Fog Carport Projekt 2022

2/5-2022 til 31/5-2022

Klasse A (Gruppe 8)

Navn:	**Cph-email:**	**Github:**
Rehman Ahmed Habib	cph-rh255@cphbusiness.dk	https://github.com/notebrr
Oscar Tuff	cph-ot58@cphbusiness.dk	https://github.com/oTuff
Mark Lundgaard	cph-mj924@cphbusiness.dk	https://github.com/Ch0min
Gustav Bøgh	cph-gb84@cphbusiness.dk	https://github.com/GBoegh98

Demovideo: https://youtu.be/HvNXmA5NLvY

Pitch video: https://youtu.be/iYvM-8vymxE

Droplet: http://167.99.252.206:8080/carport/

Admin bruger: brugernavn: a@a.dk | **kode:** 1234

Test kunde–bruger: brugernavn: b@b.dk | **kode:** 1234

CphBusiness Academy
Carport Projekt
2/5-2022 til 31/5-2022

Indholdsfortegnelse

Indledning

Rapporten handler om at lave en dynamisk hjemmeside, hvor vi indfører koncepterne, servlet klasser og jsp sider. Vi har fået tildelt en opgave fra bestyrelsen indenfor Johannes Fogs carport afdeling. Ud fra en længere video skulle vi lave vores egne use-cases af, hvad vi mente hjemmesiden skulle indeholde og kunne udover de specifikke krav, som vi er blevet tildelt. Hjemmesiden skal kunne blive brugt af brugeren til at købe en brugerdefineret carport, og give en visuel repræsentation af en carport og en stykliste af de materialer man skal bruge til at bygge en carport.

Baggrund

Johannes Fog er en kvalificeret virksomhed som tilbyder rådgivning og distribuering af byggematerialer, samt byggeprojekter. Lige fra hvad der er godt at vide angående byg, til vejledning af valg i forhold til materialer. Johannes Fog består af et Bolig og Designhus og 9 Trælast og Byggecenter butikker. I trælasten har de træ og byggematerialer til alle slags opgaver. De har et bredt og dybt sortiment, både hvad angår størrelse, pris og kvalitet.

Krav stillet af kunden

Kunden skal kunne vælge en brugerdefineret carport ud fra selvvalgte mål, både længde og bredde. Kunden skal derefter modtage en 2D tegning af deres carport. Kunden skal kunne logge ind og se sin ordrestatus. Efter betaling modtager kunden en stykliste ud fra målet kunden har angivet og efterspurgt.

Virksomheden/Forretningsforståelse

SWOT:

Strengths: Trælast og designhus med mange tillægsydelser f.eks. udlejning af udstyr, tegninger mv. Bredt og dybt sortiment - carporte i forskellige prisklasser. Skiller sig ud ved at give meget personlig rådgivning og hjælp til kunden under hele processen. Mulighed for brugerdefinerede carporte.	**Weaknesses:** Kan være for fleksibelt og bredt til den almindelige kunde. Besværlighed for brugeren. Svært ved at udvikle sig - nye produkter. Gammeldags og ineffektiv arbejdsmetoder. Programmer er dyre at vedligeholde og udvikle. Forældet program og dele af hjemmesiden i forhold til konkurrenter.
Opportunities: Lovkrav og certificering ift. materialer, hvor de kan være foran konkurrenterne. Klimaforandringer der giver mere ekstremt vejr, hvilket gør at flere skal bruge carporte.	**Threats:** Konkurrenter: - Jørgen Andersen(J-A). - Silvan, Bauhaus mv. som kan lave billigere og nemmere produkter i takt med at færre kunder har færdigheder til at bygge selv og vil have det simpelt.

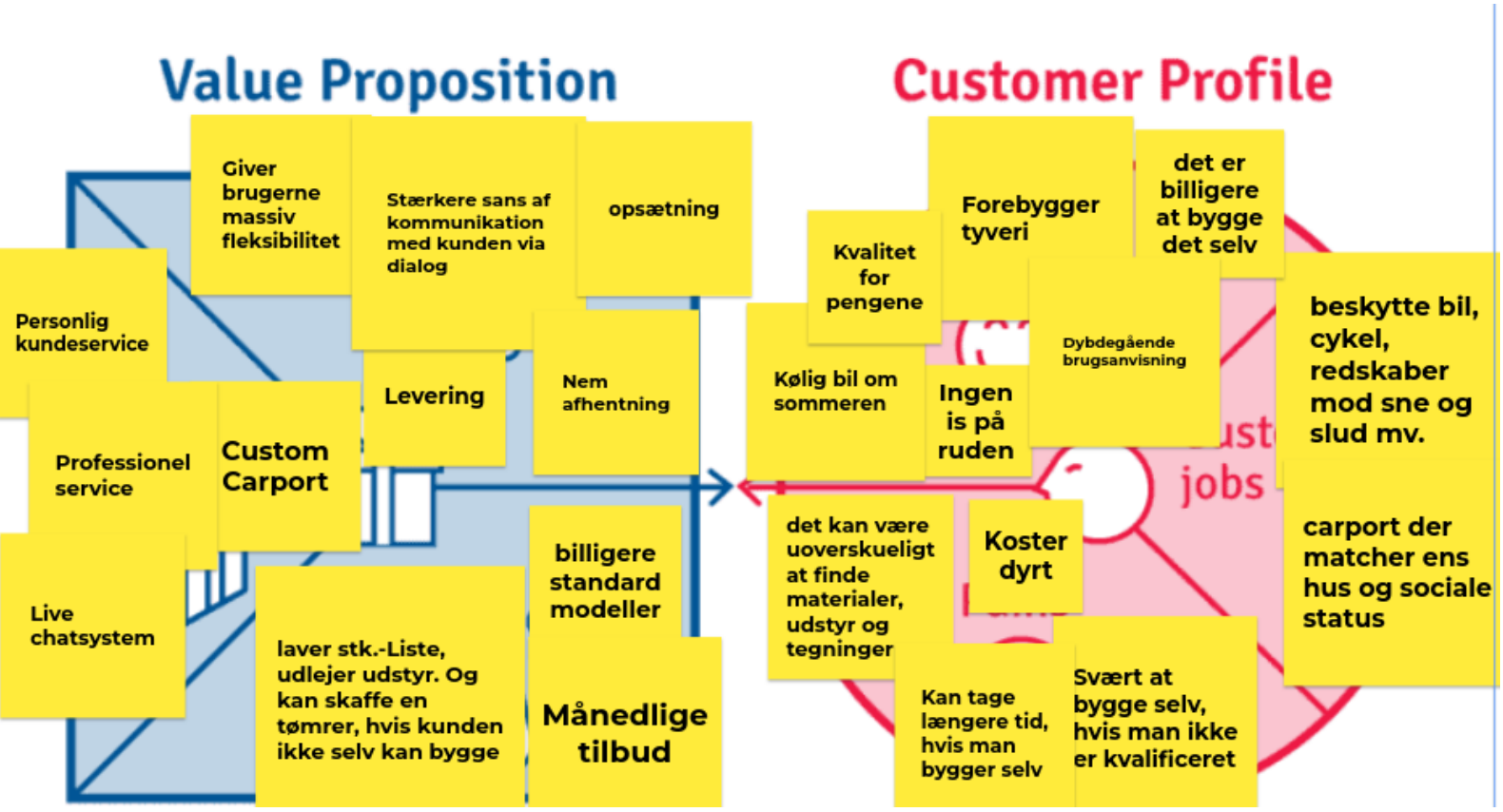
Vi har valgt at lave en TOWS matrix ud fra vores SWOT model for at få overblik over håndgribelige tiltag til at forbedre Fogs forretning.

