

CPHBusiness

# Johannes Fog Semester Projekt

---

Dat16EA, gruppe A1, 29. maj 2017

David Carl  
Kristian Krog  
Tjalfe Møller  
Kasper Breindal

## Indhold

Business Case .....	4
Vision .....	4
Activity Diagram.....	6
Domain Model .....	8
Class-Activity diagram .....	9
Arkitektur og Design Mønstre .....	15
Lagdesign .....	15
Fordele ved 3 lags arkitekturen .....	16
Designmønstre.....	16
Sekvensdiagram .....	18
Brugergrænseflade og design.....	19
Gennemgang af grænseflade.....	19
Tekniske detaljer.....	23
Komplekse løsninger .....	25
2D render proces .....	25
3D render proces .....	27
SQL Queries .....	28
Sikkerhed .....	30
Hashing af password .....	30
PreparedStatement .....	31
Features.....	32
2D tegning.....	32
3D tegning.....	32
Stykliste.....	32
Adminpanel.....	32
Testing .....	33
Manuel Testing .....	33

Automatiseret Testing (JUnit).....	33
Fremtidige implementeringer .....	35
Autogenereret PDF fil .....	35
Gamle carporte design + ordrer .....	35
Bedre admin panel.....	35
Rewrite dele af koden .....	35
Kendte Fejl.....	37
Konklusion .....	39
Installation.....	40
Dokumentation .....	40
Proces Rapport .....	41
Backlog.....	41
Tasks.....	41
Tilbageblik.....	42
1. Sprint.....	42
2. Sprint.....	42
3. Sprint.....	43
Productowner meeting med PAB .....	43
Gruppesarbejde.....	44
Rapportskrivning.....	44
Arbejdsmetoder .....	45
Værktøjer .....	45
Scrum .....	46
Agile Development .....	46
Konklusion .....	47

## Business Case

Af David

Fog ønskede sig et nyt IT-system som skulle erstatte deres 20 år gamle system som de brugte. En af begrundelserne for at vi skulle bygge det nye til dem var fordi de havde mistet source-koden til programmet så de ikke kunne opdatere det mere, og deres program begyndte at blive langsommere og mere besværligt for deres medarbejdere at bruge.

Lige nu får Fog en email når en kunde bestiller et tilbud som de selv designer, hvorefter at en medarbejder selv skal skrive værdierne ind i deres program for derefter at få pris og materialeliste. Der er desværre ingen tegning på hvordan carporten kommer til at se ud. Efter betaling af den ønskede carport får kunden udleveret en byggevejledning som PDF som en medarbejder har skulle sidde og lave manuelt med de ting som ville være relevant for kundens carport.

De havde nogle krav til det her IT-system.

Mulighed for at få kontakt med kunden, hvordan havde de ikke specificeret til os.

Fog ønskede en 2D tegning der gjorde det muligt at få en idé om hvordan carporten kom til at se ud, men det måtte ikke være for detaljeret så kunden kunne lade være med at købe carporten af dem, men købe materialerne hos en konkurrent, for derefter at bygge den selv.

De sagde samtidig også at hvis man kunne lave 3D model ville det være en ting at foretrække og vi meget gerne måtte gøre det.

Man skulle ikke kunne købe carporten direkte da fog gerne ville havde kontakt til kunden inden de begyndte at sætte sig ind i det store forløb, for at sikre sig at kunden vidste hvad de gik ind i, og så fog kunne komme med forslag og sikre sig at alt var i fineste orden.

## Vision

Af David

Vi vil forbedre Fogs carport-system for at skabe:

- Højere kundetilfredshed
- Højere effektivitet for Fogs medarbejdere

Idéen med programmet er at gøre det enklere for medarbejderne, at håndtere de 'ønsker' kunderne kommer med. Det skal samtidig være en bedre oplevelse for kunden bestille en ikke standard carport fra fog. Dette har vi planer om skulle ske ved at vi laver en visuel repræsentation af carporten undervejs, så kunden nemt kan lege rundt med forskellige mål og på den måde kan se hvordan den

ville komme til at se ud. Dette kunne vi godt tænke os at gøre med noget 3D og 2D tegning. 3D for kunden så de kunne se den fra alle vinkler, og 2D for at få det godkendt hos kommunen.

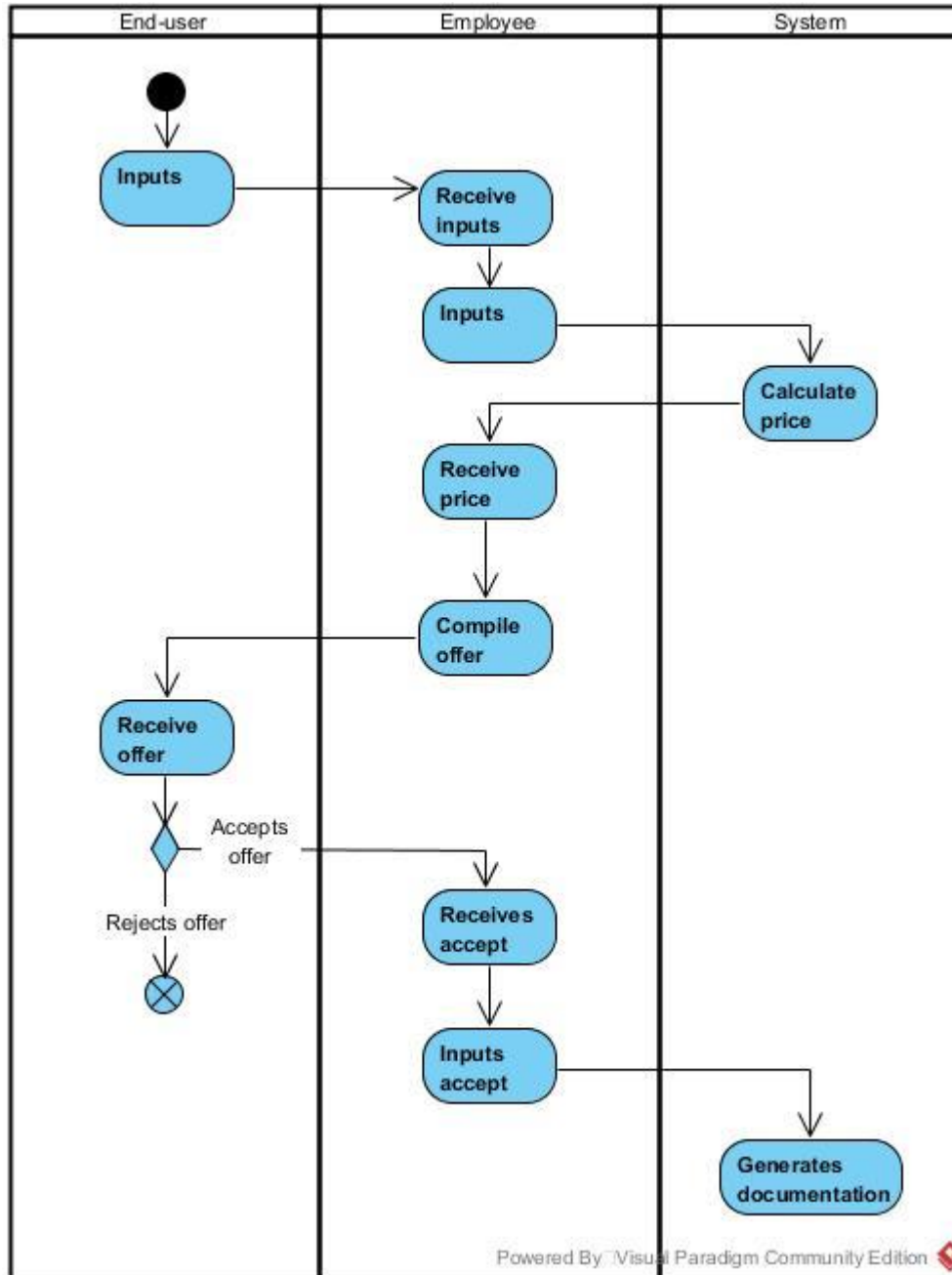
Vores mål for projektet er:

- At sænke behandlingstiden pr. ordre med 20%
- Øge kundetilfredsheden med 20% ved næste måling
- At give kunden noget at se før carporten rent faktisk står færdig

## Activity Diagram

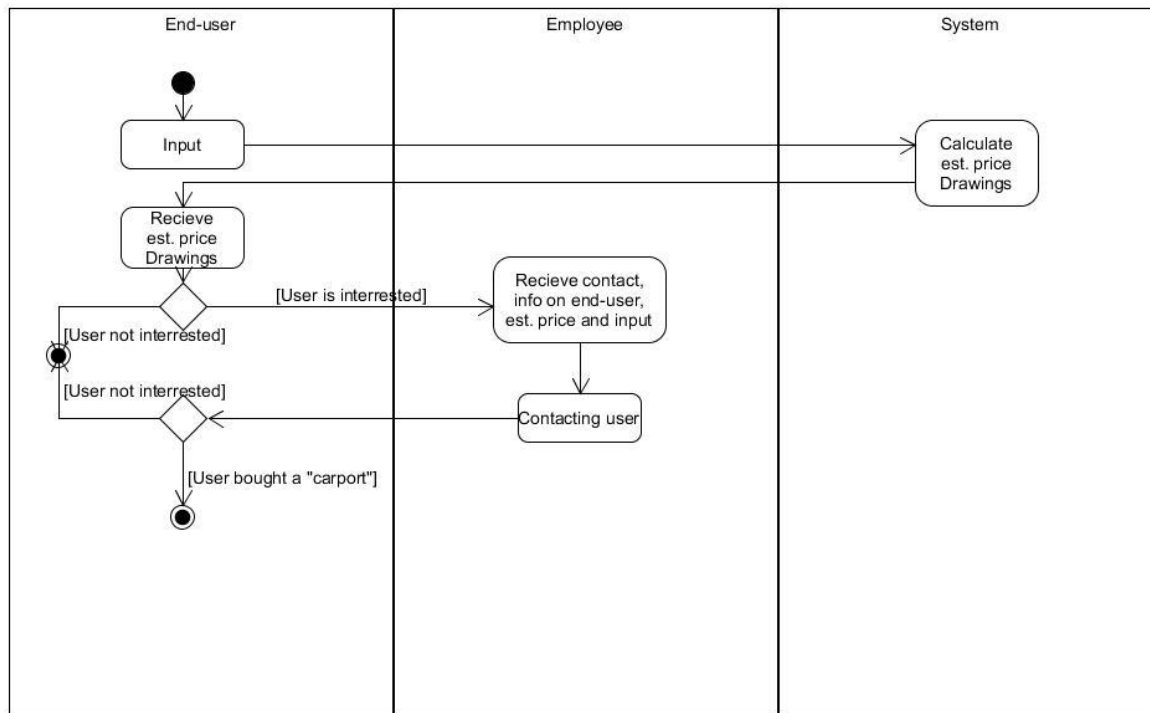
Af Kristian

*As is – som systemet ser ud nu.*



As is diagrammet viser den nuværende arbejdsgang med det nuværende system.

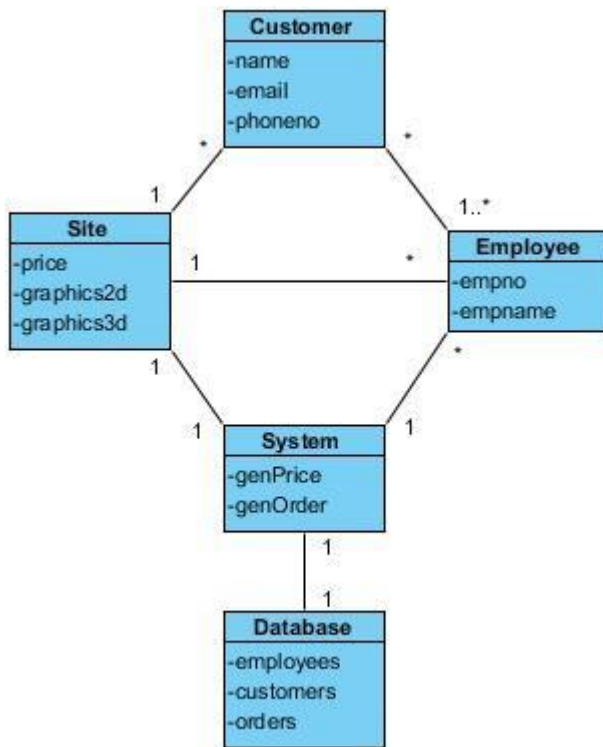
*To be – som systemet optimalt kunne se ud*



To be diagrammet viser hvordan vores program automatisk håndterer en del ting som Fog-medarbejderen tidligere skulle håndtere selv. Det vil effektivisere arbejdsgangen og give mere tid til customer maintenance ved hver salg. Det vil også skære behandlingstiden ned som helhed, da kunden kan se en estimeret pris før kunden kontakter Fog omkring endelig pris og køb.

