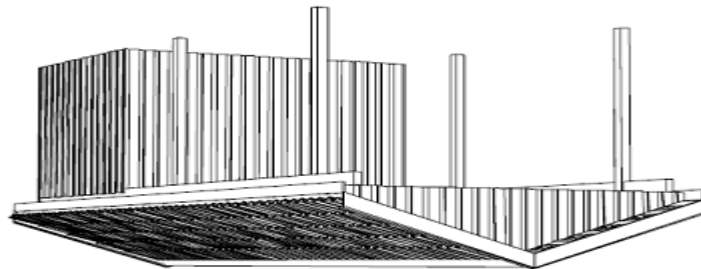


Fog-Carport 2. Semester Projekt

Datamatikeruddannelsen ved Copenhagen Business Academy Lyngby

```
at org.apache.catalina.core.ApplicationDispatcher.doInclude(ApplicationDispatcher.java:51)
at org.apache.catalina.core.ApplicationDispatcher.include(ApplicationDispatcher.java:523)
at org.apache.jasper.runtime.JspRuntimeLibrary.include(JspRuntimeLibrary.java:934)
at org.apache.jsp.index_jsp._jspService(index_jsp.java:149)
at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:438)
at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:396)
at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:340)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:105)
```



Klasse A, Gruppe 2 - Deltagere

Navn	Skole-email	Github brugerprofil
Rúni Vedel Niclasen	cph-rn118@cphbusiness.dk	Runi-VN
Camilla Jenny Valerius Staunstrup	cph-cs340@cphbusiness.dk	Castau
Asger Koch Bjarup	cph-ab363@cphbusiness.dk	HrBjarup
Malte Hviid-Magnussen	cph-mh748@cphbusiness.dk	MalteMagnussen

Projektet blev påbegyndt d. 11/04-2019 og afleveret d. 29/05-2019.

Link til projektet på GitHub - <https://github.com/HrBjarup/Fog-Carport>

Link til JavaDocs (GitHub pages) - <https://hrbjarup.github.io/Fog-Carport/docs/>

Link til projektets IP (online på DigitalOcean Droplet) - <http://bjarup.com>

Antal tegn (inklusive forside, indholdsfortegnelse og bilag):
92994 | Svarende til: 38 normalsider af 2.400 tegn

Indholdsfortegnelse

Indledning	6
Baggrund	6
Teknologivalg	6
Udviklingsværktøjer	6
Teknologier	6
Dependencies og Plugins	7
Frameworks	7
Krav	7
Firma	7
User Stories	8
SCRUM	10
SCRUM-Processen	10
SCRUM-master	11
Sprints	11
Sprint Planning	11
Tasks	11
Sprint Retrospective	11
Daily SCRUM	12
Discord	13
GitKraken	13
Taiga	14
Taiga point-estimering	14
GitHub	14
Github Wiki	15
Readme	15
Branches	15
Issues	17
Pull-Requests	17
Releases	19
Diagrammer	20
Domæne model	20
Domæne model beskrivelse	20
E/R diagram	21

Beskrivelse	22
Tabellerne i databasen	22
Navigationsdiagram	24
Beskrivelse af Navigationsdiagram	24
Sekvensdiagrammer	26
Sekvensdiagram for ViewPartslist	26
Gennemgang af ViewPartslist	26
Sekvensdiagram for ViewSVG	30
Gennemgang af ViewSVG	31
Klassediagram	34
Klassediagram beskrivelse	35
Særlige forhold	35
Session, request og validering	35
FrontController	36
Exceptions	36
DataSource	37
JSTL og EL	38
JSTL	38
EL	39
JavaScript	40
Styling	41
Stykliste algoritmer	41
Løsning af problemstilling 1	42
Løsning af problemstilling 2	43
Robusthed, kobling og kohæsion	43
Indkapsling	43
Kobling	43
Kohæsion	44
Validering af brugerinput	44
Clientside	44
Serverside	44
JavaDoc	45
Status på implementation	45
Opnået systemfunktionalitet	45
Kundens brug af systemet	45
Medarbejderens brug af systemet	45
Overblik over implementeret CRUD i forhold til tabellerne i Databasen	46
Ikke-opnået systemfunktionalitet	46

Kundens brug af systemet	47
Medarbejderens brug af systemet	47
Sikkerhed i forbindelse med login	47
Datatyper	47
Logging	47
Log Levels	48
Stack Traces	48
Database normalisering og relationer	48
Yderligere mangler	49
Mindre fejl og mangler	49
Prioritering	50
Test	50
Prioritering af test	50
Anvendte testmetoder	51
Fordelen ved automatiserede tests	51
Manuelle test	51
Følgende dele af programmet er testet	51
BaseCalc.java	51
RoofFlatCalc.java	52
RoofRaisedCalc.java	53
ShedLogic.java	54
UserMapper.java	54
OrderMapper.java	54
Bilag	55
Bilag 1 - Github og Taiga links	55
Sprint 1	55
Sprint 2	55
Sprint 3	55
Sprint 4	55
Sprint 5	56
Bilag 2 - Sekvensdiagram [Partslist]	56
Bilag 3 - Sekvensdiagram [SVG]	56
Bilag 4 - Navigationsdiagram	56
Bilag 5 - Screenshots af Programfunktionalitet	57
Forside	57
Login	58
Ordreoversigt	59
Ordrevisning	61

Stykliste	64
Tegning	66
Bilag 6 - Bills som det fremgik hos Fog	66
Bilag 7 - Ressourcer	67

Indledning

I vores 2. semesterprojekt beskæftiger vi os med udviklingen af et it-system, baseret på de emner vi har haft undervisning i, i løbet af hele 2. semester.

Herudover har vi selv opsøgt læring i emner som JavaScript og JSTL, for at få lavet en frontend der er væsentlig mere dynamisk. Vi har taget værktøjer i brug som GitKraken for bedre at kunne få det fulde udbytte af versionsstyring.

Processen med SCRUM har været en vellykket udfordring for vores gruppe og har været en af de ting, vi er gladest for at have løst, da det har rykket ved forståelsen af teamarbejde i et udviklingsprojekt, og det er blevet tydeliggjort i hvor stor grad det letter selve programmeringen.

Det endelige resultat af projektarbejdet er, på trods af de mangler der stadig er, noget vi alle fire er stolte over at have leveret og afspejler godt, at vi har været omkring alle de emner, vi har haft undervisning i og lidt til.

Baggrund

Vi er blevet bedt om at udvikle et nyt it-system til virksomheden Johannes Fog. Den del af virksomheden, der skal aftage it-systemet, laver Carporte på specialmål. It-systemet skal erstatte deres gamle it-system og skal derfor have en del af de samme kernefunktioner, samt nogle nye.

De ønsker at kunne modtage og gemme tilbuds-forespørgsler fra deres kunder, indtaste et tilbud til kunden, generere en tegning ud fra kundes mål og genere en stykliste ud fra den endelige ordre. Ansatte skal have et overblik over alle ordrer. Herudover er der et ønske om, at kunden også skal kunne logge ind i systemet og se deres ordre(r). En uddybning af firmaets krav kan ses i rapportens Krav-afsnit.

Teknologivalg

It-systemet er Javabaseret, med en MySQL database tilknyttet. Frontenden er skrevet i HTML, JSTL og JavaScript, og stylingen er foretaget med Bootstrap og custom CSS.

Udviklingsværktøjer

- Netbeans IDE (8.2)

Teknologier

- Linux Ubuntu 18.10 x64
- MySQL Version 8.0.15 for Linux on x86_64 (MySQL Community Server - GPL)
- Java: 1.8.0_181
- JavaEE Web API (7.0)
- Apache Tomcat 8.0.27.0

- Maven 4.0.0

Dependencies og Plugins

- MySQL JDBC (8.0.12)
- Maven Javadoc Plugin 2.9
- Mockito version 1.10.19
- Junit version 4.12
- Hamcrest version 1.3
- Jacoco-maven-plugin version 0.8.3
- HikariCP version 3.3.1
 - Springframework version 2.1.4.RELEASE

Frameworks

- Bootstrap 4.3.1
- JQuery-3.4.0

Krav

Kravene til it-systemet er blevet defineret, dels i en videooptagelse med Johannes Fog hvor der blev redegjort for deres nuværende it-system, dels ved møder med Product Owner (en lærer på studiet). I videoen blev der gennemgået hvilken funktionalitet de havde nu og gerne ville beholde, og hvilken funktionalitet de savnede.

Deres krav blev opsummeret som følgende liste efter videokonsultationen og det er disse krav der efterfølgende er skrevet User Stories ud fra.

Firma

- Admin skal kunne oprette ansatte
- Admin skal kunne opdatere ansattes gemte informationer
- Admin skal kunne slette ansatte
- Admin skal kunne se statistik over ansattes salg
- Admin skal kunne se statistik over en afdelings salg
- Ansatte skal kunne logge ind
- Ansatte skal kunne se alle ordrer
- Ansatte skal kunne markere en ordre som betalt
- Ansatte skal kunne se stykliste for en ordre
- Ansatte skal kunne se tegninger for en ordre
- Ansatte skal kunne se en faktura for en ordre
- Ansatte skal kunne se tilgængelige materialer
- Ansatte skal kunne redigere og slette materialer
- Ansatte skal kunne ændre materialer i en genereret stykliste
- Kunder skal kunne oprette en bruger
- Kunder skal kunne indtaste Carport med ønskede mål og indhente et tilbud uden bruger
- Kunder skal kunne indtaste Carport med ønskede mål og indhente et tilbud med bruger

- Kunder skal kunne se tidligere ordrer hvis de har en bruger
- Kunder skal kunne se stykliste for en ordre hvis de er logget ind
- Kunder skal kunne se tegning for en ordre hvis de er logget ind
- Kunder skal kunne betale for deres ordre
- Kunder skal kunne se en faktura

User Stories

Kundens krav er blevet omformuleret til i alt 26 User Stories, som er blevet prioriteret ugentligt af projektets Product Owner, som repræsenterer kundens interesser.

Derudover skulle hver User Story være forståelig for vores Product Owner. Det vil sige at der ikke må stå tekniske krav i User Stories. Tekniske krav skal holdes nede i tasks.

De User Stories, der er blevet oprettet efter videokonsultation med virksomheden, er blevet godkendt af vores Product Owner. Vores Product Owner har lagt vægt på, i prioriteringen af User Stories, at få kernefunktionaliteten af systemet færdig først. Hen mod det sidste sprint begyndte Product Owner at prioritere login og større omstrukturering af hvad man kan som medarbejder, og hvad man kan som kunde.

Vi har ikke altid været gode nok til at huske at lave Acceptance Criteria for alle vores User Stories. I de første to Sprints af projektet havde vi fokus på at lave User Stories med deres Acceptance Criteria fra begyndelsen af - præsentere dem for Product Owner og så derefter lave tasks, der matchede kriterierne, men i de senere Sprint fik vi mindre fokus på dette, primært på grund af tidsmangel og lavede i stedet de tasks, som vi synes var nødvendige for at implementere den givne User Story. De tasks, vi så lavede, endte med at definere de kriterier, som vi fra begyndelsen af skulle have lavet som en del af den givne User Story.

Uddrag af Acceptance Criteria fra vores [Stykliste Simpel User Story](#) som eksempel:

size: large ✕ dependency: high ✕ + Add tag

Som sælger vil jeg gerne kunne beregne en SIMPEL stykliste på baggrund af kundes ordre, så lagermedarbejderne kan se hvad de skal pakke ned.

Acceptance criteria:

- Som sælger skal jeg kunne indtaste mål (længde, bredde og højde) på carporten
- Det skal være muligt at tilvælge et standard skur, der ikke kan have special-mål
- Ud fra indtastet data skal der genereres en stykliste med alle relevante materialer (dvs. med søm og beslag og alting)
- Sælger skal kunne se den genererede stykliste (styklisten bliver bare vist på en webside (bliver ikke genereret som pdf))

Liste af implementerede User Stories:

- Stykliste Sempel (Sprint 1)
- Stykliste Avanceret (Sprint 2 og 3)
- Tegning Oppefra (Sprint 3)
- Gem Ordre (Sprint 3)
- Login (Sprint 4)
- Kunde Historik (Sprint 4)
- Helptext (Sprint 4)
- Ændre nuværende bruger login (Sprint 5)
- Kundeendt bestillingsform (Sprint 5)
- Kunde skal kunne se bestilling (Sprint 5)
- Prisvisning for kunden (Sprint 5)
- Pris på carport (Sprint 5)
- Design (Sprint 5)
- Pæn visning af tegninger (Sprint 5)
- Stabilt system (Sprint 5)
- Inputvalidering (Sprint 5)
- Fejlbeskeder (Sprint 5)

Liste af User Stories vi ikke nåede:

- Logging
- Tilpas Stykliste
- Se Materialer
- Opdater Materialer
- Opret Regning
- Opret Sælger
- Opdater Sælger
- Sælger Statistik
- Afdelings Statistik

SCRUM

I de følgende afsnit vil vi uddybe vores SCRUM-process, samt de værktøjer vi har anvendt for at understøtte den agile udvikling.

SCRUM-Processen



Agile manifesto¹

Dette er det første projekt, hvor vi har skulle arbejde agilt efter SCRUM-processerne. I tidligere projekter har vi været vant til at arbejde med alt i projektet på én gang og har haft forsøgt at overskue hele projektet på forhånd. Denne gang har vi taget udgangspunkt i kundes krav, opstillet User Stories ud fra disse krav, som så er blevet groft estimeret i forhold til hvor lang tid, vi antog de ville tage at implementere, samt hvor afhængige de var af, at andre dele af systemet var implementeret.

Herefter har vores Product Owner (virksomhedens repræsentant i form af en underviser) prioriteret disse User Stories efter hvad, han gerne vil have færdigt først. Han har lagt vægt på at vi færdiggjorde kernefunktionalteter i systemet først.

Vi har arbejdet med sprints på en uges varighed, og vi har inden hvert nyt sprint estimeret hvor meget tid vi hver især har haft til rådighed til at arbejde i. Efter hvert sprint har vi afholdt retrospektive møder, hvor vi har talt om, hvad der gik godt og skidt i den forgangne uge.

Til at begynde med lå der en del arbejde i at integrere SCRUM i vores arbejdsrutiner. Møderne, planlægningen osv, lød som om, det ville tage meget tid væk fra programmering, første gang vi hørte om det. I det første sprint gik der en del tid med SCRUM-relaterede opgaver, opstilling af krav og backlog, estimering m.m. og efterlod ikke meget tid til programmering, men det har vist sig at være en stor hjælp i resten af projektet.

¹ <http://agilemanifesto.org/>

Dog kunne vi godt have ønsket os at sprints havde en længere varighed, gerne 2-3 uger i stedet for kun 1 uge. At skulle have en større feature klar til at demo for Product Owner på kun 7 dage kunne til tider føles som en presset tidsplan, der var skrøbelig i forhold til uforudsete problemer.

SCRUM-master

Vi blev bedt om at vælge en person i gruppen til at være SCRUM-master og havde mulighed for at skifte personen ud i de forskellige sprints, men vi har følt, det var for stor en opgave at sidde med ene mand og har generelt været glade for en mere demokratisk process, hvor vi har taget beslutninger i fællesskab.

Sprints

Som beskrevet var alle vores sprints af en uges varighed. Product Owner bestemte suverænt hvilke User Stories der blev prioriteret i et sprint, med hensyn til den tid vi havde estimeret de forskellige User Stories til, samt den tid vi havde estimeret, vi kunne arbejde på projektet for det specifikke sprint.

Sprint Planning

Sprint planning mødet delte vi op i to møder. Vi afholdte et internt møde inden vi havde et Sprint planning møde med Product Owner. I det interne møde sørgede vi for at gennemgå vores Product Backlog, altså de User Stories der endnu ikke var implementeret, som Product Owner kunne vælge imellem i sin prioritering. Vi estimerede de øverste User Stories (som var de User Stories Product Owner ved sidste møde havde prioriteret højt) grundigt, så der var mulighed for at se, hvor meget de User Stories ville fylde i sprintet. De resterende User Stories lod vi beholde deres grove estimat. Vi splittede store User Stories op i flere mindre User Stories, og vi skrev tasks til de tidligere højt prioriterede (men ikke-implementerede) User Stories. Det var også i det interne møde, at vi estimerede vores egen tid i det kommende sprint.

Herefter havde vi et møde med Product Owner, som så udvalgte de User Stories, han gerne ville have implementeret i det kommende sprint ud fra vores eget tidsestimat og tidsestimatet på User Stories.

Tasks

I de første par sprints oprettede vi de fleste tasks i fællesskab til det interne møde inden Sprint Planning med Product Owner. Efter vi var blevet komfortable med at oprette tasks, begyndte vi at stoppe med at fokusere så meget på at få alle tænkelige tasks med i en User Story, men i stedet oprette nogle få nøgle-tasks og så oprette flere løbende, efterhånden som det gik op for os, hvad der skulle til for at få en feature implementeret.

Det betød ikke noget, for det tidsestimat, vi havde sat på for de fleste User Stories, var (set i bagklogskabens lys) meget store i forvejen, hvilket gjorde det svært at estimere dem præcist.

Sprint Retrospective

Vi har afholdt Sprint Retrospective møder i slutningen af hvert sprint. I forbindelse med dette gennemgik vi vores taiga taskboard, GitHub Issues og GitHub Pull-Requests. Vi sørgede for, at der ikke var noget, der blev overset, og alt der ikke var reviewet og merget, blev gennemgået, og

så oprettede vi en Pull-Request til Master-Branchen. Efter vi havde løst diverse merge-konflikter, oprettede vi så en Github release og afholdt vores Sprint Retrospective.

