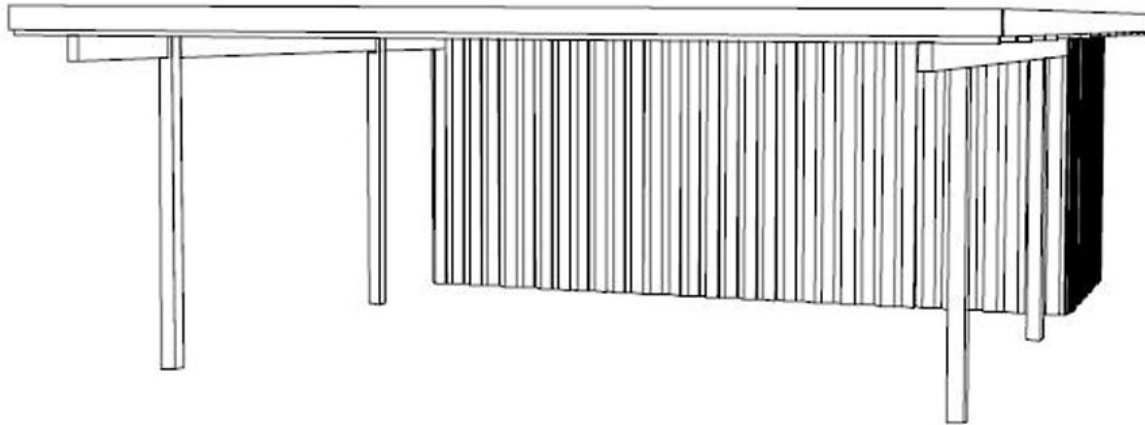


FOG Carport



Marcus Forsberg, cph-mf411@cphbusiness.dk, MarcusPFF

Jonathan Kudsk, cph-jk513@cphbusiness.dk, JonathanKudsk

Jonas Outzen, cph-jo221@cphbusiness.dk, JonasOutzen

Semesterprojekt – Hold A – 28/05/2025

Indledning	3
Problemformulering	3
Mål og succeskriterier	3
Overordnede mål	4
Sekundære mål	4
Succeskriterier	4
Forretningsforståelse	5
Interessentanalyse	5
Risikoanalyse	7
Kravspecifikation	9
Funktionelle krav	9
Ikke-funktionelle krav	9
User stories	10
Hvilke krav blev opfyldt	10
Analyse og Modellering	11
AS-IS diagram	11
TO-BE diagram	12
Domænemodel	13
UI og Design	14
Figma	14
Typografi og farvepalette	16
UX-principper (Laws of UX)	16
System environment variables	17
Brug af Scaling Vector Graphics (SVG)	17
Responsivt Design	20
Sendgrid til Mails	21
Valg af teknologier og værktøjer	25
Test	26

IntegrationsTest.....	27
UnitTest	28
Deployment.....	30
Digital Ocean.....	30
Caddy server	31
DNS.....	32
Udviklingsproces.....	33
Repositorystruktur og Arbejdsregler.....	33
CI/CD workflows (inkl. automatiserede integrationstest og JUNIT)	33
Pull request og code reviews.....	34
Brug af Github Projects (Brug af kanban koncept)	34
Databasedesign og ERD	36
Konklusion og refleksion	39
Hvad gik godt.....	39
Hvad kunne forbedres.....	39
Læring	40
Konklusion	41
Video og Links	42
Bilag	43

Indledning

Denne rapport beskriver den proces, de undersøgelser og overvejelser samt den systemudvikling, der ligger til grund for udviklingen af en hjemmeside til salg og opførsel af carporte for Johannes Fog.

Rapporten gennemgår centrale aspekter af projektet – fra problemformulering og målsætninger til analyse, design, implementering og test. Vi beskriver forretningsmæssige overvejelser, fastslår interesser, samt vurderer og stiller krav til systemet.

Udviklingsarbejdet dokumenteres gennem user-stories, UI-designs og modeller/diagrammer heriblandt AS-IS, TO-BE, domænemodeller, tillige med klassediagrammer.

Derudover redegøres for anvendte teknologier og sikkerhedshensyn i udviklingsprocessen, herunder bruges GitHub, Docker og SendGrid. Rapporten dækker også over brugen af pull requests tilmed unit- og integrationstest til kvalitetssikring. Afslutningsvis behandles deployment, udviklingsproces og databasedesign. Rapporten afrundes med en konklusion og refleksion over forløbet.

Problemformulering

Hvordan kan vi, ved hjælp af Javalin og Thymeleaf, udvikle en professionel og brugervenlig webapplikation for Johannes Fog, der gør det muligt for kunden hurtigt at konfigurere en carport med fladt tag? Løsningen skal inkludere en dynamisk visualisering af carporten via SVG-tegning, en beregner der automatisk udregner prisen baseret på brugerens valg, samt en funktionalitet til at generere og sende et tilbud via e-mail.

Mål og succeskriterier

I dette projekt har vi ønsket at skabe en webbaseret carport-konfigurations løsning til Johannes Fog, der giver en god kundeoplevelse og sørger for at kundens flow ikke går i stå. Vi har defineret en række mål og nogle succeskriterier, som vi kan bruge som en rød tråd gennem udviklingsforløbet.

Overordnede mål

Vores primære ambition har været at sætte brugeren først. Vi bruger *3-minutter-princippet*, som går ud på, at kunden skal kunne designe sin carport, og nå at få sendt et præcist prisforslag direkte på sin mail på under tre minutter. På baggrund af det, ville vi automatisere hele beregningsprocessen og gøre hele flowet stabilt, idet stabilitet er afgørende for os. Der må på intet tidspunkt ske frontend fejl, hvilket betyder, at vi skal fange så mange *exceptions*, som overhovedet muligt.

Sekundære mål

For at understøtte de overordnede mål har vi sat fokus på disse sekundære mål:

- **Responsivt design**
 - Et responsivt design er nødvendigt for kundens oplevelse, derfor har vi målsætninger om, at programmet skal kunne køre på både tablets, smartphones og computerskærme.
- **Email-integration**
 - En email-integration som leverer tilbud og stykliste til kunden.
- **Test coverage**
 - For at have et program, der kører sikkert og oprigtigt, har vi målrettet os mod en 80% dækning på unit- og integrationstests, via vores CI/CD-pipeline på GitHub Actions.

Succeskriterier

Når projektet er færdigt, vurderer vi vores success ud fra disse punkter:

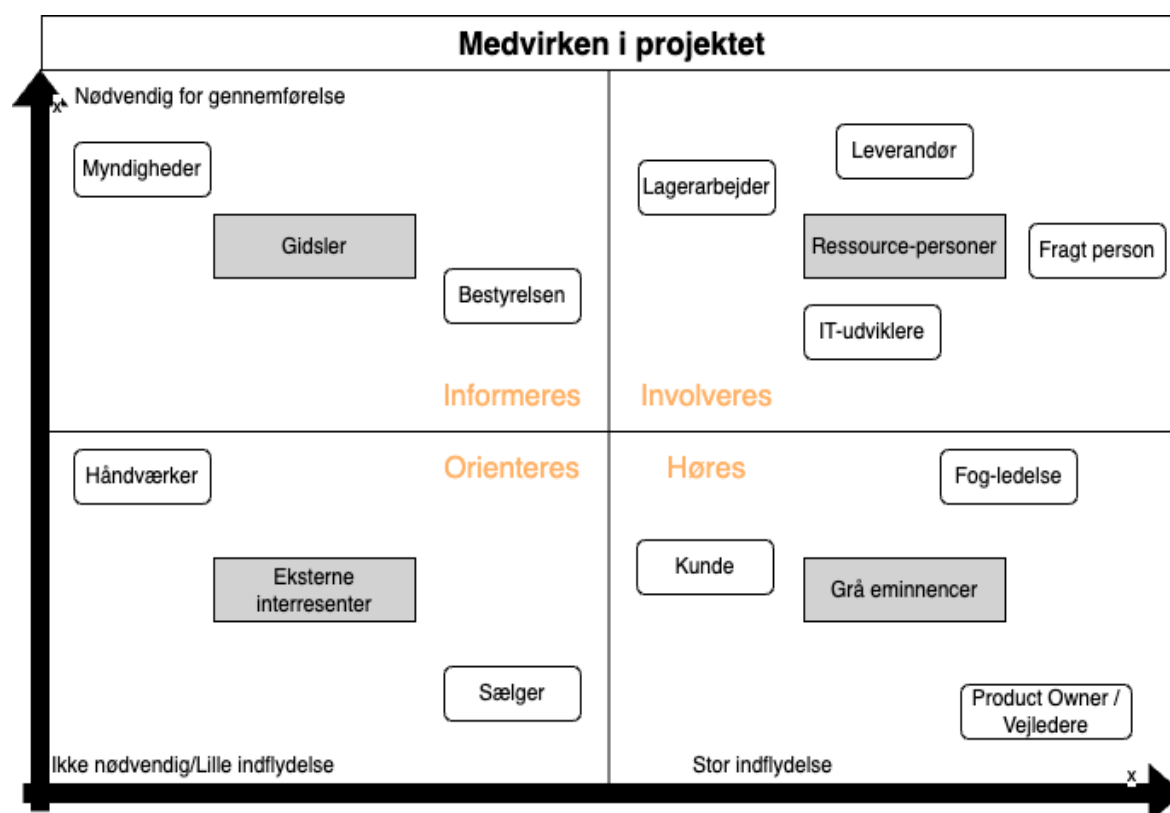
- **Funktionalitet**
 - Kunden kan gennemføre en fuld konfiguration, med en skitsetegning og modtage et fejlfrit tilbud på e-mail inden for fem minutter, og alle prisberegninger stemmer overens med virksomhedens prislister.
- **Design**
 - For at opnå et ensformigt og stilagtigt design, skal vi forinden have lavet skitseringer, der sørger for at hele teamet er på samme plan.

Forretningsforståelse

For at kunne udvikle en målrettet og værdiskabende løsning har det været essentielt for os at få en dybere indsigt i forretningen og dens behov. Allerede tidligt i forløbet samarbejdede vi om at udarbejde forskellige analyser med det formål at opnå den nødvendige indsigt og forstå, hvordan projektet har påvirket virksomheden.

Derfor har vi blandt andet udarbejdet en interessentanalyse for at identificere de forskellige grupper og roller med interesse i projektet, samt hvordan deres behov og forventninger påvirker udviklingsprocessen. Derudover har vi udarbejdet en risikoanalyse for at imødekomme potentielle udfordringer, der kunne hæmme projektets fremdrift.

Interessentanalyse



Figur 1: Interessentanalyse

Vores interessentanalyse er delt ud i fire sektioner. *Informeres*, *Involveres*, *Orienteres*, *Høres*, hvorhen i hver sektion er der de medvirkende – *Gidsler*, *Ressource-personer*, *Eksterne-interessenter*, *Grå eminencer*. I vores udarbejdede analysemodel, er det muligt at se hvilke personer vi føler, vil have større indflydelse end andre til udviklingen af vores projekt.

I *Informeres*-sektionen har vi dem, som er nødvendige for at kunne gennemføre projektet, men som ikke har direkte indflydelse på udviklingen. Her findes følgende roller – **Myndigheder** og **Bestyrelsen**. Disse aktører skal der tages højde for, da deres krav og godkendelser er afgørende for projektets fremdrift. Det kan godt være, de ikke aktivt er en del af udviklingsprocessen, men det er stadig vigtigt at de bliver informeret undervejs om status og fremgang.

I *Involveres*-sektionen har vi dem, som har stor betydning for udviklingen af projektet. Her findes følgende roller – **Lagerarbejder, Leverandør, Fragtperson** og **IT-udviklere**. Disse aktører er et essentielt fundament for at kunne gennemføre projektet, og deres aktive deltagelse er afgørende for, at løsningen bliver succesfuld.

