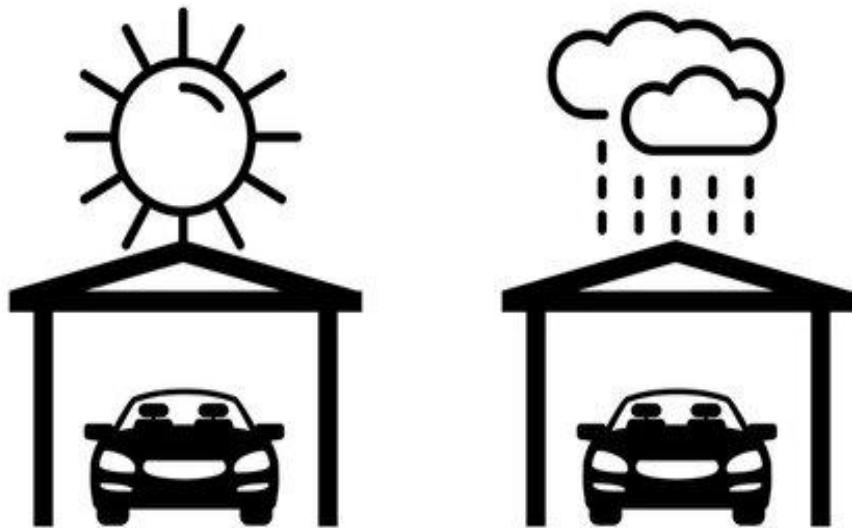


Fog Carport



Udarbejdet af:

Navn	Skolemail	Git Profil
Ahmad Alkaseb, Projektleder	cph-aa540@cphbusiness.dk	babanoelk
Hanni Salman	cph-ha186@cphbusiness.dk	HanniSalman
Lasse Kjær Hauerberg	cph-lh225@cphbusiness.dk	lassekh
Solomon Mwesigwa	cph-sm492@cphbusiness.dk	solo-king100
Youssef Badran	cph-yb48@cphbusiness.dk	badranyoussef

Link til projektet på GitHub: <https://github.com/babanoelk/FogsCarport/>

Antal anslag: 54 505 antal tegn med mellemrum

29/12 - 2023

Indholdsfortegnelsen

Studiekontrakten	4
Indledning	5
Problemstilling	6
Teknologivalg	7
Aktivitetsdiagram	8
AS-IS	8
TO-BE	8
Interessentanalyse	9
Risikoanalyse	15
Krav	18
User Stories & Acceptance Criteria	19
Domænemodel	25
Entity Relationship Diagram	27
Projektstruktur	30
Mappestruktur	30
Naming conventions	31
Klassediagram	33
Navigationsdiagrammet	35
Arbejdsproces	36
Daglig proces	36
Opgavestyring	37
GitHub branches	38
Kommunikation	39
Særlige forhold	40
Validerings logik	40
Session	41
Roller	41
Kode fremvisning	42

Repository Pattern	44
Figma	45
Test	46
Konklusion	47
Bilag 1	50
Bilag 2	51
Bilag 3	52
Bilag 4	53
Bilag 5	54
Bilag 6	55
Bilag 7	56
Bilag 8	57
Bilag 9	58
Bilag 10	59
Bilag 11	60
Bilag 12	61
Bilag 13	62
Bilag 14	63
Bilag 15	64
Bilag 16	65

Studiekontrakten

I vores gruppe deler vi en fælles vision og høje ambitioner. Det har været afgørende for os at etablere klare retningslinjer fra starten for at sikre, at vores forventninger til hinanden er tydelige. Vores højeste prioritet har været vores fælles projekt. Vi er glade for at kunne konstatere, at vi alle værdsætter og er tilfredse med hinandens arbejde, og vigtigst af alt, at vi har fulgt vores studiekontrakt.

Studiekontrakten har fungeret som en naturlig retningslinje i vores gruppe, og dens tilstedeværelse har været en selvfølge for os. Heldigvis har vi indtil nu ikke stødt på nogle udfordringer, hverken i vores samarbejde eller i at nå vores mål.¹

¹ <https://github.com/babanoelk/FogsCarport/blob/main/documents/contracts/Study-Contract.docx>

Indledning

Denne rapport er udarbejdet som en væsentlig komponent i vores anden semester eksamensprojekt ved Copenhagen Business Academy i Lyngby. Projektet tjener som en praktisk demonstration af vores dybdegående forståelse og anvendelse af den teoretiske viden, vi har opnået gennem de første to semestre af vores datamatikeruddannelse.

Formålet med denne rapport er at guide og informere medstuderende, der enten er i gang med eller har afsluttet deres andet semester, om den kompleksitet og udfordringer, som er forbundet med udviklingen af et realistisk softwareprojekt.

I dette projekt har vi fået til opgave at udvikle et skræddersyet carport bestillingssystem, der tillader kunder at foretage individuelt tilpassede bestillinger af carporte, herunder valg af dimensioner og tilføjelse af et skur. Ydermere skal medarbejdere som benytter systemet kunne logge ind og se alle ordrer, samt behandle disse i samme system. For at realisere denne opgave har vi anvendt Javalin som vores web-framework og Thymeleaf som templating engine, hvilket sikrer en effektiv integration af backend- og frontend-funktionaliteter.

Dette projekt har haft til formål at demonstrere vores tekniske færdigheder, kreativ problemløsning og evnen til at anvende teoretiske koncepter i en praktisk kontekst, f.eks. SCRUM, MVC-arkitektur, CRUD-funktionaliteter. Et primært fokus har været at designe en brugervenlig applikation, der indeholder alle de nødvendige funktioner til effektivt at håndtere kundeorder.

Problemstilling

Virksomheden Johannes Fog står over for en udfordring i forhold til deres system til bestilling af carporte med specialmål. De arbejder i øjeblikket med flere forskellige systemer for at håndtere en enkelt bestilling af en carport, hvilket er tidskrævende og ineffektivt.

Lige nu skal Fog indtaste alle informationer modtaget af kunden, manuelt i et system der hjælper dem med at udregne prisen og en stykliste. Herefter skal de i et andet system besvare kundens henvendelse med et tilbud.

Deres nuværende system er ikke integreret med deres lagersystem, hvilket gør det umuligt for ikke-vante medarbejdere at bruge systemet.

Virksomheden har derfor givet os opgaven at udvikle et nyt, omfattende system, der forener alle de eksisterende funktioner i én integreret løsning. Johannes Fog søger et nyt system, som skal simplificere processerne samt brugeroplevelsen for både kunder og medarbejdere.

Teknologivalg

I det følgende afsnit lister vi alle teknologier vi har anvendt til udvikling af systemet, herunder både teknologier der har været et krav (ses forneden) og teknologier vi har tilvalgt er JavaScript og Sendgrid 4.10.1.

- IntelliJ 2023.2.3
- PGadmin 7.3
- Java 17
- Postgres 42.6.0
- Javalin 5.6.1
- Thymeleaf 3.1.1
- JDBC
- HTML
- CSS
- Linux
- Docker
- JUnit

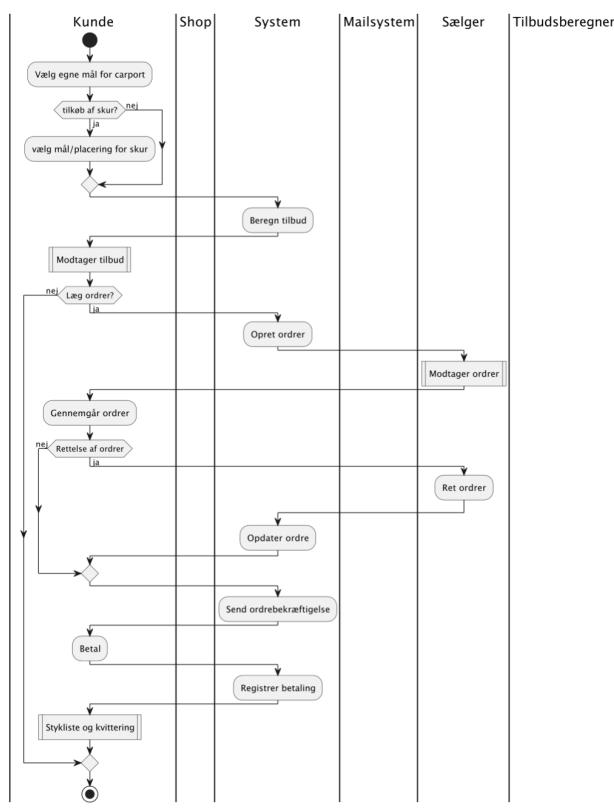
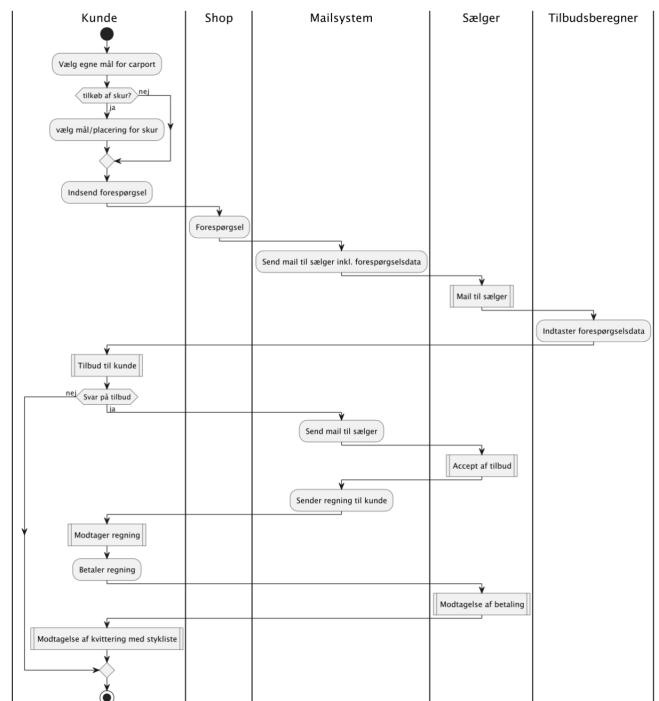
Javascript er ikke en del af pensum, men vi har valgt at anvende denne teknologi, da den kan forbedre brugeroplevelsen på klientsiden betydeligt. Vi har anvendt ChatGPT til at generere Javascript, idet ingen i gruppen er bekendt med teknologien. På trods af vores manglende kendskab til Javascript, var vi i stand til at læse og forstå den genererede kode.

Aktivitetsdiagram

AS-IS²

Efter at have talt med Martin (Fog), så kan vi konstatere at de har fem aktører, og at deres aktivitetsdiagram ser således ud.

En bestilling skal igennem fem aktører i form af systemer og personel. Fog synes processen er uoverskuelig og tidskrævende, og kun få medarbejdere kender systemet godt, men nye vil have svært ved at finde hoved og hale i deres nuværende måde at håndtere en bestilling af carport på.



² Bilag 2

³ Bilag 3

TO-BE³

Efter at have talt med Martin (Fog), så kan vi konstatere at vi kan lave en bedre løsning til dem. Vi er gået fra fem til tre aktører. Derudover bliver løsningen også meget mere overskuelig, plus at de kommer til at have færre systemer, som også kræver mindre arbejde fra sælgerne, således at de har mere tid til at kigge på flere ordrer.

Interessentanalyse

For at sikre en fremdrift i projektet, har interessentanalysen spillet en afgørende rolle for at skabe overblik over personer, som har interesse i projektet, og samtidig kan igangsætte processer. Ydermere har det givet os et indblik i hvilke fordele og ulemper der kan opstå for de enkelte interesserter og hvordan vi bedst håndterer dem.

Her ses interesserterne opdelt efter hvor meget indflydelse de har og hvor nødvendige interesserterne er for gennemførsel af projektet.

Gidsler	Ressourcepersoner
<ul style="list-style-type: none">• Medarbejder der skal bruge systemet• Kunder som skal benytte systemet• Marketingafdeling	<ul style="list-style-type: none">• Værebro Carporte (kontaktperson)• IT-afdeling (lagersystem)• Ekstern IT-konsultation
Eksterne interesserter	Grå eminencer
<ul style="list-style-type: none">• Johannes Fogs Fond• Bestyrelsen Johannes Fog A/S	<ul style="list-style-type: none">• FOGs ledelse• Jon

I skemaet nedenfor har vi set nærmere på hver intercessent, og beskrevet fordele og ulemper, som interessensten kan opleve, samt givet en vurdering af interessenstens vigtighed og hvordan vi vil håndtere den.