I Sprint Retrospective møderne snakkede vi om, hvad der var gået godt, hvad der var gået mindre godt, om der var noget vi kunne ændre for at lette hinandens arbejdsprocess og aftalte hvornår vi mødtes i løbet af det kommende Sprint. Nogle af de ting vi har haft vendt i forhold til forbedringer har været:

- at vi skulle blive bedre til at assigne os selv på tasks i Taiga, hvis der var noget man meget gerne ville lave. De tasks der ikke var nogen assignet til, var fri for alle at gå i gang med.
- at vi meget gerne måtte nedbryde tasks i mindre tasks, hvis man var i gang med en stor task, så de andre i gruppen kunne være behjælpelige.

Nogle af de ting vi har haft nævnt i forbindelse med, hvad der er gået godt i et givent Sprint har f.eks. været:

- at vi har været bedre til at anvende Issues og Pull-Requests.
- at vi var blevet bedre til at assigne os til tasks.

Daily SCRUM

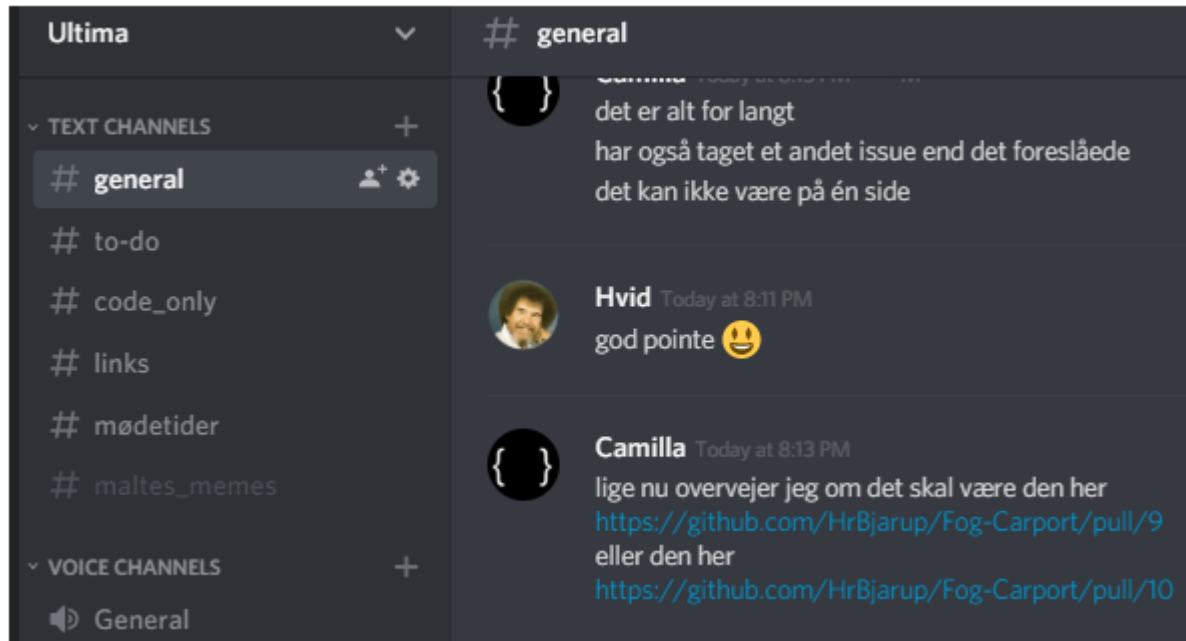
Vi har afholdt Daily SCRUM møder på alle hverdage, med meget få undtagelser. Det har enten været i en voice-chat på vores Discord-server, eller når vi har mødtes på skolen. Til disse møder har vi kort fortalt om, hvad vi har lavet siden sidste Daily SCRUM møde, hvad vi fortsætter med at arbejde på, og om man har brug for hjælp til noget specifikt.

Vi har generelt holdt vores Daily SCRUM møder til 15 minutter, men af og til er de gået over tid. Dette har som regel været på grund af kodetekniske problemer. Fordi vi har valgt at arbejde uden en dedikeret SCRUM-master, og vi alle fire deltog i samtalerne, blev det med jævne mellemrum glemt at håndhæve tidsbegrænsningen og fortsætte den tekniske snak til efter mødet.

Det at have et dagligt kort møde hvor vi talte med hinanden, hjalp meget med at holde overblik over, hvor vi som gruppe var henne i processen. Det hjalp også, hvis man var i tvivl om, hvad det smarteste var at bruge sin tid på - så kunne man få hjælp til mødet af de andre.

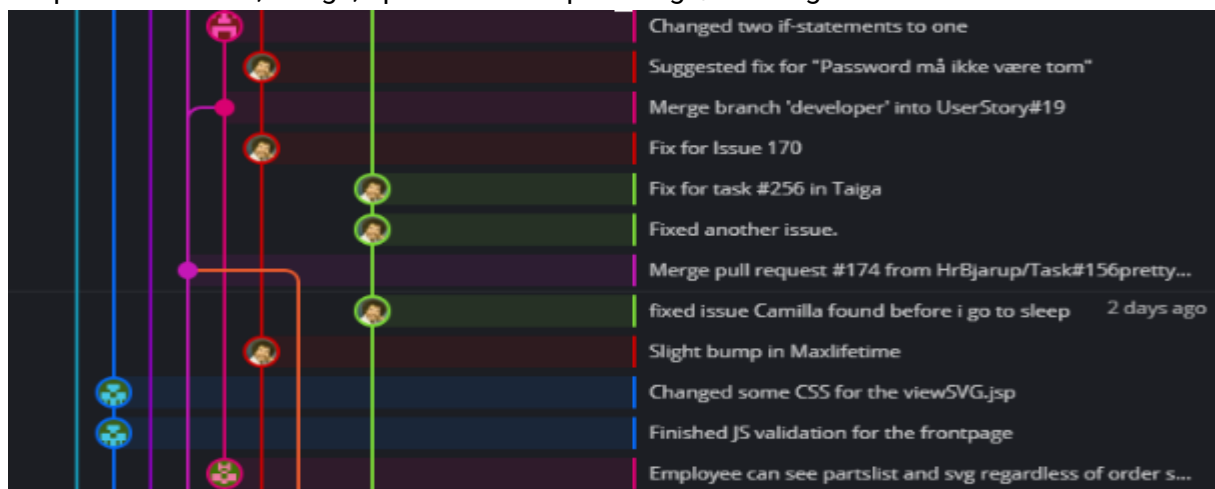
Discord

Discord er en instant-message og voice-client. Det er muligt at oprette forskellige tekst-kanaler til emner. Det er muligt at formatere tekst efter f.eks. Java-standard og dermed hurtigt dele små stykker kode. Vi har brugt det gennem hele studiet, men det har vist sig at være meget værdifuldt i SCRUM-processen, da vi har haft kontakt med en eller flere fra gruppen stort set hele tiden.



GitKraken

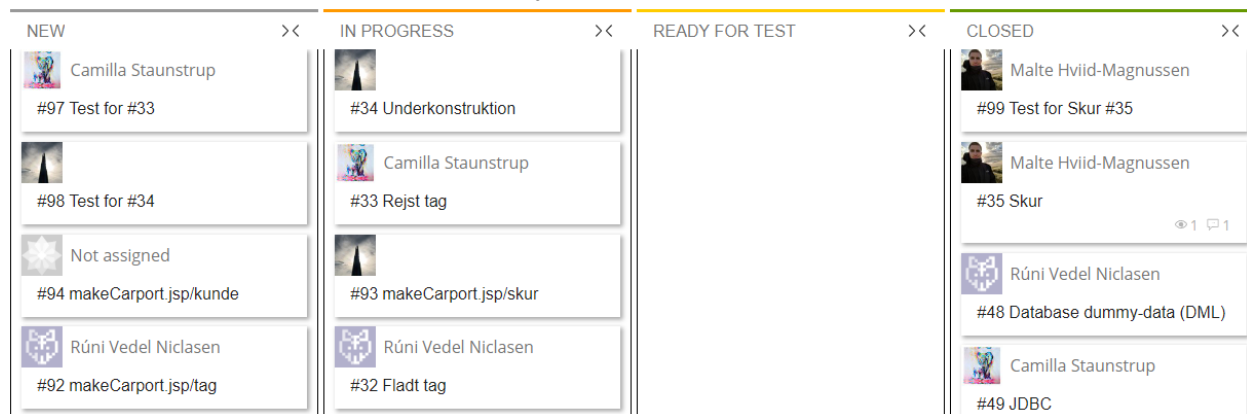
GitKraken er en git-client som overskueliggør ens repository på en meget mere visuel måde end commandline tools som f.eks. GitBash, som vi har haft brugt hidtil. Vi valgte at bruge GitKraken, da det gjorde det nemmere for os at følge med i, hvad hinanden arbejdede på - det var nemmere at oprette branches, merge, oprette Pull-Requests og løse mergekonflikter.



Taiga

Taiga² er et online værktøj der hjælper et team med SCRUM. Man kan oprette stories, sprints, tasks osv. Vi har brugt taiga til at holde overblik over vores User stories, tasks, sprints og deadlines.

I Taiga er der et virtuelt taskboard. Der findes 4 primære kategorier som vi har anvendt til at sortere tasks. "New", "In progress", "Ready for test" og "Closed".



Når man opretter en ny task under en User Story, så ligger den i "New" indtil en person assigner sig selv på den og flytter den over i "In progress". Når en task er løst sættes den over i "Ready for test", og når en task (eventuelt i sammenhæng med andre tasks i samme User Story) er merget ind på Developer-Branchen, flyttes den over i "Closed".

Taiga point-estimering

Taiga har et point-estimerings-system, som vi har anvendt. Vi har primært kigget på det som timer fremfor point, da det var lidt abstrakt at sætte os ind i, i begyndelsen af projektet. Derudover har vi lavet to typer af tags, som vi har tildelt alle User Stories. Det ene tag, er et groft størrelses-estimat (large, medium small), og det andet er et afhængigheds-estimat (high, medium, low), så det var nemmere at vurdere hvor meget tid vi skulle estimere de enkelte User Stories til, når vi sad med estimering.

GitHub

Vi har brugt GitHub som versionsstyringsværktøj. Vi har alle tidligere gjort brug af GitHub, men slet ikke i så stort et omfang, som vi gjorde i dette projekt. Vi har f.eks. ikke gjort brug af issues, pull requests og releases i tidligere projekter. Hvordan vi har anvendt det i dette projekt kommer vi nærmere ind på i de næste afsnit.

Vi lavede release kort før hver sprint-afslutning, så der lå et snapshot af vores repository fra afleveringsdatoen. Dette medførte at vi kunne arbejde videre på projektet og Product Owner ville stadig have en nem måde at tilgå projektets tilstand efter de enkelte sprints.

² <https://tree.taiga.io/project/maltemagnussen-fog/timeline>

Github Wiki

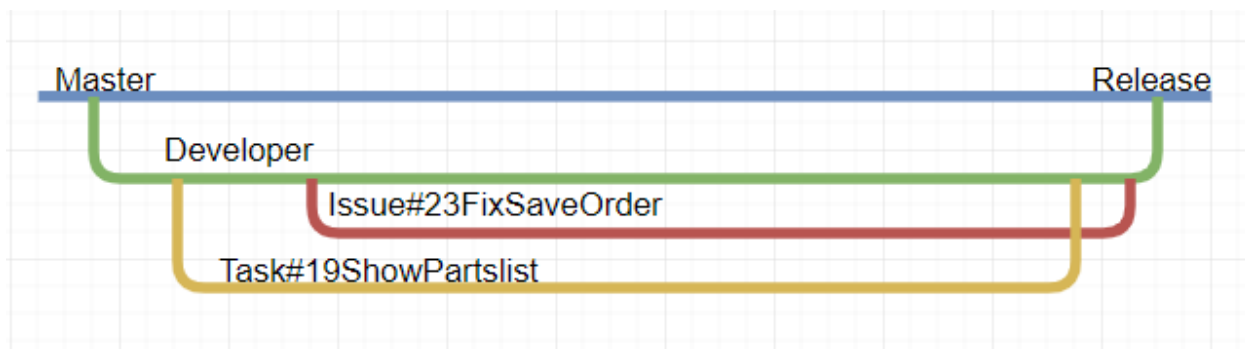
Rúni brugte GitHubs Wiki-funktionalitet i et tidligere projekt som en slags logbog. Det synes vi andre var en god idé, og det har vi ført videre til dette projekt. I [vores wiki på GitHub](#) kan man løbende se tanker og overvejelser, noter fra møder, arbejde vi fik færdiggjort, problemer vi stødte på osv. for hvert sprint.

Readme

Vi har i de tidligere projekter alle gjort meget ud af at have en god og forklarende readme på GitHub og nærmest brugt den som en rapport. I dette projekt bliver mange af de ting vi tidligere har skrevet i readme-filen dokumenteret andre steder. Dette projekts readme indeholder derfor kun links til uddybende information, JavaDoc m.m.

Branches

Inden vi lærte om branches, har vi i forrige gruppe-projekter pushet direkte på master. I dette projekt har vi haft en Master-branch, en Developer-branch fra Master-branchen og så en branch fra Developer til hver Issue/Task.



Forsimplet visning af vores brug af branches

Branches har vist sig at være utrolig brugbart på mange måder for os. Vi har kunne arbejde sideløbende på helt forskellige opgaver i hver vores branch uden at tænke på, om man ødelægger noget en anden fra gruppen sidder og arbejder på. Det har gjort det tydeligt, hvem der har arbejdet på de forskellige tasks, og det har været meget nemt at "hoppe over" på en anden branch end den, man selv er i gang med, for at hjælpe med problemløsning eller forslag.

Master bliver kun anvendt til releases. Det vil sige at når vi har færdiggjort et sprint og merget de branches, vi har arbejdet på ind på Developer og sikret os at Developer kan bygge, og programmet kan køre, som vi forventer, så merger vi Developer ind i Master og tagger den med et release.

Ud fra Developer opretter vi så to forskellige typer branches, som beskrevet og illustreret kort ovenfor. Issue-branches og task-branches. Issue-branches er til alle de problemer eller Issues vi støder på under programmeringen. Det kan f.eks. være noget JavaScript der er holdt op med at virke, eller det kan være at der bliver smidt en exception, når man prøver at gemme en ordre.

Task-branches er vores features, som er knyttet til de forskellige User Stories. Hver User Story brydes ned i mindre opgaver eller tasks, som vi så arbejder på i task-branches.

Efter en Issue- eller task-branch er testet (enten manuelt eller med automatiserede tests), har vi lavet et Pull-Request til Developer. Hertil kan der knyttes specifikke folk til review af ens arbejde, og labels der siger noget om hvilken slags feature eller issue, man har arbejdet på. Når andre gruppemedlemmer har reviewet og godkendt en Pull-Request, så kan den merges ind på Developer. I review-processen finder de andre gruppemedlemmer ofte fejl eller ting der kan forbedres inden branchen merges.

Branches tillader også at man afprøver ting og løsninger, som måske viser sig ikke at fungere. Så kan man bare efterlade den branch, uden at have gjort skade på en af de to hovedbranches.

De regler, vi har fulgt for branches under programmering af projektet, har været følgende:

- Det er kun developer der må merge ind på master.
- Task- og Issue-branches brancher ud fra Developer og merger ind igen på Developer.
- Task- og Issue-branches skal enten have et relevant navn, eller indeholde ID'en for deres respektive Taiga task eller Github Issue.

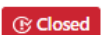
Målet med disse regler er altid at holde Developer og Master fri for bugs. Gennem Pull-Requests og tests sørger vi for, at Master altid kan bygge og køre og indeholder gennemtestet og reviewet kode.

Issues

I det følgende vil der blive vist et godt eksempel på vores Issues, som illustrerer, hvordan vi har brugt dem. Der er tale om Issue#165³ hvor der er blevet fundet følgende problem:

Der bliver tilføjet et materiale til vores stykliste med 0 i antal. Der er en beskrivende titel, et sigende label og et billede med en kort forklarende tekst der kan spore en nærmere ind på problemet.

Material with 0 quantity is added to the partslist #165



HrBjarup opened this issue 4 days ago · 2 comments

[Edit](#)[New issue](#)

HrBjarup commented 4 days ago

Owner

+ 🗨️ ...

So far I have only seen this happen when an order has an incline of 0 degrees.



HrBjarup added the **bug** label 4 days ago



HrBjarup added this to the **Sprint 5** milestone 4 days ago



HrBjarup commented 4 days ago

Author

Owner

+ 🗨️ ...

0	20	Placering: Euclyde (Bilskov)	Indgængende materialer på lager	6000	1000	0
0	20	Placering: Euclyde (Bilskov)	Indgængende materialer på lager	6000	1000	0



HrBjarup assigned Runi-VN 4 days ago



Runi-VN referenced this issue 2 days ago

Task#236 - Upgrade FlatRoofCalc #185

Merged



Runi-VN commented 2 days ago

Collaborator

+ 🗨️ ...

Please see #185



Runi-VN closed this 2 days ago

Assignees

Runi-VN

Labels

bug

Projects

None yet

Milestone

Sprint 5

Notifications

Unsubscribe

You're receiving notifications because you're watching this repository.

2 participants



Lock conversation

Pin issue

Det er sådan vi oftest har brugt Issues på GitHub. Vi har nogle få Issues som har været meget store. Dem er der blevet talt meget om, både i kommentarfelterne til Issue'et og på vores interne chat i vores gruppes Discord-kanal, men langt de fleste af vores Issues er kortfattede og relativt nemme at gå til. Når man har løst et Issue, lukker man det selv og henviser til den Pull-Request der løser Issue'et.

Pull-Requests

I det følgende vil der blive vist et godt eksempel på vores Pull-Requests, som illustrerer hvordan vi har brugt dem. Der er tale om Pull-Request#10⁴ hvor den første version af vores create script

³ <https://github.com/HrBjarup/Fog-Carport/issues/165>

⁴ <https://github.com/HrBjarup/Fog-Carport/pull/10>

til vores database lå klart. Som det fremgår af billedet, er det blevet lagt op med en kort beskrivelse og er blevet reviewet og godkendt af en anden fra gruppen og siden merget ind på Developer-branchen.