TOWS	**Opportunities**	**Threats**
	Klimaforandringer	Konkurrenter (billigere mv.)
Strength	**S-O tiltag:**	**T-O tiltag:**
Trælast og designhus med dybt sortiment og tillægsydelser samt personlig service	Differentiere sig ved at have fokus på klima. f.eks. plante træ for hvert der fældes.	Fog adskiller sig fra konkurrenterne ved at være en helheds service og ikke blot trælast eller designhus.
Weakness	**W-O tiltag:**	**W-T tiltag:**
Forældet it infrastruktur og ineffektive arbejdsmetoder	…	Fog kan effektivisere deres it systemer og derved spare omkostninger på den lange bane samt differentiere sig med f.eks. en smart hjemmeside

Det har været svært at finde opportunities for Fog, da vi ikke ved så meget om branchen. Det virker også som en meget stillestående branche derfor er der ikke noget W-O tiltag.

Vi har valgt at fokusere på W-T tiltaget: Ved at lave en hjemmeside til at sælge custom carporte, som skal effektivisere arbejdsgange og forbedre Fogs konkurrencemæssige position.

Det er en mere defensiv forretningsstrategi som vi mener er den mest relevante, da vi som sagt ikke kunne finde mange opportunities. Ineffektiv it-infrastruktur er også en tydelig weakness, som man ikke kan have i en moderne verden. Derudover er det også det tiltag som opgaven lægger op til, da vi jo skal lave et it system.



Det har været meget overfladisk arbejde med krav til SWOT og VPC. Vi har jo ikke lavet nogle foreliggende analyser til SWOT, (værdikæde, bmc, omverdensmodel, five forces). Vi har også selv valgt at lave en TOWS til at lave løsningsforslag, da SWOT er et analyseværktøj til at give overblik over virksomheden, hvorimod TOWS tager SWOT modellen og laver en strategisk analyse og plan.

VPC er på samme måde lavet lidt i blinde, da den jo er tillæg til BMC, som vi ikke har lavet. Selvom modellerne er lidt overordnet og processen lidt manglende, så har de givet overblik over Fogs forretning, hvilket også er fokus.

Teknologi valg

Til at kode projektet har vi brugt følgende teknologier, som vi mener er fine værktøjer til de krav som vi er pålagt:

- **IDE: IntelliJ 2021.2.2**
    - Brugt til at kode i. Vi har valgt denne fordi den kan kobles op på git, hvorved vi kan have flere branches. Samtidig har den god intellisense.

- **Java 17**
    - Vi har brugt den nyeste version af Java fordi Java er et populært programmeringssprog og det er nemt at komme i gang med for os da vi i forvejen har erfaring med Java.

- **HTML5 / (JSP)**
    - Vi har brug HTML til at skabe skelettet for vores hjemmeside.

- **CSS 3**
    - CSS har vi brugt til at designe vores HTML skelet, så det føles som et mere fuldendt hjemmeside design.

- **JDBC v8.0.28**
    - JDBC til at koble sql-databasen til java

- **MySQL 8**
    - Mysql til at oprette database. specifikt med Workbench til at forward engineer en database ud fra et EER diagram.

- **Bootstrap v5.1**
    - Bootstrap har vi brugt som et design library, så det giver nogle flotte kanter. Det har hjulpet til design delen.

- **Maven 3.8.5**
    - Brugt til håndtering af dependencies, primært JDBC

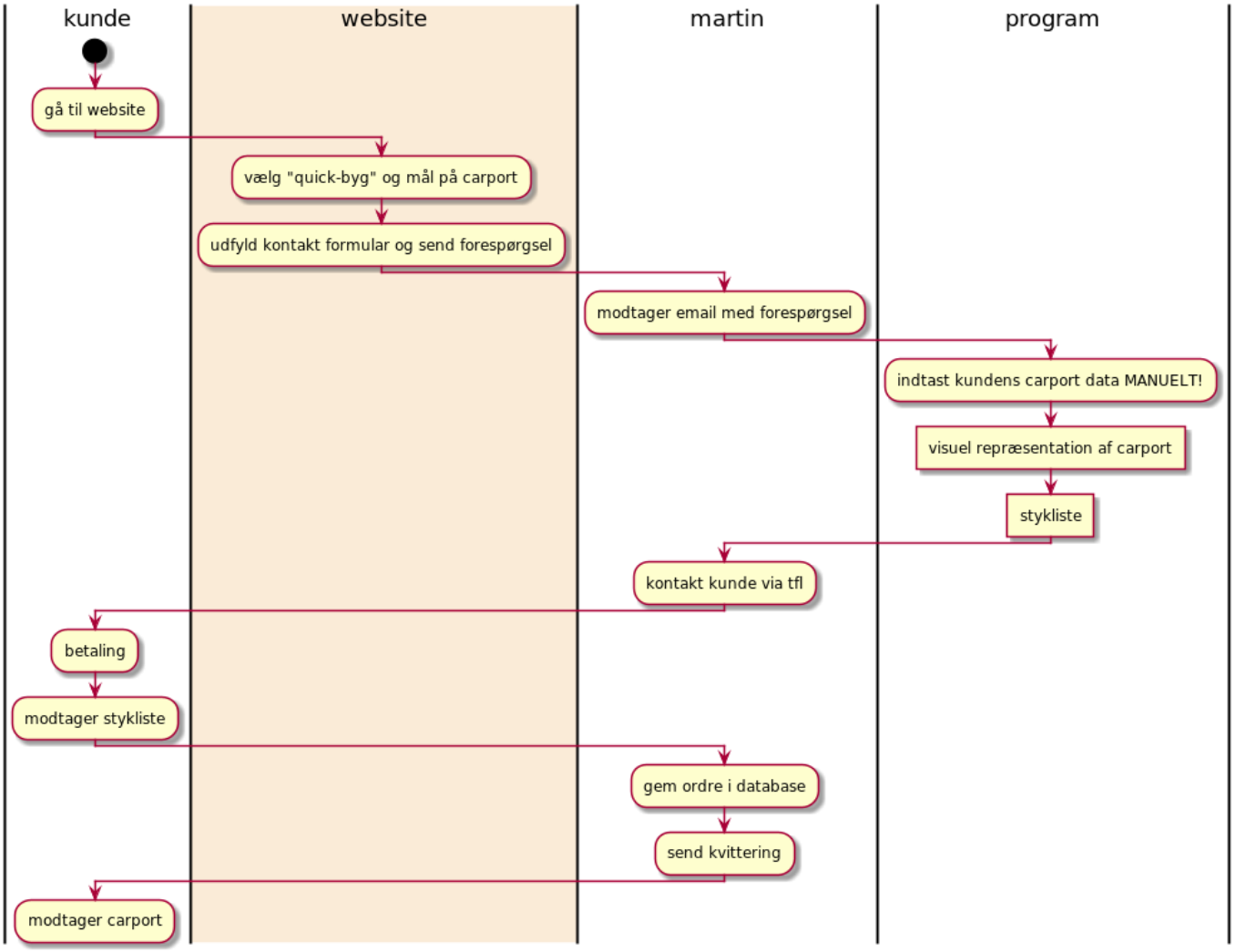
- **Git**
    - Git har vi brugt til at arbejde sammen på tværs, så vi hele tiden kunne opdatere hinanden på hvilket nye processer og opdateringer, der er blevet lavet til projektet. Vi har arbejdet i hver vores branches, samtidig med at vi har lavet branch til svære dele af opgaven.

- **Javascript**
    - Brugt til at praktiske front-end ting såsom på forsiden, for at sørge for at en form ikke er disabled længere hvis brugeren nu vil ændre sidens adresse.

