Fog Carporte

Nicklas Hansen, cph-nh192@cphbusiness.dk
github.com/Kofoedsan

Thorbjørn Jensen, cph-tj308@cphbusiness.dk
https://github.com/ThorbjoernJensen

René Stafaniuk Andersen, cph-ra147@cphbusiness.dk
github.com/Rene061089

Link til droplet:

http://64.225.101.38:8080/fog-carporte-1.0-SNAPSHOT/

Ansat login:

Brugernavn “big@boss”

Kode: “1”

Link til droplet 2:

http://165.22.27.222:8080/Fog%20Carport/

Ansat login:

Brugernavn “ansat”

Kode “1”

Link til demovideo:

https://youtu.be/J1ucSUD94-E

Github link til projekt: https://github.com/ThorbjoernJensen/Fog-Carporte

Klasse C

Gruppe 7

Dato: 28/5 2021

Table of Contents

Indledning

Kunden Johannesfog.dk er et online byggemarked, der har en afdeling der beskæftiger sig med salg af carporte. Fog har positioneret sig som et byggemarked, der leverer carporte med stor valgfrihed for kunden i forhold til materialer og dimensioner[1]. Derudover lægger de vægt på gennemsigtighed på pris, samt kompetent vejledning. Det system som kunden ønsker leveret, lægger sig i forlængelse heraf.

Der ønskes et webbaseret system, der gør det muligt for kunder, at bestille carporte på egne mål og materialer. Ud fra dette, genererer systemet et oplæg til en sælger. Dette oplæg indeholder en stykliste inkl. Pris og en dynamisk genereret tegning af carporten. Sælgeren rådgiver kunden ud fra dette oplæg, og har mulighed for at sætte en rabatpris.

Vores produkt er et bud på en sådan løsning, og denne rapport her dokumenterer vores overvejelser og specifikke valg, som er truffet løbende i dialog med repræsentanten for kunden.

Teknologi valg

- Intellij
    - IntelliJ IDEA 2020.3.2 (Ultimate Edition)
    - Build #IU-203.7148.57, built on January 26, 2021
    - Runtime version: 11.0.9.1+11-b1145.77 amd64
    - VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
    - Maven v 3.6.3
    - MySQL workbench 8.0.23 community
    - PlantUML v. 5.3.0
    - JDBC Driver - mysql-connector-java v 8.0.19
    - JSTL javax.servlet v. 1.2
    - Bootstrap v. 4.3.1
    - Junit v. 5.7.1
- Tomcat 9.0.46
- JDK java version "15.0.1" 2020-10-20
- GIT version 2.30.1.windows.1
- DigitalOcean droplet med Linux Ubuntu x64
- Google Chrome Version 90.0.4430.212

[1] VideoInterview af afdelingsleder Martin Kristensen hos Fog byggemarked. Delt internt på moodle under oplægget til projektet af “4. besøg hos kunden”

Krav

Fog carporte har på nuværende tidspunkt en webløsning der lader kunden bestille carporte ud fra egne mål.[2] I vores projekt tager vi afsæt i den idé at system ikke allerede findes - eller at vi er ved at udvikle et konkurrerende system, fra bunden.

Her kan “as is” beskrives sådan, at kunden vælger en carport fra et katalog, på faste mål og med en fast pris. Opgaven for Fogs sælgere er her evt. at rådgive kunder i forhold til hvilken af de prædefinerede carporte der passer bedst til kundens behov, men selve vejledningen er begrænset, da valgene på forhånd af truffet.

Hvad vi ønsker at implementere, “To be”, er derfor et system der dels giver valgfrihed for kunden, men også ruster sælgeren til at få en mere aktiv rolle som rådgiver. Den specifikke udformning af krav er formuleret gennem Scrum user stories.

Scrum User Stories:
Scrum User Stories er formuleret af gruppen i samarbejde med product owner. Product owner har godkendt og udvalgt de enkelte user stories der skulle implementeres. Herunder vises de user stories som product owner har valgt og som gruppen har forpligtet sig på. Vi har valgt at bibeholde navnene, og de afspejler at der er blevet valgt en del fra, og nogle er delt op/tilføjet undervejs.
Den fulde backlog findes i appendix.

US-4: Som <kunde> ønsker jeg at <kunne vælge et design på en carport ud fra egne mål og ønsker> sådan at jeg <kan designe min carport efter specifikke behov>
Givet at: jeg kan tilgå en side hvor der kan foretages bestilling
Når: kunden vælger et design og trykker send.
Så: sendes en ordre-forespørgsel der bliver gemt i en database.
Estimat: L

US-5a - Som <kunde> ønsker jeg at <kunne oprette en konto/profil> sådan at jeg kan <logge ind>
Givet at: kunden ønsker at oprette en profil.
Når: kunden udfylder brugerinfo og vælger samme password to gange.
Så: bliver kunden oprettet i systemet, og gemt i databasen.
Estimat: S

[2] https://www.johannesfog.dk/byggecenter/have--fritid/byg-selv-produkter/carporte/enkelt-carporte/

US- 5b - Som <kunde> ønsker jeg at <kunne logge ind> sådan at jeg kan <se og følge status på mine ordrer og foretage ordre-forespørgsel>
Givet at: kunden er logget ind.
Når: kunden udfylder brugernavn og password korrekt.
Så: kan kunden oprette ordre-forespørgsel og se status for ordre.
Estimat: S

US-6: Som <kunde> ønsker jeg at <kunne generere tegning af min carport> sådan at jeg kan <se hvad jeg bestiller>
Givet at: kunden har sendt ordreforespørgsel
Når: når kunden er logget ind og står på loginsiden.
Så: kan kunden få vist et billede af den bestilte carport.
Estimat: S

US-10: som <kunde> ønsker jeg at <se detaljer for min ordre> sådan at jeg kan <se status og (evt. slette/ændre i ordren)>
Givet at: man kan tilgår en ordre liste/Der er en ordre
Når: når ordren er accepteret.
Så: kan jeg opdatere og rette i ordre.
Estimat: M

US-11: Som <ansat> ønsker jeg at <kunne se alle ordrer> sådan at jeg kan <holde overblik og rette i ordre>
Givet at: man kan tilgår en ordre liste/Der er en ordre
Når: når ordren er accepteret.
Så: kan jeg opdatere og rette i ordre.
Estimat: M

US-12a: Som <ansat> ønsker jeg at <kunne se kundens ordredetaljer> sådan at jeg kan <give god vejledning>
Givet at: Der er en ordre-forespørgsel og jeg står på ordre siden.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg se ordre-forespørgsel med dimensioner.
Estimat: M

US-12b: Som <ansat> ønsker jeg at <kunne se kundens ønskede design> sådan at jeg kan <give god vejledning>
Givet at: Der er en ordre-forespørgsel og jeg står på ordre siden.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg se den tekniske tegning.
Estimat: M

US-12c: Som <ansat> ønsker jeg at <kunne se materiale listen> sådan at jeg kan <give et godt tilbud>
Givet at: Der er en ordre-forespørgsel og jeg står på ordre siden.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg se materiale listen.
Estimat: M

US-13: som <ansat> ønsker jeg at <kunne sætte prisen> sådan at jeg kan <give tilbud til kunden>
Givet at: Der er en ordre-forespørgsel.
Når: kunden er i tvivl eller synes det er for dyrt.
Så: kan man ændre prisen for at lukke salget.
Estimat: M

US-17: Som <ansat> ønsker jeg at <kunne oprette en ordre> sådan at jeg kan <sende ordren til ekspedering>
Givet at: Der er en ordre-forespørgsel og jeg står på ordre siden.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg godkende ordren så den oprettes i systemet
Estimat: M

