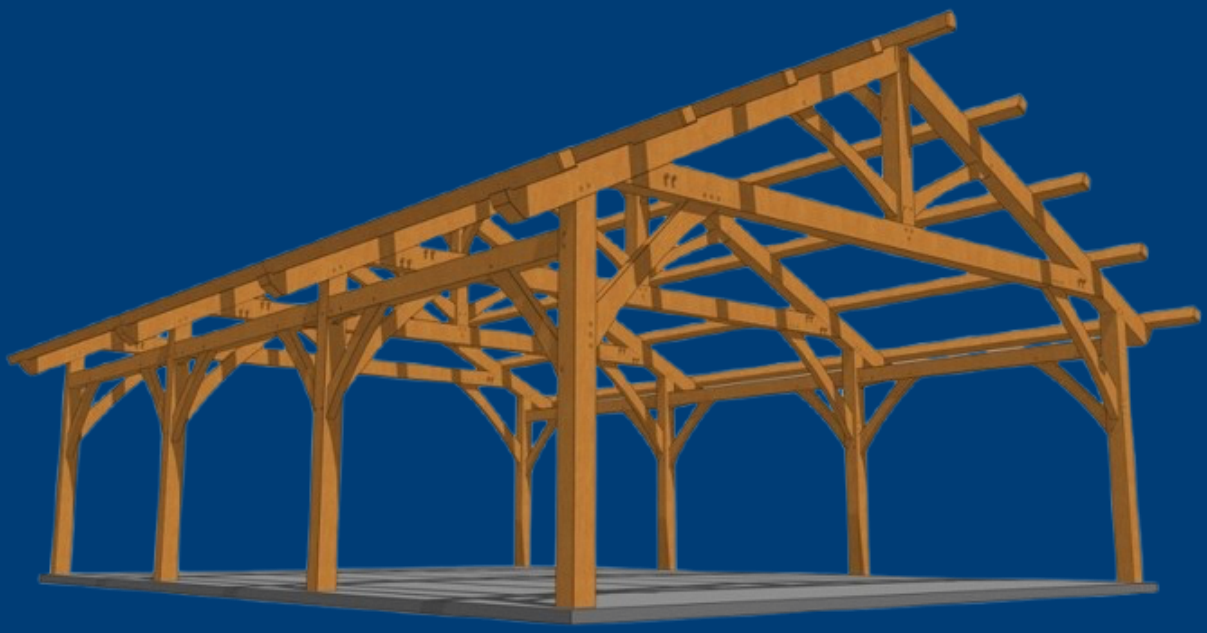
Fog®

CARPORT



Carl Emil Asly Køhn	cph-ck381@stud.ek.dk
Oliver Møller-Jensen	cph-om96@stud.ek.dk
Dino Saldic	cph-ds303@stud.ek.dk
Stephanie Tess van Slyck	cph-sv106@stud.ek.dk
GitHub:	link til GitHub
Deployed Digital Ocean Link:	link til Deployment
Demovideo på Youtube:	Link til demovideo

18-12/2025

1 Indholdsfortegnelse

2 INDLEDNING

Formålet med dette projekt er at udvikle et webbaseret konfigurationssystem til Fog Trælast- og Byggecenter, som skal modernisere og automatisere virksomhedens salgsproces for skræddersyede carporte. Projektet er udarbejdet som et semesterprojekt på 2. semester på datamatikeruddannelsen og tager udgangspunkt i et kundebesøg, hvor Fog har præsenteret deres nuværende arbejdsproces samt de udfordringer, de oplever med deres eksisterende systemer.

Virksomheden håndterer i dag deres carportordrer manuelt på tværs af flere systemer, hvilket har skabt en flaskehals i forretningen. Hver forespørgsel kræver manuel genindtastning af de specifikationer, som kunden har udfyldt i en formular. Derudover anvender Fog et forældet lagersystem, udviklet i 1999, hvilket medfører problemer med at holde lagerbeholdning, priser og varenumre opdateret på tværs af systemerne.

Den ønskede løsning er et samlet system, som nemt kan opdateres ved ændringer i priser og lagerbeholdning. Fog ønsker at automatisere dele af processen uden at gå på kompromis med kundekontakten, som er en vigtig del af virksomhedens varemærke. Systemet skal fungere som et selvbetjeningsværktøj for kunderne, hvor de kan konfigurere deres egen carport, samtidig med at sælgere får mulighed for at gennemgå, administrere samt godkende eller afvise ordrer.

Systemet skal give kunderne mulighed for at indtaste specifikationer såsom bredde, længde, tagtype og eventuelt tilvalg af redskabsrum. Formålet er at udvikle et samlet system, der kan beregne materialeforbrug baseret på de indtastede oplysninger, generere en stykliste og udregne en pris. En tidligere udviklet “nice-to-have”-funktion hos Fog var en 2D-visualisering, som i realtid viste kundens skræddersyede carport. Denne løsning kunne dog ikke integreres med virksomhedens eksisterende systemer, hvorfor der efterspørges ét samlet system, som kan løse problemstillingen.

For Fog’s sælgere skal systemet anvendes til at administrere indkomne forespørgsler. Her skal de kunne overskue og gennemgå de automatisk beregnede priser, justere prisen og/eller tilføje kommentarer samt godkende eller afvise ordrer. Ved godkendelse af en ordre skal systemet automatisk generere og sende en ordrebekræftelse samt stykliste til kunden via e-mail.

Projektet er udviklet i Java 17 med Javalin som webframework og Thymeleaf som template engine. Den indsamlede data gemmes i en PostgreSQL-database, og systemet er deployet i skyen via DigitalOcean. Der anvendes GitHub til versionsstyring, Kanban til projektledelse samt løbende tests til kvalitetssikring. Rapporten dokumenterer udviklingsprocessen fra kravspecifikation til færdig implementering og henvender sig til læsere med tilsvarende fagligt niveau og kendskab til systemudvikling, databaser og webudvikling.

3 BAGGRUND

Fog Trælast- og Byggecenter er en virksomhed, der specialiserer sig i levering af byggematerialer samt skræddersyede carportløsninger til private kunder. Virksomheden oplever i dag udfordringer med deres salgsproces for carporte, hvor en stor del af arbejdet er manuelt og fordelt på flere forskellige systemer. Det system, Fog anvender til styklisteberegning, Quick Byg, er udviklet i 1999 og har med tiden skabt betydelige begrænsninger i forhold til både effektivitet og skalerbarhed.

3.1 Den nuværende proces for håndtering af carportordrer

1. Kunden vælger en skræddersyet carport og indtaster sine specifikationer.
2. Formularen sendes automatisk som e-mail til sælgeren via et separat mailsystem.
3. Sælgeren indtaster manuelt oplysningerne på ny i Quick Byg.
4. Quick Byg beregner en stykliste, som til tider indeholder forældede varenumre og priser.
5. Sælgeren korrigerer manuelt de forældede varenumre og priser.
6. Det endelige tilbud sendes til kunden.

Denne proces medfører en høj risiko for fejl, dobbeltarbejde og generel ineffektivitet, da de samme informationer skal indtastes flere gange på tværs af systemer. Derudover er adgangsstyringen i det nuværende system meget begrænset. Der eksisterer kun én adgangskode, som indehaveren af Værebro Trælast-afdelingen har adgang til. Systemet tillader udelukkende ændringer af priser, mens lagerbeholdning, varenumre og oprettelse af nye materialer ikke kan administreres. Manglen på flere brugere og en sikker gendannelsesmekanisme øger fejlrisikoen og gør systemet sårbart, især hvis adgangen mistes.

En yderligere udfordring er forældet data og behovet for manuel korrektion. Systemet har været i brug i over 20 år, og leverandører har siden ændret både varenumre og materialespecifikationer, hvilket kræver løbende manuelle tilpasninger.

Endelig mangler Quick Byg integration med virksomhedens lagerstyringssystem, Smart Office. Der sker ingen automatisk synkronisering af lagerbeholdning, varenumre eller priser, og al dataudveksling foregår manuelt. Dette gør processen tidskrævende og fejlbehæftet for Fog's medarbejdere. Virksomheden har tidligere investeret i et nyt system, som kunne hente varer direkte fra databasen, men manglende integration med eksisterende IT-systemer betød, at løsningen aldrig blev taget i brug. Denne erfaring understreger behovet for et samlet system, der både kan generere styklister, håndtere lagerdata og automatisere e-mailkommunikation, samtidig med at den personlige kundekontakt bevares.

4 VIRKSOMHEDEN & FORRETNINGSFORSTÅELSE

For at sikre, at løsningen understøtter både de forretningsmæssige og brugermæssige behov, er der i den indledende fase af projektet udarbejdet en interessentanalyse samt en risikoanalyse. Formålet med disse analyser er at skabe overblik over projektet, identificere de centrale aktører og sikre en korrekt prioritering af udviklingsindsatsen.

4.1 Risikoanalyse

Risikoanalysen er en del af projektets planlægningsfase og har til formål at identificere potentielle risici samt minimere deres negative konsekvenser for projektets fremdrift. Risiciene er vurderet ud fra tre parametre: alvor, sandsynlighed og samlet risikoniveau. På baggrund af disse parametre er der beregnet risikopoint, som anvendes til at prioritere indsats og fokus for den enkelte risiko.

Mergekonflikter og fejl i testning blev identificeret som de væsentligste risici, da disse kan medføre forsinkelser og ustabilitet i systemet. Teamet har derfor haft særligt fokus på disse områder gennem hele udviklingsprocessen.

Risk ID	Risiko	Alvor	Sandsynlighed	Risikoniveau	Forbyggelse	Risicopoint
1	Sygdom i udviklingsteamet	Tolereres	Muligt	Medium	Syge bliver hjemme for at undgå smitte	9
2	Merge konfliker	Uønsket	Sandsynligt	Medium	Arbejder i feature branches	12
3	Gruppemedlem dropper ud	Uønsket	Usandsynlig	Høj	Fordeler deres arbejde	4
4	Bugs	Uønsket	Sandsynligt	Høj	Tester løbende kode og debugger	16
5	Server udfordringer	Uacceptabelt	Muligt	Medium	Backup server	9
6	Fejl i testing	Uønsket	Sandsynligt	Høj	Planlæg codestop i god tid	16

4.2 Interessentanalyse

Interessentanalysen identificerer de personer og grupper, der påvirkes af eller har indflydelse på projektet. Følgende interessenter er identificeret:

- **Slutkunder** udgør den primære brugergruppe af systemet. De ønsker en nem og hurtig måde at bestille en skræddersyet carport på. Selvom kunderne har lav direkte indflydelse på systemets udvikling, har de høj påvirkning, da deres tilfredshed er afgørende for systemets succes.
- **Fog's sælgere** har behov for et effektivt værktøj til administration af ordrer. De har høj indflydelse, da de anvender systemet dagligt, og deres feedback er afgørende for systemets funktionalitet. Deres påvirkning er ligeledes høj, da systemet har stor betydning for deres daglige arbejdsproces.
- **Fog's ledelse** har fokus på øget effektivitet og reduktion af fejl i ordrebehandlingen. Ledelsen har høj indflydelse, da de træffer den endelige beslutning om, hvorvidt systemet skal tages i brug. Deres direkte påvirkning er middel, da de ikke anvender systemet i det daglige.

