

Fog®Carport

2. Sem
DAT A
31/5/23

Bastian Holm
cph-bh213@cphbusiness.dk

Christian Stennicke
cph-cs519@cphbusiness.dk

Sander Roust
cph-sr319@cphbusiness.dk

Tobias Berg
cph-tb266@cphbusiness.dk

Tobias Christiansen
cph-tc184@cphbusiness.dk



INDHOLDSFORTEGNELSE

LINKS:	1
<i>Projekt</i>	1
<i>Produktvideo</i>	1
<i>Klassediagram</i>	1
<i>Logbog</i>	1
<i>Deployet version</i>	1
<i>Adminbruger</i>	1
INDLEDNING	1
BAGGRUND	2
VIRKSOMHEDEN/FORRETNINGSFORSTÅELSE	3
SWOT-ANALYSE	4
INTERESSENTANALYSE	5
TEKNOLOGIVALG	7
KRAV	7
ARBEJDSGANGE DER SKAL IT-STØTTES.....	7
<i>AS-IS</i>	7
<i>TO-BE</i>	8
USER STORIES - WEBAPPLIKATION.....	9
<i>US-1:</i>	9
<i>US-2:</i>	10
<i>US-3:</i>	10
<i>US-4:</i>	10
<i>US-5:</i>	11
<i>US-6:</i>	11
<i>US-7:</i>	11
USER STORIES - 3D-PRINT APPLIKATION	12
<i>US-1:</i>	12
<i>US-2:</i>	12
AKTIVITETSDIAGRAM	12
FØR LOGIN	13
EFTER LOGIN	14
DOMÆNEMODEL OG EER-DIAGRAM	15
<i>Domænemodel</i>	15
<i>EER-diagram</i>	15
NAVIGATIONSDIAGRAM & MOCKUPS	17
VALG AF ARKITEKTUR	19
MVC-ARKITEKTUR.....	19
FACADE-PATTERN	20
DEPENDENCY-INJECTION	20
SÆRLIGE FORHOLD	21
<i>Session</i>	21
<i>Exceptions</i>	21

<i>Brugerinputvalidering</i>	21
<i>3D-Print</i>	21
<i>Styklisteberegner</i>	22
UDVALGTE KODEEKSEMPLER	23
STATUS PÅ IMPLEMENTERING	25
<i>CRUD</i>	26
<i>Fejl/mangler på målstregen</i>	26
TEST	27
PROCES	30
ARBEJDSPROCESSEN FAKTUELT	30
ARBEJDSPROCESSEN REFLEKTERET	32
BILAG	34

Links:

Projekt

- <https://github.com/TobiasTheDanish/SemesterOpgave>

Produktvideo

- <https://youtu.be/RQtUkRdwIUU>

Klassediagram

- [class-diagram-fog.svg](#)

Logbog

- <https://github.com/TobiasTheDanish/SemesterOpgave/blob/master/Logbog/Logbog.pdf>

Deployet version

- <http://104.248.131.254:8080/Semesteropgave/>

Adminbruger

- Mail: test@test.dk
- Kode: 123

Indledning

Fog er en anerkendt leverandør af carporte, for at forbedre deres kundeoplevelse og effektivisere deres bestillingsprocess har de besluttet at udvikle en ny webapplikation. Den webapplikation vi har udviklet for Fog, giver brugerne mulighed for at logge ind som bruger, bestille en carport med specifikke mål, redigere og fjerne ordrer, derudover er der udviklet en adminside, her kan administratorer fjerne kundeordrer, se info på ordren og sætte status på ordren. Den nye webapplikation reducerer den manuelle proces og gør det nemmere for både kunder og Fog at administrere ordrer.

Baggrund

Johannes Fog A/S blev grundlagt i 1920.

I 1970 blev virksomheden omdannet fra en personligt ejet virksomhed til et aktieselskab, og i 1973 blev aktierne overdraget til en velgørende fond. Fra 2005 har de markedsført sig under navnet Fog.

Johannes Fog er en virksomhed, som hjælper den normale kunde med alle former for bygge- og/eller renoveringsprojekter. Det kan være rådgivning til f.eks. design, materialer og værktøj.

Deres byggecentre har alt til hus og have inden for bl.a. værktøj, maling, bad, beslag, el, lamper og pejseartikler.

I Trælasterne kan du finde træ og byggematerialer til alle slags opgaver lige fra carporte til hegning og terrasser. De har et stort udvalg, hvor du kan vælge mellem mange muligheder, både hvad angår størrelser, materialer og kvaliteter.

Fog består af ni Trælast & Byggecentre fordelt på Sjælland – i Hørsholm, Fredensborg, Kvistgård, Helsingør, Lyngby, Ølstykke, Herlev, Farum og Vordingborg. I Vordingborg er de eksperter i salg af jern, stål og andre metaller.

Johannes Fog's nuværende program er udviklet af en gammel kollega tilbage i 1999, Bergman IT-Service. Udfordringen med programmet er at det er en lokal applikation, hvor Fog kan logge ind med et password fra én computer. Det ville sige, at hvis computeren skulle bryde sammen eller slå fejl, så mister de adgangen til deres system.

Udover det kan de også kun ændre priser på varerne, men ikke oprette nye, slette, ændre navn osv.

Altså mangler Johannes Fog en optimeret applikation, der er mere tilgængelig for dem og giver dem mulighed for at fjerne, tilføje, slette og redigere.

Kort fortalt skal de have en applikation hvor kunden kan oprette en profil og logge ind. Herefter skal de kunne bestille en unik carport ud fra egne mål. Kunden skal kunne slette/redigere ordre, inden orden kommer i en "behandlingsfase".

Fog skal herefter have adgang til at se alle ordrer, de tilsvarende styklister og priser. De skal kunne bestemme hvilken fase orden er i, så kunden derhjemme også kan følge med.

Fog skal også have en applikation hvor de kan indtaste de givne mål fra en ordre, og derefter få en 3D version af carporten.

Virksomheden/Forretningsforståelse

Tidligt i processen blev der lavet diagrammer og modeller til projektet, så der var noget at bygge ud fra. Herved fik gruppen også sat sig sammen og lavet en SWOT-analyse, hvor alle kunne bidrage med input til de fire faktorer. Da størstedelen af gruppen har arbejdet sammen i tidligere projekter på uddannelsen, var det ikke så svært at finde både mangler og styrker. En fælles betegnelse ved styrkerne er, at det er noget, gruppen har arbejdet med før. I styrker er der blevet lagt vægt på både individuelle og samlede styrker lige så vel som der er gjort i svagheder.

SWOT-analyse

Styrker (Interne faktorer)	Svagheder (Interne faktorer)
<ul style="list-style-type: none">- Disciplin- Java (Servlets)- Tomcat- MySQL database- HTML og CSS (JSP)- God kommunikation og samarbejde- Struktur (kanban, fordeling af opgaver)- Github (branches)	<ul style="list-style-type: none">- Manglende erfaring med 3D-print- Erhvervsarbejde- Modeller og diagrammer- Droplet/DigitalOcean- Github (pull request, issues, projects)
Muligheder (eksterne faktorer)	Trusler (eksterne faktorer)
<ul style="list-style-type: none">- Forældet software hos kunden- Ressourcer til kompetenceudvikling	<ul style="list-style-type: none">- Dårlig/Ødelagt 3D-printer- Sygdom

[Figur 1](#)

Nogle af punkterne indenfor *styrker* ligger i samme boldgade, her kunne der blandt andet være tale om disciplin og struktur, hvilket er vigtigt, når der laves gruppearbejde. Den måde, det menes på, er at gruppen altid har været gode til at holde det daglige møde omkring formiddag. Så har gruppen mødtes og snakket igennem hvad status er og hvad der skulle laves den enkelte dag, hvorefter gruppen ofte har siddet klar på Discord til at hjælpe hinanden hvis det skulle være. Hvis der har været planer, har aftalen været at man måtte kode når man kun ne og så måtte man individuelt hve fat i hinanden ved spørgsmål. Noget andet der har hjulpet til strukturen og fordeling af opgaver har været Kanban-boardet inde på GitHub, hvor gruppen ellers tidligere har gjort brug af Trello. Det har været mere intuitivt at bruge den, da den ligger på GitHub sammen med resten af projektet.

Som det kan ses på [figur 1](#), kunne der ikke findes så mange trusler som vi ikke kunne påvirke, og det er heller ikke noget hvor der har været grund til bekymring. Der har heldigvis ikke været de store problemer med sygdom,

men angående dårlig/ødelagt 3D-printer, så var det lige nøjagtigt hvad vi blev mødt af. Gentagne gange forsøgte gruppen at printet en 3D-model af en carport, men printeren ville simpelthen ikke samarbejde. Det er en ulempe for projektet, men det illustrerer fint hvorfor SWOT-analysen kan fungere som god forberedelse til projekter.

