

# Carport

## Udarbejdet af:

Henrik Michael Agger

Efteråret 2018

Dato:

19/12 2018

Navn:

Henrik Michael Agger

Cph-email:

cph-ha104@cphbusiness.dk

Klasse:

118dat2ae

<https://github.com/HenrikAgger/Carport>

[www.digitalocean.com](http://www.digitalocean.com)

cph-ha104 / FRA1 - Ubuntu 16.04.4 x64

## Indholdsfortegnelse

<b>Indledning .....</b>	<b>3</b>
Baggrund.....	3
Teknologi valg.....	3
<b>Krav.....</b>	<b>4</b>
Overordnet beskrivelse af virksomheden .....	4
Interessent analyse.....	5
SWOT analyse.....	6
Arbejdsgange der skal IT-støttes .....	7
Scrum userstories .....	8
<b>Domænemodel og ER diagram .....</b>	<b>10</b>
Domænediagram .....	10
ER diagram.....	11
<b>Navigationsdiagrammer .....</b>	<b>12</b>
Navigationsdiagram vedrørende kunden .....	12
Navigationsdiagram vedrørende salgsmedarbejderen .....	14
<b>Sekvensdiagram .....</b>	<b>16</b>
<b>Særlige forhold.....</b>	<b>17</b>
<b>Udvalgte kodeeksempler.....</b>	<b>19</b>
<b>Status på implementering .....</b>	<b>21</b>

## **Indledning**

Projektet omhandler at vi som studerende skal udarbejde et IT-system for Fog. Det antages at Martin fra Fog repræsenterer kunden som skal anskaffe IT systemet.

IT systemet skal kunne bruges af kunden til at bestille carporte. Programmet skal ligeledes kunne benyttes af salgsmedarbejderen til at håndtere ordrer.

## **Baggrund**

Virksomheden som skal bruge IT systemet er et firma ved navn Fog. Et af de produkter som firmaet sælger er carporte.

Productowner's krav til IT systemet er at det skal kunne bruges af byggefirmaets kunder til at bestille carporte. Samtidig skal IT systemet kunne bruges af salgsmedarbejderen til at håndtere kundeforespørgsler.

## **Teknologivalg**

Til projektet er anvendt følgende programmer:

- Netbeans 8.2
- JDBC (bundle version 8.0.11)
- Workbench 8.0
- MySQL 8.0.11
- Apache Tomcat 8.0.27
- Maven 3.0.5

MySQL er hostet på en linux maskine på Digital Ocean. Maskinen kører på Ubuntu 16.04.4 baseret på Linux.

## Krav

### Overordnet beskrivelse af virksomheden

Virksomhedens vision for systemet, og værdi som systemet skal tilføre virksomheden, er beskrevet med SMART mål, formål og featureliste.

SMART mål omkring hvad virksomheden opnår med systemet:

Specifikt – hvad er det helt præcist, der skal opnås?

- Bedre lagerstyring (rette varenumre og opdatere priser).
- Bedre leveringstider
- Beregning af (tagkonstruktion/stolpe)-forhold
- Grafiske forbedringer

Målbart – hvornår er målet nået?

- Når sales repræsentative let og hurtigt kan rette varenumre og opdatere priser
- Når leveringstiden er halveret
- Når (tagkonstruktion/stolpeforholdet) beregnes af programmet
- Når kunden er tilfreds med den grafiske præsentation

Attraktivt – hvorfor skal målene opnås?

- For at effektiviser arbejdsprocesser
- For at højne brugeroplevelsen

### Formål:

Formålet med at implementere et nyt IT-system er at opnå bedre brugeroplevelse og at effektivisere arbejdsprocesser.

### Featureliste:

- Bedre lagerstyring (rette varenumre og opdatere priser)
- Beregning af (tagkonstruktion / antal stolper)
- Grafiske forbedringer

### Interessent analyse:

Der er flere interessenter som har indflydelse på projektet. Nedenstående tabel viser graden af påvirkning og indflydelse de forskellige interessenter har på IT projektet:

	Stor indflydelse	Lille indflydelse
Bliver påvirket af projektet	Productowner og udvikler team	Håndværkeren, leverandører og lagermedarbejderen
Bliver ikke påvirket af projektet		Marketing og referencegrupper

Martin fra Fog Trælast og programmør er "ressourcepersoner" og bliver i høj grad påvirket af byggeprojektet.

Leverandører, håndværkere og lagermedarbejdere har forholdsvis lille indflydelse på IT projektet, men bliver påvirket af projektet. Leverandøren bliver påvirket vedrørende vareudvalg, leveringstider, priser mv. Lagermedarbejderen som skal bruge IT systemet og levere materialer bliver også påvirket af projektet.

Marketing og referencegrupper er "eksterne interessenter". Martin kan blive påvirket af hvad referencegrupper så som venner, familie og eksperter mv. siger om projektet.

## SWOT analyse

Nedenstående tabel viser en SWOT analyse set fra productowners perspektiv over hvordan det nye IT projekt vil påvirke Martins/virksomhedens situation.