5 TEKNOLOGIER

Frontend er implementeret med **HTML5** og **CSS3** og renderes server-side ved hjælp af **Thymeleaf 3.1.2.**

Backend-delen af projektet er udviklet i **Java 21** og anvender **Javalin 6.1.3** som webframework til routing og håndtering af HTTP-requests. Valget af server-side rendering med Thymeleaf muliggør en klar adskillelse mellem præsentationslag og applikationslogik.

Data persisteres i en PostgreSQL-database via JDBC, hvor **HikariCP 5.1.0** anvendes til connection pooling for at sikre effektiv og stabil databaseadgang. For at reducere mængden af gentagende boilerplate-kode anvendes **Lombok 1.18.38** til generering af konstruktører, getters og setters.

Systemet understøtter afsendelse af e-mails via **Jakarta Mail 2.1.3**, med **Angus Mail 2.0.3** som implementation. Projektet bygges og pakkes som en fat-jar ved hjælp af Maven, hvilket gør applikationen nem at deploye og afvikle som en selvstændig enhed.

6 KRAV

6.1 Funktionelle kundekrav

Slutkunden skal via systemet kunne skræddersy en carport og gennemføre en bestilling med følgende funktionalitet:

6.1.1 Konfiguration

- Vælge bredde og længde på carporten ud fra gyldige mål i dropdown-menuer
- Vælge tagtype (fladt tag eller tag med rejsning)
- Tilvælge redskabsrum med angivelse af størrelse
- Se en visualisering af carporten
- Se en 2D-visualisering af den konfigurerede carport (nice-to-have)
- Se den beregnede pris baseret på valgte specifikationer

6.1.2 Bestillingsproces

- Sende en forespørgsel til sælger med alle valgte specifikationer

- Modtage ordrebekræftelse, når sælger har godkendt ordren
- Gennemføre betaling efter godkendelse fra sælger
- Modtage stykliste efter gennemført betaling

6.2 Funktionelle systemkrav

Systemet skal automatisk kunne håndtere følgende funktioner:

6.2.1 Beregning og validering

- Beregne nødvendige materialer (stolper, spær, brædder, beslag og skruer)
- Generere en stykliste baseret på mål, tagtype og eventuelt redskabsrum
- Beregne en samlet pris inklusive materialer
- Validere de valgte mål for at sikre gyldige konfigurationer

6.2.2 Dokumenthåndtering

- Generere ordrebekræftelse til kunden
- Generere en komplet stykliste til produktionen
- Eksportere styklisten til brug i produktionen
- Tildele hver ordre et unikt ordrenummer
- Gemme ordrer sikkert i databasen

6.2.3 Kommunikation

- Sende e-mail til sælger ved modtagelse af en ny forespørgsel
- Sende e-mail til kunden, når en ordre godkendes eller afvises
- Sende ordrebekræftelse og stykliste til kunden efter gennemført betaling

6.3 Funktionelle sælgerkrav

Sælgeren skal via systemet kunne håndtere kundeforespørgsler og administrere ordrer effektivt.

6.3.1 Ordrehåndtering

- Modtage afventende (pending) forespørgsler fra kunder med alle specifikationer
- Se kundens valg af mål, tagtype og eventuelt redskabsrum
- Acceptere eller afvise kundens specifikationer
- Justere ordreprisen efter behov
- Tilføje kommentarer til ordren
- Godkende eller afvise ordrer

6.3.2 Oversigt

- Se ordrehistorik for alle kunder
- Markere ordrer med status: pending, accepted eller rejected
- Godkende afsendelse af stykliste til produktionen

7 ARBEJDSGANGE DER SKAL IT-UNDERSTØTTES

7.1 As-is

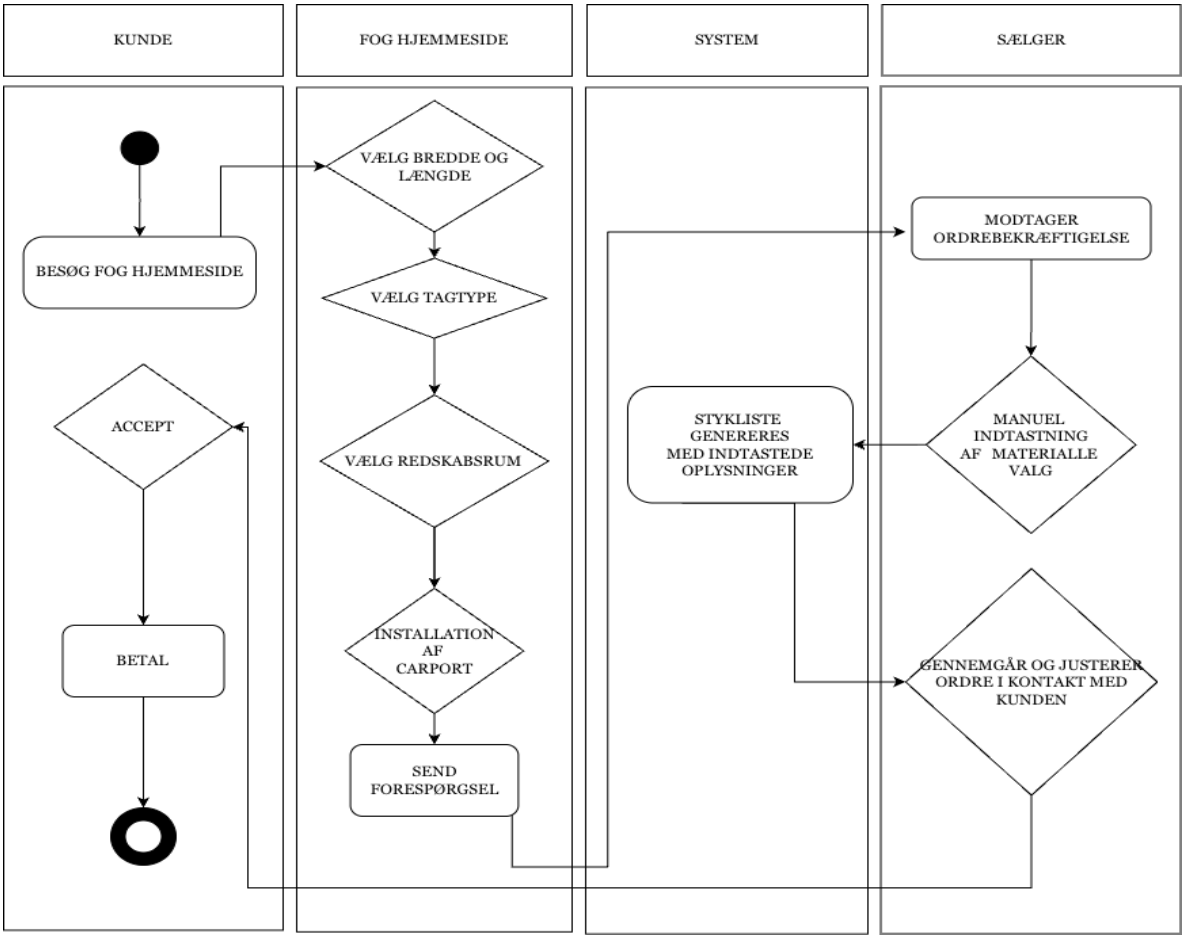
AS-IS-diagrammet viser det nuværende kundeflow fra start til slut. Diagrammet er opbygget med fire svømmebaner, som repræsenterer henholdsvis kunden, Fog's hjemmeside, carportsystemet og sælgeren.

Flowet starter med, at kunden besøger Fog's hjemmeside og indtaster sine specifikationer, herunder bredde, længde, tagtype og eventuelt redskabsrum. Herefter sendes en forespørgsel til sælgeren. Hvis sælgeren accepterer forespørgslen, skal kunden gennemføre betaling, hvorefter systemet genererer en stykliste baseret på de indtastede oplysninger.

Sælgeren validerer efterfølgende de indtastede mål manuelt ved at genindtaste oplysningerne i deres interne system. Til sidst gennemgår sælgeren ordren og foretager eventuelle justeringer i dialog med kunden.

AS-IS-diagrammet indeholder ikke et fuldt betalingsflow eller et afvisningsflow. For at bevare overblik og overskuelighed er diagrammet opbygget uden gentagne loops og med klare endpoints.

