

# Carport hjemmeside rapport

The logo consists of the word "Fog" in a large, white, sans-serif font. A registered trademark symbol (®) is positioned at the top right of the letter "g". The background is a solid dark blue.

## Gruppemedlemmer

Masih Bijan Kabiri, cph-mk330@cphbusiness.dk

Tobias Welling Hansen, cph-th401@cphbusiness.dk

Peter Janas, cph-pj282@cphbusiness.dk

Henrik Thunbo Jensen, cph-hj286@cphbusiness.dk

Umair Tafail, cph-ut38@cphbusiness.dk

Datamatiker 2.Semester, Klasse B

22. April 2024 - 24. Maj 2024

# Indholdsfortegnelse

<b>Indholdsfortegnelse.....</b>	<b>2</b>
<b>Links.....</b>	<b>3</b>
<b>Indledning.....</b>	<b>4</b>
<b>Baggrund.....</b>	<b>5</b>
<b>Virksomheden/Forretningsforståelse.....</b>	<b>6</b>
Teknologivalg.....	6
Risikoanalyse.....	7
Interessentanalyse.....	8
Krav.....	10
User stories.....	12
Domæne model og ER diagram.....	13
Navigationsdiagram & Mockups.....	16
<b>Valg af arkitektur.....</b>	<b>18</b>
<b>Særlige forhold.....</b>	<b>20</b>
Informationer der gemmes i session og request scope.....	20
Sikkerhed ift. login.....	21
Kriterier til brugeroprettelse.....	21
Automatisk stort bogstav.....	21
Håndtering af exceptions.....	23
Svg tegning.....	24
Stykliste beregner.....	25
<b>Udvalgte kodeeksempler.....</b>	<b>27</b>
Røde felter.....	27
Toggle Password.....	28
Gemme input i session.....	29
<b>Status på implementering.....</b>	<b>30</b>
Implementering af websider.....	30
Implementering CRUD-metoder.....	30
Styling.....	30
Test.....	31
SVG.....	31
Stykliste beregner.....	31
<b>Kvalitetssikring (test).....</b>	<b>33</b>
Unit Testing.....	33
Integrationstest.....	33
Teststrategi.....	35
User Acceptance tests.....	37
<b>Proces.....</b>	<b>38</b>
Arbejdsprocessen faktuelt.....	38
Arbejdsprocessen reflekteret.....	40
<b>Bilag.....</b>	<b>42</b>

Diagrammer.....	42
Logbog.....	47

## Links

Link til vores Github projekt repository -

<https://github.com/TobiOneCanobi/SemesterProjekt-B8>

Link til video demo af vores program -

<https://www.youtube.com/watch?v=YRNq-3sBFH0>

Link til deployed website af vores program -

**Login navn + kodeord til en demo admin-bruger**

Email: admin

Password: test

# Indledning

Vi har modtaget en opgave fra Fog, en virksomhed, der sælger carporte, om at udvikle et nyt website for dem. Deres nuværende website har flere problemer, som vi skal løse ved at starte helt fra bunden.

For at skabe en klar struktur og et godt overblik over projektet, begynder vi med at udarbejde forskellige diagrammer og modeller. Vi laver også et mockup af websitet, som vil fungere som en visuel guide og hjælpe os med at planlægge, hvad der skal laves, og hvordan det skal gøres.

Efter planlægningsfasen opretter vi en database, som vil blive brugt til løbende at teste websitet. Vi tilføjer gradvist flere funktioner til websitet, indtil vi har opnået det endelige produkt, som vi har planlagt.

Arbejdet med kodningen fordeles i teamet ved at tildele mindre opgaver ('tasks') til hvert medlem. Når en opgave er afsluttet, vælger vi en ny opgave baseret på, hvad der giver mening i forhold til projektets fremdrift. Nogle dele af koden afhænger af, at andre dele er færdige, så vi prioriterer opgaverne derefter.

For de større og mere komplekse opgaver anvender vi 'pair programming', hvor to teammedlemmer arbejder sammen foran en computer. Dette sikrer, at vi kan løse de svære opgaver korrekt og effektivt.

# Baggrund

Fog Trælast & Byggecenter er en virksomhed inden for byggematerialer og -værktøjer, der betjener både professionelle håndværkere og private kunder. De tilbyder et bredt udvalg af produkter såsom træ, værktøj, maling, VVS-udstyr, og meget mere, der er nødvendige for byggeprojekter og renoveringer. Fog Trælast & Byggecenter fokuserer på at levere kvalitetsprodukter og professionel rådgivning for at imødekomme kundernes behov og sikre succesfulde byggeprojekter. Deres mål er at være en pålidelig partner for alle, der er involveret i byggebranchen, og de stræber efter at opretholde et højt niveau af kundetilfredshed gennem deres service og produkter.

I dette afsnit præsenterer vi de overordnede krav, som kunden har til et nyt system som vi laver, der tager sig af bestilling af carporte. For at udforme en præcis og brugervenlig løsning har vi analyseret en video præsenteret af kunden, som detaljeret beskriver deres forventninger og behov. Denne video har været afgørende for at forstå kundens ønsker og krav til systemets funktionalitet og brugervenlighed uden at gå i detaljer med den tekniske implementering.

Kravene fokuserer på at skabe en intuitiv og effektiv bestillingsproces samt et effektivt lagersystem til nem administration af materialer. Kunden skal kunne bestille en carport via en brugervenlig bestillingsproces. Systemet skal tilbyde en funktion, hvor kunden kan generere en forhåndsvisning af carporten. Denne forhåndsvisning skal visuelt præsentere produktets dimensioner og design, så kunden får en klar forståelse af, hvordan den færdige carport vil se ud.

Derudover skal vores kunde, altså Fog selv, nemt kunne administrere deres lagersystem. Dette inkluderer muligheden for ubesværet at tilføje nye materialer, slette eksisterende materialer og ændre detaljerede oplysninger om materialerne. Systemet skal være intuitivt og brugervenligt, så ændringer kan foretages hurtigt og præcist uden behov for omfattende teknisk viden.

Kunden skal også have adgang til en overskuelig liste over alle ordrer, der er placeret i systemet. Denne liste skal indeholde relevante købsoplysninger, såsom kundens navn, ordredato og produktspecifikationer, samt nuværende status på hver ordre. Funktionen skal gøre det nemt for kunden at holde styr på deres ordrebehandling og opfølgning.

Efter kunden har indsendt en forespørgsel, skal deres ønsker registreres i systemet. Virksomheden skal kunne behandle disse forespørgsler ved at indtaste og administrere detaljerne i deres interne system. Kunden skal modtage en pris estimation baseret på deres specifikationer for produktet. Den detaljerede stykliste, som indeholder de nøjagtige komponenter og materialer, skal forblive en del af virksomhedens interne dokumentation og ikke deles med kunden før de har betalt.

Sammenfattende skal det nye system være både brugervenligt og effektivt, hvilket indebærer en intuitiv bestillingsproces, enkel lageradministration, en overskuelig ordreoversigt og præcise pris estimationer.

# Virksomheden/Forretningsforståelse

## Teknologivalg

Teknologi	Version
IntelliJ IntelliJ IDEA	2024.1.1
Postgresql	42.7.2
PgAdmin 4	8.9
Docker Desktop	4.28.0
Svg	1.1
Thymeleaf	3.1.2
Javalin	6.1.3
Java	21.0.2
Hikariconnectionpool	5.1.0
JUnit	5.10.2
PlantUML	7.10.1-IJ2023.2

# Risikoanalyse

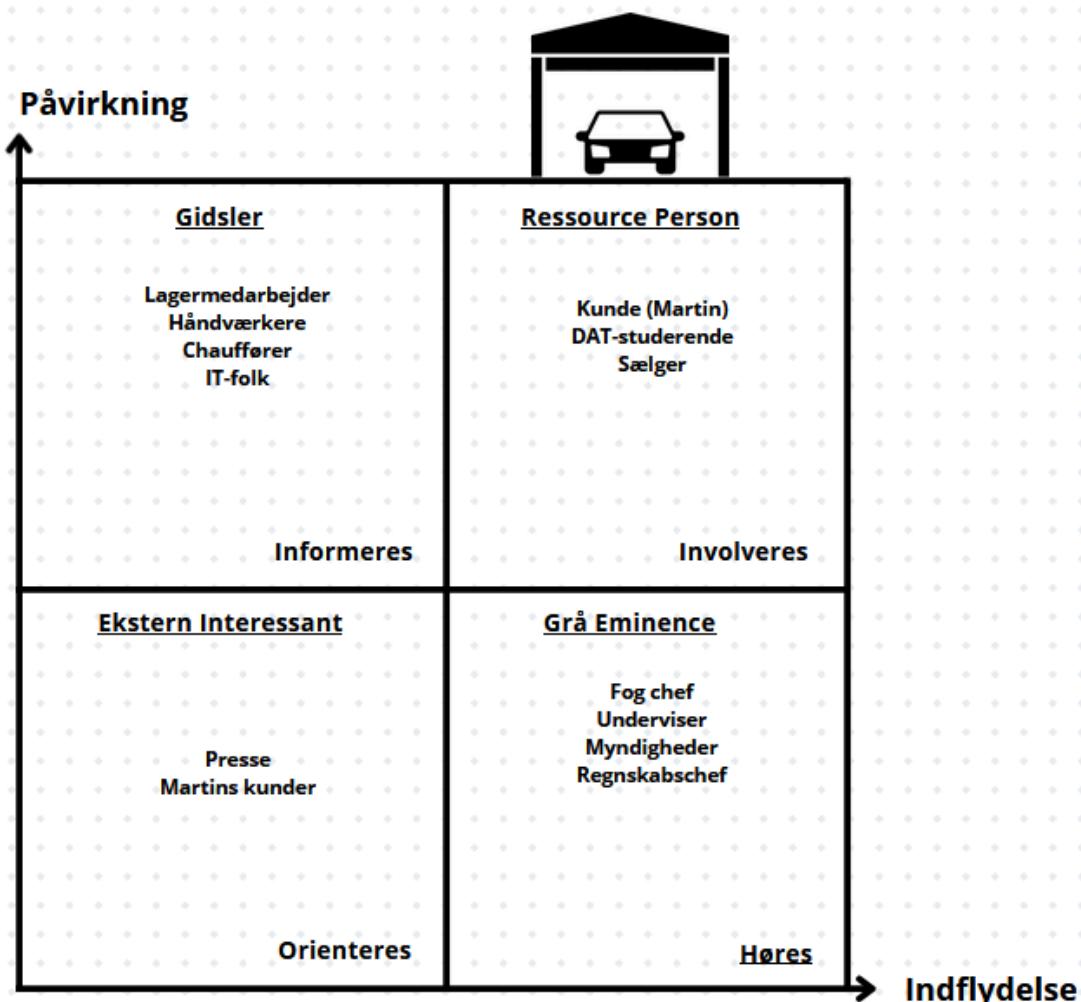
## Risikoanalyse

Semester projekt For FOG

Forebyggelse					
Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau	Plan for forebyggelse/afværgning
1	Sygdom i udviklingsteamet	Tolereres	Sandsynligt	Medium	Vi tager vores vitaminer, vacciner og holder afstand fra syge folk
2	Det regner	Uønsket	Sandsynligt	Medium	Vi tager den online istedet
3	Uhensigtsmæssig arbejde	Uønsket	Muligt	Medium	Tager kage med
4	Firmaet går konkurs	Uacceptabelt	Usandsynlig	Ekstrem	Håndtere det kontrakt mæssigt
5	Når ikke deadline	Uønsket	Muligt	Høj	Optimering af workflow, realistisk deadline
6	Kunde trækker sig tilbage	Uønsket	Usandsynlig	Høj	Kræve stykvis eller startbetaling
7	Intern konflikt	Uacceptabelt	Muligt	Høj	Konflikthåndterings-trappemodellen
8	Hardware svigt	Uønsket	Muligt	Høj	Lave hyppige backup
9	Underviser/vejleder utilængselig	Uønsket	Muligt	Høj	Aftale med vejleder nogen tidspunkter
10	Sælger som boycotter	Uønsket	Usandsynlig	Medium	Ikke gøre dem sure
11	Kunder som boycotter	Uønsket	Usandsynlig	Medium	Give kunder det de ønsker
12	FOG repræsentant skrifter job/	Uønsket	Muligt	Medium	Giv en lønforhøjelse

Som gruppe har vi i vores team snakket om nogle mulige risici, der kunne være, for kvaliteten af projektets udførelse vil falde. Vi har været enige om, hvordan disse risici skulle håndteres. Vi har formået ikke at have haft nogen merge konflikter, da vi har aftalt, at der skal være en konsekvens, hvis en person laver en fejl i merging af brances og ved en fejl kommer til at slette noget andre har lavet. I vores tilfælde var konsekvensen, at man så skulle give resten af holdet kage.

# Interessentanalyse



Interessentanalysen viser forholdet mellem påvirkningen og indflydelse af de enkelte deltagere i projektet.

Den giver et overblik over, at forstå forskellige roller i projektet, forbedre kommunikation og samarbejde samt identificere risici og sikre et godt arbejdsmiljø. I vores projekt har vi lavet den med det udgangspunkt, at vi forestiller os som værende ansatte i en virksomhed, men samtidig også at vi er elever på en skole. Her er en gennemgang af de forskellige roller, og hvad det har kunne give os.

## Gidsler

Gidsler kaldes de som har en høj påvirkning, men lav indflydelse. Det er typisk dem der udfører den del af arbejdet som rent effektivt får servicen til at blive fuldendt. Kendskab til gidsler i projektet kan forbedre projekt-flowet ved at sikre, at deres behov og udfordringer bliver fremvist, hvilket øger effektiviteten. Det fremmer bedre kommunikation og samarbejde mellem teammedlemmerne, hvilket reducerer misforståelser. Anerkendelse af gidslernes bidrag kan øge deres motivation og engagement.

### **Ressourceperson**

En person med høj påvirkning, men også høj indflydelse på hvad og hvordan projektet skal udføres, er en ressourceperson. De involveres for at sætte rammerne for hvordan projektet udføres. F.eks., hvordan carporten bygges, om der er nogle ting at tage særligt forhold til, om nogle dele ikke kan leveres til lager og alternativer derfor skal bruges, osv. Martin fra Fog har i dette tilfælde taget del i denne opgave, da det er videoen hvor vores lærer Jon besøger ham, som vi har kravene til produktet fra. Videoen har fungeret som alfa-omega for, hvad vores løsning skal være og hvad det skal indeholde. Men det er mere videoen i sig selv, og ikke diagrammets skabelse, der har hjulpet os.