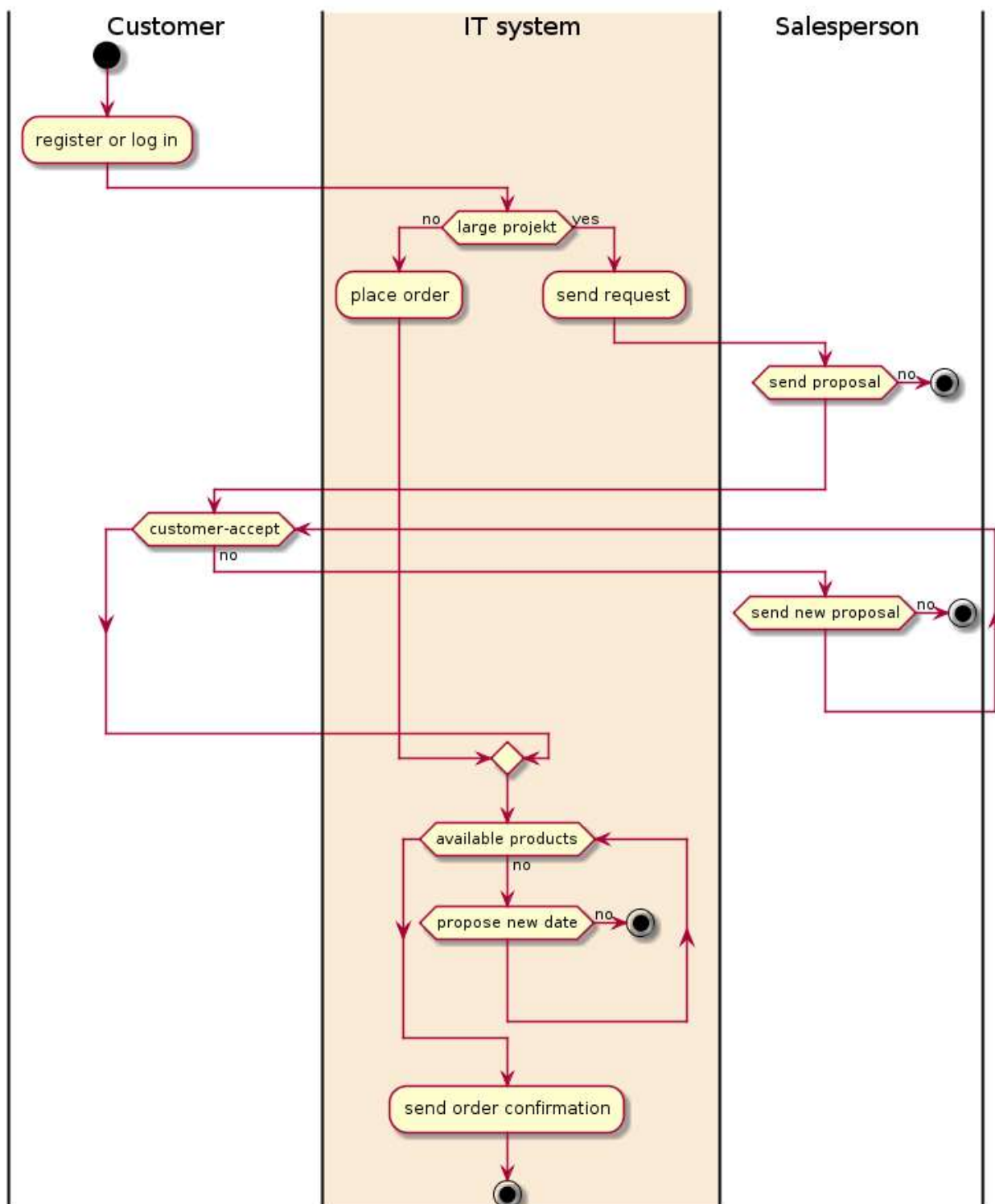
<b>Strengths</b> <ul style="list-style-type: none"> <li>• Bedre brugeroplevelse</li> <li>• Logistiske forbedringer: <ul style="list-style-type: none"> <li>- Bedre lagerstyring (rette varenumre og opdatere priser).</li> <li>- Bedre estimat af forventet leveringstid og avance.</li> <li>- Beregning af (tagkonstruktion / antal stolper) forhold.</li> <li>- Nyt program som er kompatibelt med nyt software</li> </ul> </li> </ul>	<b>Weaknesses</b> <ul style="list-style-type: none"> <li>• Konkurrenter kopierer programmet eller ideer.</li> </ul>
<b>Opportunities</b> <ul style="list-style-type: none"> <li>• Konkurrence</li> </ul>	<b>Threats</b> <ul style="list-style-type: none"> <li>• Oplæring og indkøring af nyt system tager tid</li> </ul>

En SWOT analyse over projektets chance for at blive en succes set fra udvikler teamets perspektiv.

<b>Strengths</b> <ul style="list-style-type: none"> <li>• Lave omkostninger</li> </ul>	<b>Weaknesses</b> <ul style="list-style-type: none"> <li>• Da der er tale om et eksamensprojekt er det ikke sikkert at projektet bliver færdigt.</li> </ul>
<b>Opportunities</b> <ul style="list-style-type: none"> <li>• Andre projekter</li> </ul>	<b>Threats</b> <ul style="list-style-type: none"> <li>• Konkurrenter</li> </ul>

### Arbejdsgange der skal IT støttes

De overordnede arbejdsgange efter IT systemet er implementeret, beskrives med nedenstående TO-BE aktivitets diagram.



Kunden registrerer sig eller logger ind. Hernæst sender kunden en forespørgsel på et tilbud. Salgsrepræsentanten sender et udkast. Kunden kan vælge at acceptere eller afstå udkastet. Såfremt udkastet afvises af kunden, sender salgsrepræsentanten evt. et nyt prisoverslag. Hvis kunden og salgsrepræsentanten bliver enige om tilbuddet, forespørger IT systemet om der er de nødvendige materialer. Hvis der mangler materialer, foreslås en senere leveringsdato. Til sidst sendes ordrebekræftelse mv. til kunden. Send "proposal" og "available" products er ikke blevet implementeret i koden.