Krav

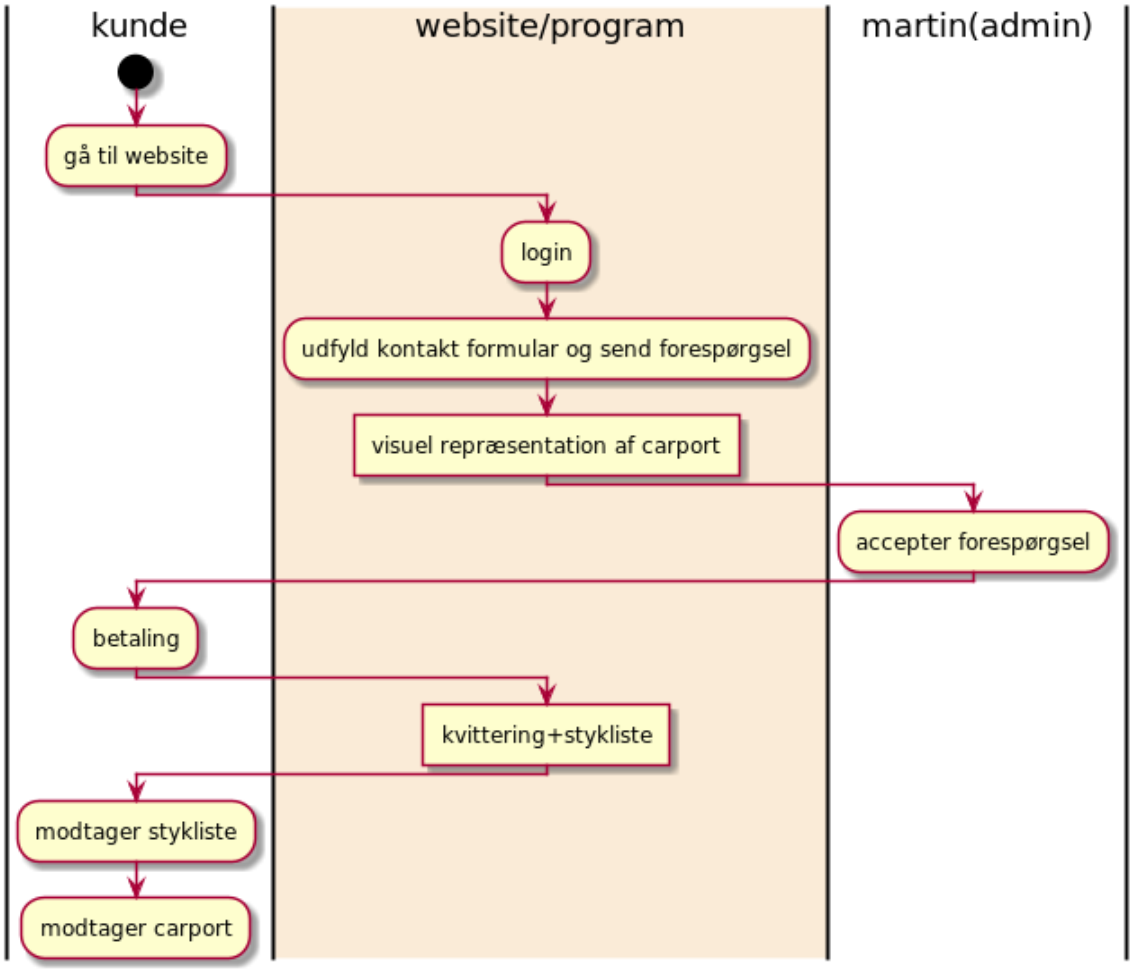
Arbejdsgange der skal IT-støttes

As-is:



Fogs arbejdsgang ift. custom carport ses i ovenstående aktivitets diagram. Det vigtigste at fokusere på er at Martin skal interagere med programmet. Som vi så i videoen med Martin, så var der meget redundant arbejde i form af at indtaste de samme data som kunden selv havde indtastet i programmet. Programmet var også gammelt og havde nogle små bugs, som gjorde det besværligt og ineffektivt at arbejde med.

To-be:



Arbejdsgange bliver effektiviseret ved at vores website/program selv udregner stykliste og laver tegning, uden at en admin skal indtaste de samme data som kunden har indtastet. Kunden får også allerede en grafisk repræsentation af carporten efter at indtaste længde og bredde.

Admin/Martin/sælgers job bliver blot at acceptere en forespørgelse ud fra en samtale med kunden, så der stadig er den menneskelige kontakt, der skaber værdi for kunden. Dette kunne i teorien også helt automatiseres, ved at kunden fik en stykliste efter at betale uden behov for at sende forespørgsel først, men fordi Fog udbyder en service ved at kigge ordren igennem først, har vi ikke gjort det på den måde.

Det er vigtigt at pointere at vores aktivitets diagrammer er forsimplet, for at give nemmere overblik, der mangler f.eks. trælast og leverandør som er en del af forløbet for at kunden kan modtage sin carport.

User Stories

Kanban board / Git Issues Link:

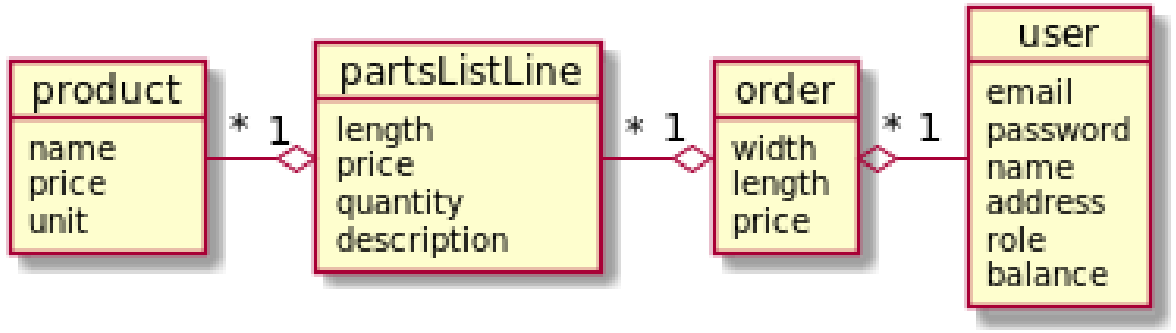
https://github.com/oTuff/carport/projects/1

- **US-1:** Som admin skal jeg kunne logge ind med det formål at kunne se og bearbejde ordrer fra kunder via et Adminpanel. (Small)
- **US-2:** Som kunde skal jeg kunne logge ind på min bruger med det formål at kunne bestille og betale/sende forespørgsel til en carport. (Small)
- **US-3:** Som kunde kan jeg oprette en konto/profil for at kunne skræddersy og købe en carport. (Small)
- **US-4:** Som kunde kan jeg vælge længde og bredde på carporten, se prisen og få en kvittering. (Large)
- **US-5:** Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder. (Medium)
- **US-6:** Som kunde eller administrator kan jeg logge på systemet med email og kodeord. Når jeg er logget på, skal jeg kunne se min email på hver side. (Small)
- **US-7:** Som admin kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt eller hvis en ordre er afvist. (Medium)
- **US-8:** Som kunde skal jeg være logget ind for at kunne skræddersy og bestille en ordre. (Small)
- **US-9:** Som admin kan jeg ændre i produktlisten, herunder varer og priser. (Medium)
- **US-10:** Som kunde kan jeg se en grafisk repræsentation af carporten. (Large)
- **US-11:** Som kunde skal jeg have mulighed for at tilføje skur til carporten. (Large)*(IKKE LAVET)*
- **US-12:** Som kunde efter betaling får jeg en kvittering samt stykliste på samme side. (Large)
- **US-13:** Som bruger kan jeg sende en forespørgsel om at få købt en carport under orders. (Medium)
- **US-14:** Som admin kan jeg acceptere eller afvise en forespørgsel fra en kunde. (Medium)

Estimat:

(Small): Maks. 1-3 timer. (Medium): Op til 1 dag. (Large): Maks. 1-3 dage.