7.1.1 Bilag: as-is



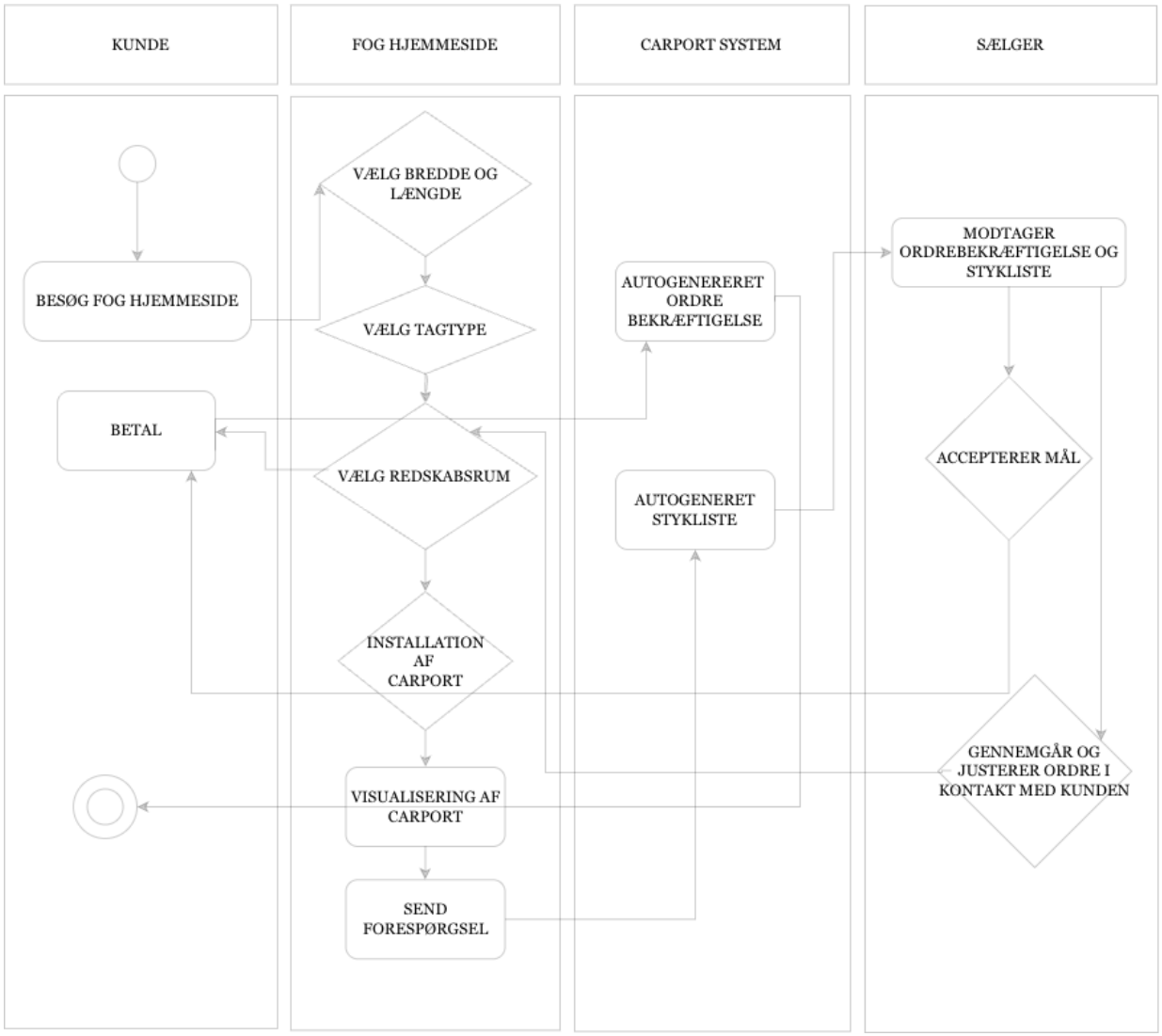
7.2 To-be

TO-BE-diagrammet viser det ønskede fremtidige flow med det forbedrede system. Diagrammet anvender de samme svømmebaner som AS-IS-diagrammet, men med et væsentligt kortere og mere effektivt flow.

Kunden besøger Fog's hjemmeside og indtaster sine specifikationer. Systemet beregner automatisk materialer og genererer en stykliste, som vises sammen med en 2D-visualisering af den konfigurerede carport. Herefter sendes en forespørgsel til sælgeren.

Sælgeren modtager en ordrebekræftelse samt den automatisk genererede stykliste direkte fra systemet. Dette adskiller sig fra AS-IS-flowet, hvor sælgeren tidligere skulle genindtaste oplysninger manuelt. Sælgeren kan herefter acceptere målene, justere ordreprisen og tilføje eventuelle kommentarer, inden ordren færdiggøres.

7.2.1 Bilag: to-be



8 USER STORIES

8.1 US-1: Valg af tagtype

Som kunde ønsker jeg at kunne vælge imellem tag med rejsning og fladt tag på min carport, så jeg kan få en løsning der passer til mine behov.

Acceptkriterier:

- Givet at kunden er på hjemmesiden
- Når kunden trykker på "Bestil skræddersyet carport"
- Så vises der to muligheder med rejsning eller fladt

8.2 US-2: Valg af bredde og længde

Som kunde ønsker jeg at kunne vælge bredde og længde på min carport i en dropdown menu, så jeg kan få en carport der passer til min grund

Acceptkriterier:

- Givet at kunden har valgt tagtype
- Når kunden er på skræddersy carport siden
- Så kunden kan indtaste længde fra dropdown
- Så kunden kan indtaste bredde fra dropdown
- Og systemet viser de valgte mål

8.3 US-3: Redskabsrum

Som kunde ønsker jeg at kunne tilføje et redskabsrum til min carport, så jeg har ekstra opbevaringsplads.

Acceptkriterier:

- Givet at kunden er på carport konfigurationssiden
- Når kunden vælger at tilføje redskabsrum
- Så kan kunden vælge om de vil have redskabsrum eller ej

- Og hvis de vælger redskabsrum, kan de vælge længde og bredde fra hver sin dropdown

8.4 US-4: 2D Visualisering

Som kunde ønsker jeg at se en realtids visualisering i 2D model af carporten med mine specifikationer, så jeg kan være sikker på at resultatet er som forventet.

Acceptkriterier:

- Givet at kunden har indtastet tagtype valg og mål
- Når kunden er på konfigurationssiden
- Så vises en 2D visualisering af carporten
- Og visualiseringen opdateres ved ændringer på mål eller valg af redskabsrum

8.5 US-5: Send forespørgsel

Som kunde ønsker jeg at kunne sende min carport forespørgsel, så jeg kan modtage et tilbud fra FOG.

Acceptkriterier:

- Givet at kunden har indtastet alle specifikationerne
- Når kunden klikker “Send forespørgsel”
- Så indtaster kunden kontaktinformation (navn, e-mail, adresse, postnummer)
- Og forespørgslen gemmer i systemet som en kundeordre
- Og kunden får en bekræftelse på at forespørgslen er modtaget
- Og systemet beregner automatisk en pris til Admin review

8.6 US-6: Modtag tilbud via e-mail

Som kunde ønsker jeg at modtage et tilbud via e-mail, så jeg kan se prisen og detaljerne på carporten.

Acceptkriterier:

- Givet at admin har sendt et tilbud
- Når e-mail er afsendt

- Så modtager kunden en e-mail med carportspecifikationerne (tagtype, længde bredde, redskabsrum)
- Samlet pris
- Eventuelle noter fra admin

8.7 US-7: Se kundeordrer med beregnet pris som admin

Som admin ønsker jeg at kunne se en oversigt over alle kundeordrer med automatisk beregnet pris, så jeg har overblik over indkomne ordrer.

Acceptkriterier:

- Givet at admin er logget ind
- Når dashboardet er indlæst
- Så vises der en oversigt over kundeordrer
    - Kundens fornavn og efternavn
    - Tagtype valg
    - Længde og bredde
    - Redskabsrum (nej eller ja med mål)
    - Ordrestatus (pending, accepted, rejected)
    - Oprettelsesdato (timestamp)

8.8 US-8: Rediger pris for ordre

Som admin ønsker jeg at kunne redigere den beregnede pris for en kundeordre, så jeg kan justere tilbuddet efter behov.

Acceptkriterier:

- Givet at admin er logget ind og har valgt en specifik ordre
- Når admin ser ordredetaljerne med den automatisk beregnede pris
- Så kan admin redigere prisen
- Og systemet gemmer den nye pris

- Og der vises en succes besked

8.9 US-9: Tilføj note til ordre

Som admin ønsker jeg at kunne tilføje noter til en kundeordre, så jeg kan dokumentere særlige ønsker eller detaljer.

Acceptkriterier:

- Givet at admin har valgt en specifik ordre
- Når admin ser ordredetaljerne
- Så kan admin tilføje en note
- Og systemet gemmer noten
- Og noten vises i orderdetaljerne

8.10 US-10: Send tilbud til kunde

Som admin ønsker jeg at kunne sende et tilbud til kunden via e-mail, så kunden kan modtage og vurdere tilbuddet.

Acceptkriterier:

- Givet at admin har indtastet eller justeret pris for en ordre
- Når admin klikker på “Send tilbud”
- Så sendes en e-mail til kunden med:
    - Længde, bredde, tagtype og redskabsrum specifikationer
    - Samlet pris
    - Eventuelle note
    - Og ordrestatus opdateres
    - Og der vises en succes besked til admin

9 DOMÆNEMODEL OG ER-DIAGRAM

9.1 Domænemodel

Den centrale entitet i modellen er Order, som repræsenterer en bestilling på en carport. En ordre indeholder oplysninger om carportens dimensioner, ordrestatus, oprettelsestidspunkt, samlet pris samt den valgte tagtype.

En ordre består af flere materialer, som anvendes til konstruktionen, og kan have tilknyttet flere kommentarer, der bruges til kommunikation og administration. Hver ordre er knyttet til præcis én kunde, mens en kunde kan have flere ordrer.

