



# Fog Carport

Datamatiker 2. Semester - cphbusiness Lyngby

## Skrevet af

New Pied Piper - B klassen

Adam Lass - *Github*: [adamlass](#) - [cph-a1262@cphbusiness.dk](mailto:cph-a1262@cphbusiness.dk)

Nikolai Perlt Andersen - *Github*: [Perlten](#) - [cph-na130@cphbusiness.dk](mailto:cph-na130@cphbusiness.dk)

## Dato

01/06/2018

## Links

GitHub - [github.com/Perlten/FogCarport](https://github.com/Perlten/FogCarport)

Kørende Projekt - [159.89.19.132/FogCarport/](http://159.89.19.132/FogCarport/)

Javadoc - [perlten.github.io/FogCarport/](http://perlten.github.io/FogCarport/)

# Indhold

<b>Indledning</b>	<b>3</b>
Instruktion	3
Teknologi valg	4
<b>Baggrund og Krav</b>	<b>4</b>
Virksomheden	4
Krav	5
Arbejdsgange der skal IT-støttes	5
Ordrestyring	5
Styring af indhold	6
Kundebestilling	6
Scrum userstories	6
<b>Domæne model og ER diagram</b>	<b>8</b>
Domæne model	8
ER Diagram	9
Forslag til forbedringer	10
Normalisering	11
Dataallokeringer	11
<b>Navigationsdiagrammer</b>	<b>12</b>
Customer	12
Employee	13
<b>Sekvens diagram</b>	<b>14</b>
<b>Arkitektur</b>	<b>15</b>
LoginSample	15
Command pattern	15
3-lags arkitektur	16
<b>Særlige forhold</b>	<b>16</b>
Email-system	16
Event System	17
Exception Handling	18
Banner	18
Logging	19
Kunde navigation	19
Brugerinput	20
Ajax	20
Sikkerhed	21
Hashing	21
Salt	21
POST og GET	21
Password håndtering	22

<b>Udvalgte kodeeksempler</b>	<b>22</b>
Udregning af spær	22
Emailing og threading	23
<b>Fejl og mangler</b>	<b>25</b>
Forslag til forbedringer	25
<b>Kvalitetssikring</b>	<b>26</b>
Interne reviewing	26
Accept-test	26
Brug af Gestalt Love	26
Unit Tests	27
Integrationstest	28
Usability test	29
Hvorfor?	29
Hvordan?	29
Udførelsen og udbytte	30
<b>Process</b>	<b>31</b>
Sprint planning	31
Burndown charts	31
Scrum master	32
PO møde	33
Daily scrum	33
Definition of done	33
Retro spective	34
Vores indtryk af scrum	34
Positive	34
Sprints	34
Scrumwise	34
Negative	35
Scrum master	35
<b>Bilag</b>	<b>36</b>
Bilag 1 - Product backlog	36
Bilag 2 - Accept test	39
Bilag 3 - ER Diagram 1.0 (første udkast)	41
Bilag 4 - Navigationsdigram	42
Bilag 5 - Burndown Charts	43
Sprint 1	43
Sprint 2	44
Sprint 3	44
Sprint 4	45
Sprint 5	45

# Indledning

Fog træhandel har siden 1990'erne brugt et IT-system udviklet af en tidligere medarbejder til at beregne mål på carporte og styklister hertil. Systemet har tidligere været frakoblet deres hjemmeside og kørt på en gammel windows xp maskine, der ikke længere er supporteret. Da produkterne med priser ikke kan ændres har den totale pris været ubrugelig i mange år, hvilket gør systemet delvist ubrugeligt. Grundet dette har Fog brug for et nyt IT-system, der er bedre integreret på deres nuværende webside og som er mere fleksibel i forhold til prisstigninger og ordrestyring.

En af de centrale elementer i denne opgave, er udarbejdet af en beregner. Denne skal ud fra kundens specifikationer kunne producere en præcis stykliste med dertilhørende tegning. Ydermere, skal den kunne bruges på et lager samt i vejledningen til kunden. Herfra skal medarbejderne have adgang til at styre ordre og ændre på indholdet mm. Derudover skal projektet kvalitetssikres ved brug af fx. Integrations tests.

Kundens brugeroplevelse er i fokus. Vi har derfor valgt at denne skal være så simpel og elegant som mulig. Vi har gennemtænkt vores valg, omhandlende hvordan hele processen er skruet sammen og har ydermere tilføjet egenskaber til siden, der skal være med til at forbedre brugeroplevelsen.

## Instruktion

1. For at køre siden gå til: <http://159.89.19.132/FogCarport/>. Her kan du bestille en carport som en kunde.
2. For at logge ind som en medarbejder gå til:  
<http://159.89.19.132/FogCarport/employeeLogin.jsp> eller tryk på linket i det grønne banner.
  - a. Tryk på "Reset Admin User" hvis brugeren angivet på siden ikke virker.
3. For at se hvordan en ordre med mange *Events* ser ud gå her:  
<http://159.89.19.132/FogCarport/FrontController?command=LoadOrder&id=222>
4. For at logge ind på databasen (begrænset adgang):
  - a. IP: **159.89.19.132**
  - b. Brugernavn: **doorkeeper**
  - c. Password: **Fortado#420**

## Teknologi valg

- Netbeans 8.2
- Tomcat 8
- Mysql
- Git
- Bootstrap 4
- Java EE 7 web
- Jsp sider
- MySQL Workbench
- Ubuntu server på DigitalOcean

## Baggrund og Krav

### Virksomheden

Fog er en dansk træhandel som sælger både til erhverv og privat. En del af deres virksomhed består af at sælge gør-det-selv carporte til kunder, som herefter selv skal bygge dem eller hyre en håndværker til arbejdet. Fog benytter deres eget IT system til varestyring. Dog har de også behov for andre IT-løsninger, eksempelvis vores, til at håndtere specifikke ordre, deres nuværende system ikke understøtter. Med hensyn til bestilling af carporte fungerer virksomheden således, at en salgsmedarbejder hjælper kunden gennem ordreprocessen. Dette sker ved en fagperson, såsom Martin Kristensen, i mellemtiden kontakter kunden omhandlende de tekniske detaljer i forbindelse med ordren. Fog ser ikke blot deres produkter, såsom brædder og skruer til byg af carporten, som værende den samlede service de sælger til kunden. De mener også, at den vejledning kunden modtager, både via ordreprocessen og samle vejledningen, er en stor del af servicen. Kunden betaler dermed ikke blot for materialer, men også for den service Fog udøver under handlen. Grundet dette er det vigtigt for Fog, at deres nye IT-system til håndtering af disse handler understøtter-, og måske endda forbedrer- denne process.

## Krav

- Kunden skal kunne bestille en carport med angivne mål og kontaktoplysninger.
- Det skal være muligt at tilpasse priser på produkterne.
- En kunde skal først modtage stykliste når carporten er købt og betalt, da dette er en del af den service Fog sælger.
- Det skal være muligt for en medarbejder at se og redigere i en foretaget ordre.
- Det skal være muligt for en medarbejder at tilpasse prisen på carporten, i ordren, inden godkendelse.

De resterende krav til systemet kan findes i form af vores [Product Backlog - Bilag 1](#).

## Arbejdsgange der skal IT-støttes

I det følgende vil de arbejdsgange der skal IT-støttes blive listet. Efterfølgende vil vi liste disse med implementationen af vores IT system.

### Ordrestyring

Vi vil i det følgende redegøre for, hvordan ordrestyringen finder sted i vores udarbejdede produkt. En salgsmedarbejder skal kunne udarbejde simple CRUD operationer på alle ordre i systemet. Når en ordre bliver indsendt af en kunde, skal den lande i en liste med de nyeste øverst. Efterfølgende skal salgsmedarbejderen kunne få en oversigt over de inputs kunden har lavet. Dette inkluderer en pris-oversigt. Medarbejderen skal under denne kunne ændre i ordrens indhold, således at den kan blive tilpasset, såfremt der er ønsket ændringer fra kundens efter indsendelse af ordren. Sidst men ikke mindst skal en stykliste være tilgængelig for lagermedarbejderen, således at han kan samle produkterne til kunden.

#### *Efter implementation:*

I det nye IT-system ønsker vi at tilføje en måde, hvorpå en medarbejder kan bekræfte en ordre. Grunden hertil er, at vi ønsker, kunden skal kunne følge med i ordreprocessen.

Ydermere ønsker vi at gøre styklisten printer- og tabletvenlig.

### Styring af indhold

En admin skal på siden have adgang til at lave CRUD operationer på de "styling" muligheder der bliver vist for kunden (f.eks. beklædningsstypen).

#### *Efter implementation:*

Under implementationen tilføjer vi en mulighed for, at ændre på de enkelte style valg og disses priser, således at prisen også kan ændres ud fra stylingen.

## Kundebestilling

En kunde skal kunne bestille en carport via siden med sine mål på carport og kontaktoplysninger.

*Efter implementationen:*

Vi tilføjer under implementationen muligheden for, at vælge sin styling direkte, for på den måde at få et overblik over ordren. Dette overblik indebærer en estimeret pris som sendes på mail med et link til en reference side.

## Scrum userstories

Her vil vi udvælge et par userstories og forklare, hvorfor vi har sat dem op på den måde, de er sat op på.

User Story	How to demo	Subtasks	Est. hours
<i>As a salesman I want to be able to edit and confirm a request/order so that I can make changes after talking with the customer.</i>	<i>Take a existing order then make a change to the size of the carport.</i>	<ul style="list-style-type: none"><li>1) <i>CRUD</i></li><li>2) <i>Logic Facade</i></li><li>3) <i>Edit functions in frontend</i></li><li>4) <i>Be able to unconfirm order and limit edit to unconfirmed orders</i></li></ul>	<b>3</b>

Denne userstory er skrevet for at sikre, at en medarbejder kan redigere og bekræfte en ordre efter at have talt med kunden. En skræddersyet carport ordre hos Fog går nemlig ikke bare direkte igennem til betaling. En medarbejder skal først bekræfte at ordren er godkendt i forhold til de valgte mål. Vores How-To-Demo beskriver, hvordan en ændring af carportens størrelse kan verificere funktionaliteten. For at implementere denne funktionalitet skal der skrives en DataMapper, der kan opdatere ordren med de nye mål. Derudover skal der skrives frontend der via *LogicFacade* og *DataFacade* snakker sammen med vores DataMapper. Efter implementationen tilføjede vi en feature, der gjorde, at man kan *Unconfirm* en ordre, og at man ikke kunne trykke *Edit* i en ordre der er *Confirmed*.

Denne userstory blev estimeret til 3 timers arbejde, før at subtask 4 blev tilføjet, hvilket vi overskred med ca. 5 timer.

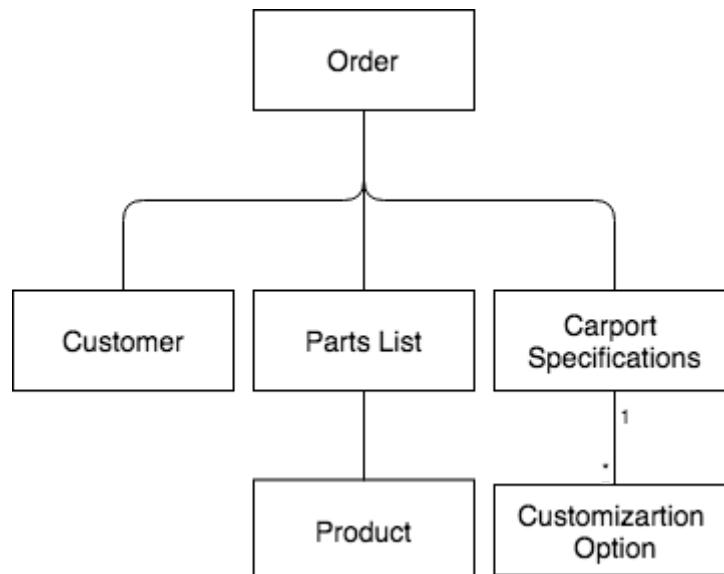
<i>As a customer i want to receive an email with a reference link when i submit my request so that i can track my request.</i>	<i>Getting an email once requested and then going to a reference page with information via the link provided.</i>	<i>1) make method to send email 2) make fog email</i>	<i>1.8</i>
--	---	---	------------

Vi tilføjede denne userstory for at sikre, at kunden ville modtage en mail med ordre bestillingen, for herefter at kunne vende tilbage til ordreferencen. Denne feature var vigtig i forhold til reference siden med events, som kunden skal vende tilbage til og tjekke status på. En How-to-demo er i dette tilfælde ret simpel, da man som kunde bare skal modtage en mail med et gyldigt link ved bestilling. Da vi estimerede denne opgave på tid, havde vi ikke kendskab til hvordan man sendte mails i Java, derfor antog vi bare at det var forholdsvis let, og at vi kunne finde en form for SMTP API der kunne hjælpe os her.



