Fog Carport

Udvikler:
Navn: Carl Emil Styrbjørn Gullacksen
Email: cph-cg207@cphbusiness.dk
Github: Styrse – www.github.com/styrse

Klasse:
I-dat-da 0924b - Sem2 - (Klasse B)

Udarbejdet: april/maj 2025

Projekt:
Side: Fog Carport – www.carport.styrse.com
Repo: Github – www.github.com/styrse/carport
YouTube: Video - https://youtu.be/A65KMGipfVI

Tegn	Modeller	Total	Normalsider
66.963	6	71.763	29,9

Indholdsfortegnelse

Indledning

Dette soloprojekt er udviklet på 2. semester med det formål at digitalisere Fog Byggemarkeds arbejdsgang for specialtilpassede carporte. Kunder kan konfigurere deres carport direkte på hjemmesiden og få pris og visualisering med det samme. Medarbejdere arbejder i et internt dashboard, hvor ordrer og materialer kan håndteres struktureret og effektivt. Projektet bygger på Java, PostgreSQL og Thymeleaf og har særlig fokus på realisme i beregning, fleksibel arkitektur og rollebaseret adgang.

Baggrund

Kunden

Johannes Fog er en dansk byggemarkedskæde, der har specialiseret sig i at levere kvalitetsmaterialer og rådgivning til både private og professionelle kunder. Virksomheden lægger stor vægt på personlig service og skræddersyede løsninger, hvilket er en vigtig del af deres konkurrencefordel.

Kundens krav og ønsker

Projektet tager udgangspunkt i et konkret ønske fra salgschefen hos Johannes Fog om at modernisere og digitalisere arbejdsgangene i forbindelse med bestilling af specialtilpassede carporte. I den nuværende arbejdsgang udfylder kunden en formular på hjemmesiden, hvorefter forespørgslen sendes som en e-mail. En medarbejder skal herefter manuelt indtaste oplysningerne i et andet system – en proces der er både tidskrævende og fejlbehæftet.

Derfor ønskede kunden et nyt system med følgende overordnede krav:

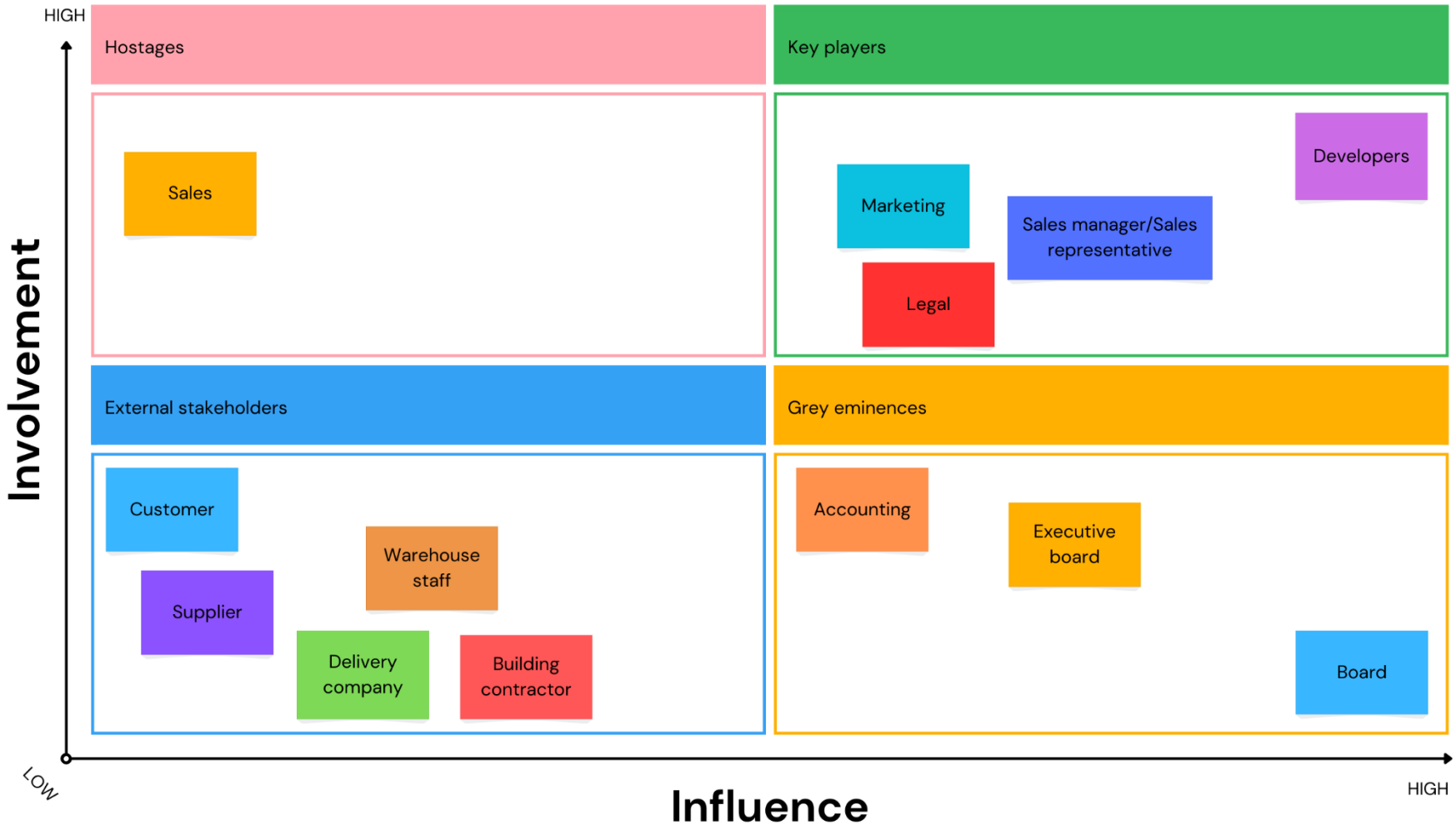
- Kunder skal kunne konfigurere og indsende forespørgsler på carporte direkte på hjemmesiden. Det skal være muligt at angive mål som bredde, længde og højde, samt vælge blandt forskellige materialer.
- Systemet skal understøtte dynamiske beregninger og visualisering baseret på kundens valg, hvilket gør løsningen mere fleksibel og tilpasset den enkelte kunde.
- Medarbejdere skal kunne logge ind på et internt dashboard, hvor de har adgang til at se og redigere ordrer, kunder og materialer.
- Systemet skal effektivisere arbejdsgangen for medarbejderne og give dem mere tid til kundekontakt, hvilket Fog betragter som en kerneværdi i deres service.
- Salgschefen skal kunne administrere materialekataloget direkte i systemet – herunder oprette, redigere og slette materialer.
- Systemet skal understøtte rolleopdeling mellem almindelige medarbejdere og ledere (f.eks. salgschefen), hvor ledere har adgang til yderligere administrative funktioner som medarbejderstyring og systemopsætning.
- Det skal være muligt at sende beskeder og betalingslinks til kunden direkte fra systemet, så en medarbejder efter en telefonsamtale nemt kan følge op uden at bruge eksterne værktøjer.

Ud over at løse konkrete, praktiske behov skal systemet også være med til at fremtidssikre Johannes Fog på markedet for specialbestilte carporte. Ved at gøre det lettere for kunderne at tilpasse deres bestilling efter

egne behov, samt give medarbejderne mere effektive arbejdsredskaber, styrker Fog deres position i forhold til konkurrenterne og lægger fundamentet for videreudvikling og skalering.

Virksomheden

Interessentanalyse



Der blev i projektets begyndelse udarbejdet en interessentanalyse for at identificere, hvilke aktører systemet påvirker, og hvordan deres behov kunne imødekommes. Analysen bidrog til at prioritere funktionalitet og brugerflow – særligt med fokus på medarbejdernes arbejdsprocesser og salgslederens behov for overblik og kontrol.

Analysen viser, at kunder, medarbejdere og ledelsen har størst indflydelse på systemets succes, og at løsningen skal balancere brugervenlighed, effektivisering og fleksibilitet. Flere øvrige interessenter – som lager, håndværkere og jura – påvirkes i varierende grad og er tænkt ind i den videre planlægning. Se interessentoversigt i bilag, tabel 3.

Risikoanalyse

Se fuld risikoanalyse i **bilag, tabel 2** samt tilhørende **risiko-matrice i tabel 3**.

Følgende nøglepunkter fremhæves:

R1 – Utilstrækkelig testdækning:
Minimeres ved krav om 90 % unit test-dækning for at forebygge kritiske fejl i pris- og styklisteberegninger.

R3 – Inputvalidering:
Vurderes som lav risiko, da Fog i forvejen kontakter kunder telefonisk og dermed fanger fejl manuelt.

R8 – Kommunikation med kunde:
Håndteres via løbende sprint reviews og godkendelser, hvilket reducerer risikoen for misforståelser.

R12 – GDPR-brud:
Mindskes ved inddragelse af Fogs juraafdeling. Vi følger gældende praksis, men erkender en vis rest-risiko ved lovændringer.

R13 – Fog går konkurs:
Risikoen vurderes som meget lav pga. opkøb, men der modtages fortsat delbetalinger som sikkerhedsforanstaltning.

Opsummering:
Risikoanalysen har bidraget til mere fokuseret planlægning og løbende risikohåndtering. Både tekniske, organisatoriske og forretningsmæssige risici er vurderet og imødegået gennem konkrete tiltag, der har skabt tryghed for både kunde og udvikler.

Teknologivalg:

Projekt konfiguration

Projekt konfiguration	Version
sourceEncoding	UTF-8

Backend

Backend	Version
Java	17
Javelin	6.5.0
ThymeLeaf	3.1.3

ThymeLeaf extras	3.0.4
sl4j	2.0.17
jackson	2.18.3

Frontend

Frontend	Version
HTML	Nyeste version
CSS	Nyeste version
JavaScript	Nyeste version

Database

Database	Version
Postgressql	42.7.5
Docker	Nyeste version
hikariCP	6.2.1

Testing

Testing	Version
JUnit	5.12.0
hamcrest	3.0

Vision

Målet med projektet er at udvikle et system, som Johannes Fog Byggemarked reelt kan anvende til bestilling og håndtering af specialtilpassede carporte. Løsningen digitaliserer kundens bestillingsproces og effektiviserer interne arbejdsgange ved at samle kundedata, ordrer og materialer ét sted.

Systemet afspejler den virkelige arbejdsgang og balancerer kundens behov for fleksibilitet med virksomhedens behov for struktur og overblik. Kunderne kan nu selv konfigurere carporten online, mens medarbejdere arbejder i et dashboard uden behov for dobbeltindtastning.

Samtidig understøtter systemet salgsledelsens behov for værktøjer til materialestyring og medarbejderroller. Ved at samle funktionaliteten i én platform styrkes virksomhedens effektivitet og evne til at levere hurtigt – uden at gå på kompromis med Fogs personlige kundeservice.

Udvidelse og fremtidige muligheder

Systemet er udviklet med en arkitektur, der understøtter fremtidig udvidelse både teknisk og forretningsmæssigt. Funktionaliteten er opdelt i klare lag (bl.a. controller, service og datalag), hvilket gør det enkelt at tilføje nye features uden at ændre eksisterende logik. Denne opbygning muliggør løbende udvikling og tilpasning, efterhånden som virksomheden får nye behov eller ønsker at optimere sin service yderligere.

En vigtig del af systemets fleksibilitet ligger i materialemodulet, som er struktureret så det på sigt kan rumme forskellige typer materialer med individuelle egenskaber. Selvom der i første version kun er implementeret de mest grundlæggende materialetyper, er strukturen allerede forberedt til at kunne

håndtere nye produktkategorier som fx **el-ladere, udendørsbelysning, gulvbelægning (fliser, sten mv.)** og andre *add-ons*. Disse kan integreres som selvstændige klasser uden større omskrivninger af systemet.

Fra et forretningsperspektiv åbner systemets struktur også op for muligheder inden for **salg og mersalg**. Et eksempel herpå er at tilbyde relaterede produkter direkte i bestillingsforløbet, så kunden før checkout bliver præsenteret for værktøj og tilbehør, der er nødvendigt for at opføre carporten. Dette kan skabe grundlag for strategisk **kryds-salg** direkte på hjemmesiden og bidrage til øget indtjening per ordre.

Krav

Kravene til systemet er identificeret i dialog med salgschefen hos Fog og afspejler både kundens ønske om fleksibilitet og virksomhedens behov for kontrol, indtjening og effektivisering. De er formuleret med udgangspunkt i praktiske erfaringer frem for tekniske specifikationer.

Et centralt krav er, at kunden først får adgang til stykliste og teknisk dokumentation, når betalingen er gennemført. Dette beskytter Fogs rådgivning mod misbrug og fastholder kundekontakten som en værdiskabende del af salget.

Kunderne skal kunne vælge materialer under konfigurationen – noget der ikke tidligere har været muligt. Der blev også udtrykt interesse for fremtidige tilvalg som fliser, el-ladere og udendørsbelysning, som systemet er forberedt til, selvom det ikke er krav i første version.

