



Datamatiker 2. semester
<https://github.com/rasm445f/Fog-Carport>

Semesterprojekt - Fog Carport

Christoffer Perch Nielsen
cph-cn332@cphbusiness.dk



Anders Meinicke
cph-am423@cphbusiness.dk



Rasmus Taul
cph-rt91@cphbusiness.dk



Afleveringsdato 31. maj 2022

Links

Git

Github repository:

<https://github.com/rasm445f/Fog-Carport>

Demo

Vi synes alle tre, at det kunne være sjovt at lave videoen, hvor vi viser hjemmesiden frem. Vi blev dog enige om at det ville blive for kaotisk, hvis vi alle 3 var med i videoen. Derfor trak vi lod om, hvem der skulle være med i den.

Demovideo af hjemmesiden:

<https://youtu.be/faUsfINCwJE>

Droplet

Link til kørende hjemmeside:

<http://165.22.88.18:8080/FogCarport/>

Der kan logges ind med følgende informationer på hjemmesiden:

e-mail: user@user.dk

kodeord: user

e-mail: admin@admin.dk

kodeord: admin

Virksomheds pitch

Link til video om virksomheden:

https://youtu.be/JyT_OukpKes

Indholdsfortegnelse:

Links	1
Git	1
Demo	1
Droplet	1
Virksomheds pitch	1
Indholdsfortegnelse:	2
Indledning	4
Baggrund	4
Virksomheden	5
SWOT analyse af Fog	5
Value Proposition Canvas	6
Teknologivalg	7
Krav	8
Funktionelle krav	8
Ikke-funktionelle krav	8
User stories	9
Aktivitetsdiagram - ud fra kundens nuværende system	11
Aktivitetsdiagram - vores system	12
Git	13
Domæne model	14
Klassediagram	15
EER diagram	16
Constraints	17
Relationer	17
Normalform	17
Navigationsdiagram	18
Arkitektur	19
Særlige forhold	20
Session	20
Exceptions	21
Stykliste	21
SVG	22

Udvalgte kodeeksempler	24
Stykliste	24
CRUD metoder	25
Status på implementering	26
Test	27
Manuelle tests	27
JUnit	28
Arbejdsprocessen faktuelt	28
Kanban og Git issues	28
Vejledning og faser	30
Arbejdsprocessen reflekteret	31
Arbejdsprocessen	31
Kanban og evaluering	31
User stories	32
Github	32

Indledning

Vi har fået til opgave at bygge et webbaseret system til Johannes Fog, som skal bruges til at bestille skræddersyede carporte på nettet. På baggrund af interviewet med Martin fra Fog har vi fået et indblik i, hvordan de driver deres forretning samt hvordan deres nuværende system fungerer. Ud fra interviewet har vi skabt nogle krav til systemet og senere omformuleret dem til user stories, som er funktionaliteten i systemet. I opgaven har vi valgt at fokusere på at gøre systemet så automatiseret som muligt. Der er tilkoblet en database til vores system, som anvendes til at gemme og læse forskellige data.

I bund og grund handler opgaven om, at man skal kunne bestille en carport. Som kunde skal man vælge længde og bredde på carporten samt vælge tagtype og om den skal have et redskabsrum eller ej. Hvis man tilføjer et redskabsrum så skal der også vælges mål til dette. Systemet skal derefter kunne generere en stykliste, som kunden får adgang til på et senere tidspunkt og en tegning af carporten, som er genereret med SVG. Sælgeren hos Fog skal have mulighed for at ændre ting som priser og status på forespørgslerne.

Vi startede forløbet i opgaven med at sætte vores versionsstyring værktøj - Github - op. Vi har gjort brug af flere værktøjer, som nævnes senere i rapporten. Disse har hjulpet os med at holde et overblik. Derudover har vi også holdt daglige møder i gruppen og været til vejledningstimer med vores lærer Jon, som har gjort at vi har haft en struktureret arbejdsproces.

Baggrund

Vores kunde i projektet er Johannes Fog. Firmaet fokuserer på at sælge forskellige produkter og services til deres kunder indenfor byggeindustrien. Fog sælger alt fra lamper og maling til carporte og drivhuse osv. Vores kundes krav er, at vi skal bygge et system til deres afdeling, der sælger skræddersyede carporte og den nye løsning skal erstatte kundens nuværende løsning. De ønsker et nyere og mere attraktivt system, som er nemmere at vedligeholde og opdatere i fremtiden. Derudover skal systemet også være mere automatiseret.

Virksomheden

Vi har lavet vores SWOT analyse og value proposition canvas VPC ud fra interviewet med Johannes Fog. I interviewet fortæller Martin Kristensen, hvordan de driver virksomheden og han forklarer nogle af de svagheder og styrker, som de selv mener de har. Interviewet har givet os et godt indblik i, hvilken virksomhed vi arbejder med. Formålet med at lave en SWOT analyse er at identificere virksomhedens styrker og svagheder, samt hvordan de kan forbedre dem og om de har nogle trusler. Analysen har vi udarbejdet på baggrund af interviewet og derudover har vi undersøgt hvilke andre brancher der kunne være en trussel for Johannes Fog. Formålet med vores VPC er at se, hvordan de driver virksomheden. Det har vi også undersøgt på baggrund af interviewet, men også ved at kigge på deres nuværende hjemmeside.

SWOT analyse af Fog

Strengths:

- De har en erfaren sælger på hver ordre til at rådgive kunderne.
- De kan redigere prisen på hver ordre og kan nemt se dækningsgraden.
- Kvaliteten af materiale samt service er god..
- De kan hurtigt og nemt skabe en stykliste og et 2d billede af carporten ud fra kundens valgte mål.
- Kunden får først styklisten når de har betalt. Dermed kan kunden ikke sende en forespørgsel og derefter gå ud i andre butikker og købe hvert enkelt materiale.

Weaknesses:

- Det kan være svært at opgradere systemet, da det skal være koblet sammen med resten af deres systemer.
- Deres hjemmeside og system er en smule outdated.

Opportunities:

- De kunne opgradere deres system så det giver mulighed for at vise 3d billeder af carporten. Det vil skabe en bedre visualisering for kunden.
- De kunne forbedre systemet til at inkludere en metode der redigerer, hvor et skur ligger på billederne af carporten.
- De kunne tilføje flere byg-selv projekter fx legehus og anneks.

Threats:

- Andre firmaer der kan lave et lignende system.
- Andre firmaer som fx Bauhaus eller Silvan, vil oftest have nogle af materialerne til billigere penge.
- Øgede omkostninger på deres materialer.
- Høj inflation.
- Grundet den manglende adgang til databasen bag systemet, er det ikke muligt at opdatere systemet selvom de har et system klart, som de har betalt mange penge for.

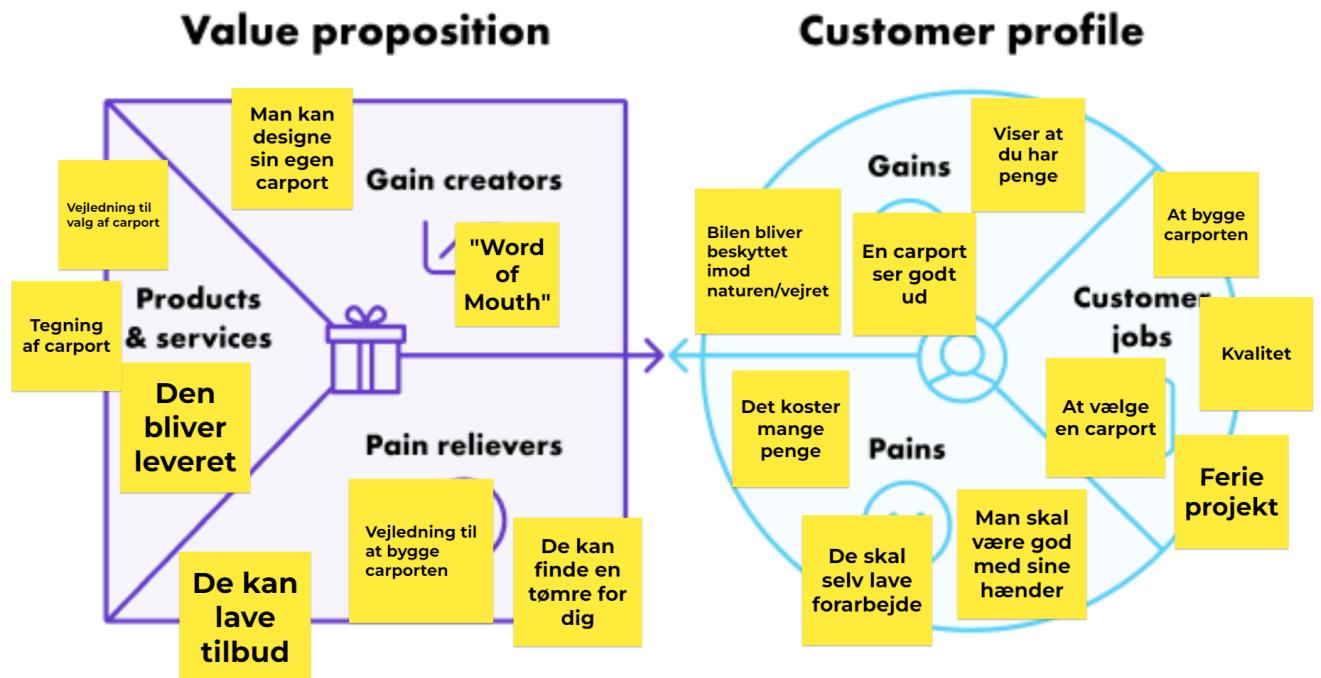
Vi vurderer at Johannes Fogs største trussel er, at andre firmaer kan skabe et lignende system som ikke har de problemer, som Johannes Fog er stødt ind i og muligvis også et nyere og mere attraktivt system. Fog vil derfor have svært ved at følge med konkurrenterne da de ikke let kan opdatere deres system. For at mindske deres største trussel er vi derfor nødt til at løse deres største svaghed, hvilket er at deres system ikke let kan opdateres.

Den bedste løsning vil være at bygge et nyt system fra bunden. Systemet skal udføre de samme metoder som det gamle system, men denne gang udføres så det er nemmere at vedligeholde fx hvis en pris opdateres, så sker det overalt i systemet.

