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

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

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

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

Cupcake Projekt Rapport

Gruppe: CRMD

Vinter 2018

Indholdsfortegnelse

[Indledning 2](#_Toc532741382)

[Baggrund 2](#_Toc532741383)

[Valg af teknologi 3](#_Toc532741384)

[Krav 3](#_Toc532741385)

[Overordnet beskrivelse af virksomheden 3](#_Toc532741386)

[Arbejdsgange der skal IT-støttes 4](#_Toc532741387)

[Scrum userstories 4](#_Toc532741388)

[Modeller og diagrammer 5](#_Toc532741389)

[Domæne model 5](#_Toc532741390)

[Forklaring til domæne model 6](#_Toc532741391)

[Begrundelse for vores valg af 1-1 relation mellem order og carport 6](#_Toc532741392)

[Begrundelse for vores valg af 1-1 relation mellem customer og address 6](#_Toc532741393)

[Begrundelse for vores valg af 1..\*-1..\* relation mellem storage/shop og employee (?) 7](#_Toc532741394)

[Konsistentitet 7](#_Toc532741395)

[Klasse diagram (burde vi slette denne?) 7](#_Toc532741396)

[EER diagram 8](#_Toc532741397)

[Navigationsdiagram 10](#_Toc532741398)

[Sekvens diagram 11](#_Toc532741399)

[Særlige forhold 12](#_Toc532741400)

[Udvalgte kodeeksempler 12](#_Toc532741401)

[Status på implantation 13](#_Toc532741402)

[Test 13](#_Toc532741403)

[Process 14](#_Toc532741404)

[Arbejdsprocessen faktuelt 14](#_Toc532741405)

[Arbejdsprocessen reflekteret 14](#_Toc532741406)

[Bilag 14](#_Toc532741407)

# Indledning

Fog træhandel har siden 1990’erne brugt et IT-system udviklet af en tidligere medarbejder til at beregne mål på carporte og styklister hertil. Systemet har tidligere været frakoblet deres hjemmeside og kørt på en gammel maskine. Derfor kan produkterne med priser ikke ændres hvilket har medført at den totale pris har været ubrugelig i mange år, hvilket gør at medarbejderen skal lave ekstra arbejde ved at finde de korrekte produkter. Grundet dette har Fog brug for et nyt IT-system, der er bedre integreret på deres nuværende webside og som er mere fleksibelt i forhold til prisstigninger og ordrestyring.

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

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

## Baggrund

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

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

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

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

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

Der skal også være en administrator del, hvor fra Johannes Fogs ansatte kan opdatere produkter, med priser, størrelser, producenter mv. Samt tilføje og slette produkter. En administrator skal også kunne behandle en kundes ordre.

## Valg af teknologi

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

|  |  |
| --- | --- |
| **Programmer:** | **Sprog:** |
| * Netbeans 8.2 * MySQL Workbench 8.0 * Java & Java SE (JDBC) 181 * Apache Tomcat 8.0.27.0 | * Java * HTML * CSS * MySQL 14.14 |

Vi har også brugt front-end framework: Bootstrap 4.1.0

# Krav

* Kunden skal kunne bestille en carport med angivne mål og kontaktoplysninger.
* Det skal være muligt at tilpasse priser på produkterne.
* En kunde skal først modtage stykliste når carporten er købt og betalt, da dette er en del af den service Fog sælger.
* Det skal være muligt for en medarbejder at se og redigere i en foretaget ordre.
* Det skal være muligt for en medarbejder at se og redigere i en foretaget ordre.
* Det skal være muligt for en medarbejder at tilpasse prisen på carporten, i ordren, inden godkendelse.

## Overordnet beskrivelse af virksomheden

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

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

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

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

## Arbejdsgange der skal IT-støttes

Dels skal afsnittet beskrive de overordnede arbejdsgange før og efter IT systemet. Gerne beskrevet med “as-is” og “to-be” som aktivitets diagrammerne.

## Scrum userstories

Dette afsnit skal beskrive de user-stories der er aftalt med product-owner. Det er vigtigt for hver user-story at man kan spore det tilbage til aktivitetsdiagrammerne fra forrige side, og at I har en håndfuld userstories som er lavet fuldt ud, dvs:

* der er beskrevet how-to-demo
* der er brudt ned i tasks
* der er lavet et estimat
* den enkelte story følger INVEST princippet
* at det er klart hvordan den tilfører værdi

Den fulde produkt backlog kan ligge som appendix.