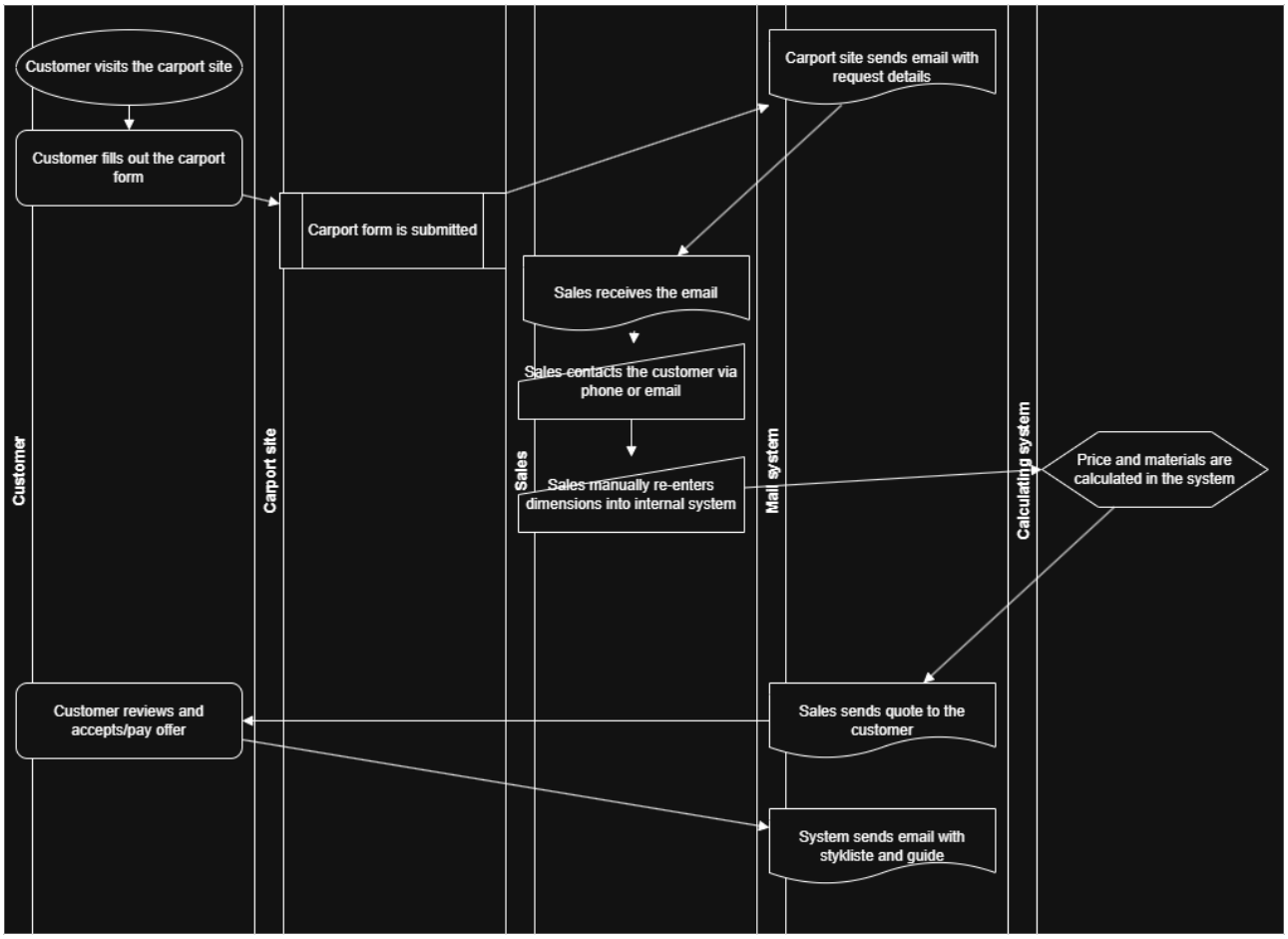
Der blev desuden nævnt et ønske om at sende en forklarende ordliste sammen med monteringsvejledningen. Dette er ikke med i første version, men kan tilføjes senere.

For at muliggøre frit materialevalg er det blevet nødvendigt at indføre dynamiske beregninger – særligt i forhold til stolpernes knækstyrke og bjælkers bæreevne. I modsætning til tidligere, hvor én fast stolpetype blev brugt, skal systemet nu kunne tilpasse konstruktionen ud fra de tekniske egenskaber ved hvert materialevalg.

Samlet spænder kravene bredt – fra konkrete ønsker til tekniske forudsætninger – og dækker både brugeroplevelsen og de interne kontrolmekanismer, der skal sikre kvalitet, fleksibilitet og forretningsværdi.

Arbejdsgange der skal IT-støttes

As-is

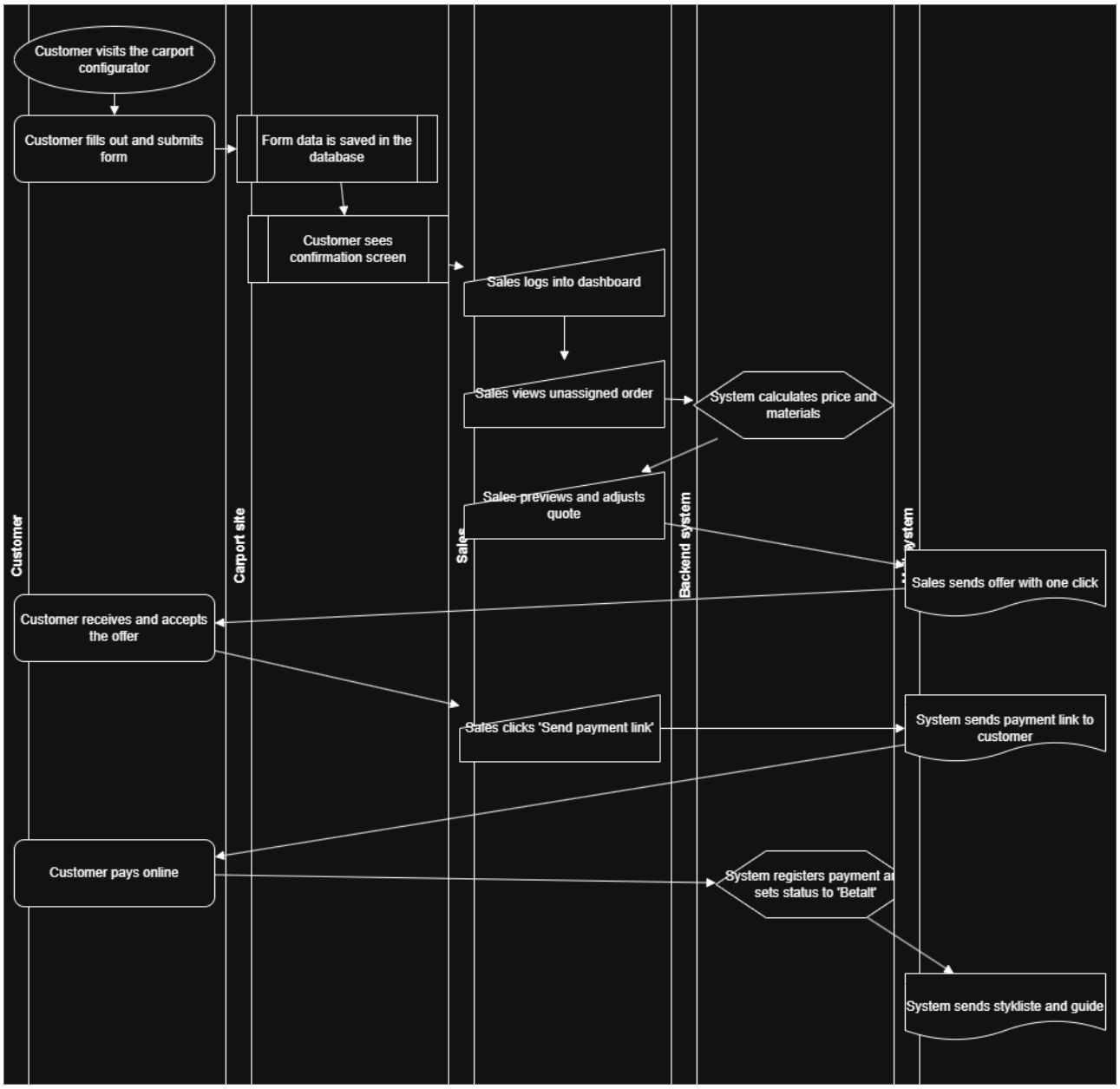


I den nuværende arbejdsgang besøger kunden carport-siden og udfylder en formular med sine ønsker og mål. Når formularen indsendes, sendes informationerne videre som en e-mail til salgsafdelingen. Herfra foregår det meste manuelt: En medarbejder læser e-mailen, kontakter kunden fortrinsvist telefonisk for at afklare eventuelle detaljer og behov, og sender efterfølgende en opfølgende e-mail med information.

Samtidig åbner medarbejderen et separat internt system, hvor alle mål og oplysninger fra formularen skal indtastes manuelt én gang til. Der findes ingen validering i formularen, så det er medarbejderens ansvar at opdage eventuelle fejl eller mangler i de indtastede data.

Når oplysningerne er indtastet, bruges systemet til at beregne pris og materialeforbrug, hvorefter der kan udarbejdes et tilbud. Hele denne proces involverer mange gentagelser og manuelle trin, hvilket øger risikoen for tastefejl og medfører unødvendigt tidsforbrug.

To-be



I den forbedrede arbejdsgang starter processen på samme måde: Kunden besøger carport-konfiguratoren og udfylder en formular med mål og ønsker. Men i stedet for blot at sende en e-mail til salgsafdelingen, gemmes oplysningerne nu direkte i systemets database. Kunden får straks vist en bekræftelsesskærm, hvilket skaber tryghed og transparens.

I salgsafdelingens dashboard vises nye kundehenvendelser automatisk under fanen "ikke-tildelte ordrer", hvor en medarbejder kan åbne og gennemgå informationerne. Alle mål og valg er allerede struktureret og klar til brug, uden at noget skal genindtastes manuelt. Herfra kan medarbejderen også ”tage” ordren, hvorefter den bliver en del af vedkommendes egen liste over aktive ordrer.

Systemet foretager automatisk en beregning af pris og materialeforbrug baseret på de indtastede data. Salgsmedarbejderen har mulighed for at gennemgå og justere tilbuddet direkte i systemet, hvorefter det sendes med ét klik. Kunden modtager tilbuddet via e-mail og kan bekræfte det.

Når kunden accepterer, klikker medarbejderen på en knap i systemet, som automatisk sender en betalingslink til kunden. Når betalingen er registreret, ændres ordrestatus automatisk til "Betalt", og systemet sender herefter en mail med både stykliste og monteringsvejledning.

Denne arbejdsgang minimerer manuelle indtastninger, sikrer korrekt dataoverførsel og giver både kunde og medarbejder en hurtigere og mere sammenhængende oplevelse.

User stories

User stories er udarbejdet på baggrund af kravindsamling, dialog med salgschefen og de forretningsmæssige og tekniske behov, der er opstået under analyse og udvikling. De beskriver funktionelle behov fra brugerens perspektiv og er nedbrudt i konkrete opgaver (tasks), estimeret med T-shirt sizes (S, M, L) og har tilhørende acceptkriterier. Den fulde samling findes i bilag.

Nedenfor vises tre udvalgte user stories, der dækker centrale områder som konfiguration, ordreoverblik og adgangskontrol:

User Story 2 – Clear Configuration Process

Som kunde ønsker jeg et tydeligt trin-for-trin forløb til at designe min carport, så jeg kan vælge størrelse og materialer uden forvirring.

- **Tasks:**
    - 2.1: Opret valg af dimensioner – *Medium*
    - 2.2: Tilføj materialevalg – *Medium*
    - 2.3: Indtast kundeinfo – *Medium*
    - 2.4: Bekræftelse og oversigt – *Medium*
    - 2.5: Visuelle indikatorer for færdige trin – *Small*
- **Acceptkriterier:**
    - Brugeren kan ikke springe trin over
    - Data gemmes lokalt undervejs
    - Opsummering viser korrekt indhold
    - Formularer validerer input (f.eks. gyldig e-mail, mål inden for grænser)

User Story 14 – Sales Dashboard Overview

Som sælger ønsker jeg at kunne se og filtrere alle ordrer, så jeg hurtigt kan håndtere åbne forespørgsler.