Domæne model

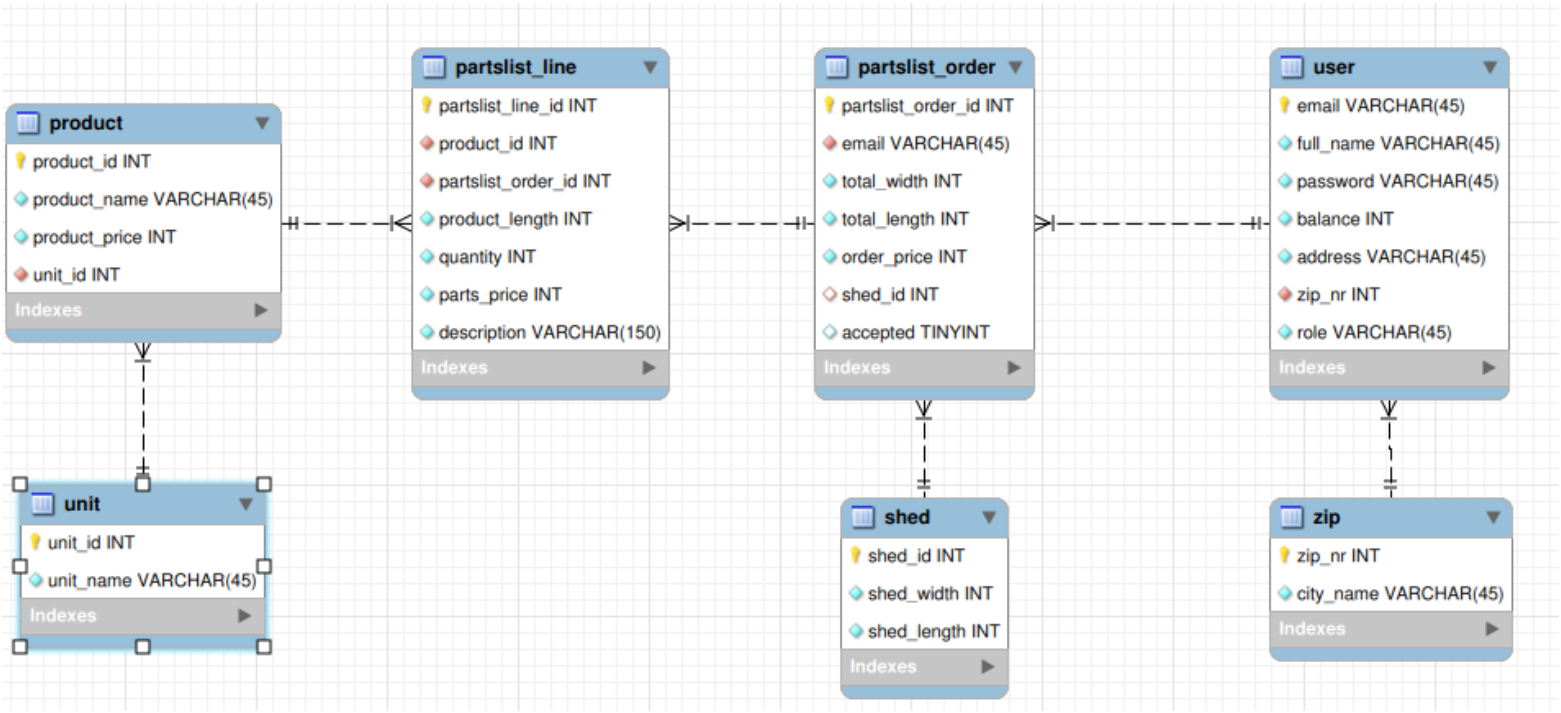


Vores domæne model giver et essentielt billede af, hvordan de forskellige entiteter i vores program taler og fungerer i forhold til hinanden. Den består af de vigtigste attributter og ingen metoder.

En “user” kan have flere “order”, så derfor har vi valgt at sætte en 1 til * relation i mellem de to entiter. Dermed kan en “order” indeholde flere “partsListLine’s”, og derfor har vi sat 1 til * relation fra partsListLine til order. Dernæst kan en partsListLine indeholde flere “product’s”, og derfor har vi også tilføjet en 1 til * relation mellem “product” og “partsListLine”

Overordnet set peger alle entiteter mod “order” entiteten, da den fungerer som den samlede skræddersyet stykliste.

EER diagram



Vi valgte at lave en separat tabel for “unit”, som vi tog ud fra “product” tabellen for at normalisere databasen til 3. normalform. Dette har vi også valgt at gøre med to andre tabeller, “shed” og “zip” tabellen. Dette gjorde det både mere overskueligt at arbejde med og mere fremtidssikret i forhold til nye ændringer i databasen.

Tabellen “partslist_line” udgør en linje/rubrik i styklisten og er tilknyttet til tabellen “partslist_order”, som fungerer som den overordnet ordre og den samlede skræddersyet stykliste. Derfor har “partslist_line” tabellen en fremmednøgle fra “partslist_order” tabellen kaldet “partslist_order_id”.

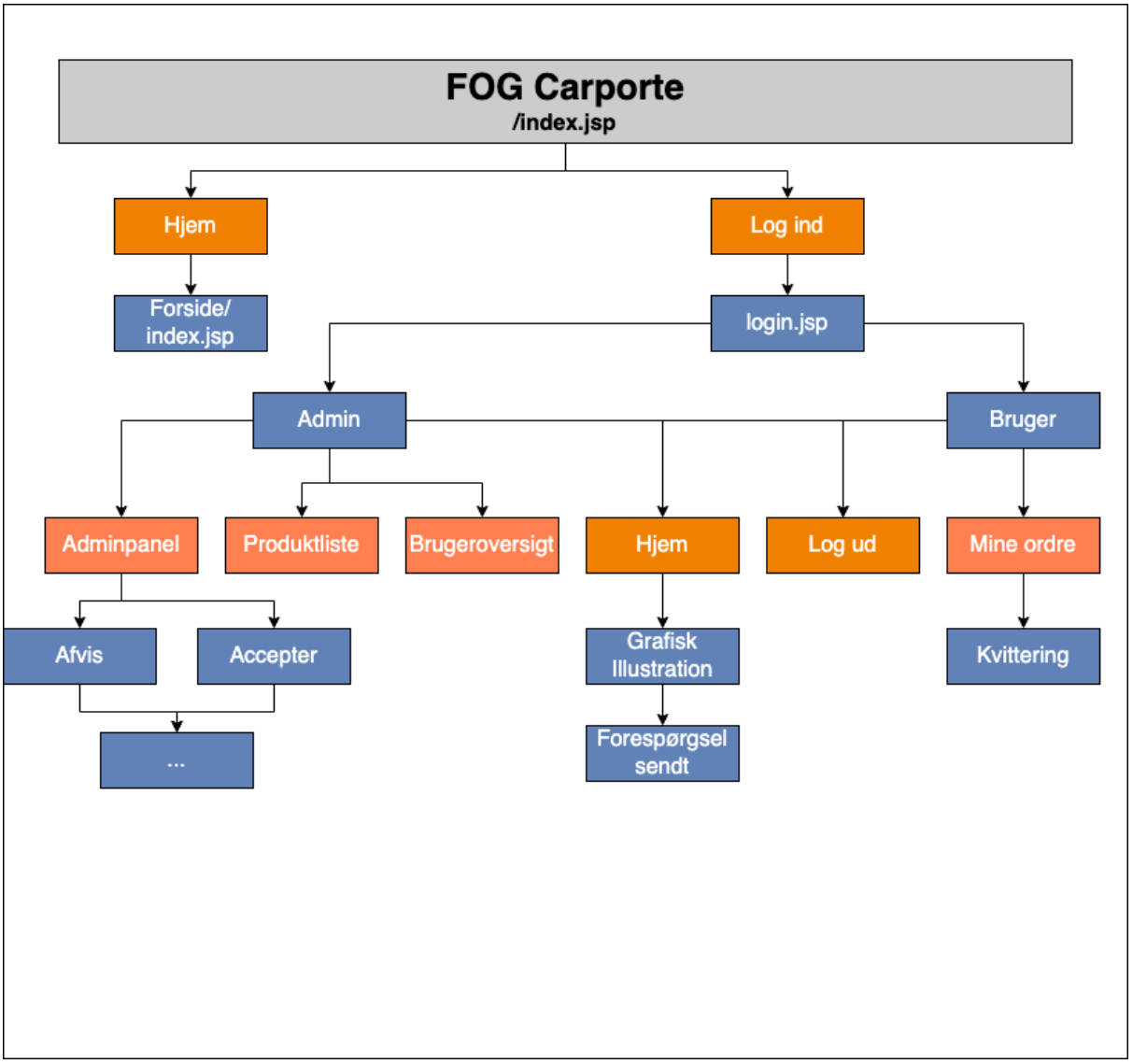
I tabellen “partslist_line” er “description” separat fra “product” tabellen, da det samme produkt kan blive brugt til forskellige dele af carporten. Derfor valgte vi at tage det ud af “product” tabellen.