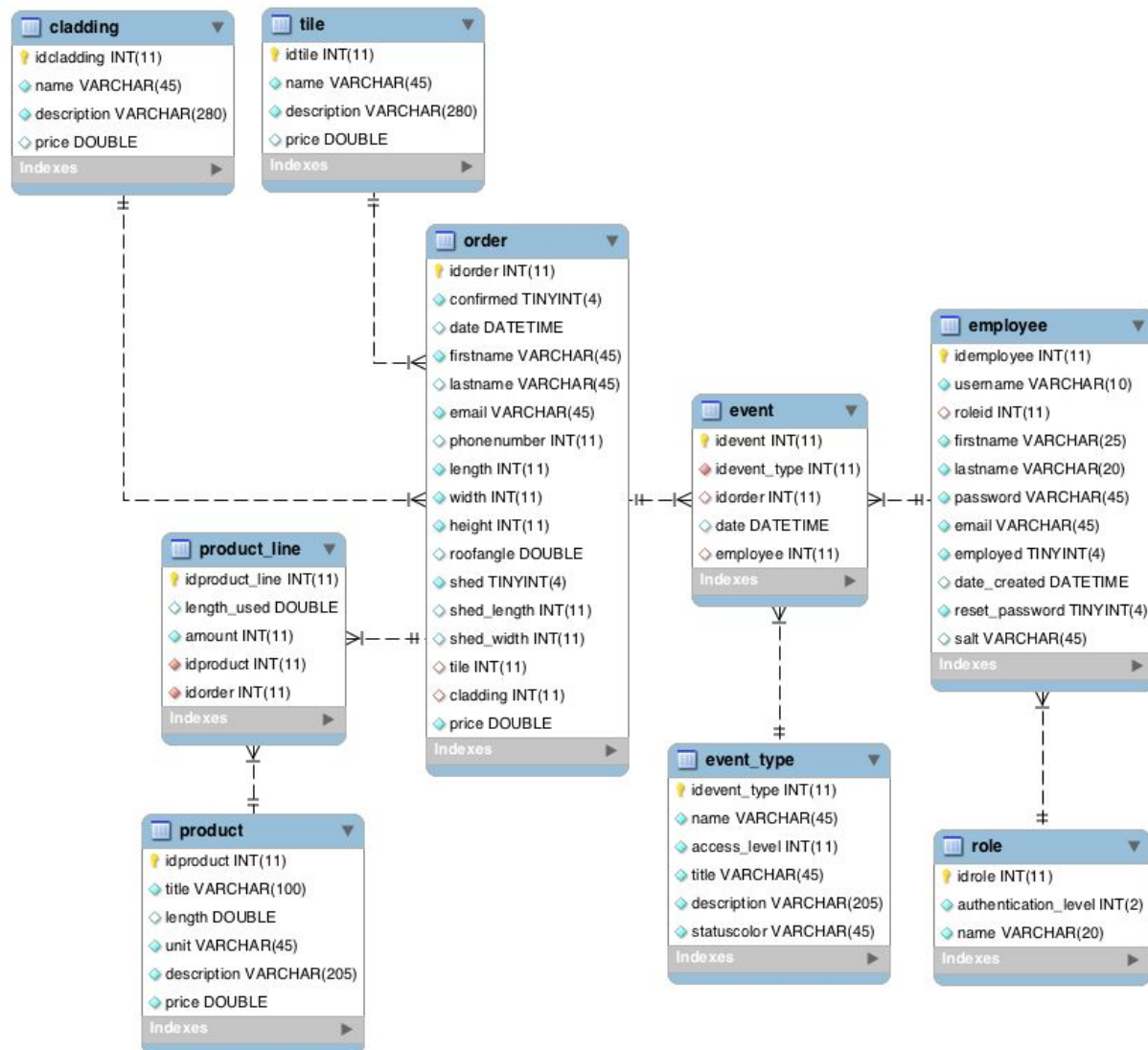
## Domæne model og ER diagram

### Domæne model



Vi ser ordre som den centrale del af Fogs domæne, som vi arbejder med. Relationer som kunder, styklister og specifikationer på carporte kan alle sætte i relation til ordren. En *Customer* bestiller en carport ved at lægge en *Order*. Her indtaster kunden *Carport Specifications*, vælger 1 til mange *Customization Option(s)* og indtaster sine personlige oplysninger. Ud fra *Carport Specifications* bliver en *Parts List* genereret, hvortil denne *Parts List* indeholder *Products*. Ovenstående var vores initiale forståelse af domænet. Vi har sidenhen lært at en *Employee* har en relation til *Order* da en eller flere *Employee(s)* kan være involveret i håndteringen af en *Order*. Ydermere fandt vi også frem til, at der findes en 1:1 relation mellem *Product* og *Customization Option*. Dette ses eksempelvis ved, at et bestemt valg af beklædningstype, har en direkte relation til et produkt af den type træsort med en pris osv..

## ER Diagram



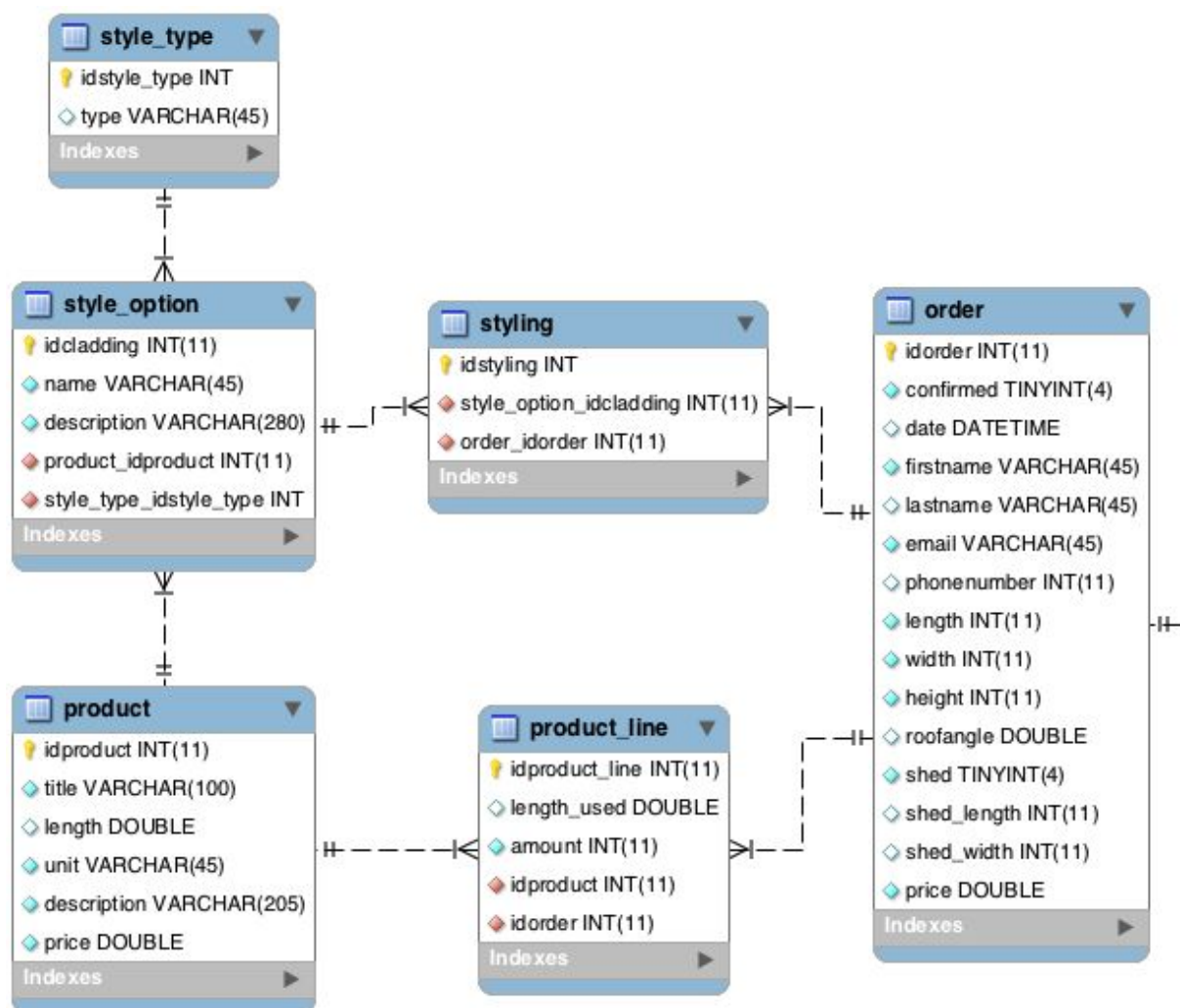
Ordre er som tidligere skrevet den centrale del af systemet. Derfor ses den også her som et central element af databasen. Vi startede med at oprette mange individuelle tabeller som customer, customization og shed, da vi lavede databasen. Vi indså her, at de alle har en 1:1 relation til *order* og derfor burde samles i én tabel (se [bilag 3](#)). Da vores system ikke skal designes til at være runtime-effektiv, er dette også en ekstra grund til, hvorfor en samling af alle tabeller med 1:1 relationer til *order* i *order* er det mest optimale.

Ift. *Customers* 1:1 relation til *order* valgte vi at designe systemet således, at en kunde ikke skal behandles som en bruger, der skal logge ind ved bestilling af carporten. Grunden til dette er, at det ikke er optimalt, at en kunde skal oprette en bruger og logge ind for at bestille

en carport, da kunden måske kun gør dette ca. 1-2 gange i sit liv. Vi har derfor valgt at lægge kundeoplysningerne direkte på ordren som felter der skal udfyldes i forbindelse med bestillingen, i stedet for at have en separat tabel med kunder.

## Forslag til forbedringer

I Bilag 3 ses en anden form for opsætning af *customization\_option*. Denne er sidenhen blevet til *StyleOption* i koden, grundet det faktum at den er mere dynamisk i forhold til oprettelsen af nye customization-typer. Efter lange diskussioner omkring designet i starten af projektet, besluttede vi at gå med en anden opsætning, hvor hver type af customization option havde sin egen tabel. Dette skyldes, at vi havde en formodning om, at der i forhold til vores stykliste-udregner ville dukke nogle attributter op, som var specifik for typen af customization\_option, og på den måde fjerne den homogene relation mellem typerne. Vi fandt senere ud af, at denne *forudsigelse* ikke blev en realitet, og at det ville være mere optimalt at lave følgende opsætning:



Dette er den samme opsætning som i *bilag 3*, med undtagelse af anden navngivning og nogle få forbedringer. I denne mere dynamiske opsætning, med brug af en mellem tabel, har vi givet *style\_option* en direkte relation til et produkt, og fjernet prisen. Som beskrevet ovenfor under afsnittet *Domæne model* har en *style\_option* en direkte relation til *product*. Det betyder også, at prisen af stylingen burde afhænge af, hvilket produkt/materiale der er valgt, og som udgangspunkt ikke burde ligge på selve *style\_option* tabellen, som den gør nu. Denne ændring af databasen når vi ikke at implementere, og den vil derfor indgå som et punkt under afsnittet *fejl og mangler*. Implementering af denne ændring er i teorien relativ simpel, da vores arkitektur har en meget lav kobling som effekt af vores 3-lags deling med tilhørende facader.

## Normalisering

Vi gjorde brug af de tre normalformer, da vi designede databasen. Dette var for at reducere redundante data og for at sikre nem vedligeholdelse af databasen.

1. Normalform/**1NF** kan ses opfyldt i designet af alle vores tabeller. De har alle en unik identifikator i form af en primary key, alle felter holder kun atomiske og ensartede værdier, alle kolonner har unikke navne og rækkefølgen ved input af data er ikke relevant.

**2NF** ses opfyldt ved *product\_line* tabellen, hvor *amount* og *length\_used* begge har en relation til den sammensatte primary key i tabellen.

**3NF** kan eksempelvis ses i ved *employee's* tilhørende *role* tabel. Ved at bruge en tabel til *role*, undgår man redundante data i *employee* tabellen, samtidigt med at man knytter *authentication\_level* til selve rollen.

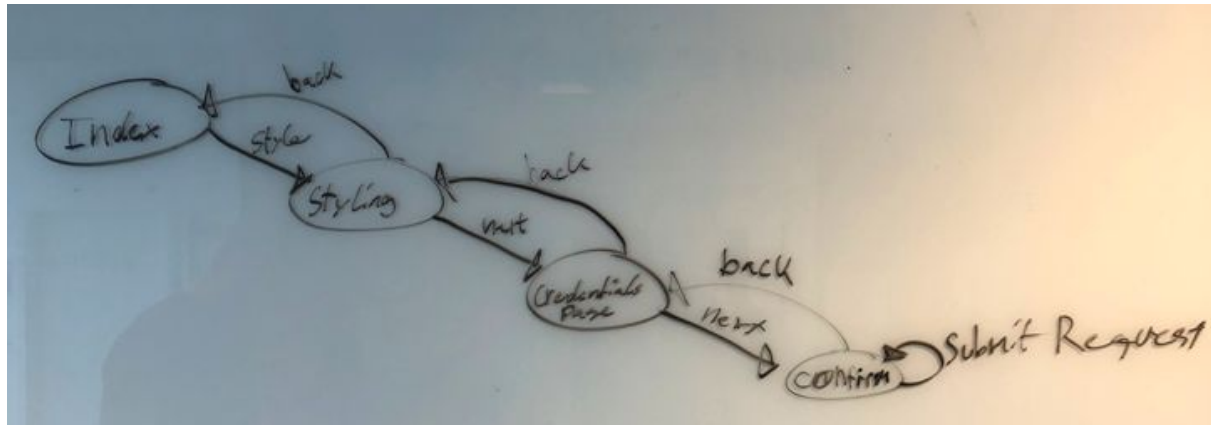
## Dataallokeringer

Da vi oprettede vores *employee* tabel, gjorde vi ekstra ud af at definere data allokeringerne for de enkelte datatyper. Dette gjorde vi ved at begrænse pladsen til hvert felt, til hvad vi estimerede var nødvendigt. Hvis en bruger giver et for stort data input, vil der komme en *SQLException* der kan fortælle en udvikler, hvilket felt der er allokeret for lidt plads til. Ved at begrænse sig til de allokeringer man skal bruge, spares der en del serverplads på længere sigt. Det ses især når systemet efterhånden bliver fyldt med større mængder data.