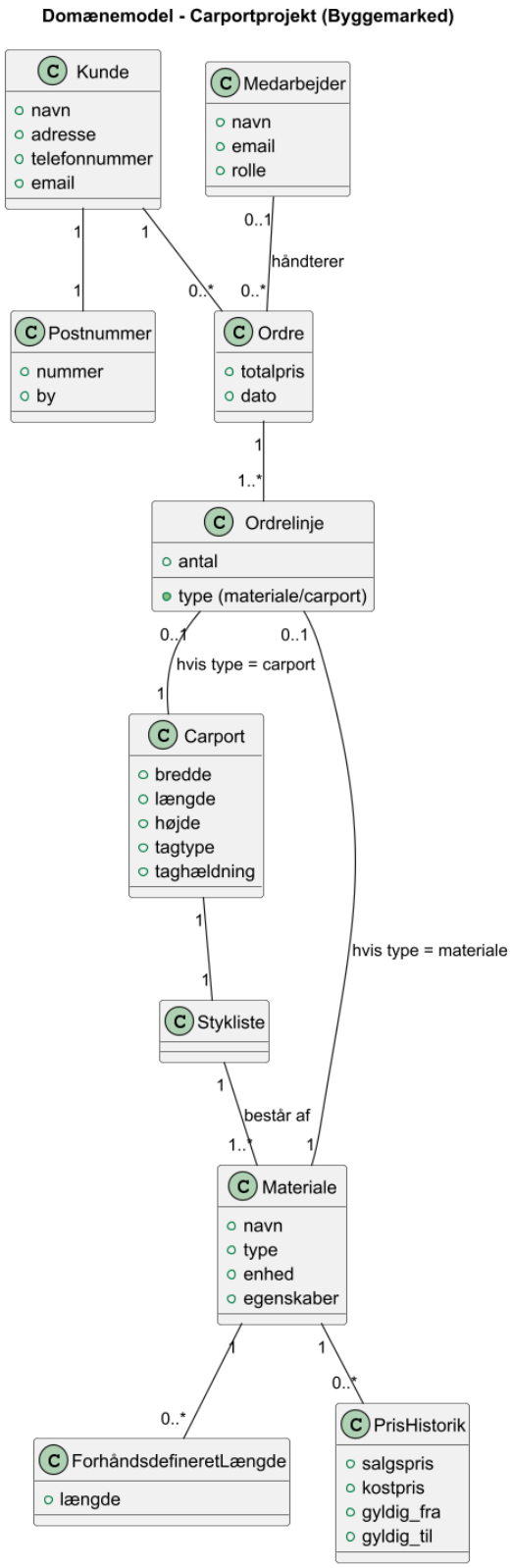
- **Tasks:**
    - 14.1: Vis ordretabel med nøgleoplysninger – *Medium*
    - 14.2: Tilføj filterfunktion for status – *Medium*
    - 14.3: Søge- og sorteringsfunktioner – *Small*
- **Acceptkriterier:**
    - Som udgangspunkt vises sælgerens egne ordrer
    - Mulighed for at se alle ordrer ved filter
    - Det er muligt at sortere efter dato og status

User Story 18 – Login for Internal Users

Som medarbejder ønsker jeg at kunne logge ind med rollebaseret adgang, så jeg kun ser funktioner, der er relevante for mig.

- **Tasks:**
    - 18.1: Login-formular med validering – *Small*
    - 18.2: Backend-autentificering med hashing – *Medium*
    - 18.3: Rollesystem og sessionstyring – *Medium*
    - 18.4: Adgangsbegrænsning til specifikke visninger – *Medium*
- **Acceptkriterier:**
    - Forkerte loginforsøg vises tydeligt
    - Roller afgør adgang til dashboards
    - Inaktive brugere logges automatisk ud efter inaktivitet

Domænemodel



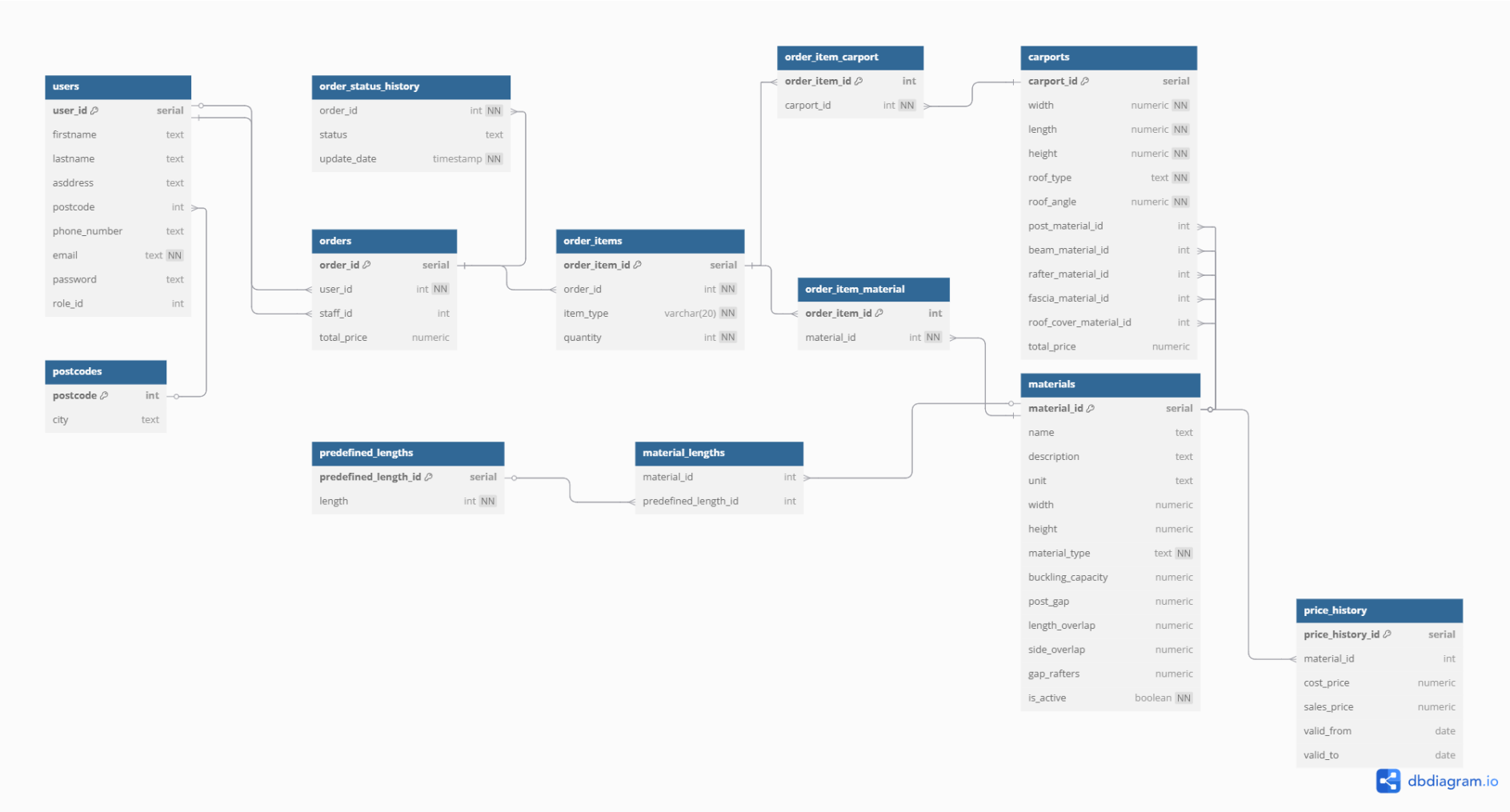
Domænemodellen illustrerer de vigtigste begreber og relationer i det forretningsdomæne, som systemet er udviklet til – nemlig bestilling og håndtering af carporte og tilhørende materialer hos et byggemarked. Modellen afspejler virkeligheden, uden at tage stilling til hvordan data konkret lagres eller struktureres teknisk.

En kunde kan afgive en eller flere ordrer. En ordre kan omfatte en færdigkonfigureret carport, løse byggematerialer eller begge dele. For at kunne repræsentere dette fleksibelt, er der introduceret en generel enhed, der repræsenterer hver enkelt bestilt vare – fx en carport eller et specifikt materiale.

En carport består i sig selv af flere materialer, såsom stolper, remme og tagbeklædning. Disse materialer samles i en stykliste, som beskriver præcist hvilke komponenter der indgår i den valgte konstruktion. Materialer kan også sælges separat uden at indgå i en carport.

Ud over kunder og ordrer indeholder modellen begreber som medarbejder (der håndterer ordrer), geografisk information (postnummer og by) samt historiske priser og faste længder på materialer. Disse elementer indgår for at afspejle de virkelige forhold og behov, der gør sig gældende i en byggevarevirksomhed som Johannes Fog.

ER-diagram


users
user_id serial
firstname text
lastname text
asddress text
postcode int
phone_number text
email text NN
password text
role_id int
postcodes
postcode int
city text
order_status_history
order_id int NN
status text
update_date timestamp NN
orders
order_id serial
user_id int NN
staff_id int
total_price numeric
order_items
order_item_id serial
order_id int NN
item_type varchar(20) NN
quantity int NN
order_item_carport
order_item_id int
carport_id int NN
order_item_material
order_item_id int
material_id int NN
carports
carport_id serial
width numeric NN
length numeric NN
height numeric NN
roof_type text NN
roof_angle numeric NN
post_material_id int
beam_material_id int
rafter_material_id int
fascia_material_id int
roof_cover_material_id int
total_price numeric
materials
material_id serial
name text
description text
unit text
width numeric
height numeric
material_type text NN
buckling_capacity numeric
post_gap numeric
length_overlap numeric
side_overlap numeric
gap_rafters numeric
is_active boolean NN
predefined_lengths
predefined_length_id serial
length int NN
material_lengths
material_id int
predefined_length_id int
price_history
price_history_id serial
material_id int
cost_price numeric
sales_price numeric
valid_from date
valid_to date
dbdiagram.io

ER-diagrammet danner grundlag for, hvordan data struktureres og behandles i systemets relationelle PostgreSQL-database. Modellen tager udgangspunkt i domænemodellen og er tilpasset med hensyn til nøgler, relationer og normalisering.

Struktur og normalisering

Alle tabeller er normaliseret til 3. normalform med entydige primærnøgler og fremmednøgler, der sikrer referentiel integritet:

- users dækker både kunder og medarbejdere via en role_id.
- postcodes er adskilt for at undgå redundans mellem bynavn og postnummer.
- orders refererer til både kunde (user_id) og ansvarlig medarbejder (staff_id).
- order_status_history gemmer statushistorik via en sammensat primærnøgle (order_id, update_date).

Ordrelinjer og polymorfi

Ordrer kan indeholde både carporte og løse materialer, håndteret via polymorfi i order_items. Her afgør item_type, om linjen refererer til order_item_carport eller order_item_material.

- Strukturen muliggør udvidelse med nye produkttyper.
- En CHECK-constraint sikrer gyldige værdier i item_type.

Carporte og materialer

En carport refererer til op til fem materialetyper via fremmednøgler og gør det muligt at ændre materialer enkeltvis.

Materialer har:

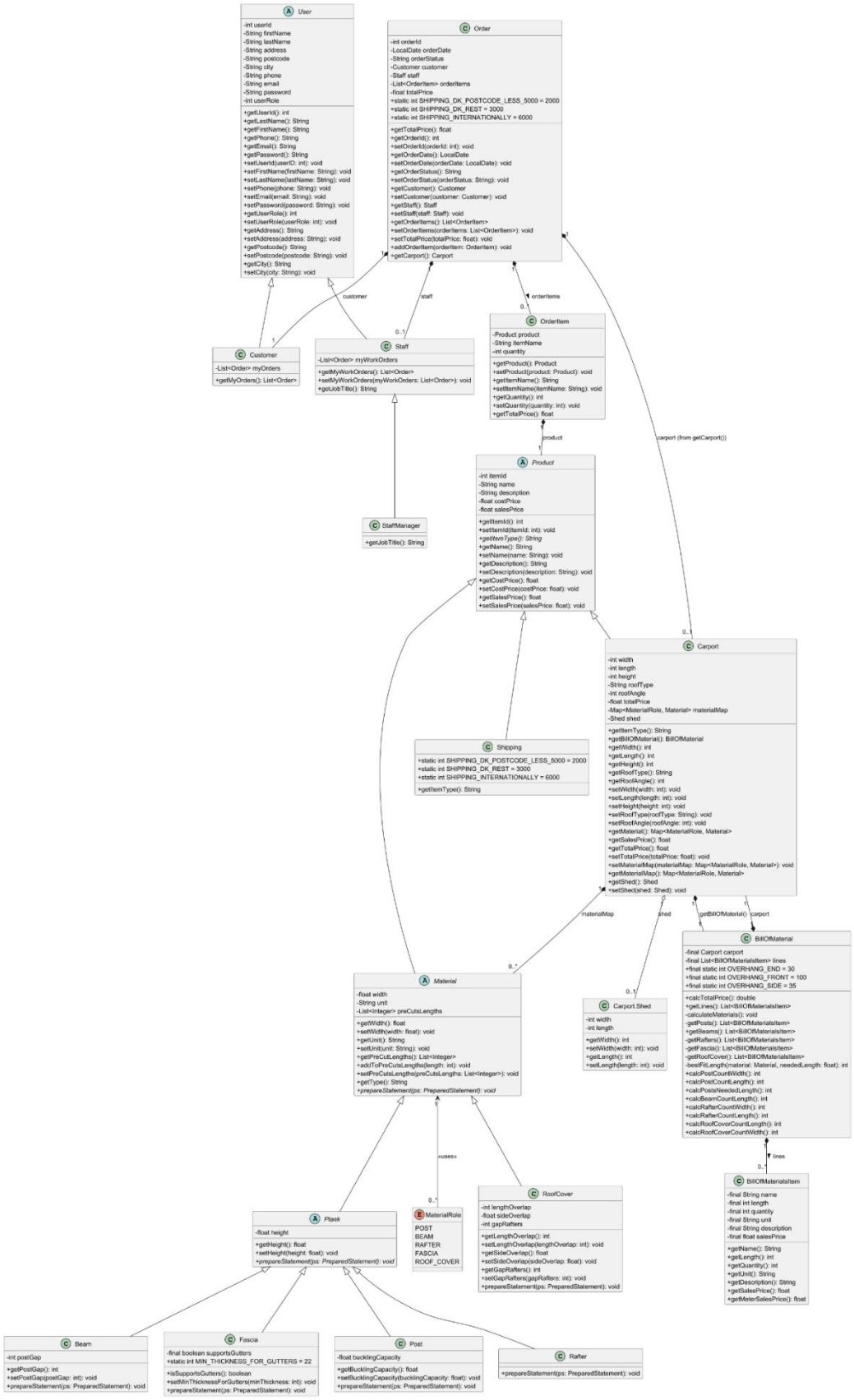
- Pris-historik i price_history.
- Mange-til-mange-relation til længder via material_lengths.

