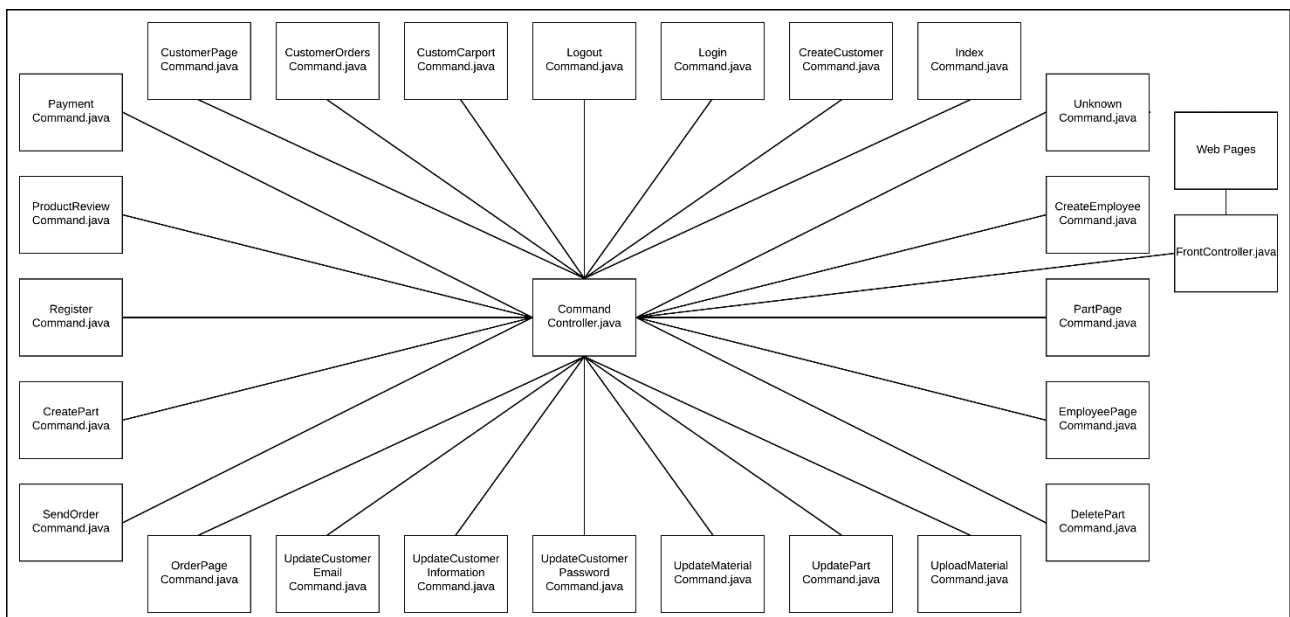
For os virkede sprint-værktøjet i særlig høj grad igennem klare mål og delmål. Dette medførte en høj arbejdsmoral og meget få, hvis overhovedet nogle, perioder med stilstand i projektet.

