Fog Carports

Gruppe 3 hold B

Nicolai Rosendahl - cph-nr135@cphbusiness.dk - github: MrJustMeDahl

Pelle Hald Vedsmand - cph-pv73@cphbusiness.dk - github: pelle112112

Carsten Juhl - cph-cj505@cphbusiness.dk - github: CarstenJuhl

Danyal Kitir - cph-dk174@cphbusiness.dk - github: DanyLoyal

31/5/2023

Antal anslag: 89.773

(Modeller og diagrammer er inkl og tæller for 800 anslag

Screendumps af kodefragmenter, samt bilag er ikke talt med, da

især kodefragmenterne kun er på få linjer)

Links:

- **Github repository:** https://github.com/MrJustMeDahl/CarportFog
- **Github pages javadoc:** https://mrjustmedahl.github.io/CarportFog/
- **Link til demovideo** : https://youtu.be/TmQujEECKKA
- **Link til webapplikation:** http://134.122.87.83:8080/carport/

  Login: admin@admin.dk
  kode: admin

Indholdsfortegnelse

Indholdsfortegnelse **1**
Indledning: **3**
Baggrund: **3**
Virksomheden/Forretningsforståelse: **4**
SWOT-analyse 4
Interessentanalyse 5
Teknologi valg: **6**
Krav: **6**
Funktionelle krav i form af user stories: 6
Ikke funktionelle krav: 7
Arbejdsgange der skal IT-støttes: **7**
User stories: **9**
US-1: 9
US-2: 10
US-3: 10
US-4: 11
US-5 11
US-6: 12
US-7: 12
US-8: 13
US-9: 13
US-10: 14
US-11: 14
Modeller, diagrammer og mockups: **15**
Domæne model: 15
EER-Diagram: 16
Klassediagram: 18
Sekvensdiagram: 19
Navigationsdiagram: 20
Figma mockups: 21
Valg af arkitektur: **22**
MVC: 22
Facade pattern: 22
Dependency: 23
Særlige forhold: **23**
Scopes: 23
Exceptions: 24
Javadoc: 24
Validering (Brugerinputs): 24
Validering (Login): 25

Brugertyper: 25
Materiale beregner (stykliste): 26
Udvalgte kodeeksempler: 27
Materiale beregner ud fra spild: 27
Javascript på produktside: 27
Data konsistens mellem Java og DB: 28
Opdater Materiale: 29
Status på implementering: 30
User stories: 30
Afsluttende bemærkninger: 32
Test: 33
Testcoverage: 33
Unittest eksempel: 36
Integrationstest eksempel: 36
Proces 37
Arbejdsprocessen faktuelt 37
Arbejdsprocessen reflekteret 40
Bilag: 42
Bilag 1.1 42
Bilag 1.2 42
Bilag 2.1 43
Bilag 2.2 43
Bilag 3.1 44
Bilag 3.2 44
Bilag 4.1 45
Bilag 4.2 46
Bilag 5. 47

Indledning:

Denne rapport er produktet af afsluttet semesterprojekt, udført af gruppe 3, Hold B. Et projekt hvor vi skulle udvikle en webshop med salg af carporte for Fog byggemarked.

Vi har valgt at have fokus på funktionalitet, samt simplicitet. *Det skal være nemt og bekvemt*. En køber skal holdes i hånden hele vejen, skal kunne bestille og i sidste ende betale for en carport uden for mange bump på vejen. På samme tid, skal en sælger kunne administrere ordrer og materialer.
I rapporten beskriver vi de forskellige teknologier og værktøjer, vi har benyttet, og hvordan vi har navigeret imellem dem, for at skabe en løsning der fungerer. Rapporten reflekterer det arbejde vi har lagt i projektet, og de succeser samt udfordringer vi har haft i forbindelse med skabelsen af webapplikationen.
Vi håber at denne rapport vil give et indblik i vores process, tanker og beslutninger i forbindelse med skabelsen af dette projekt.

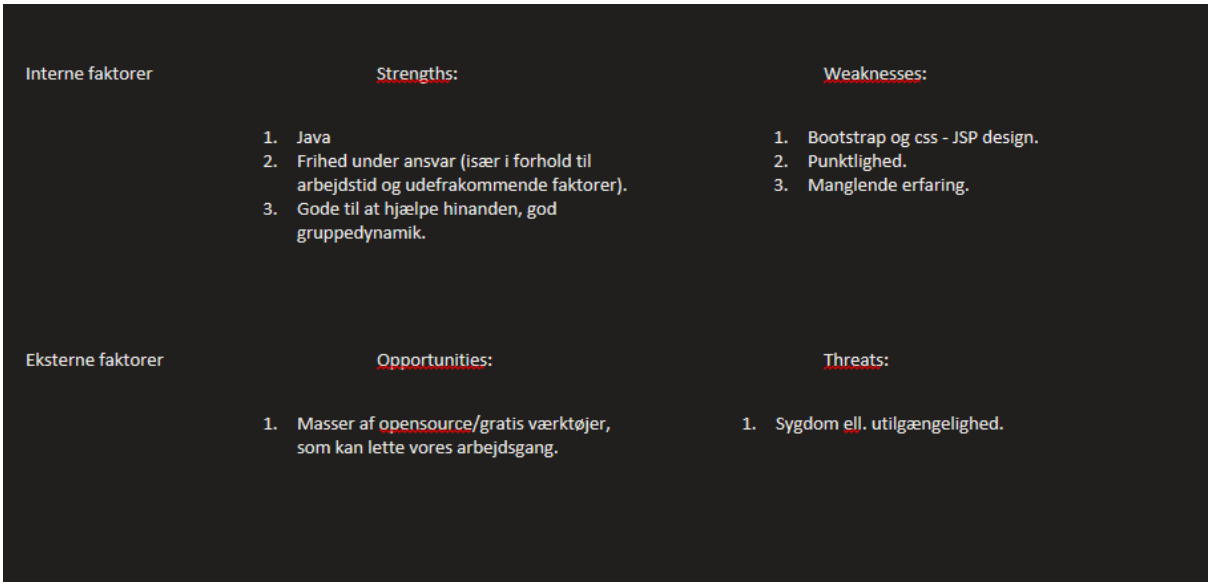
Baggrund:

- *Vi er blevet instrueret i at lave et mindre program for Fog byggemarked, og trælast. Fog specialiserer sig især inden for tømmerhandel, med samlede og "gør-det-selv" projekter*

- *Fog ønsker at få et program, som gør det nemt og simpelt at bestille en Carport, med eller uden skur, efter egne mål. Bestilling skal kunne gøres så flydende som muligt, dog uden at automatisere alle handlinger. En sælger skal være i stand til at håndtere kundeordrer inden de gennemføres, således at man kan mindske "dumme"fejl. Derudover ønsker Fog, at have muligheden for at kunne redigere/ tilføje materialer i deres sortiment.*

Virksomheden/Forretningsforståelse:

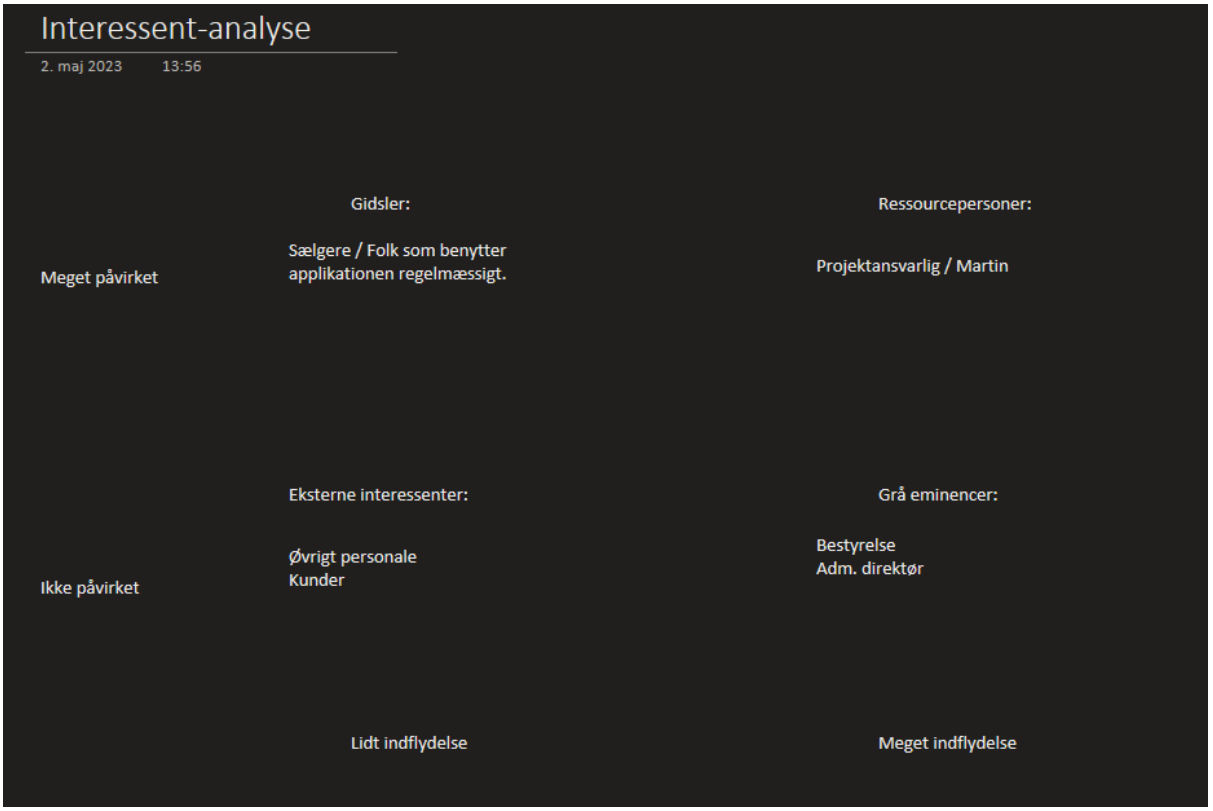
SWOT-analyse



SWOT-analysen, som det ses ovenfor, er udarbejdet på os som gruppe, hvilket afspejler sig i at det er meget begrænset, hvad vi som gruppe har haft af eksterne muligheder og trusler. Analysen er lavet med henblik på at gøre os opmærksomme på, hvad vi skal bruge ekstra energi på i projektforløbet, samt hvordan vi skal prioritere vores tid, så vi eksempelvis ikke glemmer en del af projektet, bare fordi den del ikke falder til vores højre ben. Under vores arbejde med SWOT-analysen kom vi frem til et par konklusioner om vores gruppedynamik. Selvom alle ikke er på samme niveau kodemæssigt, så har vi til dels de samme styrker og svagheder.
I vores gruppe er vi generelt gode til at programmere i Java, og det har vi helt klart kunne udnytte ved udviklingen af programmet. Vi fik sat metoderne op i vores mapper funktioner, som gjorde det nemt at udvikle hjemmesiden.
Vores gruppe arbejder kraftigt under ideen “frihed under ansvar”. Her er det muligt for gruppemedlemmerne at disponere deres tid, når de føler det bekvemt. Dette kan også ses som en svaghed i nogle grupper, hvis man ikke er i stand til at håndhæve dette.
Dog har vores gode gruppedynamik sørget for en god arbejdsgang, hvor alle har haft mulighed for at få hjælp under udviklingen.
Vores manglende erfaring indenfor Bootstrap og CSS, samt design i JSP er også noget der kan afspejles i projektets struktur. Vi har altid været fokuserede på funktionaliteten i vores program kontra design. Derudover kan vores manglende punktlighed også have et par problemer både inden for gruppearbejdet, men også for selve produktet. Vores produkt kan ses som en flydende proces, med mangel på præcise rammer, der kan beskrive programmets færdige state.
Generelt har vi ikke haft mange trusler i vores gruppe. Det, der nok kunne hindre vores projekt mest, ville være sygdom eller anden mangel på tilgængelighed. Skulle vi blive udsat

for sygdom eller anden utilgængelighed af et eller flere gruppemedlemmer, har vi aftalt at tage vores forventninger og ambitioner omkring projektet op til genovervejelse.

Interessentanalyse



Under udviklingen af interessentanalysen blev vi usikre på vigtigheden af denne analyse, og hvordan den skulle udvikles.
Er vi ansatte hos Fog eller er vi eksterne programmører?
I dette tilfælde er vi udefrakommende programmører, der udvikler programmet for Martin.
Ressourcepersonen er Martin, da han er den person der sidder og bruger programmet, ved samtidig at have stor indflydelse på designet af programmet.
Gidslerne er i dette tilfælde de folk som benytter applikationen regelmæssigt, uden at have indflydelse på udviklingen af programmet. Dette kunne være ansatte sælgere.
De grå eminencer er bestyrelsen, som bestemmer budgettet og andre beslutninger der skulle forekomme.
De eksterne interessenter er det øvrige personale, som ikke bliver direkte påvirket af programmet. Dog kan øvrigt personel blive påvirket af programmet på andre måder.
Programmet udvikler nye styklister, hvor materialerne skal plukkes ud fra disse af medarbejderne, og de kan måske bruge andre slags materialer.

Med vores funktion som udviklere, har vi haft henblik på, at Martin og gidslerne i store træk har haft primært brug for løsning.

Teknologi valg:

- Intellij v2021.3
    - Der blev benyttet 2021.3 og 2023
- Mysql connector v8.0.30 (JDBC)
    - Databasen er sat op i Mysql. Queries er kørt i workbench før programstart.
    - Workbench vi har benyttet er v8.0.
- JSTL v1.2
- Maven project
- Java
    - Vi har kørt forskellige JDKs, men primært version 11 og 18.
    - Vi har haft problemer med opsætning af Droplet, hvor Ubuntu kører med JDK 11.
- PlantUML v8059
- Javax servlets v4.0.1
- JUNIT v5.8.2
- Tomcat (local) v9.073
- Digital Ocean Droplet
- Droplet med Ubuntu, samt Java JDK 11
- Bootstrap 5

Krav:

Funktionelle krav i form af user stories:

1. Alle brugere skal logge ind for at benytte siden. Deraf skal man som besøgende have mulighed for at oprette en ny konto.
2. Som kunde skal man have mulighed for at bestille et tilbud på en carport efter egne mål.
3. Som kunde har man mulighed for at vælge om man vil tilkøbe et skur efter egne mål til sin carport, tilbuddet gives på den samlede ordre.
4. Som kunde skal man have mulighed for at følge op de tilbud man har bestilt samt godkende eller afvise tilbuddet.
5. Som kunde skal man modtage en kvittering og stykliste efter tilbuddet er betalt.
6. Administratorer skal have mulighed for at se bestilte tilbud og om nødvendigt have adgang til kundens kontaktoplysninger, hvis der skulle være opfølgende spørgsmål til bestillingen.

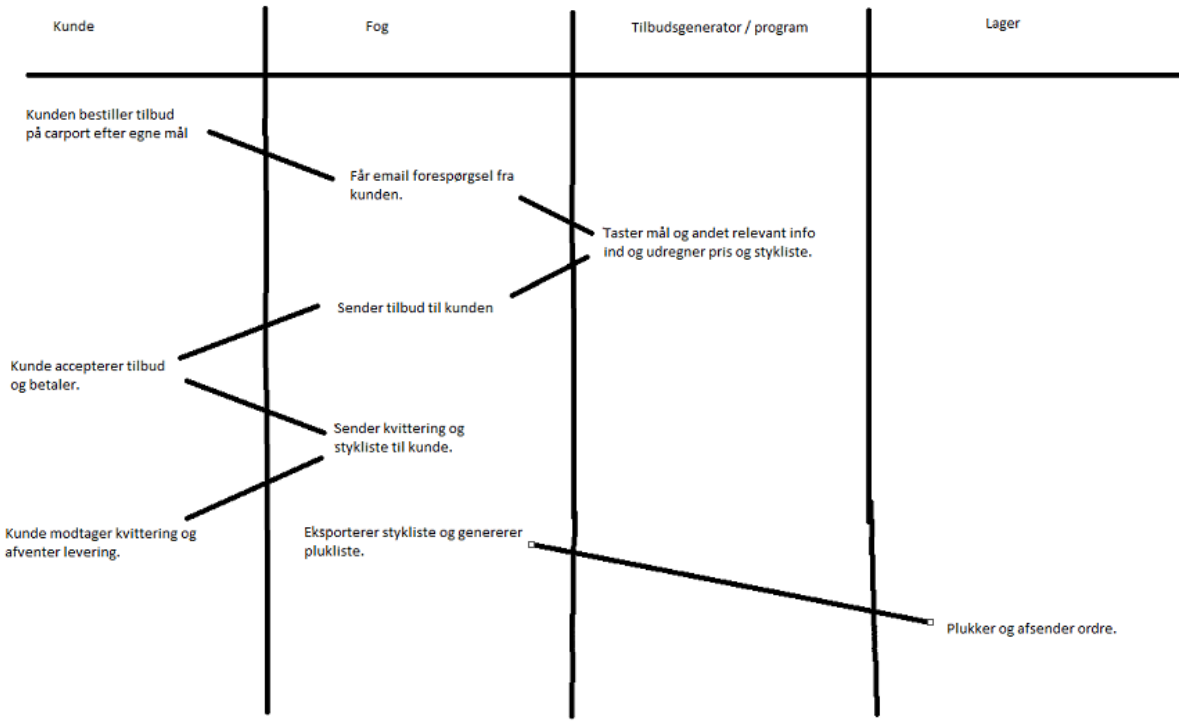
7. Administratorer skal kunne se en stykliste for de bestilte tilbud, som indeholder de nødvendige materialer der er påkrævet for at bygge produktet.
8. Administratorer skal kunne sætte en pris og sende tilbuddet til kunden.
9. Administratorer skal kunne følge op på alle ordrerne der ligger i systemet og slette dem om nødvendigt.
10. Administratorer skal have mulighed for at tilføje nye materialer, samt ændre beskrivelsen og prisen på eksisterende materialer.

