Johannes Fog - Carport Case - Rapport

May 30, 2018

Jacob Borg - cph-jb308@cphbusiness.dk - Jack-Borg
Daniel Lindholm - cph-dl110@cphbusiness.dk - Hupra
Nikolaj Thorsen Nielsen - cph-nn134@cphbusiness.dk - NikolajX4000
Stephan Marcus Duelund Djurhuus - cph-sd115@cphbusiness.dk - Stephan-MDD

Link til GitHub repository: https://github.com/NikolajX4000/Sem2Exam
Link til deployet sitet: websitet
Link til javadoc: https://nikolajx4000.github.io/Sem2Exam/

Contents

1 Indledning

Vi er blevet kontaktet af Johannes Fog Værebro, der har spurgt om vi kan lave et program og tilhørende hjemmeside til deres salg af carporte. Deres nuværende system bygger på manuelt arbejde samt en ikke tilstrækkelig adgang til ændringer i både system og database. Dette er nogle af fokuspunkterne der er stillet i vores opgave.

1.1 Baggrund

Johannes Fog er en koncern der både har design & bolighuse og trælast & byggecenter. Vi vil i dette project fokusere på Fog Værebro, der har bedt os om at lavet et nyt system til deres salg af carporte. Fog Værebro sælger som de andre trælast & byggecentre, træ, byggematerialer og alt det du behøver til hus og have inden for f.eks. maling, bad og VVS, beslag, elartikler og lamper samt haveredskaber, grill og havemøbler, men derudover har de gjort det til deres varemærker at være specialister i carporte. Det er i den forbindelse at vi er blevet bedt om at udvikle et system, med tilhørende hjemmeside, til at erstatte deres nuværende system, da de har erkendt at det er outdated. Med systemmet skal man kunne gå på hjemmesiden og bestille en carport, systemmet skal kunne udregne en stykliste af materialer, en medarbejder skal kunne gå ind og give en kunde særlige tilbud, og en kunde skal kunne se sin ordre.

1.2 Teknologivalg

- Projektet er et Maven 3.1
- Projekt skrevet i Netbeans IDE 8.2
- mysql 5.1.39
- MySQL Workbench 6.3ce
- Ubuntu 16.04.3 x64 server
- Websitet er deployet vha. apache-tomcat-8.0.32
- Sprogene der er brugt i koden er java jdk 1.8.0_141
- JAVAX 7.0
- HTML5
- JSTL 1.2 til at opbygge JSP sider
- JBCrypt 0.4 til salt og hashing af passwords

2 Krav

På websitet skal man som kunde kunne bestille en carport. Man skal kunne vælge om taget skal være med eller uden rejsning, og hvis man har valgt med skal man kunne vælge hvor meget hældning man vil have på taget. Man skal kunne vælge om man vil have et redskabskur, og hvor stort det skal være. Når man bestiller en carport skal man kunne indtaste sine oplysninger, navn, telefonnummer, email og adresse. Når man har bestilt en carport kan man se alle de ordre der er bestilt med den samme email. Man skal som bruger kunne se alle sine tidligere ordre ved at indtaste sin email.
Som medarbejder skal man kunne se alle ordre. Man skal kunne ændre status og pris på en ordre. Man skal kunne se navne og priser på de forskellige typer træ, skruer og beslag.
Systemet skal kunne udregne en stykliste udfra en ordre og en vejledende pris udfra styklisten, som en medarbejder kan bruge til at lave et tilbud til kunden.

2.1 Overordnet beskrivelse af virksomheden

Fog er en koncern med afdelinger i det meste af nordsjælland og en enkelt i Vordingborg. Af deres 10 afdelinger har de 1 afdeling med design og bolighus placeret i Lyngby, og 9 afdelinger med trælast og byggecenter placeret i Farum, Fredensborg, Helsinge, Herlev, Hørsholm, Kvistgård, Lyngby, Værebro og Vordingborg. Udover at være trælast og byggecenter, har Fog i Værebro gjort det til sit varemærke at være specialister i carporte.
Fogs arbejdsopgaver kan inddeles i følgende funktioner:

- Salgsfunktion
  - Kundebetjening
  - Markedsføring
  - Vareopfyldning
  - Sætte kunde i kontakt med tømrer
- Indkøbs- og lagerfunktion
  - Valg af leverandør
  - Varebestilling
  - Varemodtagelse
  - Modtagekontrol
- Økonomifunktion

- Opstille økonomiske mål
- Budget
- Regnskab
- Bogføring
- Løn
- Budgetkontrol

Fog er en handelsvirksomhed, da de køber byggematerialler, værktøj, osv. og sælger det videre. Da de sælger deres varer til privatpersoner, er det en detailvirksomhed. Fog har ejerforholdet aktieselskab

2.2 Arbejdsgange der skal IT-støttes

As-is

Aktivitetsdiagram: as-is kan ses i Appendix A: Diagrammer: Figure 5.
Når en kunde laver en ordre bliver der genereret en email, som en medarbejder manuelt skal indtaste i IT-systemet. Fog vil gerne derefter kunne ringe og snakke med kunden da kunde service er meget vigtigt for Fog. Efter samtale med kunden har medarbejderen mulighed for at lave ændringer til styklisten og lave et tilbud til kunden.

To-be

Aktivitetsdiagram: Kunde bestiller en carport kan ses i Appendix A: Diagrammer: Figure 6.
Når en kunde opretter en ordre vil IT-systemet udregne en stykliste og en vejledende pris ud fra styklisten. En e-mail bliver sendt til fog for at underrette om den nye ordre i systemet, en medarbejder tjekker ordren og har mulighed for at lave rettelser i styklisten og derefter kan medarbejderen sende et tilbud til kunden.

2.3 Scrum userstories

Som kunde vil jeg kunne bestille en carport så jeg kan få en carport.
how-to-demo:

- Bruger skal kunne vælge om carporten skal være med eller uden rejsning.
- Bruger skal kunne vælge størrelse mm. på en carport.

tasks:

- Opret order tabel i databasen.
- Opret shoppage med form til at angive størrelse på carport.
- Opret ordermapper.
- Opret user til databasen.
- Giv useren til databasen insert privilegier.
- Opret create metode i ordermapper.
- Opret order klasse.
- Command til bestilling.

estimat: Estimeret til en uges arbejde.
INVEST:

- Independent: For at give kunden mulighed for at bestille en carport. Så den er afhængig af flere undersystemer.
- Negotiable: Userstoryen beskriver ikke om det skal være en pre-designet eller selv-designet carport.
- Valuable: Værdien i denne userstory er stor da kunden ikke kan bestille en carport uden.
- Estimable: Vi estimerede denne til at være en ugens arbejde da det krævede at vi skulle lave siden og databasen hvorpå man kunne bestille en carport.
- Small: Denne userstory var Episk eftersom den krævede en del mere arbejde end de fleste andre.
- Testable: Vi testede denne ved at se om vores database indeholder den rigtige data efter en bestilling.

Som kunde vil jeg kunne se en tegning af min carport så jeg kan se hvad det er jeg køber.

how-to-demo:

- Brugeren skal kunne se en tegning af en carport med fladt tag fra oven.
- Brugeren skal kunne se en tegning af en carport med fladt tag fra siden.
- Brugeren skal kunne se en tegning af en carport med rejsning fra oven.
- Brugeren skal kunne se en tegning af en carport med rejsning fra siden.

tasks:

- Lav tegning for carport med fladt tag fra oven.
- Lav tegning for carport med fladt tag fra siden.
- Lav tegning for carport med rejsning fra oven.
- Lav tegning for carport med rejsning fra siden.

estimat: Estimeret til en uges arbejde.

INVEST:

- Independent: For at udregne en tegning kan der bruges placeholder mål så den er uafhængig.
- Negotiable: Storien definere ikke hvordan carporten skal tegnes, men kun at man skal få en ide om hvordan den vil se ud når man har bygget sin carport.
- Valuable: Det er af ret stor værdi for kunden at kunne se hvordan den carport man bestiller kommer til at se ud.
- Estimable: Denne user story vurderede vi til at tage en uges arbejde siden er mange detaljer der skal regnes ud.
- Small: Dette punkt er ikke opfyldt da det er en ret stor story.
- Testable: Når hele carporten er tegnet kan man manuelt tjekke om alle mål ser ud til at passe.

Som medarbejder vil jeg kunne udregne en stykliste så jeg ved hvilke materialer der skal bruges.

how-to-demo:

- Styklisten stemmer overens med tegningerne.

tasks:

- Lav stykliste for carport med fladt tag.
- Lav stykliste for carport med rejsning.
- Lav tabel i database med tilgængeligt materiale.

estimat: Estimeret til en uges arbejde.

INVEST:

- Independent: For at udregne stylisten skal man kun bruge mål fra en ordre, som nemt kan bruges nogle placeholders i stedet.
- Negotiable: Storien definere ikke hvordan styklisten skal udregnes bare hvad den skal ende ud med.
- Valuable: User storyen skaber værdi i form af der skal bruges en stykliste før man kan sælge nogle carporte.
- Estimable: Denne User story vurderede vi til at tage en uges arbejde siden er mange detaljer der skal regnes ud.
- Small: Dette punkt er ikke opfyldt da det er en ret stor story.
- Testable: Alle metoder kan vi give et input hvor vi ved hvad output skal være.