## Domain Model

Af Kristian



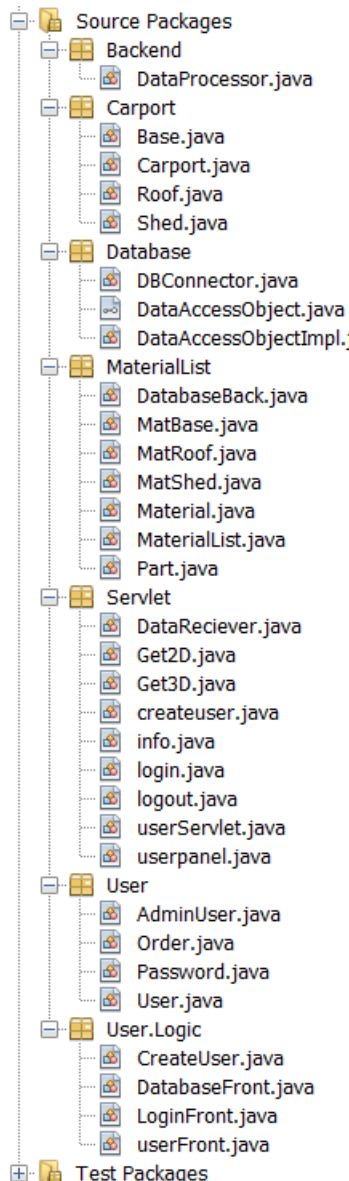
Vores domænemodel illustrerer simpelt hvordan vi har tænkt løsningen skulle opbygges. Kunder vil aldrig komme i kontakt med andet end en frontend, der vil vise dem en cirka pris og et par autogeneratede modeller af deres ønskede carport, og en af Fogs sælgere. Siden har adgang til systemet primært for at kunne generere en cirka pris, men også for at kunne gemme kundens info (sker kun hvis kunden ønsker det). Fogs ansatte har også adgang til systemet, både for at kunne se den eksakte pris, og for at kunne se en liste over materialer. Al kontakt til databasen sker via vores system og aldrig direkte.

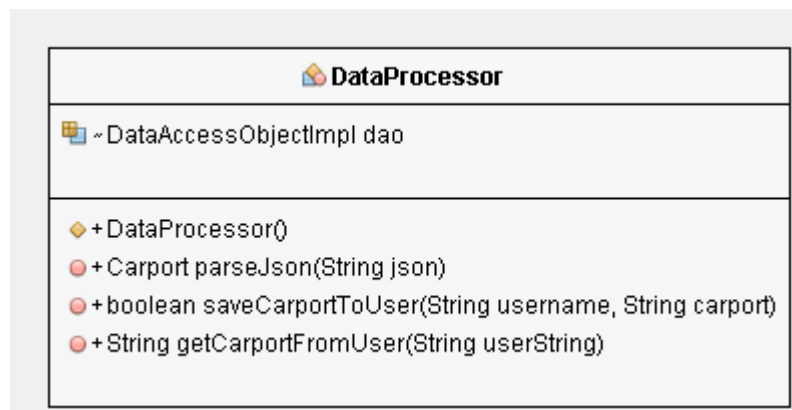


## Class-Activity diagram

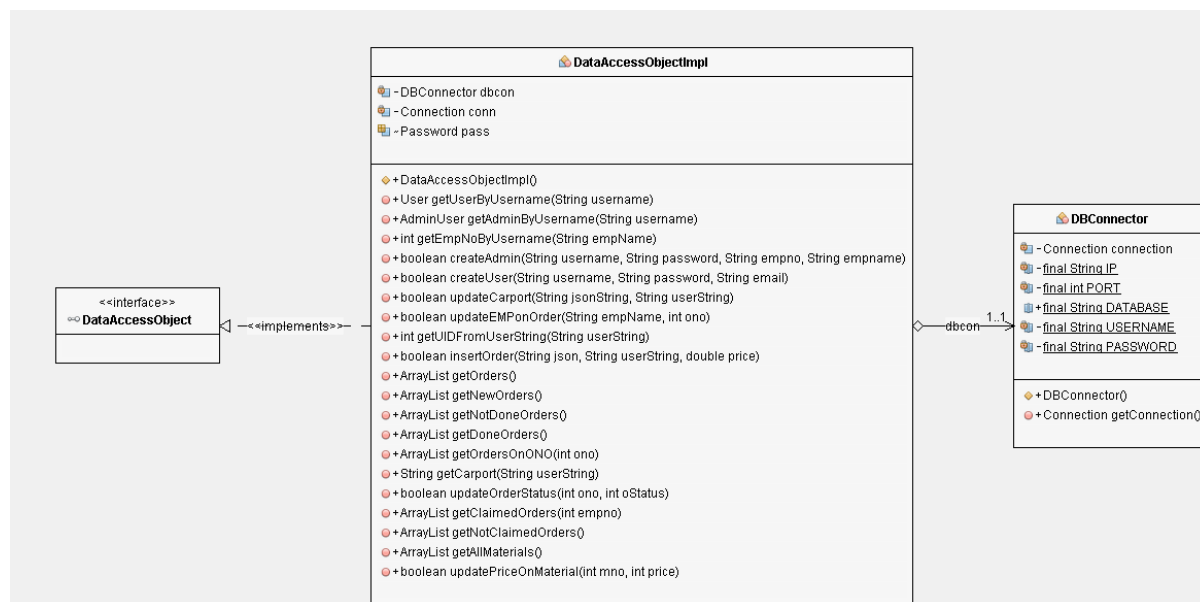
Af Kristian

Selvom vores program er 3 lags arkitektur har vi valgt at sortere de enkelte classes i packages ud fra deres direkte tilhørsforhold. Selvom nogen classes importerer fra andre classes i andre packages, har vi valgt at opdele klassediagrammet i enkelte packages, for på den måde at forbedre overskueligheden.

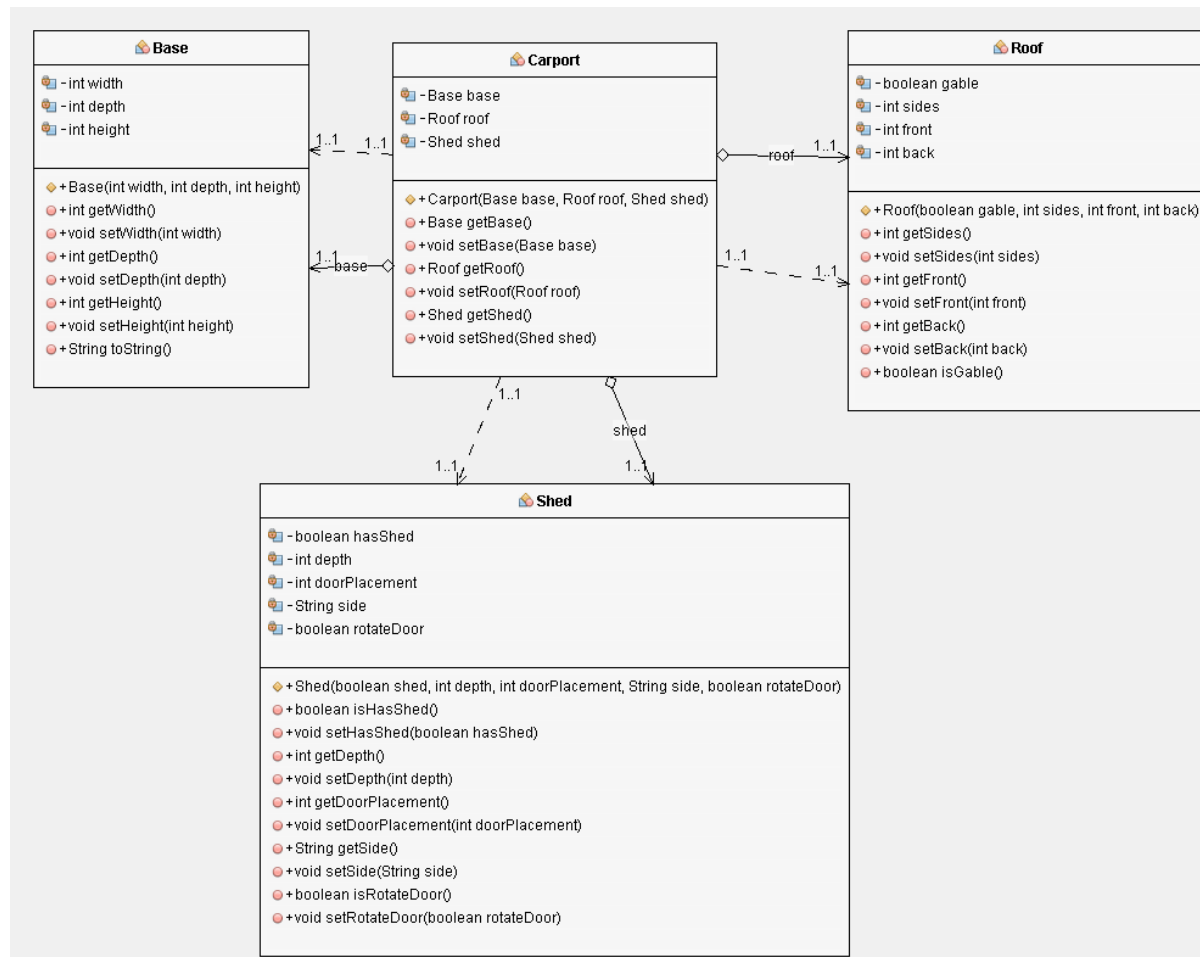




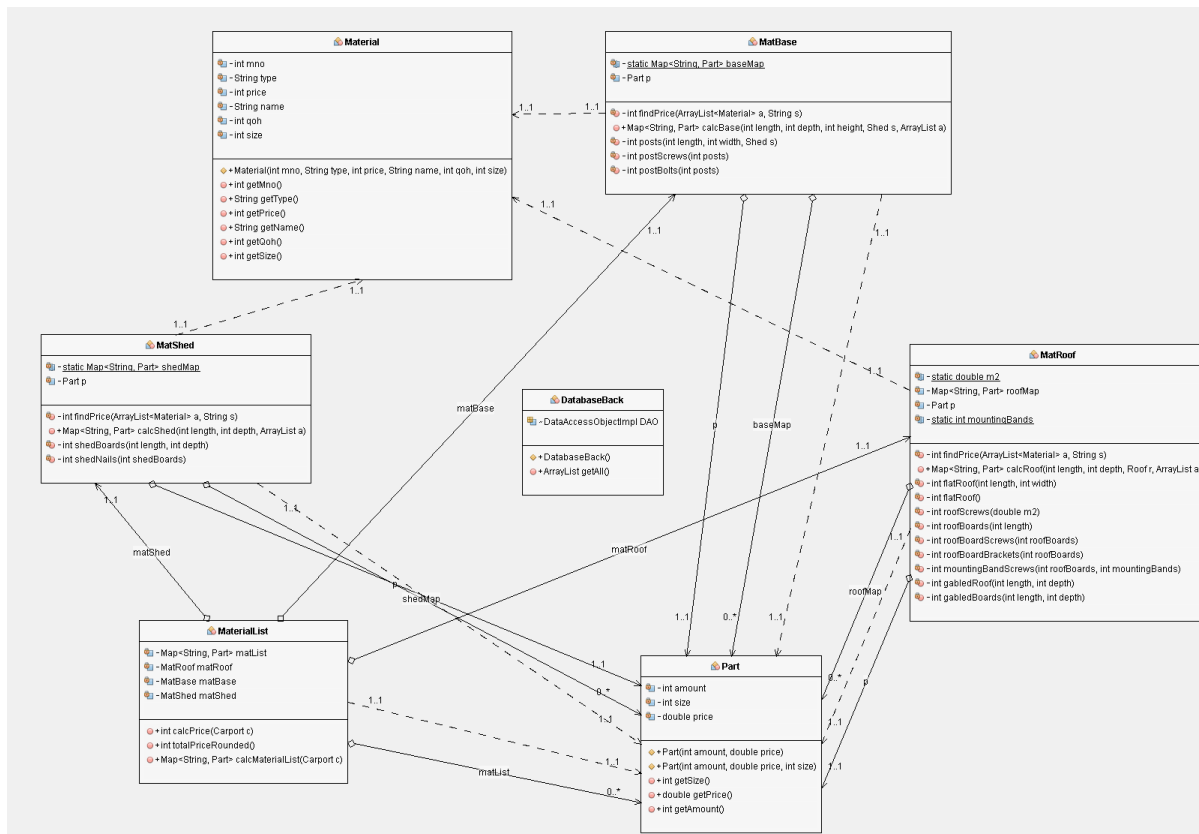
Vores backend package indeholder kun DataProcessor klassen, der bruges til både at lave en JSON string til at gemme en carport i databasen og til at hente en JSON string fra databasen og lave et carport-objekt til at udregne prisen på en carport og generere en instans af MaterialList ud fra denne.