Vi er også selv som datamatiker-studerende ressourcepersoner som har en klar påvirkning på byggelsen af projektet, da vi står for hjemmesiden. Tilmed også noget indflydelse, da vi skal kommunikere lærerne, og bruge kunden Martins krav om, hvordan tingene skal være.

### **Grå eminence**

Grå eminence er øverst i hierarkiet. De kan tage vigtige beslutninger som ikke påvirker projektet direkte, men ændrer virksomhedsstrategi, værdier, eller andet der kan ændre udgangspunktet for alle andre der er deltagere i projektet. I dette projekt som værende et skoleprojekt, har læreren skullet give vejledning og sætte rammer for hvad vi skal lave. De har i det hele taget hjulpet os med at lave projektet, med deres vejledning.

### **Ekstern interessant**

En ekstern interessant er en som kan påvirke projektet eller virksomhedens tilstand, gennem omtale og anmeldelser, men det betyder ikke noget i vores tilfælde.

## Krav

### AS-IS

#### Se bilag 1

Som Fogs system var ved projektets start, var der en besværlig proces ift. bestilling af carport både for kunden og Fog. En kunde skulle vælge sit mål og indsende en forespørgsel. Den forespørgsel havnede ved et mailsystem, som genererede en mail til Fog selv.

Fog skulle tage de oplysninger fra mailen og selv indtaste dem i et andet system, hvor der blev lavet et tilbud. Så skulle Fog selv maile tilbuddet til kunden, hvor kunden så kan afslå, omformulere ønske eller acceptere.

En anden ting var, at hvis Fog fik nye materialer til at bygge med, kunne de ikke indtaste dem i deres lagersystem. De skulle i stedet for lægge materialet ind som et andet materiales navn og selv huske på og tælle forskellen på beholdningen mellem de to materialer. De kunne kun tilpasse prisen på tilbuddet som systemet gav dem, i tilfælde af at de vil hæve eller sænke prisen af forskellige årsager. Til sidst skal det nævnes, at kunden først vil få adgang til styklisten, når de har betalt.

### TO-BE

#### Se bilag 2

I det system vi udvikler til Fog skal vi løse ovenstående problemer på en måde der er hensigtsmæssig ift. tidsforbrug og antal af trin, for både kunde og Fog.

Vi vil designe et system hvor kunden udfylder sine ønsker om en carport på den nye hjemmeside, hvorefter det bliver oprettet som en sag med status som kan ændre sig.

Et beregningssystem laver en pris og en stykliste til brugerne, og ændrer korrekt på varebeholdningen. På hjemmesiden kan Fog se prisen, tilpasse den efter behov og sende et endeligt bud. De kan også slette den.

Det kunne f.eks være hvis brugerne ombestemmer sig og vil designe en carport forfra. Fog sender tilbuddet til kunden, som de får gennem en autogenereret mail. Inde på hjemmesiden kan brugerne tjekke status for deres carport under hele forløbet.

## Aktivitetsdiagram TO/BE Efter

#### Se bilag 3

Efter projektets afslutning har vi valgt at opdatere aktivitetsdiagrammet. Processen starter med, at kunden vælger sine egne mål og sender en forespørgsel. Denne forespørgsel omdannes derefter til en ordre og sendes videre til sælgeren. Sælgeren har to muligheder: enten at afvise orden, hvilket afslutter programmet, eller at acceptere den. Hvis orden accepteres, sendes den videre til tilbuds beregneren.

Sælgeren kan også vælge at ændre eller slette orden, hvis kunden for eksempel ombestemmer sig. Tilbuds-beregneren udarbejder en stykliste og et tilbud, men sender kun tilbuddet videre til kunden. Kunden modtager tilbuddet og kan vælge enten at afslå det, hvilket afslutter programmet, eller acceptere det, hvorefter de betaler og modtager en stykliste samt betalingsinformation.

I denne version har vi valgt ikke at inkludere et mailsystem. Tilbuds-beregneren giver direkte et tilbud til kunden. Vi ændrede forespørgsler til et ordresystem, fordi der oprettes en ordre, så snart kunden sender en forespørgsel, og det er ordren, der opdateres, når der sker noget nyt i programmet.

# User stories

## Tasks

Vi har taget vores user stories og brudt dem ned i mindre tasks, som vi har udført. Deres status har vi behandlet på vores kanban-board.

Adresse til kanban-board:

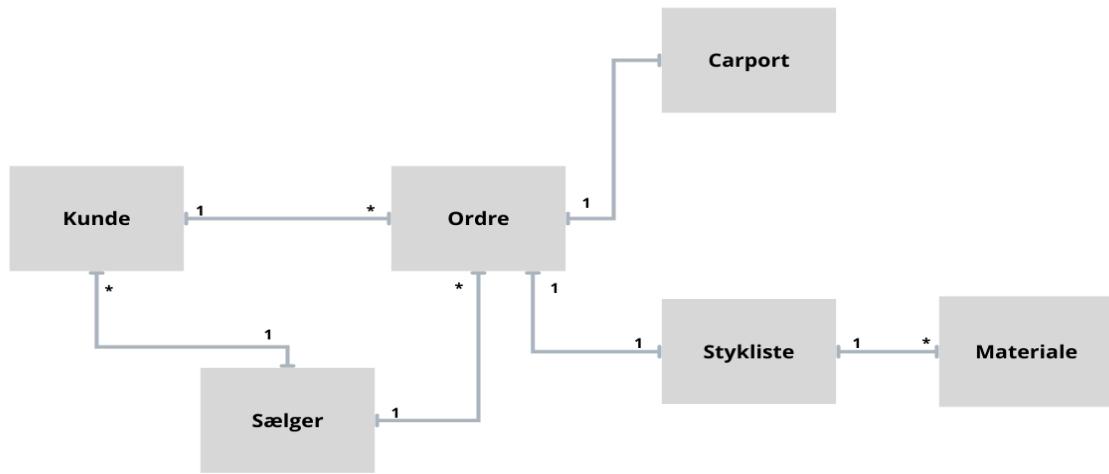
- <https://github.com/users/TobiOneCanobi/projects/4/views/1>

## Estimater

Vi har ikke lavet estimater for de forskellige tasks, da vi ikke har så meget erfaring med at estimere længden på tasks. F.eks undervurderede og fejlvurderede vi nogle tasks, som kom til at tage længere tid end forventet. Det var mest af alt validation af en brugeres information til oprettelse af en bruger. Der var flere og flere ting man kunne kræve som kriterier, så det fortsatte i længere tid, end vi lige troede. Tilmed kræver det også tid at gennemskue hvordan man laver løsningerne i kode.

Nr.	User story	Acceptance criteria
1	<b>Som</b> en bruger, <b>vil</b> jeg gerne se en oversigt over carporte, <b>for</b> at jeg kan bestille en som passer til mine behov.	<b>Givet</b> at jeg kan logge ind, og jeg er på forsiden, <b>Når</b> jeg så klikker på en carport, bliver jeg henvist til en ny side, så jeg kan se mere om den. <b>Da</b> kan jeg overveje om jeg vil købe den.
2	<b>Som</b> en bruger, <b>vil</b> jeg gerne kunne designe min egen carport, <b>for</b> at jeg kan finde en der passer til mine behov.	<b>Givet</b> at jeg er på byg-selv-siden, <b>Når</b> jeg udfylder de påkrævede detaljer, <b>Da</b> kan jeg lave en forespørgsel.
3	<b>Som</b> en bruger, <b>vil</b> jeg gerne kunne lave en forespørgsel, <b>så</b> jeg kan få en bekræftelse på min forespørgsel	<b>Givet</b> at jeg har valgt en carport jeg vil have, <b>Når</b> jeg trykker på opret forespørgsel, <b>Da</b> kan jeg vente på at jeg får et tilbud fra min sælger.
4	<b>Som</b> en sælger, <b>vil</b> jeg kunne oprette et tilbud til en kunde, <b>så</b> kunden kan få en pris på sin bestilling	<b>Givet</b> at jeg er logget ind som administrator, <b>Når</b> jeg har fået udregnings-systemets autogenererede pris, <b>Da</b> kan kunden betale for servicen og virksomheden få en indkomst
5	<b>Som</b> en bruger, <b>vil</b> jeg gerne modtage en email med tilbuddet på carporten, <b>så</b> jeg har mulighed for at købe den	<b>Givet</b> at jeg lavet en forespørgsel, <b>Når</b> sælgeren har sendt et tilbud <b>Da</b> jeg har betalt, modtager jeg en stykliste til carporten og afventer dens materialer, eller at håndværkere bygger den

## Domæne model og ER diagram

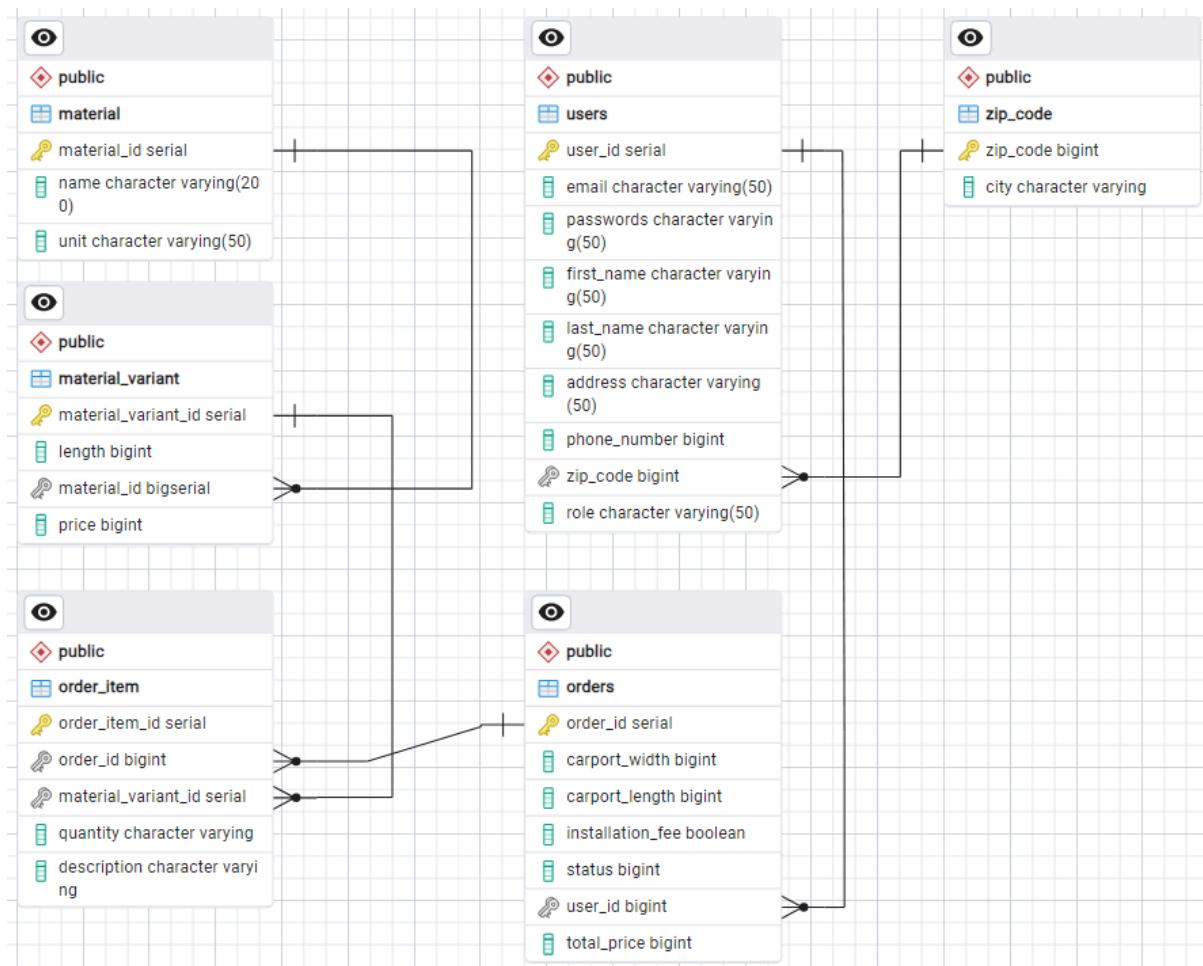


En sælger kan have mange kunder, som den anden vej betjenes af én og samme sælger.

En kunde kan have mange ordrer, som til gengæld hver især kun kan høre til en kunde. Hver ordre har kun en styklister. En styklister har mange materialer. Materialerne kan kun tilhøre én styklister.

Hver ordre kan kun have én carport. Men en kunde kan godt have flere carporte ved at have flere ordrer.

I vores opdaterede domænemodel fravalgt vi at en carport kan have et skur, fordi vi fokuserede mere på at lave en carport med et funktionelt program.



Vi har sat vores ER-diagram op, så det har seks tabeller, udover test-tabellerne. Det består af en bruger som kan lave en ordre på en carport. Carporten er bygget af en liste over materialer, som henviser til forskellige varianter af materialer, som tilhører en kategori af materialer.

### Cardinality / Entity-Relationships

Users kan have flere ordrer. Bestemte ordrer, i dette tilfælde en bestemt carport, kan kun høre til en enkelt bruger.

Orders kan have flere 'order\_items', da en carport bygges af mange dele. Et sammenhængende sæt af 'order\_items' kan kun tilhøre en ordre.

'Order\_item' er en samling af 'material\_variant', hvor hvert element der er puttet i samlingen er en enkelt variant af et 'material'. Derfor kan et 'order\_item' have én 'material\_variant', og der kan være flere forskellige 'material\_variant' i 'order\_item'.

'Material' er en overordnet kategori til varianter af varer i 'material\_variant'. Derfor kan der være mange 'material\_variant' for hver material. Omvendt kan der være ét 'material' for hver variant. F.eks tilhører varianter af spær kun til spær, og ikke til rem eller stolpe.

## **Primary keys**

I hver tabel er det id'et der er primary key, fordi det markerer at den kolonne skal være unik. Der må f.eks ikke være flere kunder med samme id, eller flere 'material' med samme id. Ellers vil man ikke kunne henvise unikt til dem. Der er en mulighed for at to kunder begge hedder Thomas Jensen og har bestilt samme carport. Måske bor de endda på samme vej. De har derfor tilknyttet et unikt id, så der let kan refereres og kendes forskel på dem.