Ikke funktionelle krav:

1. Løsningen skal baseres på en MySQL 8 database.
2. Designet skal afspejle en flerlags arkitektur, der afvikles på en Java server (Tomcat 9).
3. Webapplikationen bør bygges af almindelige Java klasser, servlets, og JSP-sider.
4. Websiderne skal være fuldt funktionelle i enten Google Chrome eller Firefox.
5. Websitet skal deployes i skyen (Digital Ocean).
6. Kildekoden skal være tilgængelig i et GitHub repository.
7. Den bestilte carport skal kunne printes på en 3D printer. I kan f.eks. aflevere en *.stl fil, som dokumentation og vise modellen frem til eksamen. I kan også indsætte fotos i rapporten af den printede udgave.

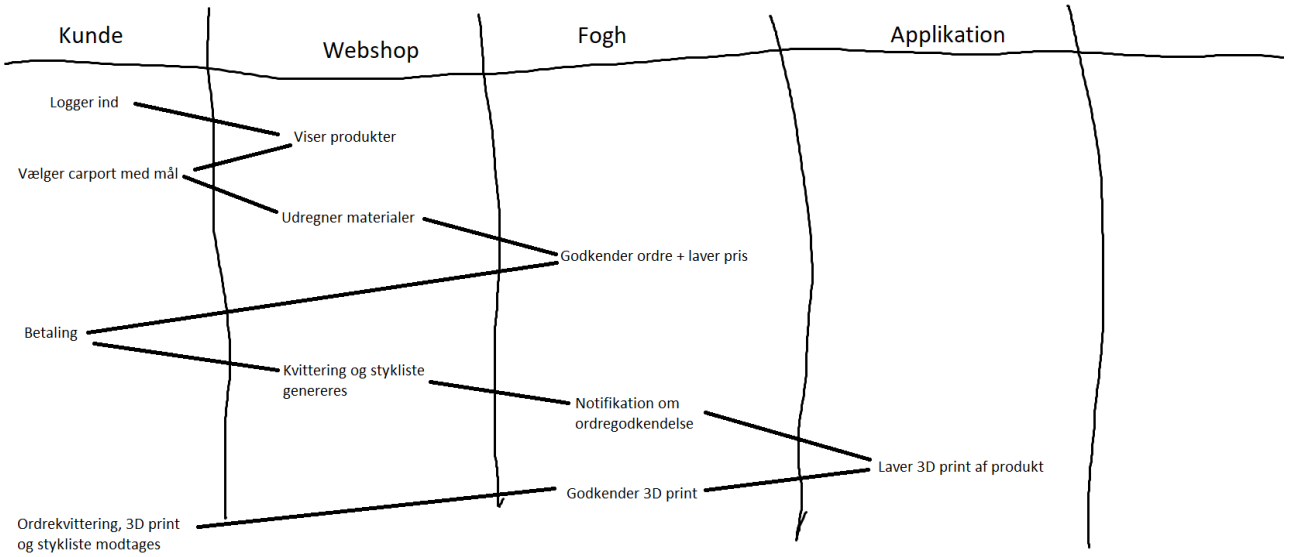
Arbejdsgange der skal IT-støttes:

Før udviklingen af vores produkt er Fog allerede i besiddelse af et fungerende system. De vil dog gerne have strømlinet et par processer i deres arbejdsgang. Herunder er et aktivitetsdiagram af Fogs håndtering af kunders ordrer af Carporte:



Ovenstående diagram er derfor "as-is", da det er den arbejdsmåde, de bruger på nuværende tidspunkt. Kunden vælger deres mål for carporten og et evt skur, for så at sende det videre til Fog. Her bruger Martin en separat applikation til at udregne materialemængder, længder og pris for carporten. Han vil derefter ringe til kunden og give dem et tilbud på carporten. Her har Martin mulighed for at ændre på målene for carporten, hvis de ikke giver mening.
Kunden har derefter mulighed for at acceptere tilbuddet samt betale.
Martin vil derefter tilsende dem en kvittering og en stykliste, samt generere en plukliste for hans medarbejdere.
Martin vil som sagt gerne have strømlinet hele denne proces, ved at samle alle applikationerne ét sted. Martin har derudover heller ikke mulighed for at redigere eller tilføje materialer i databasen.

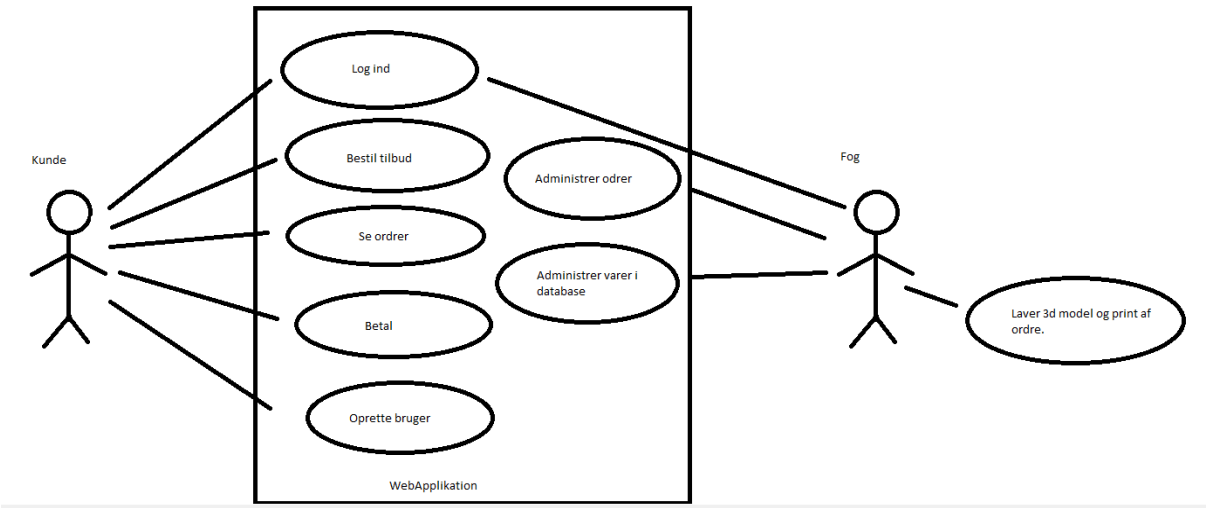
Vi har udarbejdet et lidt anderledes aktivitetsdiagram, som dog til dels ligner meget "as-is" diagrammet:



Ovenstående diagram er det såkaldte "to-be" aktivitetsdiagram, og det viser processerne, som vi gerne vil opnå ved bestilling af en carport.
De fleste af processerne sker gennem hjemmesiden, hvor både styklisten og materiale beregneren er inkorporeret.
Kunden starter med at vælge en carport på hjemmesiden, hvorefter de indtaster deres egne mål. Her vil der blive genereret en vejledende pris for materialerne. Ordren sendes nu videre til Fog, som så har mulighed for at kontakte kunden om carportens mål og pris.
Martin vil stadig gerne have det personlige touch som sælger ved at ringe kunden op med et tilbud efter at have modtaget en bestilling fra kunden, da det stadig er en del af deres service og pris.
Efter kunden har betalt for produktet, vil det blive sendt tilbage til Fog. Her vil styklisten blive genereret, hvorefter han vil have mulighed for at 3D-printe en model af carporten.

User stories:

Vi har illustreret hvilken funktionalitet der er aftalt med kunden i følgende use-case diagram og beskrevet hvordan vores user stories imødekommer den aftalte funktionalitet derefter:



Her er det relevant at påpege at det er aftalt, at alt aftalt funktionalitet skal implementeres på den samme webapplikation. Det vil sige at webapplikationen skal kunne håndtere alle funktionelle krav der vedrører Fogs kunder og personale. Undtaget er generering af 3D modellerne, der skulle implementeres som en enkeltstående applikation. Use-case diagrammet er udarbejdet med henblik på at give kunden og programmøren et fælles overskueligt billede af hvilke aktører der skal kunne udføre hvilke handlinger på webapplikationen.
I vores analyse af domænet har vi udarbejdet 11 user stories, som dækker kravene stillet af kunden. Disse 11 user stories er udarbejdet på baggrund af det første kunde interview med Martin fra Fog samt efterfølgende samtaler omkring domænet med kunden/vejledermøder. Vi vil i herunder beskrive hver af vores user stories med acceptkriterier, og komme med en kort analyse af hvad det kræver at implementere dem, samt hvor stor en opgave det bliver. Vi vurderer opgavernes størrelse ud fra om hvor meget data der skal behandles, hvor mange udregninger der skal implementeres, hvor mange lag af systemet der skal arbejdes på og hvor høje krav der er til sikre data konsistens, samt i mindre grad hvor komplekse jsp-siderne vil være (jo mere indhold på siden, des højere kompleksitet).

US-1:

Som eksisterende bruger, skal jeg kunne logge ind på siden, så jeg herefter får mulighed for at bestille produkter på siden.

Acceptkriterie: Når man trykker på “Login” knappen, bliver man redirected til loginsiden. Her skal man både indtaste email og password. Email og password er case-sensitive. Hvis man ikke indtaster begge, bliver man sendt til fejlsiden. Hvis man indtaster de korrekte oplysninger, så bliver man sendt videre til produktsiden.

US-1 er knyttet til “Log ind” funktionaliteten på use-case diagrammet. Denne user story adskiller den nye webapplikation fra den gamle, da det nu er nødvendigt at logge ind for at benytte applikationen. Dette gøres for at kunderne har mulighed for at gemme deres ordre i en indkøbskurv samt får mulighed for at administrere deres ordrer via applikationen.
For at få implementeret denne user story kræves der en side, hvorpå brugeren kan logge ind, samt at den data der er tilknyttet en bruger er på plads, herunder brugerens personlige oplysninger, ordrer, osv. Vi vurderer derfor dette til at være en large opgave, da den kræver

definition og implementering af en stor del af backenden for at være opfyldt på en valid måde.

US-2:

Som ny bruger, skal jeg kunne oprette en konto på siden, så jeg har mulighed for at logge ind.

Acceptkriterie: Når man trykker på “opret” på forsiden, bliver man sendt videre til en “opret-konto” side. Her skal man indtaste email og password, hvilket begge er case-sensitive. Hvis man ikke indtaster alle oplysninger, bliver man sendt videre til fejlsiden. Hvis man har indtastet oplysningerne korrekt, bliver man nu sendt videre til “login-siden”.

US-2 er knyttet til “opret bruger” funktionaliteten, som fremgår af use-case diagrammet. Eftersom at det på den nye webapplikation vil være et krav at logge ind for at benytte applikationen, så skal man som besøgende på sitet naturligvis have mulighed for at oprette en konto. En fuld implementering af denne user story kræver en side, hvorpå man kan indtaste brugeroplysninger, samt at den indtastede data valideres og gemmes i databasen. Vi vurderer derfor dette til at være en small opgave, da der ikke er stor sammenhængskraft mellem denne del af webapplikationen og resten af systemet.

US-3:

Som registreret kunde, skal jeg have mulighed for at bestille et tilbud på en carport efter egne mål.

Acceptkriterie: Når man vælger en carport, bliver man sendt videre til en “bestillingsside”, hvor man kan indtaste egne mål for længde, bredde og højde på carporten. Her vil der være et par standardmål, som allerede er preselected. Man vælger målene fra en drop down liste, hvor der er minimum- og maksimummål. Her kan man også se en vejledende pris på carporten. Efter man har valgt sine givne mål og har trykket på “Få et tilbud”, vil ordren nu blive tilsendt administratoren.

Denne user story er knyttet til “bestil carport” funktionaliteten og dækker alle de steps i aktivitetsdiagrammet (to-be), som kunden og webapplikationen udfører før bestillingen havner hos Fog første gang - foruden tilføjelse af skuret. Det vil sige både fremvisning af produkter, indtastning af egne mål til carporten og at applikationen udregner en stykliste som passer til det valgte produkt.
Implementering af denne user story indebærer: 3 JSP-sider, herunder en side til valg af produkt, en side til indtastning af mål på produktet og en side der fungerer som indkøbskurv, hvor man kan se de indtastede mål og få en vejledende pris før tilbuddet bestilles. Udover dette skal applikationen også kunne oprette en ny ordre i databasen, samt ændre ordrens status, for at ordren bliver overført til Fog. Til sidst kræves det at webapplikationen kan udregne de passende materialer samt mængde på dem, så de passer til enhver given ordre. Vi vurderer derfor dette til at være 2 large opgaver, hvor den ene opgave indeholder

udarbejdelse af den grafiske brugergrænseflade og oprettelse af en ordre, og den anden opgave omhandler at få implementeret og integreret en materiale beregner.

US-4:

Som registreret kunde, skal jeg have mulighed for at vælge om jeg vil bestille et skur sammen med min carport, og hvor stort dette skur skal være.

Acceptkriterie: Under bestillingen af carporten, vil der nu være en checkbox med muligheden for at tilføje et skur. Her kan man vælge længde og bredde for skuret, hvor der igen vil være minimum- og maksimummål. Derudover vil der være standardmål valgt, og det er derfor ikke muligt at vælge andre mål end de givne i dropdown menuen.

Denne user story er knyttet til "bestil carport" funktionaliteten. Den dækker over, hvad der har at gøre med at kunne tilvælge et skur efter egne mål til carporten. Fuldendt implementering af denne user story kræver refaktorering af bestillingssiden, til at kunne håndtere om carporten der bestilles er med eller uden skur, tilføjelse af materialer der bruges til skuret og udvidelse af materiale beregneren til at kunne håndtere de nye materialer. Dette er for os at se en large opgave, da der er relativt meget som skal udbygges, hvilket samtidigt er beregningstungt. Alt imens at flere jsp sider skal tilpasses til at kunne vise varierende indhold.

US-5

Som registreret kunde, skal jeg have mulighed for at se ordrestatus på alle mine ordrer.

Acceptkriterie: Når man trykker på "ordrer" knappen i nav baren, vil man blive sendt videre til ordre siden, hvor man nu har mulighed for at se igangværende og gamle ordrer. Her kan man slette gamle ordrer som ikke er blevet betalt for. Man kan ikke redigere i ordrerne som kunde. Hvis man ikke er logget ind, vil der ikke være nogen "ordrer" knap i nav baren.

US-5 falder under "se ordrer" funktionaliteten og har at gøre med det GUI, kunden skal benytte for at følge op på sine ordrer. Implementeringen af denne user story kræver en enkelt side, at hente den data der skal vises og at kunne fjerne ordrerne fra databasen. Dette er derfor en medium opgave, da der skal udføres handlinger i alle lag af webapplikationen, samtidigt er det vigtigt at sørge for, at data som bliver fjernet er gyldigt i applikationen og databasen, samt at denne data er konsistent på tværs af hele systemet.

US-6:

Som registreret kunde, skal jeg have mulighed for at følge op på mine bestilte tilbud, så jeg kan betale eller afvise min bestilling.

Acceptkriterie: Når man trykker på "ordrer" knappen i nav baren, vil man blive sendt videre til ordre siden. Her kan man betale for de ordrer, som Fog har godkendt og givet tilbud på. De ordrer som stadig ikke er godkendt, kan man dog ikke betale for. Hvis man ikke er logget ind, vil der ikke være nogen "ordrer" knap i nav baren.

US-6 beskæftiger sig også med funktionaliteten tilknyttet “se ordrer”, som illustreret i use-case diagrammet ovenfor og er en viderebygning på US-5, som løser kundens handling “at skulle kunne betale”. Her kræves kun en mindre tilføjelse til det “ordrer-GUI”, som blev udviklet ved US-5, samt en implementering af at ordrens status ændres og holdes gyldig gennem hele systemet. Dette anser vi for at være en small opgave, da den ikke kræver meget mere end lidt videreudvikling på en i forvejen implementeret løsning.

US-7:

Som administrator, skal jeg have mulighed for at se nye ordrer, så jeg kan følge op på om deres bestilling giver mening, kontakte kunden, samt sende et tilbud, som inkluderer en pris, tilbage til kunden.

Acceptkriterie: Som administrator kan man trykke på “ordrer”-knappen i nav baren. Herefter bliver man sendt videre til ordre siden. Her kan man se alle kunders ordrer. Administratoren vil have 2 knapper ud for hver ordre. Her vil de have en “redigere” og en “godkend” knap. Efter de trykker på “redigere” knappen, bliver de sendt videre til “bestillingssiden”. hvor man kan redigere mål og pris.
“Godkend knappen”, vil godkende ordren og sende den tilbage til kunden.

I denne user story arbejder vi på noget funktionalitet tilknyttet administratoren; “administrer ordrer”, der skal implementere alle de arbejdsgange som Fog skal kunne udføre før tilbuddet videresendes til kunden. Implementation af dette kræver en side som kan vise alle afventende ordrer, samt ved forespørgsel, alle ordredetaljer relevante for at Fog effektivt kan udføre deres arbejde. Udover det vil der blive stillet store krav til at al data forbliver valid og konsistent på tværs af database og webapplikation, da administratoren får mulighed for at ændre de fleste data tilknyttet en ordre, hvor at ændring af data vil føre til en genberegning af hele ordren. Dette vurderes at være en large opgave da implementationen berører alle lag systemet, stiller høje krav til dataenes validitet og alt imens det skal være relativt overskueligt for Fogs administrator at arbejde med.

US-8:

Som administrator, skal jeg have mulighed for at tilføje og fjerne materialer fra databasen, samt opdatere priser og længder.

Acceptkriterie: Når man er logget ind som administrator og trykker på “admin” dropdown menuen i nav baren, og derefter trykker på “materiale side” knappen, vil man blive sendt videre til materialesiden. Her vil administratoren nu have mulighed for at tilføje materialer til databasen. Her vil der også være mulighed for at opdatere priser og længde på eksisterende

materialer. Disse ændringer vil så opdateres i databasen.
Hvis man ikke er logget ind som admin, vil man ikke have mulighed for at se “admin” dropdown menuen i nav baren.