Denne proces vil være dyr, da man skal lave et helt nyt system. Fordelen ved at gøre det er, at Fog sikrer sig mod at konkurrenter ikke har en fordel i fremtiden.

Et andet problem der kan opstå ved at skifte systemet og databasen er at informationer vil blive slettet, hvis man ikke kan få fat i informationer fra den gamle database.

Value Proposition Canvas



Arbejdet med SWOT hjalp os til at skabe et billede af virksomheden. Efterfølgende kunne vi lettere bruge vores viden til at fokusere på de problemer vi fandt. Med udarbejdelsen af VPC'en kunne vi se hvordan forretningen virkelig kørte og det hjalp os med at sætte os i firmaets sted. Vi arbejdede med SWOT og VPC hver for sig. Det burde vi have undgået, da de to modeller kan fodre hinanden med ideer. Når vi engang skal lave et lignende projekt vil det være en fordel at arbejde på de to modeller samtidig. På den måde undgår gruppen at returnere til noget, som vi troede allerede var færdigt. Denne metode kan være dårlig, for når man kommer tilbage til en analyse, kan man have mistet den rytme man havde da man begyndte på det, hvilket bevirker at processen tager længere tid. VPC diagrammet var den sværreste at udfylde. Den største forhindring var, at vi ikke var helt sikre på hvor de enkelte ting skulle placeres. Efter en snak med en af vores lærere fik vi et bedre syn på det og skabte diagrammet som er vist ovenfor.

Teknologivalg

Nedenfor ses en tabel over teknologier som er anvendt i systemet. Dette er relevant, hvis der på et tidspunkt er nogle andre som skal overtage systemet, så de ved hvilken software der skal anvendes. Vi har ikke selv valgt disse teknologier, men det er dem vi har arbejdet med på 2. semester.

Teknologi navn	Version
IntelliJ IDEA (Ultimate Edition)	2021.3.2
JDK	17.0.1
Apache Tomcat	9.0.60
MySQL Workbench	8.0.28
MySQL JDBC	8.0.28
HikariCP	4.0.3
JUnit	5.8.2
Java Servlet API	4.0.1
Maven	4.0.0

Krav

Vi har fået udleveret kravene til IT-systemet i forbindelse med interviewet med Johannes Fog, hvorefter vi selv skulle formulere dem. I interviewet bliver der fortalt, hvordan deres nuværende system fungerer og hvilke ønsker de har til det system, som vi skulle bygge. Derudover har lærerne på skolen også stillet nogle krav til opgaven. Disse kan kategoriseres som ikke-funktionelle krav. Dernæst skabte vi vores user stories ud fra kravene og vi har valgt at fokusere på delen, hvor man selv designer sin carport. Vi har også baseret vores user stories ud fra vores niveau og den tidsramme vi har haft til at bygge systemet, da det ville være urealistisk at skulle bygge en forbedret udgave af Johannes Fog's nuværende system som studerende på 2. semester.

Funktionelle krav

- Kunder skal kunne se et valg af carporte.
- Kunder skal kunne sortere carporte ud fra forskellige kriterier.
- Kunder skal kunne se en tegning af carporten.
- Kunder skal kunne sende en forespørgsel på en carport med selvvælgte mål.
- Kunder skal kunne vælge mellem forskellige tag på carporten.
- Kunder skal kunne vælge om der skal være et redskabsrum i carporten.
- Kunder skal ikke kunne se styklisten før betaling.
- Sælgeren skal modtage forespørgslerne på de unikke carporte.
- Sælgeren skal have mulighed for at skabe en tegning af carporten ud fra målene.
- Sælgeren skal ud fra forespørgslens kriterier kunne skabe en stykliste.
- Sælgeren skal kunne ændre prisen på ordren og se dækningsgraden.
- Admin skal kunne justere priserne på materialerne.
- Admin skal kunne oprette nye varer.

Ikke-funktionelle krav

- Løsning skal baseres på en MySQL 8 database.
- Designet skal afspejle en flerlags arkitektur, der afvikles på en Java server (Tomcat 9).
- Webapplikationen bør bygges af almindelige Java klasser, servlets, og JSP-sider.
- Websiderne skal fungere i enten Google Chrome eller Firefox.
- Websitet skal deployes i skyen (Digital Ocean).
- Kildekoden skal være tilgængelig i et Github repository.

User stories

Vi har formuleret 7 user stories ud fra kundens krav. User story 5 er opdelt i to dele: A og B, på grund af at det er en stor opgave. Vi har tildelt hver user story en kategori: small, medium eller large, som afspejler hvor lang tid den vil tage og sværhedsgraden. Som udgangspunkt er small en user story, der kan laves på nogle timer, hvor medium kan tage en dag eller lidt mere og til sidst large, som har været de user stories der er brugt mest tid på.

US1:

Som kunde ønsker jeg at kunne oprette en bruger på hjemmesiden og at kunne logge ind for at bestille en carport.

Givet at jeg som kunde har oprettet en bruger, vil denne data bliver gemt i databasen, så jeg senere kan logge ind igen.

Small

US2:

Som kunde ønsker jeg at kunne sende en forespørgsel på en byg-selv carport hvor jeg kan vælge mellem et bredt antal forudbestemte mål på carporten og redskabsrummet samt hvilket tag der skal bruges.

Givet at jeg som kunde har sendt forespørgslen, vil den blive gemt i systemet så både kunden og admin kan se den.

Medium

US3:

Som kunde ønsker jeg at kunne gå ind på en side, hvor jeg kan se mine forespørgsler, og har adgang til at se de mål jeg valgte samt tagtype, prisen og status.

Givet at forespørgslen er sendt afsted, kan jeg nu se et overblik over mine forespørgsler.

Medium

US4:

Som kunde ønsker jeg at kunne se en skitse af den tilhørende carport. Den skal kunne findes ud fra hver forespørgsel på siden med kundens forespørgsler.

Givet at der er sendt en forespørgsel vil både admin og kunden kunne se en skitse af carporten på deres side, hvor de har et overblik over forespørgslerne.

Large

US5A:

Som admin ønsker jeg at kunne lave en meget simpel stykliste af carporten, altså en stykliste hvor carporten har fladt tag og ingen redskabsrum.

Givet at admin har godkendt ordren, så får kunden også adgang til denne stykliste.

Large

US5B:

Som admin ønsker jeg at kunne lave en mere avanceret stykliste af carporten, hvor der tages hensyn til både fladt og skråt tag, samt redskabsrummet.

Givet at admin har godkendt ordren, så får kunden også adgang til denne stykliste.

Large

US6:

Som både admin og kunde ønsker jeg at kunne slette min forespørgsel.

Givet at den slettes, så vil både carporten, ordren og styklisten blive slettet fra databasen.

Medium

US7:

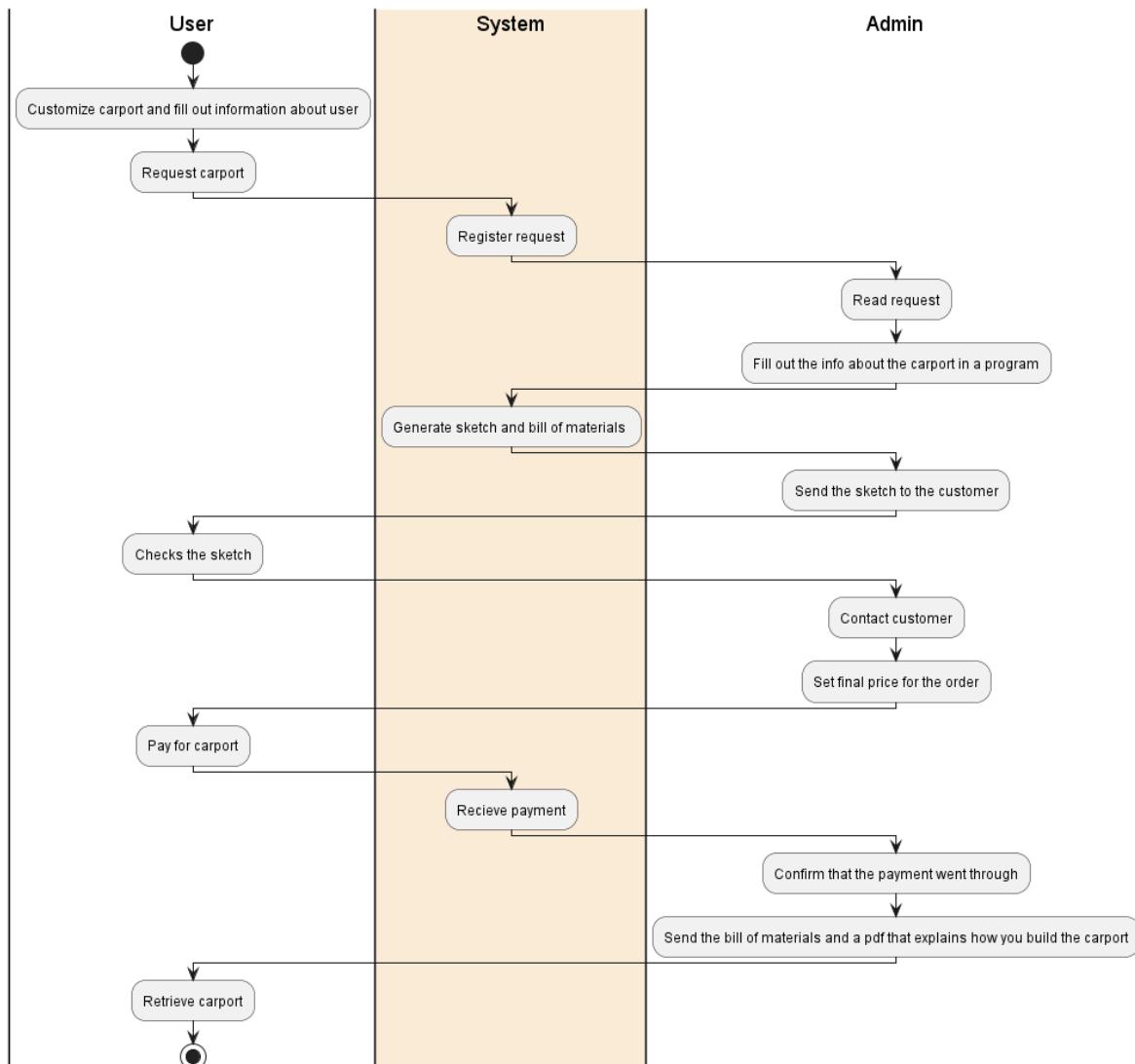
Som admin ønsker jeg at kunne tilgå en side, hvor jeg kan se alle kunders forespørgsler og få de samme informationer som kunden. Admin skal derudover også kunne se kundens navn, opdatere status på forespørgslen og opdatere prisen.

