Project report 1st year exam – Computer Science

Hand in deadline: _____

Group no: _____

Group members:

First name(s), last name	Signature
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Received: _____ _____ _____

(Name) (Date) (Time)

Indholdsfortegnelse

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen
Stefan Duro
Uffe Nedergaard

Forord - Stefan

Firmaet Hovedstadens stillads udlejning skal bruge et nyt IT system til at styre ordre, materialer og personale. Hovedstadens Stillads udlejning har en idé om at de mister ordre, fordi de ikke altid er i stand til at vurdere om de har materiale eller personale nok, til en given opgave. Det nye system skal være i stand til at benytte to lagere, samt være i stand til at tjekke om de kan levere i en given periode. Denne rapport vil vise design valgene og deres begrundelse.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen
Stefan Duro
Uffe Nedergaard

Forretningsanalyse

S.W.O.T – Hele gruppen

Styrker:

- LS er datterselskab til entreprenørfirma.
- CS har over 30 års erfaring.
- Har en fast stor kundebase.
- Kan trække på egne ressourcer.

Svagheder:

- Knaphed på monteringsressourcer.
- Kan ikke give kunder klar besked om hvornår arbejdet kan påbegyndes pga. planlægning.
- Salgspersonalet kan ikke få adgang til salgsinformation.
- To lagerbygninger skaber problemer med logistikken.

Muligheder:

- LS er datterselskab til entreprenørfirma.
- CS har et godt ry blandt deres kunder.

Trusler:

- Fare for at faste kunder falder fra under fusion.
- Forsinket byggeri.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen
Stefan Duro
Uffe Nedergaard

Vision og mål for IT-system – Hele gruppen

Vision

Visionen er at skabe et IT-system som både forbedre og forsimpler hvordan hverdagen i Hovedstadens Stillads Udlejning kommer til at foregå. Dette vil ske gennem både et simplere IT-system, som vil gøre det nemmere at oplære nye personer som skulle blive ansat hen ad vejen. IT-systemet vil gøre det nemmere at køre deres forretning til dagligt, med et IT-system som alle kan forstå hvordan virker og hvordan det skal bruges.

Mål

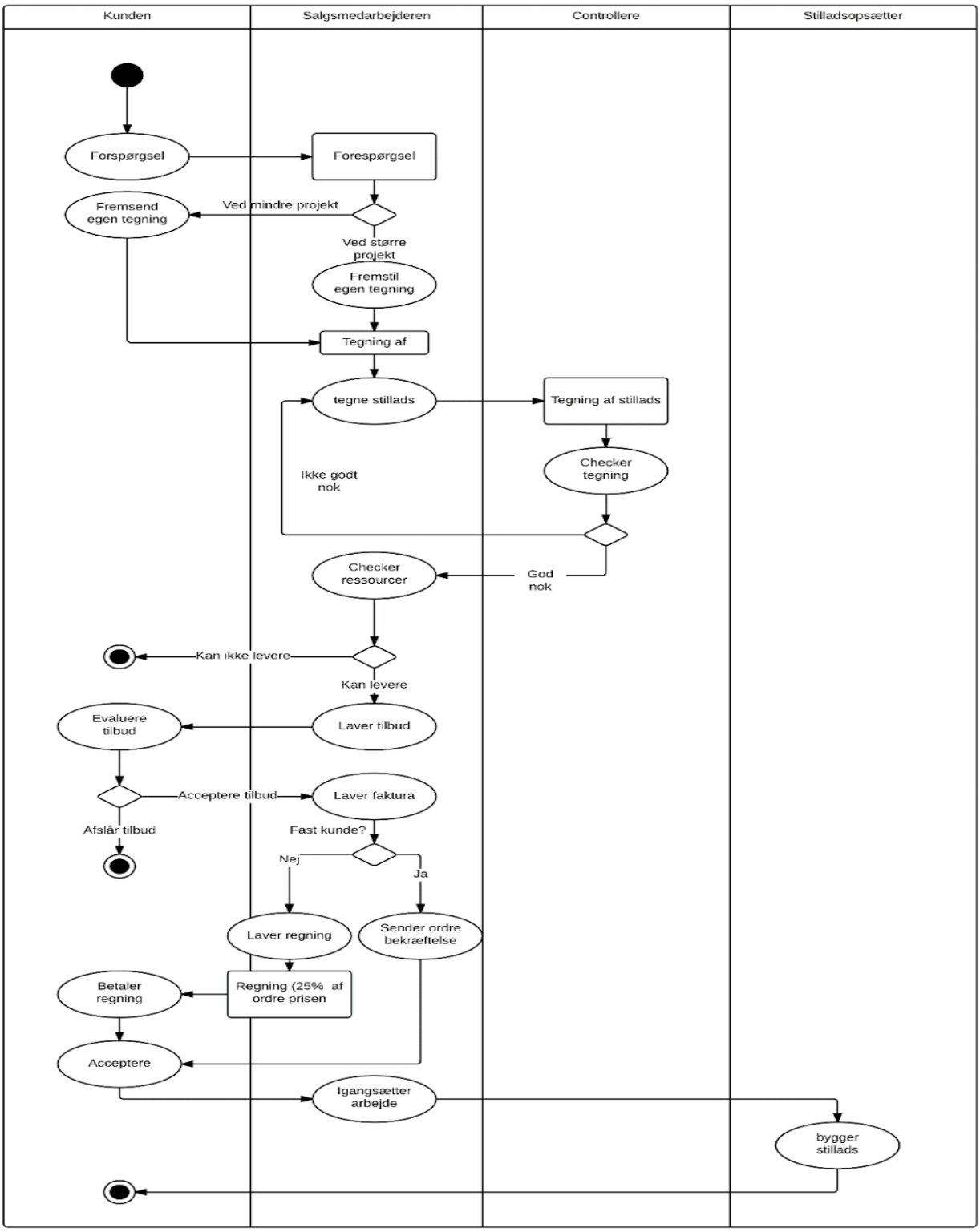
Målet er at de krav der er blevet udsendt kan blive overholdt i det nye IT-system, fremlagt fra product-owners side. Efter implementeringen af dette system vil deres frafaldne ordre mindskes og deres produktivitet stige, samt deres profitter.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen

Stefan Duro
Uffe Nedergaard

Analysemodel af arbejdsgange

Sporbarhedsmodel (As Is) – Hele gruppen



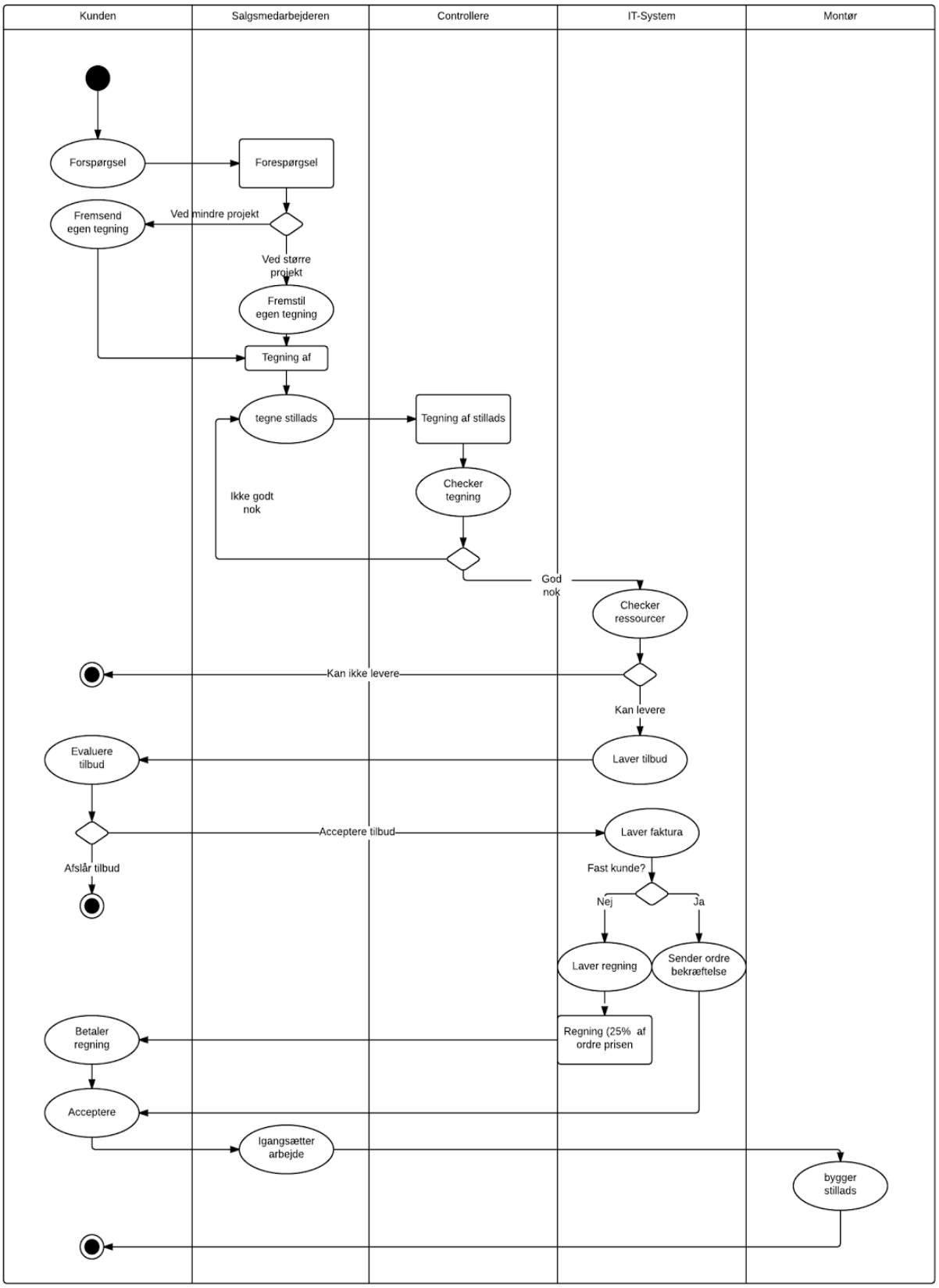
Figur 1 - Sporbarhedsmodel As Is

Denne sporbarheds model (figur 1) viser hvordan, at en ordre bliver lavet i Hovedstadens Stillads Udlejning(HSU), uden et nyt IT-system. Det er en langsom metode, da den går igennem en salgsmedarbejder, der manuelt skal lave mange ting, i stedet for en mere strømlinet metode, som det nye IT-system vil kunne give firmaet. Dette vil kunne gøre at medarbejderen hurtigt og effektivt kan oprette en ordre. Det kommer til at være en bedre og mere effektiv løsning i forhold til det regneark som der tidligere er blevet brugt.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen

Stefan Duro
Uffe Nedergaard

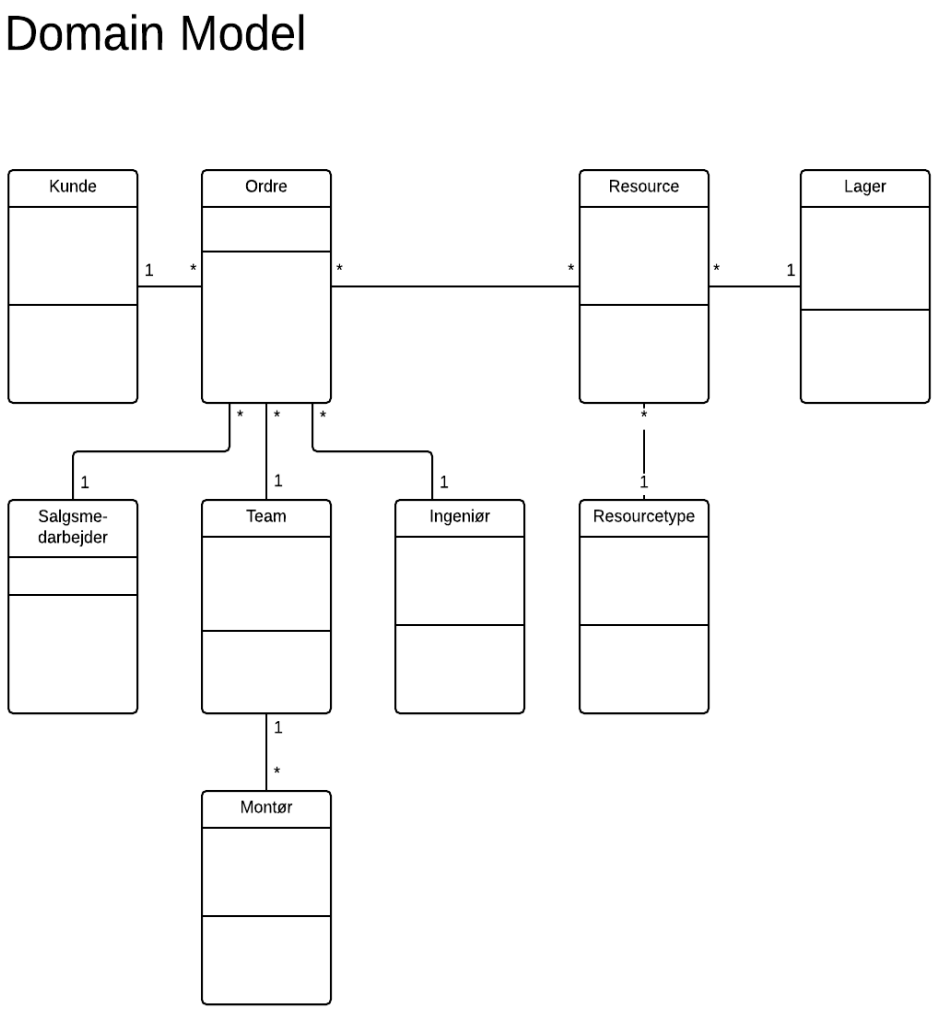
Sporbarhedsmodel (To Be) – Hele gruppen



Figur 2 - Sporbarhedsmodel TO BE

Denne sporbarhedsmodel (figur 2) viser hvordan det nye system der vil overtage mange af de processer som salgsmedarbejderen tidligere har skulle gøre. Dette vil derfor give salgsmedarbejderen mere tid til at gå igennem andre ting, og oprette flere ordre, hvilket vil betyde at der overordnet vil være færre tabte ordre.

Domænemodel – Hele gruppe



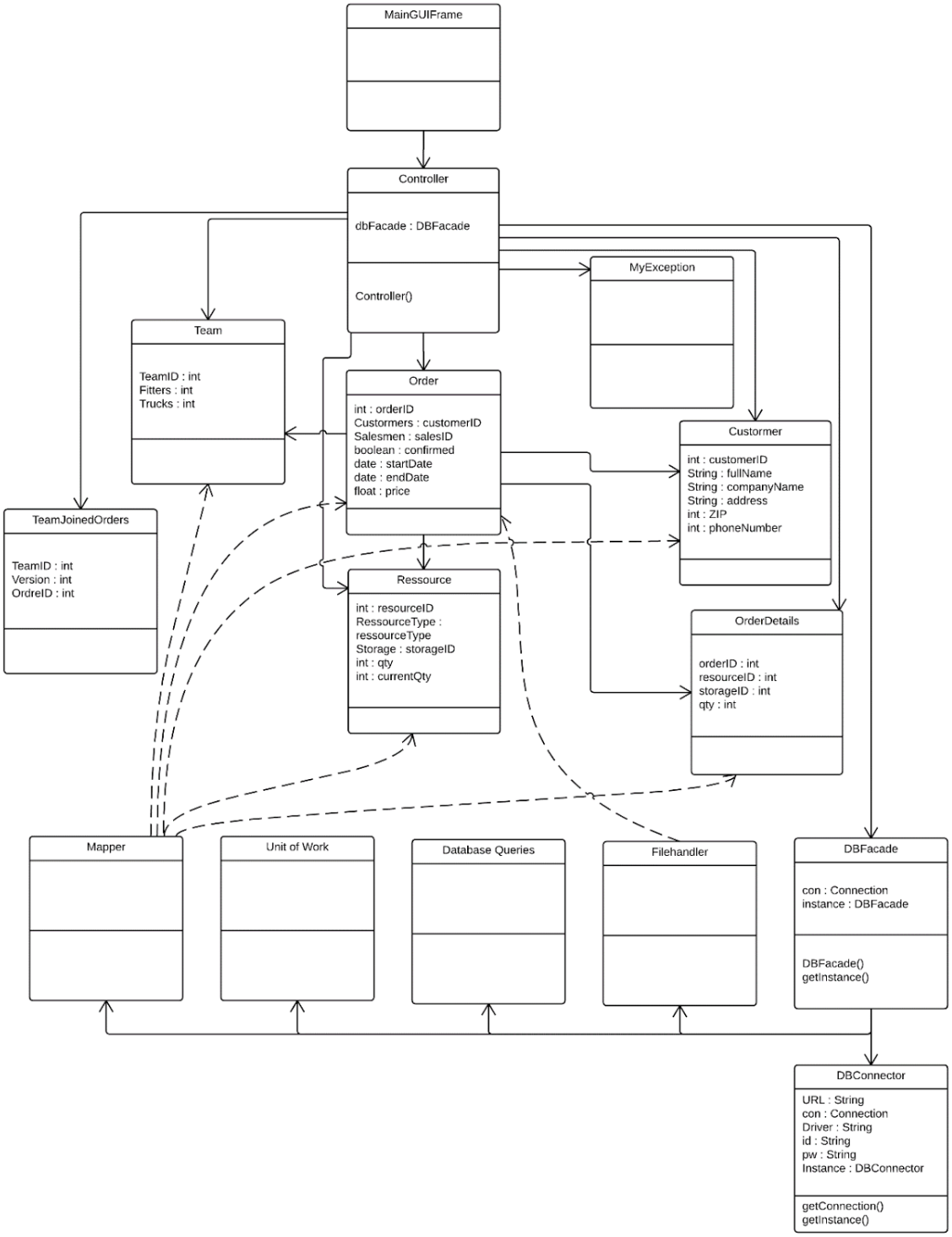
Figur 3 - Domænemodel

Denne domæne model (figur 3) viser at ordren er den centrale del af vores projekt. Det er det basis programmet blev udviklet under.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen

Stefan Duro
Uffe Nedergaard

Klassediagram – Uffe

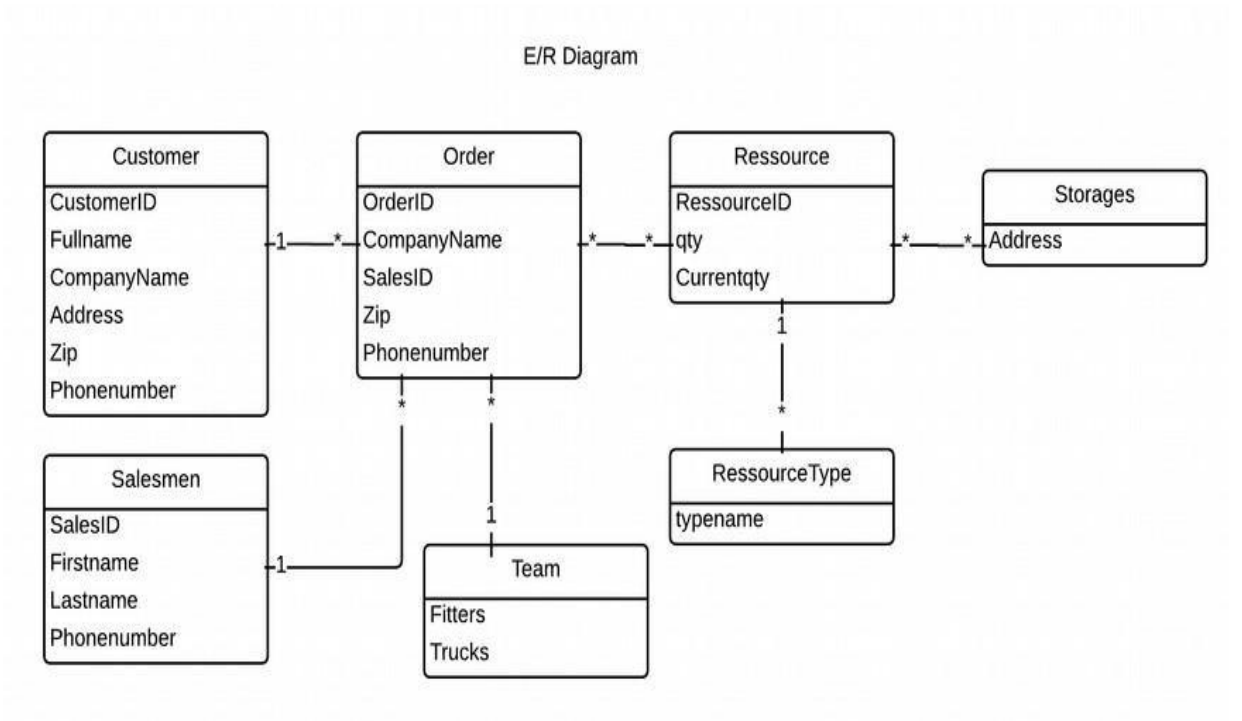


Figur 4 – Klassediagram

Øverst ses en *GUI* kasse, som inkluderer alle *GUI* klasser. *GUI* henter en instans af *Controller* og lægger det ind i et *Controller* objekt. *GUI* henter en instans i stedet for selv at oprette et objekt, da *Controller* bruger singleton mønstret. "Presentation" laget, indeholder kun *GUI* klasser. Det næste lag af klasser er "Domain" laget. "Domain" laget består af 8 klasser inklusivt *Controller* klassen. *Controller* Klassen opretter et objekt af de 7 andre klasser i "Domain" laget. Det sidste lag, "DataSource", består af "Mappe" klasser, og andre klasser der har med databasen at gøre. Det er illustreret ved at alle "DataSource" klasserne ligger i bunden af diagrammet. "Mappe" klassen er kun illustreret med en enkelt kasse, men den består i virkeligheden af 4 forskellige "Mappe" klasser.