## Scrum userstories

Product backlogs er defineret således:

1. Importance
2. Initial estimate
3. Kriterier for "done"
4. How to demo

Nedenstående ses en række sprint backlogs med tilhørende product backlogs:

1) As a <customer>, I want to <order carport> so that <I can place an ordre>.

1. Order carport er vigtig ud fra et 'business value' perspektiv. Ingen ordre, ingen handel.
2. Order carport får (2 story points). Det er en enkel opgave.
3. Opgaven er "done", når kunden kan placere en ordre via. en hjemmeside.
4. Teamet demonstrerer på en hjemmeside, at man som kunde, kan skrive i en webform en ordre hos sælger f.eks. højde, bredde og længde samt at ordren er blevet gemt.

2) As a <sales representative>, I want to <view customer orders> so that <I can see what the customers has bought>.

1. View customer orders er "overvejende" vigtigt ud fra et 'business value' perspektiv.
2. View customer orders får (3 story points). Det er en middel stor opgave.
3. Opgaven er "done" når man kan se samtlige ordre en tilfældigt valgt kunde har foretaget.
4. Opgaven demonstreres ved at teamet viser en ordrehistorik vedrørende en kunde som teamet ved har foretaget flere ordre.

3) As a <sales representative>, I want to <print line items> so that <I know exactly what needs to be sent to the customer>.

1. Print line items er en meget vigtig opgave set ud fra et 'business value' perspektiv. Ingen stykliste - ingen handel.
2. Print line items får (5 story points). Det er en stor opgave.
3. Opgaven er "done" når salgsrepræsentanten kan se/udskrive en stykliste på en tilfældigt udvalgt kunde.
4. Print line items demonstreres ved at produktejer vælger en kunde ud som har foretaget et køb og ser den pågældende kundes stykliste.

4) As a <sales representative>, I want to <view all customers> so that <I can see what I can get a view of the customers>.



1. View all customers er vigtig ud fra et 'business value' perspektiv af statistiske årsager.
2. View all customers får (2 story points). Det er en enkel opgave.
3. Opgaven er "done" når salgsrepræsentanten kan se alle kunder.
4. Opgaven demonstreres ved at teamet viser produkter alle kunder.

5) As a <sales representative>, I want to <view a specific customer order> so that <I can see what that specific customer has bought>.

1. View a specific customer order er vigtig ud fra et 'business value' perspektiv hvis salgsrepræsentanten ønsker at håndtere en specifik ordre.
2. View a specific customer order får (2 story points). Det er en enkel opgave.
3. Opgaven er "done" når salgsrepræsentanten kan se en specifik kunde.
4. Opgaven demonstreres ved at teamet viser produkter ser en specifik kundeordre.

6) As a <sales representative>, I want to <view a specific customers orders> so that <I can see what that specific customer has bought earlier>.

1. View a specific a specific customers orders er vigtig ud fra et 'business value' perspektiv hvis salgsrepræsentanten ønsker at håndtere en ordrehistorik på en given kunde.
2. View a specific a specific customers orders får (2 story points). Det er en enkel opgave.
3. Opgaven er "done" når salgsrepræsentanten kan se en specifik kundes ordrehistorik.
4. Opgaven demonstreres ved at teamet viser produkter en specifik kundes ordrehistorik.

7) As a <customer>, I want to <view all orders> so that <I can see what I have bought earlier>.

1. View all orders er vigtig ud fra et 'business value' perspektiv hvis kunden ønsker se ordrehistorik.
2. View all orders får (2 story points). Det er en enkel opgave.
3. Opgaven er "done" når kunden kan se sin ordrehistorik.
4. Opgaven demonstreres ved at teamet viser produkter hvordan en kunde ser sin ordrehistorik.

8) As an <employee>, I want to <register> so that <I can create a new employee account>.

1. Register er vigtig ud fra et 'business value' perspektiv. Hvis der bliver ansat en ny salgsrepræsentant er der behov for at kunne oprette medarbejderen i systemet.
2. Register får (2 story points). Det er en enkel opgave.
3. Opgaven er "done" når salgsmedarbejderen selv kan oprette sig som bruger.
4. Opgaven demonstreres ved at teamet viser produkter hvordan en salgsrepræsentant oprettes i systemet.

9) As a <customer>, I want to <view carports> so that <I can decide if I want to buy a carport>.

1. View carports as customer er vigtigt ud fra et 'business value' perspektiv. Kunden ønsker at se en grafisk præsentation af carporte før og efter et køb.
2. Grafik vedrørende carports får (5 story points). Det er en stor opgave. Det tager tid at udarbejde tegningerne.
3. Opgaven er "done", når designet er godkendt. Designet bør godkendes af projektejer.
4. Teamet demonstrerer grafikken vedrørende carporte over for projektejer.

10) As a <sales representative>, I want to <mark orders open/closed> so that <I know which orders I need to handle>.

1. Mark orders open/closed er en meget vigtig opgave ud fra et 'business value' perspektiv. Salgsmedarbejderen skal kunne se hvem som skal modtage ordre og om ordren stadig er igangværende.
2. Mark orders open/closed får (5 story points). Det er en stor opgave.
3. Opgaven er "done" når salgsrepræsentanten kan se om en specifik ordre er blevet behandlet/ sendt til kunden.
4. Mark ordres open/closed demonstreres ved at teamet viser status på en kunde som netop har bestilt en ordre ("open") samt status på en kunde hvor ordren er blevet sendt og derfor er "closed".