Som skrevet tidligere har gruppen lavet denne her SWOT-analyse på forhånd for projektet, så nogle svagheder er også blevet skrevet af den grund at det var nyt for os. Der kan man for eksempel nævne de tre ting fra Github "*pull request, issues, projects*". De tre ting var nye for alle, men det må siges at have virket godt. Pull requests har været rigtigt gode til at sørge for, at der blev merged korrekt ind til master. Vi har låst master-branchen, så det ikke var muligt at merge ind i den uden brug af pull request. Derudover var det også smart, at man fik set andre gruppemedlemmers kode før den blev sat ind, så det er lettere for alle i gruppen at være opdateret.

Interessentanalyse

Gruppens interessentanalyse [figur 2](#) blev lavet tidligt i processen så det var lettere at identificere og forstå de interesserter, der ville blive påvirket og samtidigt ville påvirke projektet og slutresultatet. Det var fra starten tydeligt, at ressourcepersonen var Martin fra *Fog*, da websiden lader til at være hans projekt. Det er noget, han arbejder med hver dag og han har tydeligvis både stor indflydelse og en masse holdninger til hvordan siden skal fungere. Det giver fin mening da han har arbejdet med den længe og dermed også er helt inde i hvad der tidligere har været både godt og skidt.

	Stor indflydelse	Lille indflydelse
Bliver påvirket af projektet	Ressourceperson - Kunden (Martin fra fog)	Gidsler - Medarbejdere hos fog - Slutbrugere
Bliver ikke påvirket af projektet	Grå eminence - Bestyrelse/Investorer	Eksterne interesserter - Undervisere/vejledere

Figur 2

Ressourcepersonen i dette tilfælde, Martin, prøver jo at sørge for den bedste side for både forbruger og for Fog. Derfor har det hjulpet gruppen at lave denne her interessentanalyse så vi bedre kunne forstå behov og forventninger, og hvis holdninger der var vigtigst. Hvis man skulle lede efter en konsekvens som denne analyse ligger op til, så ville det formentlig gå ud over *gidslerne* og mere præcist slutbrugerne, da det går ud over deres oplevelse på websiden.

Samlet har de forudgående analyser hjulpet gruppen med at få en klar retning og fokus på hvilke udfordringer der har kunne opstå, så der ville være større muligheder for at opnå succes. Det har samtidigt gjort at gruppen har fået en god forståelse for selve virksomheden og hvad der var behov for at blive lavet.

Teknologivalg

Webapplikationen er udviklet ved brug af følgende teknologier og software.

Teknologi	Software
Java 17 (Amazon Corretto 11.0.18)	IntelliJ 2021.3
MySQL Workbench 8.0 CE	MySQL 8.0
JDBC (mysql-connector-java 8.0.30)	Tomcat 9.0.73
HikariCP 4.0.3	
javax.servlet-api (4.0.1)	
HTML, CSS & JS	
JSTL 1.2	
Bootstrap 5.1.3	
Slf4j 1.7.36	
Junit 5.8.2	

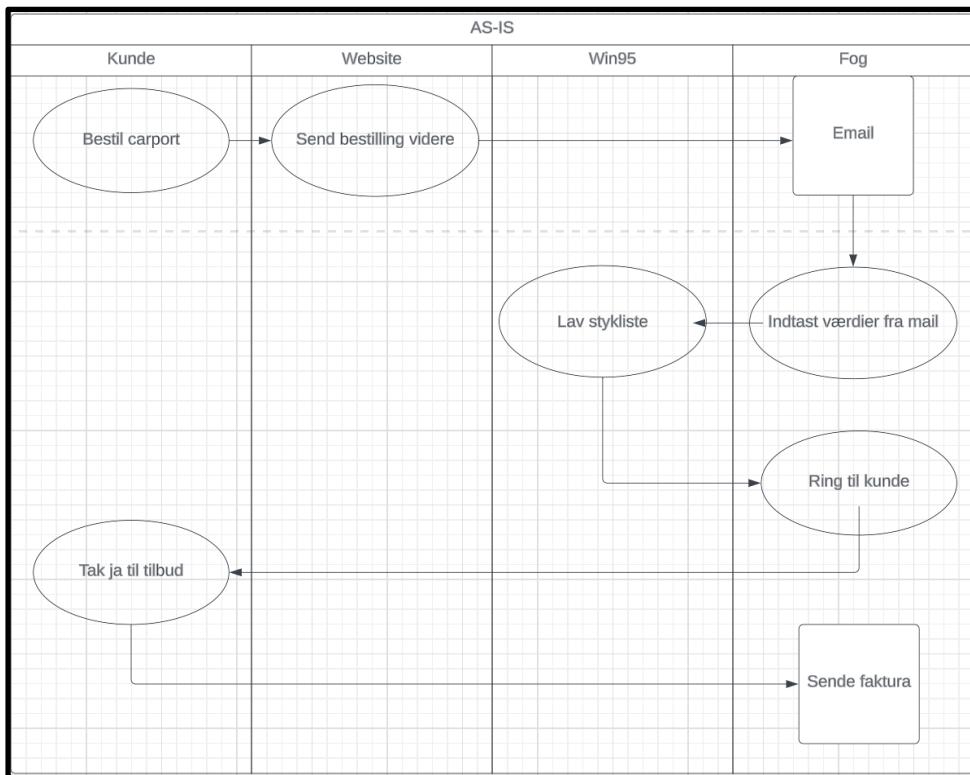
Krav

Arbejdsgange der skal IT-støttes

AS-IS

På vores AS-IS-diagram, som ses på [figur 3](#), kan man se hvordan hele bestillings- og købsprocessen fungerer. Den starter med at kunden skal bestille en carport fra hjemmesiden. Herfra sender hjemmesiden en e-mail ind til Fog om at der er

blevet lagt en bestilling med nogle specifikke værdier på de forskellige dimensioner. Herfra skal Fog manuelt tage værdierne fra e-mailen og smide ind i deres applikation som laver en stykliste og pris til kunden. Fog skal herefter ringe kunden op og oplyse om prisen, hvis kunden siger ja skal Fog til sidst sende dem en faktura.

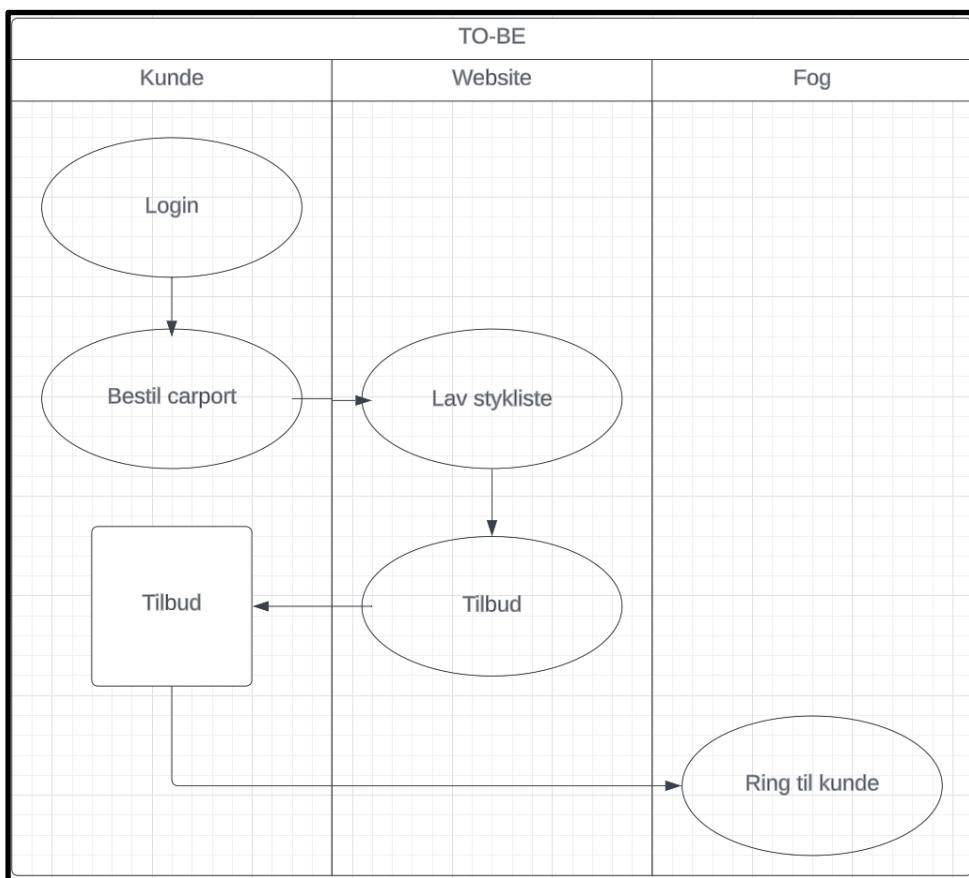


Figur 3

TO-BE

På vores TO-BE diagram, som ses på [figur 4](#), kan man se samme proces som sker på [figur 3](#), men her er det med vores applikation. Det der sker her er, at kunden logger ind og bestiller en carport, herefter laver vores applikation automatisk en stykliste til de ønskede dimensioner, samt laver en kostpris på alle materialer. Meningen er så at kunden skal modtage det her automatiserede tilbud med en aftalt dækningsgrad fra Fog, hvor de kan takke ja eller nej. Til sidst vil Fog ringe kunden op for at snakke diverse ting igennem om processen. Her har Fog altså

kun én aktiv opgave, som de først skal gøre, når der er takket ja til et tilbud.