'Zip\_code'-tabellen har ikke et ID, der er autogenereret. Vi har selv gjort 'zip\_code' attributen til primary key, fordi at postnumre til at starte med er unikke, så der behøves ikke et id til at vise det.

## **Foreign keys**

De fleste af vores foreign keys forbinder sig til andre tabeller via dets attribut af samme navn i en anden tabel.

F.eks attributen 'zip\_code' i 'users'-tabellen, der er primary key til 'zip\_code' attributen i 'zip\_code'-tabellen. Eller 'user\_id' attributen i 'users'-tabellen, der er primary key 'orders'-tabellen på dets attribut med samme navn.

Det er simpelere at udføre og nemmere at huske, hvis man skal hente data fra en tabel mens man står på en anden, hvis tabellen man skal hente data fra, forbinder sig med en attribut/foreign key med samme navn. Det vil være ineffektivt hvis 'users'-tabellen bruger 'phone\_number' attributen til at forbinde sig til 'orders'-tabellen ved at lande på 'carport\_length' attributen.

I 'order\_item' tabellen er det ikke dens primary key/id der bruges til at forbinde den videre til 'material\_variant'-tabellen, men 'material\_variant\_id'. Det er fordi at tabellens formål er at forbinde 'orders' og 'material\_variant'-tabellerne, så materiale varianterne kan vises som en 'order\_item' i order. Det har ikke brug for at hente informationer fra andre tabeller. 'Order\_items' job er at forbinde 'material\_variant'-tabellen til 'orders'-tabellen.

## **Normalformer**

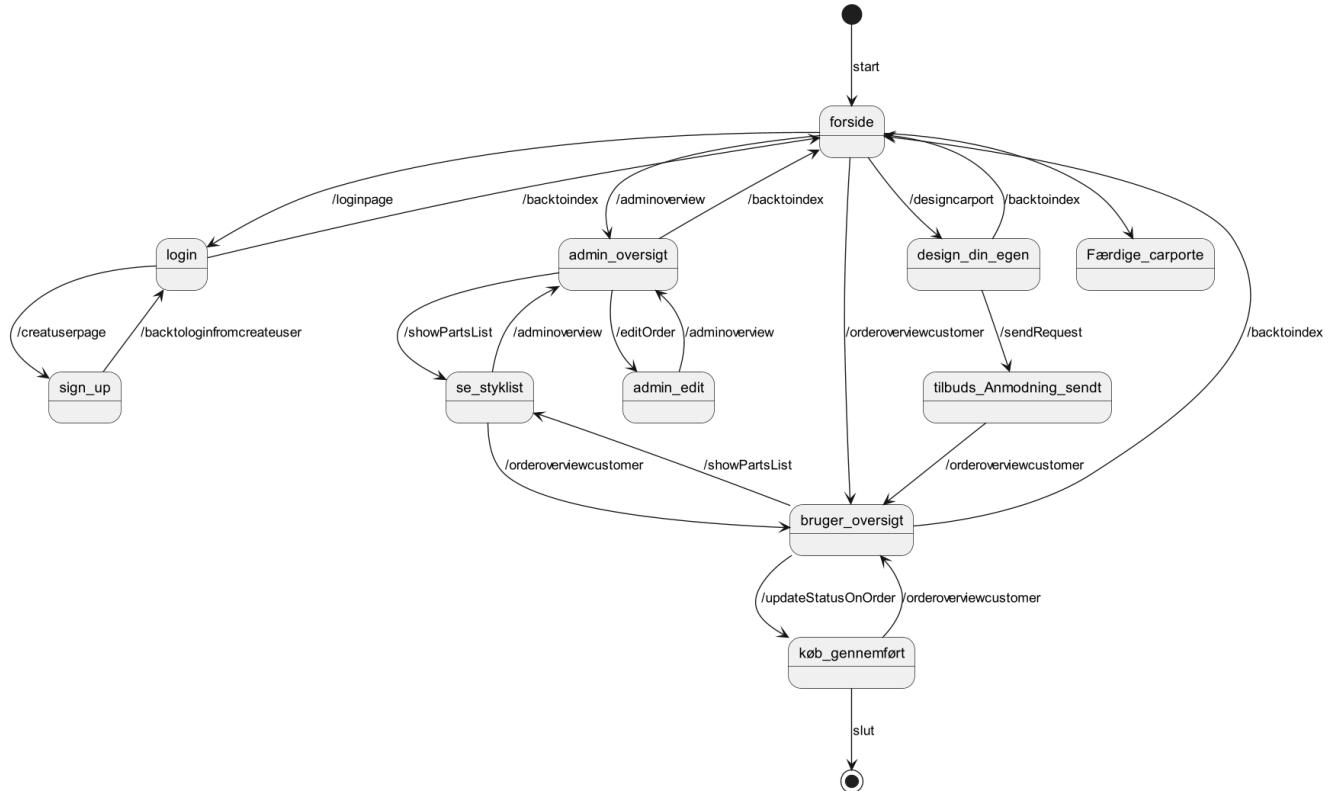
Vores ER-diagram er lavet efter opfyldning af de tre første normalformers krav.

Ifølge 1. normalform må en kolonne ikke gentage en anden kolonnes værdi. Det havde f.eks været hvis efternavnet i 'users'-tabellens 'efternavn' kolonne også stod i 'fornavn' kolonnen. På tværs af alle vores tabeller er kolonner unikt opdelt efter reglen om at kolonner skulle indeholde forskellige værdier.

Ifølge 2. normalform skal alle kolonner, som ikke er primærnøglen, være helt afhængige af primærnøglen. De afhængige kolonner kan kun blive refereret ved at referere til primærnøglen.

Ifølge 3. normalform må ingen kolonne, som ikke er kolonnens primærnøgle, afhænge af en anden kolonne, som ikke er en primærnøgle. Dvs. at kolonnerne i 'users'-tabellen ikke må afhænge af 'city' attributen i 'zip\_code'-tabellen. De skal afhænge af primærnøglen 'zip\_code'.

## Navigationsdiagram & Mockups



For at sikre, at adgangen til vores system er kontrolleret og sikker, kræver både customeroverview og adminoverview siderne, at brugeren er logget ind med de korrekte rettigheder. Specifikt skal man være logget ind som "kunde" for at få adgang til customeroverview siden, mens adminoverview siden kun er tilgængelig for brugere med "admin" rettigheder. Dette er implementeret for at sikre, at kun autoriserede brugere kan se og interagere med de respektive sider, hvilket beskytter de følsomme oplysninger, der er tilgængelige på hver side.

I forbindelse med købsforløbet har vi designet to specifikke sider, som kun er tilgængelige for kunder. Disse sider er en central del af brugeroplevelsen og guider kunden gennem processen med at bestille og betale for en carport. De to sider er:

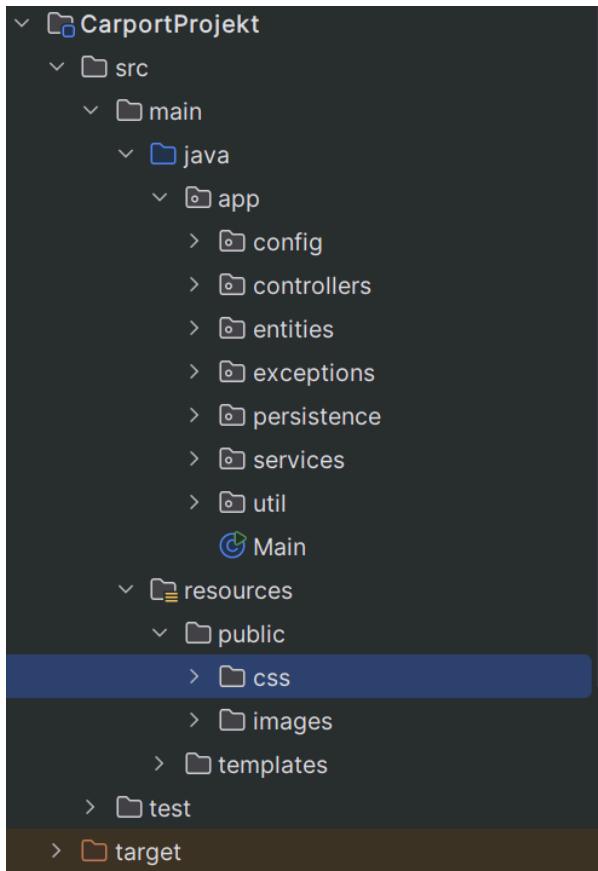
Requestsendpage, Denne side bliver kunden omdirigeret til umiddelbart efter at have bestilt et tilbud på en carport. Siden bekræfter, at deres forespørgsel er modtaget og at de herfra kan vælge til de har fået en respons tilbage fra en sælger.

Confirmation page, Efter at kunden har gennemført betalingen for deres ordre, bliver de omdirigeret til denne side. Her får de en bekræftelse på, at betalingen er gennemført, samt yderligere oplysninger om levering og forventet tidsramme.

På administratorsiden af købsforløbet har vi udviklet en side kaldet updateorder. Denne side er kun tilgængelig gennem adminoverview siden og kan derfor kun tilgås af brugere med "admin" rettigheder. Updateorder siden giver administratorer mulighed for at opdatere og administrere ordrer, hvilket er en kritisk funktion for at sikre, at alle ordrer bliver behandlet korrekt og effektivt.

En vigtig sikkerhedsfunktion i vores system er begrænsningen af adgangen til showpartlist siden for "kunde" brugere. Denne side bliver først tilgængelig for kunder efter, at de har betalt for deres ordre. Dette sikrer, at kunder kun får adgang til detaljerede oplysninger om deres ordre efter betalingen er gennemført, hvilket beskytter både kunden og systemet mod uautoriseret adgang og potentielle misforståelser. På den anden side har administratorer altid adgang til showpartlist siden, hvilket giver dem mulighed for at bistå kunder og administrere systemet uden begrænsninger.

# Valg af arkitektur



I dette projekt har vi benyttet os af en MVC-arkitektur (model view controller), som har hjulpet os med at organisere og strukturere vores klasser i pakker. Ved at følge dette designmønster har vi nemt kunnet implementere principippet om Separation of Concerns ved at opdele vores system i separate sektioner. Dette har givet os et klart overblik over en mere organiseret og vedligeholdelsesvenlig kodebase.

Projektets struktur er opdelt i to hoveddele: en "Java"-del og en "Resources"-del.

Java-delen udgør kernen af koden og er opdelt i syv pakker, hver med deres specifikke formål med henblik på at optimere vores arbejdseffektivitet under kodningsprocessen.

I de syv pakker finder vi følgende:

**Config-pakken:** Indholder to klasser, der styrer konfigurationen af Thymeleaf og session, især med henblik på deployment af programmet til et domæne på nettet.

**Controllers-pakken:** Her ligger vores kontrollerklasser, som styrer programflowet og håndterer brugerinput.

**Entities-pakken:** Indholder objekter, såsom brugere, ordrer eller materialer, som vi opretter og anvender andre steder i koden.

**Exception-pakken:** Her ligger exception-klasser, der håndterer forskellige typer undtagelser, som ikke håndteres af standard Java-fejlhåndtering. For eksempel har vi en exception-klasse til databasefejl.

**Persistence-pakken:** Indeholder vores connection pool-klasse, som forbinder projektet til en database, samt diverse mappers, som bruges til kommunikation med databasen.

**Service-pakken:** Her finder vi klasser, der muliggør brugen af Scalable Vector Graphics (SVG) i vores program, hvilket gør det muligt at visualisere selvdesignede carporte.

**Util-pakken:** Indeholder klasser med metoder, der stiller specifikke krav til brugerinput, som forekommer gennem brugen af vores program.

I "resources"-delen af vores programarkitektur håndterer vi det visuelle og funktionaliteten, som vises på websiden.

Pakke Strukturen for denne del af programmet er opdelt, så vi har en "public"-mappe og en "templates"-mappe. I "public"-mappen findes en "css"-pakke og en "image"-pakke, som henholdsvis indeholder CSS-styling til vores HTML-sider og diverse billeder, som vi bruger på siderne.

I "templates"-mappen har vi samlet alle vores planlagte HTML-sider til projektet.

# Særlige forhold

## Informationer der gemmes i session og request scope

I vores program gør vi brug af mange attributter og session-attributter. attribute er et request-scope, det vil sige at attributen kun er der i det request. sessionAttribute er i sessionen scopet, det vil sige at den eksisterer fra at vi laver session attributen til at vi terminerer den eller stopper programmet.

Nogle af de session-attributter, vi har brugt, er "current user", "guest user", "user role" og "carport length list".

**"Current user"** bliver lavet fra det øjeblik, man logger ind, og varer til at man klikker "log ud" som vil terminere sessionen ellers stopper programmet. Current user gemmer på alle de information vi har om brugeren fra vores database, altså user id, fornavn, efternavn, adresse, postnummer, telefonnummer, email, password og rolle.

```
private static void updateStatusOnOrder(Context ctx, ConnectionPool connectionPool) 1 usage ▲ Umair-230104 +1
{
    User currentUser = ctx.sessionAttribute( key: "currentUser");

    OrderMapper.updateStatus(orderId, status, connectionPool);
    if (currentUser.getRole().equals("customer"))
    {
        ctx.render( filePath: "confirmationpage.html");
    } else
    {
        ctx.render( filePath: "adminoverview.html");
    }
}
```

Her er et eksempel på, hvordan vi bruger "current user". Vi tager "current user"s rolle, og hvis den matcher med "customer", vises en bekræftelsesside. Ellers vises en oversigtsside for admin.

**"Guest user"** bruger vi, når man forsøger at oprette en bruger og har indtastet noget forkert, hvis det sker render vi siden om igen men gemmer på det info de har indtastet i alle fælderne med undtagelse af password. hvis man lykkes med at oprette en bruger eller klikker væk fra create user siden vil guest useren blive termineret.

**"User role"** bruger vi til at vise bestemte elementer. For eksempel, på index.html-siden viser vi "kundeoversigt"-knappen, hvis rollen er "customer", og "admin oversigt"-knappen, hvis rollen er "admin".

**"Carport length list"** bruger vi til at vise længder i dropdown-menuer, hvor vi henter længder fra materialer via en mapper-metode. Disse længder vises i dropdown-menuen, så man kan vælge de længder man vil have til sin carport.