I *Orienteres*-sektionen har vi dem, som ikke nødvendigvis har nogen direkte indflydelse på projektets fremdrift, men som alligevel skal holdes orienteret omkring projektets udvikling fra tid til tid. Her findes følgende roller – **Håndværker** og **Sælger**

I *Høres*-sektionen har vi dem, som har stor indflydelse på forståelsen af projektet og på at sikre, at de involverede aktører bevæger sig i den rigtige retning. Her findes følgende roller – **Kunde, Fod-ledelse, Product Owner**. Deres mening er værdifuld for at projektet udvikles med de behov og forventninger, der er blevet lagt.

Risikoanalyse

Vores risikoanalyse er en analyse, som tager forskellige risici i betragtning, og vurderer dem ud fra hvor kritisk/sandsynlige de er. Udover det, har vi lavet en plan for forebyggelse ud fra hver risiko her i teamet, så vi formindsker muligheden for evt. hæmmelse af fremdrift. Selve skemaet er lavet ud fra en formel der hedder:

$$\text{Alvor} * \text{Sandsynlighed} = \text{Risikoniveau}$$

Dette giver risikoniveauet en karakter som enten *Lav*, *Medium*, *Høj* eller *Ekstrem*. Ud fra denne vurdering kan teamet få et overblik over, hvilke risici der er mest kritiske, samt hvor stor sandsynligheden er for, at de opstår.

Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau	Plan for forebyggelse
1	Sygdom i udviklingsteamet	Acceptabel	Muligt	Høj	Kan ikke forebygges.
2	Familie nødstilfælde	Tolereres	Usandsynlig	Medium	Kan ikke forebygges, men god kommunikation kan mildne udkommet.
3	Forsinkelser	Uønsket	Muligt	Medium	Nogle forsinkelser vil være ude af vores hænder, men ellers kom afsted i god tid.
4	Dårlig kommunikation i team	Uønsket	Sandsynligt	Medium	Sørge for at have jævnlige møder og dele information
5	Dovenskab/udeblivelse	Uacceptabelt	Muligt	Ekstrem	Disciplin – dette skal ikke ske
6	Fejl i kritisk kode	Uønsket	Sandsynligt	Ekstrem	JUNIT & Integrationstest
7	Fejl i Databasen	Uønsket	Sandsynligt	Ekstrem	Sørge for at teste databasen, og sætte den rigtigt op fra starten af.

Figur 2: Risikoanalyse

	Hvad kan gå galt?	K	S	K X S	Forebyggende handlinger	Afbødende handlinger
1	Sygdom i udviklingsteamet	4	1	4	Kan ikke forebygges	Arbejd hjemmefra
2	Familie nødstilfælde	1	1	1	Kan ikke forebygges	Her kan det måske lade sig gøre at arbejde hjemmefra.
3	Forsinkelser	1	5	5	Kan være svære at forebygge.	Arbejde længere i den anden ende.
4	Dårlig kommunikation i team	3	2	6	Sørge for at holde møder.	Når skaden er sket, skal man bruge den tid det tager til at få alle i samme retning igen.
5	Dovenskab/udeblivelse	5	1	5	Disciplin.	Arbejd over en anden dag.
6	Fejl i kritisk kode	5	3	15	Masser af tests.	Sætte sig i gruppen til problemet er løst.
7	Fejl i Databasen	4	2	8	Gennemgå databasen før den laves (skitsering).	Sætte sig i gruppen til problemet er løst.

Figur 3: Risikoanalyse med risiko-score

Kravspecifikation

Funktionelle krav

Vi havde til opgave at designe et program, der med en brugergrænseflade og et backend system kunne tage en kunde gennem en bestilling af en carport. Det skulle for en kunde være nemt at åbne vores hjemmeside og starte en "quick-byg" session, hvor kunden ville være fri til selv at bestemme mål, og dernæst vælge hvorvidt de skal have et skur eller ej. Herefter skal det være muligt at skrive sine kundeinformationer, og bestille et tilbud på mail, der vil være computer beregnet, dette tilbud skal også indeholde et tilbudsnummer. Kunden skal nu have mulighed for at blive kontaktet af en sælger ud fra dette tilbudsnummer, og sammen kunde og sælger findes en slutpris. Sidst skal kunden have mulighed for at bekræfte eller afkræfte den aftalte pris. Hvis denne bekræftes, skal kunden modtage en stykliste per mail.

Ikke-funktionelle krav

For at kunne agere som en sælger og redigere priser, skal der være en sikkerhedskode og login-funktion. På en fuldkommen hjemmeside, vil det slet ikke være muligt for en kunde at trykke på et link, der redirecter til en sælger side, men da alle dele i programmet skal kunne vises, laver vi en login funktion med en pinkode på forsiden.

Programmet skal deployes med et domænenavn og køre via en droplet-server på digitalocean, dette vil gøre det nemt at tilgå fra de fleste browsere.

Programmet skal gøres brugervenligt både på telefoner og computere. Dette betyder, at vi skal sørge for at gøre programmet responsivt og tilpasset til små skærme.

Programmet skal være nemt at opdatere, så koden skal være velstruktureret og dokumenteret. Databasen skal på samme måde være velstruktureret, og overholde de tre normalformer. Sidst er det nødvendigt at hovedkode testes, så fejl ikke sker på en kunde-tilgængelig side.

User stories

US-1: Kunden skal kunne vælge sine oplysninger ud fra en dropdown menu.

US-2: Kunden skal kunne indtaste sine oplysninger inden betaling (navn, efternavn, adresse og mail)

US-3: Beregningssystem skal kunne vise en cirka pris til kunden (Dette skal være den højest mulige pris ud fra kundens mål).

US-4: Kunden skal kunne få et uforpligtende tilbud uden stykliste. Dette sendes til kundens mail.

US-5: Sælger skal have mulighed for at opdatere prisen via web-applikationen i sammenspil med kunden. (Sælger har en kode til login, så kunder ikke har adgang).

US-6: Takker kunden "ja" til tilbuddet (Prisen), skal en stykliste sendes til kunden ud fra de oplysninger, sælger har fået fra kunden.

Disse User-Stories beskrives som *Nice To Have*, og er ikke påkrævet for programmets virkning.

US-7: I tilfælde af beregninger hvor programmet "loader" i et længere tidsrum, skal der indsættes et loading ikon. (En lille cirkel der drejer rundt).

US-8: Lav Carport SVG 3D, så man kan dreje den rundt.

US-9: Fake stripe betalingsside – før kunden modtager stykliste.

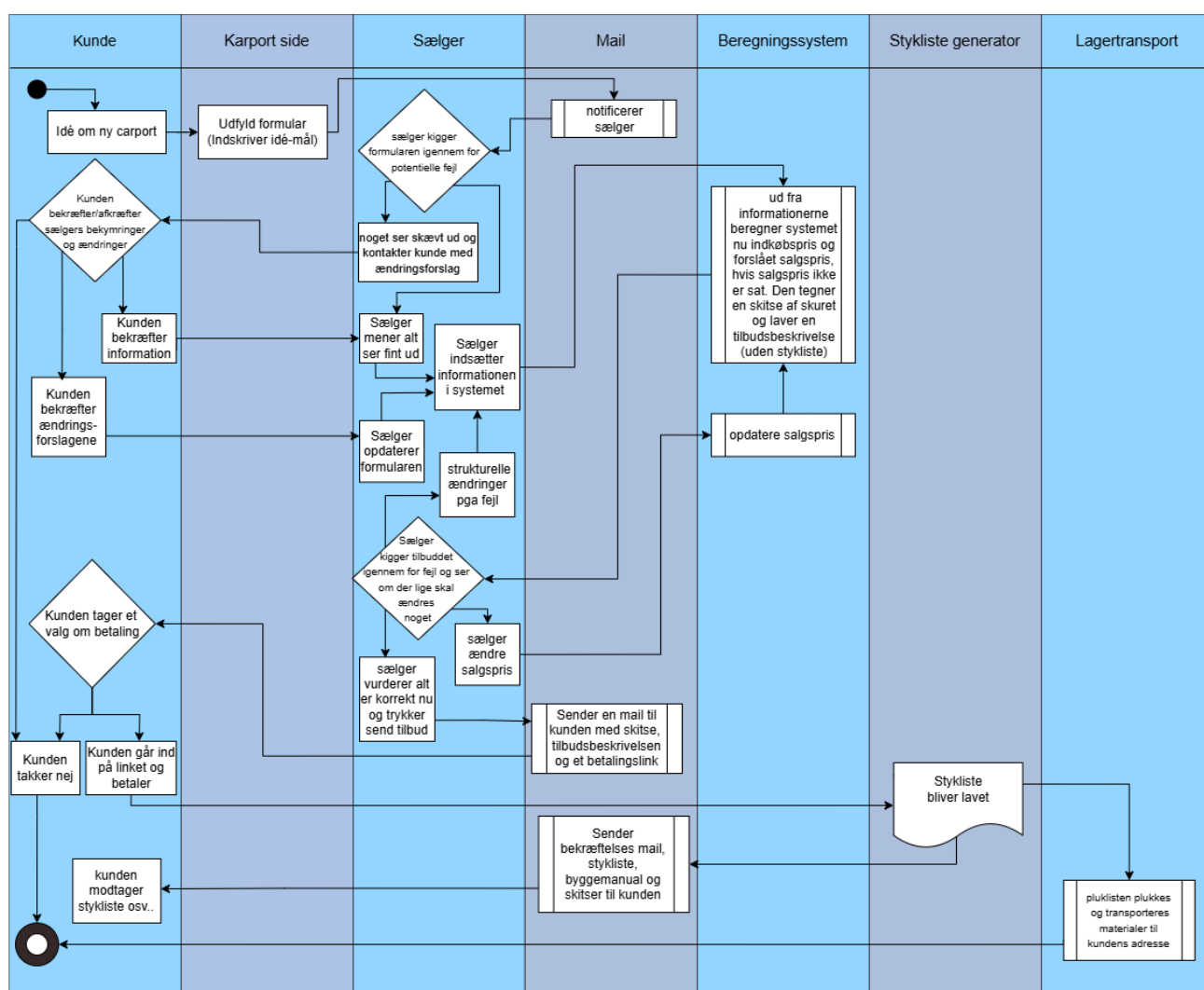
Hvilke krav blev opfyldt

Alle vores primære User Stories blev opfyldt, hvilket betyder, at den grundlæggende funktionalitet og brugeroplevelse, vi havde sat som målsætning, blev opfyldt. Vi valgte bevidst at fravælge de sekundære User Stories for at sikre, at projektets vigtigste dele blev implementeret med høj kvalitet.