Designvalg og fleksibilitet

Materialer er gemt i én samlet tabel med et type-felt og nullable kolonner til særlige egenskaber (fx knækstyrke, overlap). Dette forenkler datamodellen og matcher projektets omfang, samtidig med at det understøtter polymorfi i koden.

Strukturen er valgt for lav kompleksitet, men tillader senere opsplitning i separate tabeller, hvis systemet udvides.

Klassediagram



Klassediagrammet illustrerer den objektorienterede struktur og udgør fundamentet for systemets programarkitektur. Det bygger på domænemodellen og databasen, men inkluderer også Java-specifikke metoder og attributter, der afspejler implementeringen.

Brugerstruktur

User er en abstrakt overklasse for Customer, Staff og StaffManager, der deler fælles egenskaber og metoder. Arven sikrer, at fx kun Customer kan tilgå egne ordrer, mens ansatte har adgang til arbejdsmæssige ordrer. Roller adskilles funktionelt men håndteres polymorfisk.

Ordrestruktur

En Order består af flere OrderItem-objekter og refererer til både kunden og en eventuel ansvarlig medarbejder. OrderItem kombinerer produkt og kvantitet og kan repræsentere både carporte og materialer. Ordren beregner totalpris og indeholder logik til at identificere carportkomponenten.

Produktstruktur

Product er en abstrakt klasse for alle produkttyper (fx Carport, Material, Shipping) og sikrer fælles behandling. Carport rummer mål, tagtype, og har evt. tilknyttet Shed og BillOfMaterial, som beregner og viser nødvendige materialer.

Materialer og stykliste

Material er abstrakt og specialiseres via Plank, som har underklasser som Post, Beam, Rafter, Fascia og RoofCover. Disse indeholder egenskaber som bæreevne, overlap og afstand. Et BillOfMaterial knyttes til en carport og indeholder en række BillOfMaterialsItem, som angiver længde, mængde og pris for hvert materiale.

Roller og fleksibilitet

Materialets funktion defineres med MaterialRole (enum), fx POST, BEAM, ROOF_COVER, og mappes i Carport til det konkrete materiale via et materialMap. Denne struktur sikrer fleksibel udskiftning og korrekt anvendelse i konstruktionen – uden at ændre systemets kerne.

Navigationsdiagram

Mockup

Der blev udarbejdet et interaktivt Figma-mockup før implementering, som visualiserer brugerrejsen og layoutet på de vigtigste sider i bestillingsflowet. Det fungerede som grundlag for design og frontend, og har sammen med navigationsdiagrammet gjort det nemt at vurdere UI/UX og få tidlig feedback fra kunden.

Link til mockup: Figma - https://www.figma.com/proto/S3FPGu4FtiiFZsKL0go8LE/carport?node-id=6-55&p=f&t=l9tcWYEDJReDUdRk-1&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=6%3A55&show-proto-sidebar=1

Kundevisning

Navigationsdiagrammet viser kundens rejse fra det offentlige website til det beskyttede customer/dashboard. Bestillingsflowet består af fire trin: målvalg, materialer, kontaktinfo og bekræftelse.

Efter bestilling får kunden tilsendt en bekræftelsesemail hvori loginoplysninger også findes. Et kodeord autogeneres (fornavn + postnummer). Kunden kan nu med email og kodeord få adgang til customer/dashboardet med overblik over ordrer og profiloplysninger.

Betalingssiderne (/payment og /payment/confirm) tilgås via et link sendt af en medarbejder. De er knyttet til en specifik ordre og valideres på baggrund heraf.

En fejlside (/public-error) håndterer uautoriseret adgang og tekniske fejl. Se navigationsdiagram i bilag, figur 1.

Medarbejdervisning

Navigationsdiagrammet for medarbejdersiden viser, hvordan Fog's ansatte bruger det interne dashboard baseret på rollebaseret adgangskontrol. Brugeren skal være logget ind, og den tildelte rolle – enten Staff eller StaffManager – afgør adgangen til specifikke funktioner.

Efter login præsenteres medarbejderen for en fast topmenu, som er tilgængelig på alle dashboardsider og giver adgang til: ordrer, kunder, materialer, medarbejdere (kun for salgschefer), carportoprettelse, profil og e-mailfunktioner. Dette gør navigationen hurtig og effektiv uden behov for at vende tilbage til en forside.

Dashboardets hovedfunktion er overblik over ordrer, som kan åbnes for at se detaljer, kundeoplysninger og – hvis relevant – carportens stykliste og tegning.

Materialefunktionen gør det muligt at søge, filtrere og redigere materialer. Kun salgschefer kan oprette eller slette materialer, styret via rollen.

Under kundeadministration kan ansatte redigere kundeoplysninger, mens salgschefer har adgang til medarbejderadministration, hvor brugere kan oprettes og redigeres.

Profilsektionen gør det muligt at opdatere egne oplysninger, og e-mailmodulet muliggør afsendelse af dokumenter og betalingslinks direkte til kunden.

En dedikeret fejlside håndterer uautoriseret adgang og tekniske problemer. Se navigationsdiagram i bilag, figur 2.

Arkitektur

Systemet er opbygget efter en MVC-inspireret arkitektur (Model–View–Controller), som adskiller præsentation, forretningslogik og datatilgang. Denne struktur gør systemet overskueligt, testbart og nemt at udvide – vigtigt i et domæne som bestilling og konfiguration af specialtilpassede carporte.

Pakkeoversigt

app.controller

Indeholder controllere for både kunder og medarbejdere. Mapperne customer og dashboard afspejler brugerrollerne og håndterer HTTP-requests, inputvalidering og kald til services og visninger.

app.entities

Denne pakke repræsenterer domænemodellen og indeholder de centrale objekter, systemet arbejder med. Den er struktureret i tre hovedområder: orders, users og products.

- **orders**
  Denne del indeholder klasser relateret til ordrer og ordrelinjer. En Order repræsenterer en kundeforespørgsel og kan bestå af både en carport og/eller materialer. Ordren består af en samling OrderItem-objekter, hvor hver linje kobler et produkt med en mængde.

  Strukturen muliggør fleksible ordretyper og sikrer samtidig beregning af totalpris samt relationer til både kunde og evt. ansvarlig medarbejder. orders-pakken er central for forretningslogikken og anvendes i både kundevisning, dashboard og intern behandling.

- **users**
  Pakken indeholder den abstrakte User-klasse, der dækker fælles informationer som navn, adresse og login. De tre konkrete brugerroller – Customer, Staff og StaffManager – nedarver fra denne klasse og tilføjer rolle-specifik funktionalitet.

  Denne struktur understøtter rollebaseret adgang, differentierede brugeroplevelser og forenklet autorisationslogik. Klassen er tæt integreret med login-systemet, ordrer og dashboardvisninger.

- **products**
  Product er en abstrakt klasse, som samler fælles attributter for alle produkttyper: navn, beskrivelse, kostpris og salgspris. Både enkle produkter som Material og sammensatte som Carport nedarver herfra, hvilket sikrer ensartet behandling i fx ordrelinjer og prisberegninger.

  - **Carport**
    Indeholder klasser til konfiguration af carporte: mål, tagtype og materialer. Hver carport har et BillOfMaterial, som dynamisk beregner nødvendige materialer ud fra valgte dimensioner og typer. Pakkens klasser samarbejder med materialelogik og tegnefunktionalitet og udgør kernen i konfigurationssystemet.

  - Material
    Indeholder alle konstruktionens materialer og består af den abstrakte Material-klasse samt tekniske underpakker:

    - planks: fx Post, Beam, Rafter, Fascia med egenskaber som knækstyrke og spændvidde.
    - roof: fx RoofCover, der håndterer tagoverlap og spærafstand.

    Materialer tildeles deres funktion via MaterialRole (enum), fx POST eller ROOF_COVER, og indgår i carportens materialMap. Denne struktur understøtter fleksibel udskiftning og tekniske beregninger, uden at gå på kompromis med domænestruktur.

app.persistence.mappers

Indeholder mapperklasser, som håndterer CRUD-operationer mod databasen. Hver mapper oversætter mellem entiteter og SQL – fx OrderMapper, MaterialMapper, UserMapper. Det sikrer et entydigt og centraliseret datatilgangslag.

app.service

Indkapsler forretningslogik og transformationer, som ellers ville overbelaste controllerne. Gør det muligt at genbruge logik på tværs af brugertyper og holde systemet modulært og vedligeholdelsesvenligt.

app.utils

Indeholder hjælpeklasser uden domæneafhængighed, fx formattering og hashing.

app.exceptions

Samler systemets custom exceptions, som fx Databaseexception.

app.config

Rummer applikationens konfiguration, fx initialisering af DatabaseConnectionPool og generelle settings.

Særlige forhold

Routing/Controllers

I dette projekt er der gjort en række overvejelser og valg i forhold til struktur, sikkerhed og brugeroplevelse. Applikationen er bygget op omkring MVC-arkitekturen, hvor routingen er samlet i én controller, og hver controller håndterer en specifik del af forretningslogikken.

Til adgangsstyring anvendes session-variabler. Når en bruger logger ind, gemmes en `currentUser` i sessionen, og hvis brugeren er leder (`StaffManager`), sættes desuden en `isManager`-variabel. Disse anvendes på tværs af applikationen til at differentiere indholdet i visninger og begrænse adgangen til følsomme områder.

En vigtig sikkerhedsmekanisme er implementeret med et globalt filter, som sørger for, at kun medarbejdere har adgang til dashboardet. Dette filter tjekker, om brugeren er logget ind og er af typen `Staff`, og omdirigerer ellers til login-siden. På denne måde sikres det, at uautoriserede brugere ikke får adgang til administrative funktioner.

Fejlhåndtering foregår konsekvent i hele applikationen ved hjælp af `try-catch` blokke. Hvis der opstår en fejl, vises en brugervenlig fejlside med en passende besked. Dette bidrager til en ensartet og professionel brugeroplevelse og skjuler tekniske detaljer fra brugeren.

Der arbejdes med to centrale brugertyper i databasen: `Customer` og `Staff`. En `Staff` kan have yderligere rettigheder som `StaffManager`, hvilket giver adgang til udvidede funktioner som materialestyring og medarbejderadministration. Disse roller anvendes i både forretningslogik og visninger og gør det muligt at målrette funktionaliteten til forskellige brugere.

En særlig funktionalitet i projektet er den dynamiske SVG-tegning af carporten, som genereres ud fra brugerens input og materialevalg. Tegningen vises i både kunde- og medarbejderdelen af systemet og understøttes af en stykliste, som udregnes og vises for hver ordre. Dette giver både brugeren og medarbejderen et klart overblik over den valgte konfiguration og de tilhørende materialer.

Orders

I systemet håndteres ordrer gennem Order og OrderItem-entiteter. Hver ordre indeholder information om kunde, medarbejder, status, dato og en samling af ordrelinjer (stykliste).

Brugerroller:
Hver ordre knyttes til en Customer og evt. en Staff, som begge er hentet fra bruger-tabellen med forskellige roller (kunde og medarbejder). Disse bruges både til rettighedsstyring og visning i brugerfladen.

Styklisteberegning:
Hver OrderItem refererer til et produkt og har en metode getTotalPrice() som udregner stykpris × antal. Order summerer totalpriser fra alle ordrelinjer og kan dermed beregne en samlet pris.

Users

Brugertyper og roller

I systemet skelnes der mellem forskellige brugertyper, som hver har adgang til specifikke funktioner. Dette er implementeret i Java ved hjælp af arv, hvor den abstrakte klasse User definerer fælles information for alle brugere, herunder navn, adresse, telefonnummer, e-mail og adgangskode.

Ud fra denne fælles base findes tre konkrete klasser:

- Customer, som repræsenterer almindelige kunder, der kan oprette og se egne ordrer.
- Staff, som repræsenterer medarbejdere, der kan tilgå og behandle kundeordrer.
- StaffManager, som arver fra Staff, men har udvidede rettigheder i systemet, såsom adgang til ledelsesværktøjer og medarbejderoversigter.

Disse rettigheder er delvist illustreret i navigationsdiagrammet og bliver også beskrevet særskilt i rapporten under afsnittet om adgangskontrol.

Ved login kontrolleres brugerens adgangsniveau på baggrund af den rolle, der er gemt i databasen som et heltal (user_role). Mere om selve instansieringen af de forskellige brugerklasser kan læses i afsnittet om mapperne. Det betyder, at selvom alle brugertyper teknisk set gemmes i samme database-tabel, bliver de i programmet håndteret forskelligt afhængigt af deres rolle.

