

FOG CARPORTE

Datamatiker 2. semester

Præsentation af produktet:

Hjemmeside: <http://167.71.57.146:8080/FogSemesterProjekt-1.0-SNAPSHOT/fc/index>

GitHub: <https://github.com/AlexKenfelt/FogSemesterProjekt>

Taiga: <https://tree.taiga.io/project/alexsk2020-semesterprojekt-fog-1/kanban>



Navn	Mail	Github
Alex Simone Kenfelt	cph-ak428@cphbusiness.dk	https://github.com/AlexKenfelt
Julius Krüger Madsen	cph-jm352@cphbusiness.dk	https://github.com/PsychedelicCoder
Maja Nøhr-Nørgaard	cph-mn605@cphbusiness.dk	https://github.com/cph-mn605
Projekt udarbejdet 08-05-2021		Rapport udarbejdet 28-05-2021

Indholdsfortegnelse

Introduktion	1
Indledning	1
Baggrund	1
Teknologivalg	2
Analyse	3
Krav	3
User Stories	3
Arbejdsgange der skal it-støttes	4
Aktivitetdiagram	4
Implementering	5
Use Case	5
Domænemodellen	6
ER diagram	7
Navigationsdiagram	9
Mockups	11
Valg af arkitektur	12
Særlige forhold	14
Udvalgte kodeeksempler	17
Test	21
Proces	22
Arbejdsprocessen faktuel og reflekteret	23
Konklusion	26
Bilag	27

Introduktion

Indledning

Vi har som en del af datamatikerstudiet 2. semester på Cphbusiness fået tildelt opgaven; at udvikle en hjemmeside for Johannes Fog A/S. Hjemmesiden skal laves med formålet at carportene kan sælges med brugerdefinerede mål.

Vores gruppe startede med at lave mockups, som har sat rammen for vores hjemmeside. Gruppen har udviklet projektet sammen, herunder kode og rapporten, som vi alle tager ligeligt ansvar for.

Hjemmesiden bliver kodet ved hjælp af HTML, CSS, Bootstrap, Java og SQL. Hjemmesiden bliver kørt gennem Tomcat webcontaineren, som eksekverer programmet. Vi har arbejdet ud fra Scrum metoden, for at holde et godt overblik og god struktur over arbejdet.

Hjemmesiden er dynamisk, da der gemmes og hentes data fra en database, som bliver normaliseret på 3. normalform.

Vi har fået stillet til opgave at udarbejde en nyere version af Johannes Fogs IT-system. Der er et ønske om at få opdateret og revideret systemet til salg af deres carporte. Systemet skal kunne tilgås af både kunde, samt personale for at kunne tilgå de administrative opgaver, såsom godkendelse af kundens ordre. Der skal udarbejdes en website som gør det muligt for en kunde at oprette og skræddersy en ønsket carport med ønskede mål. Carporten skal kunne modtage specifikke mål i ønskede længder og bredder.

It-systemet skal kunne modtage den lagte ordre fra kunden, hvor der skal beregnes en stykliste med materialer som skal bruges til carporten. Udover at udregne en stykliste, skal systemet også kunne udarbejde tegninger af carporten med de ønskede mål fra kunden.

Baggrund

Johannes Fog er stiftet i 1920 og var en enkeltmandsejet virksomhed, frem til grundlæggeren død i 1970. I dag er Fog 100% ejet af Tømmerhandlen Johannes Fogs Fond. Johannes Fogs Fond er stiftet af Johannes Fogs datter Ingrid Bang og ægtefællen Ole Bang. Formålet med fonden er gennem

legater, at støtte almennyttige, almenvelgørende formål, forbedring af miljøet og samfundsudviklingen. Johannes Fogs Fonden er også en erhvervsdrivende fond.

Johannes Fog består af Bolig & Designhus websitet, samt har de ni Trælast & Byggecenter-butikker fordelt i hele Nordsjælland. I deres trælast, som der er fokus på i denne opgave, har de træ og byggematerialer til alle opgaver. Der er der et stor udvalg, hvor deres kunder kan vælge mellem mange muligheder, i form af størrelse og kvalitet.

Teknologivalg

Vi har til projektet benyttet os af følgende teknologier:

Værktøjer:

- IntelliJ IDEA 2020.3.2 (Ultimate Edition)
- Bootstrap Studio (Version)
- MySQL Workbench (Version.....)

Teknologier:

- Ubuntu (Version)
- Tomcat 9 Version - 9.0.22
- Java Version 15.0.1
- MySQL (Version 8.0)

Stack:

- HTML5
- CSS
- Bootstrap (Version 5.0)

Analyse

Krav

Hjemmesidens IT-system skal bygges op, så vi kan levere et skræddersyet tilbud af en carport til kunden. IT-systemet skal derfor kunne hjælpe Fog med følgende:

- At øge muligheden for kundekontakt, så kunderne kan få en hurtig og direkte mulighed for bestilling hos Fog.
- At give Fogs ansatte et bedre overblik, over kundens bestilling, så der sikres det bedste salg til kundens ønske af carport.
- At give Fog en større fleksibilitet over materialet og dens pris.

Disse krav er udarbejdet ud fra samtaler med Product Owneren og lavet om til vores User Stories.

User Stories

Nr.	User Story	S	M	L	XL
1	Som kunde kan jeg oprette en konto profil, så jeg kan gemme og bestille min ordre.		M		
2	Som kunde skal jeg kunne vælge målene til min carport og for at få et tilbud på carporten.		M		
3	Som kunde kan jeg få et tilbud på min carport, så jeg kan se den samlede pris.		M		
4	Som kunde kan jeg godkende tilbuddet på min valgte carport så min ordre kan blive sat igang	S			
5	Som kunde skal jeg kunne modtage plantegningerne samt styklisten efter jeg har betalt min ordre.				XL
6	Som administrator så skal jeg kunne logge ind og se ordrehistorikken, samt "pending" tilbud			L	
7	Som administrator skal jeg kunne gå ind og ændre en kundes ordre fra "pending" til "confirmed", (så kunden kan se sin plukliste og plantegning.)			L	
8	Som kunde kan jeg følge med i min ordrestatus så jeg ved hvor langt min ordre er	S			
9	Som sælger, så vil jeg gerne kunne godkende en ordre og evt. få kontakt til kunden			L	
10	Som sælger skal jeg kunne se plantegning og stykliste for bestillingen, som bliver udregnet via it-systemet				XL

Arbejdsgange der skal it-støttes

Aktivitetdiagram

I dette diagram, viser vi hvordan kunden kan bestille en carport hos Fog. Dette vil vi gøre på to måder, ved at dele aktivitetsdiagrammet op. Først vil vi vise Fogs nuværende situation, og derefter vise, hvordan det ønskede resultat kommer til at se ud.

As is:

Kunden skriver til Fog med en forespørgsel og mål til en carport. En sælger fra Fog modtager en mail på forespørgslen og indtaster målene, så systemet kan beregne forespørgslen om til et tilbud. Sælger gennemgår tilbuddet, og kontakter kunden pr. telefon. Kunden vurderer derefter tilbuddet. Hvis kunden takker nej til tilbuddet, må kunden evt. lave en ny forespørgsel til en carport. Hvis kunden takker ja til tilbuddet, bliver ordren skrevet om til den endelige faktura. Fakturaen bliver derefter sendt til kunden, så kunden kan betale for sin ordre. Kundens materialer, bliver pakket og sendt til kunden, så kunden kan bygge sin carport.

