# Fog carport rapport

- **Klasse og gruppe**
  Eksamensplan A - klassen
  Gruppe: 9

- **Deltagere:**
  Navn: Rasmus Barfod Prætorius
  github-navn: Rasm-P
  cph-mail: cph-rp134@cphbusiness.dk

  Navn: Rasmus Hemmingsen
  github-navn: Razz
  cph-mail: cph-rh178@cphbusiness.dk

  Navn: Ditlev Andersen
  github-navn:DitlevR
  cph-mail: cph-di22@cphbusiness.dk

  Navn: Ludvig Bramsen
  github-navn: LudvigB
  cph-mail: cph-lb270@cphbusiness.dk

- **Dato:**
  Projekt: 08-04-2019
  Rapport: 20-05-2019
  Projektaflevering: 29-05-2019

- **Links**
  Github project:
  https://github.com/razz7/FogCarport.git

  Javadoc:
  https://razz7.github.io/FogCarport/

  Fog-system:
  http://167.99.209.155/FogCarport/
  Demo user: admin@admin.com
  Demo password: admin

# Indholdsfortegnelse

# Indledning

Dette projekt tager udgangspunkt i udviklingen af et byg-selv carport system til Johannes Fog byggemarked, ved brug af SCRUM som process model til udviklingen af større dele i en multibruger database applikation. Denne applikation udfolder sig ved brugen af MySql database, java servlets, JSP sider som backend, HTML samt CSS og javascript i form af Bootstrap som frontend delen. Applikationen oploades på en ubuntu-baseret droplet, hvorfra den er tilgængelig i skyen. Selve funktionaliteten af systemet fungerer som en online bestillings service til byg-selv carporte, hvorigennem en kunde kan bestille en carport med specifikke mål, som så kan blive konverteret til en komplet stykliste og kan visualiseres ved brugen af SVG grafik. Denne bestilling kan derefter blive godkendt eller redigeret af en Fog-ansvarlig som kan prissætte bestillingen og derefter godkende ordren. En carport kan bestå af specifikke brugerdefinerede mål indenfor nogle bestemte rammer, og kan både være med fladt tag eller rejsnings konstruktion, samt med eller uden skur. Både bruger og Fog-ansvarlig har adgang til forskellige dele af systemet gennem en rollefordelingen i login platformen. Projektets faglige fordeling ligger hovedsageligt i udviklingen af et produkt, samt udformningen af selve arbejdsprocessen og dokumentation til projektet og arbejdsproces i form af en denne rapport.

## Baggrund

Virksomheden Johannes Fog er et udbredt trælast og byggecenter der beskæftiger sig med med salg og rådgivning indenfor træ, byggematerialer og en bred variation af f.eks. maling, bad og VVS, beslag, elartikler og lamper samt haveredskaber, grill og havemøbler. I sammenhæng med dette projekt skal Johannes Fog bruge en online løsning til deres efterhånden uddaterede system til bestilling af byg-selv carporte.

De grundlæggende krav til systemet ligger i udviklingen af et testsystem som delvist opnår de pålagte systemkrav fra Fog's side som under projektet yderligere klargøres i takt med projektets fremgang og SCRUM aftaler med product owner/user stories. Her blev det klargjort at product owneren ønskede en platform både tilgængelig af kunder og de administrerende medarbejdere hos fog, som skulle kunne navigere gennem systemet gennem en overliggende hovedmenu for hver rolle. Det blev klargjort at kunder skulle have mulighed for at kunne oprette en bruger og logge ind, udfylde et skema med valgfrie mål for sin carport, og så blive videreført til en visuel tegning som brugeren gennem menuen ville kunne navigere sig fra og til samt se deres tidligere carporte og visualiseringer. Det skulle også være muligt for en bruger at klikke sig ind på en side med de forskellige lagermaterialer som Fog har, og for en administrerende medarbejder at kunne redigere i disse materialer gennem en mindre editor. Den Fog-administrerende skulle også have mulighed for at klikke sig ind på en specifik ordre stykliste, prissætte den og derefter færdiggøre ordren gennem en redigeringsside. Disse elementer skulle derudover have en attraktiv og gennemført styling med basis i den visualisering som Fog bruger på deres egne hjemmesider.

## Teknologi valg

Af selve de tekniske formelle systemkrav, skal systemet implementere en MySQL database, samt etableres i en logisk multilags arkitektur, der kan køres på en java server. Systemet skal gøre brug både java classes, servlet og JSP sider, samt kunne bruge i en række forskellige browsere heriblandt Google Chrome og Firefox. Det udviklede system skal også etableres i skyen på en Digital Ocean droplet, og selve source koden fra projektet skal være tilgængeligt i et GitHub repository hvori der også skal ligge Javadoc for systemet.

Nedenfor ses de programmer og versioner som er blevet brugt i udarbejdelsen af produktet:

Netbeans 8.2 Java EE version, med Apache Tomcat 8.0.27 plugin.
MySql Workbench version-8.0.12 build -13312926 CE.
Digital Ocean Droplet med Ubuntu 18.10 x64 og installation af Java jdk-11.0.1,
-Mysql-server version-14.14 distribution-5.7.25, og apache-tomcat-9.0.16.
Git, Git-Bash med git-version 2.20.1.

## Overordnet beskrivelse af virksomheden

Som forberedelse til denne opgave benyttede vi os af flere metoder til at analysere vores kunde og på denne måde udforske de muligheder og udfordringer der kunne vise sig at være i forbindelse med afvikling af opgaven samt de personer der bliver påvirket af udvikling og brug af et nyt IT-system. En meget indlysende fordel ved dette projekt er at Fog, som en erfaren træhandel med mange år på markedet, allerede har et IT-system der kan modtage elektroniske ordrer fra kunder, på trods af at en del af processen, som at overføre ordremål til beregningssystemet, skal foretages manuelt. Dette er dog stadigvæk en stor hjælp for os da det giver os et strukturelt skelet over hvordan systemet skal se ud og hvordan opgaven skal fuldføres. På trods af at vi ikke har haft adgang til koden har det alligevel givet os et overordnet billede af, hvad Fog gerne vil have fra vores nye system. Dette kommer også af at mange af de funktioner vi satte os for at udvikle allerede eksisterede i Fogs gamle system, mens vores opgave i sådanne tilfælde har været at forbedre eller modernisere dem. Det betyder dog også at vores system har en standard som det utvivlsomt kommer til at blive holdt op imod og vurderet i forhold til. Derfor er det vigtigt at der også fokuseres på, hvordan vi kan forbedre det nye system i forhold til det gamle. Dokumentation i form af Overordnet Interessent-beskrivelse af virksomheden, Interessent-analyse for de implicerede aktører og deres interesser, SWOT-analyse over kunden set fra kundens perspektiv, og SWOT-analyse over projektets chance for at blive en succes, fra før projektets igangsættelse, kan henholdsvis ses i bilag 1,2,3 og 4)

## Arbejdsgange der skal IT-støttes

En anden del af forberedelsen til dette projekt involverede beskrivelsen af Fog's overordnede arbejdsgange før og efter systemets implementation, gennem as-is og to-be aktivitetsdiagrammer.

As-is aktivitetsdiagrammet beskriver hvordan Fog's arbejdsgange som virksomhed står til nu, før systemets implementation. Fog it situation tillader at man i øjeblikket kan navigere deres webside og ud fra en række forskellige carport forslag kan vælge at lave en byg selv carport af 2 typer. Brugeren bliver her bedt om at indtaste nogle mål som efter færdiggørelse bliver sendt via mail til Fog ansvarlige. Den ansvarlige hos Fog indtaster derefter manuelt målene i et forholdsvist gammelt og uddateret it system, som med nogle materialer som ikke kan ændres udregner en carport stykliste. Herefter kan den ansvarlige så gennem et andet program visualisere denne stykliste og tilknytte den til ordenes byggevejledning. I mellemtiden aftaler en salgsperson en pris med kunden, som bliver tilføjet ordren og evalueret af Fog ansvarlige. Hvis ordren godkendes sendes den videre, bliver samlet med en stykliste og færdiggjort, hvis ikke, skal den ansvarlige selv indtaste målene manuelt i systemet igen og processen starter forfra.

To-be aktivitetsdiagrammet beskriver i modsætning, hvordan systemet antages at forbedre fogs arbejdsgang efter systemets implementation. Her starter kunden med at navigere til shop siden, hvor de kan udfylde formen og oprette en ordre med brugerdefinerede mål. Ordren bliver derefter automatisk kørt gennem en carport algoritme som danner en stykliste ud fra de brugte materialer. I denne stykliste har Fog ansvarlig mulighed for at ændre i de vedhæftede materialer, alt efter navn, id, beskrivelser, mål, og kvantitet. Ved ordrens oprettelse kan en sælger tilknyttes som kan tage kontakt til kunden og forhandle en pris mens Fog ansvarlig internt i systemet kan få frembragt en visualisering af den bestilte carport ved at trykke på se order. Herefter kan den ansvarlige dømt ud fra styklisten, visualisering og korrekte navne, id'er, beskrivelser, mål, og kvantiteter af materialer, evaluere ordren. Hvis ordren godkendes videreføres styklisten og visualiseringerne til en byggevejledning som så kan sendes til kunden og markeres i systemets historie som færdiggjort. Hvis ikke kan den Fog ansvarlig blot danne en ny stykliste eller redigere i materialerne på den eksisterende, og få systemets automatiske proces til at samle og visualisere ordren så den er klar til godkendelse.

Yderligere værktøjer i form af forebyggende use case samt de komplette as-is og to-be aktivitetsdiagrammer kan ses ved henholdsvis bilag 5, 6 og 7.

# SCRUM

I afvikling af denne opgave har vi benyttet os af SCRUM. SCRUM er en agil udviklingsmetode der bruges til at dokumentere og estimere kundekrav, samt hvordan udviklere kan konvertere disse krav til et operativt format til at kontrollere deres arbejdsrutine. Altså at vurdere, estimere og håndtere udviklingen.

SCRUMs rollefordeling udfolder sig inden for rollerne, product owner, som ses som den ansvarlige projektejer/kunde, SCRUM teamet som er det pågældende project team, og SCRUM master står for organisering af SCRUM processen.

Gennem projektets udviklingsperiode, har vi gennem daglige SCRUM møder udvalgt og estimeret dagens arbejde i henhold til det aktuelle sprint og de user stories som vi havde lagt i sprintet. Disse møder varede som regel 10 - 15 minutter, hvor teamet kunne rapportere fremgange og forhindringer til hinanden. Dette involverede hovedsageligt spørgsmål vedrørende, hvad den enkelte har opnået, hvad der skal afsluttes inden næste møde, og hvad hvilke forhindringer der er fremkommet undervejs. Med SCRUM master som det produktive baghoved i den daglige SCRUM har vi kunne fordele det udvalgte sprint fra backlogen og overholde vores leverancer i forhold til deadline. Ud over at være et godt værktøj til at optimere gruppens sammensatte overblik, har det også hjulpet os til at effektivisere os selv i nedbrydelsen af større opgaver og færdiggørelsen af de tasks som hver user story indeholder.

Fra projektstart har vi ugentligt haft gennemgående møder med Product Owner hvor vi har fremlagt systemets nuværende status på baggrund af det forrige sprint samt aftalt og sammensat kompositionen af det næste sprint ved sprint-planning. Til selve møderne har product owner her haft muligheden for at kunne byde ind med konkrete krav til den fortsatte udvikling af systemet samt at vi har kunne stille product owner spørgsmål vedrørende funktionalitet og udseende. Disse krav og tilføjelser er så blevet tilføjet til sprints i form af user stories, med nogle konkrete acceptance criteria som den enkelte story skal overholde. Efter hvert sprint har vi set tilbage på, hvordan den foregående uge forløb, med udfordringer og løsninger, hvad der gik godt og hvad der skal forbedres til det næste sprint. Dette hjalp os med at hele tiden vurdere, hvad skal vi fortsætte med at gøre som vi gjorde i det forrige sprint og hvad skal vi skal gøre anderledes i det næste sprint.

Nedenstående user stories er eksempler på hvordan en reaktant ville kunne benytte forskellige funktioner/implementationer for at kunne gøre eller resulterer i et eller andet specifikt. Dette danner et effektivt krav om en funktion, der så skal sammensættes af det pågældende team, som så selv tilføjer mindre tekniske tasks til hver user story hvortil de enkelte teammedlemmer så kan blive tilføjet. Et eksempel på en user story, som i vores tilfælde blev lavet gennem den online planlægnings platform Taiga, kunne blandt andet se ud som dette eksempel fra sprint 2.

**User story:**
Som Fog ansvarlig eller salgsmedarbejder vil jeg gerne kunne oprette en stykliste, så jeg kan liste materialerne til enkelte carporte.
**Acceptance Criteria:**
1. Jeg vil gerne kunne oprette en visuel stykliste gennem systemets funktion.
2. Jeg vil gerne kunne igangsætte de enkelte styklister og deres status, efter grundig samtale med kunde.
3. Jeg kan ikke direkte redigere i selve udregningerne af styklisterne gennem denne oprettelse.
**Tasks:**
Business lag der laver stykliste objekter. 5 timer.
Udregning af stykliste ud fra Java algoritme med mål der er givet. 20 timer.
Front end funktion der generer stk. liste. 2 timer.
Stykliste i database 3 timer.
Front end Interface til redigering af stykliste data. 5 timer.

Strukturen for sammensætning af user stories som det førnævnte eksempel set ovenfor, er blevet brugt gennemgående gennem alle de andre user stories for alle projektets sprints. Da disse sprints i henhold til product owners ønsker om projektet, hovedsagelige involverede udførelsen af de mest essentielle og grundlæggende funktioner først, vil man for project backlogens sprints kunne genkende en forskel i størrelse og mængde af user stories. Dette forekommer hovedsageligt af at de første sprints tog udgangspunkt i mere krævende tasks for implementationer der tit var mere essentielle og omfattende end de senere og mindre prioriterede implementationer, hvilket også passer med product owners prioritering af elementer gennem projektet.

Alt dette i sammenhæng med opnåelsen af det førnævnte prædefinerede sprint har så været grundstenen for planlægningen af det næste sprint, om udvalget af de mest essentielle funktioner eller tilføjelse som de første userstories til at blive planlagt. På den måde vil produkt owner's mest prioriterede elementer altid være de første til at blive planlagt og implementeret i systemet. Dette har derved gjort det muligt for os at planlægge det ugentlige arbejde og optimere leveringen af færdige implementationer, til produkt owner's tilfredsstillelse.

I forbindelse med sprint retrospective møder, har vi ved afslutningen af hvert sprint, haft dybdegående samtaler om status vedrørende sprints, user stories og opnåelsen af de igangsatte implementationer samt selve anvendelsen af SCRUM-processen. Her har vi reflekteret på hvordan vi taklede disse opgaver og hvad der blev opnået samt hvad der måske stadig mangler implementation og hvilke forbedringer der kunne involveres i processen. Mange af disse møder tog derfor udgangspunkt i arbejdsmetoden og forbedringen af det fortsatte arbejde. I tilfældet af ukomplette userstories eller implementationer, blev til disse møder der i sammenhæng med det næste planlagte sprint oprettet og fordelt nye user stories som så kunne videreføres i nye implementationer, efter product owners vision.

# Krav

Fog's vision med dette system er at optimere deres lidt forældre behandling af byg-selv carport systemet, som kræver flere ressourcer end praktisk nødvendigt og ikke er kompatibelt med mange af deres eksisterende it ressourcer. Denne optimering involvere både at bringe systemet online, så det er tilgængelig for forskellige maskiner, og at involvere automatisering af eller førhen manuelle funktioner. Den værdi som systemet derfor tilføjer Fog som virksomhed, er hovedsageligt grundet i optimering af deres bestillingsproces, så der kan skabes flere resurser til behandlingen og rådgivningen af den enkelte kunde, samt øget kapacitet af kundebestillinger.f

Kunder skal kunne oprettes i systemet, være i stand til at logge, kunne indtaste mål og placere en ordre på en carport og se en visualisering af deres bestilling. Systemet skal kunne udregne en stykliste med materialer carporten skal bestå af. Forskellige medarbejdere skal også være i stand til at se ordren og prissætte og til sidst godkende den.

I forbindelse med SCRUM møder med product owner og etableringen af en lang række fundamentale user stories, opretter disse de formelle krav til systemet med det grundlag at de beskriver hvad en reaktant skal kunne gøre i/med system og hvorfor. De pågældende user stories brugt og dirigeret i henhold til product owners ønsker for dette projekt, står fordelt i deres respektive sprints som følgende, eller kan findes i deres fulde længde under bilag 8.

**Første sprint:**
Som Fog medarbejder vil jeg gerne visuelt kunne se fog-lagerets gemte materialedata så jeg kan vurdere dets relevans for en stykliste.

Som Fog medarbejder vil jeg gerne kunne ændre priser, navne og andet materialedata i systemet så jeg kan holde det opdateret og hvis der kommer nye produkter eller ændringer som vi vil bruge i stedet for noget andet.

**Andet sprint:**
Som Fog medarbejder, vil jeg gerne visuelt kunne se styklister, så jeg kan tilgå materialerne til den enkelte carport.

Som Fog ansvarlig eller salgsmedarbejder vil jeg gerne kunne visuelt se og redigere styklister, så jeg kan ændre i dem hvis situationen kræver det.

Som Fog ansvarlig eller salgsmedarbejder vil jeg gerne kunne oprette en stykliste, så jeg kan liste materialerne til enkelte carporte.

**Tredje sprint:**
Som Fog medarbejder vil jeg gerne kunne se alle stk. lister og tidligere ordre, så jag altid kan gå tilbage og se tidligere produkter.