Figur 4

User Stories - Webapplikation

US-1:

Som kunde skal jeg kunne bestille et tilbud på en carport med givne mål

Acceptkriterier: Givet at jeg som kunde er logget ind på systemet, når jeg har indtastet mål og trykket på bestil carport, så skal der oprettes en ordre i systemet og kunden skal sendes videre til en kvitteringsside.

Vigtighed: Need to have

Estimat: Large

US-2:

Som kunde kan jeg oprette en konto/profil

Acceptkriterier: Givet at jeg som kunde ikke har en konto på siden, når jeg har udfyldt en formular med e-mail, password og confirmation password, så skal jeg oprettes som bruger i databasen, og logges ind i systemet.

Vigtighed: Need to have

Estimat: Medium

US-3:

Som kunde kan jeg tilgå en profilside, hvor jeg kan redigere mine oplysninger.

Acceptkriterier: Givet at brugeren er logget ind, når de navigere til profilsiden via navbaren, så kan de redigere i de givne oplysninger.

Vigtighed: Nice to have

Estimat: Medium

US-4:

Som kunde eller administrator kan jeg logge på systemet med e-mail og kodeord. Når jeg er logget på, skal jeg kunne se min email på hver side

Acceptkriterier: Givet at man som vilkårlig rolle er logget ind på siden, så skal ens email være synlig i navbaren uafhængigt af hvilken side du befinner dig på.

Vigtighed: Nice to have

Estimat: Small

US-5:

Som administrator kan jeg se alle kunder i systemet og deres ordrer, så jeg kan følge op på ordrer, rette i ordrestatus og holde styr på mine kunder.

Acceptkriterier: Givet at brugeren har rollen administrator, når brugeren nавigerer til ordre-siden, så kan brugeren se alle ordrer i systemet, og rette i status.

Vigtighed: Need to have

Estimat: Medium

US-6:

Som kunde kan jeg fjerne en ordre fra min ordreliste eller kunne redigere min ordre (mål, størrelse osv.).

Acceptkriterier: Givet at kunden har oprettet en ordre, så skal kunden også have muligheden for at fjerne/redigere ordren igen. Dette skal kun være muligt så længe ordren har den indledende status. Når ordrestatussen ændres til "behandles", så skal det ikke længere være muligt at fjerne/redigere sin ordre via hjemmesiden. Så skal der derimod henvises til et telefonnummer/mailadresse, man kan kontakte.

Vigtighed: Need to have

Estimat: Medium

US-7:

Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.

Acceptkriterier: Givet at brugeren har rollen administrator, når de er på ordre siden kan de fjerne en ordre fra visningen, så bliver ordren markeret som inaktivt i databasen

Vigtighed: Nice to have

Estimat: Small

User Stories - 3D-Print-Applikation

US-1:

Som bruger af programmet skal jeg kunne manuelt indtaste dimensioner for den givne carport, og herefter få et slutprodukt jeg kan 3D-printe.

Acceptkriterier: Jeg skal igennem min database eller ordreliste fra webapplikationen se en ordre og dens mål. Det skal kunne skrives i nogle færdige funktioner, hvor den herefter spytter mig en 3D-model ud, som er skaleret ned.

Vigtighed: Need to have

Estimat: Large

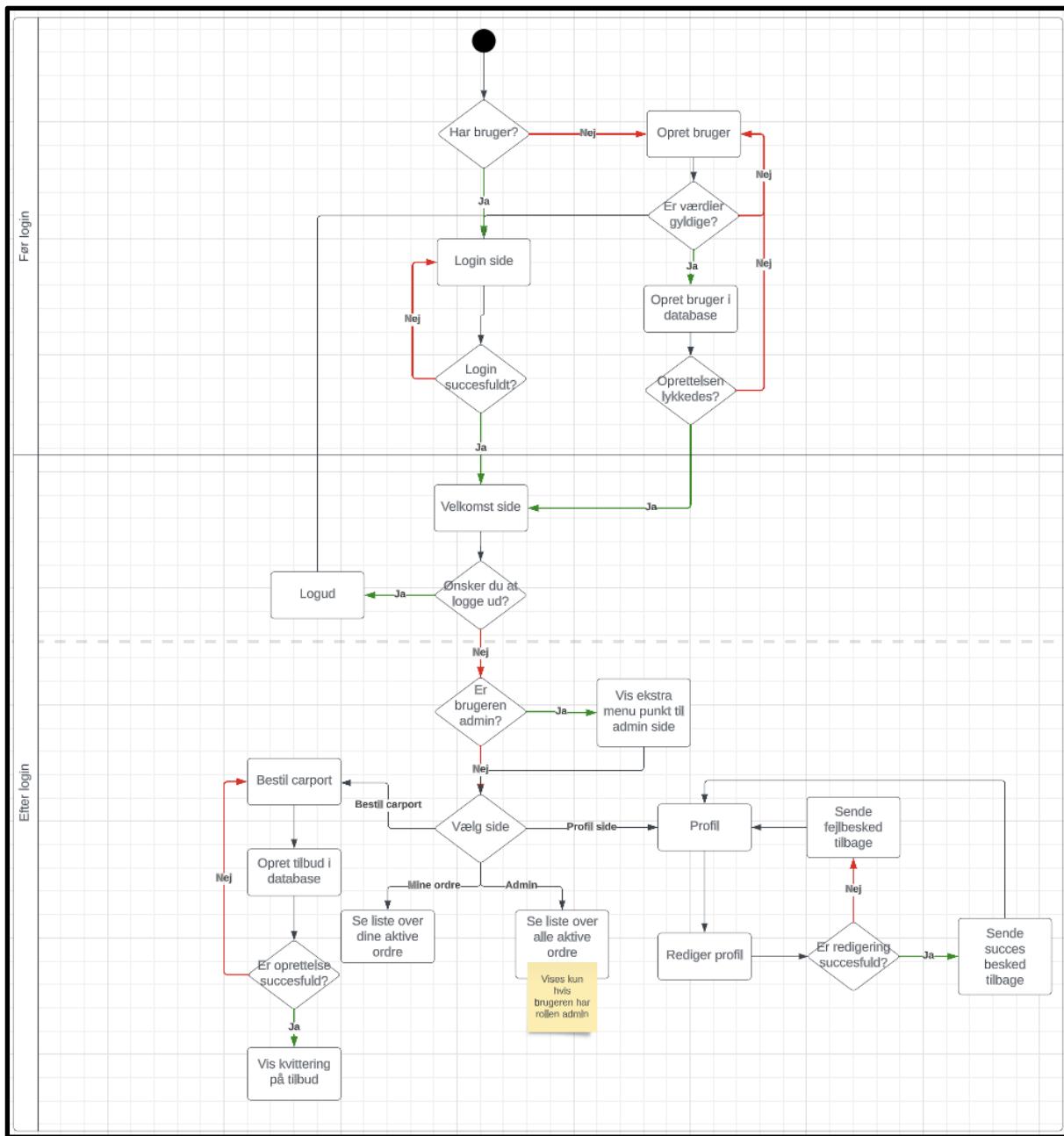
US-2:

Som bruger af programmet skal jeg kunne smide et ordrenummer i en funktion, som herefter henter alle mål og dimensioner fra databasen til det givne ordrenummer. Den skal så herefter lave 3D-modellen til carporten.

Acceptkriterier: Der skal manuelt indtastes et ordrenummer i applikationen. Ud fra ordrenummeret skal alle dimensioner og mål hentes fra databasen og laves til

Aktivitetsdiagram

Aktivitetsdiagrammet beskriver brugerens vej igennem webapplikationen. Diagrammet er opdelt i to primære dele, 'før login' og 'efter login'.



Figur 5

Før login

Webapplikationen starter når brugeren tilgår hjemmesiden, her bliver brugeren præsenteret med velkomstsiden. Her kan brugeren enten oprette en ny bruger eller logge sig ind på en eksisterende bruger. Vælger brugeren at oprette sig, bliver der oprettet en ny bruger i databasen, inden brugeren bliver ført videre.