## **Domænemodel og ER diagram**

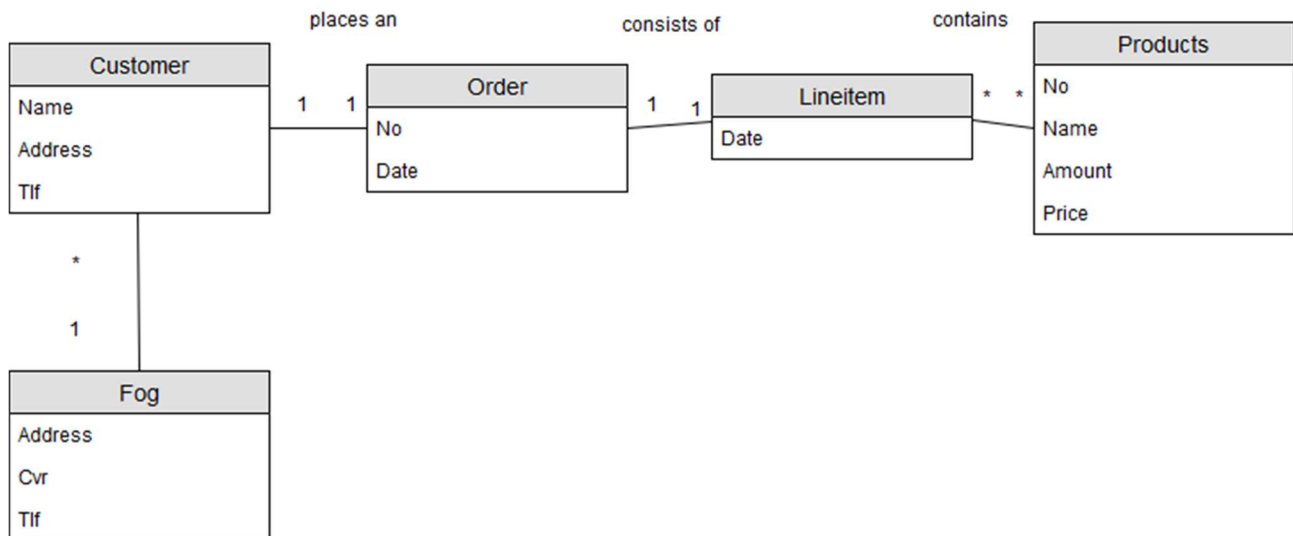
### **Domænediagram**

Domænediagrammet kan beskrives således: Fog Trælast har adgang til mange kunder. En kunde placerer en ordre. Med ordren følger en stykliste som består af mange produkter.

Entiteterne i domænemodellen er navneordene: Fog, Customer, Order, Lineitem, Products.

Attributterne i domænemodellen er: places an, consist of og contains.

Relationerne kan beskrives således. Fog har mange kunder. Kunden bestiller en ordre. Ordren består af en stykliste. Styklisten indeholder mange varer.

