michael due p., [CPH-MP447@CPHBUSINESS.DK](mailto:CPH-MP447@CPHBUSINESS.DK), MICH561D

j. cHRISTIAN rYGE, [cpH-JR221@CPHBUSINESS.DK](mailto:cpH-JR221@CPHBUSINESS.DK), rANGERrYGE

github link: <https://github.com/mich561d/ProjectFog>

weblink: <http://104.248.29.81/ProjectFog-1.0-SNAPSHOT/>

Fog Projekt Rapport

Gruppe: CRMD

Vinter 2018

Indholdsfortegnelse

[Indledning 2](#_Toc532991793)

[Baggrund 3](#_Toc532991794)

[Valg af teknologi 3](#_Toc532991795)

[Krav (\*) 4](#_Toc532991796)

[Overordnet beskrivelse af virksomheden 4](#_Toc532991797)

[Arbejdsgange der skal IT-støttes 5](#_Toc532991798)

[AS-IS 5](#_Toc532991799)

[TO-BE 6](#_Toc532991800)

[Scrum userstories 8](#_Toc532991801)

[Modeller og diagrammer 15](#_Toc532991802)

[Domæne model 15](#_Toc532991803)

[Forklaring til domæne model 16](#_Toc532991804)

[Begrundelse for vores valg af 1-1 relation mellem order og carport 16](#_Toc532991805)

[Begrundelse for vores valg af 1-1 relation mellem customer og address 16](#_Toc532991806)

[Begrundelse for vores valg af 1..\*-1..\* relation mellem storage/shop og employee 17](#_Toc532991807)

[Konsistentitet 17](#_Toc532991808)

[EER diagram 18](#_Toc532991809)

[Normalformer 18](#_Toc532991810)

[Autogenereret id 19](#_Toc532991811)

[Fremmednøgler og begrænsninger 20](#_Toc532991812)

[Forbedringer 20](#_Toc532991813)

[Klasse diagram (\*) 21](#_Toc532991814)

[Navigationsdiagram 22](#_Toc532991815)

[Sekvens diagram 24](#_Toc532991816)

[Særlige forhold (\*) 25](#_Toc532991817)

[Session (\*) 25](#_Toc532991818)

[Brugerinput validering (\*) 26](#_Toc532991819)

[Sikkerhed (\*) 27](#_Toc532991820)

[Prepared Statement (\*) 27](#_Toc532991821)

[Hash (\*) 27](#_Toc532991822)

[Salt (\*) 28](#_Toc532991823)

[Brugertyper (\*) 28](#_Toc532991824)

[SVG (\*) 28](#_Toc532991825)

[Stykliste (\*) 28](#_Toc532991826)

[Arkitektur (\*) 28](#_Toc532991827)

[Exceptions (\*) 29](#_Toc532991828)

[Logging (\*) 30](#_Toc532991829)

[Udvalgte kodeeksempler (\*) 30](#_Toc532991830)

[Carport udregning (\*) 30](#_Toc532991831)

[Stykliste (\*) 30](#_Toc532991832)

[Util (\*) 30](#_Toc532991833)

[Status på implantation 31](#_Toc532991834)

[Test (\*) 33](#_Toc532991835)

[Proces 34](#_Toc532991836)

[Arbejdsprocessen faktuelt 34](#_Toc532991837)

[Arbejdsprocessen reflekteret 35](#_Toc532991838)

[Bilag 37](#_Toc532991839)

[Bilag 1 – Klasse diagram: Webpages 37](#_Toc532991840)

[Bilag 2 – Klasse diagram: Præsentations-lag 37](#_Toc532991841)

[Bilag 3 – Klasse diagram: Carport udregnings system 38](#_Toc532991842)

[Bilag 4 – Klasse diagram: Database-lag 39](#_Toc532991843)

# 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. Derfor kan produkterne med priser ikke ændres hvilket har medført at den totale pris har været ubrugelig i mange år, hvilket gør at medarbejderen skal lave ekstra arbejde ved at finde de korrekte produkter. Grundet dette har Fog 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.

Vi har derfor i følgende projekt lavet en hjemmeside for Johannes Fog, hvor på der kan bestilles en konfigurerbar carport efter en brugers egne valg og ønsker til materialer samt udformning. Før end en bruger kan bestille skal personen oprettes i databasen, hvorfor vi har lavet en mulighed for at man kan registrere sig på siden. Herefter kan der logges ind på siden. Når en bruger er logget ind, kan de se tidligere ordre samt ændre deres oplysninger.

Vi har også lavet en administrator side, hvori der kan ændres i produkters data, pris, beskrivelse osv. Ydermere skal en administrator kunne oprette nye administratorer og behandle brugeres ordre.

## Baggrund

Johannes Fog er et byggemarked der samtidigt også består af et Bolig og Designhus. Johannes Fog dækker Sjælland med deres ni byggemarkeder, hvoraf det ene befinder sig i Vordingborg og de øvrige i Nordsjælland.

Johannes Fog Trælast blev grundlagt i 1920 og indtil 1970 var det en enkeltmandsejet virksomhed, hvorimod Fog er 100% ejet af tømmerhandler Johannes Fogs fond i dag.

De beskæftiger sig primært med træ og tilbehør hertil, men de har også stor ekspertise i salg af jern, stål og andre metaller. Johannes Fog kan levere til alle slags opgaver og fokusere på et bredt udvalg i størrelser og kvaliteter. Samtidigt kan de også levere værktøj til alt slags arbejde.

Johannes Fog ønsker et system der kan udregne materialer til en carport efter deres kunders specifikke ønsker, her skal de kunne vælge længde, bredde og højde, samt hvilken type tag de ønsker og til slut om de ønsker et redskabsskur interageret i deres nye carport. Her skal systemet kunne generere en stykliste derved får kunden hurtigt en total pris og der kan nemt og hurtigt pakkes fra lageret.

Johannes Fog ønsker at deres kunder kan logge ind på en personlig side, hvor en kunde har overblik over tidligere køb og der kan ændres i kundens oplysninger.

Der skal også være en administrator del, hvor fra Johannes Fogs ansatte kan opdatere produkter, med priser, størrelser, producenter mv. 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:** | **Sprog:** |
| * Netbeans 8.2 * MySQL Workbench 8.0 * Java & Java SE (JDBC) 181 * Apache Tomcat 8.0.27.0 | * Java * HTML * CSS * MySQL 14.14 |

Vi har også brugt front-end framework: Bootstrap 4.1.0

# Krav (\*)

* 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 den service Fog sælger.
* Det skal være muligt for en medarbejder at se og redigere i en foretaget ordre.
* Det skal være muligt for en medarbejder at se og redigere i en foretaget ordre.
* Det skal være muligt for en medarbejder at tilpasse prisen på carporten, i ordren, inden godkendelse.

## Overordnet beskrivelse af virksomheden

Fog er et af de større byggemarkeder i Danmark, de har deres kernekompetence inden for træ og tilbehør hertil. Dog beskæftiger deres afdeling i Vordingborg sig også med salget af metal.

Fog har kun byggemarkeder beliggende på Sjælland, nærmere bestemt Hørsholm, Fredensborg, Kvistgård, Helsinge, Lyngby, Ølstykke, Herlev, Farum og Vordingborg.

