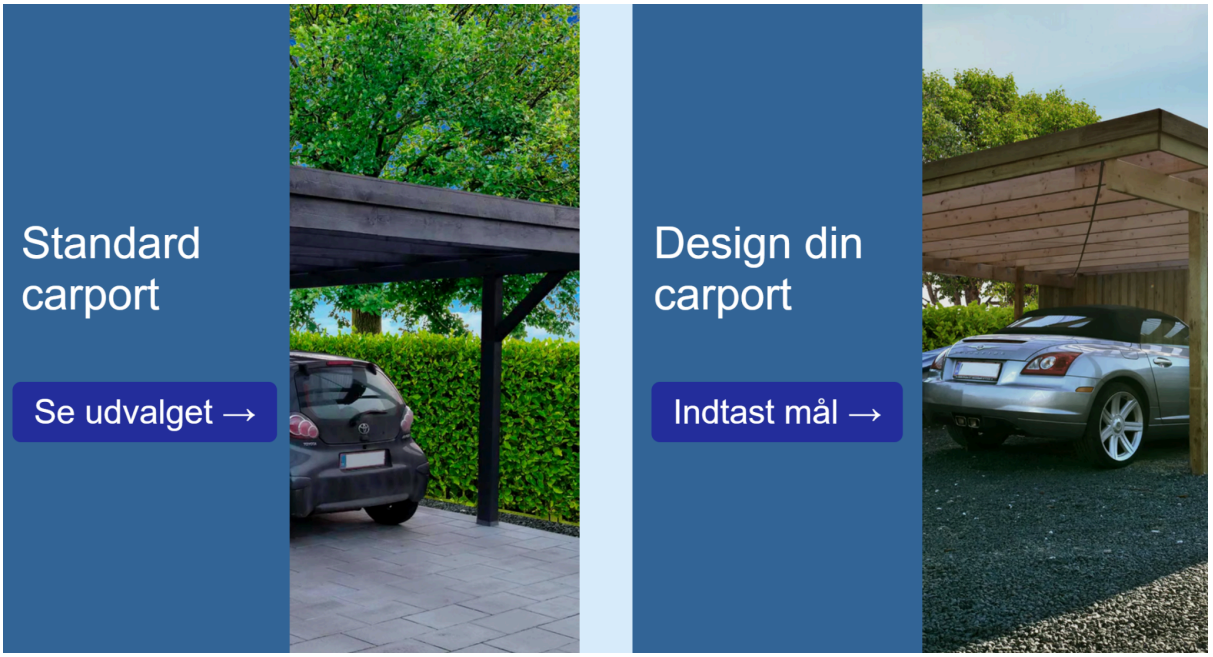
Fog Carports



Datamatiker 2. semester F24 - Hold B
24/05 - 2024

Orn-Iliya Ananta Phak Petersen: cph-op91@cphbusiness.dk (Github: *Liya-ap*)
Ömer Ümit Øcalan: cph-oo203@cphbusiness.dk (Github: *qmer05*)

Rebecca Mary Buron Sørensen: cph-rs@cphbusiness.dk (Github: *rebeccaburon*)
Teodora Milijic: cph-tm291@cphbusiness.dk (Github: *TeodoraM24*)

Tobias Thormod Birk Nielsen: cph-tn293@cphbusiness.dk (Github: Tobitastik)

Link til demovideo:

(Demo video)

Link til hjemmeside på Digital Ocean:

http://161.35.195.156:7070/

Admin bruger til hjemmeside:

E-mail: *admin@admin.dk*

Kode: *admin*

Indholdsfortegnelse

1. Indledning

Vi lever i en verden omgivet af teknologi; den er en integreret del af vores dagligdag. Full-stack udviklere spiller en stor rolle i at gøre vores liv lettere ved at skabe de apps og hjemmesider, vi bruger dagligt.

I dette projekt har vi samarbejdet med virksomheden Johannes Fog A/S . Vi har udviklet et full-stack system, hvor kunden kan designe egen carport, se sine forespørgsler, tidligere ordre og hertil lave en tilhørende administrator side, som kan håndtere og justere kundens forespørgsel og ordre. Denne rapport vil fokusere på vores analyse, design og implementering af de teknologiske valg, der er blevet foretaget. Projektet vil primært fokusere på vores arbejdsproces og de beslutninger, vi traf undervejs.

2. Baggrund

Fog tilbyder et bredt udvalg af materialer og produkter til at bygge- og renoveringsprojekter. Deres Trælast & Byggecentre på Sjælland har specialiserede afdelinger, herunder salg af jern, stål og andre materialer i Vordingborg samt bolig og design i Kongens Lyngby. De sælger og leverer alt fra byggematerialer og træ til maling, badeværelser, el, værktøj og haveredskaber. Fog står klar til at hjælpe kunder med at vælge de rette løsninger til deres behov. Udover det, er Fog specialiseret i design af carporte. (Johannes Fog A/S, 2024)

Krav

Som et krav for kundens side, ønskede Fog at skabe en mulighed for brugerne til at oprette en profil på hjemmesiden, hvor de senere kunne logge ind og oprette en forespørgsel til en carport. Kunden kan vælge både højde og bredde, samt indtaste mål og materiale for taget og redskabsrummet, hvis ønsket.

Kravene for den administrative side er, at de ønskede at skabe en mulighed for at tilføje nye materialer til databasen og ændre prisen samt beskrivelsen på materialer, der allerede eksisterer i databasen. Derudover skulle administratoren kunne få en liste over materialer og beregne prisen, som er blevet udregnet ud fra en kundes forespørgsel, ved at liste alle

materialer, der kræves for at bygge carporten samt indkøbsprisen, salgsprisen, med og uden moms og dækningsgraden i procent.

3. Forretningsforståelse

Interessentanalyse

Vi har valgt at udarbejde en interessentanalyse, som har til formål at sikre, at projektgruppen og lederen er opmærksomme på, hvilke personer (både inden for og uden for en organisation) der har interesse i projektet. Derfor er det vigtigt at identificere, hvilke personer der vil have kontakt med projektet eller blive påvirket af det både under projektets gennemførelse og efter, når projektets resultater skal implementeres. (Manaz, 2024)

Projekt *Carport hjemmeside for Johannes Fog A/S*			**Udfyldt af** *Orn-Iliya A. Petersen, Ömer Ümit Øcala og Rebecca Mary Buron Sørensen, Teodora Milijic, Tobias Nielsen*	**Dato** *23/04/2024*
Interessent	Interessenten kan opleve følgende FORDELE ved projektet	Interessenten kan opleve følgende ULEMPER ved projektet	Samlet vurdering af interessentens bidrag/position	Håndtering af interessenten
Martin	Afgørende indflydelse på hjemmesidens funktionalitet og (muligvis) design. En veldesignet hjemmeside gør det lettere at oplære medarbejdere, der skal navigere rundt i systemet.	Skal tilvænne sig et nyt system/hjemme side. Hvis det ikke er veldesignet, bliver oplæring af medarbejdere mere besværligt og tidskrævende.	Stor magt og indflydelse. Aktiv deltagelse også er nødvendigt, da hjemmesiden bliver designet til Johannes Fog A/S, og Martin repræsenterer dem. Dermed er deres accept af funktionaliteter og design en nødvendighed for fuldførelse af projektet.	Det kan håndteres gennem møder mindst én gang om ugen for at få afklaret nødvendige krav eller andre spørgsmål til systemet.

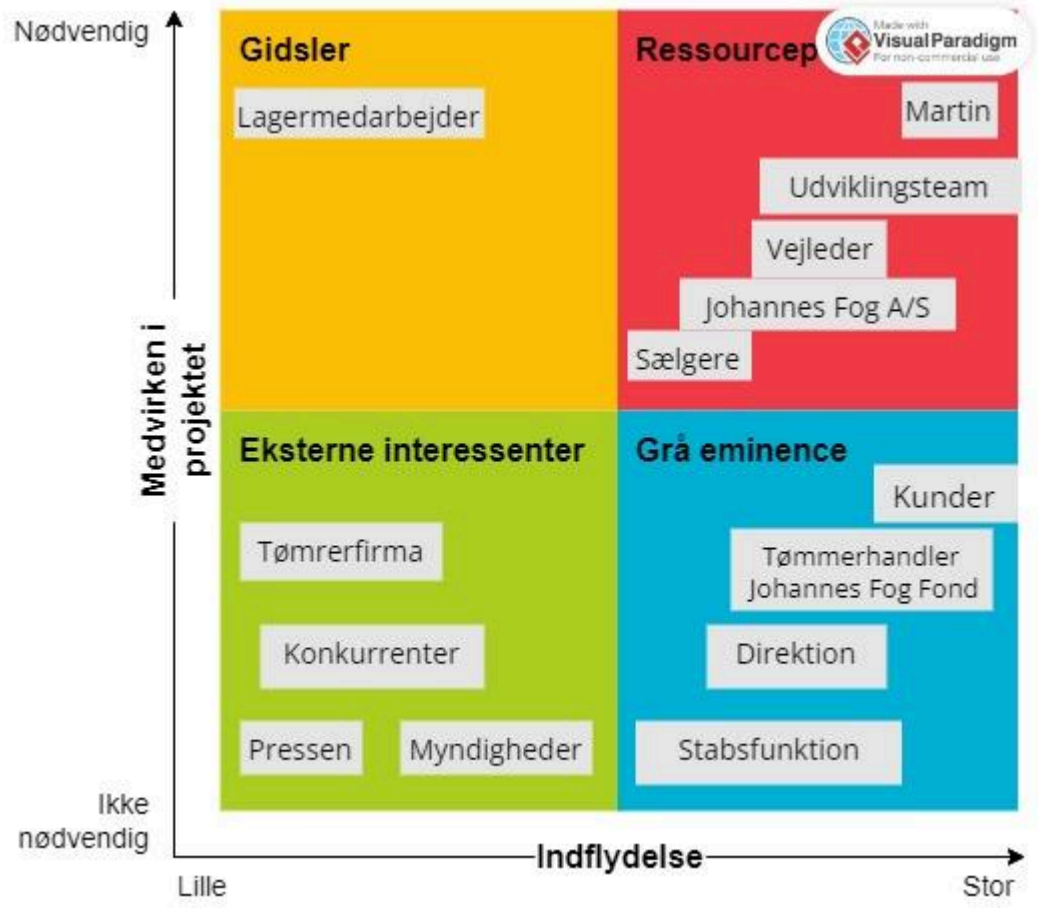
Johannes Fog A/S	En veldesignet hjemmeside kan også betyde flere kunder, der køber carporte, da kunderne ikke vil have svært ved at navigere rundt og foretage et køb. Det kan også afspejle sig positivt på virksomhedens image og omdømme.	Der kan muligvis være tab af nogle kunder, hvis hjemmesiden ikke er veldesignet, og kunderne føler, at det er svært at foretage et køb.	Stor magt og indflydelse. Aktiv medvirken er nødvendig, da hjemmesiden bliver designet til firmaet. Deres accept er nødvendig og afgørende for fuldførelsen af projektet.	Kan håndteres gennem ugentlige møder med Martin, der repræsenterer firmaet.
Kunder	En veldesignet hjemmeside og intuitiv brugergrænseflade gør det nemmere for kunderne at navigere i. Det forbedrer deres overordnede oplevelse, hvilket gør det mere sandsynligt for dem at foretage et køb af en carport.	Foretrækker den gamle måde at købe carport på frem for den nye. En stamkunde, der har vænnet sig til den gamle måde, kan have svært ved at tilvænne sig det nye.	Lille magt, men stor indflydelse, da kunderne skal kunne finde rundt på hjemmesiden og bestille carporte, og hvis designet ikke er brugervenligt, så betyder det frustrerede kunder og færre bestillinger.	Kan håndteres gennem usability-testing ved behov, så vi har mulighed for at ændre på designet, hvis nødvendigt.
Sælger	Et velfungerende system kan gøre det nemmere for sælgere at navigere i og anvende færre trin i forbindelse med beregning og salg af carporte. F.eks. kan det frigøre ressourcer (sælgers tid), hvis kunderne direkte kan få beregningen af mål, pris mm. i stedet for at sælger manuelt skal gøre det. Mere automatisering vil desuden også mindske arbejdspresset på sælgerne.	Skal tilvænne sig et nyt system/hjemmeside. Stor utilfredshed med systemet kan give modstand mod anvendelsen af det. Det kan medføre dårlig arbejdsmoral samt ringere effektivitet blandt sælgerne. Det kræver tekniske færdigheder, oplæring osv.	Stor indflydelse da, de skal inddrages i gennem hele projektet indtil det står færdigt. De er nødvendige for at projektet får succes, da de selv er direkte brugere af systemet. Det kræver en stor grad af accept. Derudover skal sælgerne også være fortrolige med hjemmesiden i tilfælde af at kunderne ønsker hjælp når der skal købes en carport.	Kan håndteres via brugerinddragelse herunder brugerinput, feedback og tilfredshedsmåling er via kvalitative og kvantitative undersøgelser. Martin kunne bruges som en form for bindeled/repræsentant for sælgerne og udviklingsteamet.