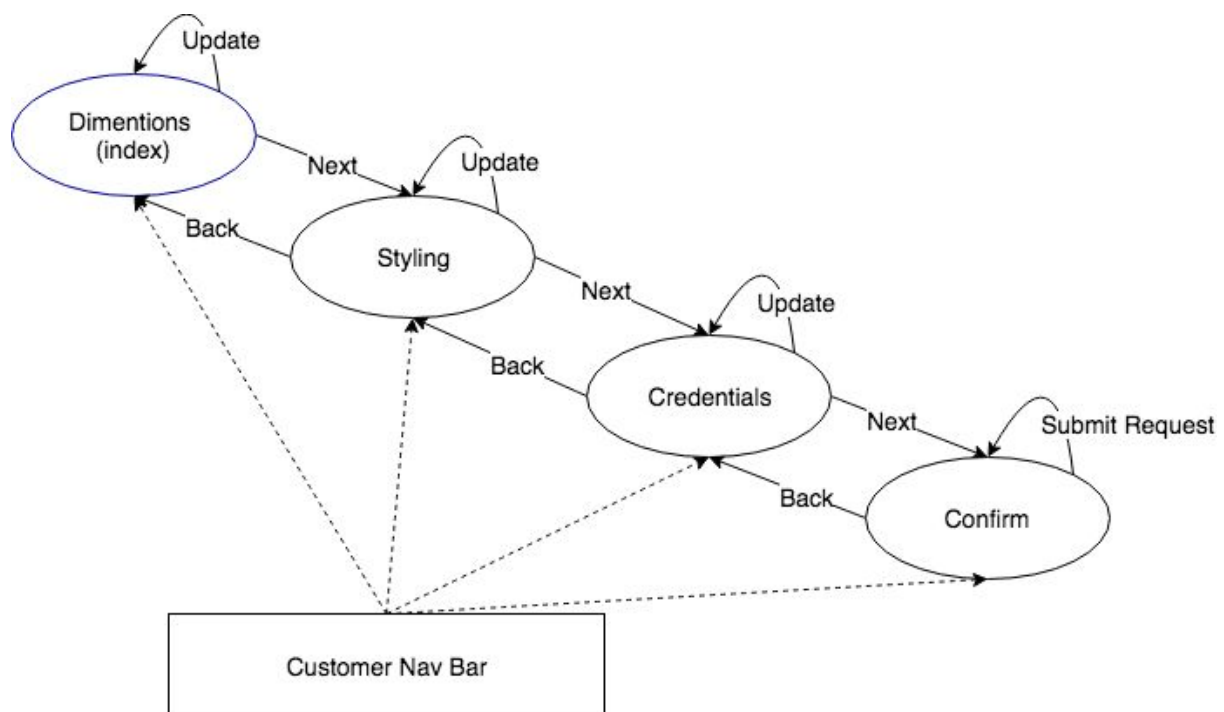
# Navigationsdiagrammer

Vores website er opdelt i en Kunde og Medarbejder del. Vi har derfor to navigationsdiagrammer for systemet.

## Customer

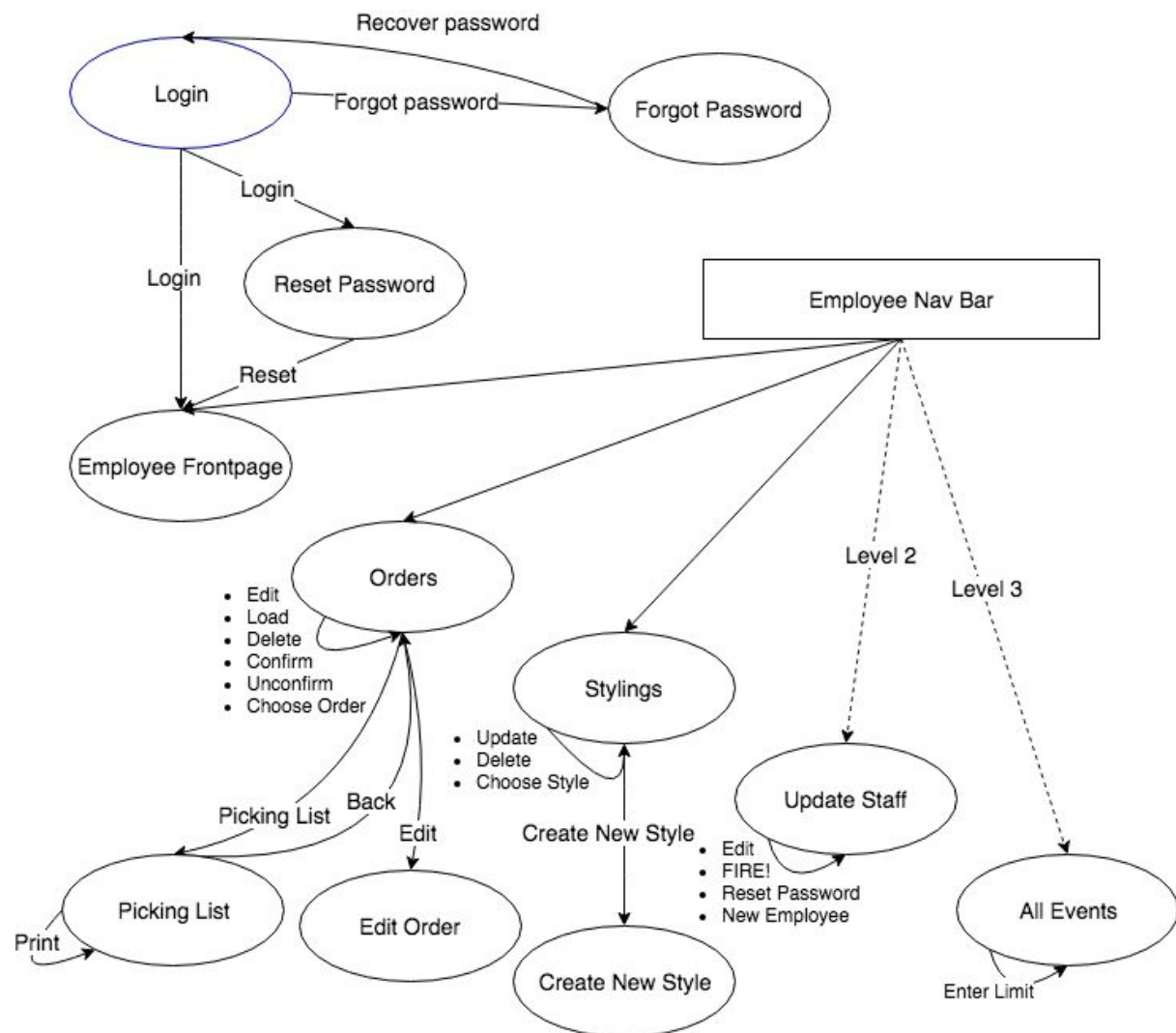


Hvis man kigger på ovenstående udkast til navigationen på kundesiden, kan det ses, at vi i store træk har fulgt denne. Udkastet blev lavet som en del af den indledende fase af projektet. Her diskuterede og analyserede omfanget af projektet. I det færdige produkt kan det ses, at vi under udviklingen har opfyldt de mål, vi havde sat os til at begynde med. Vi har dermed kun tilføjet knappen *Update* og en navigationsbar:



De stiplede pile fra Customer Nav Bar indikerer at siden kun kan tilgås, såfremt det er muligt  
(Se afsnittet [Særlige Forhold - Kunde Navigation](#)).

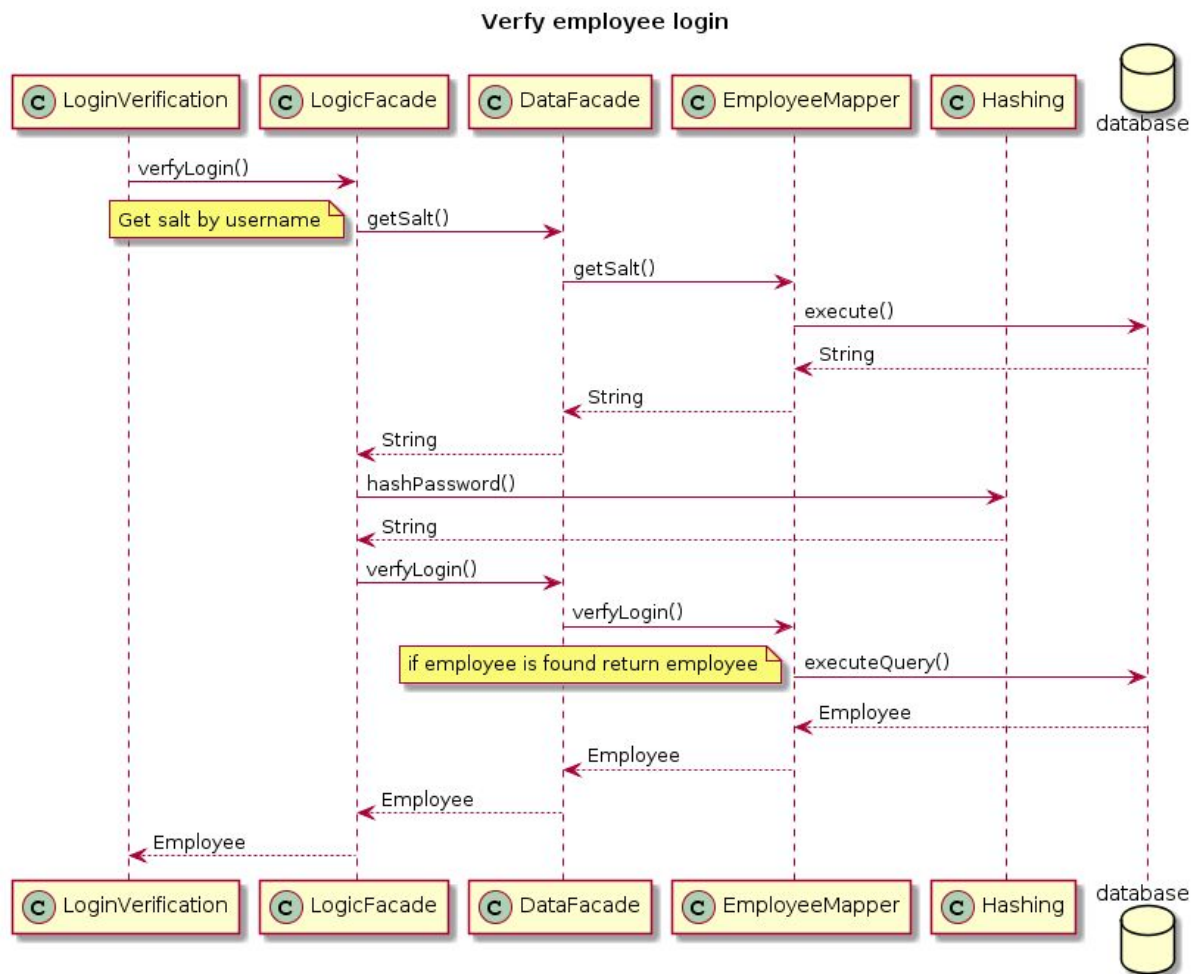
## Employee



Loginsiden tilgås via [159.89.19.132/FogCarport/employeeLogin.jsp](http://159.89.19.132/FogCarport/employeeLogin.jsp). Herfra kan en medarbejder logge ind i systemet og arbejde. Ydermere er det her også muligt at nulstille sit password, samt at logge ind. Hvis brugeren skal nulstille sit password ved næste login, kommer man forbi *Reset Password* siden, hvor man kan vælge et nyt password. Når brugeren er logget ind lander man på *Employee Frontpage*, hvor man også får adgang til *Employee Nav Bar*. Her kan man som udgangspunkt tilgå *orders* og *styling*. Har medarbejderen en rolle med tilhørende *authentication\_level* på 2 eller over, får personen adgang til *Update Staff*. Hvis medarbejderen er på *authentication\_level* 3 eller over, gives der her adgang til *All Events*.



## Sekvens diagram



Ovenstående sekvens diagram viser processen, der sker, når en medarbejder indtaster et brugernavn og et password. Systemet skal efter indtastningen kontrollere, at der findes en medarbejder med det angivne brugernavn og password. Denne sekvens illustrerer meget godt vores 3 lags arkitektur, hvor vi fra *presentation layer* kun skal kalde en metode i *LogicFacade*. Når denne er kaldt, henter metoden både data fra vores databaser og behandler denne i vores *Hashing* klasse.

# Arkitektur

## LoginSample

Dengang vi startede med at skrive vores projekt, forked vi, på opfordring af vores vejleder, vores lærers skabelon for, hvordan man kan skifte mellem sider.

Vi har siden ændret store dele af den oprindelige kode. Vi har ændret måden, hvorpå *command pattern* bliver sat op på. Vi har ændret i *Connection* for at gøre det lettere tilgængeligt at vælge og skifte mellem *connections*, med henblik på nemmere at kunne teste vores program. Den eneste klasse hvor der er sket minimal ændring er *FrontController*. Vi har dog tilføjet *logging* hvilket sker i *FrontController*, hvilket vi vil komme mere ind på [senere i vores rapport](#).