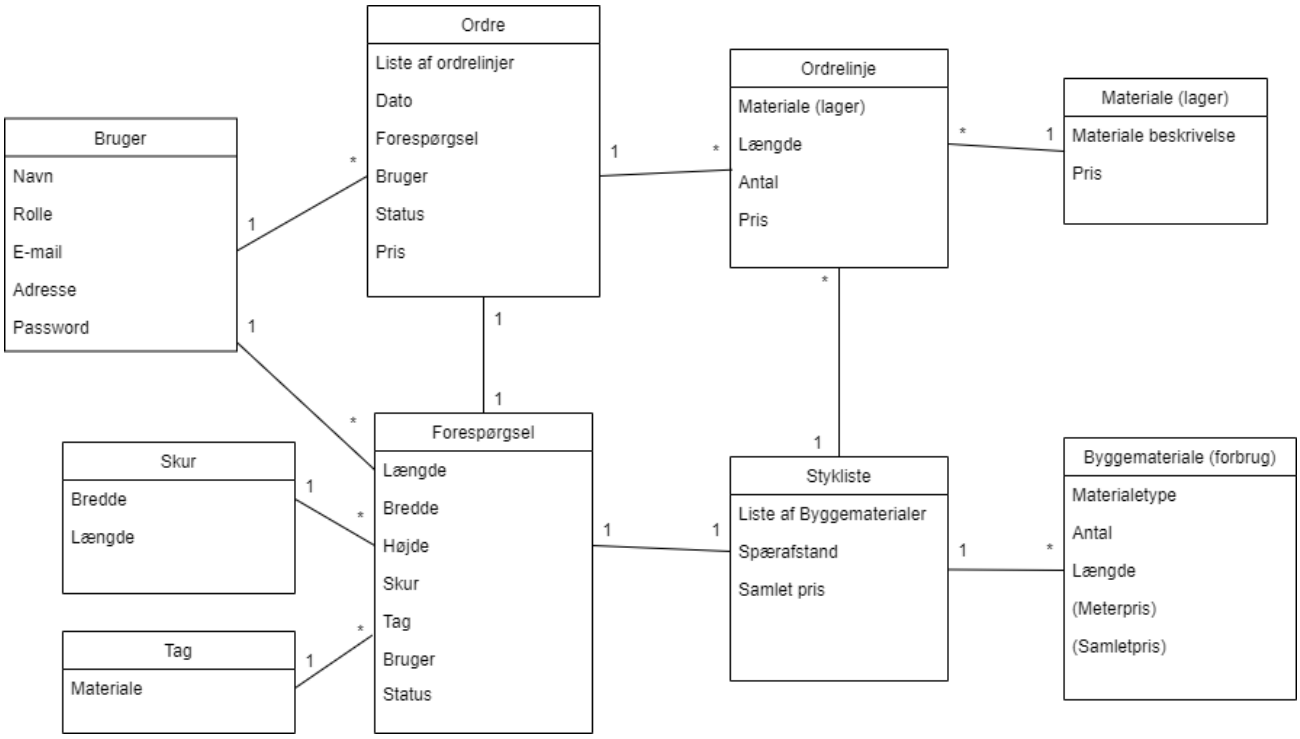
US-17c: Som <system> ønsker jeg at <kunne oprette kunder i systemet samtidigt med at der oprettes en forespørgsel> sådan at jeg kan <knytte en bruger på forespørgsler>
Givet at: jeg som kunde kan oprette en forespørgsel.
Når: kunden sender en carport forespørgsel.
Så: bliver kunden og carport forespørgsel oprettet i systemet
Estimat: M

Domæne model

Domænemodellen kan inddeles i tre hovedområder: Èn del der håndterer et carport-ønske fra en kunde (forespørgsel, skur og tag) – dernæst en sektion der repræsenterer brugerens forespørgsel som en stykliste, der består af mål og antal af ønskede byggematerialer(spær, rem, stolper m.m.). Det er denne stykliste, sammen med en dynamisk genereret tegning af carporten, som sælgeren kan rådgive kunden ud fra. Den genererede svg-tegning er også en konceptuel kandidat til diagrammet, men vi har valgt ikke at tage den med, for at undgå for mange detaljer i diagrammet.

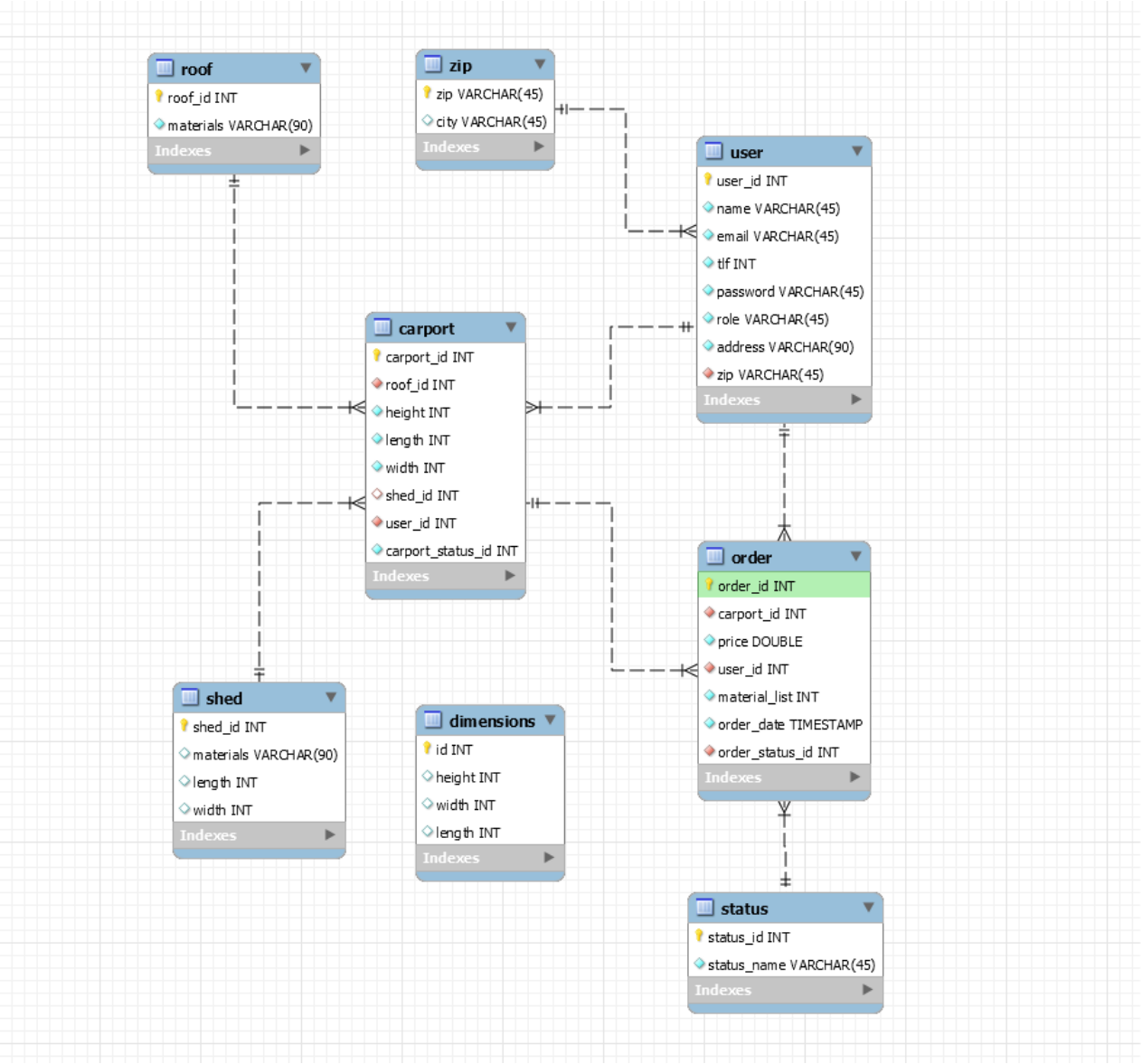
Sidst har vi ordre-delen, der oprettes når sælger godkender forespørgsel. Ordren oprettes med en samling ordrelinjer, der består af mål og antal af materialer fra lageret, alt sammen beregnet ud fra styklisten. Materiale- delen, der bliver sendt fra ordrelinje, svarer til en liste af de varer virksomheden har på lager.

Meterpris og samletpris i Byggemateriale er sat i parentes, fordi det er data der hører til i listen over lagervarer og i ordrelinjen. Men da vi i den nuværende implementering af systemet har hardcoded disse værdier, er de taget med.