Som medarbejder vil jeg kunne se en vejledende pris baseret på en styklisten så jeg har et udgangspunkt for at kunne give kunden et tilbud.

how-to-demo:

- Medarbejderen kan se en pris der er beregnet ud fra styklisten.

tasks:

- Udregning for carport med fladt tag.
- Udregning for carport med rejsning.

estimat: Estimeret til et par timers arbejde.

INVEST:

- Independent: Der kan beregnes en pris ud fra en dummy stykliste.
- Negotiable: User storyen siger at der skal beregnes ud fra en stykliste så det ligger ret fast.
- Valuable: For at kunne give et realistisk tilbud er det godt for sælgeren at have et overblik over hvor meget materiellet koster.
- Estimable: Denne user story vurderede vi til at tage et par timer da den ikke skal tage højde for ret mange forskellige ting.
- Small: Denne user story skal kun lave en enkelt udregning så den er ikke så stor.
- Testable: Man kan teste om prisen bliver det man forventer når man giver den en prædefineret dummy stykliste.

Som bruger af systemet vil jeg have et brugervenligt design så jeg kan finde rundt.

how-to-demo:

- Sitet ser godt ud og er nemt at finde rundt i.

tasks:

- Formatér dato.
- Formatér pris.
- Styling.

estimat: Estimeret til et par timers arbejde.

INVEST:

- Independent: Designet er ikke indflydelse på funktionaliteten af applikationen.
- Negotiable: Designet var ikke en høj prioritet, da der ikke var noget konkret facit udover at vores PO skulle kunne finde rundt. PO kan altid komme med ideer til designet.
- Valuable: Et godt design skaber en brugervenlighed hvilket resultere i en bedre oplevelse.
- Estimable: Designet er på alle vores JSP sider, hvilket betyder at for hver gang vi tilføjer en ny JSP side, skal der laves et design. Derfor er det svært at estimere tiden da den er dynamisk.
- Small: Dette punkt er ikke udfyldt da den ikke er ret stor.
- Testable: Design er subjektivt, men vi har prøvet at gøre det nemt og overskueligt at navigere rundt på siderne, bestille carport og håndtere rettelser. For at se om designet er tilstrækkeligt vil man til sidst gennemgå det med PO.

3 Domæne model og ER diagram

Domæne model kan ses i Appendix A: Diagrammer: Figure 1.
En ordre indeholder mange PartLines og et Roof. En PartLine indeholder et Material og høre til en specifik Ordre. Et Material, kan hører til mange PartLine. Et Roof, kan tilhøre mange ordre. Ud fra en ordre kan der genereres to tegninger, en fra siden og en fra toppen.

ER diagram kan ses i Appendix A: Diagrammer: Figure 2.
Vi har valgt at i vores løsning holder vi i ordren alle relevante informationer om kunden, samt størrelsen på carporten, skur, hældning på taget, tagtype, hvornår ordren blev oprettet, status, prisen af materiel og den endelige pris.
Vores ordre indeholder en foreign key til roofs tabellen så vi sikre os at det ikke kommer en ordre med et tag vi ikke har i databasen.
Materials indeholder alle bjælker, stolper, tagsten, rygsten, osv. Tools indeholder alle skruer og beslag.
Employees indeholder alle de logins der har adgang til admin delen af vores IT-system, password ligger hashed i databasen.

Klassediagram kan ses i Appendix A: Diagrammer: Figure 3.

- Ordre:
  - calculatePrice()
    - bruger PartLine.calculatePrice() for hver PartLine for at udregne den totale pris.
  - hasShed()
    - tjekker om shedWidth og shedLength er større end 0.
  - isFlat()
    - tjekker om angle er lig 0.
  - getStatusColor()
    - kigger på status for at finde den rigtige farve.
  - generateStringId()
    - generer en ramdom String til brug som ID i html.
  - getStringId()
    - returnere stringId og hvis det ikke er sat generer den et.
  - getDrawingSide()
    - bruger isFlat() og kører den korrekte af

            - DrawCarportFlatSide(Order).getDrawing().
            - DrawCarportAngleSide(Order).getDrawing().
    - getDrawingTop()
        * bruger isFlat() og kører den korrekte af
            - DrawCarportFlatTop(Order).getDrawing().
            - DrawCarportAngleTop(Order).getDrawing().
    - getPartList()
        * tjekker om partsList er sat ellers udregner den partsList med enten
            - FlatCarportList(Order).getParts().
            - TallCarportList(Order).getParts().

- PartLine:
    - calculatePrice()
        * hvis Material har en størrelse forskelling fra 0
            - udregn prisen ud fra Material.getPrice() * størrelsen * amount
        * udregn prisen ud fra Material.getPrice() * amount

4 Navigationsdiagram

Navigationsdiagram kan ses i Appendix A: Diagrammer: Figure 4.
Når man kommer ind på websiden kommer man først til index siden.
Derfra har man mulighed for at komme til makeCarport siden, hvor man kan bestille en carport, derfra kommer man over på specificUserOrders siden, hvor man kan se alle sine ordrer. Man kan også komme til specificUserOrders siden fra index siden ved at indtaste sin email.
Man har fra index siden også mulighed for at komme til login siden, hvor en medarbejder kan logge ind. Når man logger ind kommer man til allOrders siden, hvor man får en liste med alle ordrer.
Vi bruger en header med links, så man fra alle siderne kan komme til index siden, makeCarport siden, og specificUserOrders siden, og hvis man er logget ind kan man også komme til allOrders siden og editMaterials siden, hvor man kan se tilgængeligt materiale, og ændre på priser og længder.
Vi har også en footer, hvor vi har et link til login siden, så man kan også komme dertil fra alle siderne.

5 Sekvensdiagrammer

Placér ordre

Sekvensdiagram: Placér ordre kan ses i Appendix A: Diagrammer: Figure 7.
I dette diagram kan man se hvordan sekvensen fra kundes input bliver bearbejdet i de forskellige filer. Først indsætter kunden de givne mål og informationer til den ønskede carport i makeCarport.jsp. Disse informationer bliver hentet af CmdCreateOrder.java hvor de bliver lagt ind i et Order objekt. Efter objektet er lavet, sendes det videre til addOrder() gennem OrderMapper.java. OrderMapperen laver en forespørgsel til databasen på at tilføje ordren til databasen, hvorefter alle ordre kunden har lavet bliver hentet og lagt i requestet som desiredOrdersFromEmail. Til sidst returnere CmdCreateOrder.java siden specificUserOrders, som gennem vores FrontController kalder siden specificUserOrders.jsp, og fremviser alle kundes ordrer.

Se ordre

Sekvensdiagram: Se ordre kan ses i Appendix A: Diagrammer: Figure 8.
Kunden starter med at indtaste en email som bruger commanden 'CmdShowOrders.java' som så tager det parameter der er sendt med fra mailen og bruger metoden getOrders() til at gå ned i mapperen som så leder efter alle ordrer i databasen med den email. De bliver så returneret til mapperen som så laver det til en liste af ordrer den giver videre til 'CmdShowOrders.java'. Den ligger dataen ned i en attribut som så vil kunne blive vist frem på 'specificUserOrders.jsp'.

Opdater ordre

Sekvensdiagram: Opdater ordre kan ses i Appendix A: Diagrammer: Figure 9. Dette diagram viser hvordan en ansat kan opdatere en ordre. Ved at starte på allOrders.jsp, kan man for hver ordre inspicere og opdatere pris og status. Når ændringerne er færdige vil knappen opdater kalde kommandoen CmdUpdateOrder.java som henter de nye indsatte værdier. Disse værdier bliver sendt videre til vores OrderMapper.java gennem metoderne updatePrice() og updateOrder(). Metoden updatePrice() opdatere prisen på carporten, hvor updateOrder() opdateret statusen på ordren. Begge metoder kalder vores database og laver et UPDATE statement. Derefter hentes alle ordrer også dem som nu er opdateret gennem metoden getAllOrders() og gemmes i requestet som orders. Til sidst returnere CmdUpdateOrder.java siden allOrders som gennem FrontControlleren kalder siden allOrders.jsp og fremviser alle ordrer.

Inspicér ordre

Sekvensdiagram: Inspicér ordre kan ses i Appendix A: Diagrammer: Figure 10. På dette sekvensdiagram kan man se hvordan man inspicer en ordre. Først går man ind på alle ordre siden hvorefter man vælger den ordre man gerne vil se. Meget af dataen om ordren vil allerede vil være loadet ind på siden, men styklisten vil først blive lavet når man klikker ind på den. Dette vil ske via et ajax kald, som så vil load den ind på siden.

Se alle ordrer

Sekvensdiagram: Se alle ordrer kan ses i Appendix A: Diagrammer: Figure 11. Diagrammet ViewAllOrders viser hvordan man som ansat kan få en historik af alle ordrer. Ved at logge ind som ansat vil hans informationer blive hentet i CmdLogin.java. Disse værdier bliver sendt videre til EmployeeMapper.java som kalder databasen for at retunere et resultset til EmployeeMapper.java som bliver valideret og returneret til CmdLogin.java. Efter dette vil OrderMapper.java blive kaldt som vil forespørge alle ordrer fra databasen. Disse vil blive hentet og sendt videre til CmdLogin.java og gemt i requestetet som orders. Derefter bliver vi sendt videre til allOrders.jsp gennem vores FrontController. På siden allOrders.jsp eksekveres et JSTL for-loop som fremviser alle ordrer.