To be:

Kunden kommer ind på forsiden af hjemmesiden, her kan kunden vælge at gå videre til at bygge sin carport. Hvis kunden er logget ind, kommer kunde direkte videre til bestillings siden, hvor kunden kan indtaste en forespørgsel på sin bestilling. Hvis kunden ikke er logget ind, bliver kunden først sendt videre til login siden, så kunden derefter kan gå videre og bestille ordren. Når kunden har bestilt ordren, bliver ordren pending. Kunden venter nu på, at sælger går ind og godkender tilbuddet til kunden.

Når sælger kommer ind på forsiden af hjemmesiden, skal sælger login for at tilgå administrator indgangen. Her kan sælger se, alle igangværende tilbud. Sælger vælger et tilbud, tjekke det og kan derefter bekræfte det.

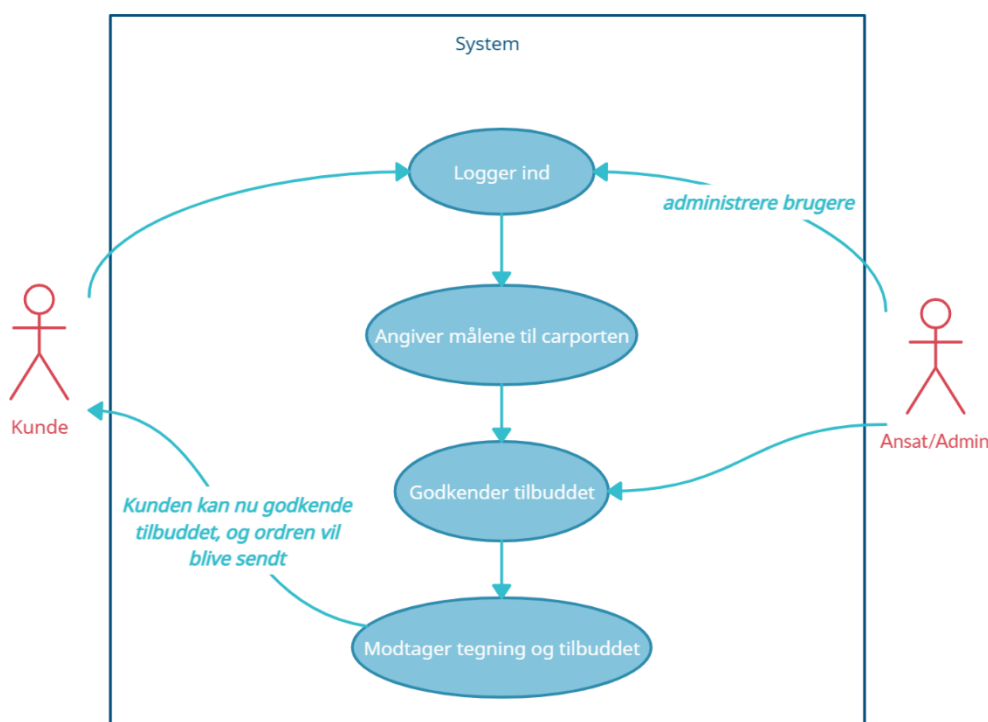
Kunden kan nu se, at sælger har godkendt og færdig beregnet tilbuddet.

Implementering

Use Case

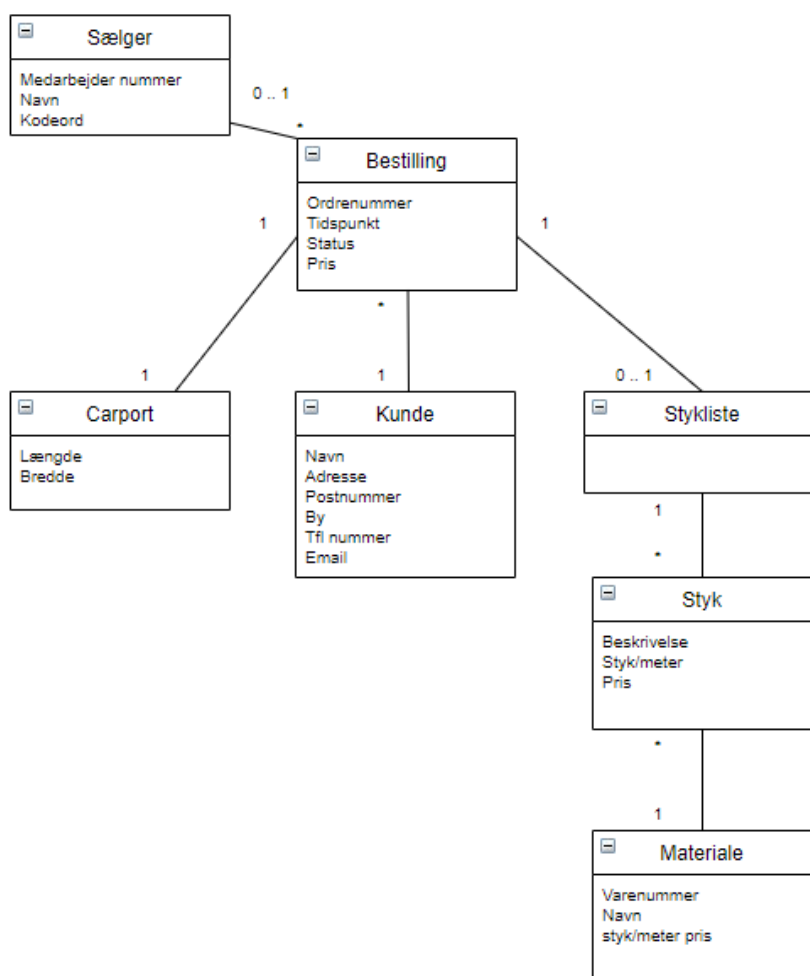
Vi har udarbejdet et Use Case diagram som ligger til grundlag for vores User Stories. Det ses at vi har en *kunde* og en *administrator*, som kan tilgå vores system. Ser vi på kundens side er *kunden* mere koncentreret omkring hjemmesidens front-end. *Administratoren* derimod er mere koncentreret på hjemmesidens back-end.

Hjemmesiden er udarbejdet således, at vi har kunnet få overblik over systemets struktur, eksempelvis skal *administratoren* godkende ordren inden *kunden* modtager styklisten og plantegningerne. Dette gøres efter eget ønske fra Fog, da de ikke ønsker *kunden*, går til konkurrenten.



Domænemodellen

I domænemodellen ses der hvilke datatyper og variabler vi forventer at bruge i vores klasser. Dette gøres ud fra de informationer vi har modtaget fra Fog. Ved brug af denne model får vi et overblik over, hvordan klasserne hænger sammen og hvordan man kan udarbejde et projekt. Modellen skal udarbejdes så den er nem at tilgå for vores kunde, da virksomheden skal kunne arbejde med vores produkt, og derved også skal kunne forstå relationerne i domænemodellen.