Som fogmedarbejder vil jeg kunne se den specifikke carport-tegning scalere sig i henhold til carportens mål, så jeg nøjagtigt kan se de visuelle refferencer mellem styklisternes forskellige mål.

Som bruger af platformen vil jeg kunne se en grafisk 2D tegning af den udvalgte carport carport fra siden, så jeg kan visualisere carporten.

**Fjerde sprint:**
Som bruger vil jeg gerne kunne logge ind i systemet, så jeg kan tilgå min konti.

Som kunde vil jeg gerne kunne oprette en bruger i systemet, så jeg kan tilføje mig til en konti.

Som sælger vil jeg kunne vurdere carporte, så jeg kan tilpasse kunden en bedre pris.

Som Fog ansvarlig vil jeg gerne kunne se og godkende carport ordre, så jeg kan sikre kvalitet og validitet af produktet.

Som kunde vil jeg gerne kunne indtaste mine egne mål og en carport type på Fogs hjemmeside, så jeg kan oprette en carport ordre.

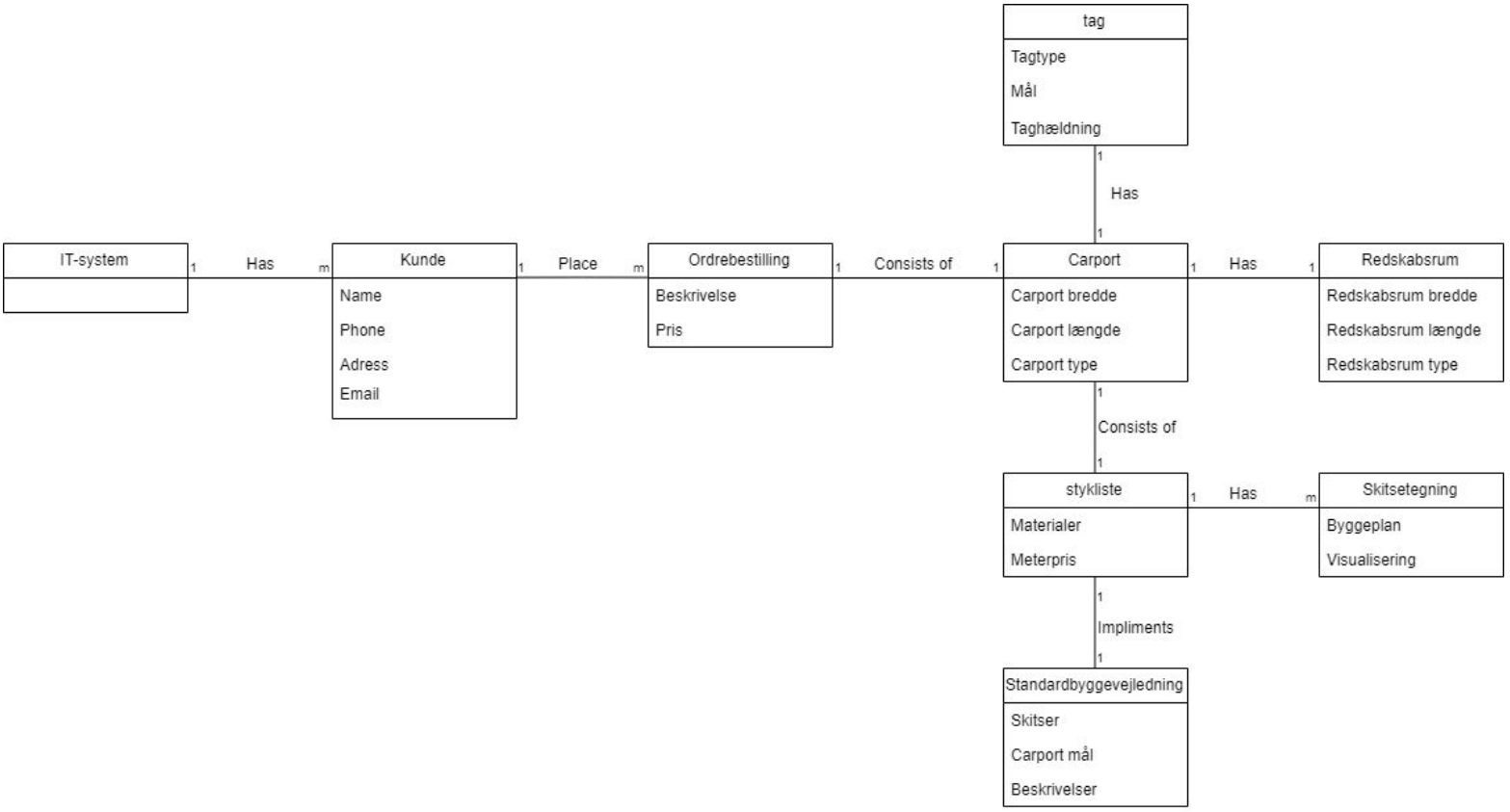
Som Fog medarbejder vil jeg gerne kunne tilgå kundernes ordre visuelt, så jeg bedst muligt kan assistere dem i deres køb.

Som kunde vil jeg gerne visuelt kunne se mine ordre, så jeg kan se deres status, brugerdefinerede mål og carport.

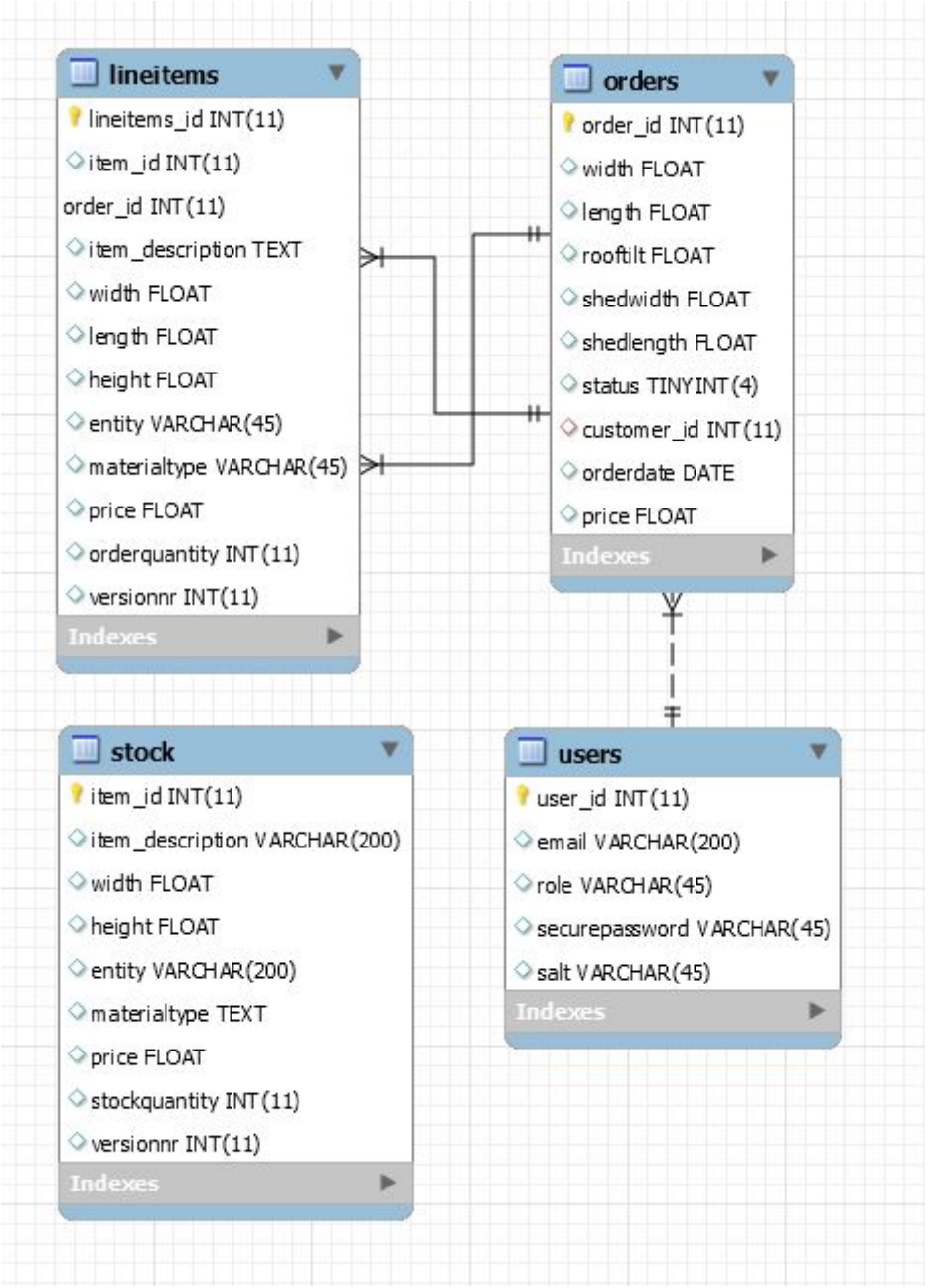
Som henholdsvis kunde og Fog medarbejder, vil jeg gerne kunne tilgå systemet online, så jeg kan tilgå systemet forskellige steder fra.

# Domæne model og ER diagram

Som del i sammensætning for projektets program koncept, sammensatte vi følgende domænemodel i starten af projektet. Denne konceptmodel tager udgangspunkt i en beskrivelse logik for den grundlæggende beviste klasse sammensætning for programmet. Her ses både den grundlæggende aktivitet og relationer mellem klasserne og koncept metoder, som de fortolker programmet. Dette diagram var en del af planlægning for programmets sammensætning.




ER-Diagrammet for domæne og database er interessant da det i et stort omfang er grundlaget for resten af systemet, med tabeller og relationer der fortæller noget om hvad systemet arbejder med. De følgende modeller og diagrammer beskriver grundlaget for relationer og den centrale data for det meste af systemet. Disse tabeller og relationer giver en ide om hvad systemet gennemgående generelt håndterer og arbejder med.




Vores database tager udgangspunkt i at hvis data'en ikke passer, bliver der ikke lageret. For at oprette en ordrer skal man have en customer_id som man får ved oprettelse og som er "auto_increment". Ordren bliver kun lavet hvis det id man indsætter som customer_id eksisterer i customer tabellen, og det samme gælder lineitems. Denne database har en meget opstillet vej, der bliver lagt stor vægt på at kun relevant data bliver indsat og i tilfælde af at data'en ikke passer, bliver der smidt fejl.

Gennemgående er der brugt "on delete cascade" på alle primær nøgler. Det gør det både praktisk og overskueligt. Det er med til at sørge for at det data man har er relevant, og at hvis man slettet noget data fra en tabel bliver alle dens nøgler i andre tabeller slettet. Hvis en kunde slettes fra systemet/databasen vil alle personens ordrer bliver slettet, i tilfælde af at man ville have en ordrehistorik ville dette hurtigt kunne tilføjes.

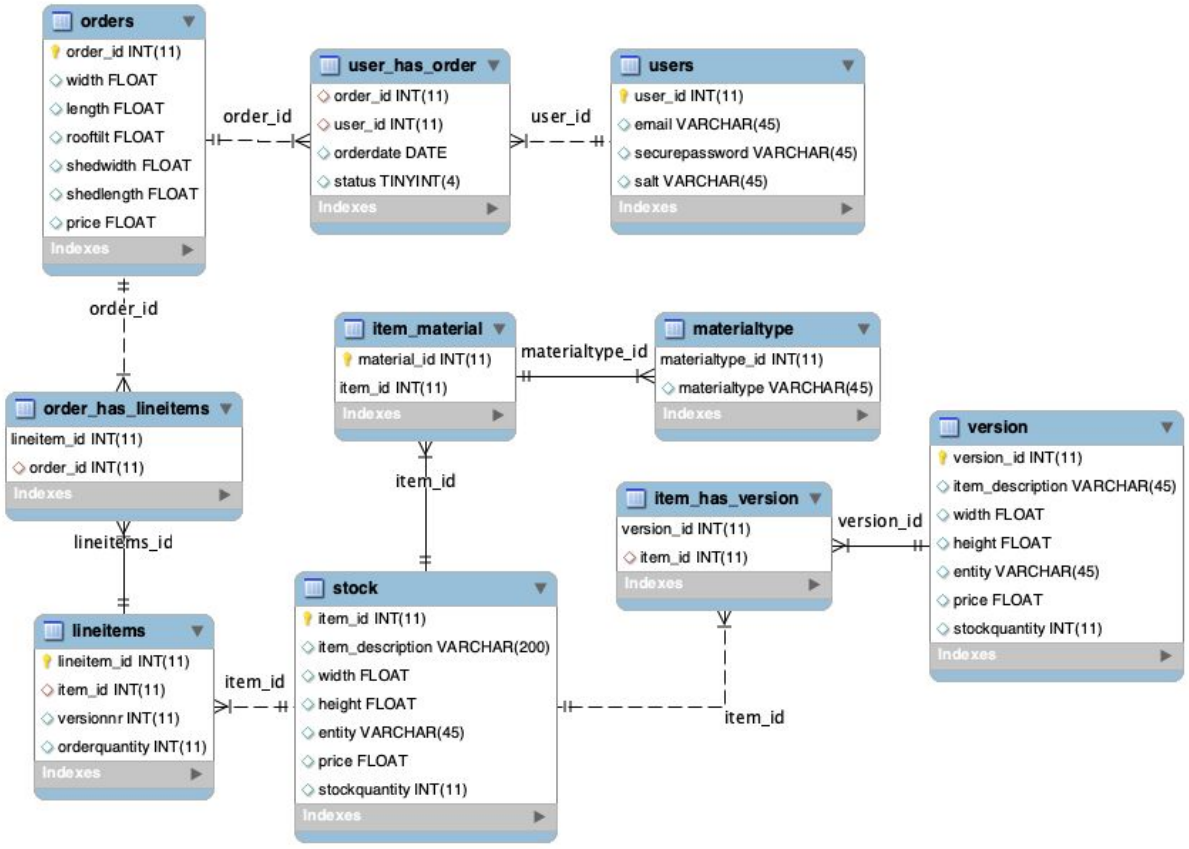
Tabellen “lineitems” som har tilknytning til orders tabellen med order_id som fremmednøgle, opfylder ikke kravene for 1. normalform. Det er gjort med hensigt på at “stock” tabellen er de vare som “lineitems” består af, er dynamisk og hyppigt skifter indhold. Derfor har vi valgt ikke at lave item_id til en fremmednøgle i lineitems, da den lineitem som bliver gemt på daværende tidspunkt kan have ændret sig. Vi har med henblik på at holde styr på versionen af data’en tilføjet attributen versionnr. Når en en funktion bliver kaldt, som trækker information fra “stock” tabellen og brugt til at lave en ordrer i form af en lineitem, er det den nyeste version der bliver brugt. Det er med til at give overblik over relevansen af den tilknyttede data til det givne lineitem. På den måde har lineitems og de items der er i stock tabellen ikke så meget tilknytning. “Stock” tabellen fungerer som en repræsentation af hvordan lagerbeholdning ser ud lige nu. Derfor kan et lineitem og item fra “stock” ikke sammenlignes desto mindre de er samme version.

Lineitems har lineitems_id og order_id som primærnøgler. lineitems_id er sat til at lave auto_increment. Grunden til at vi har valgt at have disse to som primærnøgler er dels for at have en unik nøgle til hvert lineitem og fordi at flere lineitem med samme order_id godt kan have samme item_id. For at øge effektiviteten og responstiden af databasen, har vi forsøgt at indeksere lineitemstaballen så forespørgsler bliver kaldt i henhold til order_id og lineitems_id.

Orders tabellen er på 2.normalform men da vi har valgt at have customer_id som en attribut kommer der transitive funktionelle afhængighed og den kommer ikke på 3. normalform. Det kunne løses ved at lave en tabel som bindeled mellem users tabellen og orders tabellen. På den måde vil alle non-prime attributer have direkte tilknytning til primær nøglen(order_id).

En alternativ database hvor normalformerne bliver overholdt, kan ses nedenfor. I denne alternative database er alle de nøgler som før ikke var fuldt funktionelle af primærnøglen lagt ud i deres egen tabel. For at undgå redundans er kolonnen materialetype lagt ud i sin egen tabel med reference i form af foreign key til item_id. Det gør stock material kommer på 3. normal form. I forhold til versionsstyring er tabellen "version" blevet tilføjet. Her bliver en item med samme attributter som i stock tabellen lageret. For at undgå funktionelle afhængigheder er versionnr blevet tilføjet til lineitems tabellen i form af en attribut uden key. Ved hentning af den specifikke version af et item, tilknyttet til et lineitem ville man skulle hente versionnr gennem stock-tabellen. Det ville koste noget ekstra tid, men ville være en løsning til hvordan man kunne undgår redundans i forhold til versioner af forskellige items.

For at få orders på 3. normalform er der blevet tilføjet tabellen "user_has_order" hvilket gør at der ikke er nogen attributter i orders der ikke har direkte tilknytning til order_id. Alle kolonnerne er fuldt funktionelle afhængig af primærnøglen.