# Modeller og diagrammer

## Domæne model



### Forklaring til domæne model

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

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

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

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

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

Når websystemet bruger de to dele af ordren, bruges de sommetider forskelligt, og derfor giver bedst mening i vores øjne at dele dem op. Ordren indeholder de konkrete informationer fra selve ordren, der vises når brugeren følger sin tidligere ordre, og endvidere når en medarbejderne skal vurdere ordren før afsendelse. Carporten indeholder data, om hvordan den skal bygges, hvordan styklisten generes og hvordan plantegningerne tegnes. Der havde ikke været en væsentlig forskel, hvis vi havde slået dem sammen, men vores tanke bag dette, er igen, Johannes Fogs ønske omkring at kunne integrere og ekspandere sin forretning ind i systemet. I tilfælde af at han i fremtiden integrerer en større del af hans forretning i systemet, vil det betyde, at 1-1 relationen mellem ordre og carport vil blive 1-1..\* relation mellem ordre og produkt, hvor produkt eksempelvis kan indeholde en carport, værktøj og boliginterior.

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

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

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



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

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

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

### Konsistentitet

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

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

## EER diagram



### Normalformer

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

#### Første normalform

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

* Unikke kolonnenavne
* Same type data i kolonnen
* Ingen ’Multivalues’

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

#### Anden normalform

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

* Først normalform
* Ingen partielle afhængigheder

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

#### Tredje normalform

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

* Anden normalform
* Ingen transitive afhængigheder

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



### Autogenereret id

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

### Fremmednøgler og begrænsninger

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



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

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

### Forbedringer

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

## Klasse diagram

## Navigationsdiagram



Det som brugeren oplever er en række websider hvor man kan indtaste oplysninger gå videre til andre sider. I større systemer kan det være svært at bevare overblikket over hvilke sider der er, og hvordan man kommer fra den ene til den anden. Navigationsdiagrammet er beregnet på at vise dette på en mere overskuelig måde. Som led i beskrivelsen af navigationsdiagrammet skal følgende med:

* Oversigts diagrammet. Hvis det bliver for stort må man dele det op. Men det er vigtigt at der er et overordnet diagram.
* Hvis man har benyttet sig af en “fælles navigations bar” i toppen af alle sider skal man forklare det.
* Hvis nogle sider kun kan nåes af nogle brugere (dem der har konto, dem der er logget ind, dem der arbejder i butikken,…), så skal det fremgå.
* Navne på jsp sider skal fremgå, og hvilke servlet der bringer en fra den ene side til den næste.

## Sekvens diagram



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

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

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

# Særlige forhold

Dette afsnit bruges til at beskrive særlige forhold der benyttes i programmet. Det kan f.eks. være:

* Hvilke informationer gemmes i session
* Hvordan håndterer man exceptions.
* Hvordan man på har valgt at lave brugerindput validering
* Hvordan man har valgt at lave sikkerhed i forbindelse med login
* Hvilke brugertyper der er valgt i databasen, og hvodan de er brugt i jdbc
* … andre elementer – i Fog projektet kan det være:
* Tegning
* Stykliste beregner

Husk: det er bedre med 2 linjers dokumentation end ingen.

# Udvalgte kodeeksempler

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

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

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

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

# Status på implantation

Dette afsnit skal liste hvor langt man er nået med implementationen. Typiske ting man kan have sprunget over er:

* Man har ikke nået at lave alle de jsp sider man har med i navigationsdiagrammet.
* Man har ikke nået at lave alle CRUD metoderne til alle tabellerne
* Man har ikke fået stylet sine sider
* Man har fundet en fejl “i sidste øjeblik”, men har ikke haft tid til at rette det. - F.eks. at man har brugt session forkert sådan at man på en af siderne kan komme ind uden at være logget in.
* tests der fejler på afleveringstidspunkt
* … andre mangler

# Test



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

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

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

# Process

Der skal være et afsnit hvor I beskriver jeres arbejsprocess i projekt perioden. Der skal dels være et faktuelt afsnit og et reflektions afsnit.

## Arbejdsprocessen faktuelt

Dette afsnit skal beskrive:

* Hvilke sprints der var, og hvilke user stories der blev arbejdet med.
* Hvem der var scrum master i hvilke dele af projekt perioden. Giv gerne nogle eksempler på hvad scrum master gjorde i udvalgte sprint.
* Et eksempel på et af PO-møderne, hvad der var planlagt fra jeres side, og hvordan det gik.
* Hvordan i afholdt jeres daglige standup møder.
* Hvornår i holdt retrospectives.

## Arbejdsprocessen reflekteret

Dette afsnit skal beskrive jeres overvejelser over hvilke dele der har fungeret godt og hvilke dele der måske er faldet lidt på gulvet. I kan f.eks. beskrive:

* Om scrummaster rollen fungerede, hvilke problemer I så i den, og hvad I gjorde for at rette op på det.
* Hvad der var de væsentligste emner på jeres retrospektiv møder
* Om I havde problemer med at nedbryde user stories i tasks
* Om I var spot-on med jeres estimeringer
* Om der var problemer med vejledningen og PO møderne
* Hvor langt inde i processen I fandt en rytme der var produktiv
* Andre elementer der har at gøre med at forsøge at arbejde i et scrum team

# Bilag