I vores domæne model kan man se, at *bestilling* er vores primære fokusområde. Alt på vores hjemmeside er afhængig af, at *kunden* opretter og gennemfører en bestilling. Vi har udarbejdet vores

projekt ud fra, at en *kunde* kan oprette og gennemføre flere ordre på samme tid. Da Fogs *kunder* kan både være privat forbrugere og håndværkere, skal systemet kunne oprette flere ordre pr kunde.

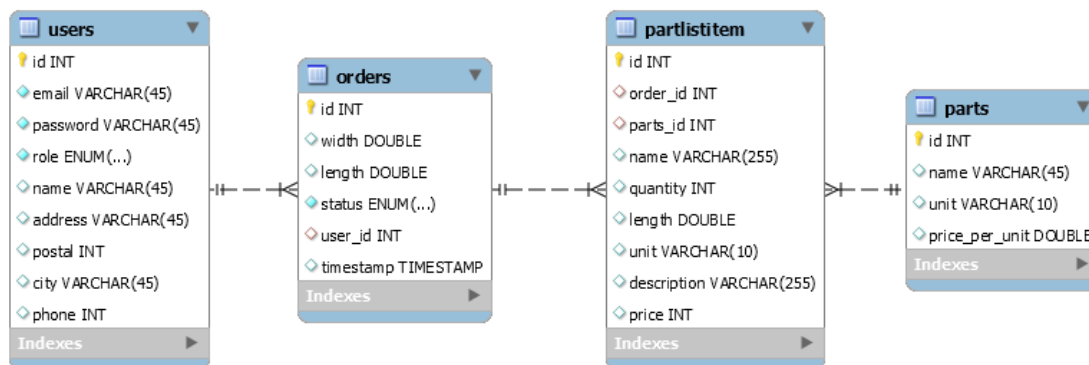
Når Bestillingen bliver oprettet, vil den bestå af en *Sælger*, *Carport*, *Kunde* og *Stykliste*. *Sælgeren* skal godkende bestillingen inden den bliver betalt. Der skal oprettes en ønsket *carport* i de rette mål, som *kunden* angiver. *Kunden* skal være oprettet i systemet med et login og korrekte personoplysninger. *Styklisten* er genereret ud fra kundens ønsker. *Styklisten* bliver unik ud fra den givne ordre, da hvert styk henviser til et specifikt Materiale.

Det ses i vores domænemodel at *sælger* og *stykliste* har en “ingen eller en” relation. Det skyldes, at der ikke behøver at være en relation mellem *sælger* og *bestilling*, samt *stykliste* og *bestilling* før senere i processen. *Sælgeren* skal først godkende bestillingen senere i processen og *styklisten* skal først oprettes, når *bestillingen* er godkendt af Sælger.

Ud fra vores domænemodel har vi kunnet få et overblik, over kundens ønsker til produktet. Vi har også fået et overblik over datatyperne til MySQL.

ER diagram

ER diagrammet er blevet udarbejdet på baggrund af vores domænemodel. Modellen viser hvilke datatyper, der er blevet benyttet og forholdet mellem de forskellige klasser i MySQL databasen. Databasens indhold er bygget op med tabeller og relationer, der næsten overholder 3. normalform. Det vil sige, at databasens kolonner er ikke-transitive afhængige af en primærnøgle. Afvigelsen fra 3. normalform sker i tabellen *users*, da der godt kan være den samme data i navn, adresse, postnummer og by. Afvigelserne giver mening at lave, da det ellers ville gøre tabellerne mere uoverskuelige at tilgå i databasen, hvis der kun er individuelle tabeller for ovennævnte afvigelser.



I alle tabellerne i vores database, er der en primærnøgle 'id', som er blevet sat som en 'auto increment' i tabellerne. Der bliver derfor skabt en unik nøgle i den enkelte tabel. I alle vores tabeller har vi valgt at bruge data typen 'Integer', som er det der bliver brugt til at auto incremente nøglen. Dette tal har vi valgt automatisk skal stige, når der bliver indsat flere rækker i kolonnerne. Vi sikrer vi os derfor, at de primære nøgler aldrig er det samme og altid bliver genereret.

Fremmednøglerne bliver også brugt i et eller flere tabeller. Fremmednøglerne identificerer en række i en anden tabel, den tabel kalder man for børnebordet, som bliver henvist til forældre bordet.

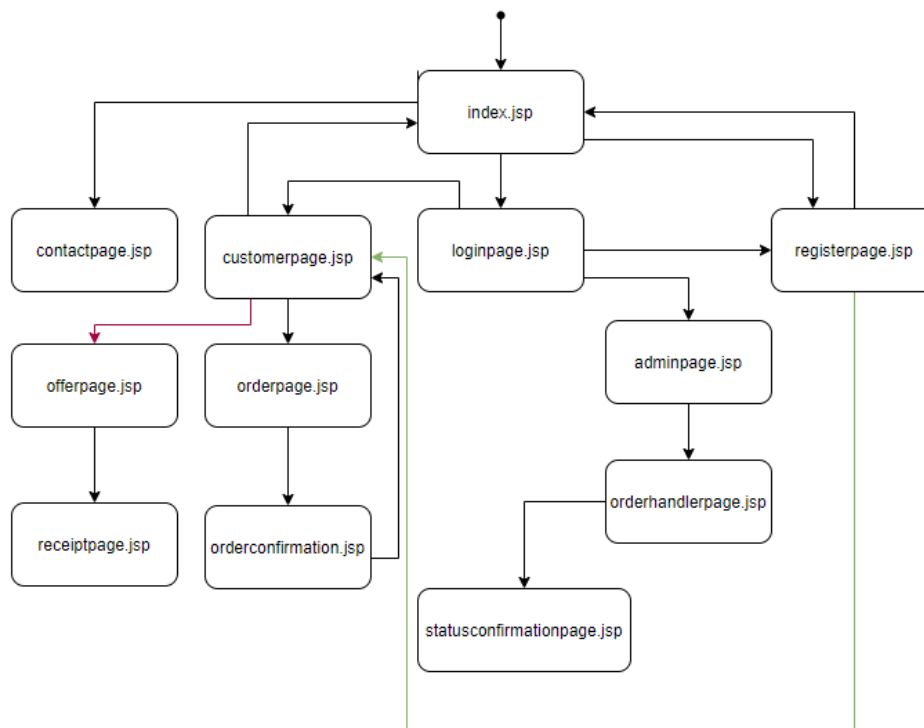
I *users* tabellen indeholder kolonnerne: *id (AI)*, *email*, *password*, *role*, *name*, *address*, *postal*, *city* og *phone*. *Users* tabellen har vi en til mange relation til *orders*.

Orders tabellen indeholder kolonnerne: *id (AI)*, *width*, *length*, *status*, *user_id (FK)* og *timestamp*.

*Partlistitem*s tabellen indeholder kolonnerne: *id(AI)*, *order_id*, *parts_id*, *name*, *quantity*, *length*, *unit*, *description* og *price*.

Parts tabellen indeholder kolonnerne: *id(AI)*, *name*, *unit* og *price_per_unit*.

Navigationsdiagram



I dette afsnit vil vi redegøre for navigationsdiagrammet for vores hjemmeside, dette indebærer navigationen for både kunder og ansatte på hjemmesiden. Vi valgte at have en navigations bar på vores hjemmeside. Navigationen baren er udarbejdet stort set på samme måde for alle brugere af hjemmesiden, men med få undtagelser.