Givet at jeg som admin ændrer statussen til godkendt, så får kunden adgang til styklisten og hvis jeg ændrer prisen, så bliver den også ændret på kundens side.

Medium

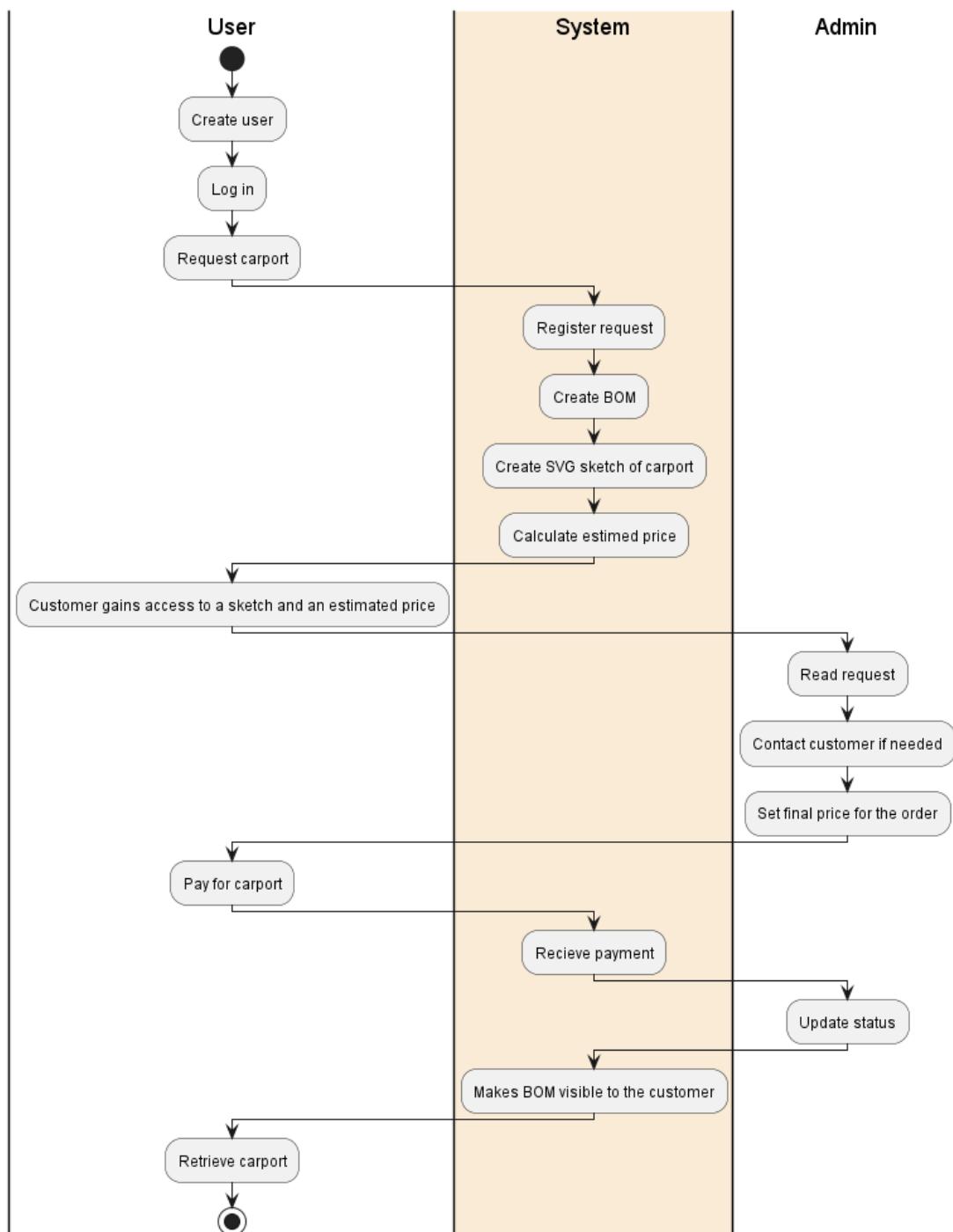
Aktivitetsdiagram - ud fra kundens nuværende system

Nedenunder kan man se det aktivitetsdiagram, som vi har lavet ud fra kundens nuværende system, altså den måde vi har fortolket deres system fungerer på ud fra interviewet. Vi har lavet to aktivitetsdiagrammer. Det ene kan beskrives som “as-is” og det andet som kommer længere nede kan beskrives som “to-be”, altså det som vi har lavet ud fra vores system. Som man kan se i diagrammet nedenunder, er der flere trin hos admin (som er sælgeren) end der er i vores “to-be” diagram. Det skyldes, at vi har ønsket at lave et system, som er mere automatiseret og hvor der kan ske færre “menneskefejl”. I Johannes Fogs nuværende system, skal sælgeren selv ind og vælge målene på carporten, tagtypen og målene på redskabsrummet i et program. I vores system er det primært kunden som kan lave fejl når de sender en forespørgsel, men det er som sagt kun en forespørgsel, og sælgeren har stadig mulighed for at kontakte kunden inden de betaler.



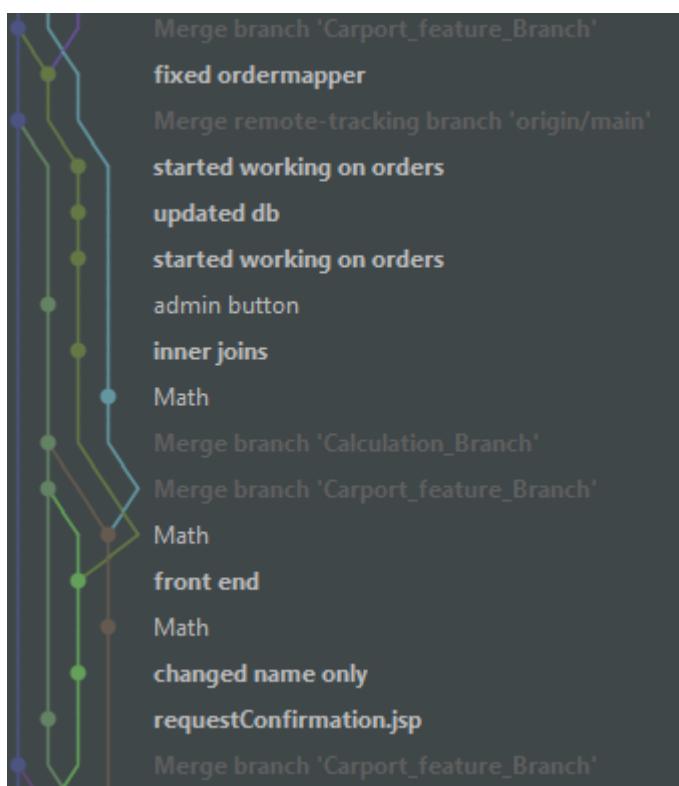
Aktivitetsdiagram - vores system

Vi har udarbejdet et aktivitetsdiagram af vores system, som ses nedenunder. Formålet med diagrammet er at give kunden (Johannes Fog) et overblik over hvordan systemet fungerer. Det er delt op i user, system og admin, hvor user er kunden til den enkelte ordre og admin fungerer som sælgeren i virksomheden. Man kan fx se at kunden ikke får adgang til styklisten før sælgeren har godkendt ordren eller systemet har modtaget betaling. Derudover opretter systemet selv styklisten og skitsen af carporten, når det modtager en forespørgsel.



Git

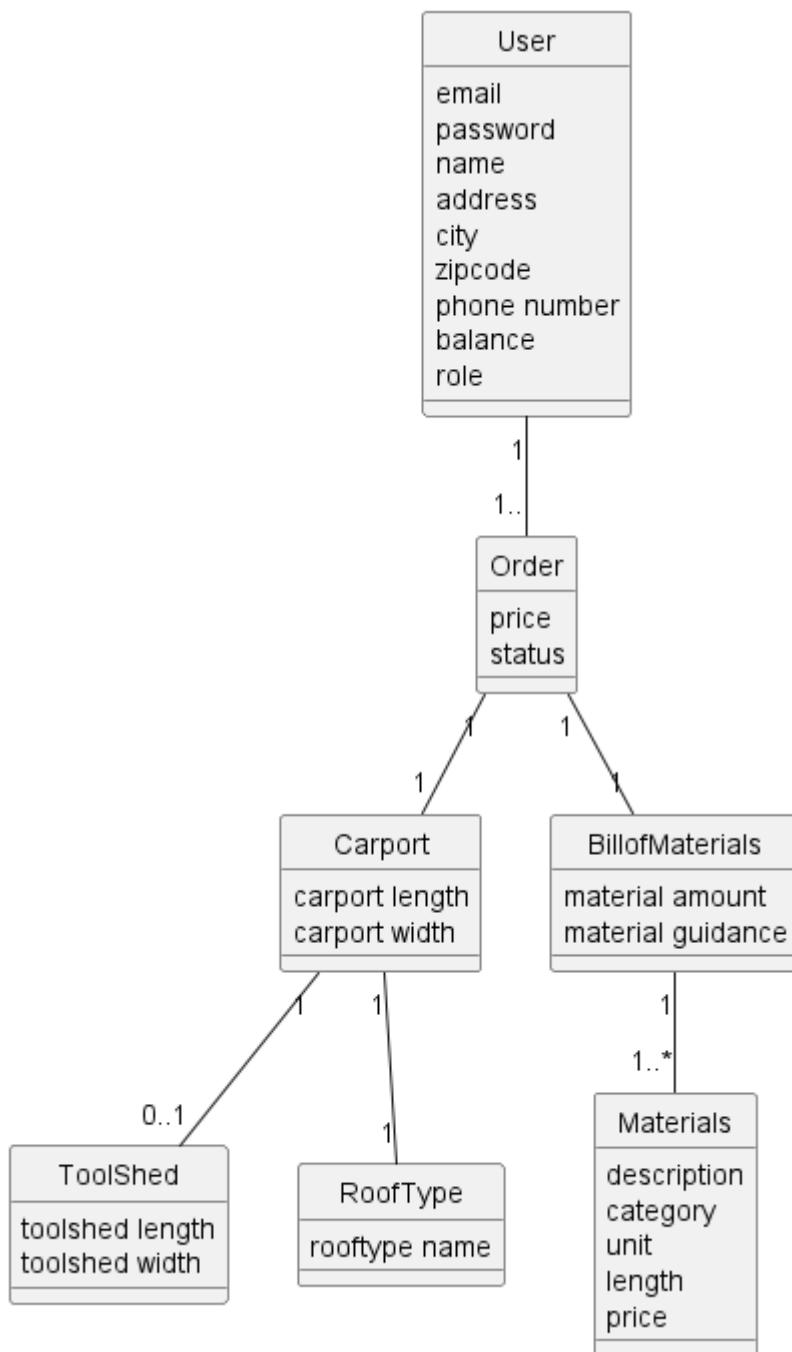
Vi har brugt Git til versionsstyring af vores projekt. Vi startede med at klone den startkode, som blev udleveret. Dernæst lavede vi en main branch, hvori vores endelige kode også ligger. Under arbejdsprocessen har vi hver især arbejdet i forskellige branches og når man var færdig med en del af koden, så gik vi den igennem sammen for at sikre at der ikke var fejl, og at det kunne pushes til main branchen. Hvis vi skulle have gjort det anderledes, skulle vi nok have haft en branch, som var opdateret med main branchen og så kunne vi pushe derover inden at vi pushede til vores endelige main branch.