Ydermere har Fog også et designhus beliggende i Lyngby, hvor de sælger inventar til boligen. Virksomheden beskæftigede i 2017 459 fuldtidsansatte, og havde en omsætning på lidt over 1,2miliarder kroner.

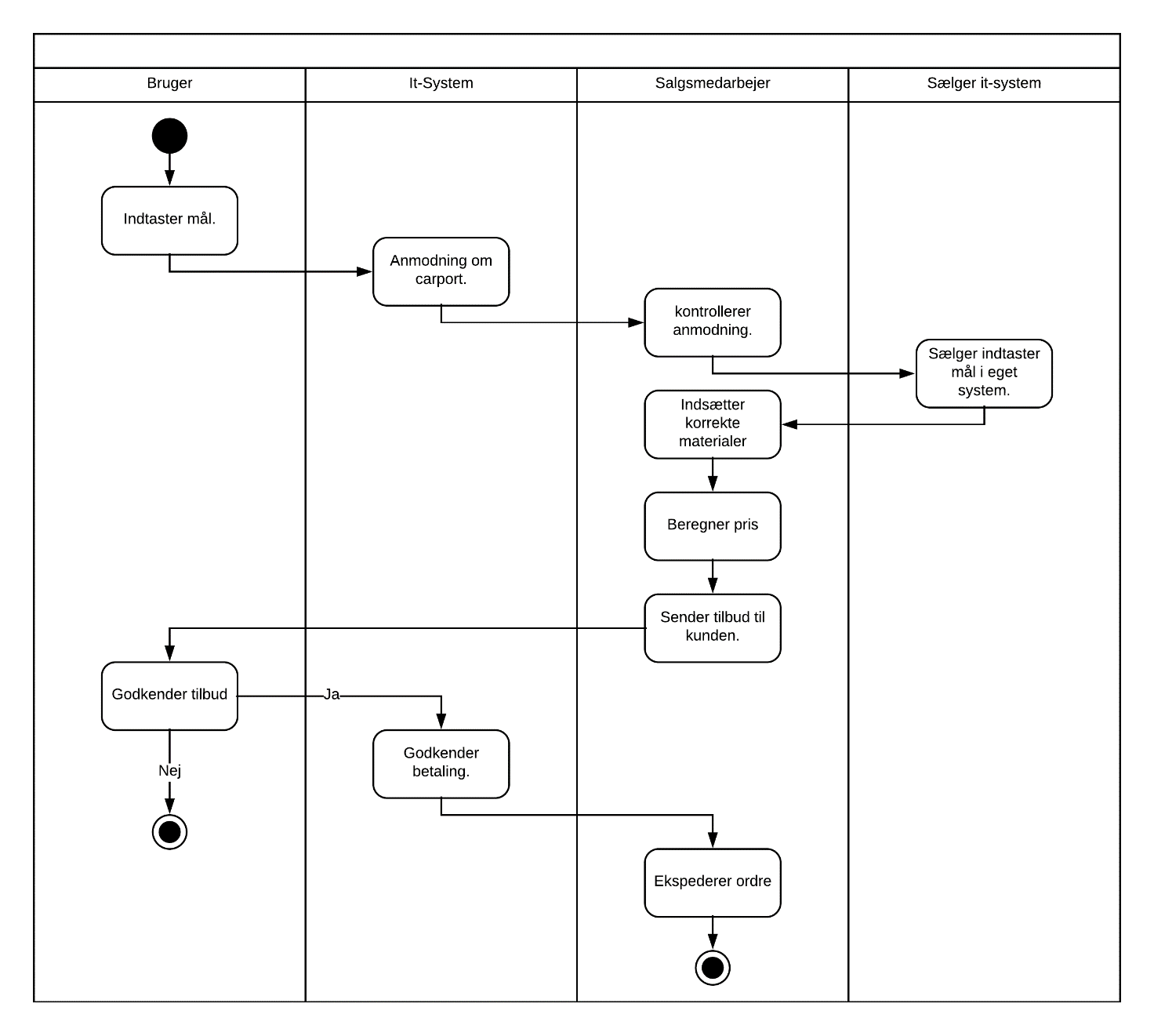
Fog benytter deres eget IT system til varestyring. Dog har de også behov for andre IT-løsninger, f.eks. vores, til at håndtere specifikke ordre, deres nuværende system ikke understøtter. Med hensyn til bestilling af carporte fungerer virksomheden således, at en salgsmedarbejder hjælper kunden gennem ordreprocessen. Dette sker ved en fagperson, såsom Martin Kristensen, i mellemtiden kontakter kunden omhandlende 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 samle vejledningen, er en stor del af servicen. Kunden betaler dermed ikke blot for materialer, men også for den service Fog udøver under handlen. Grundet dette er det vigtigt for Fog, at deres nye IT-system til håndtering af disse handler understøtter-, og måske endda forbedrer- denne process.

## Arbejdsgange der skal IT-støttes

I det følgende afsnit vil vi gennemgå den gamle metode hvorpå ordre behandles og efterfølgende redegøre for, hvordan vores nyes system vil strømline og effektivisere denne proces. Dertil har vi udarbejdet to aktivitets diagrammer som vil blive benævnt AS-IS og TO-BE.