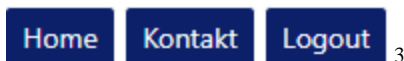
I bilaget (1) ovenover kan man se navigationsbaren for en bruger af hjemmesiden der ikke er logget ind endnu. Her har brugeren adgang til en home, kontakt, login og sign up knap.

¹ Bilag 1



Profil Home Kontakt Logout ²

I andet bilag (2) kan man se navigationsbaren for en kunde der er logget ind. Her har man altid en knap, som går tilbage til brugerens profil og mulighed for at logout. Ellers er der ikke de store forskelle for brugeren, om de er logget ind eller de ikke er.



Home Kontakt Logout ³

Det sidste bilag (3) ses der navigationsbaren for en administrator. Administratoren er i stand til at gå til *home*, *kontakt* og *logout*. Der er også oprettet en knap til *adminPage*, den har vi dog ikke med som en del af vores navigationsbar.

Først tager vi udgangspunkt i administrator navigationen på vores hjemmeside. Det første der sker for alle brugere er, at uanset om man er kunde eller ej, så rammer man *index.jsp*. Når man logger ind, som *admin* vil man blive sendt til *adminpage.jsp*. Som admin vil man kunne se alle '*pending*' ordre, i toppen af *adminpage.jsp*. Derudover vil man også kunne finde en rediger ordre knap. Hvis/Når knappen bliver trykket på, vil man blive sendt hen til vores *orderhandlerpage.jsp*. Under *orderhandlerpage.jsp* får man tre knapper tilgængelig, '*Se Indhold*', '*Fjern*', og '*Confirm order*'. Da '*Se Indhold*' og '*Fjern*' knappen ikke er fuldt ud implementeret. Dette vil blive forklaret dybere i afsnittet "Status for implementeret". Hvis man som admin trykker på '*Confirm order*' knappen ud fra en specifik ordre, vil ordren blive sat til confirmed og admin vil blive redirected hen til vores *statusconfirmationpage.jsp*.

Tager man udgangspunkt i navigationsbaren for kunden på hjemmesiden, vil man starte på *index.jsp* siden. Hvis det er en ny bruger, har vedkommende mulighed for at oprette sig som ny bruger i systemet. Når man har oprettet sig, bliver man sendt videre til *index.jsp* siden.

Når kunden vælger at bestille en ønsket carport, bliver brugeren omdirigeret hen til *orderpage.jsp*. På den side vil brugeren få mulighed for at indtaste længde og bredde for den ønskede carport. Når kunden har trykket '*bestil*', vil kunden blive videresendt til *orderconfirmationpage.jsp*. Her vil kunden modtage en bekræftelse på sin forespørgsel og kunne se sin SVG-tegning af sin carport. Kunden vil nu have mulighed for at se sin ordre status. For at kunden skal kunne se en stykliste, skal

² Bilag 2

³ Bilag 3

kunden vente på en admin har godkendt ordren. Når den er blevet 'confirmed' vil kunden få en knap op som hedder 'se indhold'. Den vil sende brugeren frem til siden '*offerpage.jsp*', hvor kunden vil se det totale beløb, samt en stykliste. Kunden vil nu kunne betale for ordren og vil blive navigeret over til *receiptpage.jsp*.

Mockups

Rammen for hjemmesidens design, er lavet ud fra følgende mockup. Vi har lagt resten i bilag:



Vores mockups lavede vi i startfasen for projektet. Alle involveret i projektet ønskede et enkelt og simpelt design, så brugerfladen for kunden og administratoren er nem at tilgå. Hvilket vi synes vi har overholdt. Vi har taget udgangspunkt i Fogs egen hjemmeside⁴. Vi har blandt andet fundet inspiration fra farverne, logoet og opsætningen.

⁴ <https://www.johannesfog.dk/byggecenter/>

Da vi udarbejdede vores mockup ønskede vi et overordnet billede af hjemmesidens brugerflade, derfor valgte vi ikke at lave specifikke sider til *kunder* og *administrator*. Mockuppen har været med til at give et overblik over vores hjemmeside.

Valg af arkitektur

Vi har udarbejdet vores projekt efter en givet startkode. Arkitekturen for startkoden har været med til at opbygge vores projekts ramme, da der fulgte en del funktionaliteter med startkoden. Der var heriblandt jsp. sider som skabte rammen for vores hjemmesides layout. Der var blandt andet en forside, registerpage og genericpage. Det er blot eksempler på de givne sider, der var implementeret fra startkoden. Login systemet er også med til at forstå logikken bag startkoden og derfor skabe en ramme for kodens arkitektur.

Forståelsen af startkoden kræver noget forarbejde, derfor vil der blive gennemgået de givne *design patterns*. De givne *design patterns* er:

- Model View Controller (MVC pattern)
- Front Controller pattern
- Command pattern
- Singleton pattern
- Facade pattern
- Dependency injection

Model view controller også kaldet *MVS*, er en hjælp til at organiserer ens kode og adskille de forskellige lag fra hinanden. *MVS* er opbygget af tre lag, med henholdsvis model, view og controller.

Ser man på *modellaget* bruges den til at definerer de mest væsentlige data i koden. Heriblandt logik og data. Hvis man tager fat i vores kode, vil det være vores mappers og service klasser som vil skabe rammen for vores logik og data i koden.

I *view* laget består den af alle de funktioner, der direkte interageres med brugeren. Det er især her man har fokus på brugerfladen hos kunden. I koden vil det være i vores html og vores jsp. sider der vil være *view* laget.

Controlleren fungerer som mellemledet mellem model og view. Det er her, der bliver modtaget brugerinput fra brugeren. Den tager fat i en forespørgsel, fra en bruger og sender forespørgslen videre til serveren. Det er her serveren bestemmer, hvad der skal ske med forespørgslen fra kunden.

Derfor hænger det sammen med *model* og *view*. Controlleren modtager data fra *modellaget* som behandler dataen og sender videre til *view* laget. I forbindelse med vores projekt vil det være vores command-klasser som udgør *controllerlaget*.

Ser man samlet på vores kode, hænger koden godt sammen med MVC-konceptet. Vi har hele vores logiske kode (*modellaget*). Vores jsp. sider udgør vores view lag som indebærer at den skal kunne tilgå hjemmesiden.

Frontcontroller pattern

Frontcontrolleren i programmet har en stor indflydelse for at systemet virker, da front controlleren er med til at omdirigerer os rundt i system. Hvilket hænger godt sammen med MVC, da den behandler forespørgsler mellem kunden og serveren.

Command pattern

Når man ser på command pattern, omhandler det et design pattern, der opsummerer en request, som er sendt fra kunden. Den sender et objekt, for at kunne gemme eller eksekvere requesten til senere brug. Denne pattern er en del af startkoden, som blev givet og er en abstrakt klasse som kan *Extendes* fra andre klasser. Ved de kan *Extendes* er pga. de alle har samme command metode, *execute()*. Ser man på underklasserne man kan extende er *CommandProtectedPage* og *CommandUnprotectedPage*. Hvis første underklasse, altså *CommandProtectedPage*, skal extende noget skal man opgive to parametre *pageToShow* og *role*. Det er altså specifikke roller som kan tilgå *CommandProtectedPage*, som i vores kode er "*Admin*". Hvorimod hvis det ønskes at alle skal tilgå hjemmesiden, skal man blot opgive en parameter *pageToShow*.