Vi overvejet og burde nok have lavet en tabel og normaliseret en “description” tabel til 3. normalform, da vi i Java blot har hardcoded alle descriptions i Calculator klassen.

Vores “user” tabel er viderebygget fra startkoden af, og vi har valgt at bruge “email” som primary key, da det umiddelbart er unikt. Det kan dog være bedre at have et “user_id” i fremtiden, hvis f.eks. en bruger vælger at ændre sin email m.m.

Vores “shed” tabel havde vi planlagt at få implementeret, men tiden løb fra os. Den har en shed_width og shed_length samt et shed_id. Planen var at brugeren skulle have en mulighed for at tilføje et skur eller ej. Hvis så brugeren vælger at tilføje et skur på sin carport, så ville der være en boolean, som blev sat til true, og dermed kunne man vælge width og length på sin shed og få den implementeret.

Navigationsdiagram



Orange: Fælles nav-links. Laksefarvet: Specifikke nav-links Blå: Nye sider.

Vi har valgt at benytte os af en fælles navigation bar i toppen af alle vores sider, da vi føler at det giver en form for brugervenlighed til Brugeren. Det er nemmere at navigere rundt i, når man nemt kan se hvilket links man har i toppen: “Log ind / Log ud” og “Hjem” er de fælles nav-links, som både Brugeren og Admin har til fælles.

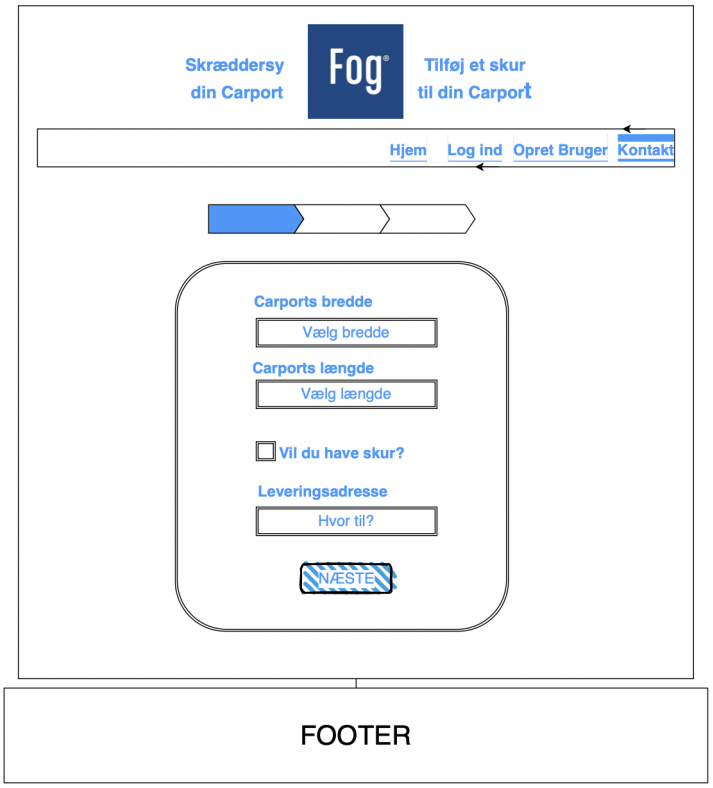
Når man er logget ind som Admin har man adgang til links “Brugeroversigt”, “Admin Panel” og “Produktliste”. Dette har man ikke adgang til som almindelig bruger, da disse nav-links udelukkende kun skal blive brugt af Admin.

Når man er logget ind som almindelig Bruger har man adgang til link "Mine ordre". Dette nav-link har Admin ikke, da Admin allerede har et overordnet Admin Panel, som viser alle brugeres aktive ordrer i stedet, hvori "Mine ordre" kun viser selve brugeres egne ordre.

Når man er logget ind som enten Admin eller bruger, så er nav-link "Log ind" ændret til et nyt link kaldt for "Log ud", herunder vil ens brugernavn/email også blive vist i toppen af højre hjørne.

Mockups

Det fulde Mockup findes under bilag.



Dette er forsiden på vores Mock-up i begyndelsen. Mock-upet har hjulpet os alle med at få en fælles forståelse for et slutprodukt, som vi alle kunne stræbe efter. Det har givet os en fælles visuel design template fra begyndelsen af, så alle var på lige fod med hinanden om det overordnet slutdesign. Det har også formindsket den forvirring der kan opstå, når flere personer prøver at lave et produkt uden at have en fælles idé af hvordan det skal se ud.

Valg af arkitektur

Vi har valgt at bruge MVC-arkitektur ud fra startkode skabelonen ligesom i cupcake projektet. Vi har udelukkende valgt denne arkitektur, da vi har arbejdet med den og har styr på den. Dette har givet os en fælles fremgangsmåde, da vi alle fire har leget med det før. Det blev vi ret hurtigt kollektivt enige om, da vi alle følte det var den rigtige løsning. Det gjorde det mere overskueligt at arbejde med og nemmere at vedligeholde kildekoden.

Særlige forhold

Når brugeren logger ind gemmes et User objekt i session. Når brugeren prøver at oprette en ordrer, og trykker "næste" på forsiden, gemmes et Order objekt også i sessionen. Exceptions håndteres enten ved et simpelt *try/catch* eller ved *throws*.

Validering af bruger inputtet på login-siden foregår ved at Java tjekker om der er en bruger med samme email og navn i MySQL via JDBC. På opret bruger-siden smides dataene i databasen uden yderligere validering. På login siden er der validering på front-end i HTML, som dog nemt kan forbipasseres. I databasen er koderne gemt uden nogen som helst form for hashing, hvilket gør systemet meget sårbart, fordi enhver med databasen har adgang til brugerens kode.

I databasen er der 2 brugertyper: admin og user. Via JDBC kan man hive data fra u*ser* databasen ned og derefter tjekke den enkelte brugers rolle. Den metode har vi brugt til at sørge for at admin-brugeren har adgang til både de ting user har adgang til, men også adgang til sider som kun admin skal have adgang til. Admin-brugeren har adgang til Adminpanel, Produktliste, og Brugeroversigt siden, som user ikke har adgang til.

Vi har håndteret fejl på flere forskellige måder. De fleste løses med try, catch statements og til tider throws exceptions. HTTP fejl 404 håndteres ved at vise en fejlside som fortæller brugeren at siden ikke virker.

Vi har brugt en minimum bredde og længde på 240 cm og og en maks bredde på 600 cm og maks længde på 780 cm, fordi det er de mål Fog har brugt på deres nuværende hjemmeside.

SVG klassen har mange attributter som primært er ints. Viewbox er dog den eneste attribut med en String, da den både indeholder ints og bogstaver, og bliver beskrevet under "headerTemplaten". Derudover indeholder klassen også nogle forskellige templates, som hjælper med at forme en 2D tegning. Den første template er "rectTemplate" som hjælper med at forme spær, rem og stolper. Den anden template "lineTemplate" hjælper med at forme linjer såsom stålbåndet. Den tredje template "arrowTemplate" former vores pile ved hjælp af linjer og pilehoveder. Herunder har vi tilføjet nogle "add metoder", som appender strings sammen ved hjælp af StringBuilder klassen, og dermed kobler dem til ServletGraphic klassen. I ServletGraphic har vi lavet en masse udregninger, som tager dataen fra Calculator klassen og bruger dem til at få udskrevet og formet en 2D svg tegning.