6 Særlige forhold

Session

Hvis man logger ind som en employee gemmer vi en reference til et employee objekt med information om den bruger. Denne information bruges til at se om man har adgang til forskellige sider.

Exceptions

Vi arbejder primært med to forskellige exceptions "CustomException" og "NoAccesException".
Hvis der opstår en anden slags exception f.eks. en SQLException eller NumberFormatExeption vil den blive grebet, og der vil blive kastet en ny "CustomException" som man så giver en brugervenlig besked, som der vil kunne blive vist til brugeren på siden med hvad der gik galt.
CustomExceptions bliver grebet i vores forskellige Command klasser hvor de bliver lagt ned i en attribut ved navn "feedback", som så vil blive skrevet ud på siden.
NoAccesException kan opstå hvis man ikke har tilladelse til at tilgå den command man prøver at lave. Der vil så blive kastet en NoAccesException som bliver grebet på vores FrontController, der så vil sende brugeren hen til en anden side.

Brugerinput validering

Der bliver lavet brugervalidering både på front og backend.
På frontend bruger vi lidt forskellige ting, men primært HTML ting som 'required' hvis et felt skal være udfyldt.
Hvis man f.eks. skal indtaste et postnummer, er der blevet brugt regular expressions til at sikre det man taster ind er et gyldigt dansk postnummer: "\d{4}"
det ovenstående statement altså "\d{4}" siger at man kun vil modtage 4 digits, og ikke nogle bogstaver, tegn eller mindre/flere end 4 digits.
Hvis man kigger på denne expression som validere telefonnumre og prøver at bryde den ned så den er til at forstå: "[+]?([\d][-]?){4,19}"

- [+]? siger at nummeret må starte med 0 eller 1 '+' tegn.
- ([\d][-]?){4,19}
  - [\d] siger der må være et digit
  - [-]? siger der må være enten et space eller en bindestreg 0 eller 1 gang.
  - {4,19} siger at den sekvens der står før må forekomme 4-19 gange

Så den siger: Der må være 0 eller 1 '+' tegn, efterfulgt af 4-19 ganges: digit med mulighed for et space eller bindestreg efter.

For at sikre os at man ikke kan lave en ordre på et skur der ikke giver mening, f.eks. 3x3 meter carport med 4x4 meter skur, er der lavet javascript(jQuery), der hver gang man ændre størrelsen af skuret, sørger for at man ikke kan lave et skur der er for stort. Hvis dette skulle gå galt bliver det også tjekket på backenden.
Fordi vi validere på frontenden antager vi at det data som bliver sendt over til vores backend er acceptabel, derfor prøver vi bare at bruge den data som er sendt med, hvis af en eller anden grund det data vi får med ikke er ordentligt vil der blive kastet en 'CustomException' og aktionen vil blive stoppet.

Login sikkerhed

Sikkerhed i forbindelse med login er lavet med BCrypt, som både står for salt og hashing af passwords.
Hvis man prøver at logge ind, vil vores program se om det indtastede brugernavn findes i databasen, hvis det gør det sammenligner man så det indtastede password med det hashed password fra databasen.

Brugertyper til databasen

Vi har to forskellige brugertype til vores database.
Den første er den vi bruger på selve sitet ved navn: "Fog" denne bruger har kun de rettigheder som den har brug for og ikke andet.
Vores anden bruger er kaldt: "fogTest" denne bruger kan gøre alt of har ikke nogen restriktioner.

7 Udvalgte kodeeksempler

```
PartLine spear() throws CustomException {
    Material material = findBestMat(spaerLength,
       findMaterials("spær"));
    double amount = Math.ceil((length - shedLength-60)
        / 89);
    if (hasShed) {
        amount += Math.ceil(shedLength / 110) - 1;
    }
    return new PartLine(material, (int) amount);
}
```

reference: Source Packages/logicLayer/TallCarPortList.java (129 - 139)
Først findes det Material som har mindst spild længde i forhold til længden.
Antallet findes derefter ved at tage længden af carporten minus det udhæng der skal være og dividere med den afstand der skal være mellem spærene.
hvis der er et skur bliver samme beregning kørt igen men denne gang med 110 som mellemrum.
der returneres en PartLine som indeholder det Material som har den rigtige længde og et antal.

```
private List<Material> findMaterials(String name)
   throws CustomException {
    List<Material> materials = new ArrayList<>();
    for (Material material : allMaterials) {
        if (material.getName().equals(name)) {
            materials.add(material);
        }
    }
    if (materials.size() == 0) {
            throw new CustomException("ingen␣matriel␣
               passer");
    }
    return materials;
}
```

reference: Source Packages/logicLayer/TallCarPortList.java (500 - 515)
Der kigges på alle Material som ligger i allMaeterials som er en attribut der indeholder alle Material og for alle Material der har samme getName som name bliver de gemt i en List, hvis der ikke bliver fundet nogen Material med det rigtige name

bliver der kastet en CustomExeption ellers bliver der returneres den List som indeholder alle de Material med det rigtige name.

```
private Material findBestMat(double length, List<
    Material> list) throws CustomException {
     double best = Double.MAX_VALUE;
     double wasted;
     Material mat = null;
     if (length <= 0) {
         throw new CustomException("mærklige␣mål");
     }
     for (int i = list.size() - 1; i >= 0; i--) {
         wasted = 1 - (length / list.get(i).getSize())
            % 1;
         if (wasted == 0) {
             return list.get(i);
         } else if (wasted < best) {
             mat = list.get(i);
             best = wasted;
         }
     }
     return mat;
}
```

reference: Source Packages/logicLayer/TallCarPortList.java (446 - 469)
Her bliver der taget en længde og en List over Material, wasted er hvor meget Material går til spilde ved at lave den length ud af det Material, så hvis wasted er 0 så er Materialet perfect længe og der vil være ingen spil så det returneres, ellers bliver det Material med lavest wasted returneret.

8 Status på implementation

Som det kan ses i Appendix B: User Stories er der user stories vi ikke har implementeret, dette er da PO har valgt at nedprioritere dem i forhold til de andre user stories. Vi har derfor ikke lave metoder til at kunne tilføje og fjerne materialer, da det var en del af en af disse user stories.

Vi har lige nu kun sat op så der sendes en email til fog med at der er lavet en ny ordre, vi kunne godt tænke os at der blev sendt flere emails med opdatering så som, at når man lavede en ordre også modtog en bekræftelse der sagde noget i stil med "Tak for din ordre du vil blive kontaktet snarest af en medarbejder". Eller når status bliver opdateret af en medarbejder at man som kunde også modtager en email der fortæller status af ens ordre er blevet opdateret.

Sådan som vores system er bygget om, udregner vi en ny stykliste hver gang den skal vises på siden. Havde vi haft mere tid, havde vi lavet en tabel i databasen, hvor vi ville kunne gemme styklisten for en given ordre med foreign key til ordrens id. Med denne ændring vil vi kunne gøre det muligt for en medarbejder at kunne ændre styklisten i tilfælde af at måden at udregne styklister på ændre sig på, så udregningen ikke længere er rigtig.

9 Test

Vores testmiljø består af JUnit test og et plugin som hedder 'TikiOne JaCoCoverage'.

Testene er lavet så vi får det forventet output ud af vores input. Dette betyder at vi f.eks. gerne vil have et element fra en tabel med et specielt id eller navn, men også at få en fejl hvis vi prøver at hente eller indsætte værdier som ikke stemmer overens med databasen eller objektet. Hvis vi eksempelvis vil indsætte en negativ pris på et objekt, eller hente information fra ting som ikke eksisterer vil der blive kastet en CustomException.

Vores testmiljø er en kombination af black box og white box testing. Vores JUnit er black box testing da vi giver et input og ser om outputtet stemmer overens med med det forventet. På denne måde ser vi ikke ind i koden så det forventet resultat kan kun blive baseret på hvad programmet skulle kunne og ikke hvad det kan.

Code Coverage kan ses i Appendix C: Code Coverage: Figure 12.
Vores white box testing er et code coverage, som giver mulighed for at se hvor meget kode der er testet. Dette plugin er specielt godt til at se om man har testet forskellige udfald i metoder som 'ifs', 'Exceptions' mm.

Vores plugin tester hvilke linjer der er blevet testet i koden, hvilket betyder at dele i metoden ikke kan blive testet, da det f.eks. kræver en SQL syntaksfejl eller en Connection der er lig null eller forkert sat op. If-statementet i vores try-catch ser om der er indhold i resultsettet, og hvis dette ikke er sandt vil vores preparedstatement ikke kunne kaste en exception da vi ikke kommer ind til vores 'ps.get...()'. Derfor er vores 'catch(SQLException | ClassNotFoundException ex)' ikke testet i vores Mapper klasser, ifølge vores code coverage plugin.

Vi har primært haft fokus på at få testet vores Mapper klasser da alle metoder i dem indeholder try-catch håndtering. Disse test er koblet til en database kaldet 'sem2examTest', da vi ikke skal ændre i den aktuelle database hvor der kunne opstå konflikter, med resultat i at applikationen ikke ville kunne fungere.

Vores test database kalder den aktuelle for hver test i vores Mapperer. Dette gøres i en @Before metode kaldet 'setup()'. I metoden opretter vi en forbindelse til vores test database ved hjælp af en 'setConnection()' metode. Metoden bruger samme attributter i alle test, da de er nøglerne til vores test database.