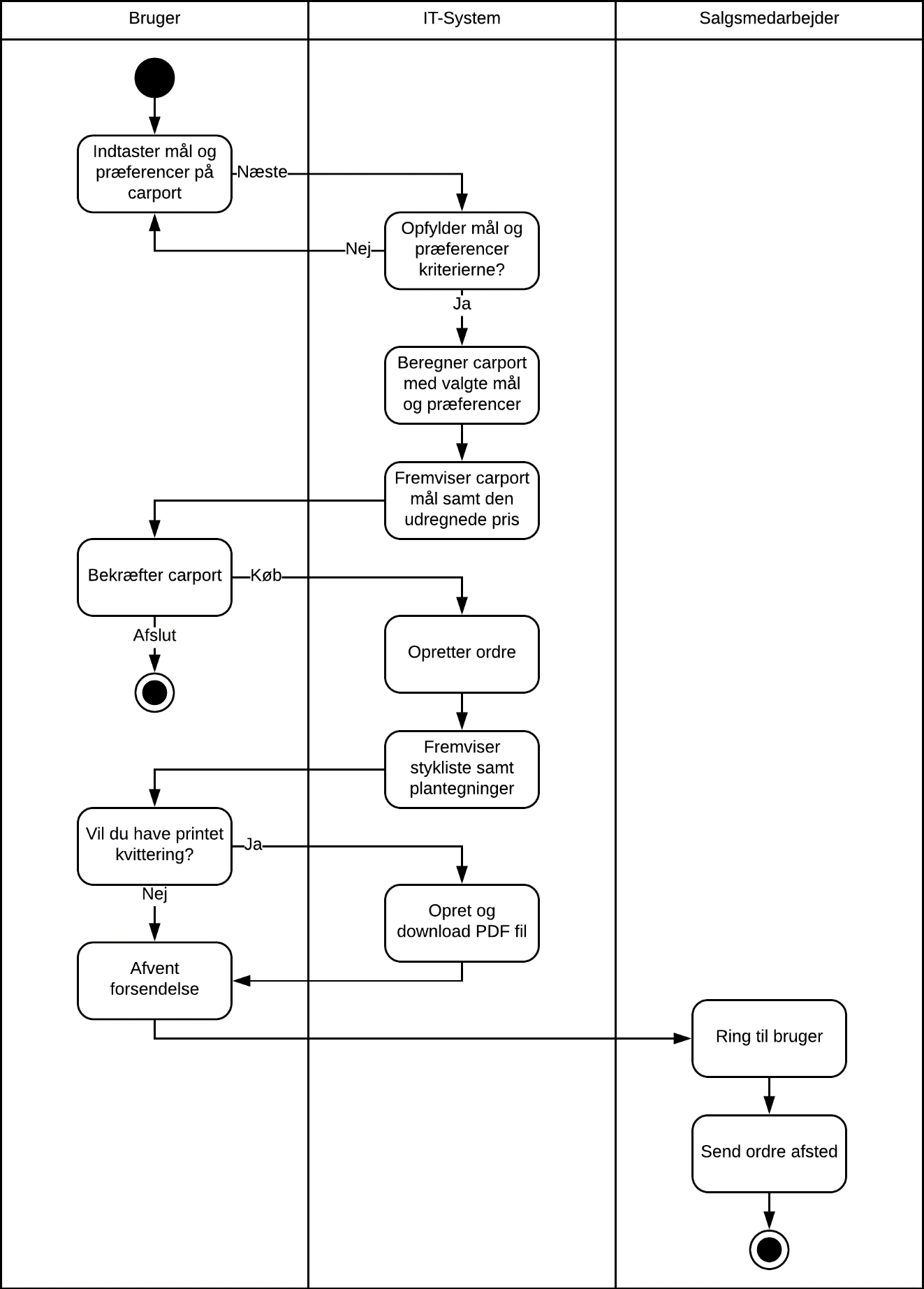
### AS-IS

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, disse er faste værdier hvilket medfører at en kunde ikke umiddelbart kan få en carport på mål der ligger uden for disse værdier.   
Efter en kunde har indtastet sine mål, vil der blive sendt en forespørgsel til Fog hvorefter en salgsmedarbejder skal gennemgå ordren. Hvis kunden har indtastet nogle mål der ikke giver mening, f.eks. en carport på mål længde: 240, bredde: 240 men samtidigt vælger et skur på størrelse længde: 400, bredde: 400. Dette bliver ikke valideret forend salgsmedarbejderen gør dette manuelt. Efter salgsmedarbejderen i samarbejde med kunden har kontrolleret ordren og sikret den. Bruger Fogs ansatte en forældet applikation, denne kan udregne hvilke materialer og hvor mange der skal bruges for at bygge kundens carport. Ulempen her ligger i, 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 de opdaterede produkter samt priser i tilbuddet.   
Når salgsmedarbejderen har udregnet et færdigt tilbud, med korrekte materialer fremsender han dette til kunden. Herefter skal kunden godkende tilbuddet hvorefter der kan genereres et salg.

****

### TO-BE

I vores system, vil processen i 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 en kunde 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 forend der bliver lagt en bestilling.   
Nu kan kunden lægge en ordre, når kunden har betalt for sin ordre, vil der blive genereret en stykliste samt blive 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, denne kan 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 mere-salg i form af træbeskyttelse. Efterfølgende kan salgsmedarbejderen ekspedere ordren.



## Scrum userstories

I dette følgende afsnit vil vi redegøre for hvordan vi aftalte sprints med Project Owner, senere forkortet PO. Samt gennemgå 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.

Første uge efter vi havde lært en smule om SCRUM, havde vi oprettet en del userstories, disse skulle rettes til og sammen med PO kom vi frem til et sprint der så således ud.



PO lagde vægt på, at med vores fremgangsmåde og syn på projektet kunne han allerede i løbet af første uge genere et salg ud fra vores system, 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 for, hvordan vores userstories blev udformet og hvordan vi måtte slette mange, da de ikke kunne opfylde senere specificerede krav.

Vi vil nu gennemgå userstoryen for Carport Calculation System.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| As A: | I want: | So that: | Done: | How to Demo: | Sub tasks. | Point |
| Sales emplyee | The system to calculate the price and parts needed for the desired product without needed to use a 3rd party software. | Customers gets their desired products faster and I can work with something else to be more productive . | At vi kan modtage alle forskellige parametre og give et korrekt resultat. | Gå igennem processen med at designe en carport adskillige gange, med forskellige parametre for at sikre at vi regner rigtigt og ikke bare har hardcodede priser ind. | Sizechart Materials.  Stolpe beregner.  Remme beregner.  Spær beregner.  Tag beregner. | 5 |

Som det kan ses i diagrammet over, har vi udarbejdet vores userstories ud fra kriterierne der er fremhævet i de grå felter. På denne led føler 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 vil blive opfyldt. Vi har også beskrevet hvordan vi anser en userstory som værende færdig samt hvordan vi kan teste om dette er tilfældet. Yderligere 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 lidt abstrakt men for os gav det et godt overblik over en opgave. Vi har disse cirka estimater på point. Ét point er en meget simpel opgave som vi planlægger, vil tage ca. én time at udfører. Fem point er en stor og krævende opgave der kan tage op til to hele arbejdsdage at lave, Vi håber dette er fyldestgørende for forståelsen af pointsystemet. Vi har udarbejdet alle userstories efter disse principper hvilket kan ses i vores bilag.   
Til vores næste møde med PO kunne vi fremvise et færdigt sprint med tilhørende tests på vores beregninger samt demonstrerer en simpel side hvorpå en bruger kunne indtaste alle mål imellem længde 240cm og 780cm. Samt en bredde på mellem 240cm og 750cm. Og en højde på mellem 200cm og 250cm. 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 bliver der også taget højde for i systemet.

I samarbejde med PO udformede vi sprintet for Uge to, efter vores demo af første uge. Dette sprint udmærkede sig ved at vi skulle kunne generere en plantegning I SVG. Samt tilføje muligheden for at beregne et tag med rejsning. Sprintet udformede sig således.



For PO var det vigtigt at kunne genere en plantegning til kunderne, 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 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 hjemmeside 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.

Her vil vi gennemgå Finishing touches Finished webpages.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| As A: | I want: | As it: | Done: | How to demo: | Sub tasks: | Point: |
| Customer. | I want the finished webpages to look nice and crisp. | it greatly increases my trust in the company and increases my overall experience. | når siderne er færdige og ser lækre ud, med alle funktioner implementeret. | vi kan navigere rundt på de færdige sider. | Finish design on webpages via. Bootstrap. | 2 |

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 til at tage 3-4 timer.

Sprintet for uge to nåede vi også uden at overskride tidsplanen.

I sprintet for uge tre var fokus på at blive færdige med alle udregninger, derfor implementerede vi en udregner for et skur, samt at dele alle calculator klasser op 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 da han forsøgte at presse os og skulle testdelen laves til userstories ville denne helt sikkert også udgøre 5 point.