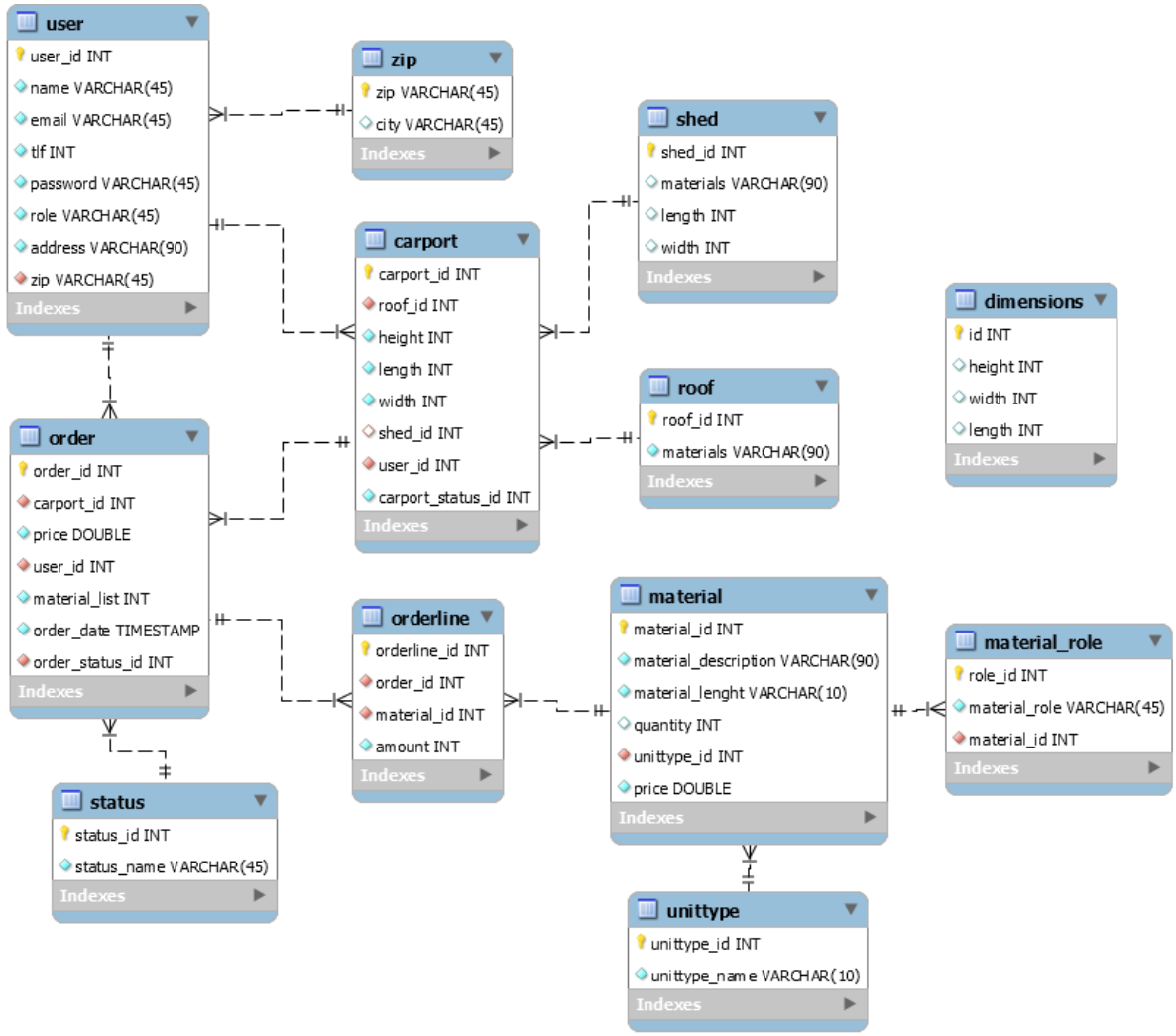


EER-diagram

Nedenstående EER-diagram svarerm til, hvad vi på afleveringstidspunkt har implementeret. Det er disse tabeller der bliver behandlet data i, via mappere i java-koden, når programmet kører.



Det nedenstående diagram, viser hvordan vi fra start har designet databasen, for at programmet ville være funktionelt. Den store forskel er, at den del der berører håndteringen af orderlines, herunder material, material_role og unittype, ikke er implementeret. Dermed persisteres indholdet af en ordre ikke, men dette er vores forslag til design af tabeller.



Diagrammet følger i vidt omfang de konceptuelle klasser som vi ser i domænediagrammet. Selve carport-tabellen svarer til forespørgslen i domænemodellen, med nogle givne mål og produktvalg, som knyttes sammen med en given bruger.

Væsentligste forskel, i forhold til domænemodellen, er at styklisten ikke er at finde, men derimod kun ordren og dens tilhørende ordrelinjer. De enkelte ordrelinjer består af et antal og en mængde inkl. en enhed, f.eks. cm ell. Stk. (som defineres i unittype). Selve ordre-tabellen og dens forbundne “orderline” og “material”, svarer til styklisten som vi har i domænediagrammet.

Tabellen “material_role” giver mulighed for at knytte de forskellige konstruktionselement/typer, med en given material_id, så man eksempelvis kan definere hvilken vare på lageret der skal benyttes til spær.

Vi har valgt kun at give mulighed for at definere én material_id til hver elementtype, så stolper kun kan anvende den samme type træ fra lageret. Det er oplagt i større skala at implementere, muligheden for at vælge mellem flere forskellige typer træ, til hver elementtype. Det ville kræve at der blev sat en link-tabel imellem material og material_role, der havde role_id og material_id som fælles primærnøgle.

For at sikre dataintegritet, er tabellerne knyttet sammen gennem primærnøgler, der anvendes som fremmednøgler i de tabeller de forbinder med. I praksis kan der således ikke sættes en værdi ind på fremmednøglens plads, hvis samme værdi ikke allerede er oprettet i den tabel hvor nøglen er primærnøgle. Vi kan f.eks. ikke oprette en bruger med et postnummer hvis postnummeret ikke allerede eksisterer i zip-tabellen. Det sætter det krav at zip-tabellen skal være udfyldt med alle gyldige postnumre, for at en ny bruger kan oprette sig med et tilfældigt dansk postnummer.

Tabellen dimensions står udenfor de øvrige tabeller. Den udfører den opgave, at samle nogle standard-mål for carporte på samme sted. Hver række i denne tabel skal ikke tænkes som en entitet, men det sparer nogle tabeller og mapperfunktioner at holde dem samlet i én tabel.

Tabellerne opfylder 1. normalform ved at de alle har en entydig nøgle. I praksis er dette sikret ved at lade primærnøglen bestå af et autogenerer heltal.

Kolonnen “adresse” i tabellen “user” tænker vi som ét navn. En sammentrækninger af vejnavn og nummer. Det kunne være relevant at dele op, men vi vurderer at det til vores brug, svarer til at skulle dele navnet Christian d. 4. op i to forskellige kolonner.

Da der ikke er nogen delte primærnøgler opfylder tabellerne også 2. normalform.

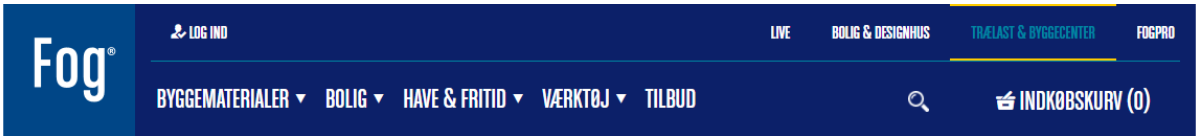
For at opfylde 3. normalform har vi indsat tabeller, der kun knytter en integer-id som nøgle sammen med et variabel navn. Dette gælder for tabellerne zip, status og unittype.

Eftersom “bruger” i ordre tabellen er afhængig af “forespørgslen”. Som vores kode er implementeret, kan man tale om et brud på 3. Normalform. Det kan dog sagtens tænkes at systemet senere kunne have en uafhængighed, mellem forespørgsel og ordre. Der kunne f.eks. placeres en ordre uden at det var med afsæt i en forespørgsel.
Dermed er der ikke afhængelighed mellem bruger og forespørgsel.

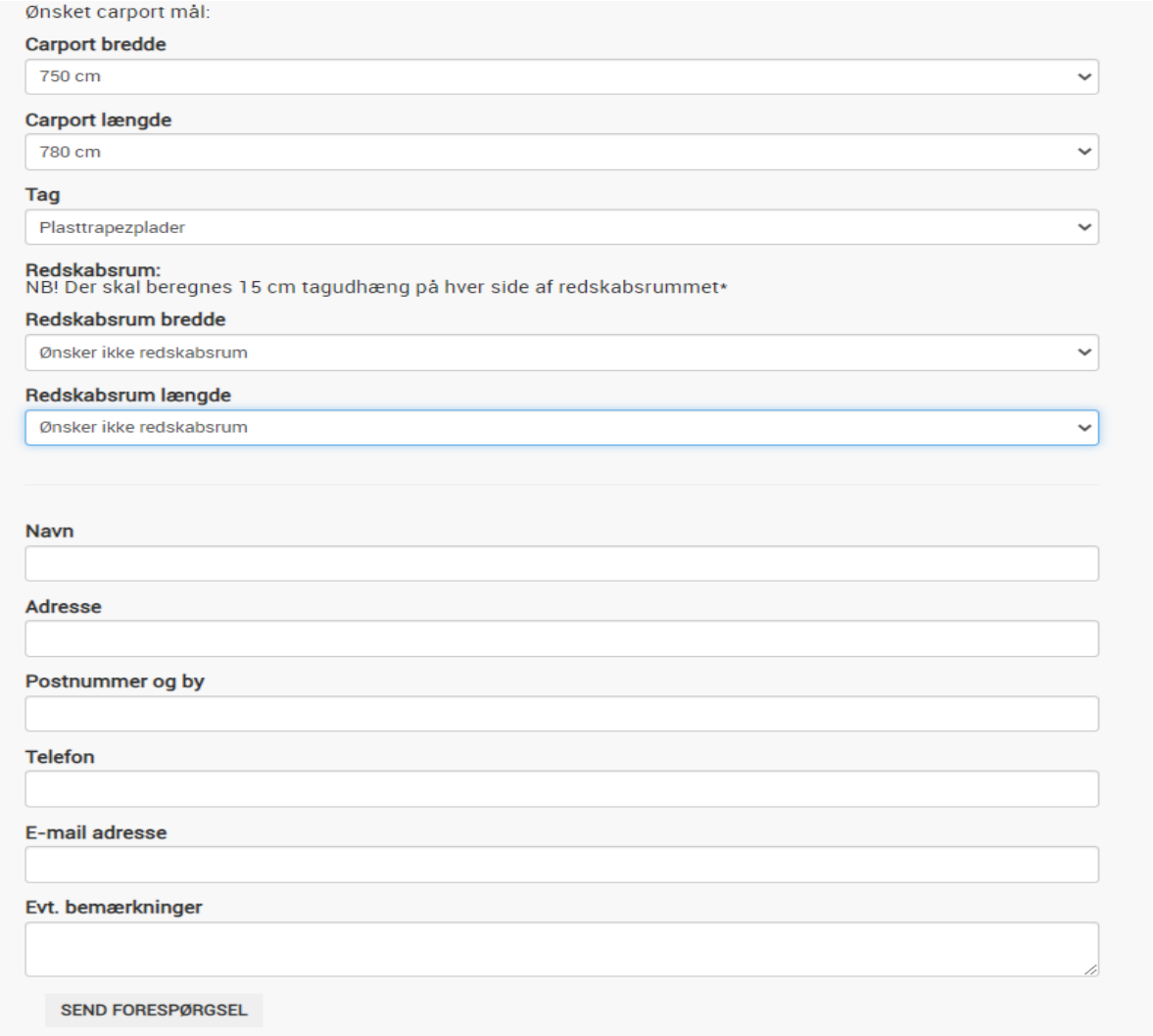
Mockups

Målet var at komme så tæt på det originale UX designm som det var muligt, af den side, som ville lade en bruger skræddersy sin carport, på johannesfog.dk. Målet var dog stadig at sætte vores eget præg på det grafiske design.

