Fog Carport

Mads Benjamin Ribberholt - **cph-mr415@cphbusiness.dk** - MadsBenjaminR

Daniel Herlev Moldavsky - **cph-dm189@cphbusiness.dk** - moldamolda

Jeppe Rønning Koch - **cph-jk469@cphbusiness.dk** - JeppeKoch

Laith Alkaseb - **cph-la356@cphbusiness.dk** - MINGO-INC

Klasse A

29/4 -24/5 2024

Indhold

Links

Til repository

https://github.com/MadsBenjaminR/eksamenfogren

Til demovideo

https://youtu.be/2-0YVdSpi4g

Til kørende version af applikation

https://fogeksamenren.danielherlev.dk/

Admin-login information

admin login: bruger: jeppe, kodeord: jj

eller

admin login: bruger: awd, kodeord: awd

Link til risikoanalyse

https://docs.google.com/spreadsheets/d/1I-vkLsyp7umx2zwr9B2hh7MLRPbQB7VAZsVsxyoGSLQ/edit#gid=0

Indledning

Projektet har til formål, at forme et program til Johannes Fog A/S's (omtales i rapporten som 'Fog') website ud fra givne krav og behov. Det sigter mod at skabe en intuitiv og brugervenlig hjemmeside, der gør det muligt for kunden at lave en forespørgsel på en carport ud fra kundens egne mål. Sælgeren skal modtage og redigere forespørgslen, samt sende et tilbud, som kunden skal kunne acceptere.

Denne rapport vil dokumentere hele projektets livscyklus og fastslå refleksioner over processen og vores resultater.

Baggrund

Følgende beskrivelse af virksomheden er Fog's egen:

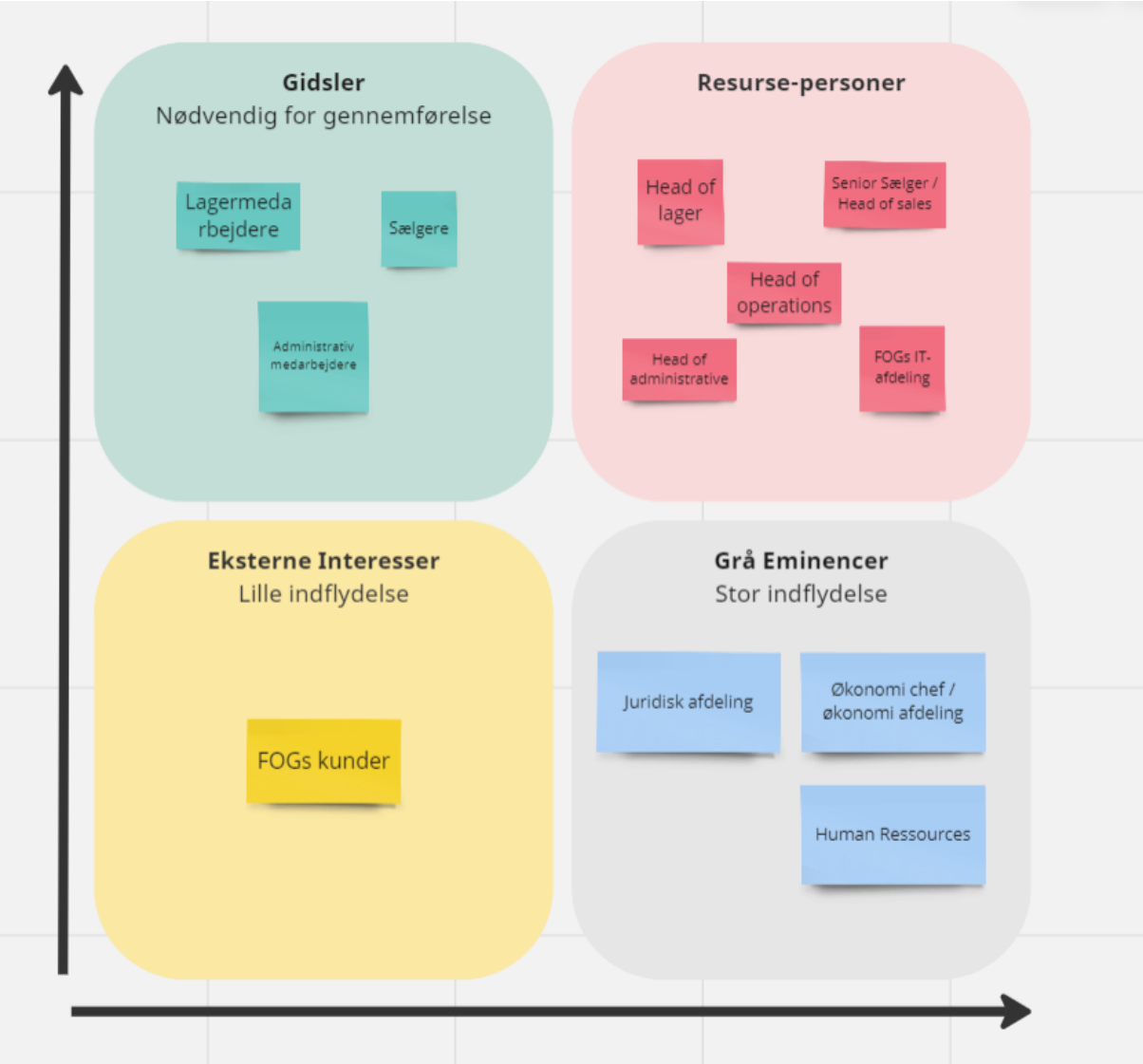
"I Fog får du kvalificeret rådgivning til den bedste løsning. Vi hjælper dig med at få overblik og komme i mål med dit byggeprojekt. Lige fra hvad der er godt at vide, til vejledning i valg af materialer. I Trælasterne kan du finde træ og byggematerialer til alle slags opgaver lige fra carporte til hegn og terrasser. Vi har et stort udvalg, hvor du kan vælge mellem mange muligheder, både hvad angår størrelser, materialer og kvaliteter."

Kundes krav

- Kunden kan tilpasse deres egen carport ved at justere dens størrelse.
- Kunden kan ændre målene på sin tilpassede carport efter at have logget ind, og systemet gemmer de indtastede mål.
- Når kunden tilpasser en carport, kan denne oprette en bruger, så Fog kan bruge dennes data i forbindelse med et tilbud.
- Sælgeren kan se detaljer om kundens ønskede carport, herunder materialeliste, indkøbspris, vejledende pris og dækning, for at kunne give kunden det bedst mulige tilbud.
- Sælgeren har adgang til et administrationspanel, hvor denne kan tilføje nye materialer til forskellige kategorier og indtaste beskrivelser og priser for hvert nyt materiale.
- Kunden får en kvittering med en oversigt over sit køb og en detaljeret materialeliste, når kunden foretager et køb, så han kan afgive ordren og se, hvad de forskellige materialer er til.
- Sælgeren kan se en liste over indgående kundeforespørgsler, redigere og godkende disse forespørgsler og sende et tilbud til kunden.
- Kunden kan se en tegning af carporten ovenfra, baseret på egne mål, for at få et overblik over carportens konstruktion.

Virksomheden

Interessentanalyse



Hvor **X-aksen** er: Interessenter, der har Indflydelse på projektet

Og

Hvor **Y-aksen** er: Interessenter, der er påvirket af projektet.

Eksterne interesser

Vi vurderer at **Fogs kunder** har en lille indflydelse på projektet, og samtidig ikke er direkte påvirket af det. Derfor er denne interessent placeret som ekstern interesse.

Gidsler

Arbejdsgangen for både **lagermedarbejdere**, **sælgere** og **administrative medarbejdere** i organisationen, vil potentielt blive påvirket af det nye system. De har samtidig en lille indflydelse på projektet.

Grå eminencer

Virksomhedens **økonomiafdeling / Økonomichef** har en stor indflydelse på projektet, når det kommer til det økonomiske aspekt. De kan have merit til at planlægge budgettet og dermed en maksimal udgift for Fog. De er dog ikke direkte påvirket af systemet, da de højst sandsynligt ikke vil anvende det i deres virke. Dette gælder også organisationens **Human Resources** afdeling. De kan til gengæld have en indflydelse på den del af systemets design, der har at gøre med medarbejdernes overenskomst og overordnede rettigheder. For eksempel fysiske forhold. Denne vil formentligt blive understøttet af virksomhedens **Juridiske afdeling**, der derfor også placeres som grå eminence.

Ressourcepersoner

Lagerchefen har en afgørende position i projektet på grund af deres omfattende ansvar for lagerdrift og logistik. Dennes direkte deltagelse i implementeringsfasen er uundværlig, da den direkte påvirker deres operationelle arbejdsgange og ansvarsområder.

På samme måde har **Senior Sales Manager/Head of Sales** en betydelig indflydelse på projektet. De er ansvarlige for den strategiske ledelse af salgsmetoder, -resultater, og deres input er afgørende, da implementeringen af det nye system har potentiale til at omstrukturere salgsprocesserne.

Driftschefen har også central position inden for den organisatoriske ramme og har indflydelse på virksomhedens effektivitet. Chefens direkte involvering i implementeringsfasen giver dem magt til at forme beslutninger om processer og arbejdsgange.

Risikoanalyse

De identificerede risici dækker et bredt spektrum af Fogs aktiviteter herunder juridiske afdeling, udviklingsteamet, og kundeservice.

Risk ID	Aktivitet	Risiko	Alvor	Sandsynlighed	Risikoniveau	Plan for forebyggelse	
US-1	Kunde kan skræddersy sin egen carport	Ønsket carport er ikke mulig at lave	Uønsket	Usandsynlig	Medium	Giv kunden nogle rammer at arbejde inden for	
US-2	Kunden kan ændre sine mål i forespørgselsprocessen	Fejl i systemet kan ødelægge ordre eller gemme forkerte mål	Uacceptabelt	Usandsynlig	Lav	Sørg for at projektet er fejlsikret	
US-3	Kunde kan oprette bruger ved en forespørgsel	Man kan miste en kunde der ikke vil oprette en bruger	Uønsket	Muligt	Medium	Accepter at man kan miste enkelte kunder for systemets bedre	
US-4	Admin kan se økonomiske detaljer for en forespørgsel	Sælger kan misbruge informationer	Uønsket	Muligt	Lav	Hyr kun de bedste sælgere	
US-4	Kunde kan få tilbud om montering	Kan gå imod fog forretningsmodel, og tilbuddet er udefrakommende	Uønsket	Muligt	Medium	Bestil tømrer man kender/stoler på og hav alle følge samme	
US-5	admin/sælger kan tilføje materialer	forskellige rolletildelinger kan skabe interne problemer	Tolereres	Muligt	Medium	Hierarki kan sørge for kun nødvendige ændringer/tilføjelser	
US-6	Kunde får kort beskrivelse/kvittering ved køb	ved kun en kort beskrivelse kan der gå meget galt i kommunikation inden levering	Acceptabel	Muligt	Lav	levering dobbeltjekkes og kommunikation med kunde er muligt	
US-6	Stykliste	Kunde får fat på stykliste og kan finde materialer billigere	Uacceptabelt	Usandsynlig	Høj	Kunden får ikke stykliste, før de betaler	
US-7	Sælger kan redigere/godkende tilbud til kunde	fejl i kommunikation, tilbuddet der sendes kan gå galt eller sælger kan give forkert tilbud	Uacceptabelt	Usandsynlig	Medium	Sælger dobbeltjekker tilbud og sikrer sig kunde modtager	
	Fogs kunder i interessantanalyse	nye system kan give problemer hos kunder	Uønsket	Muligt	Medium	test systemet inden launch	
	Sygdom hos udviklerteam	Sygdom kan forsinke processen	Tolereres	Sandsynligt	Medium	ved langtidssygdom kan en vikar findes, ved kort sygdom kan hjemmearbejde 'måske' fungerer	
	Teamet mangler ønsket kompetencer	Dele af systemet kan kun skabes med speciele kompetencer som ikke er til stede i team	Uønsket	Usandsynlig	Høj	dele af systemet kan outsources	
	Uforudsigelige forsinkelser	Kan skubbe aflevering af projekt tilbage	Uønsket	Muligt	Høj	overarbejde	
	Juridisk afdeling	Juridiske udfordringer kan hindre dele af projektet, måske hele projektet	Uønsket	Usandsynlig	Ekstrem	Kør hele projektet igennem med juridiske afdeling for godkendelse	
	Økonomi afdeling	Der er ikke penge til alle ønskede dele af systemet	Uønsket	Usandsynlig	Høj	Planlæg i forvejen og ellers må man skære visse dele af projektet ud	
	Human Ressources	Interne konflikter i virksomheden	Tolereres	Muligt	Lav	Sørg for alle er på samme side når det gælder projektet	
	Head of operations	Deres syn går imod resten af virksomheden	Uønsket	Usandsynlig	Lav	Sørg for alle er på samme side når det gælder projektet	
	Head of sales	Kan have svært ved det nye system	Tolereres	Muligt	Lav	Giv dem et lynkursus	
	Head of Lager	Igen problemer med det nye system	Tolereres	Muligt	Lav	Giv dem et lynkursus	
	Head of administratives	Dårlig planlægning	Uønsket	Muligt	Medium	Planlæg i forvejen	
	Lovændring	Nye love der påvirker udviklingsteamet	Uønsket	Usandsynlig	Medium	Følg med i mulige nye love	
	Nye sikkerhedskrav	Nye sikkerhedskrav der kan gøre projektet større	Uønsket	Muligt	Medium	Altid byg projeketet så sikkert som muligt	