Tømmerhandler Johannes Fog Fond	Johannes Fog A/S får potentielt flere kunder der bestiller carporte, hvilket øger omsætningen og gør at fonden får flere penge til rådighed til uddeling af legater.	Hvis hjemmesiden ikke er veldesignet, kan det også betyde lavere omsætning end forventet.	Kan have stor indflydelse på projektet, da fonden medvirker til udviklingen af Johannes Fog A/S. Dermed er det vigtigt, at man ikke overser dem i udviklingen af hjemmesiden, når større beslutninger bliver taget.	Kan håndteres gennem rapportering af projektets status gennem møder, brugerinddragelse herunder.
Direktion	En veldesignet hjemmeside kan tiltrække nye kunder, differentiere dem fra konkurrenterne, og drive virksomhedens vækst op.	Hvis hjemmesiden ikke er veldesignet så resulterer det muligvis i tab i omsætningen og tab af kunder, hvis kunderne har svært ved at benytte den nye hjemmeside.	Kan have stor indflydelse i projektet, da hjemmesiden repræsenterer firmaet, men deres aktive medvirken er ikke nødvendig. Dermed er det vigtigt, at man ikke overser dem i udviklingen af hjemmesiden, når større beslutninger bliver taget.	Kan håndteres gennem rapportering af projektets status gennem møder og evt. mails.
Stabsfunktioner	En veldesignet hjemmeside kan positivt påvirke væksten i virksomheden, hvilket gør det nemmere for stabsfunktioner at realisere diverse mål i firmaet.	Øget pres i forhold til muligvis at træne/informere medarbejderne om den nye hjemmeside.	Kan have stor indflydelse, men deres medvirken i projektet er ikke nødvendig. De kan rådgive og vejlede direktionen og firmaet til at tage visse afgørende beslutninger.	Kan håndteres ved at rapportere om projektet og dets status undervejs gennem mail eller møder. Kan interviewes ved behov for at få indblik i mulige udfordringer og risici.
Tømrerfirma	De vil potentielt få flere kunder, da kunderne selv har mulighed for at vælge om de er interesseret i at få hjælp fra en tømrer, når de foretager et køb. Dette er også med til at styrke samarbejdet	Hvis hjemmesiden ikke er veldesignet så kan de miste kunder, hvis det ikke er tydeligt for kunderne at de kan få hjælp til at bygge	Har ingen indflydelse og deres aktive medvirken i projektet er kun lidt nødvendigt, da god kommunikation mellem Fog og tømrerne resulterer i tilfredse kunder.	Kan håndteres gennem mail og man kan overvåge deres holdninger til hvordan hjemmesiden påvirker dem.

	mellem tømrerfirmaet og Fog.	deres carport når de har foretaget et køb.		
Konkurrenter	Hvis hjemmesiden ikke er veldesignet og kunderne har sværere ved at købe carporte hos Johannes Fog A/S, så går de formentlig til konkurrenten i stedet.	En veldesignet hjemmeside kan betyde tab af kunder for konkurrenten da kunderne får en mere positiv brugeroplevelse hos Johannes Fog A/S	Har ingen indflydelse og deres aktive medvirken i projektet er ikke nødvendigt.	Skal ikke informeres men man kan vælge at holde øje med om de får en større stigning i antallet af kunder.
Udviklingsteam	Hvis hjemmesiden er veldesignet, er der blevet lavet et godt produkt til firmaet og det viser at man har formået at analysere samt forstå de kundekrav der er blevet stillet.	Hvis hjemmesiden ikke lever op til kundens forventninger betyder det at der ikke er blevet leveret et tilfredsstillende produkt og at man ikke har formået at forstå de krav der er blevet stillet.	Stor magt og indflydelse og hvis aktive medvirken også er nødvendig, da hjemmesiden bliver lavet af udviklingsteamet, så de tager i sidste ende beslutningen om hvilke funktionaliteter og design implementationer der er mulige at indføre.	Kan håndteres ved at man laver god analysearbejde af kundens krav og ønsker. Muligvis kan man præsentere det for kunden før man rigtigt går i gang.
Vejleder	Muligheden for at bidrage til udviklingen af et praktisk og relevant projekt, der kan give værdifuld erfaring til undervisere/studerende. Mulighed for at vejleder kan følge udviklingsproces samt give feedback undervejs, hvilket giver en værdifuld læringsoplevelse. Mulighed for at se potentiale og konkrete resultater.	Risikoen for at blive overbebyrdet med for mange opgaver eller forespørgsler fra flere studiegrupper eller projekter samtidig. Forringet kvaliteten af vejledning på baggrund af udfordringer med at balancere sin egen tid mellem jeres projekt og andre. Hvis projekt ikke lever op til	Har en væsentlig indflydelse på projektet, da vejledning og feedback kan bidrage til projektets succes og kvalitet. Position som underviser giver et unikt perspektiv og ekspertise, der kan være værdifuldt for studerende i udviklingsprocessen. Vigtigt at anerkende og respektere vejlederens bidrag og position i projektet for at sikre et konstruktivt	Kan håndteres gennem ugentlige vejledermøder for at identificere evt. problemer og løses i tide. Klar og tydelig kommunikation med vejlederen om forventninger, deadlines og eventuelle udfordringer undervejs i projektet. Udnyttelse af vejlederens ekspertise og ressourcer bedst muligt ved at være åbne for feedback

		forventninger eller krav, kan det være skuffende for vejlederen.	samarbejde og en vellykket gennemførsel af projektet.	og konstruktiv kritik samt ved at søge vejledning, når det er nødvendigt.
Pressen	Pressen kan drage fordel af hjemmesiden ved at få en indsigt i virksomhedens udviklingsproces. Dette kan omfatte en historie om virksomhedens innovative løsninger, og hvordan hjemmesiden forbedrer kundeoplevelsen. Disse historier kan tiltrække læsere og forbrugere, hvilket resulterer i øget trafik og engagement på pressens egen hjemmeside.	Der kan forekomme et potentielt pres fra offentligheden, der ønsker at pressen skal dække enhver negativ udvikling eller fejl på hjemmesiden. Dette kan blive på bekostning af andre historier som pressen har mindre tid til at dække.	Har ingen indflydelse og deres aktive medvirken i projektet er ikke nødvendigt.	For at håndtere interessanten kan Martin og hertil virksomheden vedligeholde en åben og transparent kommunikation med pressen under udviklingsprocessen af hjemmesiden. Ved at være åbne omkring processen kan der blive opbygget tillid og troværdighed hos pressen.
Myndighed er	Hvis hjemmesiden er veludviklet, kan den være en kilde til vigtige oplysninger om virksomheden, dens produkter og politikker.	Der kan opstå konflikter mellem virksomhedens og myndighederne s interesser med hjemmesiden. Virksomheden ønsker at optimere for salg og markedsføring, mens myndighederne ønsker at sikre, at forbrugerbeskyt telse standarder opretholdes.	Kan have indflydelse på projektet, men deres aktive medvirker er ikke nødvendigt. Myndighederne kan kræve yderligere ressourcer til at imødekomme deres krav og forventninger.	Identificér potentielle interessekonflikter mellem virksomhedens og myndighedernes interesser tidligt i projektet.

Lagermedarbejde	Deres arbejde ville potentielt blive lettere, da ordre listen ville være præcist den samme som det kunden selv har indtastet. Her ville formentlig blive sparet noget tid ift. retur på en vare, da tastefejl fra sælgerne ikke vil blive aktuelt.	Hvis sælgeren ikke opfanger de fejl som kunden potentielt kan taste ind, vil dette føre til problemer, hvor forkerte produkter bliver sendt. Derudover ville kommunikation en mellem kundeservice muligvis øge, for at rette eventuelle fejl, kunden har foretaget.	Har ingen indflydelse og deres aktive medvirken i projektet er ikke nødvendigt.	Man kan håndtere denne interesse ved at lave en opfølgning med dem, efter hjemmesiden er kommet op at køre, og kunderne har bestilt en selv designet carport.



Figur 3.1: Interressentdiagram

Interessentanalysen har bidraget til at analysere og identificere alle de involverede parter i projektet (interessenter), sammen med deres forventninger, indflydelse og krav. Det hjalp os som gruppe med at prioritere de givne krav mere effektivt. Ydermere gav det os en indsigt i organisationsstrukturen, og hvilket risici der kunne forekomme.

Risikoanalyse

Risikoanalysen bruges til at detektere risici for et projekt, der kan hjælpe med at identificere og finde strategier samt give en løsning til at overvinde eller minimere risici.

Fog Carport					
		Forebyggelse			
Risk ID	**Risiko**	**Alvor**	**Sandsynlighed**	**Risikoniveau**	Plan for forebyggelse/afværgning
1	Sygdom i gruppen	Tolereres	Muligt	Medium	Vi vil sørge for at have så meget mulig indsigt i hinandens arbejde, så skulle det ske en gruppemedlemmer bliver syg, kan vi stadig fortsætte nogle dele af arbejdet, så det hele ikke går i stå.
2	Work life balance	Tolereres	Sandsynligt	Høj	Vi vil forsøge og planlægge ugentligt, så vi kan overholde de deadlines vi har sat. Her er det vigtigt at vi er gode til at kommunikere og informere hinanden.
3	Dårlig database design	Uønsket	Muligt	Høj	Vi tager os god tid til at tænke over vores system og følge de 3 normalformer. Vi vil her forsøge at sikre os et godt udgangspunkt for databasens opbygning, så vi lettere kan tilføje eller fjerne, hvis det bliver nødvendigt.
4	Dårlig time management	Uønsket	Muligt	Høj	Vi skal her have fokus på at fordele opgaverne i forhold til deres størrelse. Sætte nogle deadlines for hvornår noget er færdigt. Her vil vi bruge Kanban Board som et redskab til at få et bedre overblik over opgaverne.

5	Dårlig kommunikation mellem gruppemedlemmer	Uønsket	Muligt	Høj	Der er vigtigt at få italesat eventuelt dobbeltarbejde i god tid, så man kan nå og få det stoppet. Løse de personlige konflikter der måtte opstå, og minde hinanden om at det er et projekt (tidspres, træt mm.) og ikke personligt.
6	Mangel på afklaring af funktionaliteter og opfølgning af projekt med kunden	Uacceptabelt	Muligt	Ekstrem	Vi vil forsøge at holde løbende møde, og få italesat den tvivl, der måtte opstå fra vores side af samt lave en opfølgning på projektet. Her kan vi modtage feedback, ændringer på krav eller andet.
7	Mangel på test af funktionalitet	Uønsket	Muligt	Høj	Vi prioriterer at teste det mest væsentlige af systemet. Hvis fejl opstår, ville det forhåbentligt kun være mindre fejl.
8	Problemer med teknologien der anvendes i projektet	Uacceptabelt	Usandsynlig	Høj	Informere kunden om problematikken med teknologien, og eventuelt finde en alternativ løsning.
9	Mangel på kompetencer til at kunne udøve de krav kunden har	Tolereres	Muligt	Medium	Vi vil her kommunikere med kunden, om at finde en alternativ løsning der imødekommer de færdigheder vi har.
10	Kunderne kan ikke finde ud af at navigere rundt på hjemmesiden og de får en dårlig brugeroplevelse	Uønsket	Muligt	Høj	Før vi færdiggør vores produkt og lancerer det, skal vi have testet det på kunderne gennem usability testing for at få deres mening om hjemmesiden.
11	Eksterne farer såsom voldsomme storme, vandalisme, brand osv. der gør at projektet eller firmaet bliver påvirket således at de ikke har mulighed for at fortsætte med projektet	Uacceptabelt	Usandsynlig	Høj	Man er opmærksom på medierne og omgivelserne, så man potentielt kan forberede sig så vidt muligt og kan kommunikere med hinanden om hvad der skal ske.
12	Modstand fra sælgerne (brugerne), idet de ikke ønsker forandringer og at de er vant til faste rammer.	Uønsket	Muligt	Høj	Medinddragelse for at afdække sælgernes behov og ønsker. Her vi kan anerkende deres ønsker, ved at få dem implementeret i projektet.
13	Uoverenstemmelser med Fog ved opfyldelse af krav, som kan føre til manglende betaling	Uacceptabelt	Muligt	Ekstrem	Der skal laves nogle konkrete aftaler på hvad der vurderes som acceptabelt, når kravene skal imødekommes.

14	Vejleder er ikke tilgængelig	Tolereres	Muligt	Medium	Planlæg i god tid og få fastlagt flere vejledermøder over projektets periode.
15	Hvis direktionen vælger at afbryde projektet	Uacceptabelt	Usandsynlig	Høj	Forsøge at være opmærksom og have indsigt i virksomhedens økonomi og strategier.
16	Andre udbydere (konkurrenter) tilbyder billigere/bedre løsninger til Fog inden der tegnes en kontrakt	Uønsket	Muligt	Høj	Tilbyde andre værdiskabende tjenester/produkter.
17	Dårlig presseomtale af Fog som virksomhed	Uønsket	Muligt	Høj	Have nogle strategier klar til at forsvare sig mod disse.

Analysen har hjulpet gruppen med bedre forståelse af omfanget af projektet og de krav, som klienten har stillet. Derudover har den hjulpet med prioritering af opgaverne samt med at identificere de problematikker der måtte opstå, og imødekomme disse inden problemet bliver for stort. Analysen har bidraget til identificering af potentielle risici for brugeroplevelsen og dermed hjulpet os med at designe og udarbejde en Mockup-design af hjemmesiden.