Interessent	Interessenten kan opleve følgende FORDELE ved projektet	Interessenten kan opleve følgende ULEMPER ved projektet	Samlet vurdering af interessenstens bidrag/position	Håndtering af interessensten
Medarbejder der skal bruge systemet	Mere effektiv ordrehåndtering.	Skal lære et nyt system at kende.	Medarbejderne bør inddrages i brugervenligheden af systemet til håndtering af bestillinger.	Lave brugertests på medarbejdere og bruge feedback til at forbedre systemet. Give et kursus, der uddanner medarbejdere i brug af systemet. Lave dokumentation for brug af systemet.

Kunder som skal benytte systemet	Nemmere bestilling, hvor tegning fremgår med det samme, og kortere svartid på tilbud.	Kunder kan indtaste forkerte oplysninger, der gör at den foreløbige pris bliver højere eller lavere.	Kunder som skal benytte systemet er en vigtig interessen, der skal mene at systemet er brugervenligt at anvende.	Skal inddrages i udviklingen, ved at lave brugertests og bruge den respons, der gives til at lave ændringer i systemet.
Marketingafdeling	Et nemmere system gør markedsføring og konvertering nemmere.	Forsinkelse af projektet, kan gøre det sværere at planlægge marketingaktiviteter. Fejl der opstår kan falde tilbage på marketingafdelingen, hvis det bliver en dårlig kundeoplevelse.	Marketing skal markedsføre det nye forbedret system, og skal derfor orienteres om udviklingen. Det er vigtigt det ikke går ubemærket hen når det nye system er implementeret.	Gøre det klart for marketing hvilke værdier systemet leverer ud til kunden, så de ved hvad de skal lægge fokus på i deres markedsføring. Lav en videogenmgang af systemet og screenshots der kan bruges til markedsføring.

Værebrou Carporte	Ingen manuel indtastning, kun gennemgang af kundens indtastning. Styrket proces i forhold til bestilling af carporte med specialmål. Større grad af fleksibilitet i systemet - flere valgmuligheder.	Fejlkilder i og med det er kundens indtastning, der er gældende. En kunde kan regne forkert på størrelsen af carporten, f.eks. hvis de ikke har taget højde for bilens størrelse og om eventuel skur størrelse rækker til behovet.	Vigtig interessent da det er vores ultimative kontakt, og dem der ved hvad systemet skal kunne.	Lyt til hvad de siger, da det er vores ultimative kontakt, og dem der ved hvad systemet skal kunne. Vær omhyggelig i udarbejdelse af funktionelle krav og User Stories, så systemet bliver som kunden ønsker.
IT-afdeling (lagersystem)	Automatisk opdatering af priser på materialer i det nye system.	Nyt system at tage hensyn til og evt. integrationsproblemer.	Vigtig interessent i forhold til integration mellem lagersystem og carport-system.	Indhent information om hvilket lagersystem der bruges, og hvordan man kan integrere det med carport-systemet.

Vi har i høj grad oplevet at de vigtigste interesserter har været **Værebro Carporte** (i dette tilfælde ageret af vores undervisere), og **eksterne IT-konsulenter** (i dette tilfælde ageret af vores undervisere og medstuderende, samt ChatGPT).

Vi har haft et løbende behov for at kommunikere med kunden omkring hvordan systemet har helt præcist skulle sammensættes. Særligt omkring den endelige stykliste og hvad den skulle indeholde.

Derudover har vi haft brug for hjælp til programmering af systemet i nogle tilfælde. Ved simple eksempler har vi spurgt ChatGPT til råds og fået et klart og brugbart svar. Vi har også fået hjælp af vores undervisere til at integrere mail-afsendelse i systemet og tegne SVG-filen der viser carporten set ovenfra.

Risikoanalyse

Riskostyring er en kunst, der håndterer og reducerer potentielle risikofaktorer, før de udvikler sig til trusler. Der er en vis grad for usikkerhed om, hvorvidt en risiko vil manifestere sig, men hvis det sker, kan det have en negativ påvirkning på projektet. Ved at kvantificere både sandsynligheden for risikoen, samt den potentielle skade, kan vi fra start identificere områder, hvor der er behov for ekstra opmærksomhed.

Herunder ses et billede af vores risikoanalyse:⁴

Forebyggelse					
Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau	Plan for forebyggelse/afværgning
1	Sygdom hvor man kan arbejde hjemmefra eller fraværende kun 1 dag	Acceptabel	Muligt	Lav	Sikre mulighed for at arbejde hjemmefra med online kommunikation - Afgiver start og slut status/briefing omkring hvad man skal arbejde med og hvad man har udført - Benyt projektstyrings værktøj for at sikre overblik og struktur
2	Sygdom hvor man ikke kan rumme at arbejde hellere ej hjemmefra - sengeliggende	Uønsket	Usandsynlig	Medium	Implementer en sundheds agenda, hvor medarbejderne tilbydes sund kost og gratis motion på arbejdsplassen - Revurdering af mål i tilfælde af en længere sygdomsforløb - Aftal en anden deadline med kunde - Sikre en IT-partner som kan hjælpe med at løse opgaver
3	Elektronik som går i stykker	Uønsket	Usandsynlig	Medium	Sikre cloud løsninger (Backup)
4	Begynder fejl - opsætning af projekt fra start. Vi er ikke professionelle endnu og kan glemme nogle vigtige opsættninger	Uønsket	Sandsynligt	Høj	Søg hjælp ved hinanden når man er i tvivl. Sikre en oversuelig kanban overblik hvor opgaverne tydeliggøres - Programmering 2 og 2 front+backend
5	Opsigelser blandt teamet	Acceptabel	Usandsynlig	Lav	Åben og ærlig kommunikation blandt teamet. Gør det klart hvis man har tanker om at træde ud - Ugentlig dialog omkring hvordan man har det og om der er noget man har svært ved.
6	Opsigelse af kontakt person. Dvs. kontakt personen opsigter sin stilling	Uønsket	Muligt	Høj	Sikre et kontakt team fremfor udelukkende en kontaktperson hos partneren - Lav en kontrakt baseret på de funktionelle krav. Dermed sikre den fremadrettede arbejde trods en opsigelse ved partneren.
7	tekniske udfordringer i selve sammenkoblingen mellem backend og front end	Tolereres	Sandsynligt	Høj	Genbesøg Jons video 'er omkring sammenkoblingen - Sikre enighed omkring navngivning af attributter som forbinder front end og back end - Søg hjælp eksternt
8	Markedsændring. Hvis kundens kunder ønsker ex. garager fremfor carporte i højere grad	Acceptabel	Usandsynlig	Lav	Bygge et system som har et højt abstraktions niveau - Vær omstillingssparate som team
9	Det endelig produkt er anderledes end hvad kunden ønskede	Uacceptabelt	Sandsynligt	Ekstrem	afstem med kunde via figma - Lav Kontrakt på de aftalte funktionelle krav og design
10	Nye ønsker fra kunden ud over det afstemte	Acceptabel	Muligt	Lav	Ud over kundens ønsker bør vi se på projektet fra andre perspektiver og evt. komme med ideer / tilføjelser - måske skulle der laves et cykelstavt også??

⁴ Bilag 1

Vi har været ude for at flere af de risici vi forberedte os på, er blevet en realitet. Det har blandt andet været vores risici 1 og 2 i forhold til sygdom og 3 og 7⁵, der handler om tekniske og projektstyring.

Flere af os blev ramt af sygdom i løbet af projektperioden, men det er forløbet uden de helt store problemer, da vi har haft en klar plan for projektet, og ikke har lagt et for stort arbejdspres på os selv. Det har betydet at de personer der har været friske, har vidst hvad de skulle tage fat på, og vores daglige “stand-up meeting” kl. Ca. 9, har gjort at dem der var syge kunne være med hjemmefra til at orientere gruppen om de opgaver de har været i gang med.

Sygdommen i gruppen har forhindret at vi har kunne få de sidste detaljer på plads i forhold til koden, men vores ambition om at lave et code freeze i den sidste projektuge, betyder at vi er kommet i mål med et produkt der virker dog ikke fejlfrit.

Vi har haft nogle begynderfejl, der har skabt udfordringer i løbet af projektet. Blandt andet har vi ikke brugt Figma i det omfang vi burde, til at lave skabeloner for vores HTML-sider. Vi valgte at starte med at lave skabeloner i Figma til et par sider, men kom siden fra det. Det skabte problemer i forhold til strukturen på de resterende HTML-sider, der blev opsat meget forskelligt, og særligt skabte det usammenhæng i vores CSS styling. Vi burde have brugt Figma til mock-up af alle HTML-sider, set i bakspejlet.

Fra et tidligere projekt havde vi haft problemer med sammenkoblingen frontend og backend. Det var igen noget der drillede, da vi flere gange havde misinformeret hinanden om navne på data, der skulle sendes frem og tilbage mellem frontend og backend. I og med vi vidste, hvilken data der var behov for at sende frem og tilbage, kunne vi fra start af have lavet en liste over navnene på det, for at undgå konflikter i systemet.

⁵ Bilag 1

Der var også situationer hvor flere gruppemedlemmer var begyndt at arbejde på samme funktionaliteter i systemet, hvilket er uhensigtsmæssigt, da det er dobbeltarbejde og tid spildt. Her kunne vi have gjort opgavebeskrivelserne i vores kanban-board tydeligere.⁶

En risiko vi ikke havde med var risikoen for hvor meget, vores meget forskellige arbejdstider uden for studiet, kunne påvirke arbejdsgangen i projektet. Det har givet mange dage, hvor nogle har måtte gå

tidligt el. ikke har været i stand til at deltage på bestemte tidspunkter, samt givet flere dage med online gruppearbejde. I virkeligheden har det udgjort den samme risiko som sygdom har. Her kunne vi have lavet et skema ved projektstart, hvor alle kunne have skrevet deres arbejdstider ned 4 uger frem, og valgt at mødes hele dagen på de dage hvor alle har mulighed for det, samt lagt vores daglige “stand-up meetings” på tidspunkter hvor alle kunne deltage.

Alt i alt har vi haft et godt projektforløb, hvor de risici vi havde forudset, har været håndterbare. Det har betydet at vi har måtte nedprioritere dele af projektet, men har ikke haft så stor betydning at vi ikke er kommet i mål med et funktionelt system.

⁶ <https://github.com/users/babanoelk/projects/3>

Krav

Vi har fået leveret en video af Product Owner (en af vores lærere), som definerer kravene til systemet. Denne video er blevet optaget ved Fog, hvor vi møder en af deres eksperter, Martin, i at bygge og levere carporte. Martin gennemgår deres nuværende system og krav til et nyt system. Derudover har vi haft nogle møder med Product Owner som har defineret nogle krav og svaret på de spørgsmål vi havde til systemkrav for at sikre forståelsen af Fogs krav.

Krav som kunde bruger:

- Kunder skal kunne bestille carporte med egne mål.
- Kunder skal kunne logge ind med deres nye bruger efter bestilling af carport.
- Kunder skal kunne se status på deres ordre.
- Kunder skal kunne sende en besked vedr. deres ordre til firmaet.
- Kunder skal kunne se stykliste for en ordre når de har betalt for carporten.
- Kunder skal kunne se en forhåndsvisning af den bestilte carport.

Krav som sælger og admin bruger:

- Admin skal kunne logge ind i systemet.
- Admin skal kunne oprette en sælger bruger i systemet.
- Admin og sælger skal se alle ordrer i systemet.
- Admin og sælger skal kunne behandle ordren:
 - Rette prisen.
 - Give rabat.
 - Rette i kundens oplysninger (navn, adresse, etc.).
 - Sende en regning til kunden (autogenereret mail).
- Admin & sælger skal kunne se en oversigt over varelageret.
- Admin skal kunne tilføje og slette varer i varelageret.

User Stories & Acceptance Criteria

Efter en omfattende forståelse af Fogs krav til det kommende system, som vi er ansvarlige for at udvikle, har vi på baggrund af samtalen udarbejdet nedenstående user stories og tilhørende acceptance criteria⁷. Følgende user stories er lavet til at afspejle, hvad vi har talt om, og for at sikre at alle relevante aspekter af systemet er dækket.