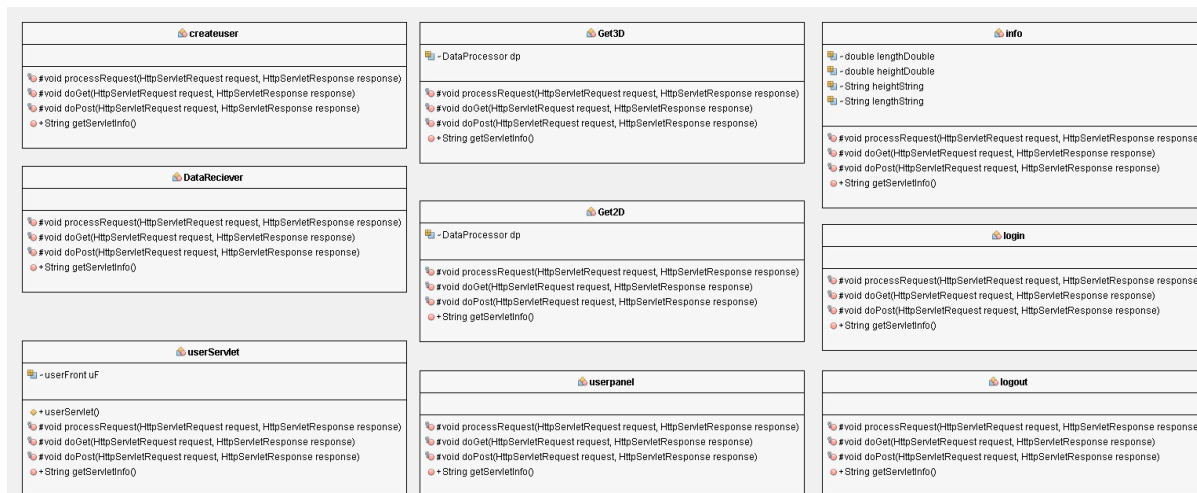
Vores Database package har et DataAccessObjekt interface og dets implementering. Database indeholder også en DBConnector class til at kontakte selve databasen.



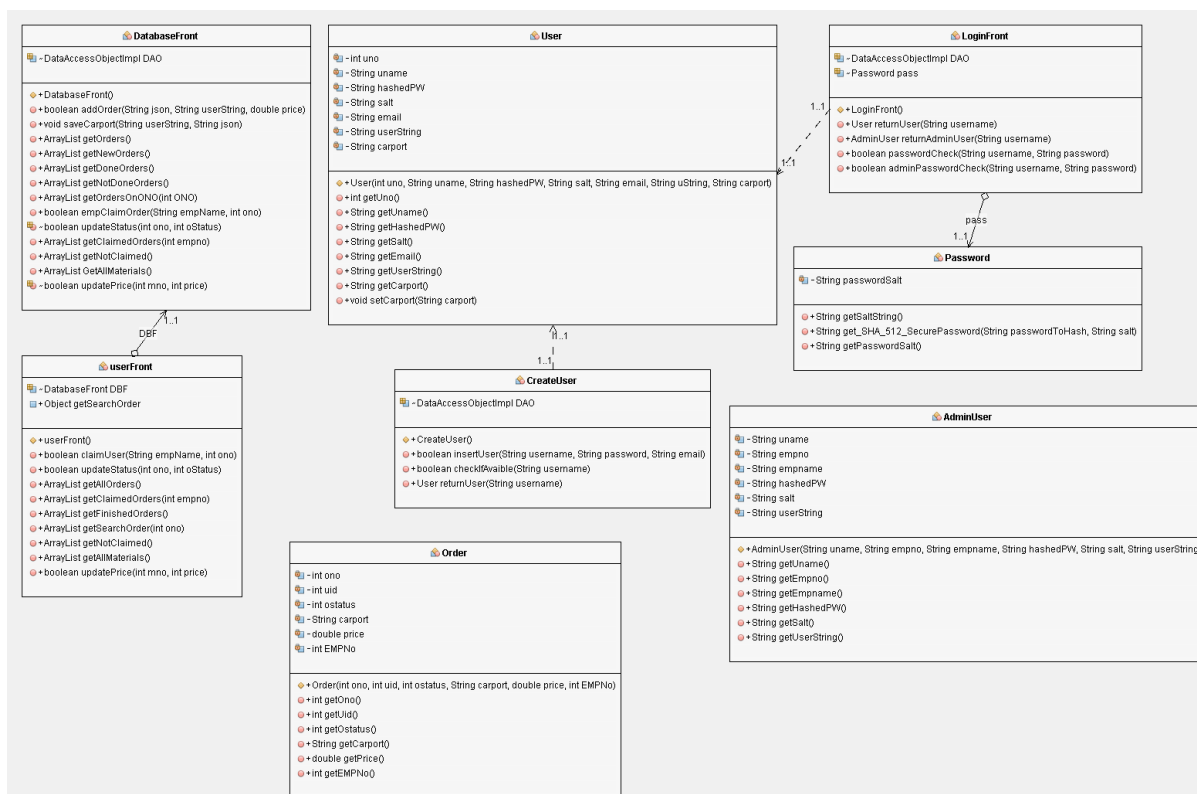
Carport package bruges til at opbevare info fra en JSON string fra vores database. Hver Carport objekt indeholder et Shed, et Base og et Roof objekt.



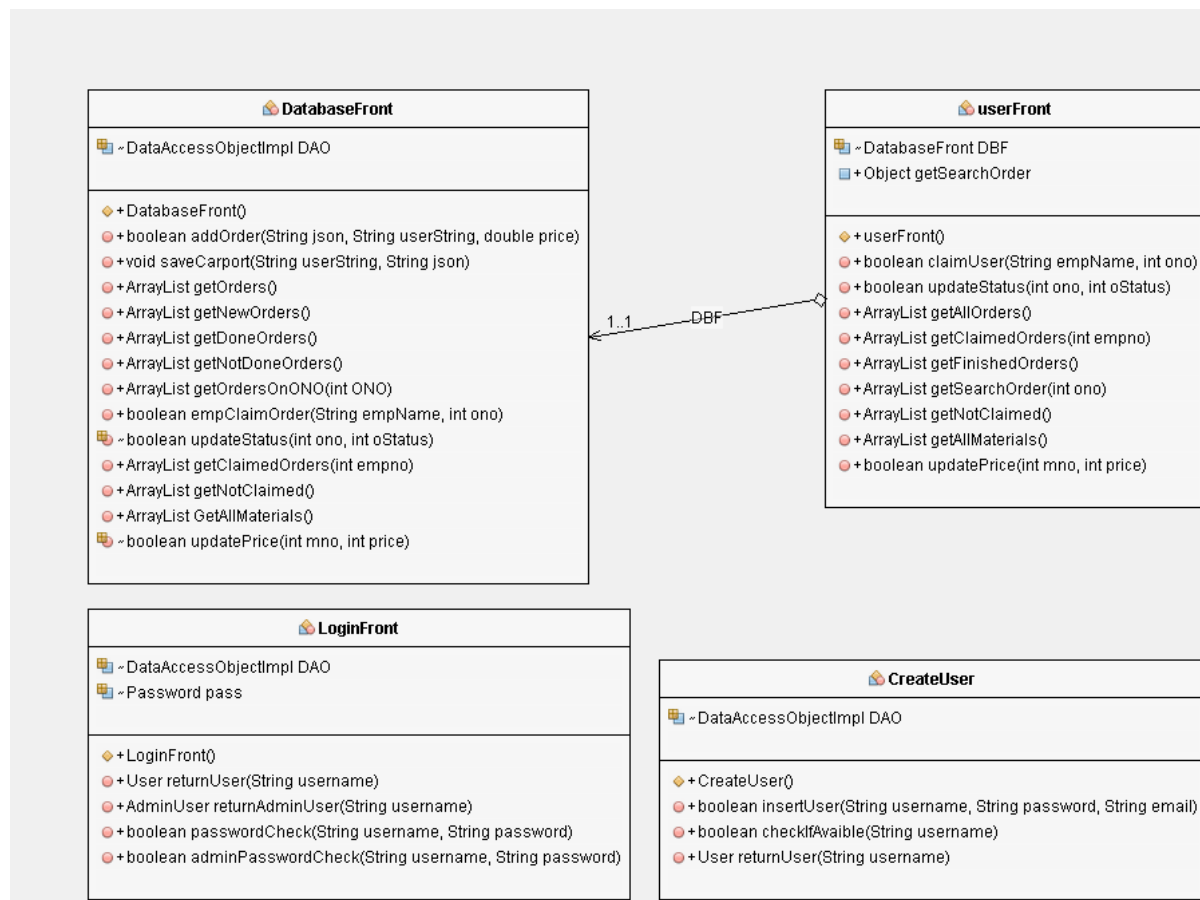
Vores MaterialList package indeholder alle de klasser der bruges til at oprette en MaterialList(stykliste) og ud fra den udregne en pris for materialer. Packagen indeholder også to constructor classes for objekterne Material og Part. Material bruges til midlertidigt at opbevare databasens material table. Part bruges da MaterialList opbevares som Hashmap med String som key og Part som value.



I vores servlet package ligger hele vores presentation layer. Alt hvad en bruger ser bliver ført gennem denne package.



Userpakagen indeholder constructor classes til user info og til oprettelse af nye users.



User.Logic indeholder de classes der kommunikerer user info mellem database og frontend.

## Arkitektur og Design Mønstre

Af David

Når man skal bygge en applikation som skal kunne vedligeholdes og skrives på af mange forskellige programmører, er det en god ting at følge en anerkendt arkitektur. Dette gør det nemmere at vedligeholde i form af at en programmør ikke skal bruge længere tid på at lære programmet at kende end at vedligeholde det. Det gør det også nemmere at arbejde på sammen flere ad gangen, da det giver nogle faster rammer til hvordan det skal se ud.

### Lagdesign

Vi har valgt at bruge 3 lags arkitekturen. Dette er en velkendt teknik blandt IT-folk så derfor gør det den nem at arbejde med. De 3 lag er følgende:

- Præsentations-laget
- Logik-laget
- Data-laget

#### *Præsentations-laget*

Dette er hvad brugeren af vores applikation kommer til at ende med at se. Det er her at brugeren kommer med input og får de output vores program giver dem. I vores tilfælde er det gennem en webbrowser, da det vi har lavet er en hjemmeside. Alt det brugeren bliver serveret sker igennem JSP filer, men bruger aktivt logik laget for at finde ud af hvad den skal vise til brugeren. Dette sker igennem servlets.

#### *Logik-laget*

Det er her den meste logik foregår, det foregår igennem servlets som bruger metoder fra andre Java classer vi har lavet. Her foregår det som brugeren ikke skal se, som i vores tilfælde for eksempel er udregning af pris og dermed også en stykliste over de ting, som kunden skal bruge til at bygge deres carport de designer. Her foregår alt kommunikation også til vores data lag, også selvom det er vores præsentation lag som skal bruge det.

#### *Data-laget*

Data lageret bliver kun brugt til kommunikation til vores MySQL database som under udvikling har ligget på vores DigitalOcean Droplet (Dette er en virtual linux server), dette sker igennem SQL calls. Dette er det eneste sted hvor der faktisk bliver snakket med databasen.

### Eksempel

Når vores præsentations lag skal vise for eksempel noget med ikke færdige ordrer, så sender den en efterspørgsel til vores logik lag. Vores logik lag finder så ud af præcist hvad det er vores præsentations lag er interesseret i og spørger efter det i vores data lag. Vores data lag henter så de data ud af vores database, og sender det tilbage til vores logik lag, som så gør det klar til at vores præsentations lag ved hvordan det skal præsenteres til medarbejderen som har efterspurgt det og på den måde fungere de 3 lag sammen.

### Fordele ved 3 lags arkitekturen

- Det er nemt at begynde at arbejde med, og forstå hvordan det allerede er sat op hvis man kender til 3 lags arkitekturen.
- Hvis man skal vedligeholde noget fra et lag behøver man nødvendigvis ikke pille hvad alt andet i de andre lag.
- Genbrugsmulighed ved de forskellige lag.
- Det er nemmere at arbejde flere på samme projekt ved 3 lags arkitekturen end ved at bare at lave det på 3 forskellige måder, da 3 lags arkitekturen har nogle faste rammer for hvordan det fungere.
- 3 lags arkitektur er også kendt for at være en arkitektur med lav kobling og høj samhørighed der som tidligere sagt gør det nemt at vedligeholde eller helt udskifte dele af programmet.

### Designmønstre

*"The interface lets you take greater advantage of polymorphism in your designs, which in turn helps you make your software more flexible." - Bill Venners, JavaWorld<sup>1</sup>*

Der er mange gode grunde til hvorfor man skal bruge noget som for eksempel interface, en af de grunde er at det bliver nemmere og skulle ind og få et overblik over hvad man allerede har af metoder. En anden ting er også det bliver nemmere at skifte motoren bagved, eller bare ændre nogle småting, det er også lav kobling som vi snakkede om tidligere og som er en god ting at have med.