Efter login

Når brugeren er logget ind, bliver de mødt af samme velkomstside som inden de havde logget ind, bare med flere navigationsmuligheder via navbaren. Navigationsmulighederne varierer baseret på brugerens rolle, og er følgende:

- **Bestil carport:**
 - Denne side er tilgængelig for alle brugere.
 - Her kan man som bruger bestille et tilbud på en carport, med en given højde, bredde og dybde.
- **Mine ordre:**
 - Denne side er tilgængelig for alle brugere.
 - Her kan brugeren se sine aktive ordrer, og redigere/fjerne ordre, så længe status er 'Bestilt'.
- **Profil:**
 - Denne side er tilgængelig for alle brugere.
 - Her kan brugeren se og redigere i sine kontaktoplysninger.
- **Admin:**
 - Denne side er kun tilgængelig for brugere med rollen 'Admin'.
 - Her kan en admin se alle aktive ordrer i systemet, og redigere/fjerne/se mere info om den enkelte ordre.
- **Log ud:**
 - Her sendes brugeren tilbage til velkomstsiden og logges ud af systemet.

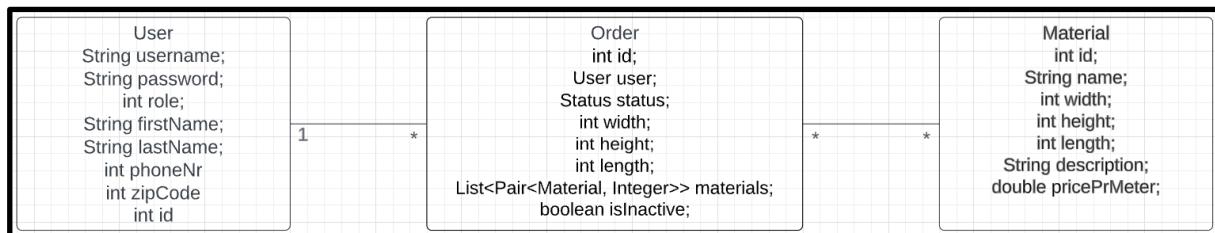
Domænemodel og EER-diagram

Domænemodel

Vores domænemodel består af 3 entiteter, 'User', 'Order' og 'Material'.

Relationen mellem 'User' og 'Order' er en-mange, det er den fordi en User kan have mange ordrer. En ordre kan dog kun have én 'User', da hver ordre er unik.

Relationen mellem 'Order' og 'Material' er beskrevet som en mange-mange relation. Det er fordi en ordre kan have mange forskellige materialer i dens styklister, og fordi en type materiale kan fremgå i mange forskellige ordrer.



Figur 6

EER-diagram

User-tabellen indeholder fire hovedelementer.

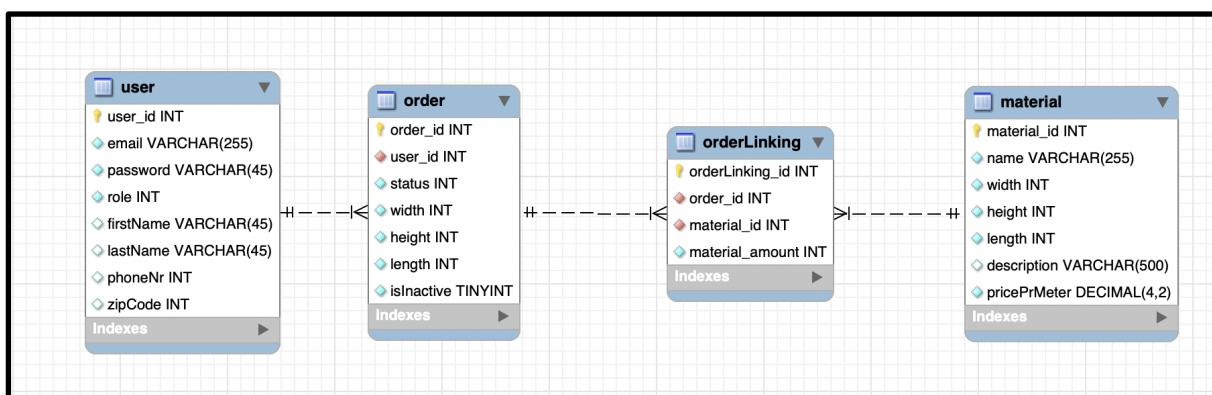
- id, så vi unikt kan identificere alle brugere
- e-mail & password, så brugeren kan lave et login.
- rolle, som er helt essentielt for hele processen med webapplikationen.

Det smarte ved at knytte en rolle til alle brugere er, at man kan få applikationen til at se forskellig ud, alt efter hvilken rolle man bærer på.

På nuværende tidspunkt er der kun to roller, nemlig 0 & 1. Rollerne er blevet deklareret som 'integers'. Har man rollen 0, så er man administrator og dermed adgang til alt, hvad applikationen indebærer. Denne rolle er som udgangspunkt forbeholdt de ansatte hos Johannes Fog. Dernæst er der rollen 1, som er forbeholdt alle andre, nemlig kunderne. Har man rollen 1, så er applikationen målrettet til netop dig som kunde. Som kunde har man naturligvis ikke adgang til nogle administrerende elementer.

Det smarte ved at bruge 'integers' til at deklarere roller er, at det er nemt skalerbart. Hvis der pludselig er behov for endnu en rolle, så får den bare id 2. Det er selvfølgelig overhovedet ikke umuligt at bruge Strings, så rollerne kunne hedde; "admin" & "user". Det var også en overvejelse der blev gjort i gruppen, men der blev generelt set spået færre fejlhåndteringer ved brug af 'integers'.

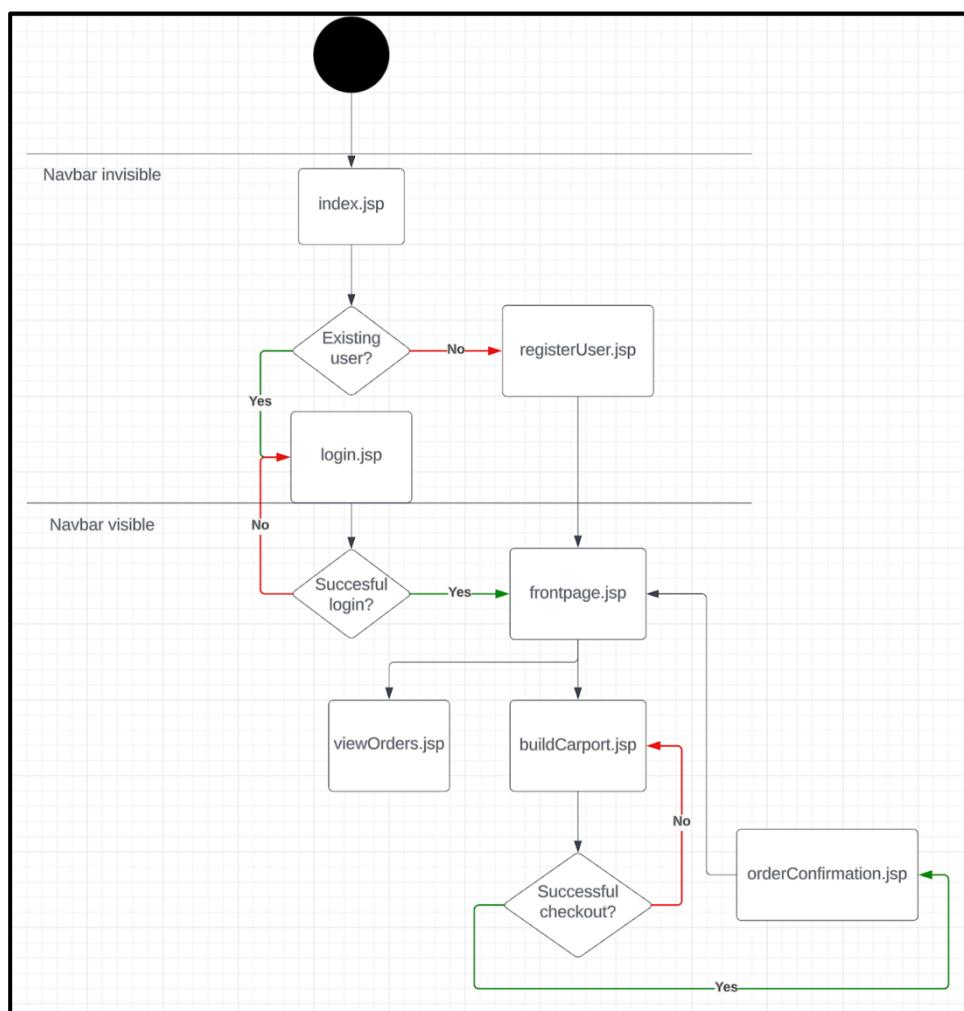
I EER-diagrammet nedenfor kan det ses, at der er benyttet en mange-mange relation i form af en linktabel. 'Order-tabellen' og 'material-tabellen' har et mange-til-mange forhold og kræver derfor en linktabel. I dette tilfælde er der oprettet en 'orderLinking-tabel', som indeholder et ordre-id og et materiale-id samt et antal af de givne materialer. Her er ordre-id'et en fremmednøgle til order-tabellen, ligeledes er materiale-id'et en fremmednøgle til material-tabellen.