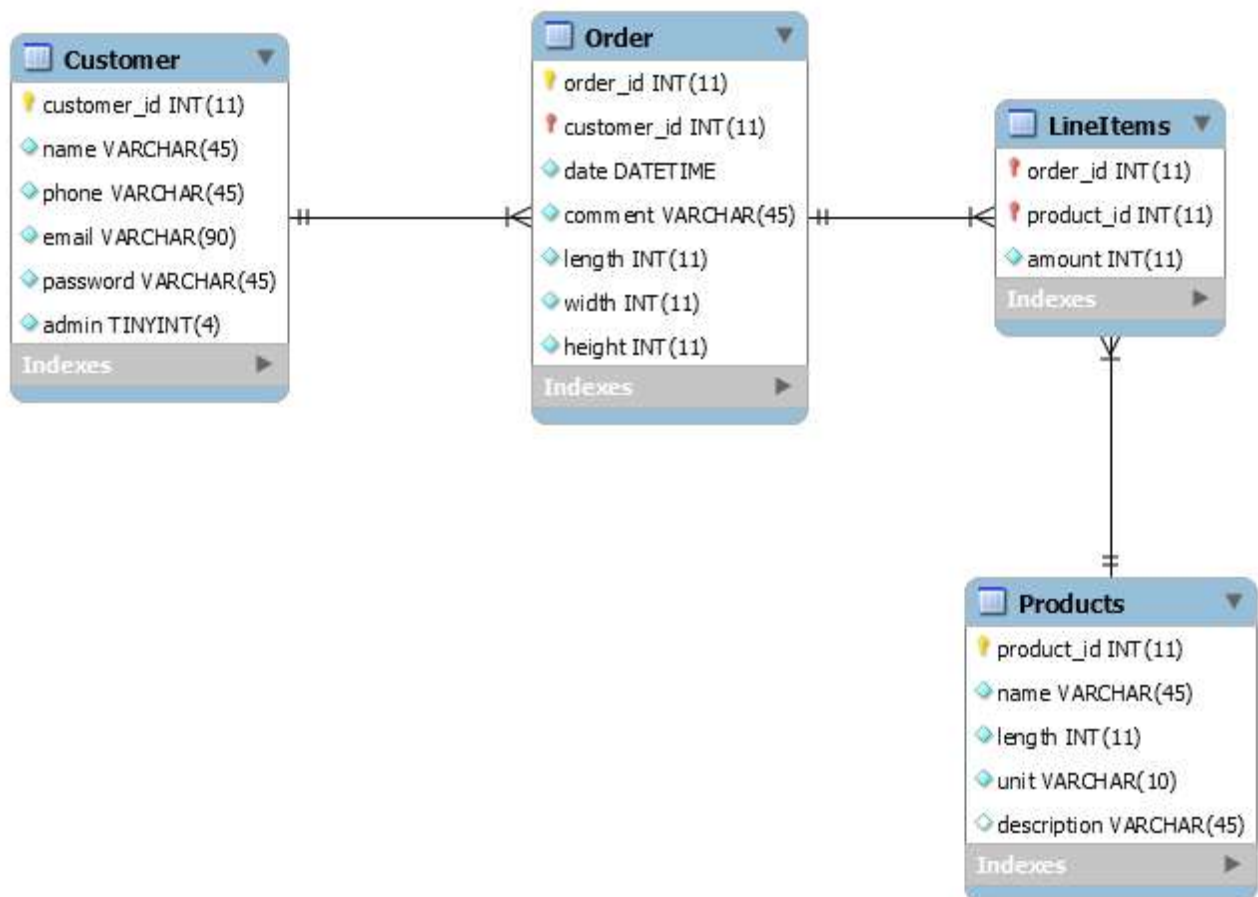


## ER diagram

Et Entity Relationship Diagram er et strukturelt diagram til brug i databasedesign. Tabellerne i databasen svarer til DTO klasserne i Java. DTO er blot objekter som holder dataene.

DAO er en klasse som har CRUD operationer så som create, read, update, and delete. F.eks. er der i `DataLayer/CustomerMapper.java` oprettet en metoden `createCustomer()`, hvor der indsættes parametre i SQL vha. `INSERT` som er hentet fra `FunctionLayer/Customer.java` der gemmes i databasen.

Nedenstående ses et ER diagram over databasen:



Et ER diagram er et logisk/strukturelt diagram, som beskriver tabellerne i den tilhørende database, samt relationerne mellem de enkelte tabeller, primærnøgler og datatyper.

Tabellerne:

Tabellen "Customer" beskriver kundens navn, telefonnummer, e-mail, password og admin. Hvor admin afgør om der er tale om administrator eller en kunde.

Tabellen "Order" beskriver dato, kommentar, længde, vide og højde på det den carport som er blevet bestilt.

Tabellen "LineItems" beskriver amount.

Tabellen "Products" beskriver produktnavn, længde, enhed i cm og evt. en beskrivelse af materialerne.

Der er en "en til mange" relation mellem Customer og Order tabellen. Da en kunde kan foretage mange bestillinger.

Der er en "en til mange" relation mellem Order og Lineltems og en "en til mange" relation mellem Products og Lineltems. Tabellen "Lineltems" er sammensat af fremmednøglerne order\_id og product\_id som forbinder tabellerne "Order" og "Products".

Der er kun valgt "Not Null" ved description. Admin i tabellen "Customer" er sat til 0 som default. Dvs. hvis man logger ind som kunde er standardværdien 0, og hvis man logger ind som administrator er værdien 1.

I tabellen "Order" ved kolonnenavn "date" er datatypen sat til DATETIME og default/expression er sat til CURRENT\_TIMESTAMP. Såfremt der foretages en bestilling af en carport registreres købstidspunktet.

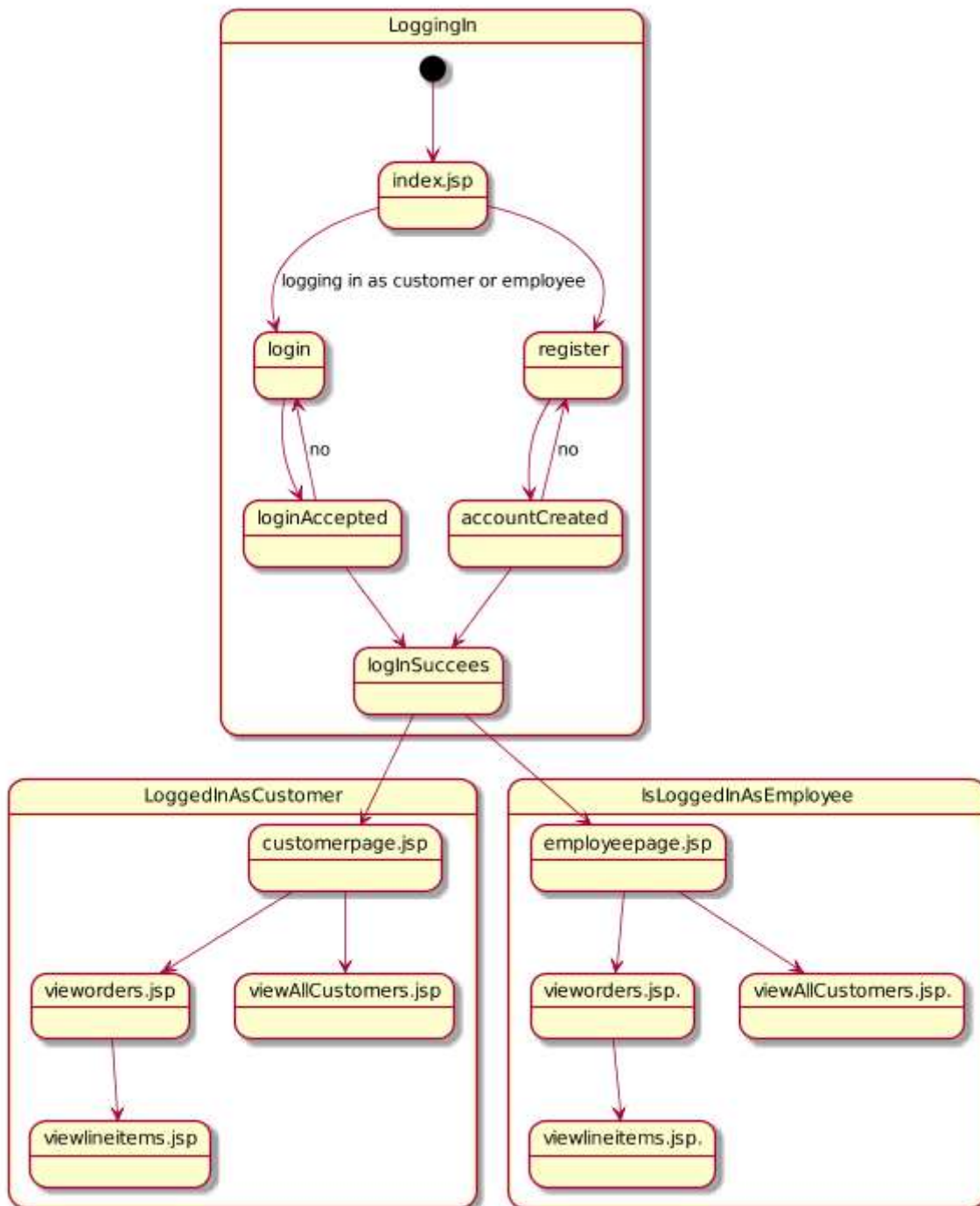
Tabellerne er normaliseret til 3. normalform idet:

1. Værdierne i kolonnerne er atomare, har samme domæne, kolonnenavnene er unikke i hver tabel og rækkefølgen af kolonnerne er ligegyldig.
2. Ingen kolonner er afhængige af en delprimærnøgle.
3. Ingen kolonner er afhængige af en kolonne udenfor primærnøglen.

### **Navigationsdiagram**

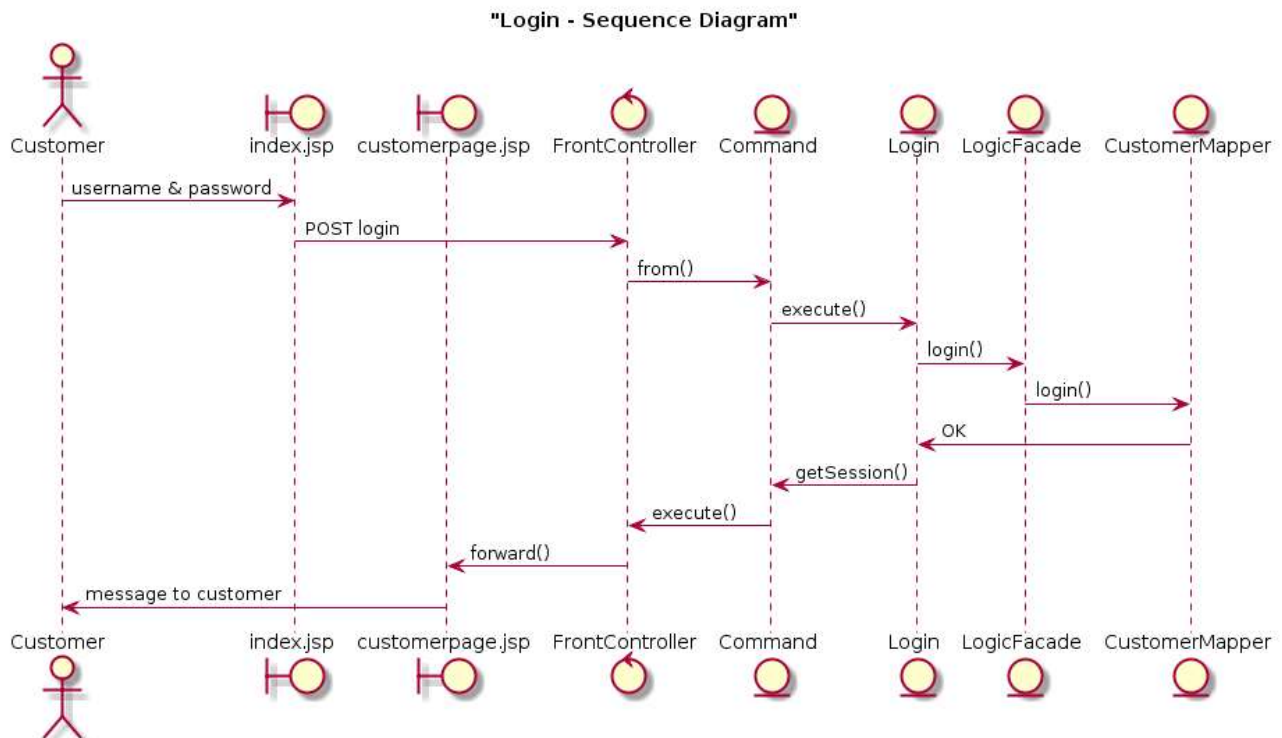
Nedenstående ses et navigationsdiagram som viser hvordan kunden eller salgsrepræsentanten logger ind eller registrerer sig som bruger. Dernæst vises de forskellige navigationsmuligheder for henholdsvis kunden og salgsrepræsentanten.

## Logging in



## Sekvensdiagram

Nedenstående ses et sekvensdiagram vedrørende en kunde som logger ind og videresendes til en ordreside.



På index.jsp findes en form med navnet login. Metoden POST sendes vha submit.

En @WebServlet indsætter en URL `"/FrontController"` ved navn FrontController.