Nummer	User Story	Acceptance Criteria
1	Som en kunde hos Fog vil jeg gerne kunne bestille en carport med egne mål, så jeg kan få en skræddersyet løsning.	Givet at jeg er på Fog's side "bestil carport med egne mål". Når jeg udfylder alle obligatoriske felter på bestillingssiden, inklusive kontaktoplysninger og eventuelle yderligere oplysninger, og vælger carport-dimensioner via dropdowns, Så kan jeg sende forespørgslen ved at klikke på knappen "Send forespørgsel". Efter at have sendt forespørgslen, modtager jeg en bekræftelse på skærmen om, at forespørgslen er

⁷ <https://cphbusiness.mrooms.net/mod/book/view.php?id=625178&chapterid=28498>

		<p>modtaget og under behandling.</p> <p>Hvis ikke alle obligatoriske felter eller dropdowns er udfyldt, bliver jeg omgående bedt om at vælge de manglende data, og en relevant fejmeddeelse vises for at guide mig i at fuldføre alle påkrævede oplysninger.</p>
2	<p>Som kunde hos Fog vil jeg have mulighed for at tilføje et skur til min carport, så jeg kan tilpasse løsningen yderligere.</p>	<p>Givet at jeg er på Fog's side "bestil carport med egne mål".</p> <p>Når jeg har valgt mine egne mål for carporten.</p> <p>Så kan jeg vælge via radio buttons, om jeg ønsker at tilføje et skur til carporten.</p> <p>Efter at have valgt skur muligheden, kan jeg fortsætte med at udfylde eventuelle yderligere oplysninger og klikke på knappen "Send forespørgsel".</p>

		Hvis ikke alle påkrævede felter, inklusive skur muligheden, er udfyldt, bliver jeg omgående bedt om at vælge de manglende data, og en relevant fejlmeldelse vises for at guide mig i at fuldføre alle påkrævede oplysninger.
3	<p>Som en kunde eller sælger hos Fog</p> <p>Vil jeg kunne logge ind med med email og kodeord</p> <p>Så jeg kan se mit navn i toppen af hver side.</p>	<p>Givet at en bruger er logget ind,</p> <p>Når brugeren er logget ind med e-mail og kodeord,</p> <p>Så kan brugeren se sit navn i toppen af hver side.</p>
4	<p>Som en kunde hos Fog,</p> <p>Vil jeg kunne se min forespørgsel,</p> <p>Så jeg kan slette min ordre.</p>	<p>Givet at kunden er logget ind,</p> <p>Når kunden er på “min-side”,</p> <p>Så kan kunden vælge at slette en ordre.</p>
5	<p>Som en salgsmedarbejder hos Fog</p> <p>Vil jeg gerne kunne logge ind på Fog's admin-side,</p>	<p>Givet at en salgsmedarbejder er på Fog's “login-side”.</p> <p>Når salgsmedarbejderen indtaster de korrekte</p>

	<p>Så jeg kan se alle indsendte forespørgsler.</p>	<p>loginoplysninger og logger ind.</p> <p>Så kan salgsmedarbejderen klikke på menu knappen “Alle ordre” og få vist en oversigt over alle indsendte forespørgsler fra kunder.</p> <p>Oversigten skal inkludere relevant information om kunde, som navn, telefon nr, dato for oprettelse af ordre samt ordre titel.</p>
6	<p>Som en kunde hos Fog</p> <p>Vil jeg kunne bestille et skur til mit carport, som maks kan være 50% af længden og 50 eller 100% af bredden,</p> <p>Så jeg kan bestille en carport i de rigtige forhold.</p>	<p>Givet at kunden er på bestillingssiden</p> <p>Når kunden vælger et skur.</p> <p>Så kan kunden bestille en carport med skur til, med det rigtige forhold.</p>

7	<p>Som en sælger hos Fog,</p> <p>Vil jeg gerne kunne godkende en forespørgsel,</p> <p>Så jeg kan sende kunden en regning for det ønskede produkt.</p>	<p>Givet at sælgeren har gennemgået detaljerne for en specifik forespørgsel.</p> <p>Når sælgeren vælger at godkende forespørgslen, hvis den opfylder kravene.</p> <p>Så ændres status for forespørgslen til ”Godkendt” og en faktura sendes til kunden.</p>
8	<p>Som en sælger hos Fog,</p> <p>Vil jeg gerne kunne give rabat på en kundes ordre.</p>	<p>Givet at sælger er blevet kontaktperson på en ordre,</p> <p>Når sælger har gennemgået ordren, og givet rabat,</p> <p>Så kan sælger sende et tilbud.</p>
9	<p>Som en sælger hos Fog,</p> <p>Vil jeg gerne kunne ændre i kundens ordre,</p> <p>Så jeg kan tilpasse kundens egne valg efter at have kontaktet dem.</p>	<p>Givet at sælger er blevet kontaktperson på en ordre,</p> <p>Når sælger har gennemgået ordren med kunden, og ændret mål,</p> <p>Så kan sælger opdatere ordren.</p>

10	<p>Som en kunde hos Fog,</p> <p>Vil jeg modtage min faktura,</p> <p>Så jeg kan betale den.</p>	<p>Givet at min forespørgsel er blevet godkendt,</p> <p>Når jeg modtager en faktura,</p> <p>Så vil jeg betale for carporten.</p>
-----------	--	--

Efter en dialog med Fog har vi nu opnået en fælles forståelse og er begge tilfredse med ovenstående. Vi er nu klar til at begynde med opgaven.

I løbet af eksamensprojektet opdelte vi hver User Story i flere use cases for at opnå en dybere forståelse af, hvilke klasser og metoder der skulle implementeres i forbindelse med hver User Story. Vi strukturerede arbejdet ved at differentiere mellem frontend- og backend-opgaver. Gennem projektperioden arbejdede vi ofte i par, hvor én person fokuserede på frontend-udvikling og en anden på backend-udvikling. Dette samarbejde gjorde det muligt for os at integrere de to aspekter og opnå en holistisk forståelse af systemet.

For at opretholde overblikket over arbejdsprocessen benyttede vi os af et Kanban-board⁸ på GitHub. Dette board hjalp os med at visualisere, hvilke user stories der var under behandling, og hvilke der var færdiggjort. Vi formåede at fuldføre alle ti User Stories i projektet.

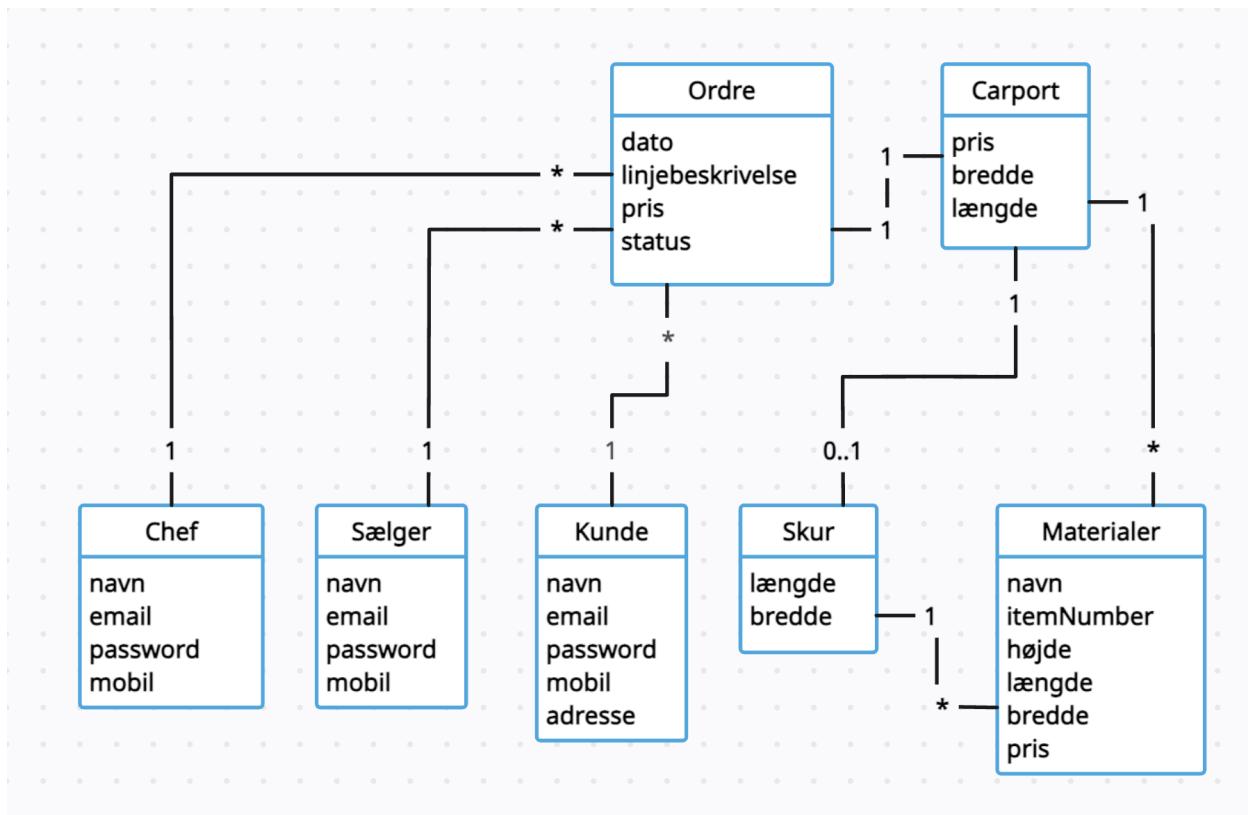
Det er værd at bemærke, at vi løbende tilføjede yderligere funktionaliteter til systemet uden at oprette nye User Stories for disse ændringer.

⁸ <https://github.com/users/babanoelk/projects/3>

Domænemodel

Domænemodellen for vores carport bestillingssystem er udformet ud fra de centrale entiteter og deres interaktioner inden for virkeligheden. Hver entitet repræsenterer en unik del af forretningen.

Her bruges der multipliciteten 0 og * samt forbindelserne er vist med en streg.



Kunde: Kunden er kernen i vores model. Denne entitet indeholder alle de nødvendige attributter om kunderne, såsom navn, e-mail, password, mobilnummer og adresse. Dette giver os mulighed for at tilbyde personaliserede oplevelser og kommunikere effektivt med kunderne.

Ordre: Ordre entiteten indeholder attributter om de individuelle bestillinger, herunder dato, linjebeskrivelse, pris og status på ordren. Denne entitet er nøglen til at spore kundens bestillinger fra start til slut.

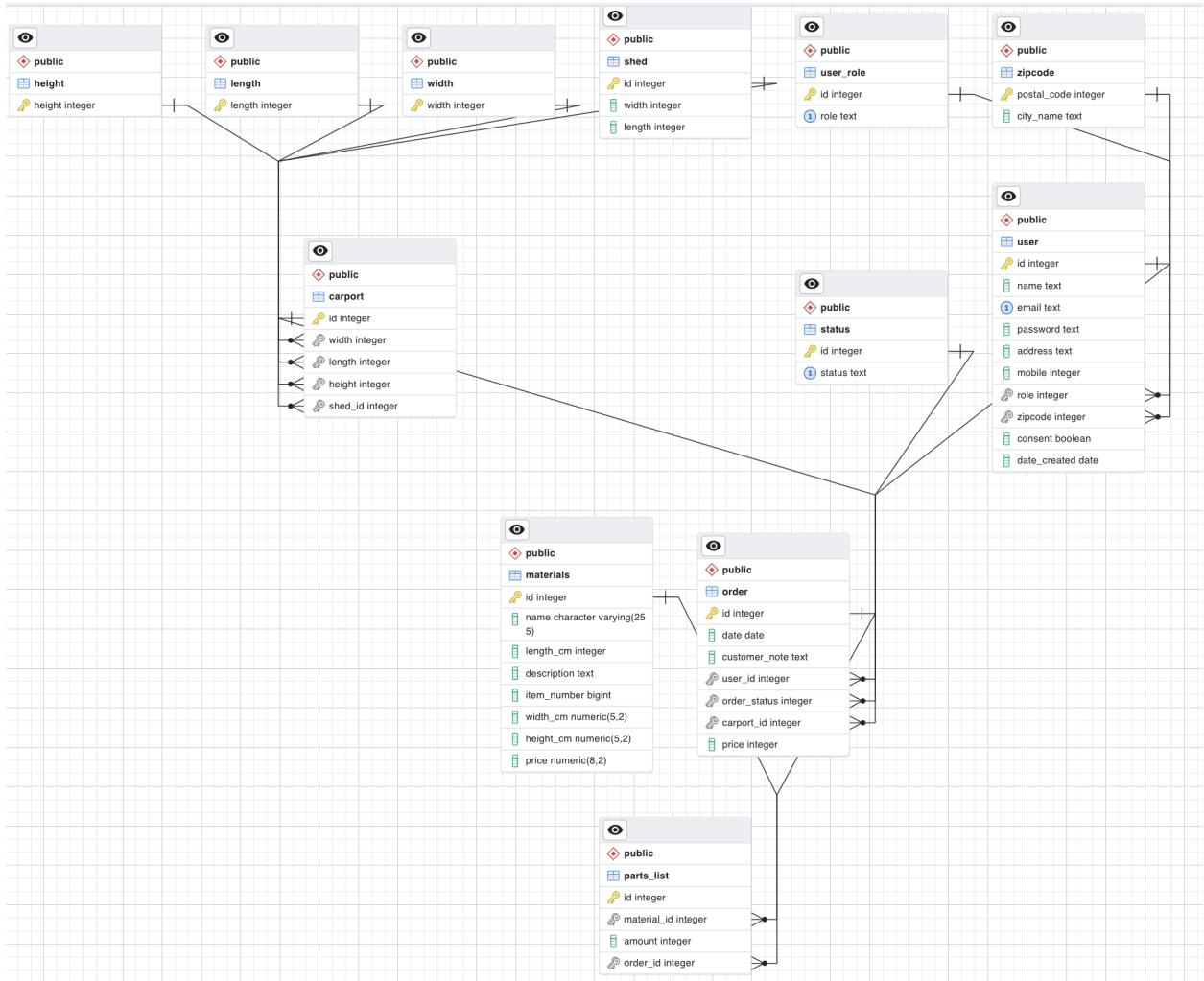
Carport: Carporten er et produkt, som kunden kan tilpasse efter behov. Entiteten omfatter attributten pris og dimensioner (bredde og længde), som kunden kan specificere.