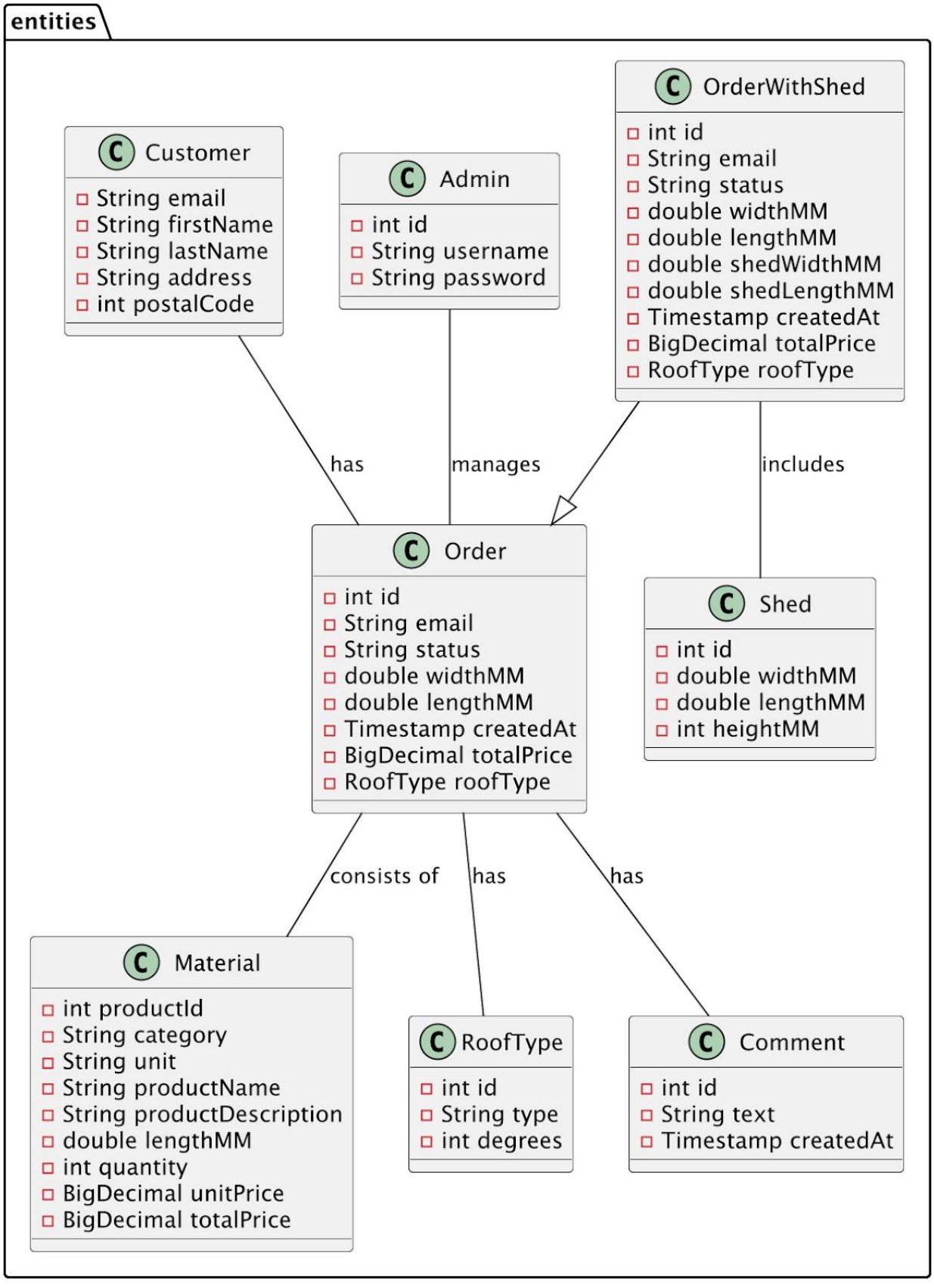
OrderWithShed er modelleret som en specialisering af Order og anvendes udelukkende, når der bestilles en carport med skur. Klassen nedarver alle attributter og relationer fra Order, hvilket betyder, at relationer til materialer, tagtype og kommentarer ikke gentages.

Dette reducerer redundans og bidrager til en mere overskuelig og vedligeholdelsesvenlig model. Kun relationen til Shed, som repræsenterer skurets dimensioner, er specifik for denne type ordre og er derfor modelleret direkte.

Derudover indgår entiteterne Material, RoofType og Comment, som understøtter ordrens sammensætning og administration. Material beskriver de fysiske komponenter, der indgår i konstruktionen, mens RoofType angiver typen og hældningen af taget.

Comment anvendes til beskeder og noter relateret til en ordre. Endelig repræsenterer Admin en systembruger, der kan administrere ordrer, f.eks. ved at se og ændre ordrestatus samt håndtere kommunikationen med kunden. Samlet set afspejler domænemodellen systemets forretningslogik og danner et solidt grundlag for den efterfølgende implementering.

9.2 Bilag: Domænemodel



9.3 ER-DIAGRAM

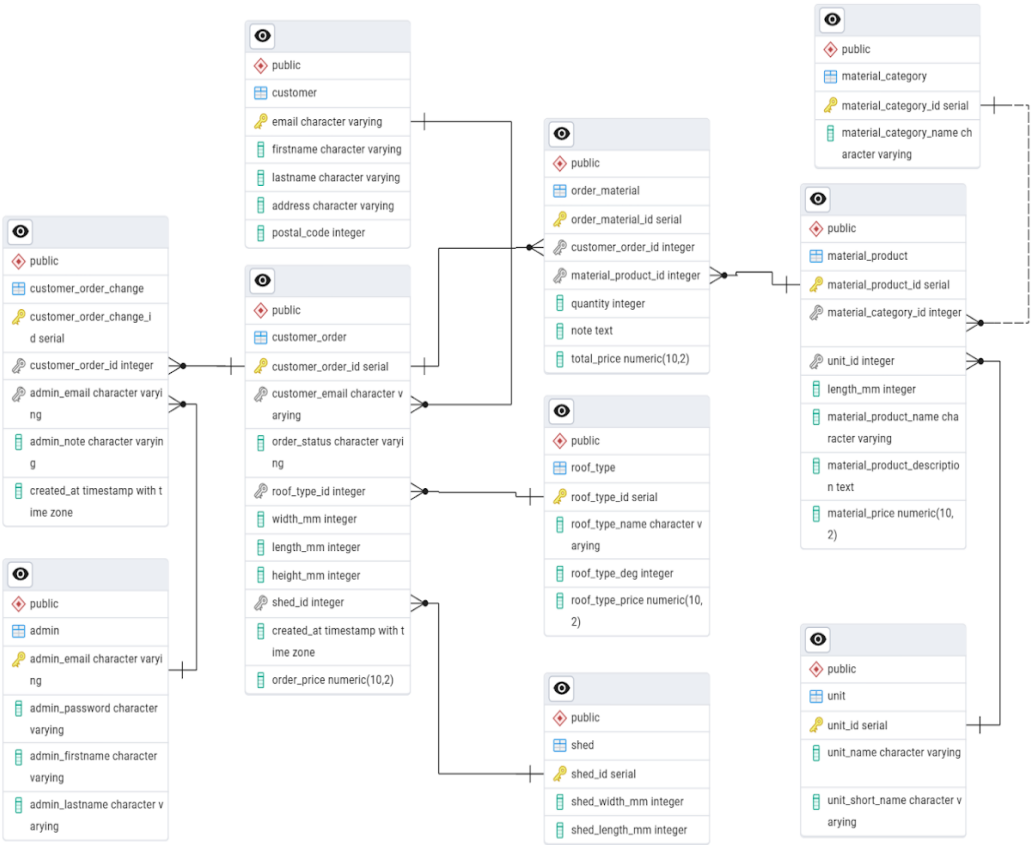
Databasen er designet til at understøtte bestilling og administration af specialbyggede carporte og er opbygget som en relationel model, der overvejende følger 1, 2 og 3. normalform. Formålet har været at undgå redundans, sikre konsistens og gøre prisberegning samt ændringer på ordrer entydige. Modellen er centreret omkring *customer_order*, som repræsenterer en konkret carportordre og er knyttet til en kunde via e-mail, en valgt tagtype samt eventuelt et redskabsskur. En kunde kan have flere ordrer, men en ordre tilhører præcis en kunde.

Materialer er modelleret uafhængigt af ordrer og opdelt i kategorier og enheder for at undgå gentagelse af data. Forholdet mellem ordrer og materialer er en mange til mange relation, som er implementeret via tabellen *order_material*. Denne tabel indeholder både reference til materialet og ordren samt mængde og samlet pris, hvilket gør det muligt at gemme en komplet stykliste for hver ordre. Ændringer foretaget af en admin på en ordre gemmes i *customer_order_change*, som er adskilt fra selve ordren for at bevare historik og sporbarhed frem for at overskrive eksisterende data.
De fleste tabeller anvender automatisk genererede primærnøgler (serial) for enkelhed og performance, mens customer og admin anvender e-mail som primærnøgle. Dette er et bevidst valg, da e-mail i domænet er entydig, stabil og bruges direkte til login og kommunikation. Alle relationer er implementeret med fremmednøgler, og der findes kun en entydig vej mellem centrale entiteter, hvilket reducerer risikoen for inkonsistens i databasen.
Der er dog også designvalg, som med fordel kunne forbedres.

Et eksempel er håndteringen af skur, hvor *shed_id* i dag ligger direkte på *customer_order*. Dette giver en tæt 1 til 1 kobling mellem ordre og skur og medfører en nullable fremmednøgle. En mere fleksibel løsning ville være at anvende en koblings tabel mellem ordre og skur, hvilket ville gøre det nemmere senere at understøtte flere skure pr. ordre eller tilføje ekstra information på relationen uden at ændre ordre strukturen. Ligeledes er *order_status* gemt som fritekst, hvilket er simpelt, men kunne forbedres ved at bruge en separat status tabel for at forhindre ugyldige værdier.

Felter som *order_price* og *total_price* er bevidst gemt i databasen, selvom de kan beregnes, for at sikre at historiske ordrer ikke ændrer pris ved efterfølgende prisjusteringer på materialer, men dette kræver, at beregning altid sker konsistent i applikationslaget.
Samlet set afspejler databasens struktur carportdomænet klart og understøtter både beregning, ordreflow og administration, samtidig med at der er taget bevidste kompromiser mellem fleksibilitet, dataintegritet og implementeringstid.

9.4 Bilag: ER-Diagram



10 NAVIGATIONSDIAGRAM

Navigations diagrammet har til formål at give et samlet og overskueligt overblik over systemets opbygning samt de mulige veje gennem applikationen, samt viser både et overordnet flow og de vigtigste sammenhænge mellem siderne, og er derfor et centralt redskab i forståelsen af brugeroplevelsen.

Overordnet kan systemets navigation opdeles i to hovedområder: et kundeflow og et adminflow. Kundeflowet starter på forsiden, som er offentligt tilgængelig uden login. Herfra kan brugeren vælge mellem en specialbygget carport eller en standard carport. Vælges standard carport, viderestilles brugeren til Fog's eksterne hjemmeside, mens valg af specialbygget carport fører videre til valg af carport typer, enten med fladt tag eller med rejsning.