I billedet ovenover ser man en del af vores “git træ” hvor den lilla linje til venstre illustrerer vores main branch og prikkerne er commits. Vi har anvendt Git igennem IntelliJ frem for terminalen, da IntelliJ kommer med en masse forklaringer og advarsler, hvorimod hvis man brugte Git i terminalen, så gør den bare det man skriver. Derudover har vi også brugt merge frem for rebase, da det er det vi har arbejdet mest med og følte os sikre i.

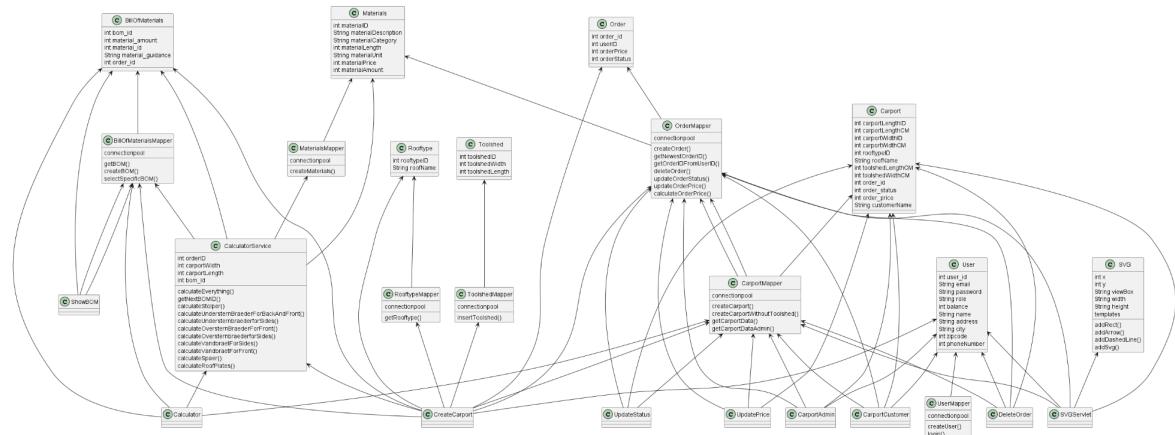
På den måde skaber det også en større historik i Git end hvis vi havde rebased som omskriver historikken.

Domæne model



Vores design til domæne modellen er simpelt, så kunden kan få et indblik i hvordan systemet fungerer. Ud fra modellen kan man se, at der er en bruger som kan have en eller flere ordrer i systemet. En ordre består af en stykliste, som siger noget om hvor mange af hver enkelt materiale der skal bruges samt en kort beskrivelse af hvordan det enkelte materiale skal anvendes. Der kan være mange materialer i styklisten og det er beskrevet med “1..*”. Derudover består ordren også af en carport, som selvfølgelig har nogle mål og den vil altid have et slags tag. Redskabsrummet er noget man kan tilvælge og ikke noget der altid vil være til stede.

Klassediagram



1

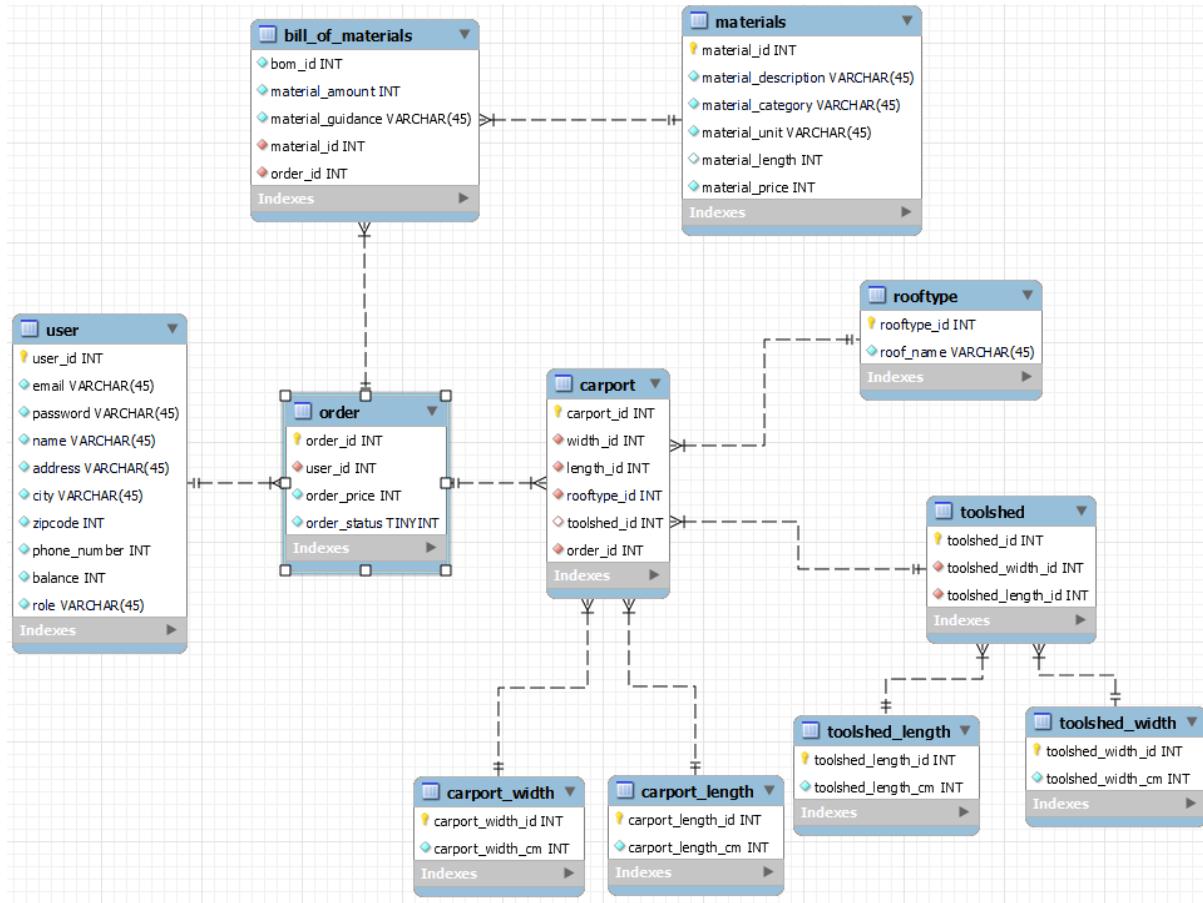
Vi har designet vores klassediagram ved hjælp af PlantUML. Formålet med at lave diagrammet er, at man skal få en ide om hvordan de forskellige klasser er forbundet med hinanden.

Da vores system består af mange forskellige Servlets, Mappers og java klasser har vi valgt at undlade nogle enkelte klasser i diagrammet for at det ikke skulle blive mere voldsomt end det allerede er i forvejen.

¹ <https://i.imgur.com/blecy1O.png>

Åbn billedet via imgur linket ellers kan det også findes i vores Github repository under documentation.

EER diagram



I vores EER² diagram har vi tildelt et ID til hver tabel som er tabellens primary key. Alle disse primary keys undtagen bom_id i bill of materials er sat til at generere et automatisk ID. Grunden til, at vi ikke har sat den til at gøre det automatisk i bill of materials er, at det tillader os at have flere rækker med det samme ID. Bill of materials er vores stykliste og styklisten består af mange forskellige materialer. Lad os sige at den består af 15 materialer, så vil de få hver deres række i databasen med det samme bom_id og det samme order_id.

Ideen bag vores diagram er, at man har en bruger, som kan have en ordre tilknyttet til sig. Ordren består af en carport, hvor carporten består af en længde, bredde, tagtype og muligvis et redskabsrum. Det ses også i diagrammet, at toolshed_id i vores carport tabel ikke behøver at eksistere. I carport tabellen er der kun ID'er og det er fordi at selve dataen som fx carportens længde, den finder man i carport_length, hvor vi manuelt har indsat alle de forskellige længder som kan vælges. Det har også betydet, at vi har skulle bruge nogle inner og left joins, for at finde længder og bredder ud fra ID'et i carporten. Et inner join er hvor man forbinder en tabel med en anden, for at få vist nogle flere relevante kolonner baseret på relationen mellem tabellerne. Til sidst, har ordren også tilknyttet en bill of materials til sig, som henter de enkelte informationer om det pågældende materiale fra vores materials tabel.