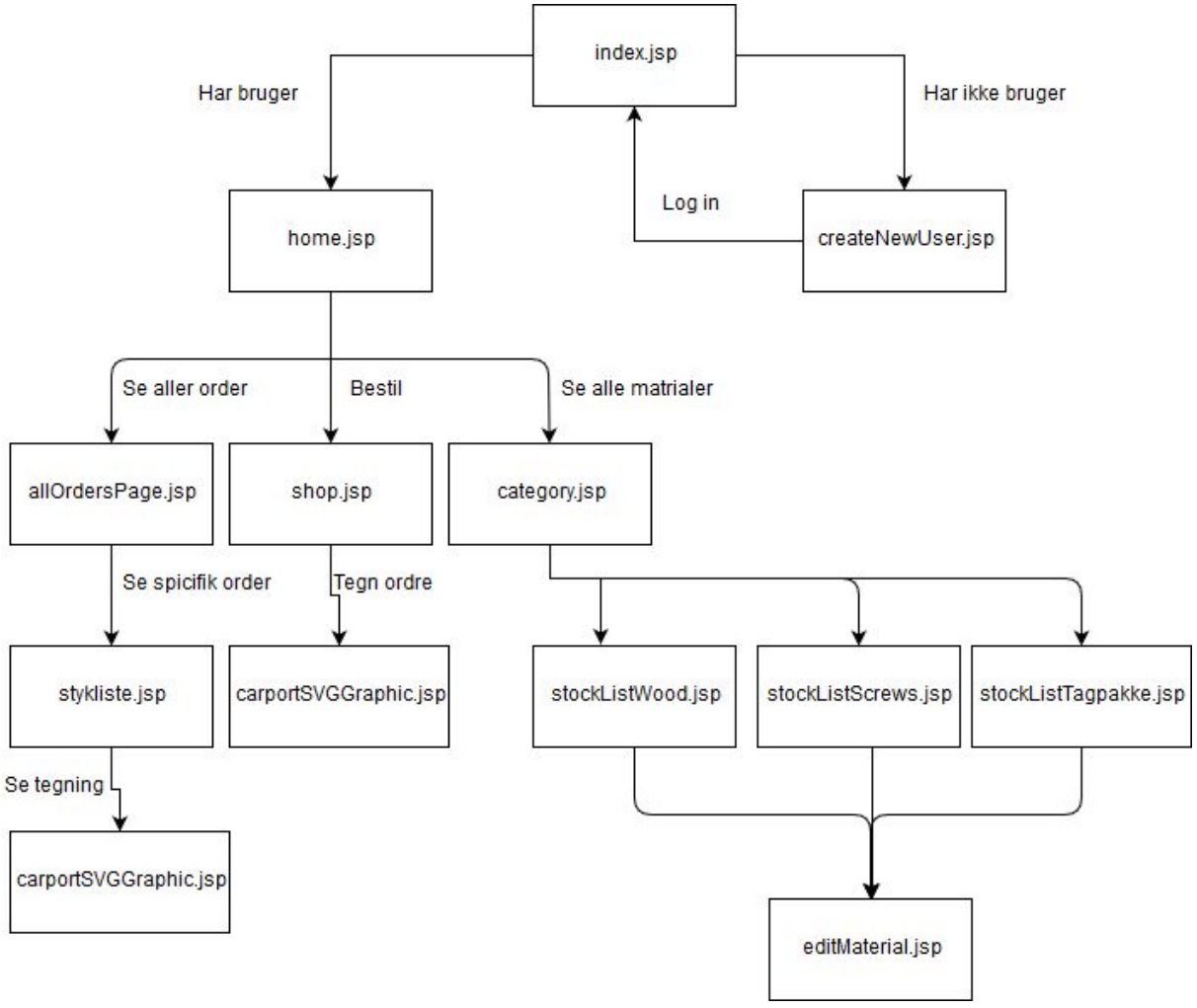



Ulempen ved denne database kunne blandt andet være at det ville tage længere tid at hente noget fra databasen. De mange tabeller ville resultere i at man skulle lave mange forespørgsler som ville koste tid og sidste ende gøre systemet langsom.

## Navigationsdiagram

Navigationsdiagrammerne nedenfor viser i et mere overskueligt format, hvordan en bruger kan navigere fra og til de forskellige sider i systemet, samt hvilke brugere, der kan få adgang til hvilke sider. Da grundlaget for navigationsdiagrammerne hovedsageligt tager udgangspunkt i bedst muligt at beskrive navigationen mellem vores jsp sider, har vi valgt følgende illustrationer, hvorpå vi kan tolke vores system i diagramform.

Navigationsdiagrammet der ses her under viser den generelle navigation i vores program. Det viser hvordan man fra index.jsp, kan hvis man ikke har en bruger komme til createNewUser.jsp og oprette en ny bruger. Herefter kommer man tilbage til index.jsp og logge ind. Vi kommer så til home.jsp hvor vi har vores 3 kategorier vi kan komme til allOrdersPage.jsp for vi kan se alle ordre og redigere dem og se tegninger og stkLister. Vi kan også prissætte ordre. Vi kan fra home.jsp også komme til shop.jsp hvor vi kan lave en ordre, se stklisten og se tegningen over den ordre. Vi kan ikke redigere ordren på denne side det skal man til allOrdersPage for at gøre. Den sidste mulighed fra home.jsp er at komme til category.jsp det er her vi kan se alle materialer på lager. Alt træ, alle skruer og alle tagpakker. Vi tjekker på alle siden om der er en bruger logget ind hvis ikke så kommer tilbage til index.jsp.



Fra home.jsp bruger vi en fælle navigationsbar der kan sende os til shop, allorders og home siden. Denne bar er på alle sider. Den bliver scalet med bootstrap så den passe lige meget hvor stor display man viser det på. Baren bliver vist på hver side ved at hver eneste .jsp side i vores program indeholder “include” direktivet. “jsp:include” blive sat som en tag i en .jsp side, sammen med linket til en anden .jsp. Når .jsp siden med “jsp:include” taggen bygges vil den blive sammenflettet med .jsp siden i “jsp:include” tagget.

På denne måde har alle .jsp sider i vores program sætningen

```
<jsp:include page='/JSP/sitemenus.jsp'></jsp:include>
```

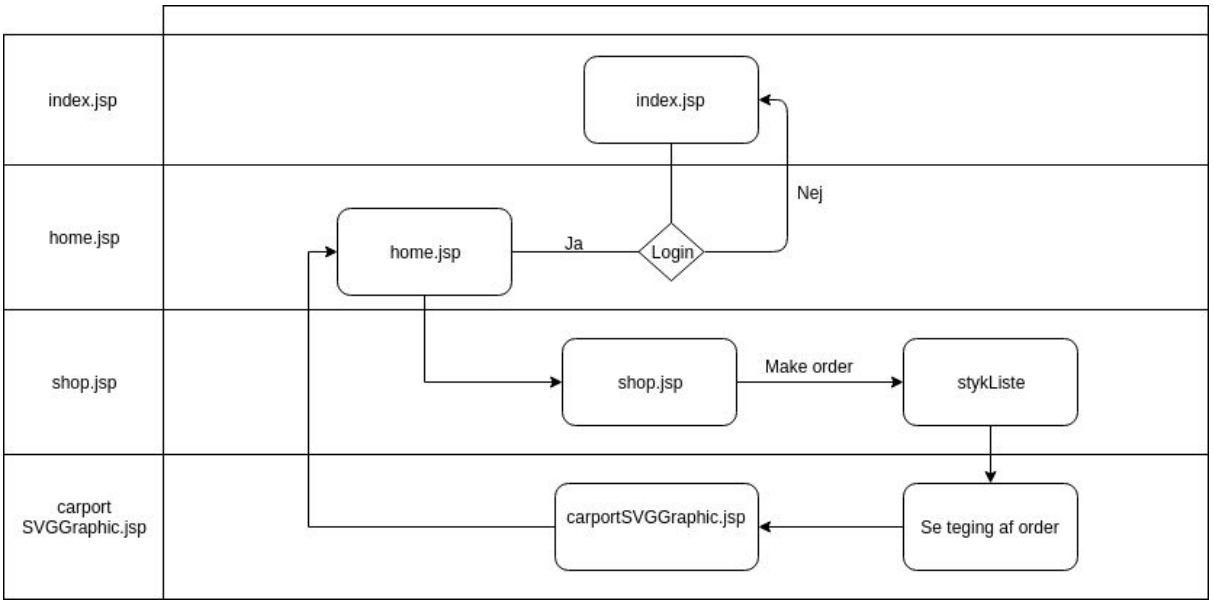
Nær starten af siden, hvor baren derfor selv er placeret.
Menuen indeholder <a href> links til de forskellige sider der vises hen til.

```
<li class="nav-item">
    <a class="nav-link" href="/FogCarport/FrontController?command=shop">Shop</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="/FogCarport/FrontController?command=AllOrders">Allorders</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="/FogCarport/FrontController?command=logout">Logout</a>
```

Disse links sendes uden at være afhængig af tidligere indtastet input og derfor kan de altid tilgås ligegyldigt hvor i systemet man er.

Det følgende diagram ses et mere konkret navigationsdiagram i såkaldte swimlanes, der hjælper med at forklare hvordan genvejene mellem de enkelte jsp sider interagere med hinanden samt under hvilke forhold gennem systemet.
Denne lidt forsimplede version er måske nemmere at forstå for nogen, og giver et mere fuldstændigt indtryk, hvilket grunden til at vi har valgt at inkludere den. Her har vi vores forside som startpunkt, og alt efter om brugeren har et login eller ej, kan de komme rundt til forskellige sider.

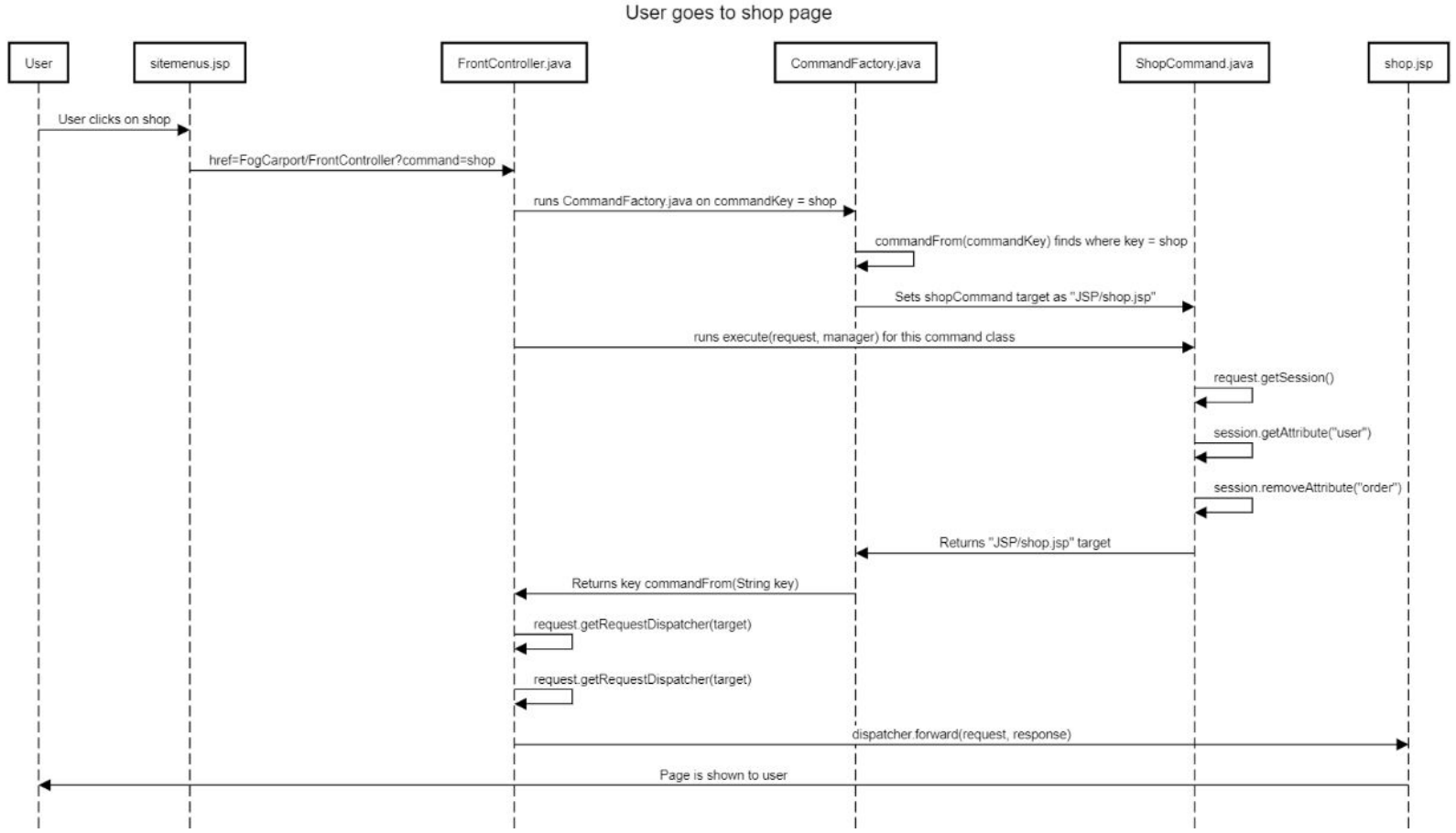
Det viser hvordan man processen vil være hvis man skal lave en ordre, se en tegning og komme tilbage til forsiden. Fra index.jsp skal vi logge ind, og så fra home kan vi komme til shoppen, lave en ordre og se tegningen over denne ordre.




Systemet og dets elementer ligger som en war. fil i en TomCat manager der ligger på en af vores droplet servere. Behandlingen af disse elementer foregår gennem forskellige java Objekter instanser baseret på brugerdefinerede input som gennem forskellige algoritmer kan behandles og skabe stykliste og bruger objekter som i sammenhæng med et order objekt kan eksekveres gennem forskellige aritekturlag som vises gennem en række frontend jsp løsninger. Selve behandlingen af http requests og responses håndteres i en front kontroller som bliver opereret gennem et command system med udgangspunkt i at samle backend metoder som skal køres inden der henføres til en bestemt front end jsp-side også kaldt 'target'.

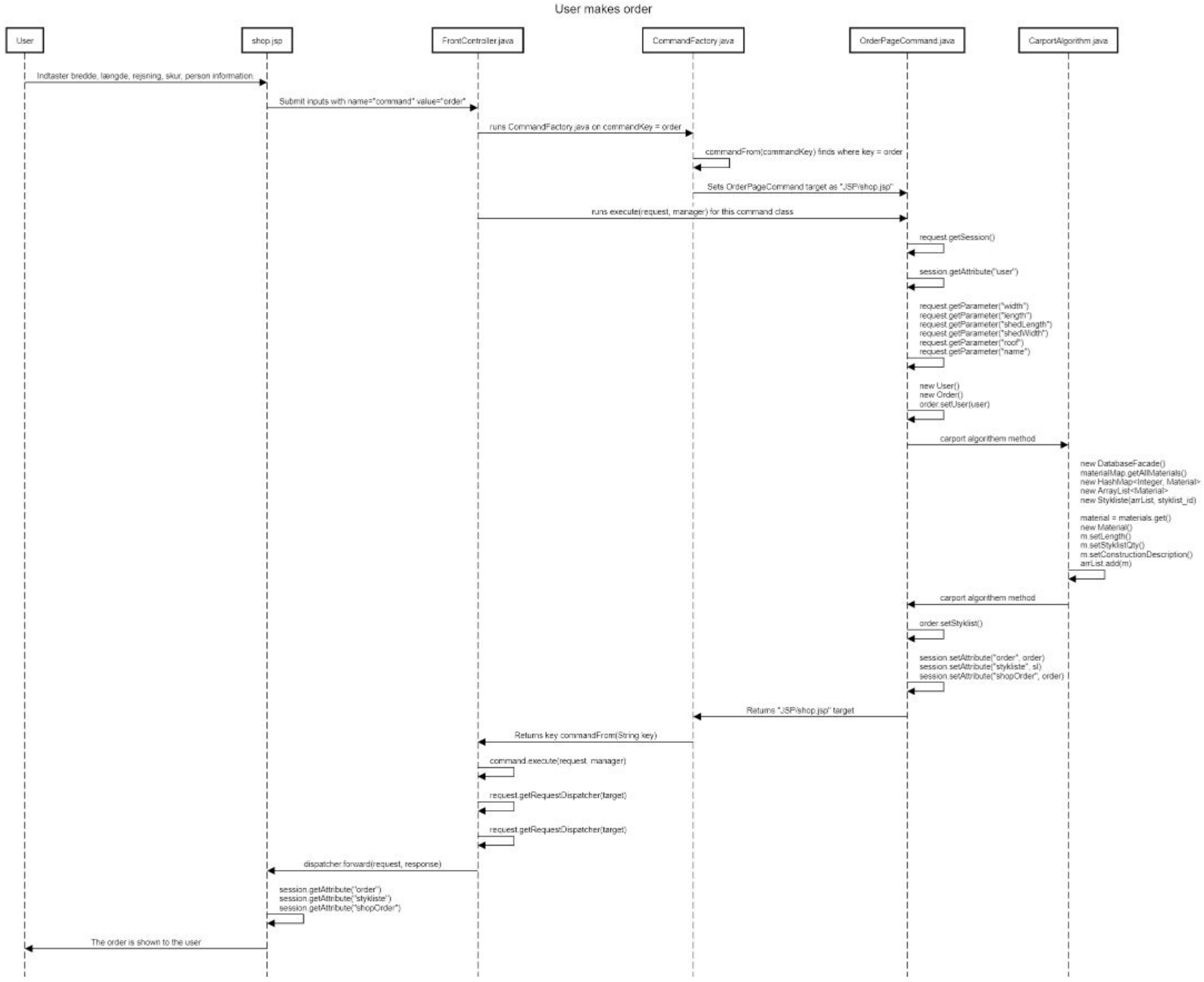
## Sekvens diagrammer

Sekvensdiagrammer bruges til at skabe overblik og forståelse for sammenhængen mellem de enkelte metoder i en sekvens af et program. En sådan diagramsekvens kan bruges til grundlæggende at vise, hvordan et typisk forløb foregår, eller til at vise et særligt vigtigt særtilfælde. Da disse sekvenser kun viser gennemgangen af mindre sektioner of funktionaliteter i programmet, laves sekvensdiagrammer normalt kun for mindre men vitale forløb i systemet. I vores tilfælde har vi valgt at lave 3 sekvensdiagrammer over hvordan en kunde henholdsvis kan navigere menuen og præsenteres for shop siden, oprette en ordre med brugerdefinerede mål gennem denne shop side, og så få genereret en visualisering af sin ordre gennem en SVG side som kunden så kan føres til. Gennem de følgende sekvensdiagrammer vil der følge forklaringer og beskrivelser for brugen af specifikke og korrekt navngivne klasser, metoder og enkelte if-sætninger som er vigtige for den visualiserede sekvens.