<u>Risiciene er vurderet ud fra:</u>

- Alvor: Angiver, hvor alvorlig en risiko er for virksomheden, fra "Tolereres" til "Uacceptabelt".
- Sandsynlighed: Chancen for at risikoen opstår, fra "Usandsynlig" til "Muligt".
- Risikoniveau: En samlet vurdering baseret på alvor og sandsynlighed.

Den samlede risikoprofil viser, at de mest kritiske områder er fejl i systemet og manglende kompetencer i udviklingsteamet. Juridiske udfordringer udgør også en ekstrem risiko for projektets gennemførelse. Hvis for eksempel vi kommer til at lække personfølsomme oplysninger.

<u>Prioritering af risici</u>

Risici med høj eller ekstrem alvor skal prioriteres højst. Dette inkludere jurdiske problemer, sygdom hos udviklerteamet og andre uforudsigelige forsinkelser

Teknologivalg

- IntelliJ IDEA 2023.3.5
- Java 11.0.20
- HTML
- CSS
- Maven 17
- Jetty 11.0.20
- Javalin 6.1.3

- Javalin-rendering 6.1.3
- Thymeleaf 3.1.2.RELEASE
- Thymeleaf-extras 3.0.4.RELEASE
- PostgreSQL 42.7.2
- PgAdmin 4 version 8.3
- Hamcrest 2.2
- Kotlin 1.9.22
- Junit 5.10.2
- HikariCP 5.1.0
- Jackson 2.17.0-rc1
- Slf4j 2.0.12

Hosting

- Docker 4.27.2
- Linux VM 7.0.12

User stories og Acceptance criteria

User Story 1

SOM kunde

VIL JEG skræddersy min egen carport

FOR AT kunne tilpasse størrelsen på carporten

AC 1

- GIVET kunden er på hjemmesiden
- NÅR kunden trykker på knappen for at skræddersy sin carport
- SÅ får kunden adgang til konfigurationsværktøjet til at tilpasse carporten
- NÅR kunden bruger konfigurationsværktøjet
- SÅ kan kunden indgive mål på carporten

User Story 2

SOM kunde
VIL JEG ændre carportens mål, når jeg skræddersyer min carport, efter jeg er logget ind

FOR AT få en carport med mine ønskede mål

AC 2

- GIVET jeg er kunde
- NÅR jeg er logget ind
- SÅ bliver mine mål gemt, men jeg får muligheden for at ændre dem

User Story 3

Som kunde

VIL JEG oprette en bruger når jeg skræddersyer min carport

FOR AT Johannes Fog A/S kan bruge mine oplysninger i forbindelse med et tilbud

AC 3

- GIVET jeg er kunde
- NÅR jeg skræddersyer en carport
- SÅ opretter jeg en bruger

User Story 4

SOM sælger

VIL JEG se detaljer om kundens ønskede carport, blandt andet stykliste, indkøbspris, anbefalet pris og dækningsgrad

FOR AT give kunden det bedst mulige tilbud for begge parter

AC 4

- GIVET jeg er logget ind som sælger/admin
- NÅR jeg har valgt en carports forespørgsel
- SÅ skal det være muligt at kunne se stykliste, indkøbspris, anbefalet pris og dækningsgrad for carporten.

User Story 5

SOM admin/sælger

VIL JEG have adgang til et administrationspanel

FOR AT kunne editere materialer til forskellige kategorier

AC 5

- GIVET jeg er logget ind som admin/sælger
- NÅR jeg navigerer til administrationspanelet
- SÅ kan jeg tilføje nye materialer til forskellige kategorier
- OG jeg kan angive beskrivelser og priser for hvert nyt materiale

User Story 6

SOM kunde

VIL JEG modtage en kvittering med en oversigt over mit køb samt en stykliste

FOR AT kunne bogføre ordren og se hvad de forskellige materialer skal bruges til

AC 6

- GIVET Jeg er en registreret kunde i systemet
- NÅR jeg foretager et køb
- SÅ modtager jeg en kort kvittering, der indeholder et overblik over de købte produkter samt den samlede pris
- OG kvitteringen vises på hjemmesiden efter køb
- NÅR varerne ankommer
- SÅ forventer jeg at modtage en detaljeret stykliste over de købte produkter

User Story 7

SOM sælger

VIL JEG se en liste over indkomne forespørgsler fra kunder

FOR AT redigere og godkende forespørgsel, samt sende et tilbud til kunden

AC 7

- GIVET Jeg er logget ind som sælger
- NÅR jeg ønsker at se en liste over indkomne forespørgsler fra kunder
- SÅ vises en oversigt over alle indkomne forespørgsler
- GIVET Jeg har åbnet en forespørgsel fra en kunde
- NÅR jeg ønsker at redigere og godkende forespørgslen
- SÅ har jeg mulighed for at ændre i prisen og godkende forespørgslen
- NÅR jeg har godkendt forespørgslen
- SÅ genereres et tilbud
- OG tilbuddet sendes direkte til kundens side

User Story 8

SOM kunde

VIL JEG se en tegning af carporten oppefra, ud fra mine mål

FOR AT danne et umiddelbart overblik over carportens konstruktion

AC 8

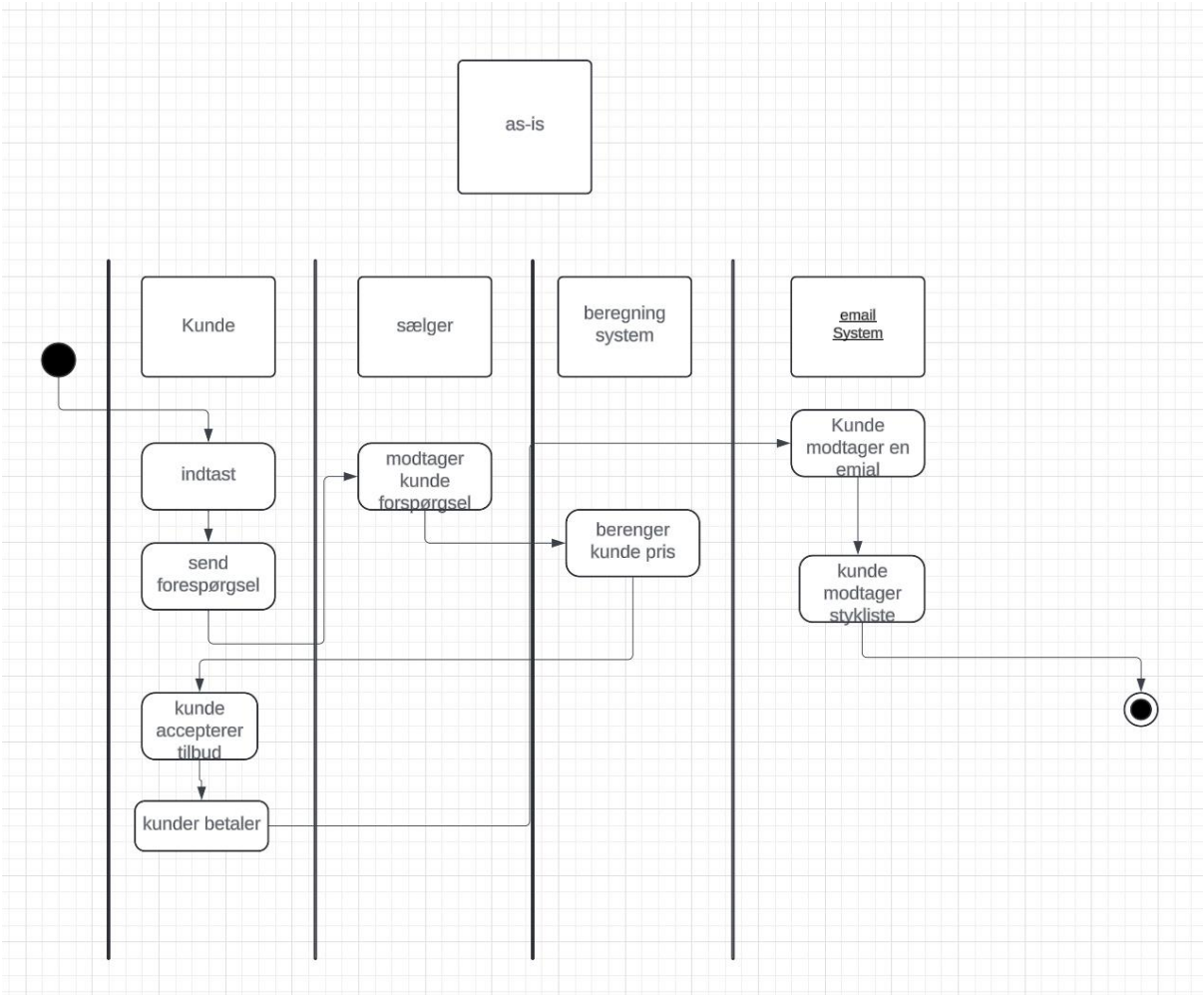
- GIVET Jeg er logget ind som kunde
- NÅR jeg kigger på tilbud fra sælger
- SÅ vises en dynamisk SVG, tegnet ud fra mine givet mål

Aktivitetsdiagram

Det er vigtigt at nævne, at 'as-is'-modellen er udarbejdet ud fra hvordan Fogs system ser ud på nuværende tidspunkt. "to-be"-modellen er designet ud fra hvordan vi ønsker at processen ideelt set skal være.