Analyse og Modellering

AS-IS diagram

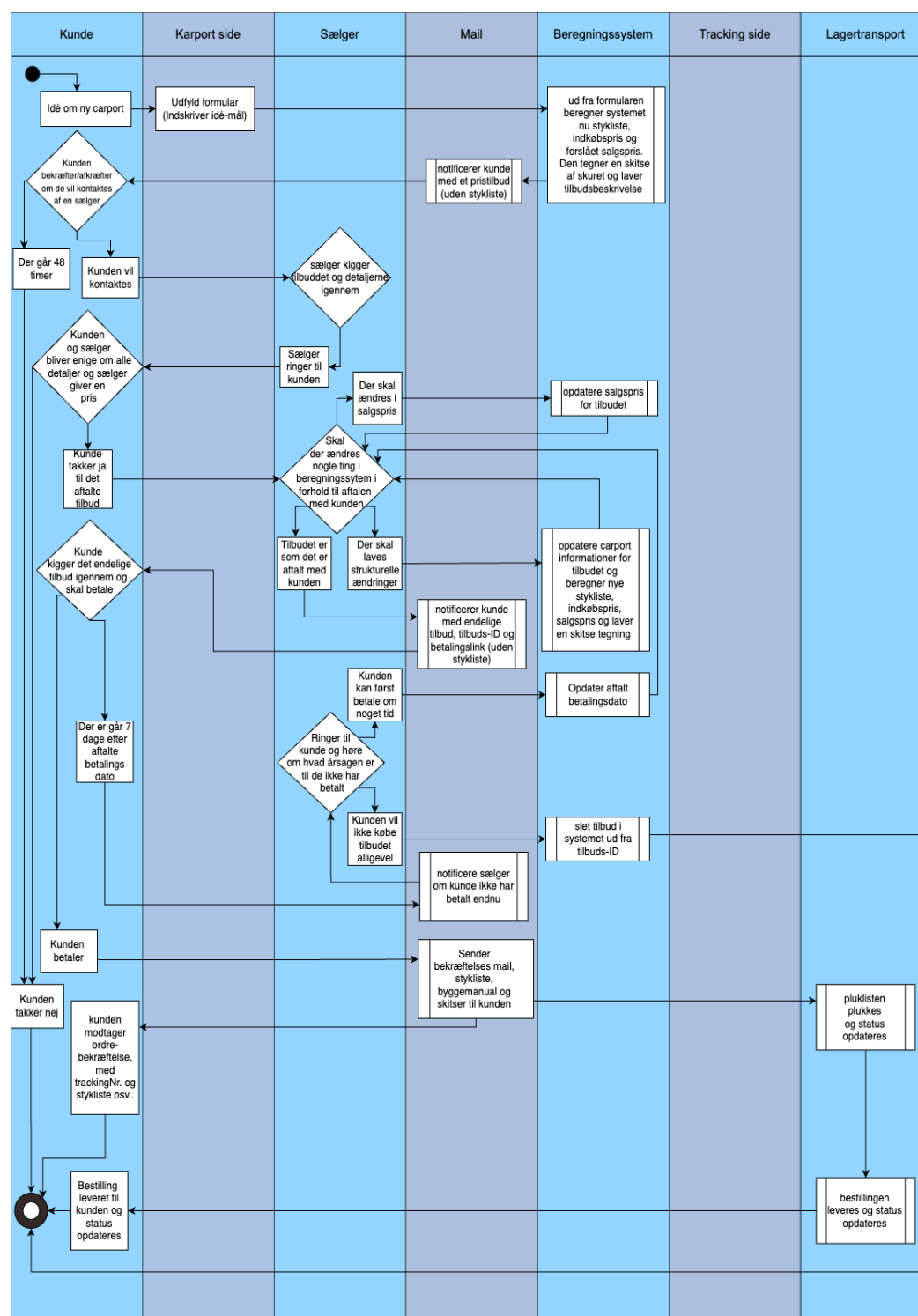
Vores AS-IS-diagram er udarbejdet på baggrund af et interview med ledelsen hos Johannesfog.dk. Formålet med dette diagram er at påpege hvordan deres nuværende process-flow fungerer fra start til slut. Det gav os et solidt udgangspunkt for at forstå virksomhedens behov og danne grundlag for udviklingen af vores forbedrede TO-BE-model.



Figur 4: AS-IS-Diagram

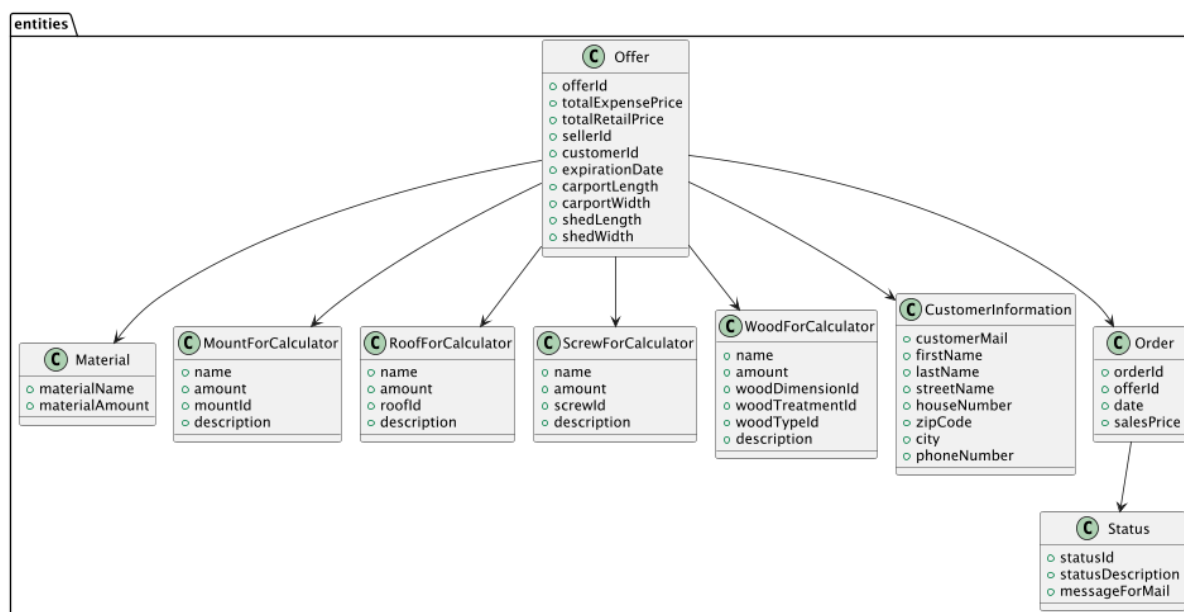
TO-BE diagram

Nedenfor ses vores endelige TO-BE-diagram, som illustrerer den ønskede fremtidige proces. Diagrammet er udarbejdet med udgangspunkt i vores analyse af den nuværende arbejdsgang samt de forbedringsmuligheder vi har identificeret.



Figur 5: TO-BE-Diagram

Domænemodel



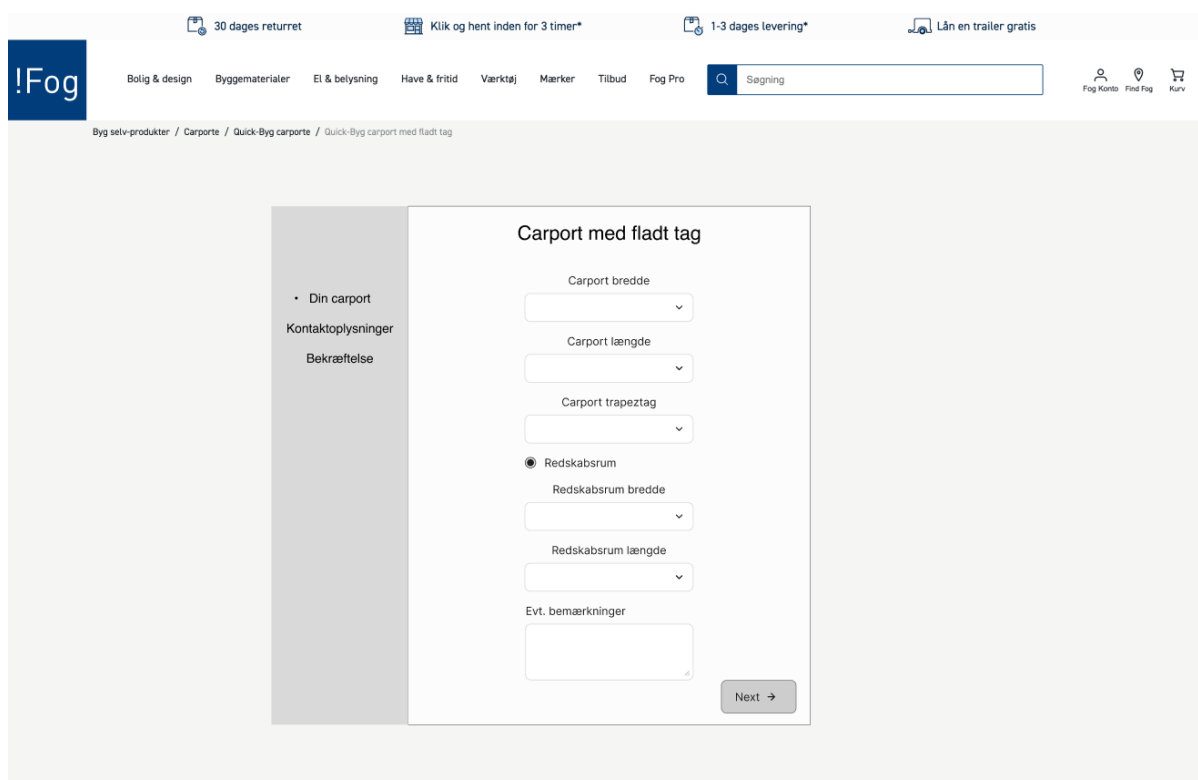
Figur 6: Domænemodel

Gennem projektet har domænemodeller, af forskellige versioner, været brugt til at holde en overordnet rød tråd. Disse sørger for at vi som team har en overensstemmelse for hvilke relationer, der skal være i programmet og hvordan selve domæne-forløbet skal forløbe. På denne måde undgås fejl i forventningen mellem teamet. Med henblik på klassediagrammer, har vi undladt at bruge disse til andet end opstarten på projektet, idet vi hver dag har siddet sammen og holdt møder om hvilke metoder og klasser, der har skulle bruges i programmet. Derudover nævnes det også, at vi bruger GitHub Projects til at overskue og opdele arbejdsopgaver.

UI og Design

Figma

Efter udarbejdelsen af både vores AS-IS- og TO-BE diagram, havde vi en idé til hvordan vores prototype af det endelige produkt skulle se ud. Ved at arbejde i Figma kunne vi sikre, at hele teamet var enige om det visuelle designforslag og den funktionelle opbygning. Samtidig fungerede prototypen som skabelon for det videregående arbejde og sørgede for, at vores designvalg blev dokumenteret undervejs.



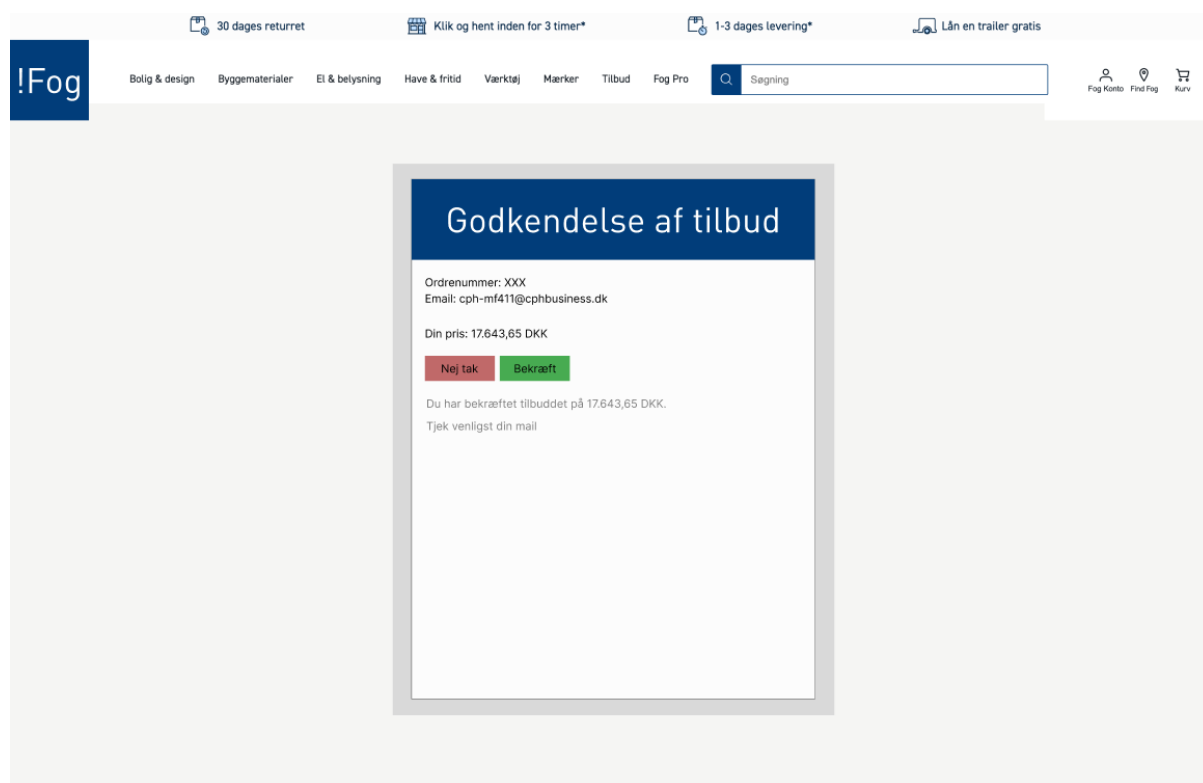
The screenshot shows a Figma design sketch of a web page for configuring a carport. The page layout includes a header with navigation links (Bolig & design, Byggematerialer, El & belysning, Have & fritid, Værktøj, Mærker, Tilbud, Fog Pro) and a search bar. The main content area is titled "Carport med fladt tag" and contains a form with the following fields: "Carport bredde", "Carport længde", "Carport trapeztag", "Redskabsrum" (selected), "Redskabsrum bredde", "Redskabsrum længde", and "Evt. bemærkninger". A "Next" button is located at the bottom right of the form. A sidebar on the left shows a progress indicator with the following items: "Din carport", "Kontaktoplysninger", and "Bekræftelse".