E/R Diagram - Stefan

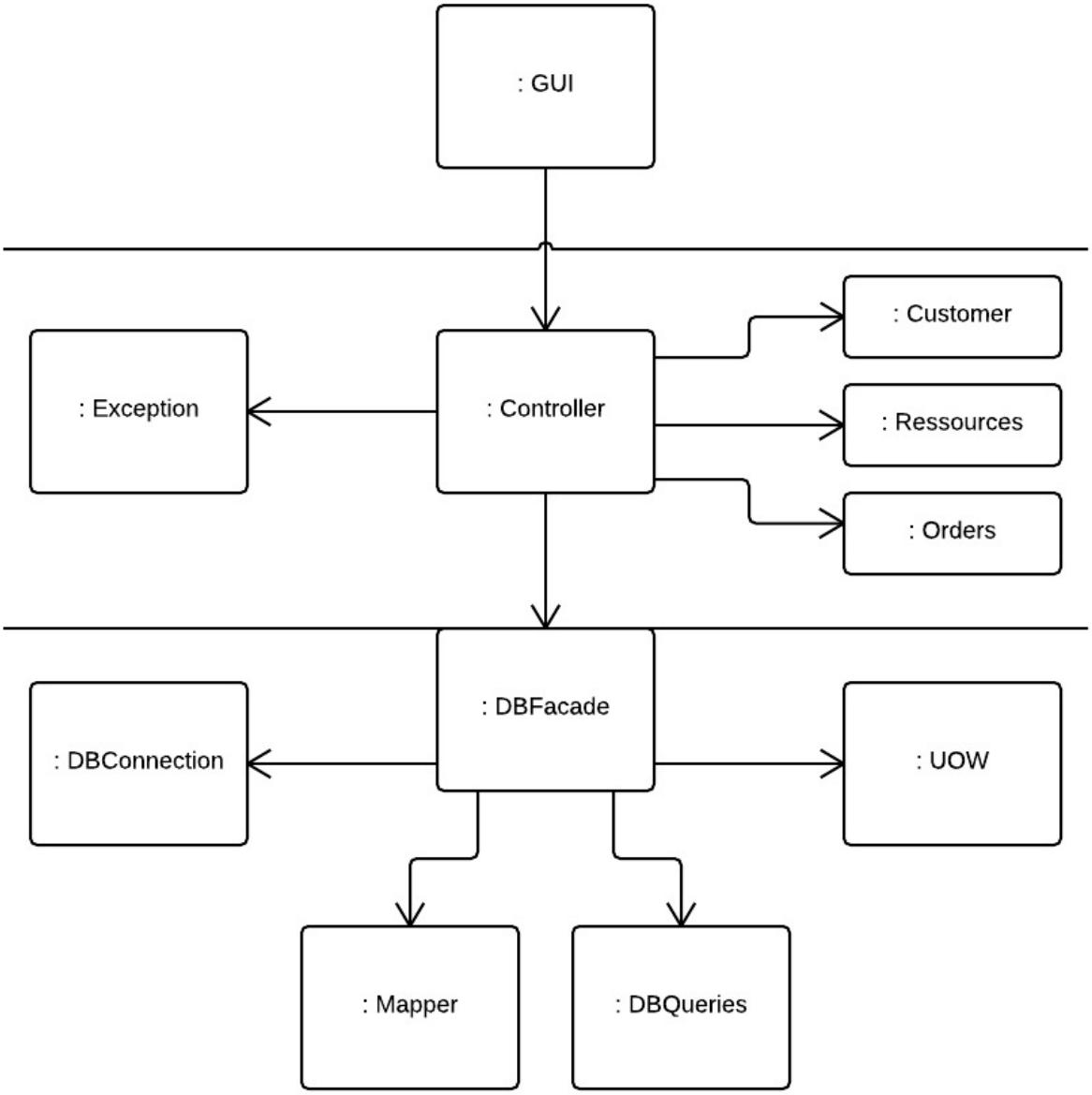
For at kunne skabe et overblik over hvilke relationer, sammenhænge og arbejdsprocesser der eksisterer i firmaet City Stilladser A/S, er der blevet udarbejdet en E/R model som er brugt til at finde relationelle sammenhænge i firmaet. Ud fra E/R modellen har det været muligt at skabe et relationelt skema, som senere er gået over og blevet tegningen for databasen.



Figur 5 - E/R diagram

Fra E/R modellen bliver det meget tydeligt at Order er den mest centrale ting i firmaet, som har relationer til Customer, Salesmen, Team og Ressource. Relationerne til Customer, Salesmen og Team er alle sammen ens, en mere interessant relation er den som går fra Order til Ressource og fra Ressource til Storages. Denne relation bliver meget interessant i udformningen af det relationelle skema.
Som det kan ses på E/R diagrammet er der mellem Order og Ressource en mange til mange relation, denne relation er vist af de 2 asterisk som er mellem Order og Ressource. Dette betyder at mange Order kan have Mange Ressource. Samtidig er der en mange til mange relation mellem Ressource og Storages, denne relation skal forstås som at der kan være mange Ressource på mange Storages og mange Storages på mange Ressource. Disse to mange til mange relationer vil senere hen vise sig at skabe problemer i forhold til implementation af samtidighed.

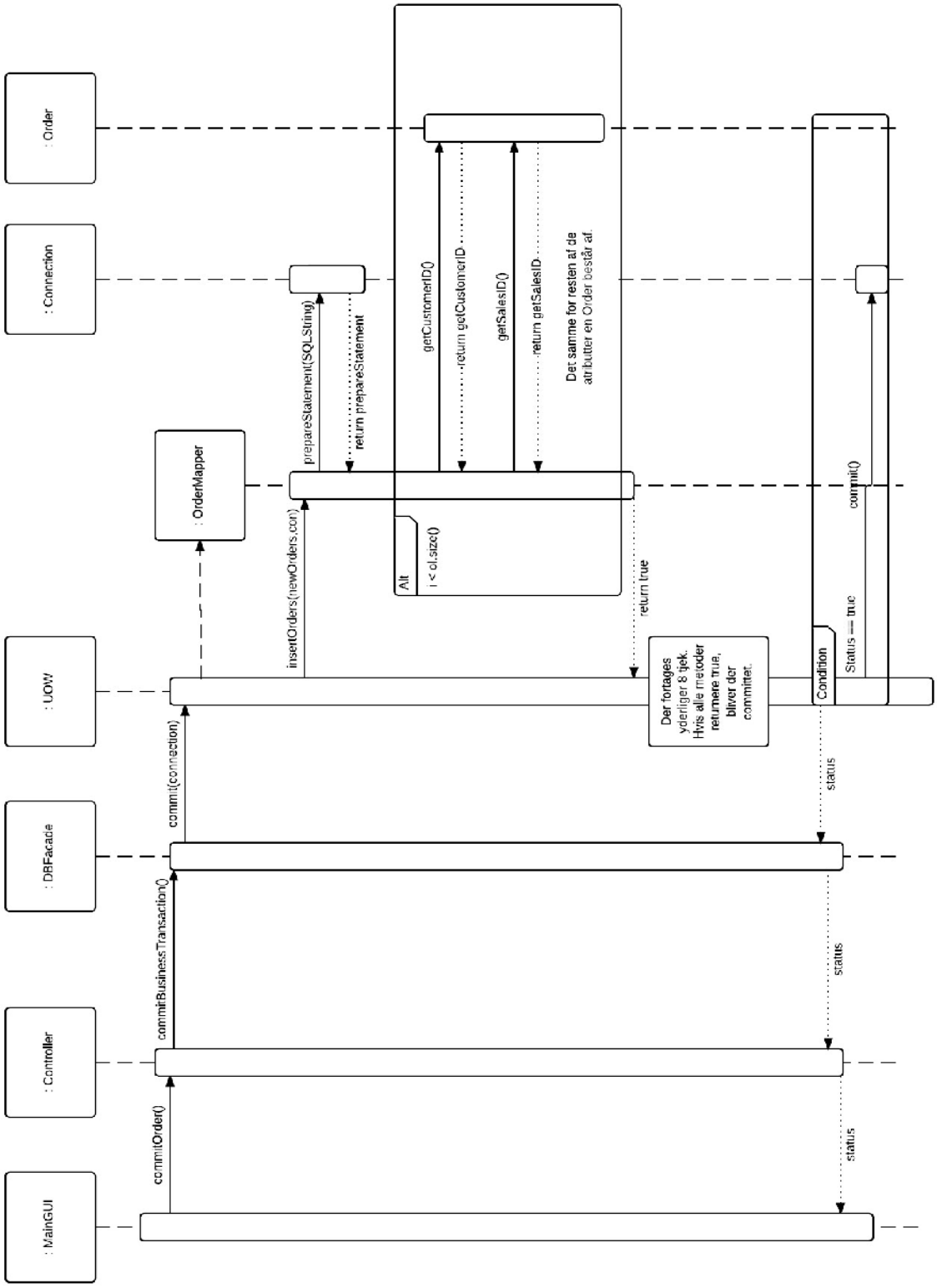
Arkitekturmodel – Morten



Figur 6 - Architecture model

Architecture Model er en forsimplet udgave af en domæne model, der giver et nemt og overskueligt overblik over programmets klasser.
Det gør det nemt for andre der ikke kender programmet, at få et overblik og lave ændringer eller tilføje udvidelser.

Sekvensdiagram – Morten



Figur 7 – Sekvensdiagram

Sekvensdiagrammet (figur 7) viser metoden commit(), hvor en business transaktion afsluttes og en ordre bliver gemt eller opdateret. Det gøres ved at kalde commit() metoden på Unit of Work. Der tjekkes for de samtidighedsproblemer der er, før der kan commites. Dette er beskrevet i afsnittet om Unit of Work.

Relationelle skema – Stefan

Ved at benytte E/R diagrammet som tegning er det blevet muligt at lave et relationelt skema, det relationelle skema giver et præcist billede af hvordan databasen kommer til at se ud. Det relationelle skema er derfor endt med at se således ud:

*Orders (OrderID, **CustomerID(Customers)**, **SalesID(Salesmen)**, confirmed, startDate, endDate, price, ver)*

Teams(TeamID, Fitters, Trucks)

*TeamOrderJoined(**OrderID, TeamID**)*

*Customers(CustomerID, fullName, companyName, address, **ZIP(ZipCodes)**, phoneNumber)*

ZipCodes(ZIP, City)

RessourceTypes(RessourceTypeID, typeName)

*Ressources(**RessourceTypeID(RessourceType)**, **StorageID(Storage)**, qty)*

*Storages (StorageID, address, **zipCode(Zipcodes)**)*

*OrderDetails (**OrderID(Orders), RessourceID(Ressources) StorageID(Ressources)**, qty, ver)*

Salesmen(SalesID, firstName, LastName, phoneNumber)

Det antages at alle disse tabeller er på 1. normalform, da det vil være trivielt at argumentere for denne normalform. Samtidig vil alle tabeller som ikke indeholder en sammensat nøgle automatisk være på 2. normalform. Da 2. normalform kun gør sig gældende for tabeller med sammensatte nøgler, vil den derfor af selvsamme grund ikke blive argumenteret for.