Databasen er automatisk koblet op og opdateret til den aktuelle database 'sem2exam'. Test databasen består af to varianter af alle tabeller, da vi skal bruge en der opdatere sig selv, og en der kan blive ændret i uden at have indflydelse på nogle andre tabeller.

For hver gang vi kalder 'setup()' metoden fjerner vi den givne test tabel hvis den eksistere for at skabe en ny og derefter indsætte værdierne fra den statiske tabel.

CONNECTION ATTRIBUTES

```
private static Connection testConnection;
private static final String USER = "fogTest";
private static final String USERPW = "password123";
private static final String DBNAME = "sem2examTest";
private static final String HOST = "159.89.9.144";
```

reference: Test Pakages/storageLayer/MaterialMapperTest.java (20 - 24)
I ovenstående reference kan man se at vi kalder 'sem2examTest' fra samme host-ip som i vores connection til vores rigtige database. Dette gøres af brugeren 'fogTest'.

TEST MED ASSERT

```
@Test
public void testGetTool() throws Exception {
        int tool_id = 9;
        Material meterial = ToolMapper.getTool(tool_id
            );
        String expResult = "4,5x70 mm. skruer";
        String result = meterial.getName();
        assertEquals(expResult, result);
}
```

reference: Test Pakages/storageLayer/ToolMapperTest.java (81 - 88)

TEST MED EXCEPTION

```
@Test(expected = CustomException.class)
public void testGetTool_posOutOfBoundsID() throws
    Exception {
        int tool_id = Integer.MAX_VALUE;
        ToolMapper.getTool(tool_id);
}
```

reference: Test Pakages/storageLayer/ToolMapperTest.java (112 - 116)
Da vi har haft meget fokus på håndtering af exceptions er mange af vores test lavet til se hvorledes der bliver kastet en exception eller ej. Vores exceptions sikre os at forkerte inputs bliver håndteret rigtigt så brugeren ikke får et forkert output.

TESTET GET METODE

```
public static Material getTool(int tool_id) throws
   CustomException {
        PreparedStatement ps = null;
        try {
                Material material = new Material();
                Connection con = Connector.connection
                   ();
                String SQL = "SELECT␣*␣FROM␣tools␣
                   WHERE␣tool_id␣=␣?";
                ps = con.prepareStatement(SQL);
                ps.setInt(1, tool_id);
                ResultSet rs = ps.executeQuery();
                if(rs.first()) {
                        material.
                        setId(rs.getInt("tool_id")).
                        setName(rs.getString("name")).
                        setPrice(rs.getInt("price")).
                        setUnitSize(rs.getInt("
                           unit_size"));
                        return material;
                }
        } catch(SQLException | ClassNotFoundException
           ex) {
                throw new CustomException( "Kunne␣ikke
                   ␣hente␣information" );
        } finally {
                closeStatement(ps);
        }
        throw new CustomException( "Kunne␣ikke␣hente␣
           information" );
}
```

reference: storageLayer/ToolMapper.java (59 - 87)

Ovenstående test reference kaster en exception, grundet at værdien på 'tool_id' er højere end det forespurgte element i databasen. Exception bliver kastet hvis ResultSettet ikke giver nogle værdier tilbage, og da dette id ikke eksistere vil der ikke komme noget ResultSet. Havde id´et været i databasen ville der blive returneret et Material object. Hvis der var indhold i ResultSettet ville metoden returnere et Material object med de forspurgte værdier.

Test Reflektering

En måde at arbejde med test på er at at lave testen først hvorefter man laver koden til at håndtere den. Dette kaldes Test Driven Development (TDD), hvilket hjælper med at opretholde metoders egentlige formål. I stedet for at skabe en metode der lever op til kriterierne, men også indeholder overfladiske elementer, kan man med TDD udarbejde en metode efter hvad testen skal kunne gennemføre.
Vi har testet mange parameter, en ting vi kunne have testet var at indsætte en for lang String i databasen. Eksempelvis ville man kunne indsætte en String på 20 karakterers længde i en kolonne i databasen hvor der er blevet sat en begrænsning til 12. Dette ville udlede i en SQLException som derefter ville kaste en CustomException.

10 Process

10.1 Arbejdsprocessen faktuelt

SCRUM-master rollen

Nikolaj blev hurtigt tildelt titlen som SCRUM-Master da han altid er den første der møder og altid har styr på datoer, tidsfrister osv. Hvis nogen havde idéer var det Nikolaj, som i sidste ende afgjorde om det blev det ene eller det andet.

Et eksempel på et PO-møderne

I et af møderne nåede vi ikke en userstory i vores sprint, hvilket resulterede i at vi skulle lave et spike. Dette havde vi forberedt inden mødet, da vi vidste at det ville blive taget op. Vores userstory handlede om at lave tegninger af carporten hvilket vi kun havde nået at gøre for en type. Spiket blev revurderet og sat som høj prioritet til næste sprint af vores PO.

Sprints og hvilke user stories der blev arbejdet på

Første sprint 4-23 til 4-26

I dette sprint arbejdede vi med følgende user stories.

- As a customer I want to be able to contact an employee so that I can get counseling
- As a customer I want to be able to order a carport
- As a customer I want to be able to see my orders and the status of my orders so I know what is going on
- As an employee I want to be able to edit the status of an order so that I can keep the customer updated

Andet sprint 4-26 til 5-3

Vi arbejdede med følgende user stories i dette sprint:

- As an employee I want to be able to cancel an order
- As a customer I want good pictures so that I can see what I'm buying
- As an employee I want to be able to create a part list for a carport

Vi fik lavet så man kunne annullere en ordre, men vi nået ikke de to andre user stories.
På dette tidspunkt havde vi fået lavet 2/4 tegninger. Der var lavet tegninger til carport med fladt tag set oppefra og fra siden, men ikke for carport med rejsning.
Vi var godt klar over at styklisten ville tage ret lang tid og vi nok ikke ville være helt færdige med den efter dette sprint.

Tredje sprint 5-3 til 5-14

Dette sprint var beregnet til at få lavet de to user stories vi ikke nået i foregående sprint færdigt:

- As a customer I want good pictures so that I can see what I'm buying
- As an employee I want to be able to create a part list for a carport

Vi fik lavet tegninger til både carporte med fladt tag og tag med rejsning, men var ikke helt tilfredse med dem, og blev enige om at give dem et sidste sprint så de kunne blive helt som vi gerne ville have dem.
Styklisten til carport med fladt tag var blevet færdig, men vi manglede stadig den til carport med rejsning.

Fjerde sprint 5-14 til 5-18

Vi blev færdige med de her user stories:

- As a customer I want good pictures so that I can see what I'm buying
- As head of department I want to be able to modify available material

I dette sprint blev vi helt færdige med tegningerne, og fik lavet mål på dem.
Vi fik også lavet den nye user story som sagde man skulle kunne modificere materialet på siden.
Vi blev ikke færdig med styklisten:

- As an employee I want to be able to create a part list for a carport

Vi havde på nuværende tidspunkt fået lavet styklister til begge slags carporte, men der var et par småfejl som vi ikke var helt tilfredse med, som vi hurtigt ville kunne få lavet færdigt, og blev enige om at gøre det i vores sidste sprint.

Femte sprint 5-18 til 5-24

Fordi vi blev færdige med styklisten i dette sprint fik vi lavet en hel del user stories:

- As an employee I want to be able to create a part list for a carport
- As an employee I want to be able to see a price based on the parts list.
- As a customer I want to see the price so that I know if I can afford the product
- As an employee I wan't the admin pages to be locked behind a login so that users can't change things they are not supposed to
- As a salesperson I want to be notified when a new order has been placed so that I can inspect it
- As a customer I want to be able to see when I made my order in a user friendly format

Da styklisten blev helt færdig var det super nemt at få lavet så prisen blev udregnet ud fra den, og vi kunne så også lave at man kunne give en personlig pris da vi nu faktisk havde en beregnet pris at gå ud fra.
På dette tidspunkt havde vi også fået lavet alle de .jsp sider som skulle laves helt færdige så vores færdige design kunne blive godkendt.
Vi havde allerede lavet login til employee i et tidligere sprint, så det var hurtigt at få lavet et system som krævede man var logget ind for at tilgå bestemte sider.
Vi havde også som bonus mål at der skulle sendes en email til fog hver gang der blev lavet en ordre forespørgsel, og det fik vi også nået.

Standup møder

Vi afholdte daglige standup møder hvor vi hurtigt gik igennem det som var blevet lavet siden sidst vi havde set hinanden. Det kunne f.eks. være over en weekend eller bare fra dag til dag. Efter det kiggede vi på hvilke opgaver der skulle laves som det næste, og diskuterede kort om vi var enige om hvordan de skulle løses, og hvem der havde tid og lyst til at gå i gang med dem.
Selve møderne blev afholdt på den måde at vi havde en tavle hvor vi kunne skrive ned hvad der skulle laves og skrive navne på de forskellige ting. Vi kunne også tegne de forskellige problemstilling der kunne opstå og blive enige om ting mens vi tegnede dem. Det kunne f.eks. være da vi blev enige om hvordan der skulle tegnes/beregnes stolper, hvor det var meget smart at kunne se hvordan det hele hang sammen.