Figur 7

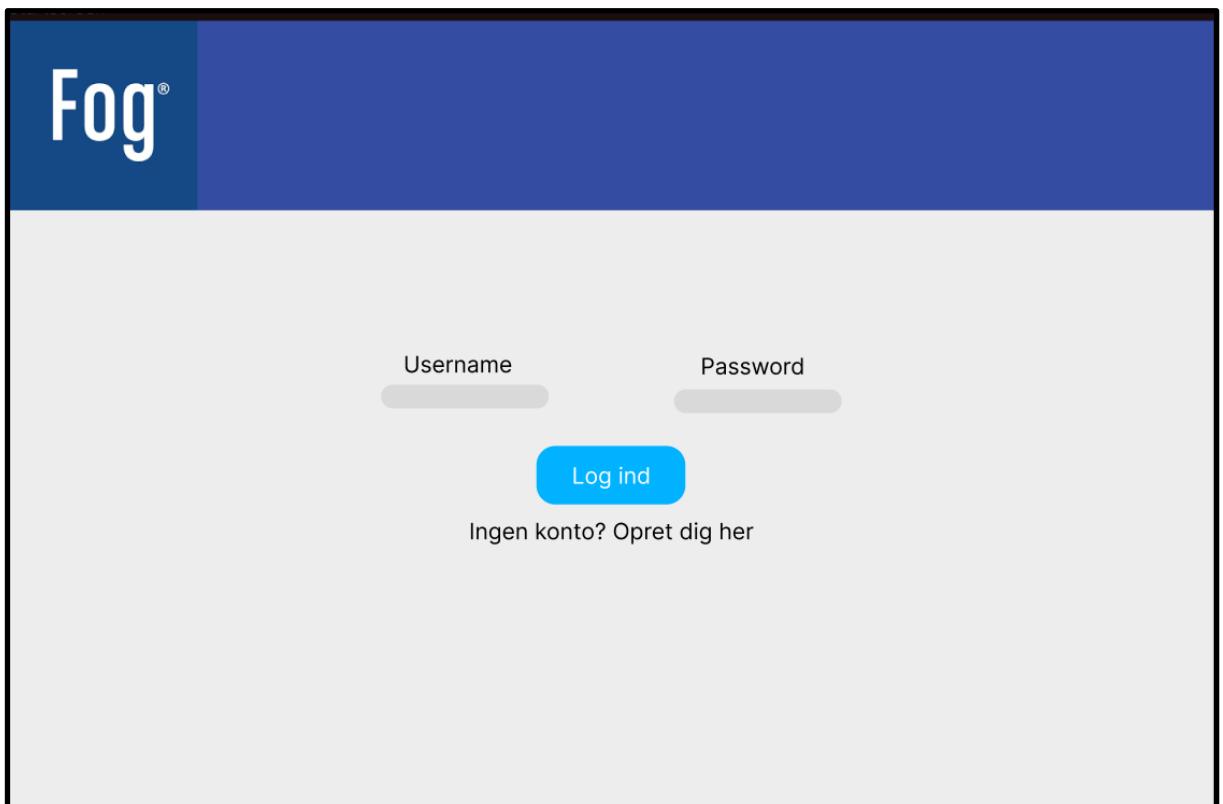
Navigationsdiagram & Mockups

På gruppens navigationsdiagram bliver man ført igennem hvordan websiden er opbygget med jsp-sider. Det er tydeligt at se hvordan man går fra loginknappen og videre til i sidste ende at få sin kvittering, hvilket kan ses på [figur 8](#) som order confirmation. På figuren kan det også ses at der er gjort brug af en navbar som først kan ses efter login. Før det kan man kun se loginknappen i navigationsbaren. Det der ikke kan ses på navigationsdiagrammet er, at selvom det er en fælles navigationsbar, så vil der være en knap mere hvis profilen er admin man så skal have nogle flere rettigheder. Som admin skal man have mulighed for at redigere i ordren sådan at den kan blive sat videre i systemet. Samtidig skal man også kunne fjerne en ordre som admin, hvis den skulle være afsluttet eller kunden har fortrudt.



Figur 8

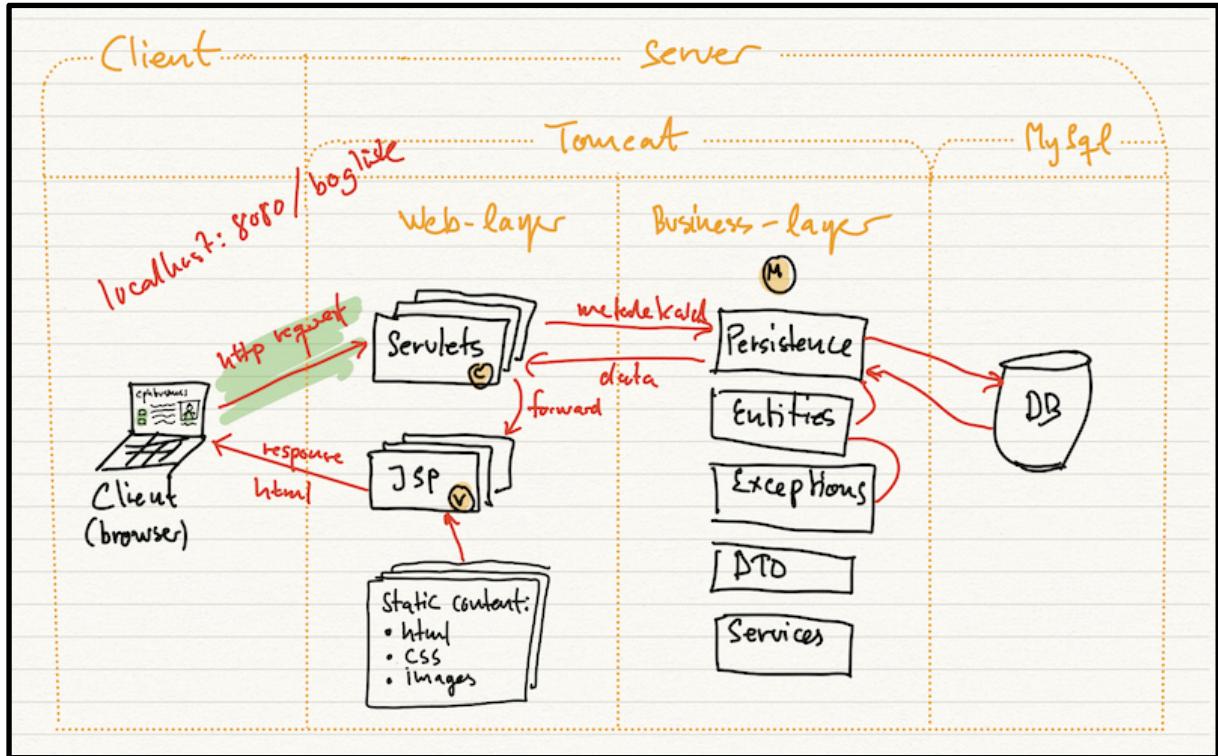
For at skabe et overblik over hvordan slutproduktet skulle ende med at se ud, blev der i starten sat en Figma op. Herunder vises et screenshot fra Figma se figur 9, den er lavet overfladisk da gruppen mere eller mindre bare skulle have noget at gå ud fra, dog ligner den meget godt det produkt der er blevet lavet.



Figur 9

Valg af arkitektur

Vores anvendte arkitektur følger MVC-arkitekturen, som vist på figur 10, hvor vi benytter facade-pattern til at hente data fra vores database.



Figur 10

MVC-arkitektur

MVC-arkitekturen består af 3 grundlæggende dele: Model, View og Controller.

Model er bl.a. entitetsklasserne i vores program som bruges til at modellere ting fra den virkelige verden, eksempelvis en ordre på en carport. Modellerne er der hvor data gemmes, uafhængigt af hvordan denne data skal præsenteres, dette betyder at en model kan bruges i forskellige views, selvom disse skal præsenteres forskelligt.

View er præsentationen af vores modeller, som henter relevant data fra modellerne og viser dem til brugeren. View i vores tilfælde er vores jsp

sider som bruger dataene fra modellerne forskelligt, alt efter hvordan brugergrænsefladen er bygget op.

Controller omsætter brugerens input til handlinger som skal udføres på modellen. Dette kan eksempelvis ses når brugeren bestiller en carport. Her indtaster brugeren de nødvendige oplysninger, som sendes videre, via en http request, til controlleren, der så opretter en ny ordre entitet i systemet. Controllerne i vores system er vores servlets.

Facade-pattern

Facade-pattern er et design pattern, hvor man forsøger at gemme kompleksiteten af et system bag en samlet facade. Dette gøres for at simplificere brugen af systemet, for eksterne brugere. Vi gør brug af facade-pattern, når vi skal hente data fra og oprette data i databasen. Eksempelvis har vi en OrderFacade-klasse, som er tilgængelige i alle klasser, her samler vi de vigtigste funktioner fra vores OrderMapper-klasse, som er den, der reelt set henter og opretter data i databasen. OrderMapper-klassen er package private, og derfor ikke tilgængelig i andre dele af programmet.