Mange af de attributter, vi har brugt, er for at give messages til brugeren, det kunne være error beskeder eller hvis lykkes med at oprette en bruger fortæller vi det her.

**"Message"** bruger vi til at vise en besked ved en bestemt handling. For eksempel, hvis en bruger indtaster et nummer, når man opretter en bruger, og nummeret allerede findes, vil der blive vist en besked om, at dette nummer allerede er brugt.

**"Order List Customer"** er en anden attribut, som bruges til at vise en liste over kundens ordrer, når kunden er på sin ordreside. Denne liste vises kun, når kunden er på denne side, og forsvinder, når kunden nавigerer væk.

## Sikkerhed ift. login

Vi har lavet sikkerhed til login ved at se efter, om der er en bruger der er logget på, når man er på forsiden og trykker 'login'. Hvis man ikke er logget ind, og det burde man ikke kunne være, hvis man kommer fra forsiden med en 'login' knap, så bliver man vist til login siden. På login siden, tjekker vi om brugerens input og tjekker om det matcher det vi har i databasen. Hvis deres input ikke matcher, hvad vi har i databasen, får de en besked om at deres email eller password er forkert.

Alle steder hvor vi bruger SQL statements for at kommunikere med databasen, gør vi brug af prepared statements for at undgå SQL injections.

## Kriterier til brugeroprettelse

Vores kriterier for input af oplysninger af en ny bruger, er lavet i java-klasser, altså backend. Validation-klassen indeholder 11 metoder, som hver udfører specifikke valideringsopgaver. Specielt for password-validering har vi inkluderet fem metoder ud af de 11 for at styrke sikkerheden. De specifikke valideringsmetoder er som følger:

1. metode: Tjekker om feltet kun indeholder bogstaver.
2. metode: Tjekker om inputfeltet indeholder både bogstaver og tal.
3. metode: Tjekker om inputfeltet indeholder mellemrum mellem bogstaver og tal.
4. metode: Tjekker om inputfeltet kun indeholder fire tal (postnummer).
5. metode: Tjekker om inputfeltet kun indeholder otte tal (telefonnummer).
6. metode: Tjekker om inputfeltet indeholder @ og bogstaver (email).
7. metode: Tjekker om password1 stemmer overens med password2.
8. metode: Tjekker om password-feltet indeholder mindst ét stort bogstav.
9. metode: Tjekker om længden af password er mindst fire karakterer.
10. metode: Tjekker om password-feltet indeholder mindst ét tal.
11. metode: Tjekker om password-feltet indeholder mindst ét symbol.

Hvis vi havde lavet det i html-klasser, altså frontend, ville det være muligt for enhver at ændre kriterierne og forbigå dem. Det kan gøres ved at inspicere elementerne på sin browser, og så ændre indholdet af html-koden.

## Automatisk stort bogstav

Når man opretter en ny bruger, har vi lavet så det første bogstav i deres fornavn, efternavn og adresse automatisk laves om til at have et stort startbogstav og gemme det i databasen.

```

public static String firstLetterToUppercase(String text)
{
    String firstLetter = text.substring(0, 1).toUpperCase();
    String restOfText = text.substring( beginIndex: 1);
    text = firstLetter + restOfText;
    return text;
}

```

En string tages ind som parameter i metoden. To nye strings erklæres, firstLetter og restOfText. I de to strings indsættes parameteren.

En substring-metode bruges til at skære den første string over ved det første bogstav, og derefter kaldes metoden toUpperCase på den for at lave den om til stort bogstav.

Den anden metode bruger substring-metoden til at fortsætte fra andet bogstav og tager resten af parameteren i sig. Til sidst konateneres de to strings, defineret som text. (den string der blev taget ind som parameter) Til sidst returneres den, og ‘text’ første bogstav er nu lavet om fra småt til stort..

```

firstName = Caps.firstLetterToUppercase(firstName);
lastName = Caps.firstLetterToUppercase(lastName);
address = Caps.firstLetterToUppercase(address);

} catch (NumberFormatException n)
{
    User guest = new User(firstName, lastName, address, zipCode, phoneNumber, email);
    errorMessages.put("validinfomsg", "Alle felter skal være udfyldt");
    ctx.setAttribute("currentCreateUser", guest);
    ctx.setAttribute("errormessages", errorMessages);
    ctx.render( filePath: "createuserpage.html");
}

```

Metoden bruges i vores createUser metode, til at lave det første bogstav på en bruger stort, på firstname, lastname og address. Kort efter når vi instantierer en ‘guest’ som et user objekt der gør brug af det.

## Håndtering af exceptions

I vores projekt har vi implementeret en struktureret tilgang til exception-håndtering for at sikre pålidelighed og brugervenlighed. Vi har oprettet en specialiseret exception-klasse kaldet DatabaseException, som arver fra Java's indbyggede Exception-klasse. Denne klasse hjælper os med at håndtere fejl, der opstår under database-operationer, på en ensartet måde.

```
public class DatabaseException extends Exception
{
    10 usages  ± TobiOneCanobi
    public DatabaseException(String userMessage)
    {
        super(userMessage);
        System.out.println("userMessage: " + userMessage);
    }

    16 usages  ± TobiOneCanobi
    public DatabaseException(String userMessage, String systemMessage)
    {
        super(userMessage);
        System.out.println("userMessage: " + userMessage);
        System.out.println("errorMessage: " + systemMessage);
    }
}
```

Når en DatabaseException kastes, kan vi give en brugerorienteret besked (userMessage) samt en mere detaljeret system-meddelelse (systemMessage). Dette hjælper os med at skelne mellem, hvad der skal vises til brugeren og hvad der er relevant for os til fejlfinding.

Ved at have separate konstruktører til bruger- og systemmeddelelser kan vi hurtigt identificere og rette fejl. System-meddelelsen giver os ofte specifikke detaljer om, hvad der gik galt, mens bruger-meddelelsen kan bruges til at informere brugeren uden at afsløre tekniske detaljer.

```
    throw new DatabaseException("Fejl ved sletning af en ordre! Ingen rækker blev slettet");
}
} catch (SQLException e)
{
    throw new DatabaseException("En fejl opstod: ", e.getMessage());
}
```

Udover at have oprettet en specialiseret klasse til database exceptions, har vi også benyttet andre typer exceptions såsom NumberFormatException, RuntimeException, NullPointerException og InterruptedException. For hver type har vi kodet passende beskeder både til brugeren og til os som udviklere.

## Svg tegning

I vores SVG-tegning ønskede vi at tilføje kryds til stolperne, og dette har vi løst på følgende måde:

```
// Variabler for de første og sidste stolpers positioner
private double firstTopPostX; 2 usages
private double firstTopPostY; 2 usages
private double lastBottomPostX; 2 usages
private double lastBottomPostY; 2 usages

private double firstBottomPostX; 2 usages
private double firstBottomPostY; 2 usages
private double lastTopPostX; 2 usages
private double lastTopPostY; 2 usages
```

Vi oprettede variabler for de første og sidste stolpers positioner fra venstre side i tegningen og anvendte dem i addPosts-metoden.

```
private void addPosts() 1 usage ± Umair-230104
{
    boolean isFirstTopPost = true;
    boolean isFirstBottomPost = true;
    for (double i = 100; i <= length; i += 200)
    {
        double x1 = i;
        double y1 = width - 37.5;
        innerSvg.addRectangle(x1, y1, height: 10, width: 10, style: "stroke:#000000; fill: #000000");

        double x2 = i;
        double y2 = 32.5;
        innerSvg.addRectangle(x2, y2, height: 10, width: 10, style: "stroke:#000000; fill: #000000");

        // Gemmer koordinaterne for den første topstolpe
        if (isFirstTopPost)
        {
            firstTopPostX = x2;
            firstTopPostY = y2;
            isFirstTopPost = false;
        }

        // Gemmer koordinaterne for den første bundstolpe
        if (isFirstBottomPost)
        {
            firstBottomPostX = x1;
            firstBottomPostY = y1;
            isFirstBottomPost = false;
        }

        // Opdater koordinaterne for den sidste bundstolpe
        lastBottomPostX = x1;
        lastBottomPostY = y1;

        // Opdater koordinaterne for den sidste topstolpe
        lastTopPostX = x2;
        lastTopPostY = y2;
    }
}
```

I metoden oprettes to booleske variabler, isFirstTopPost og isFirstBottomPost, som senere bruges til at gemme stolperne korrekt. Herefter har vi en for-løkke, der starter med at placere den første stolpe ved 100 cm/pixels og fortsætter med at placere så mange stolper som

muligt, indtil de når eller overstiger længden af carporten, med en afstand på 200 cm/pixels mellem hver stolpe.

Derefter tegner vi selve stolperne. Før dette opretter vi variablerne x1, y1, x2 og y2, som indsættes i addRectangle-metoden fra SVG-klassen. x1 og y1 angiver positionen for den nederste stolpe, mens x2 og y2 angiver positionen for den øverste stolpe.

Derpå følger en if-sætning, der kontrollerer, om isFirstTopPost er sand. Hvis dette er tilfældet, sættes x2 og y2 til værdierne af firstTopPostX og firstTopPostY, og isFirstTopPost sættes til false, så koordinaterne for denne stolpe kun gemmes én gang. Den samme proces udføres for isFirstBottomPost. Herefter opdateres variablerne med de aktuelle værdier af x1, y1, x2 og y2.

Denne fremgangsmåde sikrer, at stolperne placeres korrekt og præcist i tegningen.

## Stykliste beregner

Vi ville gerne lave en beregner til spær, hvor vi havde de krav at den altid skulle starte og slutte med et spær og at afstanden mellem spærene er den samme. For at kunne udfører dette har vi lavet en dynamisk beregner der kan beregne en distance mellem alle spær der er lige lang, ved brug af længden af carporten og bredden af spær. for at kunne beregne dette laver vi krav om mindste og største afstand mellem spærene. så beregnes hvad den mindste og største antal af spær baseret på de krav vi har. så looper den igennem mulige løsning for at finde en afstand der opfylder kravene, altså at der er samme afstand mellem alle spær og at den starter og slutter med et spær. hvis loopet ikke kan finde en løsning vil den return -1.

```
public double calculateOptimalRafterSpaceWidth(int totalLength, double rafterWidth)
{
    int minSpacing = 45;
    int maxSpacing = 60;

    int maxRafters = (int) ((totalLength + minSpacing) / (rafterWidth + minSpacing));
    int minRafters = (int) ((totalLength + maxSpacing) / (rafterWidth + maxSpacing));

    for (int n = minRafters; n <= maxRafters; n++)
    {
        double totalRafterWidth = n * rafterWidth;
        int numberofSpaces = n - 1;
        double spaceWidth = (totalLength - totalRafterWidth) / numberofSpaces;

        if (spaceWidth >= minSpacing && spaceWidth <= maxSpacing)
        {
            return spaceWidth;
        }
    }
    // Return an invalid value if no solution is found
    return -1;
}
```

Herefter vil calculateNumberOfRafters blive kaldt for at bruge den løsning der er blevet fundet til at beregne antallet af spær der skal indgå i vores stykliste. hvis der ikke blev fundet nogen løsning return den istedet 0

```
2 usages ± peterjanas
public int calculateNumberOfRafters(int totalLength, double optimalSpaceWidth, double rafterWidth)
{
    if (optimalSpaceWidth == -1)
    {
        // Return 0 if the space width is invalid
        return 0;
    }
    return (int) ((totalLength + optimalSpaceWidth) / (rafterWidth + optimalSpaceWidth));
}
```

# Udvalgte kodeeksempler

## Røde felter

Vi implementerede en feature der highlighter et felt i create-user processen, som ikke er udfyldt efter kriterierne. Feltet får en rød 'border', eller omrids, når man trykker 'Opret bruger'. Vi nåede ikke at få den til at virke fuldt ud. Til at starte med har vi fået de røde borders til at komme frem, når der ikke er noget input i felterne. Ideelt vil de røde borders komme frem, når ikke ens input fylder kriterierne for informationerne. Det vil sige at de skulle komme frem sammen med fejlbeskederne, som til gengæld virker.

En anden ting er, at når man prøver at oprette en bruger, vil man meget kort se de røde borders komme frem, i de felter man ikke har indtastet noget. Men det bliver ikke ved med at være der, da siden renderes nemlig forfra. Som andre ting ville det formodentlig skulle laves til en session. Denne feature er lavet i JavaScript, hvilket vi ikke er blevet undervist i. Vi kan med vores kendskab til java-sproget og generel kode benytte os af en begrænset mængde JavaScript til at hjælpe os med ting, som html og thymeleaf ikke kan klare. Men at skulle til at lave sessions og mere i JavaScript, ville være for kompliceret, og en ineffektiv brug af vores begrænsede tid.

Vi forsøgte først at lave denne feature med html og styling, men løb ind i problemer med hvordan vi skulle implementere det.

```
<script>
    const submitButton = document.getElementById("submitbutton");
    const emailcheck = document.querySelector("#email");
    const password1check = document.querySelector("#password1");
    const password2check = document.querySelector("#password2");
    const firstnamecheck = document.querySelector("#firstname");
    const lastnamecheck = document.querySelector("#lastname");
    const addresscheck = document.querySelector("#address");
    const phonenumbercheck = document.querySelector("#phonenumber");
    const zipcodecheck = document.querySelector("#zipcode");

1 usage  ± HenrikThunbo10
function emailValidation() {
    if (!emailcheck.value) {
        emailcheck.style.border = '3px solid';
        emailcheck.style.borderColor = 'red';
    } else {
        emailcheck.style.border = '0px';
        emailcheck.style.borderColor = 'transparent';
    }
}
```

Vi erklærer flere const variabler der defineres til at være - eller henviser til - html versionen af hvert felt der skal udfyldes. Hvert felt har så en metode der skal validere om der en værdi i feltet. Hvis ikke, så kommer der et rødt 'border' rundt om feltet. Hvis et felt er udfyldt, så skal

det pågældende ‘border’ blive gennemsigtigt. Her har vi kun vist metoderne for email validering.

```
submitButton.addEventListener('click', event => {
    emailValidation()
    password1Validation()
    password2Validation()
    firstNameValidation()
    lastNameValidation()
    addressValidation()
    phoneNumberValidation()
    zipcodeValidation()
});
</script>
```

På variablen submitButton er der kaldt en metode som udfører en handling når der trykkes på den. Herinde løber den igennem alle valideringerne. Derfor opdateres felterne kun når siden renderes forfra.