I `from()` metoden indsættes parametrene `HttpServletRequest request` fra FrontControlleren. "command" kaldes fra `form`. I `HashMap` findes key som er "login" og value der er klassen `Login()`.

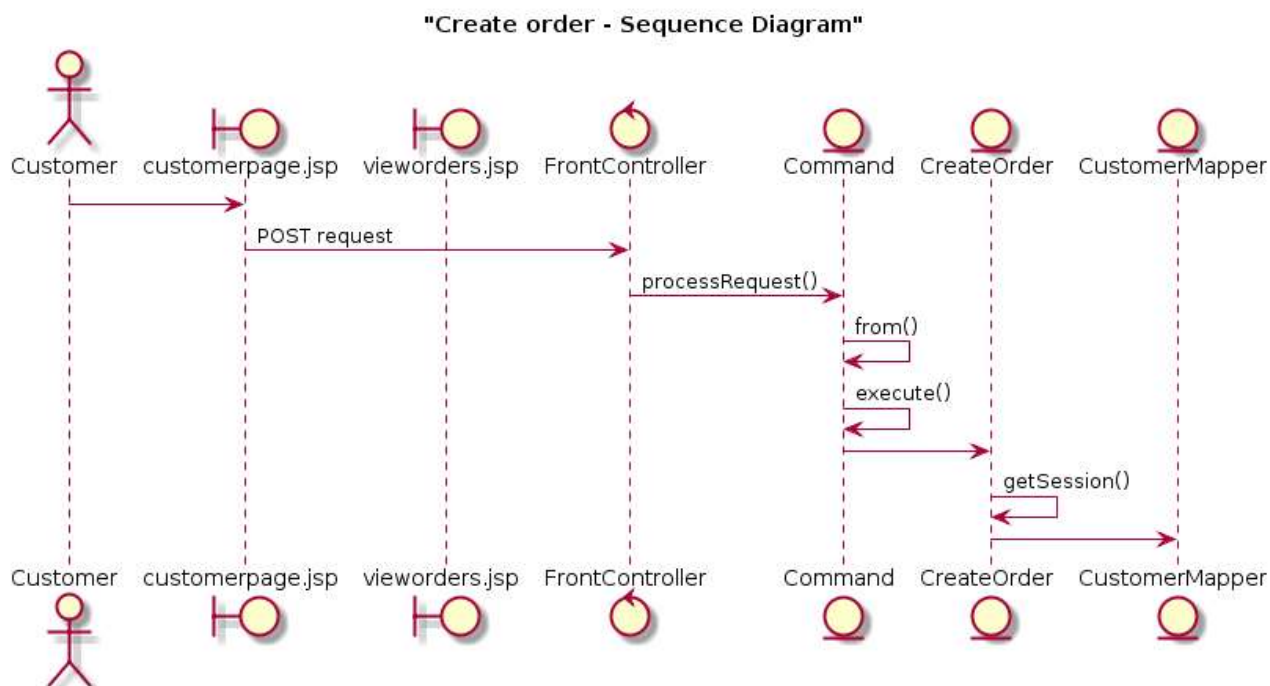
Den abstrakte klasse nedarver Login klassen. I email og password transporteres via. `FunctionLayer` til `DataLayer`, hvor metoden `login()` tager imod parametrene.

Der foretages udtræk fra databasen. `Next()` kører i en løkke indtil der ikke er flere rækker i tabellen og `next()` sættes lig false.

En `session.setAttribute` hvor customer gemmes i 30 minutter. `customer.isAdmin()` er false og "customerpage" returneres.

Stien "WEB-INF/customerpage.jsp" findes. Kunden forward'es til customerpage.jsp og modtager en besked mv.

Nedenstående vises et sekvensdiagram for oprettelse af en ordre:



Customerpage.jsp kalder FrontControlleren med en POST request af formen "createOrder". I customerpage.jsp er der et hidden field med navnet "command" og med værdien "createOrder". "@WebServlet" sender formen med typen "submit" til serveren på stien FrontController.

Metoden processRequest() i FrontControlleren kalder metoden "from" i klassen "Command", og sender en variabel med navnet "request" af typen "HttpServletRequest".

I klassen "Command" kaldes metoden "from". Variablen commandName af typen String sættes lig request objektet, hvor metoden getParameter() tildeles parameteren "command".

Metoden getOrDefault() bliver kaldt fra HashMap, som finder key og value. Hvor key er "createOrder" og value er klassen CreateOrder som instantieres. Hvis key ikke eksisterer, bliver klassen UnknownCommand() instantieret.



I FrontControlleren bliver variablen action af typen Command sat lig metoden from() af typen Command hvor parameteren request kaldes.

Alle klasser der bliver instantieret fra klassen Command skal nedarve klassen Command og implementere den abstrakte metode som hedder execute(). I metoden execute(), kaldes parametrene comment, length, width og height fra request objektet af typen HttpServletRequest.

Metoden createCustomer() bliver kaldt fra LogicFacaden, som bliver kaldt fra CustomerMapper med SQL statement INSERT.

I klassen CreateOrder oprettes et session objekt, hvor customerobjektet bliver tildelt. Session gemmer data når man skifter mellem de forskellige hjemmesider.

I FrontControlleren returnerer metoden execute() en String som er navnet på jsp siden. Dvs. "/WEB-INF/" + view + ".jsp", hvor view = "vieworders". Således at kunden bliver forwardet til WEB-INF/vieworders.jsp.

### **Særlige forhold**

Følgende informationer gemmes i session:

HttpSession objektet bruges til session management. En session indeholder information, der er specifik for en bestemt bruger på tværs af hele applikationen.