Skur: I nogle tilfælde ønsker kunden at tilføje et skur til sin carport. Ønsker kunden et skur, vil man kunne vælge attributterne længde og bredde for dette tilvalg.

Materialer: Materiale entiteten repræsenterer de forskellige materialer, som carporten og skuret kan bestå af. Den indeholder attributter som navn, varenummer, dimensioner og pris, hvilket er afgørende for at beregne den samlede kostpris og administrere lagerbeholdningen. Dette er dog ikke muligt endnu, da prisen for en ordre beregnet baseres på carportens størrelse i km².

Salgs- og Chefroller: Salgs- og chefentiteterne er inkluderet for at repræsentere de forskellige brugerroller inden for systemet. Disse to roller kan i fremtiden have forskellige niveauer af adgang og funktionalitet. På nuværende tidspunkt har de samme adgang og funktionalitet. Rollerne giver dem mulighed for at administrere ordrer og kundeoplysninger.

Entity Relationship Diagram⁹



Billedet ovenfor illustrerer vores Entity-Relationship Diagram (ERD). For at forbedre overskueligheden og undgå tomme (null) attributter har vi valgt at opdele vores tabeller. Et eksempel på dette er opdelingen af carport og skur. Denne tilgang sikrer, at carporte, der er bestilt uden et skur, ikke indeholder informationer om et skur.

Derudover har vi oprettet tre yderligere tabeller: længde, bredde og højde. Disse tabeller indeholder forskellige mål, som en bruger kan vælge, når de angiver dimensionerne for deres

⁹ <https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/eerd/2nd%20EERD.png>

carport med eller uden skur. Hver af disse tre tabeller har sin egen primærnøgle, som vi bruger som fremmednøgler i tabellen Carport.

Vi har også taget højde for, at skurets mål afhænger af carportens længde og bredde. Dette er implementeret ved at begrænse skurets længde til 50% af carportens længde og skurets bredde til enten 50% eller 100% af carportens bredde.

Et eksempel på en uddybning af en af tabellerne. User tabellen ses forneden:

User	
Int ID	Dette er brugerens primærnøgle, som vi bruger til at referere til den specifikke bruger
Text name	Brugerens navn, som er en tekst
Text email	Brugerens e-mail er unik. Det vil sige, at ingen andre brugere kan have samme e-mail (vist med et et-tal med en blå cirkel) ¹⁰
Text password	Brugerens kodeord, som er en tekst
Text address	Brugerens adresse, som er en tekst
Int mobile	Brugerens mobilnummer, som er et heltal
Int role	Brugerens rolle er et heltal samt en fremmednøgle. Dette tal refererer til tabellen user_role, som fortæller systemet, hvilken rolle vedkommende har. Der findes tre roller: 1) Standard

¹⁰ <https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/eerd/2nd%20EERD.png>

	2) Sælger 3) Admin
Int zipcode	Brugerens postnummer er et heltal samt en fremmednøgle. Dette tal refererer til tabellen zipcode, som fortæller systemet, hvilket postnummer vedkommende bor i.
Boolean consent	Dette er vores måde at, hvor vi får en accept af brugeren på at vi må kontakte dem. Dette er en boolean.
Date date_created	Datoen på hvornår brugeren blev oprettet. Dette hjælper os med at overholde den europæiske lov GDPR i forhold til hvor lang tid vi må beholde brugerens oplysninger i systemet på.

Vi valgte at opdele både by- og postnummer i vores database som en del af vores normaliseringsproces. Dette trin var afgørende for at sikre, at vores ERD fulgte de tre første normalformer (NF).

Først og fremmest sørgede vi for, at ingen værdier blev gentaget eller kombineret (1.NF). Dernæst arbejdede vi på at eliminere afhængighed af delmængder af nøgler ved at sikre, at ingen kolonner er afhængige af blot en del af den primære nøgle (2.NF). Endelig sikrede vi, at ingen ikke primærnøgle kolonne var afhængig af en anden ikke primærnøgle kolonne (3.NF).

For at opnå dette opdelte vi postnummeret og byen. Denne strategi blev implementeret på tværs af alle vores tabeller, hvilket resulterede i en normaliseret databasestruktur, der opfyldte de tre første normalformer.

Projektstruktur

Mappestruktur

Kodebasen er struktureret i et lagdelt design, hvor hver del af applikationen er klart adskilt i henhold til MVC

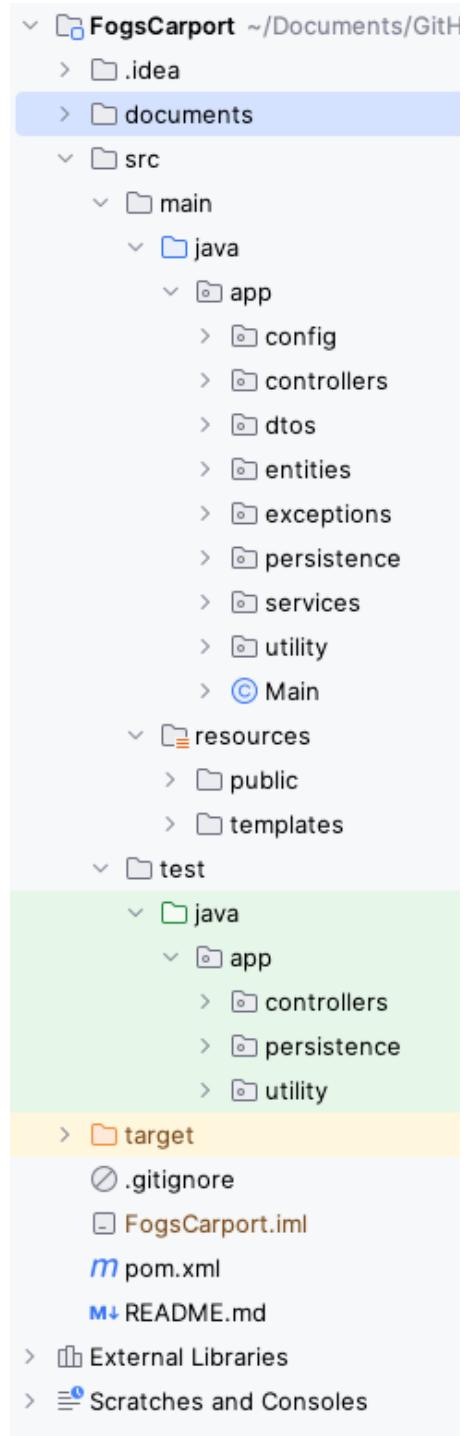
(Model-View-Controller) arkitektur mønstret. Designet er overskueligt og sikrer at udviklere ubesværet kan orientere sig i projektet. Vi har etableret mappen "app", der rummer backend-koden, og "resources"-mappen, som indeholder elementerne til projektets frontend.

"app" mappen har en uddybende pakkestruktur, der kategoriserer projektets kode efter logisk funktionalitet. Dette betyder, at der findes flere undermapper i "app", som er navngivet i overensstemmelse med klassernes roller eller funktioner.

For at give et klarere overblik over mappestrukturen, følger her en beskrivelse af, hvad hver enkelt mappe/package indeholder.

Config: Denne mappe indeholder udelukkende en klasse til at konfigurere Thymeleaf som en template engine for Java applikationen.

Controllers: Denne mappe indeholder klasser, der styrer modtagelsen og behandlingen af bruger anmodninger. De fungerer som mæglere mellem brugergrænsefladen og den underliggende applikations logik. Det vil sige, at disse klasser håndterer HTTP-anmodninger og dirigerer dem gennem de relevante forretningsprocesser.



Dtos: Forkortelsen for Data Transfer Objects: denne mappe indeholder klasser, der er designet til at bære data mellem processer. DTO'er bruges til at indkapsle data og sende det fra et systemlag til et andet.

Entities: Indeholder domæneobjekter eller entitetsklasser, som afspejler databasens tabeller og anvendes til at håndtere data i applikationens forretningslogik og datalag. Disse klasser repræsenterer de grundlæggende data konstruktioner, som applikationen arbejder med.

Exceptions: Denne mappe indeholder brugerdefinerede Exception-klasser, som håndterer særlige fejltilstande i applikationen. De brugerdefinerede undtagelser gør det muligt at reagere mere præcist på fejl, som kan opstå under kørsel af applikationen, især dem der er relateret til databasen.

Persistence: Denne mappe indeholder klasser, der er afgørende for interaktionen mellem applikationen og dens database. Det er her, alle operationer for at skabe (create), læse (read), opdatere (update) og slette (delete) data - almindeligvis kendt som CRUD-funktionerne - håndteres. Disse klasser sikrer, at data flyder korrekt mellem applikationen og databasen og understøtter således applikationens datalagring og datahentnings behov.

Services: Mappen indeholder klasser til at generere SVG (Scalable Vector Graphics), som kan bruges til at repræsentere grafiske billeder eller tegninger i applikationen.

Utility: Denne mappe indeholder hjælpeklasser eller værktøjer, som bruges på tværs af applikationen. Det drejer sig om klasser som kan udføre generelle beregninger, sende e-mails, eller andre hjælpefunktioner, som ikke direkte indgår i forretningslogikken eller datalaget.

Naming conventions

Java-koden følger camel case-konventionen, hvor hvert ord i metode-navne starter med et stort bogstav, undtagen det første, der starter med et lille bogstav. Et eksempel fra User-klassen inkluderer metoder som getName(), getEmail() og getRole(). Et mere komplekst eksempel er fra OrderController-klassen der inkluderer metoden discountPercentageOrAmount(Context ctx, ConnectionPool connectionPool).

HTML-koden er skrevet i lowercase og bruger bindestreger til at erstatte mellemrum mellem ordene. For eksempel er en HTML-side for en specifik ordre navngivet som ”admin-kd-ordre.html”. Et andet eksempel er knapper, hvor en knap til at gemme ændringer har en klasse som ”gem-knap-skur”. Denne konvention gør HTML-koden lettere at læse og skaber en mere brugervenlig opdeling mellem backend og frontend.

Klassediagram

Vi har opdateret vores klassediagram løbende gennem projektet. Til at starte med fik vi opdateret klassediagrammet dagligt, hvilket fungerede godt, da alle så kunne se hvilke klasser og metoder der var tilgængelige i systemet.

I løbet af projektet tabte vi den gode vane, og klassediagrammet blev ikke opdateret dagligt. Det resulterede i et par tilfælde med dobbeltarbejde, f.eks. to ens klasser el. at flere havde lavet den samme metode.

I rapporten har vi valgt at vise vores klassediagram fra tre forskellige stadier:

1. Det første klassediagram inden kodning
2. Klassediagrammet ved code freeze
3. Klassediagrammet efter refactoring af koden

Det giver et godt billede af hvor hurtigt koden udvikler sig, når man går i gang med at kode, og igen hvordan man til sidst kan skære en del fra igen.