² <https://imgur.com/a/g1gda4w>

Constraints

Vi har anvendt forskellige constraints i de enkelte tabeller, men størstedelen er sat til at værdien ikke må være null. Der er enkelte steder hvor det giver mening, at en værdi er null. Det er fx ikke alle materialer hvor længden er relevant, det kunne være en skrue, eller som nævnt tidligere, så behøver en carport ikke nødvendigvis, at have et redskabsrum. I tabellen user har vi også benyttet os af at bruge “default”, så når man opretter en bruger i systemet, så vil den altid have rollen “customer”. Balance har vi også sat til en default på 50.000, men vi har faktisk ikke brugt balance til noget på nuværende tidspunkt. Ideen med balance var, at man skulle kunne betale for carporten via hjemmesiden og derefter få forespørgslen godkendt.

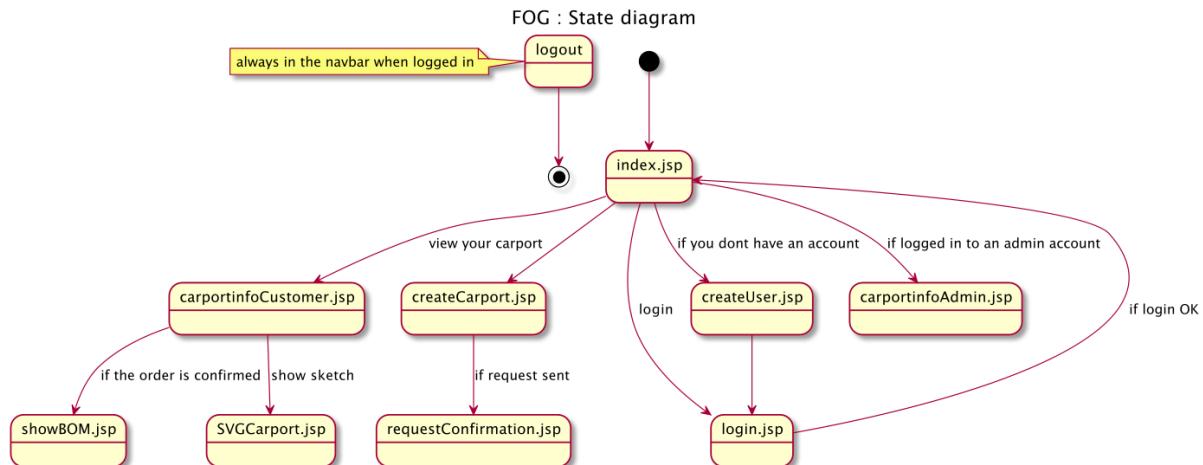
Relationer

Vores tabeller er forbundet med relationen “en til mange”. Vi har forbundet vores fremmednøgler med vores primary keys. Hvis vi tager carport som eksempel, så har den flere fremmednøgler som fx width_id og den er forbundet med tabellen carport_width, hvor den henter et unikt ID på længden af carporten.

Normalform

Vi har prøvet at holde størstedelen af databasen på 3. normalform, men der er nogle steder vi ikke har gjort det bevidst for at forsimple arbejdet. I user tabellen kunne vi have haft zipcode stående for sig selv og så have lavet en anden tabel med zipcode og city for at opnå 3. normalform, men det ville kræve en masse joins, hvis vi skulle finde alle postnumre. I materials tabellen, kunne vi også have delt den op i 2 tabeller for at opnå en højere normalform, da der er flere typer materialer fx træ eller skruer. Eller hvis man skulle have forskellige varianter af et materialer, fx et stykke træ hvor længden er 3 meter og 5 meter.

Navigationsdiagram



I vores navigationsdiagram viser vi hvordan, vi har tænkt at en bruger eller admin skal kunne bevæge sig rundt i systemet. Diagrammet indeholder kun de visuelle sider man kommer ind på i systemet. State diagrammet giver et godt overblik over for kunden, så man altid ved hvor man er og hvad man skal gøre for at komme videre. Det er isært et nyttigt væktøj, hvis man har med større systemer at gøre.

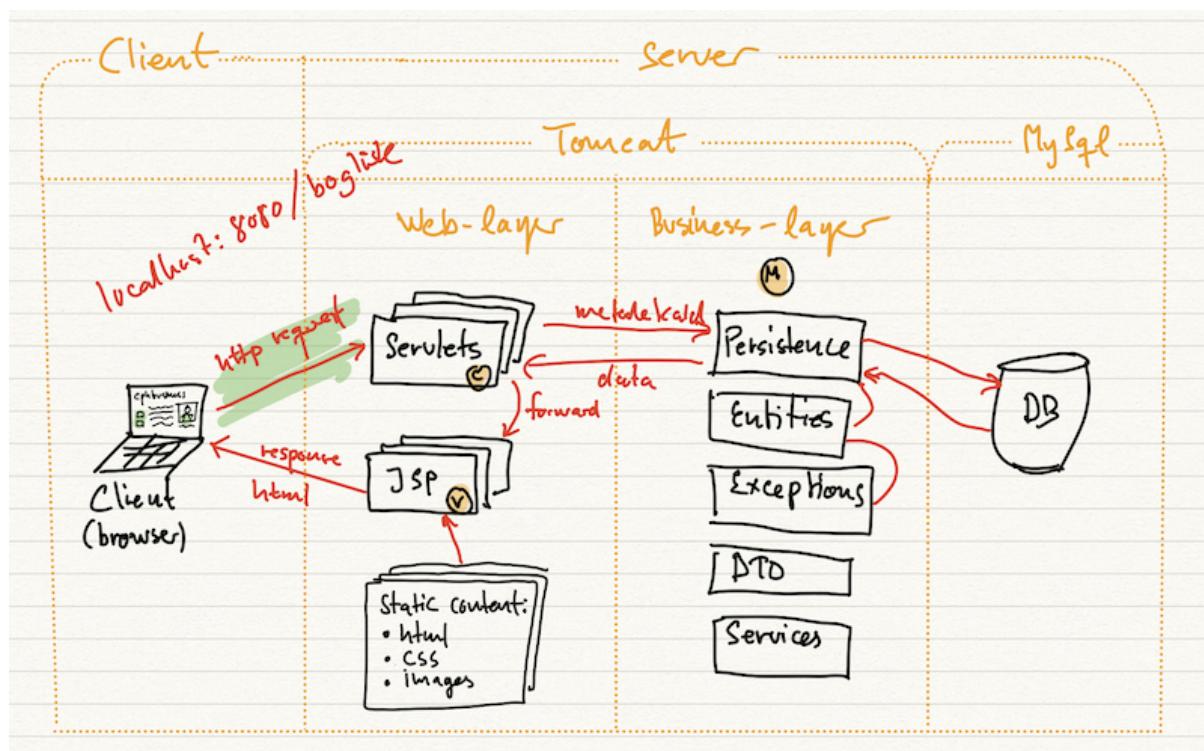
Hvis du ikke er logget ind på systemet kan du lave en bruger eller logge ind med en eksisterende bruger, først derefter kan du tilgå systemet. Når du er logget på systemet med en user rolle kan du gå ind og bestille en carport samt se den carport du har bestilt. Hvis du er logget ind med en admin rolle kan du tilgå en speciel admin side, hvor du kan se alle ordrer i systemet samt administrere dem. På alle siderne benytter vi os af en navigation bar, så det er nemt og overskueligt hvordan man skal bevæge sig rundt i systemet.

Arkitektur

I vores system har vi anvendt arkitekturen MVC, som er en forkortelse for Model-View-Controller. For at demonstrere arkitekturen er der vist et billede nedenfor af, hvordan den er bygget op på. Grundlæggende set er der en server og en client. Vi bruger Tomcat til at hoste vores server og denne er også forbundet med vores MySQL database.

Vores system er bygget op således, at der er et business-layer og et web-layer. I business-layeret har vi blandt andet "Persistence" mappen, som omhandler databasen. Det er fx her vi indhenter data fra databasen og indsætter data. I mappen "Entities" har vi vores java objekter. Man kan sige, at disse objekter næsten er forbundet med databasen, da vi har et java objekt ud fra hver tabel der er i databasen. Når vi skal indsætte eller læse data fra databasen gør vi brug af exceptions, for at gøre os opmærksomme på fejl. Et eksempel kunne være når vi skal indsætte en carport i databasen, hvis der er gået noget galt, så vil vi få fejlmeddelelsen "The carport couldn't be inserted into the database". Eller hvis vi ønsker at indlæse en carport og der sker en fejl, så får vi fejlmeddelelsen "Couldn't find carport". Vi har også en service mappe, hvor vi har en klasse der beregner stykklisten og en anden som vi bruger til at lave vores SVG med. I web-layeret bruger vi servlets til at kalde på de metoder som der er i business-layeret og derefter bliver man sendt videre til en jsp side, som består af html, css og i nogle tilfælde billeder.

Client delen er den del, hvor man tilgår serveren via sin browser. Man sender et http request til servletten, som derefter sender en videre til en jsp side, og til sidst sender et http response tilbage til browseren.



Særlige forhold

Session

I systemet bruger vi HttpSession til at gemme det meste af vores data, så vi kan få adgang til det via jsp siderne. Det bliver brugt i vores Servlets, og vi starter altid med at instantiere det i vores Servlets ved at sige HttpSession session.

```
@WebServlet(name = "carportAdmin", value = "/carportAdmin")
public class CarportAdmin extends HttpServlet {
    private HttpSession session;
```

Det er et objekt som vi henter og det har en masse nyttefulde metoder. Det som vi primært bruger i vores servlets er “sessions = request.getSession();”, som henter den nuværende session og “session.setAttribute”, som bruges til at gemme data, der kan tilgås i jsp-siderne. Efter man har gemt fx en arraylist i en session, kan man bruge den i en jsp side ved hjælp af sessionScope. Et eksempel på at man bruger en arraylist på en af jsp-siderne via sessionScope kan se ud som følgende:

```
<c:forEach items="${sessionScope.bomSpecification}" var="items">
    <tr>
        <td> <c:out value="${items.materialDescription}" /></td>
        <td> <c:out value="${items.materialLength}" /></td>
        <td> <c:out value="${items.materialAmount}" /></td>
        <td> <c:out value="${items.materialUnit}" /></td>
        <td> <c:out value="${items.materialGuidance}" /></td>

    </tr>
</c:forEach>
```

I eksemplet her bruger vi et for each loop til at gå en arraylist igennem og på den måde kan vi printe alle de relevante værdier ud i en tabel.