Målet var at have en navigation bar der ville fungere som den primære metode til at navigere rundt på de forskellige jsp sider.



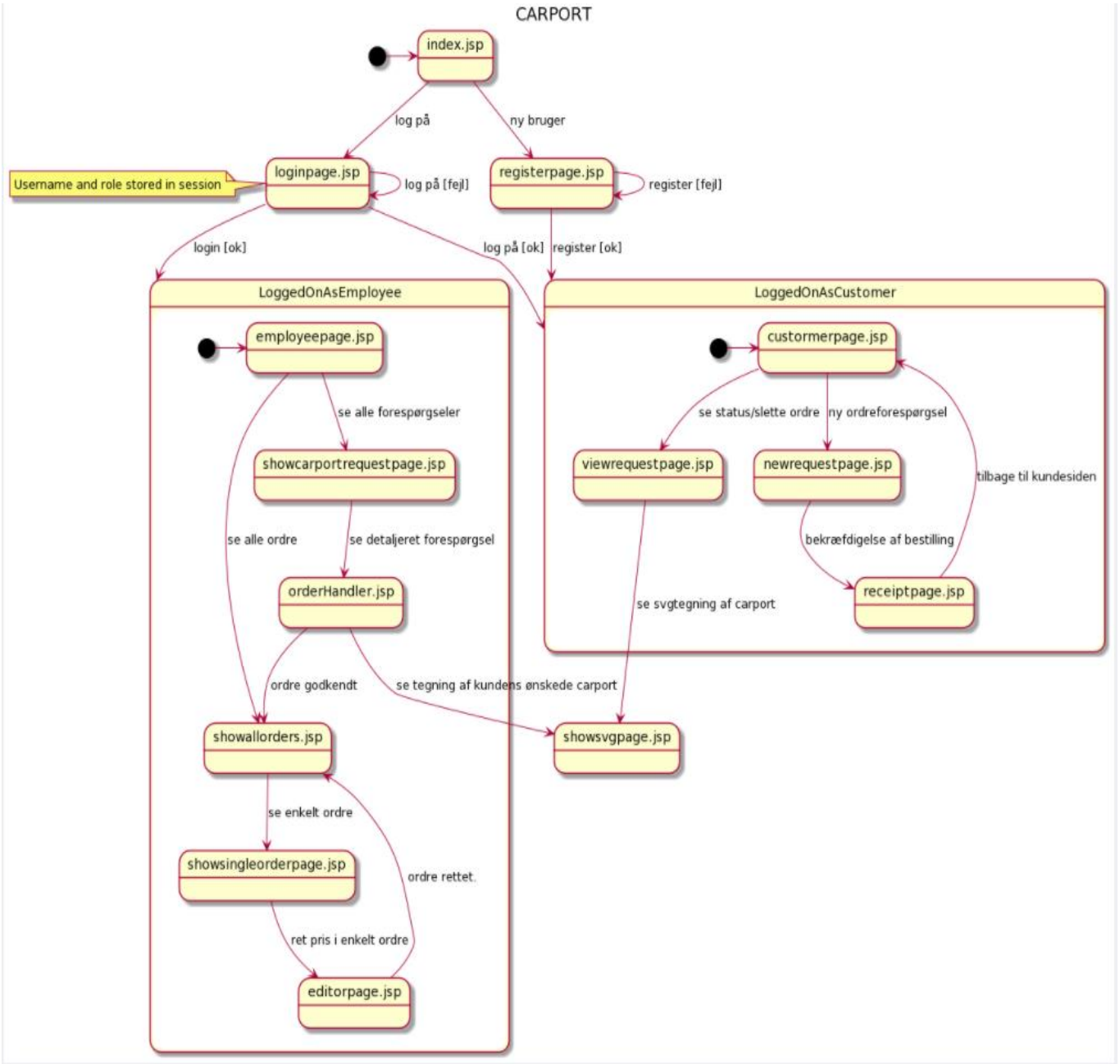
Der blev taget udgangspunkt i bestillingssiden fra johannesfog.dk. Der er dog ændret i designet og tilføjet muligheden for at vælge ønsket højde på carporten.



Kvitteringssiden, kundesiden, og de ansattes mulighed for at se ordre m.m. I systemet, har vi sat et simpelt design mål for, hvordan de skulle ende med at se ud. Vi har vægtet det funktionelle.

Navigationsdiagram

Oversigts diagram



Navigation-bar

Vi har valgt at benytte os af en “fælles navigations bar” i toppen af alle siderne. Den vil variere afhængig af om man er logget ind som kunde, ansat eller slet ikke er logget ind.

Navbar “ikke logget ind”.

Hvis man ikke er logget ind, indeholder navbaren[3] to knapper. Login sender brugeren til loginsiden(loginpage.jsp), Sign up sender en til en side hvor man kan oprette sig som ny bruger(registerpage.jsp).

Navbar logget ind som ansat

Når man er logget ind som ansat indeholder navbaren fire knapper. “Hjem” sender brugeren til employeepage(employeepage.jsp), “Ordrer” sender brugeren til ordresiden, som indeholder alle ordrer(showallorders.jsp) , “Forespørgsler” sender brugeren til siden, som indeholder alle forespørgsler(showcarportrequestpage.jsp) “Log ud” sender brugeren til index siden(index.jsp).

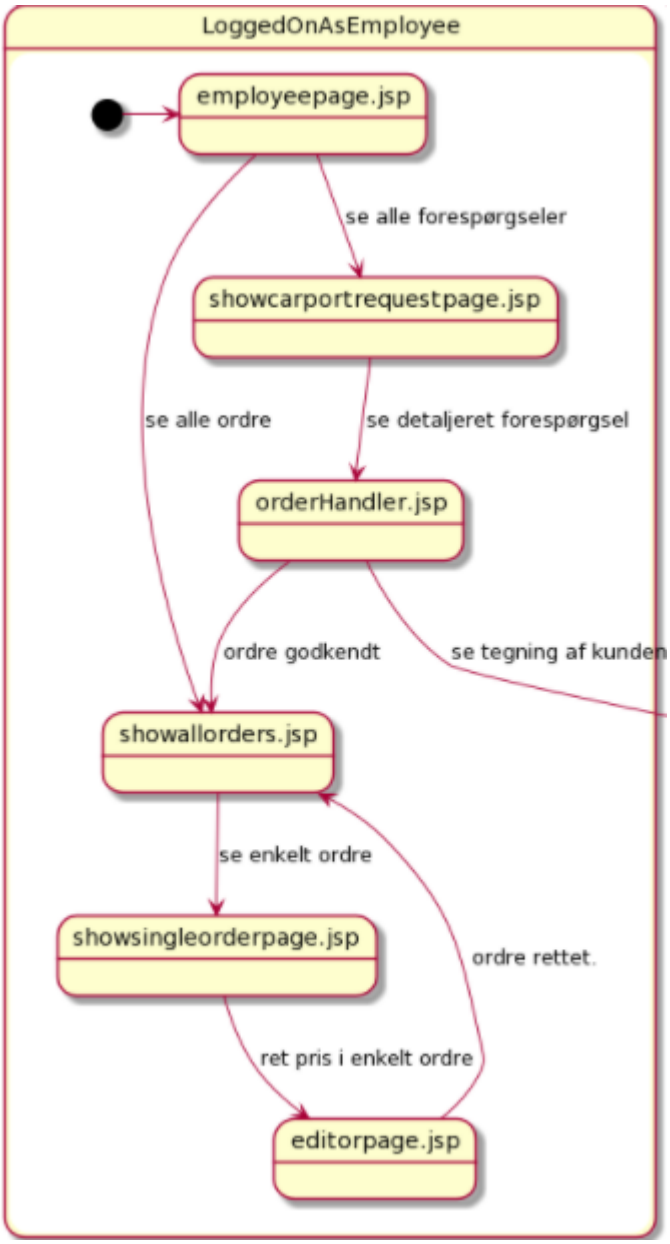
Navbar logget ind som kunde

Når man er logget ind som kunde indeholder navbaren fire knapper. “Hjem” sender kunden til customerpage(cusomerpage.jsp) “MineOrdrer” sender kunden til siden, hvor kunden kan se/slette sine ordrer(viewrequestpage.jsp), “OpretNyOrdre ” sender kunden til bestillingssiden, hvor kunden kan oprette en ny

[3] Navigations bar: En måde måde, hvor man altid kan tilgå en given side hvis man har tilladelse til det.

ordre(newrequestpage.jsp) og “Log ud” logger kunde ud og sender kunden til index siden(index.jsp).