Orders:

Er på 3. normalform fordi alle andre felter er funktionelt afhængige af OrderID.

Teams:

Er på 3. normalform fordi både Fitters og Trucks er funktionelt afhængige af TeamID.

TeamOrderJoined:

Er på 3. normal form, fordi tabellen ikke indeholder andet end én sammensat nøgle. Grunden til at denne tabel findes er for at kunne forbinde en ordre til et team.

Customers:

Er på 3. normal form fordi alle felter er funktionelt afhængige af CustomerID. Det kunne være muligt at tro, fullName og companyName felterne kunne indeholde den samme information. Dette er dog ikke tilfældet da Fullname indeholder navnet på kontakt personen i det pågældende firma, hvor companyName indeholder navnet på firmaet.

ZipCodes:

Denne tabel er også på 3. normalform, der er ikke så meget at sige om denne tabel da den kun fungerer som bindeled mellem postnumre og byer. Det ville kunne være muligt at argumentere for at denne kun var på 2. normalform, dette skyldes at nogle postnumre kunne have samme bynavn. Det er dog i designet processen af databasen blevet antaget at dette ikke er tilfældet.

RessourceTypes:

Er på 3. normalform da typeName kun er funktionelt afhængig af RessourceTypeID.

Ressources:

Er på 2. normalform da der kun er ét felt som ikke er en nøgle. Samtidig opfylder tabellen også kravene for 3. normalform da Qty kun er funktionelt afhængig af den sammensatte nøgle.

Storages:

Er på 3. normalform da både address og zipcode kun er funktionelt afhængig af StorageID.

OrderDetails:

Er på 3. normalform da både Qty og Ver er funktionelt afhængig af den sammensatte primær nøgle (OrderID, RessourceID, StorageID). Feltet Ver er versionen, dette felt er kun i tabellen for at sikre samtidighed.

Salesmen:

Er på 3. normalform da alle felter er funktionelt afhængige af SalesID.

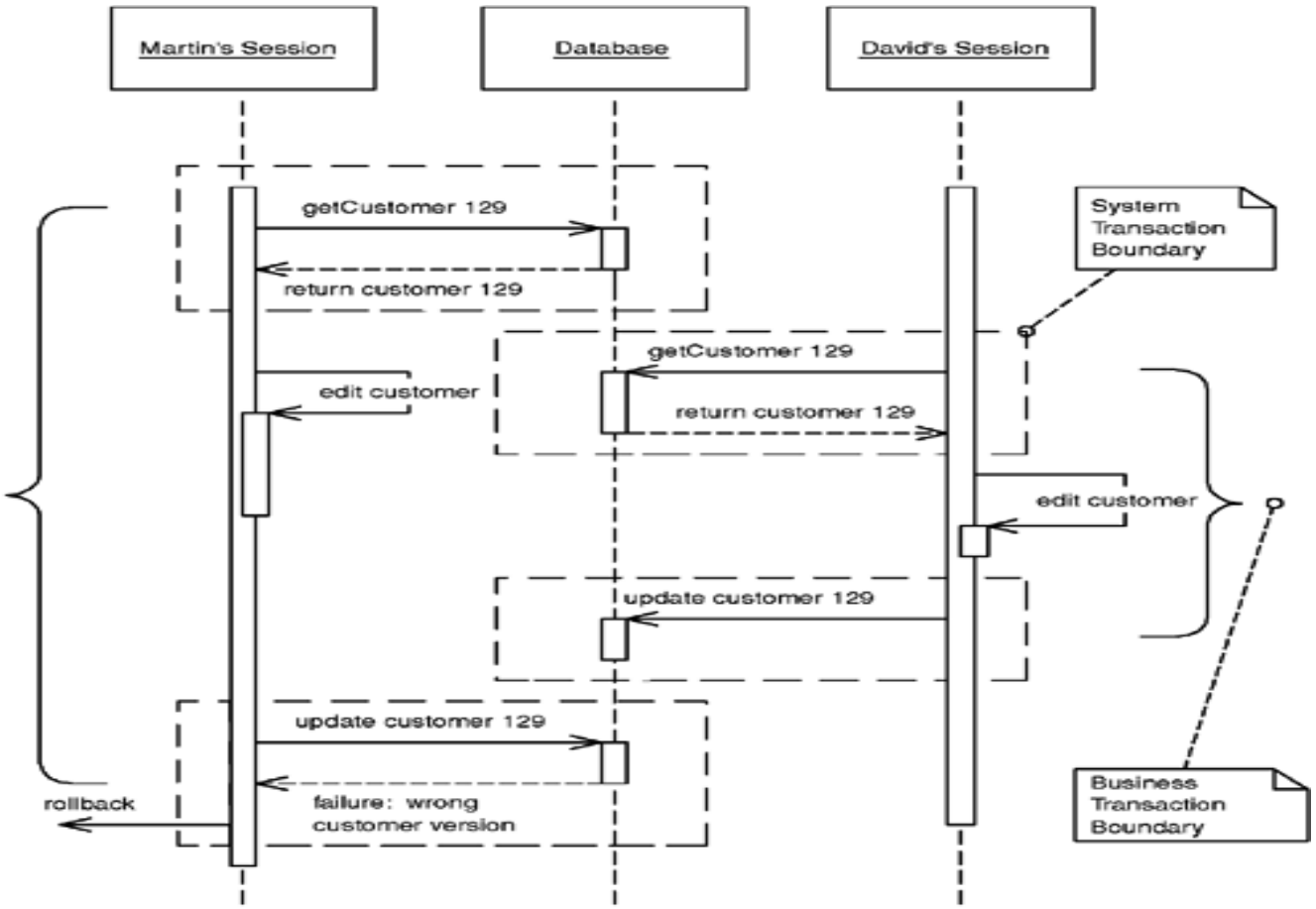
Beskrivelsen af den mest komplekse del af koden

Unit of Work – Morten

Unit of Work er en central ting i et program, hvor der arbejdes med data fra en database. Unit of Work er en proces der kontrollere, de samtidigheds-problemer der kan opstå i et flerbrugersystem.
Når man under en business transaktion opretter, ændrer eller sletter data der er i databasen gennem programmet. Der oprettes et Unit of Work der holder styr på hvilke ændringer der foretages under transaktionen. De ændringer der bliver registreret i Unit of Work, bliver kategoriseret som enten 'New', 'Dirty', 'Clean' eller 'Deleted'.
Ved afslutningen af en business transaktion, bliver samtlige ændringer kontrolleret og vurderet ud for de samtidigheds-problemer der eventuelt kunne være, før ændringerne bliver gemt i databasen. Processen bliver afviklet atomisk. Kontrollering kan ske ved to forskellige 'Patterns'. Optimistic Offline Lock og Pessimistic Offline Lock som beskrives nedenfor.

Optimistic Offline Lock – Morten

Optimistic Offline Lock bruger et versions-nummer til at tjekke om der er foretaget nogle ændringer ved at sammenligne versions-nummeret fra da man startede sin business transaktion til man afslutter den. Hvis nummeret er ændret, afbrydes transaktionen.

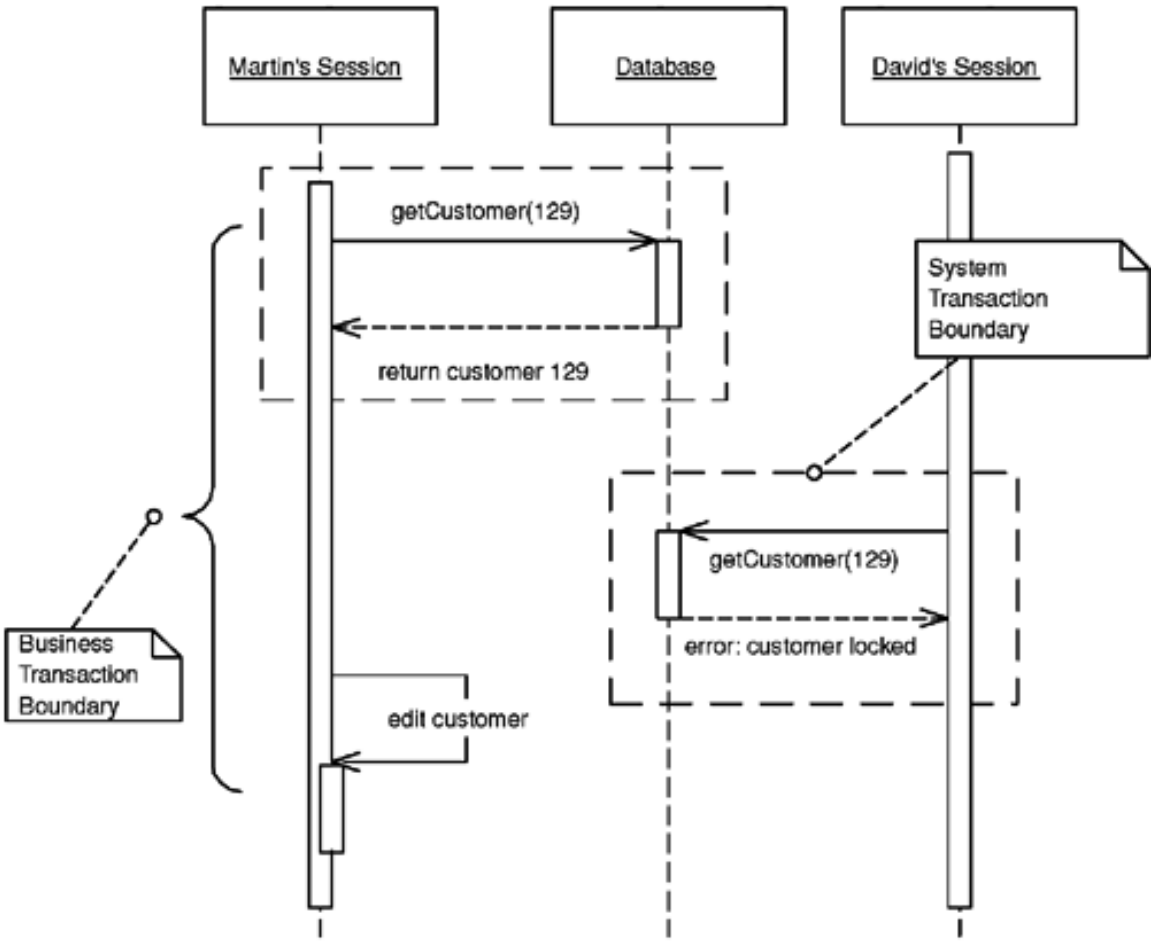


Figur 8 - Diagram over Optimistic Offline Locks funktion (Patterns of Enterprise Application Architecture by Martin Fowler s. 416)

Det uhensigtsmæssige ved at bruge Optimistic offline lock er, at der først laves et tjek af versions-nummeret i slutningen af business transaktionen. Hvis to brugere er i gang med at redigere i samme data i programmet, vil den der afslutter sin transaktion sidst, miste det udførte arbejde. Brugeren vil være nødsaget til at starte forfra, da versions-nummeret under transaktionen ændres grundet brugeren der afsluttede sin transaktion først fik gemt sine data.