US-8 omhandler den anden funktionalitet på webapplikationen, som er tilknyttet Fogs administrator, “administrer varer i databasen”. Her handler det om at imødekomme Fogs ønske om at kunne tilføje nye materialer, som ikke var muligt i deres gamle løsning, samt redigere eksisterende materialer. Implementationen af dette kræver en kompleks og gennemarbejdet jsp side inkl. servlet, og funktioner til at opdatere eller indsætte data i databasen, samt opdatering af relevante Java objekter. Dette er derfor en large opgave, da finpudsning, i forhold til at siden skal være fuldt funktionel på trods af mange forskellige inputs, og at data håndteringen skal være konsistent på tværs af hele systemet, kræver noget arbejde.

US-9:

Som registreret kunde, skal jeg have mulighed for at få en ordrekvittering og en stykliste efter et tilbud er bekræftet og betalt.

Acceptkriterie: Når man er logget ind som bruger og trykker sig ind på “ordrer” knappen i navbar, og man derefter har trykket “betal” for en ordre, så vil man modtage en ordre kvittering på samme side. Derudover vil ordren ændre status til “betalt”. Når kunden trykker på “Stykliste” knappen, vil man så have mulighed for at se styklisten, som er blevet genereret for carporten på samme side.

Denne user story beskæftiger sig med funktionen “betal” for kunden og omhandler at kunden modtager ordrekvittering og stykliste, som beskrevet i både as-is og to-be-aktivitets diagrammerne i forrige afsnit. For at lave en fuld funktionel løsning til dette kræves der en side som der bliver henvist til efter kunden har betalt, samt indlæsning af ordre data herunder stykliste. Dette anser vi for at være en small task, da al nødvendig data findes i webapplikationen i forvejen og ikke skal manipuleres yderligere før fremvisning.

US-10:

Som administrator, skal jeg have mulighed for at generere en 3D model der matcher en given ordre, sådan at jeg har mulighed for at 3D printe en prototype af ordrens indhold.

Acceptkriterie: Når man har modtaget en betalt ordre, vil admin nu have muligheden for at lave en 3D printet model af carporten med kundens givne mål. Dette vil ske i en separat applikation.

US-10 er den eneste user story som har at gøre med den funktionalitet, der ikke skal køre på webapplikationen. For at den skal implementeres kræver det at vi udvikler en separat applikation som kan tage imod data fra vores webapplikation og generere 3D modeller ud fra dette. På trods af at løsningen vil kræve at administratoren bliver trænet i brug af systemet (da implementationen ikke vil være fuld automatisk), vurderer vi at dette er en large opgave, da det ikke er noget vi har så meget erfaring med og at opgaven derfor vil tage lang tid at implementere.

US-11:

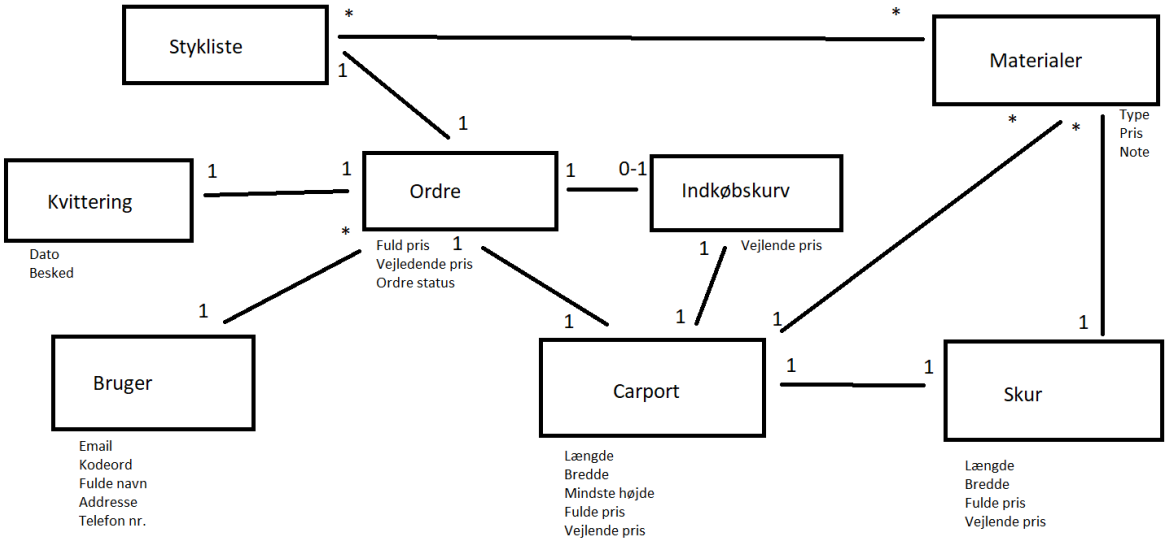
Som administrator, skal jeg kunne se alle ordrer i systemet, så jeg har mulighed for at følge op på mine kunders ordrer.

Acceptkriterie: Når man er logget ind som administrator og trykker på “ordrer” knappen i navbaren, vil man få et overblik over alle ordrer i butikken. Ordrerne vil være sorteret i betalte, godkendte og bestilte ordrer.
Som kunde vil man ikke have adgang til andre kunders ordrer. Hvis man ikke er logget ind i systemet, vil man ikke have “ordrer” knappen i nav baren.

Denne user story har at gøre med den resterende funktionalitet, som ikke er dækket af US-7, når vi taler om at “administrere ordrer” som vist i use-case diagrammet. Det indebærer at bygge en side hvor administratoren har overblik over alle ordrer afgivet af alle kunder, uanset ordre status (her medregnes ikke de ordrer som ligger i kundernes indkøbskurve) og at indhente de nødvendige data for at kunne vise ordrerne. Vi vurderer at dette er en small opgave, da der ikke skal ændres noget data.

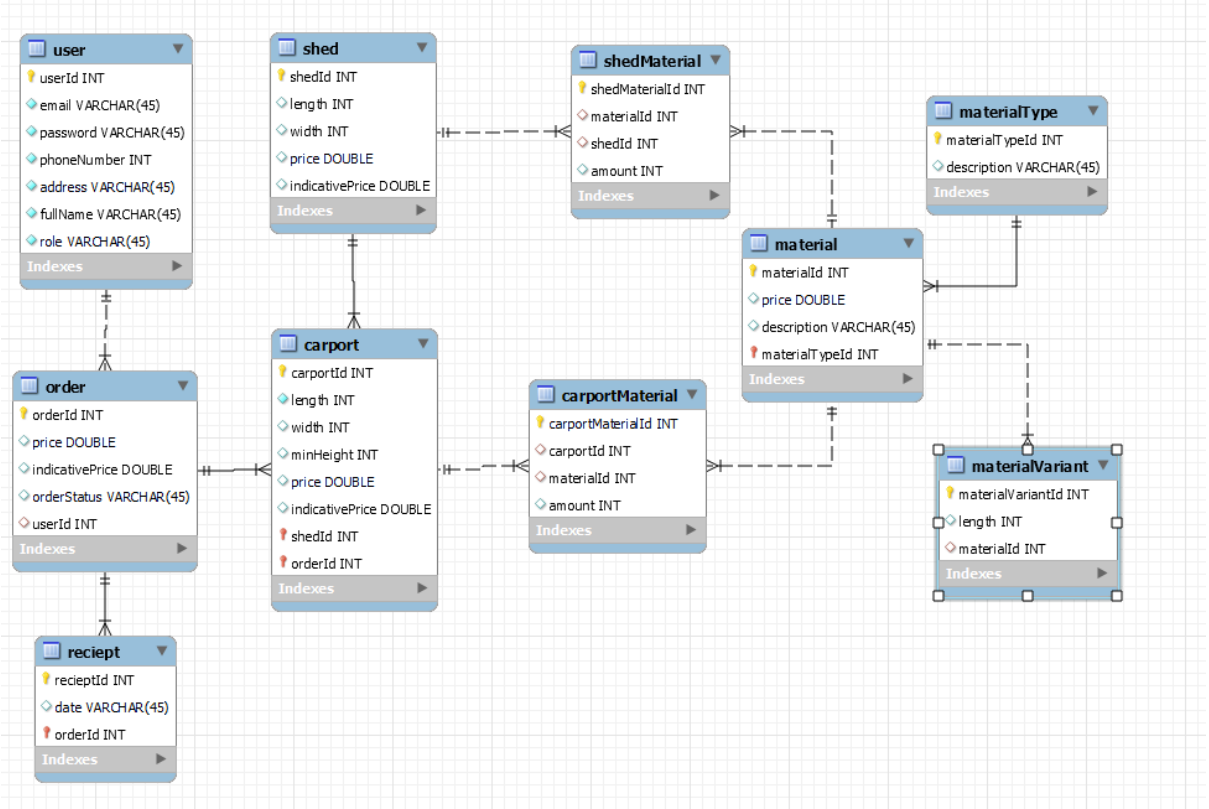
Modeller, diagrammer og mockups:

Domæne model:

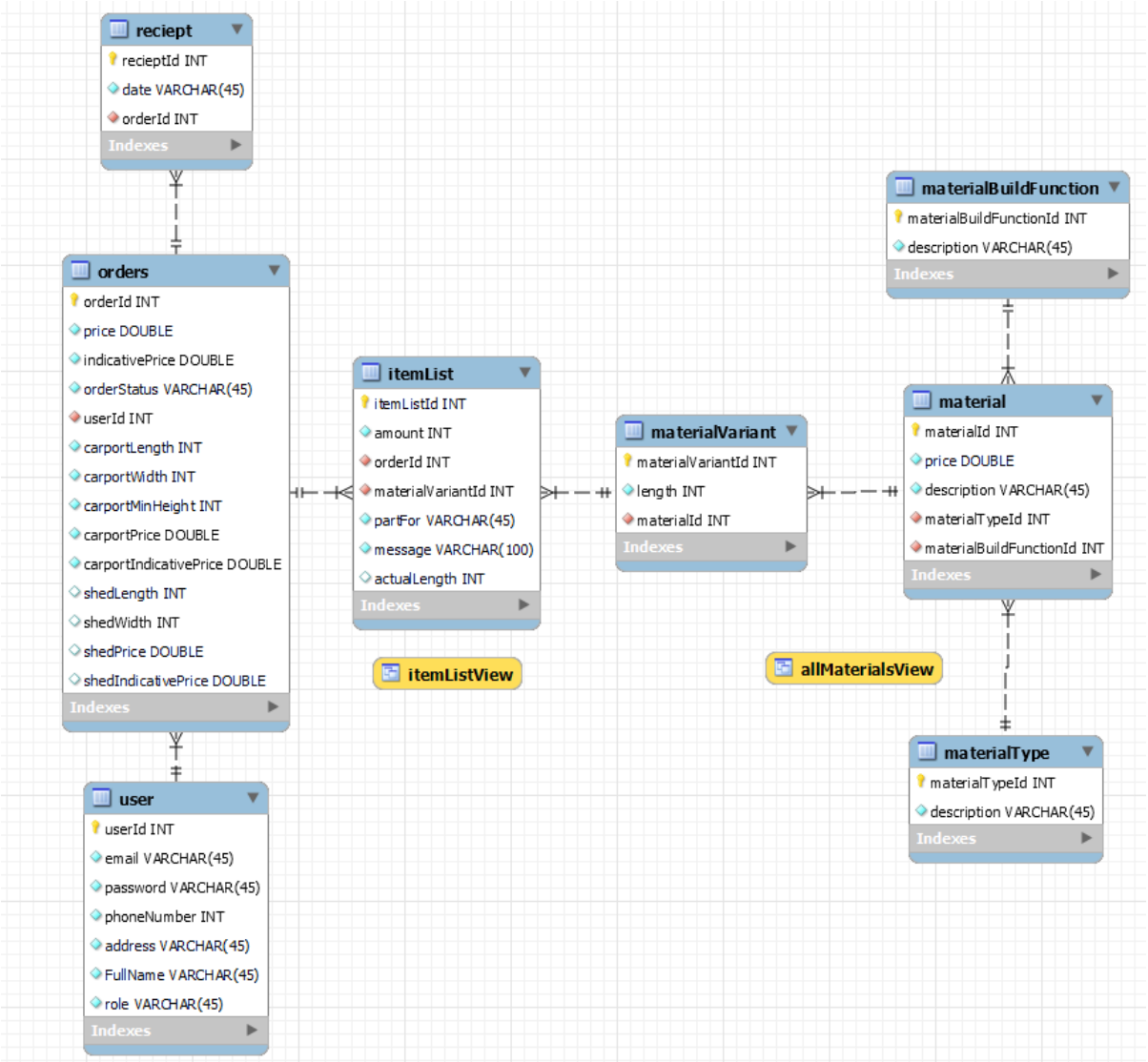


Domænemodellen ovenfor repræsenterer vores første tanker vedrørende, hvad der var essentielt for applikationens indhold for at opnå fuld funktionalitet. Dette er udarbejdet på baggrund af et kundeinterview med Martin fra Fog samt opfølgende kundemøder/vejledermøder. Modellen er lavet med henblik på at skabe et fælles grundlag for hvad der skal repræsenteres. Derudover defineres der, hvad Fog har brug for, for at kunne arbejde på webapplikationen. Et par interessante forhold kan være, at materialerne er kædet sammen med både styklisten, carporten og skuret. Dette gøres for både at have en samlet oversigt over materialerne tilhørende hele ordren, mens der samtidigt holdes styr på hvilke materialer der hører til hvilket produkt, for at kunne give bedre notater til kunden ift. Materialets funktion. Udover det er det også værd at bide mærke i, at indkøbskurven i virkeligheden repræsenterer en ordre, som ikke er blevet afsendt endnu, derfor eksisterer indkøbskurven kun indtil kunden fremsender en anmodning om at få tilsendt et tilbud.

EER-Diagram:



Dette var vores første bud på hvordan databasen kunne sættes op, fokus her var at imødekomme alle Fogs krav alt imens at vi forsøgte at holde tingene så adskilte som muligt. Vi blev dog hurtigt opmærksomme på at denne struktur havde en række svagheder, som ville gøre databasen meget svær at arbejde med. Der var f.eks. alt for høj sammenhæng mellem tabellerne, forstået på den måde at vi ville være meget låste i forhold til vores CRUD operationer, da adskillige tabeller indeholder fremmednøgler fra flere andre tabeller. Dette ville i praksis have været svært at arbejde med i forhold til indsættelse af ny data, eksempelvis skulle der ved indsættelse af en ny ordre med en carport inkl. skur skrives data ned i 5 forskellige tabeller. Vi endte derfor med at omstrukturere databasen til et mere lineært design, som vist herunder.



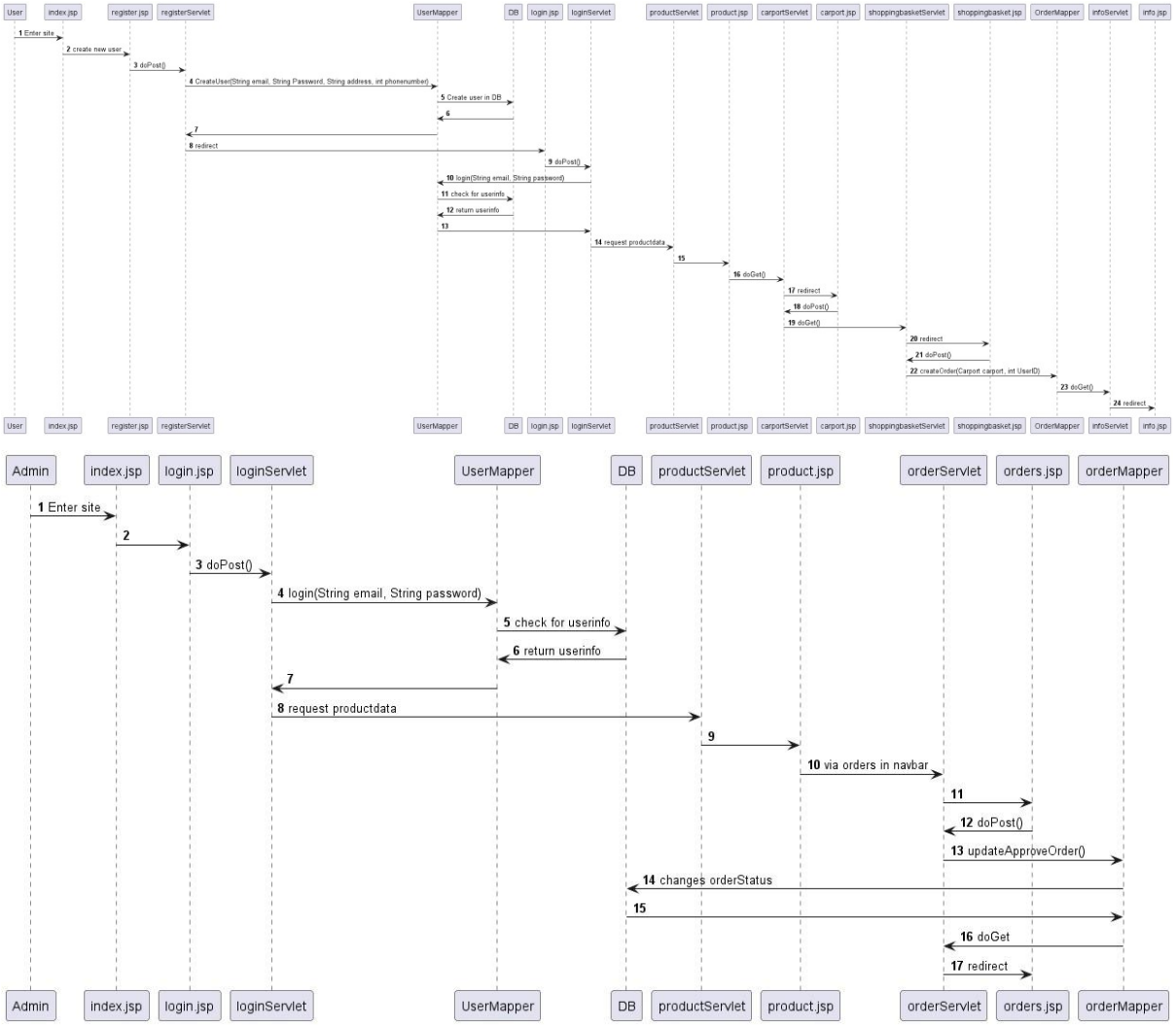
Denne struktur er markant lettere at arbejde med, da tabellerne ikke er koblet lige så tæt sammen, og at vi er sluppet af med alle 1-1 relationerne mellem tabellerne. Alle tabellerne er på 3. normalform med undtagelse af user-tabellen, som grundet adresse kolonnen ikke opfylder kravet fra 1. normalform om at der kun må være en værdi i et felt, da feltet både indeholder vejnavn + nr, postnummer, bynavn og måske endda land. Vi har taget en bevidst beslutning om at lave det sådan, da adressefeltet aldrig skal manipuleres i webapplikationen og kun skal vises som kontaktoplysninger til administratoren. Skulle vi have lavet det til 3. normalform som de andre tabeller ville det gøre det vanskeligere for nye kunder hos Fog at oprette en konto på webapplikationen, da de ville skulle indtaste flere oplysninger, som kun ville accepteres i databasen hvis de fandtes i databasen i forvejen. Eksempelvis ville en kunde med adresse i en by hvis postnummer ikke fandtes i databasen i forvejen ikke kunne oprette sig som kunde, hvilket ville være uhensigtsmæssigt for Fog. Udover dette ville det med tanke på performance ikke være hensigtsmæssigt at skulle joine adskillige tabeller sammen, blot for at samle dataen igen i Java.
At næsten alle vores tabeller er på 3. normalform har medført at vi har brug for at lave en række joins. Dette blev vi nødsaget til for at få instantieret vores objekter med korrekt data. For at løse dette har vi lavet 2 views; allMaterialsView, som joiner de 4 “material-tabeller” sammen og fjerner redundante kolonner, og itemListView, som joiner de 4 “material-tabeller