Det sidste billede viser et komplet klassediagram med alle felter og metoder i hver klasse. Vi har valgt at lave vores klassediagram hvor det er opdelt i de packages, som klasserne er opdelt i. Det betyder at vi ikke viser direkte relationer mellem klasser i forskellige packages, men i stedet imellem de packages der er.

Vores klassediagram er af den grund mere overskueligt at se på, men giver heller ikke et helt præcist billede af relationen mellem klasserne.

Vi har haft et særligt fokus på at benytte os af DTO'er i dette projekt, for at gøre det nemmere at flytte data rundt i systemet.

Det første klassediagram¹¹ afspejler meget, at vi på det tidspunkt har vist hvilke tabeller vi har haft i databasen, samt hvilke entiteter der skulle være i systemet. Vi valgte fra start af at holde det småt, da vi af erfaring vidste, at der hurtigt ville komme nye klasser på efter behov.

Vores klassediagram ved code freeze¹² er meget omfattende. Der er blandt andet metoder, der ikke bliver brugt og overflødige klasser. Et resultat af at vi har haft fokus på at få lavet et system der fungerer, og at komme i mål med vores user stories. Derudover er alle relationer mellem packages og klasser i de forskellige packages ikke tegnet.

Efter refactoring har vi i vores endelige klassediagram¹³, fået skåret unødvendige metoder og klasser fra, samt refaktoreret kode til at være mere sigende og mere lean. Alle relationer er også tegnet op.

¹¹

<https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/class/initialClassDiagram.png>

¹²

<https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/class/classDiagramBeforeRefactoring.png>

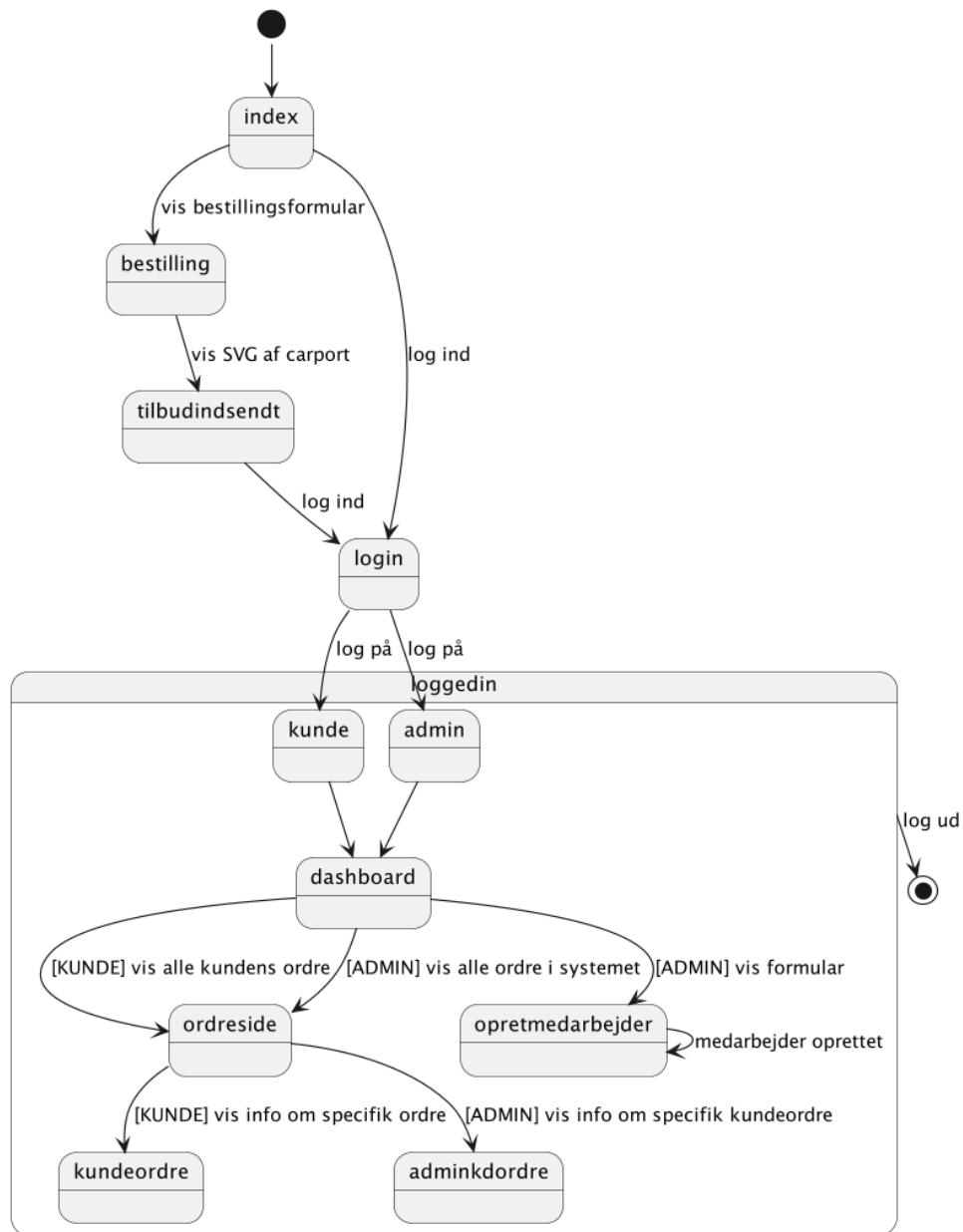
¹³

<https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/class/newFinalClassDiagram.png>

Navigationsdiagrammet

Nedenfor ses vores navigationsdiagram, der viser hvordan man kan navigere rundt på siden. Det viser ikke alle navigationsstier, men de mest oplagte, som systemet er lavet til.

Navigationsmenuen i systemet gør det muligt at logge ind eller logge ud fra alle sider og tilgå forsiden, samt bestillingssiden fra alle sider. Når man er logget ind som kunde, har man andre muligheder end når man er logget ind som en admin bruger.



Arbejdsproces

Vi har arbejdet agilt med SCRUM. Vi har ikke arbejdet 100% procent ud fra SCRUM teorien, men brugt udvalgte elementer. Det har vi gjort ved at udvælge en projektleder og holde daglige "stand-up meetings", samt lavet ugentlige mål internt i gruppen, som en form for sprints. Alle har haft mulighed for hurtigt at kunne oprette og opdatere opgaver på et Kanban-board, og vi har hurtigt kunne omstille os til at tilpasse systemet med nye el. anderledes funktionaliteter.

Med udgangspunkt i vores erfaring fra tidligere projekter, startede vi med at udpege en projektleder. Det har vi gjort for at sikre os at nogen har kunnet tage en beslutning, når der har været uenighed el. Diskussion.

Alle har fået lov til at komme med deres inputs, men i sidste ende har det været projektlederens beslutning. Det har gjort mange dele af vores arbejdsproces mere effektiv, om end ikke alle har været enige i beslutningen.

Projektlederen har også sørget for at der er blevet holdt fast i vores morgenbriefing og at alle hver dag har haft en opgave at arbejde med. Generelt har der været meget lidt spildtid pga. at vi har arbejdet med en projektleder i gruppen.

Daglig proces

Vi har haft en daglig arbejdsproces der har fulgt følgende skabelon:

1. Daglig briefing kl. ca. 9 med opsummering af gårdsdagen og planlægning af dagens opgaver.
2. Arbejde på den/de opgaver man har haft i løbet af formiddagen og tidlig eftermiddag.
3. Samling omkring kl. 12-14 med fokus på evt. udfordringer og hvad man kan bruge hjælp til.

Langt fra alle dage har været ens, en af de ting vi dog har holdt fast i dagligt, er vores morgenbriefing omkring kl. 9. Det har været med til at sikre en god arbejdsproces hele vejen.

Mange dage har dele af gruppen været på studiejob el. været syge, og nogle dage har været med arbejde hjemmefra, mens andre har været i samlet flok på skolens faciliteter.

Den første uge hvor vi lavede indledende analyser og User Stories, havde vi kun mulighed for at mødes online. Det skabte lidt udfordringer med at blive enige og færdige med den del af projektet. Samtidig var det hele nyt stof, og vi støtte oftere panden mod muren. Det var nok umuligt at undgå, men måden vi kom videre på, var ved at tage en snak om det i gruppen, og derefter blev vi straks mere effektive.

Opgavestyring

Vi har benyttet os af et Kanban-board¹⁴ til opgavestyring. Til dette har vi brugt GitHub Projects, hvor vi har arbejdet med kolonnerne:

- Tasks
- To do today (max. 5 opgaver)
- In progress (max. 5 opgaver)
- Done

Tasks, har vi brugt som backlog, hvor alle opgaver er blevet oprettet.

To do today, har kun omfattet opgaver der skal udføres i dag. Den har fået en begrænsning på max. 5 opgaver på en gang, hvilket også svarer til antallet af gruppemedlemmer i vores grupper.

In progress, har til hver en tid indeholdt opgaver, vi har været i gang med. Denne kolonne på Kanban boardet har også været begrænset til max. 5 opgaver - én pr. Gruppemedlem.

Done, der ligger alle færdiggjorte opgaver.

¹⁴ <https://github.com/users/babanoelk/projects/3>

Vi startede projektet med at opdele vores User Stories i Use Cases, og oprette hver Use Case som en opgave med en beskrivelse af hvad systemet skulle kunne når denne opgave var udført.

Senere ændrede vi det til at en opgave var mere specifik og indeholdt information om hvilket klasse og hvilke metoder der skulle laves. I sidste ende fik vi ikke brugt vores Kanban-board, og dagens opgaver blev i stedet aftalt på morgenbriefingen.

Det havde været en god idé at udpege en tovholder på Kanban-boardet, der kunne oprette alle opgaver der blev aftalt i fællesskab, og minde de øvrige medlemmer om at opdatere med nye opgaver, så havde vi nok holdt fast i brugen af opgavestyringen.

Det havde også været smart at bruge opgaverne fra Kanban-boardet til at oprette branches ud fra. Så ville vi have haft en klar sammenhæng mellem de stillede opgaver og branches. Til dette formål skulle vores opgaver også have været endnu mere specifikke, så der ikke var nogen tvivl om hvad man skulle lave.

GitHub branches

Vi har brugt GitHub som værktøj til versionsstyring. Her har vi arbejdet med en branch struktur der har set således ud:

- main-branch
 - development-branch
 - task/issue-branch

Planen med vores **main-branch** var at opdatere den hver uge, det var desværre ikke noget vi udførte i praksis.

Vores **development-branch** har vi hele tiden arbejdet ud fra, så vi først har samlet alle de nye funktionaliteter her, for efterfølgende at kunne merge dem ind i main. Development har været en protected branch, hvor minimum ét andet gruppemedlem skulle godkende en pull request.

Ud fra vores development-branch har vi lavet en ny branch ud fra en ny opgave. Vi har forsøgt at holde navngivningen for en branch forståelig, sådan at alle ved, hvad der er blevet arbejdet på, på

en given branch. Når opgaven er løst, er det seneste fra development først blevet merged ind i **task-branchen** for at sikre kompatibilitet, hvorefter der er blevet lavet en pull request til development, som skulle godkendes af minimum ét andet gruppemedlem.

Kommunikation

Til kommunikation har vi brugt Discord, hvor vi har en gruppe-server. Når vi ikke har mødtes fysisk, har vi brugt talekanaler på Discord, samt en tekst-kanal dedikeret til dette projekt. Vi har benyttet os af Code With Me igennem IntelliJ en del. Dette har gjort det nemmere at arbejde to sammen i en branch, samt vise eller hjælpe hinanden med kode når vi har arbejdet hjemmefra.

Vi har kommunikeret sammen dagligt, uanset om vi har mødtes eller ej.

I processen glemte vi nogle ting, der kunne have skabt klarere kommunikation i forhold til front-end (jf. afsnittet ‘Figma’) og navngivning i koden (jf. afsnit om ‘Projektstruktur’).

Særlige forhold

Når en kunde afgiver en bestilling på en carport, kræves det, at kunden opretter en brugerprofil i systemet til slut, under bestillingen. Dette muliggør senere adgang til at følge ordens status samt annullere orden ved behov. Kundens informationer gemmes i databasen ”fogs_carport” ved hjælp af følgende SQL-kode:

```
String sql = "INSERT INTO public.USER (name, email, password, address, mobile, zipcode, consent, role) values (?,?,?,?,?,?,?,?)";
```

Et unikt ID generes, som knytter brugeren til den specifikke carport, og oplysninger som valgte mål lagres.