Pessimistic Offline Lock – Morten

Ved brugen af Pessimistic Offline Lock forhindres dette problem, hvor en bruger må starte sin transaktion forfra. Dette sker ved at der i en Pessimistic Offline Lock låses dele af databasen i den tid man bruger på at foretage sine ændringer. Dette skaber dog et nyt problem. Ved at låse bestemte dele af databasen i længere tid, forhindre man at andre brugere kan tilgå disse data. Der findes forskellige typer af låse, hvor man f.eks. kan læse men ikke ændre i data i transaktionen.



Figur 9 - Diagram over Pessimistic Offline Locks function (Patterns of Enterprise Application Architecture by Martin Fowler s.426)

En af de vigtigste og komplekse metoder i programmet er Unit of Work. Programmet er et flerbruger-program, hvor der arbejdes med data der holdes i en database, som flere brugere har adgang til. Ved et flerbruger-program er der vigtigt med Unit of Work, da man vil undgå Inconsistent Read og Lost Update.

Ved oprettelse, ændringer eller slettelse af ordre i programmet, hvor databasen involveres opstartes der en ny business transaktion. Når business transaktionen opstartes, oprettes en ny Unit of Work. Unit of Work får tilsendt de objekter der bliver ændret undervejs i transaktionen, hvor de gemmes indtil transaktionen bliver afsluttet. Her bliver ændringerne gemt i to forskellige kategorier. 'New' og 'Dirty'. Ændringer bliver kategoriseret alt efter om brugeren opretter en ny ordre eller om brugeren redigere i en nuværende ordre.

Når brugeren er færdig med at oprette eller ændre i en ordre skal denne gemmes, hvilket afslutter business transaktionen. Ved afslutningen af business transaktionen startes der en Pessimistic Offline Lock, som låser tabellen 'ordredetails'. Dette sikre at andre brugere ikke kan benytte denne tabel. Herefter tjekkes der, ved brug af Optimitisc Offline Lock, om der er nogen ændringer i versions-numrene. Der foretages desuden et tjek omkring lagerbeholdning og om mandskab stadig er til rådighed. Der foretages i alt 8 tjek inden en transaktion kan afsluttes. Hvis Unit of Work ikke finder fejl under de 8 forskellige tjek er business transaktionen fuldført. Hvis ét trin under Unit of Work fejler vil transaktionen mislykkes, da det er en atomisk proces.

MyException – Uffe

MyException klassen er lavet sådan, at den kan kaste nogle tilpassede fejl-beskeder i stedet for de normale beskeder, som Java returnere. Disse beskeder kan blive brugt til hurtigere at forstå hvor fejlen ligger, og derfor kan fejlen hurtigere udrettes. *MyException* klassen er ikke særlig stor, da der ikke har været tid til at lave exceptions for alle tilfælde. Men der er mange tilfælde hvor at man kunne have lavet sine egne exceptions. Det gode ved at lave sin egen exception klasse er, at man som sagt før, hurtigere kan rette fejl, og spare tid. Hvis projektet havde været større, havde der været mange fordele ved at lave exception klassen, og bruge tid på denne, da det kunne spare tid på længere sigt. Det er muligt at man ville blive nød til at bruge en masse tid på at oprette alle disse fejl-beskeder og finde alle fejl-koderne. Tiden kan dog blive vundet igen ved hurtigere at kunne rette fejl.

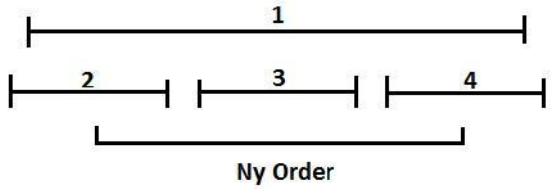
Beskrivelse af den komplekse eller essentielle del af programmets SQL forespørgsler eller opdateringer

Kompleks SQL forespørgsler – Søren

```
SELECT sum(qty)  FROM Orderdetails WHERE OrderID IN (
SELECT orderID FROM Orders WHERE (
                (endDate>=Order startDate AND Order endDate>=endDate) or
                (startdate>=Order startDate) AND Order endDate>=startDate) or
                (Order startDate>=startDate AND endDate>=Order endDate)
AND confirmed=1)
AND RessourceTypeID = 1
);
```

Figur 10

En af de mere komplicerede SQL-forespørgsler (figur 10) i programmet forekommer, når der skal tjekkes hvorvidt en ønsket ressource, er til rådighed i den ønskede periode. Eftersom der ikke er noget spor på en ressource af hvor mange af denne der er udlejet i forskellige perioder, findes alt information i Orderdetails-tabellen. Det vil sige at for at finde information om hvornår ressourcer er udlejet, skal der foretages en SQL-forespørgsel. Ved at lave en forespørgsel med en startdato, slutdato og en ressourcetype bliver der returneret antallet af denne ressource der er i brug i den forespurgte periode. Dette tal kan bruges sammen med en anden forespørgsel der giver det samlede antal af den givne ressource. Hvis det samlede antal ressourcer minus de bookede i perioden IKKE overstiger det ønskede antal ressourcer, kan en ny booking gennemføres.
Selve forespørgslen består af en select med en subselect. Der ønskes antallet(qty) fra orderdetails hvor orderID også findes i ordertabellen, og ordrens datoer falder inden for den givne periode og ordren er bekræftet. Dernæst kontrolleres der også om hvorvidt ressourcetypen i orderdetails stemmer overens med den givne ressourcetype.
Den mest komplicerede del i denne SQL er grænserne for hvorvidt datoerne matcher og dette vil blive gennemgået nedenfor.



Figur 11 - Forskellige modstridende ordre ved oprettelse af en ny ordre

Der findes 4 tilfælde (figur 11) hvor en anden ordre kan overlappe med den nye ordre.
De fire tilfælde vil blive forklaret nedenunder, den aktuelle ordre bliver benævnt som ordre og det tilfælde der bliver forespurgt bliver benævnt som case.

Tilfælde 1:

(Order startDate>=startDate AND endDate>=Order endDate)
En ordre der starter uden for ordren og hvor case startdato er MINDRE eller LIG med ordre startdato, SAMT case slutdato er STØRRE eller LIG med ordre slutdato. Dette vil fange alle ordre der overlapper hele perioden for den nye ordre.

Tilfælde 2:

(endDate>=Order startDate AND Order endDate>=endDate)
Næste tilfælde opfanger modstridende ordrer der starter før den nye ordre og slutter inde i perioden. Altså case enddate er STØRRE eller LIG med ordrens startdato. Derudover er case enddate MINDRE eller LIG med ordrens endDate. Denne afgrænsning vil også fange tilfælde 3.

Tilfælde3:

Dette tilfælde bliver fanget af begge de to afgrænsninger for tilfælde 2 og 4. Dette skyldes at de to afgrænsninger for at fange case 2 og 4 enten tester på startDato eller slutDato to gange. Altså vil case 3, der ligger præcist inde i perioden altid have end startDato der er STØRRE eller LIG med ordrens startDato og samtidig være MINDRE eller LIG med ordrens slutDato. Det samme gælder for case 3 slutDato blot med omvendt fortegn.

Tilfælde 4:

(startdate>=Order startDate) AND Order endDate>=startDate)

Denne afgrænsning fanger den sidste case, hvor en modstridende ordre starter inden i den nye ordres periode men går udover den nye periodes slutdato.

Beskrivelse af kvaliteten af vores design – Søren

Programmet er forsøgt designet således at det følger et par simple programmeringsmønster. Hovedsagligt er det lagt fokus på følgende:

- 3-lags-model(Ansvarsfordeling)
- Singleton
- Facade
- Data Mapper
- Unit of Work

Nedenfor følger nogle uddybende forklaringer af de enkelte mønstre der er blevet benyttet.

3-lags modellen

Eftersom antallet af klasser i programmet overstiger mere end en håndfuld, er det nødvendigt at inddele de forskellige klasser i forskellige lag. Dette gøres dels for forståelse, men i høj grad også for at sikre at ansvar bliver delt ud til de rigtige klasser. Altså at en klasse gør det den er kodet til og programmet har høj kohæsion. Har klassen så brug for noget andet der ligger udenfor dennes ansvarsområde sørger den for at give ansvaret videre. Et eksempel kunne være at programmets GUI KUN har ansvaret for at opsætte brugergrænsefladen. Der er derfor ikke noget kode der lige pludselig evaluere data. Når der er brug for at en bruger skal have behandlet data, sender GUI-klassen ansvaret videre til underliggende lag der kan placere ansvaret det korrekte sted. Jo større ansvarsområde en klasse/objekt har, desto svære er det at genbruge koden.

Programmet er også forsøgt designet, så afhængighed mellem to klasser kun går den ene vej. Målet er at forsøge at lave en lav-kobling så to dele klasser fx ikke er direkte afhængige af hinanden. Lad os sige at der på et tidspunkt ønskes at lave en udvidelse til programmet. Ved at have lav kobling, altså at de enkelte klasser ikke har en stor afhængighed for hinanden, kan en klasse skiftes, eller der kan blive tilføjet klasser, uden at det får stor indflydelse for resten af programmet.
For at sikre at disse ting bliver overholdt, lav kobling og høj binding, følger programmet 3-

lags-modellen. Modellen bringer en række fordele med sig. Fx gør det en eventuel udskiftning af dele af programmet langt nemmere. Skal brugergrænsefladen for eksempel skiftes fra et vindue på en computerskærm til en hjemmeside eller en mobilapp, kan dette gøres ved kun at skifte præsentationslaget. Det kunne også være at databasen var forældet og trængte til en udskiftning. Her kan hele datalaget ligeledes udskiftes uden at det påvirker resten af programmet.

Modellen består af 3-lag: Præsentations-, domæne- og data-lag.

Præsentationslaget, som er det øverste af programmets lag, står for at vise brugergrænsefladen samt modtage input fra brugeren. Det sørger for at oversætte opgaver og resultater fra programmet til noget brugeren kan forstå.

Domænelaget er det lag der koordinere programmet, uddeler og behandler opgaver og foretager behandlinger og evalueringer af data. Der styrer altså også trafik af data mellem lagene.

Datalaget er det lag hvor alt det information, programmet har behov for at gemme, bliver gemt. Det er også dette lag der hiver information ud og giver videre til domænelaget.

Singleton

For at sikre at visse klasser kun bliver instantieret en enkelt gang, har programmet flere steder et singleton pattern. Dette pattern er brugbart når det vides at der kun skal bruges præcist et enkelt objekt. Programmet har kun brug for fx et controller objekt. GUI har flere steder brug for at benytte metoder i controller klassen. For at sikre at GUI ikke kommer til at instantiere flere controller objekter, kan controller klassen kalde getInstance() (Se Figur 12).