I vores projekt har vi brugt interface til vores database del, dette er med til at gøre det nemmere hvis man pludselig finder ud af at man vil væk fra den database type vi bruger. Når vi bruger interfacet, så ved de kommende udviklere lige præcis hvilke metoder der skal være der og hvad de skal returnere

---

<sup>1</sup> <http://www.javaworld.com/article/2076841/core-java/designing-with-interfaces.html>



og hvad de skal hedde, og skal have af input. Det gør man ikke skal bruge den første halvdel af tiden på at forsøge at læse koden og forstå den, når den alligevel ikke skal bruges mere.

```
ArrayList getClaimedOrders(int empno);

ArrayList getDoneOrders();

ArrayList getNewOrders();

ArrayList getNotClaimedOrders();

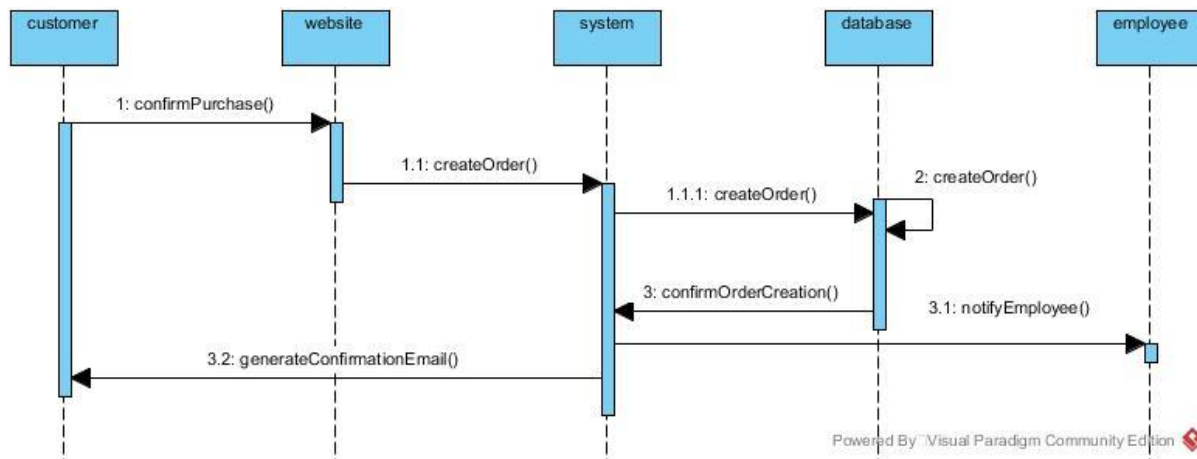
ArrayList getNotDoneOrders();

ArrayList getOrders();

ArrayList getOrdersOnONO(int ono);
```

Det her er en sammenhængende del af vores kode, vi kan se her at de alle sammen returner en ArrayList, og så også se deres navne. Vi kan samtidig se at 'getOrdersOnONO(int ono)' og i 'getClaimedOrders(int empno)' (ono = order number, empno = employ number) metoderne skal have en int ind. I den første "getOrdersOnONO" skal vi have have en order tilbage specifikt på et order nummer. I "getClaimedOrders" skal vi have alle de ordrer som den givne medarbejder har ansvaret for. I de andre skal vi bare have alle ordrer ud af en specifik type. Denne type kender vi allerede så derfor skal vi ikke sende en variable med ind til metoderne. Dette er et godt eksempel på hvordan vi har brugt det.

## Sekvensdiagram



Vores simplificerede sekvens diagram illustrerer, hvordan et ordreforløb vil foregå for kunden. Grunden til, at vi har valgt at simplificere det, er den rå mængde klasser der bliver kaldt i et realistisk forløb. Sekvens diagrammet forudsætter at kunden allerede er logget ind på en profil hos Fog. Når kunden bekræfter sit ønske om at købe en carport, vil det først blive logget i den webapplikation hvor kunden har designet sin carport. De data vil derefter blive sendt videre til selve systemet, hvor de relevante data vil blive omdannet til et ordre objekt der så bliver videresendt til databasen hvor ordren bliver oprettet, og kædet sammen med den relevante kunde. Når systemet får bekræftet at ordren er oprettet vil det, via admin panel, informere en sælger om at der er en ny mulig køber, og vil efterfølgende informere kunden om, at ordren er modtaget.

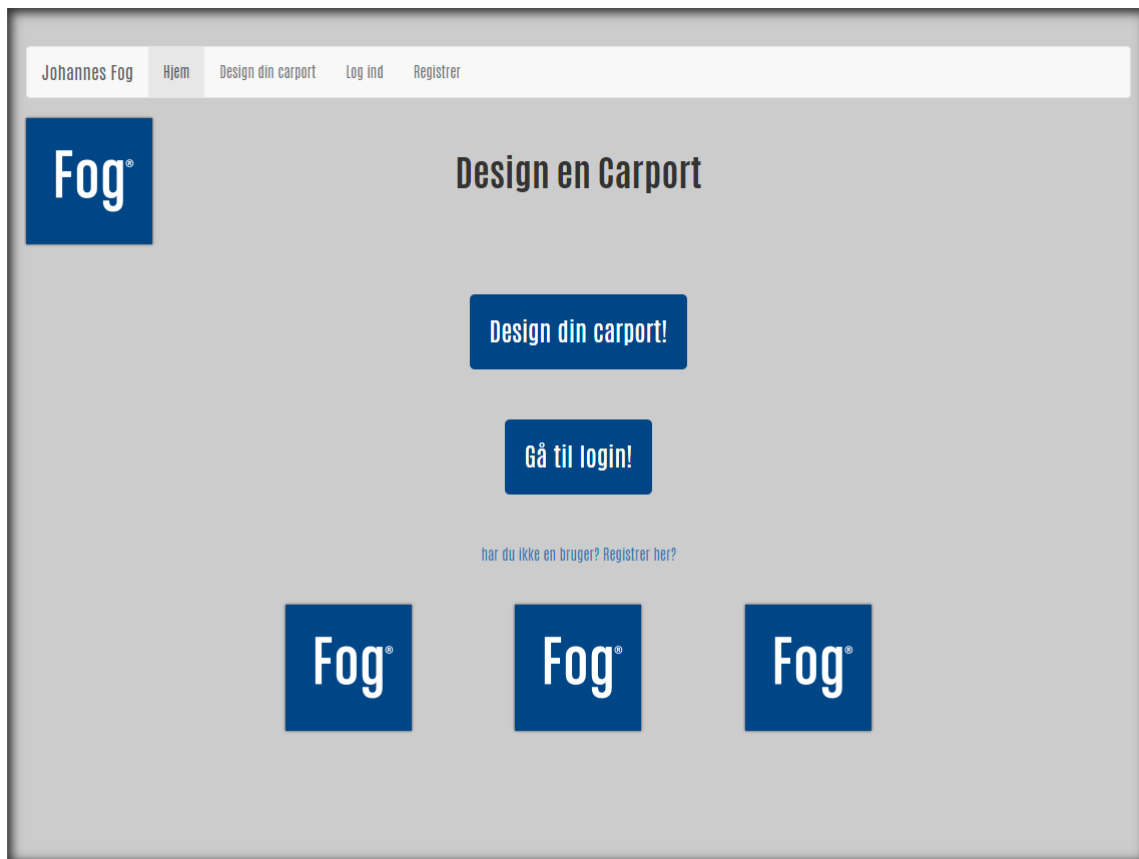
## Brugergrænseflade og design

Af Kasper

Vores mål med programmet var at skabe en nemt og velfungerende måde at designe en carport på. Det skulle føles intuitivt, og ikke skabe forvirring når man arbejder i programmet. Programmet har været igennem flere forskellige stadier, før det nåede frem til vores endelige løsning.

### Gennemgang af grænseflade

*For brugere*

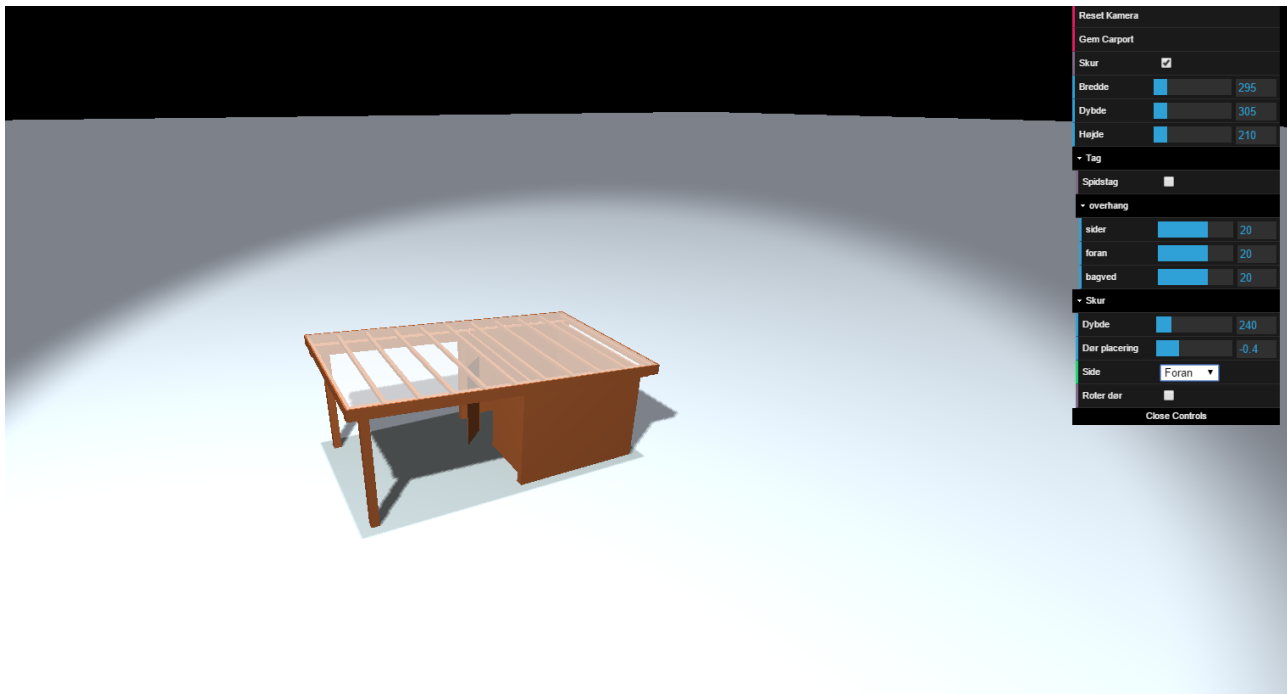


Det første der skal tages stilling til for brugeren af siden er, om man vil logge in, registrere sig, eller bare prøve at designe sin carport. Dette valg er bevidst stillet op for at tiltrække så mange potentielle kunder som muligt.

Kunderne vil ikke som det første på siden oprette sig. Derfor får de muligheden for at arbejde med designværktøjet først, og gemme det senere.

Designmæssigt har vi ikke haft mulighed for at snakke med productowner Johannes Fog, og derfor har vi implementeret et design som er simpelt og nemt at finde rundt i.

Navigationsbaren i toppen kan bruges til at lede brugeren hen på den side de leder efter og har Johannes Fogs logo sat på, så man aldrig er i tvivl om hvor man befinder sig.

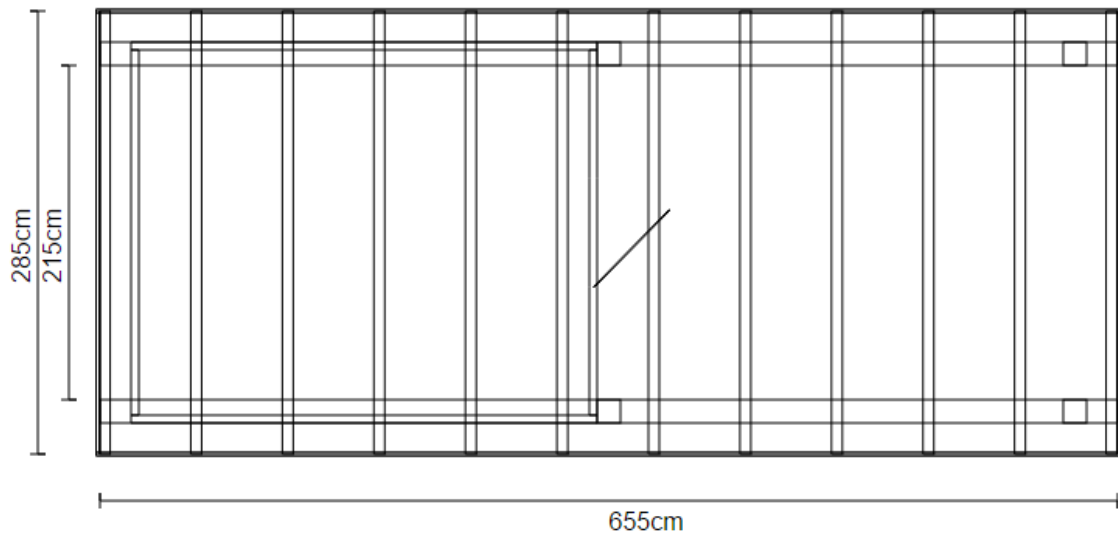


Ved tryk på "design din carport" knappen ledes brugeren til vores 3D design program. Her kan brugeren køre sliders frem og tilbage, og vurdere hvordan deres carport skal se ud, og hvilke specifikke mål den skal have. Dette har den fordel at kunderne får en grafisk præsentation af hvordan deres carport kommer til at se ud, i stedet for bare et billede af en tilfældig anden carport som firmaet engang har bygget. Når kunden føler at deres carport er som den skal være klikker de på "gem" knappen, og bliver herefter ledt videre til plantegningen af deres carport.