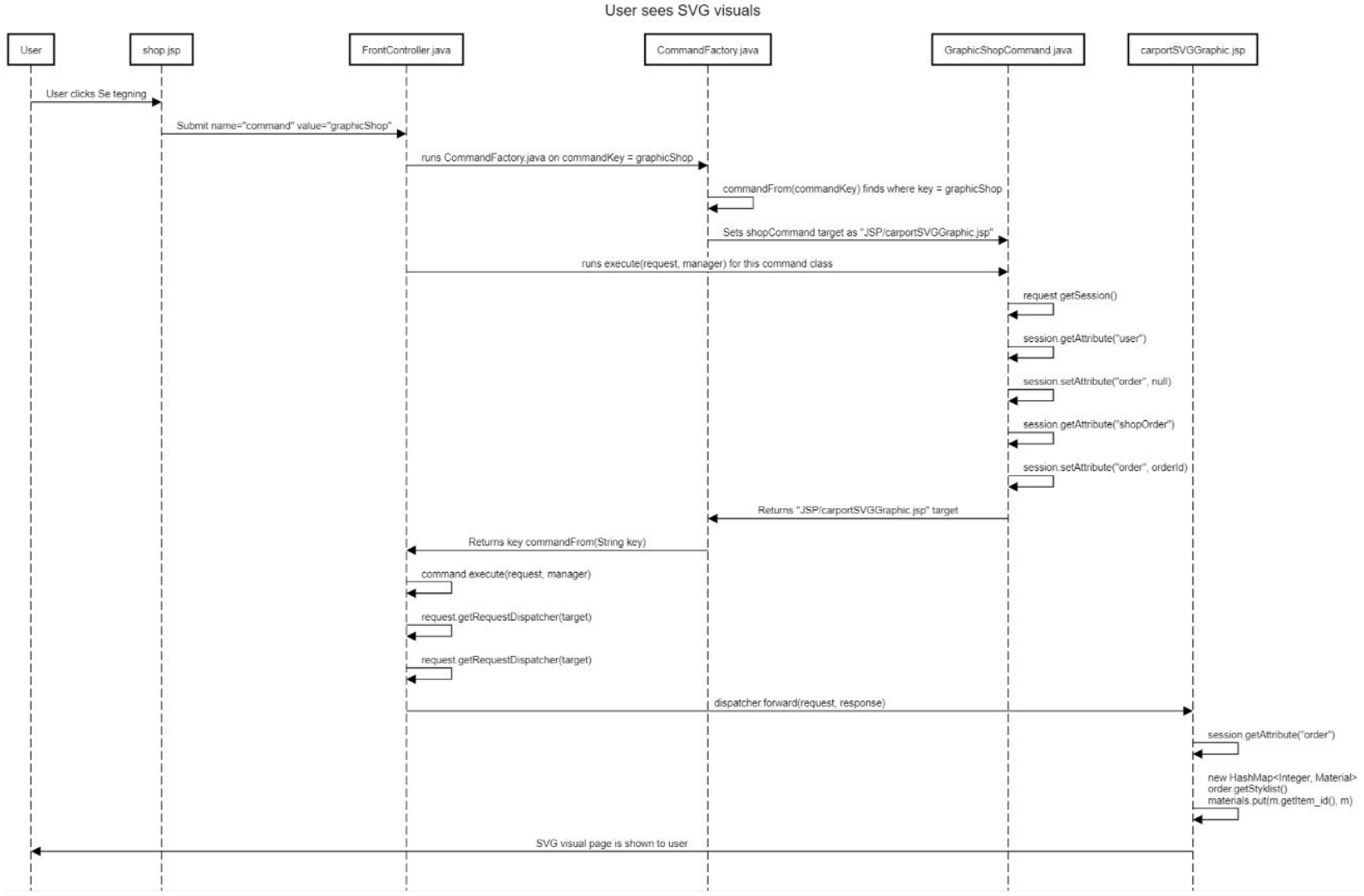

I dette sekvensdiagram klikker brugeren “User” på shop feltet i hovedmenuen sitemenus.jsp, der inkluders i bodyen på alle jsp siderne gennem et jsp:include tag. Efter klikket bliver der ført en herf reference /FogCarport/FrontController?command=shop reference til systemets servlet frontController FrontController.java, hvor der bliver oprettet en command ved CommandFactory.commandFrom(commandKey) fra CommandFactory.java, med den overførte command=shop commandKey som hentes fra http request parameteren ved request.getParameter("command"). Ved at køre CommandFactory.commandFrom(commandKey) med commandKey, hvilket køres i en instans pr. instans af servletten med som en static synchronized metode, vælges der den key fra CommandFactory() HashMap<String, Command> som matcher med commandKey. Denne HashMap indeholder keys i form af commandKey’s og en instans af en metode samt et prædefineret target parameter hvilket for shop key’en er ShopCommand("JSP/shop.jsp"). Når ShopCommand("JSP/shop.jsp") metoden bliver kørt i ShopCommand.java klassen settes "JSP/shop.jsp" som target for klassen. Herefter bliver metoden command.execute(request, manager) på kaldt ude i servletten på ShopCommand.java klassen. Når execute så bliver kørt, tjekkes der først om brugerens status er logget ind, hvilket klargøres ved at køre request.getSession() for at hente sessionen, og derefter session.getAttribute("user") som henter brugeren. Herefter bliver et hvilket andet order objekter der måtte have været gemt i sessionen fjernet ved session.removeAttribute("order"). Når disse metoder er kørt returneres target siden "JSP/shop.jsp" til frontControleren som så oprettes i en dispatcher med request.getRequestDispatcher(target), og derefter bliver videresendt med et http response

ved dispatcher.forward(request, response), så shop.jsp siden køres. På shop.jsp siden køres herefter nogle if statements der bedømmer om der er objekter gemt i sessionen og som ikke er tomme, altså at brugeren allerede har lavet en ordre. shop.jsp siden bliver herefter så ført ud og vist til brugeren.




I dette sekvensdiagram indtaster brugeren “Order” inputs i form af højde, bredde, længde, hældning, skur højde, skur bredde, og personinformation i en form, og klikker Lav order. Efter klikket bliver disse inputs sammen med en command=order reference som bliver submited til systemets servlet frontController FrontController.java som http request paramere. I frontControlleren bliver der oprettet en command ved CommandFactory.commandFrom(commandKey) fra CommandFactory.java, med den overførte command=order commandKey som hentes fra http request parameteren ved request.getParameter("command"). Ved at køre

CommandFactory.commandFrom(commandKey) med commandKey, hvilket køres i en instans pr. instans af servletten med som en static synchronized metode, vælges der den key fra CommandFactory() HashMap<String, Command> som matcher med commandKey. Denne HashMap indeholder keys i form af commandKey's og en instans af en metode samt et prædefineret target parameter hvilket for order key'en er OrderPageCommand("JSP/shop.jsp"). Når OrderPageCommand("JSP/shop.jsp") metoden bliver kørt i ShopCommand.java klassen settes "JSP/shop.jsp" som target for klassen. Herefter bliver metoden command.execute(request, manager) på kaldt ude i servletten på OrderPageCommand.java klassen. Når execute så bliver kørt, tjekkes der først om brugerens status er logget ind, hvilket klargøres ved at køre request.getSession() for at hente sessionen, og derefter session.getAttribute("user") som henter brugeren. Herefter bliver der kaldt request.getParameter() på alle de input som brugeren submittede i requesten hvorved de hentes. Disse værdier bliver gennem et if statement med nogle min og max værdier, sorteret for om der skal laves en carport eller om målene er forkerte og brugeren skal returneres til en ny tom shop.jsp side. Hvis input målene passer bliver derefter lavet et nyt User() objekt ud fra brugerinformationen og brugerobjektet som blev hentet i sessionen. Herefter bliver der lavet et Order objekt med de hentede input fra requesten, og User objektet bliver så tilføjet til Order med en setter metode. Herefter kaldes metoden manager.carportAlgorithm(), som i CarportAlgorithm.java gennem en algorithme sammensætter en stykliste med alle de konkrete materialer til en carport. I carportAlgorithm() bliver der først instantieret en DatabaseFacade(), derefter bliver getAllMaterials() metoden kaldet hvorpå alle materialer returneres i en ArrayList<Material>. Herefter Instantieres en HashMap<Integer, Material> hvori materialer kan gemmes på deres id, og der bliver oprettet en Stykliste(arrList, styklist_id), som indeholder ArrayList<Material>, og er det objekt der bliver returneret af algoritmen. For hver Materiral der bliver tilføjet arrayListen i algoritmen, bliver der kørt setLength(), setStyklistQty() og setConstructionDescription() på objektet, udover selve arrList.add() metoden der tilføjer objektet til listen. I carportAlgorithm() bedømmes der gennem nogle if statements om carporten er med eller uden skur og med eller uden rejsning, ud fra om disse værdier er 0 i Order objektet. Når carportAlgorithm() så returnere Stykliste objektet, tilføjes den med en setter til Order objektet, og både Order og Stykliste samt shopOrder for grafik funktionen, til sessionen ved session.setAttribute(). Når disse metoder er kørt returneres target siden "JSP/shop.jsp" til frontControleren som så oprettes i en dispatcher med request.getRequestDispatcher(target), og derefter bliver videresendt med et http response ved dispatcher.forward(request, response), så shop.jsp siden køres. På shop.jsp siden køres herefter nogle if statements der bedømmer om der er objekter gemt i sessionen og som ikke er tomme, altså at brugeren allerede har lavet en ordre. Hvis de ikke er tomme, fjernes udfyldningsformularen til input, og erstattes med nogle html lister der gennem toString() og list java for loops i selve siden kan kalde getAttribute() på henholdsvis Order, Stykliste og shopOrder objekterne som så vises på siden. shop.jsp siden med den udarbejdede order bliver herefter så ført ud og vist til brugeren.




I dette sekvensdiagram klikker brugeren “User” på “se tegning” feltet i shop.jsp siden, efter han har lavet en ordre, som i det ovenstående eksempel. Efter klikket bliver command=graphicShop submited til systemets servlet frontController FrontController.java som http request paramere. I frontControlleren bliver der oprettet en command ved CommandFactory.commandFrom(commandKey) fra CommandFactory.java, med den overførte command=graphicShop commandKey som hentes fra http request parameteren ved request.getParameter("command"). Ved at køre CommandFactory.commandFrom(commandKey) med commandKey, hvilket køres i en instans pr. instans af servletten med som en static synchronized metode, vælges der den key fra CommandFactory() HashMap<String, Command> som matcher med commandKey. Denne HashMap indeholder keys i form af commandKey’s og en instans af en metode samt et prædefineret target parameter, hvilket for graphicShop key’en er GraphicShopCommand("JSP/carportSVGGraphic.jsp"). Når GraphicShopCommand("JSP/carportSVGGraphic.jsp") metoden bliver kørt i GraphicShopCommand.java klassen settes "JSP/carportSVGGraphic.jsp" som target for klassen. Herefter bliver metoden command.execute(request, manager) på kaldt ude i servletten på GraphicShopCommand.java klassen. Når execute så bliver kørt, tjekkes der først om brugerens status er logget ind, hvilket klargøres ved at køre request.getSession() for at hente sessionen, og derefter session.getAttribute("user") som henter brugeren.

Herefter bliver et hvilket andet order objekter der måtte have været gemt i sessionen sat til null ved session.setAttribute("order", null), og shopOrder bliver hentet fra sessionen ved session.getAttribute("shopOrder"), samt derefter sat som ny order i sessionen ved session.setAttribute("order", order). Når disse metoder er kørt returneres target siden "JSP/carportSVGGraphic.jsp" til frontControleren som så oprettes i en dispatcher med request.getRequestDispatcher(target), og derefter bliver videresendt med et http response ved dispatcher.forward(request, response), så carportSVGGraphic.jsp siden køres. På carportSVGGraphic.jsp siden køres herefter et if statement der checker om sessionen gemmer en ordre, og hvis den gør køre session.getAttribute("order") som bruges til at instantiere ordre i siden. Herefter instantieres en ny HashMap<Integer, Material>, og der bliver kaldt order.getStyklist() på ordren, hvorefter materials.put(m.getItem_id(), m) sætter orderstyklistens indhold ind i hashmapen med hvert material id som key. Herefter bliver der gennem en række if statements kært forskellige svg sætninger i carportSVGGraphic.jsp siden, alt efter om carporten er med eller uden rejsning og med eller uden skur, baseret på om værdierne er 0 i Order objektet. carportSVGGraphic.jsp siden med carportens svg tegninger fra oven, fra siden og forfra, bliver herefter så ført ud og vist til brugeren.

## Særlige forhold og kodeeksempler

**Brug af session**

I http protokollen beskriver sessions en blivende forbindelse der oprettes mellem serveren og en tilgående maskine. Ved at lagrer objekter i sessionen ved session.setAttribute() og senere hente de objekter vi lagrede ved session.getAttribute() kan vi instantiere objekter i vores command metoder, og senere hente dem igen på .jsp siderne uden at skulle instantiere dem der. Dette foregår ved hjælp af metoder der kan skabe specifikke objekter der lagres i databasen, hvis bare de får det specifikke objekts id. Alle disse metoder kan tilgås fra klassen FunctionManager som alle command objekter får en instans af (mere om dette under afsnittet “Database og Funktioner” og “Frontcontroller”).

I dette eksempel kan vi ved valg af et enkelt objekt på en lang liste af objekter, som i listen over alle ordrer, styklister eller materialer i systemet, bruge hvert enkelt ordres, styklistes, eller materiales id. Det bruger vi ofte efter en side har udskrevet en ArrayList fyldt med alle ordre, stykliste eller materiale objekter i systemet. Herunder er et eksempel fra StockListWoodCommand.java

```
ArrayList<Material> materials = manager.getAllMaterialbyType("Træ & Tagplader");
System.out.println(materials);

session.setAttribute("stockListWood", materials);
```

stockListWood.jsp skriver så alle materialerne ud på siden, og vælger man at ændre på et enkelt materiale bliver dette objekts id sendt med som et hidden input.

```
ArrayList<Material> list = (ArrayList<Material>) session.getAttribute("stockListWood");
for (int i = 0; i < list.size(); i++) {
    //out.println("<tr><td>" + list.get(i).getItem_id() + ", " + list.get(i).getItem_description() + ", "
    out.println("<div class=\"alert alert-primary\" role=\"alert\">");
    out.println("<h5>" + " " + list.get(i).toString1() + ", StockQty: " + list.get(i).getStockQty() + "</
    out.println("<form action=\"FrontController\" method=\"post\">");
    out.println("<input type=\"hidden\" name=\"command\" value=\"editMaterial\">");
    out.println("<input type=\"hidden\" name=\"chosenStockMaterial\" value=\"" + list.get(i).getItem_id()
    out.println("<input class=\"btn btn-info  btn-sm\" type=\"submit\" value=\"Edit material\" >");
```

Når det er sendt til EditStockMaterialPageCommand kan der ud fra det id, der blev sendt som hidden input, oprettes et bestemt Material objekt, svarende til det der blev valgt fra listen, som sættes som attribut i sessionen.
Sådan ser metoden getMaterialbyID() fra MaterialDBMapper som vælger et enkelt materialeobjekt fra databasen baseret på dets id.

```
public Material getMaterialbyID(int item_id) throws MaterialSampleException {
    try {
        Connection con = dbc.connection();
        String SQL = "SELECT * FROM stock WHERE item_id = ?";
```

Herefter kan man blive sendt til en ny side hvor materiale objektet “mat”, hentes som attribut fra sessionen og bruges til at oprette et materiale objekt, lig det som man valgte på den forrige side. På denne måde kan man altid oprette et bestemt objekt hvis man har dets id.

**Håndtering af exceptions**

Når der i systemet bliver kastet en checked exception, har vi alt efter hvilket lag og hvilken class denne exception kastes i, lavet en række custom exceptions der derpå kastes og viderefører en String message samt en String target vi sættes til error.jsp siden. Disse custom exceptions er henholdsvis CommandException, OrderSampleException, MaterialSampleException, StyklistException, og LoginSampleException. Disse custom exceptions bliver ført fra den checked exception der kaster dem, og videreført op igennem lagene hvor de bliver fanget i selve Frontcontrolleren. Når de så fanges i Frontcontrolleren, logges disse custom exceptions i selve droplettens Tomcat webklient ved Logger.getLogger().log. Fra Frontcontrolleren køres så request.getRequestDispatcher() og dispatcher.forward() som videreføre os til target som er error.jsp siden, hvorpå der vises den message fejlbesked som den originale checked exception havde. Nedenfor kan ses et eksempel hvor der kastes en custom OrderSampleException.

```
} catch (SQLException | ClassNotFoundException | LoginSampleException ex) {
    throw new OrderSampleException(ex.getMessage());
}
```

**HTML input**

Ved indtastning af brugerinput i systemet er det vigtigt at der tages hensyn til at der kan indtastes forkert input, som indtastning af bogstaver når der kun skal bruges numeriske tegn, tommer felter eller lignende. Derfor kræver systemet at brugerinput checkes.