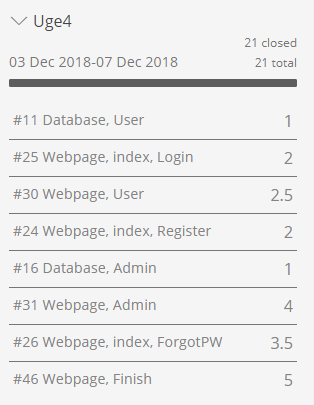
Den følgende userstory vi vil gennemgå, er Complete Calculation System. Da denne userstory samler alle udregninger og krævede en del oprydning i samtlige af vores udregnings klasser. Flere af klasserne var på flere hundrede linjer kode, disse er i dag delt ud over mange klasser med tydelige metoder der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

Vi delte userstoryen op i to sub tasks, henholdsvis ”Tie calculators together” og ”Fix calculators” Men denne burde og kunne være blevet delt 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 tydeligt i form af at vores sidste sprint var fyldt med mange små opgaver. Og da vi tilmed sad tilbage med rester fra forrige sprint. Blev den sidste uge presset tidsmæssigt.

Sprintet for uge fire endte ud således, PO mente ikke at 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 at vi havde en rest fra den forrige uge.



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

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 rekvirerer et glemt password.

Hvorfor dette sidste sprint ikke blev færdigt til tiden.

# Modeller og diagrammer

## Domæne model



### Forklaring til domæne model

Domænemodellen er centreret omkring ordren, som er delt op i to dele: 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 slåes sammen. Seperation 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 omrokeret 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 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 at 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.



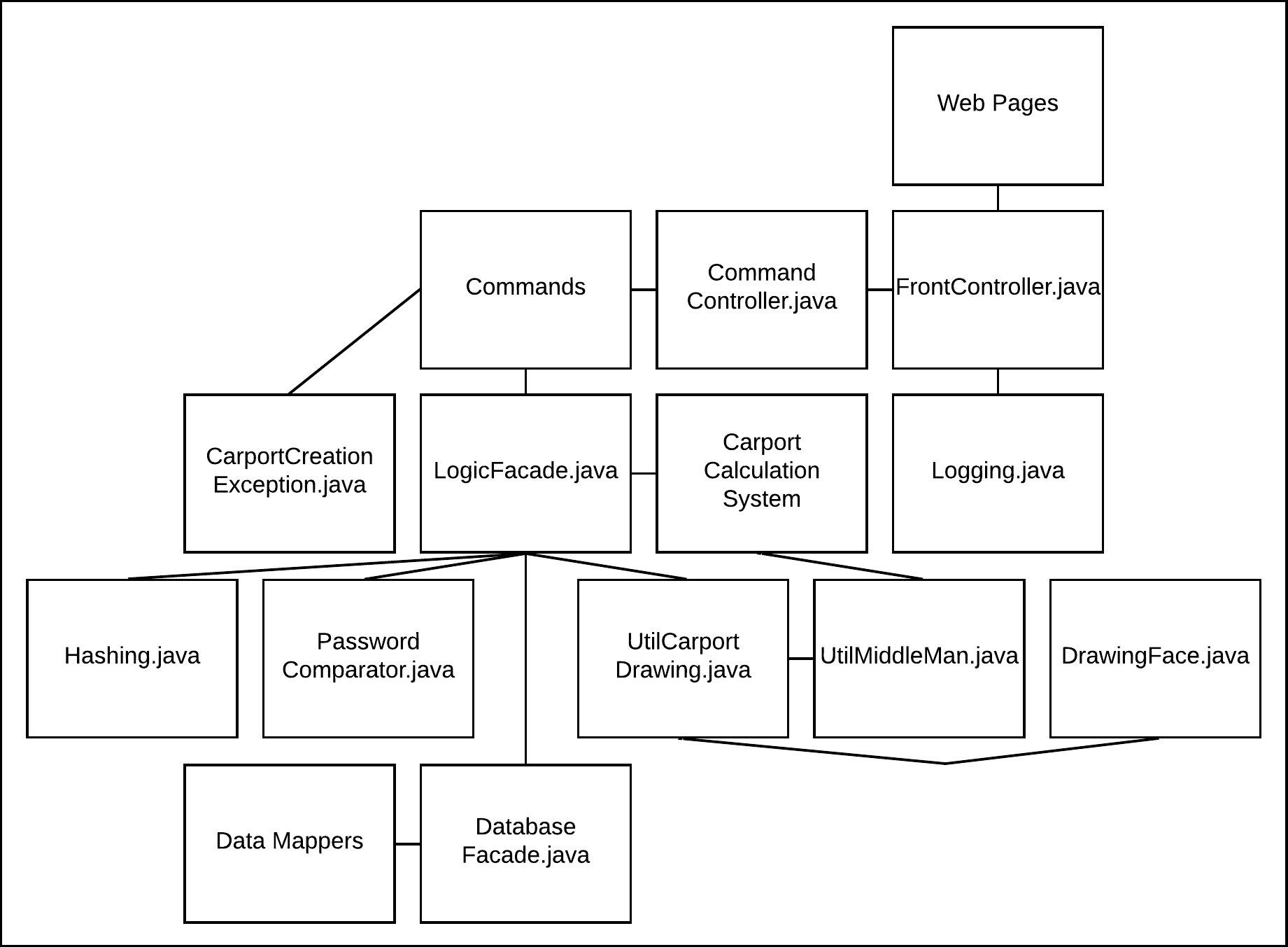
På ovenstående user tabellen kan det ses, at der er to yderligere tabeller, som har en fremmednøgle, der refererer til user tabellen. Dette betyder, at en user enten er forbundet med en customer, eller en employee. Her gør vi igen brug af ’cascade’ på ’on delete’, da vi vil fjerne dataen fra en ikke 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 yidliger nævnte forbedring omhandlende address, ville der være en anden begræsning, 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 (\*)



Dette er et klasse diagram, som lavet for at kunne give en nemt og hurtigt overblik, over hvordan klasserne snakker sammen. Vi har valgte at bruge tre lags arkitektur

* præsentations-lag
* logik-lag
* data-lag

Da vi mener at det gør systemet, nemt at vedligeholde samt videreudvikle. Tre lags arkitektur vil vi komme mere ind på senere hen under ’særlige forhold’.

Da systemet er blevet forholdsvis stort, har vi valgt at lave et for simplet version af klasse diagrammet, men vi har selfølgelig også lavet del klasse diagrammer, som viser dele af det overordnet klasse diagram. Det overordnet klasse diagram, har både java klasser, men har også reference, til et nyt klasse diagram, som følgende:

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

For at sætte en mere konkret forklaring på hvordan, vi bruger tre lags arkitektur, vil vi gerne påpeje at øverest er præsentations-laget, hvilket er stratetisk placeret, da man siger at præsentations-laget er det øverest lag, i tre lags arkitetur. Derefter har vi placeret logik laget i midten, af samme grund, da logik laget er det midderste lag. I bunden har vi derfor data laget, som igen er placeret stratetisk, da data laget er det sidste og nederest lag.

Som vist i diagrammet, gør vi også brug at facader mellem vores lag, dette gør vi for at igen,gøre det nemmere at vedligeholde samt videreudvikle. Dette vil også også komme mere ind på senere i rapporten.