4. Teknologivalg

Java:

- JDK Version 17

Javalin:

- Javalin Version: 6.1.3
- Javaling Rendering Version: 6.1.3

Thymeleaf:

- Thymeleaf Version: 3.1.2 RELEASE
- Thymeleaf Extras Java8Time Version: 3.0.4 RELEASE

Logging:

- SLF4J Version: 2.0.12

JSON Processing:

- Jackson Databind Version: 2.17.0-rc1

Database Connection Pooling:

- HikariCP Version: 5.1.0

Database:

- PostgresSQL JDBC Driver Version_ 42.7.2

Testing:

- JUnit Version: 5.10.2
- Hamcrest Version: 2.2

Maven Plugings:

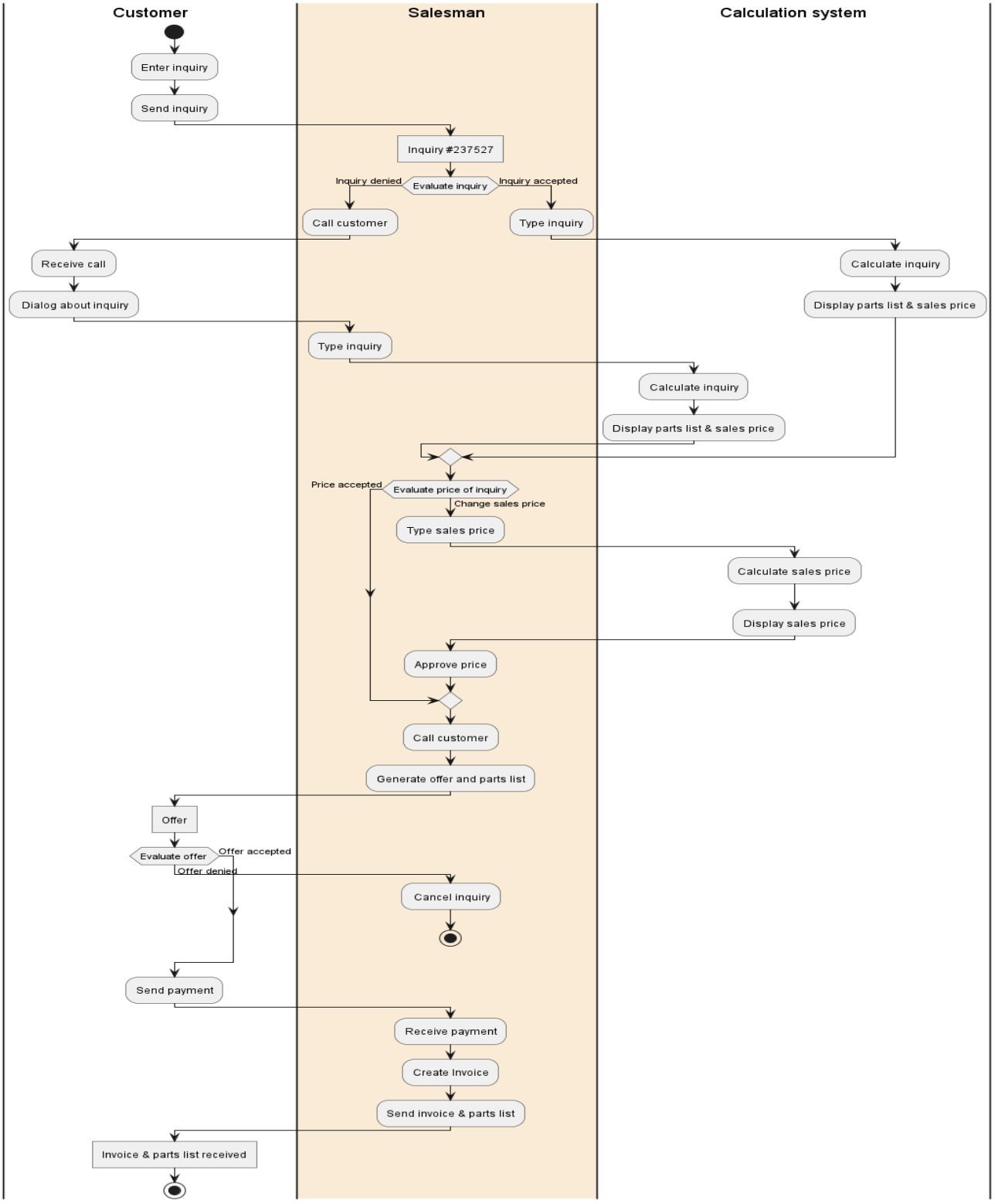
- Maven Compiler Plugin Version 3.10.1
- Maven Shade Plugin Version 3.4.1

DigitalOcean

5. Krav

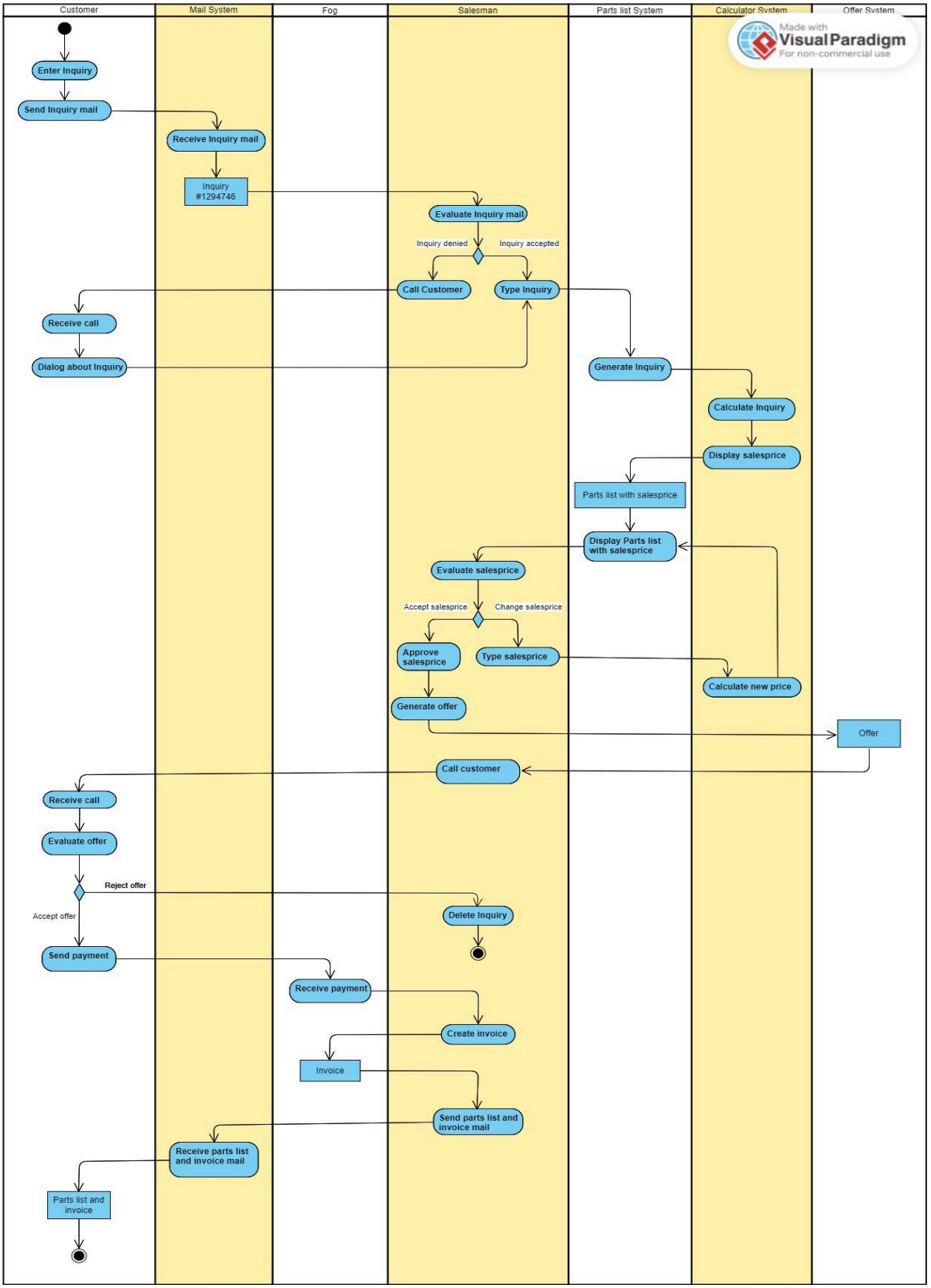
Aktivitetsdiagram

AS/IS aktivitetsdiagram



Figur 5.1: AS/IS aktivitetsdiagram

TO/BE aktivitetsdiagram



Figur 5.2: TO/BE aktivitetsdiagram

Vi har fået vist en video, der omhandler Fogs nuværende arbejdsgang, hvor vi derefter har udarbejdet et AS/IS aktivitetsdiagram. På videoen oplyser de, hvad deres ønsker er fremover, som vi har kortlagt i et TO/BE aktivitetsdiagram. Vi vil nu gennemgå de overvejelser vi har gjort os, i forbindelse med udarbejdelsen af TO/BE aktivitetsdiagrammet samt redegøre for ændringer fra AS/IS.

Vi har lagt meget vægt på at forbedre de problemer, de har med det eksisterende program. Som Martin fortæller i videoen, så har de besvær med at tilføje og ændre på de materialer som der bruges til beregning af prisen for carporten derfor har vi lagt meget vægt på at det er blevet nemmere på admin siden. Her har vi haft fokus på både at se forespørgsler og få en oversigt over hvilke kunder, der mangler at godkende eller afvise et tilbud. Dette gør det nemt at ringe til den givne kunde og følge op på spørgsmål.

Målet er også at gøre det langt nemmere, når man er logget ind som admin. Her har man mulighed for at ændre i beskrivelsen og prisen på de eksisterende materialer. Derudover skal det være nemt og tilføje nye materialer samt at slette materialer, som er udgået fra sortimentet. På kundens side er der også lagt fokus på, at det skal være en simpel og let tilgængelig måde at bestille en carport på. Derfor har vi sigtet efter at lave en minimalistisk hjemmeside både på udseendet og gjort det nemt for kunden at bestille en forespørgsel og modtage et tilbud.

Vi har på baggrund af det ovenstående udformet denne nye proces fra kunden indgiver en forespørgsel til kunden modtager ordren.

Fra de tre lanes i AS/IS (*Customer, Salesman, Calculator System*) har vi tilføjet 4 nye swimlanes: *Mail System, Fog, Part list System og Offer System.* Det er gjort for at gøre det mere klart hvor i processen, ordren bliver håndteret.

Når en kunde indsender en forespørgsel, bliver den nu grebet af *Mail System* og ikke af *Salesman* som tidligere. Her bliver den gemt i databasen. Sælger der modtager den givne sag, kan nu tage kontakt til kunden eller gå videre med at lave et tilbud. Ordren bliver herfra sendt til *Part list System* for at finde de materialer der skal bruges og *Calculator System,* der beregner prisen.

Sælgeren kan i *Calculator System* lave ændringer til prisen eller sende det færdige tilbud til kunden, som gemmes af *Offer System*. Sælgeren kan nu ringe til kunden om det færdige tilbud. Kunden modtager opkaldet og skal nu beslutte sig, om de vil tage imod tilbuddet.

Når *Fog* har modtaget betalingen, kan sælgeren lave en faktura. *Fog* gemmer fakturaen i databasen og sælgeren kan sende faktura og styklisten via *Mail System* til kunden.

Disse diagrammer gav et godt fundament, for at udarbejde specifikke User Stories, som er baseret på de nye ønsker Fog havde.

User Stories

US-1: Som kunde ønsker jeg at kunne oprette mig som bruger for at kunne bestille en carport.

Givet at jeg taster min email og et kodeord i en formular **når** jeg klikker på login-knappen, **så** bliver jeg logget ind på systemet og kan nu få lov til at bestille en carport.

Estimat: Small

US-2: Som kunde ønsker jeg at kunne sende en forespørgsel ind af en carport med egne valgte mål.

Givet at jeg er logget ind og klikker på *Quick Byg*-knappen samt indtaster tagtype, tagmål, tagmateriale og mål af redskabsrum, hvis ønsket, **når** jeg klikker på *Bestil Tilbud*-knappen, **så** sendes der en forespørgsel ind med mine egne mål og ønsker.

Estimat: Large

US-3: Som admin ønsker jeg at få vist en liste over materialer og beregne prisen på en kundes forespørgsel.

Givet at jeg er logget ind som admin og indtaster alle oplysninger fra en kundes forespørgsel, **når** jeg klikker på *beregn stykliste*-knappen **så** bliver jeg vist en liste af alle materialer der kræves for at bygge carporten samt indkøbsprisen, salgsprisen, med og uden moms, og dækningsgrad i procent.

Estimat: Large

US-4: Som admin ønsker jeg at kunne ændre i salgsprisen på en carport som er blevet udregnet ud fra en kundes forespørgsel.

Givet at jeg er logget ind som admin og har klikket på *beregn stykliste*-knappen, **når** jeg bliver vist indkøbsprisen og salgsprisen, med og uden moms, **så** kan jeg indtaste en ny salgspris med moms hvorefter salgsprisen uden moms samt dækningsgraden i procent bliver omregnet.

Estimat: Medium

US-5: Som admin kan jeg tilføje nye materialer til databasen og ændre i prisen samt beskrivelsen på materialer der allerede eksisterer.

Givet at jeg er logget ind som admin **når** jeg klikker på *alle materialer*-knappen, **så** kan jeg vælge at tilføje et nyt materiale eller ændre i beskrivelsen og/eller prisen på de materialer der allerede eksisterer.

Estimat: Medium

US-6: Som kunde kan jeg se byggevejledningen for en carport, når jeg har foretaget mig et køb.

Givet at jeg er logget ind som kunde og har købt en carport, **når** jeg klikker på *mine køb*-knappen, **så** kan jeg vælge en carport jeg har købt og se byggevejledningen for den.

Estimat: Small

US-7: Som kunde kan jeg modtage et tilbud på min forespørgsel med overordnet beskrivelse af carporten samt den totale pris.

Givet at jeg er logget ind som kunde og har indsendt en forespørgsel **når** jeg klikker på *mine forespørgsler*-knappen, **så** kan jeg vælge et tilbud jeg har fået tilbage og se den overordnede beskrivelse samt pris for en carport.

Estimat: Small

US-8: Som kunde kan jeg acceptere eller afvise et tilbud jeg har fået for et forespørgsel jeg har indsendt.

Givet at jeg er logget ind som kunde og har indsendt en forespørgsel **når** jeg klikker på *mine forespørgsler*-knappen, **så** kan jeg vælge et tilbud jeg har fået tilbage og vælge at afvise eller acceptere.

Estimat: Small

US-9: Som kunde kan jeg se egen ordrehistorik for tidligere køb.

Givet at jeg er logget ind som kunde **når** jeg klikker på ordrehistorik-knappen, **så** kan jeg se alle mine tidligere køb/invoices samt byggevejledningen der tilhører købet.

Estimat: Small

US-10: Som admin kan jeg se afsluttede ordrer og sortere/filtrere i dem.

Givet at jeg er logget ind som admin **når** jeg klikker på afsluttede ordre-knappen, **så** kan jeg se alle afsluttede ordrer der er blevet købt og jeg kan vælge at sortere/filtrere for at finde specifikke afsluttede ordrer.