og itemList tabellen. AllMaterialsView'et bruges til at instantiere objekter af alle materialerne, hvor itemListView'et bruges til at generere styklisterne tilknyttet ordrerne.
Alle vores tabeller gør brug af en autogenereret primær nøgle, hvilket vi har gjort fordi at alle de entiteter vi arbejder med er unikke og derfor skal deres primærnøgler også være unikke. Udover primær nøglerne er "email" kolonnen den eneste kolonne som er sat til at være unik. Dette har vi gjort for, at kunderne ikke skal have lov at oprette flere konti med samme email. Udover dette er alle kolonner sat til ikke at være null, udover målene på skuret i orders tabellen, som får værdien 0 som standard hvis de ikke udfyldes. Dette skyldes naturligvis at man har mulighed for at bestille en carport uden tilknyttet skur. Grunden til at alle andre værdier ikke må være null, er for at sikre os, at vi ikke gemmer ufuldstændige data.
Da vi havde det forhold, at der hører mange materialer til en ordre og at et specifikt materiale kan indgå i mange ordrer, har vi været nødt til at indskyde itemList tabellen som en mellemtabel, for at kunne holde styr på denne forbindelse. Det betyder nemlig at vi kan indskyde så mange materialer, der er behov for til hver ordre, uden at kompromittere dataen i andre tabeller. Hver række i itemList tabellen repræsenterer dermed en ordrelinje tilhørende en ordre.

Klassediagram:

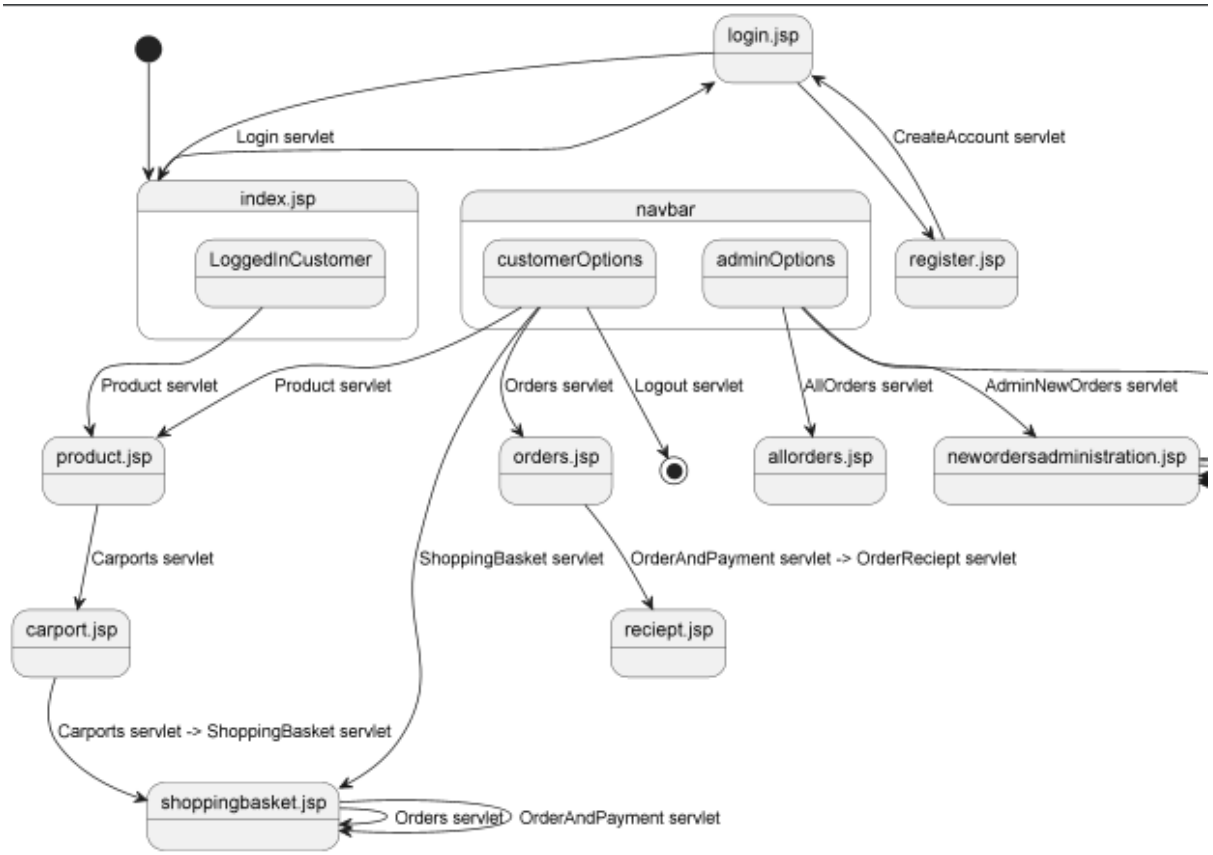
Efter at vi, ved hjælp af domænemodellen og EER-diagrammerne havde fået en bedre forståelse for domænet og kravene som helhed, har vi for vores eget overbliks skyld (eller programmører som engang skal overtage projektets skyld) udarbejdet et klassediagram der repræsenterer den implementerede løsning på webapplikationen. På bilag 1.1 og 1.2 fremgår udgangspunktet på vores klassediagram, som stammer fra før vi påbegyndte vores implementering. Efter vi påbegyndte implementeringen fandt vi dog hurtigt ud af, at vores udgangspunkt ikke var fyldestgørende nok i forhold til at løse alle funktionelle krav, og vores klassediagram blev derfor udvidet, som vist i bilag 2.1 (Bemærk at alle de forskellige materialetyper der nedarver fra abstrakt klassen Material ikke fremgår af bilaget, da den eneste forskel på dem er at prisen udregnes afhængigt af typen).
I det store hele har vi beholdt samme struktur som der var planlagt oprindeligt, men nu er der kommet en dimension på som omhandler om at generere styklister til ordrerne.
På bilag 2.2 fremgår det at vores mapper klasser er forblevet de samme, men at metoderne er rettet til så diagrammet kun viser de metoder vi fik brug for.

Sekvensdiagram:

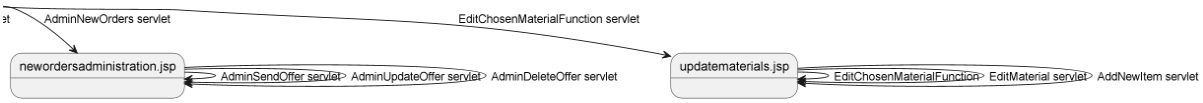


Vi udarbejdede de ovenstående sekvensdiagrammer tidligt i vores arbejdsproces, med henblik på at danne os et overblik over sammenhængen mellem vores jsp sider, servlets og database. Det første diagram viser kundens rejse fra de besøger webapplikationen til de har bestilt et tilbud på en carport. Det andet diagram viser adminens vej gennem systemet fra log ind til at se ordre oversigten over alle ordrerne i systemet. Det er vigtigt at påpege at disse sekvensdiagrammer ikke afspejler vores færdige produkt, men blev brugt som et trin i vores designfase til at lægge en plan for vores vej gennem de forskellige lag af vores MVC arkitektur, samt have en ide om hvordan henholdsvis kunden og adminen skulle navigere fra side til side. Det hjalp os til at få et overblik over, hvordan vi skulle bygge webapplikationen op fra bunden, men har også haft den betydning at vores sekvensdiagrammer ikke kan bruges til at holde sig opdateret om strukturen i projektet. Her henvises i stedet til vores klassediagrammer og navigationsdiagram, som blev holdt up-to-date løbende gennem projektets levetid.

Navigationsdiagram:



Følgende ses i forlængelse af ovenstående.



Navigations Diagrammet tager udgangspunkt i at brugeren starter ved den udfyldte sorte prik og ved første besøg på siden rammer index.jsp. På vores webapplikation er der kun 3 sider (hvis man ikke medregner error siden), som det ikke kræver en servlet at komme til - index.jsp, login.jsp og register.jsp. Alle andre sider kan kun tilgås efter brugeren er logget på og derigennem har fået nye navbar links til rådighed. Alle servlets og sider som er illustreret i diagrammet og befinder sig nedenunder index.jsp - navbar - niveauet er dermed først tilgængelige efter at brugeren er logget ind. Fra de links som ligger i navbaren har man dog flere muligheder hvis man er logget på som admin bruger, det vil sige at admin brugeren har adgang til alle de samme sider som user brugeren har, men også har mulighed for at komme til allorders.jsp, newordersadministration.jsp og updatematerials.jsp, via en dropdown, som ellers er utilgængelig for user brugeren.
Navigations Diagrammet er lavet til os programmører, da vi har haft brug for et værktøj til at holde styr på strukturen på vores site, da antallet af sider og servlets meget hurtigt bliver uoverskueligt. Vi har dog i udarbejdelsen af diagrammet opdaget hvor vigtigt det er at kalde sine sider og servlets for nogle gode navne, eftersom det selv ved hjælp af dette diagram kan være svært at gennemskue hvad det er for aktioner der bringer en fra side til side.

Figma mockups:

I vores arbejde med designet af vores sider har vi lavet 2 figma mockups, som kan ses i bilag 3.1, der viser vores forside, og 3.2, der viser bestillingssiden. Da vi udarbejdede designet fandt vi det ikke nødvendigt at lave en mockup for hver side da vi gerne ville skabe en ensartet oplevelse for kunden hele vejen gennem brugsoplevelsen af applikationen. Vi har taget en række design beslutninger som vi syntes spiller godt sammen, og som samtidigt var realistiske for os at få implementeret.

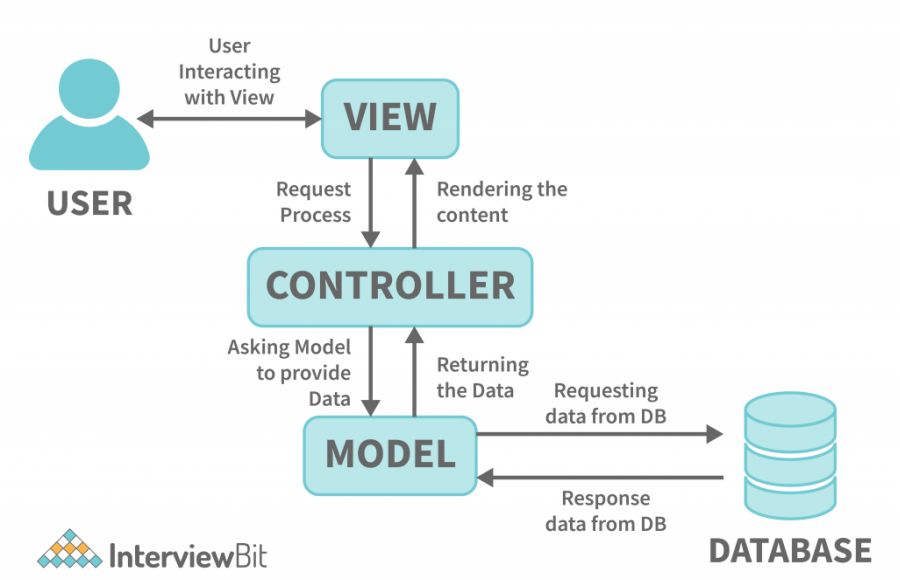
- KISS (keep it simple stupid)
- Loven om lighed (gestalt lov som den fremgår af https://www.steffengrafisk.dk/archives/83745)
- Loven om lukkethed (gestalt lov som den fremgår af https://www.steffengrafisk.dk/archives/83745)
- Hick's Law (Laws of UX som den fremgår af https://lawsofux.com/hicks-law/)

Vi synes disse principper hænger godt sammen, da de alle handler om at holde tingene simple samt ikke at overvælde brugeren. På vores mockups har vi derfor forsøgt at illustrere at alle siderne skal have den samme navbar (dog med varierende indhold), at der ikke skal være alt for meget på hver enkelt side og at elementerne på siden så vidt muligt skal kapsles ind i sektioner som tydeligt markerer at de passer sammen. I forhold til loven om lighed var det vigtigt for os at der på tværs af applikationen var lighed mellem de elementer som er ens eller hører sammen, eksempelvis ligner loginsiden og registreringssiden hinanden meget og billederne fra produktsiden går igen hele vejen gennem bestillingssiden til indkøbskurven. Det var også vigtigt for os ikke at give den besøgende på sitet for mange beslutninger at tage på hver side, hvilket er en af årsagerne til at vores navigering er blevet relativt lineær. Selvom vores primære fokus lå i funktionalitet, at gøre applikationen så simpel og ligetil som muligt, valgte vi at style vores sider ved hjælp af bootstrap, for også at gøre den behagelig og indbydende.

Valg af arkitektur:

MVC:

Vores kode tager udgangspunkt i en fælles startkode, hvor arkitekturen lægger op til at man skal benytte en MVC (Model-View-Controller) opsætning. Dvs. at brugeren via sitet ikke har direkte adgang til backenden eller den vej igennem til databasen, før deres requests har bevæget sig gennem en servlet hvor vi kan kontrollere hvad det er som skal bevæge sig videre i systemet.



En bruger modtager sin information via et view, hvilket er en jsp side. Så beder brugeren, jsp siden om at håndtere en given data i form af inputs. Jsp siden vil så videresende de inputs til en Servlet (Controller) som er programmeret til at håndtere disse data på en given måde. Som regel ved at hive fat i en Facade/Mapper klasse (Model) som så laver den/de ønskede handling(er) i Databasen.

Ved at bruge MVC strukturen kan man forhindre brugeren i at få direkte adgang til backend og database og dermed mindske uønskede eller direkte skadelige handlinger på databasen.

Facade pattern:

I samme forbindelse benytter vi et facade pattern, hvor vi “skjuler” komplekse metoder bag simple facade metoder. Det gør, ud over at gøre koden nemmere at tygge, også at vi nemmere kan rette i de komplekse metoder, uden at det nødvendigvis påvirker resten af koden. Derudover giver det også en mulighed for at lægge et ekstra lag af sikkerhed og validering, hvis der er behov for dette. I vores tilfælde benytter vi ikke validering i vores Facadelag, da vi som udgangspunkt vil lægge valideringen så tidligt i forløbet som muligt. Det beskriver jeg mere detaljeret om under særlige forhold, validering.

Dependency:

Dependencys betydning ligger i navnet, det er hvor dele af koden (klasser, og/eller metoder) er afhængig af en anden del af koden, for at kunne virke. At bruge DI, *dependency Injections*, betyder at man laver en løs opkobling f.eks. metoderne imellem. Det gør vi bl.a. når vi sætter connection pool (som er en 3. part kode, implementeret fra startkoden) ind i samtlige mappermetoder (dvs. når vi laver dyk i databasen). ud over at vi slipper for at *hard-code* opkoblingen til databasen ind i samtlige metoder, og får en ikke særlig pæn kode, så giver det os nogle forskellige fordele. For det første så gør det koden nemmere at teste, netop fordi vi så kun behøver at teste én metode af gangen, og det gør testen mere gangbar og læselig.
Men det gør det også nemmere at udvide applikationen, uden at det har indflydelse på resten af koden.
Men hvad der lige så vigtigt, så betyder det også at man direkte kan skifte dele af eksisterende kode ud, med noget nyt, og muligvis smartere.

Særlige forhold:

Scopes:

I denne opgave benytter vi alle tre former for scopes (Request, Session samt Application) til at gemme eller overføre data, fra side til side.

Vi er som udgangspunkt gået med ideen, at vi skal øge performance i vores applikation, ved at lave så få unødige nedslag i databasen som muligt, dog uden at gå på kompromis med sikkerheden.