Singleton pattern

Ved brug af *Singleton* pattern bliver der i en klasse instantieret kun et objekt af den type af den tilhørende klasse. Det vil sige, at der altid kun vil være en instans af et objekt af en klasse. I koden ville man se på eksempelvis styklisten, som man sikrer sig kun kan blive instantieret en gang.

Facade pattern

Når man benytter sig af facade pattern, er det for at gøre koden mere læselig. Det kunne blandt andet være, ved at man gemmer dele af koden væk, som bag i en facade, som også fremgår i navnet, facade. Den bliver oftest brugt når koden er for kompleks at forstå. Når man benytter sig af facade pattern, lukker man metoden sammen, så der til sidst kun er metodenavnet og hvad metoden returnerer.

Dependency injection

Når man benytter dependency injection har man et objekt som modtager andre objekter. I dette tilfælde snakker vi om databaseobjektet, som bliver kaldt i vores data mappere og facadeklasser. Alle vores datamappere og facadeklasser indeholder databaseobjektet. Dette databaseobjekt indeholder JDBC (Java Database Connectivity) som er en connection til databasen. Ved brug af dependency injection giver det mulighed for at lave en kobling til databasen og vores program, da det gør det nemt at tilgå datamapperne og vores service klasser.

Særlige forhold

Der bliver i dette afsnit, gennemgået hjemmesidens særlige forhold. De særlige forhold, gør at vi ved hjælp af eksempelvis, exceptions, kan finde fejl og gøre brugeroplevelsen bedre. Det sikrer os, den bedste kvalitet af hjemmesiden, til vores kunde Fog.

Session

Sessions indeholder vores User objekt, som bliver brugt når der bliver oprettet en ny kunde. Vi gemmer vores bruger data, som enten er en kunde eller sælger, i en session, så databasen ikke bliver

nødt til at blive kaldt hver gang. Det gør også, at sessionen ved hvilke rettigheder, hver bruger har til hjemmesiden. Hver bruger har en rolle, hvor brugerens rettigheder bliver gemt.

Exceptions og logger

Exception har vi brugt, for at give os selv et overblik, over fx if/else sætninger. Ved brug af vores exceptions, kan vi hurtigere finde frem til mulige fejl i koden, og kontrollere de fejlmeddelelser der kommer. I koden har vi især brugt User exception, som er en exception, der bliver lavet i startkoden.

Brugerininput validering

Vi har brugt brugerininput, når vi eksempelvis taster kundens navn ind, vil man ikke kunne indtaste tal. Dette gælder også når fx ordrens højde og længde bliver tastet ind, kan man kun indtaste tal ind. Grunden til det er, at vi har defineret det fra databasen af om det fx enten kun er en String eller en Integer. Det er med til at sikre der ikke opstår fejl de inputs vi får.

Sikkerhed i forbindelse med login

I forbindelse med login på hjemmesiden, har vi en LoginCommand, som tager sig af de fejl, der kunne opstå i forbindelse med login. I den har vi en try/catch, som går ind og fanger, samt udskriver hvad fejlen er.

Brugertyper

Vores hjemmeside er bygget op med to forskellige bruger indgange. Vi har en indgang for kunden, og en administrator indgang. Kunden har kun adgang, til at se kundens egen side og dermed kan kunde følge med i sin ordre.

Administrator indgangen kan sælger bruge. Her kan sælger se alle de igangværende ordrer, som er i deres system, som kommer fra databasen. Her kan sælger gå ind og bekræfte, kundes ordre, så kunden kan fortsætte til købet af ordren.

SVG-tegning

SVG-tegningen er opbygget, så den er dynamisk. Det betyder at den bliver lavet ud fra de mål der bliver givet på hjemmesiden.

SVG tegningens kode har en `StringBuilder`, hvor vi tilføjer strenge som indeholder SVG informationer. De informationer skal bruges, til at fremvise de enkelte elementer i tegningen.

```
//Making a svg object of the StringBuilder class.  
StringBuilder svg = new StringBuilder();
```

Så SVG'ens `StringBuilder` får en masse strenge, som til sidst danner en sammenhængende SVG-tegning på hjemmesiden.

Vores SVG-tegning bliver defineret i SVG klassen i services. Gennem `BuildCarportCommand` klassen. SVG-tegningen bliver oprettet, tegnet og kaldt. Det er her tegningen får sin længde og bredde.

```
private final String headerTemplate = "<svg height=\"%d%\" \" +  
    \"width=\"%d%\" \" +  
    \"viewBox=\"%s\" \" +  
    \"x=\"%d\" \" +  
    \"y=\"%d\" \" +  
    \" preserveAspectRatio=\"xMinYMin\">";
```

Den her template, bliver fx brugt, til at kunne generere de enkelte strenge.

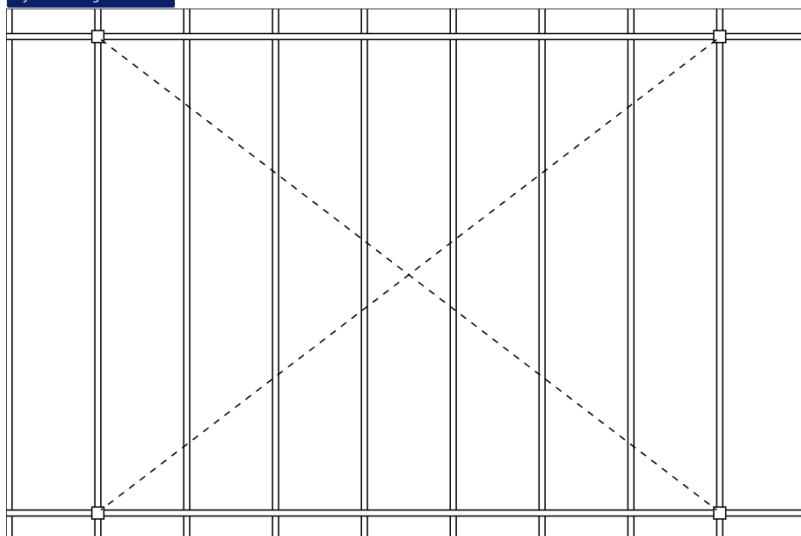
Når strengene er blevet genereret, fra målene der bliver givet på hjemmesiden. Kan SVG-tegningen blive lavet, og dermed blive vist for kunden.

Vi vil nu behandle din anmodning om en carport med målene:

Bredde: 400.0

Længde: 600.0

Tryk her for at gå til din side.



Stykliste

Vores stykliste er hardcoded data i sig, som er det der udgør styklisten. Styklisten bliver lavet i det målene for ordren kommer ind. Styklisten bliver lavet i databasen, samtidig med ordren bliver oprettet. De hardcoded elementer i styklisten er stolper, spær og rem.

Vi har også dynamiske værdier, da mængde og total prisen bliver udregnet.