Bilag 1: Skitsetegning fra Figma – QuickByg Carport med fladt tag side.

Foroven ses et eksempel på en af vores designskitser lavet i Figma. Her har vi forsøgt at visualisere så meget som muligt af det, vi ønsker i det færdige produkt, helt fra det overordnede layout til de mindste visuelle detaljer. Designskitsen dækker over headeren i toppen, indholdet i midten og ned til de mindste detaljer såsom den lille pilfigur i bunden. Målet er at lave disse sider så detaljeret som overhovedet muligt, så personen, der skal udvikle dette stykke front-end, ikke selv skal tænke på designet.

Skitsen her repræsenterer vores QuickByg carport-side og er den første side, brugeren møder når de klikker sig ind via QuickByg linket. På denne side har brugeren mulighed

for at indtaste de ønskede mål og data til deres carport. Derudover kan man vælge om man ønsker et redskabsrum og hvis det bliver tilvalgt, dukker de ekstra felter automatisk frem. Dette gør processen både intuitiv og brugervenlig, da brugeren kun bliver præsenteret for de informationer, der er nødvendige i forhold til deres valg.
















Bilag 2: Skitsetegning fra Figma – Godkendelse af tilbud

Denne designskitse er en af de sidste sider man ser i vores projekts flow. Her bliver der vist tydeligt og klart til teamet, hvad der skal laves, og hvor de forskellige elementer skal placeres. Udover det, bliver der brugt en grålig farve med en gennemsigtighed på 50%, som viser over for teamet at netop den handling først skal blive vist, når *Bekræft* knappen er trykket.

Typografi og farvepalette

Følgende farver er brugt i vores HTML til at lave brugergrænsefladen:

Primær baggrundsfarve	#FFFFFF	
Hvid nuance (Tekstfarve på mørk baggrund)	#FCFCFC	
Sort (Tekst som overskrifter eller vigtige meddelelser)	#000000	
Sort (Enkelte Skygger)	#000019	
Dyb blå (Primær tema farve – Logoer og knapper)	#013D7A	
Dyb blå (Hover baggrundsfarve til knapper)	#013469	
Lys blå (Sekundær baggrundsfarve på sælgerside)	#6C9FD1	
Lys grå (Wrappers samt små borders)	#DDDDDD	
Lys beige (Små borders på sælgerside)	#C5B79B	
Lys grå (Baggrund for sidemenuer)	#F4F4F4	
Mørk grå (Checkbox-beskrivelser)	#777777	
Klar rød (Enkelte knapper på forsiden)	#ff0000	
Mørk rød (Hover-effekt til enkelte knapper på forsiden)	#9f1818	

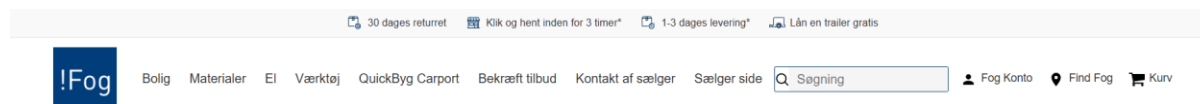
UX-principper (Laws of UX)

I designet af brugergrænsefladen har vi valgt at læne os op ad centrale principper fra *The Laws of UX*. I den kundeorienterede del af programmet, har vi blandt andet lagt et stort fokus på *Fitts* lov og *Goal gradient effect*, der henholdsvis beskriver at knapper skal være store nok til, at kunder ikke misser dem (Fitts), og at der skal være noget progress-beskrivende (Goal gradient). Et eksempel på dette er vores brugergrænseflade for Quick-Byg siden, når den skaleres ned til størrelsen af en iPhone 12 Pro, da det her er



Bilag 3: QuickByg HTML side –
smartphone skalering

muligt at se begge love. Det samme billedeksempel kan også bruges til at illustrere anvendelser af *Jakobs lov*, *Millers lov* og *proximity-loven*. Placeringen af menubaren i toppen kender kunden fra andre hjemmesider (Jakobs lov) og et meget rent og simpelt interface (Millers lov) gør det meget nemt for en kunde at finde rundt. Sidst er det nemt at finde ud af hvor inputtet hører til, da spacing mellem felterne er sat tydeligt op (Proximity-loven).



Bilag 4: Header til alle HTML sider

The Law of Similarity bruges i vores header. Her er farver og små borders med til at samle funktionaliteter. Headeren bliver på den måde delt op i en primær header (midten) og en sekundær header (toppen).

The Laws of UX har altså hjulpet os til et design, der undgår at blive uoverskueligt for en kunde.

System environment variables

Vi uploader ikke sendgrid api-keys, database password eller andet sensitive informationer man ikke ønsker i et public repository. Derfor bruger vi *System environment variables*. Disse variabler definerer man lokalt på sin computer, så systemet forstår hvilken værdi variabelen *SENDGRID_API_KEY* har.

```
String API_KEY = System.getenv("SENDGRID_API_KEY");  
String PASSWORD = System.getenv("kudsk_db_password");
```

Brug af Scaling Vector Graphics (SVG)

Vi har i projektet anvendt *Scaling Vector Graphics* eller forkortet SVG, til at skitsere plantegninger for hvordan det endelige skur ville stå. Grunden til vi har valgt at bruge SVG, frem for andre tegneprogrammer er, som det ligger i navnet, skalering. SVG gør det muligt at zoome ind og ud uden opløsningsproblemer. Dette betyder, at man vil kunne se plantegningen både på smartphones, tablets og computerskærme. Samtidig er SVG nemt at arbejde med og implementere i HTML med Thymeleaf.

For at se hvordan vores plantegning bliver lavet skal vi starte med at kigge i vores routing controller. Her findes metoden `showSvgDrawingPage()`, der bruges til at vise tegningen ud fra de mål, kunden har valgt.

```
public static void showSvgDrawingPage(Context ctx) {
    loadAllAttributes(ctx);
    Locale.setDefault(new Locale("US"));
    int carportLength = getCarportLength(ctx);
    int carportWidth = getCarportWidth(ctx);

    // Selve tegningen laves her
    CarportSvg svg = new CarportSvg(ctx, carportLength, carportWidth);
    ctx.attribute("svg", svg.toString());
    ctx.render("quick-byg-svg-drawing.html");
}
```

I den første del af metoden gemmes SVG'ens nødvendige parametre, og i den efterfølgende del bliver selve tegningen konstrueret ud fra disse værdier. For kunden sker alt dette på samme tid, men i koden bliver kundens data først loadet i `loadAllAttributes(ctx)` og dernæst sat ind på parametrenes plads på de følgende tre linjer. I nedre halvdel af koden bliver der lavet et nyt SVG-objekt via `CarportSvg` konstruktøren hvorpå parametrene `carportLength` og `carportWidth` bliver udfyldt. En længde og bredde er nu defineret.

I konstruktøren kan vi så se hvordan hver del af tegningen laves.

```
addOuterFrame();

this.hasShed = ctx.sessionAttribute("redskabsrumCheckbox");
if (hasShed) {
    int shedLength = ctx.sessionAttribute("redskabsrumLength");
    int shedWidth = ctx.sessionAttribute("redskabsrumWidth");
    addShedWithPoles(shedLength, shedWidth);
}
addPoles();
addBeams();
addRafters();
addDimensions();
```

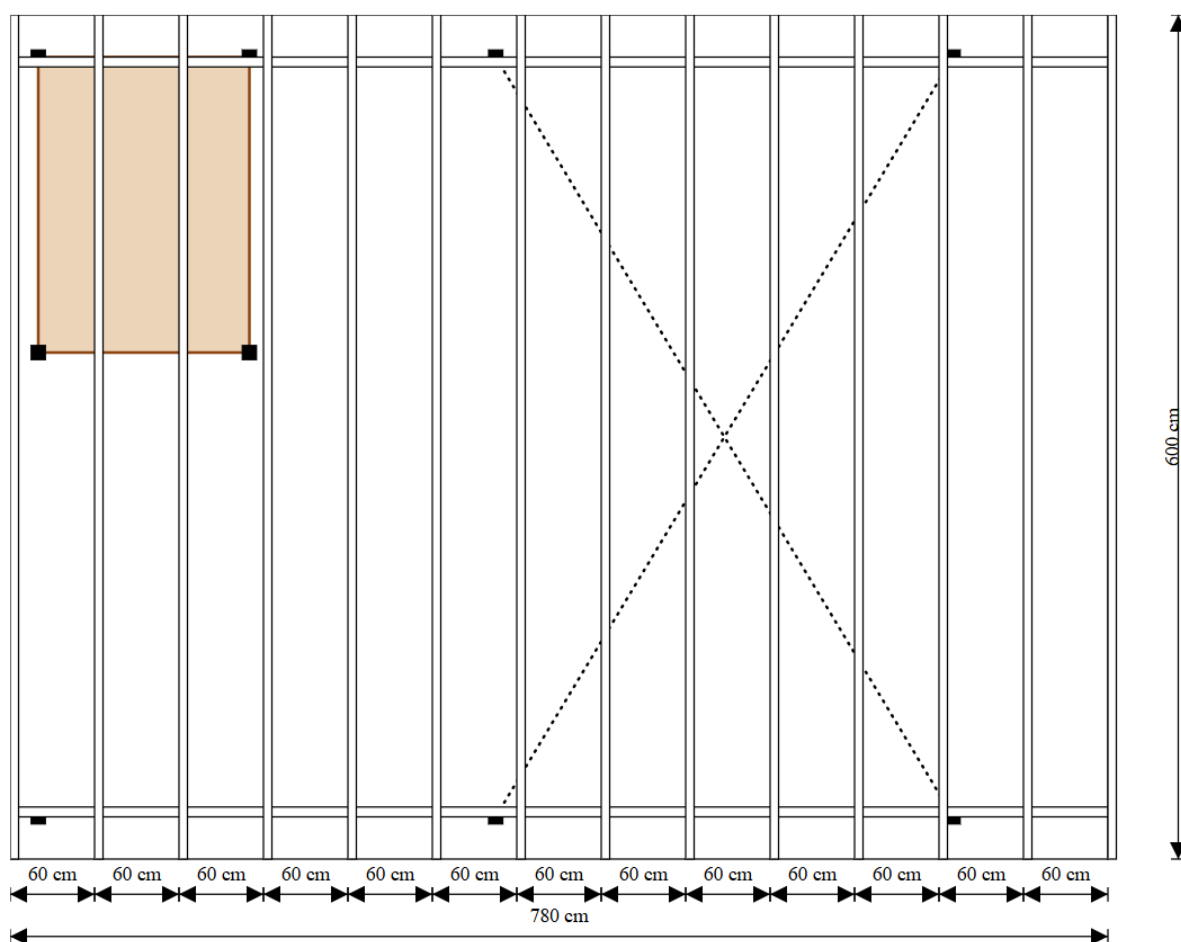
Dette er en del af konstruktøren hvor hver metode til at tegne SVG'en kaldes. Det er disse metoder, der tegner hele plantegningen for carporten, og det er altså her, vi bruger længden og bredden fra kundens oplysninger.