Ud fra diagrammet, kan vi også se at vi også gør brug at et command pattern, som er måden vi skifter fra side til side på, samt sørger for at de information vi har brug for, bliver stillet til rådighed. Dette kommer vi også mere i dybden med senere.

## 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 registere 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 startsiden. 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 såvil 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. Noget af det første vi gemmer i sessions kunne være, når en person skal oprette sig i systemet, vil de indtaste deres email samt et gyldigt password, som vil blive gemt i sessionen, så vi har en reference til disse input data, når vi skal oprette brugere i systemet, efter de har udfyldt registrerings skemaet. Efter at brugere eller medarbejder har logget ind vi deres id samt navn også blive gemt. Her er en liste over data tilknyttet til brugere og medarbejder, som vil blive gemt i sessionen:

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

Vi har også flere forskellige bannere, som kan blive aktiveret på to måde, men begge fra data gemt i sessionen, den ene måde, er hvis der bliver gemt en besked i sessionen, så vil der blive fremvist et banner, den anden måde er hvis vi gemmer et boolsk udtryk svarende til sandt i session, dette vil betyde at en allerede lavet banner skal blive fremvist. List over Banner beskeder/boolske værdier som skal aktivere bannerne:

|  |  |
| --- | --- |
| * Msg * Updated | * ErrorMsg * Error |

Når vi er på vej til siden, hvor man skal indtaste sine præferencer til carporten, sørger systemet, for at en liste af tegl vil blive gemt i sessions, som brugeren kan vælge mellem. Derefter kan bruger indtaste sine præferencer og de vil så blive gemt i sessionsen. Her er en liste over de indtastede carport værdier:

|  |  |
| --- | --- |
| * CarportLength * CarportWidth * CarportHeight * AngledRoof * Angle | * Roofing * Shed(boolean) * ShedLength * ShedWidth * Flooring |

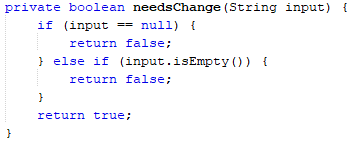
Vi gemmer også data omkring produkter, materialer og ordre som bliver brugt til fremvisning på andre sider:

* Product (data klasse carport med mål og præferencer)
* ProductPrice (pris på udregnede carport)
* ProductList (stykliste på udregnede carport)
* SVG (plantegninger af udregnede carport)
* Orders (en liste af ordre)
* Products (en liste af carporte)
* Part (en liste af materialer)
* Part (til redigering af materiale)

## Brugerinput validering (\*)

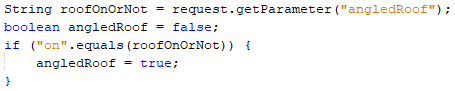
*”It’s human to make mistakes and some of us are more human than others.” – Ashleigh Brilliant*

Vi har tænkt en god del over validering af brugerinput, da vi nemt kan komme tastefejl, samt misforståelser. I vores HTML kode har vi på vores input felter, specificeret hvad type det skal være for at brugere kan lave færre fejl, som vi skal tage højde for. Der er også flere steder, hvor der bliver tjekket, om det input systemet får, er valid input, da systemet, kan smide fejl, hvis inputet ikke er korrekt. Dette er et eksempel på en hurtigt validering, som sikre systemet mod tomt data:



Vi bruger denne metode, til når brugeren skal ændre sine oplysninger, da man har mulighed for at kunne ændre alle oplysninger på engang, men man kan også vælge at ændre kun én oplysning.

Når brugeren skal indtaste sine præferencer til sin carport, har det mulighed for at kunne tilvælge skur samt tag med rejsning, men hvis de ikke gør, så kunne der have opstået problemer, dette har vi taget højde for, da vi først tjekker om de vil have skur og/eller tag med rejsning, før vi henter dataen der er tilknyttet ud:



Dette er et tjek der sørger for at hvis der er tilvalgt tag, må den godt hive vinklen ud, ellers skal den ikke prøve på det.

## Sikkerhed (\*)

### Prepared Statement (\*)

Som en del af sikkerheden i vores systemet, bruger vi Prepared Statements, da der tit forekommer brugerinput i systemet SQL statements, så uden Prepared Statements, ville der kunne forekomme SQL injection og Cross-site scripting, hvilket kunne blive et stort brud på sikkerhed, samt kunne lave en masse ravage.

### Hash (\*)

For at kunne øge sikkerheden ydeligere i systemet, har vi valgt at bruge hashing for at kunne øge sikkerheden omkring brugernes og medarbejdernes login oplysninger. I det man skal registere sig eller oprette en ny medarbejder i systemet, vil passwordet være som en ren tekst, hvor der derefter kommer ned i vores logik lag, for at blive hashet.

Måden vi hasher et password på, er ved brug af sha-1. Sha-1 hasher inputet om til 160 bit eller om til et array af 20 byte. Når sha-1 har hashet inputet, i dette tilfælde et password, vil systemet derefter blive konverteret til heximal tal, som vi derefter sender nede i data laget, for at blive gemt i databasen.

Vi har dog stadig change for brud på sikkerheden, da hvis en hacker ligger på lur, mellem præsentations laget og logik laget, vil de kunne nå at se passwordet i ren tekst, før det bliver konverteret. Dette er grundet til hvornår vi hasher passwordet.

### Salt (\*)

For at bestykke brugerne og medarbejderne, har vi gjort brug at salt. Salt er princip som gør at hackere ikke kan gå ind i databasen, og se om der er flere brugere som har det sammede hashede password. Hvis to brugere har samme password, vil det blive hashet på præcis samme måde, og i sidste ende, se ens ud, dette kan salt gøre noget ved. Salt er en tilfældig genereret tekst streng, som bliver sat sammen med brugerens password, før det bliver hashet, da dette gør at alle passwords vil være unikke. Der er dog stadig en minimal, men virkelig minimal chance for at to brugere har den samme auto generede salt værdi, samt bruger det samme password, dette kan ændre Johannes Fog, mener at det er vigtigt, ved at tjekke om den nye auto genereret salt, allerede findes i databasen.

Salt bestykker brugerne og medarbejderne mod et hacker begreb kaldet ’rainbow tables’, som er en samling af knækkede hashede passwords, som kan bruges til at slå op i, for at tjekke om en brugere har et af de hashede password i et rainbow table.

## Brugertyper (\*)

Dette IT-system, har to brugertyper. Den første er brugeren, som er Johannes Fog’s kunder, som vi enten refere til som brugere eller customer. Den anden type er medarbejder, som er Johannes Fog’s ansatte, som vi refere til som medarbejder eller employee.

## SVG (\*)

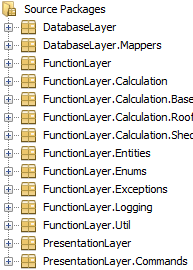
Til at tegne carportens plantegninger, bruger vi SVG (Scalable Vector Graphics), som er en god og nem måde, at tegne med. Siden at carportene har mange forskellige former, størrelser og look, ville det være svært og alt for meget arbejde, der skulle til for at have alle plantegninger gemt, derfor har vi udviklet en util klasse, som kan ud fra målene og præferencerne tegne plantegningen over carporten, fra flere forskellige sider. Mere om hvordan util klassen virker samt hvordan vi bruger den, kommer senere i kodeeksempler.