Estimat: Small

US-11: Som admin kan jeg se afventende godkendelse på tilbud og sortere/filtrere i dem.

Givet at jeg er logget ind som admin **når** jeg klikker på afventer kunde-knappen **så** kan jeg se alle tilbud der afventer kunde godkendelse/afvisning og jeg kan vælge at sortere/filtrere for at finde specifikke afsendte tilbud.

US-12: Som admin kan jeg se forespørgsler og sortere/filtrere i dem samt have mulighed for at behandle dem.

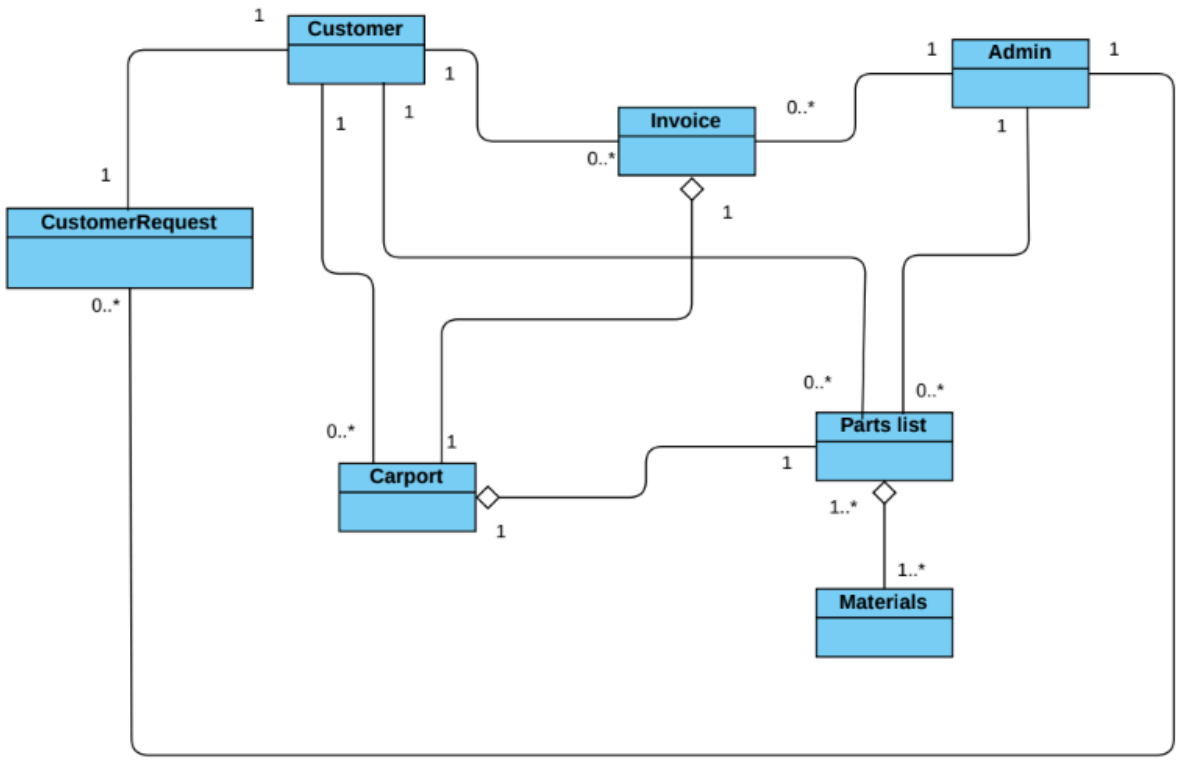
Givet at jeg er logget ind som admin **når** jeg klikker på forespørgsler -knappen **så** kan jeg se alle forespørgsler der er blevet lavet af kunder og jeg kan vælge at sortere/filtrere for at finde specifikke forespørgsler.

---

6. Domænemodel og EER diagram

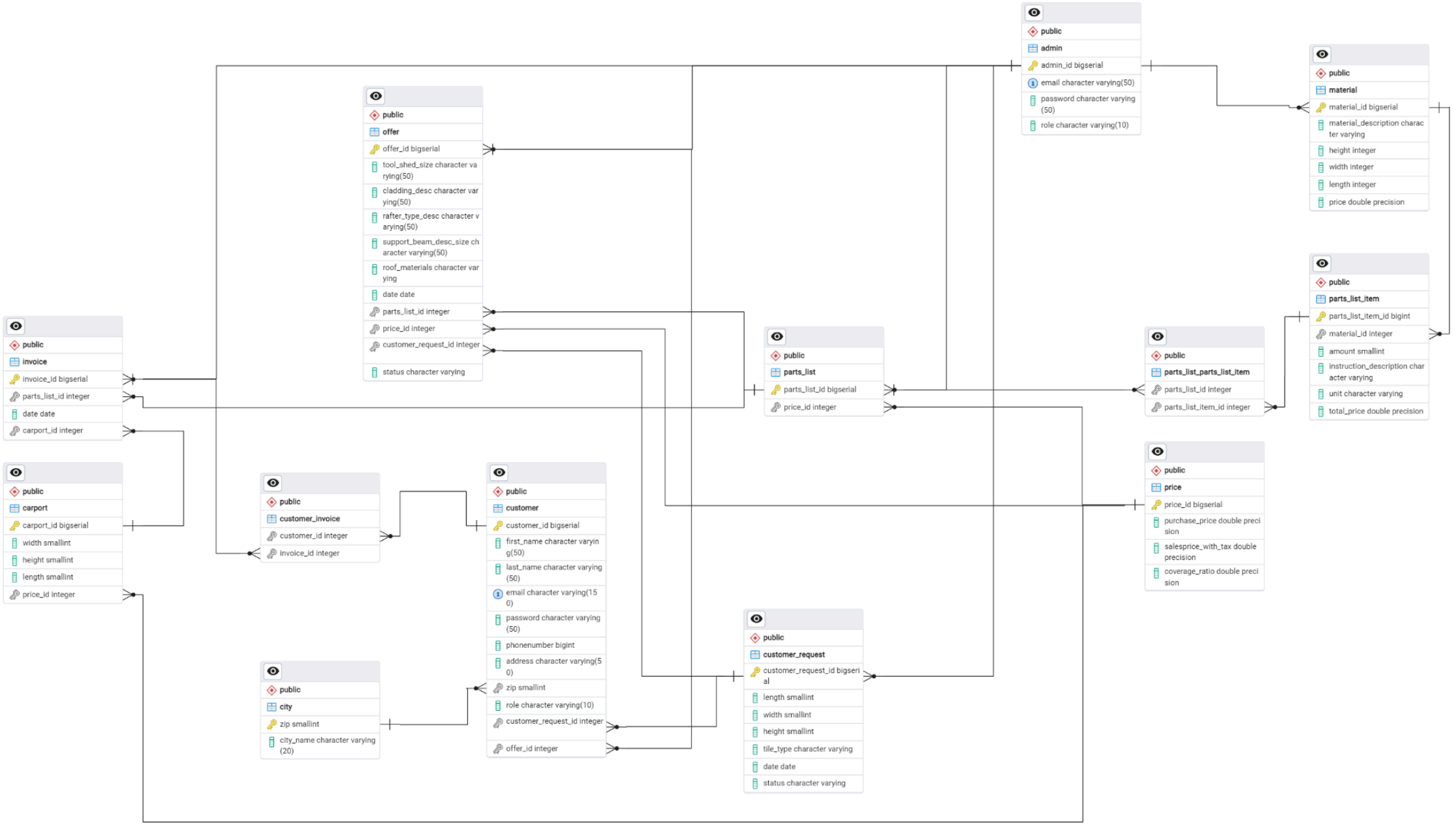
Domænemodel

I begyndelsen af projektforløbet designede vi ud fra kundeanalysen en domænemodel, der skulle afspejle objekter fra den virkelige verden. Vi udarbejdede modellen for at give os en klar, men dog konceptuel og visuel repræsentation af de enheder og relationer, vi ønskede at implementere i vores system.



Figur 6.1: Domænemodel

Ovenstående domænemodel, *Figur 6.1,* har været byggesten for det oprindelige udarbejdet ER-diagram (se bilag A, s. 1), som vi benyttede til at modellere vores database ud fra. Dog viste det sig under projektforløbet, at visse ting havde vi ikke forudset i vores ER-diagram. Dette resulterede i at databasen havde nogle fejl og mangler - både i forhold til relationerne mellem de forskellige enheder, men også den data man ønskede, at de skulle holde på. Dette rettede vi løbende, da det var essentielt for at applikationen skulle virke efter hensigten. *Figur 6.2* afspejler det endelige database design.



Figur 6.2: Endelig ER-diagram.

EER diagram

I det oprindelige ER-diagram havde *admin* tabellen flere mange-til-mange relationer til andre tabeller. Dette skyldes, at vi var for hurtige og ikke havde gennemtænkt, hvad dette egentlig ville betyde for en admin. Idéen var, at en admin skulle have mulighed for at kunne tilgå alle de enheder, den nu var forbundet til, og at systemet samtidig kunne have mere end én enkelt admin. Men dette kræver dog ikke en mange-til-mange relation, da der ikke er tale om, at mange admin kan tilgå, eksempelvis, mange tilbud - derimod skal enhver admin kunne tilgå allesammen. Dermed skulle *admin* tabellen kun have en en-til-mange relation til de forskellige tabeller, så alle mellemliggende tabeller er derfor blevet fjernet i det endelige ER-diagram.

Vi havde desuden også overset at alt pris-relateret data, havde den forkerte datatype, nemlig *integer,* hvilket vi valgte at ændre til *double*, da vi ønskede at gemme decimaltal i forhold til de resultater vi fik ud fra vores prisudregninger.

Vi opdagede derudover også at vores stykliste tabel *(parts_list)* i det oprindelige ER-diagram ikke forholdt sig korrekt i forhold til dets indhold. Til at starte med kunne *parts_list* tabellen indeholde data for bl.a. antal, pris og instruktionsbeskrivelse. Dette var dog ikke hensigtsmæssigt, da det var den enkelte stykliste, der havde den overordnede data, men derimod tilhørte dataen hvert enkelt vare i styklisten. Dette betød så at vi måtte tilføje en tabel, *parts_list_item,* der repræsenterede de enkelte varer i en stykliste.

Vi valgte også at fjerne *unit* tabellen, da den viste sig at være overflødig, og i stedet blev der bare skabt en ny kolonne i *parts_list_item* tabellen for at repræsentere *unit* direkte, hvilket stadig overholder 3. normalform. Derudover blev den mellemliggende tabel *carport* og *customer* fjernet, da en kunde ikke skal have mulighed for at tilgå en carport direkte, men kun hvis de har foretaget et køb. Carport skal kun kunne tilgås via *invoice* tabellen i den endelige ER-diagram.

Trods de mange justeringer vi måtte foretage på databasen i løbet af projektforløbet, er vores endelige ER-diagram ikke fejlfrit. Vores *offer* tabel opfylder ikke 3. normalform, da fremmednøglen *price_id* er afhængig af fremmednøglen *parts_list_id.* Den afhængighed er også repræsenteret i *parts_list* tabellen, så den direkte relation til *price* tabellen fra *offer* tabellen var unødvendigt, men dette fik vi dog ikke ændret på grund af tidspres.

Som det står til lige nu, har en kunde *(customer)* kun mulighed for at lave en forespørgsel *(customer_request)* ad gangen indtil den forespørgsel er afsluttet. Kunden kan også kun få et enkelt tilbud *(offer)* på den ene forespørgsel. Hvis forespørgslen først er afvist, er det ikke længere muligt at sende et nyt tilbud baseret på den samme forespørgsel. Kunden kan også kun købe en carport ad gangen, hvilket betyder at *invoice* tabellen har en 1-til-1 relation til *carport* tabellen. Disse 1-til-1 relationer kunne vi have ændret til 1-til-mange for nemmere fremtidige justeringer til vores hjemmeside, da kunden ikke skal begrænses på denne måde.

Imellem stykliste *(parts_list)* og den enkelte vare *(parts_list_item)* der er i styklisten, har vi lavet en mellemliggende tabel for at repræsentere mange-til-mange relationen. Dette har vi valgt, da en stykliste kan indeholde mange forskellige varer, og den samme vare kan være i mange forskellige styklister. Den samme mange-til-mange relation findes også mellem *customer* og *invoice*, men dette burde være en 1-til-mange relation, da en kunde kan have mange fakturaer, men den enkelte faktura tilhører kun en kunde.

Vi har en tabel for byer (*city),* hvor den primære nøgle ikke er autogenereret og det skyldes udelukkende at den første kolonne, som er zip koden, i forvejen er unik ud fra den by den tilhører - dog er det ikke nødvendigvis den bedste løsning, da den egentlig bruger noget rigtig data som primærnøgle, hvilket kan slå fejl hvis dataen på nogen måde ændrer sig i fremtiden.

Set i bakspejlet var der mange ting vi kunne have gjort anderledes, såsom at bruge længere tid på at finpudse vores ER-diagram, da det har været grundlaget for vores database og har væsentlig betydning for vores system. Dog er det også svært at udarbejde et fyldestgørende ER-diagram og dermed designe databasen uden fejl. Med lidt mere tid kunne vi muligvis have fanget mange af de mangler og fejl, der var i vores oprindelige database, således at vi ikke var nødsaget til at lave ændringer midt i vores projektforløb.

7. Navigationsdiagram & Mockups

Udgangspunktet for projektets design og opbygning har været på baggrund af vores Mockup, som er udarbejdet med hjælp fra design-værktøjet FIGMA (se bilag B, s.2-4).