Initial createscript for DB added to project #10 Edit

Merged MalteMagnussen merged 5 commits into `developer` from `DDL(createsriptDB)` on 24 Apr

Conversation 8 Commits 5 Checks 0 Files changed 1 +127 -0

Castau commented on 24 Apr Collaborator

Added first version of database DDL script to the project

Initial createscript for DB added to project f158029

Castau requested a review from MalteMagnussen on 24 Apr

MalteMagnussen requested changes on 24 Apr View changes

MalteMagnussen left a comment Collaborator

Looks fine.
Works in my SQL bench flawlessly.
Matches the info in the [sheet](#).

docs/Database docs/create.sql	Outdated	Show resolved
docs/Database docs/create.sql	Outdated	Show resolved
docs/Database docs/create.sql	Outdated	Show resolved
docs/Database docs/create.sql	Outdated	Show resolved
docs/Database docs/create.sql		Show resolved
docs/Database docs/create.sql		Show resolved

MalteMagnussen and others added some commits on 24 Apr

Update docs/Database docs/create.sql	Verified	52f97c2
Update docs/Database docs/create.sql	Verified	23b7e01
Update docs/Database docs/create.sql	Verified	3ae523a
Update docs/Database docs/create.sql	Verified	8651a3c

MalteMagnussen approved these changes on 24 Apr View changes

MalteMagnussen merged commit 6defaec into developer on 24 Apr Revert

Reviewers

MalteMagnussen	✓
----------------	---

Assignees

Castau

Labels

- enhancement
- suggestion

Projects

None yet

Milestone

Sprint 1 | EASTER

Notifications

Unsubscribe


You're receiving notifications because you're watching this repository.


3 participants

[Lock conversation](#)

Releases

Vi udgav som beskrevet et release⁵ efter hvert sprint, som er et snapshot af hvordan programmet så ud på daværende tidspunkt.


 v3.0

 3221268

Verified

Sprint 3

Edit

 Runi-VN released this 14 days ago · [606 commits](#) to master since this release

This is what was completed by the end of Sprint 3. Please see the [wiki](#) for more information.

[Progress](#) this week.

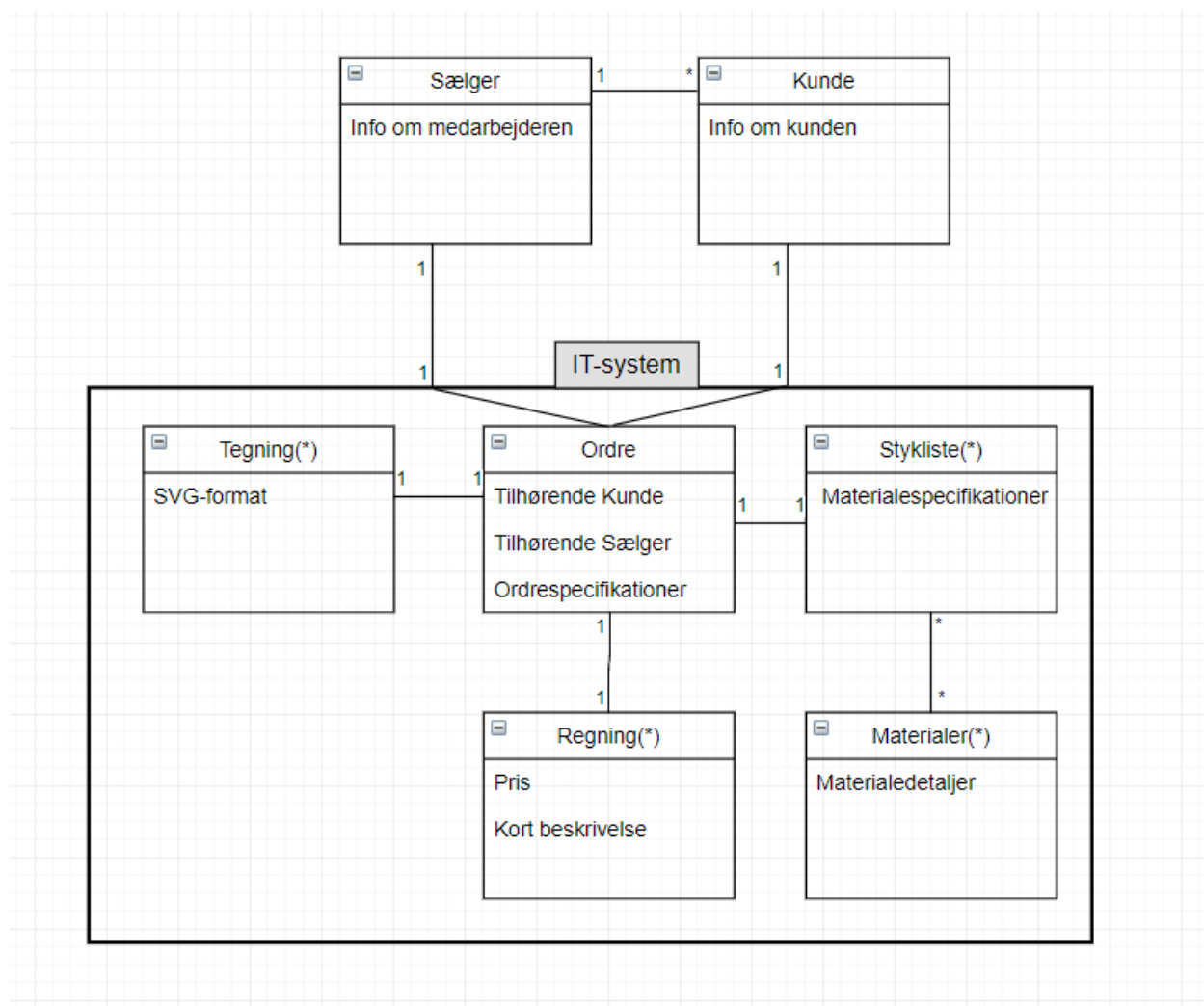
► Assets [2](#)

⁵ <https://github.com/HrBjarup/Fog-Carport/releases>

Diagrammer

Domæne model

Domæne modellen blev lavet i starten af projektet, før programmet var skrevet. Vores forestilling om programmets struktur ud fra krav og samtaler med Product Owner var som følger:

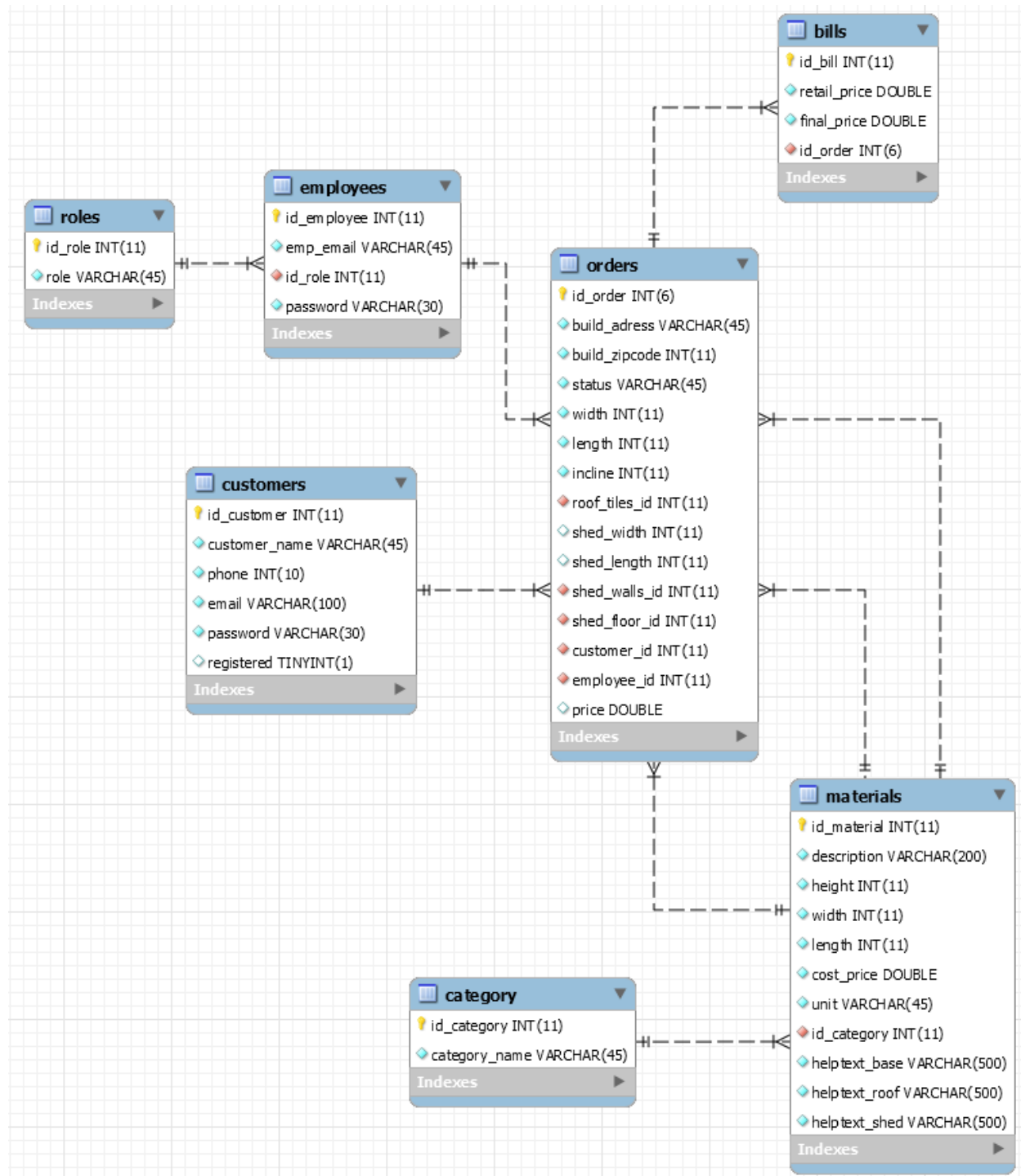


Domæne model beskrivelse

Medarbejder/sælger har mange kunder, hvorimod en Kunde kun har kontakt til én sælger. Sælger skal have adgang til en kundes ordre, den tilhørende tegning, regning og styklister. Kunden skal ligeledes have adgang til sin ordre og den tilhørende tegning, regning og styklister. En styklister har mange materialer og et materiale kan være på mange styklister.

E/R diagram

[DDL-script link](#)⁶



⁶ [https://github.com/HrBjarup/Fog-Carport/blob/master/extraFiles/Database%20docs/create\(DDL\).sql](https://github.com/HrBjarup/Fog-Carport/blob/master/extraFiles/Database%20docs/create(DDL).sql)

Beskrivelse

Alle tabellernes rækker får tildelt en automatisk generet ID som er unik for den enkelte. Dette er tabellernes Primary Key.

Tabellerne i databasen

Orders

Hver række i denne tabel repræsenterer 1 ordre hos Johannes Fog. Dette er den vigtigste tabel i systemet, og danner centrum for de andre tabeller.

Info om orders der gemmes i tabellen:

- Adressen, samt postnummeret for hvor carporten skal bygges.
- Prisen på ordren - beregnes ud fra indkøbspris på materialerne. Kan tilpasses af en sælger.
- Status på ordren - Hvorvidt ordren er betalt eller ej. Det kunne have været en boolean sådan som vi anvender den på nuværende tidspunkt, men planen var at have flere tilstande for den.
- Længde og bredde på carporten.
- Hældningen på taget i grader. Hvis den er 0 betyder det at taget er fladt.
- Længde og bredde på skuret i carporten, hvis kunden har ønsket et skur. Ellers er disse 0. Måske burde info om skur have ligget i en separat tabel, så order tabellen ikke behøvede at indeholde en masse tomme celler, hvis ordren ikke indeholdt et skur.
- Så kommer der 3 ID'er der referer til materialer. Roof_tiles_id, tagsten. Shed_walls_id, træet der bruges til beklædning på skuret. Shed_floor_id, træet der bruges til gulvet i skuret. Disse er Foreign Keys til det materiale algoritmen der genererer styklisten skal bruge.
- ID der refererer kunden der har oprettet ordren. (Foreign Key)
- ID til den sælger der er ansvarlig for ordren. Lige nu er det hardcoded til altid at være den samme sælger (linje 106 i OrderCarport.java). Planen var oprindeligt at en ordre skulle have haft en dummy sælger med ID 0 tilhæftet som standard, og så skulle sælgere selv kunne gå ind og gøre sig ansvarlig for en ordre. F.eks. hvis det er en kunde de før har haft, eller et geografisk område de har ansvar for. (Foreign Key)

Materials

Hver række i denne tabel repræsenterer 1 unikt byggemateriale. Internt i programmet har java-objektet en kvantitet på. Så når logik-laget anvender materialer fra databasen, beregnes der også hvor mange af dem der skal bruges, og så sættes det på instansen af objektet som et field.

Info om et materiale der gemmes i tabellen:

- Kort beskrivelse af materialet.
- Højde, bredde og længde på materialet.
- Indkøbsprisen for 1 af materialet.
- Materialets enhed. (Sæt, Stk, Pakke)
- En ID reference til materialets kategori (Foreign Key)
- Der 3 kolonner med hjælpetekst. Det er fordi det samme materiale kan blive anvendt til forskellige formål. F.eks. en taglægte skal både bruges til taget, men også til døren i skuret.

Customers

Hver række i denne tabel repræsenterer 1 kunde hos Johannes Fog.

Info om en kunde der gemmes i tabellen:

- Navn
- Telefonnummer
- Email adresse
- Password (hvis de har ønsket at få en bruger - dette er ikke påkrævet)
- Om de er registreret som en bruger eller ej. Man kan kun se sine ordrer hvis man vælger at oprettes som kunde i systemet. Ellers bruges kunde informationerne udelukkende som kontaktinformation for en sælger, når sælger skal behandle bestillingen i systemet.

Employees

Hver række i denne tabel repræsenterer 1 ansat hos Johannes Fog.

Derudover er der den ansattes egen email, et selvvalgt password og en id_role der er Foreign Key til role tabellen.

Category

Category bliver ikke anvendt i koden i den endelige version. Tanken med category var at anvende den i visningen til at opdele materialerne i styklisten i "Træ", "Tagpakken" og "Beslag og Skruer". Derudover skulle en employee kunne tilføje og ændre i materialerne i databasen, og i den forbindelse f.eks. kunne sortere efter kategori. Disse ting blev dog ikke implementeret af forskellige årsager. Tid var en vigtig faktor.

Role

Role bliver ikke anvendt i koden i den endelige version. En User Story den kunne have været brugt i ville være f.eks.:

"Som administrator vil jeg gerne kunne oprette nye sælgere i systemet." Eller "Som administrator vil jeg gerne kunne rette i oplysninger om sælgere i systemet."

Og det kræver, at der er en administratorrolle og en sælger rolle hæftet på employee table. Evt flere roller end det, alt efter hvad Product Owner kunne have ønsket sig. Men vi nåede ikke så langt i processen.

Bills

Bills bliver ikke anvendt i koden i den endelige version. Bills var inspireret af den regning som Fog sender ud til kunderne. ([Se bilag 6](#))

Bills var tiltænkt at have håndteret prisen på carporten ved at fungere som en en-til-mange tabel, så en ordre kunne have flere regninger (ved f.eks. tilbud)

I samarbejde med order.status skulle man kunne se forskellig data for carport prisen og manipulere denne.

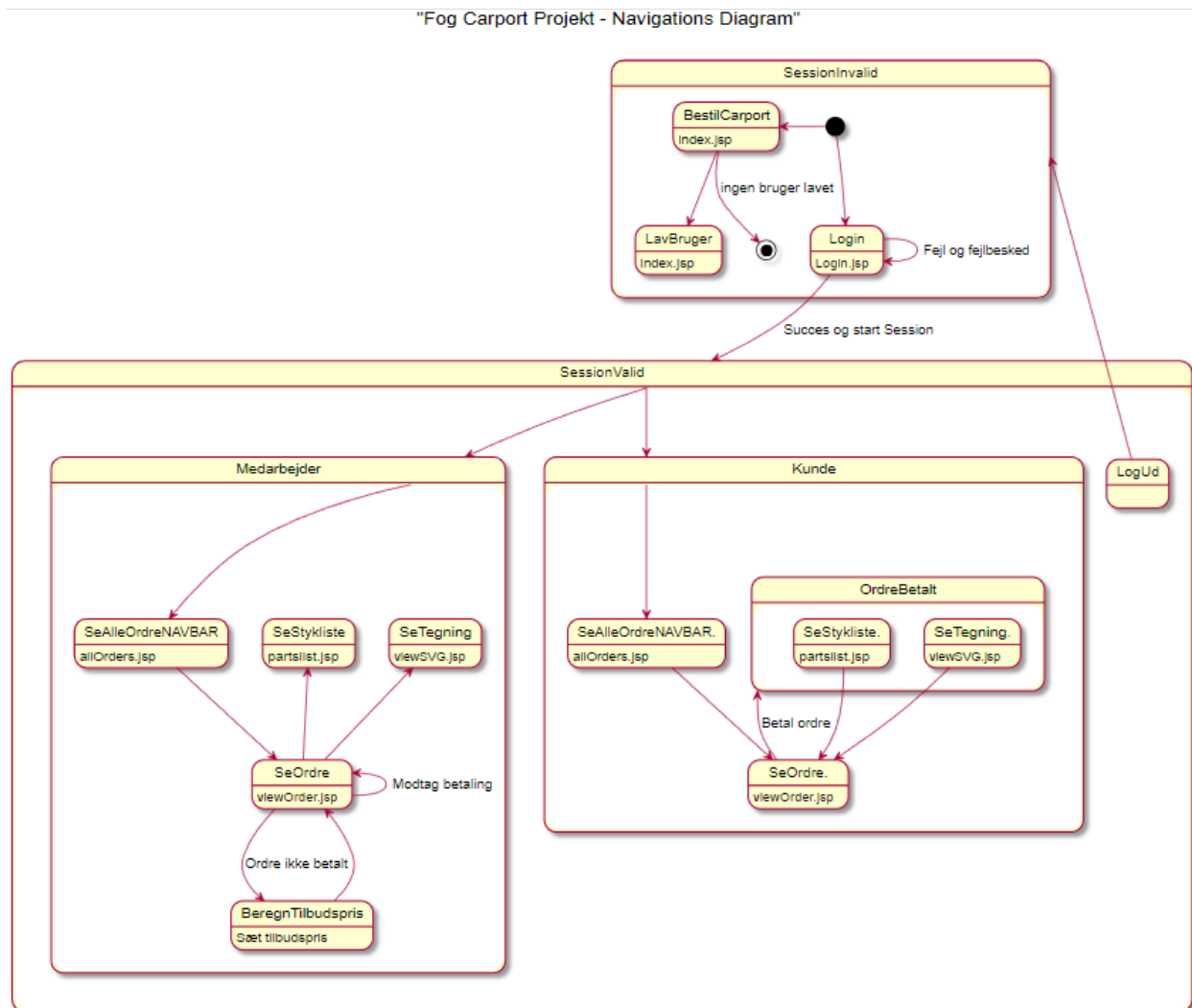
Hvis vi ville have implementeret dette, så manglede vi bl.a.:

- 1) At lave et bill-objekt i java.
- 2) At oprette bill samtidig med ordre
- 3) Lave mappers til at opdatere/gemme bills
- 4) Bills skulle så anvendes ved visning af suggestedPrice, ved tilbud & ved betaling af ordre

Som et kompromis oprettede vi order.price i [User Story #19](#), som samarbejder med order.status om at opdatere ordreprisen. Der er mere dokumentation herfor i [pull requesten for US#19](#)

Navigationsdiagram

Navigationsdiagrammet viser de muligheder en bruger har i vores system. En bruger kan være en Medarbejder eller en Kunde. Den sorte prik indikerer start når ip'en bliver tilgået.