Billedet nedenunder er fra OrderPageCommand og viser hvordan bruger inputtet til en carport valideres gennem et if statement, efter at det er blevet sendt som parametre i requesten. Først tages parametrerne og der instantieres variabler.

```
float width = Float.parseFloat(request.getParameter("width"));
float length = Float.parseFloat(request.getParameter("length"));
float shedLength = Float.parseFloat(request.getParameter("shedLength"));
float shedWidth = Float.parseFloat(request.getParameter("shedWidth"));
float roofTilt = Integer.parseInt(request.getParameter("roof"));
String name = request.getParameter("name");
```

```
if (width > 7500 || width < 2400 || length > 7800 || length < 2400 || shedLength > 6900 ||
    //throw new MaterialSampleException("Fejl i mål");

    return target;
```

Så checkes input variablenes mål. Der er et sæt mål som er maximum og minimum for hvor stor eller lille en carport og det tilhørende skur kan være og hvis disse mål over- eller underskrides så bliver man sendt til en anden side uden at have lavet en carport ordre.

Her er en anden måde hvorpå brugerinput fra editMaterial.jsp bliver valideret i UpdateMaterialCommand. Der hentes et materiale objekt, der er blevet sat som attribut i sessionen. Med getters printes materialets attributter ud i de samme input felter der bruges til at indtastede rettelser. Så kan man både se værdien i input feltet men også rette den, og hvis man kun retter i et felt sendes alle andre urettede værdier tilbage til tabellen.

```
if (session.getAttribute("stockMaterial") != null) {
    Material material = (Material) session.getAttribute("stockMaterial");
    out.println("<form action=\"FrontController\" method=\"POST\">");
    out.println("<div class=\"form-row\">");
    out.println("<input type=\"hidden\" name=\"command\" value=\"updateMaterial\">");
    out.println("<div class=\"col-md-3 mb-3\">");
    out.println("<tr><td> Material id: </tr></td>");
    out.println("<input type=\"text\" name=\"id\" value=" + "\"" + material.getItem_id() + "\""+ "readonly>");
    out.println("</div>");
    out.println("<div class=\"col-md-3 mb-3\">");
    out.println("<tr><td> Material description: </tr></td>");
    out.println("<input type=\"text\" name=\"description\" value=" + "\"" + material.getItem_description() + "\"
    out.println("</div>");
    out.println("<div class=\"col-md-3 mb-3\">");
    out.println("<tr><td> Material width: </tr></td>");
    out.println("<input type=\"text\" name=\"width\" value=" + material.getWidth() + ">");
    out.println("</div>");
    out.println("<div class=\"col-md-3 mb-3\">");
    out.println("<tr><td> Material height: </tr></td>");
    out.println("<input type=\"text\" name=\"height\" value=" + material.getHeight() + ">");
    out.println("</div>");
    out.println("<div class=\"col-md-3 mb-3\">");
    out.println("<tr><td> Material entity: </tr></td>");
    out.println("<input type=\"text\" name=\"entity\" value=" + "\"" + material.getEntity() + "\"" + ">");
    out.println("</div>");
```

UpdateMaterialCommand har så opgaven at opdaterer et bestemt materiale med ny information.

```
String regexNumber = ".*\\d.*";

if (request.getParameter("id") != null && request.getParameter("descripti
    if (request.getParameter("id").length() != 0 && request.getParameter(
        if (!(request.getParameter("entity")).matches(regexNumber) && !(r
```

Parametre bliver checket i tre if-statements. Først bliver alle parametre checket for at inputtet ikke er null, altså tomt. Så bliver alle parametre checket for at de ikke er af længden 0. Til sidst checkes det at der ikke er tal i input der kun kræver bogstav input.

Dette checkes med regex funktionen, en måde hvorpå man med tegn og symboler kan lave et udtryk der søger efter forskellige mønstre i en String.
\d betyder alle tal [0-9], og checker derfor for alle mulige tal. * eller "asterisk", indikerer at noget gerne må forekommer en eller flere gange. Så hvis et bestemt mønster forekommer bare en enkelt gang, eller 1000 gange i den tekst der skal checkes, så vil * sørge for at det opdages. Disse to tegn kombineret betyder at inputtet vil give fejl, hvis der bare er et enkelt tal et eller andet sted i strengen. Dette kodes ind i String regexNumber, der kan bruges med matches metoden til at checke input Strings.

Hvis alle disse if-statements viser sig at være sande så opdateres materialet med de nye værdier i UpdateMaterialCommand, mens de uændrede værdier også sættes ind.

```
int id = Integer.parseInt(request.getParameter("id"));
String description = request.getParameter("description");
float width = Float.parseFloat(request.getParameter("width"));
float height = Float.parseFloat(request.getParameter("height"));
String entity = request.getParameter("entity");
String type = request.getParameter("type");
float price = Float.parseFloat(request.getParameter("price"));
int qty = Integer.parseInt(request.getParameter("qty"));

manager.updateMaterialData(id, description, width, height, entity, type, price, qty);
```

Det gør det let at opdaterer materialer uden at skulle tage hensyn til om det er al information tilknyttet til materialet, der skal ændres.

**Header, navigationsbar og footer**
Som det blev nævnt under afsnittet om vores navigationsdiagram benytter vi "jsp:include" i vores program til at vise en navigationsbar på hver side. Dette blev også benyttet til at lave to .jsp sider, siteheader.jsp og sitefooter.jsp. Disse to siders funktion er åbne og lukke for alle .jsp siderne i programmet. Den fulde HTML tag som en .jsp side skal indeholde deles på denne måde op i 4 dele, med en heder, navigationsbar, selve siden og en footer. Tagget begynder i siteheader og slutter i sitefooter. Desuden indeholder disse to .jsp sider al metainformation og andet der kræves for at en .jsp side fungerer. siteheader indeholder tagget der forklarer hvordan siderne skal starte og stylies, starten på <html>, hele <head> og det begyndende <body>. Udover det indeholder siteheader også bootstrap styling der så gælder for alle .jsp siderne det inkluderes i. sitefooter indeholder hovedsageligt det afsluttende </body> og </html>. Hvis en .jsp side har en siteheader.jsp tag i starten af siden og en sitefooter.jsp tag nederst på siden så vil sidens indhold blive del af en fuldkommen .jsp side.

Derfor har de fleste .jsp sider i systemet har ingen body tag eller instruktioner om hvordan sidernes indhold skal styles overhovedet, men kun tags for siteheader.jsp i starten af siden, sitemenus.jsp bagefter og sitefooter.jsp som det sidste på siden. Tilsammen gør disse at

alle sider i systemet er inde i en HTML tag, har en tag for styling og char-set af siden og en navigationsbar.

**Forbindelse og MapperDB Klasser**
Connector klassen indeholder adressen og passwordet til databasen og sørger for at der kan oprettes forbindelse til databasen. Denne klasse virker ved at den opretter et Connection objekt. Metoden connection() checker om Connection objektet er null. Hvis det er, gives det addressen specificeret i Connector klassen, og ligegyldigt hvad, returneres der et Connection objekt, som skal bruges for at oprette forbindelse til databasen. Alle klasser der skal oprette forbindelse til databasen får et Connector objekt og hver gang en metode, der har forbindelse til databasen kaldes, kalder den connection() via dets Connector objekt og får returneret en instans af Connection objektet, som kan bruges til at kommunikere med databasen. Her er et eksempel fra OrderDBMapper, der kalder connection() og opretter et nyt Connection objekt for at få adgang til databasen.

```
@Override
public ArrayList<Order> getAllOrders() throws OrderSampleException {
    try {
        String sql = "Select * from orders;";
        Connection conn = dbc.connection();
```

Alle Mapper klasser har også mange af de samme metoder. Både OrderDBMapper og MaterialDBMapper har som ovenover en metode til at returnere alle objekter, der findes i databasen, som en ArrayList. Desuden har alle MapperDB klasserne en eller flere metoder der kan returnere et enkelt objekt baseret på enten id eller et andet unikt parameter, metoder der gør at værdier i databasen knyttet til et enkelt objekt kan ændres, et objekt kan slettes eller et nyt objekt kan blive oprettet i databasen.

**Database og Funktioner**
Vi valgte at implementere mapper klasser som instanser af abstrakte superklasser. Alle underklasser af de abstrakte superklasser indeholder en metode, getInstance(), der kan give en statisk instans tilbage af det selv. Superklasserne indeholder en metode instance(), som er i stand til at kalde på denne metode og returnere en instans af underklassen.

```
public static MaterialMapper instance() {
    return MaterialDBMapper.getInstance();
}
```

Dette er selvfølgelig kun muligt fordi at den abstrakte klasse kan godt returnere sin egen underklasse som en instans af superklassen.

Klassen FunctionManager, får så en instans af hver abstrakte mapper klasse, samt instanser af CarportAlgorithm.java og Encryption.java. Alle Mapper klassernes instance() metoder kaldes så samtidigt med at deres underklasse objekter oprettes.

```
private final StyklisteMapper StykMapper = StyklisteMapper.instance();
```

Alle relevante metoder fra alle objekterne implementeres i FunctionManager. Hver metode kalder den metode de overrider igennem de objekter, der blev oprettet i FunctionManager,

hvor metoden stammer fra.

```
@Override
public void saveLineItemsInDB(Stykliste styklist, int order_id) {
    StykMapper.saveLineItemsInDB(styklist, order_id);
}
```

Dette gør at alle funktioner i systemet, som er relateret til adgang til databasen, kryptering og beregning af carporte, kan tilgås fra et FunctionManager objekt. Samtidigt indkapsles metoderne da de netop kun kan tilgås fra FunctionManager.

**Commands og CommandFactory**
For at styre siderne og funktioner i systemet har vi brugt en Command baseret designarkitektur. Alle Command objekter er baseret på Command.java interfacet, som indeholder execute() metoden. execute() er ansvarlig for at udføre de fleste funktioner i systemet, som f.eks. at hente data fra databasen, sætte og hente attributter i sessionen og requesten, lave udregninger og returnere adressen for den næste side brugeren kommer til nu. Mange af disse funktioner kaldes igennem et FunctionManager objekt (se "Frontcontroller").

Derudover har Command interface også metoden loginStatus() som returnerer "false", hvis sessions attributen "user" er null, og "true" hvis en bruger er logget ind. Dette bruges til at verificere om brugeren er logget ind. I tilfælde af at en bruger ikke behøver at logge ind, kan man undlade at udfylde metoden ved implementation. Som en del af interfacet har Command også accesToPage() som ville skulle bruges til at verificere hvorvidt en bruger havde adgang til den givne side. Grunden til at det er lavet som en del af interfacet gør at man bliver nødt til at tage stilling til adgangsstyring ved udarbejdelse af Commands.

I vores implementation er der også en klasse CommandFactory, der er meget vigtig i forhold til navigation og funktioner i systemet. CommandFactory håndterer, hvilke commands der kaldes på hvilke tidspunkter. CommandFactory indeholder et HashMap, hvori alle mulige instanser af Command opbevares i, samt en metode commandFrom(), der kan modtage en string, der svarer til nøglen til et bestemt Command objekt i hashmappet, som derefter kan returneres. Denne funktion bruges ved håndtering af requests (mere om det under "Frontcontroller"). Konstruktøren i alle commands i CommandFactory indeholder en string, target, som er et link til en bestemt .jsp side. Det er denne string, der returneres når execute() eksekveres og fortæller hvad for en side man skal sendes til efter af en Command er blevet kaldt og eksekveret. CommandFactory indeholder en statisk, privat instans af et selv, som derfor er tilgængelig for alle andre klasser.

**Frontcontroller**
Frontcontrolleren extender HttpServlet, og er den servlet vi bruger til at håndtere vores http requests og responses samt de custom exceptions som kastes i de forskellige lag af systemet. Frontcontrollen indeholder en instans af FunctionManager og har adgang til CommandFactory.

Når der modtages en http request i Frontcontrolleren instantieres der en string variabel med samme værdi som parameteret "command" fra requesten. Denne variabel bestemmer hvad for en Command der kaldes. Derefter eksekveres CommandFactory metoden commandFrom(), der gives command variablen og der instantieres et bestemt command objekt. execute() metoden i det instantierede command objektet gives requesten og en instans af FunctionManager som parametrer, og metoden eksekveres. Så udføres forskellige funktioner, som at hente data fra databasen gennem FunctionManager objektet, udfører udregninger og hente og sætte attributter i requesten og sessionen. Til sidst returneres target adressen til .jsp siden, hvorefter brugeren sendes videre til adressen og processen afsluttes. Dette gøres gennem metoden request.getRequestDispatcher(target) i frontControlleren, som ved en dispatcher videresender et http response ved dispatcher.forward(request, response), hvilket videreføre siden "target" til brugeren.

### Carport algoritme

CarportAlgorithm.java modtager et Order objekt med målene for en carports bredde, længde, tagvinkel, skurbredde, skurlængde og stykliste id. Den udregner så hvor meget af hvilke materialer carporten med disse mål skal bruge og giver svaret tilbage i form af et stykliste objekt med et stykliste id. Til at starte med omdanner algoritmen den totale liste af stock materialer fra getAllMaterials(), til en hashmap med materialernes id som key. Herved kan algoritmen nemmere få fat på et specifik material i stedet for at skulle itererer igennem et for loop hver gang. Denne HashMap<Integer, Material> bruges så til at instantiere versioner af de pågældende Material objekter som algoritmen slavisk en efter en sætter ind i en styklistes ArrayList<Material>. Algoritmen tilføjer specifikke længer og styklisteQty kvantiteter for de enkelte materialer inden de bliver sat ind i styklisten. Da disse materialer allerede har nogle prædefinerede højder og bredder fra stock database tabellen, udregner algoritmen længde og kvantitet i henhold til både materialernes højde og bredde, samt ordrens carport mål. Hen igennem denne algoritme findes også en række if statements som køre forskellige dele af koden alt efter om carporten skur eller har en hældning, baseret på om disse værdier er 0 eller i order objektet. Her ses et eksempel på et materiale der tilføjes til styklisten:

```
if (shedLength != 0 || shedwidth != 0) {
    //Tilføj 2 remme der sadles ned i stolperne for skuret af 45x195mm.spærtræubh
    material = materials.get(5);
    m = new Material(material.getItem_id(), material.getItem_description(), material.getWidth(), material.getHeight(),
    m.setLength(shedLength + 10); //så lange som skurets sider og lidt til tilskæring
    m.setStyklistQty(2);
    m.setConstructionDescription("Remme i sider, sadles ned i stolper (skur del)");
    arrList.add(m);
}
```

### SVG grafik

Carporten tegnes fra 3 forskellige vinkler med SVG, baseret på målene og styklisten der blev lavet ud fra order målene.  Denne order hentes fra sessionen ved session.getAttribute("order"), hvor den bliver instantieret, og derefter kaldt order.getStyklist() i sammenhæng med et for-each loop og en hashmap som tilføjer alle listen materialer, så de kan kaldes på deres id. Dette gør det nemmere at kalde på de enkelte objekter, når de skal bruges til SVG målene.

Alle SVG tegningerne er tegninger i et størrelsesforhold af 1:10 pixels pr. mm for hvert mål, hvilket gør det muligt at have 3 hele illustrationer på en side. De enkelte SVG statements, er hovedsageligt baseret på rektangler med nogle x og y starts kordinater samt en længde, en bredde og noget stroke og fill farve. Foruden dette har vi gjort brug af transform til i nogle tilfælde at translate startslokationen oven på de foruddefinerede x og y koordinater, rotere SVG elementerne efte en bestemt vinkel, og i 2 tilfælde brug skewX og skewY, hvilket ændre i selve figurens interne vinkler. Nedenfor ses et eksempel på nogle af de nævnte SVG elementer:

```
<rect x=" <% out.println((order.getLength() - 300) / 10); %> " y=" <% out.println((order.getHeight()
<rect x=" <% out.println((order.getLength() - order.getShedLength() - 300) / 10); %> " y=" <% out.pri
<rect x=" <% out.println(((1000.0 / 7800.0 * order.getLength()) + (materials.get(6).getWidth()) + (31
```

Ligesom i selve carport algoritmen, har vi for SVG tegningerne også gjort brug af en helt del if statements, til at bedømme hvilken del af SVG'en i carportSVGGraphic.jsp der skal køres vedrørende om carporten har skur og har hældning. Ud over dette har vi i nogle tilfælde fast placeret SVG elementerne i henhold til målene men med samme faste variation af mellemrum, hvor andre elementer er blevet dannet ved brug af en række for loops ud fra kvantiteten af det element.

**Sikkerhed i forbindelse med brugerlogin**

Vores kryptering af password ligger i klassen "Encryption". Når bruger opretter sig bliver passwordet sendt til metoden 'hash' som tager passwordet og salt(salt bliver random genereret i metoden getSalt). Til at hashe passwordet bruger vi SHA1 (Secure Hash Algorithm 1) som bruger PBEKeySpec som key til at kryptere passwordet med. Iterations som er antal gange passwordet bliver hashet, har vi sat til 10.000 som anses for at være tilstrækkeligt. Ved forøgelse af iterations count kan systemet blive påvirket i form af lang ventetid ved login. PBKDF2WithHmacSHA1 (Password-based key-derivation function with key-hashed authentication code) som er den hash funktion vi bruger til at hashe med returnere et byte array af længden 160 bits. I metoden generateSecurePassword bruger vi base64, som har en encoder. Den bliver ofte brugt til at formidle binær data i form af tegn i stedet for 0'er og 1'er. Vi bruger metoden encodeToString som er en del af base64 biblioteket i Netbeans, til at lave en repræsentation af passwordet. Ved oprettelse af bruger bliver det hashede password sammen med salt gemt i databasen. Den samme funktion bliver brugt når en bruger logger ind. Det indtastede password bliver sammen med det lageret salt hashet og tjekket om det matcher det securePassword der blev genereret da brugen oprettede sig.