Bilag

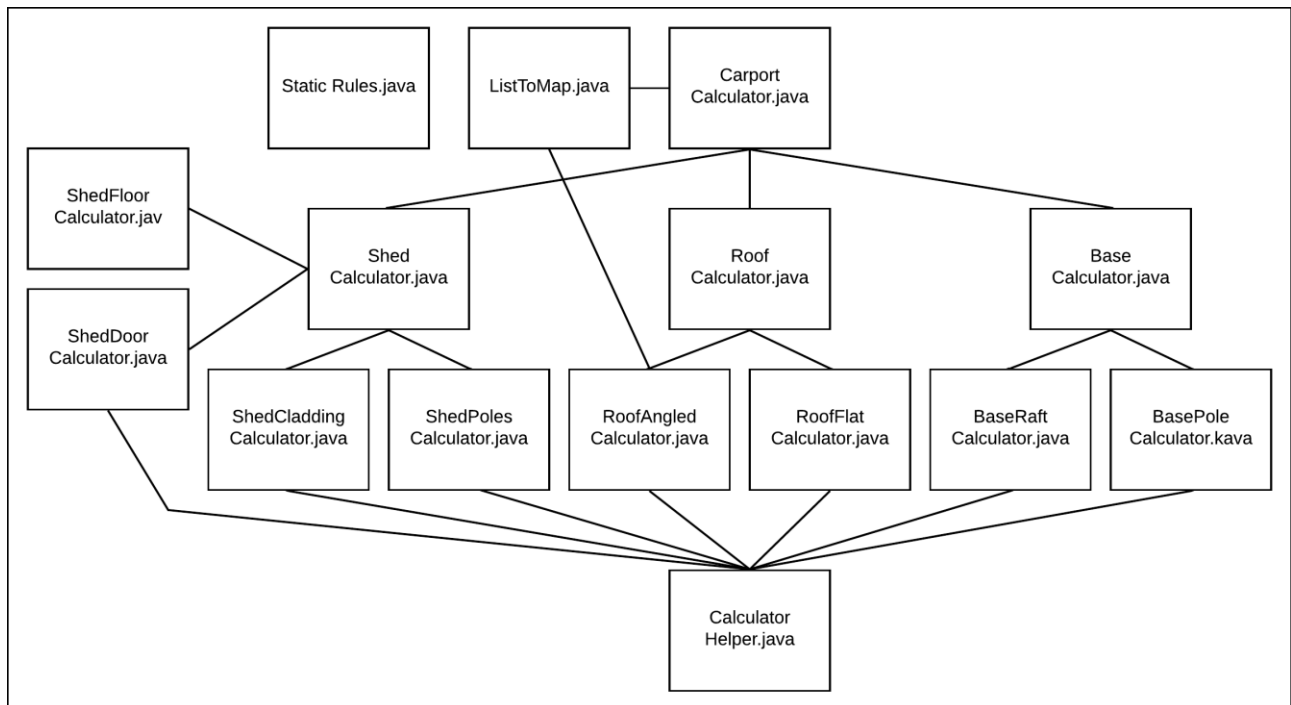
Bilag 1 – Klasse diagram: Webpages



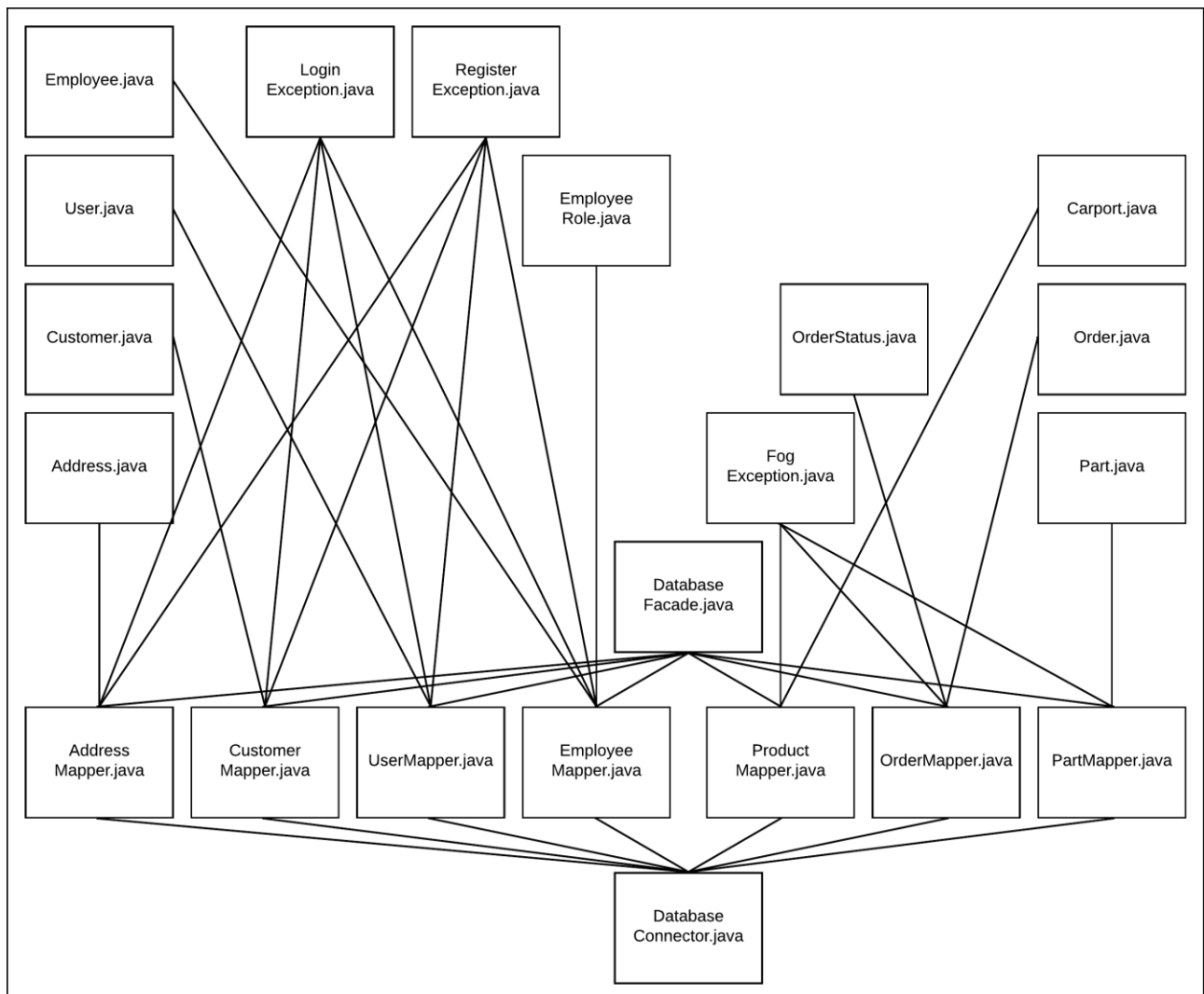
Bilag 2 – Klasse diagram: Præsentations-lag