Dependency-injection

Dependency injection er et design pattern, hvor funktioner og objekter modtager andre funktioner og objekter som de er afhængige af. Dette gøres for at skille funktioner og objekters ansvarsområder fra hinanden, og hermed skabe et løskoblet program.

Eksempelvis sendes en connection pool med som argument i alle funktioner i vores facader. På denne måde skabes der en løs kobling mellem oprettelsen af forbindelse til databasen (connectionpoolens ansvarsområde), og databehandlingen (mappere og facaders ansvarsområde).

Særlige forhold

Session

Når en bruger logger på, instantieres et User objekt, som der gemmes en reference til i session scopet. Denne reference bruges blandt andet til at vise brugerens e-mail i vores navigation bar og hente de relevante ordrer til vores User objekt under 'View Orders'.

Exceptions

For at håndtere exceptions har vi primært gjort brug af try-catch blokke og vores DatabaseException klasse. I nogle tilfælde smider vi brugeren til vores error jsp side, hvis man lander i vores exception blok.

Brugerinputvalidering

Vores brugerinputvalidering har vi lavet primært ved at fejlhåndtere på vores jsp-sider. Det har vi gjort ved at bruge attributter som 'required' og under 'type' valgt den attribut der passer til scenariet. Det kan f.eks være 'email', som sørger for at der skal være de elementer der er i en email, eksempelvis et '@'.

Derudover bruger vi også vores servlets til at validere inputtet. Det kan eksempelvis være under CreateUserServlet, hvor vi tjekker at inputtet i 'Password' og 'Confirm Password' stemmer overens.

3D-Print

Applikationen for 3D-print af carporten er udviklet separat for web-applikationen. Man åbner altså programmet separat og bliver mødt af et tekstbaseret program, som allерførst beder useren om målene på den ønskede carport til print. Efter man har indtastet sine mål, så får man øjeblikkeligt skænket en komplet guide fra start til slut af, hvordan man går fra at indtaste målene, til at stå med en færdigsamlet miniaturemodel af sin carport i hånden.

Denne guide varierer naturligvis afhængigt af hvilke mål brugerens har valgt. Applikationen er udarbejdet i frameworkt JavaCSG¹.

Programmet består af seks klasser og én abstrakt klasse. Den abstrakte klasse hedder Material.java og indeholder gettere og setttere til materialernes mål. Derefter er der én klasse til hvert materiale, altså Purlin.java, Rafter.java, Post.java. Disse arver alle sammen fra Material.java. Deres former bliver så lavet med variabler, der refererer til materialets egen længde, bredde og højde, således hele programmet er dynamisk. Materialernes højde og bredde er naturligvis forudbestemt, men længderne varierer afhængigt af brugerens mål. Når brugerens har indtastet sine mål på carporten, så bliver der oprettet et objekt af hvert materiale med de givne længder. Dette foregår i klassen 'Test.java'.

Sidst bliver Guide.java-klassen kaldt, som blot printer en komplet guide til brugerens i konsollen. Denne guide fortæller bl.a. nøjagtig hvor mange af hvert materiale man skal printe baseret på brugerens indtastede mål.

Styklisteberegner

Vores stykliste beregner er lavet sådan at vi henter højde, bredde og længde fra den relevante ordrer. Vi opretter 3 Pair variabler af typen, Material og Interger. Den første er 'posts' som tager parametrene 'connectionPool', 'length' og 'height'. Så er der 'rafters' med parametrene 'connectionPool', 'length' og 'width'. Og så har vi 'purlins' med parametrene 'connectionPool' og 'length'. De bliver alle sammen smidt ind i en ArrayList af typen Pair som hedder 'listOfPairs'.

De 3 variabler bliver lavet ud fra 3 forskellige 'get' metoder, 'getPost', 'getPosts' og 'getRafters'.

Hvis vi tager 'getPosts' som et eksempel, [figur 11](#). Så laver vi en int variabel ('amountOfPosts') som er lavet til at beregne mængden af posts ud vi skal bruge, og en Material instans ('post') som finder hvilken størrelse materiale der skal bruges. Til sidst gemmer vi vores

¹ Intro to framework created by Tobias Grundtvig @tgrundtvig

‘post’ og ‘amountOfPosts’ i et Pair, som vi så tilføjer til vores ArrayList ‘listOfPairs’ som nævnt tidligere.

```

50 @ ...
51     //Calculating length and amount of posts need
52     public static Pair<Material, Integer> getPosts(ConnectionPool connectionPool, int length, int height) throws DatabaseException {
53         //Maximum length between two posts is 310 cm.
54         int amountOfPosts = ((int) Math.ceil(((double) length) / 310) + 1) * 2;
55         Material post = MaterialFacade.getMaterialByNameAndLength(name: "Stolpe", height, connectionPool);
56         return new Pair<>(post, amountOfPosts);
57     }

```

Figur 11

Udvalgte kodeeksempler

På [figur 12](#) nedenfor ses den funktion, som udregner styklisterne til hver ordre. Funktionen tager naturligvis en ordre og en ‘connection pool’ som parametre. På linjer 18-20 bliver ordrens længde, højde og bredde gemt i tilsvarende variabler, så det er lettere at arbejde med. Dernæst opretter vi en variabel til hvert materiale med datatypen Pair. Det er oplagt at bruge pairs her, da vi kan gemme to værdier sammen. I dette tilfælde er vi interesseret i at gemme materialetypen og antallet af det vilkårige materiale.

```

17 @ ...
18     public static void calcMaterialList(Order order, ConnectionPool connectionPool) throws DatabaseException {
19         int length = order.getLength();
20         int height = order.getHeight();
21         int width = order.getWidth();
22
23         Pair<Material, Integer> posts = getPosts(connectionPool, length, height);
24         Pair<Material, Integer> rafters = getRafters(connectionPool, length, width);
25         Pair<Material, Integer> purlins = getPurlins(connectionPool, length);
26
27         List<Pair<Material, Integer>> listOfPairs = new ArrayList<>();
28         listOfPairs.add(posts);
29         listOfPairs.add(rafters);
30         listOfPairs.add(purlins);
31         MaterialFacade.createOrderLink(order, listOfPairs, connectionPool);
32     }

```

Figur 12

I de variabler vi opretter på linje 22-24, der får vi dataene fra tre forskellige funktioner længere nede i klassen, se [figur 13](#). Vi har eksempelvis en funktion, der hedder ‘getPurlins’ som tager ordrens længde som argument, og returnerer så et ‘pair’ som dette: ‘Pair<Material, Integer>’ materialet for dette ‘pair’ vil så være ‘purlin’ og antallet vil så være 2.

```

33     //Calculating length of purlins
34     @    public static Pair<Material, Integer> getPurlins(ConnectionPool connectionPool, int length) throws DatabaseException {
35         //Since we only ever need 1 purlin on each side, the amount is always 2.
36         //We keep the variable however, to make it easier to refactor later, if needed.
37         int amountOfPurlins = 2;
38         Material purlin = MaterialFacade.getMaterialByNameAndLength( name: "Rem", length, connectionPool);
39         return new Pair<>(purlin, amountOfPurlins);
40     }

```

Figur 13

På [figur 14](#) ses en funktion som egentlig gør det samme som den ovenstående på [figur 13](#), denne funktion har blot mere fyld på og er lige en tand mere kompliceret. Før skulle vi udregne hvor mange, og hvilke remme der skulle bruges til kundes vilkårlige carport. Det var relativt simpelt, da antallet af remme altid var 2.

På linje 51 på [figur 14](#), ses den tilsvarende funktion, men for stolperne. Her skal funktionen altså både have ordrens længde og højde med som argument. Højden bruges til at finde den rigtige størrelse stolpe. Længden bruges til at beregne antallet af stolper.