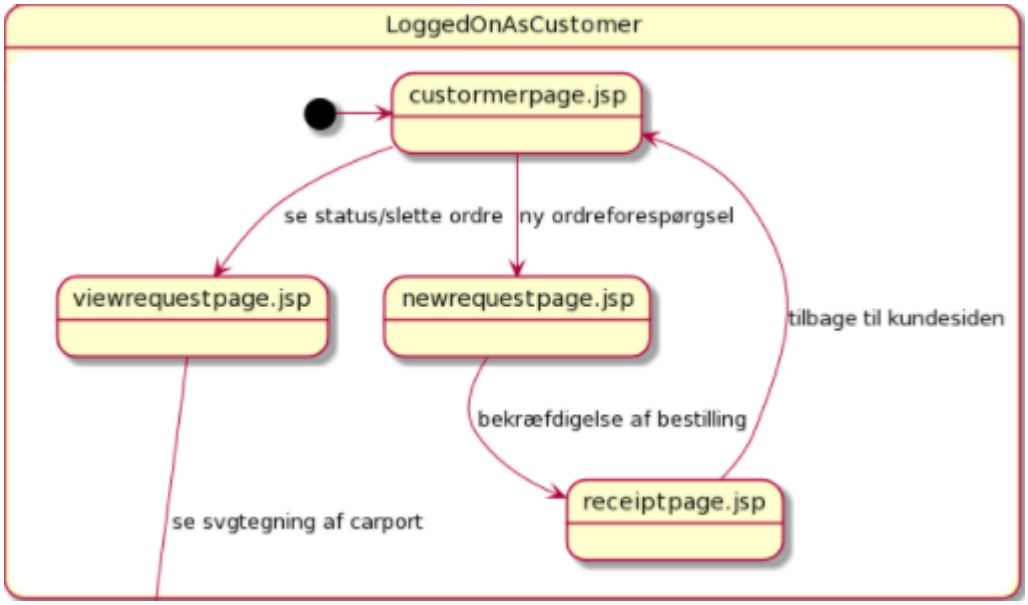
Ansat diagram



Når man er logget ind som ansat, kan man nå indholdet der er er på: employeepage.jsp, showallorders.jsp, showsingleorderpage.jsp, showcarportrequestpage.jsp, orderHandler.jsp, editorpage.jsp, showsvgpage.jsp. Hvis man ikke bruger navigationsbaren, kan man nå, showcarportrequestpage.jsp og showallorders.jsp via employeepage.jsp. Fra showcarportrequestpage.jsp kan man nå orderHandler.jsp og fra orderHandler.jsp kan man komme videre til showsvgpage.jsp og showallorders.jsp. Fra showallorders kan man tilgå siden

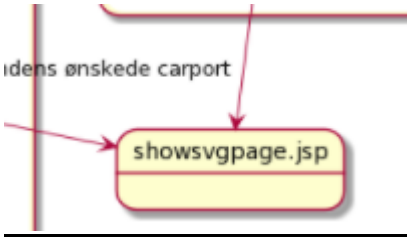
showsingleorderpage.jsp. Fra showsingleorderpage.jsp kan man nå editorpage.jsp. Det er kun muligt at nå showsingleorderpage.jsp fra showallorders.jsp og det er kun muligt at nå editorpage.jsp fra showsingleorderpage.jsp.

Kunde diagram



Når man er logget ind som kunde, kan man nå indholdet der er er på: custermorpage.jsp, viewrequestpage.jsp, receiptpage.jsp, newrequestpage.jsp, showsvgpage.jsp
Hvis man ikke bruger navigationsbaren, kan man nå viewrequestpage.jsp og newrequestpage.jsp fra customerpage.jsp. Fra viewrequestpager.jsp kan man nå showsvgpage.jsp. Fra newrequestpage.jsp kan man nå receiptpage.jsp

Der er en alternativ måde at nå receiptpage.jsp på, og det er via index.jsp. Her kan oprette en ordre forespørgsel og oprette sig som bruger på én gang. Når man trykker “send”, bliver man logget ind og sendt videre til receiptpage.jsp.



Fælles for, om man er logget ind som kunde eller ansat, er at de begge kan nå den samme side, som viser en svg-tegning af en ønsket carport. Denne side hedder showsvgpage.jsp.

Valg af arkitektur

Programmet tager udgangspunkt i den udleveret startkode.

Startkoden benytter følgende design patterns:

- Model View Controller (MVC pattern) danner grundlag for arkitekturen i opgaven.

Modellen består af dele, en controller, en view og en model.

  - Model indeholder al data og logiske regler. Dette layer fungerer uafhængigt af brugergrænsefladen.
  - View præsenterer modellens data på en brugervenlig grænseflade. I vores tilfælde er det HTML/JSP-siderne.
  - Controller håndterer input/output. Altså request/response. I vores tilfælde håndterer controlleren det meste, da den håndterer requestScopes mm.

Denne arkitektur er fordelagtig, da den gør det muligt for os som udvikler at arbejde på samme user case på samme tid. Uden vi arbejder oveni hinanden.

- Frontcontroller pattern

Programmet gør brug af frontcontroller pattern, som håndterer alle requests der kommer for at tilgå siden. Derudover håndterer den forbindelsen og godkendelsen af login til databasen hvor ordre, bruger mm. gemmes.

Frontcontrolleren logger de Userexceptions der bliver smidt af programmet og kategoriserer dem efter hvor kritisk fejlen er.

Frontcontroller består af 3 dele:

  - **Controller**

    Her håndteres alle requests og bruger godkendelse.

  - **View**

    Håndtere visning af data til bruger. View henter data fra en model. Data i programmet vises som udgangspunkt på en HTML side.

  - **Dispatcher**

    Håndterer hvor en bruger/data skal sendes videre til. F.eks. fra JSP til en java klasse.

- **Command pattern**

Programmet gør også brug af et command pattern, som modtager data fra frontcontrolleren. Command pattern finder så det objekt der kan håndtere den given command og videregiver objektet.

Command pattern er skille laget fra web layer og business layer i programmet. Herefter er det kun java og ikke længere blandet med HTML5.

- Singleton pattern

Singleton er anvendt til sikre at et objekt/klasse kun bliver dannet én gang. Altså for at undgå duplicates. Det er anvendt til at adskille brugers data, når der er flere logget på.

Singleton findes også i database connection, og sikre kun én forbindelse bliver oprettet ad gangen.

- Facade pattern

Programmet anvender facade pattern til at kommunikere med databasen.

Det sikrer en gennemsigtighed og gør programmet mere letlæseligt da alt database og SQL er adskilt fra resten.

- Dependency injection

I programmet er der forskellige dependency injections.

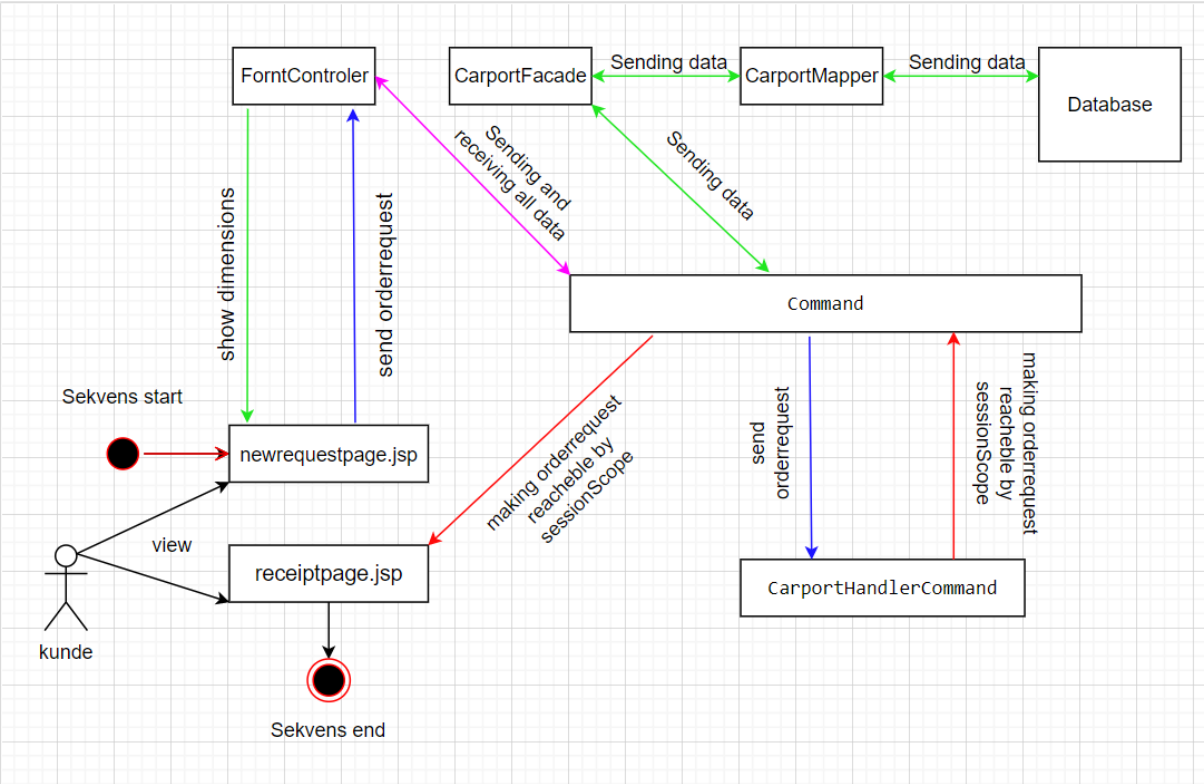
Dependency injection sker, når et objekt er afhængeligt af et andet objekt.