Denne tilgang gør det muligt at udnytte polymorfi i programmet, hvor hver brugerklasse kan have egne specialiserede metoder og samtidig dele fælles funktionalitet gennem superklassen User.

Product

Materialer

Systemets entiteter er designet med fokus på klar rollefordeling, brug af arv og datamodellering der afspejler virkeligheden. Abstrakte klasser som `User`, `Product` og `Material` bruges til at samle fælles logik, mens konkrete klasser som `Customer`, `Post` og `RoofCover` tilføjer specifik funktionalitet.

Produkt- og materialehierarkiet er struktureret, så det både understøtter polymorfi og teknisk specialisering i beregninger. Det gør systemet let at udvide og ændre, uden at bryde eksisterende funktionalitet.

Material og prepareStatement-metoden

I `Material`-klassen findes den abstrakte metode `prepareStatement(PreparedStatement ps)`. Den tvinger alle underklasser til at definere, hvordan de indsætter sig selv i databasen med alle relevante felter.

Denne metode er en del af systemets JDBC-arkitektur og gør det muligt at skrive kode, der arbejder med `Material`-objekter polymorfisk, uden at kende deres konkrete type. Underklasser som `RoofCover`, `Beam` og `Post` implementerer denne metode og tilføjer egne felter som overlap, postGap eller knækstyrke.

Dette valg sikrer fleksibilitet og korrekt databaseadfærd, samtidig med at nye materialetyper nemt kan tilføjes uden at ændre eksisterende mapper-kode.

Se også: Et konkret eksempel på, hvordan prepareStatement er implementeret i klassen Post, findes i afsnittet "Udvalgte kodeeksempler".

Plank og specialiserede materialeklasser

I begyndelsen blev alle trækomponenter (stolper, spær, bjælker og beklædning) repræsenteret som én fælles klasse: `Plank`. For at afspejle virkelige konstruktionsforskelle og fremtidssikre systemet, blev de adskilt i fire underklasser.

Denne adskillelse gør det muligt at tilføje egenskaber som **knækstyrke** for stolper, **understøttelse af tagrender** for beklædning, og **afstandskrav** for bjælker med mere. Ved at lade hver materialeunderklasse håndtere sine egne tekniske krav, undgår man at centralisere al logik ét sted, hvilket ville føre til kompleks og fejlbehæftet kode.

Samtidig betyder denne struktur, at systemet nemt kan **udvides i fremtiden**. Hvis der eksempelvis opstår behov for at understøtte nye materialetyper, som tager højde for snebelastning, vindmodstand eller bæreevne under hældning, kan disse egenskaber tilføjes i en ny specialiseret klasse – uden at ændre eksisterende logik. Det gør arkitekturen både robust og fleksibel, og sikrer, at tekniske og forretningsmæssige regler kan udvikles uafhængigt af hinanden.

Carport

I Carport-klassen er Shed defineret som en indre, statisk klasse. Den repræsenterer et valgfrit redskabsskur, der kan tilføjes som en del af carportens konfiguration. Ved at placere Shed i Carport-klassen fremhæves den semantiske sammenhæng – et skur eksisterer kun i konteksten af en carport.

Det at klassen er statisk betyder, at den kan instansieres og behandles uafhængigt af en konkret Carport-instans, hvilket gør den lettere at arbejde med i isolerede sammenhænge, som fx konfigurationsværktøjer eller tests. Samtidig signalerer klassens placering, at Shed ikke er tænkt som en generel bygningstype, men udelukkende som en del af carportmodellen.

Styklisteberegner

Styklisteberegningen er rygraden i systemet og anvendes både til priskalkulation og som grundlag for visuelle og tekniske tegninger. Systemet genererer en komplet materialeliste ud fra den carport-konfiguration, brugeren har valgt — fx længde, bredde og valgte materialetyper.

Dynamisk og ikke-persistent model

En vigtig designbeslutning har været, at styklisten ikke gemmes i databasen. I stedet gemmes kun den valgte carportkonfiguration, inkl. dimensioner og valgte materialer (én for hver rolle). Når carporten hentes fra databasen, dannes styklisten automatisk ud fra konfigurationen. Det gør løsningen mere fleksibel og forenkler databasen, fordi man undgår at gemme detaljerede styklister separat. Samtidig sikres det, at ændringer i beregningslogik eller materialedata altid anvendes konsekvent.

Generering af stykliste

Når en carport skal visualiseres eller der skal genereres en ordre, oprettes en instans af BillOfMaterial. Denne klasse indeholder en metode til at generere en komplet stykliste ud fra dimensionerne og de valgte materialer. Hver linje i styklisten er repræsenteret af en BillOfMaterialsItem, som indeholder oplysninger om navnet på komponenten, længde, antal, enhed, beskrivelse og pris.

Styklisten udregnes ved at bruge tekniske egenskaber fra materialerne, fx tagoverlap, knækstyrke og afstand mellem stolper.

Materialer og enum-roller

Materialerne kategoriseres efter deres funktionelle rolle i konstruktionen vha. enum-typen MaterialRole. Denne anvendes flere steder i systemet, fx:

- Til at knytte det rigtige materiale til sin rolle i Carport via en Map<MaterialRole, Material>

- Til at generere grupperede visninger af styklisten (fx alle POST-komponenter)

- Til at bestemme rækkefølgen og layout ved visuel rendering

Specialiserede materialeklasser

For at tage højde for tekniske krav indeholder visse materialer specialfelter:

- Post har feltet bucklingCapacity (knækstyrke), som bruges til at afgøre, hvor meget belastning én stolpe kan bære.

- Beam definerer postGap, dvs. hvor langt der må være mellem stolper afhængigt af bjælkens bæreevne.

- RoofCover har lengthOverlap og sideOverlap, som bruges til at beregne hvor mange tagplader der kræves for at dække hele taget uden utætheder.

- Fascia angiver, om materialet understøtter tagrender afhængigt af tykkelse.

Ved beregning af f.eks. antal stolper tages der højde for både knækstyrken fra Post og den tilladte afstand fra Beam, så der opnås en sikker og stabil konstruktion. Den endelige placering og mængde afhænger derfor af en kombination af flere materialespecifikke egenskaber.

Mapperstruktur og databaseadgang

Systemets databasetilgang er struktureret omkring mapperklasser, hvor hver mapper har ansvar for en bestemt del af domænet, f.eks. brugere, materialer, ordrer og carporte. Disse klasser fungerer som bindeled mellem databasen og de tilhørende Java-entiteter og bruger JDBC med prepared statements for at undgå SQL-injektion og sikre korrekt typehåndtering.

Alle mapper benytter `connectionPool` for at optimere ressourceforbrug og gøre databasetilgange genanvendelige. Undtagelser håndteres konsekvent ved at kaste en `DatabaseException` med beskrivende fejlbeskeder og en oprindelig SQLException som årsag. Dette sikrer både fejlsporing og robusthed.

Systemet anvender polymorfi aktivt – især i `MaterialMapper`, hvor `instanceof` bruges til at identificere konkrete underklasser af `Material`, så den korrekte `prepareStatement()`-logik anvendes ved indsættelse i databasen. Dette gør det muligt at håndtere forskellige materialetyper ensartet i mapperkode, men stadig sikre korrekt behandling af specialfelter som fx `bucklingCapacity` i `Post` eller `overlap` i `RoofCover`.

I OrderMapper håndteres noget af den mest komplekse datalogik i systemet. Mapperen forbinder ordrer med både brugere, carporte og tilknyttede materialer og genskaber dermed et fuldt objekthierarki. Dette sker gennem flere SQL-forespørgsler og join-operationer, som tilsammen samler alle relaterede data til én konsistent og sammenhængende struktur i Java.

`CarportMapper` anvender en `Map<MaterialRole, Material>` til at gemme og hente materialer knyttet til en carport. Denne struktur gør det muligt at bruge enum-værdier som nøgler og dermed nemt gruppere materialer efter deres rolle i konstruktionen, f.eks. `POST`, `BEAM` eller `ROOF_COVER`. Det giver både struktur og fleksibilitet i opbygningen af carportobjekter, og gør mapperen let at udvide med nye roller og materialetyper.

Users

Metoden `mapUser(ResultSet rs)` i `UserMapper` er ansvarlig for at genskabe den korrekte brugerinstans ud fra databasen. Systemet arbejder med tre typer brugere: `Customer`, `Staff` og `StaffManager`, som alle arver fra den abstrakte klasse `User`.

I databasen er brugertypen gemt som en integer i feltet `role_id`. Ved læsning bruges dette tal til at afgøre hvilken subklasse der skal instansieres. Dette er et centralt punkt i systemets brug af polymorfi, da det sikrer at logikken omkring adgang, dashboardvisninger og rettigheder bindes korrekt sammen med den konkrete brugertype.

Eksempelvis vil en bruger med `role_id = 1` blive oprettet som `Customer`, mens `role_id = 3` oprettes som `StaffManager`, som har adgang til yderligere funktioner i systemet. Hvis der findes en ukendt rolle i databasen, kaster metoden en `SQLException`, hvilket er en vigtig sikkerhedsmekanisme for at undgå fejltilstande.

Metoden indlæser desuden alle relevante brugerfelter fra databasen (navn, adresse, email, osv.) og overfører dem til konstruktøren for den valgte brugerklasse.

Denne metode er et godt eksempel på, hvordan databasen og domænemodellen forbindes på en fleksibel måde, hvor systemet kan udvides med nye brugertyper uden at ændre grundlæggende logik andre steder i systemet.

Se kodeeksemplet i afsnittet “Udvalgte kodeeksempler” for implementeringen.

Material – createMaterial

Metoden `createMaterial(Material material)` i `MaterialMapper` er ansvarlig for at indsætte et materialeobjekt i databasen. Systemet benytter polymorfi aktivt ved at lade den konkrete `Material`-subklasse selv definere, hvordan dens felter indsættes via `prepareStatement()`. Dette sikres ved, at `Material` definerer en abstrakt metode `prepareStatement(...)`, som alle underklasser – f.eks. `Post`, `Beam` eller `RoofCover` – skal implementere.

På den måde håndteres både fælles felter som navn og pris samt specialiserede felter som knækstyrke (i `Post`) og overlap (i `RoofCover`) på en fleksibel og skalerbar måde. Denne tilgang muliggør polymorfi i praksis og betyder, at mapperklassen ikke behøver kende til de enkelte underklassers felter. Det skaber en adskillelse af ansvar og gør det nemt at udvide systemet med nye materialetyper uden at ændre mapperens

logik.

Som vist og beskrevet i kodeeksemplerne for `Material` og `Post`, kaldes `prepareStatement(...)` direkte på det givne materialeobjekt, og det er op til objektet selv at definere hvordan det skal indsættes korrekt i databasen.

Efter materialet er oprettet i hovedtabellen, håndterer metoden også materialets længder (preCuts), som er gemt i en liste.

Denne metode er et godt eksempel på brugen af polymorf databaseadgang og effektiv datalagring. Den kombinerer fleksibilitet, skalerbarhed og præcis domænemodel-logik i én sammenhængende implementering.

Order – createOrder

Metoden `createOrder(Order order)` i `OrderMapper` er ansvarlig for at gemme en komplet ordre i databasen – herunder selve ordren, dens status og tilhørende ordrelinjer (`OrderItem`). Det kræver flere på hinanden afhængige indsættelser, som tilsammen danner en logisk helhed. Derfor udføres hele operationen inden for én database-transaktion.

Transaktionen håndteres eksplicit med `connection.setAutoCommit(false);`. Ved at slå autocommit fra, sikrer man, at alle ændringer først bliver gemt permanent, når man kalder `connection.commit();`. Hvis der opstår en fejl undervejs, kan `connection.rollback();` kaldes, så alle ændringer fortrydes. Det beskytter mod inkonsistente data og sikrer,
at enten hele ordren bliver gemt – eller intet bliver gemt.

Denne tilgang er afgørende for dataintegriteten i systemet, da ordrer består af mange delelementer, der er gensidigt afhængige. Hvis f.eks. kun ordren bliver oprettet, men ikke dens ordrelinjer, vil systemet stå i en ufuldstændig tilstand.

Dette er også brugt i flere andre mappere for at sikre dataintegriteten i systemet.

SVG

Systemets visualisering af carporte er implementeret med SVG-teknologi (Scalable Vector Graphics), som muliggør dynamisk generering af tegninger baseret på brugerens carportkonfiguration. Denne funktion er ikke blot dekorativ, men har også en praktisk rolle i formidlingen af dimensioner, struktur og ikke mindst som vejledning.

Tegnesystemet består af flere specialiserede klasser med tydelig ansvarsfordeling:

• `CarportSvg` fungerer som wrapper og opretter en SVG med de korrekte mål baseret på carportens bredde og længde. Den bruger en intern `Svg`-instans til at opbygge selve grafikken.

• `Svg` er en hjælpeklasse som tilbyder metoder til at tilføje SVG-elementer såsom rektangler, streger, pile og tekst. Den håndterer selv opbygningen af korrekt XML og tilføjer nødvendige metadata såsom viewBox og pil-definitioner.