Ved oprettelsen af brugeren på bestillingssiden er der implementeret betingelser, som skal opfyldes for adgangskoden. Disse inkluderer krav om både store og små bogstaver, tal og mindst et specialtegn. Kunden informeres øjeblikkeligt, hvis disse krav ikke opfyldes.

Validerings logik

For at forbedre systemets struktur og undgå gentagne valideringsmetoder kunne det overvejes oprette en utility-klasse kaldet ”Validator”. Denne klasse kunne indeholde metoder til validering af brugerinput, og den kunne bruges forskellige steder i systemet.

Især i OrderMapper klassen, hvor der er metoder til at modtage og validere input fra frontend, kunne det være hensigtsmæssigt at opdele lange metoder i separate valideringsmetoder.

Dette gælder også for administrationssiden (admin-kd-ordre.html), hvor den samme valideringsform anvendes gentagne gange, f.eks. ved ændringer af navn, adresse, telefonnummer, e-mail, carportens længde osv. Ved at implementere en mere modulær tilgang kunne man gøre systemet mere vedligeholdelsesvenligt og let forståeligt.

Session

I metoden updateOrderUser(Context ctx, ConnectionPool connectionPool)¹⁵ anvender vi ctx.sessionAttribute("user") til at hente brugeroplysninger fra sessionen. Først hentes det eksisterende brugerobjekt fra sessionen, og derefter opdateres brugeroplysningerne ved at sammenligne dem med de nye inputværdier, der er indsendt via formparametrene. Et nyt User-objekt oprettes med de opdaterede oplysninger, og dette objekt bruges til at opdatere brugeroplysningerne i databasen ved hjælp af UserMapper.updateUser metoden. Efter at have opdateret brugeroplysningerne i databasen, opdateres også brugerobjektet i sessionen ved at bruge ctx.sessionAttribute("user", oldUser). Dette sikrer at de opdaterede oplysninger er tilgængelige i sessionen og afspejles korrekt ved næste sidevisning.

Roller

Systemet har tre forskellige brugerroller: Standard, Sælger og Admin. Når en kunde opretter en bestilling, tildeler systemet automatisk brugerrollen "Standard" til kunden. Når kunden senere logger ind for at tjekke ordrestatus, bruger vores metode getAllOrders(Context ctx, ConnectionPool connectionPool) brugerens rolle til at bestemme, hvilke ordrer der skal vises. For en bruger med standardrollen vises kun egne ordrer.

Sælgerrollen har omfattende adgangsrettigheder. En sælger kan se alle ordrer i systemet, ændre ordrestatus og få adgang til alle kundeoplysninger. Dette inkluderer muligheden for at kontakte kunder, justere priser og udstede fakturaer. Det er vigtigt at bemærke, at sælgerrollen kun kan tildeles af adminrollen.

Adminrollen er foruddefineret fra starten og har de højeste adgangsrettigheder. Ud over at udføre alle handlinger, som sælgerrollen kan, har adminrollen også beføjelse til at oprette nye medarbejdere i systemet. Denne struktur sikrer, at de mest kritiske beføjelser er begrænset til adminrollen, og at andre brugerroller har mere begrænsede rettigheder baseret på deres ansvarsområde.

¹⁵ Bilag 16

Kode fremvisning

Vi har valgt og fremvise metoden `discountPercentageOrAmount(Context ctx, ConnectionPool connectionPool)`, som ligger i klassen `OrderController`. Denne metode håndterer rabatter og opdateringer af priser for en carport.

Først analyserer metoden indkommende formparametre såsom `orderID`, `total_price`, `discountPercentage` og `discountAmount`, som er afgørende for rabat operationerne. Disse fire værdier konverteres fra String værdier til numeriske værdier ved hjælp af `parseInt` og `parseFloat`. Derefter udføres en logik, der differentierer mellem rabatprocent og rabatmængde. Hvis en rabatprocent er specificeret, anvendes funktionen `Calculator.discountCalculatorPercentage`, for at trække den ønskede procent mængde fra. Hvis der er angivet en rabat mængde, anvendes i stedet funktionen `Calculator.discountCalculatorSubtraction`. Hvis ingen rabatter er angivet, forbliver den oprindelige pris uændret.

Efter beregningen af rabatter, bliver den opdaterede pris ændret i databasen ved hjælp af funktionen `OrderMapper.updateOrderPrice`.

For at håndtere mulige konverteringsfejl, fanges en `NumberFormatException`, der kan opstå ved konvertering af String til numerisk værdi.

Samlet set tjener denne metode som en nøglefunktion i systemets ordrebehandlings flow, der muliggør beregning af rabatter og opdatering af priser i overensstemmelse med de specificerede inputparametre.

```

public static void discountPercentageOrAmount(Context ctx, ConnectionPool connectionPool) throws DatabaseException {
    try {
        int order_ID = Integer.parseInt(ctx.formParam( key: "orderID"));
        float firstPrice = Float.parseFloat(ctx.formParam( key: "total_price"));
        String discountPercentageInput = ctx.formParam( key: "discountPercentage");
        String discountAmountInput = ctx.formParam( key: "discountAmount");

        float discountedPrice;
        if (discountPercentageInput != null && !discountPercentageInput.isEmpty()) {
            float discountPercentage = Float.parseFloat(discountPercentageInput);
            discountedPrice = Calculator.discountCalculatorPercentage(firstPrice, discountPercentage);
        } else if (discountAmountInput != null && !discountAmountInput.isEmpty()) {
            float discountAmount = Float.parseFloat(discountAmountInput);
            discountedPrice = Calculator.discountCalculatorSubtraction(firstPrice, discountAmount);
        } else {
            discountedPrice = firstPrice;
        }
        OrderMapper.updateOrderPrice(order_ID, discountedPrice, connectionPool);

        ctx.attribute("orderID", order_ID);
        ctx.sessionAttribute("totalPrice", discountedPrice);

        FormController.loadMeasurements(ctx, connectionPool);
        ctx.render( filePath: "admin-kd-ordre.html");

    } catch (NumberFormatException e) {
        e.printStackTrace();
        ctx.attribute("message", e.getMessage());
        ctx.render( filePath: "fejlsidé.html");
    }
}

```

I afsnittet om særlige forhold, kommer vi ind på gentagelse af kode i systemet, især vedrørende validering af brugerinput. Der opstår gentagne linjer kode, der udfører validering, og denne validerings logik anvendes også i metoden discountPercentageOrAmount(Context ctx, ConnectioPool connectionPool). På grund af aftalt code-freeze nåede vi desværre ikke at implementere en separat metode, som kunne kaldes hver gang, der var behov for at validere brugerinput. En sådan metode ville have gjort koden væsentligt mere simpel og overskuelig, samtidig med at den ville have elimineret gentagelsen af valideringslogikken.

Repository Pattern

Repository pattern er et populært mønster som benyttes til at strukturere adgang til datalaget.

I vores projekt har vi navngivet mappen for "persistence" som indeholder klasser der tilgår datalaget.

Repository har ansvar for at abstrahere adgang til databasen. Hver klasse har en række metoder som hver især har et formål: at udføre Create, Read, Update eller Delete (også kaldet CRUD) operationer på databasen.

.persistence

- > [CarportMapper](#)
- > [ConnectionPool](#)
- > [MaterialMapper](#)
- > [MeasurementMapper](#)
- > [OrderMapper](#)
- > [ShedMapper](#)
- > [StatusMapper](#)
- > [UserMapper](#)

Ved at bruge denne design pattern sikres at forretningslogikken og alle CRUD metoder adskilles således at forretningslogikken ikke har direkte adgang til databasen. Derudover gør det projektet nemmere at vedligeholde og at teste.

Figma

I begyndelsen af vores projekt brugte vi Figma til at designe en foreløbig skitse¹⁶ af vores hjemmeside. Figma er et effektivt værktøj til dette formål, da det tillader design af detaljerede layouts for forskellige enheder som laptops, iPads og iPhones. En af de store fordele ved Figma er dets evne til at understøtte samarbejde; det gør det muligt for hele teamet at arbejde sammen på designet og se ændringer i realtid.

Til trods for Figmas potentiale, bevægede vi os hurtigt videre fra designfasen til udviklingen af den første template (forsiden). For at få en konkret følelse af vores produkt. Efter dette punkt blev Figma sat til side, og vi vendte ikke tilbage til det og fortsatte med at designe hele webapplikationen¹⁷.

Set i bakspejlet ville en mere grundig brug af Figma have været fordelagtig. Vi kunne have udnyttet værktøjet til at eksperimentere med layout, farver og brugerfladens flow, og fastlægge designet for alle HTML-sider. Dette ville have gjort HTML- og CSS-udviklingsprocessen mere strømliniet og ensartet, med en klarere forståelse af, hvordan hver del af hjemmesiden skulle udføres.

¹⁶ Bilag 7 & 8

¹⁷ Bilag 9, 10, 11, 12, 13, 14 og 15

Test

For at sikre kvaliteten af vores software har vi implementeret en række tests, der tjener som et fundament for kvalitetssikringen af projektet. Vi har valgt at arbejde med test driven development på nøglekomponenter, som er kritiske for applikationens funktionalitet.

Vi har lavet unittests:

- UserControllerTest: En test klasse, der bekræfter vores brugers håndtering af anmodninger og sikrer, at korrekte responser genereres for varierede scenarier¹⁸.
- CalculatorTest: Sikrer at vores applikations kerne beregninger udføres præcist og effektivt¹⁹.

Vi har lavet integrationstest:

- UserMapperTest: Tester forbindelsen til databasen samt CRUD-operationerne, hvilket garanterer, at brugerdata håndteres korrekt²⁰.

Disse tests reflekterer et udpluk af vores tilgang til kvalitetssikring. Vi erkender, at en fuldstændig testdækning er ideel, men på grund af fokus på at komme i mål med en funktionsdygtig webapplikation og tidsbegrænsninger, har vi prioriteret test, der har indvirkning på brugeroplevelsen og systemets stabilitet. Vi planlægger at udvide vores testdækning i fremtiden for at omfatte yderligere moduler og scenarier, hvilket vil forbedre robustheden og kvaliteten af vores kodebase.

¹⁸ Bilag 4

¹⁹ Bilag 6

²⁰ Bilag 5

Konklusion

I løbet af dette eksamsensprojekt har vi demonstreret vores evne til at integrere teoretisk viden med praktisk anvendelse inden for fullstack webudvikling. Selvom vi ikke helt nåede alle de mål, vi havde sat os, lykkedes det os at realisere alle User Stories (US), hvilket resulterede i en funktionel og brugbar webapplikation.

Vi er lykkedes med at lave et mere automatiseret system med færre aktører. Blandt andet bliver kundens carport automatisk indsat i systemet, og sælger kan med det samme se omkostninger til materialer, samt en foreslået pris, der udregnes på baggrund af kvadratmeter.

Når kunden har indsendt en forespørgsel vises med det samme et billede af den bestilte carport ovenfra. Vi har også sørget for at kunden kan logge ind i systemet, efter at have indsendt en forespørgsel, så kunden kan få et overblik over sin ordre.

Det er muligt for en sælger el. admin bruger at ændre i alle kundens og ordrens oplysninger. Når forespørgslen er gennemgået, kan sælger med et klik fremsende faktura til betaling, og når betaling er registreret, fremsendes automatisk en stykliste.

Denne rejse har været en dyb læringsoplevelse, og vi ser frem til at bygge videre på de erfaringer og færdigheder, vi har erhvervet. Med udviklingen af denne carport webapplikation har vi opnået en grundig forståelse af, hvordan backend og frontend kan sammenkobles effektivt. Vi er ivrige efter at udvide vores viden, styrke vores færdigheder og udforske nye områder inden for webudvikling.

Afslutningsvis har projektet ikke kun styrket vores tekniske og analytiske færdigheder, men også givet os indsigt i den iterative natur af softwareudvikling. Denne proces, som understreger vigtigheden af løbende forbedring og tilpasning til skiftende brugerbehov, vil være uvurderlig i vores fremtidige projekter og professionelle bestræbelser. Vi ser med entusiasme frem til at anvende disse læringer i både fremtidige akademiske projekter og i den praktiske verden.