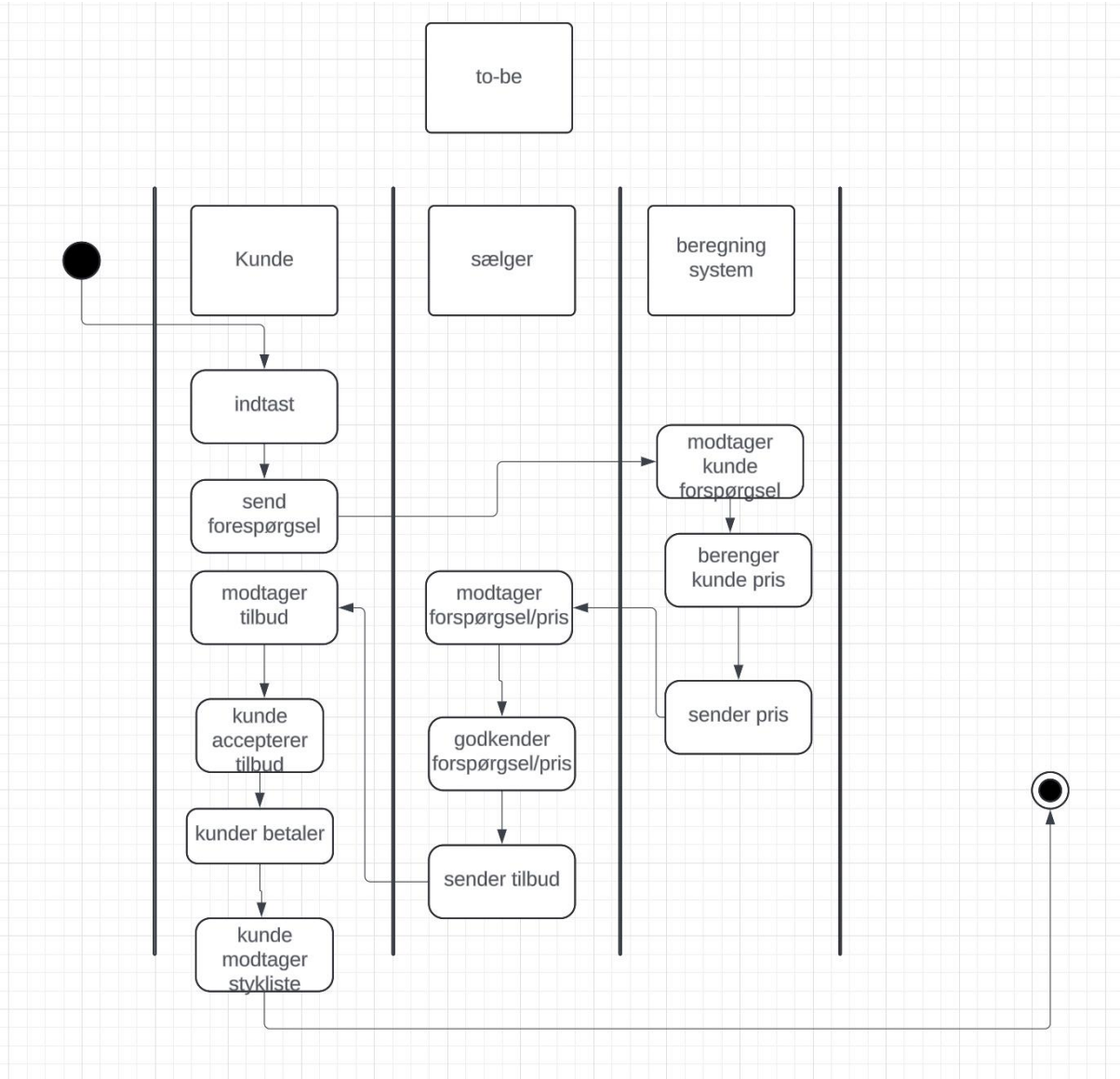
AS-IS

Nedenfor kan man se vejen kunden følger fra at bestille en skræddersyet carport. Kunden indtaster deres ønskede oplysninger og sender en forespørgsel til sælger. Sælgeren modtager kundens ønskede mål og beregner en pris og sender et tilbud kunden. Kunden kan godkende tilbuddet, betale og modtage en stykliste igennem e-mailen.



TO-BE

"TO-BE"-modellen nedenfor er næsten uændret i forhold til "AS-IS"-modellen. En bemærkelsesværdig ændring er, at vi har fjernet email-systemet (swim-lane). Som kan ses på diagrammet, er der nu også et mere klart start-til-slut-flow sammenlignet med 'as-is' modellen.



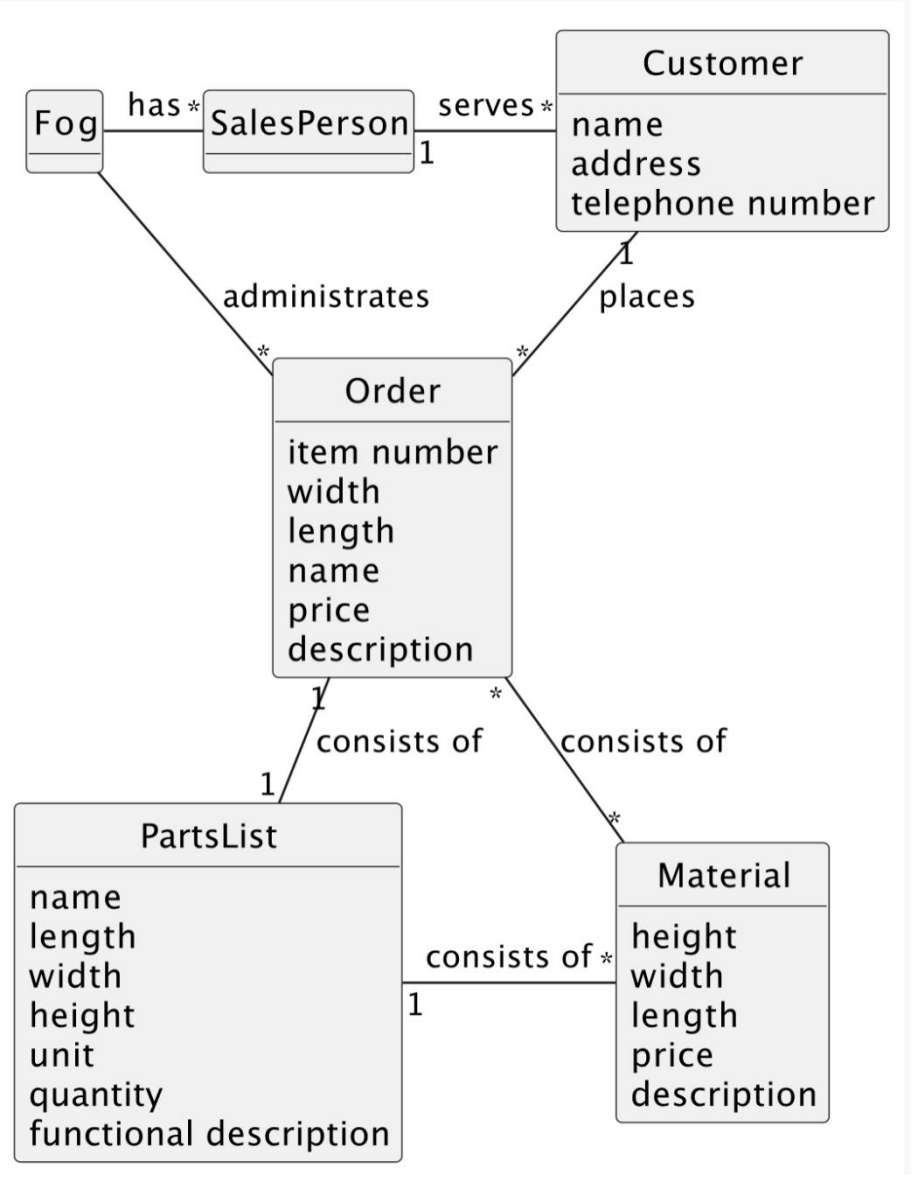
Domænemodel og EER

Domænemodel

I domænemodellen på billedet nedenfor kan man se de forskellige relationer mellem entiteterne. Vi har forsøgt at forme disse ud fra hvilke entiteter der eksisterer, når en kunde bestiller og køber en carport i den virkelige verden.

Modellen har 'Fog', som har en "has *" relation til 'SalesPerson', hvilket betyder, at Fog kan have mange sælgere, og en sælger kan have mange kunder. En kunde kan have mange ordrer, og en ordre kan have mange materialer, mens et materiale består af kun en 'PartsList'. 'PartsList' har en en-til-en relation til 'Order'.

Grunden til, at 'PartsList' og 'Material' har en en-til-mange relation er at en 'PartsList' kan bestå af mange materialer. Det er derimod ikke tilfældet, at et materiale kan bestå af mange PartsLists. Når der bliver lavet en stykliste, vil denne kunne bestå af mange materialer.



EER-diagram

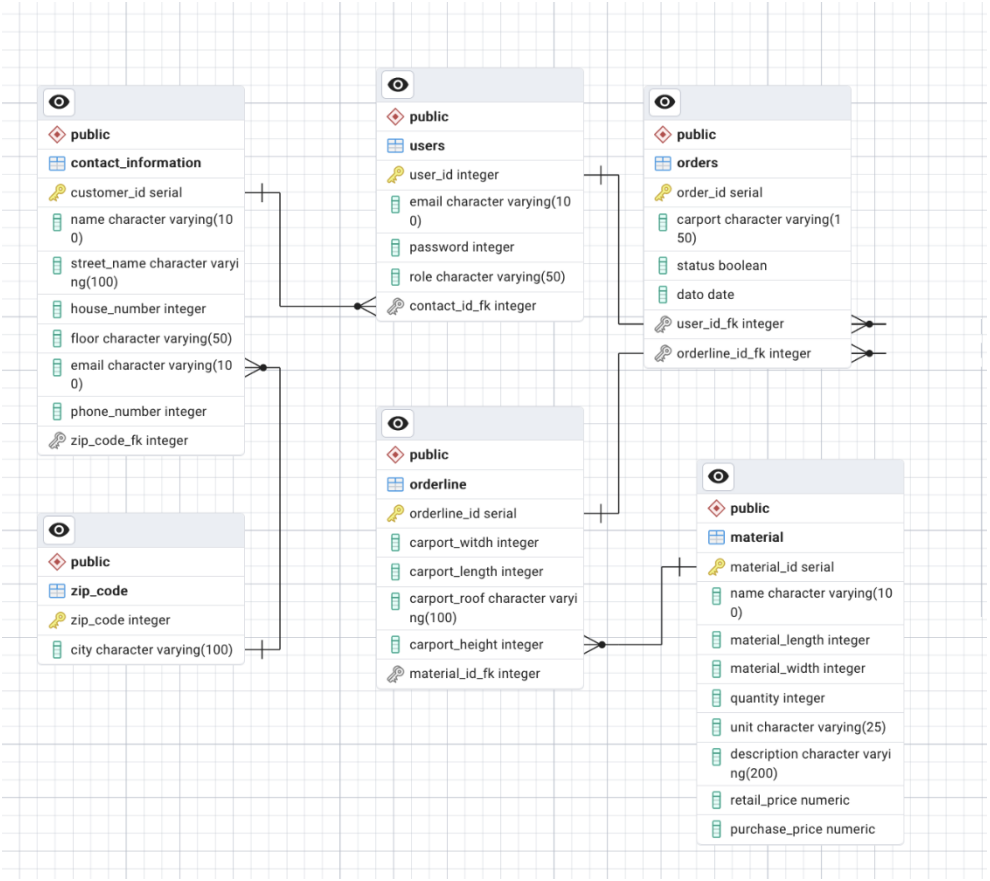
Normalformer

Vi har valgt at opfylde normalformer op til tredje normalform gennem hele databasestrukturen.