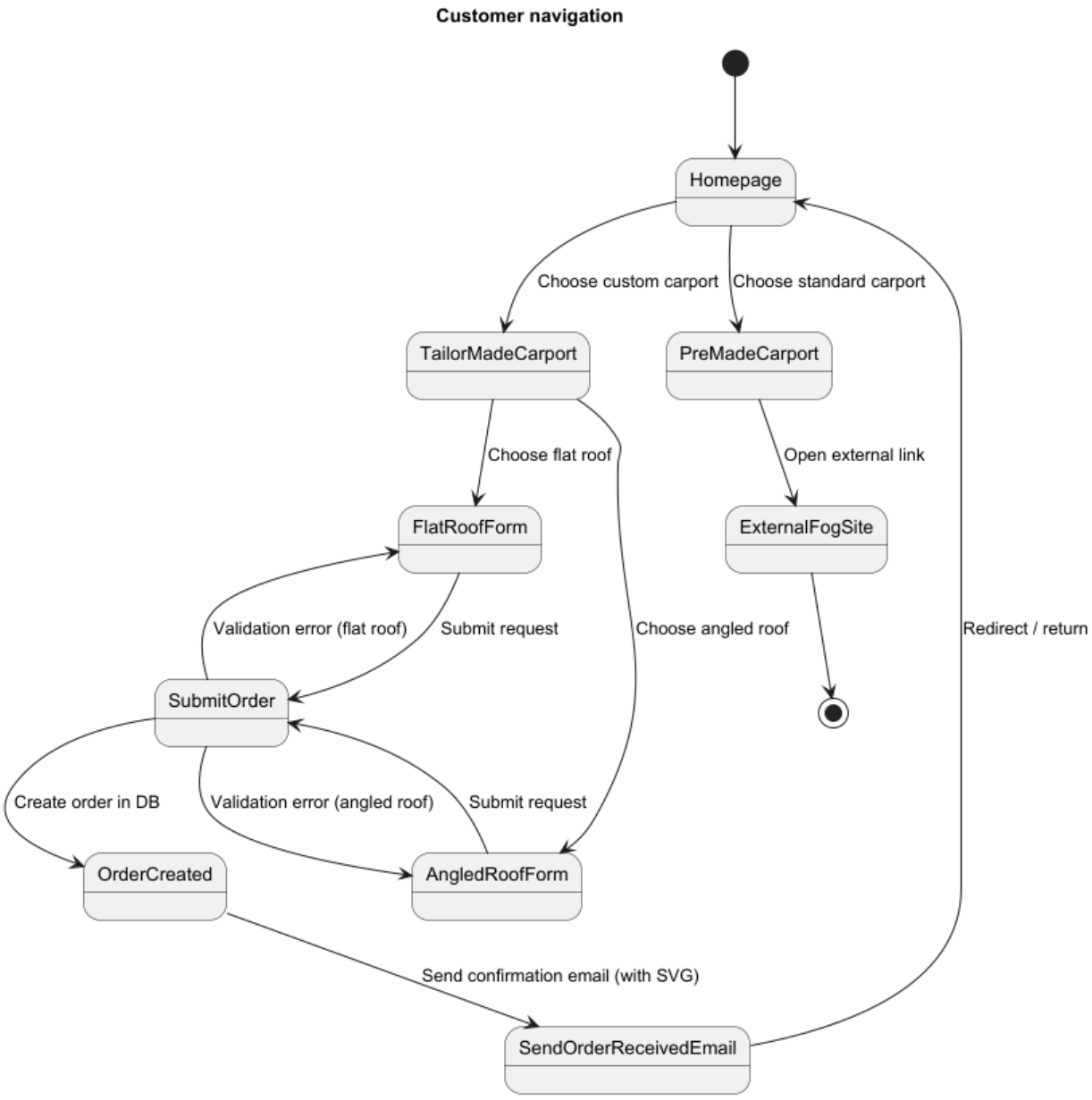
Afhængigt af dette valg navigerer brugeren til en konfigurationsside, hvor alle nødvendige oplysninger indtastes, herunder carportens mål, eventuelt redskabsskur samt kontaktoplysninger. Når alle felter er korrekt udfyldt, kan brugeren indsende en ordre forespørgsel. Hvis der mangler eller er ugyldige oplysninger, forbliver brugeren på samme side og får mulighed for at rette input. Ved korrekt indsendelse oprettes ordren, der sendes en bekræftelsesmail til brugeren, hvorefter brugeren omdirigeres tilbage til forsiden.

Kunde Siderne er forbundet gennem en fælles navigation, som sikrer en ensartet brugeroplevelse på tværs af siderne. Denne navigation gør det muligt at vende tilbage til forsiden eller skifte carport valg uden at skulle gennemføre hele bestillingsflowet igen og bidrager dermed til et mere brugervenligt og overskueligt system.

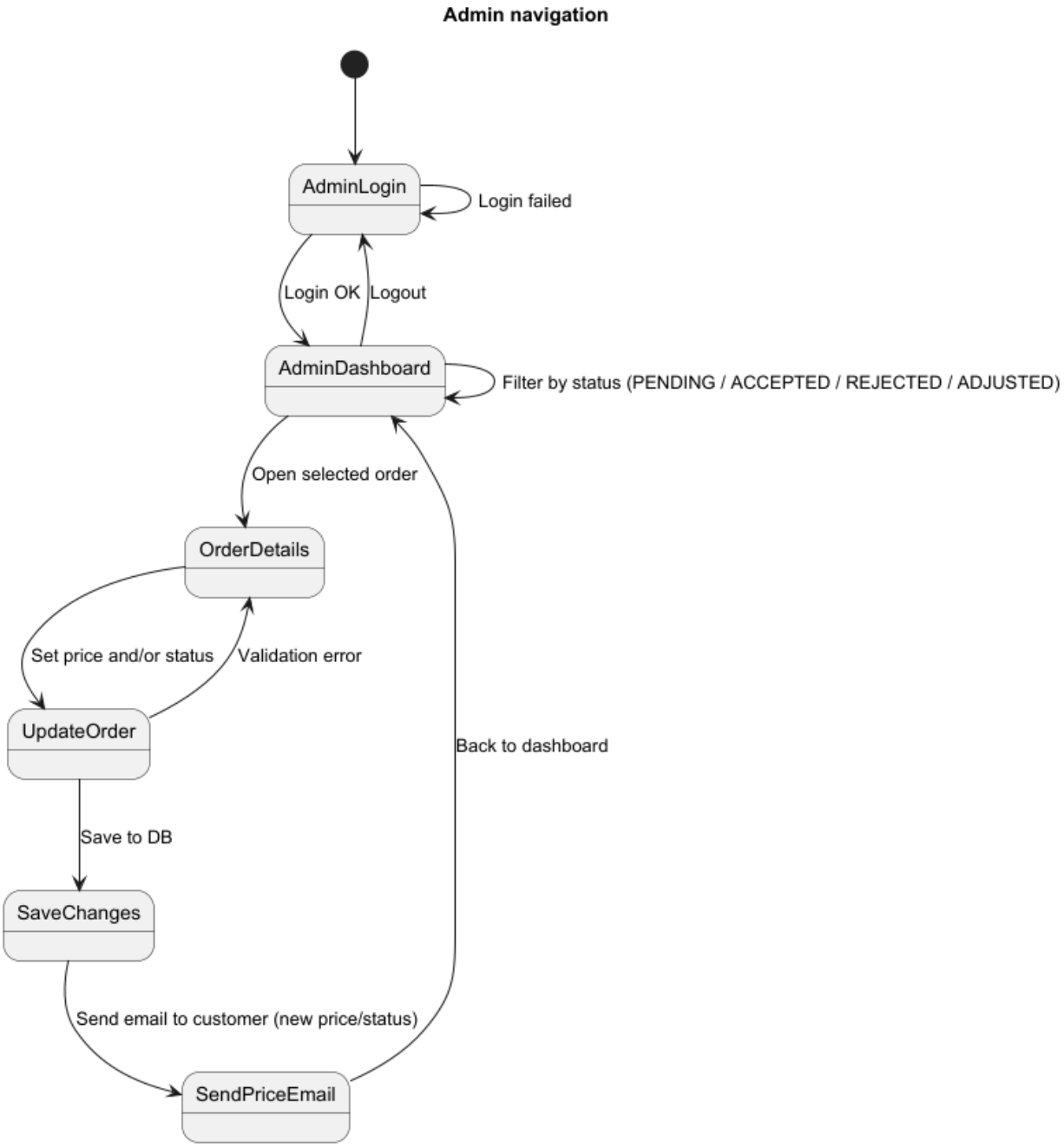
Admin flowet er adskilt fra kundeflowet og kræver login. En admin logger ind via en dedikeret loginside og får herefter adgang til et dashboard, som kun er tilgængeligt for brugere med gyldig admin session. Hvis login mislykkes, forbliver brugeren på login siden, og får fortalt at deres username og password var forkert. I admin-dashboardet vises en oversigt over alle ordrer, som kan filtreres efter status, f.eks. afventende, afviste, accepterede eller justerede ordrer. En admin kan vælge en specifik ordre for at se detaljer og fastsætte eller ændre prisen. Når prisen opdateres, sendes der automatisk en mail til kunden med den nye pris.

Navigations diagrammet viser samtidig sammenhængen mellem de anvendte Thymeleaf skabeloner, og de HTTP-routes, der forbinder dem. Hver side i diagrammet repræsenterer en konkret Thymeleaf template, mens pilene angiver de routes, der anvendes til at navigere mellem siderne. På den måde fungerer navigationsdiagrammet som et bindeled mellem den tekniske implementering og den oplevede navigation i systemet

10.1 Bilag: Kundenavigation



10.2 Bilag: Adminnavigation



11 VALG AF ARKITEKTUR

Projektet er opsat i en Model-View-Controller inspireret struktur (MVC). I IDE'et er klasserne delt op i forskellige enheder, packages, som giver en overskuelig struktur for programmets sammensætning. Der er følgende pakker som ses på figuren; config, controllers, entities, exceptions, persistence, services. Config holder på Thymeleaf-konfigurationen, hvilket bruges til at opsætte logikken der renderer HTML koden fra templates-folderen.

Entities holder på de klasser der kører som instanser i Java delen, som bliver brugt af mapperne. Mapperne ligger i persistence-pakken, og muliggør der kan hentes og indsættes data i databasen via metoder, der anvender entities og SQL til at overføre informationerne.