Kildehenvisninger

Opgavebeskrivelsen:

<https://cphbusiness.mrooms.net/mod/book/view.php?id=639642>

GitHub repository:

<https://github.com/babanoelk/FogsCarport>

Kanban-boardet:

<https://github.com/users/babanoelk/projects/3>

Studiekontraktten:

<https://github.com/babanoelk/FogsCarport/blob/main/documents/contracts/Study-Contract.docx>

Aktivitetsdiagrammer (AS-IS & TO-BE):

<https://github.com/babanoelk/FogsCarport/tree/main/documents/diagrams/activity>

Klassediagram:

<https://github.com/babanoelk/FogsCarport/tree/main/documents/diagrams/class>

<https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/class/initialClassDiagram.png>

<https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/class/classDiagramBeforeRefactoring.png>

<https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/class/newFinalClassDiagram.png>

Domænemodellen:

<https://github.com/babanoelk/FogsCarport/tree/main/documents/diagrams/domain>

ERD:

<https://github.com/babanoelk/FogsCarport/blob/main/documents/diagrams/eerd/2nd%20EERD.png>

Use-cases diagram

<https://github.com/babanoelk/FogsCarport/tree/main/documents/diagrams/use-cases>

PlantUML:

<https://plantuml.com/class-diagram>

<https://plantuml.com/activity-diagram-beta>

Videopræsentation af vores hjemmeside:

<https://www.youtube.com/watch?v=Y4nrjOmtcEY>

ChatGPT:

<https://chat.openai.com/>

Hvordan man sætter man e-mail delen op:

<https://github.com/dat2Cph/content/blob/main/sendgrid/sendgrid.md>

Hvordan laver man en SVG fil:

<https://github.com/dat2Cph/content/blob/main/svg/README.md>

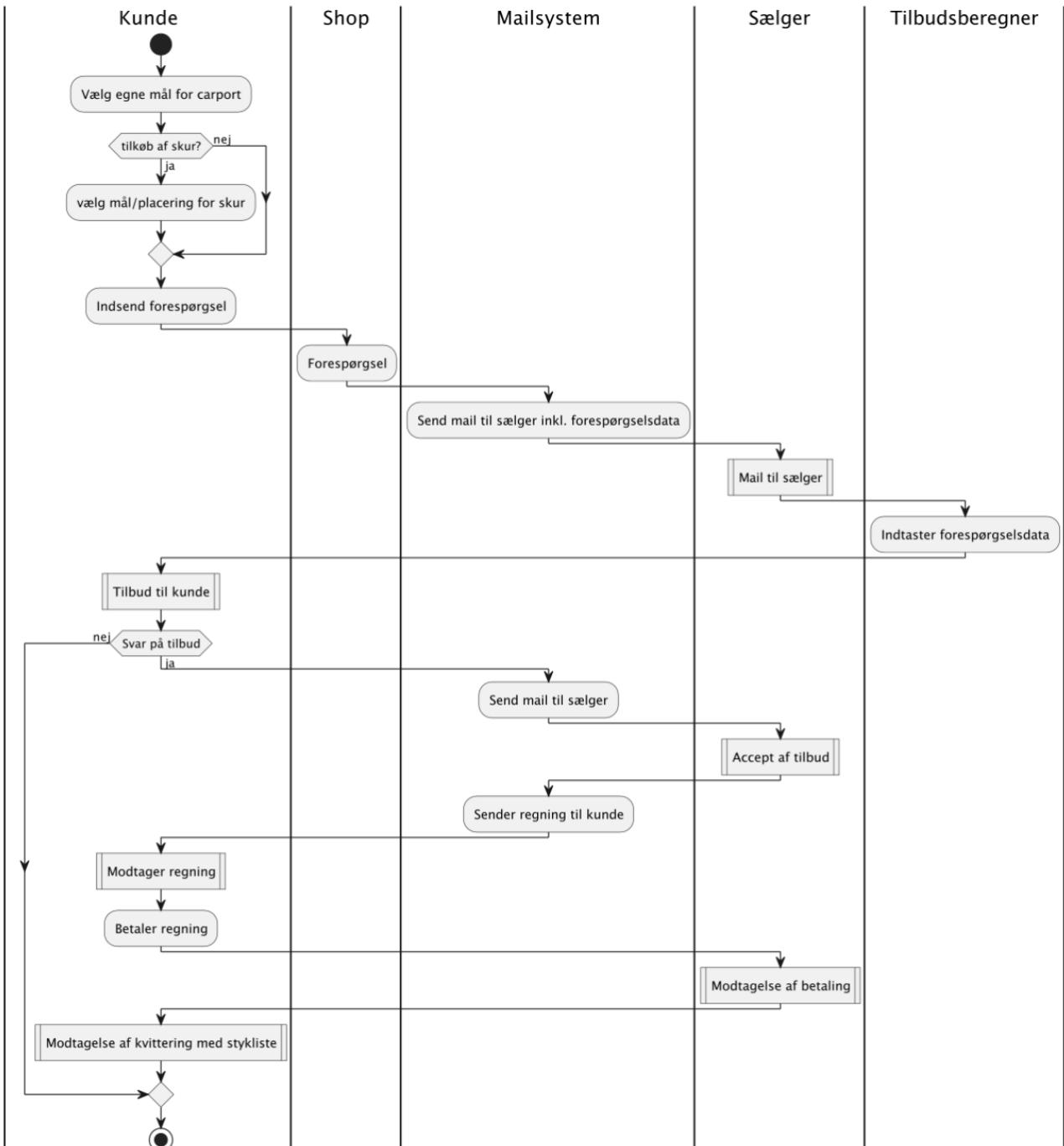
Vigtigt link i forhold til andre nyttige tutorials:

<https://github.com/dat2Cph/content/tree/main>

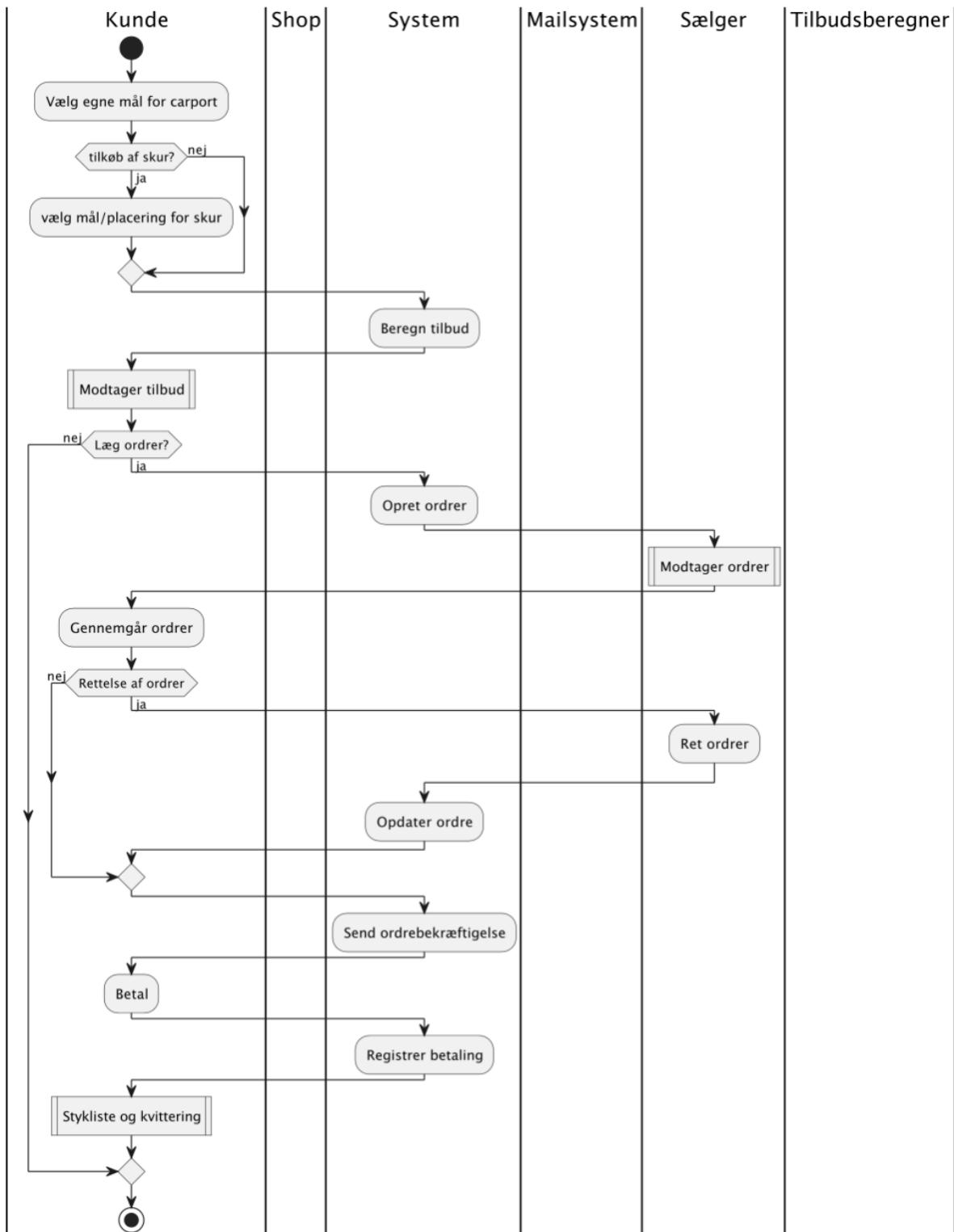
Bilag 1

Forebyggelse					
Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau	Plan for forebyggelse/afværgning
1	Sygdom hvor man kan arbejde hjemmefra eller fraværende kun 1 dag	Acceptabel	Muligt	Lav	Sikre mulighed for at arbejde hjemmefra med online kommunikation - Afgiver start og slut status/briefing omkring hvad man skal arbejde med og hvad man har udført - Benyt projektstyrings værktøj for at sikre overblik og struktur
2	Sygdom hvor man ikke kan rumme at arbejde hellere ej hjemmefra - sengeliggende	Uønsket	Usandsynlig	Medium	Implementer en sundheds agenda, hvor medarbejderne tilbydes sund kost og gratis motion på arbejdspladsen - Revurdering af mål i tilfælde af en længere sygdomsforløb - Aftal en anden deadline med kunde - Sikre en IT-partner som kan hjælpe med at løse opgaver
3	Elektronik som går i stykker	Uønsket	Usandsynlig	Medium	Sikre cloud løsninger (Backup)
4	Begynder fejl - opsætning af projekt fra start. Vi er ikke professionelle endnu og kan glemme nogle vigtige opsætninger	Uønsket	Sandsynligt	Høj	Søg hjælp ved hinanden når man er i tvivl. Sikre en overskuelig kanban overblik hvor opgaverne tydeliggøres - Programmering 2 og 2 front+backend
5	Opsigelser blandt teamet	Acceptabel	Usandsynlig	Lav	Åben og ærlig kommunikation blandt teamet. Gør det klart hvis man har tanker om at træde ud - Ugentlig dialog omkring hvordan man har det og om der er noget man har svært ved.
6	Opsigelse af kontakt person. Dvs. kontakt personen opsigter sin stilling	Uønsket	Muligt	Høj	Sikre et kontakt team fremfor udelukkende en kontaktperson hos partneren - Lav en kontrakt baseret på de funktionelle krav. Dermed sikres den fremadrettede arbejde trods en opsigelse ved partneren.
7	tekniske udfordringer i selve sammenkoblingen mellem backend og front end	Tolereres	Sandsynligt	Høj	Genbesøg Jons video 'er omkring sammenkoblingen - Sikre enighed omkring navngivning af attributter som forbinder front end og back end - Søg hjælp eksternt
8	Markedsændring. Hvis kundens kunder ønsker ex. garager fremfor carporte i højere grad	Acceptabel	Usandsynlig	Lav	Bygge et system som har et højt abstraktions niveau - Vær omstillingssparate som team
9	Det endelig produkt er anderledes end hvad kunden ønskede	Uacceptabelt	Sandsynligt	Ekstrem	afstem med kunde via figma - Lav Kontrakt på de aftalte funktionelle krav og design
10	Nye ønsker fra kunden ud over det aftalte	Acceptabel	Muligt	Lav	Ud over kundens ønsker bør vi se på projektet fra andre perspektiver og evt. komme med ideer / tilføjelser - måske skulle der laves et cykelstativ også??