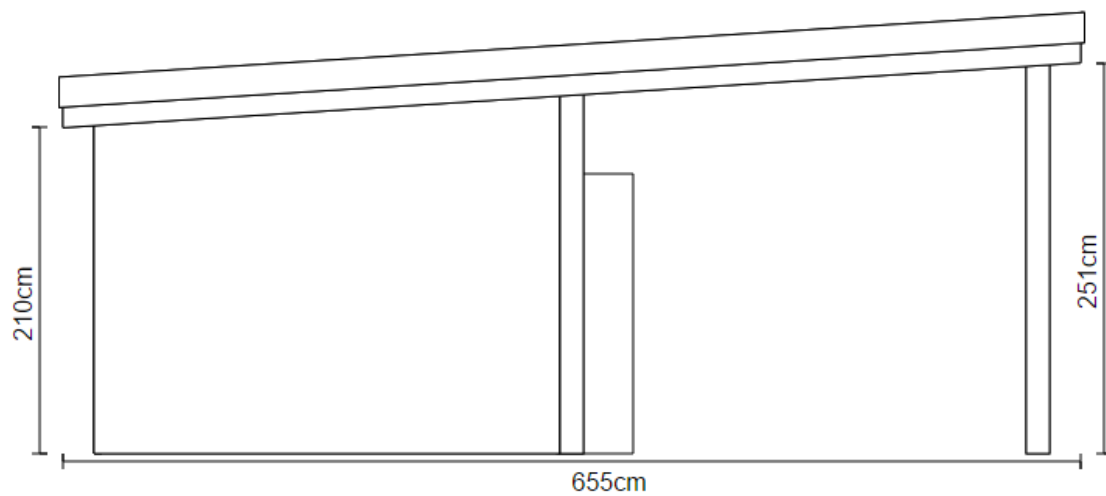
[Tilbage til brugerpanel](#)

Pris: 10000

Top:



Side:



Her får kunden en 2D-plantegning af den carport de netop har designet i 3D-programmet. Tegningen har ikke alle mål sat på - dette er gemt til kunder der vælger at købe den carport de har designet.

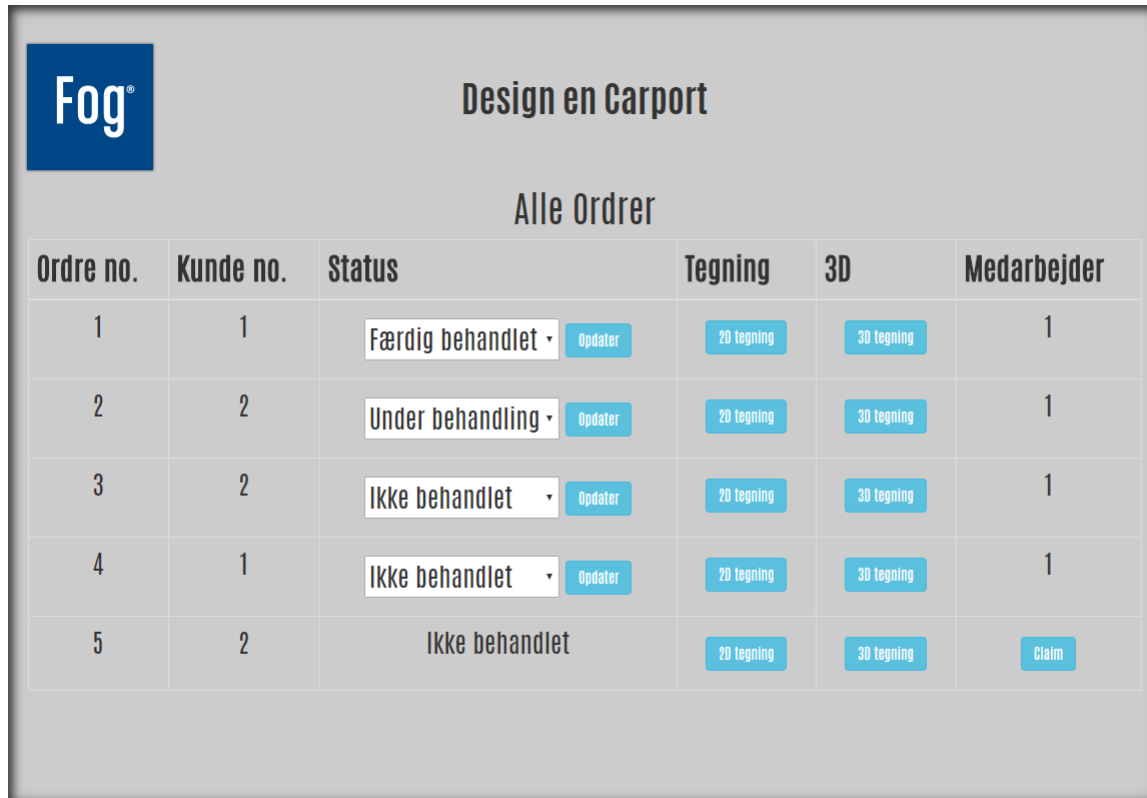
Samtidig med at tegningen er blevet genereret er carporten blevet gemt til brugeren, hvis man er logget ind. Hvis ikke, vil den bagefter bede dig om at logge in eller oprette dig.



Til slut kommer brugeren til brugerpanelet. Her kan de se den carport de har gemt, og kan trykke bestil, hvorefter ordren bliver sendt til databasen, til videre behandling af Fogs medarbejdere.

*for Fogs medarbejdere*

Fogs medarbejdere kan benytte dette samme system til behandling af ordrerne. Hvis en medarbejder logger ind vil de blive taget til administratorpanelet. Her kan medarbejderne se status på de forskellige ordrer - om de er helt nye, under behandling af en kollega eller færdigbehandlet og klar til levering. Medarbejderne kan i forvejen om de kun vil se deres egne, ledige, eller andre specifikke sager.



Ordre no.	Kunde no.	Status	Tegning	3D	Medarbejder
1	1	Færdig behandlet • Opdater	2D tegning	3D tegning	1
2	2	Under behandling • Opdater	2D tegning	3D tegning	1
3	2	Ikke behandlet • Opdater	2D tegning	3D tegning	1
4	1	Ikke behandlet • Opdater	2D tegning	3D tegning	1
5	2	Ikke behandlet	2D tegning	3D tegning	Claim

### Tekniske detaljer

Websiderne i projektet er skrevet i JSP, CSS og JavaScript, og benytter sig af JSTL og eksterne libraries så som THREE.js, JQuery og Bootstrap til at forbedre udseendet af projektet og gøre projektet kompatibelt med flere forskellige browsere og styresystemer. Så vidt muligt bliver disse libraries hentet eksternt, via et Content-Delivery Network (CDN). Dette skaber den fordel at brugerne kan have disse pakker cached fra tidligere brug på andre websider i forvejen, og derfor ikke behøver at hente dem igen. Det betyder også at pakkerne ikke kommer til at ligge på firmaets server, og derfor kan lette datatrafikken på serveren, og derved køre mere stabilt.

Workflowet er udarbejdet efter at kunne indsættes i et hvert webside-design, og til at kunne omskrives til andre programmeringssprog.

Dette giver os som gruppe muligheden for at udvikle i det sprog vi kender til (Java), samtidig med at det kan omskrives til, for eksempel, en C# og ASP.Net baseret kodebase, hvilket er meget udbredt i Danske virksomheder.



## Komplekse løsninger

### 2D render proces

Af Tjalfe

Fog ønskede at systemet var i stand til at give en tegning af carporten ud fra de indtastede mål.

Vores første intention var alt holde alt koden væk fra kunden og lave tegningen på serveren, så derfor skulle vi finde en måde at tegne i Java. Vi kom hurtigt frem til to metoder indbygget i Java, `BufferedImage` og `Graphics2D`, som vi kunne lave og gemme billeder i mange forskellige file typer (jpeg, bmp, png). Dog var der nogle ulemper ved denne måde i form af filstørrelse og pixelering ved skalering.

Første version - se Draw – `BufferedImage` på næste side.

Da vi i anden omgang kom forbi og skulle tilføje flere detaljer til 2D tegningen valgte at det skulle hen til SVG, da dette ville forbedre filstørrelsen, men også give bedre mulighed for skalering, uden at billedet og teksten blev pixeleret. Samtidig havde vi det problem at vores to forskellige måder til at tegne carporten på var i forskellige filer og sprog, så små ændringer til metoden vi lavede carporten på skulle rettes forskellige steder. Dette ville ikke være til at holde styr på i den lange bane.

Så vi valgte at SVG skulle generes i JavaScript, og gøre så både 2D og 3D blev tegnet ud fra det samme udregninger, så ændringer eller nye features ville automatisk komme ind på begge.

På dette tidspunkt havde vi allerede fået skrevet 3D-render om til at være objekt-orienteret, så andre filer kunne bruge det samme JavaScript på andre måder. Så vi manglede bare at skrive noget til at oversætte dette til SVG. Vi fandt et eksternt JavaScript-library, `SVG.js`, så vi ikke skulle til at lave alt sammen fra bunden. Så vi manglede bare et JavaScript imellem `SVG.js` og vores JavaScript der gav alle målene til carporten. Hvilket blev til `SvgMaker.js` i sidste ende.

### *Draw – BufferedImage/Graphics2D*

```
public BufferedImage drawTest() {  
    int width = 200;  
    int height = 200;  
    BufferedImage bimage = new BufferedImage(width, height, BufferedImage.TYPE_BYTE_INDEXED);  
  
    //background color  
    Graphics2D g2d = bimage.createGraphics();  
    g2d.setColor(Color.WHITE);  
    g2d.fillRect(0, 0, width, height);  
  
    //other  
    g2d.setColor(Color.red);  
    g2d.fill(new Ellipse2D.Float(0, 0, 200, 100));  
    g2d.dispose();  
  
    //savePNG(bimage);  
    return bimage;  
}
```

### *Servlet – konvertering af billedet/prep for html*

```
ByteArrayOutputStream baos1 = new ByteArrayOutputStream();  
ImageIO.write(d2d.drawImage(heightInt, depthInt), "png", baos1);  
baos1.flush();  
byte[] imageInByteArray1 = baos1.toByteArray();  
baos1.close();  
String b64_Side = javax.xml.bind.DatatypeConverter.printBase64Binary(imageInByteArray1);  
request.getSession().setAttribute("b64_Side", b64_Side);
```

### *JSP – Recieve og display*

```
<p>Taget:</p>  

```

## 3D render proces

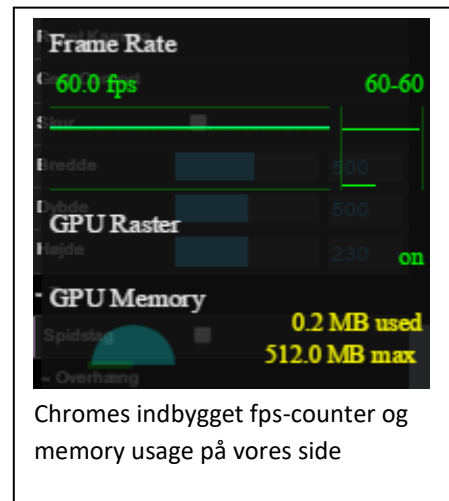
Af Tjalfe

I en samtale med Fog havde de gjort udtryk for, at godt kunne tænke sig at få en 3-dimensional repræsentation af carporten.

Kort tid inde i forløbet begyndte vi derfor at undersøge hvilke muligheder, der var inden for 3D render i browseren. Vi landede hurtigt på THREE.js pga. dens hurtige responstid, interaktive muligheder (kontrol af kameraet og live-opdatering af objekterne) og lave systemkrav for at køre.

Samtidig bruger THREE.js WebGL, som er integreret i næsten alle browsere uden et krav for extra plug-ins, så selv diverse mobile platforme som mobiler og tablets har mulighed for at køre dette, så længe JavaScript aktiveret.

Vores intentioner blev hurtigt at hele design fasen skulle ske i 3D, da det gav den bedste visuelle repræsentation, derfor havde vi brug for en eller anden form for GUI til at tage imod brugerens input. Dette kunne gøres på to måder, gennem html elementer (input-field, sliders, dropdown osv.) eller gennem JavaScript, vi fandt JavaScript til at være overlegen dels i design, databehandling og i forhold til placering over THREE.js canvas. Derfor valgte vi at bruge dat.gui.js som gjorde, at vi kunne have det som et overlay på vores canvas og sliders kunne give brugeren begrænsninger for at indtaste absurde/forkerte mål.