Dette CarportSvg-objekt bliver nu sat på svg-attributten i routing controlleren via en `svg.toString()`, så alle oplysninger nu kan omskrives til SVG-syntaks.

Når atributten er udfyldt, går vi videre til HTML-delen hvor tegningen realiseres.

```
<div th:if = "${svg != null}" class = "carport-svg-topview">
  <div th:utext = "${svg}"> </div>
</div>
```

Her bliver SVG'en tegnet, såfremt der er indtastet gyldige mål – altså at længde og bredde ikke er tomme. Dette sikrer, at tegningen først vises, når kunden har angivet de nødvendige oplysninger.



Figur 7: Endelig SVG-tegning Carport med skur (780 cm * 600 cm)

Til slut vil en plantegning kunne se således ud.

Ved at benytte SVG kan vi altså skabe en skalerbar og præcis tegning, som automatisk tilpasser sig kundens mål. Dette sikrer en god brugeroplevelse på tværs af enheder, uanset om man bruger smartphone, tablet eller computer, og gør det nemt at videreudvikle og tilpasse løsningen fremadrettet.

Responsivt Design

En stor del af vores fokus som udviklere har ligget på brugervenlighed og tilgængelighed på tværs af enheder. Derfor har vi implementeret et responsivt design, der gør programmet nemt at bruge på både smartphones, tablets og computerskærme. Dette er ideen bag et responsivt design og er opnået gennem brugen af *media queries* i CSS, som ændrer styling baseret på skærmens bredde.

```
@media (max-width: 830px) {...}  
@media (max-width: 650px) {...}  
@media (max-width: 500px) {...}
```

Disse medieførespørgsler justerer layoutet i tre trin, når siden bliver smallere. Så f.eks. menuer, knapper, tekststørrelser og kolonne-inddelinger tilpasses automatisk. På den måde sikrer vi, at brugeren altid får en optimal oplevelse – uanset enheden, og intet ligger udenfor, hvad kunden kan se.

Sendgrid til Mails

En af vores user stories lød således:

US-4: Kunden skal kunne få et uforpligtende tilbud uden stykliste. Dette sendes til kundens mail. For at opnå denne user story skulle kunden have mulighed for at modtage et uforpligtende tilbud direkte i sin mail – uden en vedhæftet stykliste. Til dette formål valgte vi at anvende en platform ved navn SendGrid, som gør det muligt at sende automatiserede e-mails fra systemet med

fuld kontrol over indhold, modtagere og leveringssikkerhed. SendGrid fungerer ved at man opretter en skabelon ud fra hver mail, der har forskelligt indhold. Under dette forløb endte vi med at have fem forskellige skabeloner.

Når man opretter en mail skabelon, genereres der automatisk et unikt Template ID, som man bruger i

TEMPLATE

- > 2SemFogProjekt-LastAcceptAndMaterialListMail
- > 2semFogProjekt-SellerMailAccept
- > 2semFogProjekt-SellerMailContact
- > 2semFogProjekt-FørsteMail
- > 2SemFogProjekt-AndenMail


Bilag 5: Mail-skabeloner der sendes via koden

backenden til at bestemme, hvilken skabelon den pågældende mail skal benytte. For eksempel har skabelonen 2semFogProjekt-FørsteMail følgende Template ID: d-c1ab968df32b4e1dabf28de9e5d863e3.

2semFogProjekt-FørsteMail

Template ID: d-c1ab968df32b4e1dabf28de9e5d863e3 ⓘ

DYNAMIC VERSION



Fog - Første mail
 ACTIVE

Bilag 6: Mail med template ID

I vores java klasse som man finder i vores repository ved stigen:

```
src/main/java/app/persistence/util/MailSender.java
```

Har vi udarbejdet en mail sender metode ud for hvert eneste unikt Template ID. Til visning tager vi udgangspunkt i vores `sendFirstMail()` metode. I denne metode indsætter vi skabelonens Template ID vi ønsker at bruge.

```
try {
    request.setMethod(Method.POST);
    request.setEndpoint("mail/send");

    mail.templateId = "d-clab968df32b4e1dabf28de9e5d863e3";
    request.setBody(mail.build());
    Response response = sg.api(request);
} catch (IOException ex) {
    System.out.println("Error sending mail 1");
    throw ex;
}
```

Det er dog ikke det eneste, man skal tage højde for, selvom vores metode i grove træk er koblet op på en SendGrid-skabelon via et Template ID. Vi skal stadig sende nogle værdier videre, som kan bruges til visning i vores afsendte mail

I vores første mailskabelon har vi fire forskellige Dynamic Template Data-variabler, som indsættes ved hjælp af placeholders i SendGrid. Disse er:

```
{{name}}, {{salesPrice}}, {{offerId}}, {{searchForOfferLink}}.
```

Forneden er der et indblik i, hvordan vores første mail ser ud, når den bliver sendt videre. Naturligvis bliver vores placeholder udfyldt med aktuelt data, når kunden modtager mailen.

Kære {{name}}

Tak for din forespørgsel! Vi har modtaget den og sendt dig et tilbud baseret på de oplysninger, du har givet os.

Estimeret carportpris: {{salesPrice}} DKK

Tilbudsnr: {{offerId}}

For at blive kontaktet af en sælger vedrørende dit tilbud skal du trykke på linket forneden og indtaste dit tilbudsnummer: {{searchForOfferLink}}.

Hvis du har spørgsmål eller har brug for yderligere assistance, er du altid velkommen til at kontakte os.

Venlig hilsen,

FOG Teamet



Bilag 7: Mail-skabelon med placeholders

Måden, vi sender vores placeholder data videre, er ved hjælp af dette stykke kode i vores metode `sendFirstMail()`.

```
public void sendFirstMail(String to, String name, float salesPrice, int offerId, String searchForOfferLink) throws IOException {}
```

Vores metode tager imod følgende parameter

```
String to,  
String name,  
float salesPrice,  
int offerId,  
String searchForOfferlink
```

Når vi tilføjer en dynamisk værdi, oprettes der et nøgle-værdi-par, som SendGrid anvender til at udfylde de relevante felter i skabelonen som for eksempel {{name}}. Disse værdier sendes videre som Dynamic Template Data, som vores skabelon kan læse og indsætte i skabelonens indhold.

Eksempelvis tager vi variablen "name" fra Java-metoden og knytter den sammen med den tilsvarende placeholder {{name}} i mailskabelonen, så modtageren får en

personlig og dynamisk e-mail. Dette sker også for de andre variabler.

```
Email from = new Email("no-reply@marcuspf.com");
from.setName("!Johannes Fog – Team");

Mail mail = new Mail();
mail.setFrom(from);

String API_KEY = System.getenv("SENDGRID_API_KEY");

Personalization personalization = new Personalization();
personalization.addTo(new Email(to));
personalization.addDynamicTemplateData("name", name);
personalization.addDynamicTemplateData("salesPrice", salesPrice);
personalization.addDynamicTemplateData("offerId", offerId);
personalization.addDynamicTemplateData("searchForOfferLink",
searchForOfferLink);

mail.addPersonalization(personalization);
mail.addCategory("carportapp");

SendGrid sg = new SendGrid(API_KEY);
Request request = new Request();
```

Valg af teknologier og værktøjer

- IntelliJ IDEA 2024.2.5 (Ultimate Edition)
- Java SDK (Corretto 17.0.14)
- Maven 17
- Javalin 6.5.0
- Thymeleaf 3.1.3
- JUnit 5.12.0
- Postgres 16.2 (via Docker)
- Docker
- PgAdmin 4 (8.13)
- SendGrid
- PDFBox

Test

Vi har opbygget vores projekt med en test-struktur, der har til formål at sikre kernefunktionaliteten og dataintegrationen i databasen. Dette er henholdsvis gjort med unittest og integrationstest. Vores nuværende testsetup dækker hovedsageligt over beregningslogik og databaseinteraktioner, og vi har derfor lavet en `CreateTestSchemaDatabase()` metode. Denne opbygger en testdatabase, der stemmer overens med den oprigtige database, og kører før hver test. Forinden slettes den eksisterende testdatabase og et rollback køres efter alle test, for at sikre at hver test kan køre uafhængigt af hinanden.

```
src/test/java/app/persistence/mappers/OfferMapperTest.java

@BeforeAll
static void beforeAll() throws SQLException {
    String USER = "postgres";
    String PASSWORD = System.getenv("kudsk_db_password");
    String URL = "jdbc:postgresql://134.122.71.16/%s?currentSchema=test";
    String DB = "Fog_Carport";
    connectionPool = ConnectionPool.getInstance(USER,PASSWORD,URL,DB);
}

@AfterAll
static void afterAll() throws SQLException {
    try (Connection conn = connectionPool.getConnection()) {
        conn.createStatement().execute("ROLLBACK;");
    }
}

@BeforeEach void setUp() throws SQLException {
    try (Connection conn = connectionPool.getConnection()) {
        createTestSchemaWithData(conn);
    }
    offerMapper = new OfferMapper();
}
```

Vi har primært testet vores calculator-metoder med JunitTest. Til dette bruges en typisk AAA opsætning altså(Arrange, Act, Assert), hvilket gør dem nemt læselige. Vi

bruger derudover integrationstest til at teste vores mapper-metoder, som sikrer at vi har stabile og korrekte data input i databasen, og at ConnectionPool håndteres korrekt.

IntegrationsTest

```
src/test/java/app/entities/forCalculator/ScrewForCalculatorTest.java
```

```
@Test
void screwForRoofCalculator() throws SQLException, DatabaseException {
    // Arrange
    int carportOneLength = 510; // Skrue 1
    int carportOneWidth = 330; // Skrue 1
    int expectedAmountOne = 2; // Skrue 1
    int carportTwoLength = 690; // Skrue 2
    int carportTwoWidth = 240; // Skrue 2
    int expectedAmountTwo = 1; // Skrue 2
    int expectedScrewId = 1; // Skrue 1 og 2
    String screwName = "Plastmo bundskruer 200 stk.";
    String description = "Skruer til tagplader.";

    // Act
    ScrewForCalculator screwOne =
screwCalculator.screwForRoofCalculator(connectionPool, carportOneLength,
carportOneWidth, screwName);
    ScrewForCalculator screwTwo =
screwCalculator.screwForRoofCalculator(connectionPool, carportTwoLength,
carportTwoWidth, screwName);

    // Assert
    assertEquals(screwName, screwOne.getName()); // Skrue 1
    assertEquals(expectedAmountOne, screwOne.getAmount()); // Skrue 1
    assertEquals(expectedScrewId, screwOne.getScrewId()); // Skrue 1
    assertEquals(description, screwOne.getDescription()); // Skrue 1
    assertEquals(screwName, screwTwo.getName()); // Skrue 2
    assertEquals(expectedAmountTwo, screwTwo.getAmount()); // Skrue 2
    assertEquals(expectedScrewId, screwTwo.getScrewId()); // Skrue 2
    assertEquals(description, screwTwo.getDescription()); // Skrue 2
}
```

UnitTest

```
src/test/java/app/persistence/mappers/rafterCalculator.java
```

```
@Test
void rafterBeamAmountCalculator() {
    // Arrange:
    int carportWidthInCm = 700;
    int expected = 3;
    int carportWidthInCm2 = 600;
    int expected2 = 2;

    // Act:
    int result = calculator.rafterBeamAmountCalculator(carportWidthInCm);
    int result2 =
calculator.rafterBeamAmountCalculator(carportWidthInCm2);

    // Assert:
    assertEquals(expected, result, "Der bør være 3 remme.");
    assertEquals(expected2, result2, "Der bør være 2 remme.");
}
```

Vi har derudover lavet manuelle acceptance test som sikrer at systemet, fra brugerens synspunkt, kører som det skal, og at User Stories overholdes. Dette har hjulpet os med at finde fejl i vores kode, og se hvordan vores metoder taler sammen, hvilket vi efterfølgende har kunnet rette. Derudover har vi også gjort det klart, at vi ikke bare har skitseret et produkt, men lavet dét, kunden har efterspurgt.