Beskrivelse af Navigationsdiagram

<http://www.bjarup.com> leder til index.jsp. Denne side er bygget op så den altid indeholder:

- navbar.jsp
- skiftende sideindhold
 - starter med homepage.jsp som har bestillingsformen, samt brugeroprettelsen.
- footer

Hvilket vil sige, at man fra alle sider har adgang til `navbar.jsp`, som vil have skiftende indhold alt efter om det er en Medarbejder eller en Kunde, der er logget ind. Begge brugertyper, samt brugere uden Session har adgang til linket til Login/Logud i `navbar.jsp`.

Login linket i `navbar.jsp` fører til `login.jsp`, og begge brugertyper kan logge ind i formen på siden. Når man er logget ind vil man få vist forskelligt sideindhold, alt efter om man er Medarbejder eller Kunde.

Logget ind som Medarbejder:

- Medarbejderen starter på `allOrders.jsp`, hvor de får vist en liste over alle ordrer i systemet.
 - Medarbejderen kan i navbaren altid komme tilbage til denne side via "Se alle ordrer".
 - Klikker Medarbejderen sig ind på en specifik ordre kommer de ind på `viewOrder.jsp`, hvor de kan se ordreinformationer om denne ordre.
 - Der er funktionalitet til at beregne dækningsgraden på en indtastet pris, og der er mulighed for at sætte en indtastet pris som tilbud. Denne mulighed ophører når ordren er betalt.
 - Der er funktionalitet til at registrere, at en Kunde har betalt for ordren. Dette kan gøres én gang og kun, hvis Kunden ikke selv har betalt for ordren online.
 - Der er links til at se henholdsvis ordrens Stykliste og Tegning, som altid er tilgængelig for Medarbejderen.
 - `partslist.jsp` viser indholdet i ordrens stykliste. Der kan klikkes på "Tilbage til din valgte ordre", som fører Medarbejderen tilbage til `viewOrder.jsp`.
 - `viewSVG.jsp` viser en tegning over ordren. Der er funktionalitet til at toggle de forskellige tegningsdele til og fra. Der kan klikkes på "Tilbage til din valgte ordre", som fører Medarbejderen tilbage til `viewOrder.jsp`.

Logget ind som Kunde:

- Kunden starter på `allOrders.jsp`, hvor de får vist en liste over alle egne ordre.
 - Kunden kan i navbaren altid komme tilbage til denne side via "Se dine ordrer".
 - Klikker Kunden sig ind på en specifik ordre, kommer de ind på `viewOrder.jsp`, hvor de kan se ordreinformationer om denne ordre.
 - Der er funktionalitet til at betale for ordren. Dette kan gøres én gang og kun, hvis Medarbejderen ikke har modtaget betaling og registreret dette.
 - Der er links til at se henholdsvis ordrens Stykliste og Tegning, som bliver tilgængelige for Kunden når ordren er betalt.
 - `partslist.jsp` viser indholdet i ordrens stykliste. Der kan klikkes på "Tilbage til din valgte ordre", som fører Kunden tilbage til `viewOrder.jsp`.
 - `viewSVG.jsp` viser en tegning over ordren. Der er funktionalitet til at toggle de forskellige tegningsdele til og fra. Der kan klikkes på "Tilbage til din valgte ordre", som fører Kunden tilbage til `viewOrder.jsp`.
 - Klikker Kunden på "Bestil Carport" kommer de ind på `homepage.jsp` og kan udfylde endnu en bestilling, men kan ikke oprette en ny bruger.

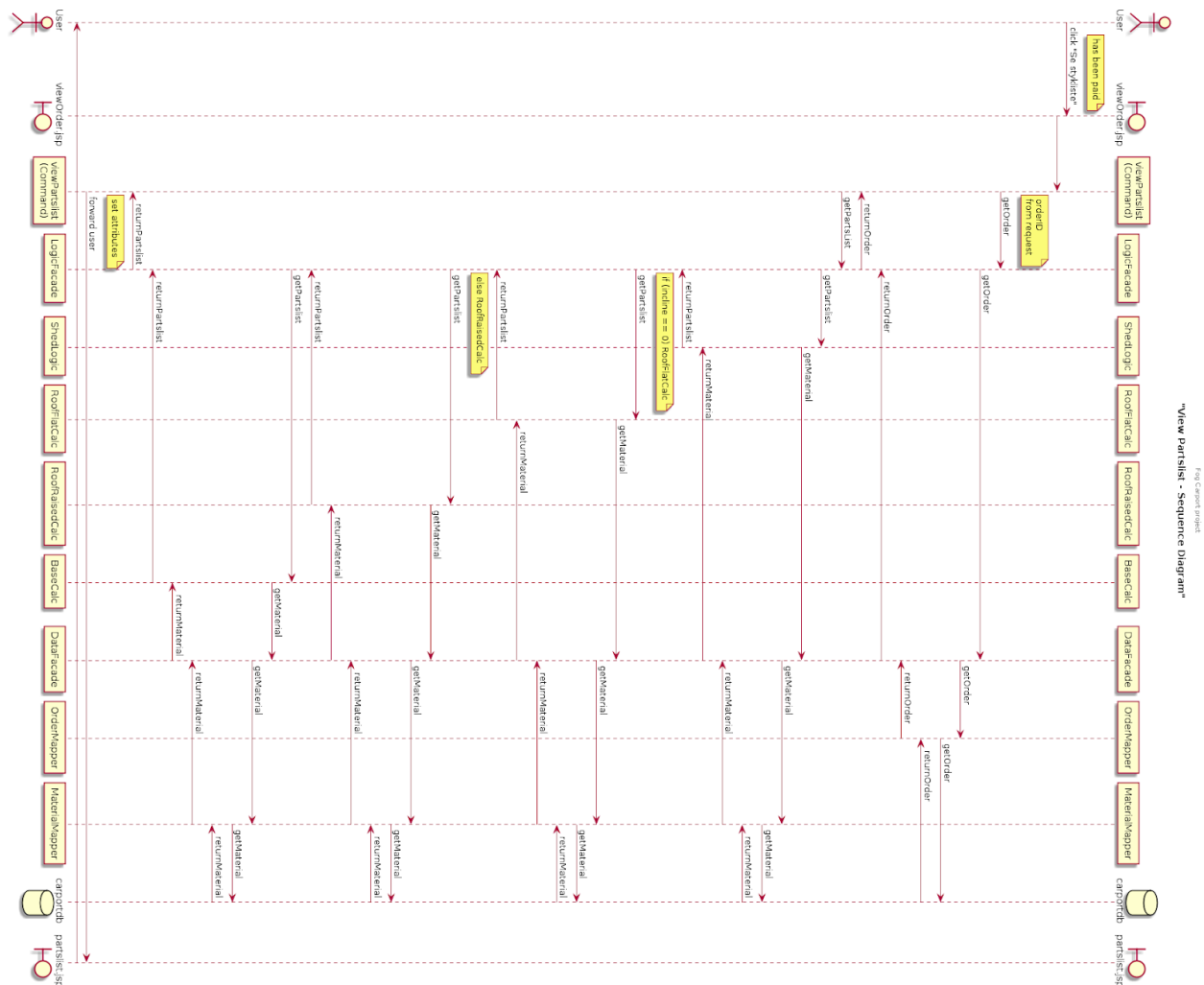
Sekvensdiagrammer

Formålet med et sekvensdiagram er at skabe et overblik over essentielle dele af programmet.

For os har det været vigtigt at overholde arkitekturen i programmet. Ved at vise sekvensdiagrammer for henholdsvis styklisten og SVG-tegningerne formår vi at vise et detaljeret billede af ikke bare to kernefunktioner i programmet, men også hvordan dataen rejser igennem de forskellige lag.

Sekvensdiagram for ViewPartslist

[Se Bilag 2 - Sekvensdiagram \[Partslist\]](#)



Gennemgang af ViewPartslist

Det antages, at brugeren i starten af diagrammet er på ordresiden for en specifik ordre ("viewOrder.jsp"), og diagrammet viser hvad der sker fra øjeblikket brugeren klikker på knappen Se Stykke og indtil brugeren er fremme på siden, som fremviser styklisten ("partslist.jsp").

PartslistModels ansvar

`PartslistModel` er en klasse, der indeholder en `ArrayList` af `MaterialModels` ("billOfMaterials"), en totalpris for disse materialer og fields, der bruges til generering af SVG-tegninger. Klassen har også metoder til at tilføje enkelte `MaterialModels` og andre `PartslistModels` til `billOfMaterials`.

Mod slutningen af projektet blev der diskuteret, om klassen kunne have været lavet som en `extended ArrayList`:

```
public class PartslistModel extends ArrayList<MaterialModel>
```

Dette var desværre ikke noget, vi havde tid til at hverken implementere eller undersøge.

Commandens ansvar ("ViewPartslist")

Commanden er klassen vi lander i, efter brugeren har trykket på *Se stykliste*.

Herfra sker der følgende - vi:

1. Validerer ordre-ID'et fra requesten med vores `Validate`-klasse.*
2. Hent ordren gennem logik-laget
3. Hent partslist gennem logiklaget
4. Sæt disse på som attributter.
5. Returnerer til FrontControlleren som videresender til `partslist.jsp`.

`partslist.jsp` håndterer så disse elementer og viser dem frem på forskellige måder afhængig af, om man er customer eller employee. ([se bilag 5](#))

**Ikke vist på sekvensdiagrammet*

Logik façadens ansvar ("LogicFacade")

`LogicFacade` sørger for at indhente de relevante metoder og klasser fra underliggende lag.

I forhold til `OrderModel` går den videre ned i laget til Data Facaden for at hente objektet tilbage.

I forhold til `PartslistModel` instantierer og henter den én overordnet `PartslistModel`, der indeholder alt materiale. Før den returnerer denne, henter den andre `PartslistModels` fra Calculation klasserne og lægger dem sammen. `LogicFacade` sørger samtidig for kun at hente det nødvendige:

```
PartsListModel getPartslistModel(OrderModel order)7
{
    (...)
    if (order.getShed_width() != 0 &&
        order.getShed_length() != 0)
    {
        //Add shed
    }
    if (order.getIncline() == 0)
```

⁷ `LogicFacadeImpl.java`, linje 87-113

```

        {
            //Add flat roof
        }
        else
        {
            //Add raised roof
        }
        //Add base
    (...)
}

```

Både `OrderModel` & den samlede `PartslistModel` returneres til commanden.

Calculations klassernes ansvar ("ShedLogic", "RoofFlatCalc", "RoofRaisedCalc", "BaseCalc")

Disse klasser returnerer en `PartslistModel` med materialerne (`MaterialModel`) for en carport for en given ordre.

Klasserne forsøger så vidt muligt smartest at udregne materialer, men det er samtidig forsøgt gjort på samme måde som Fog (Se [bilag 7](#)).

Materialerne samles i en instans af klassen `PartslistModel`, der fungerer som en udvidet `ArrayList`, designet til at indeholde `MaterialModels`.

`PartslistModel`-klassen har egne fields, som sættes under beregningerne. Det er disse, som bruges til beregning af antal elementer i SVG-tegningerne.

For hver klasse gælder det, at der returneres en `PartslistModel` til `LogicFacade`.

Data façadens ansvar ("DataFacade")

`DataFacade` sørger for at indhente de relevante metoder og klasser fra underliggende lag.

Først skal den håndtere `OrderModels`.

```
public OrderModel getOrder(int id)
```

Det gør den ved at instantiere en `OrderMapper`, sætte dennes `DataSource` (`Database Connection`) og kalde `orderMapper.getOrder()`.

Denne `OrderModel` sendes retur.

Så skal den håndtere `MaterialModels`.

```
public MaterialModel getMaterial(int id, String helptext)
```

Det fungerer meget på samme måde: Den instantierer en `MaterialMapper`, sætter en `DataSource` og kalder `materialMapper.getMaterial(id, helptext)`.

Denne `MaterialModel` sendes retur.

Mappernes ansvar ("OrderMapper", "MaterialMapper")

Mapperne sørger for det vigtige, når det kommer til at hente data fra databasen. Det er her metoderne har SQL-forespørgsler og oprettelse af `OrderModels` og `MaterialModels`.

For begge mappere gælder det, at vi med et *try-with-resources*-statement åbner for en connection og laver et PreparedStatement, der indeholder vores SQL-query.

Før vi sender det afsted, sørger vi for at opdatere SQL-querien til at indeholde det relevante ID, så vi kun får en række tilbage.

Hvis vi får en række tilbage, sørger vi for at oprette et objekt og gemme rækkens information i dette objekt. Hvis der sker en fejl, smider vi en custom DataException med en relevant besked. Objektet returneres.

For `getMaterials()` gælder det, at vi også sætter en String ("**Help**text") ind i vores SQL-query. Denne kommer helt oppe fra Calculations-klasserne og er til for at vælge hvilken række af helptext, de skal vælge fra databasen (*helptext_shed, helptext_base, helptext_roof*). Før vi fortsætter med databasekaldet, tjekker vi om den medsendte Help**text** er en af de tilladte Strings.

[Se bilag 3 - Sekvensdiagram \[SVG\]](#)



Gennemgang af ViewSVG

SVG står for **Scalable Vector Graphics**. SVG-elementer kan bruges i HTML ligesom almindelige HTML-elementer. Her følger et eksempel:

```
<circle cx="125" cy="25" r="20" stroke="#000000" stroke-width="10"
stroke-dasharray="5,5" stroke-linecap="round" fill="#FF0000"
style="opacity: 0.5;"/>
```

Vores fire klasser for hver del af carporten (*Base*, *Shed*, *FlatRoof*, *RaisedRoof*) samarbejder om at bygge disse SVG-elementer ud fra en given ordre.

Det foregår ved at hver klasse returnerer et *String*-objekt, som indeholder de relevante elementer. Alle disse *String*-objekter hentes af *LogicFacade*-klassen, hvorefter de af *ViewSVG.java* bliver sat som request-attributter, for endeligt at blive vist i *viewSVG.jsp*.

Det antages at brugeren i starten af diagrammet er på ordresiden for en specifik ordre ("*viewOrder.jsp*"), og diagrammet viser hvad der sker fra øjeblikket brugeren klikker på knappen *Se Tegninger* og indtil brugeren er fremme på siden som fremviser tegningerne ("*viewSVG.jsp*").

Commandens ansvar ("*ViewSVG*"):

Commanden er klassen vi lander i efter brugeren har trykket på *Se Tegninger*.

Herfra sker der følgende - vi:

1. Validerer ordre ID'et fra requesten med vores *Validate*-klasse.*
2. Henter den givne ordre fra databasen.
3. Genererer en stykliste for den givne ordre.
4. Opretter et *String*-objekt der indeholder SVG-elementerne for underkonstruktionen og skur og skur-pile. (hvis ordren indeholder skur)
5. Sætter *String*-objektet på som request-attribut.
6. Genererer *String*-objekter der indeholder SVG-elementerne for pile og etiketter (længde/bredde af carport)
7. Sætter disse på som request-attribut.
8. Genererer *String*-objekt for tag. (Afhængig af *incline* kommer der tegning for fladt eller rejst tag)
9. Sætter dette på som request-attribut.
10. Sætter ordre ID'et på som request-attribut
11. Returnerer til *FrontController*en som videresender til *viewSVG.jsp*.

**Ikke vist på sekvensdiagrammet*

Logik façadens ansvar ("LogicFacade"):

Se [gennemgangen af sekvensdiagrammet for ViewPartslist](#) for en detaljeret gennemgang.

LogicFacade håndterer calculation-klasserne og sørger for at få fat i styklisten til udregning SVG-elementer.