Det betyder i praksis, at allerede ved login, hvor brugeren bliver ført fra login siden, til login servlet (her bliver login informationerne håndteret). Der bliver brugeren sammenlignet med de brugerinformationer, der ligger i databasen, og ved et match bliver alle informationer hentet ud og gemt på session scope. Således at brugerens informationer vil blive gemt, så længe vedkommende er logget ind (eller i 30 min hvis brugeren lukker applikationen uden at logge ud). Første gang der logges på applikationen, vil der også laves en række nedslag i databasen, hvor alle materialedata hentes ud og gemmes på applikation scopet. Således at materialerne er tilgængelige gennem hele applikationens levetid og vi undgår derved at skulle hente alle materialer ud af databasen hver gang en ordre skal genereres. Vi bruger samme princip i en del af vores servlets, f.eks. hvis admin skal have vist alle ordrerne i systemet, vil ordrerne blive hentet ud af databasen og gemt på session scopet, men dette gøres kun hvis de ikke allerede er blevet hentet en gang før. Dette gøres for at undgå at hente den samme data adskillige gange.

I forbindelse med request scopet, så bliver det primært benyttet, hvis der er informationer der kun skal bruges fra den ene side til den næste. f.eks. når kunden skal vælge mellem en carport med skur, eller uden skur. her vel der bliver gemt en værdi (0 eller 1) alt afhængig af

om kunden ønsker uden skur, eller med. Den information bliver så videreført til carport siden, og hvis et skur er fravalgt, så vil alle valgmuligheder der involverer skuret, ikke være synligt.

Og modsat, hvis et skur er tilvalgt, så vil der komme nogle dropdown menuer, hvor kunden kan vælge bredde samt længde for skuret.

Så i korte træk bruger vi:

- Request scope, hvis vi skal bruge noget information på den næste jsp side, men som ikke er vigtigt i forhold til resten af applikationen.
- Session scope, hvis vi skal bruge information på kryds af applikationen, men som kun er vigtigt i forhold til den aktuelle bruger.
- Applikation scope, hvis vi skal bruge noget på kryds af applikationen, som ikke er baseret på den aktuelle bruger, og det ikke har betydning for applikationens eller brugerens sikkerhed.

Exceptions:

I forhold til exceptions har vi brugt den samme fremgangsmåde gennem hele applikationen. Vi har taget udgangspunkt i "*startkoden*", hvor exceptions bliver kastet fra mapperen, med en fejlbesked, hele vejen op til servletten, hvor brugeren vil blive dirigeret over til en fejlside (error.jsp), her vil fejlbeskeden så blive fremvist.
Den hyppigst brugte, og den vi har mindst kontrol over er Databaseexceptions. Da denne opstår hvis vi af ukendte årsager mister forbindelsen til databasen.
I forbindelse med MaterialCalculator klassen, har vi lavet vores egen Exception, kaldet "*NoMaterialFoundException*" som håndterer exceptionen på samme måde som en DatabaseException. Men er mere beskrivende i forhold til problemet, altså at den ikke kan finde et materiale der opfylder de kriterier der bliver sat ved beregningen af materialer til carporten.

Javadoc:

Vi har dokumenteret løbende i Javadoc, som primær dokumentation. Dvs. at i samtlige Java klasser, og servlets. er der en kort beskrivelse i starten af klassen, som giver et hurtigt overblik over klassens funktion. Herefter er der en beskrivelse før hver metode, som giver en beskrivelse af metoden, hvilke parametre der bliver brugt, hvilke Exceptions der bliver kastet samt hvilken udvikler der har skrevet metoden. Dette gør vi for at sikre en ensartet kodestandard og for at give hinanden og andre et bedre overblik over vores kode.

Validering (Brugerinputs):

Vi har taget nogle af brugerens input fejl på forskud, og forsøgt at mindske fejl ved bl.a. at pre-definere hvilke inputs brugeren kan komme med helt ude i frontend. Dette gør vi bl.a. når kunden skal vælge mål på sin Carport og/eller skur. Så kan man "kun" vælge mellem en minimum- samt maximumsværdi med et 30cm interval, i form af en dropdown menu. Det

samme gælder for admin på materialesiden. Hvor brugeren i nogle tilfælde “kun” kan vælge imellem bestemte grupper. Heriblandt materialetype eller funktion. I de tilfælde hvor brugeren skal indtaste inputs enten i form af tekst eller tal, har vi efter bedste evne forsøgt at guide brugeren direkte på jsp siden. f.eks i feltet hvor admin kan ændre priser for materialer, kan der KUN indtastes tal, med en trinstigning på 0.1 krone og der er som parameter indført at man ikke kan taste et negativt tal. i tilfælde af input fejl bliver disse håndteret via exceptions. Men i enkelte tilfælde. f.eks når admin skal ændre på beskrivelsen af et materiale, kan denne beskrivelse være hvad som helst. Her må vi gå ud fra, at vedkommende ved hvad der skal skrives, derfor har vi ikke lavet nogen check eller kontrol, så vi ikke ligger en utilsigtet hindring i vejen for brugeren.

Validering (Login):

Ved login skal brugeren indtaste 2 inputs; en email og et password. Disse har brugeren selv bestemt via registreringssiden. Igen holder vi valideringen ude i frontend, ved at definere hvilke input bokse der skal bruges. Fx så kræver en input boks med en type email, at der f.eks. er et @. Efter brugeren har indtastet disse, vil et nedslag i databasen sammenligne med samtlige brugere der er registreret. Ved et match vil al brugerdata blive hevet ud, og et user objekt vil blive oprettet og gemt på session scopet.
Hvis enten email eller kode ikke stemmer overens med hvad der står i databasen, vil brugeren modtage en fejlbesked, hvor de oplyses, at enten email eller kode er forkert indtastet.
Hvis fejlen opstår inden kontakten til databasen bliver etableret, så bliver der kastet en Databaseexception, og brugeren får af vide at der skete en fejl i databasen i stedet.

Brugertyper:

Vi har valgt at bruge 2 brugertyper i projektet (user og admin). Ved et større projekt, eller ved flere funktioner, kunne man overveje om der skulle laves flere, fx sales, som har adgang til salgsfunktioner. Marketing, hvis der skulle reklameres på forsiden. Den primære årsag til dette er at mindske adgang til dele af systemet, som er irrelevant for brugeren. På den måde vil man kunne mindske uhensigtsmæssig brug af systemet.

Som systemet er sat op i dag, er det kun admin brugeren som har mulighed for at lave ændringer i databasen.

Materiale beregner (stykliste):

Vores materialeberegner er en smule kompleks. Den er sat op, så den selv beregner hvor mange af hvert materiale der skal bruges og hvilken længde af et givent materiale der skal benyttes for at give mindst muligt spild, ud fra de krav der er i ordren. Vi har medtaget en lille bid, eksemplet herunder beregner hvor mange stolper der skal benyttes.

```
public Set<ItemListMaterial> calculatePosts(List<Post> allPosts) throws NoMaterialFoundException {
    Set<ItemListMaterial> posts = new HashSet<>();
    Post chosenPost = null;
    int numberOfPosts = 4;
    for (int i = 0; i < allPosts.size(); i++) {
        if (allPosts.get(i).length >= minHeight + 90 + 10) {
            chosenPost = allPosts.get(i);
            break;
        }
    }
    if (length - 130 > 310 && length - 130 < 620) {
        numberOfPosts += 2;
    } else if (length - 130 > 620) {
        numberOfPosts += 4;
    }
```

Udrag fra calculatePosts metoden i MaterialCalculator.java - linje 52 - 66

I det udvalgte kode afsnit sætter vi et minimum af stolper til 4, og så vælger vi de stolper som lever op til minimumshøjden, for carporten + 90 cm som stolpen skal graves ned, og ekstra 10 cm for at få hældning på taget.
Herefter vil der, alt afhængig af hvor lang carporten er, blive tilføjet yderligere 2, eller 4 stolper, baseret på ikke at overskride den maksimale længde, der må være mellem hver stolpe for at byggeriet er stabilt.
De 130 cm der bliver trukket fra, svarer til det udhæng (30 cm på bagsiden og 100 cm på forsiden) der skal være på det færdige produkt.
Materiale beregneren tager udgangspunkt i den byggevejledning, der er udleveret af Fog, som ses i bilag 4.1 og 4.2. Vi har taget udgangspunkt i at alle de maksimale mål fra disse tegninger er det maksimale tilladte mål, f.eks. må der maksimalt være 310 cm mellem stolperne, 35 cm tagudhæng på hver side eller maksimalt 55 cm mellem spærene.
Materiale beregneren implementerer kun en løsning på at finde længder der passer til kravene, eftersom vi i vores system ikke tager højde for vægt eller på anden måde om produkterne er konstrueret efter alle fysikkens love. Vi tager derfor slet ikke højde for højde og bredde i vores implementering, beregneren vil altså ikke være i stand til at vælge den mest optimale stolpe hvis den skal vælge mellem en 97x97mm. stolpe eller 105x105mm. stolpe i samme højde.

Udvalgte kodeeksempler:

Materiale beregner ud fra spild:

```
waste = (wasteWidth * (numberOfRoofsNeeded / numberOfRoofsPerRow)) + wasteLength;
if (waste < lowestWaste || lowestWaste == -1) {
    lowestWaste = waste;
    chosenRoof = r;
    actualNumberOfRoofsNeeded = numberOfRoofsNeeded;
    actualNumberOfRowsNeeded = numberOfRowsNeeded;
}
```

Udrag fra calculateRoofs metoden i MaterialCalculator.java - linje 208 - 214

Vores materialeberegner udregner hvilket materiale, af en given materialetype, der giver det mindste spild. Foroven ses et eksempel på dette fra den metode som udvælger den mest optimale tagplade og udregner hvor mange af den specifikke tagplade der skal bruges. Forud for eksemplet løbes en liste af alle tilgængelige tagplader igennem, og for hver af dem udregner den så hvor mange tagplader der skal bruges for at dække hele taget, hvor mange rækker tagplader der kommer til at være og hvor meget spild der vil være henholdsvis på længden og bredden. Efter beregningerne er udført i hver iteration vil der blive tjekket om det samlede spild for tagpladen er mindre end det forhenværende laveste spild, og hvis dette er true, vil tagpladen samt de værdier resterende værdier blive gemt i variablene der ses ovenfor. Til sidst vil disse variable blive brugt til at instantiere et ItemListMaterial objekt, som er den datatype, styklisten består af.

Javascript på produktside:

```
<script>
    function submitFormNoShed() {
        document.getElementById("formNoShed").submit();
    }

    function submitFormShed() {
        document.getElementById("formShed").submit();
    }
</script>
```

Udrag fra product.jsp - linje 45 - 53

På produktsiden har vi valgt at benytte en ganske lille smule Javascript til at gøre kundens oplevelse af siden bedre. Ideen er at på praktisk taget alle andre webshops har man mulighed for at klikke hvor man har lyst til inden for et produkts felt for at vælge det. For at genskabe den effekt var vi nødt til at gøre hele cards klikbare, og den letteste måde at gøre det på var ved ovenstående løsning, som submitter den form som er tilknyttet cardet, uanset

hvor der bliver klikket inden for cardets ramme. Alternativt skulle vi have gjort alle elementer af cardet til links eller knapper, som alle sammen fører det samme sted hen. Dette virkede dog uhensigtsmæssigt at skulle udvikle videre på, og derfor endte vi med at bruge ovenstående løsning. Selve funktionen er egentlig meget simpel, når der bliver klikket på et card kaldes funktionen med ved det navn, som står anført i det tagget i det cardets tag. Derefter vil funktionerne submitte den data der ligger i formen til den servlet form tagget henviser til.

Data konsistens mellem Java og DB:

```
try {
    if(OrderFacade.deleteOrder(orderID, connectionPool)){
        List<Order> newOrders = (List<Order>) session.getAttribute( s: "newOrders");
        Order chosenOrder = null;
        for(Order o: newOrders){
            if(o.getOrderID() == orderID){
                chosenOrder = o;
            }
        }
        newOrders.remove(chosenOrder);
    }
} catch (DatabaseException e){
```

Udrag fra doPost metoden i AdminDeleteOffer.java - linje 52 - 63

Som tidligere nævnt i afsnittet om særlige forhold: Scopes; har det været et mål for os ikke at lave flere nedslag i databasen end højst nødvendigt. Dette betyder selvfølgelig at når vi ikke henter de nyeste data fra databasen hele tiden, så skal vi sørge for at hver gang der laves ændringer i databasen, skal de samme ændringer udføres på alle de forskellige scopes hvor det er relevant. I eksemplet ovenfor forsøger admin at slette et tilbud, som ikke er blevet sendt til kunden endnu. Hvis det lykkes at fjerne ordren i databasen, så leder vi ordrelisten fra session scopet igennem, finder ordren og fjerner den derfra også, sådan at ordren ikke længere vil figurere på siden, efter den er blevet fjernet fra databasen.
Strukturen som vises ovenfor, hvor vi først kalder en metode som udfører en handling i databasen og derefter retter data på relevante scopes, går igen gennem hele vores applikation, men udføres altid kun i vores controllere.

Opdater Materiale:

```
try {
    if (materialfunction == 1) {
        session.setAttribute( s: "editmateriallist", postList);
        if (chosenmaterials == postList) {
            chosenmaterialId = Integer.parseInt(request.getParameter( s: "materialdescription"));
            for (Material m : chosenmaterials) {
                if (m.getMaterialVariantID() == chosenmaterialId) {
                    chosenMaterial = m;
                }
            }
        } else {
            chosenmaterialId = -1;
        }
```

control/EditChosenMaterialFunction.java linje 73 - 85

I sig selv er denne kode bid ikke yderligere spændende. Men det er samspillet mellem Session scope, og request scope der gør koden noget særligt.
Den modtager et materialfunction id, fra en formaction på jsp siden. via en knap.
hvis dens værdi er lig 1, så vil postList (som er sat tidligere i koden) blive loadet ind på sessionscopet og gemt til senere brug. efterfølgende bliver der lavet et check, om variablen chosenmaterials har været valgt før, og derfor er det samme, som postList. Hvis dette er tilfældet, vil den tage imod det id, den modtager for næste dropdrown (materialdescription). Til sidst vil den løbe hele chosenmaterials listen igennem, holdt op imod Material klassen, og så længe VariantId'et er det samme som chosenmaterialId, så vil den sætte chosenMaterial til at have denne værdi.
senere i koden vil både chosenmaterial og chosenmaterialId blive sat på request scopet. Hvis chosenmaterials ikke matcher postList, som nævnt tidligere (eller nogen af de andre lister, som har deres egne metoder. vil værdien blive sat til -1, og så siger den at man skal vælge en kategori på jsp siden, samt at dropdown boksen er gjort inaktiv.

Status på implementering:

Overordnet set lever produktet op til de krav vi satte ved planlægningen af projektet. Under udviklingsprocessen kom vi frem til en række user cases, som skulle løses. Disse cases er enten blevet løst eller forbedret, dog med undtagelse af 3D-modellen.
Selve arbejdsprocessen og vejen mod løsningen kan ses i "Proces" afsnittet.

Hvordan vurderer man, om produktet er færdigt? Som vi også beskriver i Proces-afsnittet senere i rapporten, så blev der udarbejdet en række krav for kodestandarden. Dette kaldes også for "Definition of Done" (DOD).

Vi har løst alle vores user cases, med undtagelse af US-10, som omhandler et 3D print. Som tidligere beskrevet i User case afsnittet under US-10, så har vi vurderet det til at være en stor opgave. 3D-printet skal stamme fra en stand-alone applikation, som skal virke uafhængig fra vores website.
Vi danner os ellers en idé om, at applikationen skulle læse data fra databasen, for så at eksportere en .STL fil. Denne STL fil vil Fog have muligheden for at 3D-printe og sende videre til kunden. Denne user case krævede for meget tid ift vores tidsplan, og vi vurderede derudover også at vi ikke se den forretningsmæssige værdi, i at man kan udskrive en 3D model af materialerne på en lokal maskine, udenom vores droplet.

User stories:

Herunder vil vi beskrive de user stories, som endte med at få en anderledes løsning end deres acceptkriterier. Vi har udeladt US-10, som er beskrevet i afsnittet ovenfor. De User cases som ikke bliver nævnt er fuldt implementeret som beskrevet i acceptkriterierne:

I vores **US-2**: "*Som ny bruger, skal jeg kunne oprette en konto på siden, så jeg har mulighed for at logge ind."*
Her definerer vi vores acceptkriterie til at skulle tage email og password som input data, men her valgte vi også, at kunden skal indtaste fulde navn, tlf nr og adresse. Dette kan bruges af Fog til at kunne kontakte kunden, Og er gjort med henblik på ikke at kunne komme til at lave user objekter som ikke er i en gyldig tilstand.

I vores **US-3**: "*Som registreret kunde, skal jeg have mulighed for at bestille et tilbud på en carport efter egne mål."*
Vores acceptkriterie endte med at være en smule forskellig fra vores løsning. I acceptkriteriet står der, at efter man har valgt sine givne mål og trykket på "få tilbud" knappen, så vil ordren sendes videre til admin. Her har vi dog indsat en JSP side og Servlet som skal afspejle en indkøbskurv. Man bliver derfor redirected til indkøbskurven i stedet for ordre siden. Her har man så mulighed for at trykke "få tilbud" på ordren, hvorefter den sendes videre til admin.