# Status på implementation

Når det kommer til statussen på implementationen af vores system, fungerer størstedelen af de implementationer som product owner prioriterede, og alle programmets core funktionaliteter virker efter hensigten, og med forholdsvis stor stabilitet. Der er dog også en del ting som vi stadig godt kunne have tænkt os at implementer, ændre eller overvejende genopbygge. Nogle af disse elementer involvere yderliger ønskede funktioner der under tidspresset og prioriteringssamtaler med product owner aldrig blev fuldendt. Andre har at gøre med fejl i overlapningen af de tilstedeværende funktionaliteter, eller mindre komplikationer som mangler konkrete løsninger. Størstedelen kommer af at vi gennem forløbet i at arbejde med systemet, er blevet en del klogere på hvordan en sådan webservice etableres og struktureres.

**Systemets konkrete status**

Systemets konkrete status er at man i form af en admin rolle, som bruger gennem vores online platform kan logge sig ind på platformen, og navigere en hovedmenu som består af Home, Materials, Shop, AllOrders og Logout. Home funktionen bringer brugeren hjem til front siden hvor der kan navigeres til Materials, Shop og AllOrders. Materials navigere brugeren til en side med valgmuligheder for at se og redigere stock materialer af forskellige kategorier, Shop bruger brugeren til shop siden hvor der kan oprettes en ordre gennem et skema og derefter vises en stykliste samt genereres en SVG tegning for den ordre. AllOrders siden navigere brugen igennem en liste med alle ordene i databasen, hvor der er mulighed for at klikke sig ind og redigere på de enkelte ordre, samt få dannet både en stykliste og en SVG tegning til den specifikke ordrer. Til sidst har brugeren muligheden for at benytte Logout, som logger brugen ud af systemet og returnere dem til login siden. Disse funktionaliteter involvere Størstedelen af de user stories som vi i samarbejde med product owner tog udgangspunkt i, og opfylder derfor de vigtigste funktionaliteter som var ønsket for product owners side. Det meste af den følgende sektion vil derfor hovedsageligt involvere userstories eller tilknyttede tasts der på demoens afleveringstidspunkt ikke var færdiggjort, eller kunne af hensigtsmæssige årsager have været tilknyttet.

**Brugere**

I henhold til bruger rollerne som skulle være tilknyttet rolle funktionen, aftalte vi med product owner at det over det 4 sprint var mere relevant at kunne se de administrerende funktioner hvorpå man kunne reagere med hele systemer. Derfor beslutede vi forinden afleveringsfristen at færddigære brugerfunktionerne uden den aktive rollefordeling som skulle have været basere i 2 roller af henholdsvis kunder og Fog administrerende. Vi ville gerne have tilpasset platformen så hvis man var kunde, at man kunne logge ind og så kun kunne se sine egne helt personlige ordre, shop og en tegning, i stedet for at det kun er en administrerende rolle der kan gøre det for alle systemets brugere. Dette er et punkt som vi ved mulighed for yderligere udbyggelse af systemet derfor godt ville have videreudviklet på, men som til demoen måtte efterlades med fokus på at kunne demonstrere systemets hovedfunktioner. Systemet indeholder dog allerede enkelte metoder og database data til denne rollefordeling, hvilket er grundstenene for denne funktionalitet som vi desværre ikke kunne nå at have med i denne demo.

**Brugerdefineret indhold**
Ved implementationen af det førnævnte rolle system, havde vi tænkt os at introducere brugen af mere brugerdefineret indhold. Dette var igen endnu en af de mindre prioriterede user stories som vi delvist nåede at implementere. Den nuværende del af implementationen der er aktivt i demosystemet, er det at en bruger har et unikt brugernavn, email, rolle og id. Disse elementer skulle i forlængelsen af denne implementation kunne være med til at bestemme hvilke elementer der skulle vises for den enkelte bruger på de forskellige jsp sider. Dog blev denne funktionalitet i forbindelse med samtaler med product owner ikke prioriteret for demoen, men var en funktionalitet som vi havde i mente og faktisk har nogle implementerede grundelementer for i det nuværende system.

**Aktiv fejlhåndtering og brugerkorrektion**
Da vi meget sent i dette forløb havde om emnet error handling, var det begrænset hvad vi kunne nå at implementere og bruge for product owner. Derfor blev den error handling som vi implementerede i systemet, brugt i nogle lidt simplere træk ind de kunne være. Den nuværende status er at vi har en række custom exceptions der kastes i forskellige dele af programmet når der mødes checked exceptions. Disse custom exceptions har deres egen target page ved error.jsp som, og en unik massage, som videreføres fra den checked exception der originalt kastede custom exceptionen. Disse custom exceptions føres op igennem lagene og fanges i front controlleren, hvor de logges i serverens Tomcat klient. Herefter forwardes error.jsp siden, hvorpå den videreførte besked fra den originale checked exception, eller en hvilken som helst anden custom message, så bliver vist på jsp siden. Dette kunne i lange træk have været gjort meget mere konkret og brugervenligt, med målrettede beskeder for fejlhåndtering af forskellige metoder. Vi kunne også have implementeret mere front end håndtering af brugerinput med blandt andet noget javascript, men igen var det her noget som vi ikke blev præsenteret for fær lang henne mod projektets slutning, men som vi har gjort os nogle tanker over.

**Database**
Et mere grundlæggende element som på mange måder ligger lidt i vejen for systemets status, er databasen som vi på mange måder godt kunne tænke os at udbygge og rette op på yderligere. Da databasen er en fundamental del for meget af systemet, og indgår i alle de forskellige datamappers metoder, er de funktionelle og strukturelle afvigelser forholdsvis kritiske for vores system, og meget svært at rette på nuværende tidspunkt. Den nuværende version af vores database var opbygget efter behov, og virker umiddelbart også efter de behov, men strukturen for yderliger udvidelser og forholdet til overholdelsen af normalformerne, gør den særdeles svær at arbejde med på nuværende tidspunkt, og bestemt på længere sigt.

**Funktioner relateret til ordrer**
Under listen over alle ordre i systemet er der en funktion der viser om en ordre er “klar”. En ordre starter med at være “ikke klar”, og efter at den er blevet prissat ændres dens status til “klar”. Dette kunne udbygges med to mulige funktioner. Først bør det at ændre en ordres status være en separat handling fra at den prissættes. I stedet for at den automatisk ændrer status når den er blevet prissat bør dette ske med en knap. Desuden bør der også være

mulighed for at ændre en ordres status fra “klar” tilbage til “ikke klar”. En metode der kan ændre en ordres status til “ikke klar” findes allerede i OrderDBMapper, unFinalizeOrder(). Denne metode kunne implementeres, også med henblik på andre forhold, som at det er et tryk på en knap der ændrer en ordres status. På samme tid kunne man for fremtiden også implementere at en ordres pris vises sammen med andet information for en ordre på allOrders siden.

**Fejlhåndtering relateret til brugerinput**
Fire forskellige funktioner i systemet skal håndtere brugerinput, henholdsvis: indtastning af mål til en carport ordre, rettelse af et materiales mål og beskrivelse, oprettelse af et nyt materiale og prissætning af ordre.

Ordre input checkes for om målene er for store eller små, men ikke om input felterne er tomme eller om de indeholder tegn andre end tal. Hvis de gør det vil det give fejl hvis disse parametre forsøges indsendes.
Ved opdatering af materiale sendes der både input med tal, bogstaver og tal og kun bogstaver. Dette input checkes for at felterne ikke er tomme eller at de felter der kun skal have bogstav input ikke modtager tal. Den kan dog ikke checke for at et felt der kun skal modtage tal ikke modtager bogstaver. Ved oprettelse af et nyt materiale eller prissætning af en ordre checkes input overhovedet ikke.

Inputsystemet kunne i første omgang forbedres ved at koden der checker ved opdatering af materiale kunne tilpasses og implementeres i alle de andre funktioner der håndterer brugerinput. Derudover bør alle brugerinput også kunne checkes for om det indeholder bogstaver, hvor det ellers kun bør modtage tal.
Der er heller ikke nogen fejlmeddelelse ved fejlinput. Hvis man taster mål der over- eller underskrider maximum og minimum i shoppen bliver man bare sendt tilbage til shoppen mens alt det input man skrev er ryddet. Hvis man prøver at indsende fejl input når man redigerer et materiale bliver man sendt tilbage til listen over alle materialer, ligeså uden en fejlmeddelelse. Alle sider kunne forbedres ved en fejlbesked ved forkert brugerinput så brugere af systemet ved at de har gjort noget galt.

**Styling**
Vores sider er stylet, men kun med bootstrap, det var ikke en prioritet at for os at putte css på. Man kan også komme langt med kun bootstrap. Vi ville derudover gerne have lavet lidt javascript til at gøre siden endnu mere interaktiv. Vi har brugt bootstrap til at lave alle vores menuer og indtastningsfelter pæne. Vi kunne have brugt css til at style vores side endnu mere og gøre nogle af vores bokse endnu pænere. Hvis vi havde tid ville vi gerne have leget med javascript til at indsætte ting som loading mellem siderne. Grunden til at vi valgte bootstrap i stedet for css er at bootstrap er meget hurtigt og simpelt at implementere. I siteheader.jsp har vi en enkelt linje der indeholder et bootstrap link, og siden alle andre .jsp sider er del af siteheader.jsp indeholder alle sider ligeså bootstrap. På denne måde kan man style hele programmet meget hurtigt. Vi kunne i fremtiden forbedre vores programs visuelle design ved at implementere css. Med css kan man lave meget mere præcis styling da forskellige instanser af det samme type element kan få forskellige styling.
Siden editMaterial.jsp er ikke blevet stylet, som en af de eneste.

**SVG**

Vores program er i stand til at lave en forståelig, men ikke perfekt repræsentation af en carport. Når en tegning med rejsning ses bagfra er taget ikke tegnet helt perfekt, da nogle af brædderne der skal dække enden af taget ikke går helt op. I øjeblikket er vores program i stand til at tegne en specifik carport set fra siden, forfra og nedefra. En mulighed, der var alt for tidskrævende, kunne være at tegne carporten set isometrisk, altså fra oven, fra siden og forfra. Det vil give kunden et lidt mere virkelighedsnært billede af carporten.

## Arbejdsprocessen faktuelt

Vi havde 4 sprints som er beskrevet i vores SCRUM sektion. Vi har ikke direkte haft en fast konkret SCRUM master, og vi har løftet denne opgave i fællesskab, da det er første gang vi arbejde med SCRUM. Normalt ville man have en person som var SCRUM master. Til hver PO-møde forberedte vi hvad vi skulle spørge vores PO om. I de senere stages af forløbet gav vi PO mulighed for at sidde med programmet selv og gøre de ting som var lavet til den givne sprint. Vores PO-møder gik fint da vi altid forberedte noget at snakke om og noget vi skulle have afklaret i forhold til prioritet. Vores daglige møde holdt vi inde vi startede vores arbejde dagligt. Snakkede om hvad folk kiggede på, om man havde brug for hjælp og om der var noget der viste sig at være større end først antaget. Efter hver sprint holdt vi vores retrospectives møder, vi brugte dem til at review vores sprints stories og de task der hørte til, som nåede vi det hele, havde vi estimeret tiden rigtigt.

## Arbejdsprocessen reflekteret

I og med det var vores første rigtige brug af SCRUM, var det lidt af en tilvænningssag ikke at være for ambitiøs når der skulle laves sprint-log. Vi brugte en masse tid på at omrokere user-stories fra de forskellige sprints som vi ikke havde nået gennem forløbet. Det ville have været meget behjælpeligt hvis vi havde lavet user-stories lidt mere uafhængig af hinanden så man kunne abstrahere fra resten af systemet når man havde sprint. På grund af vores manglende erfaring med SCRUM blev mange af vores user-stories vurderet med for lidt tid i forhold til hvor lang tid det reelt tog. I fremtidig SCRUM projekter vil det være en del lettere for os at estimere efter som vi har noget at sammenligne med. Efter som man kom i rutine med den agile udviklingsmetode blev det nemt at uddelegere opgaver og have styr på hvem der lavede hvad. Det gav et klart overblik over hvilke tasks der var i process og hvad hvilke tasks der manglede opmærksomhed. Alt i alt har det været en lærerig proces og man har fået nogle værktøjer der gør det nemt at overskue større projekter med mange elementer.

# Test

Vi har unit testet de metoder, som har mest betydning for at programmet virker. De funktioner som har med databasen at gøre, er ret afgørende for om siden fungere optimalt, og at tingene bliver lagret på den rigtige måde.

Det primære formål ved at lave disse test, er at se hvad metoderne gør i tilfælde af at en rigtig eller forkert parameter indtræder, og at kontrollere at der ikke er nogle konkrete systemfejl. Her vil det være væsentligt at finde ud af om en metode lagrer nogle forkerte parameter, eller den er bygget til at fejlhåndtere.

Nedenstående er de test som vi har lavet i forbindelse med projektet, og som kan findes fordelt i 3 testklasser. Disse testklasser er henholdsvis FogDataTest.java som tester medtoder i datalaget, FogFunktionTest.java som tester på metoderne i funktionslaget, og CommandTest.java som tester på nogle af præsentationslagets commands ved brug af test dependencien Mockito og mock objekter. Alle datamapper test testes i forbindelse med en testdatabase der fungere som et tro kopimiljø af systemets rigtige database.

**FogDataTest.java**

| Hvilke klasser er testet | Hvilke metoder er testet | Dækningsgrad af tests |
|---|---|---|
| MaterialDBMapper.java | testGetMaterialbyID()<br><br>teastGetAllMaterialbyType()<br><br>testAddNewMaterial()<br><br>testGetAllMaterials()<br><br>testUpdateMaterialData()<br><br>teatDeleteMaterial() | For MaterialDBMapper klassen testes alle klassens metoder for om de kan fremkalde og instantiere materialer fra en testdatabase. Her testes både for om noget af dataen indeholder det rigtige og ikke er null, samt om vi så kan redigere, opdatere, fjerne eller tilføje materialer samt om vi kan læse hele lister fordelt både efter alle materialer og materialer efter kategori uden at få konkrete systemfejl. |
| OrderDBMapper.java | testSaveOrder()<br><br>testGetAllOrders()<br><br>testGetOrderFromId() | For OrderDBMapper klassen testes alle klassens metoder for om de kan fremkalde og instantiere ordre fra en testdatabase. Her testes både for om dataen indeholder de rigtige ordre |

| | | |
|---|---|---|
| | testFinalizeOrder()<br><br>testDeleteOrder() | og ikke er null, samt om vi kan gemme ordre, hente alle ordre, hente ordre fra deres id, rediger de lineitem der er tilføjet et bestemt order id, færdiggøre ordrestatus, og slette en ordre uden konkrete systemfejl. Her tester vi også på om den hentede data passer med den prædefinerede data fra test databasen. |
| StyklisteDBMapper.java | testSaveLineItemsInDB()<br><br>testEditLineItemsFromOrderID()<br><br>testGetMaterialFromLineItems() | For StyklisteDBMapper klassen testes alle klassens metoder for om de kan fremkalde og instantiere hele styklister samt specifikke lineItems fra en testdatabase. Her testes både for om dataen indeholder de rigtige lineItem materialer og ikke er null, samt om vi kan gemme både enkelte lineItems såvel som hele styklister, hente styklister, hente lineItems fra deres id, og rediger de lineitem der er tilføjet en bestemt stykliste uden konkrete systemfejl. Her tester vi igen på om den hentede data passer med den prædefinerede data fra test databasen. |
| UserDBMapper.java | testCreateUser()<br><br>testVerifyUser()<br><br>testGetUserByEmail()<br><br>testLogin()<br><br>testRemoveUser() | For UserDBMapper klassen testes alle klassens metoder for om de kan fremkalde og instantiere users fra en testdatabase. Her testes både for om dataen indeholder de rigtige users og ikke er null, samt om vi kan oprette users, validere eksisterende users, hente users fra deres user email, logge en user med rolle ind og aktivt teste på om de |

| | | |
|---|---|---|
| | | eksistere,og fjerne en user fra databasen uden konkrete systemfejl. Her tester vi igen på om den hentede data passer med den prædefinerede data fra test databasen. |

**FogFunktionTest.java**

| | | |
|---|---|---|
| CarportAlgorithm.java | testCarportAlgorithm() | For klassen CarportAlgorithm tester vi på om algoritmens metode der sammensætter en carport ud fra en carports mål, kan lave specifikke carportstyklister der variere i indhold efter de tilføjede mål. Her testes både på styklistens forskelighed for carport med fladt tag og med rejsning, med og uden skur, samt for stylisternes længder og at den stykliste der returneres ikke er null. |
| Material.java | testMaterial() | For klassen Material tester vi for om vi kan instantiere et material objekt, tilføje en stylist quantity og tilføje et lineItem id, samt derefter kalde get metoder på disse, og kontrollere at det er de samme værdier vi får tilbage. |
| Order.java | testOrder() | For klassen Order tester vi for om vi kan instantiere et order objekt, tilføje en order status, tilføje en date og sætte en pris, samt derefter kalde get metoder på disse, og kontrollere at det er de samme værdier vi får tilbage. |

| | | |
|---|---|---|
| Stykliste.java | testStykliste() | For klassen Stykliste tester vi for om vi kan instantiere et stykliste objekt, tilføje en arrayList af materials, og tilføje et stykliste id, samt derefter kalde get metoder på disse, og kontrollere at det er de samme værdier vi får tilbage. |
| Encryption.java | testGetEncryptWord()<br><br>testGenerateSecurePassword()<br><br>testVerifyUserPassword() | For klassen Encryption tester vi at oprette et encryptWord word scheme som ikke er null, teste at vi kan hashe et bestemt password på et bestemt prædefineret word encryptWord word scheme, og så få en bestemt securePass krypterings kodeord ud af det, samt at vi kan verificere et hashet securePass password ved at kende kodeordet og det tilhørende encryptWord word scheme. |
| User.java | testUser() | For klassen User tester vi for om vi kan instantiere et user objekt, tilføje et krypteret kodeord, samt derefter konkludere at brugeren nu har et hashet krypteret kodeord ud fra et prædefineret encryptWord word scheme. |