Bilag 2



Bilag 3



Bilag 4

```
1 package app.controllers;
2
3 > import ...;
4
5
6     lassekhs
7
8     class UserControllerTest {
9
10
11         lassekhs
12
13     @BeforeEach
14     void setUp() {
15
16     }
17
18
19     @Test
20     void addAdminUser() {
21
22         String name = "name";
23         String address = "address";
24         int zip = 4000;
25         int phone = 23232323;
26         String email = "hej@lasse.dk";
27         String password = "1234";
28         boolean consent = true;
29         int role = 2;
30
31         User user = new User(name, email, password, address, phone, zip, consent, role);
32
33         assertEquals("Hans", user.getName());
34     }
35 }
```

Bilag 5

```
11 ▶ class UserMapperTest {
12
13     1 usage
14     private static final String USER = "notactive";
15     1 usage
16     private static final String PASSWORD = "notactive";
17     1 usage
18     private static final String URL = "jdbc:postgresql://localhost:5432/notactive";
19     1 usage
20     private static final String DB = "notactive";
21     no usages
22     private static String API_KEY = System.getenv( name: "SENDGRID_API_KEY");
23     2 usages
24     User user;
25
26     2 usages
27     User user1;
28     1 usage
29     private static final ConnectionPool connectionPool = ConnectionPool.getInstance(USER, PASSWORD, URL, DB);
30
31
32     ± badranyoussef
33     @BeforeEach
34     void setup() {
35         user = new User( name: "TestName", email: "test@testing.dk", password: "TestPassword", address: "TestAddress", mobile: 23232323, zipcode: 4000, consent: true, role: 1);
36     }
37
38     ± badranyoussef
39     @Test
40     void addUser() {
41         try {
42             user1 = UserMapper.addUser(user, connectionPool);
43         } catch (DatabaseException e) {
44             throw new RuntimeException(e);
45         }
46         assertEquals( expected: 10, user1.getId());
47     }
48 }
```

Bilag 6

```
10 ▶ class CalculatorTest {
11     4 usages
12     Carport carport;
13     1 usage
14     Shed shed;
15     ± badranyoussef
16     @BeforeEach
17     void setUp() {
18
19         carport = new Carport( id: 1, width: 200, length: 330, height: 300);
20         shed = new Shed( id: 1, carportID: 2, width: 100, length: 150);
21     }
22     ± badranyoussef
23     @Test
24     void carportPriceCalculator() {
25         float expectedPrice = 20000;
26         assertEquals(expectedPrice, Calculator.carportPriceCalculator(carport));
27     }
28     ± badranyoussef +1
29     @Test
30     void shedPriceCalculator() {
31         // Example test - replace with your actual method and expected value
32         float expectedPrice = 4000; // This should be the expected price for your shed
33         //assertEquals(expectedPrice, Calculator.shedPriceCalculator(shed));
34     }
35     ± badranyoussef +1
36     @Test
37     void carportPriceCalculator2() {
38         // Replace with a meaningful test
39         float expectedValue = 10; // This should be the expected value
40         //assertEquals(expectedValue, Calculator.carportPriceCalculator2(carport));
41     }
42     ± badranyoussef *
43     @Test
44     void piecesOfPost() {
45         assertEquals( expected: 10, Calculator.amountOfPost(carport));
46     }
47 }
```

Bilag 7

MacBook Air - 3

Bolig og design Bolig og design Bolig og design Bolig og design

Bestil Quick-Byg tilbud - carport med fladt tag

Carport med fladt tag
Udfyld nedenstående omhyggeligt og tryk "Bestil tilbud". Så vender vi hurtigst muligt tilbage med et tilbud til dig.

Carport bredde*

Carport længe*

Carport trapetztag*

RedskabsrumNB!
Der skal beregnes 15 cm tagudhæng på hver side af redskabsrummet*

Redskabsrum bredde*

Redskabsrum længde*

Evt. bemærkning / sælger ønskler

Kontaktoplysninger

Navn*

Adresse*

Postnummer*

By*

Telefon nr*

Email*

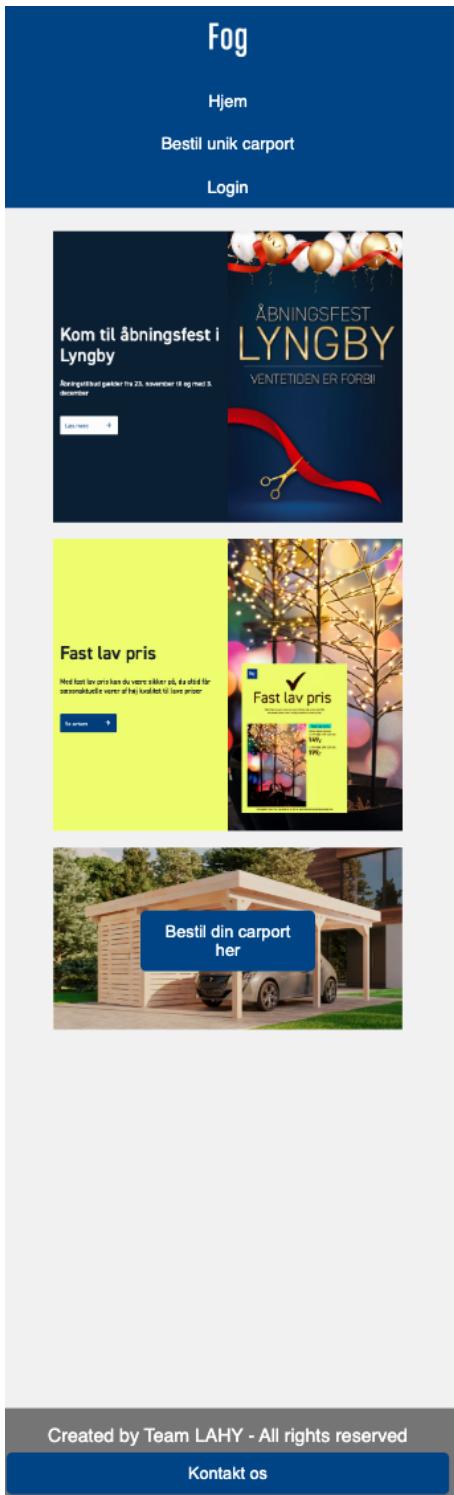
Password*

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

Bilag 8



Bilag 9



Bilag 10



Salgsordre	
Hans er logget ind	
Søg i ordre...	<button>Søg</button>
Filtrer efter status... ▾	
ORDRE ID	301
KUNDE NAVN	Youssef Badran
TLF NR	30117195
E-MAIL	b@b.dk
ORDRE DATO	2023-12-20
KOMMENTAR	
PRIS	20000.0
STATUS	Betalt
HANDLINGER	<button>Gem ændring</button> <button>Se ordre</button>
ORDRE ID	298
KUNDE NAVN	Hans
TLF NR	23456789
E-MAIL	morten.pedersen@email.com
ORDRE DATO	2023-12-19
KOMMENTAR	
PRIS	3600.0
STATUS	Betalt
HANDLINGER	<button>Gem ændring</button> <button>Se ordre</button>
ORDRE ID	306
KUNDE NAVN	jkanfajknf
TLF NR	12345678
E-MAIL	authia11e@live.dk

Bilag 11

Varelager oversigt									
Her kan du tilføj og slette vare på varelageret.									
Tilføj vare									
Vare navn	Længde(cm)	Beskrivelse	Varer nummer	Bredde(cm)	Højde(cm)	Pris(dkk)	tilføj		
ID	Vare navn	Beskrivelse		Længde(cm)	Bredde(cm)	Højde(cm)	Varer nummer	Pris(dkk)	Handling
1	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		300	4	14	14380471500300	105	<button>Delete</button>
2	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		330	4	14	14380471500330	126	<button>Delete</button>
3	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		360	4	14	14380471500360	151	<button>Delete</button>
4	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		390	4	14	14380471500390	181	<button>Delete</button>
5	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		420	4	14	14380471500420	217	<button>Delete</button>
6	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		450	4	14	14380471500450	261	<button>Delete</button>
7	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		480	4	14	14380471500480	313	<button>Delete</button>
8	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		510	4	14	14380471500510	376	<button>Delete</button>
9	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		540	4	14	14380471500540	451	<button>Delete</button>
10	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		570	4	14	14380471500570	541	<button>Delete</button>
11	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		600	4	14	14380471500600	650	<button>Delete</button>
12	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		630	4	14	14380471500630	780	<button>Delete</button>
13	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		660	4	14	14380471500660	936	<button>Delete</button>
14	Spærtræ	Spærtræ anvendes primært som en strukturel komponent i tagkonstruktioner for at give støtte og stabilitet til taget. Det bruges til at danne den bærende ramme i taget og understøtte tagdækningen.		690	4	14	14380471500690	1123	<button>Delete</button>

Bilag 12

The screenshot shows a web application interface. At the top left is a logo with the word 'Fog'. To the right are four small blue buttons: 'Hjem', 'Bestil unik carport', 'morten.pedersen@email.com', and 'Log ud'. Below this is a dark grey header bar with the word 'Dashboard' in white. Underneath the header are four dark blue rectangular buttons, each containing white text: 'Se alle ordre', 'Varelager', 'Opret medarbejder', and 'Logud'. At the bottom of the page is a dark grey footer bar with the text 'Created by Team LAHY - All rights reserved' on the left and a 'Kontakt os' button on the right.

Bilag 13

The screenshot shows a login form for a website called 'Fog'. At the top left is the 'Fog' logo. At the top right are three buttons: 'Hjem', 'Bestill unik carport', and 'Login'. Below these is a 'Login' heading. The main area contains two input fields: 'Email' and 'password'. Underneath the password field is a blue button labeled 'Vis/Skjul kode'. Below that is another blue button labeled 'Log på'. At the bottom of the page, a dark footer bar contains the text 'Created by Team LAHY - All rights reserved' on the left and a 'Kontakt os' button on the right.

Bilag 14

The screenshot shows a web page for ordering a carport. At the top, there is a blue header bar with the word "Fog" on the left and three buttons on the right: "Hjem", "Bestil unik carport", and "Login". Below the header, the title "Bestil Quick-Byg tilbud- carport med fladt tag" is displayed. The form consists of several input fields and sections:

- Carport bredde***: A dropdown menu showing "240cm".
- Carport længde***: A dropdown menu showing "240cm".
- Carport højde***: A dropdown menu showing "230cm - Kan kun bestilles med denne højde".
- Carport trapetztag***: A dropdown menu showing "Standart tag".
- RedskabsrumNB!**: A note stating "Der skal beregnes 15cm tagudhæng på hver side a redskabsrummet".
- Ønsker du et redskabsrum:** Two radio buttons: "ja" and "nej".
- Evt. bemærkninger**: An empty text area.
- Kontakt Oplysninger**:
 - Navn***: An empty text area.
 - Adresse***: An empty text area.
 - Postnummer***: An empty text area.
 - Telefon nr***: An empty text area.

Bilag 15

Fog

Hjem Bestil unik carport Login

Kom til åbningsfest i Lyngby
ÅBNINGSFEST LYNGBY VENTETIDEN ER FORBI!
Åbningstilbud gælder fra 23. november til og med 3. december
Læs mere →

Fast lav pris
Med fast lav pris kan du være sikker på, du altid får sæsonaktuelle varer af høj kvalitet til lave priser
Se avisen →

Created by Team LAHY - All rights reserved Kontakt os

Bilag 16

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

    String updateName;
    String updateAddress;
    int updateZipcode;
    int updateMobile;
    String updateEmail;

    //The chosen users editable information page
    User user = ctx.sessionAttribute(key: "user");

    User newUser = new User(user.getId(), updateName, updateEmail, updateAddress, updateMobile, updateZipcode);

    //Opdater brugere oplysningerne
    try {
        UserMapper.updateUser(newUser, connectionPool);
    } catch (DatabaseException e) {
        ctx.attribute("message", "error updating user" + e.getMessage());
        ctx.render(filePath: "admin-kd-ordre.html");
    }

    int orderId = Integer.parseInt(ctx.formParam(key: "orderID"));

    ctx.attribute("orderID", orderId);

    User oldUser = null;
    try {
        oldUser = OrderMapper.getOrderDetails(orderId, connectionPool);
    } catch (DatabaseException e) {
        ctx.attribute("message", "error loading order" + e.getMessage());
        ctx.render(filePath: "admin-kd-ordre.html");
    }
    ctx.sessionAttribute("user", oldUser);
    FormController.loadMeasurements(ctx, connectionPool);

    //Load the page
    ctx.render(filePath: "admin-kd-ordre.html");
}
```