Dette har vi gjort, så det er nemmere for kunden at have et overblik over sin bestilling. Vores løsning giver programmet et ekstra lag af sikkerhed for kunden, så der er mindre mulighed for at bestille en ordre, som ikke skulle være bestilt µ

I vores **US-4**: "*Som registreret kunde, skal jeg have mulighed for at vælge om jeg vil bestille et skur sammen med min carport, og hvor stort dette skur skal være."*
Her skulle kunden ifølge vores acceptkriterier klikke på en checkbox under bestillingssiden, som skulle tilføje et skur, herefter vil kunden være i stand til at ændre på målene for skuret.
I vores egentlige løsning endte vi med at lave to forskellige produkter, istedet for ét produkt med en checkbox:

- Carport med fladt tag
- Carport med fladt tag inkl. Skur

Under implementeringen af denne user story var det visuelt bedre for kunden at få et billede af, hvordan skuret egentlig ville komme til at se ud og det virkede uhensigtsmæssig at skulle opdatere siden (og dermed miste eventuelle mål du allerede havde skrevet ind), hver gang man klikker skuret fra eller til.

US-7: "*Som administrator, skal jeg have mulighed for at se nye ordrer, så jeg kan følge op på om deres bestilling giver mening, kontakte kunden, samt sende et tilbud, som inkluderer en pris, tilbage til kunden."*
Implementeringen af denne user case endte med langt hen af vejen at have en forskellig løsning fra vores acceptkriterier.
I vores løsning skal administratoren trykke på "admin" dropdown menuen og derefter vælge "Nye ordrer" siden, hvor vi i acceptkriteriet ville have brugt samme ordreside som kunden benytter. Dette var vi valgt at gøre for at adskille admins funktioner fra almindelige brugere, sådan at det ikke bliver blandet for meget sammen, men er helt tydeligt at de ting som ligger i denne dropdown er de sider som adminen skal udføre deres arbejde på. Her ville administratoren kunne se alle nye ordrer i butikken i stedet for kun at kunne se ordrer tilknyttet en enkelt user. Administratoren får også muligheden for at slette en ordre, kontra acceptkriteriet.

US-8: "*Som administrator, skal jeg have mulighed for at tilføje og fjerne materialer fra databasen, samt opdatere priser og længder."*
Der mangler at blive lavet en mulighed for at fjerne materialer via materiale update siden, dette blev ikke implementeret. Dette er en *nice-to-have* funktion, og er også et ønske fra kunden, men da opgaven var større end vi først havde regnet med nåede vi ikke at implementere dette. Det gik op for os at vi ville ugyldiggøre en række ordredata ved uhindret at få lov at fjerne materialer fra databasen, derfor gik vores overvejelser på at implementere en soft-delete løsning, som ikke kompromitterede anden data, men eftersom dette krævede en omstrukturering nede på database niveau og dermed en refaktorering hele vejen gennem systemet kunne vi ikke nå det.

US-9: *Som registreret kunde, skal jeg have mulighed for at få en ordrekvittering og en stykliste efter et tilbud er bekræftet og betalt."*
I denne user story bliver kunden redirected til en kvitteringsside, hvor de modtager en kvittering og en stykliste. Denne stykliste ville vi gerne inkludere i en email eller som en download funktion senere hen.
Vi har ikke implementeret nogen knap eller funktion for kunden til at kunne se kvitteringen og styklisten igen. Kunden har nemlig kun mulighed for at se den efter betaling for ordren. Hvis vores produkt skulle tages i brug af Fog, så skulle kunden have en mulighed for at kunne se kvitteringen og styklisten på hjemmesiden udover betaling. Dette ville som sagt også forekomme gennem email. Det var meningen vi ville have gemt når en ordrer var blevet betalt, som det også fremgår af vores EER-diagram, men eftersom det er nice-to-have at gemme kundernes kvitteringer endte vi med at nedprioritere dette, da vi begyndte at løbe ind i udfordringer med tidsfristen.

US-11: "*Som administrator, skal jeg kunne se alle ordrer i systemet, så jeg har mulighed for at følge op på mine kunders ordrer. "*
Ligesom i US-7, så benytter vi os af en anden JSP side og servlet. Man tilkommer denne ordreside ved at trykke på "admin" dropdown menuen. Her har administratoren mulighed for at trykke på "alle ordrer"-knappen for at se samtlige ordrer i butikken, både nye og gamle. Det var også på denne side vi ville have integreret webapplikationen på 3D-print-applikationen, men eftersom vi ikke fik implementeret 3D-print-applikationen virker den funktionalitet på siden ikke.

Afsluttende bemærkninger:

Der er ingen kendte fejl eller bugs, foruden de elementer som ikke er færdigimplementeret. Generelt set er vores produkt stabilt. Man kan navigere rundt i programmet og bestille carporte med forskellige mål, ændre på ordrerne og slette dem igen, uden at webapplikationen melder nogle fejl.

Derudover er designet endt med at blive bedre end forventet. Der er blevet brugt tid på at forbedre JSP sidernes design gennem Bootstrap. Det er ikke svært at se at der er blevet kælet for detaljerne, og at vi har forsøgt at gøre en indsats for at gøre applikationen indbydende for brugeren.

Vi er kommet i mål med at lave test til alle metoder, som vi havde planlagt at teste. Det er dog ikke i alle tests, hvor der bliver undersøgt andet end at tingene går som forventet. Vi vil komme ind på årsagen til dette i det efterfølgende afsnit - Test.

Havde vi haft mere tid er der en række ting vi gerne ville have implementeret:

1. Færdiggjort implementationen af US-8, som indebærer at indsætte en måde for adminen at soft-delete materialer.
2. Færdiggjort implementationen af US-9 og givet kunden en mulighed for at finde deres kvitteringer igen på et senere tidspunkt, samt givet kunderne en mulighed for at downloade deres kvittering.
3. Lavet en 3D applikation og fået den integreret med webapplikationen.
4. Videreudviklet på eksisterende test for at gøre dem mere valide.
5. Udvide materiale beregneren til at højde for vægt eller andet, som gør den i stand til at udvælge materialer afhængigt af materialets bredde og højde.

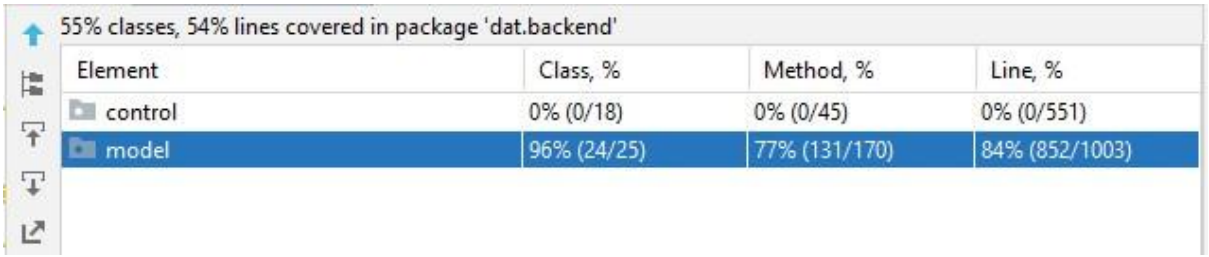
Test:

Testcoverage:

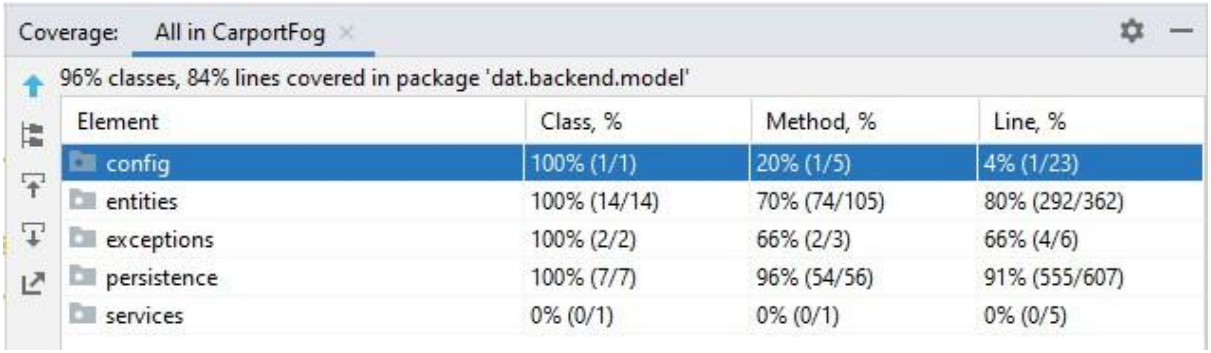
Vi har som udgangspunkt valgt at vores testcoverage skal dække alle Java metoder, som laver beregninger, manipulerer data eller ændrer objekters tilstand. Vi ønskede ikke at teste gettere, settere eller servlets. Vi har valgt ikke at teste gettere og settere, fordi de generelt ikke har noget indhold. Servlets har vi taget en beslutning om ikke at teste, fordi der sker så mange forskellige ting i dem, at de er svære at gøre testbare. Skulle man gøre dette, ville vi have været nødt til at opdele alle elementerne i vores servlets i mindre metoder, for derefter at teste dem individuelt. Det ville dog stadig have været svært at teste det samlede outcome, unittest af de mindre dele.

Vi har forsøgt at fange så mange mulige fejlinputs ude i frontend, med de biblioteker der importeres i de forskellige inputsformer. Så ved *double* inputs, kan brugeren kun benytte tal, ved valg af længder, kan brugeren kun benytte udvalgte længder, så udformningen af vores tests reflekterer dette.

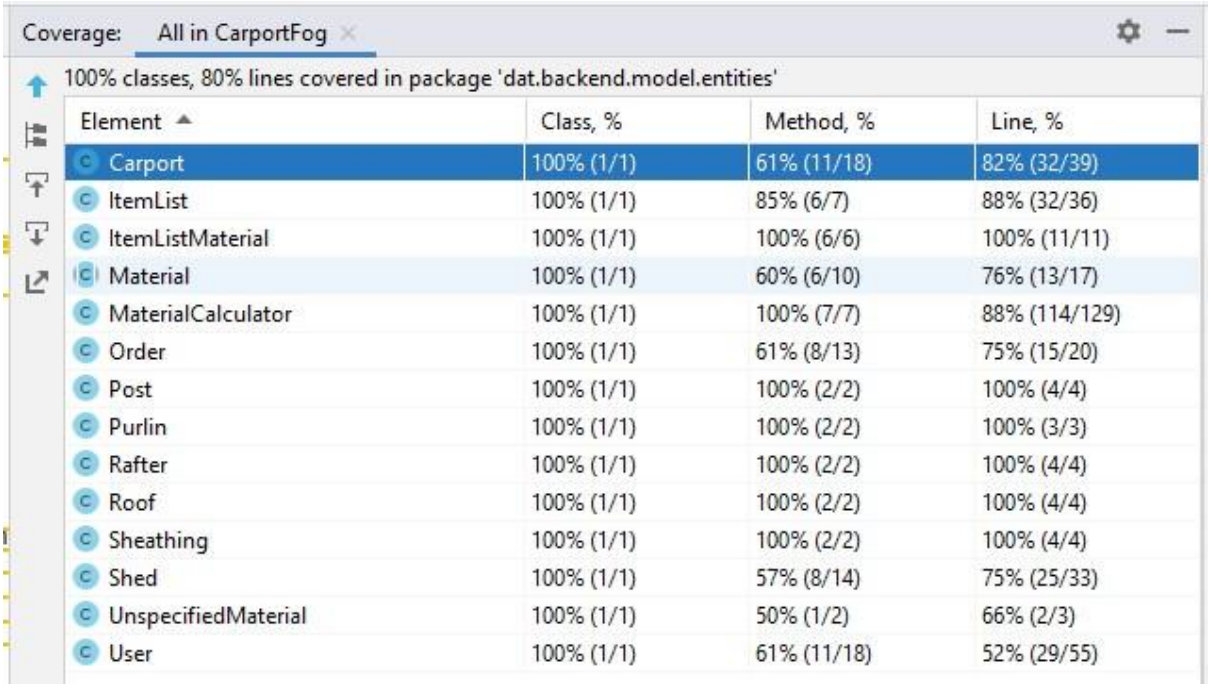
Vi har lavet tests løbende efterhånden som projektet blev bygget op, oftest i form af individuelle breakpoints. Dvs at vi har kodet en mindre eller større kode sekvens, og så har vi efterfølgende skrevet tests til den del, hvorefter alle tests skulle gennemføres før kode sekvensen blev fuldt integreret i projektet.



Som illustreret herover, ligger vores java klasser i model pakken, og alle vores servlets ligger oppe i control mappen. Her fremgår det tydeligt at vi på forhånd havde taget en beslutning om ikke at teste servlets, i resten af afsnittet vil vi derfor kun have fokus på klasserne der findes i model-packagen.

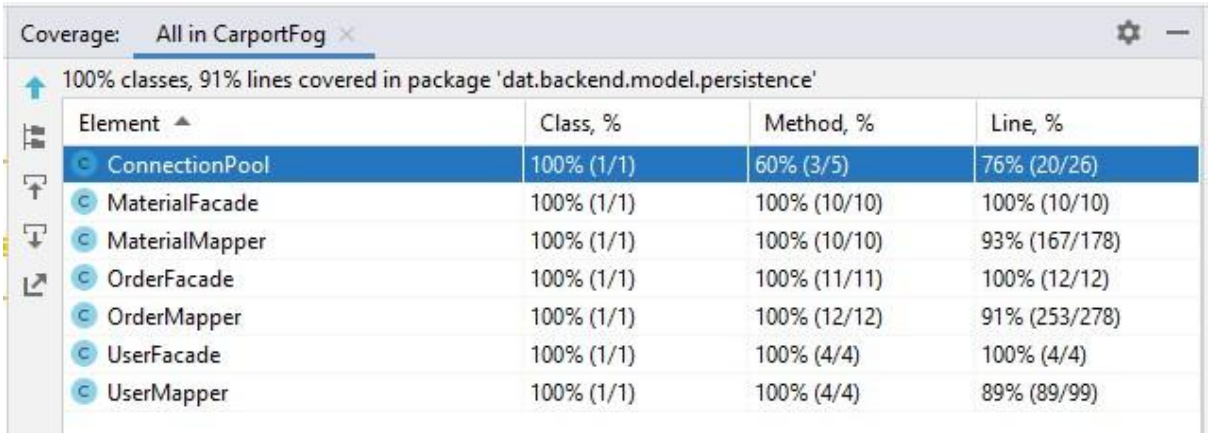


I ovenstående billede kan man se at vi tester 100% af alle de klasser, vi selv har skrevet. *Applicationstart*, der ligger i config pakken, og *Authentication*, der ligger i services pakken. Var begge en del af startkoden, og er kun delvist testet. De metoder i entities, exceptions og persistence pakkerne der ikke er testet, er alle gettere, eller i et enkelt tilfælde en abstract klasse, hvor det ikke giver mening at teste koden eftersom der ikke kan instantieres objekter af denne klasse og at metoder som falder ind under hvad vi vil teste, alligevel bliver implementeret i subklasserne, som bliver testet.



Element	Class, %	Method, %	Line, %
Carport	100% (1/1)	61% (11/18)	82% (32/39)
ItemList	100% (1/1)	85% (6/7)	88% (32/36)
ItemListMaterial	100% (1/1)	100% (6/6)	100% (11/11)
Material	100% (1/1)	60% (6/10)	76% (13/17)
MaterialCalculator	100% (1/1)	100% (7/7)	88% (114/129)
Order	100% (1/1)	61% (8/13)	75% (15/20)
Post	100% (1/1)	100% (2/2)	100% (4/4)
Purlin	100% (1/1)	100% (2/2)	100% (3/3)
Rafter	100% (1/1)	100% (2/2)	100% (4/4)
Roof	100% (1/1)	100% (2/2)	100% (4/4)
Sheathing	100% (1/1)	100% (2/2)	100% (4/4)
Shed	100% (1/1)	57% (8/14)	75% (25/33)
UnspecifiedMaterial	100% (1/1)	50% (1/2)	66% (2/3)
User	100% (1/1)	61% (11/18)	52% (29/55)

De to vigtigste mapper i denne sammenhæng er *entities* (ses åbnet herover), hvor vi har alle vores entitetsklasser (alm. java klasser) hvor vi i gennemsnit har testet 70% af vores metoder, men som sagt indledningsvis, så har vi kategorisk ikke testet gettere og settere, hvilket er de metoder der udgør de resterende 30%.



Element	Class, %	Method, %	Line, %
ConnectionPool	100% (1/1)	60% (3/5)	76% (20/26)
MaterialFacade	100% (1/1)	100% (10/10)	100% (10/10)
MaterialMapper	100% (1/1)	100% (10/10)	93% (167/178)
OrderFacade	100% (1/1)	100% (11/11)	100% (12/12)
OrderMapper	100% (1/1)	100% (12/12)	91% (253/278)
UserFacade	100% (1/1)	100% (4/4)	100% (4/4)
UserMapper	100% (1/1)	100% (4/4)	89% (89/99)

Det samme gør sig gældende i *persistence* klassen (ses åbnet herover), hvor alle vores facade og mapper klasser ligger. I disse klasser har vi ingen gettere og derfor er testningsgraden langt højere end i entitetsklasserne. Grunden til vi ikke når helt op på 100%, er fordi her ligger ConnectionPool klassen, som er en del af startkoden.

Unittest eksempel:

```
@Test
void calculateRoofs() throws NoMaterialFoundException{
    MaterialCalculator mc1 = new MaterialCalculator( length: 780,  width: 600,  minHeight: 210,  hasShed: true,  shedLength: 210,  shedWidth: 600);
    Set<ItemListMaterial> setRoofs = mc1.calculateRoofs(roofs);
    ItemListMaterial itemListMaterial = null;
    for(ItemListMaterial i: setRoofs){
        itemListMaterial = i;
    }
    assertEquals( expected: 40, itemListMaterial.getAmount());
}
```

Ovenstående Unittest beregner vi hvor mange tagplader der skal bruges til et givent areal af tag ud fra de kriterier vi har sat i metoden, bla. med et overlap på 20 cm.
I dette tilfælde forventer vi at få 40 plader. Selve metoden står nærmere beskrevet i vores udvalgte koder. I denne metode bruger vi også vores egen exception NoMaterialFoundException, som er beskrevet tidligere i rapporten.