Retrospectives

Efter hvert PO-møde brugte vi ca. 10 minutter på lige at tale om hvad der blev nævnt til møderne og om der var noget som var ekstra vigtigt at tage med fra dem. Vi havde mange dage hvor vores møder lå meget tidligt på dagen, så vi mødte direkte til dem. De dage blandede vi vores retrospectives sammen med vores standup møder, da vi synes dette gav god mening da de efter et sprint meget handlede om det samme, altså hvad vi ikke havde nået eller hvad der lige pludselig var meget vigtigt at få lavet færdigt først.

10.2 Arbejdsprocessen reflekteret

SCRUM-master rollen

Vi synes at SCRUM-master rollen fungere helt fint, der var ikke rigtigt nogle problemer i den.

Retrospectives

De væsentligste emner til vores retrospectives var lige at tale de ting igennem vi fik at vide til det PO-møde vi lige havde været til, og være sikker på alle havde forstået de ting som blev sagt på samme måde. Så talte vi også meget om der var noget der efter mødet var super vigtigt at komme i gang med.

Nedbrydning af user stories i tasks

Vi kunne godt have problemer med at nedbryde en user story i tasks.
Der kunne man f.eks. kigge på:

- As an employee I want to be able to create a part list for a carport

hvor vi havde følgende tasks:

- Stykliste for carport med fladt tag
- Stykliste for carport med rejsning

Disse skulle have været delt op i mange flere task som blandt andet kunne være:

- Udregning af spær
- Udregning af stolper
- Udregning af skruer
- osv.

Estimering

Det var meget svært at estimere hvor lang tid det ville tage at få lavet styklisten og tegningerne. Vi havde sat begge to til vores højeste estimat hvilket var en uge, så det kunne godt være at de som også nævnt tidligere skulle have været delt op i flere forskellige user stories eller måske vi skulle have sat vores øvre grænse på et estimat op.
Ellers var vi var meget gode til at få lavet de forskellige user stories på den anslåede tid.

Vejledning og PO-møder

Der har ikke været så mange problemer ved hverken vejledning eller PO-møder, vi har generelt været meget glade for den feedback som vi har fået, der var dog til et af vores første tekniske reviews hvor vi fik at vide, at vi skulle lave det på en måde så kunder ikke havde en bruger, men blot lavede en bestilling. Dette er noget vi er blev spurgt meget ind til af andre vejleder senere hen til andre tekniske reviews. Hvor de spurgte hvorfor vi havde "valgt" at gøre det på den måde. Selvom det ikke rigtigt var et valg, men noget vi havde fået at vide. Det skal dog siges at hvis det havde været op til os havde vi nok også lavet det sådan at man ikke har en bruger, men ens ordre bare er tilknyttet en email.

Arbejdsrytme

Vi fandt super hurtigt en god rytme for hvordan vi arbejde. Vi gjorde det meget på den måde at hvis der var noget man gerne ville lave, fik man lov til at lave det. Ved at gøre det på den måde fik folk lov at lave de ting som kunne holde dem motiveret, men samtidig også noget de var gode til. Vi mødte også op skolen hver dag da vi synes det var en god måde at arbejde på hvor vi kunne hjælpe hinanden eller spørge ind til noget hvis man var usikker eller ville diskutere forskellige løsningsforslag.