På linje 53 på [figur 14](#), ses den matematiske del af funktionen. Her dividerer vi længden af orden med 310, da den maksimale længde mellem to stolper er 310 cm. Vi bruger ‘Math.ceil’ til at runde tallet op, og dernæst lægges der 1 til resultatet. Der lægges 1 til resultatet, fordi der altid skal være en stolpe i hjørnerne. Er din carport f.eks. 350 cm lang, så laver funktionen regnestykket $(350 / 310) = 1,129$, her vil vores ‘Math.ceil’ så runde tallet op til 2, men da carporten er 350 cm, og hver stolper maksimalt må have 310 cm imellem sig, så skal vi altså bruge en ekstra. Derfor lægges der 1 til. Den udregning regner kun stolperne til den ene side, og derfor bliver resultatet afslutningsvist ganget med 2.

```

50     //Calculating length and amount of posts need
51     @    public static Pair<Material, Integer> getPosts(ConnectionPool connectionPool, int length, int height) throws DatabaseException {
52         //Maximum length between two posts is 310 cm.
53         int amountOfPosts = ((int) Math.ceil(((double) length) / 310) + 1) * 2;
54         Material post = MaterialFacade.getMaterialByNameAndLength( name: "Stolpe", height, connectionPool);
55         return new Pair<>(post, amountOfPosts);
56     }

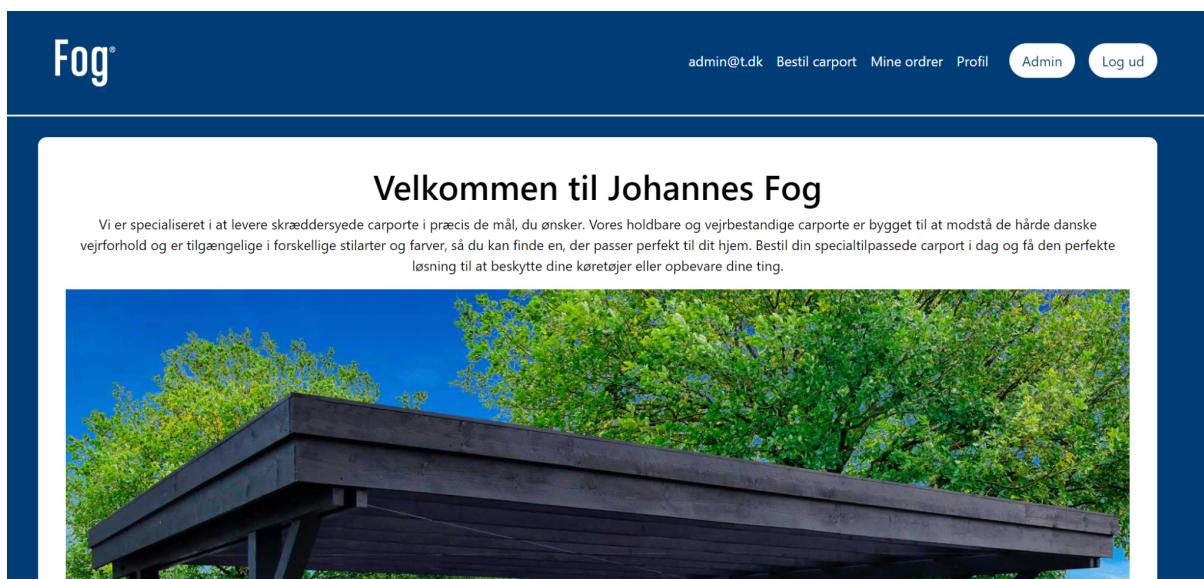
```

Figur 14

Status på implementering

Gruppen er meget tilfredse med det produkt som der er blevet lavet, siden den virker som planlagt. De user stories som der er blevet sat op fra starten er blevet fulgt, så der ville komme et brugbart produkt til køberen.

Der er blevet tilføjet ens styling til alle jsp sider for at få et simpelt og samtidigt stilrent look, som ville være forventet af Fog. Vi har valgt at bruge Fogs blå farve som den mest dominerende på vores side og så bruge en neutral hvid farve i midten af vores sider. Det giver en side som også minder meget om Fogs eget logo med den primære blå farve og så en snert af hvidt på selve bogstaverne. Der er også blevet highlightet nogle vigtige knapper som det kan ses på [figur 15](#). De er blevet highlightet med hvid rundt om. Det er gjort fordi det skal gøre så intuitivt som muligt. Adminknappen skal skille sig ud fra resten da det hovedsageligt er det, som skiller adminsiden fra brugersiden. Log ind-knappen er farvet, fordi det er et af de første steder, vi vil have brugerens øjne hen, når de besøger siden. Når brugeren så er logget ind, bliver log ind-knappen byttet ud med en log ud-knap. Her er det så naturligt at bibeholde den eksisterende ‘style’.



Figur 15

CRUD

CRUD bliver brugt i store dele af opgaven når der bliver interageret med databasen. *Create* bliver brugt når der skal oprettes nye brugere i vores database. *Update* bliver brugt til at opdatere vores brugere og til at man kan opdatere hvor langt i processen den givne ordre er. *Update* bruges også hvis man vil opdatere sine oplysninger på profilen eller redigere den ordre som man allerede har sat i gang før den ryger videre i systemet. Til sidst bliver *delete* også brugt, men den bliver ikke slettet helt nede i databasen. Fordi man som admin skal kunne slette orden fra siden, men vi vil stadig gerne beholde dataene. Det er altid smart at beholde data som disse, da det kan bruges senere hvis det skulle være nødvendigt.

Fejl/mangler på målstregen

Havde vi haft mere tid, så ville vi have implementeret således, at administratorer kan vælge meterprisen for alle materialer. Dette ville være en naturlig ting for et firma at have adgang til. Hvis vi havde fået at implementere det, så havde vi nok lagt det som en underside på adminsiden, så der kommer en liste frem med alle de tilgængelige materialer og deres nuværende pris. Her skulle man så kunne trykke 'opdatér' og på den måde skulle styklisteberegneren så dynamisk ændre sig efter de aktuelle priser på materialerne.

Test

Testet klasser	Testet metoder	Dækningsgrad af test
MaterialListAlgorithm	1. void calcMaterialList() 2. void getPurlins() 3. void getRafters() 4. void getPosts()	100%
UserMapper	1. void login() 2. void invalidPasswordLogin() 3. void invalidUserNameLogin() 4. void createUser()	25%

Figur 16

I begge test klasser starter vi med at oprette forbindelse til en testdatabase. Den er lavet sådan at hvis man ikke har testdatabasen endnu, så bliver den oprettet en identisk til den rigtige igennem SQL queries (Linje 42-47, på [figur 17](#)).

Vi bruger anmærkning @BeforeAll, så vi er sikre på at der altid er oprettet forbindelse til databasen, inden vi gør andet.

```

24     private final static String USER = "dev";
25     private final static String PASSWORD = "*****";
26     private final static String URL = "jdbc:mysql://206.81.31.86:3306/seminsteropgave_test";
27
28     private static ConnectionPool connectionPool;
29
30     private Order order;
31
32     @BeforeAll
33     public static void setUpClass()
34     {
35         connectionPool = new ConnectionPool(USER, PASSWORD, URL, test: true);
36
37         try (Connection testConnection = connectionPool.getConnection())
38         {
39             try (Statement stmt = testConnection.createStatement())
40             {
41                 // Create test database - if not exist
42                 stmt.execute( sql: "CREATE DATABASE IF NOT EXISTS seminsteropgave_test;" );
43
44                 stmt.execute( sql: "CREATE TABLE IF NOT EXISTS seminsteropgave_test.user LIKE seminsteropgave.user" );
45                 stmt.execute( sql: "CREATE TABLE IF NOT EXISTS seminsteropgave_test.order LIKE seminsteropgave.order" );
46                 stmt.execute( sql: "CREATE TABLE IF NOT EXISTS seminsteropgave_test.orderLinking LIKE seminsteropgave.orderLinking" );
47                 stmt.execute( sql: "CREATE TABLE IF NOT EXISTS seminsteropgave_test.material LIKE seminsteropgave.material" );
48             }
49         }
50         catch (SQLException throwables)
51         {
52             System.out.println(throwables.getMessage());
53             fail("Database connection failed");
54         }
55     }

```

Figur 17

I begge test-klasser har vi også en setUp funktion (På [figur 18](#) er setUp metoden fra MaterialListAlgorithmTest), som starter med at fjerne alt data fra de relevante tables (Linje 65-68, [figur 18](#)), hvor den herefter tilføjer data der skal bruges til testene (Linje 71-78 [figur 18](#)).

Grunden til vi gør det på den her måde, er fordi så kører vi altid hver test med samme data, det gør altså at der ikke kan komme nogle afvigelser i testene. Vi bruger anmærkningen @BeforeEach, så vi kalder setUp inden hver test, så databasen altid har samme data i testene.

```

57 @BeforeEach
58 void setUp()
59 {
60     try (Connection testConnection = connectionPool.getConnection())
61     {
62         try (Statement stmt = testConnection.createStatement())
63         {
64
65             stmt.execute("delete from semesteropgave_test.user");
66             stmt.execute("delete from semesteropgave_test.orderLinking");
67             stmt.execute("delete from semesteropgave_test.order");
68             stmt.execute("delete from semesteropgave_test.material");
69
70
71             stmt.execute("insert into semesteropgave_test.user (email, password, role) " +
72                 "values ('user','1234',1),('admin','1234',0), ('ben','1234',1)");
73             stmt.execute("INSERT INTO semesteropgave_test.material (name, width, height, length, description, pricePrMeter) " +
74                 "VALUES ('Spar',5,20,360,'',54.95), ('Spar',5,20,390,'',54.95), ('Spar',5,20,420,'',54.95), ('Spar',5,20,450,'',54.95), " +
75                 "('Stolpe',9,9,208,'',39.95), ('Stolpe',9,9,238,'',39.95), ('Stolpe',9,9,268,'',39.95), ('Stolpe',9,9,298,'',39.95), " +
76                 "('Stolpe',9,9,369,'',39.95), ('Rem',5,20,360,'',54.95), ('Rem',5,20,390,'',54.95), ('Rem',5,20,420,'',54.95), ('Rem',5,20,450,'',54.95), " +
77                 "('Rem',5,20,480,'',54.95), ('Rem',5,20,510,'',54.95), ('Rem',5,20,540,'',54.95), ('Rem',5,20,570,'',54.95), ('Rem',5,20,600,'',54.95);");
78             User user = UserFacade.login(username: "user", password: "1234", connectionPool);
79             order = new Order(user, Status.BESTILT, width: 400, height: 290, length: 550);
80         }
81     }
82     catch (SQLException | DatabaseException throwables)
83     {
84         System.out.println(throwables.getMessage());
85         fail("Database connection failed");
86     }
87 }
88

```

Figur 18

I testen på **billede z**, tester vi funktionen `getPurlins()`. Det gør vi ved at oprette 2 Pair, en der hedder `expected` og en der hedder `actual`. Vi giver vores material nogle værdier vi ved, der svarer til at de skal have 2 af dem til styklisten.