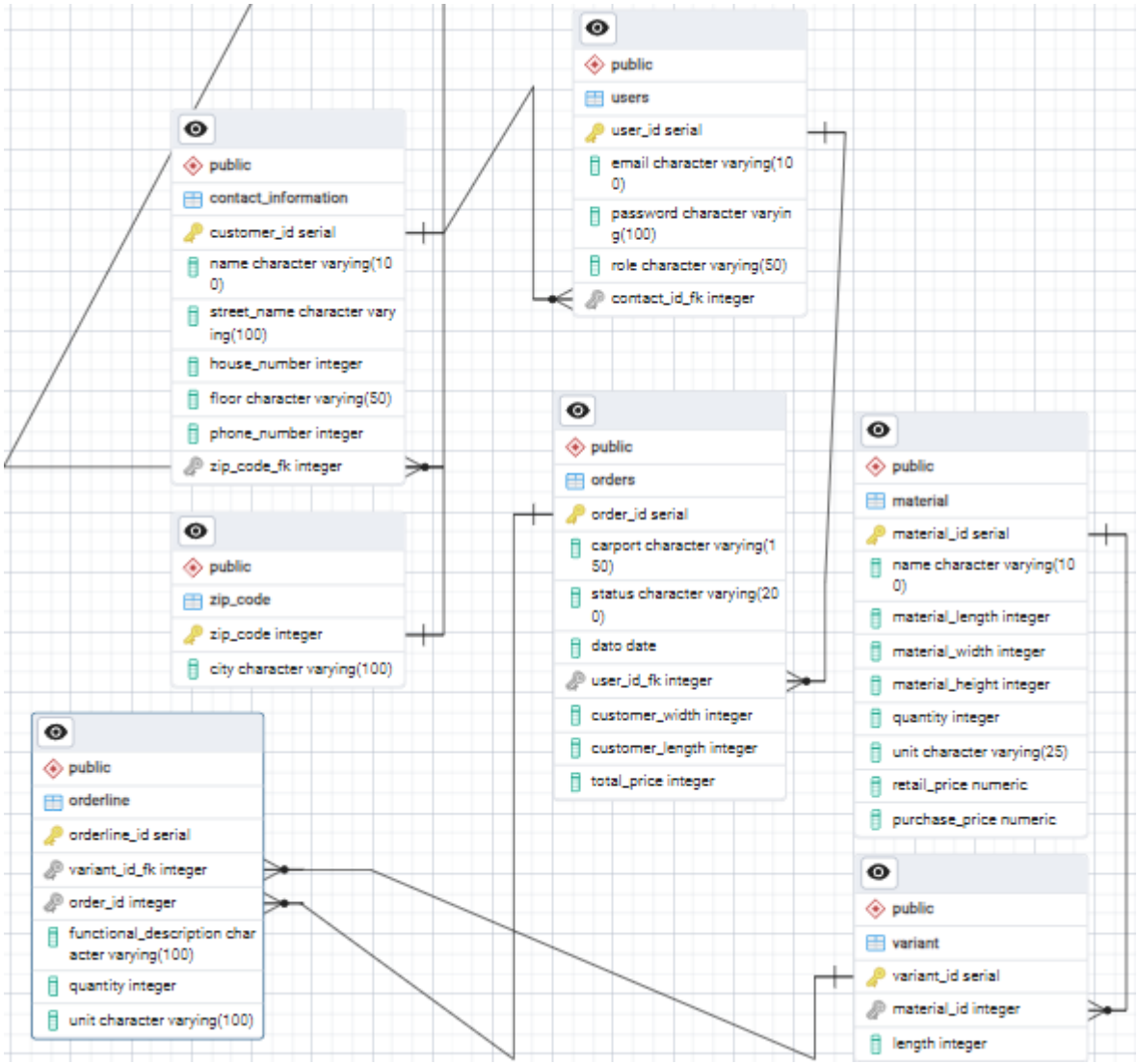
Variant

Som det kan ses i det tidligere EER nedenfor, har vi udeladt en 'variant'-tabel. Vi havde ikke overvejet, at mens dybden og bredden på et bræt er konstant, kan længden på brættet variere, afhængig af kundens valgte dimensioner. Derfor, jævnfør det nuværende EER, har vi tilføjet et 'variant'-tabel.

Tidligere EER



Nuværende EER



Relationer

zip-code til contact_information: en-til-mange

contact_information til users: en-til-mange

users til **orders**: en-til-mange

orders til **orderline**: en-til-mange

material til **variant**: en-til-mange

variant til **orderline**: en-til-mange

Zip-code-tabellen

Vi vurderer at den nuværende opsætning af de to tabeller 'zip-code' og 'contact_information', er struktureret og normaliseret. 'contact_information' opbevarer

data om kontaktens adresse sammen med en reference til 'zip-code' tabellen for bynavn og postnummer. Det skal nævnes, at vi har valgt ikke at gøre ID for 'zip-code' automatisk genereret, men derimod er et ID et postnummer. Det gør det nemmere at skabe overblik over hvilket postnummer en kunde har, når man søger gennem tabellen 'contact_information'.

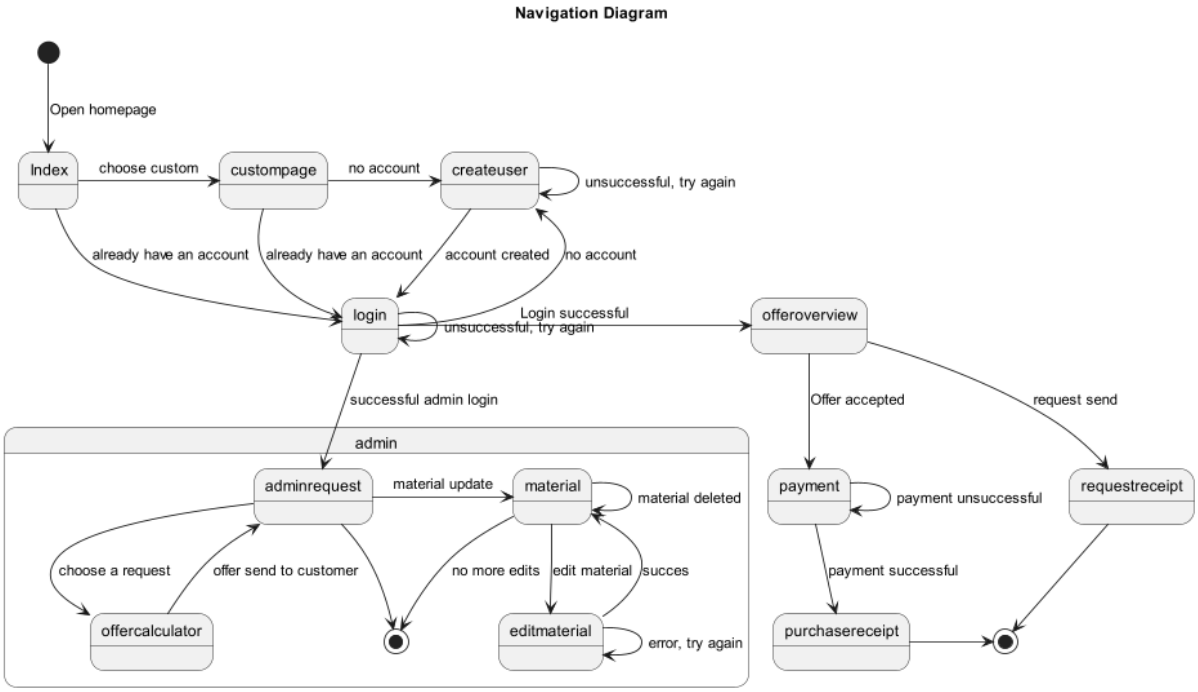
Fremmednøgle i 'contact_id_fk'

'contact-id-fk' i tabellen 'users' har vi valgt godt kan være 'null', for at gøre det muligt at oprette en bruger med rollen 'admin' i databasen.

Orders

I starten af projektet tog vi en beslutning om, at en ordre kun skulle bestå af en carport. Hvis Fog i fremtiden vil give mulighed for kunden også at kunne bestille et skur med sin carport, skal der tages højde for dette ved redesign af denne tabel. Man kunne for eksempel inkludere målene på skuret i tabellen.

Navigationsdiagram



Det som brugeren oplever er en række websider hvor man kan indtaste oplysninger og gå videre til andre sider. I større systemer kan det være svært at bevare overblikket over hvilke sider der er, og hvordan man navigerer mellem disse. Navigationsdiagrammet er beregnet til at vise dette på en overskuelig og let læsbar måde.

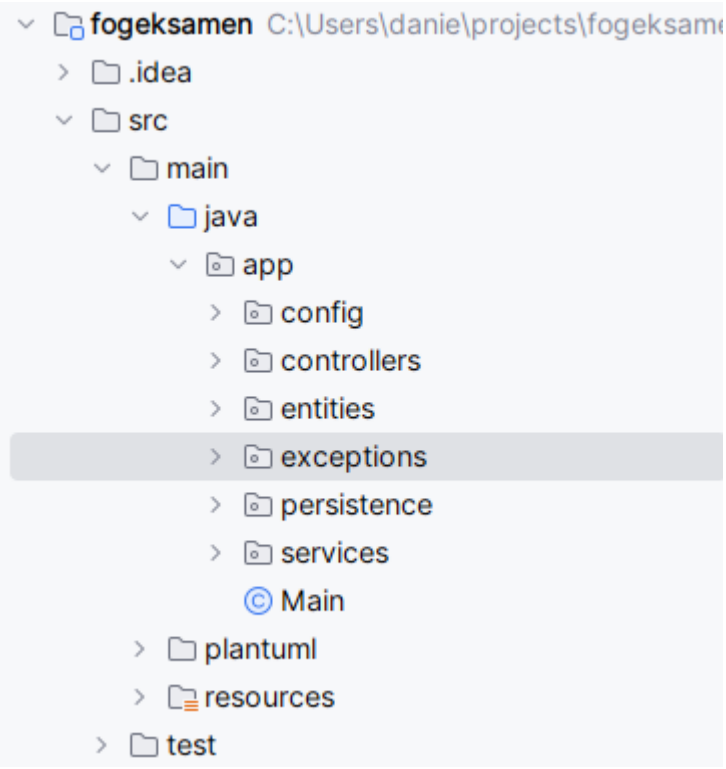
Hvad der ikke er illustreret i diagrammet, er de pile der går tilbage til index fra alle .html siderne. Hvis man er logget ind, er der en log ud knap samt Foss logo der et hyperlink, der begge fører brugeren tilbage til 'index', med hver deres formål.

'admin'-siden kan man kun komme til hvis man har log ind til dette. Man kan ikke oprette en 'admin'-bruger ligesom man kan som almindelig kunde. Admin giver adgang til at ændre i materialer og gennemgang af forespørgsler fra kunder.

Vi er i demo videoen gået igennem hele flowet.

Valg af arkitektur

I dette semester har vi arbejdet med en 'model view controller' inspireret arkitektur til at strukturere vores projekt. Denne arktitektur adskiller projektets entiter, brugergrænseflade, controllers og mappers hvilket gør koden mere vedligeholdesesvenlig og overskuelig.



Vi har arbejdet med følgende packages:

1. Config
    a. Indeholder konfigurationsklasser, der sætter applikationen op såsom, Thymeleafconfig og sessionconfig.
2. Controllers
    a. Indeholder controller-klasserne som håndterer HTTP-forespørgsler og svarer tilbage. Det kunne f.eks være brugerens login-forsøg som gribes og videresendes hvis succesfuldt.
3. Entities
    a. Indeholder entitetsklasserne, som repræsenterer dataobjekter i systemet. Disse klasser er typisk lig med database-tabeller.
4. Exceptions
    a. Indeholder brugerdefinerede exceptionsklasser, som hjælper med at håndtere database-fejl og undtagelser i applikationen.
5. Persistence
    a. Indeholder klasser, der håndterer dataadgang og kommunikation med databasen. F.eks 'UserMapper' der håndterer user tabellen i database.
6. Services
    a. Indeholder serviceklasser som ikke hører til i persistence. I dette tilfælde SVG og calculator klasser.
7. Recources
    a. public: Indeholder offentligt tilgængelige filer såsom billeder og css stylesheets.

    b. templates: Indeholder HTML-templates.
8. Test
    a. Indeholder testfiler og testklasser, der bruges til at sikre at programmet kører som forventet.