## Toggle Password

Vi implementerer en feature der gør, at man kan skifte mellem at kodeord er vist eller skjult. Vi har placeret en billedfil/ikon ind i de input fields, der skal indtastes kodeord i. På det lille ikon kan man trykke med musen, så ens input skifter tilstand.

```
<script>
    let eyeicon = document.getElementById("eyeicon")
    let eyeicon2 = document.getElementById("eyeicon2")

    let password = document.getElementById("password1")
    let password2 = document.getElementById("password2")
    eyeicon.onclick = function () {
        if (password.type == "text") {
            password.type = "password";
            password2.type = "password";
            eyeicon.src = "images/PasswordEye.png"
            eyeicon2.src = "images/PasswordEye.png"

        } else {
            password.type = "text";
            password2.type = "text";
            eyeicon.src = "images/PasswordEyeslash.png"
            eyeicon2.src = "images/PasswordEyeslash.png"
        }
    }
}</script>
```

En variabel ‘eyeicon’ er erklæret og defineret til at være lig med et billede af et øje.

Denne metode er f.eks brugt til vores login side. Det virker ved, at to password variabler er erklæret og sat til at være lig med det felt man indtaster sit kodeord i.

Med et klik på ikonet kører en funktion. Hvis passwords variablene type er text, så defineres det som password. Når det er password skal den vise det ikon af et øje der ikke har en diagonal streg igennem sig.

Hvis passwords type efter et klik er text, defineres 'eyeicon' til at være et ikon med en streg over sig.

Det som gør at kodeordet er skjult, er typen på html-elementerne password1 og password2, som kan nås fordi de er refereret af de to 'let', som er initialiseret øverst i koden. Helt grundlæggende i html, så viser en 'input' typen 'text' bogstaver, mens 'password' viser de prikker, som skjuler kodeordet.

Grunden til, at der er et password2, er at der skal være to password felter i create user processen, som begge har en toggle password funktion. Her skal begge ikoner ændre tilstand når man trykker på en af dem, så et tryk ikke bare vil gemme den ene, mens den anden bliver vist.

## Gemme input i session

Vi gemmer inputtet i felter i create-user processen, til hvis man trykker opret, og siden renderer igen, og giver den besøgende en eller flere fejl beskeder. På den måde skal de ikke inputte de samme oplysninger flere gange. Når en bruger oprettes med succes, termineres sessionen. Den termineres også hvis man fra create-user siden går tilbage til forsiden.

Hvis en bruger oprettes med succes, invalideres sessionen.

```
try
{
    UserMapper.createUser(firstName, lastName, address, zipCode, phoneNumber, email, password1,
    ctx.attribute("message", "Du er hermed oprettet med email: " + email +
        ". Nu kan du logge på.");
    ctx.req().getSession().invalidate();
    ctx.render( filePath: "loginpage.html");
}
```

Hvis man trykker tilbage fra create user processen, så invalideres ens session også.

```
<div class="return">
    <form th:action="@{/backtologinfromcreateuser}" method="get">
        <button type="submit" class="button-red">Tilbage</button>
    </form>
</div>
```

# Status på implementering

## Implementering af websider

Vi har nået at lave alle de websider vi gerne ville inkludere. En forside, login-side, opret bruger-side, design carport-side, bekræftelsesside til carport bestilling, bekræftelsesside til betaling af carport. en brugers oversigt-side og admins kundeoverblik-side.

Nogle af websidernes endelige opbygning har ændret sig lidt fra vores indledende design. Det kom naturligt i løbet af projektet, fordi vi fandt ud af hvordan det var bedst at lave nogle løsninger. F.eks visningen af færdiglavet carporte man kan ”købe”, her viser vi bare et billede som viser brugeren kan klikke på for at blive videresendt til Fog’s hjemmeside, hvor man har en mere fyldestgørende beskrivelse.

## Implementering CRUD-metoder

Vi har nået at implementere alle CRUD-metoder, så data fra databasen kan behandles gennem hjemmesiden.

Brugeren kan:

- lave en carport (create)
- se oprettede data om sin carport(e) på ‘mit overblik’ (read)
- betale og derved ændre status på carporten (update)

Administratoren kan:

- også lave en carport (create)
- se oprettede data om alle kunders carport(e) på sin ‘kundeoversigt’ (read)
- acceptere forespørgsel på en carport fra en kunde og derved ændre dens status (update)
- slette en forespørgsel om en carport fra en kunde (delete)

## Styling

Vi har implementeret grundlæggende CSS-styling på vores HTML-sider for at sikre, at de er visuelt tiltalende og brugervenlige. Alle HTML-sider er designet til at være responsive, hvilket betyder, at de automatisk tilpasser sig skærmstørrelsen og oplosningen på den enhed, som brugeren anvender. Dette gør, at indholdet præsenteres optimalt, uanset om det ses på en desktop, tablet eller mobiltelefon.

En undtagelse fra denne responsive design regel er tabellerne i admin-oversigten over ordrer og kundens oversigt over ordrer. Grunden til dette er, at hvis vi tillod disse tabeller at skalere ned til mindre skærmstørrelser, ville teksten og dataene i tabellerne blive så små, at de ville blive ulæselige. For at bevare læsbarheden har vi valgt at bevare tabellernes

oprindelige størrelse, hvilket betyder, at brugere på mindre skærme muligvis skal scrollle for at se hele tabellen.

Når det kommer til valg af farver og designæstetik, har vi ladet os inspirere af Fogs egen hjemmeside. Dette betyder, at vi har valgt farver og designvalg, som minder om dem, der bruges på Fogs hjemmeside, for at skabe en visuel sammenhæng og genkendelighed for brugerne.

Mod slutningen af projektet har vi dog identificeret nogle mangler i vores styling. For eksempel skulle Fogs logo have været placeret på alle sider for at sikre fuld sammenhæng og genkendelighed, uanset hvor i programmet man befinner sig. Desuden havde vi ønsket at implementere en fælles bjælke, der viser, hvem der er logget ind, på samtlige sider efter loginsiden. Desværre fungerer denne funktionalitet kun på et begrænset antal sider.

## Test

Vi har oplevet problemer med metoderne `delete()` og `orderById()`, som beskrevet under emnet kvalitetssikring. Vi har udført tests på et tilstrækkeligt antal metoder for at sikre deres funktionalitet, og disse metoder fungerer som forventet på vores website.

## SVG

Vi har udviklet en detaljeret SVG-tegning af en carport set ovenfra, hvor brugerens specifikke bredde og længde bruges til at generere tegningen. Udeover selve carportens kontur har vi også inkluderet pile og tekst i en ydre SVG, som angiver dimensionerne af carportens sider.

Selvom vi har skabt en funktionel grundtegning af carporten, er der stadig flere elementer, vi mangler at implementere for at fuldende projektet. Disse elementer omfatter en måler, der viser afstanden mellem hvert spær, samt hvis længden af carporten gør, at der er behov for flere remme og hvor stolper skal placeres for at være bedst bærende konstruktion. Vi planlægger også at inkludere en side tegning af carporten, der viser højden og detaljerne fra siden. Derudover ønsker vi at tilføje muligheden for at tegne et skur, hvis kunden ønsker dette.

Vores primære fokus har været at udvikle et funktionelt program til tegning af carporten, hvilket har medført, at nogle af de mere detaljerede elementer endnu ikke er blevet implementeret.

## Stykliste beregner

Stykliste beregneren er blevet udviklet og fungerer tilfredsstillende. Hvis vi havde haft mere tid, kunne vi have udbygget den med flere funktioner. Som det er nu, inkluderer styklisten kun beregninger for stolper, remme og spær. En fremtidig udvidelse kunne inkludere flere elementer såsom byggematerialer til et skur.

De længder og bredder værdier man kan give til vores beregner går fra 300-600 med 60 som interval. Det vil sige at vores beregner aldrig oplever at længden på carporten overgår max længden på vores spærtræ som er 600 cm. Her kunne vi havde viderebygget sådan at hvis længden var højere end max længden på vores spærtræ, ville den gennemgå vores materiale varianter og finde et eller flere spærtræ der tilsammen vil dække den påkrævede længde og minimere spild.

Vi havde også tænkt at lave en prisberegner, der skulle tjekke, hvad for nogle materialer der blev brugt og beregne en pris ud fra det. Denne del valgte vi at droppe, da vi så det som en nice to have feature og gerne ville fokusere på andre dele af projektet. Som det er nu er prisen altid hardcoded til at være 19.999 plus at den vil se om booleanen installationFee checkbox er true og så lægge en flat 5.000 til.

Alt kommunikation der skulle ske mellem kunden og sælger er noget vi "leger" der sker. Det eneste tidspunkt hvor kunden og sælger vil blive gjort opmærksom på at en ordre er blevet ændret er hvis order statussen er 1 så kan sælger "send et tilbud" med en knap der ændrer order statussen til 2. Herefter ville kunden kunne "betale".

# Kvalitetssikring (test)

For at sikre en højere kodekvalitet og mere effektiv testning har vi valgt at oprette separate klasser dedikeret til enhedstest (unit testing) af de interne funktioner i vores program. Vi indså hurtigt, at det var både tidskrævende og ineffektivt at teste funktionerne gennem main-klassen hver gang, vi foretog ændringer. Denne erfaring førte til vores beslutning om at organisere koden bedre og implementere separate test klasser.

## Unit Testing

Vi har udarbejdet unit tests for Validation-klassen, hvor vi har dækket både positive og negative scenarier. Derudover har vi implementeret en unit test for Caps-klassen, som indeholder en metode, der gør det første bogstav i en tekst stort, hvis det er lille. Vi har også udviklet unit tests til Calculator-klassen, hvor vi har testet funktionerne til beregning af stolper, remme og spær samt en extractor-metode, der udtrækker bredden af et spær baseret på beskrivelsen.

Implementeringen af unit tests har givet os flere væsentlige fordele. Først og fremmest har det øget vores kodekvalitet ved at sikre, at nye ændringer ikke viser fejl. Ved at have et sæt automatiserede tests, der kører hver gang vi laver ændringer i koden, kan vi hurtigt identificere og rette fejl, før vi pusher og merger til main. Vi har også oplevet en betydelig forbedring i vores udviklingsproces, da unit tests gjorde det lettere for alle i gruppen at forstå og arbejde med koden.

Brugen af unit testing har hjulpet os med at opretholde en høj standard for kodekvalitet, hvilket er afgørende for vores projektets succes. Det har reduceret antallet af bugs i vores applikation og øget vores tillid til det program, vi leverer. Vi ser frem til at fortsætte med at bruge unit testing som en integreret del af vores udviklingsproces og at udforske yderligere testmetoder, som kan bidrage til endnu højere kvalitetsstandarder.

## Integrationstest

For at sikre at vores CRUD-funktioner (Create, Read, Update, Delete) fungerer korrekt med databasen, har vi også gennemført en integrationstest af OrderMapper-klassen. Her kontrollerede vi, at data flyder korrekt mellem klasserne og databasen, samt at de arbejder sammen som forventet.

Men det gik ikke som forventet, da vores metoder i OrderMapper gav problemer. Delete-metoden fungerede ikke korrekt, og vi havde svært ved at forstå årsagen. Efter en grundig gennemgang af metoden af hele teamet opdagede vi, at metoden fejlede, fordi en ordre består af flere ordre-items. For at løse problemet og få det hele til at fungere korrekt, var vi nødt til at implementere kode, der først slettes de tilknyttede ordre-items, før selve ordren kunne slettes. Denne indsigt hjalp os med at forstå afhængighederne mellem de forskellige dele af vores system bedre og sikrede en mere robust og fejlfri sletteproces.

Vi fandt også ud af at getOrderId() fejlede i testen fordi vi havde sat et user-objekt in Order konstruktøren men inde i OrderMapper havde den ikke et user-objekt i konstruktøren.

Vi opdagede også, at getOrderId() fejlede i testen. Årsagen var, at vi havde inkluderet et user-objekt i Order-konstruktøren, men i OrderMapper-koden var der ingen tilsvarende user-objekt i konstruktøren. Dette mismatch førte til, at testen ikke kunne udføres korrekt. Efter en nærmere undersøgelse fandt vi ud af, at der faktisk ikke var behov for et user-objekt i konstruktøren i OrderMapper-klassen. For at løse problemet fjernede vi user-objektet fra Order-konstruktøren i testklassen og opdaterede vores testcases i overensstemmelse med denne ændring. Denne justering var afgørende for at sikre, at getOrderId()-metoden kunne hente ordrer korrekt og pålideligt under testene.

*Screenshot af den test som fejlede:*

```
117 @Test new *
118 void getOrderId()
119 {
120     try
121     {
122         User user = new User( userId: 1, firstName: "pepande", lastName: "pandestejsen", address: "pepandevej 20",
123                             zipCode: 2600, phoneNumber: 12341234, email: "pepande@pepande.dk", password: "Pepande2!", role: "customer");
124         Order expected = new Order( orderId: 1, carportWidth: 1, carportLength: 1, installationFee: false, orderStatusId: 1, totalPrice: 10, user);
125         Order actualOrder = OrderMapper.getOrderId( orderId: 1, connectionPool);
126         assertEquals(expected, actualOrder);
127     } catch (DatabaseException e)
128     {
129         fail("Database fejl: " + e.getMessage());
130     }
131 }
132 }
```

*Screenshot af den test som bestod:*

```
117
118 @Test ± MASIH1903 +1 *
119 void getOrderId()
120 {
121     try
122     {
123         Order expected = new Order( orderId: 1, carportWidth: 1, carportLength: 1, installationFee: false, orderStatusId: 1, totalPrice: 10);
124         Order actualOrder = OrderMapper.getOrderId( orderId: 1, connectionPool);
125         assertEquals(expected, actualOrder);
126     } catch (DatabaseException e)
127     {
128         fail("Database fejl: " + e.getMessage());
129     }
130 }
131
132 }
```

Integrationstesten har lært os, at alle CRUD-operationer udføres korrekt, og at systemet opretholder dataintegritet og konsistens. Den har også givet os en dybere forståelse af, hvordan forskellige dele af systemet samarbejder og kommunikerer med hinanden. Dette har hjulpet os med at identificere og rette eventuelle fejl i dataflowet. Derudover har integrationstesten bidraget til at styrke tilliden til systemets funktionalitet og dets evne til at håndtere komplekse opgaver effektivt.