## Stykliste (\*)

Vi startede med at lave styklisten ved brug af ’ArrayList’, men vi fandt ret hurtig ud af at det var et dårligt valg, da der skulle en del arbejde til hver gang vi skulle bruge den, da den viste hver ting, så det blev hurtigt en meget lang liste af materialer. Vi prøvede flere forskellige måder, at få arraylisten til at fungere som planlagt, men after at have brugt for lang tid på det, lod vi den være sådan der. Da vi endelig kom tilbage til styklisten, var vi kommet på en god idé med at lave et ’Hashmap’, som indeholdte en streng som ’key’, og en arrayliste som ’value’. Dette viste sig at være en god løsning, da den viste en meget pænere stykliste, som var meget mere overskuelig end før. Eksempel på nuværende stykliste se bilag 5. Mere om koden til stykliste kommer i kodeeksempler.

## Arkitektur (\*)

I dette projekt har vi gjort brug af nogle arkitekturiske metoder, som gør at systemet, nemt kan opdateres, videreudvikles samt gør det nemt for nye udviklere at forstå programmet, og nemt gå finde rundt. Vi har først og fremmest gjort god brug af ’tre lags arkitektur’ og dette ses tydeligt, hvis man åbner koden i NetBeans. Eksempel fra NetBeans:



Her kan vi se at vore pakker er navngivet på dne måde at vi har laget, derefter noget information omkring hvad de indeholder.

For at bygge videre på tre lags arkitekturen, har vi valgte at benytte os af ’facade pattern’, som fungere godt sammen. Facade pattern hjælper systemet med at få lav kobling og en høj binding. Når vi skal fra præsentations laget og videre ned, så går vi gennem ’logic facade’, som derfra kalder vi nødvendige metoder. Hvis vi var brug for at skulle ned i data laget, går vi igennem ’database facade’, som at kunne komme ud til de korrekte mappers.

For at holde præsentations laget overskueligt og nemt at videre udvikle på, har vi gjort brug at ’frontcontroller pattern’, som er en enkel servlet, som tager imod alle forespørgelser. Dertil bruger vi ’command pattern’, som vi mener passer perfekt sammen med frontcontroller pattern, da man nemt kan oprette en kommando der skal gøre noget og derefter viderestille en til en ny side, med hjælp fra frontcontrolleren.

## 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 i Logging. Til sidst vil vi blive viderestillet til den side som bruger fejlen til noget kontruktivt, 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 exception godt, så man hurtigt får en god idé om hvad de bruges til. RegisterException, bliver smidt hvis der sker noget som helst galt med oprettelse at brugere eller medarbejdere. LoginException har med fejlene omkring login at gøre, så hvis man skriver forkert password, eller man ikke eksistere i systemet. CarportCreationException bruger vi til at kune fortælle brugeren, hvis de har lavet noget dårligt brugerinput, når de er igang med at lave deres carport. FogException tager sig af resten af fejlene, dem som har med systemet at gøre.

For at gøre vores exceptions bedre når de skal logges, bliver de tildelt et ’level’, som viser hvor fatal fejlen var, så når man læser loggen, kan man sortere de små fejl fra, nemt.

## Logging (\*)

For at kunne rette fejl, som vi ikke lige kender til, gør vi brug af konceptet logging, som gør at de fejl der opstår, når brugerene og medarbejderne får når de bruger systemet, vil blive skrevet i en fil, hvor det er muligt for os at læse det, så vi ved præcis hvilke fejl de får, og nemt kan teste det igennem, og får løst problemet. Eksempel på fejl i loggen:

C:\Users\Michael\AppData\Local\Microsoft\Windows\INetCache\Content.Word\LogExample.png

Her kan vi se datoen hvorpå fejlen opstod, samt kan vi se hvor slem fejlen var, i dette tilfælde var den ’info’, hvilket betyder det er bare brugerinput fejl, til sidst ser vi hvad fejl beskeden var. Dette eksempel kommer fra en der var en person, der prøvede at logge ind på en bruger eller medarbejder som ikke fandtes i systemet.

# Udvalgte kodeeksempler (\*)

Det er ikke sikkert at censor (eller eksaminator) finder alle jeres guldkorn i selve koden. Derfor er det en god ide at vælge særlige kode stumper ud og vise dem i rapporten.

De eksempler der er givet uder “særlige forhold” afsnittet kan man godt tage og illustrere med kode direkte i rapporten.

Det kommer til at virke særligt overbevisende hvis den kode man vælger ud indgår som led i et af sekvensdiagrammerne.

Der er mange af jer der vil skrive jeres ting i word eller googledocs. Vær opmærksom på hvordan i formaterer jeres kode. Man vælger ofte en lidt mindre font, en der er “monospaced” (alle bogstaver optager samme bredde). Der er også nogle der sætter små skærmbilleder fra Netbeans ind. Det er OK, men så husk at vælge et tema fra netbeans med hvid baggrund og mørke/farvede bogstaver da nogle censorer skriver rapporten ud på blækprintere som ikke gengiver lyse bogstaver på sort baggrund særligt godt.

## Carport udregning (\*)

## Stykliste (\*)

## Util (\*)

# Status på implantation

Da dette projekt som tidligere nævnt var planlagt til 5 sprints fra vores side, kom vi i tidspres grundet gruppens størrelse. Her vil vi liste de få mangler vi har i vores system, i forhold til vores egne mål.

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 prioriteringer lå i at lave et færdigt og funktionelt system. Har vi forsømt en del på JSP siderne, dette ses i form af at meget af fores kode der, er samlet med lappe løsninger. Dette kunne ville ikke være tilfældet havde vi været flere i gruppen. Eller at vi havde fokuseret mindre på at bygge en rigtig carport, med alle dele der skal bruges dertil.

PDF.  
Grundet tidspres fik vi ikke implementeret en metode hvori en bruger kan udskrive en PDF, hvori stykliste samt SVG-tegninger indgår. Igen fokuserede vi andet steds.

Mailsystem.  
Vores mål fra starten var at oprette et mailsystem, dette skulle bruges til blandt andet at sende bekræftelser på køb til brugere, skabe kontakt mellem brugere og salgsmedarbejdere. Ydermere skulle en bruger også kunne resette sit password, skulle vedkommende have glemt dette. Det ny generede password skulle så sendes gennem mailsystemet til brugeren.

Calculator.  
For at systemet skulle være 100% færdigt, skulle vi have snakket med en specialist i carporte. Da vi kun har brugt internettet til at finde information omkring udformningen af en carport føler vi at der kan være mangler i vores konstruktion.

Manglende CRUD-metoder.  
I følgende mappers har vi mangler i vores CRUD.  
AddressMapper, har vi ingen separat delete del.  
CustomerMapper, har vi ingen separat delete del.  
EmployeeMapper, har vi hverken separat update eller delete del.  
Disse mangler bliver dog hentet i vores UserMapper ved en cascade delete.  
OrderMapper, har ingen delete del.  
ProductMapper, har hverken update eller delete del.   
Disse mangler skyldes udelukkende tidspres.

Vi har dog komplet CRUD i både PartMapper og UserMapper.