Bilag 3 – Klasse diagram: Carport udregnings system



Bilag 4 – Klasse diagram: Database lag



Bilag 5 – Nuværende stykliste

Stykliste

Type	Størrelse	Detaljer	Antal	Stykpris
Firkant Skive	40x40x10mm	Firkant Skiver til montering på bræddebolt. 50stk pr. pakke.	1	279.0 kr.
Basic skrue	4.5x60mm	Basic skrue til udendørs brug. 200stk pr. pakke.	3	73.95 kr.
Vandbrædt	19x100mm 240cm	Tryk impreneret vandbræt	24	23.88 kr.
Vingetagsten	100x100cm	Tagsten kvadratmeter	7	156.69 kr.
Stolpe	97x97mm 300cm	Tryk impreneret Stolpe af fyr	4	113.85 kr.
Basic beslag	190mm	Vinkel beslag, til montering af spær	28	43.95 kr.
Bræddebolt	10x20mm	Bræddebolt til montering af spær. husk firkant skiver. 25stk pr. pakke.	1	269.0 kr.
Spær	47x200mm 300cm	Ubehandlet Spærtræ, høvlet	22	131.85 kr.

Pris

Total: 6954.86 kr.

Heraf moms: 1738.71 kr.

Bilag 6 – Sprints

▽ Uge1	
	15.5 closed
09 Nov 2018-23 Nov 2018	15.5 total
#9 Carport calculation system	5
#27 Webpage, Design your own carport	5
#32 Database, Products	1
#20 Database, Order	1
#29 Webpage, Cart	3.5

∨ Uge2

19 Nov 2018-23 Nov 2018 13.5 closed
13.5 total

#35 carport drawing system.	5
#34 Roof calculation system	5
#23 Webpage, Index	1.5
#47 Finishing touches, Finished webpages.	2

∨ Uge3

23 Nov 2018-30 Nov 2018 15 closed
15 total

#33 Shed calculation system	5
#36 Complete calculation system	5
#48 Complete carport drawing system.	5

✓ Uge4

21 closed

03 Dec 2018-07 Dec 2018

21 total

#11 Database, User	1
#25 Webpage, index, Login	2
#30 Webpage, User	2.5
#24 Webpage, index, Register	2
#16 Database, Admin	1
#31 Webpage, Admin	4
#26 Webpage, index, ForgotPW	3.5
#46 Webpage, Finish	5