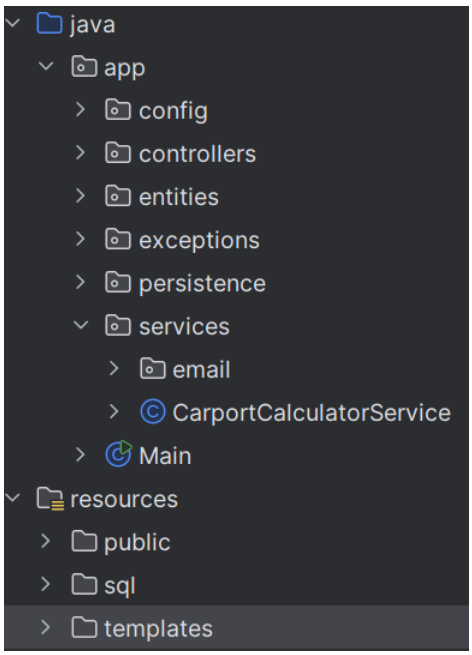
Controllerspakken indeholder de klasser som står for logikken mellem brugergrænsefladen (HTML) og dataene, der kommer fra backend. Exceptions pakken indeholder exception-håndteringen, som gør det muligt at have klasser til at håndtere mere specifikke fejl, der kan opstå.

Services pakken holder på email systemet, styklisteberegneren, og svg-generer, altså systemer der har mere specifikke formål, men som ikke har indflydelse på MVC-strukturen. Denne arkitektur gør systemet meget let at opdele. Fordelen ved denne separation, er at udviklerne i teamet kan arbejde på samme tid og mere uafhængigt af hinanden, samtidig med at koden er lettere at teste, udskifte, genbruge og skalere. Kildekoden gør brug af et par forskellige singletons, hvor det giver mening at have klasser, vis metoder skal kunne anvendes globalt.

Her er der følgende: ConnectionPool, en klasse givet af skolen til projektet, som står for at skabe forbindelse til databasen med en Hikari JDBC.
Så er der Mapper klasserne, der skal kunne kaldes fra relevante controllers og ikke behøver at instansieres, da de bruger den fælles JDBC. At de er singleton, betyder også at controllernes vil bruge samme mapper logik, og være mere ressourcevenlige, da der er færre objekter i hukommelsen. Til at konkretisere koden, bruges der en del hjælpeklasser.

Beregninger laves med Math klassen, som simplificerer koden i at den indeholder de basale numeriske operationer der bl.a. anvendes i styklisteberegneren. Email-systemet er oprettet med Jakarta, som er et hjælpebibliotek der indeholder en masse funktioner til dette. Både forbindelsen til mailserver og afsendelsen håndteres med metoder herfra. Struktureringen af email indholdet kommer fra hjemmelavet hjælpeklasse kaldet EmailTemplateBuilder.
Til at de mange data der skal passeres anvendes Lists hjælpebiblioteket. Lists er et interface, hvor der typisk bruges ArrayList klassen til at gemme objekter der skal enten udpakkes og sendes til databasen, eller fra databasen og pakket til at blive anvendt i Java. Entities' konstruktører, getters/setters og toString metoder bliver genereret med Lombok, hvilket reducerer kompleksiteten og gentageligheden.

11.1 Bilag: Systemarkitektur



12 SÆRLIGE FORHOLD

Dette afsnit beskriver særlige tekniske valg og løsninger, der er anvendt i applikationen, herunder håndtering af scopes, exception-håndtering, validering, sikkerhed og domænespecifik funktionalitet i Fog-projektet.

12.1 Scope-håndtering (Application, Session og Request)

Applikationen anvender forskellige **HTTP scopes** til at håndtere data på tværs af HTTP-anmodninger og sikre en effektiv brugeroplevelse uden unødvendig databaseadgang.
Session scope anvendes til data, der skal bevares gennem en brugers session.

Dette omfatter blandt andet den loggede admin-bruger, som gemmes i sessionen efter succesfuldt login for at opretholde autentificering mellem sider. Derudover gemmes kundens midlertidige carport-konfiguration (f.eks. mål, tagtype og eventuelt skur), mens kunden justerer sin ordre, så data ikke mistes ved genindlæsning af siden. Session scope bruges generelt til at bevare brugerdata mellem HTTP-requests.

Request scope anvendes til kortvarige data, der kun er relevante for én enkelt HTTP-anmodning. Dette inkluderer brugerinput fra formularer, valideringsfejl der vises til brugeren, samt succesbeskeder som bekræftelser på handlinger (f.eks. at en pris er gemt eller en ordre er opdateret).

12.2 Exception-håndtering og logging

Applikationen anvender proaktiv exception håndtering for at undgå nedbrud og for at sikre stabile flows, så anvendes der, try-catch blokke, i både mappers og controllers. Database relaterede exceptions såsom *SQLExceptions* opstår typisk i mapper laget og kastes videre til controllers gennem *DatabaseException*, som er en subklasse af superklassen *ApplicationException*.

I controllerne fanges disse exceptions og omsættes til fejlmeddelelser, som præsenteres for brugeren uden at eksponere tekniske eller følsomme detaljer. Under udvikling logges fejl via standard Java-mekanismer såsom *printStackTrace ()*. I et produktionsmiljø kan det erstattes af et dedikeret logging-framework for bedre sporbarhed og fejlanalyse.

12.3 Application scope

benyttes til globale ressourcer, som deles mellem alle brugere i systemet. Her gemmes blandt andet en database connection pool, der sikrer effektiv og skalerbar databaseadgang. Derudover lagres statiske konfigurationsdata såsom værdier til dropdown menues (f.eks. tagtyper og materialer), som kun indlæses én gang ved applikationens opstart.

12.4 Brugerinput-validering

Validering af brugerinput sker primært på server siden for at sikre data integritet. Input fra formularer valideres i controller-laget ved tjek for tomme værdier, gyldige talformater, såsom at priser skal være positive tal.
Parsing af numeriske værdier håndteres via try-catch for at fange format fejl, og eventuelle valideringsfejl sendes tilbage til brugeren som beskeder, der vises i Thymeleaf templates. Eventuel client-side validering kan anvendes som supplement for bedre brugeroplevelse, men server-side validering er den primære sikkerhedsmekanisme.

12.5 Sikkerhed og login

Loginfunktionaliteten er session baseret. Brugere autentificeres via email og password, som valideres mod data i databasen. Ved succesfuldt login gemmes admin-objektet i session scope. Adgang til beskyttede områder, såsom admin dashboardet, kontrolleres ved at tjekke en aktiv session med en logget admin.
Logout invaliderer sessionen fuldstændigt og fjerner dermed al brugerrelateret session-data. Passwords håndteres via sikre hash-mekanismer, hvilket forhindrer lagring af klartekst-kodeord.

12.6 Brugertyper og roller i databasen

Systemet anvender rollebaseret adgang, primært med rollerne Admin og Customer. Roller er defineret i databasen og repræsenteret i applikationen via entiteter og JDBC-mappere (f.eks. AdminMapper og CustomerMapper). JDBC anvender prepared statements for at beskytte mod SQL injection. Roller bruges til at begrænse adgangen til funktionalitet, så administrative handlinger som prisændringer og overblik over ordrer kun er tilgængelige for admins.

12.7 Fog-specifik funktionalitet

Fog-projektet indeholder domænespecifik funktionalitet, herunder stykliste og prisberegning, som udføres i service-laget baseret på kundens carport-konfiguration (mål, tagtype og materialer). Systemet understøtter desuden email afsendelse, hvor kunden informeres om ændringer, f.eks. opdaterede priser. Disse elementer er integreret med frontend via Thymeleaf og med databasen via en connection pool for skalerbarhed.

13 UDVALGTE KODEEKSEMPLER

13.1 Materialmapper

```
private static int toDbLength(int mm) {
    return (int) Math.ceil(mm / 10.0);
}

private static Integer fromDbLength(Integer dbLen) {
    if (dbLen == null) return null;
    return (dbLen > 0 && dbLen < 1000) ? dbLen * 10 : dbLen;
}

private static Material findByMinLengthAndCategory(int minLengthMM, String
category)
        throws DatabaseException {

    String sql = """
        SELECT
            mp.material_product_id,
            mp.material_product_name,
            mp.material_product_description,
            mp.length_mm,
            mp.material_price,
            u.unit_name,
            u.unit_short_name
        FROM material_product mp
        JOIN unit u ON mp.unit_id = u.unit_id
        JOIN material_category mc ON mp.material_category_id =
mc.material_category_id
        WHERE mc.material_category_name = ?
          AND mp.length_mm >= ?
        ORDER BY mp.length_mm ASC
        LIMIT 1
        """;

    int dbMin = toDbLength(minLengthMM);

    try (Connection connection =
ConnectionPool.getInstance().getConnection();
         PreparedStatement ps = connection.prepareStatement(sql)) {

        ps.setString(1, category);
        ps.setInt(2, dbMin);

        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                return new Material(
                        rs.getInt("material_product_id"),
```

```
                        rs.getString("material_product_name"),
                        rs.getString("material_product_description"),
                        fromDbLength(rs.getObject("length_mm",
Integer.class)),
                        rs.getBigDecimal("material_price"),
                        rs.getString("unit_name"),
                        rs.getString("unit_short_name")
                );
            }
        }

        throw new DatabaseException(
                "Manglende materiale",
                "Ingen materialer i kategori '" + category +
                "' med min længde " + minLengthMM + " mm"
        );

    } catch (SQLException e) {
        throw new DatabaseException(
                "DB-fejl",
                "findByMinLengthAndCategory failed: " + e.getMessage()
        );
    }
}
```

Koden her viser den centrale database adgang i **MaterialMapper**. Metoden **findByMinLengthAndCategory** anvendes til at hente det korteste materiale i en given kategori, som opfylder et minimumskrav til længde. Dette gøres ved at filtrere på materialekategori og minimumslængde, sortere materialerne stigende efter længde og vælge det første match.

For at sikre konsistens mellem applikationen og databasen anvendes hjælpemetoderne **toDbLength** og **fromDbLength** til at konvertere længder mellem millimeter i applikationen og det format, længderne er lagret i databasen. Databasen tilgås via PreparedStatement, hvilket sikrer korrekt parameterbinding og adskillelse mellem SQL og forretningslogik. Resultatet mappes til et **Material** domæneobjekt, som efterfølgende anvendes direkte i beregningslogikken i **CarportCalculatorService.**

13.2 CarportCalculator

```
public static void calculate(Order order) throws DatabaseException {
    List<Material> materials = new ArrayList<>();

    addPosts(order, materials);
    addRems(order, materials);
    addRafters(order, materials);
    addSterns(order, materials);
    addRoofSheets(order, materials);
    addShed(order, materials);

    BigDecimal materialsTotal = materials.stream()
            .map(Material::getTotalPrice)
            .reduce(BigDecimal.ZERO, BigDecimal::add);

    BigDecimal slopePrice = calculateSlopePrice(order);

    order.setMaterials(materials);
    order.setTotalPrice(materialsTotal.add(slopePrice));
}
```

CarportCalculatorService har en statisk calculate metode, som tager en ***Order*** og beregner både materialelisten og den samlede pris for ordren.