Integrationstest eksempel:

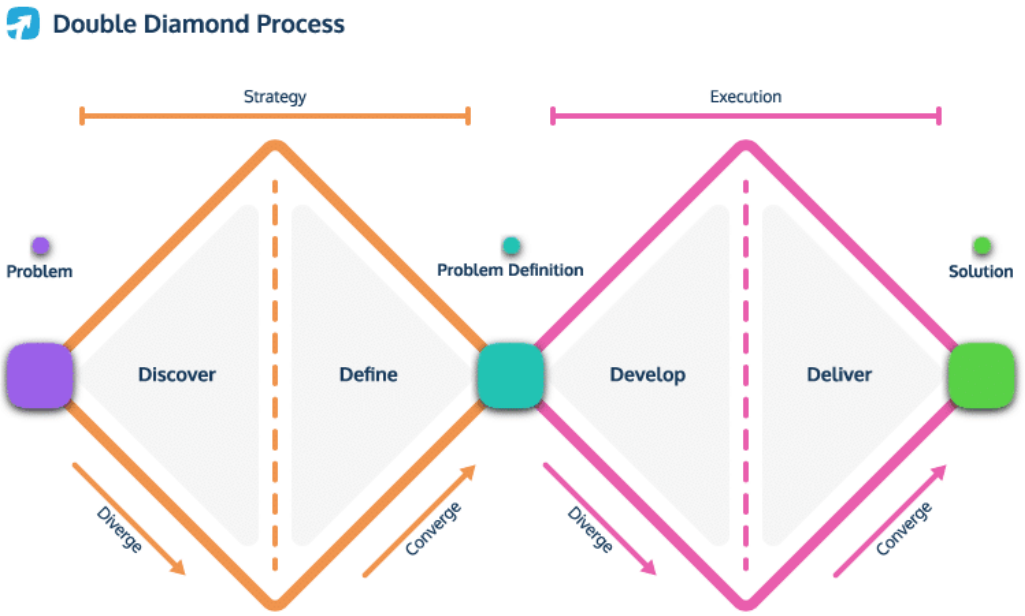
```
@Test
void sendOfferToCustomer() throws DatabaseException{
    assertTrue(OrderFacade.sendOfferToCustomer( orderID: 3,  salesPrice: 500, connectionPool));
    assertFalse(OrderFacade.sendOfferToCustomer( orderID: 4,  salesPrice: 2000, connectionPool));
}
```

I ovenstående eksempel tester vi send-tilbud-til-kunden metoden. Foregående har vi lavet en beforeEach metode, som sørger for at lægge 3 ordrer ind på vores testdatabase.
Dette er en integrations test, da vi tester hvordan vores applikation integreres med en del af systemet som ligger udenfor webapplikationen.
Så i første omgang siger vi, at testen skal give os et (true) godkendt tilbage, fordi vi sender et tilbud der siger at Order id = 3, på 500kr, skal sendes til kunden. Det bliver godkendt, fordi som beskrevet før, lagde vi 3 ordrer i testdatabasen.
Men i anden omgang, forventer vi at få et (false) ikke-godkendt tilbage, fordi vi i testen skriver at vi vil sende et tilbud med ordre id = 4, med en værdi til 2000kr, tilbage til kunden. Dette kan selvfølgelig ikke lade sig gøre, da der ikke findes nogen ordre med ID nr. 4. Derfor er testen godkendt, da vi både har testet at metoden virker, men også at den ikke gør, hvis den ikke får de korrekte variabler med. Denne test kan kun laves på metoder der returnerer en boolean.

Proces

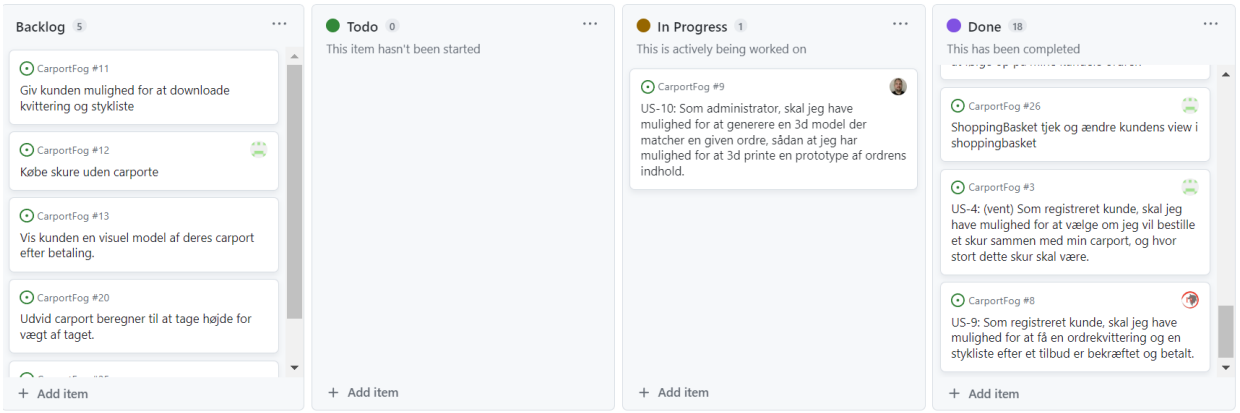
Arbejdsprocessen faktuelt

Under et projektforløb benytter vi os af en model kaldet “Double Diamond”.



I den første uge af projektet aftalte vi rammerne for vores projektarbejde. Vi udarbejdede analyser og modeller, der senere skulle hjælpe os med opsætning af database og javakode.
I cupcake projektet benyttede vi os af Github Projects med stor succes, så det valgte vi at bruge igen for semesterprojektet.
Med Github Projects såkaldte “issues” havde vi mulighed for at holde et overblik over de opgaver der skulle laves. Vi havde derudover også mulighed for at assigne folk til de forskellige issues, samt oprette nye issues løbende.
Vi oprettede separate branches for hver issue, så man ikke sad og arbejdede på samme branch ved flere forskellige opgaver.

Herunder er et screenshot af vores projekt view, taget d. 25/5, hvilket er under en uge før vi skulle aflevere projektet:



Her har issues 4 forskellige statusser:

- Backlog
- Todo
- In Progress
- Done

Backlog er ekstraopgaver, som er sidste prioritet, og er noget som man kan nappe af når alle andre issues er løst.
Todo er issues der mangler for at løse projektets user stories.
En issue må først sættes i "Done", når den løser vores definition of done.
Vores definerede DOD (Definition of done):
Opfylder alle funktionelle krav defineret ved acceptkriterierne.

- Relevante Integrationstest og Unittest skal lykkedes. (Metoder som laver beregninger, manipulerer data eller ændrer objekters tilstand)
- Javadoc laves til metoder og klasser og indeholder følgende:
    - Beskrivelse
    - Author
    - Parameter beskrivelse
    - Exception beskrivelse, hvis vi selv kaster exceptions.

I første fase, hvilket var første uge af projektforløbet, kodede vi slet ikke. Udover vores modeller og diagrammer, samt virksomhedsanalyse, lavede vi en gruppekontrakt, som dannede rammerne for vores samarbejde under projektforløbet. Gruppekontrakten (bilag 5), indeholder en række krav for medlemmerne af gruppen, både om vores arbejdsgang, men også hvem der har hvilke roller i gruppen.
Pelle fik ansvar for logbogen og Danyal for Databasen og Droplet. Vi gennemgik også Fogs kundevideo, og prøvede derefter at udarbejde en plan for en løsning til Martins ønsker. Herefter definerede vi user stories.
Ved første vejledningsmøde med Signe havde vi forberedt diagrammer og user stories, som vi skulle gennemgå og have godkendt. Vi fik godkendt user stories, men de manglede acceptkriterier, som vi senere løste.
Første del af vores første fase er "Discover" delen af Double Diamond modellen, hvor vi prøvede at danne os et overblik over problemet. Senere prøvede vi at definere problemet,

samt skabe en plan for vores projektstruktur. Ved dannelse af vores user stories og acceptkriterier, har vi så skabt en såkaldt problemformulering.

I anden fase af projektforløbet, hvilket var efter mødet med Signe d. 3/5, færdiggjorde vi vores modeller og diagrammer samt acceptkriterier. Herefter gik vi i gang med at programmere. Vi mødtes hver formiddag kl. 10 på Discord for at afholde et scrum-møde vedrørende vores daglige arbejdsgang, succeser og udfordringer.
Her nåede vi at implementere startkoden og vores database. Vi definerede hvilke begrænsninger vi skulle sætte for kunderne i form af hvad de skulle have af valgmuligheder på de forskellige sider.

Her kom vi også frem til, at nogle af vores issues er for store og besværlige at assigne til én person. Vi opdelte derfor nogle af vores user stories til mindre issues.
Det var i denne fase vi gik i gang med Double Diamond modellens develop fase, hvor vi åbner scopet for udvikling.

I tredje fase af forløbet efter mødet med Signe d. 10/5, havde vi en semi-funktionel hjemmeside, som vi kunne vise. Her snakkede vi primært om inkludering af test. Vi havde en del problemer med vores test database, som ikke viste de korrekte relationer mellem tabellerne i form af manglende fremmednøgler. Vi konkluderede, at vi ikke kunne nå 100 % coverage af tests for programmet. Istedet skulle vi overveje hvilke metoder der skulle testes. På dette tidspunkt havde vi en næsten fuldt funktionel side, hvor næsten alle vores user stories er løst. Programmet indeholder dog en del bugs samt mangel på design.

I fjerde fase efter møde med Signe d. 17/5, blev vi nødt til at definere en ny plan for færdiggørelsen af projektet. Vi har på dette tidspunkt for mange opgaver i gang, og vi klargører os derfor på, at vi muligvis ikke når alle de ting vi har tænkt os. På dette tidspunkt mangler vi at refaktorere koden, så kunden har mulighed for at tilvælge et skur til carporten. Vi mangler også 3D-printet, som vi så ikke nåede, hvilket vi også beskrev i "status på implementation" afsnittet.
Vi er i sidste del af Double Diamond modellen, hvor vi skal indskrænke projektet og færdiggøre vores issues, så vi snart står med et færdigt produkt.

Efter møde med Signe d. 25/5, går vi nu ind i femte og sidste fase af projektforløbet. Vores program er 95 % færdigt. Det eneste vi mangler er deployment samt 3D-print.
Under deployment af hjemmesiden stødte vi på en del problemer med Tomcat på serveren. Ved mødet med Signe snakkede vi om rapporten, om vores kodeeksempler, samt at vi skal prøve at holde en rød tråd gennem rapporten.
Vi er på dette tidspunkt i gang med rapporten. Derudover sidder vi og løser et par designmæssige opgaver, samt løser de bugs, vi finder løbende.
Ift. Double Diamond modellen er vi nu i allersidste del, hvor vi afslutter de sidste par opgaver. Vi prøver så vidt som muligt ikke at begive os ud i nye store kode mæssige opgaver.

Arbejdsprocessen reflekteret

Under projektforløbet er vi kommet frem til mange konklusioner angående vores tilgang og arbejdsstruktur for projektet. Vores erfaring for arbejde i en gruppesammenhæng stammer primært fra cupcake projektet, som vi satte vores rammer ud fra.
Når man ser tilbage på vores arbejdsgang, kan vi konkludere, at vi kom fint fra start i vores Discover fase. Vi brugte tid på at opsætte korrekte modeller, som gjorde det nemmere for gruppen at danne et overblik over projektets struktur. Tidligere i cupcake projektet manglede vi sekvensdiagrammer, der funktion for funktion viser kundens og administratorens gang i websiten. Projektet er meget nemmere at arbejde på, hvis man fastlægger brugen af servlets, JSP-sider og mapper funktioner, samt den måde de spiller sammen på.

Projektledelsen i gruppen er ikke styret af et enkelt medlem, men det er i stedet styret af alle. Vi tager i de fleste tilfælde ikke beslutninger, uden at have snakket med resten af gruppen. Denne styreform har både fordele og ulemper. Generelt set kræver vores projektstyring et ansvar fra alle medlemmer. Alle medlemmerne skal melde ind, og man har selv ansvar for at fremlægge eventuelle ideer og tanker til resten af gruppen. Man kan diskutere om denne form for projektstyring er bedre, end fx at have en gruppeleder. Begge måder at arbejde på har både fordele og ulemper. Når vi kigger tilbage på vores gruppedynamik, kan man dog godt ønske en mere struktureret tilgang til uddelegering af opgaver. Når vi kommer frem til en opgave der skal løses, så kræves der, at et af gruppemedlemmerne frivilligt melder sig til at løse opgaven. På denne måde kan nogle af gruppemedlemmerne ende med at arbejde mere end andre. Det kan dog også være en god måde for folk at melde sig på de opgaver, de gerne vil løse.

Vi havde som tidligere beskrevet scrum møder hver morgen kl 10 på Discord, hvis ikke andet var aftalt. Her plejer vi at snakke om status på de opgaver vi er i gang med. Her kan man også spørge resten af gruppen til råds, hvis man har siddet fast i en opgave. Derefter planlægger vi målet for resten af dagen, og i nogle tilfælde resten af ugen. Det var som regel efter mødet med Signe, at vi begyndte at udarbejde ugeplaner. Vi prøvede at vurdere, hvor meget vi derfor ville nå før næste møde.

Som tidligere beskrevet i user stories afsnittet, så havde vi ikke estimeret længderne på vores user cases før senere i opgaven. Dette gjorde det besværligt for gruppen at arbejde med, og en opgave, der virkede simpel, kunne gå over og tage længere tid end forventet. Dette resulterede i mangel på tid for vores gruppe. Da vi ramte midtpunktet af projektforløbet, følte vi os bagud, og vi begyndte derfor at prioritere de opgaver der var mest essentielle for løsningen af vores projekt.
Dette problem kan løses med bedre estimeringer af opgaver, samt ugeplaner over de opgaver der skal løses.

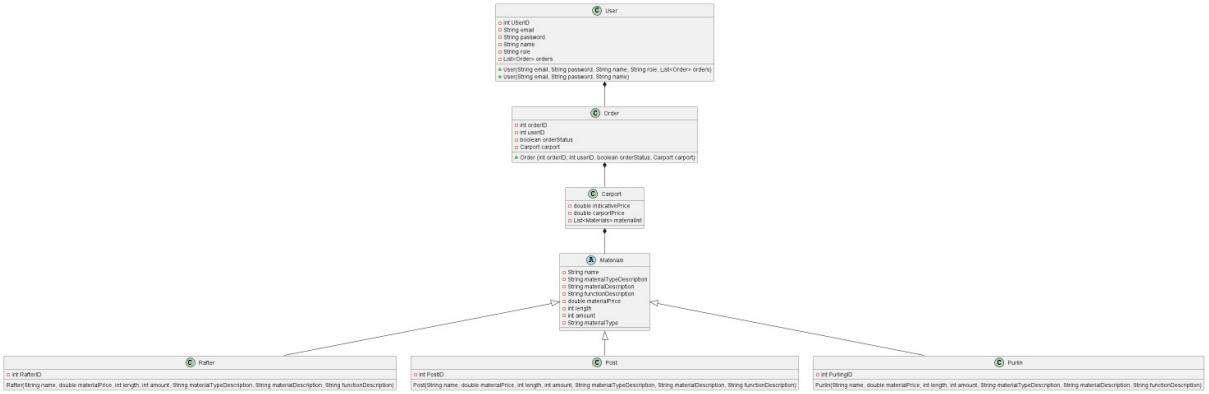
Generelt havde vi ikke de store problemer med at dele kode via Github. Medlemmerne af gruppen er så småt begyndt at danne god erfaring med kodning i grupper og arbejde under de rammer der gælder.
Som tidligere beskrevet i afsnittet, så havde vi opsat regler for DOD (Definition of Done), samt merging af kode. Vi har en main branch, som er beskyttet bag pull-requests, samt en

“dev” branch. Dev branchen blev benyttet som et ekstra lag af sikkerhed, før man kunne merge kode ind i main. Vi gjorde det også til et krav, at man først mergede dev ind i sin workbranch og løste eventuelle merge konflikter, samt sikrede sig at alle tests fortsat blev gennemført, før man mergede det ind i dev. Dette fungerede glimrende, og vi havde ikke nogen problemer med deling af koden.
Main branchen skulle repræsentere den deployede udgave af projektet, hvilket vil sige at vi ikke fik benyttet pull requests ret ofte i projektperioden. Det havde nok i højere grad været relevant for os at benytte denne struktur, hvis vi havde arbejdet videre på et projekt, der allerede var deployet, så vi kunne bruge pull requests til at beskytte “produktionen”. Flere pull requests løbende gennem projektet kunne dog have hjulpet os til at opdage fejl og mangler løbende gennem projektet. Vi har eksempelvis brugt meget tid sidst i projektet på at tilføje manglende dokumentation eller fjerne redundant kode. Dette kunne have været løst løbende ved hjælp af flere mindre review sessions, eller måske endda at have udvidet konceptet til også at beskytte vores dev-branch bag pull requests. Vi vurderede dog at dette ikke var en realistisk arbejdsgang, når vi havde så kort tid til at løse opgaven, da vi havde risikeret lange ventetider mellem code reviews.
Det var heller ikke i alle tilfælde, at alle medlemmerne af gruppen havde udført tests eller Javadoc dokumentation ved tilføjelse af nye metoder, servlets eller andet. Det var dog generelt meget bedre end i cupcake projektet.