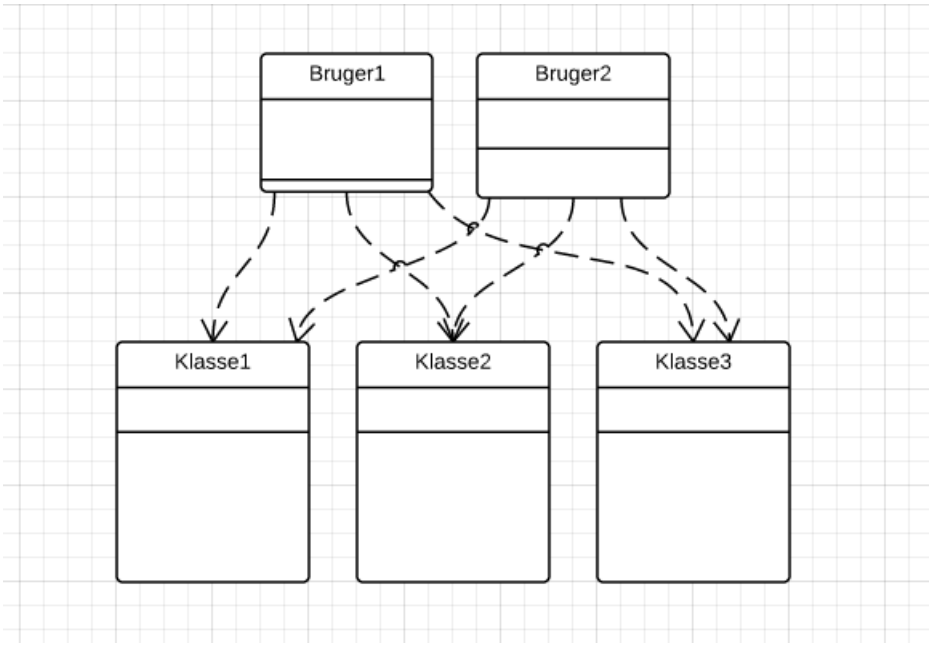
```
public static Controller getInstance()
{
    if (instance == null) {
        instance = new Controller();
    }
    return instance;
}
```

Figur 12

Eftersom en singleton klasse har en private constructor kan denne ikke kaldes af andre klasser. Alle andre steder i programmet, der ønsker at gøre brug af Controlleren må derfor kalde getInstance(). Denne metode sørger for at hvis klassen ikke har instantieret den statiske variable instance, altså den er null, så kan den selv sørge for at kalde den private constructor og derefter returnere instance.
Udover Controller-klassen følger DBFacade-og DBConnector-klassen også singletonpattern

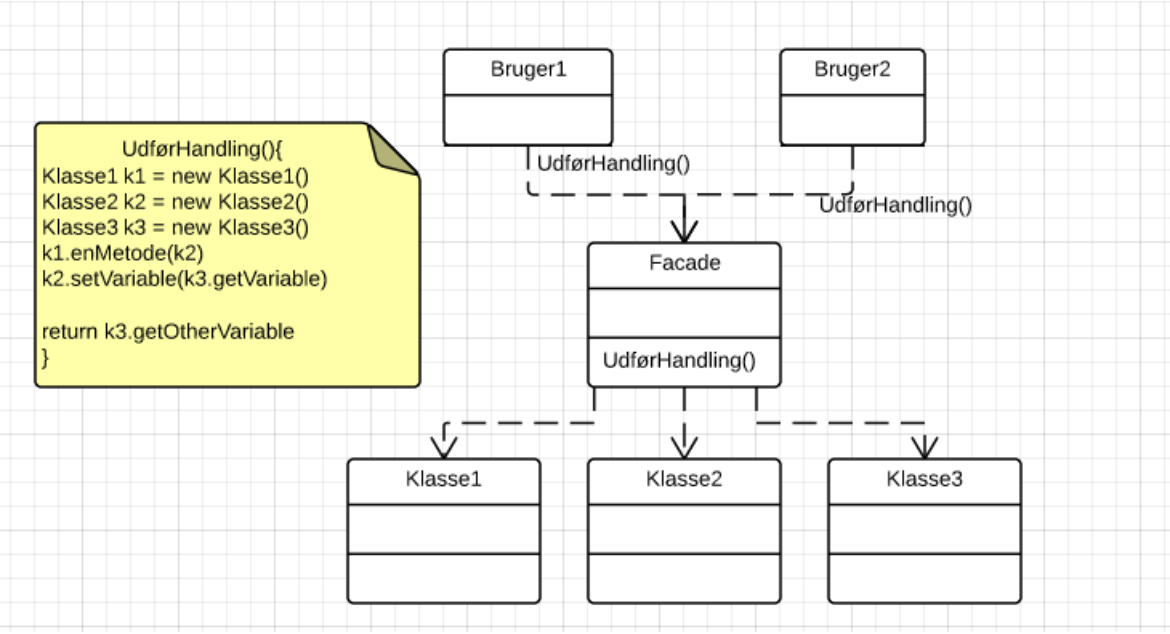
Facade

For at sikre et forståelig og simpelt interface benytter programmet Facade-mønstret i Controller- og DBFacade-klassen. De fungerer som mellemled mellem de forskellige lag i 3-lags-modellen og forsimpler derfor kald fra et lag til et andet, ved at disse kald SKAL ske igennem facaden. Når brugeren vil forsøge at lave en handling der har noget med databasen at gøre, sørger programmet for at alt skal ske igennem databasefacaden. Så fremfor at controlleren skal have kendskab til samtlige mapper-klasser, kan controlleren nøjes med at have kendskab til databasefacaden. Facaderne holder ikke selv de mest komplekse dele af koden, men holder styr på hvor ansvaret skal deles ud, og sørger for at give denne information på tværs af lagene fx til en bruger.



Figur 13 - Diagram der viser hvordan to brugere foretager en handling uden en facade

Et simpelt eksempel (figur 13) ville være at to bruger-klasser skulle lave den samme handling på en række underliggende klasser. For at de begge kan lave handlingen på klasserne skal bruger-klasserne kende samtlige underliggende klasser. Ved at få dem til at gå igennem en facade mindskes koblingen mellem lagene.



Figur 14 - To brugere foretager nu en handling igennem en facade

I dette diagram (figur 14) går kaldet igennem facaden hvilket hjælper med forståelse men også som det beskrevne ”samlingspunkt” mellem lagene.

Datamapper

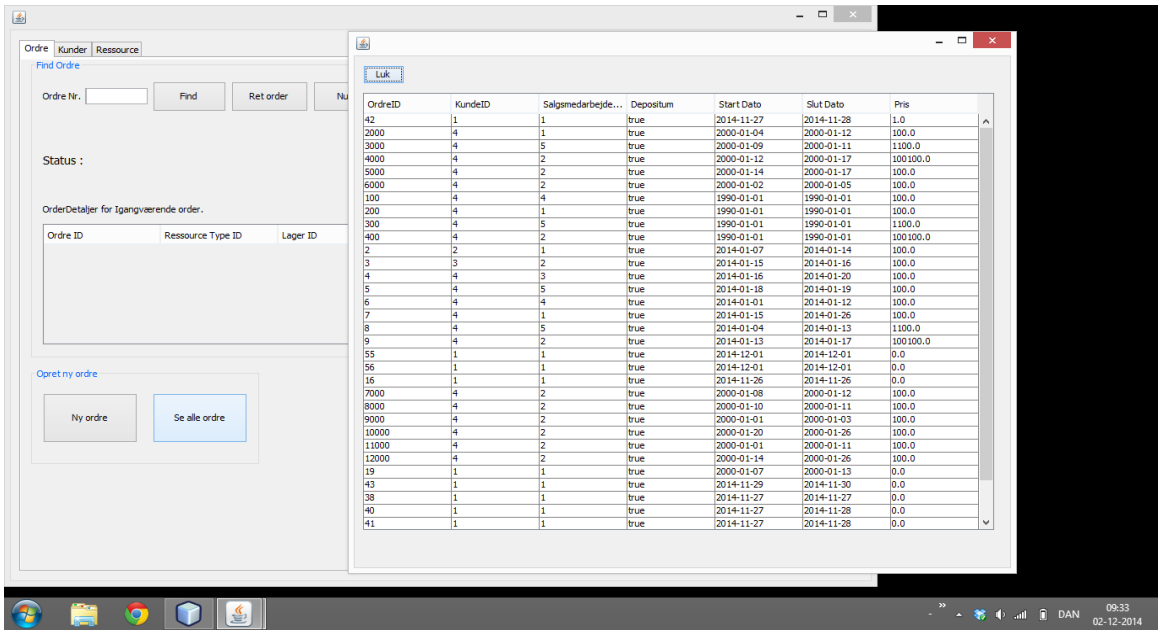
For at flytte data fra programmets data-lag til domæne-lag, benytter programmet et Data mapper pattern. Dette sikre at data i databasen kan blive omdannet til data i hukommelsen, således at programmet kan arbejde med data som objekter. Det går også den anden vej, så et objekt i entitetslaget der har været arbejdet med i programmet skal gemmes til databasen, kan en data mapper sikre at data bliver gemt korrekt. Selve klassen der bliver mappet til i domænelaget kender ikke til mapperen.
Programmets data mappers følger det samme pattern og indeholder omtrent de samme metoder: insert, update og delete. Metoderne trækker data ud fra et objekt i entitetslaget, eller trækker data ud fra et resulset og omdanner disse til et objekt.
Overvejelserne omkring mapper klasserne har gået på hvorvidt en mapper skal trække stort set alt data fra databasen ud for at oprette et objekt. Et eksempel er at programmets ordremapper. En ordre indeholder mange informationer om andre entiteter, og derfor har det været vigtigt at vurdere hvor meget der skulle mappes til det nye orderobjekt. For at sikre at programmet forbliver objektorienteret bliver en kunde mappet i ordremapperen. Dette

skyldes at databasen ikke kender til objekter og derfor kun knytter en kunde til en ordre via en integer. Dette kan medføre at nogle mappers i princippet kan blive overflødige. En ordre har ordredetaljer, og en ordredetalje er sin egen entitet i domænelaget. Dog vil der aldrig blive behov for kun at mappe en ordredetalje da den knytter sig så meget til en ordre, at det ikke vil være fordelagtigt at lave en mapper kun til ordredetaljer.
Der er flere steder hvor programmet kan udvides med flere mapninger af referencer til andre objekter frem for kun referencer databasen forstår via integers.