Calculator klassen har også mange attributter som primært er ints der fungerer som mellemregninger. Derudover har den også et Order object, en PartsListLine ArrayList og en Product ArrayList. Alle udregninger bliver lavet ud fra width og length, som er de data kunden har valgt som carportens bredde og længde. Der sættes en masse PartsListLine objekter ind i Array-Listen med et Product og de beregnede data, som til sidst kobles til Order. Derudover kaldes en metode til at udregne den samlede pris for styklisten.

Udvalgte kodeeksempler

Metode til fremvisning af Admin Panel:

```
@Override
public List<Order> retrieveAllOrders() throws DatabaseException {
    Logger.getLogger( name: "web").log(Level.INFO,  msg: "");

    List<Order> orderList = new ArrayList<>();

    String sql = "SELECT partslist_order_id, email," +
            "total_width, total_length," +
            "order_price, shed_id," +
            "accepted FROM carport.partslist_order";

    try (Connection connection = connectionPool.getConnection()) {
        try (PreparedStatement ps = connection.prepareStatement(sql)) {
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                int partslistOrderId = rs.getInt( columnLabel: "partslist_order_id");
                String email = rs.getString( columnLabel: "email");
                int width = rs.getInt( columnLabel: "total_width");
                int length = rs.getInt( columnLabel: "total_length");
                int orderPrice = rs.getInt( columnLabel: "order_price");
                int shedId = rs.getInt( columnLabel: "shed_id");
                boolean accepted = rs.getBoolean( columnLabel: "accepted");
                Order order = new Order(partslistOrderId, email, width, length, orderPrice, shedId, accepted);
                orderList.add(order);
            }
        }
    } catch (SQLException ex) {
        throw new DatabaseException(ex, "Error while loading 'carport' from Database.");
    }
    return orderList;
```

Metoden vi har brugt til at fremvise alle ordrer, der er blevet lavet af kunder. Denne specifikke metode er med til at tale til databasen, altså holde styr på ordrer samt tilføje nye ordrer til en liste i databasen. Denne metode tager kun fra en tabel “partslist_order”. Metoden bliver lavet i OrderMapper klassen, og bliver brugt til Admin Panelet så Admin kan holde styr på alle sine kunders ordre. *se ovenstående billede.*

Admin specifikke knapper:

```
<nav class="navbar navbar-expand-md navbar-light bg-light" rel="stylesheet">
    <div class="container">
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavAltMarkup"
                aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse justify-content-start" id="navbarNavAltMarkup">
            <%--LEFT NAV SIDE--%>
            <div class="navbar-nav mr-auto">
                <c:if test="${sessionScope.user.role == 'user' }">
                    <a class="nav-link" href="${pageContext.request.contextPath}/servletmyorders">| MINE ORDRE |</a>
                </c:if>
                <c:if test="${sessionScope.user.role == 'admin' }">
                    <a class="nav-link" href="${pageContext.request.contextPath}/servletadminpanel">| ADMINPANEL</a>
                </c:if>
                <c:if test="${sessionScope.user.role == 'admin' }">
                    <a class="nav-link" href="${pageContext.request.contextPath}/servletadminproducts">|
                        PRODUKTLISTE |</a>
                </c:if>
                <c:if test="${sessionScope.user.role == 'admin' }">
                    <a class="nav-link" href="${pageContext.request.contextPath}/servletuseroverview">BRUGEROVERSIGT
                        |</a>
                </c:if>
            </div>
        </div>
```

Bootstrap og Java kode brugt til at fremvise Admin specifikke knapper, så kun en Admin kan se dem. I billedet ovenfor bliver der fremvist hvordan vi via sessionScope tager fat i objektet user, og hvis user objektet har admin rollen, bliver knapperne vist via pagetemplate.tag siden. Vi er opmærksomme på at vi kun har brug for at tjekke om user er i lig med admin en gang, og at vores kode derfor sagtens kunne effektiviseres ved at fjerne de to nederste if statements. *se ovenstående billede.*

Calculator klassen:

```
private void splitCheck() {//if length is longer than 720 which is the longest board you need two boards.
    if (length > 720) {
        boardQuantity = 4;//doubles the quantity
        boardLength = length / 2;//split the board length in half(a post should be in the middle)
    }
}

private void calcRafterQuantity() { rafterQuantity = (int) Math.ceil(length / 52.0); }

private void calcPostsQuantity() {
    postQuantity = (int) Math.ceil(length / 300.0) * 2;
    if (postQuantity < 4) {
        postQuantity = 4;
    }
}

private void calcRoofing() {
    roofQuantity = (int) Math.ceil(width / 100.0);// roof is 120 wide and need to overlap with 20.

    //you will always get 600cm roof. even if you only need 350.
    partsList.add(new PartsListLine(products.get(7),  length: 600, roofQuantity,
            description: "tagplader monteres på spær"));
    if (length > 600) {
        partsList.add(new PartsListLine(products.get(7),  length: 360, roofQuantity,
                description: "tagplader monteres på spær"));
        roofQuantity = roofQuantity * 2;
    }
    partsList.add(new PartsListLine(products.get(8),  length: 0, (int) Math.ceil(roofQuantity / 4.0),
            description: "Skruer til tagplader"));
}
```

Calculator klassen beregner styklisten ud fra brugerens valgte længde og bredde, samt en liste af produkter der kommer fra databasen.

Vi lavede matematikken bag Calculator klassen ud fra det uddelte materiale med stykliste. Udvalgte udregninger af styklisten blev delt op i deres egne metoder for at give bedre overblik. *se ovenstående billede.*

Status på implementering

På nuværende tidspunkt er størstedelen af hjemmesiden funktionel. Der er enkelte mangler:

- Der er en fejl i "Opret bruger" siden, der gør det muligt at indsætte sin egen kode og få den kørt på serveren, via et XSS-angreb.
- Der er ingen fejlhåndtering af andre fejlkoder end HTTP fejl 404.
- Ingen af siderne er mobilvenlige og de er ikke blevet testet på skærme større end 16".
- Siden er ikke testet på alle browsere.
- Brugernes kode gemmes i databasen uden hashing.
- Ingen CAPTCHA/sikkerhed ved log ind siden eller opret bruger siden - man kan nemt lave en bot til at prøve at cracke folks koder eller til at oprette tusindvis af brugere. Der er heller ingen CAPTCHA som stopper brugeren fra at spam-oprette en masse ordrer.
- US-11, implementering af skur fik vi ikke færdiggjort, da vi havde mangel på tid, og da vi prioriteret andre ting. Dette er ellers den eneste user story/case vi ikke nåede at få implementeret. En løsning til hvordan vi kunne have fået det implementeret var, at få tilføjet mere udregning og udvide Calculator klassen samt SVG klassen. Hermed også i en servlet, som ville checke om der var et skur tilføjet eller ej.
- 'Balance/Saldo' under 'user' bliver ikke brugt. Tanken med denne attribut kommer fra Cupcake projektet. Vi ville gøre så brugeren selv kunne indtaste et saldo beløb på sin konto, og dermed betale for en carport ligesom i Cupcake projektet, hvor brugeren skulle betale for et antal cupcakes. Denne kollektive beslutning fandt vi længere hen irrelevant for vores opgaves mål, da vi valgte at gå en anden vej, hvor brugeren sender en forespørgsel i stedet for, at betale over hjemmesiden.
- 'View' ville vi have brugt i SQL, men fandt vi hurtigt ud af ikke var nødvendigt for vores løsning. Vi ville have lavet en join til styklisten, hvor vi hentet data ned fra forskellige tabeller, og dernæst sat op som et View. Dette valgte vi ikke at gøre, i stedet lavet vi et SQL join statement i en Mapper. Grunden til det var, at vi ikke følte vi skulle bruge joinen til andre ting, andet end til den ene Mapper.
- "Glemt kodeord" under log ind virker ikke. Tilføjet som et eksempel på en mulig brugervenlighed.
- Log ind knappen på forsiden skal man vælge bredde og længde før man kan klikke på den. Overvejelser på bedre brugervenlighed kunne være at forsiden udelukkende kun

er en velkomst og "Log ind knap", og først efter man er logget ind viser custom carport forsiden.