**CommandTest.java**

| | | |
|---|---|---|
| PriceCommand.java | testPriceCommand() | For Command klassen PriceCommand tester vi ved brug af Mockito, at vi kan tilføje tomme mock elementer som stedtrædere for reel data, og derefter instantiere PriceCommand |

| | | |
|---|---|---|
| | | metoden, med en bestemt jsp side, og køre execute metoden med en mock request og en moc manager facade, for at se om vi får den rette jsp side tilbage, som simulering af et korrekt udfald af denne command. |
| EditLineItemCommand.java | testEditLineItemCommand() | For Command klassen testEditLineItemCommand tester vi ved brug af Mockito, at vi kan tilføje tomme mock elementer som intentionelt forkerte stedfortrædere for reel data, og derefter instantiere PriceCommand metoden, med en bestemt jsp side, og køre execute metoden med en mock request og en moc manager facade, for at se om vi får index.jsp siden tilbage, som simulering af et forkert udfald af denne command ved en bruger der ikke er logget ind. |
| LoginCommand.java | testLoginCommand() | For LoginCommand klassen PriceCommand tester vi ved brug af Mockito, at vi kan tilføje tomme mock elementer som stedtrædere for et bruger objekter med et password og en email, samt derefter instantiere LoginCommand metoden, med en bestemt jsp side, og køre execute metoden med en mock request og en mock manager facade, for at se om vi får den rette jsp side tilbage i tilfældet af at en bruger succesfuldt logger ind, som simulering af et korrekt udfald af denne command. |

# Bilag

## 1. Overordnet Interessent-beskrivelse af virksomheden

Siden IT-systemet, ud over oprettelse af ordrer, kun er tilgængeligt for Fog-medarbejdere kommer det ikke direkte til at påvirke kunde eller andre personer uden for Fog. En vigtig undtagelse er dog at det nye system opretter styklister og tegninger fuldkommen automatisk, uden behov for en Fog-ansat tage målene fra kundens ordre og indtaster dem i carport beregningsprogrammet. Dette giver mindre chance for fejl samt en hurtigere afvikling af ordrer.

Alt dette kommer nok til at påvirke mange ansat i Fog positivt. Lederen af enkelte Fog butikker kommer til at nyde godt af en automatisk afvikling af ordrer, både fordi det sker hurtigere og mere fejlfrit, men også fordi at det betyder at der er mindre brug for ansættelse af personer, der ellers ville have udført opgaven at skulle indtaste ordremål ind i beregningsprogrammet. Derfor er det vigtigt at Fog ansvarlig løbende drages ind i projektets eventuelle problemer og løsninger hertil, da denne mere end nogen andre brugere af dette system kommer til at blive påvirket af det færdige program. Der bør derfor tages hensyn til forskellige forhold, ændringer og nye funktioner som den Fog ansvarlige vil have implementeret i systemet. Af disse grunde bør Fog ansvarlig få en demonstration af demoen efter hvert sprint, så denne er i stand til at vurdere videre forholdsregler som udviklerne skal tage hensyn til.

Programmet, specielt automatisering af styklist oprettelse, kommer også til at påvirke Fog-salgsmedarbejderne positivt, da de vil være i stand til at generere en stykliste og tegning af carporten til vurdering, så snart at en ny ordre er blevet modtaget i systemet. Dette vil gøre dem i stand til at vurdere ordren og kontakte kunden meget hurtigere. Overordnet set er salgsmedarbejderens involvering i systemet mindre vigtig, men især på området med behandling af, og funktioner relateret til, ordre bør salgsmedarbejderen mindst et par gange have adgang til en demo af systemet samt mulighed for at give sin mening om programmet. Fog-pakkefolk vil også blive positivt påvirket af automatisering, da disse kan gå i gang med at samle styklisten lige så snart at salgsmedarbejderen har vurderet styklisten. Pakkefolk skal ligesom salgsmedarbejderen have mulighed for at kommentere på specifikke funktioner, specielt dem relateret til adgang til styklisten og adgang og funktioner relateret til materialer.

Personer der førhen har haft opgaven at indtaste ordremål ind i beregningsprogrammet vil sandsynligvis blive påvirket negativt. Med programmet automatiseret vil der ikke længere være brug for at de udfører deres opgave. Selv hvis der ikke er nogen fastansat til at overføre ordremål, vil det nye program stadigvæk frigøre mange arbejdstimer. Dette kunne gøre at Fog er nødt til at fyre nogle medarbejdere der førhen, blandt andet, har indtastet ordremål og udregnet carporte. Det ville dog på samme tid være positivt for Fog, da de nu skal lønne færre personer.

Både Fog byggeeksperter og kunder hos Fog vil begge kun opleve fordele. De har ikke rigtigt adgang til den indre del af systemet, og vil kun nyde fordele af et hurtigere,

automatisk system. Overordnet set kommer de fleste interessenter til at opleve fordele ved det nye system.

## 2. En Interessent-analyse, hvor I gør rede for de implicerede aktører og deres interesser i projektet.

Denne Interessent analyse blev lavet i forbindelse med forberedelsen til projektet, og er derfor en forbyggende analyse af systemets aktørere og ders interesser før peojektes igangsættelse.

| | Stor indflydelse | Lille indflydelse |
|---|---|---|
| Bliver påvirket af projektet | Ressourceperson | Gidsler |
| Bliver ikke påvirket af projektet | Grå eminence | Eksterne interessenter |

| **Interessent** | **Interessenten kan opleve følgende FORDELE ved projektet** | **Interessenten kan opleve følgende ULEMPER ved projektet** | **Samlet vurdering af interessentens bidrag/position** | **Håndtering af interessenten** |
|---|---|---|---|---|
| Fog ansvarlig -Ressourceperson | Mere effektiv overførsel af information | Mister kontrol over deres salgsproces | Det samlede udkom af projektet vil nytte dem | Interessenten skal etableres i funktionen af det nye system, med fokus på en målrettet effektivisering af deres platform og vedligeholdelse af kundeforhold. Der skal derved defineres efterfølgende milepæle, ansvarspersoner og opfølgnings-/evalueringsrutiner for projecktet. |
| Fog pakkefolk -Grå eminence | Ingen ændringer. | At processen går hurtigere hvilket vil sige at de skal pakke hurtigere | Der sker ikke ændringer i deres arbejdsgang. | De skal introduceres til at der kommer ændringer i hvordan det nye system fungere. Hvordan han selv og hans overordnet kan ændre priser. |
| Fog salgsperson -Gidsler | De kan komme igennem flere | Mindre kontrol over processen, da mere | Ikke så meget at bidrage. | Ny arbejdsgang for hvordan de får besked om |

| | | | | |
|---|---|---|---|---|
| | kunder hurtigere og får flere ressourcer til salgsdelen. | af det vil foregå i websystemet. | | hvor langt en kunde er i bestillingsprocessen.<br>De sidder ikke selv med det i hånden men de får bare en stykliste i hånden. |
| De anbefalede eksterne byggeeksperter -Eksterne interessenter | Mindre chance for fejl ved indtastning af data i systemet, og hurtigere afvikling af bestillinger. | Ingen umiddelbare ulemper | Projektet vil kun nytte dem | De skal informeres om at det nye system virker uden manuel overførsel af data |
| Kunderne -Eksterne interessenter | Give dem et produkt hurtigere. | Ingen umiddelbare ulemper | Nyt it-system | Kunder bliver introduceret til nyt system. Ny afvikling af ordren. |

## 3. En SWOT-analyse over kunden set fra kundens perspektiv. I skal simpelthen hjælpe Fog til at se deres situation i markedet, og hvordan det nye IT-projekt vil påvirke deres situation.

Denne SWOT-analyse over kunden set fra kundens perspektiv blev lavet i forbindelse med forberedelsen til projektet, og er derfor en forebyggende analyse for hvordan It-projektet vil påvirke Fog's situation før projektets endelige igangsættelse.

| **SWOT** | Helpful<br>*…to achieving the objective* | Harmful<br>*…to achieving the objective* |
|---|---|---|
| Internal Origin<br>*(Attributes of the organization)* | ***STRENGTHS*** | ***WEAKNESSES*** |
| | Mange års arbejde med carporte. Kan give præcise rammer for deres program. Det nye system kan gøre Fog mere konkurrencedygtige i den digitale verden. Mindre chancer for menneskelavet fejl eller forsinkelser og hindringer. | Programmet skal måske modificeres så det passer til nye computere. Integrationen med andre af Fogs systemer, kunne forårsage enkelte konflikter. |
| External Origin<br>*(Attributes of the environment)* | ***OPPORTUNITIES*** | ***THREATS*** |
| | Hurtigere afvikling af ordre og afvikling af de enkelte ordre med aktuelle materialer og priser. Muligheden for at redigere i databasens materialelister samt automatiseringen af deres nuværende manuelle ordresystem. Ved automatisering gennem en online platform, vil Fog kunne spare en masse ressourcer som førhen blev brugt på manuel indtastning og ordreoprettelse. | Ikke rigtigt noget, dog kunne det tænkes at noget af kontrollen der førhen lå hos Fog nu flyttes til selve systemet. |

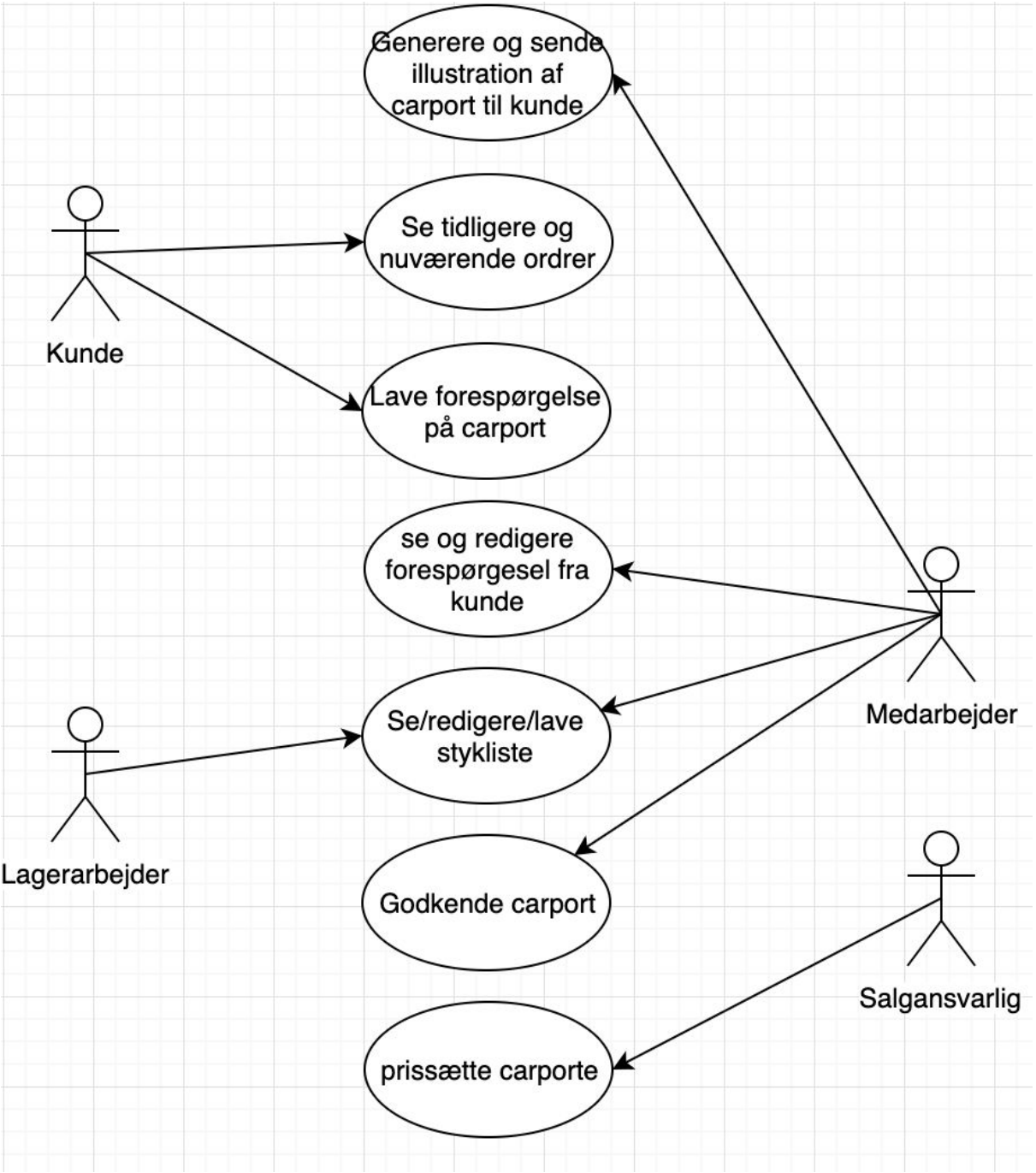
## 4. En SWOT-analyse over projektets chance for at blive en succes set fra udvikler teamets perspektiv.

Denne SWOT-analyse over projektets chance for at blive en succes blev lavet i forbindelse med forberedelsen til projektet. Analysen er derfor en forebyggende analyse for hvad chancen er for at It-projektet kan blive en succes fra udvikler teams perspektiv, før projektets endelige igangsættelse.

| **SWOT** | Helpful<br>*…to achieving the objective* | Harmful<br>*…to achieving the objective* |
|---|---|---|
| Internal Origin<br>*(Attributes of the organization)* | ***STRENGTHS*** | ***WEAKNESSES*** |
| | Programmet der udregner carporten findes allerede og skal bare bygges ovenpå. Kan køre på og i forbindelse med Fogs nye it systemer. | Fog har mindre direkte kontakt med indtastningen og valget af materialer til en ordre, som nu vil foregå i en komplet automatiserede proces istedet for en manuel proces. |
| External Origin<br>*(Attributes of the environment)* | ***OPPORTUNITIES*** | ***THREATS*** |
| | Hurtigere afvikling af ordrer med mindre muligheder for fejl. Færre arbejdstimer bruges på indtastning af data. Der kan redigeres direkte i både priser og materialedata. | Ovekomplicering eller konkrete fejl på systembasis, fundamental ændring af en ordreprocess ved automatisering. |

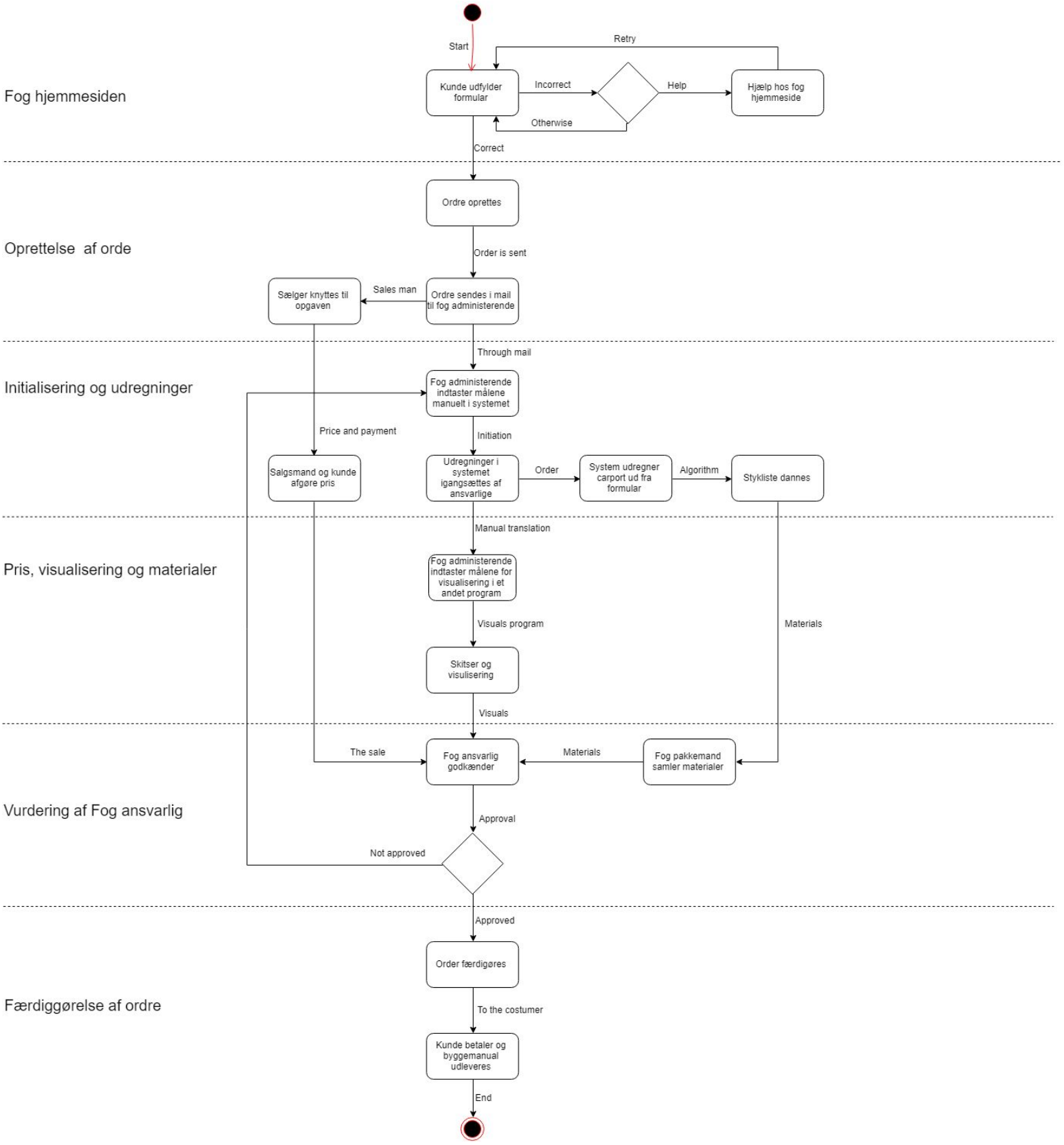
# 5. Use case diagram for planlægning af systemet

Dette Use case diagram blev lavet i forbindelse med forberedelsen til projektet, og er derfor et forebyggende diagram over de planlagte use cases for systemets forskellige aktørere, før projektets endelige igangsættelse.