## SQL Queries

Af Kasper

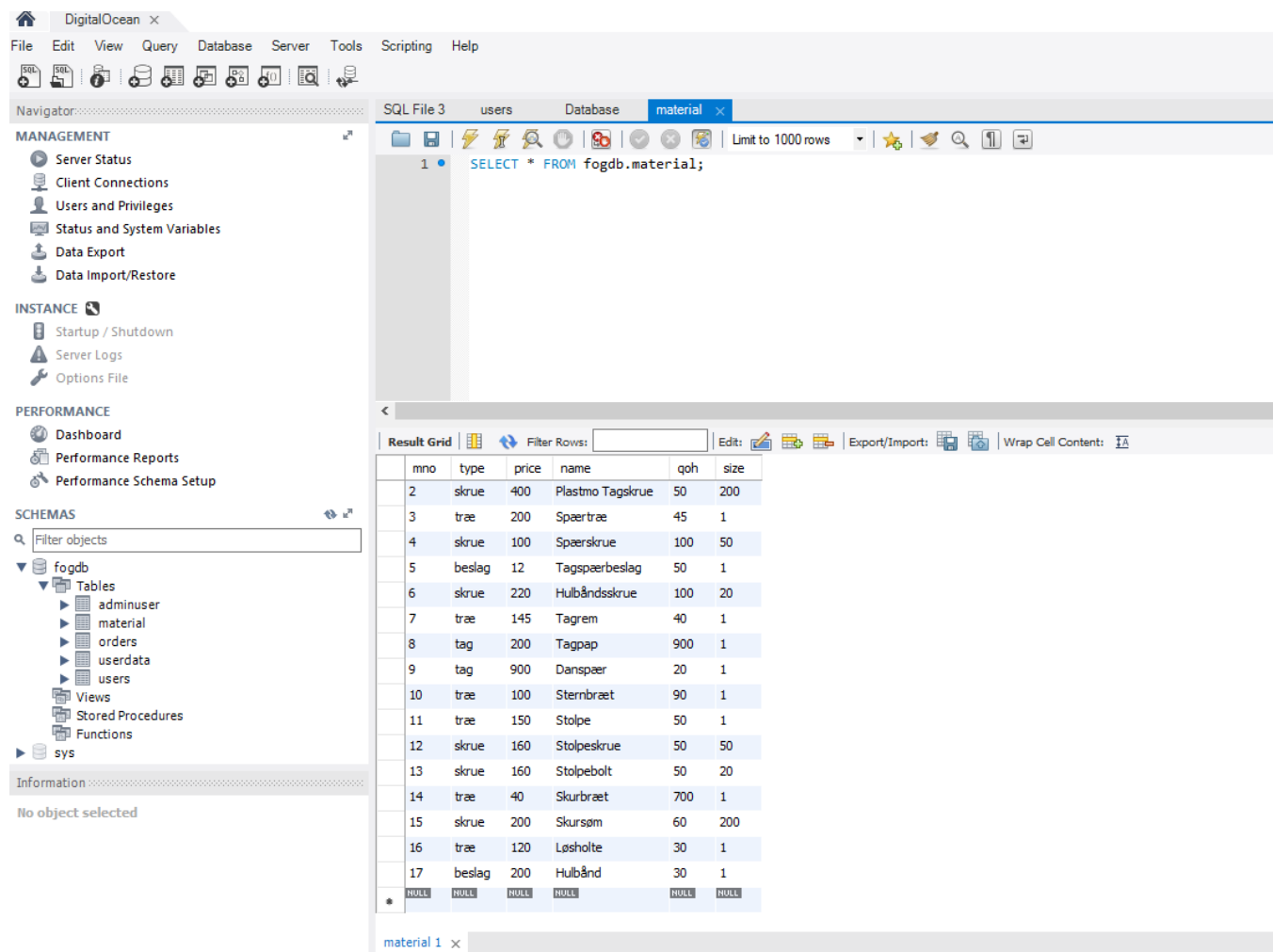
Alle metoder til at behandle SQL statements til vores MySQL database findes i DataAccessObjekt-klassen. Vi benytter os af PreparedStatements for at gøre systemet hurtigere og mere stabilt, og for at sikre os mod SQL-injection angreb – nærmere detaljer forefindes i kapitlet "Sikkerhed".

```
public User getUserByUsername(String username) throws SQLException {
    User user = null;
    PreparedStatement stmt = null;
    try {
        stmt = dbcon.getConnection().prepareStatement("SELECT * FROM users WHERE
uname = ?;");
        stmt.setString(1, username);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            int UID = rs.getInt("uid");
            String usernameRetrieved = rs.getString("uname");
            String passwordRetrieved = rs.getString("password");
            String saltRetrieved = rs.getString("salt");
            String emailRetrieved = rs.getString("email");
            String userString = rs.getString("userstring");
            String carportRetrieved = rs.getString("carport");

            user = new User(UID, usernameRetrieved, passwordRetrieved,
saltRetrieved, emailRetrieved, userString, carportRetrieved);
        }
    } finally {
        try {
            if (stmt != null) {
                stmt.close();
            }
        } catch (Exception e) {
        }
    }
    return user;
}
```

PreparedStatement indsætter de strings der bliver givet med til metoden der hvor spørgsmålstegnene står. Dette er ligegyldigt hvad der står i den string der kommer med, derfor bliver SQL-injection umuliggjort her.

MySQL-Databasen er opbygget specifikt efter vores egne behov. Derfor er den ikke særlig fleksibel. Dog er tabellen med dele til carporten bygget op så både skruer, brædder, søm osv. kan være i samme tabel. dette er gjort ved at definere forskellige dele-id'er som markerer forskellige typer materialer.



The screenshot shows the DigitalOcean MySQL interface. The left sidebar contains navigation menus for MANAGEMENT, INSTANCE, PERFORMANCE, and SCHEMAS. The SCHEMAS section is expanded, showing the 'fogdb' database with tables like 'adminuser', 'material', 'orders', 'userdata', 'users', 'Views', 'Stored Procedures', and 'Functions'. The 'material' table is selected. The main area displays a SQL query: `SELECT * FROM fogdb.material;` and its results in a table grid. The table has columns: mno, type, price, name, qoh, and size. The results list 17 items, including various types of screws (skrue), wood (træ), and other materials (beslag, tag, tagrem, tagpap, danspær, sternbræt, stolpe, stolpebølte, hulbånd, hulbåndsskrue, skursøm, løsholte). The last row shows NULL values for all columns.

mno	type	price	name	qoh	size
2	skrue	400	Plastmo Tagskrue	50	200
3	træ	200	Spærtræ	45	1
4	skrue	100	Spærskrue	100	50
5	beslag	12	Tagspærbeslag	50	1
6	skrue	220	Hulbåndsskrue	100	20
7	træ	145	Tagrem	40	1
8	tag	200	Tagpap	900	1
9	tag	900	Danspær	20	1
10	træ	100	Sternbræt	90	1
11	træ	150	Stolpe	50	1
12	skrue	160	Stolpeskrue	50	50
13	skrue	160	Stolpebolte	50	20
14	træ	40	Skurbræt	700	1
15	skrue	200	Skursøm	60	200
16	træ	120	Løsholte	30	1
17	beslag	200	Hulbånd	30	1
	NULL	NULL	NULL	NULL	NULL

## Sikkerhed

Da vi startede projektet var sikkerhed en stor prioritet for os. Vi har derfor hashed de password som bliver gemt i vores database. Når vi kommunikerer med vores database bruger vi også PreparedStatement.

### Hashing af password

Det er desværre set flere gange at virksomheder får hacket deres database og får den dumpet (lagt ud på internettet til offentlig skue), så derfor har vi snakket om at det er vigtigt at vi får beskyttet password, da mange mennesker desværre bruger det samme password til forskellige tjenester. Vi har så valgt at hashe vores passwords ved SHA-512. Det sker ved at vi genererer en tilfældig 18 cifferet streng, hvilke hedder 'salt'. Salten er genereret tilfældig per bruger.

Vores bruger, 'test1', har også 'test1' som password. Salten som er genereret til denne bruger er den følgende 'F15TDDLMREW2QOM6HC'. Når vi så har hashed vores password bliver det til et 128 karakter password i databasen i forhold til de 5 karakter det er i plain text (Det kaldes det før det bliver hashed). Passwordet ser sådan her ud i databasen 'ffa7348493c146a3826c0514fbff98a10a783b439af6367cb8d3a6436b68c6f8cfec53Daca0c34242741c449c31d6602960bcb91297d356c701db927478f1193'. Ud fra det kan vi se at i tilfælde af at vores database bliver lagt ud på nettet er det besværligt at gætte kodeordet.

Her er koden som hasher vores password, og vi definere så at vi bruger SHA\_512 i MessageDigest.

```
public String get_SHA_512_SecurePassword(String passwordToHash, String salt) throws
UnsupportedEncodingException {
    String generatedPassword = null;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-512");
        md.update(salt.getBytes("UTF-8"));
        byte[] bytes = md.digest(passwordToHash.getBytes("UTF-8"));
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < bytes.length; i++) {
            sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
        }
        generatedPassword = sb.toString();
    } catch (NoSuchAlgorithmException e) {
        System.out.println("ERROR Password 1:");
        e.printStackTrace();
    }
    return generatedPassword;
}
```

## PreparedStatement

Vi bruger PreparedStatement til alt vores SQL kommunikation af en meget simpel grund. Det er sikkert, og det kan hjælpe med at forhindre SQL-injections hvis det bliver brugt rigtigt. Grunden til det er på den måde at user inputs bliver sendt ind på.

1. "INSERT INTO users VALUES (" + ono + ", " + username + ", " + password + ");"
2. "INSERT INTO users VALUES (?, ?, ?);"

Her har vi 2 forskellige SQL sætninger. Den ene er der mulighed for at lave SQL injection på, den anden er det ikke. Grunden til, at sætning 1 er mulig at exploite, er på grund af den måde som den i sig selv er sat op på. Når variablerne bliver sat ind i sætningen, bliver de en del af selve SQL-sætningen når det bliver behandlet i databasen.

I sætning 2 bliver vores variabler indsat på en lidt anden måde, det sker ved `setString` / `setInt` / `setDouble` metoder. Der bliver dataen sendt ind til databasen sammen med SQL sætningen og vores database kan på den måde vide at det bliver sendt ind som parameter og ikke en del af SQL sætningen. På den måde ved databasen at hvis der står " Robert'); DROP TABLE users;-- ? " at den ikke skal tage det som en del af statementet, men derimod som et parameter, og det bliver derfor tilføjet som sætningen står i databasen, i stedet for at ødelægge hele databasen.

## Features

Af Kristian

### 2D tegning

Det første feature vores løsning tilbyder er en 2D tegning af den carport kunden har bestilt. Der er intet arbejde i det for Fogs ansatte, da programmet automatisk skalerer antal ben og placering af ben etc. alt efter carportens størrelse. Både 2D og 3D tegningen kan skales i realtime via en slider i et lille UI på den side Fogs kunder ser.

### 3D tegning

Ud over en 2D tegning tilbyder vores løsning også en 3D tegning. En fordel ved en 3D tegning er at kunderne kan få lov at se deres carport for sig før de køber, hvilket vil gøre valget nemmere for mange. En anden fordel er, at man vil kunne videreudvikle systemet omkring 3D tegningen yderligere og tilføje flere features til den senere i udviklingsforløbet.

### Stykliste

Før i tiden skulle Fogs ansatte selv smide en stykliste sammen for produktet mens de lavede den .pdf kunden skulle modtage. I vores løsning bliver styklisten autogenereret baseret på carportens størrelse og de valg kunden har foretaget i forbindelse med købet (med/uden skur, fladt tag eller skråt tag). Når 3D modellen udvides til at kunne vise forskellige typer af træ/tagdækning vil dette kunne sendes videre til styklisten og denne vil blive opdateret derefter.

### Adminpanel

Vores adminpanel giver Fogs ansatte nem adgang til både nuværende og tidligere ordrer, hvilket mindsker den tid der skal bruges på tastearbejde og på at fremsøge en ordre ud fra ordrenummer. En feature ved admin panelet er muligheden for nemt at kunne tilpasse de enkelte materialer i Fogs database med hensyn til pris. Det er også nemt for Fogs ansatte at ændre status på en kundes ordre (ny/påbegyndt/færdig etc.).



## Testing

Af Kasper

I softwareudvikling er det vigtigt at sikre sig, at det der bliver produceret også fungerer som det skal. Firmaer benytter software til både behandling af persondata, lønstyring og behandling af anden fortrolig data, og derfor er det vigtigt, at alt er sikkert og fungerer efter hensigten når det bliver sendt live.

### Manuel Testing

Vi har alle i gruppen individuelt gået igennem alle funktioner i programmet, og sikret os at det hele fungerer. Dette har inkluderet at prøve at sende forkert data i input-felter, se sider, hvor man skal være logget ind som administrator, uden at være det og tjekke efter ødelagte links. Vi har derefter fikset alle fejl der blev fundet, for at levere et bedre produkt.