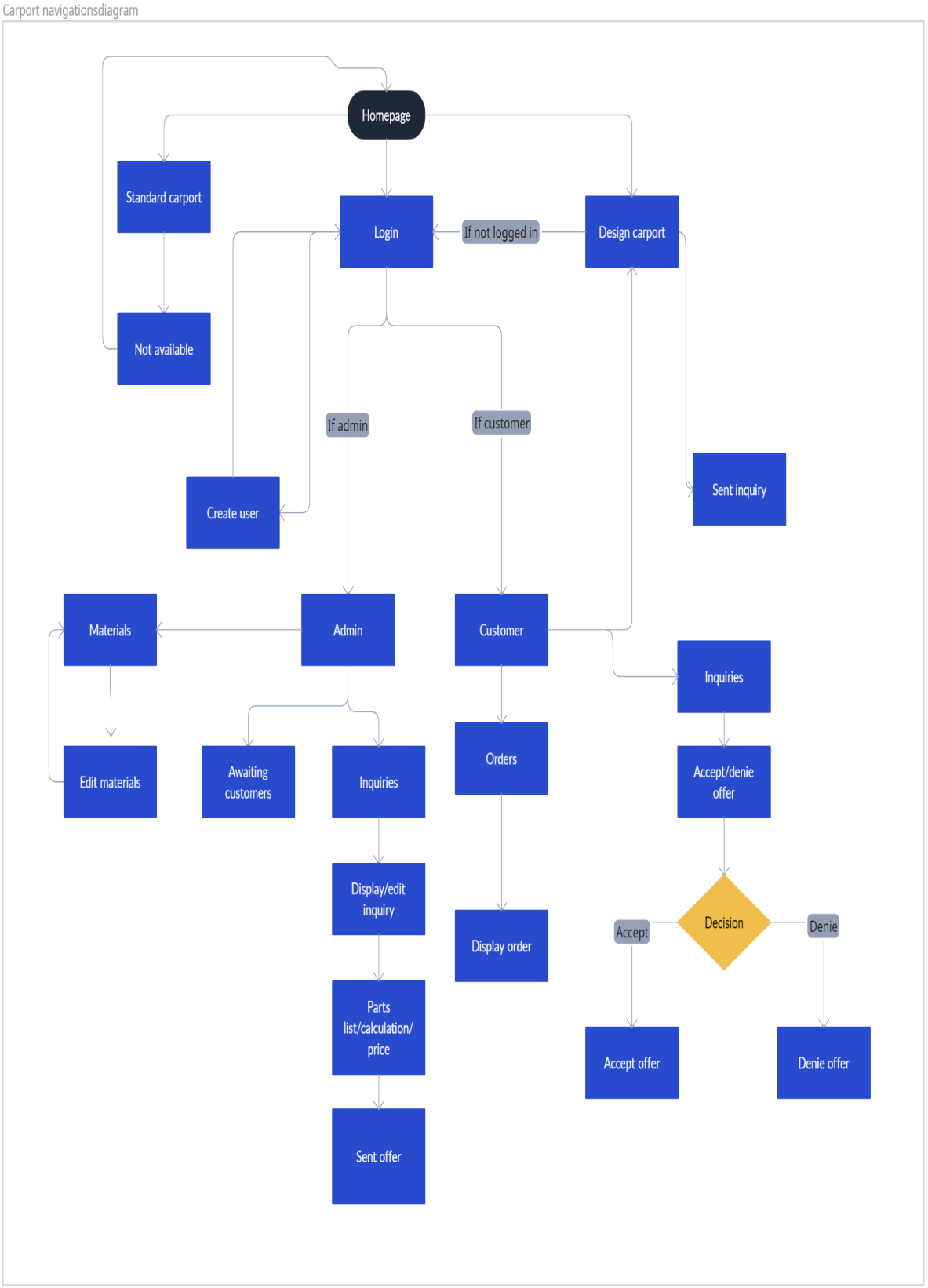
Vi har så vidt muligt forsøgt at udarbejde et intuitivt UI for at give en flydende UX vha. retningslinjer fra Laws of UX (Yablonski, 2024) herunder primært gestalt samt vores egen kreativitet, som har været præget af moderne hjemmesider. Vi har f.eks. grupperet objekter efter deres nærhed til hinanden og holdt det simpelt ift. formen på objekterne. F.eks. har vi mange firkanter med afrundede hjørner, idet forskning viser, at mennesker er visuelt bedre til at håndtere og huske mindre komplekse figurer frem for komplekse.

På grund af begrænset tid, har vi kun lavet et design der understøtter pc-skærm størrelser, selvom målet også var tablets og mobiltelefoner. Af samme årsag er responsiviteten af vores hjemmeside ikke blevet prioriteret, men vi har forsøgt at have det i mente, mens de enkelte sidder blev kodet.

Mockupen er udarbejdet i samme omgang som forretningsforståelsen, hvortil der her været tilføjelser og ændringer løbende. Det har gennemgået en iterativ proces, da vi løbende kunne se fordele og ulemper på den måde vi havde opstillet hjemmesiden i Mockupen, samt at der var forhold, vi ikke havde taget højde for. F.eks. blev materiale og udregningssiden lavet på bagkant, da vi så det fordelagtigt for koderen at have frie tøjler til at designe for at fremme en mere agil proces, hvor vi efterfølgende vurderede resultatet i fællesskab.

Mockupen har haft til hensigt at virke som en skabelon, som vi kunne læne os op ad til frontend-delen herunder HTML og CSS, men også backend-delen herunder hvordan siderne skulle kobles gennem controller- og routing-konfigurationer.

I Mockupen har vi navigationslinjer der viser, hvor man bliver ført hen, når man interagerer ved at klikke på diverse objekter såsom knapper, logoer mv., men navigations diagrammet udarbejdet for at fjerne forstyrrende elementer og derved give et overblik over, hvordan man navigerer rundt på vores hjemmeside.



Figur 7.1: Navigationsdiagram

Fælles navigationsbar

Øverst på siden er der en gennemgående og fælles navigationsbar for besøgende, kunder og admins, der er logget ind. Uafhængigt om man er logget ind, har man mulighed for at klikke på Fogs logo for at komme til forsiden. Alt efter hvem man er logget ind som, vil menu-knappen indeholde forespørgsler, ordrehistorik mv. tilpasset den aktuelle bruger herunder kunde/admin. Derudover kan begge brugere logge af fra hvilken som helst side.

Navigationsdiagram

Hjemmesiden består af adskillige sider, som kan tilgås af besøgende (ikke logget ind), kunder og admins, der er logget ind. Der er sider der kan tilgås af alle og sider, der kan tilgås afhængigt af om man er logget ind som kunde eller admin.

En besøgende kan tilgå design carport siden og vælge en højde, længde og bredde på en carport og trykke på forespørgsels-knappen, og hvis den besøgende ikke er logget ind vil blive stillet om til login siden, hvor den besøgende kan logge ind eller oprette en bruger. Den besøgende har ikke andre muligheder, hvis denne ikke er logget ind.

Hvis den besøgende (kunden) er logget ind, vil der efter man trykker på forespørgselsknappen blive vist en side, hvor Fog har modtaget forespørgslen. Kunden kan endvidere navigere til forespørgselssiden, hvor kunden kan se status på sin forespørgsel. Hvis denne status er klar, kan kunden klikke sig videre til siden og se et tilbud fra Fog. Kunden kan derefter vælge at acceptere eller afvise tilbuddet. Hvis tilbuddet accepteres, kan kunden efter Fog har bekræftet accepten, se styklisten, prisen og tegningen af carporten under "Mine ordrer"-siden.

Admin tilgår sine admin-rettigheder gennem en admin login og har andre sider tilgængelig end en kunde. En admin kan navigere til "Materiale"-siden hvor man kan se, redigere, tilføje og fjerne materialer fra listen. Derudover kan admin se alle forespørgsler fra potentielle kunder og vælge at ændre på carportens længde, bredde og højde (i samråd med kunden telefonisk) og derefter navigere videre til beregnings- og stykliste siden. Her kan admin ændre på pris og materialer samt se tegning af carporten. Derefter kan admin afsende et tilbud til kunden. Når kunden har accepteret (eller afvist) tilbuddet, kan admin se dette under "Afventende kunder"-siden. Hvis statussen er klar, kan admin sende stykliste, pris og tegning.

JSP og servlets

Som alternativ til JSP, anvendes ThymeLeaf templating engine, da vi arbejder med en Java-baseret webapplikation.

Thymeleaf konfiguration: Applikationen har en `ThymeleafConfig`-klasse, der konfigurerer en `TemplateEngine` med en `ClassLoaderTemplateResolver`, hvilket peger på Thymeleaf-skabeloner.

En servlet er en Java-klasse, der håndterer HTTP-anmodninger fra en klient (f.eks. en webbrowser) og genererer et svar (f.eks. en HTML-side). Når en bruger interagerer med en webapplikation, kan deres handlinger udløse forskellige servlets, som bestemmer, hvilken side brugeren ser næste gang.

Anvendelse i projektet

Brugeren anmoder om en side: Når en bruger anmoder om en bestemt URL, bliver denne anmodning sendt til en bestemt servlet, baseret på URL-mappingen defineret i vores applikation herunder de respektive Controllers.

Servleten behandler anmodningen: Servlet-klassen behandler anmodningen, udfører eventuelle nødvendige forretningslogik, og bestemmer hvilken Thymeleaf-skabelon, der skal bruges til at generere svaret.

Servleten videresender eller omdirigerer: Efter behandling af anmodningen, kan servleten enten videresende anmodningen til en skabelon (Thymeleaf-fil) for at generere HTML-svar, eller omdirigere til en anden URL, hvilket kan udløse en anden servlet.

Eksempel på anvendelse

```
app.get("/", ctx -> ctx.render("customer/carport-index.html"));
```

```
<a href="../login/login-page.html" th:href="@{/login-page}">
```

```
app.get("/login-page", ctx -> ctx.render("login/login-page.html"));
```

Fra carport-index.html til login-page.html
URL: /login-page
Controller: `LoginController`
Beskrivelse: Når den besøgende klikker på konto-knappen, sendes en GET-anmodning til `LoginController`, som derefter viser login-page.html.

```
<a href="/createuser">Opret en bruger</a>
```

```
<form class="generic-form" action="/createuser" method="post">
```

```
app.post("/createuser", ctx -> createCustomer(ctx, connectionPool));
```

Fra create-customer.html til create-customer.html

URL: /createuser

Controller: `CustomerController`

Beskrivelse: Når den besøgende klikker på konto-knappen, sendes en POST-anmodning til `CustomerController`, som derefter kalder på `createCustomer`-metoden, hvor der i denne bliver indhentet form parametre og derefter skrevet i databasen ved hjælp af en `CustomerMapper`-klasse med en `createUser`-metode.

8. Valg af arkitektur

I projektet har vi valgt at opbygge strukturen for vores webapplikation ved brugen af MVC-arkitektur (Model View Controller). *Model* håndterer data logikken, *View* præsenterer data visuelt og Controller styrer brugerinteraktionen og danner sammenhængen mellem View og Model. Formålet ved brugen af MVC er at få skabt en dynamisk struktur og forståelse for koden og hertil lave en alsidig opdeling, der gør det lettere at vedligeholde og teste koden. Hver opdeling får sine egne ansvarsområder, der skal sørge for at flowet i programmet kører.

MVC – arkitektur

Bilag C viser en oversigt over, hvordan vores package-struktur forløber sig. Med udgangspunkt i MVC arkitektur har vi opdelt systemet i *controllers*, *persistence* og *resources*. Ydermere har vi packages *config, entities, exceptions, services*, *validators* og *test*. Da vi havde to former for brugerinteraktion i systemet, valgte vi at opdele *controllers, persistence og resources* yderligere i *customer* og *admin* packages. Dette skabte et bedre overblik over hvilke sider, der skulle fremvise og hvilke klasser og tilhørende funktionalitet, der hængte sammen med brugerinteraktionen.

I controlleren håndteres brugerens input, interaktionen med persistence (Model) og sætter rammerne for det visuelle udfald for resources (View). Denne komponent indeholder servlets der håndterer HTTP-forespørgsler og styrer web-applikationens flow.

I persistence håndterer vi datalogikken af en request. Vores mapper-klasser interagerer med vores database, og ved brugen af pågældende CRUD operations, får vi hentet, aflæst, opdateret og slettet data fra vores database. Vi bruger denne komponent udelukkende til datahåndtering, da vores controllers ikke skal have nogen direkte indvirkning på vores data.

I resources er der fokus på visuel datapræsentation, hvor den pågældende data som controllers videresender fremvises. Resources er delt op i public, som indeholder CSS style-sheets, Images og JS, og templates som indeholder HTML-filer. Udover opdeling af admin, customer, login og svg, valgte vi at lave en fragments-package. Denne package indeholder et *header* layout som vi genbruger konsistent på forskellige HTML sider. Fordelen ved brugen af fragments, er at få defineret nogle af de fælles elementer og hertil sørger for, at skulle der foretages nogle ændringer, foretages det kun ét sted.

Config package indeholder to konfigurationsklasser, som hver er dedikeret til en bestemt del i applikationen. ThymeleafConfig bruges til at generere dynamisk HTML og SessionConfig konfigurerer sessions-håndtering.

Entities indkapsler de entiteter som står i relation til vores database. Dette gør det lettere at udføre CRUD operationer, da databasetabellerne mappes til objektklasser, som kan genbruges. Vi valgte at bruge entities-klasser til at kunne arbejde med databaseoperationer på et objektorienteret plan.

Exceptions bruger vi til at håndtere fejl på en struktureret og konsistent måde. Vi anvender en DatabaseException klasse, som undersøger fejl, der relaterer sig til vores database. Her inkluderer vi både bruger- og systemmeddelelser, der har hjulpet os til at få bedre indsigt i fejlen.

Services package er opdelt i to yderligere packages, calculator og svg. Disse to packages håndterer forretningslogikken, hvor udregningen af en styklisten, prisen for produktet og de pågældende svg tegninger bliver genereret.

Validators bruges til tre former for validerings-logik, *CityValidator*, *EmailValidator* og *PasswordValidator*. Vi valgte at lave en validator-package, for at kunne ændre i valideringslogikken uden at påvirke resten af applikationen og hertil genbruge valideringen forskellige steder, hvis nødvendigt.