LogicFacade håndterer tegningerne for:

1. SVGBase (Underkonstruktion) + SVGShed (Skur) hvis ordren ønsker det
(getSVGbase)
2. SVG Base Arrows (Viser carport størrelser) & Labels (etiketter) (Længde/Bredde)
(getSVGbaseArrowLength, getSVGbaseArrowWidth, getSVGbaseLabelWidth & getSVGbaseLabelLength)*
3. SVGRoof (Tagdelen) - Vælger mellem fladt og rejst tag afhængig af ordres hældning
(OrderModel.getIncline)
(SVGDrawingFlatRoof eller SVGDrawingRaisedRoof)

**Umiddelbart ser det ud til, at der laves mange metodekald, men det gøres fordi, der skal laves to pile og to etiketter for både carport og tag, altså i alt 8 elementer.*

Den samlede PartslistModel og String-objektet sendes retur til commanden.

Calculations klassernes ansvar ("ShedLogic", "RoofFlatCalc", "RoofRaisedCalc", "BaseCalc"):

Se [gennemgangen af sekvensdiagrammet for ViewPartslist](#) for en detaljeret gennemgang.

Disse klasser returnerer en PartslistModel med materialerne (MaterialModel) for hver deres del af carporten for en given ordre.

Denne PartslistModel og dens materialer indeholder information om antal af elementer til SVG-tegningerne.

Et eksempel herpå kunne være fieldet int rafterCount, som anvendes af klasserne DrawingFlatRoof og DrawingRaisedRoof til at generere antal spær-elementer.

Denne PartslistModel sendes retur til LogicFacade.

Datafaçadens ansvar ("DataFacade"):

Se [gennemgangen af sekvensdiagrammet for ViewPartslist](#) for en detaljeret gennemgang.

DataFacade sørger for at indhente de relevante metoder og klasser fra underliggende lag.

I tilfældet for ViewSVGskal den håndtere OrderModels & MaterialModels - det får den henholdsvis en OrderMapper og en MaterialMapper til at klare.

Disse OrderModels & MaterialModels sendes retur til LogicFacade.

Mappernes ansvar ("OrderMapper", "MaterialMapper"):

Se [gennemgangen af sekvensdiagrammet for ViewPartslist](#) for en detaljeret gennemgang.

Mapperne sørger for det vigtige, når det kommer til at hente data fra databasen. Det er her metoderne har SQL-forespørgsler og oprettelse af `OrderModels` og `MaterialModels`.

Disse `OrderModels` & `MaterialModels` sendes retur til `DataFacade`.

SVGDrawing klassernes ansvar ("SVGDrawingBase", "SVGDrawingShed", "SVGDrawingFlatRoof", "SVGDrawingRaisedRoof"):

Vi har nu adgang til `OrderModel` og `PartslistModel`, som indeholder vigtig data for beregning af SVG tegningerne.

Som beskrevet ovenfor, så gælder for for hver klasse, at de skal returnere et `String`-objekt med deres SVG-elementer. Hver klasse bruger både `OrderModel` og `PartslistModel` til at udregne dimensioner samt antal SVG-elementer. Et godt eksempel herpå kan findes i starten af `SVGDrawingRaisedRoof` hvor der sættes fields:⁸

```
int svgExtraPadding = 100;
int halfPadding = (svgExtraPadding / 2);
int roofWidth = (order.getWidth() / 10) + 60; // + overhang
int roofLength = order.getLength() / 10;
int rafterCount = roofRaisedBOM.getRafterCount(); //partslist
```

Brugen af disse fields kan f.eks. ses ved udregningen af spær⁹:

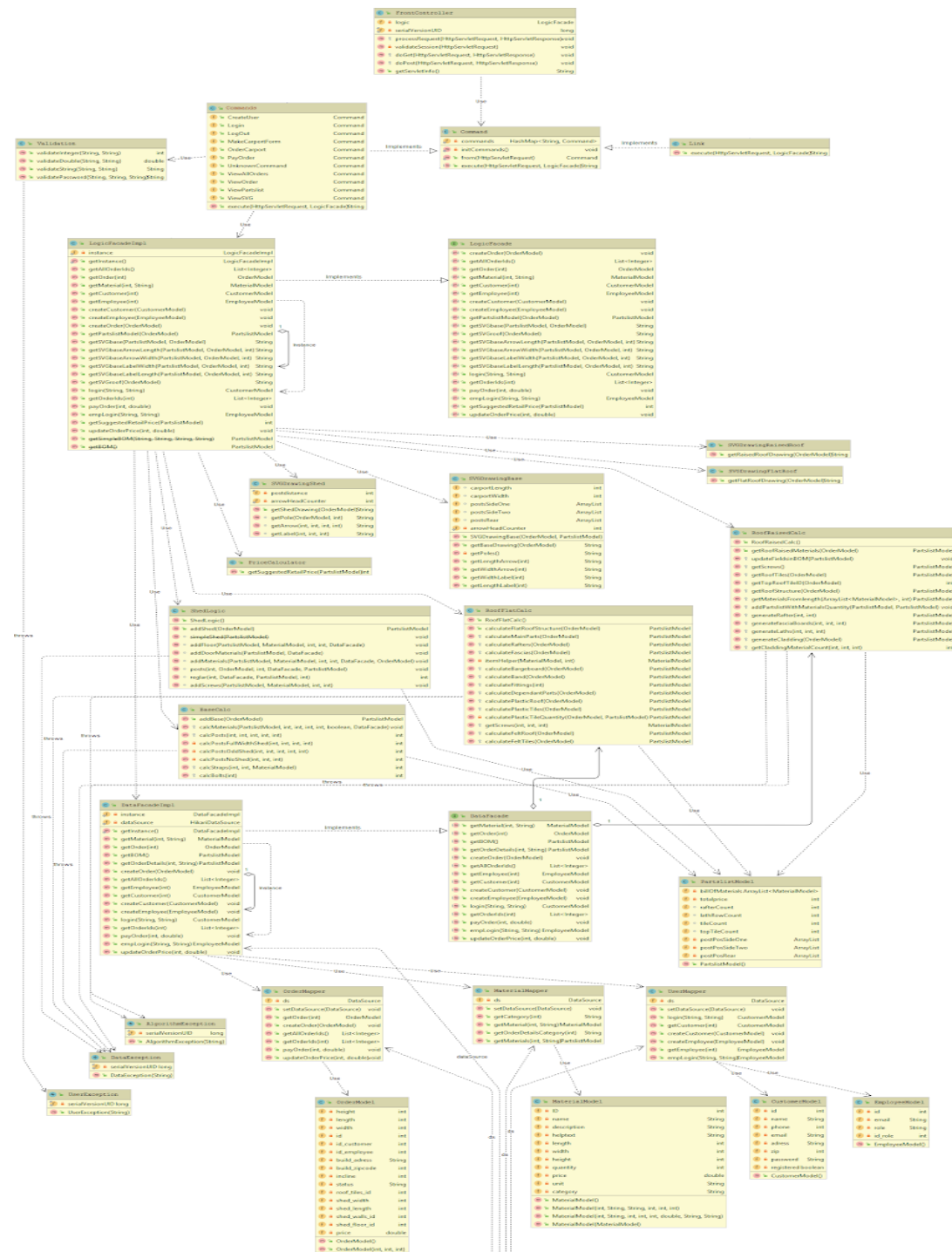
```
int rafterDist = roofLength / (rafterCount - 1);
int rafterPlacement = halfPadding;
for (int i = 0; i < rafterCount; i++) {
    String rafter = " <rect x=\"\" + rafterPlacement + "\""
        y=\"\" + halfPadding + "\"" width=\"6\" height=\"\"
        roofWidth+\"\"+\"style=\"fill:gainsboro;stroke:black;stroke-w
        idth:1;\"/> \n";
        rafterPlacement = rafterPlacement + rafterDist;
        stb.append(rafter); //stringbuilder
    }
}
```

Til sidst returneres en `String` bestående af alle SVG-elementerne, som så håndteres af `viewSVG.jsp`.

⁸ `SVGDrawingRaisedRoof.java`, linje 19-23

⁹ `SVGDrawingRaisedRoof.java`, linje 38-45

Klassediagram



Klassediagram beskrivelse

Klassediagrammet kan ses i [stor opløsning på vores GitHub](#)¹⁰.

På klassediagrammet er der nogle detaljer, der er blevet ændret for overskuelighedens skyld. Nogle klasser bliver vist uden fields (fordi der er ekstremt mange af dem, hvilket gør klasserne meget store så de fylder det hele) og nogle af klasserne bliver vist uden metoder, fordi de kun har getters, setters, equals, hash og toString-metoder, som igen fylder rigtig meget uden at vise noget særligt.

Klasser der vises uden fields:

- `BaseCalc.java`
- `ShedLogic.java`
- `RoofFlatCalc.java`
- `RoofRaisedCalc.java`

Klasser der vises uden metoder:

- Alle Model-klasserne.

Alle klasserne, der implementerer `Command.java` er blevet lagt sammen i én klasse, der har metoden `execute(HttpServletRequest request, LogicFacade logic)` for at vise, at de alle minder om hinanden, idét de kun har den ene metode og alle bliver brugt af `FrontController`n.

`Link.java` er blevet lagt separat, da den ikke helt har samme funktion som commands, men blot eksisterer for at dirigere programmet i særlige tilfælde, hvor brugeren skal have adgang til systemet selvom de ikke er logget ind (f.eks. på login-siden eller opret-bruger-siden).

`Link.java` returnerer kun et target og ikke andet (den har ingen logik og har derfor heller ikke nogen forbindelse til `LogicFacade.java`)

Særlige forhold

Session, request og validering

Når en bruger er blevet valideret ved login, oprettes der enten et `CustomerModel`-objekt (hvis det er en kunde der logger ind) eller et `EmployeeModel`-objekt (hvis det er en medarbejder der logger ind) i programmet. Dette objekt gemmes i session.

Vi anvender objekterne i session, hver gang en bruger passerer igennem vores `FrontController`, til at validere hvorvidt brugeren har en valid session. Hvis brugeren er på en side, der kræver at de er logget ind før de må se den, men de ikke har et objekt tilknyttet i session, så bliver de omgående sendt videre til en vores forside, hvor de godt må være, med beskeden "Du skal være logget ind."

Alle andre objekter gemmer vi på request-objektet, fordi det kun skal anvendes en enkelt gang, og vi undgår at session-objektet bliver kontamineret med en masse data der er ubrugelig på de andre jsp.-sider eller at der decideret ligger gammel data der kan ødelægge funktionaliteter der er afhængige af specifikke objekter. Der er nogle steder i systemet, hvor f.eks. et ordre-objekt kan bruges på mere end én side, men vi har besluttet ikke at lægge objektet i session, fordi vi i

¹⁰

<https://raw.githubusercontent.com/HrBjarup/Fog-Carport/master/extraFiles/Diagrams/Class%20Diagram%20Carport.png>

så fald skulle lave check, der fjerner den fra session, på alle andre sider som ikke skal bruge ordren, hvilket ikke ville være optimalt. Ordre bliver derfor genereret og lagt på request-objektet, hver gang en side skal bruge en ordre.

FrontController

FrontControlleren er servleten, der sørger for at modtage og håndtere requests og responses. Med metoden `processRequest()` som bliver kaldt, når servleten modtager en `GET`-request eller en `POST`-request, sørger FrontControlleren for at behandle brugerens input og forward'er til `index.jsp`. Dette foregår i flere steps:

Først valideres sessionen og hvis valideringen fejler, får man en fejlbesked og bliver sendt til en passende side.

Hvis valideringen lykkes, oprettes den passende `Command`, som returnerer det passende target (`String`) som FrontControlleren sætter på request-objektet. Derefter forwarder FrontControlleren til `index.jsp`. `Index.jsp` vil ud fra target have importeret den korrekte jsp-side, så clienten får den efterspurgte side frem:

```
<jsp:include page='${target}.jsp'></jsp:include>
```

Skulle brugeren efterspørge en ressource, som ikke eksisterer, får de standard 404-siden at se - vi har ikke en skræddersyet fejlside til den situation. Vi bruger ikke et catch-all system, hvor FrontControlleren fanger alle request der kommer til serveren, så derfor er der ikke speciel fejlhåndtering til situationer som f.eks. den ovennævnte, hvor brugeren efterspørger en ukendt ressource.

Når der bliver kastet exceptions længere nede i systemet, bliver disse exceptions fanget i FrontControlleren, som sørger for at forward'e det rigtige sted hen, alt efter hvilken slags exception den fanger. Exception-objektet vil altid have en passende fejlbesked, som bliver vist i Navbaren i toppen på siden.

FrontControllerens fejlhåndtering er dog ikke perfekt, da den samme slags exceptions kan blive kastet flere forskellige steder i systemet. FrontControlleren kigger ikke på, hvilken besked de forskellige exceptions har med sig, men kigger blot på hvilken slags exception, den modtager, og handler ud fra det.

Som eksempel bliver der kastet en `UserException` både hvis brugeren forsøger at logge ind med en ikke-eksisterende bruger, og når man prøver at lave en ny bruger, hvor brugernavnet allerede eksisterer i databasen, i forbindelse med man bestiller en carport. De to situationer foregår på vidt forskellige sider, men begge resulterer i, at man bliver sendt tilbage til login-siden.

For at forbedre FrontControllerens fejlhåndtering, kunne man enten forøge antallet af custom exceptions i systemet, eller få FrontControlleren til at kigge på de beskeder, der ligger i exception-objekterne og reagere ud fra dem. (Se mere i Exception-afsnittet nedenunder)

Exceptions

Vi har 3 custom exceptions:

- `UserException`
- `DataException`
- `AlgorithmException`

Vi kaster `UserException`, når der har været fejl i bruger-input. Hvis user f.eks. indtaster forkert info i login formen, så bliver der smidt en `UserException` med en custom message på dansk, der vises til bruger i .jsp a la "Kunden eksisterer ikke i databasen." Her er et eksempel fra vores validering på password:¹¹

```
if (password == null || password.isEmpty() ||
    password2 == null || password2.isEmpty()){
    throw new UserException(name + " må ikke være tom.");
}
if (!password.equals(password2)){
    throw new UserException("De to passwords stemte ikke overens.");
}
```

Når der fanges en `UserException` i vores `FrontController` så sendes user til `login.jsp`.

Vi kaster `DataException` når der er sket fejl i forbindelse med fremkaldelse af data fra databasen. Når der fanges en `DataException` eller `AlgorithmException` i vores `FrontController`, så sendes user til vores `errorMessage.jsp` side.

Vi kaster `AlgorithmException` når der har været fejl i logik-laget i forbindelse med et metodekald i algoritmerne. F.eks. bliver der lavet følgende tjek i en metode i en af algoritmerne:¹²

```
if (order == null || order.getLength() <= 0 ||
    order.getWidth() <= 0 || order.getIncline() <= 0) {
    throw new AlgorithmException("Ordren er ikke udfyldt korrekt.
Der bliver muligvis divideret med 0 eller der er indtastet felter med
værdier på 0 eller mindre.");
}
```

De beskeder, vi sender med i exceptions, står på dansk, fordi de skal vises til brugeren i brugergrænsefladen.

Vi havde overvejet at benytte exception-beskederne til mere end bare information til brugeren. Vi havde tænkt at fejkoder skulle være en del af beskederne, og at programmet ud fra fejkoderne kunne beslutte hvad der skulle gøres og hvor brugeren skulle redirectes. Desuden ville det også gøre det nemmere for os selv som udviklere at finde de steder, hvor tingene gik galt, hvis der var fejkoder i beskederne. Fejkoderne blev ikke en del af systemet, da der var andre ting, der blev prioriteret højere.

DataSource

Førhen har vi brugt `DriverManager` til at få forbindelse til vores SQL database. `DriverManager` understøtter dog ikke connection pools. Derudover skal man selv håndtere forbindelserne. En `DataSource` håndterer forbindelserne for en, og man kan få fat i en

¹¹ Validation.java, linje 108-115

¹² RoofRaisedCalc.java, linje 96-98

forbindelse i runtime, og man kan anvende en connection pool. Når man 'lukker' en connection fra en DataSource med connection pool, bliver den bare returneret til poolen, og er klar til at blive genbrugt.

Vi har i dette projekt endt op med at bruge HikariCP¹³ som vores DataSource. Undervejs i projektet har vi brugt forskellige måder til at forbinde til vores MySQL database på. De forskellige løsninger har præsenteret forskellige problematikker, men til sidst valgte vi at anvende Hikari. Den benchmarker højere end de andre datasources, og den har fornuftige default settings.

Hikari anvender en connection pool. Efter at have læst om Pool Sizing¹⁴, viser det sig at man kan slippe af sted med meget færre connections end først antaget. Vi har derfor konfigureret vores DataSource til max at anvende 3 connections ad gangen i sin pool, og selv dette er måske overdrevet.

I vores Mapper klasser kan man sette datasourcen. Dette er praktisk, fordi det gør det enormt let at skifte ens connection ud med en anden. Det gør det også let at teste via mocking. Man setter bare mock-forbindelsen på mapperen, og så kan man teste uden at ødelægge produktionsdatabasen.

I vores datafacade implementation har vi en singleton datasource, der sendes ned til mapperklasserne. Det sørger for, at vi bruger den samme connectionpool til kaldene til vores database. Det sikrer også, at det er den korrekte forbindelse der bliver brugt, siden den datasource vores facade anvender er konfigureret til vores produktionsdatabase.

JSTL og EL

I dette projekt har vi anvendt JSTL (JavaServer Pages Standard Tag Library) og EL (Expression Language) i vores .jsp sider. I forrige projekter hvor vi har anvendt .jsp sider, har vi brugt scriptlets til at opnå den samme funktionalitet.