Metoden opretter først en lokal liste af Material, hvorefter den kalder en række hjælpe metoder (**addPosts, addRems, addRafters, addSterns, addRoofSheets og addShed),** som hver især beregner mængde og pris for en bestemt del af carporten og tilføjer materialer til listen.

Når alle materialer er tilføjet, summeres den samlede materialepris ved at reducere **getTotalPrice()** for hvert materiale. Derefter beregnes en eventuel ekstra pris baseret på tagets hældning, og til sidst sættes både materialelisten og den samlede pris på ordren via **order.setMaterials(materials) og order.setTotalPrice(...).**
Der anvendes **BigDecimal** til prisberegninger for at undgå afrundingsfejl, som kan opstå ved brug af double i finansielle beregninger.

13.3 addShed

```
private static void addShed(Order order, List<Material> materials) throws
DatabaseException {
    if (!(order instanceof OrderWithShed ows)) return;
    Shed shed = ows.getShed();
    if (shed == null) return;

    int perimeter = 2 * (shed.getWidthMM() + shed.getLengthMM());
    int boards = (int) Math.ceil(perimeter / 100.0) * 2;

    Material board =
MaterialMapper.findCladdingForHeight(SHED_WALL_HEIGHT_MM);

    BigDecimal price =
board.getUnitPrice().multiply(BigDecimal.valueOf(boards));

    materials.add(new Material(...));
}
```

addShed tilføjer kun materialer til skuret, hvis ordren er en **OrderWithShed**, og om der faktisk er angivet et shed objekt. Derefter beregnes skurets omkreds ud fra bredde og længde.

Ud fra omkredsen beregnes et antal beklædningsbrædder, hvor der afrundes opad for at sikre nok materiale. Det relevante beklædningsmateriale hentes fra databasen via **MaterialMapper**, prisen beregnes med **BigDecimal**, og materialet tilføjes til materialelisten, som senere indgår i totalprisen.

13.4 Tests

```
@Test
Void calculate_orderWithShed_setsMaterials_and_price() throws
DatabaseException {
   RoofType roof = new RoofType(1, "Fladt tag", 0, BigDecimal.ZERO);
   Order order = new OrderWithShed(
           "calc2@fog.dk",
           "PENDING",
           roof,
           3000,
           2200,
           5400,
           new ArrayList<>(),
           null,
           BigDecimal.ZERO,
           new Shed(2100, 2400)
   );
   CarportCalculatorService.calculate(order);
   assertNotNull(order.getMaterials());
   assertFalse(order.getMaterials().isEmpty());
   assertTrue(order.getTotalPrice().compareTo(BigDecimal.ZERO) > 0);
}
```

Denne snippet tester på ***CarportCalculatorService.calculate(order),*** som er en kernemetode for systemets forretningslogik. Hvis testen skulle fejle, ville der ikke kunne leveres en tryg beregning af materialer og pris, og derfor er det altså en central del af testningen, at dette lykkes.

Her bliver afhængigheden af **MaterialMapper** også indirekte testet, idet *calculate()* funktionen benytter den til databasefunktionaliteten. Med ***assertFalse(...)* og *assertTrue(...),*** får man altså en sikker test på resultatet af beregningen, dog testes der ikke hvordan.

Ergo hvis beregningsalgoritmen skulle blive ændret, ville testen stadig kunne godkendes, og dette gør den stærk til refaktorering. Grunden til at ***AssertTrue(...compareTo(BigDecimal.ZERO) > 0)*** tester om prisen udregnes til større end nul, er at det undgår en skrøbelig test, hvilket prisberegningen er pålidelig til, gennem dens afhængighed af flere variable faktorer.

13.5 Exceptions

```
protected ApplicationException(String userMessage, String systemMessage) {
   super(systemMessage);
   this.userMessage = userMessage;
}
```

Håndtering af exceptions foregår ved en abstrakt ApplicationException klasse, der nedarver til specialiserede exception handlers. Metoden der bruges, tager en userMessage og en systemMessage i parameter til fejlbeskeder, som der bliver videregivet til henholdsvis brugerne og udviklerne.

Primært anvendes DatabaseException i systemet, da de fleste exceptions der kan forekomme, vil være fejl i databaseinteraktionerne. Her er det vigtigt at brugeren får en fejlbesked uden systemoplysninger som kan lække sårbare data, mens udviklerne får en systembesked med tekniske oplysninger der kan logges og hjælpe debugging. Dette gøres via metoden vist.

14 STATUS PÅ IMPLEMENTERING

Der er ikke implementeret fuld CRUD-funktionalitet for alle tabeller i databasen. Dette er i flere tilfælde et bevidst designvalg baseret på systemets forretningslogik og krav til dataintegritet. F.eks. kan kunder og ordrer ikke slettes, da disse indeholder historiske forretningsdata, som skal bevares. I stedet anvendes statusfelter til at styre ordre flowet. Hvor CRUD-funktionalitet mangler, skyldes det således enten bevidste fravalg eller tidsmæssige begrænsninger i projektet.

Der blev ikke udviklet en egentlig betalingsløsning til kunden. Når en sælger fastsætter prisen på en carport, modtager kunden en email med den samlede pris, men mailen indeholder ikke et link til en betalingsside. En egentlig betalingsproces ville kræve integration med en ekstern betalingsmåde, hvilket ikke er nået inden for projektets tidsramme.

Der blev heller ikke udviklet et brugerinterface til oprettelse af nye administratorer. Selvom der findes persistent logik til oprettelse af administratorer, er der ikke implementeret et tilhørende admin interface, og der er heller ikke fuldt implementeret sikker håndtering af administrator-passwords, herunder hashing.

Desuden er oprettelse af en SVG-tegning af carporten (US-4) ikke blevet implementeret. Det var oprindeligt planen, at kunden skulle modtage en SVG-visualisering af carporten baseret på de indtastede mål, sendt som vedhæftet fil i en email og vist på hjemmesiden til kunden, men denne funktionalitet blev nedprioriteret for at færdiggøre kernefunktionaliteten i systemet.

Afslutningsvis kan der forekomme mindre fejl eller mangler, som er identificeret sent i udviklingsforløbet, men som ikke er nået at blive rettet inden aflevering. Dette kan f.eks. omfatte utilstrækkelig session-håndtering på enkelte sider. Der kan desuden være tests, som ikke er fuldt færdiggjort eller som fejler på afleveringstidspunktet.

15 KVALITETSSIKRING

15.1Automatiserede tests

Testklasse	Testet klasse	Testede metoder	Testtype
AdminMapperTest	AdminMapper	login()	Integrationstest
CustomerMapperTest	CustomerMapper	registerCustomer() getCustomerByEmail() isEmailInSystem() getAllCustomers()	Integrationstest
MaterialMapperTest	MaterialMapper	getAllMaterialsFromOrder()	Integrationstest
OrderMapperTest	OrderMapper	getOrderByOrderId() changeOrderPrice()	Integrationstest
CarportCalculatorServiceTest	CarportCalculatorService	calculate()	Integrationstest/servicetest

Der testes med JUnit 5 for at kvalitetssikre systemet. Hovedsageligt er det integrationstest, der er vigtige, da hele systemet afhænger af databasen. Hvis ikke databasen spiller ordentligt sammen med koden, vil systemet slet ikke kunne køre. Testene fungerer følgende:

15.1.1 AdminMapperTest

AdminMapperTest målsætter login() funktionen, som via SQL verificerer at en admins brugernavn og password er i databasen. Testen virker ved at den seeder en bruger i test-databasen med @BeforeEach, og så prøver login() funktionen med henholdsvis den seedede brugers oplysninger, og én hvor oplysningerne er forkert. Altså tester den både det positive og negative scenarie der

kan ske når der logges ind. Dette giver 100% dækning af login() som er den eneste metode i mapperen.

15.1.2 CustomerMapperTest

CustomerMapperTest tester alle funktioner fra CustomerMapper, heraf oprettelse af en kunde i registerCustomer(), hentning af en kunde via email i getCustomerByEmail(), kontrol for om email findes i systemet med isEmailInSystem(), og hentningen af alle kunder i getAllCustomers(). Her testes der fra data der er lagt direkte i test-databasen som kan findes i /resources/testDatabase.sql. Dækningsgraden her er ikke helt gennemført, da der ikke testes under manglende data og fejl, dog bliver forbindelsen og korrekt passering af data verificeret.

15.1.3 MaterialMapperTest

MaterialMapperTest fokuserer primært på en enkelt metode, getAllMaterialsFromOrder(). Der verificeres om ordre og materialer joines korrekt, om mapping af mængde, pris, noter og materialedata er korrekt, og om håndteringen længdekonvertering fra databaseformat fungerer. Dækningsgraden her er ret lav da det kun er den mest kritiske persistence-funktion der bliver dækket med integrationstest. Dog bliver interne hjælpefunktioner som findByMinLengthAndCategory() testet indirekte gennem service-laget.

15.1.4 OrderMapperTest

OrderMapperTest tester de centrale forretningskritiske metoder getOrderById()og chageOrderPrice(). Disse tests undersøger om opbygningen af Order og OrderWithShed er korrekt. Om ordreprisen opdateres rigtigt, og om historikken oprettes ordentligt i customer_order_change. Dækningen her er mere minimal, og der kunne godt blive testet en del mere, dog var der steder som måtte prioriteres pga. tidspres.

15.1.5 CarportCalculatorServiceTest

CarportCalculatorService er mere en både en integrationstest og service test, som sikrer kvaliteten af den vigtige styklisteberegner. Her verificeres både en almindelig carport, samt en carport med skur om, at ordren bliver korrekt udregnet med calculate(). Der tjekkes om materialelisten udfyldes, om materialer vælges rigtigt med MaterialMapper og om at prisen er korrekt. Dette dækker de vigtigste scenarier i koden.

Som det fremgår, er koden desværre ikke fuldt dækket af tests, hvilket ville have været optimalt. Alle metoder i persistens er ikke direkte testet, og det er uden tvivl en sårbarhed for koden, at der ikke er denne sikring. Samtidig er det primært succes-scenarier der testes, hvor ting som

datamangel og ugyldige input ikke testes alle steder - som også giver nogle huller i kvaliteten. Givet mere tid skal dette selvfølgelig ændres.