Denne testtype har ledt til flere korrespondancer om hvorvidt projektet manglede noget for at opnå alle User Stories (Forventningskravet fra kunden.

Test-klasserne er godt organiseret og klart adskilt mellem forskellige testtyper. Der er dermed ikke nogle blandinger af integrationstest og unittest, hvilket gør at vores `@Beforeeach` kun køres om nødvendigt. Den organiserede struktur gør det også nemt at navigere rundt i testene. Vi har en coverage på mindst 85% af linjerne i vores unit tests og en lidt lavere, men rimelig coverage på vores integration tests. Dog har vi ikke

tests til alle udfald af metoderne, hvilket er en fejl, der vil kræve større fokus, næste gang et projekt som dette laves.

Som sikkerhed har vi opsat automatisk testkørsel på alle pull requests i GitHub. Disse skal bestå før man kan merge koden ind, hvilket sikrer en kvalitet og stabilitet gennem udviklingen og nedsætter drastisk chancen for store fejl i produktionskoden ved merge. Samlet set giver denne teststruktur stor sikkerhed i systemets kernefunktioner. Teststrukturen sikrer, at metoder virker, som de skal, og at vores system snakker korrekt sammen med databasen. Dog skal der nok være et større fokus på at automatisere acceptance test og udvide testscenarier for at øge sikkerheden og stabiliteten af testene samt gøre systemet mere skalerbar i fremtidige projekter.

Deployment

Digital Ocean

Vi har i dette projekt anvendt droplet-servere i Digital Ocean til at deploye vores program. Ved at oprette en virtuel maskine (droplet) har vi haft kontrol over server miljøet. Vi har containeriseret applikationen med *Docker* og styrer dermed hele miljøet med *docker-compose*.

Vores Java-applikation er bygget som en "eksekverbar" JAR-fil. Vi laver et Carport directory i `jetty@.../deployment`. Så laver vi en kopi af vores JAR-fil og placere den ind i `jetty@.../deployment/Carport`. Så laver vi en Dockerfil i Carport directory som vi fylder med:

```
FROM amazoncorretto:17-alpine COPY ./app.jar /app.jar EXPOSE 7070  
CMD ["java", "-jar", "/app.jar"]
```

Denne Dockerfile instruerer Docker i at bruge en letvægts-Java 17-base, kopiere JAR-filen ind i containeren, og starter applikationen med at køre JAR-filen. Så laver vi det Docker-container(Docker-Compose.yml), hvilket gør det nemt at starte flere services på samme tid, og det gør vi på følgende måde:

```
version: '3.9'  
  
services:  
  
  caddy:  
    image: caddy:2.7.6  
    container_name: caddy  
    restart: unless-stopped  
    ports:  
      - "80:80"  
      - "443:443"  
    volumes:  
      - ./Caddyfile:/etc/caddy/Caddyfile  
      - ./site:/srv  
      - ./caddy_data:/data  
      - ./caddy_config:/config
```

```
carport:
  build:
    context: ./carport
    dockerfile: Dockerfile
  container_name: carport
  restart: unless-stopped
  ports:
    - "7070:7070"
  environment:
    - DEPLOYED=PROD
    - JDBC_USER=postgres
    - JDBC_PASSWORD=<din-cloud-db-password>
    - JDBC_CONNECTION_STRING=jdbc:postgresql://<din-cloud-db-host>:5432
      /Fog_Carport?currentSchema=public
    - SENDGRID_API_KEY=<din-sendgrid-nøgle>
  networks:
    - default

networks:
  default:
    name: deployment_default
    driver: bridge

volumes:
  caddy_data:
  caddy_config:
```

Caddy server

Vi opsætter en Caddyfile, som bliver læst af vores docker-compose og konfigurerer webserveren til at håndtere både *reverse proxy*, og statisk hosting af projektets hjemmeside. Reverse proxy betyder, at Caddy fungerer som en mellemmand, der modtager brugerens forespørgsel og sender den videre til backend-applikationen. Dette betyder at brugeren kun kommunikerer med Caddy, der skjuler og beskytter selve applikationen (vores kode), samtidig med at den håndterer HTTPS-forbindelser og kan fordele trafik effektivt.

Eksempel på Caddyfile:


```
{
  email cph-jo@cphbusiness.dk
}

carport.outzenprogramming.dk {
  reverse_proxy carport:7070
}

outzenprogramming.dk {
  root * /srv/
  file_server
}
```

Dét der sker her er, at al trafik til `carport.outzenprogramming.dk` sendes videre til en backend-service med navnet `carport`, som kører på port `7070`. Det betyder, at den sender alle forespørgsler videre til vores Java-kode, som kører i Docker-containeren med navnet “`carport`”.

DNS

Vi anvender DNS til at knytte domænenavnet eksempelvis “`marcuspf.com`” til den tilsvarende IP-adresse på vores Caddy-server. Når en bruger indtaster domænet i sin browser, slår DNS’en op i zonefilen og returnerer den korrekte IP, hvor applikationen hostes.

A	marcuspf.com	134.122.93.103	600
A	carport.marcuspf.com	134.122.93.103	600

Udviklingsproces

Repositorystruktur og Arbejdsregler

Det var vigtigt for os, at opsætningen af repositoryet var struktureret korrekt helt fra starten. Ud over det basale som en README og en wiki-side, sørgede vi også for at udarbejde en præcis .gitignore-fil. Dette sikrede at unødvendige filer og midlertidige data ikke blev uploadet til repositoryet og dermed holdt koden ren og overskuelig. Vi valgte at ignorere ting som .class filer, logfiler, .jar filer og mapper fra vores udviklingsmiljø som f.eks. .idea/. Vi fjernede også systemfiler fra mac, som .DS_Store. For sikkerhedsmæssige årsager, lavede vi nogle branch protection rules som lød således

1. Ikke muligt at push til main branch
2. Kræve pull request før merge
3. Mindst en reviewer approval før pull request kan merges
4. Kræv at alle kommentarer er håndteret, før man kan merge

CI/CD workflows (inkl. automatiserede integrationstest og JUNIT)

Endnu en sikkerhedsmæssig protocol vi sørgede for, var at tilføje et Java CI/CD-pipelinen som lavede automatiserede integration- og JUNIT test. Disse test blev kørt automatisk for hvert eneste pull request og merge man lavede, og blev det ikke godkendt, så kom der et rødt kryds som indikator.



Bilag 8: Eksempel på CI/CD workflowet

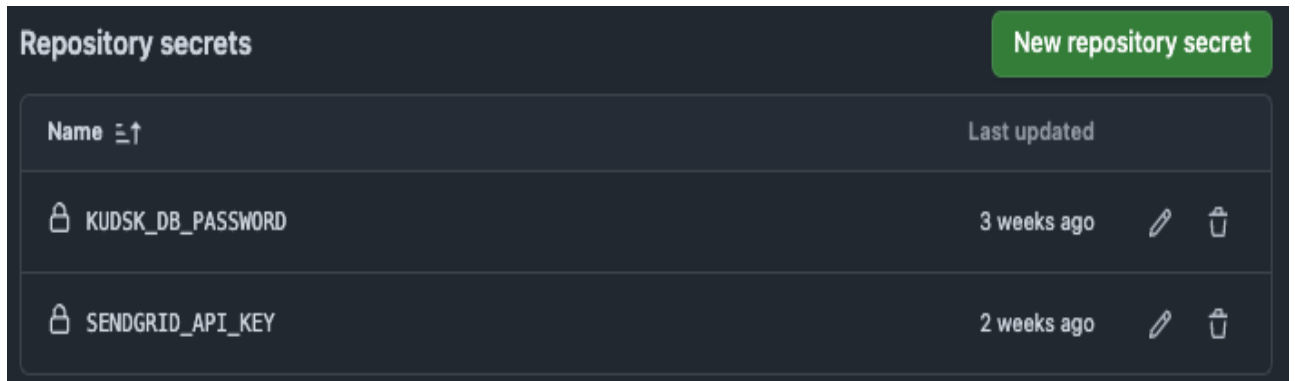
Disse automatiserede test krævede at vi tilføjede en mappe ude i root med navn [.github/workflows](#) og derefter køre et Github Actions workflow script







`ci.yml`

Da workflowet var sat op, var vi nødt til at tilføje nogle *repository secrets*, fordi testene krævede adgang til vores hemmelige miljøvariabler for at kunne køre korrekt.

```
jobs:
  build:
    runs-on: ubuntu-latest
    env:
      kudsk_db_password: ${ secrets.KUDSK_DB_PASSWORD }
```

Secrets and variables -> Actions -> Repository secrets



Repository secrets		New repository secret	
Name ↕	Last updated		
 KUDSK_DB_PASSWORD	3 weeks ago		
 SENDGRID_API_KEY	2 weeks ago		

Bilag 9: System-enviornment-variables

Pull request og code reviews

Som der blev nævnt før, så krævede et merge et godkendt pull request. Derfor lagde vi meget vægt på at lave Pull request og code reviews ordentlig, for vi hjalp hinanden med at overholde de kodestandarder der var blevet lagt. Ved at vi var så grundige med vores code reviews, gjorde det hele hang sammen til sidst strukturmæssigt og kodemæssigt.

Brug af Github Projects (Brug af kanban koncept)

Vi anvendte Github Projects til alt igennem forløbet og sørgede for at lave branches igennem vores issue-kort derinde fra, så hver branch fik et unikt id som var nemt at holde styr på. Github Projekt tager brug af kanban konceptet. Kanban er et visuelt planlægningsværktøj, der bruges til at organisere og styre opgaver i et projekt. Vi havde fem kategorier

1. Backlog

Her placerer vi alle opgaver, som endnu ikke har fået en planlæggelses dato. Dette kan være nye idéer eller funktioner til projektet. De kort her er typisk ikke færdigskrevet eller klar til at blive flyttet til *Ready*

2. Ready

Når en opgave i *backlog* er færdigskrevet, bliver den flyttet til *Ready*. Det betyder, at opgaven er tydeligt beskrevet, og at alle forstår, hvad der skal laves. Det er herfra teammedlemmerne vælger opgaver, de vil arbejde på.

3. In progress

I den her kolonne opstår de kort som teammedlemmerne aktivt har valgt at begynde udviklingen på. Med at kortene kommer i sådan en kolonne her, hjælper det med at holde overblik.