SVG.  
Vi har ikke nået at færdiggøre SVG tegningerne, igen blev vi overrumplet af mængden af arbejde der kom fra sprint 3. Derfor tegner vi kun én tegning, denne bliver dog generet ud fra den nævnte Util klasse.   
Vores mål var også at genere en tegning fra siden og forfra. For at give et bedre overblik

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

# Test (\*)



Der skal være lavet test. Du kan dokumentere tests ved at beskrive i tabel form:

* Hvilke klasser er testet
* Hvilke metoder der er testet
* Dækningsgrad af dine tests for de valgte metoder og klasser

Desuden kan du beskrive hvordan i systematisk har arbejdet med at teste koden før den er blevet gjort til en del af master branch.

# Proces

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

## Arbejdsprocessen faktuelt

Sprints og userstories.

Vi havde i dette projekt fire sprints, disse er vedlagt som Bilag og kan også ses i afsnittet ”Userstories”. En komplet liste af userstories er også vedlagt i bilag, som en JSON-fil.

SCRUM Master.  
Vi brugte SCRUM master i den første uge af forløbet, 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 fungere når man er to.

PO møder.  
Hver uge havde vi møde med PO, Her var der ikke fokus på det tekniske, men derimod hvilke krav Fog havde til os som udviklere.   
Vi vil her 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 komme meget langt. Efter vores demonstration havde PO nogle få ønsker til rettelser og forbedringer. Blandt andet Farve valg og design ændringer.  
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 meget tilfredse med vores indsats og følte at mødet gik over alt forventning. PO afslørede at han ikke havde forventet at vi var kommet i mål med hele sprintet.

Daglige SCRUM-møder.  
Hver dag startede vi med at gennemgå, 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 startede hver mandag med at gennemgå 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 fordi vi kun var to i gruppen. Dette ville dog ikke være tilfældet havde vi været tre eller fire. Den største faktor for at det ikke fungerede, var at vi begge to tog meget ansvar under hele projektet. F.eks. rettede vi begge to til i Taigo.io og vi aftalte i fællesskab hvilke stykker arbejde der skulle udføres og af hvem.

Retrospektive møder.  
I vores retrospektive møder snakkede 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 nemt at nedbryde userstories i mindre tasks, men om disse opfylder alle SCRUM-kriterier vides ikke. Vi fandt dog, at vores system virkede og at vores userstories var håndgribelige og nemme at gå til.

Estimeringer.  
Selv syntes vi at vores estimeringer holdt stik over 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 og derfor har vi sat os for at lære mere om dette i fremtiden.

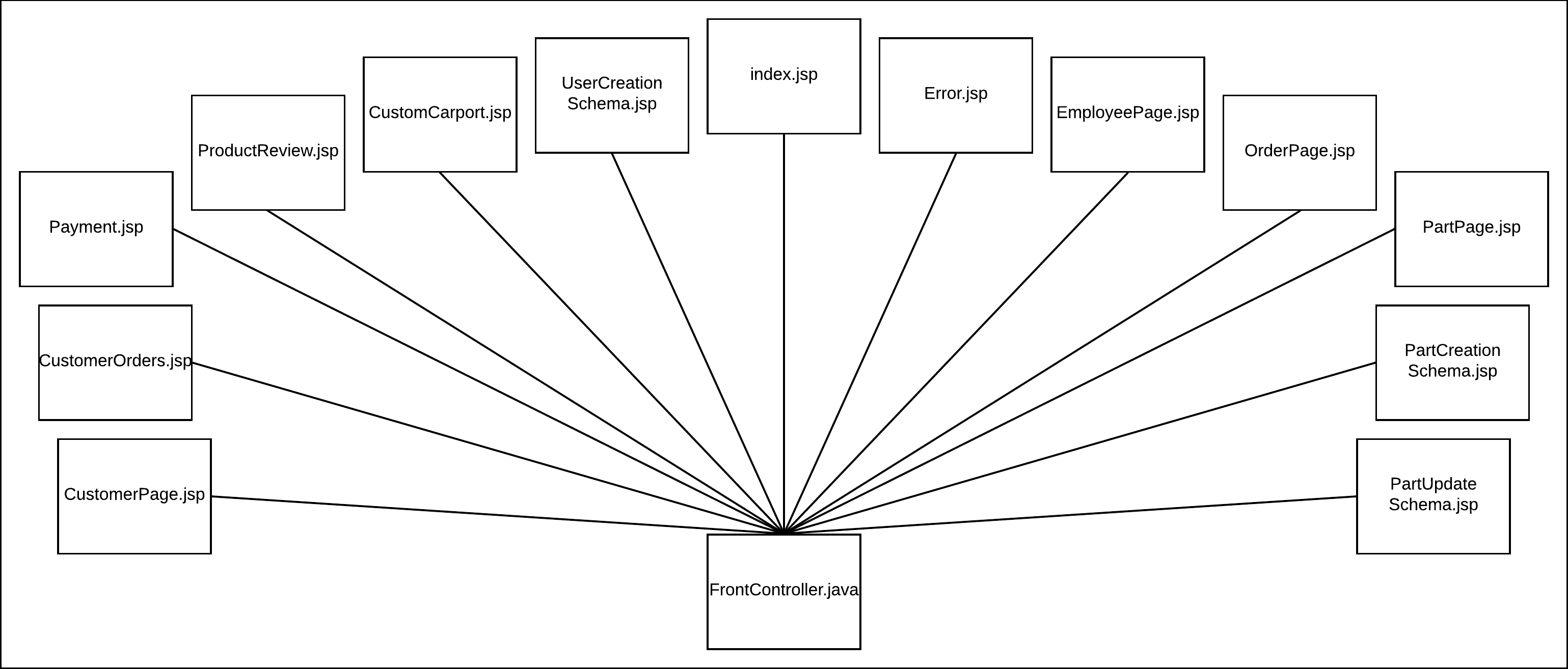
Vejledning og PO møder.  
For vores vedkommende, fungerede møderne rigtig godt. Vi værdsatte virkelig den hjælp vi blev tilbudt. Samtidigt med at vi ikke fik kørt ud af en tangent og fik lavet noget kunden ikke ønskede. Vi føler at dette var en rigtig god introduktion til hvordan det fungere i erhvervslivet.