- Styklisten, der printes ud på kvitteringssiden er ikke altid i "rigtig" rækkefølge, ift. den udleverede stykliste. Dette kunne løses ved at bruge "order by" i sql statementet, når styklisten hentes fra databasen.

Test

	Calculator	UserMapper	OrderMapper	ProductMapper
Metoder:	calcPartsList() splitCheck() calcRafterQuantity() calcPostQuantity() calcRoofing() calcPrice()	login() invalidPasswordLogin() invalidUserNameLogin()	insertOrder() retrieveAllOrders() retrieveOrder() acceptOrder() deleteOrder()	retrieveProduct() retrieveAllProducts() getProductId() updateProduct()
Dækningsgrad:	65%	~50%	~85%	100%

Vi har valgt at teste Calculator klassen samt de forskellige mappers. Calculator classen har også nogle getters og setters som ikke er testet, hvilket giver den lave dækningsgrad. Vi har testet lidt løbende, men primært til allersidst. Vi har været bevidste om at det i et professionelt miljø ville være bedst at teste funktionerne løbende, men vi valgte fra start af at gøre det til sidst fordi projektet er lille og det samtidig er tidskrævende for os at lave tests grundet manglende erfaring.

Vi endte med at refaktorisere især Calculator klassen for at prøve at gøre den mere testbar. Calculator klassen har metoder med sideeffekter, hvor de påvirker variable og returnere void. Det var i første omgang grundet til at vi ikke mente at metoderne udover calcPartsList() behøvede at være public eller returnere noget, hvilket viste sig at gøre det sværere at teste. Derfor lavede vi getters og setters til nogle af variablene for lettere at kunne teste.

Proces

Arbejdsprocessen faktuelt

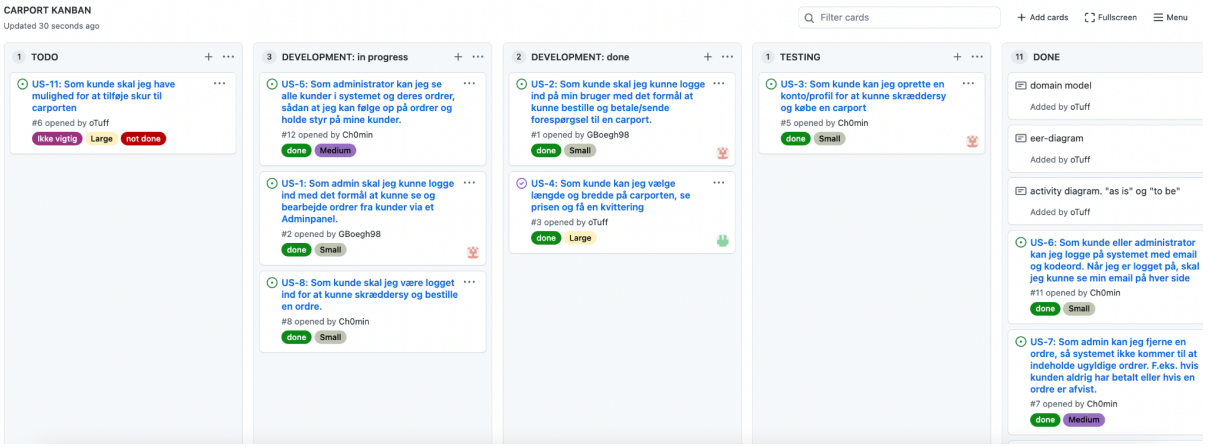
Vi startede projektet med at sætte os godt ind i kravspecifikationerne og hvad der krævede af os. Vi aftalte herefter at skrive en logbog for vores arbejdsproces i slutningen af hver dag. Derefter begyndte vi at lave på et EER-diagrammet i MySQL Workbench, fordi det dannede grundlag for hele projektet. Derefter oprettede vi et mock-up af hvordan hjemmesiden skulle se ud. Helt i starten brainstormede vi de klasser, som var de vigtigste, oprettede dem og delte klasserne op så alle havde noget at lave. Vi aftalte også kode standarden som ser sådan her ud:

- Alle servlets skal starte med navnet "Servlet" fx "ServletLogin".
- Alle url patterns skal være lowercase fx "hejmeddig".
- Curly braces skal være på samme linje som funktionen.
- Alle jsp skal være lowercase fx "index.jsp".
- Alle sql tabeller skal være lowercase.
- Alle values i sql tabellerne skal have "_" mellem hver mellemrum(snakecase), og lowercase fx "total_price".
- Java syntaks skal være camelcase, gælder også for klasser/id i HTML fx "getProductId".
- Alle billeder har vi valgt at være png.
- Koden skal være på engelsk, med få undtagelser.
- Hjemmesiden skal være på dansk.

Vi tog derefter inspiration fra SCRUM-modellen og delte projektet op i mindre bidder, for derefter at mødes hver dag for at kode, se hinandens kode igennem og lave en opsamling samt logbog.

Da vi begyndte at kode lavede vi hver vores egen branches, og til bestemte user stories oprettede vi også deres egen branch. Herefter tog vi hver en eller flere user stories og begyndte at kode.

Github Issues: TODO - DEVELOPMENT(In progress) - DEVELOPMENT(done) - TESTING - DONE



Vi brugte en funktion 'issues' i github til at oprette et kanban board, for at holde styr på hvor langt vi var med de forskellige user stories/cases. Dette var et essentielt værktøj til at holde styr på ugens planlægning. Vi udvalgte dermed en 'Kanban Mester' hver uge, som skulle holde styr på Kanban boardet under github issues værktøjet. Personen skulle også holde styr på en fælles 'logbog', som vi hver især opdaterede før og efter en dags arbejde.

Faser:

- **UGE 1:** Planlægningsfase - herunder blev der snakket om Github, KanBan, Database, Mock-up, Kodestandard. Her blev der kollektivt snakket om, hvordan vi ville gribe opgaven an. Vi fik lavet et EER Diagram, og opbygget en database.
- **UGE 2:** Funktionalitet og backend fase - herunder begyndte vi med at kode, og starte på programmets funktionalitet. Vi arbejdet på Calculatoren og forespørgselsdelen. Samt også, hvordan vi kunne få arbejdet på at få udskrevet en stykliste.
- **UGE 3:** SVG og testing fase - Den her uge arbejdet vi på at få forberedt og implementeret en SVG tegning ind i vores program. Dette gjorde vi fælles, og kom i mål. Dernæst arbejdet vi alle sammen på at få testet.
- **UGE 4:** Rapport fase - Denne uge arbejdet vi alle sammen på rapporten samt Front-end.

Til hver vejledningsmøde havde vi kollektivt forberedt os til, hvad vi ville snakke om, og hvilket spørgsmål, vi ville stille til specifikke emner og problemer. Et eksempel kunne være uge 2, hvor vi var i tvivl om, vi skulle gå i en retning af at brugeren skulle betale på selve siden, eller om brugeren bare skulle sende en forespørgsel af sted, som dernæst Admin kunne acceptere. Denne vejledning hjalp os med, at finde en løsning om hvad der kunne være en bedre løsning under det her projekt, hvilket i sidste ende var forespørgselsdelen.