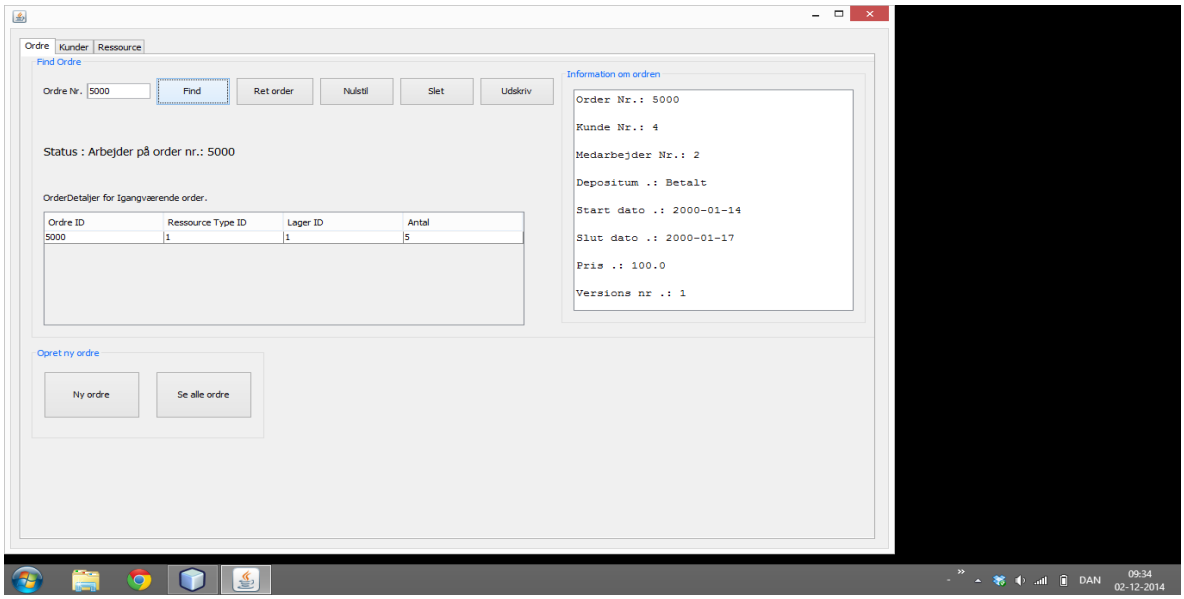
Unit Of Work

Programmet benytter Unit of Work pattern som er beskrevet i afsnittet om de mest komplekse dele af koden.

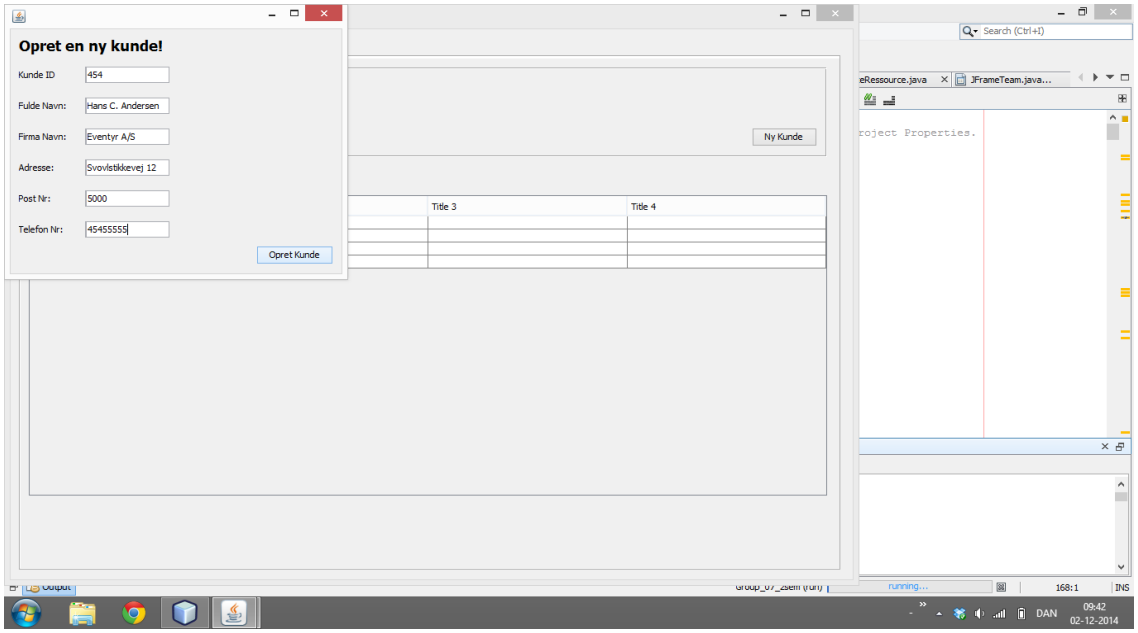
Screen Dumps – Uffe



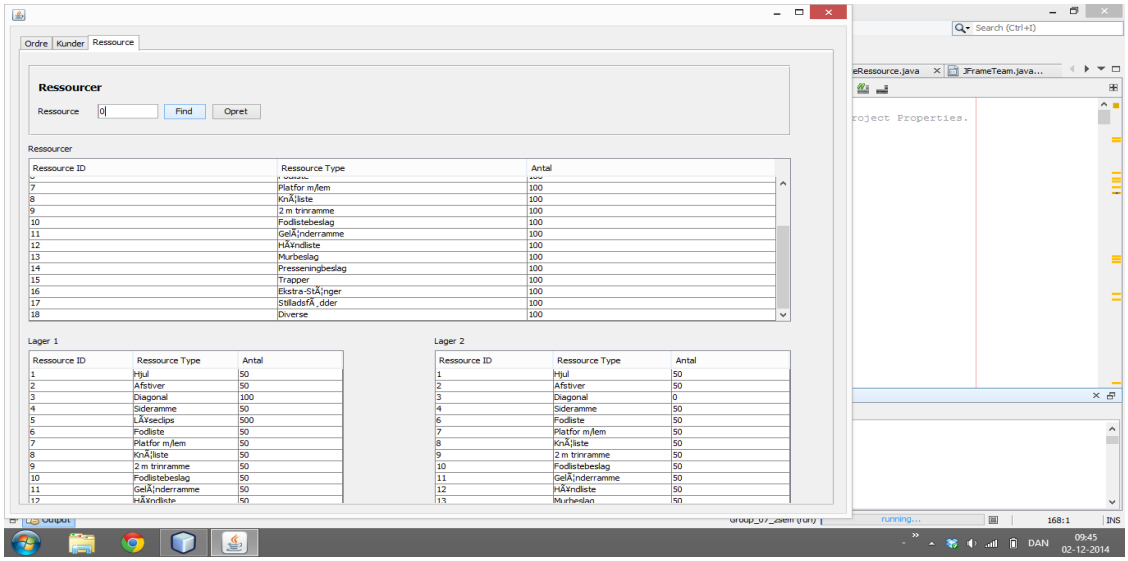
Her bliver der trykket på knappen "Se alle ordrer". Denne knap findes i "Ordre" fanebladet. Derefter ses der en liste, som indeholder alle ordrer i databasen.



Her forsøges der at finde ordren med ordre nr. 5000. Efter et tryk på "find" kommer der nogle informationer frem til højre, samt nogle detaljer om diverse ressourcer kommer frem nedenunder.



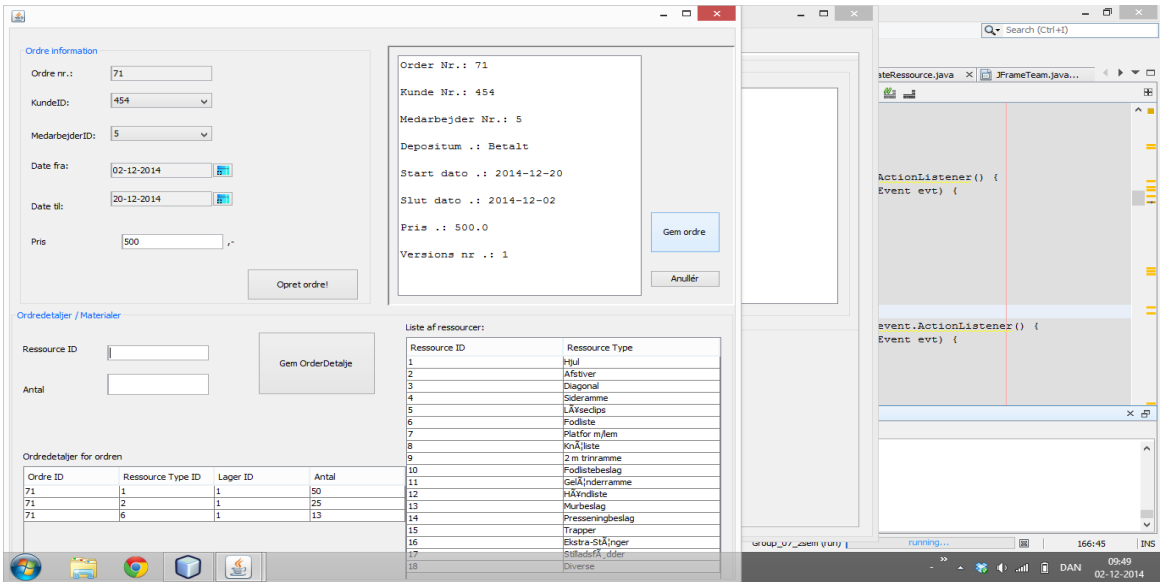
Her bliver der oprettet en kunde. Dette kan man gøre efter at have trykket på knappen "Ny Kunde" som ses til højre i billedet. Knappen findes i "Kunde" fanebladet.



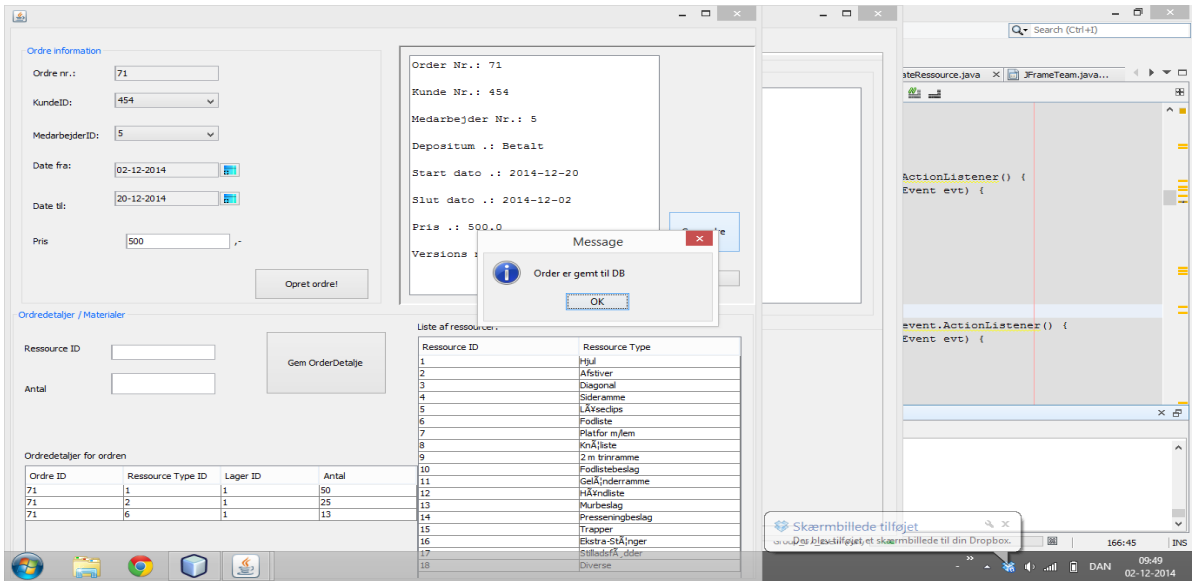
Her kan man søge på forskellige ressourcer for at se hvor mange der er tilgængelige. Men også hvor mange der er tilgængelige på de to lagre. Hvis man skriver "0" i ressourceID feltet og trykker "Find", får man vist alle ressourcer på lageret.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen

Stefan Duro
Uffe Nedergaard



Her bliver der oprettet en ordre med tilhørende materialer. Billedet er taget efter at knappen "Opret Ordre" er trykket, og efter at 3 forskellige ressourcer er tilføjet. Ordren er stadig ikke gemt i databasen, da det først skal tjekkes om der er blevet booket ressourcer af andre brugere, inden den kan gemmes.