Særlige forhold

Brug af Debugging og PrintStackTrace

Som gruppe har vi adopteret en effektiv tilgang til fejlfinding i Java. Vi er begyndt at gøre stor brug af debugging-værktøjet, da det giver os mulighed for at forstå koden bedre og identificere fejl hurtigt og præcist.

Brug af Debugging

Vi benytter ofte debugging-funktionaliteten i Intellij, som gør det muligt for os at køre vores kode trin for trin. Det hjælper os med at forstå flowet af programmet og hvordan flowet opfører sig når koden bliver uoverskuelig.Ved at sætte breakpoints på strategiske steder i koden kan vi observere programmets tilstand og flow, hvilket oftest er stor hjælp, når vi forsøger at finde og rette fejl.

Brug af PrintStackTrace

Udover debugging bruger vi også printStackTrace() til fejlhåndtering. Når en exception opstår, giver printStackTrace() os en detaljeret fejlmeddelelse der viser hvor fejlen opstod. Dette gør det lettere at spore årsagen til fejlen og bestemme, hvor i koden problemet er opstået.

Ved at inkludere printStackTrace() i vores catch-statements sikrer vi, at vi får feedback, når noget går galt. Det hjælper os med hurtigt at identificere problemer uden at skulle gennemgå koden manuelt flere gange. Specielt i dette projekt hvor routes og metoder går på tværs af hinanden.

Session Attribute

I kodeeksemplet nedenfor kan man se, at ‘session.currentLength == 0’ bliver brugt. Det er implementeret for at bevare brugerinput og overføre det til andre sider, så værdier gemmes.

```
<div th:if="${session.currentUser == null}">
    <form name="form1" id="form1" action="/custompage" method="post">
        <select name="length" id="length" required>
            <option value="" disabled th:selected="${session.currentLength == 0}">Længde</option>
            <option value="240" th:selected="${session.currentLength == 240}">240 cm</option>
```

I første omgang i 'loadpage' bliver både 'currentLength' og 'currentWidth' sat til at være 0 (det sker når siden bliver loadet første gang, så vil 'session.currentLength == 0' være det der bliver valgt på input feltet som default). Herefter renderes 'custompage.html', hvor kunden kan vælge de ønskede mål til en carport.

```
2 usages   Mads Benjamin Ribberholt
public static void loadpage(Context ctx){
    ctx.sessionAttribute("currentLength",0);
    ctx.sessionAttribute("currentWidth",0);
    ctx.render( filePath: "custompage.html");
}
```

Når kunden går videre til enten 'login' eller opretter en bruger, vil disse værdier blive gemt i 'currentLength' og 'currentWidth'. På denne måde bliver de gemt i 'sessionAttribute'. Dette gør at kunden får mulighed for at ændre de mål, man valgte i starten. Det hensigtsmæssige ved at gemme dem som 'sessionAttribute' er at vi altid har mulighed for at tage fat på det specifikke 'sessionAttribute', i stedet for at lave en liste i koden og gemme værdierne. Til sidst i if-statementet på linje 140 og 141 bliver 'currentWidth' og 'currentLength' opdateret med de nye værdier og gemt i 'sessionAttribute'.

```
131 @    public static void login(Context ctx, ConnectionPool connectionPool) {
132          String email = ctx.formParam( key: "email");
133          String password = ctx.formParam( key: "password");
134          String widthParam = ctx.formParam( key: "width");
135          String lengthParam = ctx.formParam( key: "length");
136          if (widthParam != null && !widthParam.isEmpty() && lengthParam != null && !lengthParam.isEmpty()) {
137              try {
138                  int w = Integer.parseInt(ctx.formParam( key: "width"));
139                  int l = Integer.parseInt(ctx.formParam( key: "length"));
140                  ctx.sessionAttribute("currentLength", l);
141                  ctx.sessionAttribute("currentWidth", w);
142
143              } catch (NumberFormatException e) {
144                  throw new RuntimeException(e);
145              }
```

SVG

For at vise en tegning af kundens carport oppefra har vi valgt at lave en dynamisk SVG, der automatisk justeres i forhold til kundes input af bredde og længde. Derfor er der i vores backend lavet en package, der kaldes 'services'. Klassen 'Svg' bygger ved hjælp af en 'Stringbuilder' primitiver, som <line> eller <rectangle>, kaldt for at bygge SVG.

```
public CarportSvg(int customerWidth, int customerLength) {
    this.customerWidth = customerWidth;
    this.customerLength = customerLength;
    carportSvg = new Svg( x: 0, y: 0, width: "200%", viewBox: "0 0 850 690");
    carportSvg.addRectangle2( x: 40, y: 40, customerLength, height: 600, style: "stroke:#000000; fill:#ffffff");
    addRafters(customerLength);
    addBeams(customerLength);
    addPoles(customerLength);
    addMountingTape();
}
```

Derudover har vi valgt at lave 2 klasser, der hver især bygger en SVG, ved at anvende 'Svg'-klassens konstruktør og metoder som skabelon.

Ovenstående er et billede fra klassen 'CarportSvg', der i konstruktøren instantiere et objekt af klassen 'Svg'. I Konstruktøren anvendes 'Stringbuilderen' for første gang. I metoderne 'appendes' de forskellige primitiver til tekstrengen 'Svg2StringBuilder'.

```
41      private void addRafters(int customerLength){
42
43
44          for (double i = 95 ; i < customerLength ; i += 55.714){
45              rafters.add(i);
46              carportSvg.addRectangle2(i, y: 40, width: 4.5, height: 600, style: "stroke:#000000; fill:#ffffff");
47          }
48
49
50      }
```

For-løkken ovenfor initialiserer en variabel 'i' til 95. Den vil køre så længe 'i' er mindre end 'customerLength'. Ved slutningen af hver iteration øges 'i' med 55.714.

På linje 46 tilføjes en rektangel til 'carportSvg' med de specifikke dimensioner.

Positionerne i 'rafters'-listen bruges som referencer for placeringen af andre komponenter, som stolper (poles) og hulbånd (mounting tape).

```java
public static void showCarport(Context ctx, ConnectionPool connectionPool){

    Locale.setDefault(new Locale( language: "US"));
    User user = ctx.sessionAttribute( key: "currentUser");

    try {
        Orders order = OrderMapper.getCustomerDim(user.getUserId(), connectionPool);

        int customerWidth = order.getCustomerWidth();
        int customerLength = order.getCustomerLength();

        CarportSvg carportSvg = new CarportSvg(customerWidth,customerLength);

        ArrowSvg arrowSvg = new ArrowSvg(customerWidth,customerLength);
        String combinedSvg = arrowSvg.getArrowSvg2().addSvg2(carportSvg.getCarportSvg2());

        List<Orders> orderList =OrderMapper.getAllOrders(connectionPool);
        ctx.sessionAttribute("ordersList", orderList);
        ctx.attribute("svg",combinedSvg);
        ctx.render( filePath: "payment.html");

    } catch (DatabaseException e) {
        //Fejlmeddelelsen fra DatabaseException tilføjes som en attribut til kontekstobjektet.
        ctx.attribute("message", e.getMessage());
        ctx.render( filePath: "login.html");
    }

}
```

Metoden 'showCarport' ovenfor, henter brugerens ordredetaljer fra databasen og sammensætter SVG-tegningerne for carporten, det vil sige SVG'en fra klassen 'ArrowSvg' samt SVG'en fra klassen 'CarportSvg'. Vi kombinerer tegningerne og gemmer dem i konteksten. Derefter renderes betalingssiden med de to kombinerede SVG'er.

Vi er klar over at denne metode ikke burde rendere en .html-side, men blot skal have til formål at lave strengen 'combinedSVG' og gemme dem i konteksten. På den måde er der mulighed for at 'displaye' den kombinerede tegning hvor som helst på siden.

Udvalgte kodeeksempler

Login

I vores loginmetode har vi valgt at have 4 attributter, som består af 'email', 'password',' widthparam' og 'lengthparam'. 'email' og 'password' bliver brugt i resten af metoden som ses på billedet nedenfor.

```
public static void login(Context ctx, ConnectionPool connectionPool) {
    String email = ctx.formParam( key: "email");
    String password = ctx.formParam( key: "password");
    String widthParam = ctx.formParam( key: "width");
    String lengthParam = ctx.formParam( key: "length");
    if (widthParam != null && !widthParam.isEmpty() && lengthParam != null && !lengthParam.isEmpty()) {
        try {
            int w = Integer.parseInt(ctx.formParam( key: "width"));
            int l = Integer.parseInt(ctx.formParam( key: "length"));
            ctx.sessionAttribute("currentLength", l);
            ctx.sessionAttribute("currentWidth", w);

        } catch (NumberFormatException e) {
            throw new RuntimeException(e);
        }
    }
```

'widthparam' og 'lengthparam' bliver brugt til at gemme brugerens valg af længde og bredde på carporten. Vores 'if-statement', som man kan se på billedet, tjekker om 'widthparam' ikke er 'null' og ikke er tom (en tom streng). Hvis 'widthparam' er 'null' eller er tom, vil 'if-statement' blive ignoreret, og koden fortsætter. Grunden til 'if-statement' er inkluderet er hvis en bruger skal logge ind direkte fra 'index.html' (forsiden), eller en 'admin' skal logge ind. Så behøver de ikke at have udfyldt værdierne for 'widthparam' og 'lengthparam', hvilket betyder, at de gerne må være 'null'.

Hvis en kunde derimod vælger dimensioner og herefter logger ind, bliver dimensionerne gemt som nedenstående:

String widthParam = ctx.formParam("width");

String lengthParam = ctx.formParam("length");

```
    try {
        User user = UserMapper.login(email, password, connectionPool);
        //Hvis login-operationen lykkes, gemmes brugeren som den aktuelle bruger i sessionsattributterne i kontekstobjektet.
        ctx.sessionAttribute("currentUser", user);


        if (user.getRole().equalsIgnoreCase( anotherString: "customer")) {
            boolean status= OrderMapper.doesOrderByUserIdExist(user.getUserId(),connectionPool);
            if (status==true){
                List<Orders> orders=OrderMapper.getOrderFromUserId(user.getUserId(),connectionPool);
                ctx.sessionAttribute("orders",orders);
                String orderStatus = orders.get(0).getStatus();
                ctx.sessionAttribute("status", orderStatus);
                ctx.render( filePath: "offeroverview.html");
            }else {
                ctx.render( filePath: "custompage.html");
            }
        } else {
            OrderController.displayAllOrders(ctx, connectionPool);
        }
    } catch (DatabaseException e) {
        //Fejlmeddelelsen fra DatabaseException tilføjes som en attribut til kontekstobjektet.
        ctx.attribute("message", e.getMessage());
        ctx.render( filePath: "login.html");
    }

}
```

Vi starter ovenfor med at lave en ‘try-catch’ for nemmere at kunne fange og håndtere eventuelle fejl. I første linje opretter vi et ‘user’-objekt ved at kalde vores ‘login’-metode i ‘mapperen’. Herefter gemmer vi ‘user-objektet’ i en ‘sessionAttribute’, for at identificere den nuværende bruger.