### Automatiseret Testing (JUnit)

Vi har Unit Tests på de mest essentielle metoder i klassen DataAccessObjekt, da dette er den klasse der foretager de fleste vigtige metodekald.

```
public class DataAccessObjektTest {  
  
    DataAccessObjekt instance = null;  
    Random r = new Random();  
  
    public DataAccessObjektTest() {  
    }  
  
    @BeforeClass  
    public static void setUpClass() {  
    }  
  
    @AfterClass  
    public static void tearDownClass() {  
    }  
  
    @Before  
    public void setUp() throws Exception {  
        instance = new DataAccessObjektImpl();  
    }  
  
    @After  
    public void tearDown() {  
    }  
}
```

```
/**
 * Test of getUserByUsername method, of class DataAccessObjektImpl.
 */
@Test
public void testGetUserByUsername() throws Exception {
    String username = "test1";
    String expectedResult = "test1@test1.com";
    User u = instance.getUserByUsername(username);
    String result = u.getEmail();
    assertEquals(expResult, result);
}
```

Her ses et eksempel på en test af DataAccessObjekt. Der oprettes en ny version af objektet hver gang en metode kaldes, for at sikre at en tidligere test ikke har ændret noget i objektet.

Testen verificerer at man kan logge ind som en bruger. Testen er dog ikke optimal i det den bruger den faktiske databaseforbindelse til at tjekke på en faktisk bruger, der er lagt i databasen til testbehov. Det optimale vil være at skrive en mock-database connection med for eksempel Mockito. Dette vil give den fordel at man kun tester om metoden gør det, der er meningen, ved at mocke de svar der skal sendes til metoden. Dette er kernen af en Unit-test – at teste en specifik metode, for at sikre at de enkelte dele i projektet fungerer som de skal.

Fordelen ved at teste med den faktisk database koblet på projektet er, at man samtidig metoden, også får testet om ens metode fungerer som planlagt. Dog bevæger dette sig mere over i det område det kaldes Integration Tests i stedet for Units Tests.

Det er dog vigtigt at have både Unit- og Integration tests, da begge er vitale for at se om ens projekt fungerer som planlagt.

En anden fremtidsplan vi havde er at implementere Selenium-testing. Dette gør det muligt at teste hele systemet fra brugerfladens side, i stedet for de rene metoder. På denne måde behøver vi som gruppe ikke at sidde og teste hver eneste knap igennem for at sikre os, at de fungerer. Dette vil Selenium-testen prøve, og kan derfor også hjælpe med at teste vores JavaScript, der ellers ikke er unit-testet.

## Fremtidige implementeringer

Vi har gjort os en masse overvejelser over hvad vi gerne ville havde med i programmet. Nogle af de ting som vi gerne ville havde haft som vi desværre ikke nåede / havde erfaringen til det.

### Autogenereret PDF fil

Vi kunne godt havde tænkt os at få autogenereret den PDF fil som fog skal udlevere til kunden med bygnings instrukser. Dette ville gøre det både nemmere og lækre for kunden at arbejde med da de ville få en tegning med deres egne mål på. Dette ville også gøre det nemmere for fog da en medarbejder så ikke mere skulle bruge tid på at sidde og plukke for diverse andre manualer for at få lavet en så god til kunden som muligt.

### Gamle carporte design + ordrer

På det givne tidspunkt er det ikke muligt at se mere end den sidste gemte carport. Dette kunne vi godt tænke os at udvide til man for eksempel kunne se ens 50 sidste byggede carporte, for at give det bedre overblik over hvad man har lavet, og på den måde vælge den man er mest tilfreds med.

Det er desværre ikke muligt for kunden at se de ordrer, de har placeret, og hvor langt de er i processen med at få dem behandlet. Dog har vi valgt at det ikke var den vigtigste feature, da fog ville være i telefonisk kontakt eller kontakt med kunden over mail.

### Bedre admin panel

Det admin panel som vi har på det givne tidspunkt, fungerer til de fleste basale ting. Det kunne dog godt ønskes at havde flere muligheder som for eksempel at søge information om en bruger ud fra deres kunde nummer som en fog medarbejder for udleveret når de tager en ordre. Det kunne også tænkes at udvide den del af siden, hvor medarbejderne ser de forskellige ordrer, så der også står et telefon nummer /email på kunden, og eventuelt en rubrik hvor medarbejderen kan tage nogen noter efter samtale med kunden. Så kan man på den måde bedst kunne finde ud af, hvordan det hele skulle hænge sammen efter diverse samtaler man har haft med kunden.

## Rewrite dele af koden

Nu når vi er kommet så langt i projektet har vi siddet, og kigget på det, og har fundet ud af at der er flere steder hvor vi måske kunne koge 2 filer ned 1, eller skrive det mere effektivt og bedre. Så en af vores næste store planer ville være at bruge tid på ville være at gå hele kodebasen igennem og forbedre den, og finde ud af hvad der er 'ligegyldigt', og hvad der vil give mening at putte over i en anden klasse. Vi har også gjort os flere tanker her til sidst på at lave en front controller til at håndtere alle request fra JSP, og så dele det ud til det en af flere 'hovedcontrollers', som så står for at sende det videre ud.

Vi skal også have tjekket efter om der er noget kode som ikke længere skulle bruges, eller om der faktisk er noget af det som er duplicate, så det bare skal fjernes.

Det samme gælder også vores JavaScript, da der muligvis også er noget kode som kunne skrives om og gøres på en bedre måde.

## Kendte Fejl

Af Tjalfe

Programmet er ikke 100% færdiggjort så der er et par fejl i vores kode grundet i vi ikke havde tiden til at færdiggøre disse ting.

### JavaScript (2D/3D):

Det er muligt at sætte målene over maksimum, under minimum og udenfor de steps der er sat i GUI'en.

Disse ulovlige mål kan sendes ind til Java og gemmes men eftersom hver ordre kommer forbi en Fog medarbejder må vi på dette tidspunkt forvente at de validere målene inden ordren kommer videre.

For at genskabe denne fejl:

- Gå til siden /Get3D
- Ændre i json String (som er i et hidden input) kan gøres direkte på html eller via JavaScript commands i console
- Åbne browserens console og køre getJson()
- Trykke knappen i GUI'en "Gem Carport"

Konsekvensen:

- Der kommer korrupt data ind til Java og databasen som ikke var forventet

Forslag til løsninger:

- Validering af målene i Java lige når det kommer ind, da JavaScript kan manipuleres eller undgås ved at bruge browser-console direkte

### Admin funktioner:

Det er muligt at ændre prisen på materialerne og ordres status uden at være admin. Hvis man kender til dens url opsætning. Man kan dog ikke komme ind på admin siden og få nogen visuel bekræftelse på hvad man har gjort, så dette vil kræve et stort kendskab til opbygningen af programmet.

Grunden til dette er at vi i dette tilfælde har flyttet valideringen af "useren" ind i de pågældende .jsp sider og redirecter men JSTL. Vi ser nu det ikke var den bedste idé men havde ikke tiden til at fixe det.

Eksempel pris ændring:

For at genskabe denne fejl:

- <http://localhost:8080/ProjektFog/userServlet?updatePrice=&mno=1&price=200>

Konsekvensen:

- Ikke administrators kan ændre i priser og ordre status

Forslag til løsninger:

- Userservlet burde have et tjek på om det er en admin der er logget ind før det udfører opgaver

## Konklusion

Af Kasper

Vi er udmærket tilfredse med vores projekt ved aflevering. Det har de fleste af de features implementeret via de userstories, vi har fremstillet. Gruppearbejdet har også fungeret fint, og vi har fået fremstillet et projekt, som vi kan se tilføje værdi til Johannes Fog som virksomhed, og som opfylder vores egne krav der blev stillet i starten. Det har ikke været et perfekt stykke arbejde hele vejen igennem, hverken projektmæssigt eller rapportmæssigt, men generelt er vi tilfredse med det projekt vi har fået fremstillet.

## Installation

Af Kasper

Programmet er udgivet og kørende på en DigitalOcean webserver på adressen <http://46.101.97.181:8080/ProjektFog>

Alternativt kan projektet hentes på <https://github.com/KasperOnFire/FogEksamensProjekt> og selv compiles. Projektet åbnes i undermappen `./ProjektFog/` i NetBeans eller lignende. Tilføj derefter MySQL JDBC-Connectoren til projektet, hvis ikke dette er sket automatisk.

I tilfælde af at projektet stadig ikke fungerer, skal der tilføjes `org.json` biblioteket, som findes i `./ProjektFog/lib/json/` mappen.

Derefter kan projektet køres og testes.

Der findes 2 test-brugere – en "kunde" og en "admin".

Logininformationerne er som følger:

Kunde:

Username: test1

Password: test1

Admin:

Username: admin

Password: admin

## Dokumentation

Af Kasper

Generelt kan denne rapport ses som en dokumentation af projektet. Derudover er der javadoc – det er live på hjemmesiden <http://breindal.me/FogEksamensProjekt/>



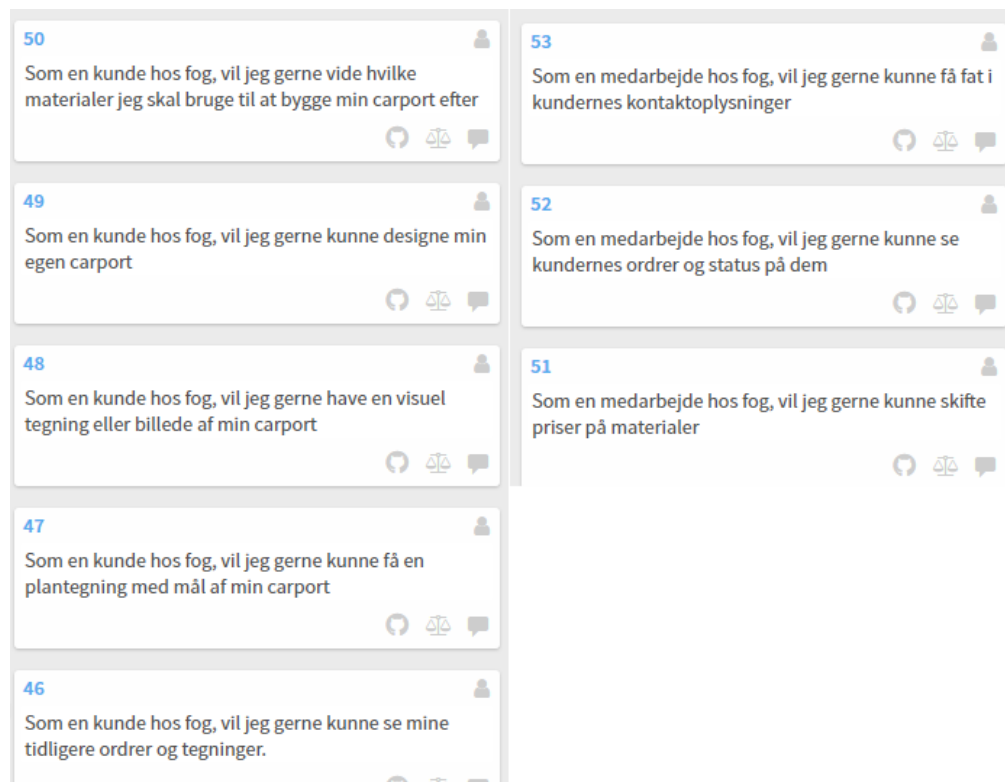
## Proces Rapport

### Backlog

Af Kasper

I starten af projektet fandt vi på nogle userstories, vi mente der ville blive nødvendige for projektet. Det viste sig hen ad vejen at vi ikke havde ramt helt plet, men dog gav det os et solidt grundlag for at starte projektet. Dog blev vi nødt til hen ad vejen at lave mindre ændringer, og tilføje enkelte userstories til vores backlog.

I forhold til tidsallokering tog vi noget oftere fejl. Nogle userstories og tasks tog noget længere tid end vi regnede med – andre gange gik det den modsatte vej.



### Tasks

Af Kasper

Vi besluttede ikke at koble de enkelte tasks i programmet specifik sammen med userstories. En del tasks kan direkte forbindes med den userstory den er relevant for, men mange tasks er opgaver, der har vist sig nødvendige for at projektet samlet kunne fungere. Dette inkluderede især tasks relateret til databaseopbygning og brugergrænsefladedesign. Begge dele er ting som er absolut nødvendige før

projektet kan køre – men ingen af dem kan siges at være noget brugeren selv lægger mærke til, eller syntes der skal tilføjes.

## Tilbageblik

Af Kasper

### 1. Sprint

Dette var første gang vi 4 arbejdede sammen som gruppe. David, Tjalfe og jeg (Kasper) har arbejdet sammen i flere projekter i løbet af året. Efter at have sat os sammen og lavet de første userstories viste det sig at være udfordrende for os at lave tidsplaner for hvad vi kunne nå, da 2D og 3D tegning ikke er noget vi har rørt før dette projekt. Dette resulterede i at vi i det første sprint kom længere end vi troede, og havde et basalt program kørende, der endda kunne lave en 2D-plantegning til kunden. Derfor kunne vi senere fokusere mere på detaljer i de andre features.