• `SvgDrawingService` indeholder selve tegnelogikken. Her implementeres konkrete metoder til at tilføje visuelle repræsentationer af f.eks. stolper (`posts`), spær, mål og dimensioner. Tegningen er baseret på data fra `Carport`-objektet og materialevalgene fra stykliste eller carportkonfiguration.

Ved at adskille selve tegnelogikken (i `SvgDrawingService`) fra præsentationslaget (`Svg`) og konfigurationen (`CarportSvg`), opnås høj genbrugelighed og fleksibilitet. Systemet kan fx nemt udvides med sidevisning eller skurtegning uden at omskrive den eksisterende struktur.

SVG-klasserne er vigtige, da de kombinerer forretningslogik og visuel feedback i én løsning, hvilket skaber transparens for brugeren og understøtter arbejdsgange internt.

PasswordUtil – Adgangskodehåndtering med hashing og salt

`PasswordUtil` håndterer alle adgangskoder i systemet og anvender den sikre algoritme BCrypt til at beskytte brugernes data. Når en bruger opretter en adgangskode, kaldes `hashPassword(...)`, som genererer en hash med en indbygget, tilfældig salt. Denne salt er unik for hver adgangskode og betyder, at selv to brugere med samme kodeord vil få forskellige hash-værdier.

Salt er vigtigt, fordi det beskytter mod såkaldte rainbow table-angreb, hvor man prøver at matche kendte hash-værdier med kodeord. Ved at kombinere kodeordet med en tilfældig salt, bliver sådanne angreb stort set umulige.

Kodeord uden hashing og salt	Kodeord med hashing og salt
Pw2	"$2a$10$LxX3YiRR9ozVD.rDpvs9fOqESjgRTWHqiq8S3ioWiXXm/mwh.82Qu"

SendGrid og e-mailkommunikation

Systemet anvender SendGrid til at sende automatiserede e-mails til kunder og brugere. Dette inkluderer blandt andet:
• Bekræftelser ved kontooprettelse
• Kommunikation omkring ordrer og tilbud
• Andre former for beskedudveksling direkte fra medarbejdere til kunder

Der er oprettet en række e-mail-skabeloner (templates) direkte i SendGrid, og hver skabelon har en unik `templateId`. Når en besked skal sendes fra systemet, refererer man til denne ID for at vælge hvilken skabelon der skal bruges.

Det er bevidst valgt at holde selve SendGrid-templaten relativt enkel. I stedet for at lade skabelonen stå for åbningshilsen ("Hej [Navn]") og afslutning ("Venlig hilsen ..."), genereres disse formalia allerede i Java-laget og vises direkte i tekstboksen via Thymeleaf. Det betyder, at medarbejderen ser den komplette besked – inklusive hilsen og afslutning – mens de skriver eller tilpasser den.

På den måde undgår man usikkerhed om, hvad der bliver tilføjet automatisk. Medarbejderen skal ikke gætte på, om der fx står "Hej" i templaten i forvejen. Det giver bedre kontrol, færre fejl og en mere ensartet kommunikation.

Metoden som anvendes til at sende mails, er bygget så den modtager parametre som modtagerens e-mailadresse, emnelinje, beskedens indhold og `templateId`. Ved afsendelse indsættes indholdet i den

dynamiske del af templaten via SendGrids API.

Et eksempel på hvordan en e-mail sendes fra systemet kan ses i afsnittet “Udvalgte kodeeksempler”.

Udvalgte kodeeksempler

Nedenfor ses udvalgte kodeeksempler, som illustrerer nogle af de vigtige principper beskrevet i 'Særlige forhold'-afsnittet.

Routing/controllers

Adgangskontrol med before-filter

Dette filter sikrer, at kun medarbejdere har adgang til dashboardet:

```
app.before("/dashboard/*", ctx -> {
    User user = ctx.sessionAttribute("currentUser");
    if (user == null || !(user instanceof Staff)) {
        ctx.redirect("/login");
    }
});
```

Login-håndtering i AuthController

Eksempel på hvordan login og rollevalidering håndteres:

```
User user = userService.authenticate(username, password);
if (user instanceof StaffManager) {
    ctx.sessionAttribute("isManager", true);
}
ctx.sessionAttribute("currentUser", user);
```

Entities

Material – preparedstatement

I `Material`-klassen er `prepareStatement` defineret som en abstrakt metode. Det sikrer, at alle underklasser selv håndterer, hvordan deres data skal indsættes i databasen.

```
public abstract void prepareStatement(PreparedStatement ps) throws SQLException;
```

Denne metode viser, hvordan en `Post`-instans indsætter sine data i en `PreparedStatement`. Fælles felter som navn og enhed håndteres først, efterfulgt af det specialiserede felt `bucklingCapacity`. Øvrige felter, som ikke er relevante for denne type materiale, sættes som `NULL`.

```
@Override

public void prepareStatement(PreparedStatement ps) throws SQLException {

    ps.setString(1, this.getName());

    ps.setString(2, this.getDescription());

    ps.setString(3, this.getUnit());

    ps.setFloat(4, this.getWidth());
```

```
    ps.setFloat(5, this.getHeight());
    ps.setString(6, this.getClass().getSimpleName());
    ps.setFloat(7, this.getBucklingCapacity());
    ps.setNull(8, Types.NUMERIC);
    ps.setNull(9, Types.NUMERIC);
    ps.setNull(10, Types.NUMERIC);
    ps.setNull(11, Types.NUMERIC);
}
```

Mappers

User – Roller

Metoden mapUser genskaber den korrekte brugerinstans baseret på role_id fra databasen. Den bruges til at oprette enten en Customer, Staff eller StaffManager, afhængigt af brugerens rolle.

```
private static User mapUser(ResultSet rs) throws SQLException {
    int role = rs.getInt("role_id");
    String firstname = rs.getString("firstname");
    String lastname = rs.getString("lastname");
    String address = rs.getString("address");
    String postcode = rs.getString("postcode");
    String city = rs.getString("city");
    String phone = rs.getString("phone_number");
    String email = rs.getString("email");
    String password = rs.getString("password");

    if (role == 1) return new Customer(...);
    if (role == 2) return new Staff(...);
    if (role == 3) return new StaffManager(...);

    throw new SQLException("Unknown user role: " + role);
}
```

Material – createMaterial

Denne metode viser, hvordan polymorfi og `instanceof` anvendes til at håndtere forskellige materialetyper og deres specialfelter. Samtidig indsættes materialets længder i en separat tabel med fokus på effektivitet.

```
public void createMaterial(Material material) throws DatabaseException {
    try (Connection connection = connectionPool.getConnection()) {
        PreparedStatement ps = connection.prepareStatement(SQL);
```

```
        material.prepareStatement(ps);
        ps.executeUpdate();

        for (int length : material.getPreCutLengths()) {
            PreparedStatement psLength = connection.prepareStatement(SQL_LENGTH);
            psLength.setString(1, material.getName());
            psLength.setInt(2, length);
            psLength.executeUpdate();
        }
    } catch (SQLException e) {
        throw new DatabaseException("Could not create material", e);
    }
}
```

Order – createOrder

Metoden opretter en komplet ordre inkl. status og ordrelinjer som én samlet transaktion. `setAutoCommit(false)` anvendes for at sikre dataintegritet og mulighed for rollback ved fejl.

```
public void createOrder(Order order) throws DatabaseException {
    try (Connection connection = connectionPool.getConnection()) {
        connection.setAutoCommit(false);

        PreparedStatement ps = connection.prepareStatement(SQL_INSERT_ORDER,
Statement.RETURN_GENERATED_KEYS);
        ps.setDate(1, order.getOrderDate());
        ps.setString(2, order.getStatus());
        ps.executeUpdate();

        ResultSet rs = ps.getGeneratedKeys();
        if (rs.next()) {
            int orderId = rs.getInt(1);
            order.setId(orderId);
        }

        for (OrderItem item : order.getOrderItems()) {
            PreparedStatement itemPs = connection.prepareStatement(SQL_INSERT_ITEM);
            itemPs.setInt(1, order.getId());
```

```
            itemPs.setString(2, item.getName());
            itemPs.setFloat(3, item.getSalesPrice());
            itemPs.setInt(4, item.getQuantity());
            itemPs.executeUpdate();
        }

        connection.commit();
    } catch (SQLException e) {
        connection.rollback();
        throw new DatabaseException("Could not create order", e);
    }
}
```

SendGrid- Email

For at sikre en ensartet og professionel e-mail-kommunikation, genererer systemet automatisk formelle hilsner og afslutninger i Java, som derefter indsættes i HTML-formularen via Thymeleaf. På den måde får medarbejderen **forhåndsvist hele beskeden**, som den vil blive sendt – inkl. navn, kontaktoplysninger og hilsen.

Dette gøres f.eks. i controlleren, hvor medarbejderens og kundens oplysninger lægges i modellen:

```
model.put("customerName", user.getFirstName() + " " + user.getLastName());
model.put("staffName", staff.getFirstName());
model.put("staffEmail", staff.getEmail());
model.put("staffPhone", staff.getPhone());
```

I HTML vises beskeden i et <textarea>, hvor tekstens begyndelse og afslutning sættes op direkte via Thymeleaf:

```
<textarea id="message" name="message" rows="14" required
  th:text="'Hej ' + ${customerName} + ',&#10;&#10;&#10;Med venlig hilsen&#10;' +
${staffName} + '&#10;Email: ' + ${staffEmail} + '&#10;Telefon: ' +
${staffPhone}"></textarea>
```

Denne løsning sikrer:

- At medarbejderen ikke behøver tænke på formalia
- At kommunikationen fremstår ensartet og fejlfri
- At modtageren får en tydelig og korrekt e-mail – uanset hvem der sender den

Status på implementering

Projektet er gennemført med et højt ambitionsniveau og stor fokus på kvalitet, testbarhed og brugervenlighed. Samtlige kernefunktioner er implementeret og testet, herunder hele konfigurationsforløbet, kunde- og medarbejderdashboard, login med rollebaseret adgang samt automatisk styklisteberegning og prisudregning.

Der er dog enkelte funktioner og detaljer, som enten ikke er implementeret fuldt ud, eller som er blevet prioriteret fra på grund af tid og kompleksitet:

Funktioner, som ikke er implementeret

- **Skur (shed):** Funktionaliteten til at tilføje et skur til carporten er under udvikling. I det nuværende system kan brugeren vælge en passende skurstørrelse baseret på carportens dimensioner, så skuret ikke overstiger carportens dimensioner. Det er dog endnu ikke muligt for en medarbejder at tilføje eller redigere skurinformation i dashboardet. Desuden bliver skurdata hverken gemt i databasen eller medtaget i styklisteberegningen på nuværende tidspunkt. Funktionaliteten er forberedt i Carport-klassens struktur og kan integreres i kommende iterationer.
- Forsendelse: Mulighed for at vælge eller beregne forsendelse er ikke en del af den nuværende version. Dette vil kræve ekstra domænelogik og udvidelse af databasen.
- **Småmaterialer (skruer, søm, beslag):** Styklisteberegneren medtager i øjeblikket kun de primære konstruktionselementer. Småmaterialer kan nemt og hurtigt tilføjes i en senere iteration.
- **Taghældning på fladt tag:** Det er almindeligt at give selv et "fladt" tag en svag hældning for afvanding. Denne funktion er ikke implementeret, men nævnt i overvejelserne.
- **Skrå hældning:** Systemet understøtter i øjeblikket kun standard flade tage. Muligheden for hældning til siden er ikke tilføjet og vil kræve både visuelle og strukturelle ændringer.
- **Præcis beregning af stolpehøjde:** Stolpernes længde beregnes i dag uden at medregne behovet for nedgravning i jorden eller tilpasning til hældning. En fremtidig forbedring kunne inkludere justering af længde baseret på jordtype og lokal bygningsreglement, samt tilpasning ved tag med hældning.

Overordnet vurdering

De mest centrale funktionaliteter i systemet er fuldt implementeret og grundigt testet. Systemet er designet med fokus på robusthed, stabilitet og brugervenlighed – særligt med henblik på at fungere som et pålideligt arbejdsredskab for Fog's medarbejdere. Der er lagt vægt på at opbygge en arkitektur og kodebase, som er let at vedligeholde og nem at videreudvikle, så systemet også i fremtiden kan tilpasses nye behov uden store omstruktureringer.

Der er lagt vægt på at demonstrere bredden og potentialet i løsningen ved at implementere en stor del af både kundevendte og interne funktioner. Målet har været at vise, hvordan systemet allerede nu fungerer som et stærkt og praktisk værktøj — og hvordan det let kan udbygges. Dette fokus på helhed og sammenhæng har betydet, at enkelte ikke-essentielle elementer som skuret og hældning på taget endnu ikke er fuldt implementeret, men fundamentet er lagt i både domænemodellen og kodebasen.

Kvalitetssikring (test)

Unittest

Systemets styklisteberegner (BillOfMaterial) er en af de mest kritiske komponenter. Den omsætter brugerens input og materialevalg til en konkret opskrift på, hvordan carporten skal bygges – både i form af antal, længder og typer af materialer. Enhver fejl i denne beregning kan få direkte konsekvenser for både økonomi, logistik og kundens oplevelse. Derfor er netop denne komponent testet meget grundigt.