Vi har et ‘if-statement’, der kontrollerer om brugerens rolle er "customer". Hvis dette er tilfældet, går vi videre ind i ‘statementet’. Først opretter vi en ‘boolean variabel’ kaldet ‘status’, der tjekker om brugeren har en eksisterende ordre i databasen. Herefter følger et nyt 'if-statement', som tjekker, om ordren findes. Hvis ‘status’ er lig “true”, fortsætter vi ved at oprette en liste med datatypen ‘Orders’, som vi også kalder ‘orders’. I denne liste gemmer vi ordren for brugeren baseret på brugerens ID.

Listen gemmes i en ‘sessionAttribute’, så vi kan vise den på hjemmesiden. Vi henter også ‘status’ for den nuværende ordre og gemmer denne ‘status’ i en ‘sessionAttribute’. Dette gør det muligt for os at tjekke ordrestatus i html-koden senere i projektet. Til sidst renderer vi brugerens overblik over ordrer.

Hvis brugeren ikke har en ordre i systemet, bliver de omdirigeret til siden, hvor de kan indtaste ønskede mål for en carport. I det sidste ‘else-statement’ kalder vi ‘displayAllOrders’ metoden som kun er for ‘admin’. Vi viser så sælgeren alle de ordrer der ligger i systemet.

PriceListCalc

```
public static int calcBeams(int customlength, ConnectionPool connectionPool) {
    int bestMatch = 0;

    List<Variant> variant = VariantMapper.selectVariantLengthById( materialId: 5, connectionPool);
    int variantCount = variant.size();
    if (customlength > 600) {
        int length = customlength / 2;
        for (int i = 0; i < variantCount; i++) {
            int variantLength = variant.get(i).getLength();
            bestMatch = variantLength;
            if (bestMatch >= length) {
                break;
            }
        }
    } else {
        for (int i = 0; i < variantCount; i++) {
            int variantLength = variant.get(i).getLength();
            bestMatch = variantLength;
            if (bestMatch >= customlength) {
                break;
            }

        }

        return bestMatch;
    }
    return bestMatch;
```

Denne metode bruges til at beregne den bedst egnede rem-længde baseret på brugerens ønskede længde. Vi starter med at initialisere en 'int'-variabel, som vi kalder 'bestMatch'. Derefter opretter vi en liste af datatypen 'Variant', hvor vi henter variant-længderne med 'materialId' nummer 5, da vi ved at dette materiale skal bruges til remme. Vi erklærer endnu en 'int'-variabel, som holder størrelsen på vores 'variant-liste'.

Vi bruger et 'if-statement' til at kontrollere, om brugerens ønskede længde er over 600cm. Hvis dette er tilfældet, findes der ikke spær der er lange nok, og vi går derfor ind i dette 'if-statement'. Vi halverer brugerens længde, da vi senere skal bruge denne værdi. Derefter laver vi et 'for-loop', som itererer igennem 'variant-listen'. Inde i løkken henter vi længden af den aktuelle variant, og gemmer den som en 'integer-variabel' 'variant Length'. Vi sætter 'bestMatch' til at være lig 'variantLength'. En ny 'if-statement' tjekker om 'bestMatch' er større end eller lig med brugerens halverede længde. Hvis ja, 'breaker' vi løkken og returnerer 'bestMatch'.

Hvis brugerens ønskede længde er mindre end 600 cm, bruger vi igen 'for-loop' til at iterere igennem variant-listen. Loopet gør det samme som før, og 'bestMatch' returner så den værdi som er tættest på, men større end brugerens ønskede længde

På denne måde sikrer vi, at hvis længden er over 600 cm, får brugeren to remme, som kan hvile på en midterste stolpe i stedet for at skulle lappe to remme sammen uden støtte nedefra.

partsListPriceCalcById

```
private static float partsListPriceCalcById(Context ctx, ConnectionPool connectionPool) throws DatabaseException {
    int orderId = Integer.parseInt(ctx.formParam( key: "orderId"));
    List<Orderline> userOrder = OrderlineMapper.getPartsListByOrderId(orderId, connectionPool);
    float totalPrice = 0;
    for (int i = 0; i <= userOrder.size()-1; i++) {

        Orderline orderline = userOrder.get(i);
        Material material = orderline.getMaterial();

        float retailPriceTotal = material.getRetailPrice();
        int quantity = orderline.getQuantity();
        int customerLengthInMeters= orderline.getVariant().getLength()/100;

        totalPrice += quantity * (retailPriceTotal * customerLengthInMeters);

    }
    return totalPrice;
}
```

I vores 'partsListPriceCalcById'-metode beregner vi helt simpelt en totalpris for kundens carport. Vores valg af datatype her er en 'float', hvilket betyder, at vi kan få en komma-værdi tilbage som totalpris.

Vi starter med at have en 'int' 'orderId', som er den nuværende 'orderId' på kunden som er logget ind. Derefter laver vi en liste ('OrderLine'-datatype), hvor vi gemmer alt fra 'getPartsListByOrderId' ud fra 'orderId'. Der oprettes en 'float'-variabel, som er lig med 0, og den vil blive opdateret igennem 'for-loopet'. Her løber vi gennem 'userOrder.size() - 1'. I koden kan vi se, at vi har 'OrderLine orderline = userOrder.get(i)'. Denne linje henter alt ud fra den ene 'OrderLine', så vi derefter kan hente materialet. I retailPriceTotal, af datatypen 'float', gemmes den nuværende værdi af materialet i attribut "retailprice" . I 'quantity', af datatypen 'int', gemmes antallet af det specifikke materiale, som spær eller remme, her kunne 'quantity' f.eks. være 2. I 'customerLengthInMeters' henter vi længden og dividerer med 100, for at gå fra cm til meter. På den måde er det nemmere at udregne totalprisen, der til sidst opdateres i 'totalPrice'.

'totalPrice' tager de forskellige værdier, og sætter dem sammen til en totalpris, som kan returneres.

Offeroverview frontend if-else statement

```
<div>
    <form class="table" id="table-styling" method="post">
        <div th:if="${session.status == 'Tilbud sendt'}">
            <table>
                <thead>
                <tr>
                    <th>Bredde</th>
                    <th>Længde</th>
                    <th>Tag</th>
                    <th>Total Pris</th>
                </tr>
                </thead>
                <tbody>
                <tr th:each="order : ${session.orders}">
                    <td th:text="${order.customerWidth}">Width</td>
                    <td th:text="${order.customerLength}">Length</td>
                    <td>Trapez</td>
                    <td th:text="${order.price}">Price</td>
                    <td>
                        <button class="button-blue" type="submit" th:value="${order.orderId}"
                                th:formaction="@{/payment}">Gå til tilbud
                        </button>
                    </td>
                </tr>
                </tbody>
            </table>
        </div>
    </form>
</div>
```

Dette kodeeksempel viser hvordan en 'if-else-statement' fungerer i 'html'.

Billedet ovenfor er et 'if-statement'. Hvis der ligger en 'status' med 'Tilbud sendt' i som 'sessionAttribute', vises en tabel hos brugeren med længden, bredden, tagtype og totale pris på dennes ordre. Dette er ikke en stykliste, men blot en oversigt. Statussen på ordren bliver hentet fra databasen i 'mapperen' og bliver sat som 'sessionAttribute' i controlleren. I forbindelse med tabellen vises en knap, der fører 'useren' videre til tilbudssiden, for den specifikke ordre, ved hjælp af det gemte 'orderId'.

```
<div th:unless="${session.status == 'Tilbud sendt'}">
    <h1>Ordre venter på bekræftelse fra sælger</h1>
    <h2>Tjek tilbage senere</h2>

    <h3>Brug for hjælp?</h3>
    <h5>kontakt os på:</h5>
    <p>E-mail: webshop@johannesfog.dk</p>
    <p>tlf: 90909090</p>

</div>
```

Billedet ovenfor viser 'unless'-statement. Hvis der ikke ligger et 'Tilbud sendt' i 'sessionAttribute', vises teksten i de forskellige <h> og <p> tags. Vi har valgt at gøre det på denne måde, for at have så få .html-sider som muligt.

Status på implementering

Ufuldstændige websider

Da vi udformede navigations-diagrammet, var intentionen at vise brugeren de "premade" carporte på Fogs hjemmeside. efter nogle dage fandt vi ud af det ville være mere relevant at bruge vores tid på den skræddersyet del, for at komme så langt som muligt med vores user stories.

Styling og templates

Fra starten af havde vi aftalt, at vi i vores stylesheet ville lave en form for "templates" til vores header, footer, knapper osv. Dette gjorde vi for at sikre konsistens og effektivisere stylingprocessen. Efter at have brugt for lang tid på styling i det sidste projekt, besluttede vi denne gang at holde det mere minimalistisk. Som følge heraf blev styling ikke et problem i dette projekt, da vi hurtigt kunne genbruge de definerede templates og fokusere på andre vigtige aspekter af udviklingen.

Fejl i sidste øjeblik

På baggrund af vores erfaring vidste vi, at det er umuligt at nå en fuldstændig fejlfri afslutning på et projekt af denne størrelse. Vi var derfor bevidste om, at vi til sidst ville stå med små fejl og problemer, som ikke kunne prioriteres inden for den givne tidsramme. Vi sørgede for at fokusere på de mest kritiske funktioner og fejlretninger, så projektet kunne leveres til tiden med en høj grad af kvalitet og funktionalitet.

Test og mangler

Refaktorering og Kode-forbedringer

Vi mangler at refaktorere mange klasse- og metode navne og html-sider. Vi har valgt at prioriterer at få vores kode/program til at virke i forhold til vores user-stories. Eksempelvis vil vi gerne have at admin kan se en SVG tegning når han går en på af de order der ligger klar på "offercalculator.html" vi fandt også ud af at vi skulle refaktorere metoden 'displayCarport()' i klassen 'Ordercontroller' for at kunne vise SVG'en korrekt.

Justering af carportens SVG

På den nuværende SVG af carporten, er spærrende placeret på den første kolonne af stolper, og den sidste kolonne af stolper. Til sidst i kodeforløbet gik det op for os at vi også vil have spær i både starten og slutningen af remmen, for at sikre en holdbar konstruktion. Derudover mangler hulbånd i styklisten har vi ikke med, men det er med i SVG

Admin-funktionalitet

Vi satte os i starten for at rollen 'admin' skal kunne tilføje materialer til databasen. Dette har vi ikke nået at få implementeret. Dette gør også at User story 5 ikke er godkendt, da admin ikke kan tilføje materialer.

Kvalitetssikring (test)

Automatiserede test

Klasse	Metode testet	Testtype	Beskrivelse
UserMapper	login	Integrationstest	Tester validering af bruger. Hvis bruger findes i systemet er testen en succes
OrderMapper	doesOrderByUderIdExist	Integrationtest	Tjekker om ordre existere ud fra brugerens id

PartsListCalc	calcPosts	unit test	Regner mængden af nødvendige stolper ud
PartsListCalc	calcBeams	unit test	Regner den bedste remlængde ud
PartsListCalc	calcBeamsQuantity	unit test	Regner den nødvendige remmængde ud, ud fra carportens længde

Testarbejde inden integration i 'Master-branch'

1. Udvikling af test
    a. Vi fokuserede på at teste metoder, der omhandlede beregninger af carporte, for at sikre præcision og korrekthed i de kritiske funktioner.
    b. Tests skrives parallelt med udviklingen af de nye funktioner for at sikre, at de fleste dele af koden testes, inden den integreres i 'master branch'.
2. Brug af test framework
    a. Vi bruger JUnit til at skrive og køre unit- og integrationstest.
3. Integrationstest
    a. Integrationstests blev brugt til at teste metoder, der arbejder sammen med databasen.
    b. For at sikre dette har vi oprettet et test schema i databasen med de nødvendige tabeller, så vi kan teste databasemetoderne isoleret fra vores reele data.
4. Unit test
    a. Unit tests blev brugt til at teste metoder, der havde til formål at regne styklisten.
    b. Metoderne blev testet for at sikre at sælgeren får den rette anbefalede pris at vide og at kunden får den rette stykliste.