Exceptions

I vores system har vi brugt flere forskellige exceptions, så vi nemt har kunne se hvad der var galt, når der opstod problemer. Når vi starter vores system kan vi se en server log, som løbende bliver opdateret. Hvis der via hjemmesiden opstår en fejl, som bliver fanget af en af vores try catch metoder, så får vi en meddelelse om hvad der er galt. Vi har primært brugt DatabaseException, som er en klasse vi fik udleveret i startkoden, der nedarver javas Exception objekt.

```
public class DatabaseException extends Exception {  
    public DatabaseException(String message) {  
        super(message);  
        Logger.getLogger("web").log(Level.SEVERE, message);  
    }  
    public DatabaseException(Exception ex, String message) {  
        super(message);  
        Logger.getLogger("web").log(Level.SEVERE, message, ex.getMessage());  
    }  
}
```

Når vi bruger vores DatabaseException, kan vi tilknytte en besked, som bliver sendt i server loggen. Det vil vi vise i eksemplet nedenunder:

```
    } else {  
        throw new DatabaseException("Wrong username or password");  
    }  
}  
} catch (SQLException ex) {  
    throw new DatabaseException(ex, "Error logging in. Something went wrong with the database");  
}
```

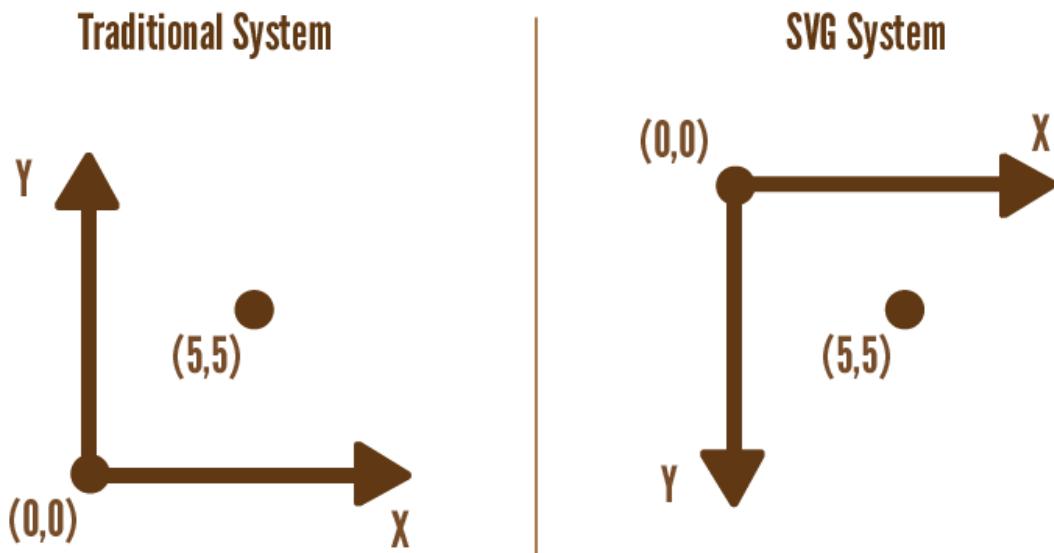
Denne kode er fra vores UserMapper klasse, som vi bruger når man skal logge ind eller oprette en bruger på hjemmesiden. Vi har brugt try catch og det vil sige at systemet prøver at køre et stykke kode og hvis det eventuelt fanger en fejl, så sender den en fejlmeddelelse.

Stykliste

Vi udregner styklisten med nogle simple udregninger. Hver af disse udregninger er sat til en unik funktion hvor de bliver kaldt, når man har brug for dem. Metoderne tager imod den højde og bredde som er blevet udvalgt af kunden. Vi bruger disse værdier til udregne mængden af materialer som skal leveres til kunden. Vores materiale dimensioner er konstante, hvilket gør det lettere at udregne styklisten, samt fjerner det ekstra arbejdet fra Fog, da det antages at kunden godt selv kan skærer materialerne til. Vi sender så vores stykliste til databasen, hvor den bliver gemt. I databasen bliver styklisten også linket sammen med kundens profil, så det kun er den bestemte kunde, der kan tjekke styklisten. Styklisten er ikke direkte tilgængelig for kunden da det kunne skade forretningen, så den bliver kun vist hvis en admin har godkendt ordren. Det giver også mulighed for, at en sælger kan ringe til en kunde, hvis de ser en carport der ikke giver mening.

SVG

Scalable Vector Graphics er et HTML tag som man anvender til få noget visuelt op på en side. For at vise noget skal man have en viewBox. En viewBox virker som et koordinatsystem, dog er y-aksen ikke op men ned. Du tegner altså inde i viewBoxen ved at angive x og y koordinater. Det smarte ved vektorgrafik er, at man kan skalere det uden billede mister kvalitet og bliver grynet som fx andre filformater som png.



Eksempel på en SVG viewBox vs traditionelt koordinatsystem.

Til at tegne SVG anvender vi en SVG klasse, hvori der er lavet string templates og metoder til at tilføje/append disse templates til vores Stringbuilder med vores egne koordinater. På den måde tilføjer vi forskellige dele af tegningen fx rectTemplate og arrowTemplate med metoder som addRect() og addArrow().

```
rectTemplate = "<rect x=\"%d\" y=\"%d\" height=\"%d\" width=\"%d\" style=\"stroke:#000000; fill: #ffffff\" />";
```

I metoderne anvender vi String.Format til at ændre værdierne %d (Int) i vores string templates, dette gør vores templates dynamiske. Ved at kalde disse metoder i SVGServlet kan vi altså generere en carport tegning med de rigtige mål og i de rigtige størrelsesforhold.

```

//lodret spær
for (int x = 0; x < spærAmount; x++)
{
    if (x == 0)// første spær
    {
        carportSVG.addRect( x: 0, y: 0, carportWidth, width: 4);
    } else if (x == spærAmount - 1)// sidste spær
    {
        carportSVG.addRect( x: carportLength - 4, y: 0, carportWidth, width: 4);
    } else {
        carportSVG.addRect( x: 55 * x - 1, y: 0, carportWidth, width: 4);
    }
}

```

Billedet ovenfor viser måden vi tilføjer de lodrette spær til vores tegning. Først bestemmer vi mængden af spær ved at sige længden af carporten/pladsen imellem spærene. Derefter kører vi et for loop til at sætte alle spærene.

Til at vise vores tegning anvender vi viewBox. Vi har 2 forskellige viewBoxes en indre og en ydre. Den indre viewBox bruger vi til at tegne carporten set ovenfra med stolper, spær osv. Den ydre bruger vi til at sætte mål på tegningen i form af pile med tal.

Udvalgte kodeeksempler

Stykliste

```
public BillOfMaterials calculateStolper() throws DatabaseException {  
  
    String prerequisiteOne = "97x97 mm. trykimp. Stolpe";  
    int length = clength;  
    int materialLength = 300;  
    int amountOfStolper = 2;  
    int spaceFromBeginningToFirstStolpe;  
    while (length > 310) {  
        amountOfStolper++;  
        length = length - 310;  
    }  
    amountOfStolper = amountOfStolper * 2;  
    spaceFromBeginningToFirstStolpe = length / 2;  
    for (Materials materials : materialList) {  
        if (materials.getMaterialDescription().equals(prerequisiteOne) && materials.getMaterialLength() == materialLength) {  
            BillOfMaterials billOfMaterials = new BillOfMaterials(bom_id,amountOfStolper, materials.getMaterialID(), materialG  
        }  
    }  
    return null;  
}
```

Dette er et lille stykke af den kode vi bruger til at skabe en stykliste. I eksemplet kan man se hvordan vi beregner, hvor mange stolper der skal bruges til carporten.

Vi lavede mange forskellige metoder der udregnede hvor mange antal af de enkelte materialer man havde brug for. Måden vi fandt frem til mængden af materialerne, er ved tage den længde og bredde kunden havde givet deres carport, og så satte vi dem igennem nogle udregninger vi var kommet frem til. Vi brugte en stykliste som Fog har lavet, som vores facit.

CRUD metoder

Vi har brugt en masse CRUD metoder i vores system. CRUD står for create, read, update og delete. Det har været relevant, når vi skulle styre vores database.

```
String sql = "INSERT INTO carport (`width_id`, `length_id`, `rooftype_id`, `toolshed_id`, `order_id` ) " +  
    " VALUES (?, ?, ?, (SELECT MAX(toolshed_id) as toolshed_id FROM toolshed), ?)" ;
```

I eksemplet ovenover opretter vi en carport med et redskabsrum i databasen, men da vi både havde en tabel for carporten og redskabsrummet, måtte vi finde en måde hvorpå vi kunne finde redskabsrummets ID på. I vores Servlet hvor man opretter en carport, har vi kodet, at man først kører metoden med den følgende sql sætning:

```
String sql = "insert into toolshed (toolshed_width_id, toolshed_length_id) values (?,?)";
```

Sætningen indsætter et redskabsrum i databasen. Derefter kører vi metoden som opretter carporten, hvor den så tager det nyeste redskabsrum ved at sige “SELECT MAX(toolshed_id) as toolshed_id FROM toolshed”.