TestFactory: Systematisk testopbygning

For at kunne skrive præcise, målrettede og læsbare unit tests er der udviklet en række **factory-klasser**, herunder TestCarportFactory, TestPlankFactory og TestRoofCoverFactory. Disse klasser har til formål at oprette testobjekter med definerede egenskaber på en struktureret og DRY måde (Don't Repeat Yourself).

Formål med test factories:

- **Reducere redundans**: Man slipper for at gentage oprettelse af fx materialer og carporte med samme konfiguration i hver test.
- **Forbedre læsbarhed**: Testene bliver mere fokuserede og nemmere at forstå, fordi setup-delen er kapslet væk.
- **Fremme isolerede tests**: Ved at generere deterministiske testobjekter undgår man uforudsigelig adfærd og testafhængigheder.
- **Understøtte black-box testtilgang**: Factory-klasserne skjuler implementeringsdetaljer og gør det lettere at skrive tests, som alene fokuserer på input/output.

Brugen af test factories følger principperne for **test design patterns**, navnlig **Object Mother** og **Test Data Builders**, hvor testdata konstrueres med kendte og relevante egenskaber.

BillOfMaterialTest: Grundig test af beregningslogik

Styklisteberegningen udgør et af de mest kritiske elementer i systemet: Den er direkte ansvarlig for at udregne mængden og typen af materialer, der skal bruges til hver carport, og fungerer som grundlag for både tilbud og indkøb. Enhver fejl kan føre til økonomisk tab, spild af materialer eller fejl i levering. Derfor er der lagt stor vægt på at validere netop denne komponent gennem **grundige og målrettede unit tests**.

For at sikre høj testbarhed og lav vedligeholdelsesbyrde er der udviklet **dedikerede test factories** til oprettelse af testdata – fx carporte og materialer med kendte og relevante egenskaber.

Som en del af teststrukturen er der anvendt etablerede principper fra softwaretest, herunder designmønstrene *Object Mother* og *Test Data Builder*, til systematisk opbygning af testdata. Disse mønstre er implementeret i en række dedikerede factory-klasser, som f.eks. TestCarportFactory, TestPlankFactory og TestRoofCoverFactory.

Formålet med denne tilgang har været at reducere kompleksiteten i testopbygningen og sikre, at alle tests bygger på et ensartet og gennemskueligt fundament. I stedet for at gentage konstruktion af identiske testobjekter i hver testmetode, centraliseres oprettelsen i disse factories. Det giver ikke alene mere læsbare tests, men gør det også væsentligt nemmere at vedligeholde og udvide testsuiten over tid.

- **Object Mother**-mønstret anvendes til at oprette "gyldige standardobjekter", der repræsenterer en typisk carport eller et typisk materiale, som kan genbruges på tværs af tests.
- **Test Data Builder**-mønstret gør det muligt at specificere varianter af disse objekter gennem fleksible konfigurationsmetoder. Det anvendes især ved edge case-tests, hvor man ønsker at ændre enkelte værdier (f.eks. spærafstand eller materialeegenskaber), men stadig genbruge resten af strukturen.

Disse mønstre er særligt værdifulde i systemer med mange afhængigheder og domæneobjekter, da de bidrager til:

- **Høj testbarhed**: Det er nemt at konstruere præcise testscenarier med målrettede værdier.
- **Lav vedligeholdelsesbyrde**: Ændringer i objektstruktur eller standardværdier kræver kun opdatering ét sted.
- **Bedre overblik**: Tests indeholder kun den nødvendige logik og undgår teknisk støj fra lange objektoprettelser.

Samlet set har brugen af disse test design patterns været afgørende for at kunne teste beregningslogikken i BillOfMaterial effektivt og konsistent – og har bidraget væsentligt til at løfte både kvalitet og struktur i testsuiten.

Hovedområder i testdækningen:

- **Antal af stolper, spær, remme, tagplader** baseret på carportens dimensioner
- **Beregning af nødvendige længder** ud fra overlaps og afstande
- **Best-fit valg af præskårne materialelængder**
- **Prisberegning** ud fra de valgte materialer og mængder
- **Opbygning af korrekt BillOfMaterialsItem-liste** med beskrivelse, enhed og pris

Testmetoder og -principper:

- **White-box testing**: Flere tests er udformet med kendskab til interne algoritmer (fx calcPostCountWidth()) og tjekker, at specifikke beregninger matcher forventede værdier.
- **Black-box testing**: Samtidig testes højere niveauer (fx hele calculateMaterials()), hvor man kun evaluerer output ift. input, uanset intern implementering.
- **Edge cases**: Eksempler som minimale carporte, carporte med præcis nok længde til ét ekstra spær, eller uventet afrunding er inkluderet.
- **Stress test**: Der er blevet test på meget små og meget store carporte for at sikre systemets robusthed ved ekstreme tilfælde.
- **Assertions på totalpris**: Prisberegninger er testet ved at sammensætte kendte materialer og verificere den samlede værdi.

Kvalitetsmål

- Alle tests kører automatisk og dækker samtlige beregningsfunktioner.
- Der er balance mellem granuleret (test af enkelte metoder) og helhed (test af stykliste som samlet struktur).
- Testene udgør fundamentet for at kunne stole på systemets tilbudsberegning og materialespecifikation.

Fremhævet test

Styklisteberegneren er en af systemets mest gennemtestede komponenter. Der er udviklet mere end 100 individuelle testmetoder, som tilsammen dækker både typiske anvendelser, grænsetilfælde og særligt krævende scenarier.

Styklisteberegnedækning

Fil	Metodedækning	Linjedækning	Grendækning
BillOfMaterial	100%	100%	100%

Denne fulde dækning sikrer, at alle grene, metoder og linjer i BillOfMaterial-klassen er afprøvet under kontrollerede forhold.

Testene omfatter:

- **Almindelige tests**: Verificerer korrekt beregning og styklisteudtræk ved almindelige carportkonfigurationer.
- **Grænseværditests** (edge cases): Dækker minimale og maksimale målangivelser, præcise afstande mellem materialer samt overgangssituationer mellem to forskellige beregningslogikker.
- **Stresstests**: Tester systemets opførsel med store carporte, mange materialer og omfattende kombinationer, med henblik på ydeevne og stabilitet.

Kombinationen af testtyper og den høje dækningsgrad bidrager til et stabilt og gennemprøvet system, hvor beregningerne kan anvendes som grundlag for fakturering, produktion og kundetilbud.

Integrationstest

Systemet benytter et lag af mapperklasser til at håndtere databasetilgange, og der er derfor gennemført en række **integrationstests**, som verificerer at samspillet mellem database og domænemodel fungerer korrekt. Testene er kørt mod en separat testdatabase og dækker de vigtigste CRUD-operationer for hver entitet.

Mapper	Metodedækning	Linjedækning
CarportMapper	100%	88%
MaterialMapper	100%	90%
OrderMapper	70%	63%
UserMapper	72%	61%

Testene sikrer blandt andet korrekt oprettelse, opdatering og læsning af data fra databasen. Derudover er der fokus på at håndtere fejltilfælde, fx manglende poster, ugyldige fremmednøgler og uventet opførsel ved tomme resultatsæt.

Integrationstest

Systemet benytter et lag af mapperklasser til at håndtere databasetilgange, og der er derfor gennemført en række integrationstests, som verificerer at samspillet mellem database og domænemodel fungerer korrekt. Testene er kørt mod en separat testdatabase og dækker de vigtigste CRUD-operationer for hver entitet.

Mapper	Metodedækning	Linjedækning
CarportMapper	100%	88%
MaterialMapper	100%	90%
OrderMapper	70%	63%
UserMapper	72%	61%

Testene sikrer blandt andet korrekt oprettelse, opdatering og læsning af data fra databasen. Derudover er der fokus på at håndtere fejltilfælde, fx manglende poster, ugyldige fremmednøgler og uventet opførsel ved tomme resultatsæt.

Testfilosofi

Testarbejdet er blevet prioriteret gennem hele udviklingsforløbet og har været et centralt værktøj til at skabe sikkerhed og stabilitet. I stedet for at gemme test til slutningen af forløbet er de udviklet løbende i takt med koden – med fokus på:

- Isolering af logik i testbare enheder
- Validering af både korrekte og grænseoverskridende scenarier
- Genbrug og strukturering via factories
- Hurtig feedback under udvikling og ved refaktorering

Målet har ikke kun været "grønne tests", men også at opnå **troværdighed i systemets kernefunktionalitet** – især omkring stykliste, pris og materialelogik.

User Acceptance tests

Da dette projekt er udført som en skoleopgave, har det ikke været muligt at gennemføre egentlige user acceptance tests i samarbejde med Fog. I stedet er der udarbejdet UAT-scenarier for de vigtigste user stories, som illustrerer, hvordan systemet kan valideres i praksis.

Udvalgte User stories og acceptkriterier

User Story ID	Beskrivelse	Acceptkriterier	Status
US-2.1	Som kunde vil jeg kunne angive ønskede mål på min carport.	Brugeren kan angive bredde, længde og højde. Validering sker løbende.	Opfyldt
US-2.2	Som kunde vil jeg kunne vælge materialer.	Materialer vises efter kategori. Valg gemmes til ordren.	Opfyldt
US-3.2	Som kunde vil jeg modtage en	Systemet sender automatisk e-mail med	Opfyldt

	ordrebekræftelse per mail.	ordredetaljer efter bestilling.	
US-14.1	Som sælger vil jeg kunne se mine ordrer og filtrere dem.	Dashboard viser ordrer for logget ind bruger og kan filtreres.	Opfyldt
US-18.2	Som intern bruger skal jeg kunne logge ind og få adgang til mit dashboard.	Loginform fungerer og giver adgang baseret på rolle.	Opfyldt
US-31.3	Som kunde vil jeg kunne se en visuel tegning af carporten.	SVG-genereret tegning baseret på brugerens input vises i browseren.	Opfyldt

Proces

Arbejdsproces og værktøjer

Projektet har været struktureret omkring agile arbejdsprincipper med brug af GitHub Issues og GitHub Projects til planlægning og opgavestyring. Indledningsvist blev projektet opdelt i fire sprints, hvor hvert sprint bestod af en række konkrete issues, der dækkede både udvikling, design og test. Det var planlagt, at gruppen skulle afholde daglige standups for at sikre fremdrift og koordinering.

Til at styre og dokumentere processen blev følgende værktøjer brugt:

- **GitHub Issues:** til at oprette, prioritere og opdele arbejdet i overskuelige opgaver.
- **GitHub Projects (kanban-bord):** til at gruppere issues i sprints og visualisere fremdrift.
- **Markdown og skabeloner:** til at sikre ensartede og tydelige beskrivelser med acceptkriterier.

Efter gruppen blev opløst, fortsatte projektet som en enmandsopgave. Her blev sprintstrukturen opgivet, og fokus flyttede i stedet til en mere fleksibel og kontinuerlig proces, hvor opgaver blev løst én ad gangen ud fra en veldefineret backlog. Da hele backloggen var udarbejdet af undertegnede tidligt i forløbet, var det muligt at bevare overblik og momentum uden at følge en formel sprintstruktur.

Refleksion

Overgangen fra gruppesamarbejde til soloarbejde ændrede den daglige rytme, men havde ikke negativ indflydelse på fremdriften. Takket være den grundige foranalyse og det veldefinerede opgavegrundlag, var det muligt at arbejde effektivt og målrettet som enkeltperson.

De agile principper blev fortsat fulgt i ånden – med iterativ udvikling, løbende prioritering og hurtig feedback via tests og refaktorering. Det har været en klar fordel at arbejde ud fra et GitHub-baseret workflow, da det sikrede både overblik og dokumentation gennem hele forløbet.

Arbejdsprocessen faktuelt

Projektet er i hovedtræk udført som en enmandsopgave, hvilket betød, at de traditionelle Scrum-roller som Kanban-mester og Product Owner blev varetaget af én og samme person. Arbejdet har dog fulgt agile principper med backlog, sprintopdeling og løbende planlægning.

Allerede i opstartsfasen blev der lagt stor vægt på analyse og formulering af user stories. Det grundige forarbejde skabte en solid og prioriteret backlog, som blev det naturlige udgangspunkt for det videre udviklingsarbejde. Denne tidlige forretningsforståelse og systemafklaring har haft stor betydning for projektets stabilitet og fremdrift.

Arbejdsprocessen – Reflekteret

Projektets arbejdsproces har været præget af høj motivation og målrettethed, men også af de særlige betingelser, der følger med at være alene om en opgave, som normalt løses i et team. Roller som Kanban-mester, udvikler og produktansvarlig blev samlet i én person, hvilket skabte en effektiv og fokuseret arbejdsform, hvor fremdriften i høj grad blev styret af produktmål frem for procesritualer.

I praksis betød det, at mange opgaver blev løst direkte fra idé til kode – uden behov for at omsætte alt til issues eller arbejde systematisk med estimering og sprintstruktur. User stories og backlog fungerede som overblik og rettesnor, men den daglige arbejdsrytme blev drevet af intuition og indsigt i, hvad der teknisk og funktionelt var nødvendigt at implementere.

Denne fleksible tilgang fungerede godt i solosituationen, hvor der ikke var behov for koordinering. Det gav flow, hurtige beslutninger og mulighed for at arbejde kontinuerligt uden afbrydelser. Ulempen var, at ikke alle opgaver blev dokumenteret i GitHub-issues, men arbejdet blev alligevel gennemført med høj kvalitet og testdækning.