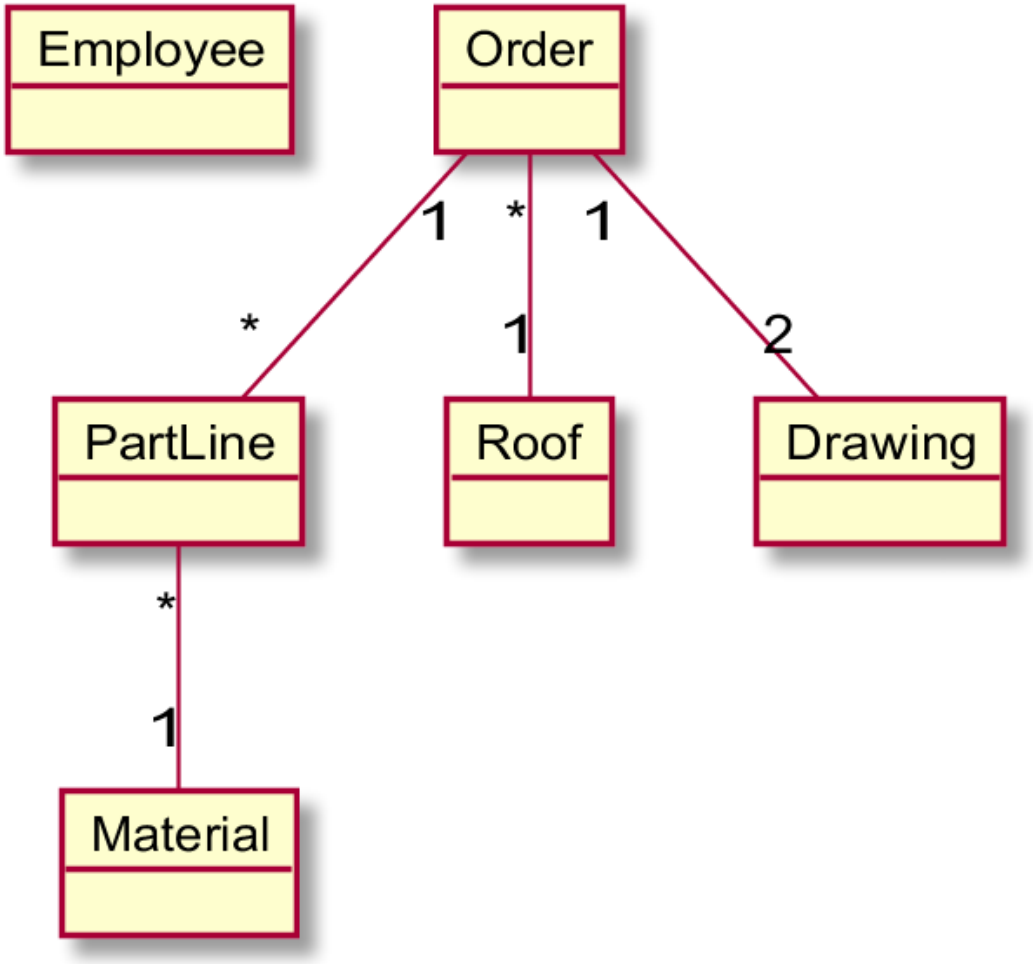
11 Appendix

List of Figures

11.1 Appendix A: Diagrammer

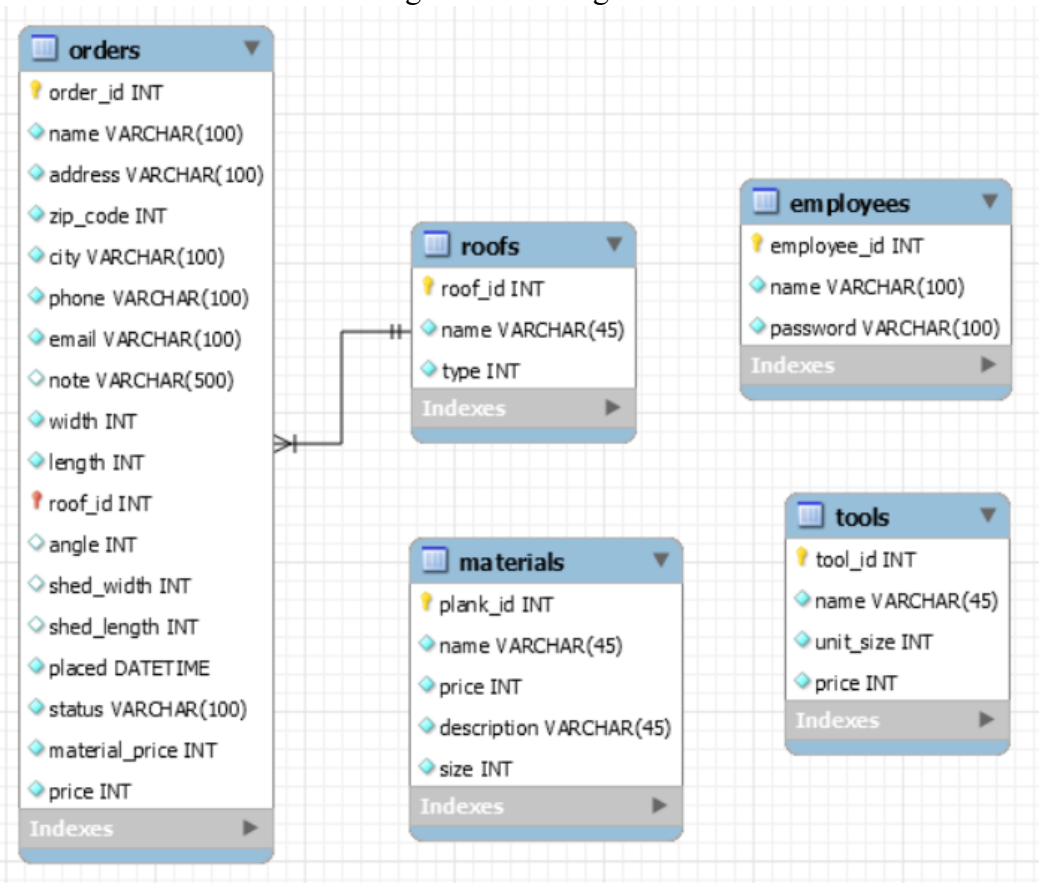
11.1.1 Domæne model

Figure 1: Domæne model



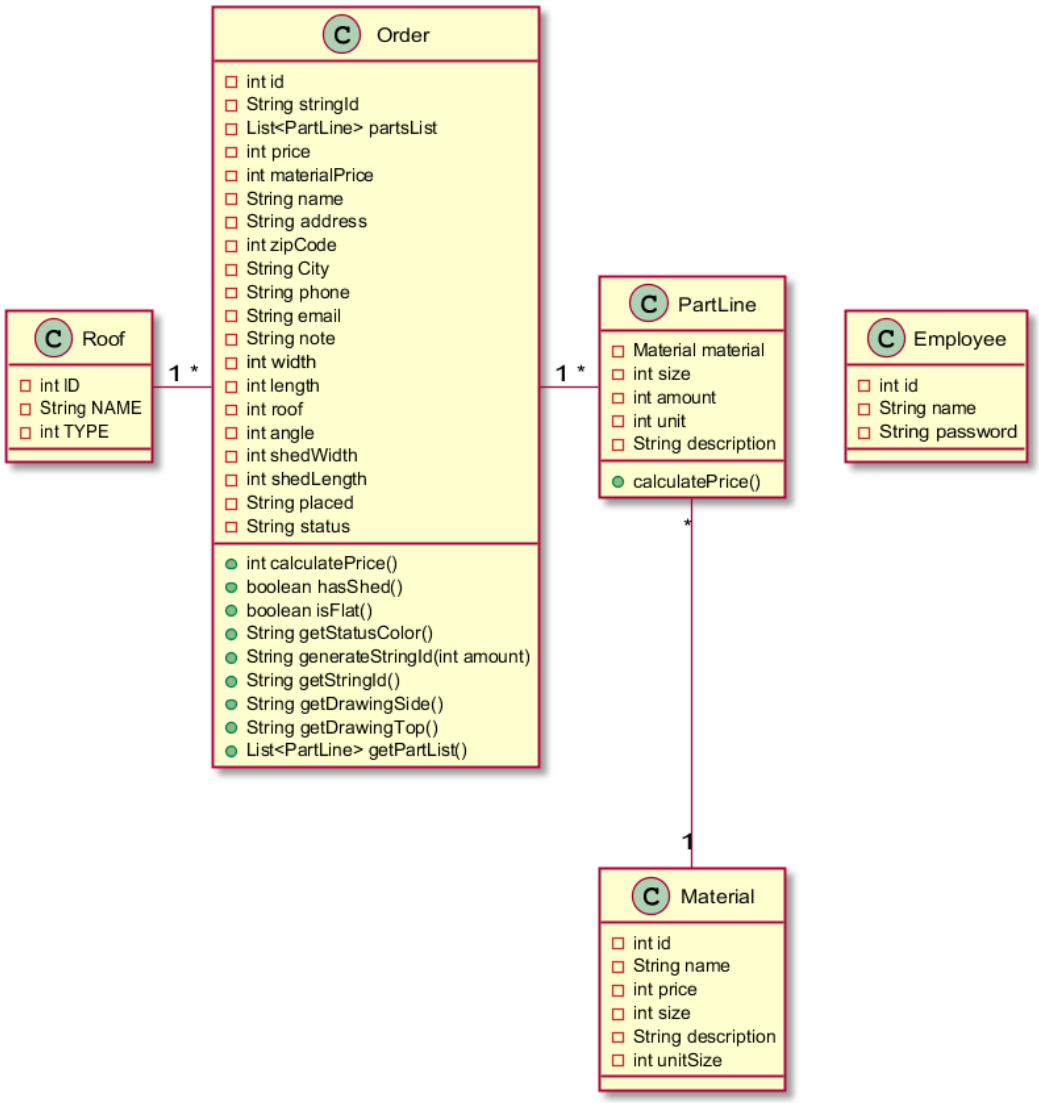
11.1.2 ER diagram