Når en bruger indtaster en hjemmeside for første gang HttpSession opnås via request.getSession (), får brugeren et unikt ID til at identificere sin session.

Dette unikke id kan lagres i en cookie eller i en anmodningsparameter.

HttpSession forbliver i live, indtil den ikke er blevet brugt til mere end den timeout-værdi, der er angivet i tag i deployment descriptor-fil (web.xml).

Standard timeout værdi er 30 minutter, dette bruges, hvis der ikke er angivet værdi i tag. Dette betyder, at når brugeren ikke besøger den angivne webapplikationstid, bliver sessionen ødelagt af servletbeholderen. Den efterfølgende anmodning vil ikke blive serveret fra denne session længere, servlet-beholderen vil oprette en ny session.

Brugeroplysningerne gemmes i sessionsobjektet ved hjælp af setAttribute () -metoden og senere, når det er nødvendigt, kan disse oplysninger hentes fra sessionen.

For at få værdien fra session bruges `getAttribute ()` -metoden til `HttpSession`-grænsefladen. Her hentes attributværdierne ved hjælp af attributnavne.

```
HttpSession session = request.getSession();  
Customer customer = (Customer) session.getAttribute("customer");  
customer.getCustomer_id();
```

Hvis en attribut allerede eksisterer, erstatter denne metode de eksisterende attributter.

Hvis der ikke findes noget objekt for den angivne attribut, returnerer metoden `getAttribute ()` null.

Exceptions håndteres således:

I `CustomerMapper` og `OrderMapper` håndteres exceptions vha. `try catch`.

```
try {  
} catch (SQLException | ClassNotFoundException ex) {  
    throw new LoginSampleException(ex.getMessage());  
}
```

`getMessage()`: Henter JDBC driverens fejlmeddelelse vedrørende en databasefejl. I dette tilfælde hentes en `SQLException` og/eller en `ClassNotFoundException`.

Brugerininputvalidering er lavet således:

I servletten `Register` sammenlignes `password1` med `password2`. Hvis de to passwords er ens, returneres `"customerpage"`. Ellers kastes en `LoginSampleException` med beskeden `"the two passwords did not match"`, og kunden skal igen indtaste brugeroplysninger.

```
if (password1.equals(password2)) {  
    Customer customer = LogicFacade.createCustomer(name, phone, email, password1);  
    HttpSession session = request.getSession();  
    session.setAttribute("customer", customer);  
    return "customerpage";  
}
```

```
} else {  
  
    throw new LoginSampleException("the two passwords did not match");  
  
}
```

Sikkerhed i forbindelse med login er lavet således:

I tabellen Customer i kolonnen email er valgt VARCHAR hvor der er sat flueben ved "Unique index". Dvs. at e-mailadressen skal være unik.

Følgende brugertyper er valgt i databasen:

I databasen er valgt datatyperne INT(), VARCHAR() og TINYINT(). I MySQL er TINYINT() og boolean synonymer. MySQL driveren konverterer TINYINT() til false, i Java, da Default/Expression er sat lig '0'.

PK, NN, UQ, AI, Default/Expression

hvor

PK: Belongs to primary key

NN: Not Null

UQ: Unique Index

AI: Auto Incremental

F.eks.: customer\_id er et heltal, som er sat til primærnøgle, hvor der skal være en værdi som automatisk forøges med 1 for hver gang der indsættes en ny række. Kolonnen email er sat til Unique Index. Dvs. hver e-mailadresse skal være unik for at kunne oprettes. Når kolonnenavn er markeret med blå, kræves det, at der indsættes en værdi i databasen.

### Udvalgte kodeeksempler

Command klassen er valgt ud som kodeeksempel:

Command klassen er en abstrakt klasse. Abstrakte klasser minder om interfaces. Du kan ikke instantierer dem, og de kan indeholde et mix af metoder deklareret med eller uden implementation.

I Command klassen findes HashMap. HashMap er en collection klasse som er brugt til at gemme Key & value pairs. HashMap<String, Command> kaldes, hvor klassen instantieres med Command.

```
abstract class Command {
```

```
private static HashMap<String, Command> commands;
```

```
private static void initCommands() {  
    commands = new HashMap<>();  
    commands.put( "login", new Login() );  
    commands.put( "register", new Register() );  
    commands.put("createOrder", new CreateOrder());  
    commands.put( "viewOrders", new ViewOrders());  
    commands.put("viewLineItems", new ViewLineItems());  
    commands.put("viewAllCustomers", new ViewAllCustomers());  
}
```

```
static Command from( HttpServletRequest request ) {  
    String commandName = request.getParameter( "command" );  
    if ( commands == null ) {  
        initCommands();  
    }  
    return commands.getOrDefault(commandName, new UnknownCommand() );  
}
```

```
abstract String execute( HttpServletRequest request, HttpServletResponse response )  
    throws LoginSampleException;  
}
```

## **Status på implementering**

Siderne mangler at blive stilet:

Der er ikke udarbejdet SVG tegninger.

Følgende usecases er ikke blevet implementeret:

As a <customer>, I want to <view carports> so that <I can decide if I want to buy a carport>.

As a <sales representative>, I want to <view carports> so that <I can give customer advice>.

Der mangler at blive udarbejdet en plantegning vha. SVG som viser hvordan carporten ser ud.

As a <sales representative>, I want to <mark orders open/closed> so that <I know which orders I need to handle>.

Arbejdsprocessen i projekt perioden er ikke blevet beskrevet.