Test package indeholder test af persistence, calculator og validators packages. Den primære test, er foretaget af mapper klassernes metoder, vi har i vores persistence package. Vi vælger at fokusere på dette, for at sikre den korrekte interaktion med vores database.

Vores arkitektur og package-struktur for web-applikationen har givet os et overblik og hjulpet os til at vedligeholde en robust struktur. (se bilag C, s. 5-6)

9. Særlige forhold

Kunde: rolle

Som bruger af systemet har man mulighed for at logge ind eller lave en ny kundekonto. En kunde er begrænset i forhold til, hvad de kan tilgå i databasen og har kun mulighed for at se data, der er relevant for kunden der er logget ind. Som kunde kan man kun lave en forespørgsel, acceptere eller afvise et tilbud og se faktura for de carporte, man har købt.

Kunde: kvittering session variabler

Vi anvender sessioner til blandt andet at opretholde brugerens kontekst mellem de forskellige HTTP-request. Et eksempel på dette er i *InvoiceController*, hvor metoden *displayCustomerOrderHistory* gør brug af dette. Når en bruger logger ind, gemmes brugerobjektet i sessionattributtet ved hjælp af *ctx.sessionAttribute* (*currentUser*). Dette tillader applikationen til at huske brugerens identitet på tværs af flere sider og giver hertil mulighed for at tilpasse indholdet baseret på brugerens rolle. For at tjekke hvilken rolle brugeren har, indsætter vi det i HTML-sidernes fragments på de sider, hvor brugerens rolle er essentiel. Herefter sættes samme brugerobjekt i session med nøglen *currentCustomer*. *currentCustomer* skal bruges til at finde den specifikke kundes ordrehistorik, som er en liste af de kvitteringer (*invoice*) som kunden har. Databaseinteraktionen foregår ved hjælp af JDBC i *InvoiceMapper*, hvor metoden *getCustomersOrderHistory* henter data for en kvittering, hvor kundens id, dato for køb og kvitteringens id returneres. For at imødekomme de pågældende exceptions der måtte forekomme, anvender vi vores *DatabaseException*-klasse hvor bruger- og systemmeddelelser bliver kastet, skulle der være en fejl i programmet. (se bilag D, s. 6-7)

Admin: Rolle

Som bruger af systemet har man mulighed for at logge ind som en admin, der gør at man har adgang til andre funktioner samt andre rettigheder i forhold til en almindelig bruger. En admin har mulighed for nærmest at tilgå alt i databasen og kan fjerne samt tilføje til databasen.

Admin: forespørgsel session variabler

Når man er logget ind som en admin bruger og vælger den forespørgsel, man ønsker at lave et tilbud til, så vil kundens navn og ID samt forespørgsel ID'et gemmes i sessionen. Dette har vi valgt at gøre, da de efterfølgende sider er afhængige af den forespørgsel, man har valgt, og benytter den til at udregne styklisten baseret på den data forespørgslen indeholder. Desuden bruger vi kundens navn til at blive vist på de efterfølgende sider, når man har valgt en forespørgsel, og når vi sender tilbuddet, gør vi brug af kundens ID. Hvis man vælger at gå tilbage og væk fra den valgte forespørgsel, bliver disse værdier fjernet fra sessionen - dette gælder også når man har valgt at sende et tilbud af sted.

Admin: forespørgsel input validering

Når en admin vælger at ændre på en kundes forespørgsel, er der på HTML-delen sørget for, at man kun kan indtaste mål indenfor et bestemt interval, afhængigt af om det er bredde, højde eller længde. Hvis admin indtaster et nummer uden for disse intervaller, vil de få vist en besked, der fortæller dem at de skal indtaste indenfor et bestemt interval. Derved har admin ikke mulighed for at gemme de indtastede mål før de alle er gyldige. Desuden kan admin ikke fortsætte til at beregne styklisten og pris, hvis målene er ugyldige, og hvis de nye indtastede mål ikke er blevet gemt.

Admin: stykliste beregner

Når en admin vælger at lave et tilbud ud fra forespørgslen, bliver styklisten også beregnet. I vores nuværende stykliste beregner vi indtil videre kun spær, remme og stolper for den ønskede carport. Her er der blevet taget højde for minimalt spild af spærtræ, ud fra nogle fastsatte spærtræslængder. Derudover har vi gjort nogle overvejelser om placeringen af stolperne i forhold til placeringen af remmene, således at det giver den pænest designede carport. Vi ønsker, at der skal være en stolpe lige under remmen, som den kan sadles ned i, når længden af carporten er for lang til kun at bruge en enkelt rem. Udover det vil vi ikke have at der bare bliver tilføjet endnu en stolpe for at støtte remmene, men at længden af remmene skal være baseret på antallet af stolper, vi i forvejen har udregnet.

Login funktionalitet

Login-processen finder sted i metoden *login* inden for *LoginController*. Denne metode er ansvarlig for at behandle login-forsøg via *login*-endepunktet. Den henter e-mailen og adgangskoden fra de formparametre, som brugeren har indsendt. Den bestemmer brugerens rolle, der forsøger at logge ind, ved at kalde metoden *isAdmin* fra klassen *LoginMapper*. *isAdmin*-metoden kontrollerer, om den angivne e-mail svarer til en administrator-bruger ved at forespørge *admin*-tabellen. Hvis metoden returnerer *true*, er brugeren en administrator; ellers behandles de som en kunde.

Hvis brugeren identificeres som en administrator, fortsætter processen ved at kalde metoden *login* fra klassen *AdminMapper*. Metoden i klassen *AdminMapper* forespørger databasen med den angivne brugerindtastning (e-mail og adgangskode) for at godkende administratoren. Hvis godkendelsen lykkes, returneres et *Admin*-objekt indeholdende administrator detaljer. Attributten *currentUser* i sessionen er sat til det godkendte *Admin*-objekt ved at bruge: *ctx.sessionAttribute("currentUser", admin).*

Hvis brugeren identificeres som en kunde (*isAdmin* er *false*), fortsætter processen med at godkende kunden ved at kalde metoden *login* fra klassen *CustomerMapper*. Tilsvarende med administratorens godkendelse forespørger `login`-metoden i *CustomerMapper* databasen med den angivne brugerindtastning for at godkende kunden. Hvis godkendelsen lykkes, returneres et *Customer*-objekt indeholdende kundeoplysninger. Attributten *currentUser* i sessionen er sat til *Customer*-objektet ved at bruge: *ctx.sessionAttribute("currentUser", customer)* **.**

Efter vellykket godkendelse sætter metoden en succes besked for både administrator og kunde ved at bruge: *ctx.attribute("message", message)*. Derefter omdirigeres brugeren til den passende side ved hjælp af: *ctx.redirect()* **.**

Hvis godkendelsen mislykkes, eller der opstår en fejl, sættes en passende besked ved at bruge: *ctx.attribute("message", errorMessage)*. Login-siden vises derefter igen med fejlmeddelelsen.

Vi valgte at bruge en enkelt *LoginController* med rollebaseret godkendelse og interaktioner med separate mapper af flere årsager. Ved at bruge en controller i stedet for to separate, undgår vi at duplikere kode til formular indsendelse, fejlhåndtering og datavalidering. At

have én controller forenkler vedligeholdelsesarbejdet, da opdateringer kan anvendes på et centralt sted, og det giver også en ensartet brugeroplevelse.

Tegning af Carport

I dette afsnit beskrives de særlige forhold, der er blevet anvendt i programmet til at tegne en carport baseret på kundens input. Programmet benytter sig af SVG (Scalable Vector Graphics) til at skabe præcise og skalerbare tegninger af carporten. Tegningerne inkluderer både et top-down og et side-view perspektiv, hvilket giver en mere omfattende visning af carportens dimensioner og strukturelle elementer.

Programmet består af flere Java-klasser, som samarbejder om at generere SVG-tegningerne. Grundstenen i denne implementering er *Svg*-klassen, som håndterer oprettelsen af SVG-elementer. Denne klasse indeholder skabeloner til forskellige SVG-elementer såsom rektangler, linjer, pile og tekst. Konstruktøren i *Svg*-klassen opsætter både en ydre og en indre SVG. Den ydre SVG bruges til at tegne målelinjer og pile, mens den indre SVG indeholder selve tegningen af carporten. Metoderne *addRectangle*, *addLine*, *addArrow* og *addText* bruges til at tilføje de grafiske elementer til SVG'en.

For at skabe en side-view tegning af carporten, anvendes *CarportSvgSideView*-klassen. Denne klasse tager længde og højde som parametre og opsætter en SVG-tegning med disse dimensioner. Konstruktøren initialiserer en ny instans af *Svg*-klassen, og tilføjer derefter de nødvendige strukturelle elementer som rem, spær og stolper ved hjælp af metoderne *addBeams*, *addRafters* og *addPoles*.

På samme måde håndterer *CarportSvgTopDownView*-klassen oprettelsen af en top-down view tegning af carporten. Konstruktøren i denne klasse tager længde og bredde som parametre og opsætter en SVG-tegning med disse dimensioner. Herefter tilføjes de strukturelle elementer som rem, spær, stolper og krydsstøtte ved hjælp af metoderne *addBeams*, *addRafters*, *addPoles* og *addCross*.

Visningen af SVG-tegningerne håndteres af *SvgController*-klassen. Denne controller håndterer HTTP-anmodninger for at vise tegningen af carporten. Metoden *displayDrawing* henter kundens ønskede dimensioner fra databasen ved hjælp af *AdminRequestMapper* og genererer SVG-tegningerne ved hjælp af de ovennævnte klasser. *CarportSvgTopDownView*

og *CarportSvgSideView*. De genererede SVG-tegninger bliver derefter sendt til de pågældende HTML-sider for visning.

Som et eksempel på en genereret SVG-tegning af carporten, kan følgende HTML-kode illustrere, hvordan tegningerne integreres i en webside:

```
<div th:utext="${svgSideView}"></div>

<div th:utext="${svgTopDownView}"></div>
```

Disse div-elementer indeholder de genererede SVG-tegninger, som vises på HTML-siden. SVG-koden genereres dynamisk baseret på kundens specifikationer og kan vise en præcis visualisering af carportens dimensioner og strukturelle elementer.

Det er vigtigt at bemærke, at tegningen er vejledende, men forholdsvis korrekt i forhold til styklisten. Dette betyder, at selvom tegningen giver en god visuel repræsentation af carporten, kan der være mindre afvigelser i forhold til den faktiske bygning. En væsentlig mangel i den nuværende implementering er, at der ikke er mållinjer med afstandsmål mellem for eksempel stolperne. Dette kan gøre det sværere at forstå de præcise placeringer af elementerne uden yderligere måledata.

Stolperne i carporten er placeret med en vis afstand til enderne, og når længden overstiger et bestemt mål, tilføjes en midterstolpe. Vi har vurderet, hvor langt det skal være, før der kommer en midterstolpe, for at sikre strukturel integritet. Denne vurdering er ikke baseret på standard praksis og erfaringer inden for bygningskonstruktion, men kan variere afhængigt af specifikke krav og betingelser.

En stor fordel ved vores tilgang er, at tegningen er nemmere at tilpasse, da vi tilstræber os på ikke at hardcode unødvendige værdier. De eneste steder, hvor der er hardcodet, er i forhold til materialernes faste mål. Dette gør det muligt hurtigt at ændre dimensioner og placeringer baseret på kundens behov uden at skulle ændre grundlæggende dele af koden.

Konklusionen på denne implementering er, at brugen af SVG-tegninger kan give en præcis og skalerbar metode til at visualisere kundens ønskede carport-design. Ved at anvende Java til at generere SVG'er, sikrer vi en høj grad af fleksibilitet og præcision i tegningerne, som kan tilpasses efter kundens specifikationer. Dette giver kunden mulighed for at få en detaljeret og nøjagtig visualisering af deres carport, hvilket kan hjælpe med at sikre, at det endelige produkt lever op til deres forventninger.

10. Udvalgte kodeeksempler

Beregning af antal stolper

```
public int calcPostQuantity() {
      int maxDistancePost = 340;
      return (2 * (2 + (carportLength - removeTotalLength) /
      maxDistancePost)); }
```

Til at beregne antallet af stolper der skal bruges for at bygge en bestemt carport, bruger vi ovenstående metode *calcPostQuantity* der returnerer en int. Vi har her antaget, at den maksimale afstand mellem hver stolpe er 340 cm. Variablen *removeTotalLength* repræsenterer den længde der skal fjernes fra carportens totale længde *carportLength*, da stolperne foran og forenden af carporten starter længere inde. Den forreste stolpe starter 100 cm fra den totale længde, mens den bagerste stolpe starter 30 cm længere inde, så dermed er værdien for *removeTotalLength* = 130. Længden der er tilbage, skal derefter divideres med *maxDistancePost*, for at finde ud af hvor mange stolper, der kan være inden for den længde. Da der som minimum i forvejen altid er to stolper for at holde taget på en carport oppe, så skal de to stolper tilføjes til resultatet. Derefter ganges det med to, da stolperne skal være på begge sider af carporten.