Arbejdsprocessen reflekteret

Størstedelen af tingene i vores proces er gået godt, der er også noget som er gået dårligt. Dét at starte med at sætte os godt ind i kravspecifikationerne og forstå begreberne gjorde at vi hver især kendte opgaven bedre. Det fungerede også godt at lave et EER-diagram efter, fordi det var essentielt for at kunne få vores kode til at fungere. KanBan mesteren byttede ud med et andet gruppemedlem hver uge. Det gjorde at der var bedre styr på hvem der var i gang med hvad, så vi ikke kom til at kode de samme user stories. Det gav også et overblik over, hvor langt folk var kommet med hver user story. KanBan mesteren var også til dels en leder under ugen og tog beslutninger, som ville gavne og hjælpe gruppen.

Derfor var KanBan mesteren en hjælp, da det nogle gange kunne blive problematisk, fordi man måske skulle ind i klasser andre arbejdede på, for at tilføje noget som skulle bruges til ens egen user story. De klasser vi hver især arbejdede på fremgik ikke i KanBan boardet, så man skulle huske at informere de andre via vores “logbog”.

Efter at have oprettet vores EER-diagram satte vi os sammen og aftalte syntaksen for koden, og oprettede et par af de vigtigste klasser for derefter at oprette vores egen branches. Da vi senere kodede, så koden bedre og mere ensformig ud. Det var også meget nemmere at udføre CRUD operationer, fordi vi vidste at databasen ikke ville ændre sig. Vi havde kun mindre merge konflikter i git, som vi relativt hurtigt fik løst, fordi vi for enden af hver arbejdsdag én efter en pullede, mergede og pushede til main branchen.

På grund af vores smarte planlægning med at prioritere de vigtigste ting i starten, fandt vi hurtigt en arbejdsrytme som var produktiv, og gjorde at vi ikke stressede. Vi fik nået alle vores mål enten til tiden, eller før tid undtagen US-11. Det gjorde også at vi hver især bidrog til projektet, og var engagerede. Vores estimeringer var altid planlagt, fordi Kanban mesteren og gruppen var fælles om hvad målet var for hver uge og dag. Dermed havde vi forenden af ugen altid et produkt som stemte overens med vores planer for ugen.

Vores største problem var i starten, hvor vi var usikre på om hvorvidt vi skulle opbygge siden på en måde, der gjorde at brugeren skulle lave en forespørgsel som admin skulle acceptere, eller om brugeren kunne "købe" ordren med det samme uden en admins godkendelse. Vi diskuterede det både internt i gruppen, men også med vores vejleder, og prøvede at vende det på forskellige måder. Dermed kom vi frem til at Johannes Fog udbyder deres kunder en service ved at lade professionelle kigge på ordren inden de accepterer den, i deres nuværende system. Derfor besluttede vi kollektivt, at det ville være bedst med at lave en forespørgsel fremgangsmåde.

Et af vores andre store problemer var at få fremvist SVG i jsp-siden, fordi den kode vi havde fået udleveret ikke virkede. Vi løste det ved at vi alle sammen kiggede på koden fælles på en computer, og prøvede at komme med input hver især. På den måde kom vi hurtigt frem til en løsning på problemet.

Som tidligere nævnt, nåede vi heller ikke at lave US-11, på grund af manglende tid. Taget i betragtning af at det er den eneste use case/story som vi ikke nåede, og at det heller ikke var et fast krav fra brugeren, så ser vi vores arbejdsproces for at være positivt.

Helt overordnet set har vores kommunikation i gruppen været fremragende. Vi har mødtes fysisk stort set hver dag, og hjulpet hinanden, der hvor der var brug for det. Derudover har vores fælles logbog, hvor vi kunne se hvor langt vi hver især var nået, været en kæmpe stor hjælp.

Kommunikationen har været en stor del af projektet, fordi det har været vigtigt at uddelegere opgaverne, men samtidig støtte hinanden fordi mange af opgaverne overlappede hinanden.

Det vi har lært af processen er helt klart at arbejde bedre sammen i grupper, samtidig med at vi har fået en bedre forståelse for Git og servlets. Vi er blevet bedre til at uddelegere opgaver, og hjælpe hinanden med at løse små problemer og diverse bugs. Vi har også hjulpet og lært hinanden de forskellige værktøjer bedre at kende. Det har gjort os bedre som gruppe og til at løse adskillige problemstillinger sammen som gruppe. I praksis fungerede det godt at dele tingene op i små bider og derefter samle op på det. Det gjorde at ingen af os kom til at spilde for lang tid på unødvendige ting. Det gjorde også at vi kom i mål med at lave en fuldendt funktionel hjemmeside.

Bilag

Mockup:

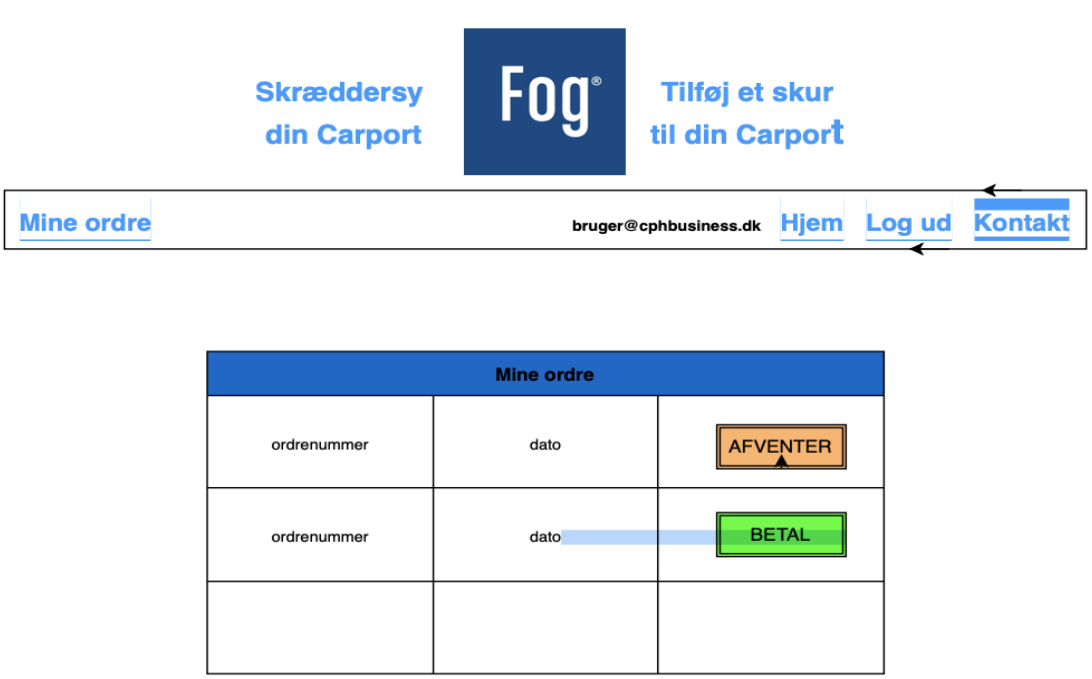
Vejledende pris: 11999 kr.


Skræddersy din Carport
Fog
Tilføj et skur til din Carport
Mine ordre
bruger@cphbusiness.dk
Hjem
Log ud
Kontakt
Tak fordi du valgte os!
Vi vil besvare din forespørgsel hurtigt muligt.
Du kan finde og betale under 'ordre'
Mine ordre
Til forsiden

Tak for din bestilling!
Vi vil behandle din ordre hurtigst muligt

KVITTERING

STYKLISTE		


Skræddersy din Carport
Fog
Tilføj et skur til din Carport
Mine ordre
bruger@cphbusiness.dk
Hjem
Log ud
Kontakt
Mine ordre
ordrenummer
dato
AFVENTER
ordrenummer
dato
BETAL

Adminpanel Ordre		
kunde	leveringsadresse	ACCEPTER AFVIS
kunde	leveringsadresse	ACCEPTER AFVIS
kunde	leveringsadresse	SLET