JSTL

I forrige projekter, hvis vi ville iterere over et objekt i session/request, og printe det ud i table rows, så skulle man skrive følgende scriptlet:

```
<%
    Object object = (Object) request.getAttribute("object");
    if (object != null){
        List<String> list = object.getList();
        if (list != null){
            for(int i = 0; i < list.getSize(); i++){
                out.println("<tr>" + list.get(i) + "</tr>");
            }
        }
    }
}
```

¹³ HikariCP - <https://github.com/brettwooldridge/HikariCP>

¹⁴ <https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing>

%>

Men med JSTL kan vi bare skrive:

```
<c:forEach var="item" items="${object.list}">
    <tr>${item}</tr>
</c:forEach>
```

JSTL tilbyder mange funktioner som `forEach`, der kan hjælpe en med at kontrollere flowet i sine .jsp sider. I vores .jsp sider har vi primært anvendt `if` og `forEach` til at styre, hvad der skal vises på siden. Med `if` statements tjekker vi bl.a. for, om det er en customer eller en employee, der er logget ind, og så kan vi nemt styre, hvad de hver især skal se, også selvom de er på samme .jsp side.

Vi har brugt [JSTL's FMT](#) (`FormatNumber`) til visning af priser i dansk format. Vi har ved .JSP sider, som har skullet vise pris, sat `FMT.Locale` til `da_DK` og formatteret på følgende måde:

```
<span id="suggestedretailpriceTOSTRING"><fmt:formatNumber
value="${suggestedprice}" type="currency" currencySymbol="" /></span>15
```

EL

Før i tiden, hvis vi ville have 'name' fra et objekt fra request i sin .jsp, skulle man skrive følgende i en scriptlet:

```
<%
    Object object = (Object) request.getAttribute("object");
    if (object != null) {
        String name = object.getName();
        if (name != null) {
            out.print(name);
        }
    }
%>
```

Men med Expression Language og JSP 2.0 eller højere, så kan man bare skrive

`${object.name}` direkte i koden, og opnå samme resultat, uden brug af scriptlet. Det gør ens .jsp sider meget mere læsbare.

Der er dog nogle ting, der skal være på plads, før man kan få fat i objekter med EL:

1. Ens objekter skal overholde `JavaBean`¹⁶ standarderne.
2. Objektet skal ligge i tilgængeligt enten i request via `setAttribute` eller i session.

¹⁵ `viewOrder.jsp` linje 87

¹⁶ <https://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html> & <https://www.stackoverflow.com/a/3295517>

Man kan også søge specifikt i et bestemt scope via f.eks. `sessionScope`. Så kan man fange præcis, det man vil, hvis der ligger flere objekter med samme navn i forskellige scopes, hvilket vi gør brug af flere steder, f.eks. i `viewOrder.jsp`.

JavaScript

Vi har valgt at benytte JavaScript som en del af vores frontend, på trods af at det er en ny teknologi, som vi ikke har haft undervisning i før. Vi så nok fordele i at bruge JavaScript til at prioritere tid på indlæring af JavaScript. Ved brug af JavaScript har vi opnået betydeligt mere dynamiske sider, hvilket for det første giver mindre arbejde til serveren og for det andet giver brugeren en mere flydende oplevelse af hjemmesiden. Brugeren oplever ikke, at hjemmesiden hele tiden skal genindlæse, som konsekvens af de valg de tager - f.eks. når de vælger nogle bestemte dimensioner på carporten, som dermed afgør hvad dimensionerne på skuret må være. JavaScriptet sørger for at opdatere alle dropdown-menuer og hjælper brugeren med at sørge for at få udfyldt alle felter, før de sender formularen ind til serveren. JavaScriptet er derfor både med til at minimere antallet af fejl, der kunne opstå, og giver samtidig brugeren en mere behagelig oplevelse.

Fordi JavaScript er helt nyt for os, er der nogle dele af JavaScriptet, der kunne være skrevet kortere og mere effektivt. Der er nogle steder i JavaScript-filen, hvor der mangler refaktorering, men dette er igen blevet nedprioriteret til fordel for andre ting i projektet.

Alt vores JavaScript ligger i én fil, der bliver importeret på master-jsp-filen (`index.jsp`). Der er derfor tilfælde, hvor noget JavaScript der er lavet til én jsp-side skaber problemer på en anden side, hvor det egentlig ikke burde være loadet. Dette problem har vi løst på flere måder: Den mest brugte løsningen er at kigge efter bestemte elementer på siden og tjekke om de er null (ikke eksisterer):

```
if (document.getElementById("varpriceinput") !== null) {  
    doStuff();  
}
```

En anden løsning, som næsten blev udfaset, var at kigge på URL'en for at tjekke på parametrene:

```
if (window.location.toString().indexOf("command=viewSVG") !== -1)  
{  
    doStuff();  
}
```

Denne løsning kræver dog, at man sørger for altid at bruge GET-metoden i sine HTML-forms de steder, hvor der skal bruges JavaScript, for at sørge for at der rent faktisk ligger parametre i URL'en.

Alle disse tjek ville ikke være nødvendige, hvis vi havde mange separate JavaScript-filer, som vi kun importerede på de relevante jsp-sider. Vores manglende viden om JavaScript, kombineret med at vores tekniske vejleder ønskede JavaScript samlet i én fil, betød at vi formentlig ikke har løst det på den mest optimale måde.

Styling

Vi har holdt vores styling ens på tværs af alle .jsp siderne. Vi har f.eks. anvendt det samme design for vores knapper på alle .jsp siderne.

Vi har anvendt ¹⁷Bootstraps CSS og JS biblioteker til at style og designe vores projekt. Vi har importeret disse biblioteker i vores kode via deres CDN links.

Bootstrap er udviklet mobile-first, det vil sige at mange af deres elementer og klasser tilpasser sig selv, hvis man ændrer størrelsen på viewPort. Det understøtter også de fleste browsere, samt android og iOS. Deres værktøjer har gjort det meget nemmere at gøre vores hjemmeside pæn, og det har sparet os en masse tid på styling.

Derudover har vi prøvet at holde vores sider så fokuserede som muligt. Hver sides formål og funktion er tydelig. Der er ikke en masse forskellige funktionaliteter blandet sammen på siderne.

Alt er dog ikke lavet i Bootstrap. Vi har vores egen CSS fil (global.css), der indeholder yderligere tilpasning af visningen til brugeren. Den er 350 linjer lang, så der er en del ting vi har poleret og strømlinet for at gøre vores system pænt. Det meste af det er farve, padding, position og overskrivning af nogle Bootstrap-klasser.

Stykliste algoritmer

Kernefunktionen i programmet er sammensætningen af styklisten ud fra brugerens selvvalgte dimensioner. Dette var et meget logik-tungt krav, og vi endte med at dele kravet op i en simpel udgave og den avancerede udgave. I den simple udgave lavede vi beregningerne ud fra hardcodede dimensioner og med antagelsen om, at taget var flat. Dette var den første User Story vores Product Owner ønskede, at vi løste, da den fungere som et "proof of concept". Både i den simple og i den avancerede version af algoritmen opdelte vi den i fire dele og arbejdede på en del hver.

I den avancerede udgave stødte vi alle på de samme problemer, og specielt vil vi gerne fremhæve to problemer, som vi endte med at løse i én af de fire del-algoritmer og gerne ville have implementeret i de resterende. Dette fravalgte vi at gøre, da manglen af denne implementering ikke betød, at algoritmerne ikke virkede, de var bare ikke så optimale, som de kunne have været.

De to problemstillinger vi stødte på og løste i `RoofRaisedCalc.java` var:

1. En hel del af materialerne i virksomhedens katalog findes i flere længder, det vil sige at samme materiale kan fås i eksempelvis 2 meter, 3 meter og 5 meter.
Vi ønskede i vores beregninger at tage højde for dette, så hvis man skulle bruge 6 meter i alt af et materiale, så ønskede vi at der blev beregnet 1 af dem på 5 meter og 1 af dem på 2 meter, frem for 2 af dem på 5 meter, både for at kunden ikke skal betale for mange meter ekstramateriale og for i det hele taget at undgå spild.
2. Den anden problemstilling bestod i at nogle af materialerne var bærende og kunne ikke bare beregnes ud fra den samlede længde af antallet af den type materiale, da dette er baseret på, at man kunne save materialerne over, som algoritmen dikterer. Dette er ikke

¹⁷ <https://getbootstrap.com/>

muligt i f.eks. spær, her skal et spær helst være sat sammen af hele materialer, da spærret bærer vægten af taget og helst ikke skal have nogle svage punkter.

Løsning af problemstilling 1

Metoden `getMaterialsFromlength()` tager en Arrayliste af de materialer (`MaterialModel`), der er at vælge mellem og den samlede længde (`length`), der skal findes materialer til, ind som parametre. Arraylisten sorteres descending, så det længste materiale ligger på index 0, da resten af beregningen er afhængig af, at materialerne ligger i denne orden.

Der bliver oprettet et `HashMap`, og alle materialerne i `ArrayListen` bliver sat ind med materialet som key og en `Integer`, der repræsenterer antallet, som value. Alle values bliver sat til 0 her.

Beregningen er lavet i to loops, et while-loop yderst der fortsætter så længe, der er en `restLength` tilbage (længden af de fundne materialer bliver trukket fra `restLength`, som starter med at være lig `length`) og et forloop baseret på antallet af materialer i den sorterede Arrayliste.

Det forloopet kigger på for at se hvilket materiale, der er bedst, er en ratio, der er lig `restLength` divideret med materialelængden `material.getLength()` for materialet på det index, forloopet er kommet til. Hvis divisionen er lig præcis 1 betyder det at `material.getLength()` er lig `restLength`. Er divisionen større end 1, f.eks. kunne den være 8,3, så betyder det at der skal 8 `material.getLength()` plus en rest til at dække `restLength`. Er divisionen mindre end 1, f.eks. 0,7, betyder det at `restLength` er kortere end den `material.getLength()` vi kigger på.

For at kunne se om et materiale er bedre end andre materialer, så gemmes materialet og dets ratio ved første gennemløb af forloopet (gemmes i `ratioBest` og `bestMaterial`). Før det første gennemløb er `bestMaterial` null og `ratioBest` er -1. Så vi antager at det første materiale er det bedste. I de resterende gennemløb bliver der kigget på hvilket materiales ratio, der er tættest på 1, og dette materiale og dets ratio gemmes så.

Når forloopet er færdigt, har vi fundet det bedste materiale til den givne `restLength`. For at finde antallet (`materialAmount`) af det `bestMaterial`, skal `ratioBest` rundes ned til nærmeste heltal, hvis den er større end 1 og rundes op til nærmeste heltal, hvis den er mindre end 1.

Herefter bliver den fundne `bestMaterial` value hevet ud af `HashMappet`, den ovenfor beregnede `materialAmount` bliver lagt til denne value, og herefter bliver den nye value sat tilbage i `HashMappet`. Til sidst i while-loopet bliver `ratioBest` sat til -1 igen, og `material.getLength()` ganget med `materialAmount` bliver trukket fra `restLength`. Det hele starter så forfra, så længe `restLength` er større end 0.

Til sidst i metoden, når while-loopet er færdigt, bliver alle materialerne i `HashMappet` med en value større end 0 hevet ud, får sat deres quantity til deres tilsvarende value, og bliver tilføjet til `ArrayListen` i den del-stykliste (`PartslistModel`), der bliver returneret af metoden.

Løsning af problemstilling 2

I denne [metode](#), `addPartslistWithMaterialsQuantity()` bliver der taget to styklister (`PartslistModel`) ind, den ene er den der skal tilføjes fra (`calcBOM`) og den anden er den der skal tilføjes til (`returnBOM`). Der bliver kigget på Arraylisterne i begge disse `PartslistModel` (`calcList` og `returnList`).

Hvis `returnList` er tom, så tilføjer vi en kopi af det først materiale i `calcList`. Vi bruger en `boolean` til at tjekke, om vi har tilføjet et materiale-antal (`quantity`) fra `calcList` til `returnList` - dette sker kun, når det samme materiale findes i begge lister.

Hvis et materiale i `calcList` findes i `returnList`, så opdateres `quantity` og `totalPrice` i `returnList`. Hvis materialet i `calcList` ikke findes i `returnList`, så tilføjes en kopi af materialet som det ser ud i `calcList` (med `quantity` og `totalPrice` osv.).

På denne måde kan man lægge flere styklister sammen, og de vil få opdateret deres `quantity` korrekt, hvilket medfører, at der kun er én variant af hvert materiale i den endelige `PartslistModel` med den korrekte `quantity` sat.

Robusthed, kobling og kohæsion

I designet af vores program har vi forsøgt at tage højde for de forskellige klassers ansvarsområder, hvor let det vil være at udskifte enkelte dele af programmet, og hvor afhængige de enkelte dele er af andre dele af programmet.

Indkapsling

Vi har anvendt Facade Pattern ved brug af Interfaces til at indkapsle vores mere komplekse klasser og metoder i koden. Via vores facader har man adgang til simple metoder med tydelige navne og JavaDocs for hvad, deres funktion er. Det gør koden mere overskuelig og læsbar. Dette er især vigtigt når man arbejder i grupper. Eksempelvis kan et gruppemedlem have siddet og implementeret en avanceret algoritme i logik-laget, men der er en tydelig `public facade-metode` tilgængelig, mens resten af metoderne i klasserne er `protected` og utilgængelige udefra. Dette gør at andre gruppemedlemmer stadig kan anvende koden, uden nødvendigvis at have læst hele algoritmen selv. For eksempel vores `getPartslistModel()` metode i `LogicFacade`, er ret nem at forstå, på trods af de mange hundrede linjer kode bagved. Man afleverer en ordre til metoden, og så returnerer den en stykliste.

Kobling

For at opnå så lav kobling som muligt, har vi forsøgt at holde klassers kendskab og brug af andre klasser til et minimum.

I data-laget er de forskellige `Mappers` adskilt og kender ikke til hinanden. Vores `connection` (`DataSource`) til databasen bliver sat i `DataFacadeImpl` og bliver givet med når `mappers` bliver oprettet. Dette gør det nemt både at lave ændringer eller skift af `mappers` og vores måde at lave en `connection` på.

I logic-laget anvender de fleste klasser og metoder en `OrderModel` til at hive den data, der skal bruges til beregning, ud af databasen og returnerer en `PartslistModel` til `LogicFacade`.

I presentation-laget gør vi brug af Command Pattern, hvilket sørger for at de enkelte `Commands` ikke kender til hinanden og let kan skiftes ud eller ændres.

Kohæsion

Vi har begrænset de mange forskellige klassers ansvarsområde så meget som muligt. I logic-laget har en klasse kun ansvaret for udregningen af en specifik del af carporten eller tegningen. I data-laget har de enkelte mappers ansvaret for at mappe hvert deres databaseobjekter, og i presentation-laget er hver command ansvarlig for en specifik funktion, her dog med få undtagelser. I presentation-laget har det været svært at bibeholde enkelte ansvarsområder og samtidig overholde Command Pattern og begrænse session-brug. F.eks. Er `login.jsp` nødt til udover at have ansvar for at logge en bruger ind også at have ansvar for at sætte en liste med ordre-id'er for at kunne vise det, som den første side, der bliver logget ind på, skal indeholde. Ligeledes er flere `commands` nødt til at beregne en `Partslist` for at kunne beregne prisvisning og tegning.

Validering af brugerinput

Validering af brugerinput foregår dels hos client og dels på serversiden.

Clientside

Vi benytter både JavaScript og HTML til at holde styr på det input, som brugeren kan give. Validering med JavaScript foregår sådan, at diverse dropdowns som brugeren kan vælge værdier i, bliver udfyldt og opdateret alt efter hvad brugeren tidligere har valgt. Hvis brugeren f.eks. ikke har valgt, at han vil have et skur i sin carport, så kan han heller ikke vælge størrelsen på skuret.

De forskellige dropdowns bliver opdateret dynamisk med JavaScript alt efter brugerens input, sådan at f.eks. skurets mulige dimensioner aldrig bliver større end selve carportens dimensioner.

Det er ikke muligt at indsende kontaktformularen uden at have udfyldt alle de påkrævede inputfelter - dette er lavet med JavaScript. Der er yderligere et tjek på email-inputfeltet med html.

Serverside

På serveren (i javakoden) har vi en klasse, der bruges til at tjekke den data brugeren har angivet. Hvis nu brugeren skulle have formået at sende en tom værdi med i en ordre, så bliver det tjekket og håndteret af valideringsklassen på serveren. Brugerens input bliver valideret som det første, når en form bliver sendt ind til serveren - på den måde håndterer vi fejl så tidligt som muligt i systemet, så vi undgår at der bliver sendt tom eller null data videre ned i systemet. Og skulle det alligevel ske er der flere steder i beregningerne hvor der igen bliver tjekket på værdierne og givet en fejlbesked til frontenden hvis de er ugyldige.

JavaDoc

Der er skrevet intern dokumentation i form af [JavaDoc](#) til langt det meste af programmets klasser og metoder for at gøre koden så let som muligt at forstå og samtidig dokumentere funktionaliteten.

Status på implementation

Det var ikke muligt at nå alle Firma-kravene i inden for deadline, men vi har nået langt de fleste kernefunktioner, som kunden ønskede. Se [bilag 5](#) for screenshots af det færdige program eller se det [live](#).¹⁸

Opnået systemfunktionalitet

Følgende Firma-krav er opfyldt:

- Ansatte skal kunne logge ind
- Ansatte skal kunne se alle ordrer
- Ansatte skal kunne markere en ordre som betalt
- Ansatte skal kunne se stykliste for en ordre
- Ansatte skal kunne se tegninger for en ordre
- Kunder skal kunne oprette en bruger
- Kunder skal kunne indtaste Carport med ønskede mål og indhente et tilbud uden bruger
- Kunder skal kunne indtaste Carport med ønskede mål og indhente et tilbud med bruger
- Kunder skal kunne se tidligere ordrer hvis de har en bruger
- Kunder skal kunne se stykliste for en ordre hvis de er logget ind
- Kunder skal kunne se tegning for en ordre hvis de er logget ind
- Kunder skal kunne betale for deres ordre

Disse krav er indeholdt i de færdiggjorte User Stories beskrevet i Krav-afsnittet.