# Teststrategi

Vi har systematisk arbejdet med at teste koden før den blev en del af main branch ved at følge disse trin:

## Enhedstest (Unit Testing):

- Individuelle metoder og klasser blev testet for at sikre korrekt funktionalitet.
- Brug af JUnit til at skrive og køre tests.

## Integrations prøver (Integration Testing):

- Test af integrationen mellem forskellige klasser for at sikre korrekt dataflow og samarbejde.

## Systemtest (System Testing):

- Test af hele systemet som en helhed for at sikre, at det opfylder alle krav og fungerer i et realistisk miljø.

## Code Reviews og Pair programmering:

- Gennemgang af alle kodeændringer af teammedlemmer før integration i main branch.
- Brug af parprogrammering for at sikre høj kodekvalitet og reducere risikoen for fejl.

Ved at følge denne systematiske teststrategi, har vi sikret, at vores program er robust, pålideligt og klart til at blive brugt i produktion.

## tabel form:

Klasse	Metoder	Dækningsgrad
Validation	validateLetterOnly()	100%
Validation	validateTextContainsLetterAndNumber()	100%
Validation	validateTextContainsWhiteSpaceBetweenLetterAndNumber()	100%
Validation	validateFourNumbersOnly()	100%
Validation	validateEightNumbersOnly()	100%
Validation	validateTextContainsAtAndLetter()	100%

Validation	validateEqualPasswords()	100%
Validation	validateOneUppercaseLetterPassword()	100%
Validation	validateLengthOfPassword()	100%
Validation	validatePasswordContainsNumber()	100%
Validation	validatePasswordContainsSign()	100%
Caps	firstLetterToUppercase()	100%
OrderMapper	getAllOrders()	31%
OrderMapper	insertOrder()	36%
OrderMapper	getOrderById()	31%
OrderMapper	delete()	36%
OrderMapper	updateOrder()	36%
Calculator	calcPostQuantity()	6%
Calculator	calcBeamsQuantity()	6%
Calculator	calcOptimalRafterSpaceWidth	6%
Calculator	calcRafterQuantity	6%
Calculator	extractPartWidth	6%

I henhold til vores tabel viser dækningsgraden 100% for Validation- og Caps-klasserne. Dette skyldes, at der udelukkende testes simple metoder, hvor funktionerne ikke indeholder kode, der stopper andre kodelinjer i funktionen.

Ordermappen og Calculatoren viser en varieret dækningsgrad, da den indeholder kode, der kan springe over andre kodelinjer. Klassen omfatter for eksempel try-catch-sætninger og if-statements, som kan forhindre visse kodelinjer i at blive udført. Dette gør det mere komplekst at opnå fuld dækning, da flowet i koden kan ændre sig afhængigt af betingelserne og eventuelle undtagelser, der opstår under kørsel. Derfor kræver test af denne klasse en mere omfattende tilgang for at sikre, at alle veje gennem koden bliver testet grundigt.

## User Acceptance tests

Nr.	User Stories	Acceptance Criteria	Kommentarer
1	<b>Som</b> en bruger. <b>Vil</b> jeg gerne se en oversigt over carporte. <b>For</b> at jeg kan bestille en som passer til mine behov.	<b>Givet</b> at jeg er på forsiden. <b>Når</b> jeg så klikker på en carport, bliver jeg henvist til en ny side, så jeg kan se mere om den. <b>Da</b> kan jeg overveje om jeg vil købe den.	Godkendt
2	<b>Som</b> en bruger. <b>Vil</b> jeg gerne kunne designe min egen carport. <b>For</b> at jeg kan finde en der passer til mine behov.	<b>Givet</b> at jeg er på design carport siden. <b>Når</b> jeg udfylder de påkrævede detaljer. <b>Da</b> kan jeg få lavet en skitse af en carport med de givne mål.	Godkendt
3	<b>Som</b> en bruger. <b>Vil</b> jeg gerne kunne designe min egen carport. <b>For</b> at jeg kan finde en der passer til mine behov.	<b>Givet</b> at jeg er på design carport siden. <b>Når</b> jeg udfylder de påkrævede detaljer. <b>Da</b> kan jeg lave en forespørgsel.	Godkendt
4	<b>Som</b> en sælger. <b>Vil</b> jeg kunne oprette et tilbud til en kunde. <b>Så</b> kunden kan få en pris på sin bestilling.	<b>Givet</b> at jeg er logget ind som administrator. <b>Når</b> jeg har fået udregnings-systemets autogenererede pris. <b>Da</b> kan kunden betale for servicen og virksomheden få en indkomst.	Godkendt
5	<b>Som</b> en bruger der har modtaget et tilbud. <b>Vil</b> jeg kunne betale. <b>Så</b> jeg kan få adgang til styklisten.	<b>Givet</b> at jeg har sendt en anmodning. <b>Når</b> jeg har modtaget et tilbud. <b>Da</b> kan jeg betale og få adgang til styklisten.	Godkendt

# Proces

## Arbejdsprocessen faktuelt

### Kommunikation

Som en gruppe på fem kan man hurtigt miste overblikket over status på forskellige tasks, hvis der ikke kommunikeres effektivt. I første uge snakkede og diskuterede hele gruppen vores diagrammer og modeller, så der var ingen problemer. Men da vi begyndte at kode, indførte vi daglige 'stand-up meetings' om morgenen.

Disse møder hjalp os med at undgå at arbejde ind over hinanden og forhindrede dermed spild af tid. Ved at opdatere hinanden dagligt på vores fremskridt og kommende opgaver kunne vi sikre, at alle vidste, hvad de andre arbejdede på, og koordinere vores indsats bedre.

### Kanban board

For at undgå netop dette, var vi også opmærksomme på at vi skulle være gode til at bruge vores kanban-board og opdatere det så snart vi har kommet videre med noget.

Til at starte med bestod vores kanban boards af vores user stories. Ud fra dem snakkede vi om, hvad mindre opgaver i hver user story kunne være og satte dem så ind i vores 'to-do'. Undervejs i projektet opdagede vi nogle features, som vi gerne vil inkludere i vores projekt. Nogle af dem var 'nice-to-have' features, andre af var noget som burde være på hjemmesiden. Vi var enige fra start om, at vi gerne ville have ekstra features med, som gjorde brugeroplevelsen nemmere og mere effektiv, men som ikke var påkrævet. Grunden er, at vi havde en forestilling om, at vi havde bemanding nok til det.

Der er ikke nogen der var udpeget til kanban board mester. Vi var enige om at vi selv skulle opdatere de tasks vi har arbejdet på.

### Arbejdsfordeling

Vi designerede nogle gruppemedlemmer til bestemte dele af projektet, som de ikke havde brugt meget tid på i vores sidste projekt. En af medlemmerne lavede i sidste projekt meget backend, en anden lavede meget frontend. Derfor byttede vi rollerne for at alle kunne have en større forståelse for projektets helhed.

Små to uger henne i projektet byttede vi lidt med hinanden, for at få noget nyt at leve og få gejsten lidt op igen, og for at få andre øjne på de ting som vi havde lavet.

Efter andet møde havde vi efterhånden nogle større problemer som vi hver især sad med. På det tidspunkt begyndte vi at bruge noget af vores egen tid derhjemme, fuldt fokuseret, på at komme videre med dem. Med lyd fra andre studerende på skolen, varme temperaturer og varierende ergonomi afhængig af rum, borde og stole, kan det nemlig være svært at fokusere og samle mod til at gå i gang med svære eller store opgaver. Derfor var hjemmearbejdet en hjælp ift. til at få nogle gennembrud i nogle af vores problemer. Vi holdt

hinanden ajour med hvad der var lavet, så vi kunne se hinandens kode igennem inden vi kom igennem og arbejde med hinandens kode igen.

### Vejledningsmøder

Undervejs havde vi vejledningsmøder med vores lærer. Disse møder fungerede som opfølgninger på vores fremskridt og sikrede, at vi holdt en fornuftig kurs i forhold til projektets resterende tid. Vi fremlagde vores fremskridt og udfordringer og modtog feedback på, hvordan vi kunne optimere løsninger eller forbedre vores workflow, især i forhold til kodning i IntelliJ.

Efter vores præsentationer spurgte læreren ind til områder, de mente, vi manglede at dække, eller kontrollerede, hvordan vi havde implementeret bestemte løsninger. Dette gav os værdifuld indsigt og hjalp os med at holde projektet på rette spor.

### Faser

Den første uge, indtil vi havde det første vejledningsmøde, var en fase med opsætning af hjemmesiden. Det var ment til at være en uge med arbejde, der skulle sørge for at vi havde et godt grundlag for at teste de mere avancerede blokke af koder.

Den anden uge, efter andet vejledende møde, var vi i en fase, hvor vi skulle implementere noget af det mere avancerede kode.

Det var f.eks kriterier der er til en bruger om, hvordan man må udfylde feltene i bruger-oprettelse. F.eks. at navne skal starte med stort og at postnummer skal være med 4 cifre og kun tal. Også at få metoderne til at virke gennem HTML og dermed inde på hjemmesiden. Vi var mange gange igennem oprettelse af en bruger for at teste at det virkede som det skulle. Det krævede mange gange at der skulle ændres noget, for at et kriterie virkede som det skulle. Et endnu større problem var at få alle kriterier til at virke samtidigt. Der opstod nogle bugs, som vi brugte nogle timer på at løse.

Rapportskrivning begyndte også så småt efter andet møde, op mod tredje møde.

Et par af medlemmerne i gruppen startede efter det andet møde på svg. Først med at se videoer og sætte sig ind i hvordan det virker, og derefter arbejdede de fokuseret på at få det til at virke, så vi med en kundes input på hjemmesiden kunne generere en carport i en todimensionel tegning.

Efter tredje fase begyndte vi at finpudse vores frontend. Vi var enige om at starte med det funktionelle i projektet med at lave frontend. Det vil sige, vi fik kun sat html op uden nogen former for styling.

I det sidste projekt, vores gruppe lavede, startede vi rigtigt tidligt med html og styling, hvilket resulterede i mange merge konflikter. Derudover havde vi generelt ikke en god ide om hvad de andre lavede og derfor mistede vi overblikket over hvad der skete i backend-delen.

### Deling af kode

Deling af kode gik godt og næsten helt gnidningsfrit. I vores sidste projekt havde vi mange merge konflikter, men denne gang var vi meget bedre til at committe og pushe vores

branches oftere. Vi opdelte tasks på en måde, så det var helt klart, hvem der lavede hvad, og hvad indholdet i hver task var.

Dette gjorde, at chancen for at redigere i de samme kode blokke på forskellige branches var meget lille. Den forbedrede koordinering og hyppigere commits bidrog til en langt mere effektiv og problemfri kode integration.

## Arbejdsprocessen reflekteret

### Kanban board roller

Vi har ikke haft brug for en kanban board-mester i dette projekt. I vores sidste projekt var vi ikke særlig gode til at bruge kanban boardet, men der var, især i starten, én person, der holdt styr på det. Denne gang redigerede vi alle boardet, når vi opdagede nye tasks, der skulle oprettes, og når vi havde færdiggjort en task.

Dette viste sig at være en mere effektiv tilgang for os. Vi føler, at det ville tage meget tid for én person og være en byrde for vedkommende at skulle redigere boardet alene, så snart der var noget nyt. Ved at dele ansvaret mellem os kunne vi holde boardet opdateret uden at overbelaste nogen.

### Evalueringsmødernes emner

Til evalueringsmøderne snakkede vi om alle aspekter af projektet og arbejdet.

Vi gik ikke ind til dem fra et dårligt udgangspunkt, hvor vi manglede hjælp for at komme videre. Vi var hver gang godt med og brugte mere møderne som forsikring på, om vores tempo var passende ift. hvor meget tid vi havde tilbage.

Vi følte ikke behovet for at fremlægge vores projekt som en demo, til vores vejleder, som om han var en kunde. I stedet fortalte vi om hvor langt vi var og hvad vi arbejdede på, som var det en normal samtale.

Her fik vi feedback tilbage om, hvordan det var en god ide at balancere det krævede indhold af hele opgaven, mod nice-to-have features.

Undervejs havde vi dog nogle spørgsmål til detaljer som løsninger af kode, omfang af coding standard, diagrammer og user stories.

I det første møde fik vi feedback på arbejdet fra vores første uge, som var ren planlægning. Vi fik nogle korrektioner på vores user stories, så de var formuleret på en mere forståelig måde, der understøtter kundens kriterier til hvad.

### User stories til tasks

Det var ikke et problem for os at nedbryde user stories til task, men vi opdagede helt sikkert flere løbende som vi kom igennem projektet. Men det gjorde ikke vores planlægning i første uge var ufyldestgørende. Det føltes som en naturlig proces at projektets små detaljer udviklede sig med tiden som vi arbejdede.

## Oprettelse af bruger kriterier

Vi var ikke helt præcise med udførelsen af kriterierne for brugeroprettelse, hvilket tog meget længere tid end forventet. Vi startede med lidt co-pairing i et par dage, men derefter gik den ene videre til andre opgaver. Det, som den anden forventede at færdiggøre hurtigt, viste sig at tage næsten en uge.

Først skulle vi identificere nye kriterier, der burde være inkluderet. Derefter fulgte implementeringen af disse kriterier, hvilket afslørede flere bugs. Et yderligere problem var, at nogle komponenter forhindrede hinanden i at fungere korrekt, hvilket krævede ekstra tid at løse.

## Arbejdsrytme

Vi startede med at kode individuelt, med undtagelse af to personer, som begyndte sammen ved at opsætte user entity, mapper og controller gennem pair programming. Dette sikrede en fejlfri start på projektet. Efter denne indledende fase valgte hver person en task at arbejde på, efterhånden som vi færdiggjorde de opgaver, vi havde påtaget os.

For visse tasks fokuserede vi på at implementere funktionaliteten på en måde, så vi kunne teste, om det virkede korrekt, selvom det kun blev implementeret ét sted. Et eksempel på dette var en feature, der gav et rødt omrids til et felt, der ikke var udfyldt. Vi bemærkede, at denne funktion forstyrrede kriterierne for felterne, og derfor færdiggjorde vi nogle komponenter, før vi gik videre med andre, så de kunne kategoriseres som færdige på vores kanban-board.

Når en task ikke kunne færdiggøres, fordi den afhæng af en anden, begyndte vi at arbejde på en ny opgave i mellemtiden. Dette sikrede, at vi altid havde noget produktivt at lave og kunne holde projektet i konstant fremdrift.