Herefter sammenligner vi de 2 Pairs på `Value0` som er materialet og efter `Value1` som er Integers for at tjekke de har samme mængde.

```

102
103     @Test
104     void getPurlins() throws DatabaseException {
105         Pair<Material, Integer> expected = new Pair<>(new Material( name: "Rem", width: 5, height: 20, length: 570, description: null), 2);
106         Pair<Material, Integer> actual = MaterialListAlgorithm.getPurlins(connectionPool, order.getLength());
107         assertEquals(expected.getValue0(), actual.getValue0());
108         assertEquals(expected.getValue1(), actual.getValue1());
109     }
110

```

Figur 19

I testen på [figur 20](#), tester vi funktionen createUser().

Først instantieres en user igennem createUser metoden, som gemmes i variablen 'newUser'.

Herefter instantierer vi endnu en user, til variablen 'logInUser', via loginmetoden (login metoden testes i en anden unit test) og en expectedUser som bliver instantieret med User klassens konstruktør, med de samme værdier som både newUser og logInUser.

Til sidst sammenligner vi vores expectedUser med vores newUser og vores logInUser.

```
100  @Test
101  void createUser() throws DatabaseException
102  {
103      User newUser = UserFacade.createUser(username: "jill", password: "1234", role: 1, connectionPool);
104      User logInUser = UserFacade.login(username: "jill", password: "1234", connectionPool);
105      User expectedUser = new User(username: "jill", password: "1234", role: 1);
106      assertEquals(expectedUser, newUser);
107      assertEquals(expectedUser, logInUser);
108  }
109 }
```

Figur 20

Proces

Arbejdsprocessen faktuelt

Fase 1:

Planlægning af projektet og udarbejdelse af en Figma-prototype.

Lave user stories til projektet og oprettelse af diagrammer.

Møde med vejlederen for at præsentere og få feedback på de tænkte user stories og diagrammer.

Fase 2:

Oprettelse af et GitHub-projekt som KanBan board til projektstyring.

Skrivning af alle user stories og tildeling af prioritering og omfang.

Implementering af user stories 1 (bestil carport), 2 (login/opret), 3 (profilside) og 4 (user/admin e-mail i navbaren).

Til vores andet møde med vejleder/kunde havde vi forberedt et spørgsmål om at implementere en ordrestatus funktion, så kunden kunne følge med i, hvor

langt deres ordrer var. Kunde/vejleder synes godt om ideen, da det ville skabe bedre struktur for firmaet, og ville skabe større kundetilfredshed.

Fase 3:

Implementering af user stories 5 (Admin alle ordrer), 6 (rediger/fjern ordrer) og 7 (Admin fjern ordrer).

Implementering af en ordrestatus funktion for kunder.

Efter implementering af alle user stories, mødtes vi med kunden, for at vise vores ide til et færdigt produkt. Kunden var rigtig tilfreds med det produkt vi fremlagde. derefter havde vi en dialog, med vejleder om, hvordan vi skulle bygge vores 3D-print applikation op, om det skulle være med inde over webapplikation eller det skulle være separat fra vores webapplikation.

Fase 4:

Oprettelse af applikation til at 3D-printe carport.

Styling af alle sider.

Start på rapportskrivning.

Fejlrettelser og bugfixing.

Vi havde alle i gruppen et fælles ansvar for at der var styr på vores Github Projects. Vi oprettet opgaver og lagde dem under “Backlog”, og hvis man skrev sig på en af opgaverne, havde man et ansvar om at konvertere det til et issue og rykke det i forhold til hvor langt man var nåede, det gjorde at KanBan boardet altid var opdateret. Derudover mødtes vi dagligt i et Discord-opkald, hvor vi bl.a. oprettede en kanal som hed “Opgaver”. Her kunne man uddeletere opgaver som andre kunne gå i gang med, så alle hele tiden var i gang med noget, hvilket skabte en god struktur.

Til vores første vejledningsmøde, havde vi forberedt spørgsmål til vores 7 user stories og diagrammer. Dialogen med vores vejleder hjalp os med rettelser af diagrammer, og hvis vi var i tvivl, hvilke user stories der var “Need

to have” eller “Nice to have” f.eks. var vi tvivl om det var “Need to have” at have en profilside, hvor vi efter dialog med vejleder fandt frem til at det kunne være en fed feature at have med på vores applikation, men at den ikke var højt prioriteret. Alt i alt gik vejledningsmødet rigtig godt, og hjalp os med at forme projektet, så vi kunne komme ud af starthullerne og gå i gang med projektet.

Arbejdsprocessen reflekteret

Da vi er et lille team og der er plads til vi alle kan snakke på tværs og ytre diverse holdninger og ideer, så har vi ikke en direkte projektledelse. Det var meget et sammenspil om hvad der skulle laves inden for den givne deadline, altså hvad der var “must haves” kontra “nice to haves”. Der er selvfølgelig også altid nogle som har bedre overblik og er dygtige til at uddeletere opgaver, det benyttede vi os også af, men det var ikke det dominerende i gruppen.

Vores evalueringsmøder havde vi dagligt om morgenens. Her kunne vi snakke på tværs af ideer og opgaver, der var opstået løbende. Her udnytter vi også meget at vise frem hvad man har lavet siden den forrige dag, så man kan få et par friske øjne og en ny tankegang til at komme med konstruktiv kritik. Så det mest væsentlige i møderne var at evaluere hinandens arbejde.

Alle vores user stories blev lagt i vores KanBan som en helhed. Altså brudt vi den ikke ned i mindre dele, men én user story var én opgave.

Det havde både sine ulemper og fordele. Ulempen var, at nogle user stories ikke kunne laves før den forrige user story var lavet. Så man kunne risikere at komme lidt i et vente stadie. Heldigvis var der altid noget at lave, i form af logbog, optimering, forberedelse osv.

Omvendt fungerede det godt da det var en “større” opgave man fik under armen og det gjorde det lettere at komme i dybden med den ene user story.

Vores estimeringer i forhold til hvad der kunne nås og hvordan vi lavede et godt produkt, har været gode. Det har været grundet vores første fase/uge.

Her har vi lavet rigtig meget forberedende arbejde, diagrammer og sat prioriteringer på user stories. Derudover har vi også mødtes med en vejleder for at hjælpe os med at give os feedback som ”kunde”. Det har hjulpet os med at vi hverken har overskudt eller underskudt projektet.

Vi fandt hurtigt en rytme, der fungerede for os og som skabte en høj produktivitet. Vi har arbejdet sammen før som gruppe og har derfor fået en håndfuld erfaring på hvad der fungerer og hvad der ikke fungerer. De eneste udfordringer der kunne hindre vores produktivitet, var mangel på opgaver, både store og små.

Github i form af pull request, issues og projects var en udfordring i starten. Det var det fordi det var første gang vi delte kode på den måde. Men efter der var kommet styr på de processer som nok tog 2-3 kode dage, så fungerede det bedre end hvad vi plejede at arbejde med (Trello og ingen pull request). Det skaber nogle trygge rammer i forhold til der ikke sker fejl på master branchen.

Bilag

Figur 1	4
SWOT-analyse	
Figur 2	6
Interessentanalyse	
Figur 3	8
AS-IS-diagram	
Figur 4	9
TO-BE-diagram	
Figur 5	13
Aktivitetsdiagram	
Figur 6	15
Domænemodel	
Figur 7	16
EER-diagram	
Figur 8	17
Navigationsdiagram	
Figur 9	18
Figma-mockup	
Figur 10	19
MVC-arkitekturen	
Figur 11	23
Udvalgt kodesektion	
Figur 12	23
Udvalgt kodesektion	
Figur 13	24
Udvalgt kodesektion	
Figur 14	24
Udvalgt kodesektion	
Figur 15	25
Screenshot af den færdige webapplikation	
Figur 16	27
Overblik af test	
Figur 17	28
Udvalgt kodesektion	
Figur 18	29
Udvalgt kodesektion	
Figur 19	29
Udvalgt kodesektion	
Figur 20	30
Udvalgt kodesektion	