Beregning af antal remme

```
public int calcAvgDistanceBetweenPosts() {
      return (int) Math.floor((double) (carportLength -
      removeTotalLength) / (((double) calcPostQuantity()
      /2)-1));}
```

For at beregne antallet af remme for en bestemt carport skal vi først finde ud af den gennemsnitlige længde mellem hver stolpe. Længden der er tilbage efter at *removeTotalLength* værdien er blevet fjernet fra *carportLength* skal divideres med antallet af afstande, der er mellem stolperne på den ene side af carporten, da siderne alligevel er symmetriske. Dvs. at hvis det totale antal af stolper f.eks. er 6, så divideres det med 2 for kun at tage den ene side. Antallet af afstande mellem de 3 stolper er kun 2 - fra første stolpe til midterstolpe og midterstolpe til sidste stolpe. Derfor trækker vi antallet fra med 1. Vi runder

ned, hvis resultatet er et decimaltal, for at holde os inden for længden af (*carportLength* - *removeTotalLength*). Ud fra den gennemsnitlige stolpeafstand kender vi nu den gennemsnitlige længde for remmene mellem hver stolpe.

```
private void calcBeams() {
    int avgBeamLength = calcAvgDistanceBetweenPosts();
    int avgBeamLengthLast = avgBeamLength + removeLengthBehind;
    int amountOfDistancesWithAvg = ((calcPostQuantity()/2)-1);

    int remainingBeamLength;
    if (calcPostQuantity() == 4) {
      remainingBeamLength = avgBeamLength + removeTotalLength;
    } else if (calcPostQuantity() == 6) {
      remainingBeamLength = avgBeamLength + removeLengthFront;
    } else {
      remainingBeamLength = calcRemainingDistance() +
      removeLengthFront; } .... }
```

Ud fra antallet af stolper der skal bruges for at bygge carporten, kan vi komme frem til, hvad de forskellige remmelængder er. Hvis carporten kun har 4 stolper så har carporten kun en rem på hver side, hvis længde er *avgBeamLength* + *removeTotalLength*. Før blev *removeTotalLength* fjernet i *calcAvgDistanceBetweenPosts*, men da remmene starter længere ude end stolperne, skal det tilføjes her. Hvis carporten f.eks. har 6 stolper i alt, så har carporten 2 forskellige remmelængder på hver side af carporten. Vi har udregnet at den bagerste rem er *avgBeamLengthLast*, som er *avgBeamLength* længde plus 30. Den bagerste rem begynder 30 cm længere ude end den bagerste stolpe. Herefter beregner vi den forreste rem som *remainingBeamLength*, hvis længde er *avgBeamLength* plus 100, da den forreste rem starter 100 cm længere ude end den forreste stolpe. Hvis carporten har over 6 stolper, har carporten mindst 3 forskellige remme på hver side. Vi benytter metoden *calcRemainingDistance* til at udregne den forreste og længste stolpe, hvor vi så ligger værdien i *remainingBeamLength*.

Beregning af antal spær

```
public int calcAmountOfRafters(int rafterWidthMM) {
      double spaceBetweenCM = 55.0;
      double rafterWidthCM = rafterWidthMM/10.0;
      return (int) Math.ceil(carportLength / (spaceBetweenCM +
      rafterWidthCM));}
```

For at beregne antallet af spær, der skal bruges til en bestemt carport, benytter vi ovenstående metode *calcAmountOfRafters*(*int rafterWidthMM*). Metoden tager en int millimeter værdi ind i parameteren og omregner det til cm. Mellem hvert spær er der som udgangspunkt en maksimal afstand på 55 cm. Derefter dividerer vi længden på carporten med afstanden mellem spærene samt tykkelsen på hvert spær. For det totale antal spær bliver resultatet rundet op til det nærmeste hele tal, så afstanden mellem spærene ikke overstiger 55 cm.

Fragments og if statements i HTML

```
<div class="main-header" th:fragment="main-header(user)">
      <th:block th:if="${#strings.equals(user,'admin')}">
            <a href="../customer/carport-index.html"
                  th:href="@{/loginpage-admin}">
                        <img src="/public/images/fog-logo.png"
                  th:src="@{../images/fog-logo.png}" alt="Fog
                  Logo">
             </a>
      </th:block>
</div>
<th:block th:if="${#ctx.session.currentUser != null}" >
      <div th:replace="~{fragments/header ::
            main-header(${#ctx.session.currentUser.getRole()})}"
      >
      </div>
</th:block>

<th:block th:if="${#ctx.session.currentUser == null}" >
      <div th:replace="~{fragments/header :: main-header('')}">
      </div>
```

```
</th:block>
```

Koden er designet til at generere dynamisk HTML-indhold afhængigt af brugerens rolle og sessionsstatus i et fragment, så den lettere kan genbruges på tværs af flere HTML-sider. Den indeholder specifikke instruktioner til at vise en header, som varierer baseret på om brugeren er logget ind og hvilken rolle brugeren har.

Først defineres et fragment kaldet *main-header*, som tager en parameter *user*. Fragmentet indeholder et conditional statement, der kontrollerer om brugeren er en admin. Hvis brugeren er en admin (hvilket kontrolleres med `${#strings.equals(user,'admin')}`), vises et link til admin login-siden (*/loginpage-admin*) sammen med et billede af "Fog Logo". Dette sikrer, at admin-brugere får en specifik visning med et relevant dropdown menu.

Dernæst findes en blok, der håndterer situationen, hvor en bruger er logget ind. Denne blok (`<th:block th:if="${#ctx.session.currentUser != null}" >`) tjekker om der er en aktiv bruger i sessionen. Hvis der er, hentes fragmentet *main-header* igen, men denne gang med brugerens rolle som parameter (`${#ctx.session.currentUser.getRole()}`). Dette betyder, at headeren tilpasses baseret på brugerens rolle, som kan være forskellig fra admin og i dette tilfælde en customer.

Til sidst er der en blok, der håndterer tilfælde, hvor ingen bruger er logget ind. Denne blok (`<th:block th:if="${#ctx.session.currentUser == null}" >`) tjekker om der ikke er nogen aktiv bruger i sessionen. Hvis ingen bruger er logget ind, hentes fragmentet *main-header* uden nogen specifik brugerrolle (parameteren sættes til en tom streng). Dette giver en generisk header, som kan vises til besøgende eller brugere, der ikke er logget ind.

Sammenfattende, gennem disse tre hovedsektioner, sikrer koden, at headeren på websiden dynamisk tilpasses baseret på, om en bruger er logget ind, og hvilken rolle brugeren har. Hvis brugeren er en admin eller customer, vises en henholdsvis specifik dropdown menu til hver af rollerne. Hvis ingen bruger er logget ind, vises en generisk header. Dette giver en fleksibel og brugertilpasset oplevelse på webstedet.

INNER JOIN og 3NF

For at give kunden et overblik over sine ordrer og kvitterings-detaljer, valgte vi at implementere siderne *Forespørgsler* og *Mine ordrer*. Hvis kunden klikker på Mine ordrer, kommer siden *viewOrderHistory* frem, som skal give kunden overblik over deres tidligere køb. Hvis kunden ønsker at læse mere om den specifikke ordre, trykker man på knappen *Læs mere* og siden *viewInvoiceDetails* fremvises, hvor metoden *getCustomerInvoicedetails* fra *InvoiceMapper* blandt andet anvendes.

Koden benytter flere INNER JOIN operationer for at kombinere data fra forskellige tabeller for at hente alle relevante oplysninger om kundens kvittering.

```
String sql = "SELECT"
        customer.customer_id,
        invoice.invoice_id,
        material.material_description,
        material.length,
        parts_list_item.amount,
        parts_list_item.unit,
        parts_list_item.instruction_description
```

For at imødekomme første normalform, er der én nøgle, der entydigt identificerer den enkelte tabel, heriblandt *customer_id* og *invoice_id*. Ydermere må de enkelte felter indeholde én værdi og må ikke gentages, hvor *materiale_description*, length og amount blandt andet repræsenterer dette.

```
INNER JOIN public.customer_invoice
ON customer.customer_id = customer_invoice.customer_id
INNER JOIN public.invoice
ON customer_invoice.invoice_id = invoice.invoice_id
```

```
INNER JOIN public.parts_list
ON invoice.parts_list_id = parts_list.parts_list_id
INNER JOIN public.parts_list_parts_list_item
ON parts_list.parts_list_id =
parts_list_parts_list_item.parts_list_id
INNER JOIN public.parts_list_item
ON parts_list_parts_list_item.parts_list_item_id =
parts_list_item.parts_list_item_id
INNER JOIN public.material
ON parts_list_item.material_id = material.material_id
WHERE customer.customer_id = ?
AND invoice.invoice_id = ?";
```

For at imødekomme anden normalform har relationerne mellem tabellerne *customer*, *invoice*, *parts list* etc., sikret at dataen er normaliseret, da de har entydige identifikationer af henholdsvis en kunde (*customer_id*), kvittering (*invoice_id*) og stykliste (*parts_list_id*).

For at imødekomme tredje normalform, har vi designet vores database således, at kolonerne er direkte afhængig af primærnøglen. Et eksempel på dette er *materiale_description* , der har en direkte afhængighed af *material_id*, som dermed får en indirekte relation til *parts_list_item*. Det er her brugen af join-operationer bliver essentielle for at opfylde de pågældende krav.

Metoden *getCustomerInvoicedetails* er ét eksempel på brugen af INNER JOIN operationer, som har gjort sig gældende for adskillige metoder i vores mapper klasser. Dette har sikret normalisering op til tredje normalform og skabt en entydig identifikation af vores databasedesign.

11. Status på implementering

Admin: afventer kunder

Denne side blev lavet i sidste øjeblik, da vi opdagede at både siden og de tilhørende funktionaliteter manglede. Dette gjorde at vi måtte fokuserer på funktionaliteterne frem for designet. Siden har det mest nødvendige såsom header, baggrund etc. men den er manglende i forhold til at man godt kunne have vist bl.a. kundenavne eller andet relevant data.

Kunde: invoice details

Vi valgte at implementere siden *viewInvoiceDetails*, der skulle vise styklisten for det afsluttet køb og tilhørende svg tegning. Dog blev den visuelle fremstilling af styklisten ikke optimal. Styklisten skulle fremvise en liste, der vertikalt skulle lave kolonner, afhængig af hvor meget materiale der skulle anvendes til at bygge carporten. Men da vi skulle sammensætte svg-tegningen med styklisten, opstod der nogle fejl og vi havde problemer med at få det visuelle til at hænge sammen. Denne implementering havde til formål, at hjælpe kunden med at få et bedre overblik over sin stykliste, men grundet tidsmangel blev denne side ikke implementeret korrekt.

Test

Vores test fungere optimalt, når testklasserne bliver kørt hver for sig. Dog fejler en del af dem, når man forsøger at køre dem allesammen på én gang. Dette skyldes muligvis at hvert test klasse ikke er opsat helt korrekt, og at de ikke bliver nulstillet ordentligt. Grundet tidspres var det ikke muligt at udbedre det, da måden vi hver især tester på har været vidt forskelligt.

Offer

Efter at have inspiceret koden og den overordnede struktur på hjemmesiden, bemærkede vi et problem relateret til tilbud, som kunderne modtager. En kunde kan gå frem og tilbage mellem at acceptere og afvise et tilbud, hvilket ikke var den oprindelige hensigt. Meningen var, at en

kunde kunne acceptere eller afvise et tilbud én gang. Dette kan opnås uden at ændre databasen ved at tjekke session-attributten og betinget vise formularens elementer.

Customer Request

I *CustomerRequestController*, findes der en metode kaldet *getAllCustomerRequests*, som returnerer en liste af *CustomerRequests*. Oprindeligt, i begyndelsen af udviklingen, var meningen, at alle forespørgsler skulle vises (både aktive og afsluttede). Programmet er designet således, at en kunde kun kan have én aktiv forespørgsel ad gangen, og da vi senere besluttede, at kun aktive forespørgsler skulle vises, refaktorerede vi ikke metoden fra at returnere en liste til at returnere en enkelt forespørgsel.

12. Test

Automatiserede tests

Helt i begyndelsen af projektforløbet blev vi alle enige om, at vi ville teste de mest væsentlige og vigtige dele af vores implementationer, før det blev merged til vores main-branch. Dette har vi overordnet overholdt, hvilket også har gjort, at vi ikke skulle tilføje alle test helt til sidst i projektforløbet. Dog havde vi ikke aftalt, hvordan vi skulle lave testene, og om vi skulle benytte samme fremgangsmåde, hvilket har resulteret i at måden vores system er blevet testet på er lidt forskelligt. Ud over det skulle vi også være opmærksomme på at løbende køre testene igennem igen for at dobbelttjekke, at de stadig fungerer, jo mere systemet er vokset igennem projektforløbet. Ret ofte fejlede testene, men dette var fordi, at der var kommet flere afhængigheder i vores database, så der skulle slettes data fra flere tabeller end hvad man startede ud med.

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	92.3% (12/13)	66.7% (42/63)	67.7% (487/719)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
app.persistence	66.7% (2/3)	52.9% (9/17)	49.2% (62/126)
app.persistence.admin	100% (7/7)	69.2% (18/26)	73.5% (216/294)
app.persistence.customer	100% (3/3)	75% (15/20)	69.9% (209/299)

Figur 12.1: Overordnet overblik over code coverage for hele projektet

Gennem IntelliJ fik vi autogenereret en Code Coverage rapport, som kan findes i vores github under *doc → CodeCoverageRapport → index.html. Figur 12.1* giver et overordnet overblik over det samlede code coverage for alle test. Her kan vi se at vi mangler at teste en klasse under mappen *app.persistence* og code coverage kun er på *66.7%*, hvor man normalt vil sigte efter *80%*.