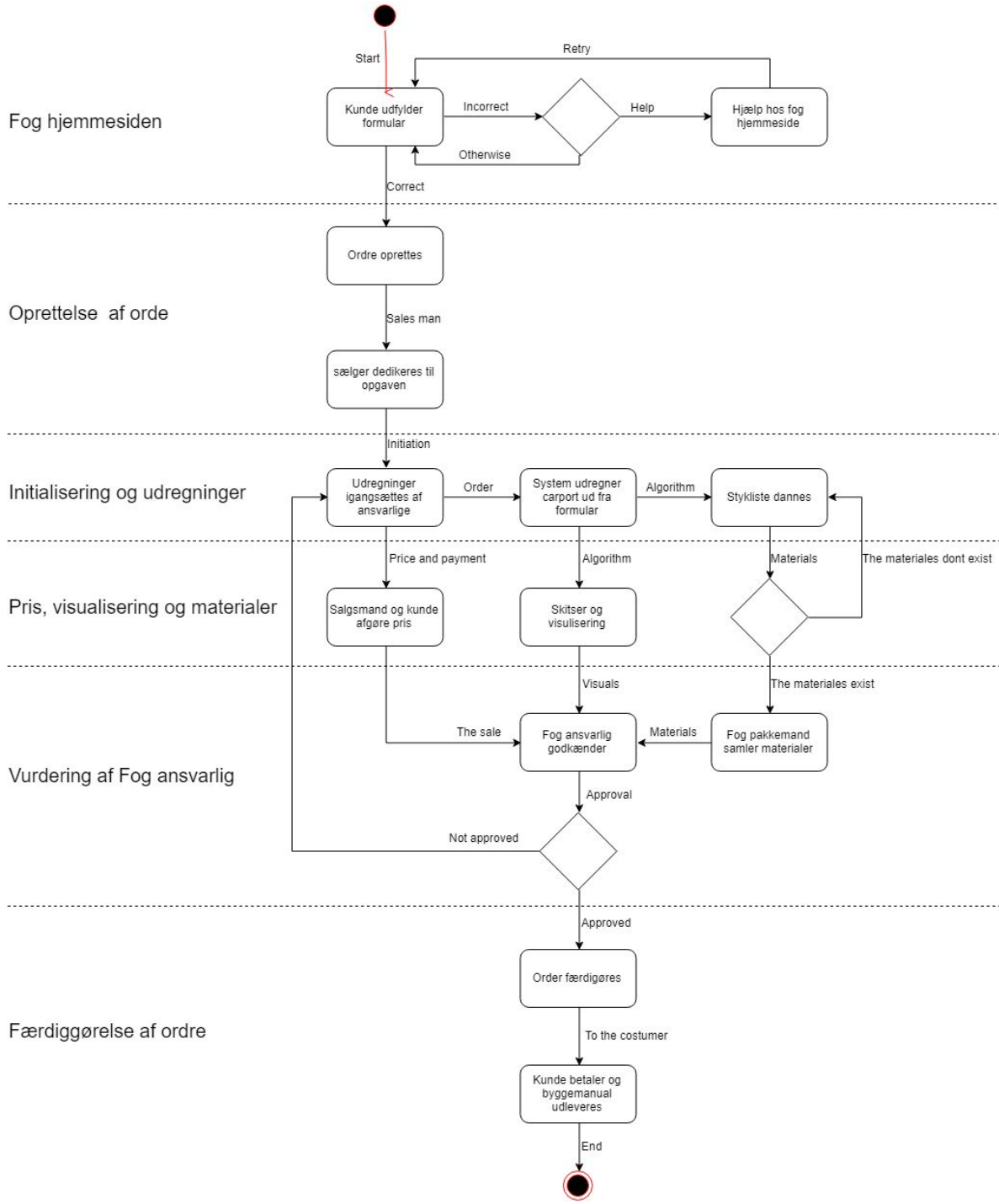
# 6. As-is Aktivitetsdiagram for planlægning af systemet

Dette as-is Aktivitetsdiagram blev lavet i forbindelse med forberedelsen til projektet, og er derfor et forebyggende diagram over Fog nuværende position i forhold til it-løsninger. Dette blev igen lavet som forberedelse før projektets endelige igangsættelse.



# 7. To-be Aktivitetsdiagram for planlægning af systemet

Dette to-be Aktivitetsdiagram blev lavet i forbindelse med forberedelsen til projektet, og er derfor et forebyggende diagram over de planlagte aktivitetsflow for systemets forskellige faser fordelt i lag. Dette blev igen lavet som forberedelse før projektets endelige igangsættelse.



# 8. Den komplette project backlog for SCRUM user stories, acceptance, criteria og tasks.

De følgende sektioner indeholder vores brugte SCRUM user stories, fordelt over 4 sprints, som vi i sammenhæng med product owner planlagde og prioriterede gennem projektet. Under hvert sprint vil der også være et link til den tilhørende side i online platformen Taiga, som er der hvor planlægningen officielt foregik.

Projekt Backlog:
https://tree.taiga.io/project/razz7-fog/backlog

## Sprint 1

https://tree.taiga.io/project/razz7-fog/taskboard/forste-sprint-3

| |
|---|
| Som Fog medarbejder vil jeg gerne visuelt kunne se fog-lagerets gemte materialedata så jeg kan vurdere dets relevans for en stykliste.<br><br>**Acceptance Criteria:**<br>1. Jeg skal visuelt og struktureret kunne se fog-lagerets gemte materialedata.<br>2. Jeg skal visuelt og struktureret kunne se pris, id , højde, brede osv for fog-lagerets gemte materialedata.<br>3. Jeg skal ikke direkte kunne redigere i materialedata gennem denne visualisering af fog-lagerets gemte materialedata.<br><br>**Tasks:**<br>Business lag java funktion til objekter for materialedata. 3 timer<br>Front end interface til visualisering af materialedata 3 timer<br>DB funktion til at trække aktuel materiale data fra databasen. 5 timer |
| Som Fog medarbejder vil jeg gerne kunne ændre priser, navne og andet materialedata i systemet så jeg kan holde det opdateret og hvis der kommer nye produkter eller ændringer som vi vil bruge i stedet for noget andet.<br><br>**Acceptance Criteria:**<br>1. Jeg skal kunne oprette og gemme materialedata i systemet.<br>2. Jeg skal kunne fjerne materialedata der er gemt i systemet.<br>3. Jeg skal kunne opdatere materialedata der er gemt i systemet.<br>4. Jeg skal ikke kunne fjerne hele informationslagere for materialedata i systemet.<br>5. Jeg skal ikke kunne lave nye informationslagere for materialedata i systemet.<br><br>**Tasks:**<br>Front end Interface til redigering materialedata. 3 timer<br>DB funktion til redigering i materialedata. 5 timer<br>Business lag java funktion til redigering og erstatning af materialedata object. 3 timer |
| **Storyless tasks:** |

Oprettelse af businesslag og en facade class. 1 time
Oprettelse af DB lag og db-connector - 3 timer
Oprettelse af præsensationslag og front controller. 1 time

## Sprint 2

https://tree.taiga.io/project/razz7-fog/taskboard/anden-spring

Som Fog medarbejder, vil jeg gerne visuelt kunne se styklister, så jeg kan tilgå materialerne til den enkelte carport.

**Acceptance Criteria:**
1. Som Fog-medarbejder kan jeg se en visualisering af alle styklister til alle carporte.
2. Jeg vil gerne kunne klikke på en stykliste og få en liste af materialer visualiseret.
3. Jeg vil gerne kunne se en visualisering af relevant bruger og orderdata tilknyttet styklisten.
2. Jeg kan ikke direkte ændre i styklisterne gennem denne visualisering.

**Tasks:**
DB Funktion der trækker materialer  fra databasen. 3 timer
Front end funktion til præsentation af stykliste. 3 timer

Som Fog ansvarlig eller salgsmedarbejder vil jeg gerne kunne visuelt se og redigere styklister, så jeg kan ændre i dem hvis situationen kræver det.

**Acceptance Criteria:**
2. Jeg skal kunne visuelt vælge, hente og visuelt se styklister.
3. Jeg skal kunne redigere i de essentielle dele af styklisten for henholdsvis materialer, pris, mængde osv..
4. Jeg skal ikke kunne redigere direkte i udregningen af styklisten.

**Tasks:**
Business lag java funktion til redigering og erstatning af stykliste object. 5 timer
DB funktion til redigering i stykliste tabellen. 2 timer

Som Fog ansvarlig eller salgsmedarbejder vil jeg gerne kunne oprette en stykliste, så jeg kan liste materialerne til enkelte carporte.

**Acceptance Criteria:**
1. Jeg vil gerne kunne oprette en visuel stykliste gennem systemets funktion.
2. Jeg vil gerne kunne igangsætte de enkelte styklister og deres status, efter grundig samtale med kunde.
3. Jeg kan ikke direkte redigere i selve udregningerne af styklisterne gennem denne oprettelse.

**Tasks:**

Business lag der laver stykliste objekter. 5 timer
Udregning af stykliste ud fra Java algoritme med mål der er givet. 20 timer
Front end funktion der generer stk liste. 2 timer
Stykliste i database 3 timer
Front end Interface til redigering af stykliste data. 5 timer

**Storyless tasks:**
Front end Interface der viser stk liste. 2 timer
Oprettelse af Front end JSP, bootstrap og css style. 1 time

## Sprint 3

https://tree.taiga.io/project/razz7-fog/taskboard/3-sprint-93

Som Fog medarbejder vil jeg gerne kunne se alle stk. lister og tidligere ordre, så jeg altid kan gå tilbage og se tidligere produkter.

**Acceptance Criteria:**

1. Som Fog-medarbejder kan jeg se en visualisering af alle styklister til alle carporte.
2. Jeg vil gerne kunne klikke mig ind på den specifikke stykliste.
3. Jeg vil gerne kunne se relevant data om styklisten.
4. Jeg vil gerne kunne klikke mig tilbage og derved skifte mellem de tidligere styklister.

**Tasks:**
Præsentationslag command til forberedelse af styklister
Versionsstyring af materialer 2 timer
Stykliste og ordre db-metoder
Frontend - visning af ordrer og stykliste 3 timer

Som fogmedarbejder vil jeg kunne se den specifikke carport-tegning skalere sig i henhold til carportens mål, så jeg nøjagtigt kan se de visuelle referencer mellem stylisternes forskellige mål.

**Acceptance Criteria:**

1. Jeg kan se ændringer på min tegning hvis genere den igen med nye mål
2. Jeg kan klikke videre og se en stk liste.
3. Jeg kan se carportens højder, bredder og længder skalere sig i henhold til stk listen

**Tasks:**
Front-end oprettelse af SVG grafiske enheder i JSP
Præsentationslag command og front controller sider til grafik funktionen
Front-end oprettelse og konvertering af carport mål til SVG i JSP

Som bruger af platformen vil jeg kunne se en grafisk 2D tegning af den udvalgte carport

carport fra siden, så jeg kan visualisere carporten.

**Acceptance Criteria:**

1. Jeg kan trykke på en knap som generer og viser en tegning.
2. Min tegning vises på en ny side.
3. Min tegning vises fra siden
4. Carport skal med eller uden skur

**Tasks:**
Carport tegning på ny side

**Storyless tasks:**
Shop - 3 timer
Start på unit-tests Black/White-box
Finde ud af hvorfor der er dataloss i "<input ="type" format> 30 timer

## Sprint 4

https://tree.taiga.io/project/razz7-fog/taskboard/sprint-4-2213

Som bruger af platformen vil jeg gerne kunne logge ind i systemet, så jeg kan tilgå min konti.

**Acceptance Criteria:**

1. Jeg skal kunne logge ind med mit eget password og mit eget brugernavn gennem en visuelt menu.
2. Jeg skal kunne logge af min bruger/ konti.
3. Jeg skal kunne tilgå en online session og forlade den når jeg logger af.

**Tasks:**
Frontend funktion til login af bruger

Som bruger af platformen vil jeg gerne kunne oprette en ny bruger i systemet, så jeg kan tilføje en ny konti.

**Acceptance Criteria:**

1. Jeg skal kunne skrive informationer om mit navn, efternavn, adresse og telefonnummer.
2. Jeg skal kunne tilgå min egen konti efter oprettelse.
3. Jeg skal visuelt kunne se informationer om min egen konti.
4. Jeg skal ikke direkte kunne tilgå andre brugeres konti.

**Tasks:**
Opret kundetabel
Frontend funktion til oprettelse af bruger

Som Fog medarbejder vil jeg kunne vurdere carporte, så jeg kan tilpasse kunden en bedre pris.

**Acceptance Criteria:**

1. Jeg skal kunne hente, åbne og se visualiseringen carporten
2. Jeg skal kunne hente, åbne og se visualiseringen af styklisten
3. Jeg skal kunne udregne og oprette en pris ud fra de visualiserede styklister.
4. Jeg skal kunne redigere prisen efter oprettelse.

**Tasks:**
Frontend funk til prissætning af ordre
Frontend funktion til visualisering af carporten
Frontend funktion til print af stykliste

Som Fog ansvarlig vil jeg gerne kunne se og godkende carport ordre, så jeg kan sikre kvalitet og validitet af produktet.

**Acceptance Criteria:**

1. Hvis jeg er Fog ansvarlig vil jeg gerne kunne færdiggøre en bestilling af en carport samt dets status.
2. Jeg kan godkende en carport, så den er klar.
3. Jeg kan fuldende en ordre.
4. Jeg kan ikke direkte prissætte en carport.

**Tasks:**
Frontend funktion til præsentation af ordre
Frontend funktion til godkendelse af ordre

Som bruger af platformen vil jeg gerne kunne indtaste mine egne mål og en carport type på Fog's hjemmeside, så jeg kan oprette en carport ordre.

**Acceptance Criteria:**

1. Jeg kan tilgå et visuelt udfyldeligt skema.
2. Jeg kan indtaste højde i systemet.
3. Jeg kan indtaste længde i systemet
4. Jeg kan indtaste bredde i systemet
5. Jeg kan tilføje en carport type.
6. jeg kan ikke redigere i selve carport udregningerne.
7. Jeg kan ikke se andre kunders carporte.
8. Jeg kan ikke oprette flere carporte i en ordre.

**Tasks:**
Frontend funktion til indtastning af carportmål

Som Fog medarbejder vil jeg gerne kunne tilgå kundernes ordre visuelt, så jeg bedst muligt kan assistere dem i deres køb.

**Acceptance Criteria:**

1. Jeg vil gerne visuelt kunne se enkelte kunders ordre.
2. Jeg vil gerne visuelt kunne se enkelte kunders styklister.
3. Jeg vil gerne kunne se visualisering af enkelte kunders carporte.
4. Jeg vil ikke kunne ændre direkte i kundens ordre, styklister eller carporte gennem visualiseringen.

**Tasks:**
Frontend funktion til adgang til SVG tegning
Frontend funktion til adgang til ordre

---

Som bruger af platformen vil jeg gerne visuelt kunne se mine ordre, så jeg kan se deres status, brugerdefinerede mål og carport.

**Acceptance Criteria:**

1. Som kunde kan jeg visuelt se den bestilte carport.
2. Jeg kan visuelt se mine brugerdefinerede detaljer om carporten.
3. Jeg kan visuelt se information om ordrens status.
4. Jeg kan ikke direkte redigere på ordren gennem visualiseringen.
5. Jeg kan ikke se information om prisudregning eller information om den endelige stykliste før fuldendelse af ordren.

**Tasks:**
Status på ordre
Adgang til brugerdefinerede detalje i ordren
Frontend funktion ordre stykliste

---

Som bruger af platformen, vil jeg gerne kunne tilgå systemet online, så jeg kan tilgå systemet forskellige steder fra.

**Acceptance Criteria:**

1. Jeg skal kunne tilgå en web-adresse der føre mig direkte til systemet.
2. Jeg skal kunne bruge systemet gennem en række websider.
3. Jeg skal kunne indtaste informationer gemmes og overføres mellem de forskellige sider gennem en online session.
4. Jeg skal kunne interagere med et sted hvor data kan blive gemt gennem brugen af selve onlinesystemet.

**Tasks:**
Database lægges på droplet
System lægges på droplet

---

**Storyless tasks:**
Test database oprettes på droplet
Unit tests
Error handeling