Udvalgte kodeeksempler

I dette afsnit vil vi tage fat i nogle kodeeksempler, som viser hvordan noget af vores kode er bygget op. Vi har valgt kode eksemplerne ud fra, hvad vi finder essential for vores kode.

```
public class BuildCarportCommand extends CommandProtectedPage
{
    OrderFacade orderFacade = new OrderFacade(database);
    BomFacade bomFacade = new BomFacade(database);

    public BuildCarportCommand(String pageToShow, String role)
    {
        super(pageToShow, role);
    }

    @Override
    public String execute(HttpServletRequest request, HttpServletResponse response) throws UserException
    {
        HttpSession session = request.getSession();

        //Variables.
        Order order;
        double length;
        double width;

        try //Here we request the width and length entered by the customer.
        {
            length = Double.parseDouble(request.getParameter("length"));
            width = Double.parseDouble(request.getParameter("width"));
        }
        catch (NumberFormatException ex)
        {
            throw new UserException("Length or Width is missing");
        }

        //And setting the length and width in the Session, to be used elsewhere.
        request.setAttribute("length", length);
        request.setAttribute("width", width);

        //Here we do the calculations and add everything to our Bill of materials.
        Bom bom = new Bom();
        BomService bomService = new BomService(database);
        bom.addToBill(bomService.calculatePosts(length));
        bom.addToBill(bomService.calculateRafters(width,length));
        bom.addToBill(bomService.calculateBeams(length));

        //Here we pull the user ID from the session and create a new order object.
        User user = (User) session.getAttribute("user");
        order = new Order(length, width);

        try //Then here we are creating the new order, and inserting everything into the database.
        {
            orderFacade.createOrder(order, user.getId(), bom);
        }
        catch (Exception e)
```

```
{  
    e.printStackTrace();  
}
```

I det første eksempel herovenover tager vi fat i en lille del af vores *“BuildCarportCommand”* klasse, som er det it-systemet rammer efter kunde har indtastet målene til sin carport. Det første der sker når kunden har indtastet længde og bredde til carporten er, at vi henter de to parametre fra *‘session’* og parametrene bliver sat. Vores *‘request.getParameter’* er omringet af et *‘try, catch statement’*, hvori vi kaster vores *‘UserException’*. Det sker for eksempel, hvis kunden glemmer at indtaste en værdi ind i en af klasserne.

Derefter laver vi et nyt objekt, af vores *‘Bom’* klasse som vi kalder bom. Det gør at vi kan kalde på vores metode *‘addToBill’*, som ligger i *‘Bom’* klassen. Vi laver også et nyt *‘BomService’* objekt. Vi bruger den til at kalde på vores beregninger, som ligger i *‘BomService’*. Vi giver den de parametre, den har brug for med til beregningerne. I vores beregninger returnerer vi et objekt af vores *‘CarportItems’*, hvilket er det *‘addToBill’* metoden i Bom klassen modtager.

```
public class Bom  
{  
    //This is our list that contains the items for our Bill of materials.  
    private List<CarportItems> bomLines;  
  
    public Bom()  
    {  
        this.bomLines = new ArrayList<>();  
    }  
  
    public List<CarportItems> getCarportItems()  
    {  
        return bomLines;  
    }  
  
    public void addToBill(CarportItems carportItems)  
    {  
        bomLines.add(carportItems);  
    }  
}
```

I vores *‘Bom’* klasse der har vi en ArrayList af *‘CarportItems’*, som vi kan tilføje ting til ved hjælp af *‘addToBill’* metoden. Vi kan kalde på *‘bomLines’* med *‘getCarportItems’* metoden.

Dette er nødvendigt så vi kan kalde på Array af *‘CarportItems’*, når vores stykliste bliver lavet samtidig med ordren, bliver oprettet.

```
public class OrderMapper  
{  
    private Database database;  
    private long now = System.currentTimeMillis();  
    private Timestamp sqlTimestamp = new Timestamp(now);  
}
```

```

public OrderMapper(Database database)
{
    this.now = now;
    this.sqlTimestamp = sqlTimestamp;
    this.database = database;
}

//Here we add our order to the database and "insertIntoPartListItem" is called foreach order to add order_id.
public void createOrder(Order order, int userId, Bom bom) throws Exception
{
    int orderId = 0;
    String status = "pending";
    try (Connection connection = database.connect())
    {
        String sql = "INSERT INTO orders (width, length, status, user_id, timestamp) VALUES (?, ?, ?, ?, ?)";
        try (PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS))
        {
            ps.setDouble(1, order.getWidth());
            ps.setDouble(2, order.getLength());
            ps.setString(3, status);
            ps.setInt(4, userId);
            ps.setTimestamp(5, sqlTimestamp);
            ps.executeUpdate();

            ResultSet ids = ps.getGeneratedKeys();
            ids.next();
            orderId = ids.getInt(1);
            order.setOrderId(orderId);

            for (CarportItems carportItems : bom.getCarportItems())
            {
                insertIntoPartListItem(orderId, carportItems);
            }
        }
        catch (SQLException ex)
        {
            throw new UserException(ex.getMessage());
        }
    }
    catch (SQLException | UserException ex)
    {
        throw new Exception(ex.getMessage());
    }
}

//This is where we populate our 'PartListItems' table in MySQL. //Also referred too as 'Bom', 'CarportItems' or 'Bill of materials'.
public void insertIntoPartListItem(int orderId, CarportItems carportItems) throws UserException
{
    try (Connection connection = database.connect())
    {
        String sql = "INSERT INTO partlistitem (order_id, parts_id, name, quantity, length, unit, description, price) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        try (PreparedStatement ps = connection.prepareStatement(sql))
        {
            ps.setInt(1, orderId);
            ps.setInt(2, carportItems.getParts_id());
            ps.setString(3, carportItems.getName());
            ps.setInt(4, carportItems.getQuantity());
            ps.setDouble(5, carportItems.getLength());
            ps.setString(6, carportItems.getUnit());
            ps.setString(7, carportItems.getDescription());
            ps.setInt(8, carportItems.getPrice());
            ps.executeUpdate();
        }
        catch (SQLException ex)
        {
            throw new UserException(ex.getMessage());
        }
    }
    catch (SQLException ex)
    {
        throw new UserException(ex.getMessage());
    }
}

```

Henne i vores *OrderMapper* klasse er det her, det hele bliver samlet og skubbet op i databasen. I vores *createOrder* metode får vi tre parametre med, et *Order* objekt, *user_id* og et *Bom* objekt. Derefter skubber vi selve ordren op i database i de korrekte kolonner. Her bliver styklisten lavet. Vi har et 'foreach loop' der laver et nyt *CarportItems* objekt for hver *bom.getCarportItems*, som er den der returnerer det Array vi lavede tidligere. For hver af dem kalder vi på *insertIntoPartListItems*, hvor vi giver den *order_id*'et med fra vores *createOrder* metode. Vi sikrer os at de matcher ind. Den modtager også et *CarportItems* objekt. Metoden *insertIntoPartListItems* skubber så hele styklisten op til databasen.

Status på implementering

I vores opgave har vi udarbejdet ti user stories som ligger til grund for vores opgave. Ud af vores ti user stories nåede vi ikke at udarbejde fem, ni og ti. Dette skyldes at vi oplevede uventede vanskeligheder, samt var de af lavere prioritet. Vi valgte at have vores fulde fokus på den bagvedliggende kode såsom at oprette en ordre, godkende en ordre osv. Vi ville nok have nået dem med mere tid og færre uventede vanskeligheder.