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

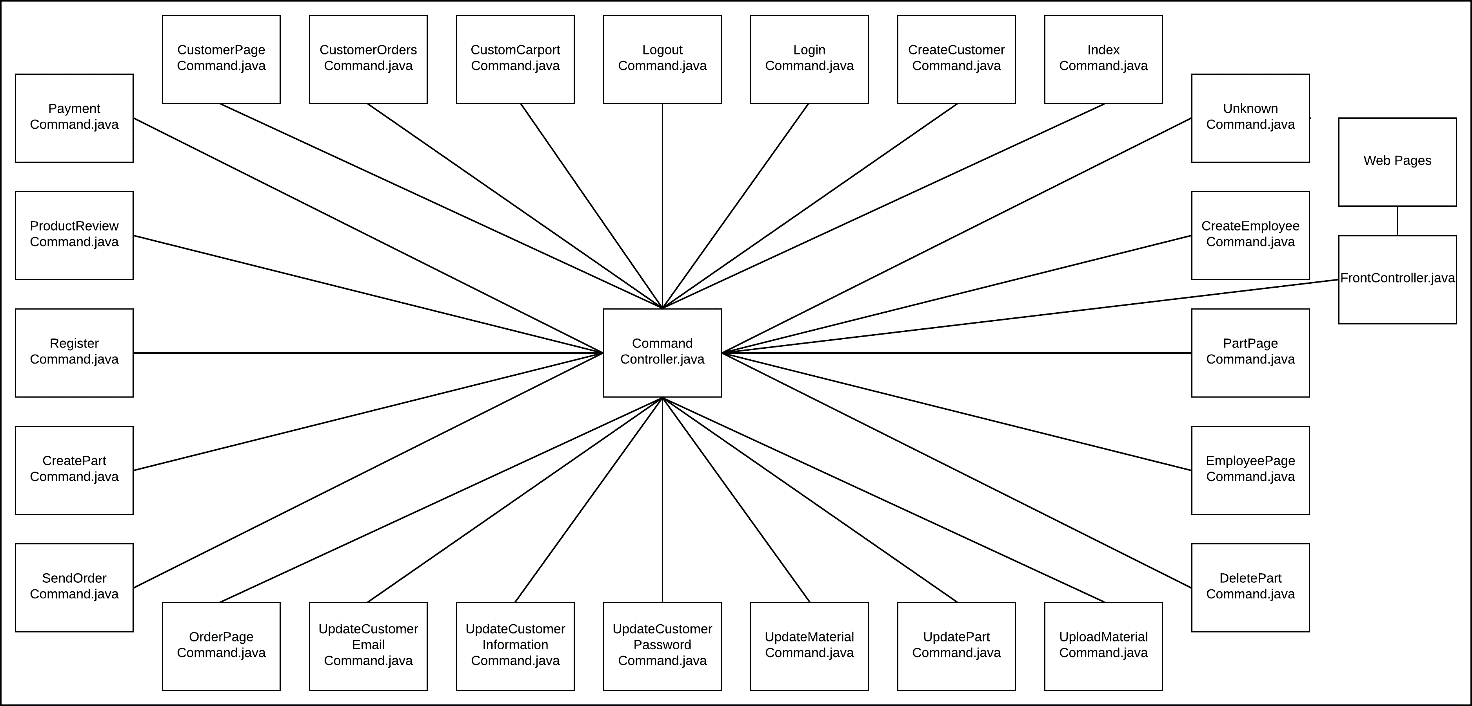
SCRUM generelt.  
Overordnet har vi indset vigtigheden af at arbejde i SCRUM og kan tydeligt se hvor vigtigt det er i projekter. Derfor vil vi i fremtiden forsøge at arbejde efter SCRUMS principper i alle projekter. Ved at vi oprettede userstories og havde klare mål for hvornår og hvordan de blev gjort færdige hjalp os utroligt meget. Det lettede arbejdet i forhold til tidligere projekter at vi havde en væsentligt bedre og stærkere struktur.   
For os virkede sprintsne i særlig høj grad, og gav klare mål og delmål dette medførte en høj arbejdsmoral og meget få, hvis overhovedet nogle perioder med stilstand i projektet.

# Bilag

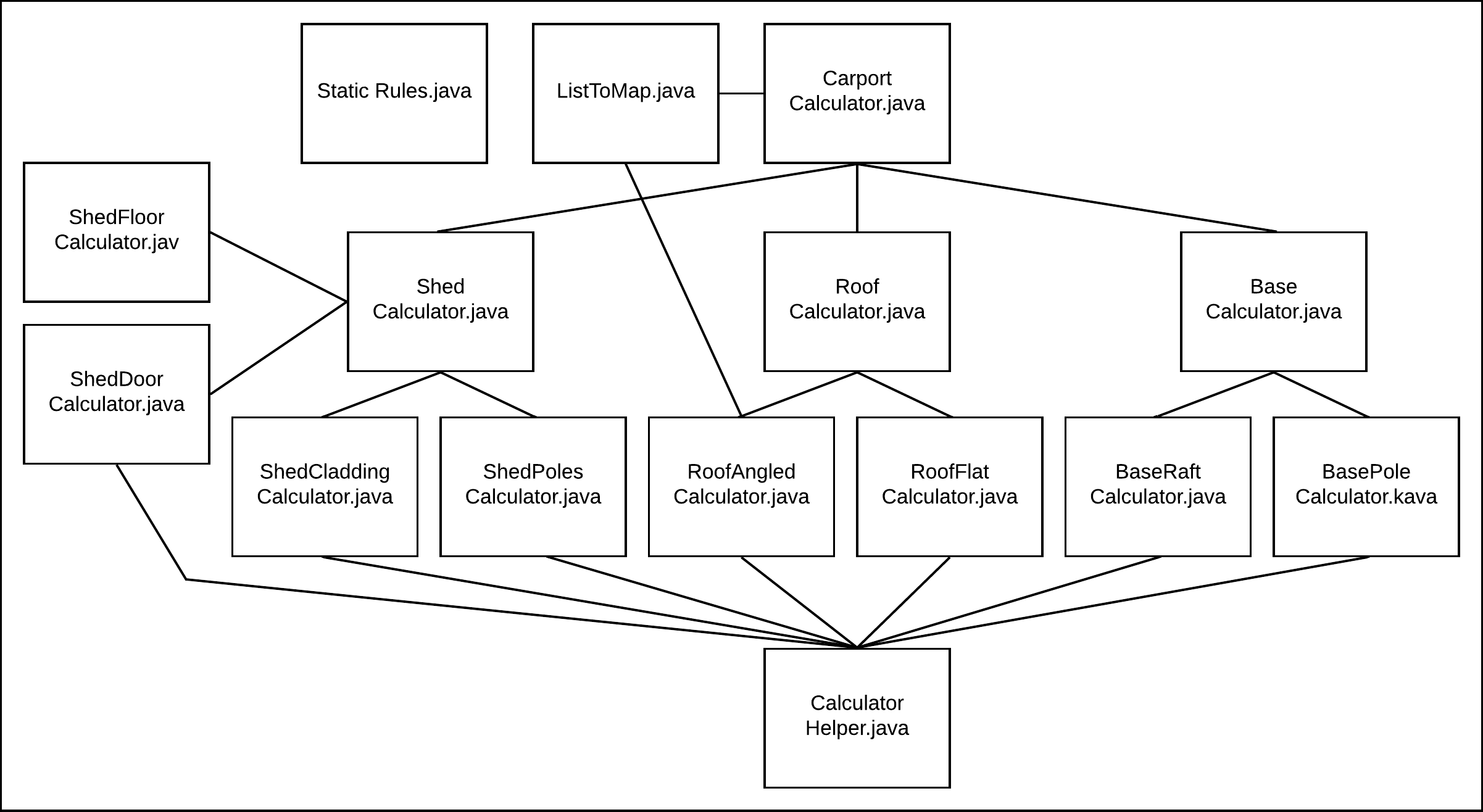
## Bilag 1 – Klasse diagram: Webpages



## Bilag 2 – Klasse diagram: Præsentations-lag



## Bilag 3 – Klasse diagram: Carport udregnings system



## Bilag 4 – Klasse diagram: Database-lag