## Command pattern

Når man taler om *arkitektur*, kan man ikke lade være med at tale om *command pattern*. Vi gør stor brug af dette design mønster, da vi bruger det næsten hver gang vi skifter side og/eller manipulerer vores data. Det kan både være i vores database eller data vi bruger på siden til fx. at vise en ordre.

Men hvad er *command pattern*?

For at have et *command pattern* skal man bruge to typer objekter; et *command object* og et *invoker object*. Vores *Command* klasse indeholder en metode kaldet *execute()* som tager imod et *HttpServletRequest* og *HttpServletResponse* i dens parameter. Dette kan bruges til at indsætte data, som kan hentes på jsp siderne. Metoden *execute()* returnerer en *String*, som er navnet på den side brugeren skal sendes til. Udover at returnere en *String* kan vi bl.a. bruge vores *logic facade* til at hente data op fra databasen samt skrive data ned på databasen. Vi kan også kontrollere *brugerinput* og kaste *exceptions*, hvis der er brug for det.

Den anden del i vores *command pattern* er vores *Invoker*. *Invokeren* indeholder et *hashmap* med alle vores *commands*, samt metoden *from()* som tager imod *HttpServletRequest* i dens parameter. På den måde kan vi, når der laves en form på vores jsp sider, lave en hidden input med navnet *command*, og som indeholder navnet på den key i vores *hashmap*, som indeholder det *Command object*, vi vil benytte. Vi kalder *from()* fra en *servlet* vi har valgt at



kalde *FrontController*. Når vores command er fuldendt returnerer den en *String* i form af en jsp side, som vi kan forwarde til.

Endnu en fordel ved at bruge Commands er, at vi kan stakke vores commands. På den måde kan vi i stedet for at returnere en String, instantiere en ny instans af en Command, som eksemplet herunder viser. På denne måde undgår vi at gentage kode i rigtig mange commands.

```
return new GetOrdersPage().execute(request, response);
```

## 3-lags arkitektur

Vores system er delt op i tre lag så der nemt kan blive skiftet dele af systemet ud uden at vores arkitektur skal ændres på. De tre lag lyder således:

1. Datalag
2. Forretnings-/Logiklag
3. Præsentationslag

Vi har sørget for, at koblingen mellem de tre lag er så lav som mulig. Måden hvorpå dette er gjort, er ved hjælp af facader. Vi har vores DataFacade som kobler datalaget og logiklaget sammen, og LogicFacade der kobler logiklaget og præsentationslaget sammen. På grund af projektets størrelse har dette bidraget til bedre strukturering, hvilket virkelig har hjulpet på vores overblik over projektet. Samtidig har det gjort os i stand til at implementere og ændre features forholdsvis simpelt.

## Særlige forhold

### Email-system

Det at kunne sende mails, er en væsentlig del af vores projekt, idet al kommunikation, både med kunder og medarbejdere, sker via disse.

Kunden modtager en mail hver gang han opretter en ny ordre. Denne mail indeholder en besked, hvor firmaet takker ham for at han har valgt FOG. Ud over denne besked indeholder mailen også et link han kan trykke på. Linket fører ham over på en side, hvor der er et overblik over hans ordre. Siden er forbundet til kundens *ordre id* så hver gang siden refresher, opdaterer siden med det nyeste data fra ordren. Denne funktion er nødvendig, idet

en medarbejder bl.a. kan ændre i ordren, godkende ordren mm. Derudover skal kunden kunne følge denne proces undervejs.

Medarbejderne gør også brug af mails. Når en ny medarbejder skal oprettes, indtaster en admin medarbejderens navn, email, og brugernavn. Admin indtaster dog ikke et password. I stedet bliver et tilfældigt password sendt ud til den nye medarbejders mail. Dette password skal den nye medarbejder benytte som password første gang medarbejderen logger ind. Samme princip gør sig gældende såfremt en medarbejder har glemt sit password.

I det fleste sammenhænge kan vi nøjes med at sende en enkelt mail, men ikke altid. Når en kunde opretter en ordre, vil vi udover at sende en mail til kunden selv, også sende en mail til alle aktive medarbejdere i systemet, med en besked om, at der er kommet en ny ordre. Det skaber dog et problem, da det tager tid at sende disse mails. Kunden vil derfor være på en side og vente på, at den næste side loader grundet disse medarbejder mails. Måden vi løser ventetiden på, er ved at køre hele mail processen på en separat thread. På den måde kan brugeren frit navigere rundt på vores sider, mens vi sender e-mails i baggrunden.

## Event System

For at Fog som virksomhed skal være bedre i stand til at se hvad deres medarbejder og kunder laver på siden, har vi implementeret et event logging system. Hver gang en medarbejder logger ind, ændre en ordre, sletter en ordre, eller på anden vis manipulere på data, vil det blive gemt i vores system. Det betyder at en admin kan se, hvad en specifik medarbejder har lavet, og hvilket tidspunkt det er forekommet på.

Vi bruger det også til at kommunikere ordre ændringer til brugeren. Hver gang en medarbejder ændrer på en ordre vil ordrens reference side afspejle dette, som set nedenfor.

**Order Confirmed**  
Your order has been confirmed by one of our employees. You will be contacted regarding the payment soon.

**Order updated!**  
Your order has been updated with new specifications.

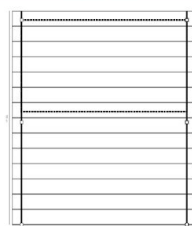
**Order has been unconfirmed**  
Your order was unconfirmed by one of our employees. Please contact our support team for more information!

**Order Confirmed**  
Your order has been confirmed by one of our employees. You will be contacted regarding the payment soon.

**Order updated!**  
Your order has been updated with new specifications.

**Request Submitted!**  
You have received an email with the details of your order. We will contact you once we have processed your request.

**Admin's Carport Request**  
Est. Price: **6420.0 DKK**  
Requesting 100%



**Dimensions**

Length	Width	Height	Roof angle	Shed Length	Shed width
700	600	210	12.0	300	540

**Styling**

Tile	Cladding
Tegula	Pine

**Contact Information**

Firstname	Lastname	Email	Phonenumber
Admin	Admin	FogCarportAdm@gmail.com	12345678

Måden vi har implementeret dette på, gør det nemt at skalere og dermed logge flere brugere og medarbejder handlinger.

## Exception Handling

Der bliver kastet exceptions fra mange steder i vores projekt. Både fra vores Commands og fra vores mappers. Vi har valgt at lave vores egen exception som vi kalder *FOGException*. Dette gør vi fordi, det giver os friheden til nemt at skifte elementer ud i vores program uden at skulle ændre på måden vi håndterer exception. Et eksempel herpå kunne være, hvis vi ville ændre på vores datalag så i stedet for MySQL ville vi bruge en anden form for database, så skulle alle metoder ændres fra at have en throws clause der kaster SQLExceptions til f.eks. en SQLite exception. Når vi i stedet laver vores egen exception, kan vi i vores mappere fange den exception, som kunne forekomme, og i stedet kaste en FOGException. Alle vores exceptions bliver til sidst fanget og håndteret i vores FrontController. Udover den funktionalitet der kommer af, at vi arver fra Exception, har vi også tilføjet ekstra funktionalitet. Vi har lavet en constructor der tager imod en fejlbesked og et *Level*, som senere bliver brugt i *logging*.

## Banner

Når en exception er fanget, vil vi gerne fortælle brugeren om, hvad der er sket. Det første der sker i denne process, er at vores FrontController gemmer vores fejlbesked i session under navnet error. Derefter sender den os videre til error.jsp. Efterfølgende kører error.jsp

et script, der først laver en cookie med variabelnavnet `refresh` og sætter værdien til `true`. Derefter kalder vi metoden `window.history.back()`. Denne metode sender os tilbage til den side, som brugeren sidst var på. Grunden til vi opretter en cookie er fordi, `window.history.back()` sender brugeren tilbage til forrige side, i det samme stadie som før. Det vil sige, at siden ikke kan se at der er oprette en ny error besked på session, og vi bruger derfor cookien til at fortælle siden den skal refreshe så den er opdateret med den nyeste information. Vi indsætter derefter et banner i toppen af siden med vores fejlbesked, som forsvinder hvis brugeren skifter side eller trykker på bannerets kryds.

## Logging

Når vi fanger en exception i `FrontController` skriver vi den korte fejlmeddelelse (fra `.getMessage()`) på vores linux server på stien `/logging/Fog.log`. For at projektet også kan køre lokalt tager programmet udgangspunkt i at vi kører lokalt, og tjekker herefter om filen findes ved at oprette et `File` objekt på stien og køre metoden `.exists()` der returnerer en boolean således:

```
String path = Conf.LOGDEVFILEPATH;

if (new File(Conf.LOGFILEPATH).exists()) {
    path = Conf.LOGFILEPATH;
}
```

Når vi har logget fejlen sørger vi for at lukke alle handlers så der ikke opstår fejl ved næste logging. På vores logs bruger vi den *message level identifier* der er gemt i den kastede Exception som `level`. På denne måde får vi en status, på hvor vigtig hændelse er, gemt sammen med vores fejlkode.

## Kunde navigation

Vi har valgt at lave en dynamisk kunde navigations bar der ved hjælp af et Boolean `HashMap` holder styr på, hvilke tabs man kan navigere mellem i bestillingsprocessen af en carport. Selve `HashMappet` bliver styret af de forskellige commands, så når kunden f.eks. indtaster carportens dimensioner, får man lov til at gå til den næste side *styling*. Når man til sidst trykker “submit order”, bliver muligheden for at gå tilbage til de andre tabs fjernet.

## Brugerinput

Vi validerer alt brugerinput, og dette gør vi på mange måder. Den første validering der kan ske er på vores jsp side med JavaScript. Valideringen bruges fx. hvis brugeren skal indtaste nogle mål, og vi på forhånd har bestemt en max værdi. Det gøres direkte i et <input> tag med nøgleordet "max".

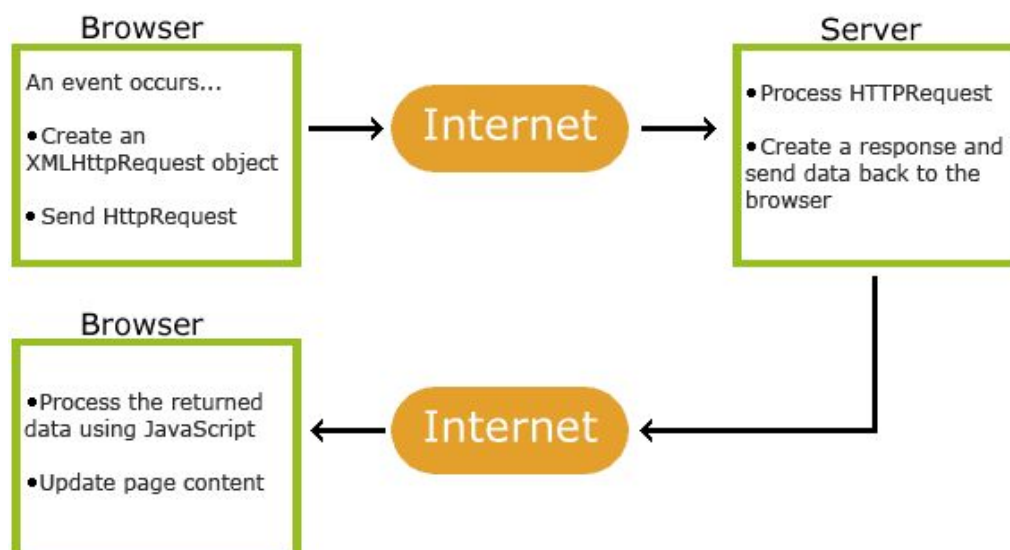
En anden form for brugervalidering kan foregå i en af vores Command objekter. Her kan der testes for hvorvidt brugeren overhovedet har indtastet noget i feltet, eller om ordren på en anden måde ikke stemmer overens med hvad vi kan acceptere. Hvis det er tilfældet, kan vi blot kaste en FogException og derefter tager vores Exception håndtering over.

Den sidste form for validering der kan foretages, er ved hjælp af prepared statements. Hvis vi på nogen måde skal gemme data i vores database der kommer som brugerinput, kører vi det igennem et Prepared Statement. Dette gøres for at undgå SQL injection, og Cross-site scripting.

## Ajax

Når vi skal hente en oversigt fra databasen, og vise den til en medarbejder, er det ikke effektivt at hente og vise dem alle på en gang. Grunden til dette er, at det vil tage utrolig lang tid, når der begynder at være mange ordre. Måden hvorpå vi har valgt at løse denne udfordring er ved hjælp af ajax.

Ajax gør os i stand til at hente data fra vores server, generere html, og derefter opdatere siden uden at den skal genindlæses.



# Sikkerhed

## Hashing

Vi er meget forsigtige med hvordan vi håndterer Fog's medarbejders password. Når en medarbejder indtaster sit nye password, hasher vi det i vores logiklag, inden vi gemmer det i vores database.

Måden vi hasher på, er ved hjælp af sha-1. Sha-1 hasher til 160 bit eller til 20 byte i form af et byte array. Vi omdanner derefter dette byte array om til heximal tal, som bliver gemt på databasen.