Current scope: all classes | app.persistence

Coverage Summary for Package: app.persistence

Package	Class, %	Method, %	Line, %
app.persistence	66.7% (2/3)	52.9% (9/17)	49.2% (62/126)

Class	Class, %	Method, %	Line, %
ConnectionPool	100% (1/1)	71.4% (5/7)	73.3% (22/30)
LoginMapper	0% (0/1)	0% (0/2)	0% (0/13)
OfferMapper	100% (1/1)	50% (4/8)	48.2% (40/83)

Figur 12.2: Overordnet overblik over code coverage for klasser under *app.persistence*

Current scope: all classes | app.persistence.admin

Coverage Summary for Package: app.persistence.admin

Package	Class, %	Method, %	Line, %
app.persistence.admin	100% (7/7)	69.2% (18/26)	73.5% (216/294)

Class	Class, %	Method, %	Line, %
AdminMapper	100% (1/1)	50% (1/2)	68.8% (11/16)
AdminOfferMapper	100% (1/1)	66.7% (2/3)	80% (28/35)
AdminRequestMapper	100% (1/1)	80% (4/5)	78.8% (41/52)
MaterialMapper	100% (1/1)	71.4% (5/7)	65.7% (65/99)
PartsListItemMapper	100% (1/1)	50% (1/2)	77.8% (14/18)
PartsListMapper	100% (1/1)	66.7% (2/3)	75% (21/28)
PurchaseMapper	100% (1/1)	75% (3/4)	78.3% (36/46)

Figur 12.3: Overordnet overblik over code coverage for klasser under *app.persistence.admin*

Current scope: all classes | app.persistence.customer

Coverage Summary for Package: app.persistence.customer

Package	Class, %	Method, %	Line, %
app.persistence.customer	100% (3/3)	75% (15/20)	69.9% (209/299)

Class	Class, %	Method, %	Line, %
CustomerMapper	100% (1/1)	62.5% (5/8)	56.2% (63/112)
CustomerRequestMapper	100% (1/1)	100% (8/8)	86.9% (113/130)
InvoiceMapper	100% (1/1)	50% (2/4)	57.9% (33/57)

Figur 12.4: Overordnet overblik over code coverage for klasser under *app.persistence.customer*

Hvis vi dykker ned i *app.persistence* set i *Figur 12.2* kan vi se at vi ikke har testet metoder i *LoginMapper* klassen. For *Figur 12.2, 12.3* og *12.4* kan man se at udover metoderne i *CustomerRequestMapper* så er mange af de andre metoder i de andre klasser ikke blevet testet fuldt ud. Hvis man kigger igennem code coverage rapporten, viser den at nogle metoder slet ikke er blevet testet, mens de fleste metoder kun til dels er blevet testet. Vi har for det meste kun lavet positive tests til vores metoder, hvilket også gør at alle linjer kode i en metode ikke nødvendigvis bliver berørt, hvilket har indflydelse på den overordnet code coverage for en enkel metode.

Overordnet set ville det have været en god idé hvis vi både havde lavet positive og negative tests for hver metode for at dække alle branches eller statements i metoden. Muligvis kunne man også have overvejet at følge en bestemt test-teknik, såsom white-box-testing, for at dække alle linjer kode og dermed få en højere code coverage.

Usability Testing

Den reelle kunde i dette projekt er Fog herunder deres repræsentant Martin. Det ville være det mest hensigtsmæssige at få Martin til at gennemføre en User Acceptance test af vores applikation, da det skal sammenlignes med de krav der er blevet stillet (User Stories) i samarbejde med virksomheden. Da vi ikke har fået gjort dette, har vi i stedet valgt at gøre brug af Usability Testing i stedet, hvor vi har fået hjælp fra en person fra vores eget netværk til at teste vores hjemmeside.

Usability Testing er en type brugerundersøgelse, der evaluerer brugerens oplevelse, når denne interagerer med en hjemmeside eller app. Det hjælper designere og produktteams med at vurdere, hvor intuitivt og brugervenligt produkterne er. Gennem Usability Testing kan man identificere potentielle problemer og områder for forbedring, hvilket resulterer i mere tilfredsstillende og effektive brugeroplevelser. Dette er afgørende for at sikre, at produkterne opfylder brugernes behov og forventninger, og for at forblive konkurrencedygtig på markedet. (Maze 2024)

Der er forskellige måder man kan udføre Usability Testing på, hvor vi har valgt at udføre det, som gav mest mening for vores projekt. Vores testperson er blevet sat ind i, hvad man skal kunne på vores hjemmeside, og derefter fik testpersonen til opgave at teste funktionaliteter baseret på vores User Stories. Vores prioritet med testpersonen har været på UX, navigation samt oplevelser af funktionaliteten. Der har været mindre fokus på design. Personen blev ikke vejledt undervejs og blev bedt om at tænke højt for at få så meget ud af testforløbet som muligt. Undervejs i testforløbet skulle vi logge ind som admin,og skulle acceptere og afsende en forespørgsel, tilbud, stykliste, pris og tegning, hvorfor testpersonen skulle logge ud og ind igen. Testpersonen har kun testet customer-siden og ikke admin-siden, da testpersonen skulle agere som en potentiel kunde, da det kun er Fog sælgere der har indsigt i admin funktionaliteterne. Der anvendes et skema med user stories med godkendt/ikke godkendt kolonne samt kommentarer fra testpersonen.

User Story	Godkendt/ikke godkendt	Kommentar
US-1 Oprette bruger og logge ind	Godkendt	Testpersonen får vist login siden efter, at testpersonen forsøger at oprette en forespørgsel uden at være logget ind. Testpersonen anerkender årsagen til omstillingen til login siden. Valideringer fungerer fint, da testpersonen forsøger at indsætte invalide input og kan se beskrivelser for acceptable input.
US-2 Design carport med egne mål	Godkendt	Valideringer fungerer fint, da testpersonen forsøger at indsætte invalide input og kan se beskrivelser for acceptable input. Testpersonen anerkender at forespørgslen ikke blev indsendt, da det krævede login. Testpersonen indsendte en ny forespørgsel.
US-6 Se byggevejledning	Delvis godkendt	Styklisten var forvirrende, men tegningen så fin ud. Der mangler dele af byggevejledningen.
US-7 Modtage tilbud	Godkendt	Testpersonen finder tilbuddet og gennemgår denne, men undrer sig over hvorfor tagmaterialet er sat til plasttrapezplader og andre øvrige materialer, da man ikke havde mulighed for at vælge.

US-8 Acceptere/afvise tilbud	Godkendt	Kan godkende, men forespørgslen forsvinder ikke og man kan godkende gentagne gange, men indser at man skal vente på at sælger skal godkende det før man modtager ordren.
US-9 Se ordrehistorik	Godkendt	Fin adgang til at få vist ordre i ordrehistorik.

Kommentar fra testperson

Det var ikke svært at navigere rundt på siden, og der var forholdsvis god logik ift. knappers placeringer mv. samt flowet. Det gav mening at man efter, at man har indsendt en forespørgsel, skulle afvente et svar, hvor man derefter fik et tilbud man kunne acceptere/afvise. Dog forsvandt forespørgslen ikke efter at, at man har accepteret tilbuddet, hvorfor det var forvirrende om sælger fik besked på, at man ønskede at acceptere.

Menuknappen med drop-down listen viste kun delvise tekster og indeholdt samme værdier, som når man klikkede ind på Fog konto-logoet, hvilket måske var lidt overflødigt. Det var udmærket med, at man vendte retur til forsiden, når man klikkede på Fog logoet i øverste venstre hjørne. Når man modtog ordren var det forvirrende, at se på styklisten, da det var en lang række uden forklaringer. Tegningen så fin ud. Generelt var hjemmesiden simpel og brugervenlig med nogle små fejl.

De ovennævnte fejl/mangler var vi bevidste om, og grundet tidspres og codefreeze, har vi ikke kunnet udbedre fejlene. Det ville være vores mål at gennemføre flere brugertests i flere iterationer og udbedre fejlene for hver fejl, der nu måtte opstå/være undervejs. Vi ville selvfølgelig starte med de fejl, vi var bevidste om før vi ville tage testpersoner i brug. Hvis tiden var til det, ville vi også udføre kvalitativ interview med forskellige testpersoner for at få input på designet, og hvordan denne kan forbedres til at give en bedre UX.

13. Proces

Arbejdsprocessen faktuelt

Første uge af projektprocessen valgte vi at fokusere på de diagrammer, som skulle skabe et fundament for projektets forløb. Vi udarbejdede en interessent- og risikoanalyse, og aktivitetsdiagrammer (AS-IS og TO-BE). Dette hjalp os til at forstå projektets domæne, og vi kunne herfra udarbejde nogle User Stories med tilhørende acceptkriterier. Herefter fik vi udarbejdet en database og et ER diagram, for at skabe et overblik over databasens relationer og dets kompleksitet. Derefter lavede vi et Kanban-board og tilhørende opgaver, baseret på vores User Stories. Ud fra disse kunne vi uddelegere opgaver og igangsætte kodningsprocessen. Vi forsøgte og opdele opgaverne op, så hver gruppemedlem fik lavet en Controller, Mapper, HTML og CSS, som minimum. Dette ville sørge for, at vi alle fik en indsigt og erfaring inden for diverse områder, der gør sig gældende for at udvikle web-applikationen. De første par dage i kodningsprocessen, var der fokus på HTML og CSS og efterfølgende fokus på de forskellige Controller og Mapper. Dog stødte vi ind i nogle problemer, da Login funktionalitet ikke var opsat korrekt fra starten af. Dette gjorde at vores Controller klasser kumulerede, da Controller som skulle bruge en *currentUser* skulle lave et nyt objekt af Customer, da vi ikke kunne anvende session attribute. Vi var derfor nødsaget til at refaktorere funktionerne i de relevante Controllere, hvilket skabte dobbeltarbejde.

Trods udfordringer fik vi sammensat projektet og hertil færdiggjort 9 ud af de 12, User Stories. US-10, US-11 og US-12 er delvist færdiggjort, idet vi mangler at implementere en søgefunktionalitet på hver af dem.

Arbejdsprocessen reflekteret

Kanban og Dagbog:

Vi havde svært ved at delegere projektet, da vi ikke nedbrød vores user-stories i mindre opgaver. I stedet for dette indsatte vi hele controllere og mappere som én opgave, hvilket vi senere fandt ud af ikke ville fungere. Som gruppe var vi heller ikke de bedste til at opdatere Kanban-boardet, når forskellige opgaver var udført. Derudover var tidsestimeringen for opgaverne ikke optimal, og vi måtte rykke flere gange på tidsplanen for at blive færdige. Vores primære kommunikationskilde var Discord, og i mellemtiden glemte vi ofte at opdatere

vores board. Dog var der meget kommunikation mellem gruppemedlemmerne, hvor vi tilstræbte at have daglige standup-meetings. Til møderne gennemgik hvert medlem kort, hvad der var lavet og hvad man på daværende tidspunkt arbejdede på. Efter opsamlingen koordinerede vi det videre forløb og uddeling af flere opgaver.

For at dokumentere de daglige opgaver i projektforløbet, valgte vi at skrive dagbog. Dette hjalp de gruppemedlemmer, der var fraværende den pågældende dag.

Entity klasser:

Det vi kunne have gjort fra begyndelsen, var at lave vores entity klasser sammen. Fordi projektet er stort, arbejdede nogle af os på de samme controllere og mappere, der brugte de samme entity klasser. I stedet for dette lavede vi vores egne entity klasser, og nogle af variabelnavnene matchede ikke. Dette førte til problemer, når vi skulle merge projektet i main, på GitHub, hvor vi manuelt måtte ændre variabelnavnene alle steder, det blev brugt.

Sammensætning af projektet:

Hvad vi kunne gøre bedre er selve sammensætningen af projektet. Da vi alle har arbejdsliv, skoleliv og privatliv, var det svært at finde tidspunkter at sætte programmet sammen i mindre klynger, som senere førte til routing problemer og overordnet flow i programmet.

Konklusion

Dette projekt har været en lærerig rejse gennem softwareudviklingsprocessen, hvor vi har arbejdet på at skabe en brugervenlig og velfungerende webapplikation til Fog og deres carporte. Fra de tidlige faser med risikoanalyse og udvikling af brugerhistorier til den endelige implementering af en kompleks database og brugergrænseflade, har vi stået over for udfordringer og lært meget undervejs.

I vores indsats for at skabe en solid databasestruktur og effektiv dataintegration har vi benyttet os af SQL-operationer og så vidt muligt overholdt principperne for normalisering. Dette har været afgørende for at sikre en gnidningsfri præsentation af kundens ordrehistorik og -detaljer. Ved at kombinere data fra forskellige tabeller har vi givet både kunder og administratorer et klart og omfattende overblik over ordrer, forespørgsler og materialer, hvilket har gavnlig indvirkning på effektiviteten af administratorrollen samt givet kunderne en mere sammenhængende brugeroplevelse.

Vores tilgang til testprocessen har også været omhyggelig. Vi har investeret i automatiserede tests og brugertests for at sikre kvaliteten af vores produkt. Disse tests har hjulpet med at identificere styrker og områder til forbedring, og de har været afgørende for at skabe et pålideligt og brugervenligt system.

I arbejdsprocessen har vi stået over for udfordringer med opgaveopdeling og kommunikation. Ved at reflektere over disse udfordringer og tilpasse vores tilgang har vi lært værdifulde lektioner om samarbejde og projektledelse. Gennem daglige standup-møder og brug af Kanban-board har vi forsøgt at opretholde en struktureret tilgang til projektstyring, selvom der har været plads til forbedringer.

På trods af de udfordringer, vi har stået over for, er vi stolte af det produkt, vi har skabt. Vi har skabt en funktionel og brugervenlig webapplikation, der opfylder mange af vores oprindelige mål. Gennem dette projekt har vi udviklet vores færdigheder som softwareudviklere og fået værdifuld erfaring, som vi vil kunne drage nytte af i fremtidige projekter.

14. Kildehenvisninger

Johannes Fog A/S (2024). Lokaliseret d. 23-05-2024:

https://www.johannesfog.dk/?msclkid=40587f64c2d41e0adb08192bdb2faaab&utm_source=bing&utm_medium=cpc&utm_campaign=BD_Search_Own-Brand_(_B_Webshop_)&utm_term=fog&utm_content=Fog

Manaz (2024). *Interessentanalyse.* Lokaliseret d. 23-05-2024:

https://www.mannaz.com/da/projektmodel/analysefasen/interesssentanalysen/

Manaz (2024). *Risikoanalyse.* Lokaliseret d. 23-05-2024:

https://www.mannaz.com/da/projektmodel/analysefasen/risikoanalyse-metodebeskrivelse/

Maze (2024). *What is usability testing?* Lokaliseret d. 23-05-2024:

https://maze.co/guides/usability-testing/

Yablonski, J. (2024). *Laws of UX*. Lokaliseret d. 23-05-2024:

https://lawsofux.com/