Ved at udvikle tests for kritiske dele af systemet, sikrer vi at koden er pålidelig og så vidt muligt fejlfri, inden den merges ind i 'master branch'. Tilgangen hjælper os med at opretholde høj kvalitet i koden.

User Acceptance test

US-01	Accepteret
US-02	Accepteret
US-03	Accepteret
US-04	Accepteret
US-05	Ikke godkendt - admin kan ikke tilføje materialer
US-06	Accepteret
US-07	Accepteret
US-08	Accepteret

Proces

Arbejdsprocessen faktuelt

Fase 0

I undervisningen, forud for projektstart, lavede vi noget af det analyserende arbejde som øvelse til projektet. Dette bestod af en risikoanalyse, interessentanalyse, aktivitetsdiagram og relevante user-stories samt acceptance-criteria. Dette fik vi feedback på, som efterfølgende blev implementeret i fase 1.

Fase 1

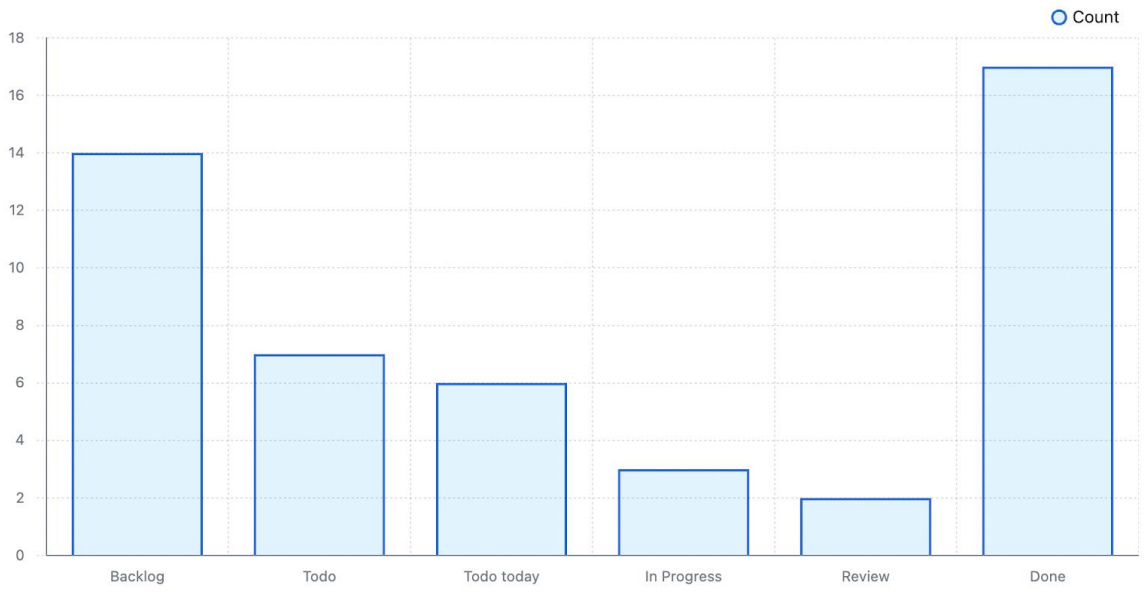
Efter endt kickoff, startede vi med at aftale processen gennem forløbet, lave gruppekontrakt med kode-kodeks, hvordan vi versionstyrer og logbog.
Vi mødtes dagligt på skolen, hvor vi startede med at kigge på 'KanBan', og lægge en plan for dagen. Vi havde ikke nogen decideret 'KanBan'-mester. Alle beslutninger blev taget demokratisk i plenum.
For at sikre 'master branch' mest muligt, blev repository på Github sat op således, at et 'pull- request' skal have et godkendt review fra en anden person i teamet før det kan blive 'merged' ind i 'master branch'. Derudover skal man først opdatere 'master branch', hvorefter man 'merger master branch' ind i den 'working branch' man sidder i, før man kan 'commit' og 'pushe'. Således undgår man at få mange 'merge'-konflikter undervejs.

Der blev ikke kodet Java i fase 1. Processen bestod af gennemgang og segmentering af alt det forud-liggende analysearbejde fra fase 0, samt navigationsdiagram, domænemodel, 'deployment' på 'droplet', opsætning af database og ER-diagram.

Fase 2

Efter endt kundemøde begyndte vi at kode. Vi startede med at lave alle de html-sider der var planlagt i henhold til 'navigations-diagrammet'. Vi besluttede os for ikke at lave 'styling', hverken css eller flexbox i denne fase, da vi fokuserede på at have et funktionelt ordreflow klar til kundemødet, der allerede var lagt onsdag.

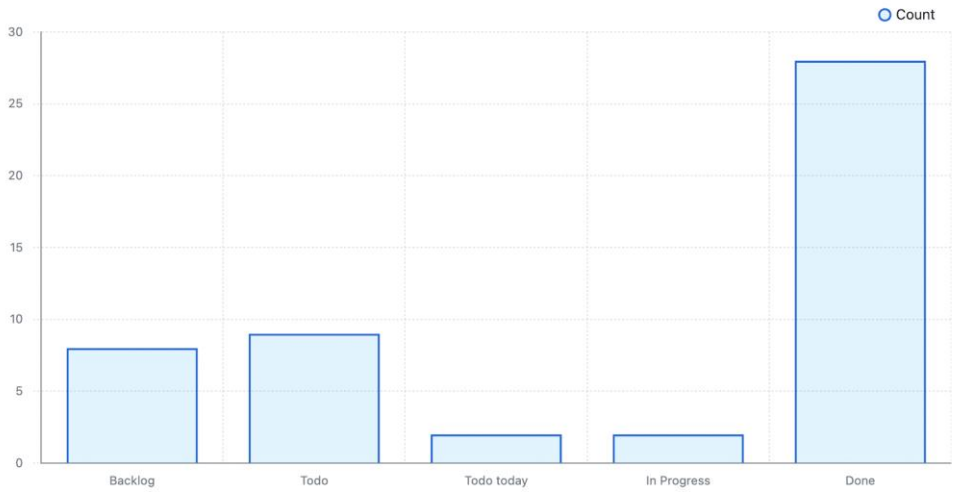
Fase 3



Planen i fase 3 var at blive færdig med alt kodning, både backend og frontend inden code freeze, så vi kunne fokusere på at skrive rapport efter kundemødet.

Vi forsøgte, så vidt muligt, at dele user stories ud så hvert medlem af teamet havde sit eget område af koden at fokusere på. Nogle user stories, der var store, blev lavet i par via 'code-with-me' i IntelliJ.

Fase 4



Sidste fase er rapportskrivning. Her blev overordnede hovedpunkter uddelegeret. De større af dem som, arbejdsprocessen reflekteret, status på implementering og udvalgte kode eksempler, er diskuteret og skrevet fælles.

Team-kontrakt og kode-kodeks

Team kontrakt

- Når man er færdig med sin nuværende opgave, assisterer man teamet ved at hjælpe en af de andre, eller igangsætte andre opgaver, hvis kanban tillader det.
- Alle bestræber sig på at støtte hinanden og møde andre teammedlemmer, hvor de er fagligt.
- Vi forpligter os til at give feedback i form af burgermodellen (positivt, kritikpunkt, positivt), således at vi holder en produktiv og ordentlig tone.
- Hvis en konflikt opstår, skal alle parter have mulighed for at dele deres perspektiv på situationen. Modparten bestræber sig på at sætte sig ind i den anden persons perspektiv. Der tages højde for at alle er forskellige og har gode intentioner.
- Hvis et teammedlem A er vidne til en form for overtrædelse af ovenstående fra teammedlem B, skal A venligt påminde B om dette.

Kode-kodeks

- 'Camelcase' i backend, 'snake case' i database
- Altid ental i variabel- , klasse-, og metode-navne, undtagen i gennemløb og andre specielle forhold
- Klasse- variabel- og metode-navne skal være læsbare for enhver
- Hvis du har 'commited' og 'pushet' en branch, aldrig editér i den pågældende branch ige

- Der bruges 'CSS Flexible box layout', også kaldet 'Flexbox', til at skalere til både pc og smartphone
- Der anvendes 'best practices' generelt

Logbog

Fase 0

Vi var i gruppen overbevist om at de øvelser vi lavede i ugen op til kickoff, kun var øvelser og ikke noget vi konkret skulle bruge til eksamensprojektet. Derfor forekommer der ikke noget af logbogen fra denne periode.

Fase 1

- Gennemgang af forud-lavede diagrammer: aktivitetsdiagram (AS-IS og TO-BE)
- Påbegyndt domænemodel samt ER-diagram
- Oprettet 'repository' på 'Github' med alle i studiegruppen som "collaborator"
- Oprettet KanBan med alle som "collaborator"
- Korrekt opsat projekt i Intellij med 'packages' og filepathing
- Udfyldt samt godkendt kode-kodeks og gruppekontrakt, af alle i studiegruppen
- Opsætning / Skabelon af rapport
- Bliv enig om navigationsdiagram, lavet i hånden herefter i PlantUML i IntelliJ
- Template til projektet opsat, demo i figma
- Kodet navigationsdiagram i Intellij
- Startet på HTML side opsætning (minus css og flexbox)
- Haft hjemme kode dag, med løbende kontakt over discord
- 13 html sider færdiggjort (minus flexbox)
- Deployment af database på droplet
- Kundemøde: god feedback på analyser og diagrammer

Fase 2

- Uddelegeret ansvarsområder at kode
- Påbegyndt backend-kodning til forespørgsel order flow
- Noget kode kunne vi genbruge fra tidligere cupcake projekt, create user og login
- Hurtigt færdig med order flow
- God feedback til kundemøde. En kommentar vi fik: "i har grundstenene til et alsidigt program"
- Vi holder fri torsdag og fredag, da det er helligdag

Fase 3

- Tung kodeuge lagt for, da der er codefreeze søndag d. 19/5/2024
- User stories uddelegeret, alle har egne ansvarsområder for koden
- Vi har i teamet besluttet at opretholde code freeze. Det betyder, at vi har enkelte mangler, der ikke blev nået at kode eller fået til at virke efter vores ønske
- Der blev ved en fejl pushed og merged et personligt password med til github da unit tests blev lavet. For ikke at blive hacket eller leake det private password, har vi derfor besluttet at sætte repository til private og oprette et nyt public repository til når vi skal aflevere
- En mangel vi stadig har, er flexbox på hele projektet til skalering til smartphone

Fase 4

- Sidder hjemme mandag og taler over discord, da skolen er lukket på grund af helligdag
- Overordnede overskrifter uddelegeret til teammedlemmer
- Fejl og mangler fra fase 3 er skrevet ind og beskrevet under særlige forhold i rapporten

Vejledningsmøde

Dagsorden

- Bestillingsflow gennemgang
- Inspiration fra Molslinjen
- Bestillingsprocessen
- Start af SVG
- Bestillingsflow
- Beregning af stykliste

Bestillingsflow

- God inspiration at hente hos Molslinjens hjemmeside.

Trin i bestillingsprocessen

- første trin: indtast bredde og længde
- andet trin: indtast email og kodeord
- tredje trin: forespørgsel sendes

Start af SVG-tegning

Vi begynder med at udvikle SVG-grafik, og starter samtidig arbejdet på bestillingsflowet. Arbejdet deles, så to personer arbejder på admin-siden og to på kundesiden.