En af vores svagheder i form af sikkerhed ligger i, hvornår vi hasher medarbejderens password. Hvis en hacker kigger med mellem presentation layer og vores function layer kan personen stadig se password som ren tekst.

## Salt

For at beskytte vores password yderligere gør vi brug af salt. Salt er en tilfældig streng som bliver indsat i et password, inden det bliver hashet. Da vi igen skal bruge saltet når en medarbejder logger ind, gemmer vi også saltet i vores database, og skifter det ud hver gang et password bliver skiftet. Formålet med dette er, at undgå at et password får den samme hashværdi med andre identiske passwords.

På denne måde undgår vi flere ting; Vi undgår at en person, der har kendskab til en Fog medarbejders password, gennem databasen kan se om, andre har det samme password.

Vi undgår derudover også rainbow tables. Et rainbow table er en samling af allerede knækket hashes, som en person kan slå op i og se om et hash i vores database stemmer overens med et hash i et rainbow table. Da alle vores hashes er unikke pga. salt, undgår vi dette problem.

## POST og GET

Måden vi sender data fra en side til en anden er oftest med POST eller GET. Når vi skal vælge hvorvidt vi skal bruge POST eller GET, skal vi overveje hvilken form for data, det er vi sender. Grunden til dette er, at hvis vi benytter os af get, kommer variable navnet og værdien til at stå i brugerens URL. Vi kan dog roligt benytte GET, hvis vi ikke arbejder med personfølsomme data. Det kan eksempelvis være et *order id*, eller længden på brugerens

carport. Hvis vi derimod arbejder med personfølsomme data, såsom password, kontooplysninger eller lign., gør vi brug af POST til at videresende data.

## Password håndtering

Et af vores mål når det kom til password, var at kun medarbejderen selv skulle have kendskab til sit password. Dette løste vi på flere måder.

1. Når en ny medarbejder først bliver oprettet indtaster admin kun email, username og medarbejderens for- og efternavn. Derefter sender systemet selv en email ud til medarbejderen med en tilfældig streng af længden 20, som er hans password. Den første gang han logger ind med dette password, vil han blive bedt om at lave et nyt password.
2. Hvis en medarbejder skulle komme til at glemme sit password, kan han trykke på "Forgot password" hvorefter han indtaster sin mail. Hvis mailen er tilknyttet en medarbejder, bliver der igen sendt en mail ud med en tilfældig streng af længden 20 som medarbejderne skal bruge for at logge ind, hvorefter han skal skifte den når han logger ind.
3. Hvis en admin mener, at det er på tide en medarbejder skal udskifte sit password, kan personens password nulstilles. Det betyder at næste gang medarbejderen logger ind, vil han blive bedt om at lave et nyt password.

I alle tilfælde, kan medarbejderen, når han skal skifte password, ikke ændre det til sit forrige password.

## Udvalgte kodeeksempler

### Udregning af spær

Når vi beregner spær i Calculator, starter vi med at finde ud af hvor mange spær vi skal bruge i forhold til carportens længde:

```
int amountOfRafters = (int) (cust.getLength() / 50);
```

Vi har ud fra tegningerne fra Fog antaget at der pr. 50 cms længde, skal ligge et spær. Derfor dividerer vi længden med 50, og caster det til et heltal.

Efterfølgende beregner vi, hvor mange gange et spærtræ kan ligge på bredden:

```
double remainder = WIDTH / RAFTERWOODLENGTH;
    if (remainder % 1 != 0) {
        remainder++;
    }
```

Dette gøres ved først at dividere bredden med længden af et spærtræ. Herefter tjekker vi om antallet ikke er et heltal, og hvis det ikke er, ligger vi et ekstra brædde på for at sikre, at vi kan dække hele bredden.

Nu kan vi blot gange mængden af spærtræ op med remainder for at finde ud af, hvor mange vi skal bruge i alt:

```
amountOfRafters *= remainder;
```

Inden vi gemmer spærtræet på listen products, beregner vi længden af hvert spærtræ, vi skal bruge:

```
double lengthOfRafters = (WIDTH / remainder);
```

Vi gemmer her et objekt af produktet på listen kun med de parametre vi skal bruge for at gemme det i datamapperen.

Som det sidste skal vi nu finde ud af, hvor mange gange vores spær skal støttes af en pæl. Dette gøres ved at trække 1 fra remainderen og gemme det i en variabel vi kalder "pitstops":

```
int pitstops = (int) (remainder - 1);

polesAndBeams(pitstops);
```

Denne variabel sender vi efterfølgende over i metoden *polesAndBeams(int pitstops)*, der sørger for at tilføje de nødvendige pæle, og remme som spærtræet skal monteres på.

## Emailing og threading

Når vi skal sende mails skal vores mail klasse bruge 3 String's; Modtager, title, og beskeden.



```
String title = "New password";
String text = "Here is your new password... DONT LOSE IT AGAIN...\n\n"
              + "Password: " + password;
SendEmail emailSender = new SendEmail(email, title, text);
```

Vi kan også vælge at give klassen en liste af mails som der skal sendes til. Dette gøres med en overloaded constructor således:

```
public SendEmail(List<String> mailList, String title, String
textMessage) {
    this.title = title;
    this.textMessage = textMessage;
    this.empList = mailList;
}

public SendEmail(String mailTo, String title, String textMessage) {
    this(Arrays.asList(mailTo), title, textMessage);
}
```

Vi opdagede hurtigt, at når vi skulle sende mange mails på en gang, ville serveren bruge en del tid på. Dette gik ud over brugeroplevelsen, når der blev trykket på en knap, og der gik op til 10 sekunder, før næste side ville blive vist. Måden vi løser dette på er ved at gøre processen med at sende en mail på en anden thread. Det første vi gør er at implementer interfaced Runnable, for så at override den abstrakte metode run().

```
public class SendEmail implements Runnable {
```

For at køre run() skal vi oprette et Thread object som tager mod typen Runnable i sin parameter. Dette gøres fra metoden i LogicFacade. Herefter mangler vi blot at kalde metoden start() som Thread indeholder.

```
Thread thread = new Thread(emailSender);
thread.start();
```

Dette starter en ny thread og kalder run i den. Hvorefter run starter mail processen:

```
@Override
public void run() {
    if (session == null) {
        makeSession();
    }
    for (String mailTo : empList) {
        sendEmail(mailTo);
    }
}
```

## Fejl og mangler

Her vil vi liste de fejl og mangler som vores opgave har.

- Den *selected* tab i Customer Nav Bar virker kun på index.jsp.
- Rette tidszone på databasen.
- Vores Calculator har vi kun beregnet de sværeste dele af styklisten. Derfor har vi ikke dækket styklistens omfang 100%.

### *Forslag til forbedringer*

- *Vi kunne* optimere databasen ved at implementere forslaget til forbedring forklaret under afsnittet [ER Diagram](#).
- *Vi kunne* sørge for at tooltip på styling siden ikke overlapper hinanden.
- *Vi kunne* gemme de felter der var brugbare ved en input exception.
- *Vi kunne* gøre det mere tydeligt over for kunden at det er *events* i venstre side på siden confirm.
- *Vi kunne* gøre det muligt at rette priser på produkterne som Calculatoren bruger. *Med adgang til en MySQL client ville en medarbejder kunne rette fejlene her.*
- *Vi kunne* tilføje en mulighed for at brugeren kunne færdiggøre ordren med fuld adresse og betaling når ordren er i status Confirmed.

# Kvalitetssikring

## Interne reviewing

Vi har under hele projektet gået meget op i kvalitets sikring. En af de metoder vi har gjort brug af er interne reviews. En af punkterne i vores "definition of done" var vi skulle sætte os sammen og læse koden igennem og komme med konstruktiv kritik. Vi har haft en rigtig god oplevelse ved dette, bl.a. fordi man hele tiden havde i baghovedet at ens kode skulle læses igennem og vurderes af en anden part.

Et andet positivt aspekt ved vores review er, at alle har et dybt kendskab til hele projektet. Både når det kommer til frontend og backend. Dette kendskab gjorde det væsentlig nemmere at bygge videre på noget, som en anden havde lavet.

## Accept-test

For at dokumentere aftalen omkring aflevering af vores produkt med vores product owner, lavede vi en accept-test. Vi opstillede en liste over alle userstories på vores product backlog, og gennemgik samtlige features på den færdige side. Dette sikrer, at alle parter er enige om, hvad der er blevet leveret. Accept-testen kan fremtidigt fungere som et juridisk dokument, der kan sikre vores team som udviklere i en retssag, hvis f.eks. Fog sagsøger os for at udelade visse aftalte user stories/features. Samtidigt kan det også skabe et overblik for en kommende Fog medarbejder, hvad der har været aftalt ift. systemets kunnen.

## Brug af Gestalt Love<sup>1</sup>

For at sikre en god brugeroplevelse valgte vi at tage brug af gestaltlovene da vi designede siden. Lovene er ret basale, og derfor benytter man dem oftest uden at være bevidst omkring det når man laver grafisk arbejde.

**Law of Similarity** ses brugt på siden Orders, hvor vi har givet de bekræftede ordre en grøn farve. Dette sikrer at medarbejderen hurtigt kan skelne mellem hvilke ordre der er bekræftet,

---

<sup>1</sup> <http://architectingusability.com/2011/05/26/using-the-gestalt-laws-of-perception-in-ui-design/>

og hvilke der ikke er. Alt dette gøres uden at skrive nogle steder at den er bekræftet, den grønne farve hjælper også her med vise at den er bekræftet.

**Law of Proximity** bruger vi næsten på samtlige af vores sider. Eksempelvis på hele kundens bestillings process hvor vi har opdelt siden i to forskellige dele lodret. Her kommer Bootstrap's *row* system også til gode da det er et godt værktøj til nemt at holde en opdeling af siden som også er dynamisk ift. enhedens skærmstørrelse. At opdele siden med funktionalitet til venstre og overblik til højre gør det samtidigt nemmere for kunden at gennemskue hvor de skal trykke og gøre ting.

#### Dimentions

Length	Width	Height
100	100	100

**Law of closure** ses i vores styling af tabellerne ovenfor i kunde overblikket i højre side. Ved brug af en tynd streg og en lidt tykkere streg skabes der en illusion af en tabel som normalt har kanter i siden lige som denne:

#### Dimentions

Length	Width	Height
100	100	100

## Unit Tests

Vi har valgt at lave junit test på vores calculator. Dette har dog vist sig at være en smule besværligt, da vi ikke har et fast facit. Derfor har vores fremgangsmåde i stedet været at teste om calculatoren opfører sig som vi forventer. Dette har vi blandt andet gjort ved at se om der vil komme flere tegl på taget hvis taget ikke er fladt, kontra hvis det var fladt.

```
@Test
    public void testAngledRoof() throws Exception {
        Order order1 = new Order(null, new Customization(600, 600, 100, 0, null,
new StyleOption("test", "test", 0), new StyleOption("test", "test", 0)));
        Calculator cal = new Calculator(order1);
        cal.calculate();
    }
```

```
int order1Amount = productAmount(8, cal.getProducts());

Order order2 = new Order(null, new Customization(600, 600, 100, 45, null,
new StyleOption("test", "test", 0), new StyleOption("test", "test", 0)));
Calculator cal2 = new Calculator(order2);
cal2.calculate();
int order2Amount = productAmount(8, cal2.getProducts());

assertTrue(order2Amount > order1Amount);
}
```

## Integrationstest

Vi har også lavet integrationstest med vores MySQL database. Vi har lavet test til alle metoder i vores mappere, men vi har dog valgt at være særlig grundig med vores EmployeeMapper, med det mål at ramme 100% test coverage. Vi har dog kun ramt 83,74 grundet at næsten alle vores metoder har en try catch der fanger en SQLException. Vi har testet verifyLogin for om den kommer ind i denne try catch, og dette kunne vi gentage for alle de andre metoder for at opnå de 100%. Vi føler dog ikke dette er nødvendig for at sikre kvaliteten.

Når vi tester vores mappere gør vi brug af en test database. Vores test database har den fuldstændige samme struktur som vores production database, men vi har selv valgt hvilke data der skal indsættes. Vi gemmer test databasen i vores project folder som test\_db.sql i mappen sql. Inden vi kører en Integrationstest bruger vi en BufferedReader til at læse hele filen ind i en String. Dette gøres i constructoren, og derfor sker det også kun en gang, hvilket øger performance.

Inden hver test, bliver metoden setUp() kaldt. Den bruger vores sql String til at rive hele vores test database ned, for så at bygge den op igen.

På denne måde sikre vi at der altid er den forventet mængde data, og testene kan køre uafhængige af hinanden.

Et eksempel på denne del er vist nedenfor:

```
public EmployeeMapperTest() throws IOException, ClassNotFoundException,
SQLException {
```

```
        BufferedReader br = new BufferedReader(new InputStreamReader(new
FileInputStream("sql/test_db.sql"), "UTF-8"));
        StringBuilder sb = new StringBuilder();

        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line);
            sb.append(System.lineSeparator());        }
        this.sql = sb.toString();

        this.con = new TestConnection().connection();
        this.mapper = new EmployeeMapper(con);
    }

    @Before
    public void setUp() throws SQLException, ClassNotFoundException {
        Statement statement = con.createStatement();
        statement.executeUpdate(sql);
    }
```

## Usability test

[Link til video](#)

*Hvorfor?*

Som en del af vores kvalitetssikring valgte vi at lave en usability / tænke højt test for sikre kvaliteten af kundeoplevelsen. Altså testede vi kun den del vi mener er den vigtigste ift. brugervenlighed, nemlig kundesiden. Da vi designede kundesiden gjorde vi bl.a. brug af gestaltlovene for at sikre brugeroplevelsen, dette testede vi så for at se hvorvidt der var en effekt.