Kundens brug af systemet

- Forsiden af webapplikationen er en bestillingsformular, som kunden kan udfylde med deres ønskede mål og materialevalg til en carport. De kan vælge at oprette en bruger under denne bestilling, som giver dem adgang til at logge ind. Når en kunde er logget ind, kan de se deres ordre (og eventuelt tidligere ordre), klikke ind på den specifikke ordre og se den vejledende pris, se en eventuel tilbudspris og betale for deres ordre. Når en ordre er betalt af kunden, får de med det samme mulighed for at se styklisten til deres ordre og en tegning ud fra ordrens specifikke mål.

Medarbejderens brug af systemet

- I venstre side af navigationsbaren er der en login-knap som medarbejdere (og kunder med en bruger) kan bruge til at komme til login-siden. Når en medarbejder er logget ind på systemet, har de med det samme adgang til at se en liste over alle ordrer. Hvis en medarbejder klikker sig ind på en specifik ordre, der endnu ikke er betalt, så har de

¹⁸ www.bjarup.com

mulighed for at beregne dækningsgraden på en tilbudspris og sætte den ønskede tilbudspris på ordren i stedet for den vejledende salgspris. Sælger har ligesom kunden mulighed for at se styklisten og tegningen på samme måde som kunden, men sælger er ikke begrænset af om der er betalt for ordren for at få adgang til disse.

Overblik over implementeret CRUD i forhold til tabellerne i Databasen

customers-table:

Create -	Implementeret
Read -	Implementeret
Update -	Ikke implementeret
Delete -	Ikke implementeret

employees-table:

Create -	Ikke implementeret
Read -	Implementeret
Update -	Ikke implementeret
Delete -	Ikke implementeret

materials-table:

Create -	Ikke implementeret
Read -	Implementeret
Update -	Ikke implementeret
Delete -	Ikke implementeret

orders-table:

Create -	Implementeret
Read -	Implementeret
Update -	Implementeret
Delete -	Ikke implementeret

Ikke-opnået systemfunktionalitet

Følgende Firma-krav er ikke nået:

- Admin skal kunne oprette ansatte
- Admin skal kunne opdatere ansattes gemte informationer
- Admin skal kunne slette ansatte
- Admin skal kunne se statistik over ansattes salg
- Admin skal kunne se statistik over en afdelings salg
- Ansatte skal kunne se en faktura for en ordre
- Ansatte skal kunne se tilgængelige materialer
- Ansatte skal kunne redigere og slette materialer
- Ansatte skal kunne ændre materialer i en genereret stykliste
- Kunder skal kunne se en faktura

Kundens brug af systemet

- Det er ikke muligt for kunden at se forskel på deres ordre i det samlede ordre-view afhængigt af, om de er betalt, dato for bestilling og lignende.
- Det er ikke muligt for kunden at se en decideret regning for en specifik ordre, dette er blevet delvist lagt sammen i den specifikke ordres view, men var en ønsket funktionalitet.

Medarbejderens brug af systemet

- Det er ikke muligt for en medarbejder at tilpasse styklisten, efter den er genereret.
- Medarbejderen har ikke mulighed for at se de materialer, der er tilgængelige og har derfor heller ikke mulighed for at tilføje nye materialer eller ændre i de materialer, der ligger i forvejen.
- Det er ikke muligt for Admin at oprette nye medarbejdere uden at gøre det direkte i databasen, og det er heller ikke muligt at opdatere den info, der ligger på den enkelte medarbejder.
- Der er ingen funktionalitet for medarbejderstatistik eller afdelingsstatistik. Planen for dette var, at en medarbejder med admin-role skulle kunne se statistik for sine medarbejders produktivitet.

Sikkerhed i forbindelse med login

Passwords gemmes i databasen uden nogen form for kryptering. Vi ville gerne som minimum have fået implementeret hashing af passwords, inden de bliver gemt.

Datatyper

Der er blevet brugt Double til Price i databasen. Dette endte med at skabe en del problemer i logik-laget, da mange del-styklisters skulle lægges sammen. Grundet tidsnød blev det løst ved at runde alle priser op. Når nu vi har med penge at gøre, ville det nok have været bedre at bruge en anden datatype, f.eks. BigDecimal eller bare integers. Vi kunne også have gjort brug af følgende: `java.util.Currency` eller det udvidede `JodaMoney-library`.

Der er brugt varchar til Status-kolonnen i Ordre-table, her burde vi have brugt enums, da der på nogle af .jsp-siderne bliver tjekket på, hvad indholdet af den er, og der bliver vist forskellige ting alt efter ordrens Status.

Logging

Logging, helt kort, er at gemme informationer om en fejl, der er opstået i koden, så man senere kan læse hvad der skete. I sine catch-clauses indsætter man øverst en logger der gemmer informationer om fejlen i en logfil, før man håndterer fejlen yderligere. Dette er en funktionalitet, vi gerne ville have nået, da det ville have ført til et mere vedligeholdelsesvenligt system og gjort det lettere at spore fejl, der eventuelt måtte opstå efter programmet er gået i produktion. Logging var ikke en høj prioritet sammenlignet med funktionaliteter for brugeren, så det blev skubbet nedad af et par omgange, og blev aldrig færdig-implementeret. Der blev gjort et kort forsøg på branch "Logging", som man kan se på, hvis man vil. Der ligger nogle levn i pom.xml fra dette.

Hvad man normalt gemmer i sin logfil, når der opstår en fejl i koden:

- Tidsstempel
- Fejlbesked
- Klassen hvor fejlen opstod
- Fejlens niveau
- Stack Trace

Log Levels

Når man logger et event, kan man tildele det forskellige niveauer, alt efter hvor alvorligt det er: "Severe" er det højeste niveau, og "finest" er det laveste niveau. Det kan hjælpe en udvikler med at prioritere, hvilke issues der skal løses først.

Stack Traces

Når Java-compileren støder på et problem i koden, så opretter den en StackTrace. Det hjælper udvikleren til præcist at identificere, hvor fejlen opstod i programmet. Derfor er det en god idé at logge hele StackTracen. Så når koden er gået i produktion, og der opstår en fejl, kan man åbne logfilen og se hele StackTracen og forhåbentlig regne ud, hvor fejlen stammer fra, og hvad dens root cause er.

Database normalisering og relationer

I vores oprindelige databasedesign overholdt databasen de to første normalformer, men under udviklingen har vi tilføjet og ændret tables og kolonner, med tanke på at det skulle være midlertidigt og har så ikke haft tid til at ændre det inden vores deadline.

Som det ser ud nu, kan det diskuteres om vi overholder første normalform, idet alle kolonner indeholder én værdi, men vi har et table, materials, der har tre kolonner med helptext som egentlig er inde for det samme domæne .

Med hensyn til anden normalform bryder vi den i vores materials-table, da de tre helptext-kolonner er partielt afhængige af id_material. Løsningen til både første og anden normalform, som vi ikke nåede at implementere, ville i vores tilfælde have været et separat tabel til helptexts og så en relationstabel imellem det nye table og materials-table.

Tredje normalform har været en udfordring for os, da vi i vores orders-table har en indbyrdes afhængighed mellem incline og roof_tiles: inclines værdi afgør hvad roof_tiles kan være. Dette er altså en transitiv funktionel afhængighed, mellem to non-prime attributter (to attributter, der ikke er primærnøgle-kandidater) og dette bryder den 3. normalform. Løsningen på dette ville have været et separat table med incline og roof_tiles_id.

Vi har kun én til mange relationer imellem vores tabs i databasen. Vi har forsøgt at undgå én til én relationer, da der ikke er nogen grund til ikke bare at have det stående i det samme table, og mange til mange relationer ville vi have undgået med relationstabeller mellem de tables, vi har nu, og de ekstra tables vi ville have indført til helptext f.eks.

Yderligere mangler

Udenfor firmakravene har der været et par ekstra User Stories eller funktionaliteter, som vi gerne ville have haft med.

Ansatte skulle gerne kunne:

- Se en detaljeret ordreoversigt i stedet for kun ordre ID'er.
 - Hertil skulle vi have lavet en OrderMapper-metode, der hentede ordre-objekter.
- Bestille en carport for en kunde.
 - Dette er delvist implementeret, og formen herfor kan findes ved at klikke på Fog's logo i Navbaren. Der mangler bare felter til e-mail, navn og telefonnummer. Hvis man forsøger sig med at gennemføre ordren, får man en fejl på manglende e-mail.
- Sortere ordrer efter status, ejerskab (ansatte tildelt ordren) og overtage disse fra andre ansatte.
 - Her var det ideen, at nye ordrer skulle være herreløse og kunne overtages af først tilkomne sælger.

Mindre fejl og mangler

- Man kan ikke indtaste en tilbudspris som kommatal, fordi form-typen er sat til number uden at have `step="0.01"` (to decimaler). Samtidig så laver vi valideringen på en Integer i stedet for en double¹⁹:
 - `int finalPrice = validation.validateInteger`
 - `(request.getParameter("finalPrice"), "Pris felt");`
- Validation-klassen bliver ikke udnyttet godt nok. Vi kunne sagtens have valideret mere herinde, f.eks. ordrer, og sørget for at udnytte den alle steder. (viewPartslist.java mangler den blandt andet)
- Validation-klassen smider altid `UserException`. (Som returnerer brugeren til `login.jsp`)
- Vi har forsøgt så godt som muligt at overholde navnekonventioner, men der er enkelte steder hvor der er glippet.
- `SVGDrawingFlatRoof.java` samt dennes algoritme `RoofFlatCalc.java` har ved en fejl lagt ekstra længde (for afvanding) til bredden i stedet for den bagerste side som det fremgår af byggemanualen.
- De forskellige del-algoritme klasser beregner selv hvor mange skruer, der skal bruges i deres underdel. Eftersom skruer kommer i pakker af flere hundrede styk, så ville det have været optimalt at lave en fælles endelig beregning af antallet af pakker af skruer i f.eks. `PartslistModel`. Som det ser ud nu, får den endelige styklisten en del for mange pakker af skruer af denne grund.
- Algoritmen for udregning af fladt tag `FlatRoofCalc` udregner i skrivende stund en ekstra lille tagplade med en quantity på 0 og tilføjer denne til styklisten. Dette var faktisk fikset i [pull #185](#) men blev ved en fejl stashet før merge.
- Algoritmen for grundkonstruktionen tager højde for, om skurets bredde er den samme som carportens bredde. Dog er muligheden for at have et skur med fuld bredde blevet

¹⁹ ViewOrder.java, linje 54

fjernet i JavaScriptet, hvilket resulterer i, at der er en del af algoritmen, der aldrig bliver brugt. Dette kan fikses ved at ændre nogle få ting i JavaScriptet, men problemet blev opdaget for sent.

- Det skal være en mulighed at bestille et skur uden belægning og/eller uden gulv, men dette er ikke en mulighed (man er tvunget til at vælge gulv og beklædning - ellers kan man ikke trykke "bestil"). Igen tager algoritmen for skuret højde for situationer, hvor brugeren ikke vil have gulv og/eller beklædning, men JavaScriptet tillader det ikke.

Mange af vores mangler kunne have været fikset som beskrevet ovenfor, men vi valgte at deploye den endelige version af projektet d. 28/05-2019 og ikke røre mere ved det derfra. Dette blev gjort for at sørge for, at al funktionalitet fungerede som det skulle og samtidig for at teste, at vores JDBC, noget som vi har haft mange problemer med, virkede efter hensigten på dropletten.

Prioritering

Vi har prioriteret SCRUM meget højt i dette projekt, da det er et nyt koncept for os og en helt ny og anderledes måde at arbejde på i et team. Det har blandt andet ført til at vores programmering blev nedprioriteret en smule specielt i starten af projektet, men det har samtidig gavnet os rigtig meget i forhold til overblik, samarbejde og at få lavet et produkt, der er funktionsdygtigt.

Vi har prioriteret at teste vores logik meget og så har vi eksperimenteret en del med Mockito-test af to af vores mappers. Denne prioritering er foretaget, da det fra start blev gjort klart for os at kernefunktionaliteten, som Product Owner ønskede, var udregningen af styklisten på baggrund af ordreinformation.

Vi har prioriteret dokumentation i form af denne rapport, samt intern dokumentation i form af JavaDoc og kommentarer i koden, frem for sidste øjeblikks ekstra-funktionalitet. Vi mener, det er vigtigt at dokumentere vores arbejde og vores kode, frem for at få streget et ekstra krav af listen, da det gør vores arbejde nemmere at vedligeholde, hjælper os selv til en bredere forståelse af al koden i programmet og giver os tid til at reflektere mere over de valg, vi har truffet undervejs.

Test

Vi har testet vores logik-klasser og vores dataMapper klasser²⁰

Prioritering af test

Vi har prioriteret at teste vores logik-lag, samt det der kunne blive tid til i datalaget og herudover fokusere på at få så meget funktionalitet med i vores system som muligt. Det er første gang, vi har arbejdet med Mockito-tests, og derfor var det en prioritet for os at få det med, men det krævede en omstrukturering af hele vores datalag og det design pattern, vi havde valgt, så det var ikke muligt at nå mere inden for den givne tid.

²⁰ <https://hrbjarup.github.io/Fog-Carport/jacoco/>

Anvendte testmetoder

Vi har anvendt automatiserede JUNIT test til vores logik-lag, og vi har lavet automatiserede Mockito tests til vores DataMapper klasser. Resten af programmet er testet manuelt.

Fordelen ved automatiserede tests

Hver gang man har implementeret nogle ændringer, kan man køre de automatiserede tests. Så ved man kort tid efter om koden stadig fungerer hvis alle tests lyser grønt.

Og hvis der er nogle tests der fejler, så ved man præcis hvor fejlen er opstået, og ofte også hvad fejlen er, hvis man sørger for at teste tit. Så er fejlen et sted i den kode, man har skrevet siden sidst man testede.

Manuelle test

Vi har lavet en del manuelle test både af vores connection til databasen, commands og javascript. Dette er primært foregået med `System.out.print()` i Java-delen af programmet specielt i forbindelse med mange kald til databasen i et loop for at genskabe og spore connection-fejl og `console.log()` i vores JavaScript-fil for at teste at vores metoder der virkede efter intentionen.

Herudover har vi ihærdigt prøvet at "broke" systemet fra frontenden og generelt prøvet at fremprovokere fejl for at finde ud af, hvor vi skulle sætte ind med validering på f.eks. bruger-input.

Følgende dele af programmet er testet

BaseCalc.java

Følgende metoder er testet direkte:

- `addBase()`
- `calcPosts()`
- `calcStraps()`
- `calcBolts()`

Følgende metoder bliver testet indirekte gennem de andre test:

- `calcMaterials()`
- `calcPostsFullWidthShed()`
- `calcPostsOddShed()`
- `calcPostsNoShed()`

Testene dækker alle metoderne i klassen. Nogle af metoderne kan kun testes indirekte, da de er `private` og eksisterer som hjælpemetoder for en anden metoder. Dette gælder:

`calcPostsFullWidthShed()`, `calcPostsOddShed()` og `calcPostsNoShed()` der alle bliver kaldt af metoden `calcPosts()`. Metoden `calcMaterials()` bliver kaldt af hovedmetoden `addBase()`.

Testene er parametriserede og der er i alt 10 set af testværdier, som alle testene kører igennem.

Konstruktøren i testklassen sørger for at oprette opdatere de værdier, som testmetoderne skal bruge. Selvom det optimale er, at have fuldstændigt uafhængige test, har dette ikke været muligt i forbindelse med testing af `BaseCalc.java` da klassen har metoder, der opdaterer fields, som senere skal bruges af andre metoder. For ikke at skulle kalde flere forskellige metoder i den samme test, har vi valgt at importere `org.junit.FixMethodOrder`; i testklassen, hvilket giver os mulighed for at køre testene i en bestemt rækkefølge.

Instansen af `BaseCalc.java` bliver oprettet som et `static` field i testklassen, så dens egne interne værdier ikke bliver nulstillet for hver gang en test har kørt.

Nogle af testene er blevet navngivet med et nummer (f.eks. `test1CalcPosts()`) så rækkefølgen af testene er sikret.

Der er ikke blevet lavet nogen negative tests for `BaseCalc.java` (test der forventer at fejle) da vi har haft begrænset tid. Vi har prioriteret at teste de tilfælde, hvor der ikke kommer ulovlige værdier ned til algoritmen, hvilket naturligvis ville skabe fejl - det input som brugeren submitter, bliver i forvejen tjekket længere 'oppe' i systemet bl.a. af `Validation.java` så udvidelse af testklassen for `BaseCalc.java` er blevet nedprioriteret.

RoofFlatCalc.java

Følgende metoder er testet:

- `CalculateRafters()`
- `CalculateFascias()`
- `CalculateBargeboard()`
- `CalculateFittings()`
- `CalculateBand()`
- `CalculatePlasticTiles()`
- `GetScrews()`

Følgende metoder er ikke testet:

- `calculateFlatRoofStructure()`
- `calculateMainParts()`
- `itemHelper()`
- `calculateDependantParts()`
- `calculatePlasticRoof`
- `calculatePlasticTileQuantity()`

Ovenstående bliver enten testet indirekte igennem de testede metoder eller anses som en hjælpemethode der ikke gør meget andet end at samle materialerne og returnere dem.

- `calculateFeltRoof()` - ikke testet.
- `calculateFeltTiles()` - ikke testet.

Udover disse bliver der også testet for edge-cases, såsom skæve vidder og exception throwing. Typisk sammenlignes der med værdier beregnet ud fra egne beregninger, men i et enkelt tilfælde sammenlignes objekter.

Der blev forsøgt brug af `setUpClass`, men det gav ikke tilfredsstillende resultater, så derfor blev alt sat instantieret i konstruktøren.