GitHub blev anvendt som versionsstyringsværktøj og til at strukturere arbejdet i branches, som afspejlede funktionelle opgaver. Det gjorde det muligt at arbejde fra flere enheder og at holde koden organiseret, selvom der ikke var behov for team-koordination, code reviews eller pull requests i traditionel forstand.

Vejledning og teamstruktur

I starten af projektet oplevede gruppen udfordringer med at finde en fælles arbejdsmæssig rytme og struktur. På den baggrund valgte vejlederen at opdele gruppen, hvilket betød, at jeg fortsatte projektet som en soloopgave. Selvom det i praksis gav stor fleksibilitet og mulighed for at arbejde effektivt uden at skulle koordinere med andre, betød det også, at faglig sparring og idéudveksling udeblev.

Jeg havde et ønske om at indgå i et større team, da det bedre afspejler en realistisk udviklingssituation, hvor opgaver fordeles på tværs af forskellige faglige kompetencer. I sådan et samarbejde bidrager hver deltager med deres styrker, og det samlede produkt får ofte højere kvalitet, fordi flere perspektiver bringes i spil undervejs i udviklingen. Derfor ville jeg gerne have haft muligheden for at samarbejde tæt med en eller flere ligesindede, der deler ambitionen om at bygge noget solidt og professionelt.

Set i bakspejlet gav det god mening, at projektgruppen blev opdelt tidligt i forløbet, da det muliggjorde et mere fokuseret og målrettet arbejde ud fra de enkelte deltageres engagement og ambitioner. For mit vedkommende betød det, at jeg kunne arbejde koncentreret og effektivt med at realisere den samlede vision for projektet.

Bilag

Tabeller

Tabel 1

Projekt: Carport Udfyldt af: Carl Emil Styrbjørn Gullacksen Dato: 25/4-2025

Interessent	Interessenten kan opleve følgende FORDELE ved projektet	Interessenten kan opleve følgende ULEMPER ved projektet	Samlet vurdering af interessentens bidrag/position	Håndtering af interessenten
Gidsler				
Medarbejdere	Mindre manuelt arbejde, mere tid til kundekontakt	Læringskurve ifm. nyt system	Meget høj	Løbende sparring og involvering i behovsafdækning
Eksterne interessenter				
Lager	Bedre forudsigelighed omkring materialeforbrug og leverancer	Krav om opdatering af lagerdata for at sikre valide bestillinger	Medium	På sigt mulig integration – Holdes opdateret
Kunde	Større frihed i valg og nemmere bestilling	Mindre personlig kontakt i første led	Meget høj	Fokus på brugervenlighed og valgfrihed
Leverandør	Mere struktureret bestilling og mulighed for integration samt øget salg	Behov for opdaterede produktdata	Lav til medium	Informeres ved ændringer, evt. fremtidig integration
Fragt	Flere ordrer kan medføre øget aktivitet i fragtleddet	Ingen ændring i arbejdsrutiner	Lav	Indirekte påvirkning
Håndværker	Mere præcise guides og tegninger	Forventninger til præcision stiger	Lav	Indirekte – kun påvirket via ordrespecifikationer
Ressourcepersoner				
Udviklere	Udviklingen tager udgangspunkt i en realistisk forretningssituation med	Enkeltmandsarbejde med stor opgavemængde	Kritisk	Selvstyring og løbende prioritering

	reelle krav og behov			
Markedsføring	Kan bidrage med at tilpasse produktet bedre til Fogs kunder	Individuelle marketingtilpasninger kræver flere ressourcer sammenlignet med en standardløsning	Medium	Holdes orienteret om funktionalitet, ikke aktivt inddraget
Jura	Bedre dokumentation og datasporing	Skal evt. gennemgå nye vilkår og databehandling	Lav	Inddrages hvis nødvendigt for GDPR eller jura
Salgsleder/repræsentant	Overblik, materialestyring, medarbejderstyring og kundeopfølgning i ét samlet system	Skal tage ansvar for onboarding og tilpasning til daglig drift	Kritisk	Tæt samarbejde og løbende afstemning af behov og ønsker
Grå eminencer				
Regnskab	Bedre overblik over ordrer og mulighed for struktureret dokumentation	Krav om ændringer i interne processer og rapportering	Lav til medium	Holdes orienteret ved relevante ændringer
Direktion	Effektivisering af arbejdsgange og forbedret kundeoplevelse kan styrke virksomhedens profil	Implementering af nyt system kræver ressourcer og ændringsparathed	Medium	Informeres om projektets fremdrift og resultater
Bestyrelse	Strategisk styrkelse af virksomhedens konkurrenceevne	Langsigtede investeringer skal give målbar værdi	Lav	Indirekte påvirkning – Informeres gennem ledelsen

Tabel 2

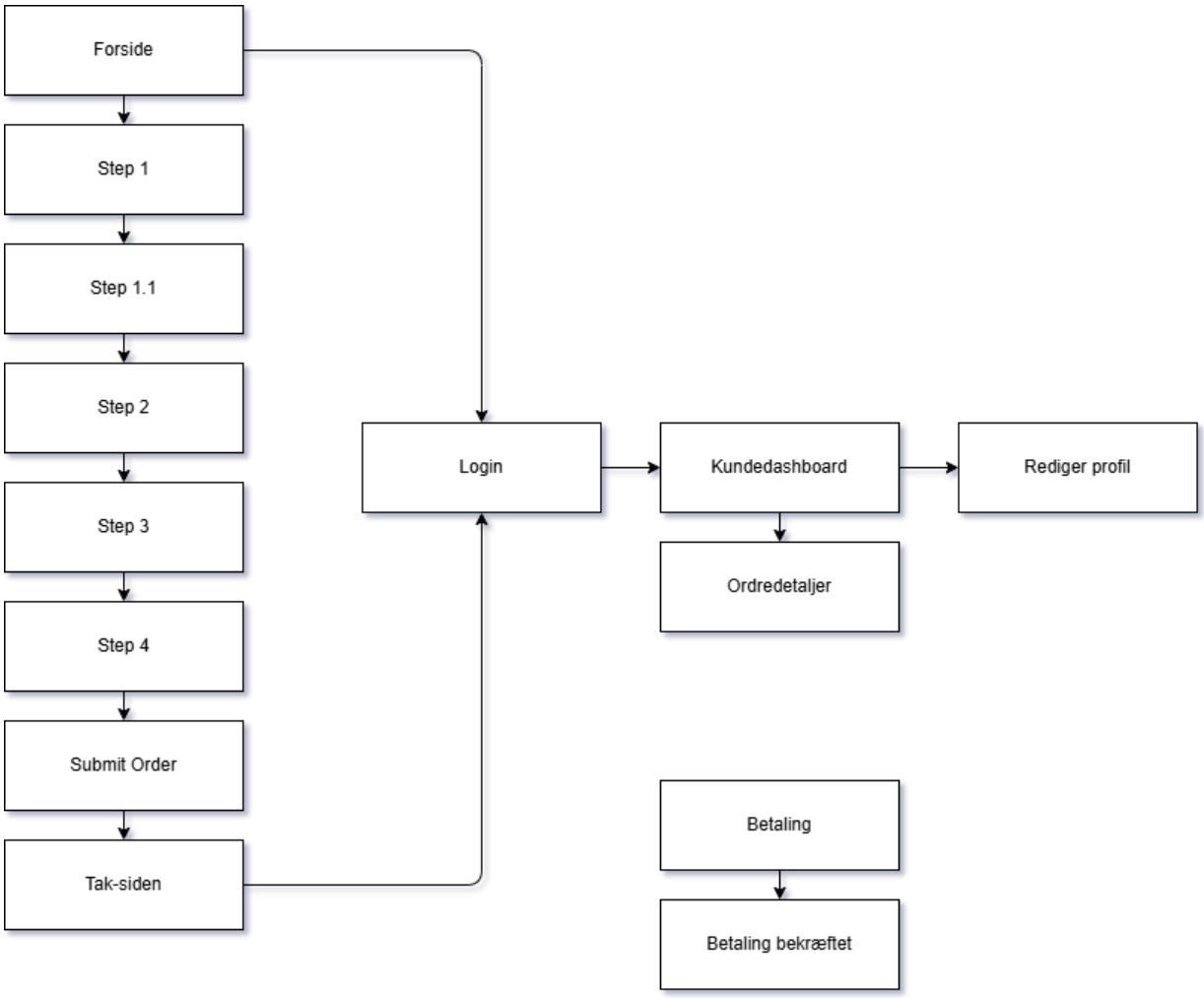
ID	risiko	sandsynlighed	konsekvens	risiko niveau	tiltag/håndtering
R1	fejl i beregningsalgoritme	4	5	20	grundig unit test
R2	dårlig brugeroplevelse ui/ux	2	2	4	user test
R3	utilstrækkelig input validering	1	1	1	sælger kontakt
R4	datatab eller mistet kundeforespørgsel	1	4	4	backup
R5	forsinkelser i udviklingen	3	4	12	Evaluer estimater
R6	sikkerhedsproblemer	3	5	15	frameworks
R7	fejl ved integration af database/mail	1	3	3	integrationstest
R8	uklar komunikation med kunde (fog)	3	4	12	afholde regelmæssige statusmøder
R9	browser / enheder understøttes ikke	3	1	3	responsiv design og browser test
R10	ændringer i krav fra kunde undervejs	2	4	8	løbende godkendelser
R11	hosting problemer	2	5	10	pålidelig hosting
R12	gdpr overtrædelser	3	4	12	cookies accept
R13	Fog går konkurs	1	5	5	Modtag delbetalinger
R14	Hosting leverandør går ned	1	5	5	Overvåg site. Manual til opsætning.
R15	Projektet bliver lukket før tid	2	4	8	Indskriv kompensation i kontrakt
R16	Sygdom	2	5	8	Undgå lav busfaktor
R17	Merge konflikter	2	3	6	Brug branches og PR
R18	Manglende test	3	4	12	Mistillid til komplicerede metoder

Tabel 3

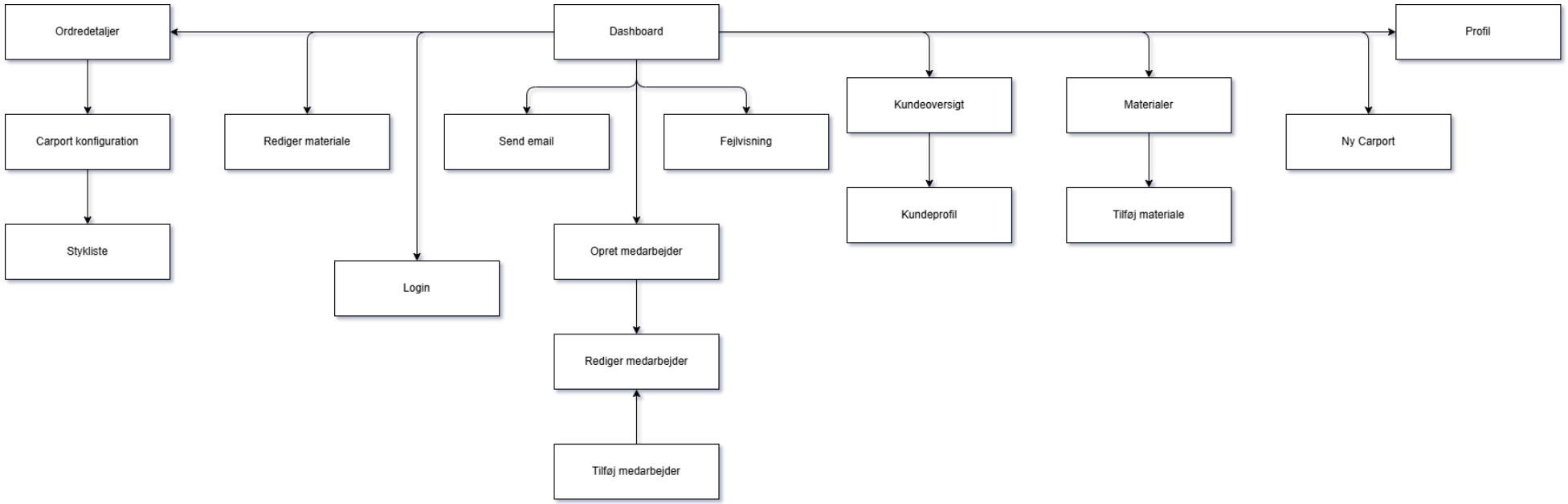
Risiko-matrice	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

Figurer

Figur 1



Figur 2


Ordredetaljer
Dashboard
Profil
Carport konfiguration
Rediger materiale
Send email
Fejlvisning
Kundeoversigt
Materialer
Ny Carport
Stykliste
Login
Opret medarbejder
Kundeprofil
Tilføj materiale
Rediger medarbejder
Tilføj medarbejder

Iconer:

https://www.flaticon.com/authors/cah-nggunung
https://www.flaticon.com/authors/vectorsmarket15
https://www.freepik.com/
https://www.flaticon.com/authors/backwoods
https://www.flaticon.com/authors/sympnoiaicon
https://www.flaticon.com/authors/pixel-perfect
https://www.flaticon.com/authors/zlatko-najdenovski