16 USER ACCEPTANCE TEST

User story	User Acceptance Status
US-1: Valg af tagtype *Som kunde ønsker jeg at kunne vælge imellem tag med rejsning og fladt tag på min carport, så jeg kan få en løsning der passer til mine behov.*	Accepteret
US-2: Valg af bredde og længde *Som kunde ønsker jeg at kunne vælge bredde og længde på min carport i en dropdown menu, så jeg kan få en carport der passer til min grund*	Accepteret
US-3: Redskabsrum *Som kunde ønsker jeg at kunne tilføje et redskabsrum til min carport, så jeg har ekstra opbevaringsplads.*	Accepteret
US-4: 2D Visualisering *Som kunde ønsker jeg at se en realtids visualisering i 2D model af carporten med mine specifikationer, så jeg kan være sikker på at resultatet er som forventet.*	Ikke-Accepteret. 2D visualiseringen blev desværre fraskåret, da problemer med visningen i HTML ikke nåede at blive løst i tide. SVG service-klasserne kom altså ikke med i deployment.
US-5: Send forespørgsel	Accepteret

Som kunde ønsker jeg at kunne sende min carport forespørgsel, så jeg kan modtage et tilbud fra FOG.	
US-6: Modtag tilbud via email Som kunde ønsker jeg at modtage et tilbud via email, så jeg kan se prisen og detaljerne på carporten.	Accepteret
US-7: Se kundeordrer med beregnet pris som admin *Som admin ønsker jeg at kunne se en oversigt over alle kundeordrer med automatisk beregnet pris, så jeg har overblik over indkomne ordrer.*	Accepteret
US-8: Rediger pris for ordre *Som admin ønsker jeg at kunne redigere den beregnede pris for en kundeordre, så jeg kan justere tilbuddet efter behov.*	Accepteret
US-9: Tilføj note til ordre *Som admin ønsker jeg at kunne tilføje noter til en kundeordre, så jeg kan dokumentere særlige ønsker eller detaljer.*	Accepteret
US-10: Send tilbud til kunde *Som admin ønsker jeg at kunne sende et tilbud til kunden via email, så kunden kan modtage og vurdere tilbuddet.*	Accepteret

17 PROCES

17.1 Arbejdsprocessen faktuelt

Projektet blev planlagt til at blive gennemført over fem faser og anvende TDD, hvor hver fase skulle svare til perioden mellem vejledningsmøderne. Denne opdeling skulle understøtte en iterativ udviklingsproces med løbende planlægning og prioritering af funktionalitet.

I den første fase skulle opgaven analyseres med fokus på forretningskrav og kundens behov. Der skulle udarbejdes relevante diagrammer, herunder databasemodellering og overordnede systemdiagrammer, som skulle danne grundlag for fastlæggelsen af projektets centrale krav og funktioner.

I anden fase skulle projektets grundstruktur etableres. GitHub projektet skulle opsættes, Maven konfigurationen (pom.xml) skulle integreres, og projektets pakkestruktur skulle defineres. Derudover skulle de nødvendige entiteter oprettes, og arbejdet med persistens laget skulle påbegyndes gennem implementering af mappers.

Tredje fase skulle have fokus på færdiggørelse af mappers samt påbegyndelse af udviklingen af controllers. Samtidig skulle carportberegneren implementeres, og der skulle arbejdes med håndtering af CSV-data med henblik på korrekt databehandling.

I fjerde fase skulle controllers færdiggøres, exception-håndtering implementeres, og systemets resterende funktionalitet afsluttes. Der skulle afsættes tid til deployment og efterfølgende fejlfinding for at sikre, at systemet fungerede stabilt i et produktions lignende miljø.

Den femte og sidste fase skulle anvendes til udarbejdelse af rapporten samt kvalitetssikring og rettelse af eventuelle fejl, der kunne opstå i forbindelse med deployment og afsluttende tests. Kommunikationen i teamet var planlagt til at være struktureret og kontinuerlig gennem hele projektforløbet. Gruppen havde aftalt, at der skulle afholdes fysiske møder hver hverdag kl. 09.30, undtagen i weekender, og at Discord skulle anvendes til løbende koordinering, vidensdeling og varsling af eventuelle forsinkelser. Forud for hver uge skulle der udarbejdes en plan, så alle gruppemedlemmer havde et klart overblik over opgaver og ansvar. Der skulle anvendes KanBan til at følge projektets fremdrift, hvor opgaver skulle organiseres efter status og gøres synlige for hele teamet.

17.2 Arbejdsprocessen – reflekteret

Arbejdsprocessen i projektet var overordnet velstruktureret, men udviklede sig løbende i takt med gruppens erfaringer og de udfordringer, der opstod undervejs. I den indledende fase fungerede den faste struktur med daglige møder kl. 09.30 og klare faser godt og gav et tydeligt overblik over opgaver og ansvar. Dette bidrog til hurtigt at etablere en fælles forståelse af både forretningskrav og teknisk retning. Samtidig viste det sig, at denne struktur krævede løbende vedligehold for at forblive effektiv.

Kanban blev anvendt som værktøj til at skabe overblik over projektets fremdrift. Kanban Master-rollen fungerede grundlæggende, men der opstod udfordringer med konsekvent opdatering af boardet. I perioder blev opgaver løst uden at blive flyttet korrekt mellem kolonnerne, hvilket reducerede gennemsigtigheden. Dette gjorde det vanskeligere at vurdere fremdrift og arbejdsbelastning. Erfaringen herfra var, at Kanban kun fungerer optimalt, hvis alle i gruppen tager ansvar for løbende opdatering, og at rollen som Kanban Master med fordel kunne have været mere aktiv.

Evalueringsmøderne havde fokus på fremdrift i forhold til user stories, tekniske udfordringer og prioritering af funktionalitet. Især carportberegneren, databaseintegration og ordreflowet var tilbagevendende emner. Møderne var mest effektive, når de tog udgangspunkt i konkrete problemstillinger og beslutninger, hvilket tydeliggjorde vigtigheden af forberedelse og klare mål for hvert møde.

Estimeringerne var ikke altid præcise, særligt i projektets anden halvdel. Dette skyldtes blandt andet undervurdering af kompleksiteten i forretningslogikken samt den tid, der blev brugt på fejlsøgning og integration mellem systemets lag. Når estimeringerne ikke holdt, blev opgaver opdelt yderligere eller flyttet til senere iterationer. Dette lærte gruppen at være mere fleksibel i planlægningen og bedre til at justere undervejs.

En positiv og produktiv arbejdsrytme blev fundet i første halvdel af projektperioden, hvor gruppen havde opnået en fælles forståelse af arkitektur, opgavefordeling og tekniske udfordringer. En væsentlig udfordring i processen var håndtering af sygdom i gruppen, da der ikke var en fast proces for kompetencedeling. Dette førte til enkelte setbacks og resulterede i, at én af user stories ikke blev fuldt implementeret. Erfaringen herfra er, at tydeligere dokumentation og en klarere overdragelsesproces kunne have reduceret konsekvenserne.

18 KONKLUSION

I dette semesterprojekt har vi udviklet et webbaseret konfigurations- og ordresystem til Fog, der skal reducere manuelt arbejde og fejl i deres nuværende carport proces. Løsningen samler kundens bestilling, beregning af stykliste og pris samt sælgerens administration i et system, så oplysninger ikke længere skal genindtastes på tværs af flere platforme. Kunden kan konfigurere carportens mål, tagtype og eventuelt redskabsrum og sende en forespørgsel, hvorefter en admin kan få overblik over ordrer, justere pris, tilføje note og sende tilbud via email. Dermed understøtter systemet et kortere og mere effektivt flow, hvor centrale dele af processen automatiseres uden at fjerne kundekontakten.

Teknisk er projektet bygget i Java med Javalin og Thymeleaf, og data persisteres i PostgreSQL via JDBC. Vi har arbejdet med en MVC inspireret arkitektur, hvor ansvar er adskilt i controllers, persistence (mappers), services og entities. Det har gjort koden mere overskuelig og nemmere at udvide og teste. I databasedesignet har vi fokuseret på normalisering og tydelige relationer, så materialer, enheder og kategorier kan genbruges på tværs af ordrer, og så historik på ændringer kan gemmes frem for at overskrive data. Særligt pris og materialeberegningen i servicelaget har været en kerneopgave, hvor BigDecimal er anvendt for at sikre stabile finansielle beregninger.

Projektet blev kvalitetssikret med integrationstests i JUnit 5, primært fordi systemets funktionalitet er tæt koblet til databasen. Samtidig viser vores evaluering, at test dækningen ikke er fuld, og at der især mangler flere fejl og edge-case tests. Ikke alle planlagte funktioner blev implementeret inden deadline, herunder 2D/SVG-visualisering og en egentlig betalingsløsning, og enkelte administrative features (fx fuldt interface til admin-oprettelse og komplet password-håndtering) er kun delvist gennemført.

Procesmæssigt har projektet understreget, hvor stor en forskel et aktivt Kanban board gør for overblik og fremdrift. I starten gav boardet en fælles retning og tydelig prioritering, hvilket gjorde det nemt at koordinere opgaver og holde tempo. Undervejs blev det dog tydeligt, at værdien falder, hvis boardet ikke vedligeholdes løbende: når opgaver ikke bliver synlige og opdateret, bliver prioriteringer mindre klare, og det er sværere at opdage flaskehalse i tide. Erfaringen er derfor, at Kanban fungerer bedst som et dagligt, fælles styringsværktøj – ikke kun som planlægning i starten.

Projektet har også vist os, at risikoanalyse giver mest værdi, når den omsættes til konkrete aftaler og arbejdsgange. Vi havde identificeret relevante risici, men vi kunne have operationaliseret dem bedre, fx med tydeligere dokumentation, faste rutiner for videns overdragelse og en mere bevidst fordeling af kritiske opgaver, samt en fastlagt lederrolle der blev passeret videre, hvis lederen blev syg. Det ville have gjort os mere robuste ved fravær og uforudsete hændelser og dermed styrket vores evne til at holde planen.

Samlet set har projektet givet os solid erfaring med at omsætte forretningskrav til en konkret webapplikation med database, arkitektur, beregningslogik, deployment og test — og samtidig lært os, at procesdisciplin (Kanban + reel risikohåndtering) er afgørende for at holde retning, kvalitet og tempo i et teamprojekt.