User stories vi ikke nåede

5	Som kunde skal jeg kunne modtage plantegningerne samt styklisten efter jeg har betalt min ordre.
9	Som sælger, så vil jeg gerne kunne godkende en ordre og evt. få kontakt til kunden
10	Som sælger skal jeg kunne se plantegning og stykliste for bestillingen, som bliver udregnet via it-systemet

Manglende CRUD

Når man tager fat i CRUD som står for create, read, update and delete mangler vi to operationer ud af fire i AdminPage. Vi oplevede problemer i forhold til vores create operation som gjorde at vi ikke nåede at oprette metoderne til read og delete knapperne i AdminPage. Vi valgte fortsat at have knapperne på vores hjemmeside, men de gør ingenting når man klikker på dem. Derudover når man vil lave en ordre har man mulighed for at tilføje et skur. Dette nåede vi heller ikke at oprette en metode til. Derfor passer vores SVG-tegninger heller ikke til at man kan oprette en ordre med et skur.

En anden ting vi manglede til AdminPage er at telefonnummeret til kunden skulle stå der. Da vi havde problemer med vores foreach loop i html gjorde at den udskrev tingene op til flere gange. Da vi ikke

kunne finde en løsning til dette problem, så valgte vi at fjerne det fra vores hjemmeside. Det var dog noget vi arbejdede på men opgav til sidst.

Hvis man tager fat i vores stykliste, havde vi mange tanker om hvornår den skulle vises for kunden. Da Fog havde sagt i det materiale vi fik udgivet, at de ikke ønskede at kunden så styklisten før ordren var betalt. På vores hjemmeside kan det ses at styklisten kommer før man betaler. Dette var noget vi ønskede at ændre til først i kvitteringen, men igen nåede vi ikke at rette det.

Noget andet i forhold til styklisten er at vi ikke rammer tre operationer i forhold til CRUD. Det er read, update og delete da vi har været nødt til at hardcode vores navn og description ind. Vi ønskede at få den til at implementeres fra vores databasen, men måtte nedprioritere og derfor vælge den “nemme” løsning i forhold til den del.

Da vi skulle lave vores SVG-tegning lavede vi den i BuildCarportCommand. Vi ønskede at rykke metoden til en anden command klasse. Dog stødte vi på en del problemer undervejs. Dette medførte at vi havde problemer med at oprette vores SVG-tegning ovenpå en anden SVG, da vi ikke kunne kalde på en metode i execute metoden. Dette skabte os utroligt mange problemer, hvilket også ligger til grund for vi ikke har det ekstra flotte layout til vores SVG-tegning.

Styling

Når vi ser på vores styling, har vi ikke udarbejdet vores sider responsive. Det kan blandt andet ses når vi gør siden mindre, så bliver teksten i footeren ikke gjort mindre sammen med billedet. Det ville have krævet mere viden omkring bootstrap og en større viden indenfor HTML. Da vi lagde det fulde fokus på vores bagvedliggende kode, blev stylingen desværre nedprioriteret.

Test

Vi har valgt at benytte os af følgende test former til at teste vores hjemmeside. Integrations test, Unit test, og ad hoc test.

I vores unit test har vi valgt at have fokus på udregningen af stolper, bjælker og spær, da det er en essentiel del af vores program, og vi følte det var vigtigt at sikre os at alle beregningerne er korrekte. Vi har udført vores unit test ved at give nogen testen en fiktiv længde og bredde, som vi allerede

kender resultatet på i forvejen. Vi tjekker ved hjælp af “AssertEquals” metoden om det er det resultat vi ender med.

I integrationstesten havde vi oprettet en test database, hvor i vi opretter nogle test data i test databasen. Vi valgte bla. at teste vores “BomMapper” klasse som har en funktion til at beregne den totale pris. Her har vi så indsatte nogen test data i databasen for at tjekke metoden beregner er det samme, som det vi forventer.

Vi har også benyttet os af mange små ad hoc test, dem har vi haft stor gavn af at lave. En ad hoc test er en lille test man udfører undervejs igennem projektet. Et eksempel kunne være en klasse med en main metode, hvor i man kan kalde en specifik metode i sit program, for at se hvordan det virker. Man kan også give metoden nogen hardcoded data, så man har mere kontrol over det forventelige resultat.

Coverage Breakdown

Package	Class, %	Method, %	Line, %
business.persistence	80% (4/ 5)	45% (9/ 20)	27.8% (88/ 316)
business.services	28.6% (2/ 7)	6.5% (2/ 31)	6.5% (6/ 92)

I dette diagram kan vi se den samlet udregninger i procenter for vores test. Det vil sige, vi kan se hvor meget af vores program, der er dækket af test. I vores Business Persistence klasse bliver fire ud af fem klasser dækket af test. Herunder er det ni ud af 20 metoder som bliver dækket af test og det er tilsvarende 88 ud af 316 linjer koder, der bliver dækket.

I Business Services klasse bliver to ud af syv klasser dække, seks ud af 31 metoder bliver dækker og tilsvarende bliver 47 ud af 92 linjer dækket af test.

Proces

For at få vores projekt til at fungere bedst muligt, har vi arbejdet med værktøjerne GitHub, Taiga, Google docs og Zoom. GitHub har samlet projektet, i og med vi har haft en main branch og hver person i gruppen har haft sin egen branch, der har kunne arbejdes ud fra.

Google docs og Taiga, de har gået hånd i hånd, når vi har ændret i det ene program, er det også blevet skrevet ind eller flyttet i det andet. De har vi brugt til at skrive logbog, fx noter ned, med de næste steps og hvad vi har aftalt med vores vejledere, Caroline eller Jon.

Arbejdsprocessen faktuel og reflekteret

I dette afsnit, vil vi beskrive hvordan vores sprints- og gruppearbejde har forløbet, under projektet. Inden vi gik i gang med projektet, forventningsafstemte vi i gruppen, hvordan arbejdsmoralen skulle være i gruppen. Det gjorde vi, så det bedst ønskede resultat for gruppen, kunne blive opnået. Vi lavede en aftale om at arbejde fra kl. 9 til ca. 16 hverdag i ugen, og hvis man havde noget der gik ind over den planlagte tid, skulle man aftale det med gruppen. Vores arbejdsproces i det tidsrum, har derudover også inkluderet daglige møder på starten af dagen. Her har vi sat dagsordenen og sat forventning til, hvor mange eller hvor meget af en user stories vi gerne ville nå. Derudover satte vi estimer for vores user stories.

Vi gik ind til projektet med den holdning, at alle har ansvar for koden og rapporten. Vi har uddelegeret nogle af opgaverne, men størstedelen af opgaverne har alle været inde over. Da vi som gruppe, har været gode til at spare med hinanden, og på den måde har kunne løse opgaverne mere effektivt. Fra projektets start til slut, har Alex været Scrum master, da det har fungerede bedst for os i gruppen. Det har sikret os, at vi har bevaret et overblik samt haft en guid line under hele projektet.