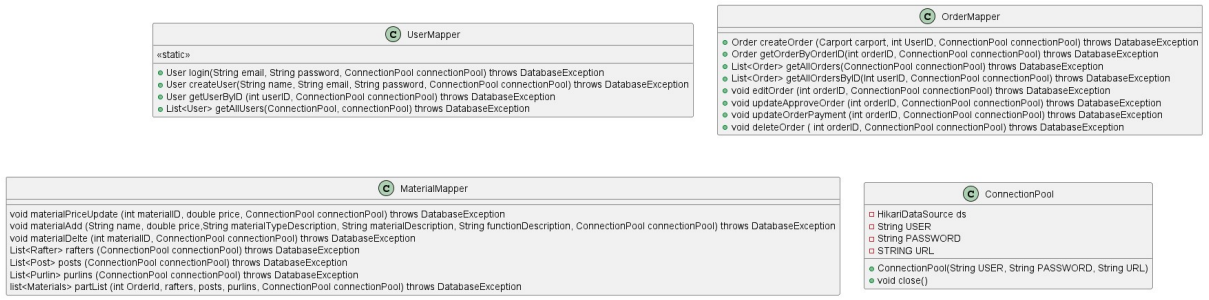
Generelt set er vi tilfredse med projektet, rapporten og koden. Vi vurderer det nemt for udefrakommende programmører at overtage projektet, da programmet både kører uden problemer, samt vi har inkluderet nødvendige tests og beskrivelser.

Bilag:

Bilag 1.1

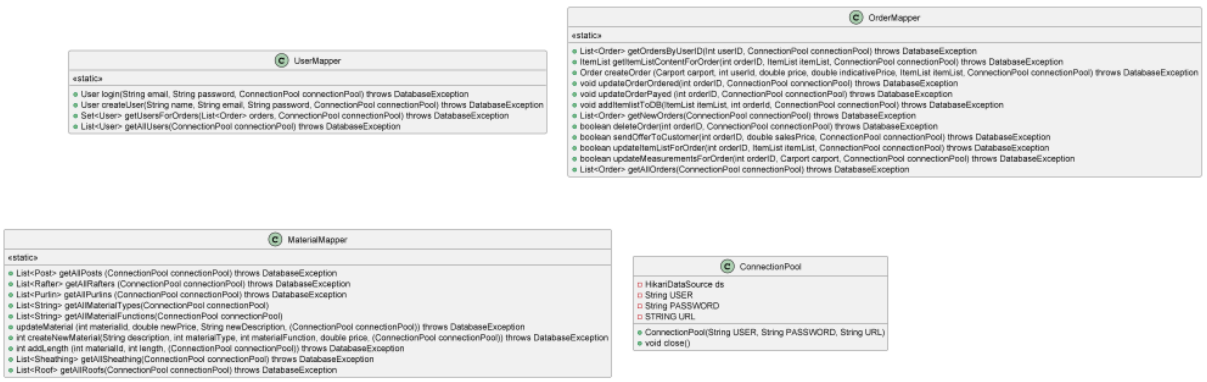


Bilag 1.2

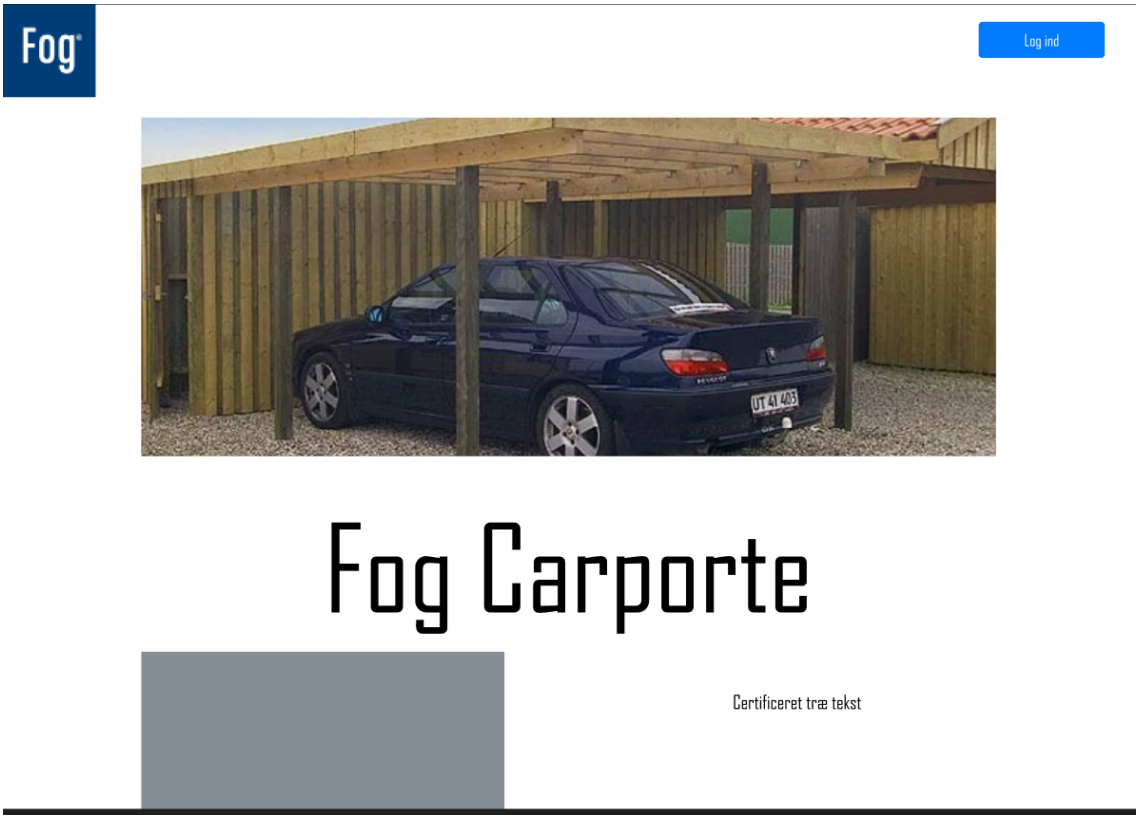


Bilag 2.1

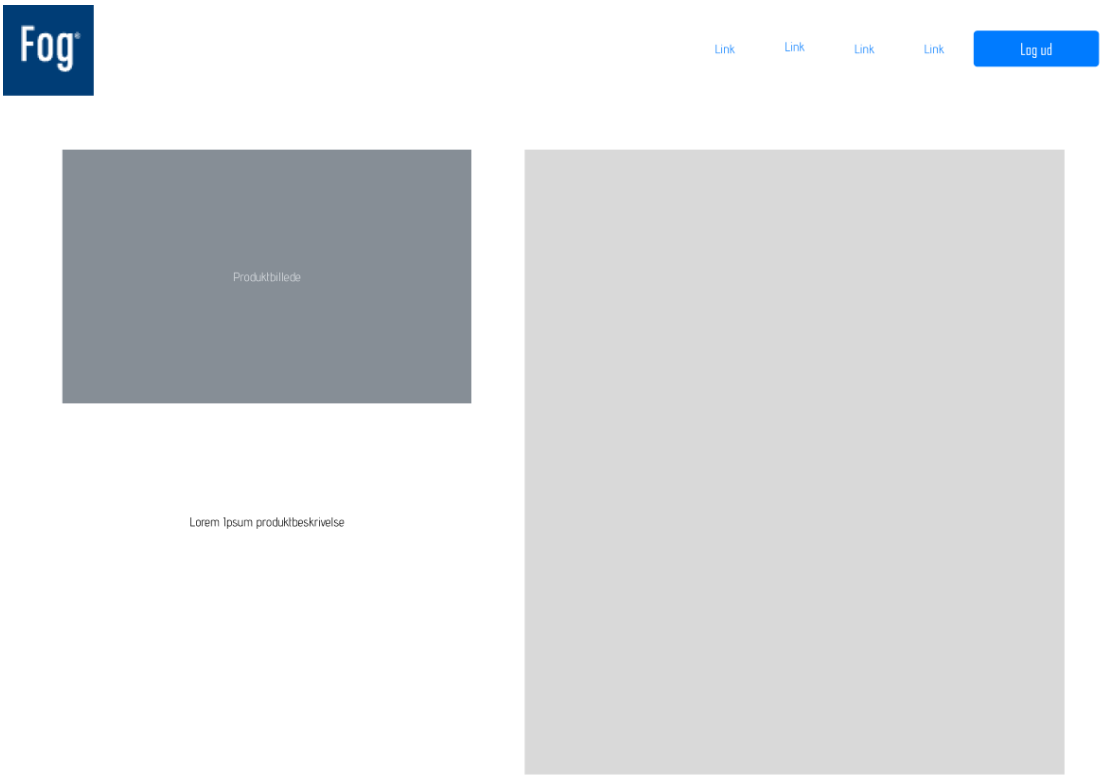
Bilag 2.2



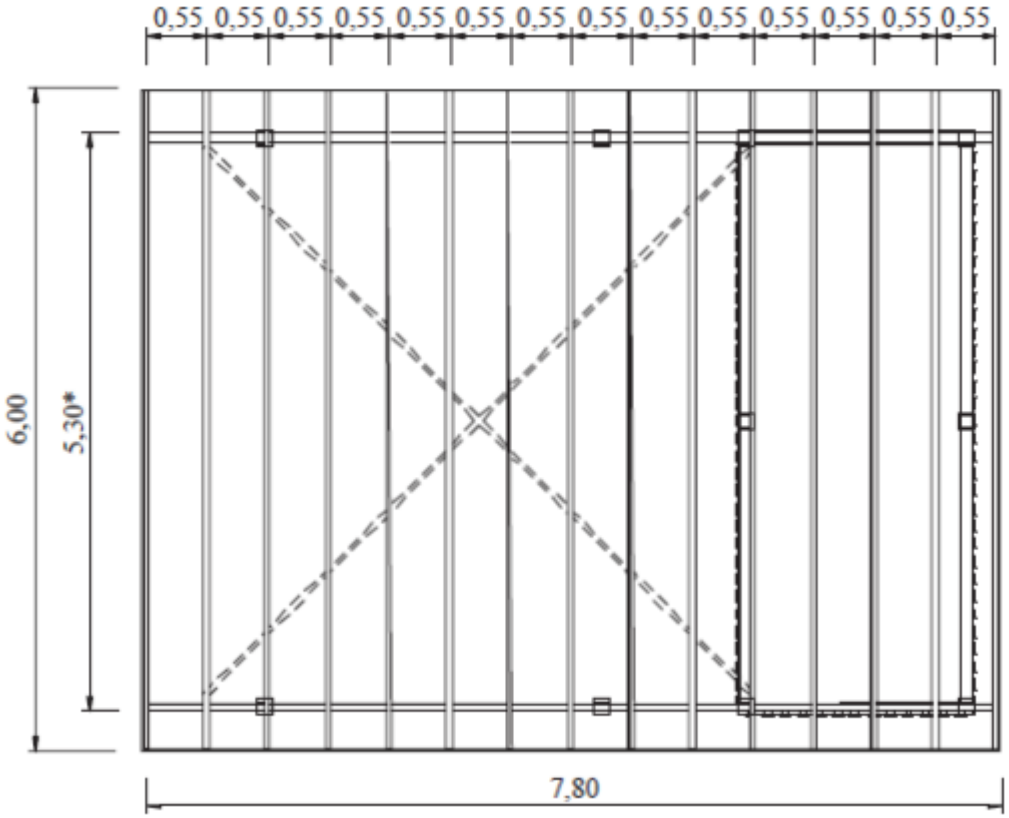
Bilag 3.1



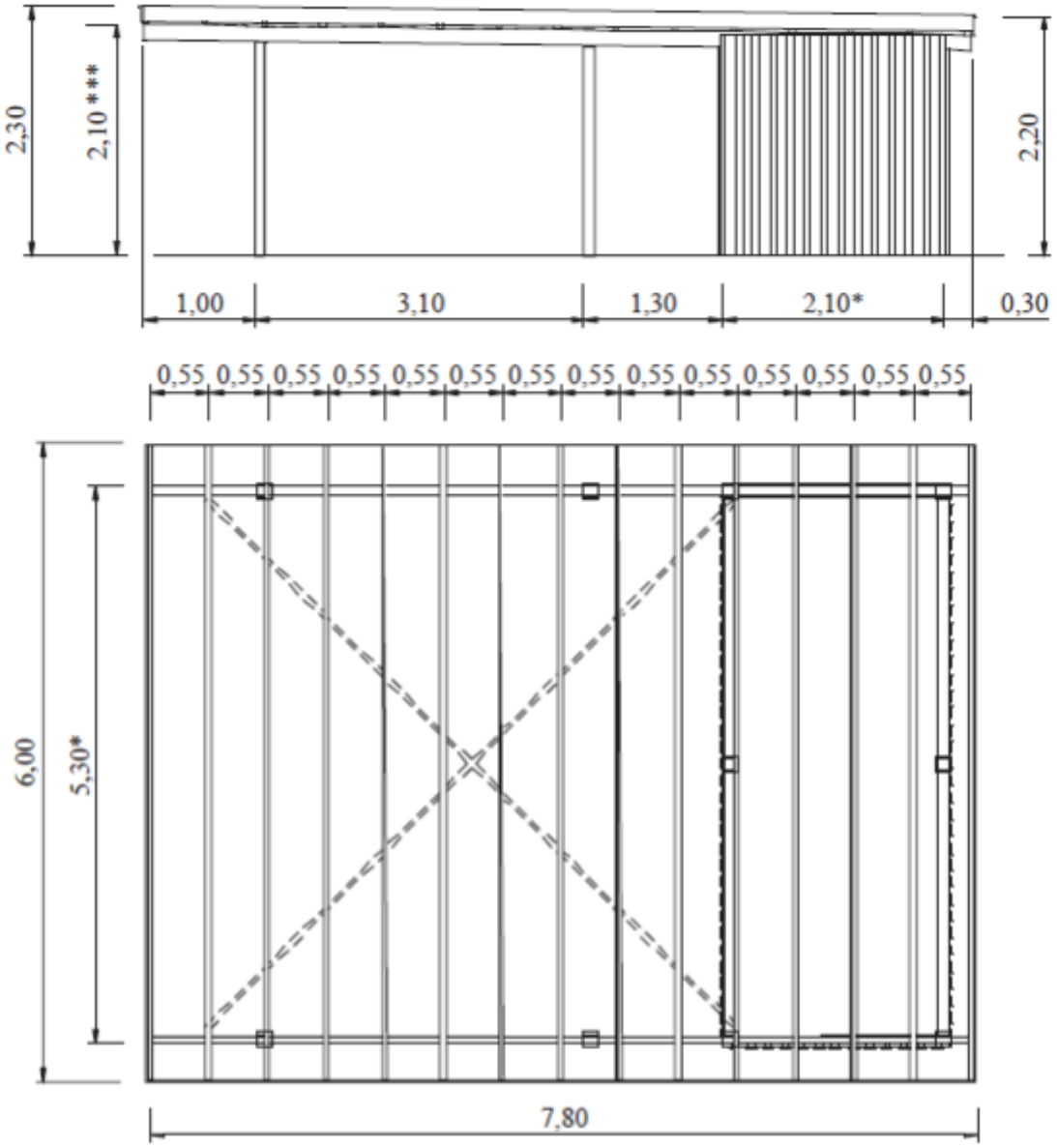
Bilag 3.2



Bilag 4.1


0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55
6,00
5,30*
7,80

Bilag 4.2


2,30
2,10 ***
2,20
1,00
3,10
1,30
2,10*
0,30
0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55 0,55
6,00
5,30*
7,80

Bilag 5.

Gruppekontrakt:

- Arbejdstid - Hvor meget tid bruger vi?
  Man hopper selv på opgaverne, og er selv ansvarlig for at arbejde inden for en rimelig tidsgrænse. Vi har ikke deciderede deadlines, men der er en forventning om en rimelig arbejdsgang, der ikke hæmmer gruppens fremskridt i projektet.
  Vi vil gerne være færdige med modeller og alt det indledende arbejde som gruppekontrakt og planlægning onsdag i første uge.

- Hvor ofte mødes vi? - Hvad skal vi snakke om?
  - Vi skal mødes alle dage i ugen kl 10 på Discord (medmindre andet er aftalt), undtagen lørdag og søndag.
  - Et scrum-møde kan tage 10-30 minutter, baseret på hvad vi skal snakke om.
  - Vores møder vil omhandle igangværende og planlagte issues i Github.
  - Når vi har vejledningsmøde, mødes vi en time før på skolen. Her kører vi vores normale scrum-møde, herefter vil vi diskutere de vigtigste ting vi skal tage med vejleder. Herefter kan vi foretage gennemgang af kode.
- Arbejdsgang/Projektstruktur/ambitionsniveau
  - Det er generelt svært at planlægge, hvor meget tid vi vil bruge på planlægning, kodning og rapport. Umiddelbart vil vi gerne så hurtigt som muligt i gang med kodningen, og så sætte 3-4 dage af til rapporten til sidst. Derudover vil vi også gerne sætte 2-3 dage af til ren integration og finpudsning af koden.
  - Vi vil gerne have et sammenhængende projekt, hvor strukturen og designet af koden er det samme hele vejen igennem.
  - Vi skal nå alle vores user stories.
  - Vi benytter os af en developer branch udover vores main branch, så vi kan kode med vores deploy. Derudover benytter vi os af branches, der er navngivet efter de opgaver man arbejder på.
    Vi beskytter Main Branch bag pull requests, så alle skal ind og godkende, før man kan merge ind i Main.
  - Vi assigner opgaver i Github Issues. Derudover prøver vi at benytte os af Github Projects på en mere professionel måde.
- Skriv selv vs kodegenerering
  - Man må gerne benytte sig af "template-kode", men man skal kunne forstå og forklare koden til resten af gruppen, kode for kode.
- Roller/ansvarsområder
  - Vi giver 1 person ansvar for logbogen, men alle må gerne gå ind og skrive notater til vigtige overvejelser eller problemer man har haft.
  - Pelle får ansvar for logbog
  - Danyal får ansvar for Droplet/DB.
  - Uddelegering af andre roller kan komme løbende, men det er ikke noget vi aftaler på dette tidspunkt.