Det sker primært via metode injektion. Hvor en metode er afhængig af et objekt.

Det er gjort for at gøre koden mere letlæselig og brugervenlig, da en metode kun afhænger af ét samlet objekt som kommer fra objektets constructior.

Sekvens diagrammer

Sekvens diagram over bestillingssiden når man er logget ind som kunde.



Når man er logget ind som kunde, kan man tilgå newrequestpage.jsp, som er forbundet med klassen CommandNewRequest. På denne side har kunden mulighed for at vælge højde, længde og bredde dimensioner kunden ønsker carporten skal have.

Muligheden for at vælge forskellige dimensioner, sker via en forEach-løkke på newrequestpage.jsp. Her looper den igennem en liste af forskellige dimensioner. Dette gøres både for højde, længde og bredde. Listen ligger i applicationScope i klassen ‘’FrontController’’. Alt data der bliver send mellen siderne og fra de forskellige tilhørende klasser, bliver sendt igennem ‘’FrontController’’.

‘’FrontControlleren’’ får listen fra ‘’CarportFacede’’ og tildeler listen et attribute navn, som er ‘DimensionLis’.

I klassen ‘’Carportfacade’’ ligger en metode som hedder getAllDimensions denne metode returnerer en en metode fra ‘’CarportMapper’’, som også hedder getAllDimensions.

I ‘’Carportmapperen’’ er metoden getAllDimensions, som henter data fra vores Carport database med tabel navnet ‘dimensions’. Dette gøres med en sql sætning “SELECT * FROM dimensions”. Så oprette den en liste med navnet ‘dimensionList’,

og putter dimensionerne ned i den. Dette gøres via en while-løkke, så længe databasen med table navnet ‘dimensions’ indeholder input, som stemmer overens med constructoren fra klassen ‘’Dimensions’’.
Når kunden trykker på send forespørgsel bliver den sendt til klassen ‘’CarportHandlerCommand’’ her bliver den oprettet og stemplet i databasen via en if() statement, hvilken if() statment afhænger af, om man har bestilt med eller uden skur. I dette tilfælde er det uden skur, men der findes en identisk sekvens, hvis bestillingen er med skur. Her er metode navnene “withShed” i stedet for “withoutShed”.

Måden den bliver oprettet på er ved at kalder metoden “insertCarportWithoutShed”, som ligger i klassen Carportfacade. Metoden “insertCarportWithoutShed” i klassen “CarportFacade” kalder metoden med samme navn fra klassen “CarportMapper”.

I klassen “CarportMapper” bliver ordren så stemplet ned i databasen via metoden “insertCarportWithoutShed” med sql sætningen “INSERT INTO carport”, hvor den venter på en gennemgang af en ansat. Metoden virker kun hvis den tilsvarende constructor fra klassen “Carport” bliver udfyldt.

Kundens ønsker stemplet i sessionen og kunden ryger videre til receiptpage.jsp som fungerer som kvitteringssiden, hvor bestillingen bliver hevet ud af sessionen via sessionScop op printet ud til kunden.

Særlige forhold

- Informationer som gemmes i session
    - Ved brugerlogin:
        - Email, navn, tlf, rolle.

    - Hvis bruger har oprettet en forespørgsel:
        - I hver brugersession gemmes Bill of Materials, som beregner hvilke materialer en carport skal bruge for at blive bygget. Der gemmes derfor en attribut med navnet samletpris.
        - Carport objektet som indeholder carportId, roofId, højde, længde, bredde, skur, samt brugerId og status på ordren.

    - Hvis burger har oprettet en ordre:
        - Ordre objekt, bruger objekt

    - Som ansat:
        - Email, navn, tlf, rolle.
        - Orderlist, orderId på den ordre den ansatte klikker sig ind og godkender, prisen & newPrice som bruges hvis en anset giver et tilbud til kunden.
        - newPrice bliver fjernet igen fra sessionen, efter den er sat til den pågældende ordre.

  Fejlhåndtering gemmes i requestScope.

- Hvordan håndterer man exceptions og logger.
    - Der er anvendt en UserException klasse som arver fra klassen Exception og afgiver en besked/msg.
    - I frontcontrolleren er der en logger, som logger fejlmeddelserne til os eller brugeren. Loggen bliver ikke gemt i databasen eller som fil. Kun imens programmet er kørende.

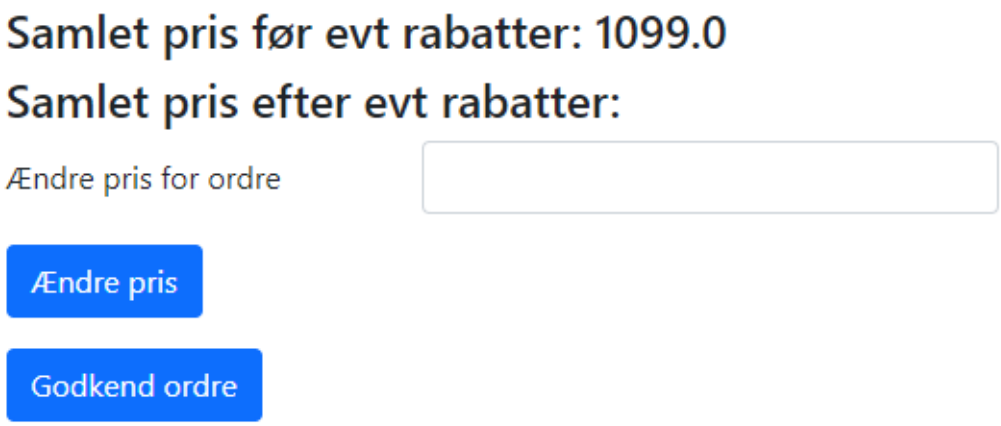
- Hvordan man på har valgt at lave brugerinput validering
    - Ved oprettelse af en carport anvendes “select” med forskellige options/muligheder.
      Brugeren kan altså ikke selv skrive ukorrekte oplysninger til sin carport forespørgsel, men kun vælge imellem nogle given muligheder.
    - Brugerinfo bliver valideret ved HTML input kun kan modtage typen tal/number, de steder, at tal forventes. F.eks. ved telefon nr.
    - Java tjekker at begge password matcher, at e-mail indeholder “@”.

- Hvordan man har valgt at lave sikkerhed i forbindelse med login
    - Password bliver gemt i databasen som en VARCHAR. Ikke i sessionen.
      Ideelt skulle de være hashet/krypteret.

      Forbindelsen til databasen burger ikke SSL kryptering. Det er derfor muligt at lytte og intercepte data.
      Dog er databasen localhost. Så det er i dette tilfælde ikke så relevant som hvis databasen havde været remote.

- Hvilke brugertyper der er valgt i databasen, og hvordan de er brugt i jdbc
    - Der er 2 brugertyper, employee og customer.
    - De er anvendt til sætte hvilke commands en given rolle har adgang til. Det sikre derfor en burger som er logget ind som customer, ikke kan tilgå en side som kun en ansat skulle kunne tilgå.

Udvalgt kode eksempel



For at kunne ændre prisen før en carport forespørgsel har fået tildelt et orderId, kræver det følgende kode:

```
<form action="${pageContext.request.contextPath}/fc/updatePrice" method="post">

    <div class="row mb-3">
        <label class="col-sm-2 col-form-label" for="price">Ændre pris for ordre</label>
        <div class="col-sm-3">
            <input id="price" class="form-control" type="number" step="0.01" name="price" value="Ændre pris:">
        </div>
    </div>

    <input type="number" hidden id="orderId" name="orderId" value="${sessionScope.orderId}">
    <input type="submit" value="Ændre pris" class="btn btn-primary">
</form>
```

På skærmbilledet står vi på siden orderHandler, hvor man som ansat kan godkende en forespørgsel.
Et af vores US var, at den ansatte skulle kunne ændre den udregnede pris baseret på materialet, sådan at der kunne gives unikke rabatter til en kunde.