Vores definition på estimeringen af en user story:

- Small: Halv dagsarbejde.
- Medium: Hel dagsarbejde.
- Large: Halvdelen dagsarbejde
- Extra-large: To dages arbejde.

Sprint 1:

Her var fokuset at få lavet de grundlæggende elementer for projektet. Vi ville have styr på User Stories, lavet domænemodel, ER diagram og udkast til navigation modellen.

Sprintet indeholdtes følgende user stories:

- US 2: Som kunde kan jeg oprette en konto profil, så jeg kan gemme og bestille min ordre.

- US 3: Som kunde skal jeg kunne vælge målene til min carport og for at få et tilbud på carporten.

Begge user stories vurderede vi til at være medium opgaver. Dette sprint fik vi overestimeret, hvor hurtigt vi kunne komme videre med, i forhold til hvad vi havde talt med Caroline om. Vi endte med at gå videre med user stories 7. og 8. også, da vi vurderede, at de passede bedst, til det næste step i processen i forhold til, hvad vi havde aftalt med Caroline. Vi nåede derfor også at blive færdige med user story 7. Vi havde vurderet dem henholdsvis til en at være large og medium størrelser.

- US 7: Som administrator så skal jeg kunne logge ind og se ordrehistorikken, samt “pending” tilbud.
- US 8: Som administrator skal jeg kunne gå ind og ændre en kundes ordre fra “pending” til “confirmed”, (så kunden kan se sin plukliste og plantegning.)

Vi skulle i dette sprint, også være bedre til, at bruge Taiga, og ikke kun bruge docs dokumentet til at dokumenter processen i.

Sprint 2:

I dette sprint, var fokuset at få udregningen til styklisten, samt plantegningen til at blive færdige. Vi fik flere bolde i luften i dette sprint, i forhold til sprint 1.

Sprintet indeholdtes følgende user stories:

- US 4: Som kunde kan jeg få et tilbud på min carport, så jeg kan se den samlede pris. (M)
- US 6: Som kunde skal jeg kunne modtage plantegningerne samt styklisten efter jeg har betalt min ordre. (XL)
- US 8: Som administrator skal jeg kunne gå ind og ændre en kundes ordre fra “pending” til “confirmed”, (så kunden kan se sin plukliste og plantegning.) (M)
- US 11: Som sælger skal jeg kunne se plantegning og stykliste for bestillingen, som bliver udregnet via it-systemet. (XL)

Vi estimerede user historierne 4: medium, 6: Ekstra large, 8: medium og evt. 11 til ekstra large.

Vi underestimeret dette sprint, i forhold til hvad vi havde regnet med. Der kom andre ting udefra der påvirkede projektet meget, både i privaten og tekniske vanskeligheder. Vi havde fx problemer med IntelliJ på Alex computer, som ikke ville løse sig. Det løste sig til sidst med hjælp fra Jon. Udover

det ramte vi en mur med nogle af udregningerne, som tog længere tid end forventet. De fleste af problemerne var lige ved at være færdige, men vi fik dem først enten løst på dagen vi havde talt med Caroline, eller dagene efter.

Sprint 3:

Her havde vi fokus på at få SVG, styklisten og unit- samt integrations-test lavet færdige. Vi ville også sikre os, at layoutet blev finpudset, samt at få gjort så mange af de resterende user stories færdige.

Sprintet indeholdt følgende user stories:

- US 5: Som kunde kan jeg få godkendt tilbuddet på min valgte carport så min ordre kan blive sat i gang. (S)
- US 9: Som kunde kan jeg følge med i min ordrestatus så jeg ved hvor langt min ordre er. (S)
- US 10: Som sælger, så vil jeg gerne kunne godkende en ordre og evt. få kontakt til kunden.
- US Test: Unit test/integration test
- Layout: Finpudsning af HTML

Vi estimeret tiden på user stories meget godt - 5: small, 9: small, 10: large, Test: medium og layout: small. Vi var kommet til et punkt i det her sprint, hvor vi blev nødt til at prioritere vores tid og energi. Selvom energien og koncentrationen også var lidt præget, var vi gode til, stadig at holde fast i vores struktur og sparre med hinanden. Vi blev samtidig nødt til at være gode til, at lægge opgaven fra os selvom der var små ting som kunne fikses, gøres bedre eller arbejde videre på, så vi kunne komme i gang med rapporten.

Vores rytme med at arbejde i hverdagene fra kl. 9 - 16, fandt vi ud af var en god struktur for os, lige fra starten af. Vi har kunne sidde sammen over Zoom, og hele tiden være med i alt hvad der har foregået. Hvis projektet har spidset til, har vi også ud over de, aftalt møder om aften eller i weekenden. Vi inden da, har uddelegeret nogle arbejdsopgaver imellem os, så alle har været sikker på hvad der hele tiden har skulle ske.

Konklusion

I konklusion vil vi gerne komme ind på vores tankegang, udfordringer og de erfaringer, vi har fået gennem projektet.


Vi er tilfredse med, vores layout for hjemmesiden og at vi kan få en kunde godt fra a til b. Vi kom igennem så vi har en funktionel hjemmeside. Vi ønskede at den skulle have været endnu mere funktionel og i bedre kvalitet. Selvom vi gerne ville have kodet videre, fuldført de mål og de intentioner som vi havde sat os for. På et tidspunkt blev vi nødt til at stoppe, så vi havde en reelle mulighed, for at lave rapporten ordentlig færdig.

Under projektet opstod der forskellige problemer; både med IntelliJ, merge konflikter i GitHub eller når vi sad fast i vores kode. De fleste af de problemer vi havde, fik vi løst ved at sparre med hinanden, tage problematikken step-by-step, søge på nettet og sidste udvej, spørge om hjælp. Det har været med til at give os en god sammenknytning af den viden vi har fået på 2. semester.

Selvom vi ville have ønsket en endnu mere funktionel hjemmesiden, det var et valg vi endte med at tage. Vi blev nødt til at sikre, at vi kunne komme igennem projektet og komme ud med et godt resultat som vi er tilfredse med.

Bilag

Fog®

LOG IND 

BYG DIN CARPORT

Carport bredde

▼

Carport længde

▼

Tag

▼

Redskabsrum

☐ Ja tak

☐ Nej tak

Navn

▼

Adresse

▼

Postnummer og by

▼

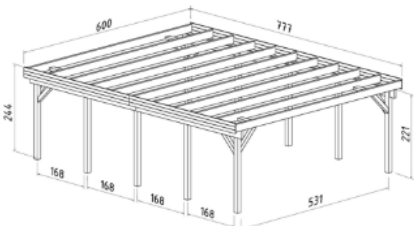
Telefon

▼

Email adresse

▼

SEND FORESPØRGSEL



Webshop

Kontakt

Webshop

Johannes Fog A/S - Firskovvej 20 - 2800 Lyngby - CVR-nr. 16314439

Login

Brugernavn

Password

Login

Opret bruger

Webshop

Kontakt

Webshop

Johannes Fog A/S - Firskovvej 20 - 2800 Lyngby - CVR-nr. 16314439

Opret bruger

Navn

Adresse

Postnummer og by

Telefon

Email adresse

Opret bruger

Webshop

Kontakt

Webshop

Johannes Fog A/S - Firskovvej 20 - 2800 Lyngby - CVR-nr. 16314439