*Hvordan?*

Vi startede med at finde en person uden kendskab til vores system, for at sikre at brugen af siden var neutral, og ikke påvirket af tidligere erfaringer ved brug. Dette var en pige ved navn Tilde som højst sandsynligt aldrig ville bestille en carport i sit liv, hvilket både kan have

fordele og ulemper ift. testen. En af fordelene er at det i nogen grad er en ekstrem test af brugervenligheden, hvilket kan være nyttigt hvis man er ude på at sikre tilgængelighed for alle typer kunder. En af ulemperne er at testpersonen ikke er en fagperson på området og derfor ikke har den nødvendige viden ift. det tekniske og hvordan det virkelige domæne fungerer. Vi skrev efterfølgende en række opgaver/udfordringer som testpersonen skulle løse, for at strukturere hvad vi ville teste. For at undgå at klippe i videoen sørgede vi for at testpersonen ikke skulle bruge lang tid på at åbne sin mailboks, ved at bede hende om at kopiere linket til hendes mail.

### *Udførelsen og udbytte*

Den første opgave hed: "Indtast dimensioner og gå til næste side". Her tester vi brugervenligheden af vores input felter, og om brugeren kan finde et system i navigationen på siderne. Dette gik smertefrit og brugeren fandt selv ud af at tilvælge "Angled roof" og "Shed" da hun skulle indtaste mål herunder og navigere til næste side.

Næste opgave hed: "Vælg din "Styling" på din carport uden at gå til næste side". Her testede vi om brugeren kunne differentiere de 2 typer af styling, om hun kunne finde ud af at vælge en styling for hver type, om hun lægger mærke til tool tippet med beskrivelse af typen og om hun finder "Update"-knappen. Dette gik næsten også uden problemer. Testpersonen kom ved en fejl til at gå til næste side, men fandt "Update"-knappen da hun gik tilbage, og bekræftede også her, at hun kunne finde ud af at navigere tilbage. Hun havde dog en bemærkning til tooltips som overlappede hinanden. Dette kunne man nemt fikse med et script der lukkede de forrige tips når et nyt åbnes.

Følgende opgave hed: "Indtast dine virkelige oplysninger". Testpersonen kom her til at skrive sin email uden et "@", hvilket var meget positivt da vi her fik bekræftet at brugeren forstod fejlmeddelelsen der kom op.

Herefter fik testpersonen opgaven: "Ændre skur længden til 290 cm". Denne opgave skrev vi for at teste om brugeren ville bruge navigationsbaren, da det er langt nemmere at navigere tilbage ved brug af denne. Det viste sig at testpersonen valgte at bruge "Back"-knappen i stedet for, hvilket nok må betyde at vi burde ændre i den måde navigationsbaren er stilet på. Dette kunne f.eks. være ved at ændre farverne og evt. ændre knapperne til pile i stedet.

Efterfølgende skulle testpersonen bestille carporten, hvilket gik uden problemer. Brugeren læste også event-meddelelsen og forstod at hun skulle åbne en mail med linket for at vende tilbage til reference siden, hvilket var den følgende opgave. Da testpersonen åbnede siden igen læste hun event-meddelelser fra før som om at det var en ny meddelelse hun skulle læse, og forstod ikke helt at det bare var det første event der blev skrevet. Dette kunne man fikse ved at gøre det mere tydeligt at der var tale om events, eksempelvis ved bare at skrive "Events" over listen.

Alt i alt var denne test meget positiv, både i forhold til udførelsen, men også på den feedback vi fik på de elementer som kunne forbedres.

## Process

### Sprint planning

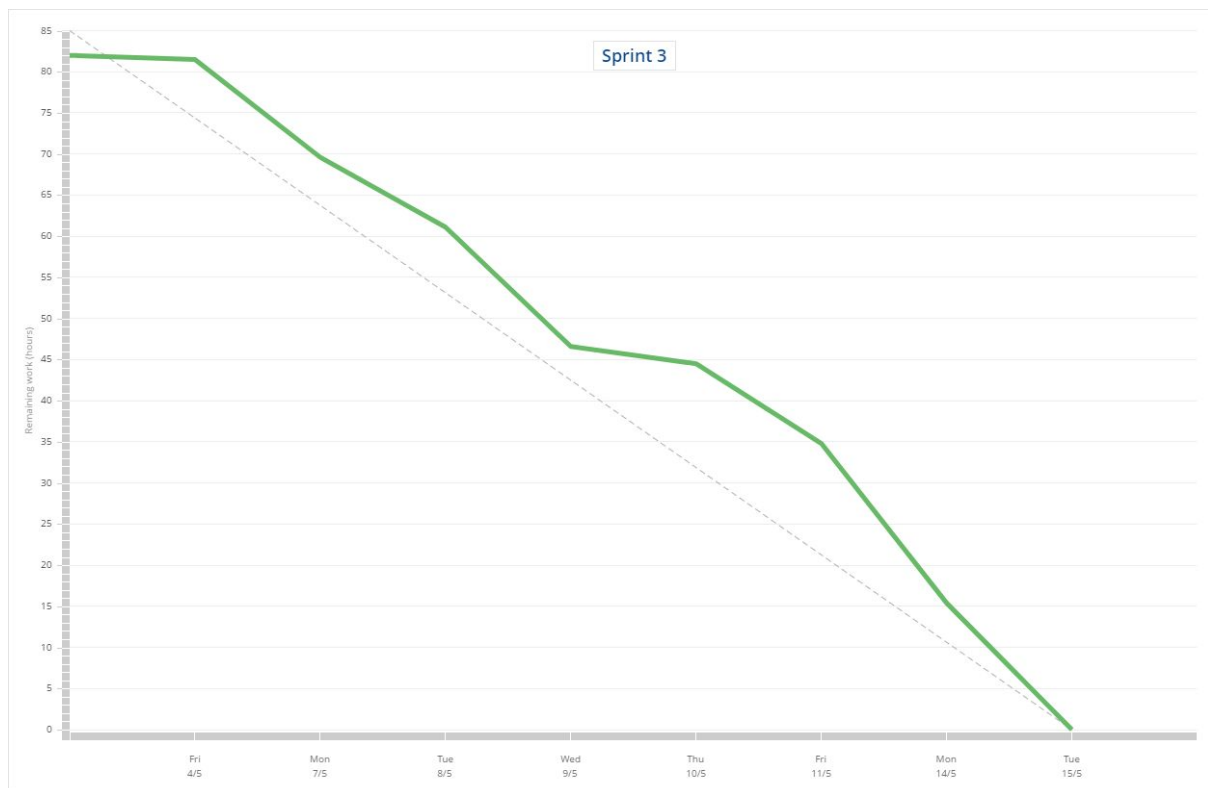
Inden hvert sprint ville vi starte med at se på, hvor meget tid vi ville have til rådighed. Vi kunne som udgangspunkt arbejde på projektet i 30-35 timer om ugen hver, hvor vejledermøder og klasseundervisning ikke var inkluderet. Ud fra dette tal var vi så istand til at planlægge hvilket task vi kunne nå i løbet af sprintet, på baggrund af de estimer vi lavede i starten af projektet af hver user story.

### Burndown charts

Vi har under hvert sprint været i stand til at følge om hvorvidt vi fulgte vores tidsplan ved at kigge på sprintets burndown chart. Dette har været en stor hjælp både i form af motivation,



og organisering af gruppens tid.



Hvis vi bruger billedet ovenfor kan vi se at i Sprint 3, ikke kom godt fra start. Dette kan ses ved at dag 1 har en flad kurve, og på grund af dette var vi bagud gennem hele sprintet. Vi indhentede dog det tabte i løbet af de sidste dage.

## Scrum master

Vi har gennem hele projektet haft den samme scrum master i Nikolai Perlt. Vi har dog ikke brugt denne rolle i en stor grad. Vi tror grunden til dette er at vi kun har været 2 i gruppen og derfor har nødvendigheden ikke været lige så stor som hvis vi var 4 gruppemedlemmer. Vi har dog brugt rollen i et mindre omfang. Blandt andet når vores vejleder har skrevet til os med henblik på at, høre hvor langt vi var nået. Her har han taget kontakt til scrum master, og scrum masteren har svaret igen. Vores værktøj til scrum(Scrumwise) skulle også fornyes undervejs i projektet, dette var også scrum masters ansvar.

## PO møde

Vi har næsten hver uge haft et PO møde. Under disse møder har det tekniske aspekt ikke været i fokus, men derimod hvilke krav Fog har til os som udviklere. Vi har oftest startet ud med at vise hvilke nye features og hvilke forbedringer vi har lavet på tidligere features, som PO så har givet feedback på. Vi har været rigtige glade for disse møder da det sikrede at vi ikke gik i en retning som kunden ikke ville have.

## Daily scrum

Hver arbejdsdag har vi givet en opdatering i form af et daily scrum til hinanden. Her snakkede vi om hvad vi har gennemført, og hvad vi har igangsat af nye task siden sidst. Vi har dog ikke helt fulgt scrum på dette punkt da vi ikke har afsat et bestemt tidspunkt på dagen hvor vi har taget dette møde. Vi har i stedet afholdt mødet når det virkede naturligt.

## Definition of done

Det vi startede projektet var en af de første ting vi skulle aftale vores *definition of done*. De lød således.

- **Styled** with bootstrap according to standards (if visible)
- **Tests** is written (if possible)
- **Tests** passed
- **Reviewed** informally
- **Documented** (if possible)

Vi har forsøgt at overholde disse så meget som muligt. Det har dog ikke altid helt været tilfældet. Især med henblik på javadoc har vi ikke altid været helt skarpe med at lave det undervejs. Vi har dog været rigtig glade for at der fra starten var sat klare linjer om hvad der forventes af en færdig feature, og på den måde er der undgået unødige uenigheder i gruppen.

## Retro spective

Efter hvert PO møde har vi på selve dagen, eller dagen efter, afholdt et retro spective. Vi har på dette møde snakket om hvad vi syntes var gået godt, og dårligt i det seneste sprint. Vi har også kigget på hvordan vores estimer i forhold til userstories har været, og om vi har nået alt hvad vi ville i sprintet. Vi opdagede f.eks. i et af det første sprint, at vi havde en tendens til at undervurdere hvor lang tid det ville tage af lave front-end, og aftalte at vi fremadrettet skulle sætte mere tid af til dette. På denne måde har retro spective været med til at tilpasse vores scrum, samtidigt med at sætte os på rette kurs med fremtidige sprints.

## Vores indtryk af scrum

Vi har overordnet været rigtig tilfredse for scrum i det her projekt, og hvis vi i fremtiden får muligheden for at vælge kunne vi sagtens finde på at bruge det igen. Vi har dog haft mere succes med nogle ting end med andre.

### Positive

#### Sprints

Vi har i særdeleshed været glade for at dele projektet op i sprints og derved følge divide and conquer konceptet. Det har været nemt at overskue hvor mange task det var muligt at nå i et sprint, da vi på forhånd havde lavet et estimat på hvor lang tid en task ville tage. Det har også hjulpet på motivationen når vi havde nået målet for et sprint til tiden.

#### Scrumwise

Da vi startede vores projekt lavede vi vores userstories i appen trello. Vi skiftede dog hurtigt over til scrumwise da vores vejleder gjorde os opmærksomme på værktøjet, hvilket vi ikke har fortrudt. Scrumwise var nemlig designet til scrum, og havde en masse features indbygget, som trello ikke havde. En af disse features var automatisk generet burndown charts, som har været 100% live.

## Negative

### Scrum master

Vi har ikke haft den store succes med at bruge en scrum master. Vi tror dog dette skyldes at vi kun har været 2 i gruppen, og det derfor ikke har været nødvendigt at have en udvalgt person til at løse teamets udfordringer.