Calculator

Beregning af styklisten skal påbegynde næste uge. Vi starter med fire personer på dette arbejde. Efter indledende arbejde kan to personer fortsætte med SVG-arbejdet, mens de andre to fortsætter med beregneren.

Materialer og priser

Vi kigger på priser for spærtræ og derefter på stolper, remme og spær.

Hvordan mødet gik

1. Gennemgang af bestillingsflow
    - Vi gennemgik Molslinjens bestillingsflow som inspiration
    - Diskuterede hvordan vi kunne tilpasse og implementere et lignende flow til vores projekt
    - Noterede nødvendige trin og besluttede at prioritere brugervenlighed
2. Planlægning af implementeringen af bestillingsprocessen
    - Vi besluttede, at første trin skulle være indtastning af bredde og længde for at generere en tegning
    - Andet trin skulle være indtastning af email og kodeord
    - Tredje trin ville involvere at sende en forespørgsel
3. Statusopdatering på SVG-arbejdet
    - Gruppemedlemmerne på SVG-arbejdet rapporterede om deres fremskridt
    - Vi diskuterede eventuelle udfordringer og brainstormede løsninger
    - Bekræftede, at arbejdet på SVG-erne var på rette spor, og at de kunne integreres i bestillingsflowet
4. Fordeling af opgaver til beregning af styklisten med calculator
    - Vi fastlagde, at alle fire teammedlemmer ville starte sammen på beregningen af styklisten
    - Efter de indledende trin ville to fortsætte med SVG-arbejdet, mens de andre to fokuserede på videreudvikling af beregneren
    - Aftalte at holde regelmæssige briefing-sessioner for at sikre, at alle var opdaterede og kunne bidrage effektivt

Resultat af Mødet

- Klar plan for bestillingsflowet og trin i processen

- God fremdrift på SVG-arbejdet og klarhed om næste skridt
- Effektiv uddeling af opgaver og tidsplan for beregning af styklisten
- Aftale om regelmæssige opdateringer for at sikre løbende koordinering og problemløsning

Mødet sikrede, at alle teammedlemmer var på samme side, og vidste præcis, hvad deres roller og ansvar var i den næste fase af projektet. Dette forbedrede vores effektivitet og samarbejde.

Arbejdsprocessen reflekteret

Kanban-mester-rolle

Vi har ikke haft en KanBan-mester, men en i teamet har været primus motor i forhold til flowet i arbejdsprocessen. I næste projekt kan det være en fordel at uddele posten til et teammedlem, så der altid er en der har ansvaret for at have et større overblik over arbejdsprocessen og selve Kanban-board.

Evalueringsmøder

Vi har stort set hver dag haft et planlægningsmøde om morgenen, men vi kunne godt have haft et evalueringsmøde ved slutningen af hver dag for at have et bedre arbejdsflow og overblik over huller i projektet og deadlines.

User stories og tasks

Vi kunne godt være bedre til at uddele 'user stories' og individuelt løse dem til ende. Vores approach i praksis var mere sporadisk end førnævnte.

Vi havde fra starten af gjort os den fejl at vi lavede for brede 'user stories'. Vi måtte derfor bryde dem ned i mindre stykker og kunne derfor godt sidde flere om en user story.

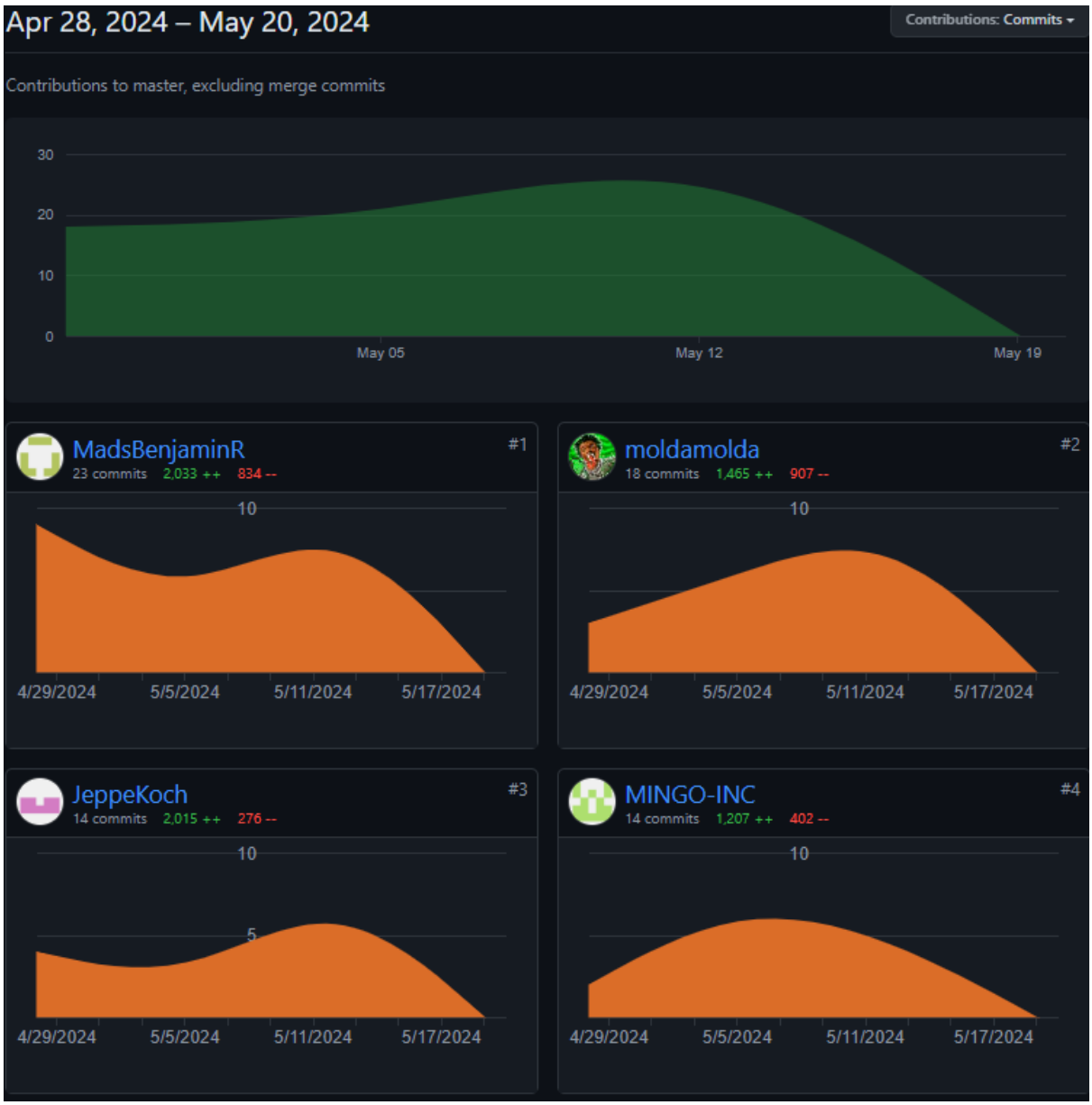
Tid og deadline

I og med at vi lagde en plan for dagen hver morgen vi mødtes, havde vi langt størstedelen af tiden godt styr på hvor langt vi var og hvad vi skulle lave. Vi havde aldrig estimater for hvor lang tid vi skulle sidde. Vi udførte vores opgave, og ved dagens ende tog vi hjem og lavede måske en aftale om en smule hjemmearbejde eller lavede en 'TODO' for i morgen.

Vejledning og demoer

Fra starten af projektet lagde vi en solid grundstruktur, som gjorde det muligt for os at komme hurtigt i gang. Den klare retning betød, at vi fra begyndelsen vidste præcis hvad der skulle laves. Som resultat fulgte vi tidsplanen nøje gennem størstedelen af projektet, og vores demoer levede op til de fastsatte mål til hvert kundemøde.

Identificering af produktivitet



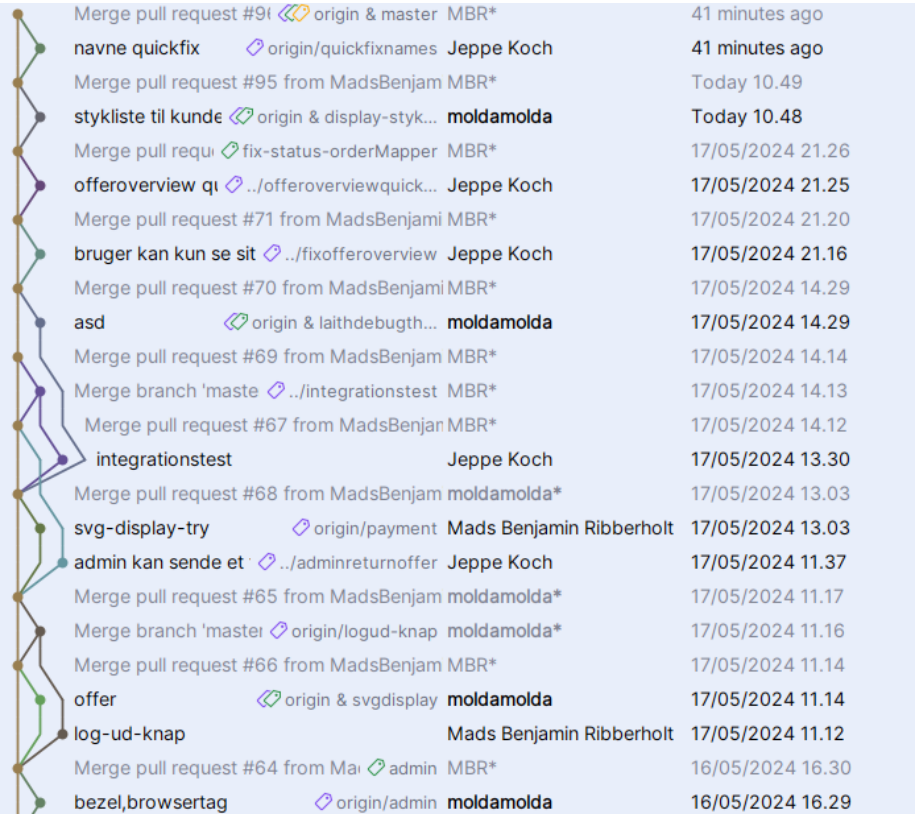
Vi etablerede hurtigt en rytme, hvor vi delte store opgaver op i mindre dele. Når man så blev færdig med sin egen delopgave, vendte man først fokus mod teammedlemmer, der ikke var færdige med deres delopgave. Herefter gik man enten selvstændigt i gang med endnu en delopgave, eller fandt ud af det i fællesskab. Som man kan se på billedet nedenfor, har alle i teamet deltaget og kodet deres respektive del. Der skal dog tages højde for at noget er kodet i code-with-me og kontributionen er derfor ikke 100% retvisende. Se billedet ovenfor.

Github

Grundet den branch protection vi havde sat op, og en omfangsrig gitignore fil, kunne 99% af koden auto merge når en collaborator havde lavet et review. Kun 2 gange under

hele projektet var der mergekonflikter på masterbranchen ved pull request, der dog nemt og hurtigt blev udredt.
Vi havde en forseelse ved et review af pull request, der resulterede i at et personligt password kom med ind i master branchen. Da det er en længere og risikabel proces at fjerne gamle commits fra historien, gjorde vi det private repository, hvorefter vi lavede et nyt public repository at aflevere i.

Ekstra bemærkninger



Vi skal fremover være bedre til at lave retvisende og præcise navne for vores klasser, metoder, branches og kommentarer i commits. Se billedet ovenfor.