RoofRaisedCalc.java

Følgende metoder er testet:

- `getRoofRaisedMaterials()`
- `getScrews()`
- `getRoofTiles()`
- `getRoofStructure()`
- `getMaterialsFromlength()`
- `addPartslistWithMaterialsQuantity()`
- `generateRafter()`
- `generatefasciaBoards()`
- `generateLaths()`
- `generateCladding()`
- `getCladdingMaterialCount()`

De eneste metoder i `RoofRaisedCalc.java`, der ikke er testet, er `updateFieldsinBOM()` og `getTopRoofTileID()`. Disse er ikke testet, da den ene blot kalder en set-metode i den endelig `PartslistModel` for at opdatere variabler, og den anden er en hardcoded switch-case, der sørger for, at der bliver valgt den samme type top-teglsten som de teglsten, der er valgt til resten af taget.

Alle de andre metoder i denne klasse er testet, og de fleste også med tests, der forventes at fejle. En metode der kan fremhæves, er den metode, der bliver brugt af fire andre metoder til at beregne hvilke og hvor mange materialer, der skal bruges. Denne metode er `getMaterialsFromlength()`, som er testet med både normale værdier, store og små værdier, tom materialeliste, negativ længde, nul (zero) længde, og med materialelængder på nul (zero).

Den metode er essentiel for at type og mængde af materiale bliver beregnet korrekt i følgende metoder:

- `generateRafter()`
- `generatefasciaBoards()`
- `generateLaths()`
- `generateCladding()`

De eneste metoder der beregner deres egne materialer, er `getRoofTiles()` og `getScrews()`, da de henholdsvis skal tage højde for mere end én dimension (teglsten skal beregnes både ud fra bredde og længde) og skruer beregnes på tværs af alle materiale-genererende metoder i klassen og har ingen dimensioner, der kan bruges til beregning i `getMaterialsFromlength()`.

Af de resterende metoder der er testet i denne klasse, bliver der lavet edge-case tests og fail-tests, der hvor der er mulighed for, at en `AlgorithmException()` bliver kastet.

ShedLogic.java

Følgende metoder er testet:

- 1. `addDoorMaterials()`
- 2. `addFloor()`
- 3. `addScrews()`
- 4. `posts()`
- 5. `reglar()`
- 6. `simpleShed()`

I `ShedLogic` er der en overmetode, der kalder disse undermetoder. Vi har valgt at teste undermetoderne med små enkle tests, så det er nemt at vedligeholde, hvis man skifter en undermetode ud senere. Der er en solid dækningsgrad, siden alle undermodulerne, der bruges til at lave styklisten for skuret, er testet.

UserMapper.java

Følgende metoder er testet:

- 1. `login()`
- 2. `createCustomer()`

Vi har brugt en metode til at teste både login af user og creation af user. Vi har testet dette med Mockito. En customer bliver oprettet, og der bliver skrevet hvad mock connection skal returnere (arrange). Så sætter vi customer ind i den mock'ede database og hiver vedkommende ud igen med login (act). Derefter tjekker vi, om den customer vi har fået ud af login, er den samme som forventet (assert).

I forhold til dækningsgrad så er der endnu ikke tests af `empLogin()` og `createEmployee()`, så cirka halvdelen af mapperen er testet. Dog er metoderne meget ens.

OrderMapper.java

Følgende metoder er testet:

1. `createOrder()`
2. `getOrder()`

Vi har testet de to vigtigste metoder i `OrderMapper` via testen `testCreateGetOrder()`. Tre metoder er ikke testet. En af dem henter alle IDs ud for orders. En henter alle IDs ud for orders for en bestemt kunde. Og den sidste skifter status på en ordre. Hvis vi havde mere tid, ville denne også være blevet testet. Men de to vigtigste metoder blev prioriteret for tests.

Bilag

Bilag 1 - Github og Taiga links

Wiki før sprints:

Uge 1: <https://github.com/HrBjarup/Fog-Carport/wiki/Week-1-%7C-Business-demands>

Uge 2: <https://github.com/HrBjarup/Fog-Carport/wiki/Week-2-%7C-SCRUM>

Sprint 1

Github milestone link for issues and pull requests:

<https://github.com/HrBjarup/Fog-Carport/milestone/1?closed=1>

Taiga sprint taskboard:

<https://tree.taiga.io/project/maltemagnussen-fog/taskboard/sprint-one-170?kanban-status=1592823>

<https://tree.taiga.io/project/maltemagnussen-fog/us/8?milestone=227420>

Wiki:

Uge 3, 4: <https://github.com/HrBjarup/Fog-Carport/wiki/Week-3-&-4-%7C-Holiday-&-First-sprint>

Sprint 2

Github milestone link for issues and pull requests:

<https://github.com/HrBjarup/Fog-Carport/milestone/2?closed=1>

Taiga sprint taskboard:

<https://tree.taiga.io/project/maltemagnussen-fog/us/111?milestone=229524>

Wiki:

Uge 5: <https://github.com/HrBjarup/Fog-Carport/wiki/Week-5-%7C-Second-sprint>

Sprint 3

Github milestone link for issues and pull requests:

<https://github.com/HrBjarup/Fog-Carport/milestone/3?closed=1>

Taiga sprint taskboard:

<https://tree.taiga.io/project/maltemagnussen-fog/taskboard/sprint-3-5193?kanban-status=1592823>

Wiki:

Uge 6: <https://github.com/HrBjarup/Fog-Carport/wiki/Week-6-%7C-Third-Sprint>

Sprint 4

Github milestone link for issues and pull requests:

<https://github.com/HrBjarup/Fog-Carport/milestone/4?closed=1>

Taiga sprint taskboard:

<https://tree.taiga.io/project/maltemagnussen-fog/taskboard/sprint-4-3495?kanban-status=1592823>

Wiki:

Uge 7: <https://github.com/HrBjarup/Fog-Carport/wiki/Week-7-%7C-Fourth-sprint>

Sprint 5

Github milestone link for issues and pull requests:

<https://github.com/HrBjarup/Fog-Carport/milestone/5?closed=1>

Taiga sprint taskboard:

<https://tree.taiga.io/project/maltemagnussen-fog/taskboard/sprint-5-2337?kanban-status=1592823>

Wiki:

Uge 8: <https://github.com/HrBjarup/Fog-Carport/wiki/Week-8-%7C-Fifth-Sprint>

Bilag 2 - Sekvensdiagram [Partslist]

Sekvensdiagram for viewPartslist.

Diagrammet er skrevet i *PlantUML*.

PlantUML koden kan findes [her](#) og i samme mappe findes billedet i PNG-format. ([her](#)) (det anbefales at åbne billedet i en ny fane).

Der er også en forsimplet version tilgængelig [her](#). Formålet med denne er rydde op i alle metodekaldene fra *Calculators* (ShedLogic, RoofFlatCalc, RoofRaisedCalc, BaseCalc) som ligner hinanden og derfor skabe et nemmere overblik.

Bilag 3 - Sekvensdiagram [SVG]

Sekvensdiagram for viewSVG.

Diagrammet er skrevet i *PlantUML*.

PlantUML koden kan findes [her](#) og i samme mappe findes billedet i PNG-format. ([her](#)) (det anbefales at åbne billedet i en ny fane)

Partslist-generering er forkortet med vilje. For en mere detaljeret gennemgang, se *bilag 2*. Der bliver foretaget validering af orderID i begyndelsen af *ViewSVG*-commanden. Dette er ikke inkluderet i diagrammet.

Bilag 4 - Navigationsdiagram

Navigationsdiagram for Fog Carport hjemmesiden.


Diagrammet er skrevet i *PlantUML*.

PlantUML koden kan findes [her](#) og i samme mappe findes billedet i PNG-format. ([her](#)) (det anbefales at åbne billedet i en ny fane)

Bilag 5 - Screenshots af Programfunktionalitet

Forside

Kunde- og Medarbejder-visning

Bestil Carport

Log ind

Velkommen til Fog Quickbyg

Vi gør dine carportdrømme til virkelighed

Bestil Carport

Længde i cm

Vælg længde

Bredde i cm

Vælg bredde

Taghældning

Vælg Taghældning

Tagtype

Vælg Tagtype

☐ Vælg skur

Kontaktinfo

E-mail (Dette bliver dit brugernavn)

Eksempel@mail.dk

Navn

Navn

Adresse

Adresse

Telefon

Telefonnummer

Postnummer

Postnummer

Jeg vil gerne have en bruger

Bestil tilbud



Login

Kunde- og Medarbejder-visning



Bestil Carport

 Log ind



Fog Carporte - Login

LOG IND

[Opret Bruger](#)



Ordreoversigt

Kunde-visning



Bestil Carport

Se dine ordrer

Hans Jensen

 Log ud

Ordreoversigt

Kundeoplysninger

Hans Jensen

56781249

hans@email.dk

Kundenummer: 72

Ordrenr: 70



Medarbejder-visning - Uddrag



Se alle ordrer

camilla@fog.dk

Log ud

Medarbejderoplysninger

Medarbejdernr: 4

Personalemail: camilla@fog.dk

Ordreoversigt

Ordrenr: 1

Ordrenr: 2

Ordrenr: 3

Ordrenr: 4

Ordrenr: 5

Ordrenr: 6

Ordrenr: 7

Ordrenr: 8

Ordrenr: 9

Ordrenr: 10

Ordrenr: 11

Ordrenr: 12

Ordrenr: 13

Ordrenr: 14

Ordrenr: 15

Ordrenr: 16



Ordrevisning

Kunde-visning - Tilbud givet, ordre ikke betalt



Bestil Carport | Se dine ordrer

Hans Jensen

[Log ud](#)

Kunde informationer

Kundenummer: 72

Hovedvejen 12 3600

Hans Jensen

56781249

hans@email.dk

Fog Trælæst og Byggecenter

Ordre information

Medarbejdernummer: 1

Medarbejder kontaktmail: admin@fog.dk

Ordrenummer: 70

Status: Afventer betaling

Carport				Skur			
Bredde	Længde	Hældning	Tagsten	Bredde	Længde	Vægmaterialer	Gulvmaterialer
5400mm.	4200mm.	5 grader	B & C Dobbelt Tagsten (Blå)	2700mm.	1200mm.	25x200 mm. trykimp. Bræt (Birk)	30x250mm alm. planke (Eg)

Vejledende salgspris: ~~+6.528,00~~ DKK

Tilbudspris: **15.995,00** DKK

[Betalt ordre](#)



Medarbejder-visning - Tilbud givet, ordre ikke betalt

Fog TRÆLAST OG BYGGECENTER

Se alle ordrer

admin@fog.dk

Log ud

Kunde informationer

Kundennummer: 72
Hovedvejen 12 3600
Hans Jensen
56781249
hans@email.dk

Fog Trælast og Byggecenter
Ordre information

Medarbejdernummer: 1
Medarbejder kontaktemail: admin@fog.dk
Ordrenummer: 71
Status: Færdigbehandlet

Carport				Skur			
Bredde	Længde	Hældning	Tagsten	Bredde	Længde	Vægmaterialer	Gulvmateriale
5400mm.	4200mm.	5 grader	B & C Dobbelt Tagsten (Blå)	2700mm.	1200mm.	25x200 mm. trykimp. Bræt (Birk)	30x250mm alm. planke (Eg)

Vejledende salgspris: ~~16.526,00~~ DKK

Tilbudspris: **15.995,00** DKK

Dækningsgrad for vejledende salgspris: 50.0%

Dækningsgrad for tilbudspris: 45.2%

Afgiv tilbud

Indkøbspris: 11.019,00 DKK

Dækningsgrad: 36.1%

Send tilbud

Ved kontant betaling eller betaling via bankoverførsel, så bekræft modtagelse af betaling

Bekræft modtagelse af betaling

Se Styklister

Se Tegninger

Kunde- og Medarbejder-visning - Ordre betalt

[Bestil Carport](#)[Se dine ordrer](#)**Hans Jensen**[Log ud](#)

Kunde informationer

Kundenummer: 72
Hovedvejen 12 3600
Hans Jensen
56781249
hans@email.dk

Fog Træløst og Byggecenter

Ordre information

Medarbejdernummer: 1
Medarbejder kontaktmail: admin@fog.dk
Ordrenummer: 70
Status: Færdigbehandlet

Carport				Skur			
Bredde	Længde	Hældning	Tagsten	Bredde	Længde	Vægmaterial	Gulvmateriale
5400mm.	4200mm.	5 grader	B & C Dobbelt Tagsten (Blå)	2700mm.	1200mm.	25x200 mm. trykimp. Bræt (Birk)	30x250mm alm. planke (Eg)

Ordren er betalt: 15.995,00 DKK

[Se Styklister](#)[Se Tegninger](#)

Stykliste

Kunde-visning - Uddrag



Bestil Carport | Se dine ordrer

Hans Jensen

[Log ud](#)

Antal	Enhed	Beskrivelse	Hjælpetekst	Længde mm	Bredde mm	Højde mm
1	Sæt	Stalddørsgreb 50x75	Til dør i skur	0	750	500
1	Stk	38x73 mm. taglægte T1	Til z på bagside af dør	5400	73	38
2	Stk	T-hængsel 390mm	Til dør i skur	0	390	0
11	Stk	30x250mm alm. planke (Eg)	Til gulvet i skuret	4800	250	30
1	Pakke	4,5x50mm Skruer (300 stk)	Til montering af inderste bræt ved beklædning	0	0	0
40	Stk	25x200 mm. trykimp. Bræt (Birk)	Helptext not available	5400	200	25
1	Pakke	4,5x50mm Skruer (300 stk)	Til montering af inderste bræt ved beklædning	0	0	0
1	Pakke	4,5x70mm. skruer (400 stk)	Til montering af yderste bræt ved beklædning	0	0	0
1	Stk	97x97 mm. trykimp. Stolpe	Helptext not available	3000	97	97
3	Stk	45x95 Reglar ubh.	Løsholter i skur	3600	95	45
2	Stk	45x95 Reglar ubh.	Løsholter i skur	2400	95	45



Medarbejder-visning - Uddrag

Varenr.	Antal	Enhed	Beskrivelse	Hjælpetekst	Længde mm	Bredde mm	Højde mm	Stk. pris
17	1	Sæt	Stalddørgreb 50x75	Til dør i skur	0	750	500	85,00
12	1	Stk	38x73 mm. taglægte T1	Til z på bagside af dør	5400	73	38	39,00
18	2	Stk	T-hængsel 390mm	Til dør i skur	0	390	0	55,85
51	11	Stk	30x250mm alm. planke (Eg)	Til gulvet i skuret	4800	250	30	75,00
27	1	Pakke	4,5x50mm Skruer (300 stk)	Til montering af inderste bræt ved beklædning	0	0	0	167,75
56	40	Stk	25x200 mm. trykimp. Bræt (Birk)	Helptext not available	5400	200	25	42,75
27	1	Pakke	4,5x50mm Skruer (300 stk)	Til montering af inderste bræt ved beklædning	0	0	0	167,75
26	1	Pakke	4,5x70mm. skruer (400 stk)	Til montering af yderste bræt ved beklædning	0	0	0	185,00
4	1	Stk	97x97 mm. trykimp. Stolpe	Helptext not available	3000	97	97	65,00
7	3	Stk	45x95 Reglar ubh.	Løsholter i skur	3600	95	45	27,50
6	2	Stk	45x95 Reglar ubh.	Løsholter i skur	2400	95	45	22,00
40	40	Stk	45x95 Reglar ubh.	Løsholter i skur	0	0	0	22,00

Tegning

Kunde- og Medarbejder-visning

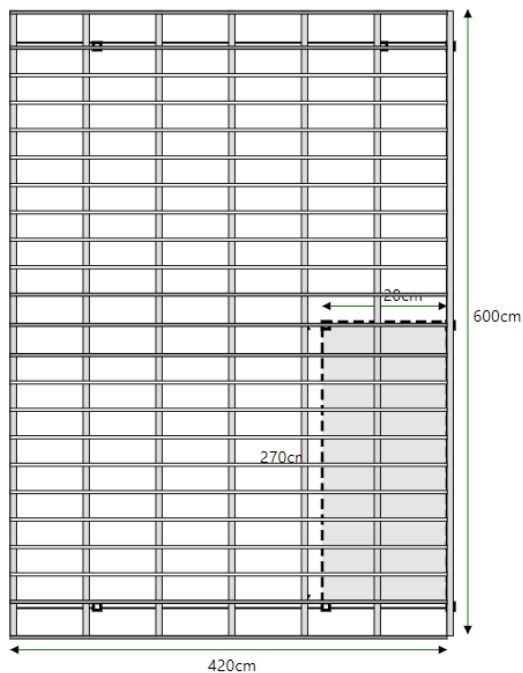
Tegninger af Carport.

Underkonstruktion

VIS GEM

Tagkonstruktion

VIS GEM



Bilag 6 - [Bills](#) som det fremgik hos Fog

I videoen fra Fog fremgår det at de sender en regning til kunden før de får adgang til stykliste og tegninger. Sådan foregår det også i den endelige version af programmet, men vi har ikke et bills-objekt at gemme i databasen.

Konstruktionsbeskrivelse:

Tilbud nr. 13623

Carport 4,2 x 6,6 mtr.

Redskabsrum 2,1 x 2,1 mtr.

Beklædning: K33 16x100mm

Gulv: Husmandsgulv 21x110mm

Tag:

Spærtype: Spær med rejsning

Remtype: Spærtræ 45x195mm

Tagmateriale: Benders brun

Salgspris:

kr. 22328

Bilag 7 - Ressourcer

Carport byggevejledning - **fladt tag**:

https://datsoftlyngby.github.io/dat2sem2019Spring/Modul4/Fog/CP01_DUR.pdf

Carport byggevejledning - **rejt tag**:

https://datsoftlyngby.github.io/dat2sem2019Spring/Modul4/Fog/CAR01_HR.pdf