# Bilag

## Bilag 1 - Product backlog

User Story	How to demo	Subtasks	Est. hours
<i>As an employee i want to be able to see a list of all the orders so that i can have an overview of the incoming orders.</i>	<i>Show all orders so far, then add a new order into the system, then show that that the order list now includes the new order</i>	<ol style="list-style-type: none"> <li>1) Database</li> <li>2) Entity</li> <li>3) Logic Facade</li> <li>4) Frontend</li> <li>5) Mapper</li> </ol>	5
<i>As a salesman i want to be able to edit and confirm a request/order so that i can make changes after talking with the customer.</i>	<i>Take a existing order then make a change to the size of the carport.</i>	<ol style="list-style-type: none"> <li>5) CRUD</li> <li>6) Logic Facade</li> <li>7) Edit functions in frontend</li> <li>8) Be able to unconfirm order and limit edit to unconfirmed orders</li> </ol>	3
<i>As an employee i want to receive an email whenever a customer places a request online so that i'm not supposed to sit and refresh the order list all day long.</i>	<i>Make a order then show that a email has been send.</i>		3
<i>As an employee i want to be able to edit the product selection in the different categories so that the content can be updated.</i>	<i>Change the price of an item, and and add a new item to a category the make a new order with the new item and the item with new price</i>	<ol style="list-style-type: none"> <li>1) Employee frontend editor page</li> <li>2) CRUD</li> <li>3) Find a way to deal with fk constraints</li> </ol>	13.5
<i>As a warehouse employee i want</i>	<i>Print item list</i>		2

<i>to have a parts list so that i can pick the right parts for the order.</i>			
<i>As an employee i want to be able to see and edit the economic aspects of the order so that i can see earnings and set a price.</i>	<i>Change the total price of an order</i>	<ol style="list-style-type: none"> <li>1) crud for reading and writing price.</li> <li>2) make front end where an employee can alter the price</li> </ol>	3.7
<i>As a customer i want to receive an email with a reference link when i submit my request so that i can track my request.</i>	<i>Getting an email once requested and then going to a reference page with information via the link provided.</i>	<ol style="list-style-type: none"> <li>3) make method to send email</li> <li>4) make fog email</li> </ol>	1.8
<i>As a employee i want the orderlist to start by only showing 10 orders and then expand by 10 more when i press a button because it will make the site run faster and be more approachable.</i>		<ol style="list-style-type: none"> <li>1) change order.jsp page to only show 10 orders and at a click of a button show 10 more</li> <li>2) Make a method in Order Mapper that extract only user info</li> </ol>	3.5
<i>As a customer i want to see a visualization of the my customized carport so that i can get a feel of what it is going to look like.</i>	<i>Undefined: Depends on the approach.</i>	<ol style="list-style-type: none"> <li>1) Draw image</li> </ol>	20
<i>As a customer i want to see the price of the customized carport so that i can decide to proceed with the submission.</i>	<i>Seeing an est. price after the configuration.</i>		1
<i>As a user i want to be redirected to a errorpage when an error occurs so that i know that the system is failing.</i>		<ol style="list-style-type: none"> <li>1) Build upon what Kasper have already made when catching exceptions</li> <li>2) Make frontend page</li> </ol>	3.5

		<i>to be redirected to</i>	
<i>As a customer i want to see the events list of my request when i submit my order so that i can see a timeline of what has been done to my request.</i>		<ol style="list-style-type: none"> <li>1) <i>Coloring of events on type</i></li> <li>2) <i>Do rest of events</i></li> </ol>	9
<i>As an employee i want to be able to log in to an administrative part of the website so that i can manager orders and products.</i>		<ol style="list-style-type: none"> <li>1) <i>Make login verification in employee Mapper</i></li> <li>2) <i>Make frontend where employee enter a username and password and is only accessible from url</i></li> <li>3) <i>reset password via mail</i></li> <li>4) <i>Make simple Welcome page for employees that can be build upon</i></li> <li>5) <i>encrypt password</i></li> </ol>	8.2
<i>As an administrator i want to be able to manage employee users so that organizational changes can be replicated in the system.</i>		<ol style="list-style-type: none"> <li>1) <i>make crud function for employee</i></li> <li>2) <i>make frontend</i></li> </ol>	5

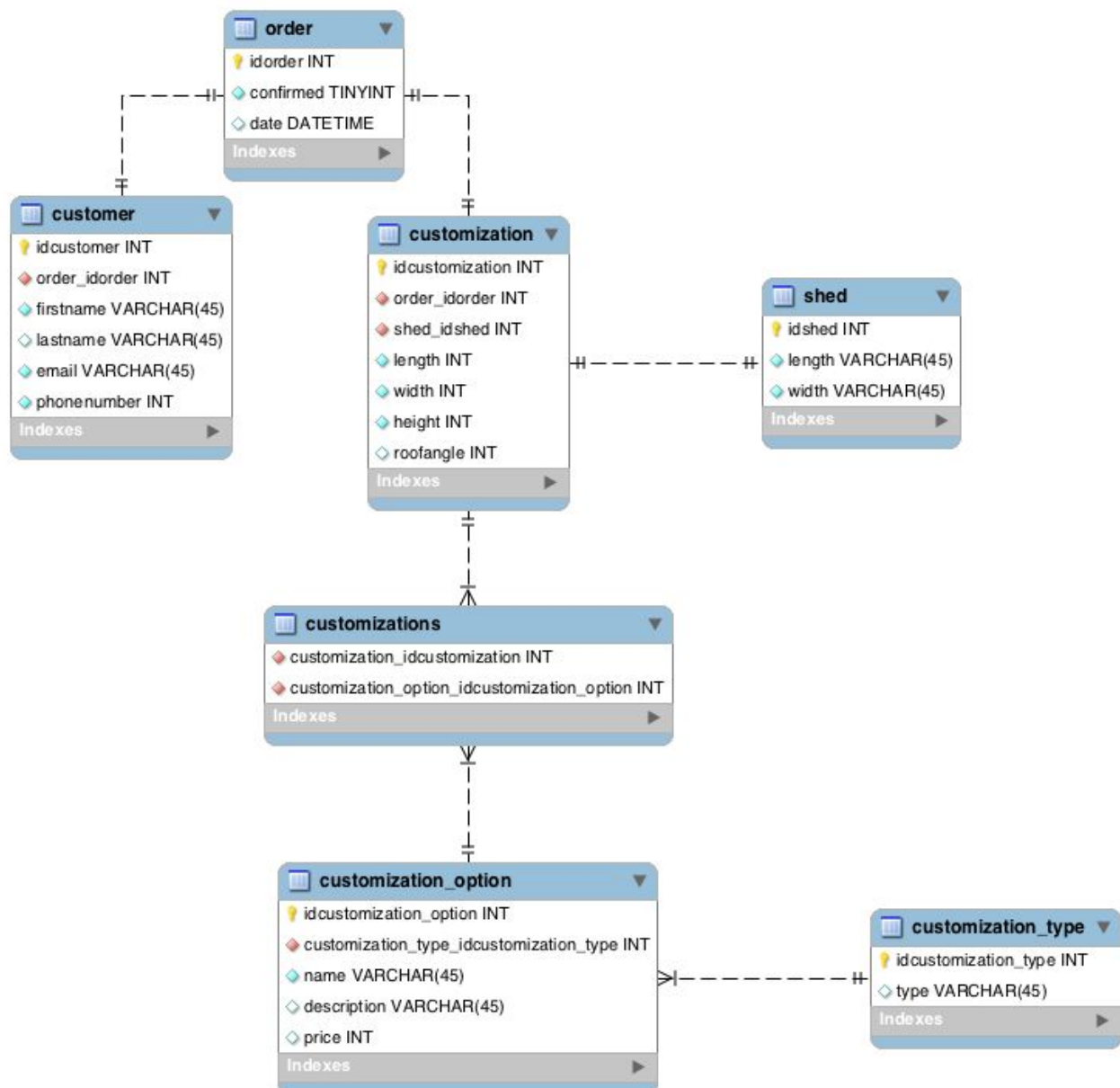
## Bilag 2 - Accept test

User Story	Godkendt
<i>As an employee i want to be able to see a list of all the orders so that i can have an overview of the incoming orders.</i>	Ja
<i>As a salesman i want to be able to edit and confirm a request/order so that i can make changes after talking with the customer.</i>	Ja
<i>As an employee i want to recieve an email whenever a customer places a request online so that im not supposed to sit and refresh the order list all day long.</i>	Ja
<i>As an employee i want to be able to edit the product selection in the different categories so that the content can be updated.</i>	Ja
<i>As a warehouse employee i want to have a parts list so that i can pick the right parts for the order.</i>	Ja
<i>As an employee i want to be able to see and edit the economic aspects of the order so that i can see earnings and set a price.</i>	Ja
<i>As a customer i want to receive an email with a reference link when i submit my request so that i can track my request.</i>	Ja
<i>As a employee i want the orderlist to start by only showing 10 orders and then expand by 10 more when i press a button because it will make the site run faster and be more approachable.</i>	Ja
<i>As a customer i want to see a visualization of the my customized carport so that i can get a feel of what it is going to look like.</i>	Ja
<i>As a customer i want to see the price of the customized carport so that i can decide to proceed with the submission.</i>	Ja
<i>As a user i want to be redirected to a errorpage when an error occurs so that i know that the system is failing.</i>	Ja

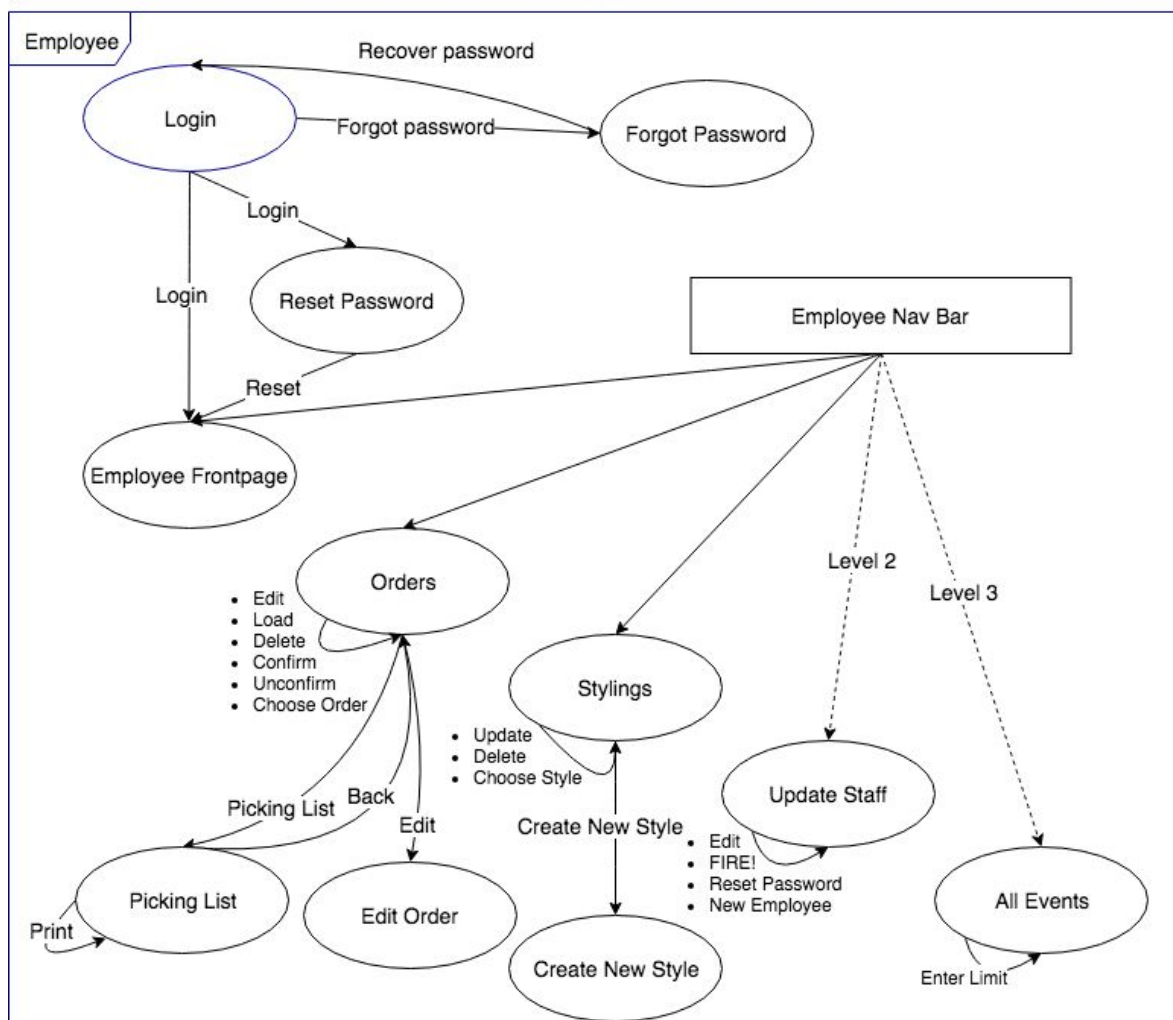
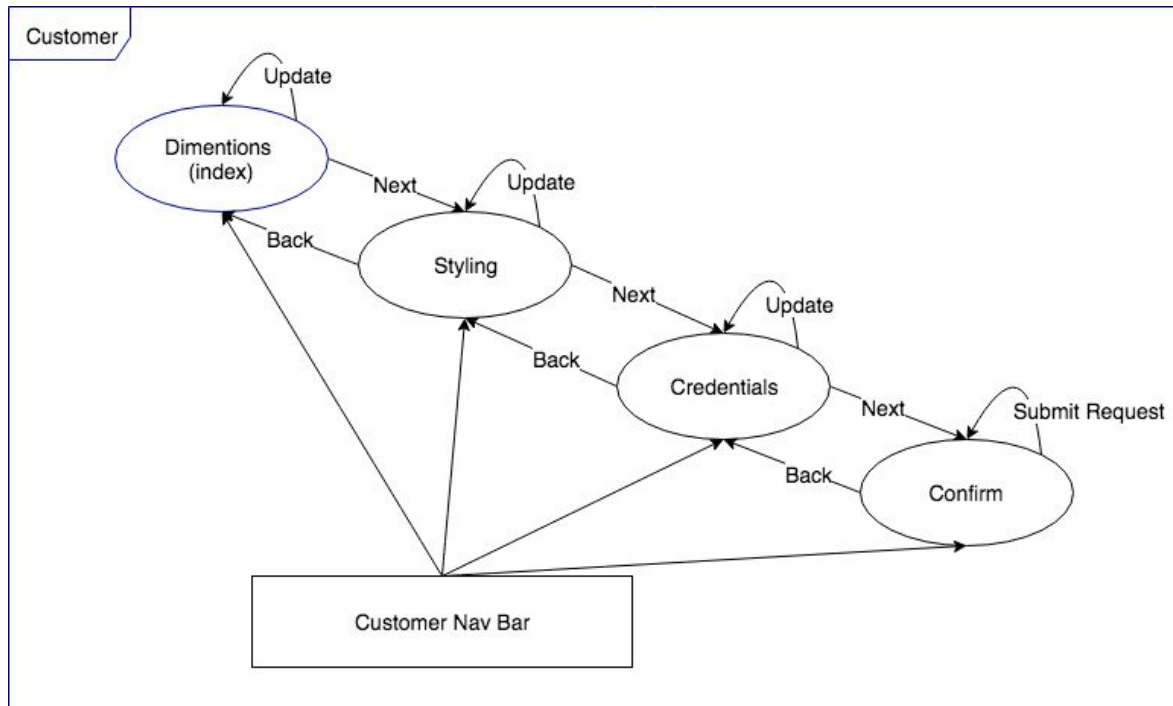