#### *Tech Review med Alpha og TM*

Til dette møde snakkede vi om starten af vores projekt, og der blev lagt væsentligt fokus på god kodestruktur og at virkelig følge scrum-metoden til mindste detalje. Dette kom lidt bag på os, da vi havde forventet mere fokus på selve koden, sådan at vi kunne forbedre os her. Dog fungere det fint til at give os et boost i den rigtige retning. Der blev også fokuseret på vores modeller og diagrammer, hvilket var noget vi især manglede at arbejde med. Dette blev derfor hurtigt en af de ting vi skulle arbejde med i næste sprint.

#### *Productowner meeting med PAB*

Her var fokus på userstories og scrum. Der var nogle områder, hvorpå vores implementering af scrum ikke var god nok, og dette fik vi diskuteret så arbejdsprocessen kunne køre bedre i fremtiden. Derudover var der nogle af userstories der skulle revideres, før productowner var tilfreds med dette. Desuden havde vi vores demomodel at vise frem, som productowner var glad for.

#### *Sprint retrospektiv*

Alt i alt var vi selv udmærket tilfredse med første sprint. Vi var fint med teknisk og kodemæssigt, og skulle derfor arbejde mere med diagrammer og scrum som teknik, hvilket vi også havde forventet. Samarbejds-mæssigt var vi meget tilfredse, da alt arbejde skred fremad i fint tempo, og alle havde cirka samme forventninger af hinanden.

### 2. Sprint

#### *Tech Review med TOG*

Gruppen kunne til dette review fremvise første revision af vores 3D-render af en carport. Dette blev mødt med positivitet, indtil vi afslørede at det er skrevet i JavaScript. Dog var det ikke noget der blev talt mere om, da negativiteten vidst mere skyldtes personlige holdninger end egentlig aktualitet og

nytte.

Der blev desuden diskuteret acceptkriterier, da vi i gruppen havde været alt for løse på det punkt. I stedet for i forvejen at bestemme hvad der definerer de enkelte userstories som færdige, havde vi arbejdet ud fra en fælles opfattelse af, hvad målet var. Dette fungerede fint i gruppen, men det manglede at komme på tekst. Dette er jo vigtigt i scrum processen, da ideen med dette er, at det ikke kun er udviklerne der ved hvornår noget er færdigt – men at productowner og måske endda brugere også kan se, hvor langt de enkelte dele er.

### *Sprint retrospektiv*

I dette sprint er vores projekt skredet udmærket fremad. Dog har det til tider været svært at fokusere præcis på at samle de enkelte dele, da forskellige folk i gruppen har arbejdet på forskellige grene af projektet – 2D, 3D, materialeliste. Derfor havde vi problemer med at få samlet det hele, hvilket ledte til en mindre agil form for udvikling end planlagt. Dog var opholdende ikke store, og vi havde nu også en fungerende 3D modellering af carportene med i projektet.

## 3. Sprint

### *Tech Review med Alpha og TM*

Ved dette møde havde vi netop modtaget et codereview pr. mail fra Alpha, som vi var ved at rette igennem og kigge på. Dette omhandlede primært "rengøring" af koden – ubrugte variabler, tomme metoder og lignende. Vi snakkede også om testing frameworks, og hvordan man kan forbedre både tests og projektet generelt ved at benytte mocks og bygge dependency injection ind i projektet.

### *Productowner meeting med PAB*

I det sidste møde med productowner havde vi en funktionel version af programmet kørende, som så senere kan udbygges med flere features. Dette var productowner fint tilfreds med, da dette jo er den egentlige mål helt fra starten af, når man arbejder i scrum. Vi snakkede også om hvad man eventuelt kunne tilføje senere, der kunne tilføje ekstra værdi for virksomheden i programmet – så som muligheder for marketing og at sikre kundeloyalitet. Desuden mente gruppen, at vi manglede viden om hvad Fog gerne ville have inkluderet i programmet for medarbejdernes side, da fokus indtil videre primært havde handlet om kunderne og deres behov.

### *Sprint retrospektiv*

Det sidste sprint var et mindre effektivt sprint for os. Programmeringen kom til at gå op i at finpudse meget små detaljer, selvom der egentlig stadig var nogle små- og mellemstørrelse features tilbage der enten kunne tilføjes eller justeres til, før de var fuldt funktionelle. Dog var det ikke et "spildt" sprint – der blev fixet mange bugs og fejl, og nogle segmenter blev refaktorert til bedre at passe ind i datastrukturen, være et mere fornuftigt placeret i koden.

## Gruppesarbejde

Af Kasper

Vores gruppe startede med at David, Tjalfe og jeg (Kasper) satte os sammen, da vi i de seneste par måneder har lavet mange gruppeprojekter sammen som alle er gået godt, og derfor naturligvis ville fortsætte den gode stime. Kristian blev hurtigt tilføjet til gruppen efter at TM mente, at han mere var på vores niveau end på niveau med den gruppe han var i.

Samarbejdet har som sådan fungeret fint i løbet af dette projekt. Dog har der i perioder, især mod vejs ende, været situationer hvor det har været svært at etablere kontakt til hinanden, og der har generelt gennem projektet været flere situationer med miskommunikation end vi brød os om.

## Rapportskrivning

Rapporten vidste sig at give flere problemer end vi havde troet. For det første kom vi sent i gang, idet nogle af medlemmerne brugte mere tid på bugfixes og justering end på rapport skrivning. Dette resulterede i en stresset periode, hvor der flere gange opstod tvivl om hvem der skrev hvilke dele af rapporten, om de var begyndt eller færdige, indtil kort tid før aflevering. Dette kan ikke siges at være optimalt. Rapporten er dog blevet færdiggjort, og hver man har taget sit eget ansvar.

## Arbejdsmetoder

Af Kasper

En del af opgaven var at følge Scrum og Agile principper til at få fremstillet projektet. Gruppen tog dette nogenlunde til sig, dog krævede det en tilvænningsperiode i starten af samarbejdet før alle var med på hvordan dette skulle udføres.

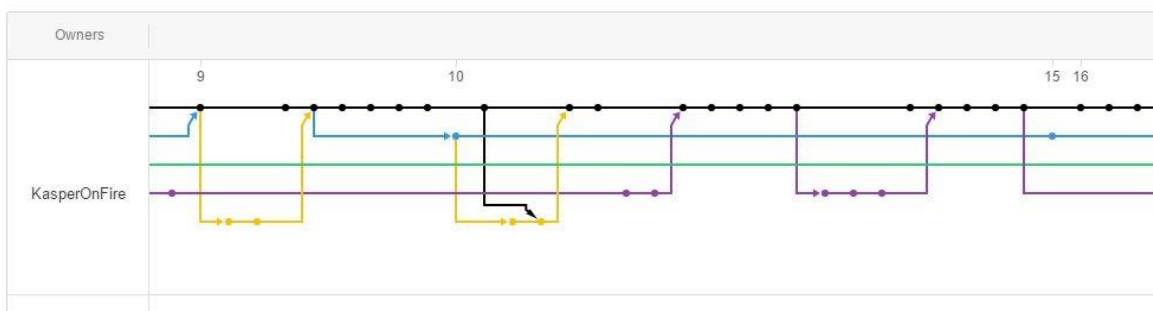
Vi har været 4 mand i gruppen, og selvom Scrum bygger på grupper af 6-8 mand, har dette har været en meget passende størrelse i forhold til uddelegering af opgaver, og så er man ikke for mange til møderne, som derfor bliver overkommelig længde.

Derudover gør det mindre antal gruppemedlemmer også det, at det ofte er nemmere at komme til enighed i diskussioner, men at alles stemme bliver hørt, da hver mand jo tæller for 25%.

Vi har i vores gruppe aftalt at vi de fleste dage mødtes på skolen for at arbejde, for at sikre os at alle får foretaget det de skal, og det på den måde bliver nemmere at hjælpe hinanden med problemer man måtte møde undervejs.

## Værktøjer

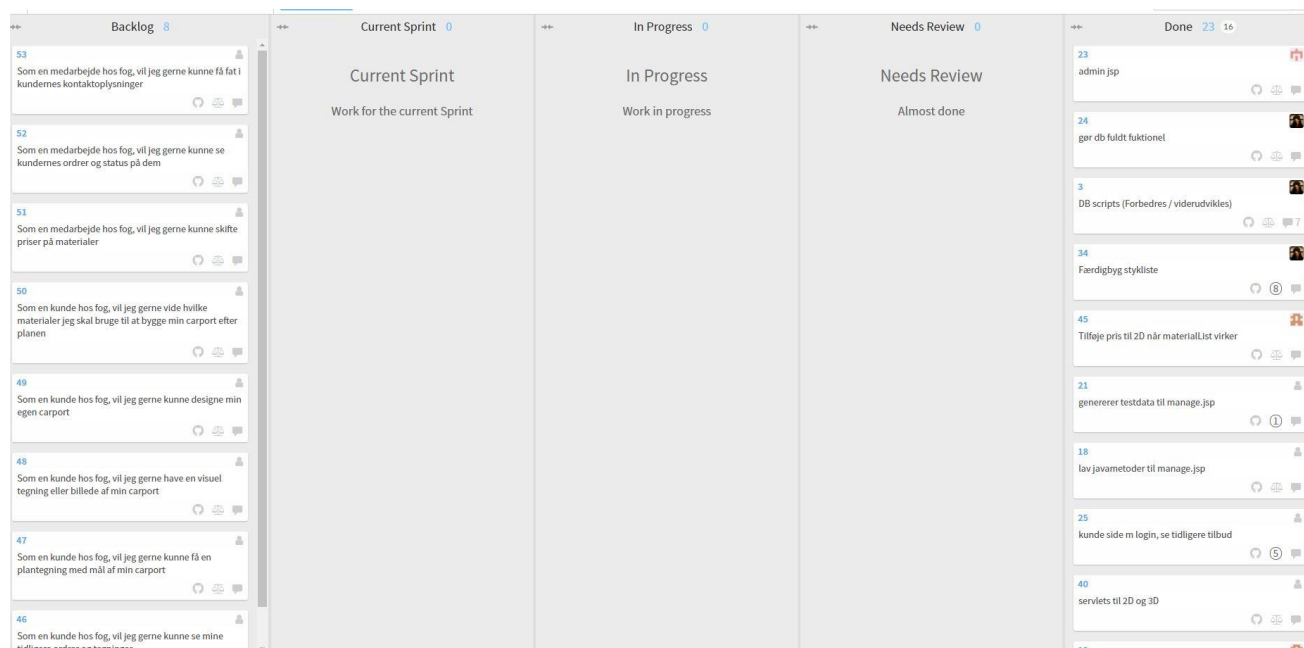
Til versionsstyring har vi benyttet os af Git og GitHub. Dette har fungeret til al tilfredsstillelse, og vi har sjældent haft merge conflicts. Gruppen har sigtet efter at arbejde efter feature-branching, altså at hver ny tilføjelse laves i sin egen branch, og derefter merges ind i master til det fungerende projekt. Dette kan ses på figuren her.



Brugen af git har givet os god træning i versionsstyring, som i nogle medlemmer af gruppens optik er noget nær det vigtigste at holde styr på praktisk i et projekt af større størrelse. Da dette også bruges af store firmaer til styring af softwareprojekter, bliver dette også meget nyttigt for os i fremtiden.

Deruder har vi brugt waffle.io til styring af tasks og userstories. Dette har fungeret udmærket, og kan hjælpe til med at give et overblik over hvem der er ved at foretage sig hvilke tasks. Vi har brugt det til

at sortere tasks efter prioritet og tid. Dette har været vigtigt da der i gruppen ofte har været diskussioner om hvilke ting der skulle implementeres hvornår.



## Scrum

Vi har arbejdet efter scrum i dette projekt. Dette har inkluderet daily standup meetings, sprint planning, sprint reviews og sprint retrospective meetings. Dette projekt har været god øvelse i at arbejde efter scrum's principper. Især daily standups har været effektive, da det hurtigt giver et overblik over hvad gruppen arbejdede med dagen før, og hvad der skal arbejdes med denne dag. Vi kan ikke påstå at være mestre i denne form for arbejde endnu, men projektet har givet os en masse erfaring, som alle i gruppen føler vil komme dem til gode i fremtiden.

## Agile Development

Agile development kom nemt til gruppemedlemmerne som en teknik, da alle fokuserede på at levere funktionel kode og at være produktive individuelt. Enkelte gange har problemer krævet at 2 satte sig sammen og par-programmerede for at løse dette. Vi ser mange fordele i denne form for arbejde, da det giver ekstra flow til softwareudviklingsprocessen.

## Konklusion

Af Kasper

Projektet har været en lang proces, og har ofte krævet meget arbejde af os som gruppe. Både med lange dage i skolen, men også lange aftener derhjemme. Det har hjulpet os i mange aspekter, og har generelt udviklet os som softwareudviklere generelt. At arbejde med scrum har været en lærerig proces, som ikke altid har været nem, men den har udviklet os, og hjulpet os til bedre at forstå hvordan virksomheder arbejder med software projekter, så vi føler os mere forberedt på vores fremtid.