Når den ansatte trykker på knappen “ændre pris” bliver siden opdateret og den nye pris får attribut navnet “newPrice”, som er sat i sessionscope.
Derefter bliver objektet “order” opdateret til at prisen skal være den værdi der står i “newPrice”.

```
@Override
public String execute(HttpServletRequest request, HttpServletResponse response) throws UserException {
    HttpSession session = request.getSession();
    Order order;
    order = (Order) session.getAttribute( s: "order");

    double price;
    try {
            price = Double.parseDouble(request.getParameter( s: "price"));
            session.setAttribute( s: "newPrice", price);
            order.setPrice(price);

    } catch (NumberFormatException e) {
        return pageToShow;
    }


    return pageToShow;
}
```

Derfra bliver man sendt tilbage til orderHandler siden, altså den side man allerede var på.
Feltet “Ændre pris for ordre” vil nu indeholde den nye pris, som kan ændres så mange gange det ønskes.

```
String token = request.getParameter( s: "token");
if (token.equals("5")) {
    double newPrice = 5;
    if (session.getAttribute( s: "newPrice") == null) {
        orderStatus=2;
        order.setOrderStatusId(orderStatus);
        order = orderFacade.carportToOrder(order);
        carportFacade.updateCarportStatus(orderStatus, carportId);
        request.setAttribute( s: "orderId", order.getOrderId());
        return orderListCommand.execute(request, response);

    } else {
        newPrice = (double) session.getAttribute( s: "newPrice");
        orderStatus=2;
        order.setOrderStatusId(orderStatus);
        order.setPrice(newPrice);
        order = orderFacade.carportToOrder(order);
        carportFacade.updateCarportStatus(orderStatus, carportId);
        request.setAttribute( s: "orderId", order.getOrderId());
        session.removeAttribute( s: "newPrice");
        return orderListCommand.execute(request, response);
    }

}
return pageToShow;
```

Når den ansatte trykker på “Godkend ordre” bliver ordren lagt i databasen med status 2, som indikerer at den nu er godkendt af en ansat.
Da newPrice denne gang ikke længere er null, vil “else” statementet i “if” sætningen blive kørt.
Endelig bliver det ordreId, som carporten bliver tildelt sat i requestScope, så det kan vises på siden. NewPrice bliver herefter fjernet fra session, sådan at de næste ordre den ansatte skal gennemgå, indeholder prisen fra den forrige carport.

Status på implementering

Med udgangspunkt i vores egne user stories og det, som var aftalt med product owner, nåede vi meget godt i mål.

Vi har implementeret de user-stories der står nævnt i krav under Scrum User Stories.

Der er nogle enkelte mangler som kunne være ‘nice to have’ på nogle user stories.

- I US-10 kunne et ‘nice to have’ være at brugeren kunne se prisen på carporten, før det blev sat i ordren. F.eks. At den skulle have status 3, som indikerer den var accepteret af brugeren.

General implementering status:

- Vi har ikke haft som mål at lave alle CRUD-metoder til tabellerne, men har oprettet dem efter behov.
- Alle siderne er stylet med bootstrap. Der kunne sagtens være lagt mere vægt på det visuelle, men det var en vurdering med produkt owner og i gruppen, at den tid ville være bedre brugt på funktionaliteten.
- Password er IKKE hashet, de bliver dog ikke gemt i sessionen. Kun i databasen som ren String/VARCHAR.
- I EER diagrammet fremgår tabellerne orderline, material, material_role samt unittype.
  Disse bliver i praksis ikke anvendt, da der ikke var tilstrækkeligt tid til at nå den de user stories hvori de blev involveret.
  Det er efter afstemning i gruppen blevet afgjort, at vi lod tabellerne være til at vise hvordan vi havde valgt at implementeret disse, såfremt de blev relevante til de pågældende user stories.
  Som set under EER-diagram har vi lavet et slutdiagram som det kom til at se ud, da vi havde implementeret den sidste user story.
- Materiale bliver KUN beregnet på træ.
  Det var først muligt at tilvælge skur på nogle hardcoded mål. Dette gik vi væk fra, da skuret ikke blev inkluderet i prisen, eller SVG tegningen.
- Da vi ikke anvendte test driven development var testing ikke fundamentet for kodeningen. Der er derfor kun testet i et meget begrænset omfang. Dvs. integrationstest mm. Mangler.
- Styklisten blev oprindeligt samlet i en klasse “BillOfMaterials” der bestod af klasser “Spær”, “Stolpe” osv. Dette design virker i praksis, men det virkede oplagt at lave denne stykliste som en Arraylist, da det er lettere at håndtere med loops i forhold til udskrivning af tabeller, og i brug af mapper-funktioner. Dette krævede en justeret beregning af stykliste, og også beregning af SVG.

Det endte med at lykkes, men da der var enkelte kommandoer der stadig anvendte den første metode, valgte vi at beholde begge stykliste-strategier.
- Der er endnu klasser der ikke har fået tilføjet exception handling.
- Der er ikke taget højde for, hvad der har været best practise i henhold til scopes. Det betyder, at der er primært er anvendt sessionScope, selvom requestScope i nogle tilfælde havde været tilstrækkeligt.

Test

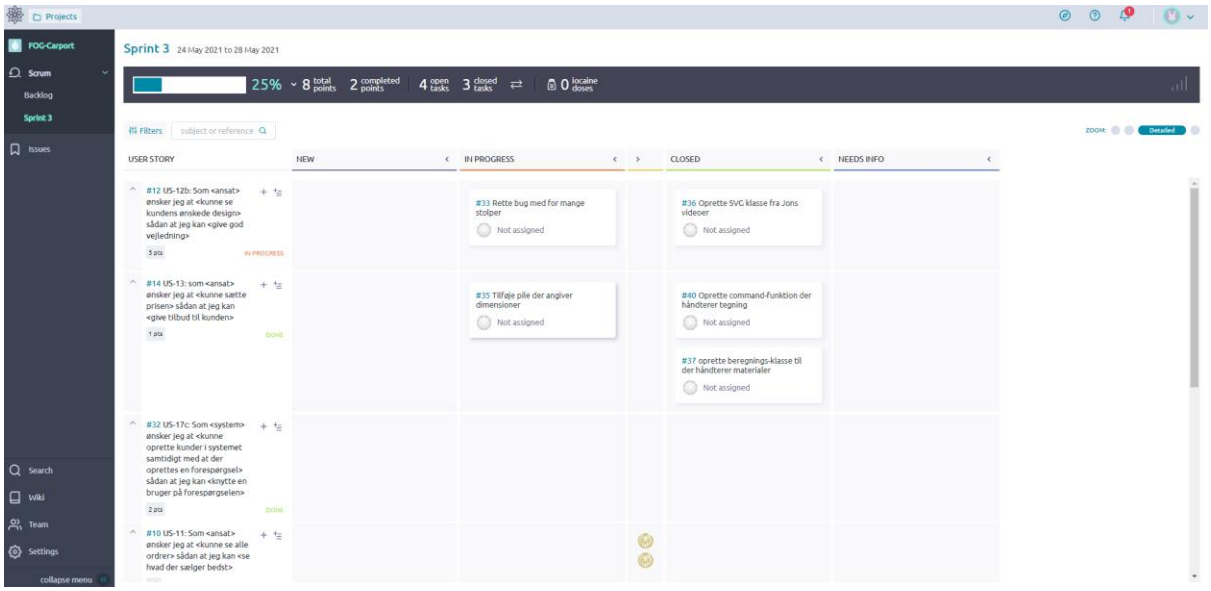
Der er lavet unit-test af funktionerne i klassen CalculateBom, som er den klasse der beregner stykliste.

Generelt har vi testet via browseren, og kontrolleret at der blev skrevet/ændret i databasen som forventet.

Der er lavet simpel integrationstest af “usermapper.login” og connection til databasen.

Proces

Klippet ud fra afsnittet Scrum user stories:



Gruppen har formået at få implementeret langt de fleste user stories.

Der er anvendt taiga.io til at håndtere de tre Scrum sprints, som projektet har været igennem.

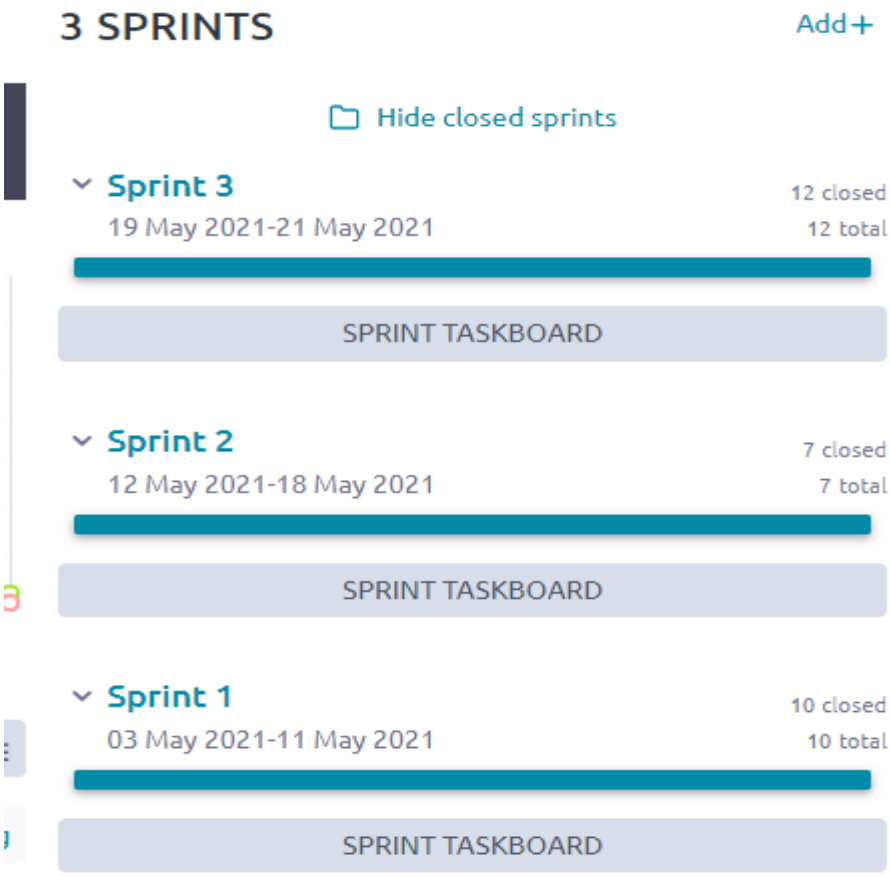
Hvert sprint fik tildelt 10 points i taiga. Hver userstory fik et bestemt antal point, alt efter hvor langt tid, det blev estimeret at den pågældende user story ville tage.