Vi har flere metoder hvor man opdaterer databasen med fx ordre pris eller status.

```
public void updateOrderPrice(int order_id, int order_price) throws DatabaseException {  
    Logger.getLogger("web").log(Level.INFO, msg: "");  
    String sql = "UPDATE fogcarport.order SET order_price =" + order_price + " WHERE order_id =" + order_id;  
  
    try (Connection connection = connectionPool.getConnection()) {  
        try (PreparedStatement ps = connection.prepareStatement(sql)) {  
            ps.executeUpdate();  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

I eksemplet ovenover ser man vores metode som admins har adgang til. Den anvendes til at opdatere prisen på en ordre. Vi har udviklet det således, at man skal bruge et ordre id og en ny pris for at køre den.

Vi har også en metode til at slette en ordre, hvor metoden kræver et specifikt ordre id. Det vigtige her, er at man husker at slette den tilhørende carport og stykliste samtidig. Det har vi gjort på følgende måde:

```
public void deleteOrder(int order_id) throws DatabaseException {  
  
    Logger.getLogger("web").log(Level.INFO, msg: "");  
    String sqlone = "delete from `bill_of_materials` where order_id = ?;";  
    String sqltwo = "delete from `carport` where order_id = ?;";  
    String sqlthree = "delete from `order` where order_id = ?;";
```

Rækkefølgen af sql sætningerne er også vigtigt, da en stykliste eller carport ikke kan eksistere uden en ordre tilknyttet til dem.

Til sidst vil vi vise den måde vi henter carportens længde, tagtype og redskabsrummets længde og bredde. Det gør vi ved brug af følgende kode:

```

String sql = "SELECT * FROM carport c\n" +
    "inner join carport_length cl on c.length_id = cl.carport_length_id\n" +
    "inner join carport_width cw on c.width_id = cw.carport_width_id\n" +
    "inner join rooftype rt on c.rooftype_id = rt.rooftype_id\n" +
    "left join toolshed ts on c.toolshed_id = ts.toolshed_id\n" +
    "left join toolshed_length tl on ts.toolshed_length_id = tl.toolshed_length_id\n" +
    "left join toolshed_width tw on ts.toolshed_width_id = tw.toolshed_width_id\n" +
    "inner join fogcarport.order o on c.order_id = o.order_id\n" +
    "WHERE user_id =" + user_id;
    . . .

```

Som man kan se, så vælger man først alle kolonner i carport tabellen, hvorefter man laver et join af de andre tabeller, for fx at finde længden eller navnet på taget.

Status på implementering

Nedenfor er en tabel med hver user story og status på implementeringen af den givne user story. I vores system har vi bygget et fundament, som er nemt at bygge ovenpå. Som man kan se nedenunder, så mangler vi primært at lave en mere avanceret SVG skitse og at videreudvikle på vores beregningsmetode af styklisten. Vi har en klasse i vores system, der hedder CalculatorService, hvor vi nemt kan smide flere beregninger ind til fx beslag og skruer. Derefter vil det automatisk blive lagt oven i styklisten som i bund og grund er en arraylist, hvor de enkelte elementer bliver gemt i databasen under samme id. I forhold til SVG, når vi skal videreudvikle den del, så har vi en klasse der hedder SVG, hvor vi har forskellige metoder til fx at lave en rektangel eller en linje og der skulle vi så tilføje flere metoder for at få en mere avanceret skitse.

Vi har selvfølgelig også arbejdet med andre ting som ikke er beskrevet direkte i vores user stories, som fx jsp, css, diagrammer og modeller. Vi har fokuseret meget på backend delen og kunne måske godt have ønsket at style vores jsp sider lidt mere med css. Det er også en ting, som der forhåbentligt vil være lidt tid til inden eksamen.

User story	Status
US1	Fuldt implementeret, man kan både oprette en bruger og logge ind. Email, kodeord osv. bliver gemt i databasen, så man kan logge ind med de samme værdier igen senere.
US2	Fuldt implementeret, når man kommer til siden hvor man “bygger” sin carport, så indlæser den de forskellige længder, bredder og tagtyper fra databasen, som så bliver vist i vores drop down menuer.
US3	Fuldt implementeret, som kunde kan man se alle ens forespørgsler i en tabel, hvor man kan se carportens længde, bredde, tagtype og redskabsrummets længde og bredde. Desuden kan man også se status på forespørgslen og prisen. Og hvis den er godkendt via en admin, kan man også tilgå styklisten.

US4	Indtil videre kan man se en 2d skitse af carporten som er udviklet med SVG under kundens forespørgsler. Man ser carporten oppefra med spær, stolper, brædder og mål, men skitsen kunne forbedres ved at man også så redskabsrummet og afstanden mellem spær i mål. Derudover ville vi også gerne have haft en skitse, hvor man ser carporten fra siden. Dette er noget vi prøver at få løst inden eksamen.
US5-A	Fuldt implementeret, når man opretter en carport i systemet, så genererer den automatisk en stykliste, som bliver vist i en tabel. Den er dog først synlig for kunden, når der har været en admin inde og godkende forespørgslen.
US5-B	Ikke implementeret, det er også en af de user stories, som vi kommer til at prøve at løse inden eksamen, så vores stykliste tager højde for redskabsrummet og at der også bliver beregnet hvor mange beslag og skruer der skal bruges.
US6	Fuldt implementeret, både bruger og admin kan slette forespørgsler, og dataene til den tilsvarende forespørgsel bliver også slettet fra databasen.
US7	Fuldt implementeret, som admin kan man ændre status og prisen på forespørgslen og når det ændres på admin siden, så bliver det også ændret på kundens side med det samme.

Test

Vi har testet systemet på flere forskellige måder, for at se at det virker og ikke crasher. Det har vi gjort primært via manuelle tests og til dels JUnit. Det var vigtigt for os, at koden virkede før den blev en del af vores main branch.

Manuelle tests

Med manuelle tests, mener vi, at vi har testet nogle at metoderne ved at udskrive dem i konsollen via "System.out.print()" eller kigget efter fejlmeddelelser i server loggen. Et eksempel kunne være, da vi skulle beregne prisen på ordren ud fra materialernes pris. Der brugte vi følgende SQL sætning:

```
String sql = "SELECT * FROM bill_of_materials b\n" +
    "inner join materials m on b.material_id = m.material_id\n" +
    "WHERE order_id =" + order_id;
```

Det der sker er, at den først læser alt data fra vores bill of materials tabel ud fra et specifikt ordre id og derefter laver vi et inner join med materials for at finde priserne på materialerne. Derefter tog vi og smed materiale mængden og prisen ind i en arraylist og for at finde den samlede pris måtte vi lave et for each loop, hvor man gik hvert materiale igennem og samtidig lagde mængden sammen med prisen. Denne metode drillede en smule, da det var vigtigt, at for each loopet var placeret rigtigt, men ved at kalde på metoden i en main klasse, kunne vi bruge "System.out.print()" til at se resultatet.

JUnit

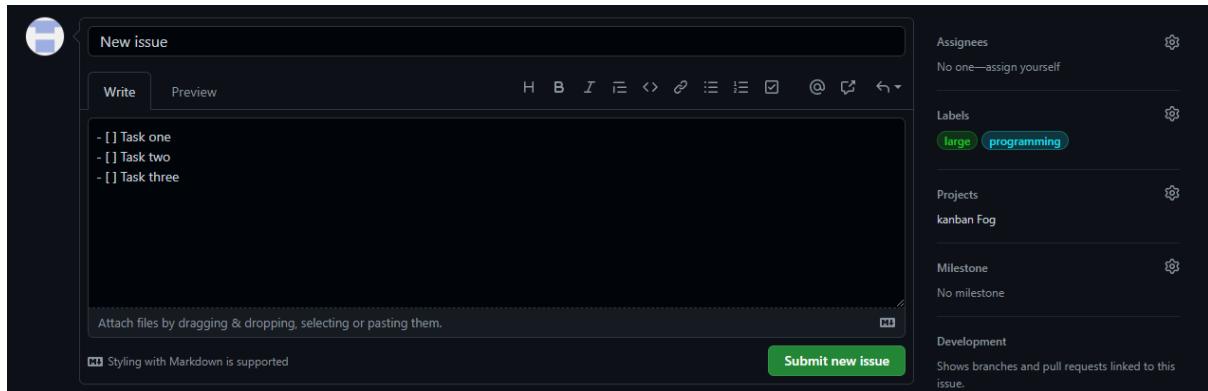
Vi har ikke været særlig gode til at integrere unit tests i vores system, men inden eksamen vil vi meget gerne have fået at lave nogle tests til vores styklister beregner, “OrderMapper” og andre klasse og metoder, som er relevante at teste. Det er noget, som vi kommer til at arbejde på i en anden branch end main branchen, og så har mulighed for at vise til eksamen.

Arbejdsprocessen faktuelt

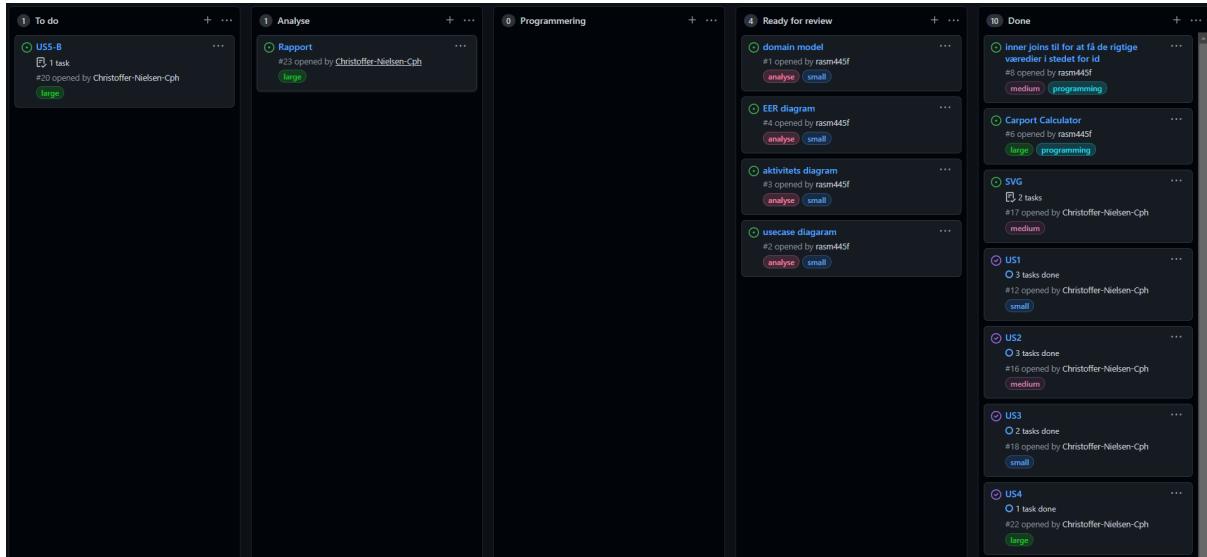
Kanban og Git issues

Under projektet mødtes vi hver dag på skolen, så der var mulighed for at spørge hinanden om hjælp, men også få vejledning fra tutorerne, Nikolaj og Jon. Derudover har vi også lavet en discord server i gruppen. Discord er en gratis tjeneste, hvor man kan oprette en server og har mulighed for at sidde i tekst kanaler og voice kanaler. På den måde har vi også kunne holde kontakten når vi ikke har været på skolen.

Den første uge af projektet startede vi med at forstå opgaven og det var også her vi fik lavet vores user stories, domæne model og EER diagram. For at skabe et overblik over hvor langt vi var nået i processen, startede vi ud med at oprette nogle issues på Github og efterfølgende et Kanban board.



Det smarte med Github er at man kan dele sin kode derinde og at man kan tilknytte sine issues til sit repository og et Kanban board. Et issue er i bund og grund en opgave og som man kan se i billedet ovenover, så giver man opgaven et navn og så kan man lave en tjekliste med forskellige delopgaver. Derudover kan man også tildele den labels, som fx small, medium eller large afhængig af opgavens størrelse eller en kategori som programmering.



Gennem Github har vi oprettet et Kanban board³, hvor vi har mulighed for at inddrage de issues som vi har lavet. I Kanban boardet har vi 5 forskellige kolonner: To do, Analyse, Programmering, Ready for review og Done. Vi startede med at lægge vores issues ind under To do og så kom det over i analyse, hvor vi i gruppen har snakket om den enkelte ting. Hvis det omhandlede programmering, så kom den derover som det næste og når man mente at man var færdig med at kode en del så kunne den rykkes til review, hvor vi gennemgik koden sammen i gruppen. Når koden blev pushet til vores main branch i Git, så kunne den enkelte issue rykkes til done.

I forhold til Kanban så har vi efter den første uge udvalgt en Kanban Mester og vi har roteret så der var en ny i hver uge. Opgaven som Kanban Mester var at holde styr på vores Kanban board, vores issues og at skabe et bedre overblik for gruppen over hvor langt vi var nået. Den første uge var det Christoffer, så blev det Anders og til sidst Rasmus.

Under projektet har vi haft vejledning sammen med Jon en gang om ugen på ca. 30 minutter.

³ <https://github.com/rasm445f/Fog-Carport/projects/1>

Vejledning og faser

Til den første vejledning, var det begrænset hvor meget vi havde at snakke om, da vi kun lige var begyndt at kode. Men de andre vejledninger var en kæmpe hjælp for os, da vi sørgede for at skrive ned hvad vi havde brug for hjælp til, og så kunne Jon hjælpe med de problemstillinger.

For at beskrive faserne i projektet har vi delt dem op i 6 faser:

- Fase 1: Før første vejledning
- Fase 2: Fra første vejledning til d. 11 maj
- Fase 3: Fra d. 11 maj til d. 18 maj
- Fase 4: Fra d. 18 maj til d. 25 maj
- Fase 5: Fra d. 25 maj til afleveringen af opgaven.
- Fase 6: Fra afleveringsdatoen til eksamen.

I **Fase 1** brugte vi primært tid på at forstå opgaven og lave diagrammer, men vi nåede også at kode en smule, som fx at få opret bruger og login til at fungere sammen med databasen. Vi lavede også en side hvor man kunne bestille en carport, og her indlæste vi målene til carporten fra databasen.

I **Fase 2** blev der brugt meget energi på at lave vores mappers og crud metoder til databasen, og vi begyndte også at overveje hvordan man skulle beregne styklisten. Her blev der lavet nogle meget store inner joins, så det var rart med den vejledning vi fik.

I **Fase 3** blev der arbejdet på vores klasse CalculatorService som er den klasse, der udregner styklisten til carporten. Vi var også stadig i gang med at tilføje flere metoder til vores mappers, og undervejs fik vi også justeret vores EER diagram.

I **Fase 4** blev der stadig udviklet på vores stykliste, men vi gik også i gang med vores SVG tegning her. Det var også her, at vi lavede nogle metoder i vores mappers, som gør at man fx kan ændre prisen på ordren og status ved at trykke på en knap, som vælger den specifikke ordre. I slutningen af denne fase havde vi også et system, som var kørende, og så begyndte vi så småt at starte på rapportskrivning.

I **Fase 5** handlede det om at finpudse koden og skrive rapporten.

I **Fase 6** i denne fase har vi tænkt os at videreudvikle vores SVG og stykliste i en anden branch, så vi til eksamen kan vise noget af det som vi ønskede var lavet til afleveringsdatoen.

Arbejdsprocessen reflekteret

Arbejdsprocessen

Den første uge brugte vi til at aftale mødetidspunkter samt hvad vi forvetede af hinanden. Det var godt at få det aftalt inden projektet egentlig gik i gang. Vi aftalte at vi som udgangspunkt, altid mødtes på skolen og at vi startede ved omkring 8.30 tiden. De fleste dage blev vi på skolen til kl 14 eller lidt senere, men det var tit, at vi så arbejdede videre individuelt derhjemme og samlede op dagen efter.

Det gjorde at vi havde nogle strukturerede uger og at vi ikke følte at de var gået tabt. Vi holdt altid hinanden opdateret via discord med fx mødetider og hvilke lokaler vi skulle mødes i.

Kanban og evaluering

Vi startede som sagt med at bestemme en ny Kanban Mester til hver uge. Dette fungerede fint dog følte vi ikke at Kanbans potentiale blev udnyttet til dets fulde. Vi brugte meget tid på at forstå opgaven og blive enige om hvordan vores endelige system skulle fungere. Da vi kun var 3 i gruppen og mødte op fysisk på skolen hver dag, så var det ofte nemmere for os, at holde et lille statusmøde ud fra vores user stories og snakke om hvad planen for dagen og efterfølgende dag var. Det var dog rart med noget visuelt i form af Kanban boardet, da vi hurtigt kunne se hvor langt vi ca. var nået. Vi tror, at hvis vi havde været i en gruppe som havde bestået af 5 eller 6 personer, så ville Kanban have været vigtigere, da det kan være vanskeligere at kommunikere ordentligt når man er så mange.

Vi udnyttede primært vores evalueringsmøder med Jon til at se, hvor langt vi var kommet i processen og spørge om hjælp. Til vores evalueringsmøder, startede vi ofte med at vise vores kørende hjemmeside ved hjælp af skærmdeling over Zoom. Derefter snakkede vi om hvad vi havde lavet siden den sidste evaluering og hvad vores mål i den kommende fase var. Det var rart at holde disse møder, da vi fik en ide om hvorvidt vi var på rette spor eller ej. Til hvert møde snakkede vi om forskellige emner og de mest væsentlige har nok været inner joins, hvordan man skulle sætte beregneren af styklisten op og hjælp til SVG, da vi først sprang ud i SVG i midten af projektet.

Der var ingen store problemer med vejledning, vores møder med Jon var altid brugbare og han var en stor hjælp. Vejledning mens man var på skole var mindre effektivt, da man nu skulle vente på sin tur i stedet for at man havde rampelyset for sig selv. Men det fandt vi en anden løsning på, for det var tit at nogle af de andre studerende havde de samme problemer som os. Et eksempel kunne være, da vi stødte på et problem, hvor bogstaver som æ, ø og å ikke blev indsat rigtigt i databasen. Det forstod vi ikke, da vores database og connection pool var sat op med UTF-8, men vi fandt en løsning ved at spørge en anden gruppe. De andre studerende var altid behjælpelige og da mange af dem havde samme niveau som os, var det altid forståeligt.

User stories

Da vi havde skrevet vores user stories i den første uge, fandt vi hurtigt ud af, at der var nogle af vores user stories, som vi blev nødt til at lave lidt om så de var nemmere at bryde ned til issues og håndtere.

Mange af vores user stories kunne deles op i nogle mindre delopgaver, som vi også benyttede os af, da vi skulle oprette vores issues i Github. Vi følte, at det var lidt svært at putte et rigtigt estimat på vores user stories i starten af forløbet, altså i form af small, medium og large. Men til vores overraskelse, så passede de faktisk meget godt i sidste ende. De steder hvor tidsestimatet ikke har været spot on, har primært været når vi skulle lave diagrammer og modeller.

De user stories, som vi havde tildelt small, var hurtigt overstået og det skyldes nok, at vi forinden eksamensprojektet lige havde arbejdet med et lignende projekt. I det forrige projekt, skulle vi lave en hjemmeside med samme arkitektur, hvor man kunne bestille cupcakes med forskellige bottoms og toppings. Derudover skulle man også have en bruger på hjemmesiden, så vores user story 1 var noget vi har prøvet flere gange, der skulle bare gemmes lidt flere informationer omkring brugeren i dette projekt.

Hvis vi kigger tilbage på vores user stories, kunne vi måske have delt nogle af dem op i mindre bidder, men da vi kun var 3 i gruppen, så var det rart at arbejde på en user story, som havde lidt kød på sig. I starten af forløbet snakkede vi med Jon omkring vores user stories, som fortalte os, at vores user stories så fornuftige ud, men at det kunne være en god ide, at dele nogle af dem op i A og B. Det gjorde vi så med vores user story, som omhandlede styklisten, da det var vigtigt for os at lave noget der virkede i første omgang og så kunne man altid bygge videre på det senere.

Github

Den første uge satte vi et Github repository op, så vi havde et sted at dele koden og muligheden for at bruge issues og kanban boardet. Vi havde alle brugt Github før, så det var ikke noget problem at lave nye branches og merge koden når en feature var færdig. Hvis man kigger på vores repository kan man se, at der er mange branches. Vi skulle nok have været lidt mere specifik i navngivningen af nogle af vores branches og der er også nogle af dem, som næsten ikke blev brugt. En af grundene til, at der er mange branches er nok fordi, at vi har været bange for at ødelægge noget i vores main branch, og så til tider har oprettet en ny branch for at teste noget i den. Hvis vi skulle have gjort det anderledes, ville vi nok have lavet en main branch og en main test branch, så vi ikke pushede direkte til main i første omgang. Derudover kunne vi også have haft en branch navngivet efter vores navne, som var vores egen, hvor vi prøve nogle ting af. Og til sidst, så ville vi selvfølgelig have navngivet vores feature branches en smule mere detaljeret og specifikt.

Det fungerede godt med at bruge Git igennem IntelliJ, da det er meget simpelt, når man skal commit, push eller pull. Det bedste med Git igennem IntelliJ var, at man altid fik en lille varsel inden man gjorde noget, og at det var nemt at rollback til det forrige commit. Det var dog rart, at vi havde haft et kursus, som omhandlede Git, så vi forstod teorien bag de forskellige commands.