## Brug af pull requests

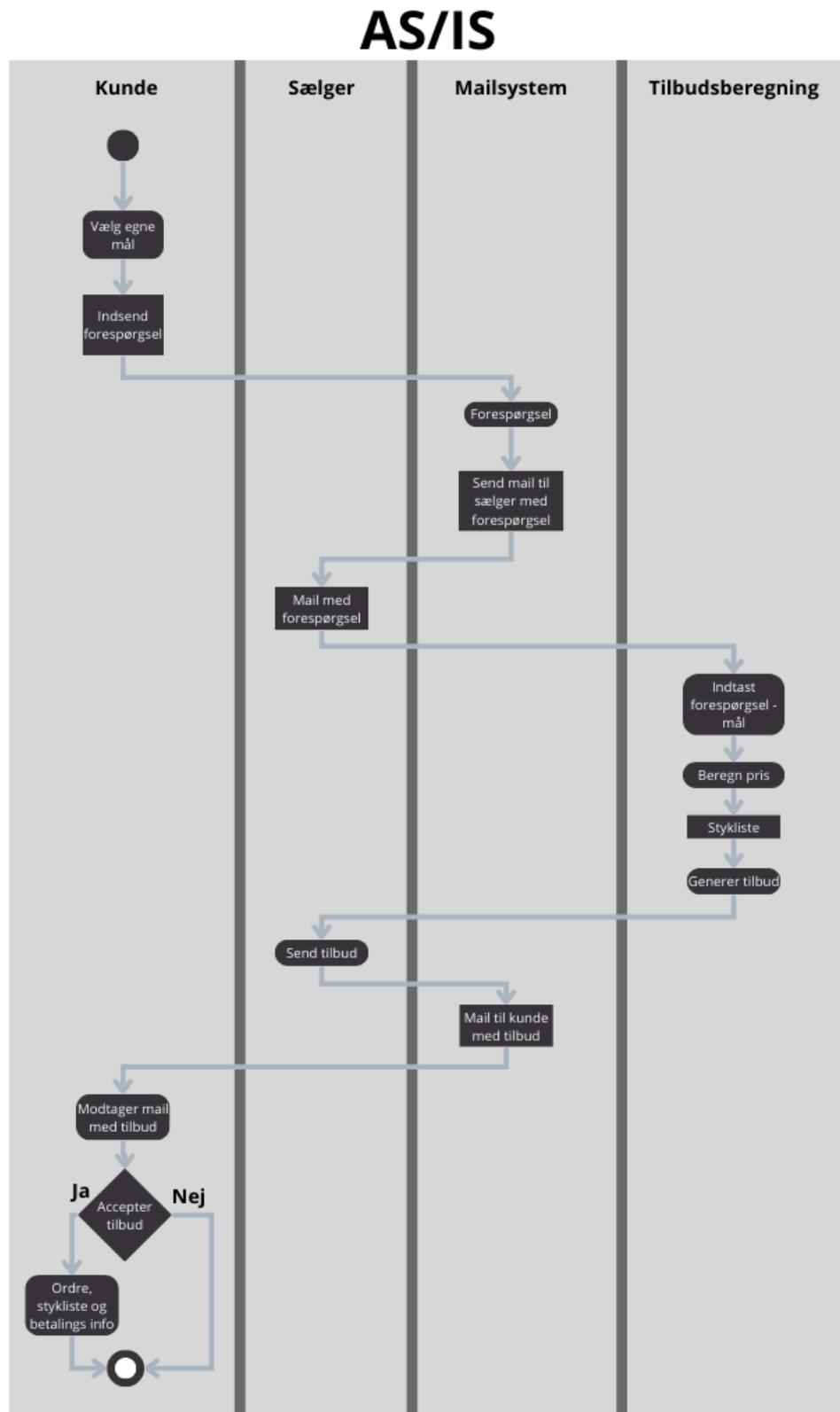
Vi brugte pull requests flittigt, især på dage hvor vi ikke var fysisk sammen i gruppen, som for eksempel på helligdage. Hver person blev tildelt specifikke opgaver, og når en opgave var færdiggjort, blev der lavet en pull request i stedet for at merge ændringerne direkte ind i koden. Disse pull requests blev derefter gennemgået og godkendt af et andet gruppemedlem.

Fordelen ved denne tilgang var, at vi kunne undgå merge-konflikter og andre potentielle problemer, der kunne opstå ved direkte merges. Vi anvendte dog ikke pull requests så ofte, fordi vi som regel arbejdede sammen fysisk på skolen. Når vi var samlet, foretog vi merges direkte, idet vi kunne gennemgå ændringerne i fællesskab og løse eventuelle problemer med det samme.

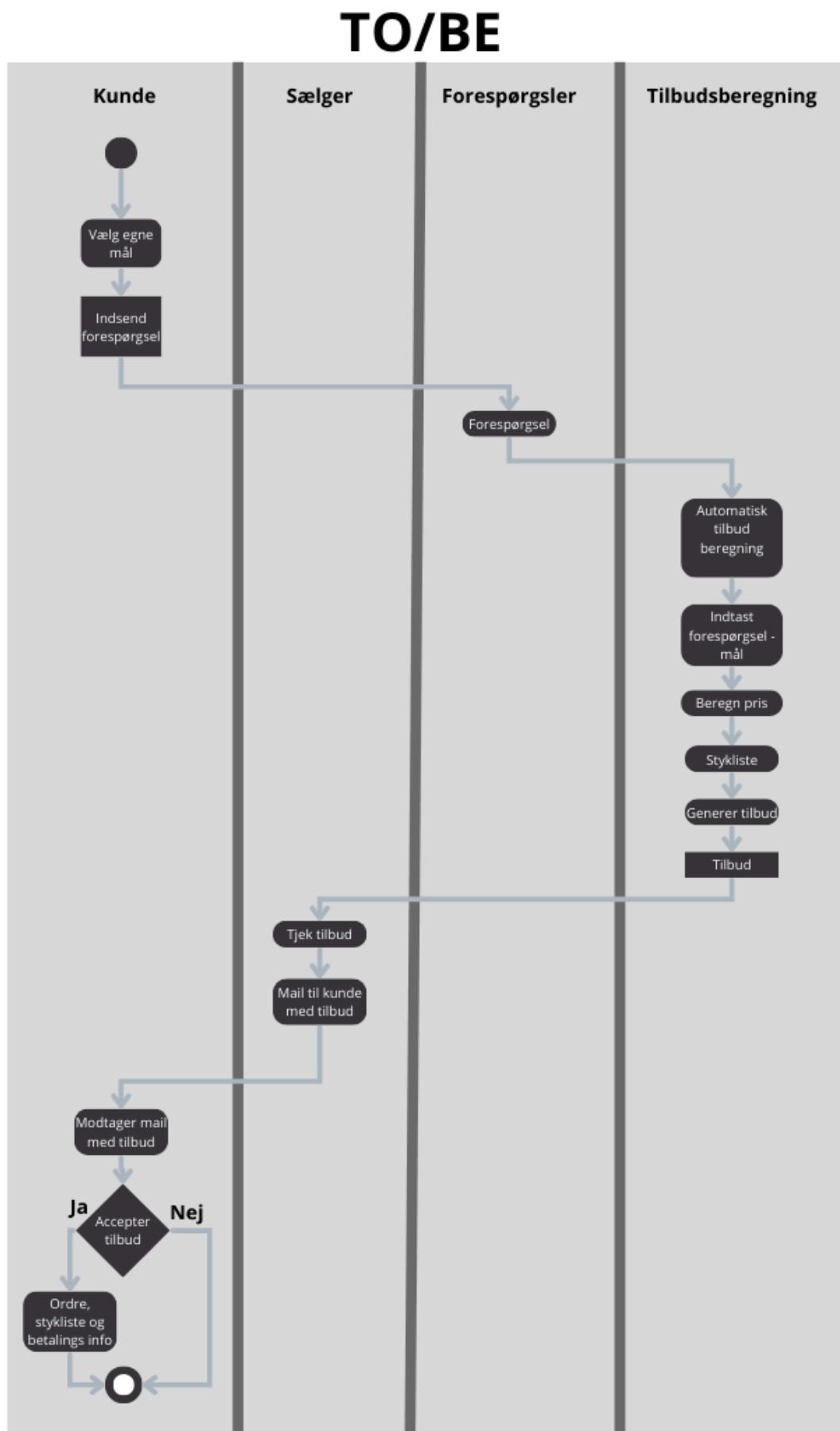
# Bilag

## Diagrammer

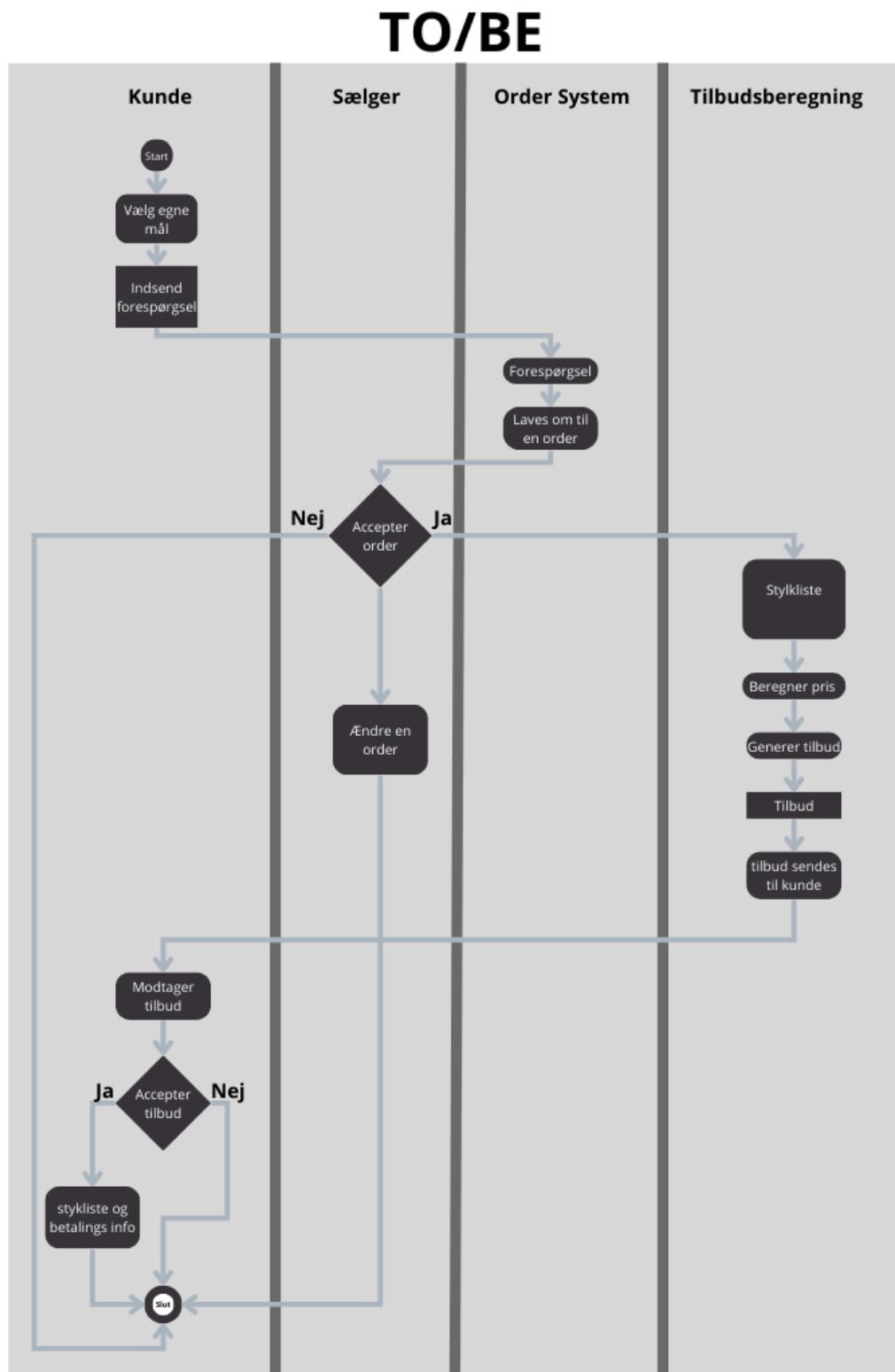
Bilag 1 (AS-IS aktivitetsdiagram):



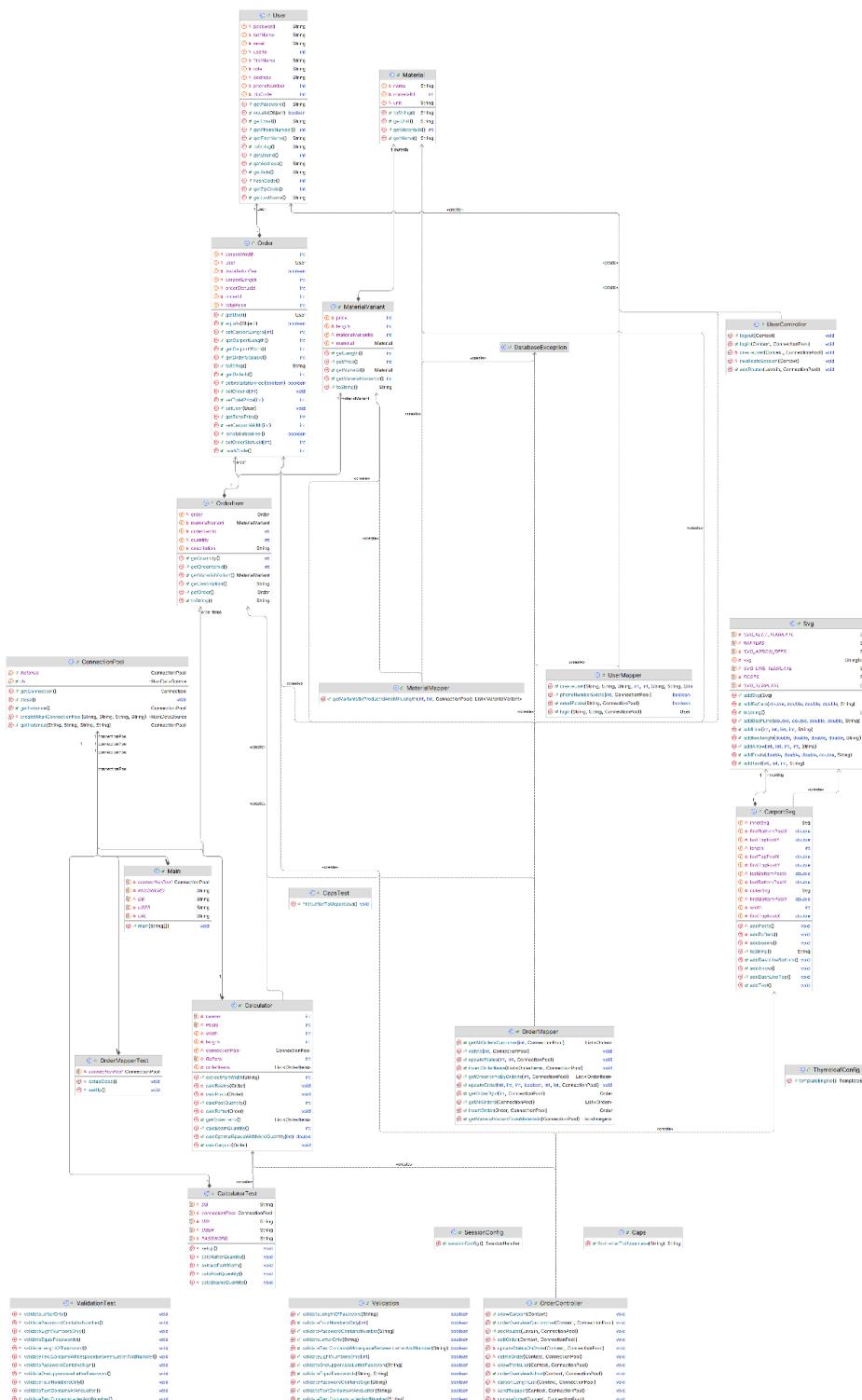
Bilag 2 (gamle TO-BE aktivitetsdiagram):