Figure 2: ER diagram



11.1.3 Klassediagram

Figure 3: Klassediagram



11.1.4 Navigationsdiagram

Figure 4: Navigationsdiagram

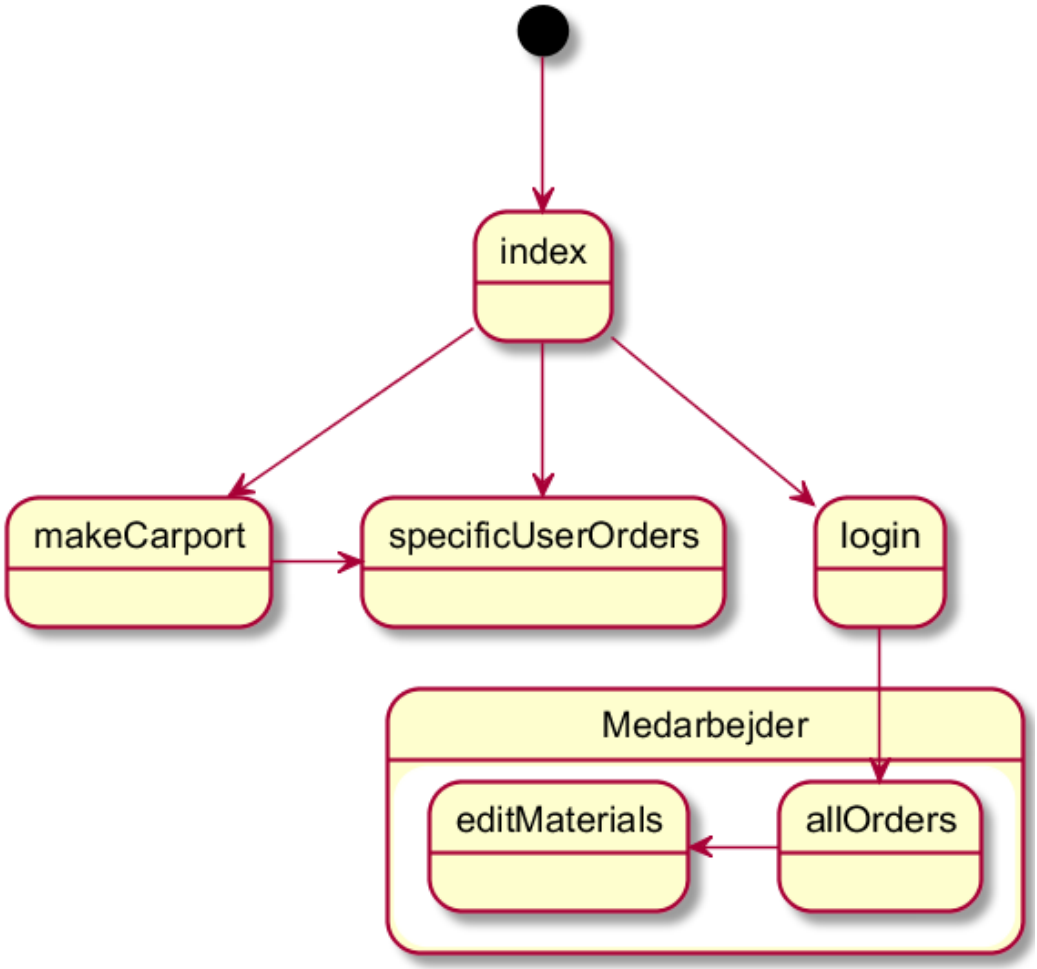
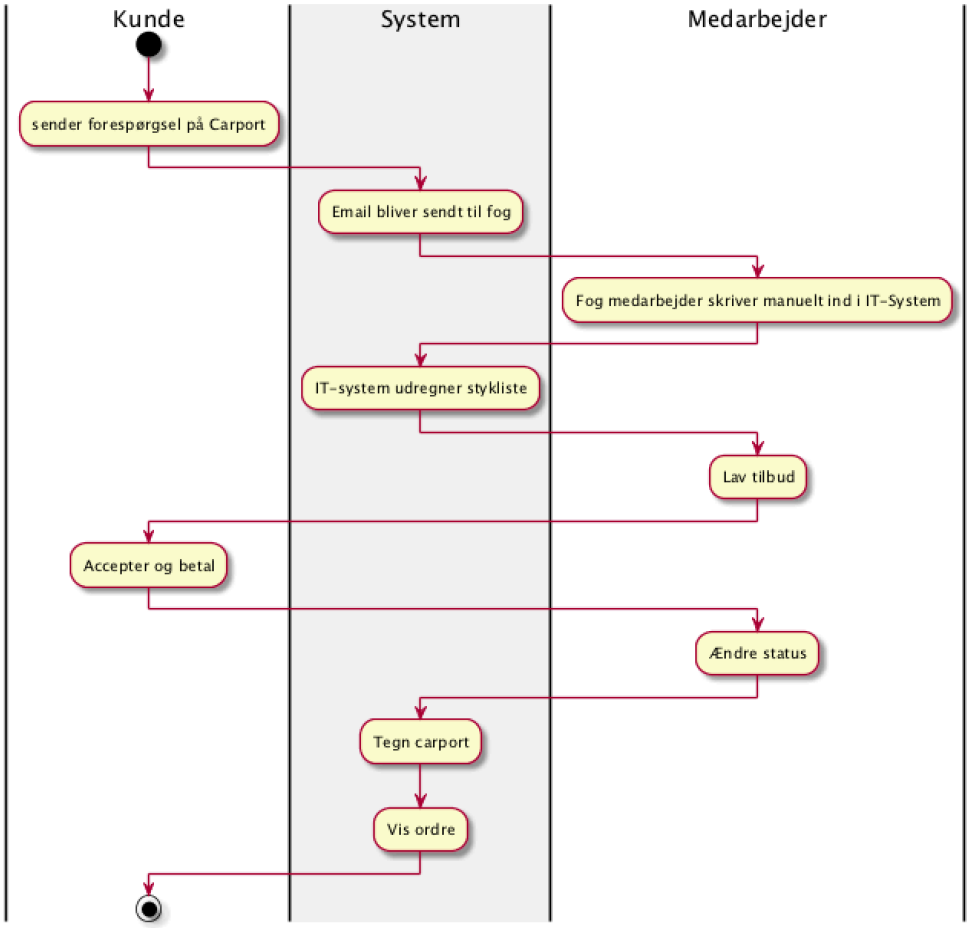
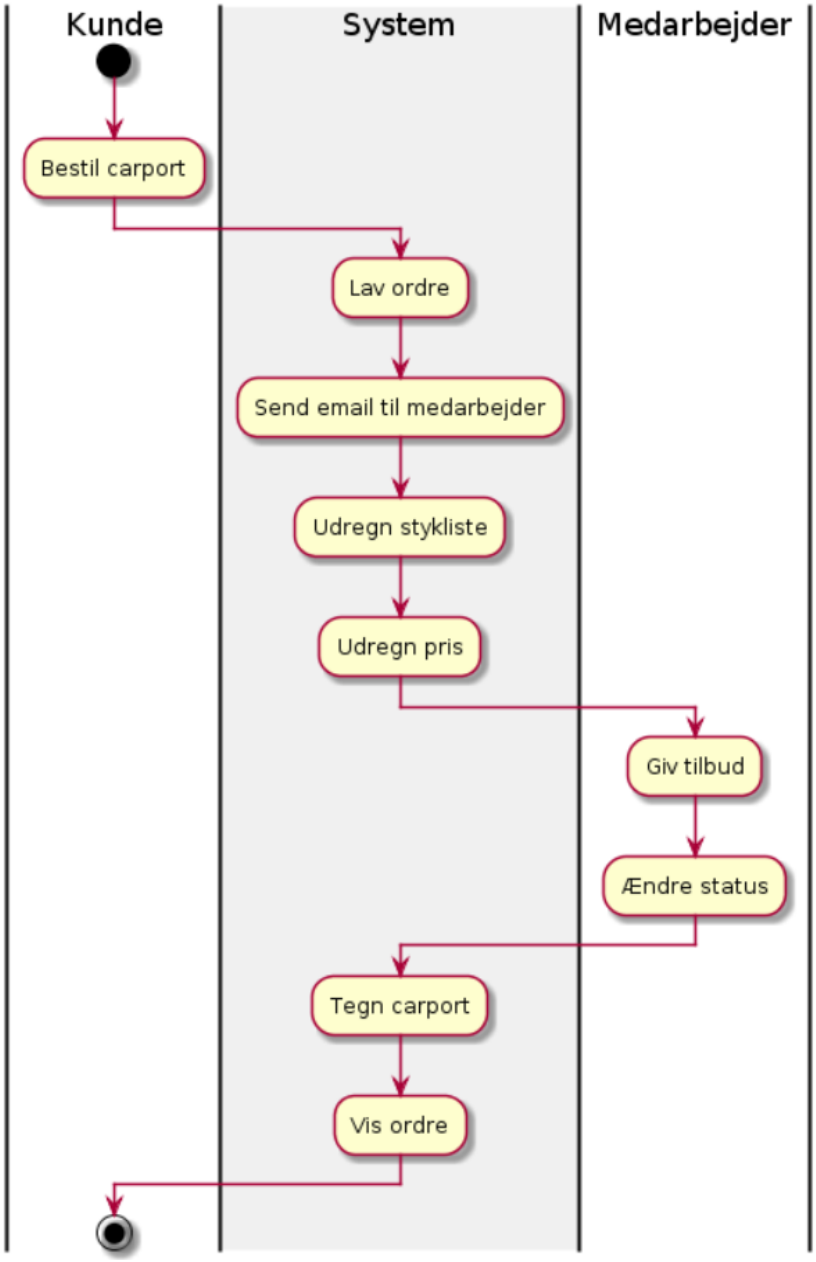