Hvert medlem fra gruppen har adgang til taiga projektet, hvor man kan assigne sig til en user story.
På den måde sikrede vi, at alle havde noget at lave, og alle vidste hvad de andre lavede. Derudover brugte vi zoom til video support, hvis der var samarbejde eller hjælp.

Hver hverdag mødtes gruppens medlemmer på zoom kl. 9.00. Hvor der blev lavet et daglig scrum meeting. Dette kunne nogle gange tage langt tid.

Arbejdsprocessen faktuelt

Der har i alt været 3 sprints i perioden.



- Sprint 1 03.05 - 11.05 arbejdede vi med US-4, US-5b, 12a & 12b.
- Sprint 2 12.05 – 18.05 US-17, US-5a.
- Sprint 3 19.05 – 24.05 US-12b, US-13, US-17c, US-11, US-10, US-6.

Officielt sluttede sprint 3 efter pinseferien, hvorefter gruppen havde afsat tid til rapportskrivning.

Scrum Master:

Der har ikke været én definitiv scrum master, gruppen ville gerne have en demokratisk holdning til beslutningerne.

Dog har René ubevidst fået lov til at tage den endelig beslutning en stor del af de gangene, hvor der har været uenighed.

Product Owner:

Der har været 3 møder i de 3 sprints.

Det første møde med product owner var kun planning.

De resterende har der været review, accepttest & planning.

Review går ud på at vise PO hvad der blev lavet i det sidste sprint, og planning er hvad der skal ske fremadrettet i det næste sprint.
Accepttest er PO godkendelse til planen.

Første møde med PO blev usercases lavet og accepteret. Sådan at vi i gruppen vidste præcist hvad kunden ønskede at se, næste gang vi skulle møde.
Det har gjort det muligt at bruge kræfterne på at udvikle de specifikke funktionaliter PO ville se.

Daily Scrum Meetings:

Gruppens har holdt daglige scrum meetings hver hverdag kl 9.00 om morgen.

Her var der obligatorisk mødepligt hvor gruppen lavede en status og løste evt. Konflikter & problemer.

Scrum Retrospectives:

Efter hvert møde med PO blev der lavet et Retrospectiv scrum. Hvor gruppen vurderede hvad der gik godt, hvad der var gået mindre godt og hvad man skulle fortsætte med.

Værktøj & arbejdsprocessen:

Taiga.io er anvendt til at håndtere vores user stories og de 3 sprints vi har været igennem.

Zoom har været anvendt til kommunikere og gjort det muligt at kommunikere med PO og hinanden i real-time.

Arbejdsprocessen reflekteret

Scrum master:

Det havde været lettest hvis der var en scrum master i gruppen, til at tage beslutninger så gruppens diskussioner kunne være blevet en del kortere.

Omvendt har alle gruppens medlemmer haft en indflydelse på beslutningsprocessen ved at bruge en mere demokratisk tilgang.

Retrospektiv møder:

De fleste US estimater var indenfor margin of error.
Der var dog 2 user cases, US-17 samt US-12b som tog længere tid end forventet. De blev derfor splittet op i nogle mindre US efter samtale med PO og overført til næste sprint, US-12b om at lave en dynamisk tegning af carporten, blev nedskaleret til IKKE at inkludere et skur. Dette var efter samtale med PO.

Andet:

En bedre intro til startkoden/skabelonen havde været dejligt.
Der blev brugt en del tid i gruppen på finde ud af hvordan skabelonen var sat sammen, f.eks. hvis man ville ændre headerens titel på en JSP side, men uden ændre i den genericpage.jsp som de alle arver fra.

Appendiks

US-1: Som <kunde> ønsker jeg at <kunne se en liste af standard carporte> sådan at jeg kan <se de forskellige carporte>

US-2: Som <kunde> ønsker jeg at <se den enkelte carport> sådan at jeg kan <se en detaljeret beskrivelse>

US-3 Som <kunde> ønsker jeg at <kunne sortere efter kriterier> sådan at jeg kan <filtrere carporte til kun at få vist dem som møder mine krav>

US-4: Som <kunde> ønsker jeg at <kunne vælge et design på en carport ud fra egne mål og
ønsker> sådan at jeg kan <kan designe min carport efter specifikke behov>
Givet at: jeg kan tilgå en side hvor der kan foretages bestilling
Når: kunden vælger et design og trykker send.
Så: sendes en ordre-forespørgsel der bliver gemt i en database.
Estimat: L

US-5a - Som <kunde> ønsker jeg at <kunne oprette en konto/profil> sådan at jeg kan <logge ind>
Givet at: kunden ønsker at oprette en profil.
Når: kunden udfylder brugerinfo og vælger samme password to gange.
Så: bliver kunden oprettet i systemet, og gemt i databasen.
Estimat: SUS- 5b - Som <kunde> ønsker jeg at <kunne logge ind> sådan at jeg kan <se og følge status på mine ordrer og foretage ordre-forespørgsel>
Givet at: kunden er logget ind.
Når: kunden udfylder brugernavn og password korrekt.
Så: kan kunden oprette ordre-forespørgsel og se status for ordre.
Estimat: S

US-6: Som <kunde> ønsker jeg at <kunne generere tegning af min carport> sådan at jeg kan <se hvad jeg har bestilt>
Givet at: kunden har sendt ordreforespørgsel
Når: når kunden er logget ind og står på loginsiden.
Så: kan kunden få vist et billede af den bestilte carport..
Estimat: S

US-7: Som <kunde> ønsker jeg at <få rådgivning og tilbud på den valgte carport> sådan at jeg kan <tage beslutning om at købe på et informeret grundlag>

US-8: som <kunde> ønsker jeg at modtage relevant dokumentation inkl. Stykliste monteringsvejledning på min carport> sådan at jeg kan <se hvordan jeg samler carporten>

US-9: som <kunde> ønsker jeg at <se en liste over fakturaer og se betalingsstatus> sådan at jeg <kan se hvor meget jeg skylder, så jeg kan betale>

US-10: som <kunde> ønsker jeg at <se detaljer for min ordre> sådan at jeg kan <se status og (evt. slette/ændre i ordren)>

US-11: Som <ansat> ønsker jeg at <kunne se alle ordrer> sådan at jeg kan <holde overblik og rette i ordre>
Givet at: man kan tilgår en ordre liste./Der er en ordre
Når: når ordren er accepteret.
Så: kan jeg opdatere og rette i ordre.
Estimat: M

US-12a: Som <ansat> ønsker jeg at <kunne se kundens ordredetaljer> sådan at jeg kan <give god vejledning>
Givet at: Der er en ordre-forespørgsel og jeg står på ordre siden.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg se ordre-forespørgsel med dimensioner.
Estimat: M

US-12b: Som <ansat> ønsker jeg at <kunne se kundens ønskede design> sådan at jeg kan <give god vejledning>
Givet at: Der er en ordre-forespørgsel og jeg står på ordre siden.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg se den tekniske tegning.
Estimat: M

US-12c: Som <ansat> ønsker jeg at <kunne se materiale listen> sådan at jeg kan <give et godt tilbud>
Givet at: Der er en ordre-forespørgsel og jeg står på ordre siden.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg se materiale listen.
Estimat: M

US-13: som <ansat> ønsker jeg at <kunne sætte prisen> sådan at jeg kan <give tilbud til kunden>

US-14: Som <ansat> ønsker jeg at <kunne rette i monteringsvejledningen> sådan at jeg kan <forbedre vejledningen efter kunde feedback>

US-15: Som <ansat> ønsker jeg at <have adgang til databasen> sådan at jeg kan <kan ændre vareinfo, tilføje varer, slette varer mm.>

US-16: Som <ansat> ønsker jeg at <få genereret en liste af materialer der er blevet solgt> sådan at jeg <kan automatisere opfyldning af lager>

US-17: Som <ansat> ønsker jeg at <kunne godkende en ordre forespørgsel og dermed oprette en ordre> sådan at jeg kan <gemme ordre detaljer>
Givet at: Der er en ordre-forespørgsel.
Når: kunden har sendt en ordre-forespørgsel.
Så: kan jeg godkende ordren så den oprettes i systemet
Estimat: M

US-17b: Som <ansat> ønsker jeg at <kunne omsætte styklisten ordrelinjer> sådan at jeg kan <sende ordren til ekspedering>
Givet at: Der er en materialliste og en ordreforespørgsel.
Når: kunden har sendt en ordre-forespørgsel.
Så: bliver ordrelinjer oprettet i systemet
Estimat: M

US-17c: Som <system> ønsker jeg at <kunne oprette kunder i systemet samtidigt med at der oprettes en forespørgsel> sådan at jeg kan <knytte en bruger på forespørgselen>
Givet at: At jeg som kunde kan oprette en forespørgsel .
Når: kunden sender en carport forespørgsel.
Så: bliver kunden og carport forespørgsel oprettet i systemet
Estimat: M