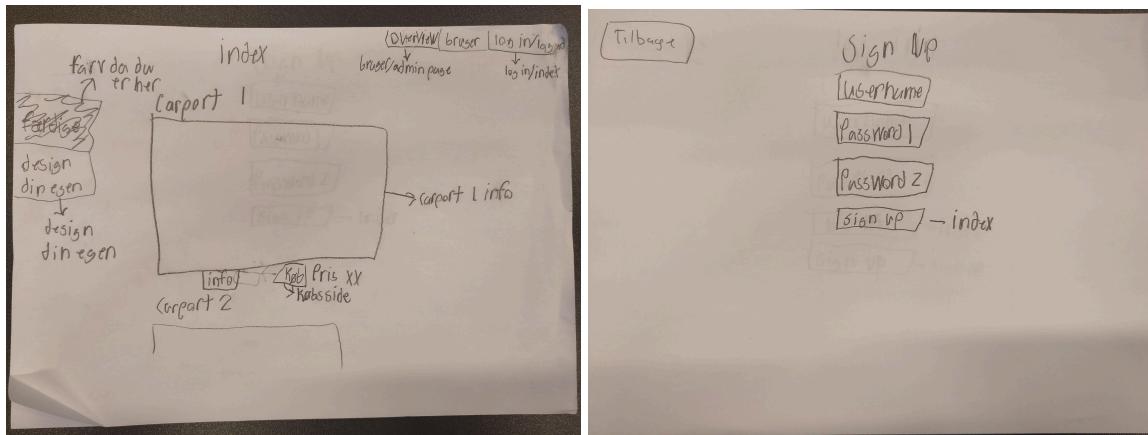
Bilag 3 (ny TO-BE aktivitetsdiagram):



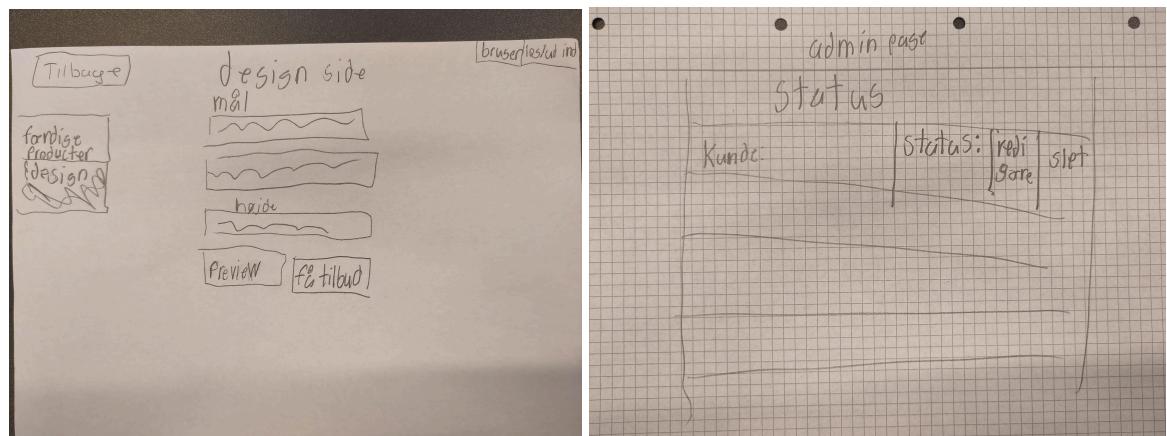
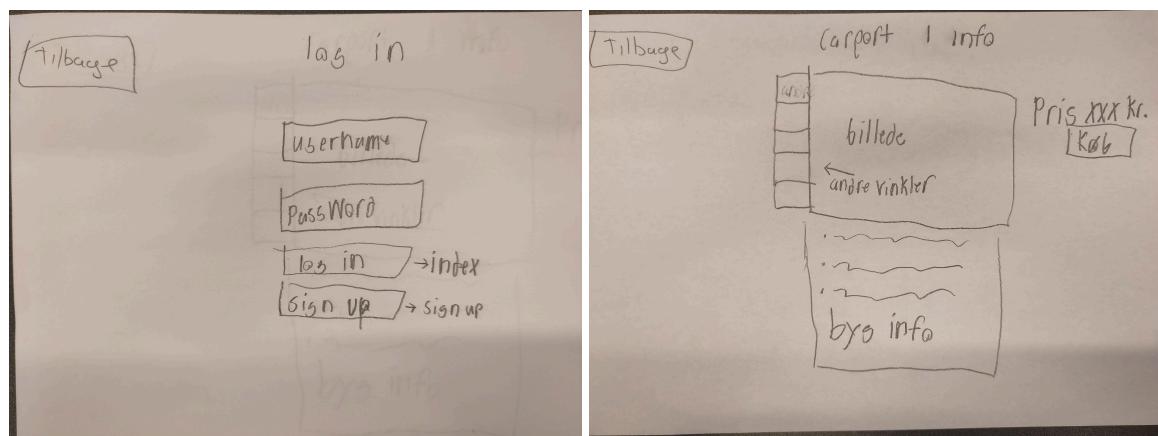
## Bilag 4 (klassediagram): Se i intelliJ under diagrammer for, at se i højere oplosning.

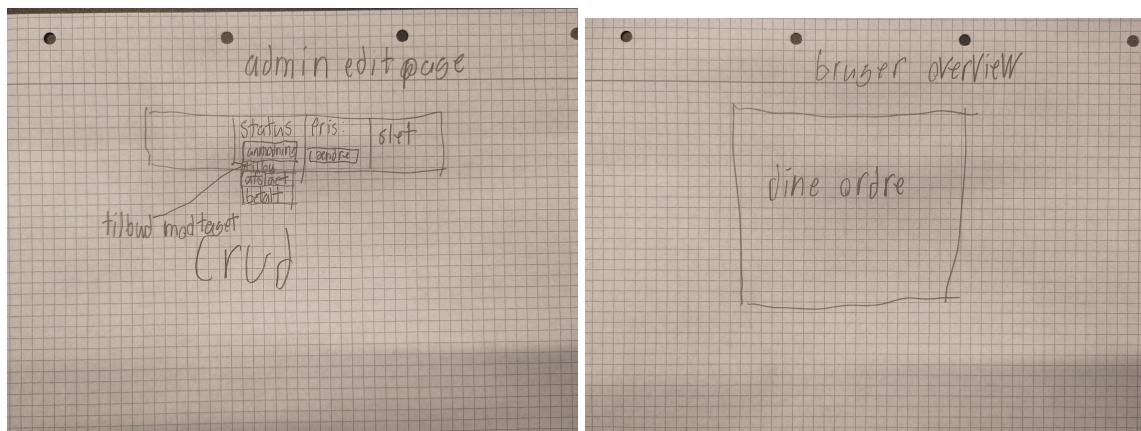


## Bilag 5 (mockups til hjemmeside design):



Sign Up  
[Username]  
[Password 1]  
[Password 2]  
[Sign up] → index





## Logbog

**Man 29/4:** Derefter kom vi stille og roligt i gang med at starte op på nogle diagrammer og diskutere detaljer om projektet,

**Tir. 30/4:** Vi har lavet om på vores as-is og to-be diagrams, user stories, risikoanalyse og interessenstanalyse fra første uge om forretningsforståelse, på baggrund af feedback fra Kim. Vi lavede også et aktivitetsdiagram

**Ond 1/5:** I dag er at vi har fået rettet, og godkendt vores domæne-model af Jon, samt fået lavet og godkendt et ERD "mockup" som vi har fået indtastet i en database på Peters' PgAdmin lige nu. Vi har derudover lavet mockups, både på papir og i canva, på Umair's pc.

**Tor 2/5:** Vi har fået færdiglavet vores mock up til hjemmesiden. Vi har også fået lavet et navigationsdiagram og et klassediagram. Til sidst fik vi sat en firewall om vores droplet på DigitalOcean. Inde på vores IntelliJ produkt fik vi så alle forbundet os til Masihs database, gennem postgreSQL plugin. Til sidst gjorde vi det også på pgAdmin.

**Fre 3/5:** Vi har haft møde, så med vores vejleder og fortalt hvor langt vi er. Vi opdagede at vores user stories var lidt forældede, så vi opdaterede dem til at passe til vores opgave. Vi satte controllers og mappers op, samt routing, så html kan kommunikere med java. Vi lavede metoder så vi kan logge ind og ud af vores hjemmeside. Hjemmesiden havde fået sat knapper osv. op, så vi rent faktisk havde noget at teste vores metoder på.

**Man 6/5:** Vi har lavet create user metoder i mapper og controller, og testet det med succes. Vi har fået sat flere sider (basalt) op på vores domæne uden styling, og fået de knapper til at virke, som viser brugeren ind på de forskellige sider. En fra gruppen lavede finpudsninger i order controller- og mapper, og så også SVG videoer for at sætte sig ind i det, når vi snart skal lave det.

**Tir 7/5:** Vi har lavet sign-up criteria til udfyldning af felter, når man laver en ny bruger. Vi indsatte de basale dele af SVG og, at det ligger det rigtige sted i vores kode, hvor vi skal

bruge SVG. Vi har brugt Jons mål/materialer indtil videre for at se om det virker i vores projekt.

**Ons 8/5:** vi vil lave en session til createUser processen, så det man udfylder i felterne ikke forsvinder, hvis man har inputtet noget forkert. Vi lagde mærke til, at sessionen blev gemt, så vi rettede det med terminate session. Vi vil også korrekte, at man ikke får server error og nullpointerException ved felter som kræver en int, men ikke er indtastet. Vi laver også så at input bliver rettet til at være med stort bogstav ved navne og adresse, hvis ikke brugeren har skrevet det sådan.

**Fre 10/5:** Der er implementeret fejlbeskeder til hvert felt i createuserpage. Koden i createUser metoden i userController er effektiviseret. User klassen, userController, userMapper, Caps klassen og Validation klassen er trimmet. fundet og indsæt materialer ind i vores database som hermed kan bruges til blandt andet "calculator" og til SVG tegning. Der blevet lavet at der kan tegnes en carport udefra kundes mål dog uden pile og tekst.

**Søn 12/5:** Har pudset html og css, bl.a. ændret titler, beskrivelser, farver og størrelse på knapper

**Man 13/05:** Der er lavet så et rødt border kommer om 'firstname' hvis ikke det er udfyldt. Senere skal det laves så det også kommer, hvis ikke de andre kriterier er opfyldt, også for alle andre inputs. Der er også lavet et 'toggle password visibility' til password felterne. Diverse html og deres css er blevet strømlinet, så opsætningen er ens på alle sider. Svg Tegningen blev til, at man kan generere en carport med både pile og tekst som viser hvor lang og bred carporten er.

**Tir 14/05:** En mand er gået i gang med at skrive rapport. Lavet ændringer i css for at imødekomme målet om skalerbarhed i forhold mellem pc og mobil. Der er lavet at der bliver lavet en carport med kryds fra stolper og det hele kører som det skal med SVG. Der er blevet testet på validation igen og specifikt på postnummer og adressefeltet. En bruger kan nu kun taste tal ind og ikke bogstaver på de to felter. Udover det fandt vi nogle huller i email og adresse feltet. De to er lidt tungere end andre, da der er både bogstaver og tal, der kræves. Arbejdet blev ikke færdigt i dag, men planen er at færdiggøre det i løbet af ugen.

**Ons 15/05:** En mand har testet mere validation i createUser processen og opdaget flere ting der kan optimeres. Har også gjort så felter med manglende input bliver rødt, men det forsvinder når der er et nyt render af en side. Der er også lavet så session blive ugyldig, når man trykker 'tilbage' fra create user siden.

Vi har fået lavet alle slags CRUD metoder for ordermapperen. En admin kan nu gå på sin admin overblik side både redigere en order eller vælge at slette den. Lavet en form i design carport siden, som tager alt info for den nuværende session af en bruger, samt lavet små ændringer i html & css for at ting ser påne ud.

**Tor 16/06:** Lavet styling og omplacering af slet knappen under edit knappen, samt sat edit formularen ind i en pæn styling.

Vi fik lavet integrations test på alle vores CRUD metoder i orderMapperen udover, at testen for delete virker ikke endnu. Vi fik lavet, at en kunde kan se sine ordre på sin “kunde overblik” side og der mangler lige, at en kunde skal betale også skal statussen opdateres hos admin, at kunden har betalt.

**Fre 17/5:** Vi fik sat up flowet fra en bruger vælger en carport til de har købt den, og testet hele flowet mellem brugeren og admin siden.

Fik lavet så der er en dropdown over længder man kan vælge til sin carport.