4. In review

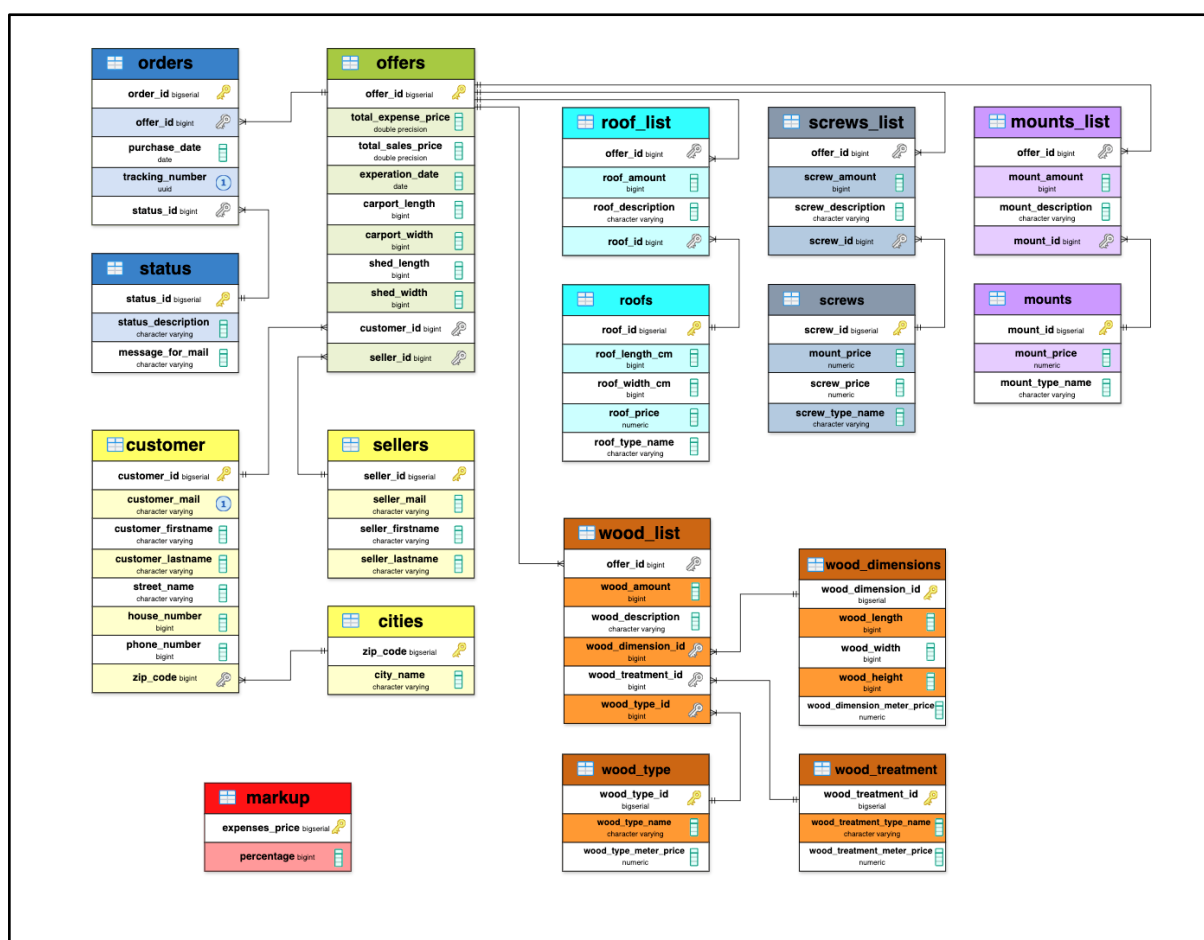
Når en opgave er færdigudviklet og der laves et pull request, kommer den automatisk i *In review*. Her gennemgår et andet teammedlem koden i et code review, for at sikre koden følger vores kodestandarder samt finde eventuelle fejl eller forbedringer, inden det bliver merget i produktionsbranchen.

5. Done

Når pull request er godkendt, og merged i produktionsbranchen, bliver kortet automatisk flyttet til *Done*. Dette viser at opgaven er fuldt afsluttet.

Databasedesign og ERD

Vi har designet en relation-baseret database med henblik på nøjagtig og nem håndtering af kunder, ordrer, tilbud, træ, skruer, beslag og tag. Databaseen er bygget via relationer, ved at data er opdelt i unikke/respektive entiteter, som repræsenterer sin egen tabel. En primær del af opbygningen af databaseen har været ved brugen af *Primary-Keys* og *Foreign-Keys* for en sikker sammenhæng mellem tabellerne.



Figur 8: ERD til databaseen

Datatyper er valgt med fokus på præcisionen og funktionaliteten af dataen. Vi anvender *bigserial* til at genere ID'er automatisk, hvilket sikrer, at hver datarække får en unik *primary-key* uden manuel håndtering. Vi har gjort brug af *numeric* til priser, for at få en præcis lagring af priser uden afrundingsfejl. Vi har brugt *UUID* til trackingnummre for en øget sikkerhed og global unik identifikation. Bigserial er med tilbageblik ikke det bedste valg til offer_id, da det kan betyde, at kunder potentielt kan

gætte den næste persons tilbudsnummer, idet det er serials. Med dette i tankerne ville det nok have været en bedre ide at bruge UUID til offer_id, da man kan sætte constraints med "random autogen", hvilket ville gøre det næsten umuligt at gætte andres tilbudsnummer.

Relationerne mellem tabellerne er bygget med foreign-keys, hvilket gør, at rækker peger på allerede eksisterende rækker i andre tabeller. Man kan fx ikke oprette en ordre med et offer_id uden, at der allerede er et eksisterende offer_id i databasen. Dette sikrer, at vi ikke har non-refererbar data i nogle af tabellerne. Vi har tildelt en *CASCADE Action On Delete*, til alle foreign-key constraints for offer_id, hvilket sørger for, at hvis et offer slettes i offers tabellen, så slettes alle tilhørende rækker i alle andre tabeller også. Dette sørger for, at vi ikke har løsthængende data, som bare fylder. Vi har også "mange til mange" relationer, som bruges ved skruer, træ, beslag mm. Disse refererer til liste-tabeller, hvor hver række indeholder et offer_id, et eller flere materialer-id'er, og en mængde. Dette skaber mulighed for at lave lister af objekter til hvert offer_id, og dermed lave unikke kombinationer. Vi har valgt at dele træ-objektet op i tre dele, da prisen stiger ud fra træets "behandling". Dette betyder, at vi kun skal rette priser enkelte steder og ikke på hvert stykke træ med den behandling. Dette gør det mere brugbart for produktion og prisberegning.

Med et tilbageblik ville det nok have været smartere at skabe en relation fra tabellen *wood_type* til tabellen *wood_dimension*, og lave *wood_length* om til *wood_length_id* (foreign-key) i *wood_dimension*, og lave en ny tabel(*wood_lengths*), hvor kolonnen har *wood_length_id* som primary-key og bigserial, og en kolonne med *wood_length_in_cm*. Dette ville gøre det muligt, at redigere en længde på træ hurtigere, hvis man nu fx ikke længere har 240cm, men nu 250cm.

Med henblik på normalisering har vi arbejdet efter mindst tredje normalform. Hvilket betyder, at alle tabeller indeholder autogenererede unikke værdier(1st. Normalform). At non-key kolonner er fuldstændig afhængige af primary-key'en(2nd. Normalform). At non-primary-key kolonner ikke afhænger af andre non-primary-key kolonner for at undgå unødvendig dublerede data(3rd. Normalform). Fordelen ved denne normalisering er, at hver oplysning kun findes et sted, som

mindsker chancen for ikke konsistent data, og gør det nemmere at opdatere data uden fejl. Da der undgås duplikering, sparer vi plads, hvilket øger performance, dog kun betydeligt hvis datamængden bliver stor nok. Det hjælper også på vedligeholdelse, da data kun skal opdateres/rettes et sted i tilfælde af ændringer. Systemet fungerer godt til, at det nemt kan udvides med nye typer af materialer mm. Uden at skulle ændre på den nuværende struktur.

Ulemper ved denne normalisering er, at der skal bruges lange og komplekse sql-statements, med mange joins, for at hente komplet data, hvilket kan påvirke ydeevnen negativt, især når databasen vokser. Vi har valgt denne struktur idet, vi vurderer denne til at være en tilstrækkelig databasestruktur med henblikket, at det skal være let at vedligeholde. Dertil regnes der ikke med at være flere tusinder af ordre. Samtidig har vi gjort klar til at auto slette alle ikke købte tilbud efter syv dage. Det gør, at databasen aldrig vokser sig for stor i forhold til performance.

Konklusion og refleksion

Hvad gik godt

Der var flere dele i projektet, som forløb effektivt og efter planen. Et af de elementer som fungerede bedst i teamet, var brug af Kanban i Github Projects. Dette sørgede for at vi fordelte arbejdet så ligeligt ud som overhovedet muligt, og gav et stærkt overblik over de forskellige udviklingsopgaver, der skulle laves. Det medførte, at vi i teamet konstant var opdateret på hinandens fremskridt og opgaver.

Efter udarbejdelsen af vores risikoanalyse havde vi som team fastlagt en plan for, hvilke handlinger der var acceptable, og hvilke der ikke var. Dette bidrog til, at vi havde stor respekt for hinandens ønsker og tid. Hvis der opstod udfordringer i teamet, var vi gode til at kommunikere dem, hvilket gjorde det muligt at håndtere potentielle forsinkelser og risici i tide.

Vi har valgt at holde et stand-up møde hver dag, hvor vi snakkede om, hvad der skulle laves frem mod næste gang. Ved at sætte interne deadlines i teamet, sikrede vi fremdrift i projektet, og alle vidste hele tiden, hvad de skulle arbejde på.

Et andet element som fungerede rigtig godt var vores tekniske samarbejde. Vi lagde en fælles kodestandard fra start, som alle i teamet holdt sig til gennem hele projektet. Pull requests var en stor hjælp, fordi vi kunne give hinanden feedback og sikre, at alle fulgte standarden. Dette gjorde, at det eneste som kom ind i "main" produktionskoden, var kode som følger standarden og var nemt at læse.

Hvad kunne forbedres

Der er plads til at bruge nogle bedre datatyper i databasen, idet den meget objektorienterede database struktur, vi har lavet, er let at vedligeholde, men halter performancemæssigt, når der begynder at være større mængder data. Vi skal skabe en bedre kundeoplevelse, såsom tekst der forklarer, at der sker noget, når kunden trykker på knapper som "byg tilbud". Her ville det være passende med en besked, der beskriver at kundes tilbud faktisk udregnes.

Vi kan blive bedre til at forklare, hvordan vi tænker koden skal fungere helt konkret, eksempelvis hvilken type af objekter, der bygges i systemet, samt hvordan backenden sender og håndterer disse.

Fremover bør vi lave automatiseret acceptance test på vores frontend. Dette vil sørge for, at der automatisk testes for uventede funktionelle ændringer/fejl. Eksempelvis ved tryk af knapper, der kalder metoder i Routing Controller. Det har vi i dette projekt gjort manuelt(monkey-test). Hvilket hverken er en særlig tidseffektiv eller pålidelig test, idet der nemt kan ske menneskelige fejl.

Læring

Gennem arbejdet med dette projekt har vi opnået værdifuld erfaring med samarbejde og projektorganisering i en gruppe. Vi har lært, hvordan man arbejder effektivt sammen mod et fælles mål, og vi har erfaret os med bedre kommunikation, koordinering samt respekt for hinandens ideer. Vi har arbejdet med at fordele ansvar og opgaver mellem hinanden indbyrdes, specielt ved brug af KanBan i GitHub projects. Vi har lært selvstændigt at tage fat på opgaver og tage ansvar for egen indsats, idet projektet er for stort til, at vi som team ikke kan kontrollere eller overvåge hinandens arbejde. Vi har lært at bryde projektet ned i mindre og overskuelige dele, hvilket har gjort det lettere at tage nye opgaver op. Disse erfaringer har styrket vores evne til at arbejde i grupper og har givet os indsigt i, hvordan man som team gennemfører et projekt effektivt.

Konklusion

Afslutningsvis kan vi konkludere, at vores arbejde med hjemmesiden til Johannes Fog's carport-løsning har været struktureret, med fokus på den oplevelse, kunden møder, såvel som de interne processer, der sker bag om carport-salget. Vi startede med en grundig forretnings- og interessentanalyse, der hjalp os til at forstå alle aktørers behov. Dette gav et godt udgangspunkt, til at tegne det nuværende AS-IS- og ønskede TO-BE-handlingsforløb, samt at skabe en domæne- og klassesdiagramstruktur, der sikrer, at systemets arkitektur hænger sammen og er let at vedligeholde.