<i>As a customer i want to see the events list of my request when i submit my order so that i can see a timeline of what has been done to my request.</i>	Ja
<i>As an employee i want to be able to log in to an administrative part of the website so that i can manager orders and products.</i>	Ja
<i>As an administrator i want to be able to manage employee users so that organizational changes can be replicated in the system.</i>	Ja

## Bilag 3 - ER Diagram 1.0 (første udkast)

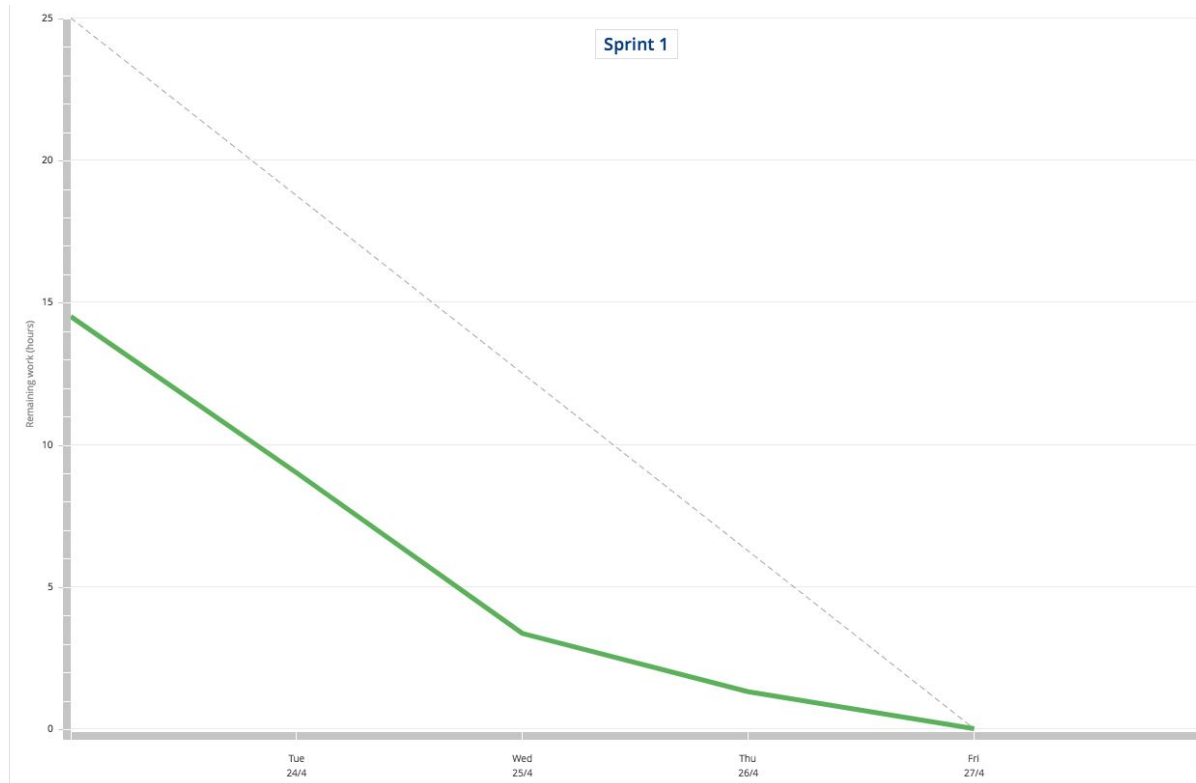


## Bilag 4 - Navigationsdiagram

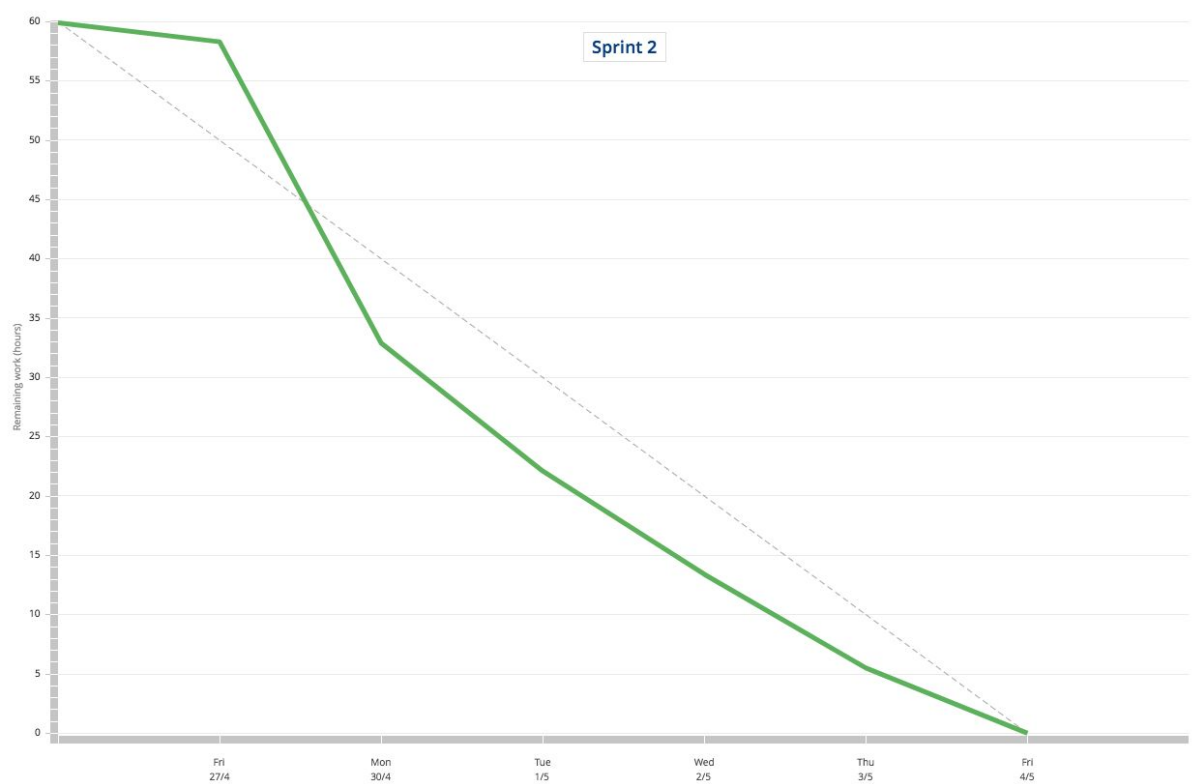


## Bilag 5 - Burndown Charts

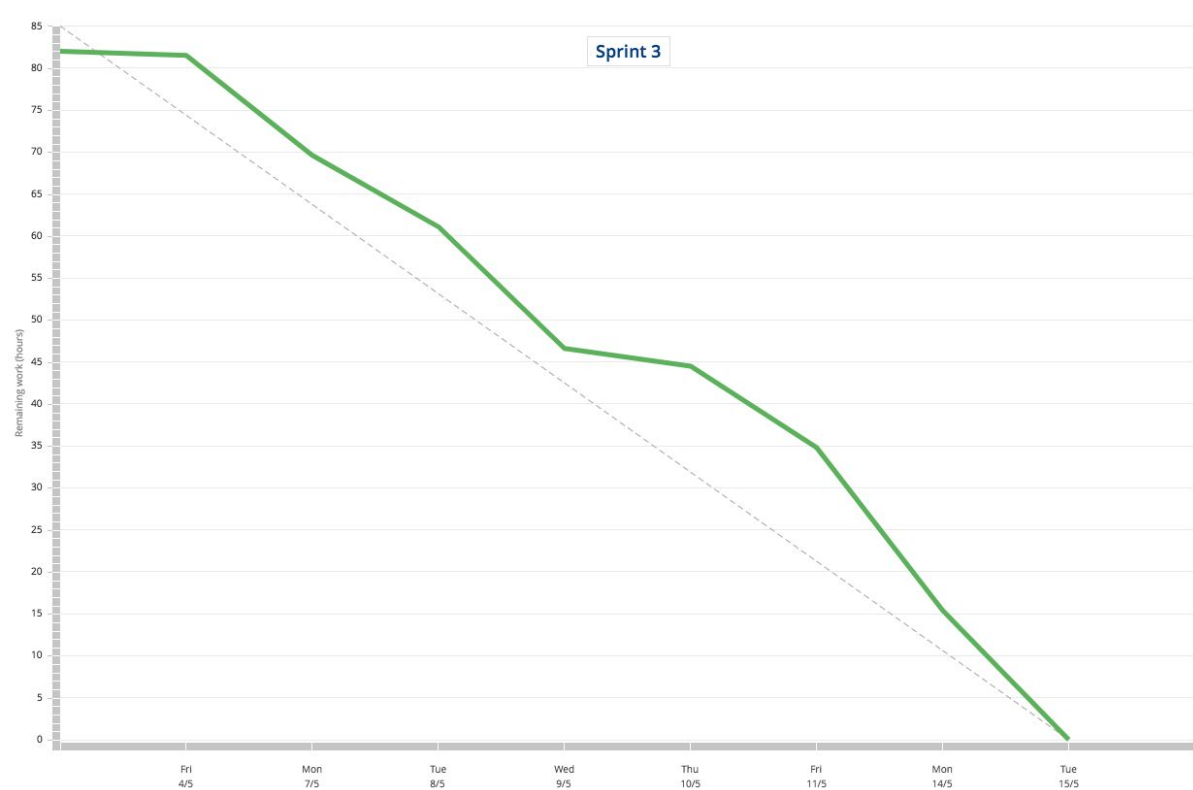
### Sprint 1



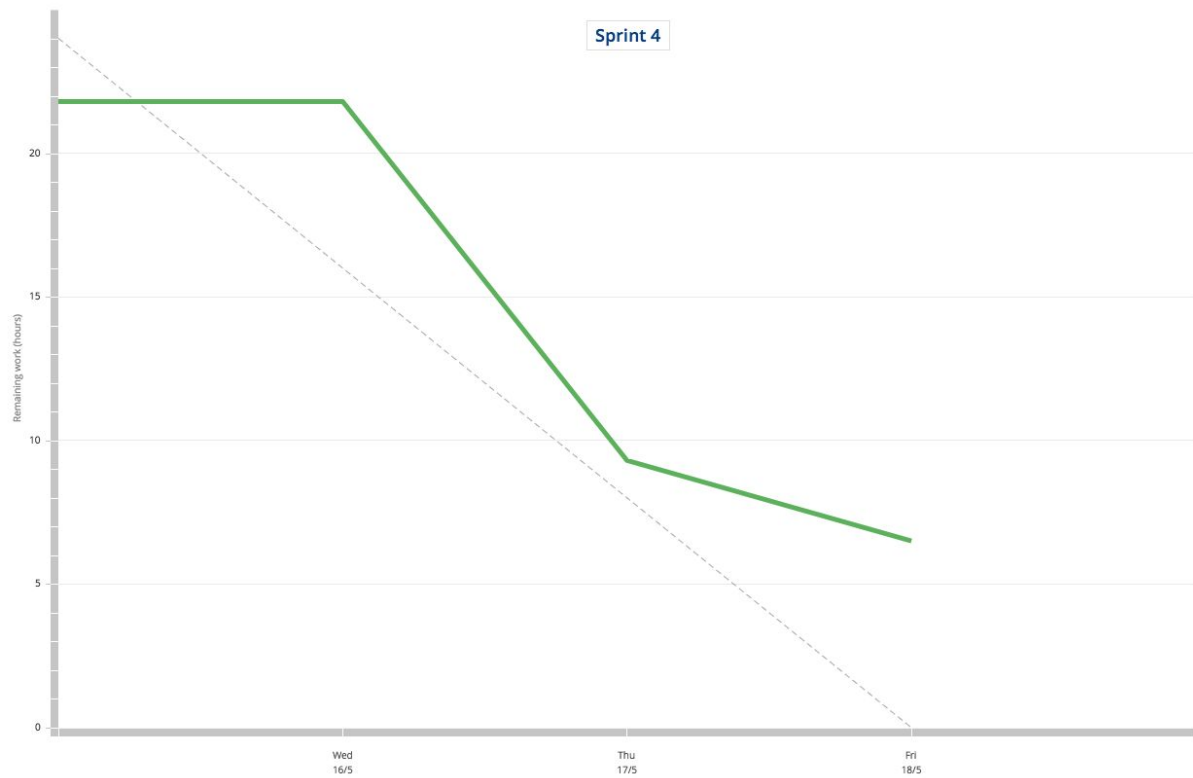
## Sprint 2



## Sprint 3



## Sprint 4



## Sprint 5