Figure 5: Aktivitetsdiagram: as-is



11.1.5 Aktivitetsdiagrammer

Figure 6: Aktivitetsdiagram: to-be



11.1.6 Sekvensdiagrammer

Figure 7: Sekvensdiagram: Placér ordre

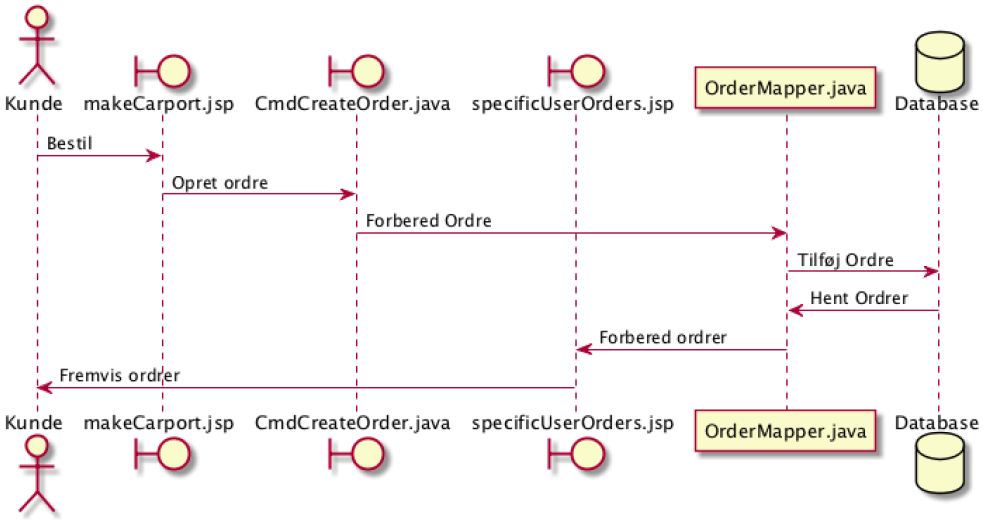


Figure 8: Sekvensdiagram: Se ordre

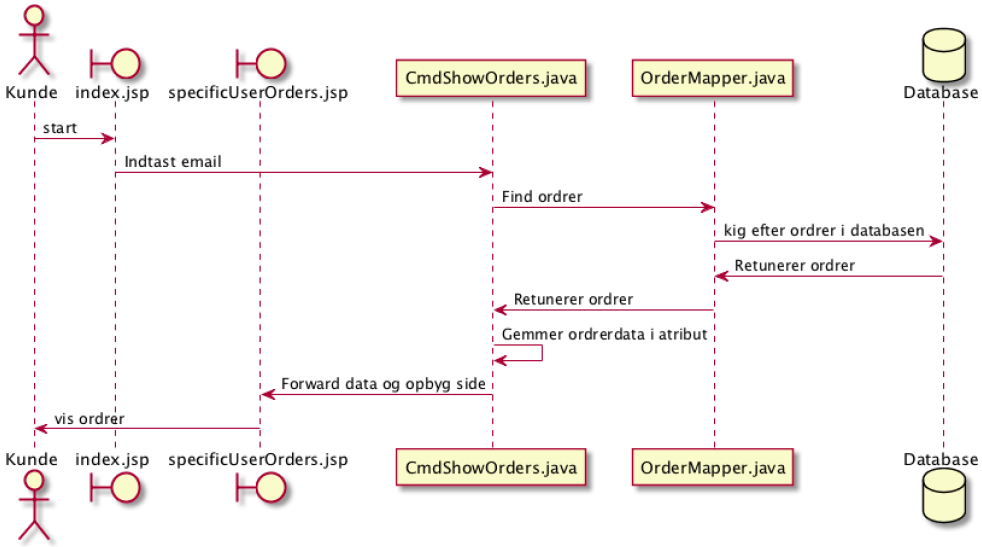


Figure 9: Sekvensdiagram: Opdater ordre

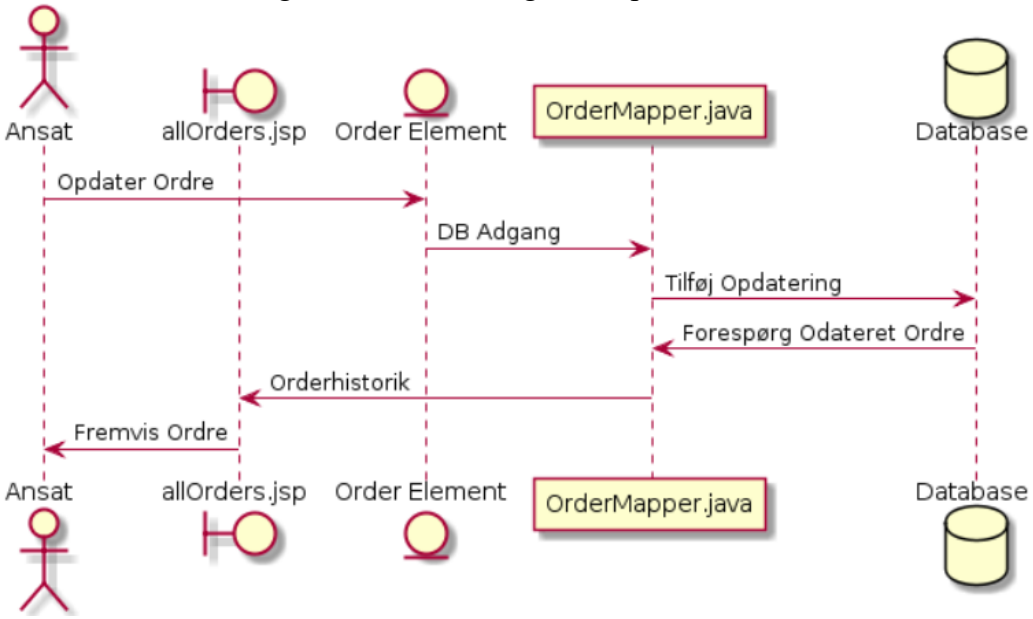


Figure 10: Sekvensdiagram: Inspicér ordre

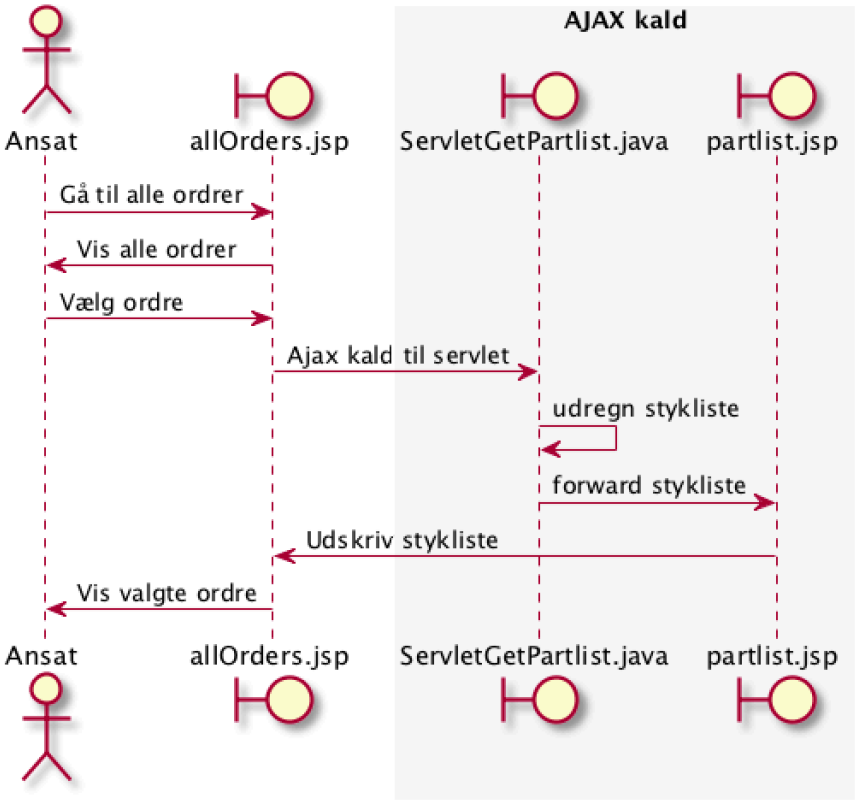
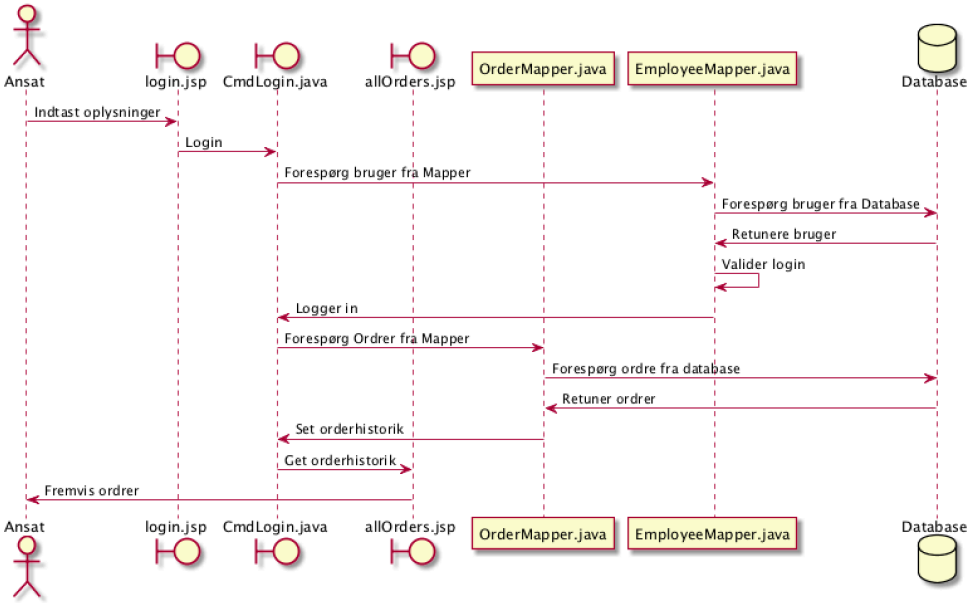


Figure 11: Sekvensdiagram: Vis alle ordrer



11.2 Appendix B: User Stories

11.2.1 Implementeret

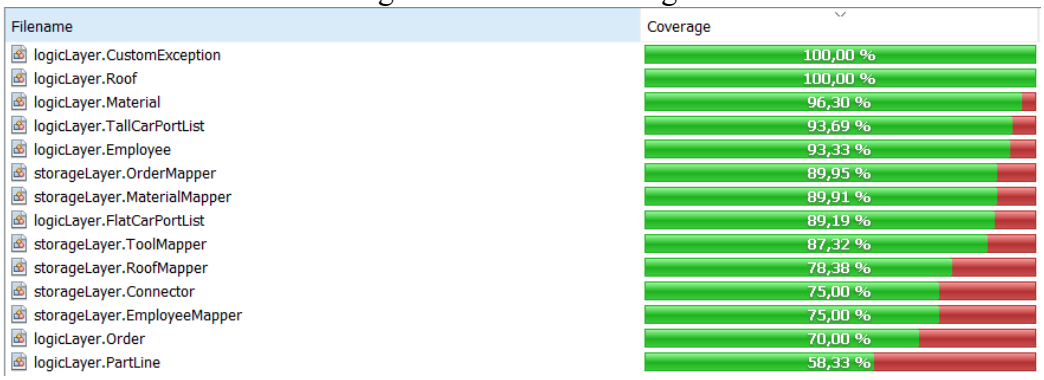
- Som bruger af systemet vil jeg have et brugervenligt design så jeg kan finde rundt
- Som kunde vil jeg have mulighed for at kontakte en medarbejder så jeg kan få rådgivning
- Som kunde vil jeg have mulighed for at kunne se mine ordre og deres status så jeg kan holde mig opdateret
- Som kunde vil jeg kunne bestille en carport så jeg kan få en carport
- Som kunde vil jeg kunne se en tegning af min carport så jeg kan se hvad det er jeg køber
- Som kunde vil jeg kunne se hvornår jeg har afgivet min ordre i et læseligt format så jeg kan se hvornår jeg har bestilt min carport
- Som kunde vil jeg kunne se prisen på min carport så jeg ved om jeg har råd til den
- Som medarbejder vil jeg have besked når en ny ordre er blevet placeret så jeg hurtigt kan hjælpe kunde med at få sin carport
- Som medarbejder vil jeg have medarbejderspecifikke operationer gemt væk bag et login så kunder ikke har adgang
- Som medarbejder vil jeg have mulighed for at ændre status for en ordre så kan holde kunden opdateret
- Som medarbejder vil jeg kunne se en vejledende pris baseret på en styklisten så jeg har et udgangspunkt for at kunne give kunden et tilbud
- Som medarbejder vil jeg kunne udregne en stykliste så jeg ved hvilke materialer der skal bruges
- Som medarbejder vil jeg være i stand til at annullere en ordre så det er muligt at annullere sin ordre
- Som medarbejder vil jeg være i stand til at ændre på tilgængelige materialer så jeg ved hvilke materialer jeg har

11.2.2 Ikke implementeret

- Som kunde vil jeg kunne opdatere mine informationer så jeg kan få min carport leveret til den rigtige adresse
- Som kunde vil jeg kunne se et estimat for hvornår jeg kan få min carport så jeg ved hvornår jeg får min carport
- Som medarbejder vil jeg kunne lave et nyt kodeord hvis jeg glemmer mit kodeord så jeg stadig kan arbejde hvis jeg glemmer mit kodeord
- Som medarbejder vil jeg kunne slette og tilføje materialer så jeg kan tilpasse mig ændringer i tilgængeligt materiale
- Som medarbejder vil jeg kuune ændre en stykliste så jeg kan være sikker på at den er rigtig

11.3 Appendix C: Code Coverage

Figure 12: Code Coverage



Filename	Coverage
logicLayer.CustomException	100,00 %
logicLayer.Roof	100,00 %
logicLayer.Material	96,30 %
logicLayer.TallCarPortList	93,69 %
logicLayer.Employee	93,33 %
storageLayer.OrderMapper	89,95 %
storageLayer.MaterialMapper	89,91 %
logicLayer.FlatCarPortList	89,19 %
storageLayer.ToolMapper	87,32 %
storageLayer.RoofMapper	78,38 %
storageLayer.Connector	75,00 %
storageLayer.EmployeeMapper	75,00 %
logicLayer.Order	70,00 %
logicLayer.PartLine	58,33 %