I udviklingsfasen benyttede vi Java med Javalin og Thymeleaf, Docker og PostgreSQL til at skabe et skalerbart backend-miljø, mens vi med Scaling Vector Graphics og Figma-skitser sikrede en visuelt overskuelig og responsiv frontend. Integration med SendGrid gjorde det muligt automatisk at sende tilbud og styklister til kunden via mail, så al kommunikation foregår fejlfrit. Kvaliteten er sikret gennem en teststrategi med både unit-, acceptance- og integrationstest, som kører automatisk inden GitHub-merge. Løbende pull-request reviews, daglige stand-ups og opgavefordeling via Kanban i GitHub Projects har samtidig sikret god og hurtig håndtering af udfordringer. Når vi ser fremad, vil vi indføre automatiserede frontend-acceptance-tests og overveje endnu stærkere databasestruktur, eksempelvis med konsekvent brug af UUID'er til tilbudsnumre. Vi vil desuden styrke kundeoplevelsen ved at gøre det klart synligt, når beregninger foretages.

Vi har leveret et brugervenligt og kriterieopfyldende løsning, der matcher kundens ønsker og Johannes Fog's interne idéer. Sidst er det værd at konkludere; at erfaringerne fra projektet styrker vores evner til at arbejde effektivt i teams, arbejde med forretningslogik og levere en løsning, der opfylder virksomhedens behov, men samtidig kan videreudvikles.

Video og Links

Youtube link:

<https://youtu.be/3M9TCkKiWSs>

GitHub repository link:

<https://github.com/MarcusPff/Carport-Fog>

Links til deployment:

<https://carport.outzenprogramming.dk/>

<https://carport.kudskprogramming.dk/>

<https://carport.marcuspff.com/>

Bilag

30 dages returret
Klik og hent inden for 3 timer*
1-3 dages levering*
Lån en trailer gratis

!Fog
Bolig & design
Byggematerialer
El & belysning
Have & fritid
Værktøj
Mærker
Tilbud
Fog Pro
Søgning
Fog Konto
Find Fog
Kurv

Byg selv-produkter / Carporte / Quick-Byg carporte / Quick-Byg carport med fladt tag

Din carport
• Kontaktoplysninger
Bekræftelse

Kontaktoplysninger

Navn	Efternavn
<input type="text"/>	<input type="text"/>
Adresse	Postnummer
<input type="text"/>	<input type="text"/>
Postnummer	By
<input type="text"/>	<input type="text"/>
Telefon	Email
<input type="text"/>	<input type="text"/>

☐ Kontakt samtykke
Fog må benytte de afgivne oplysninger til at kontakte mig i forbindelse med tilbud på QuickByg carpor

Next →

30 dages returret
Klik og hent inden for 3 timer*
1-3 dages levering*
Lån en trailer gratis

!Fog
Bolig & design
Byggematerialer
El & belysning
Have & fritid
Værktøj
Mærker
Tilbud
Fog Pro
Søgning
Fog Konto
Find Fog
Kurv

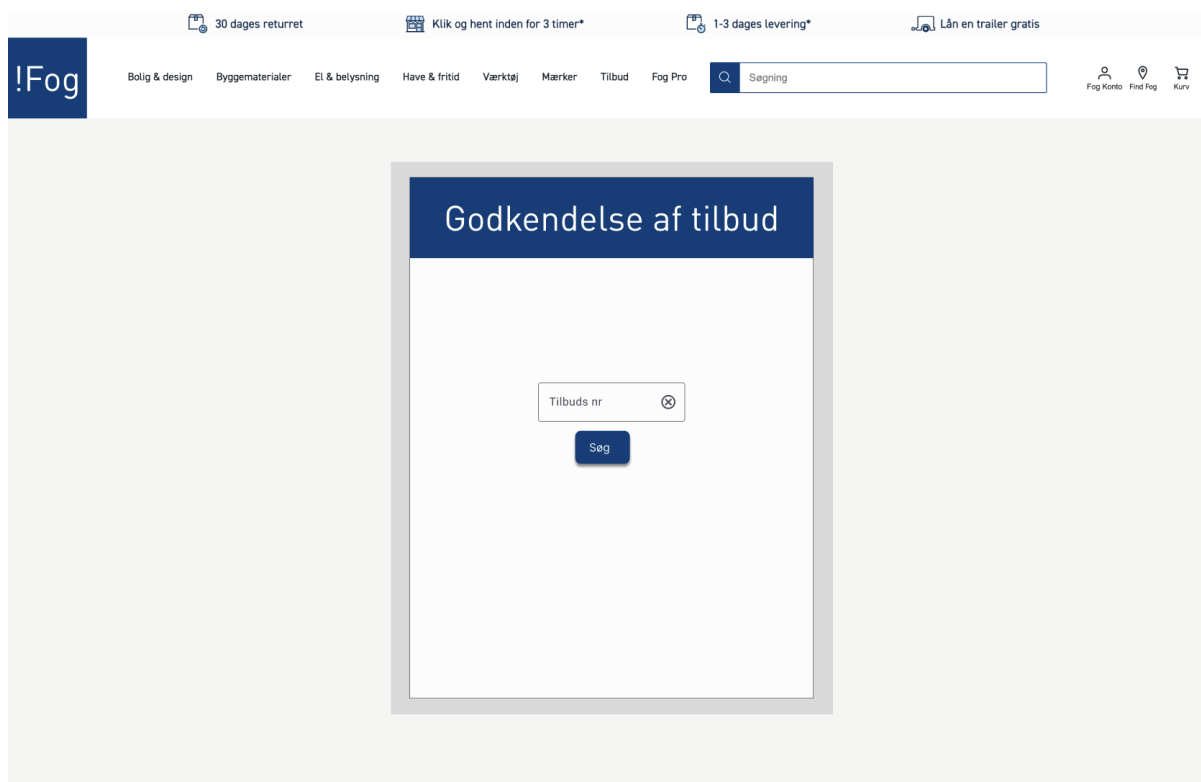
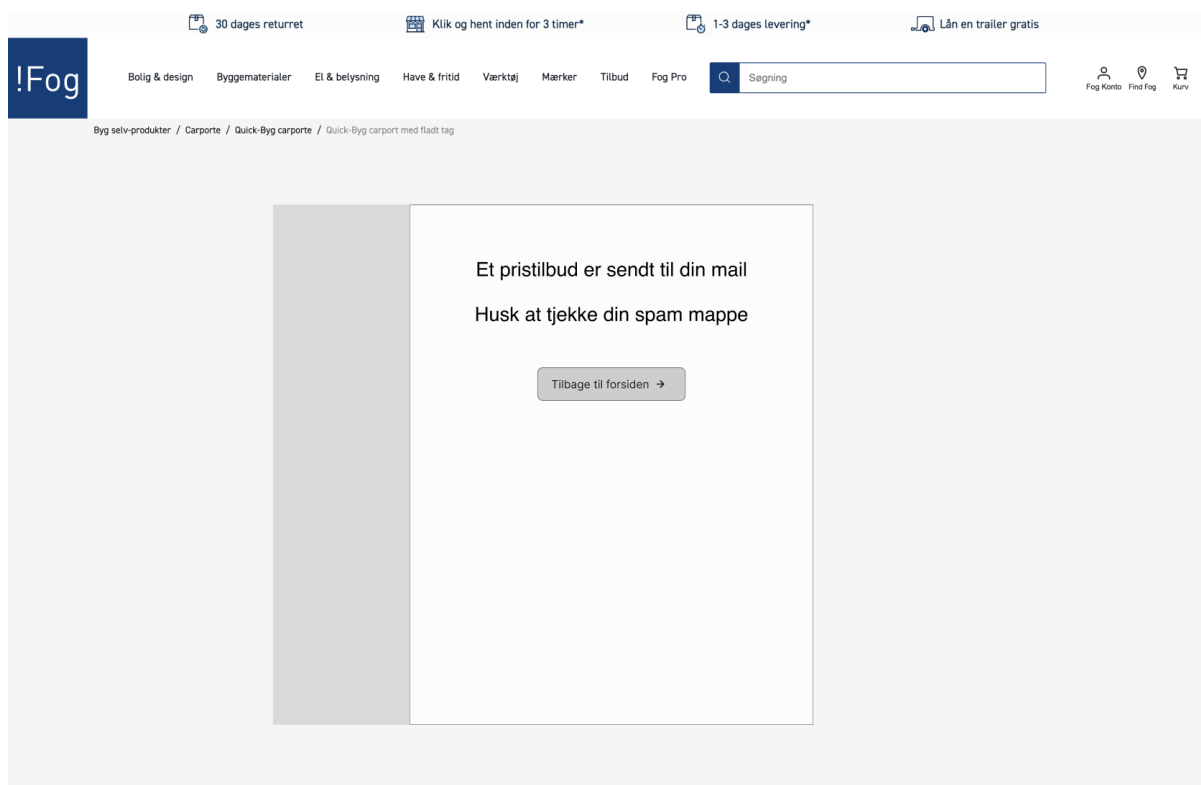
Byg selv-produkter / Carporte / Quick-Byg carporte / Quick-Byg carport med fladt tag

Din carport
Kontaktoplysninger
• Bekræftelse

Bekræft information

Bredde:
Længde:
Trapeztag:
Redskabsrum: Ja/Nej
Redskabsrum Bredde:
Redskabsrum Længde:
Evt. bemærkninger:

Bestil tilbud



30 dages returret

Klik og hent inden for 3 timer*

1-3 dages levering*

Lån en trailer gratis

!Fog

Bolig & designByggematerialerEl & belysningHave & fritidVærktøjMærkerTilbudFog Pro

Søgning

Fog KontoFind FogKurv

Bliv kontaktet af sælger

Tilbuds nr

Bekræft

30 dages returret

Klik og hent inden for 3 timer*

1-3 dages levering*

Lån en trailer gratis

!Fog

Bolig & designByggematerialerEl & belysningHave & fritidVærktøjMærkerTilbudFog Pro

Søgning

Fog KontoFind FogKurv

Godkendelse af tilbud

Ordrenummer: XXX
Email: cph-mf411@cphbusiness.dk

Din pris: 17.643,65 DKK

Nej tak

Bekræft

Du har bekræftet tilbuddet på 17.643,65 DKK.
Tjek venligst din mail

The screenshot displays the !Fog website's header and a central login form. The header includes the !Fog logo on the left and navigation links for 'Bolig & design', 'Byggematerialer', 'El & belysning', 'Have & fritid', 'Værktøj', 'Mærker', 'Tilbud', and 'Fog Pro'. On the right, there are icons for 'Fog Konto', 'Find Fog', and 'Kurv'. Above the navigation links, three promotional banners are visible: '30 dages returret', 'Klik og hent inden for 3 timer*', and '1-3 dages levering*'. A search bar with the text 'Søgning' is also present. The main content area features a dark blue header with the text 'Sælger administration'. Below this, there is a white box containing a text input field labeled 'Sælger kode' with a clear button (X) on the right. Below the input field is a dark blue 'Login' button.

The screenshot displays the !Fog website's 'Sælger administration' (Seller Administration) interface. At the top, there is a navigation bar with the !Fog logo and several menu items: 'Bolig & design', 'Byggematerialer', 'El & belysning', 'Have & fritid', 'Værktøj', 'Mærker', 'Tilbud', and 'Fog Pro'. A search bar is located on the right side of the navigation bar. Below the navigation bar, the main content area features a dark blue header with the text 'Sælger administration'. Underneath the header, there is a search bar with the placeholder text 'Søg efter tilbuds ID' and a 'Søg' button. The central part of the interface contains a table with five columns: 'ID', 'materiale', 'dimensioner', 'Indkøbspris', and 'Antal'. To the right of the table, there are several buttons: 'Opdater struktur', 'Indkøbspris', 'SalgsPris', 'Rediger tilbudspriser', 'Send endelige tilbud', and 'Slet tilbud'.