Billedet er taget efter at knappen "Gem Ordre" er blevet trykket. Programmet checker at alle ressourcer stadig er ledige, og hvis de er, så gemmer den dem i databasen.

Dokumentation af alle test aktiviteter – Stefan

I projektet er der kun lavet én testklasse som sørger for at teste alle metoder i *OrderMapper* klassen. Denne klasse har kun til formål at fungere som bindeled mellem databasen og programmet. I dette tilfælde er det for *Order* klassen og dens underklasse *OrderDetail* som mapperen håndterer.

Der bliver brugt en anden database som test database end den database som programmet benytter, dvs. alle tests der bliver udført i testklassen er uafhængig af hoveddatabasen. Dog skal der gøres opmærksom på at begge databaser er blevet sat op, med det samme SQL script (*DBscript.sql*), så de er identiske.

I test pakken bruges klassen kaldet *Fixture* til at opsætte den test data som *OrderMapperTest* benytter sig af. Metoden Fixture.setUp bliver afviklet inden hver test bliver udført i *OrderMapperTest*, da kaldet Fixture.setUp er placeret i setUp()h metoden i *OrderMapperTest.* JUnit sørger for at udføre dette inden hver test bliver udført.

JUnit Tests i *OrderMapperTest*

Test metode	**Test Grundlag**	**Hvad metoden tester**
testGetOrderIDZero	Prøver at hente en order med OrderID = 0	Order retuneret = null
testGetOrderNegative	Prøver at hente en order med OrderID = -1	Order retuneret = null
testGetOrderIDExist	Prøver at hente en order med OrderID = 990	Order retuneret har OrderID = 990
testInsertOrderDetailsEmptyList	Prøver at indsætte en tom OrderDetail ArrayList	Retuneret true, det lykkedes at indsætte ingen OrderDetail
testInsertOrderDetailsOneOrderDetail	Prøver at indsætte én OrderDetail	Retunerert true, det lykkedes at indsætte én OrderDetail
testInsertOrderDetailsMultipleOrderDet	Prøver at indsætte flere	Retunereret true, det

ials	OrderDetail	lykkedes at indsætte flere OrderDetail
testUpdateOrderDetailsOneElement	Prøver at updatere Ordre med OrderID 990, ResType 1, StorageID 1	Tester at Mapper metoden retunerer true og at det er ændret i databasen
testUpdateOrderDetailsMultipleElements	Prøver at updatere flere Ordre	Tester at Mapper metoden retunerer true og at det er ændret i databasen
testInsertOrdersMultipleOrders	Prøver at indsætte flere Orders	Tester at Mapper metoden indsætter alle ordre
testInsertOrdersOneOrder	Prøver at indsætte én Order	Tester at Mapper metoden indsætter orderen
testInsertOrdersEmptyList	Prøver at indsætte en tom Order ArrayList	Tester at Mapper metoden indsætter ingen Ordre
testGetAllOrders	Prøver at hente alle Order	Tester om størrelsen på den ArrayList som retuneres er lig 5

Nogle af test tilfældene er meget banale fx. testes der for om det er muligt at bruge metoden insertOrders(ArrayList<Order>, Connection) til at indsætte en ArrayList uden nogle *Order* objekter, i det givende tilfælde vil det forventes at metoden returnerer true.
Målet med testningen er at ramme grænsetilfældene i den givende metode. For at kunne gøre dette kræves der en god del tid til at vurdere, hvad man forventer der skal ske i de givende tilfælde.
Nogle tilfælde bliver helt bevidst undladt at blive testet. Tilfældet hvor fx. Connection i InsertOrders metoden kunne være null. Det er muligt med fuld sikkerhed at fravælge at teste

for dette, da Connectionen hentes fra *DBConnector* som bruger singleton pattern.

Konklusion – Mads

Produktet indeholder en måde hvorpå man kan oprette en ordre, der bliver gemt i databasen og derefter kan blive fundet inde i GUI 'et, så man kan se at den er blevet gemt. Det er også muligt for programmet at tilknytte ressourcer til en ordre, og tjekke en lagerstatus på diverse ressourcer der skal bruges til at opsætte stilladset. Derudover er det muligt at gå ind og tilknytte teams til en ordre, via GUI 'et. Det er også muligt for en bruger at gå ind og udskrive en pakkeliste.
Det er ikke den fulde liste af specifikationer der kunne opfyldes i programmet, grundet der ikke var tilstrækkelig tid. Men det er dog et program der er funktionelt og har nok basis funktioner, til at kunne fungere og kunne producere det som blev prioriteret af product owner.

Instruktioner i hvordan man installere programmet – Uffe

1. Indsæt USB-stik i computeren.

2. Udpak "group_07_2sem.zip" til den ønskede location.

3. Åbn NetBeans 8.0 eller nyere.

4. Åbn projektet ved at trykke på: File > Open Project, naviger til den valgte location fra punkt 2.

5. Klik "Run Project" eller tryk F6.

"Print ordre" - knappen opretter en textfil i projektmappen. Den findes normalt her:

C:\Users\"Username"\Documents\NetBeansProjects.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen

Stefan Duro
Uffe Nedergaard

SCRUM

User stories – Hele gruppen

Færdigt
Undervejs
Ikke begyndt

ID	Name	Importance
1	Visning af stillads materialer	60
2	Visning af team	70
3	Tilføj team	75
4	Oprettelse af ressourcer	85
5	Sletning af ressourcer	35
6	Fremsendelse af tilbud til kunde	40
7	Oprette en kunde	80
8	Ændring af kunde	50
9	Udskrivning af pakkeliste	90
10	Ændring af resourcer	45
11	Tilknytning af teams til en ny ordre	65
12	Oprettelse af ordre	100
13	Ændre ordre	55

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen
Stefan Duro
Uffe Nedergaard

ID	How to demo
1	Tilføj ressource. Se om status ikke er booket. Book materiale ved tryk i GUI. Se på listen om ressource nu er booket
2	Tilføj et nyt team. Se om det er blevet oprettet ved at tjekke inde i team listen
3	Tilføj team. Se om status ikke er booket. Se på listen om teamet nu eksisterer.
4	Tryk opret ressource i GUI. Indtast information. Kig i liste over ressourcer efter den nye ressource.
5	Find og marker ressource til sletning i liste af ressourcer. Tryk slet. Tjek listen igen for ressourcen.
6	Opret en bookning og tryk send. Se nu om tilbuddet er under sendte tilbud.
7	Tryk opret kunde under kunde GUI. Indtast informationer og en ny kunde. Kig i liste over kunder efter den nye kunde.
8	Find og marker kunde i kundelisten. Skift informationer til ændring i formularen. Se i listen over kunder efter ændringer.
9	Tryk print pakkelisten og se om tekst filen bliver oprettet og find den inde i mappen.
10	Find og marker en ressource der skal ændres. Udfyld formularen og check inde i ressource tabben om ressourcen er blevet ændret.
11	Gå ind i ordre ordre listen og check om teamet der skulle tilknyttes ordren vises på listen under ordre tabben.
12	Tryk opret ordre i ordre tabben og indtast de nødvendige informationer, og opret ordren. Check under ordre tabben om ordre nummeret er der, og om informationerne er der.
13	Gå ind i ordre GUI 'et og find ordren der skal ændres. Ændre de nødvendige informationer, og gem ændringerne. Gå derefter ind i ordre listen og se om de informationer der skulle ændres, er blevet ændret.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen
Stefan Duro
Uffe Nedergaard

ID	User Story
1	En bruger skal kunne se om stillads materialer er booket eller ej.
2	En bruger skal kunne se om et team er booket eller ej.
3	En bruger skal kunne se om et team, som derefter skal kunne vises i en liste over all teams.
4	En bruger skal kunne oprette ressourcer.
5	En bruger skal kunne slette ressourcer.
6	En bruger skal kunne sende et tilbud til en kunde.
7	En bruger skal kunne oprette en kunde i systemet.
8	En bruger skal kunne ændre informationer om en kunde i systemet
9	En bruger skal kunne udskrive en pakkeliste med booket materiale til det valgte projekt.
10	En bruger skal kunne ændre informationer om et stillads materiale.
11	En bruger skal kunne tilknytte ledigt personale og køretøjer til en given ordre
12	En bruger skal kunne oprette en ordre og tilknytte materialer til ordren.
13	En bruger skal kunne ændre en ordre.

Morten Schultz Laursen
Søren Tromborg Larsen
Mads Rosenmeyer Poulsen

Stefan Duro
Uffe Nedergaard

ID	Tasks
1	Tilføj tabel for materialer. Oprette GUI til at se ledige ressourcer.
2	Tilføj tabel for teams. Tilføjelse til GUI for at se om teams er bookede eller ej.
3	Tilføj tabel til ressourcer. Tilføjelse til GUI for at se om det nye team er blevet tilføjet
4	Tilføjelse til GUI der kan modtage bruger-input og placere dette i relevant tabel.
5	Kunne vælge en ressourcer fra eksisterende liste i GUI og slette denne database samt GUI.
6	Tilføjelse til GUI: Oprette skabelon til udfyldelse af tilbud. Ud fra en skabelon skal systemet kunne udregne en pris og sende prisen til kunden
7	Tilføjelse til databasen: Oprette en kundetabel Tilføjelse til GUI: brugergrænseoverflade skal kunne modtage input og placere dette i databasen.
8	Tilføjelse til GUI: Visning af information om kunde og knap til fomulat til ændring af information ti database.
9	Tilføjelse til GUI: Knap til at udskrive pakkelisten fra en ordre til en tekst Oprettelse af funktion der samler samtlige bookede ressourcer til en streng.
10	Tilføjelse til GUI: Knap til åbning af formular til at ændre information i database.
11	Funktion til at tjekke ledige teams. Funktion til at knytte teams til en ordre.
12	Tilføjelse til databasen: Ny tabel for ordre. Funktion til at udregne ledige tider hvor der er materiale til rådighed til den valgte ordre. Tilføjelse til GUI: Skabelon til oprettelse af ny ordre til indsættelse i DB. Tilføjelse til GUI: Visning af ordre. Tilføjelse til GUI: Visning af ordre. Oprettelse af database. Oprettelse af nødvendige diagrammer.
13	Tilføjelse til GUI: Brugergrænseoverflade til at kunne gå ind og ændre en ordre via en knap, knyttet til et ordre nummer.

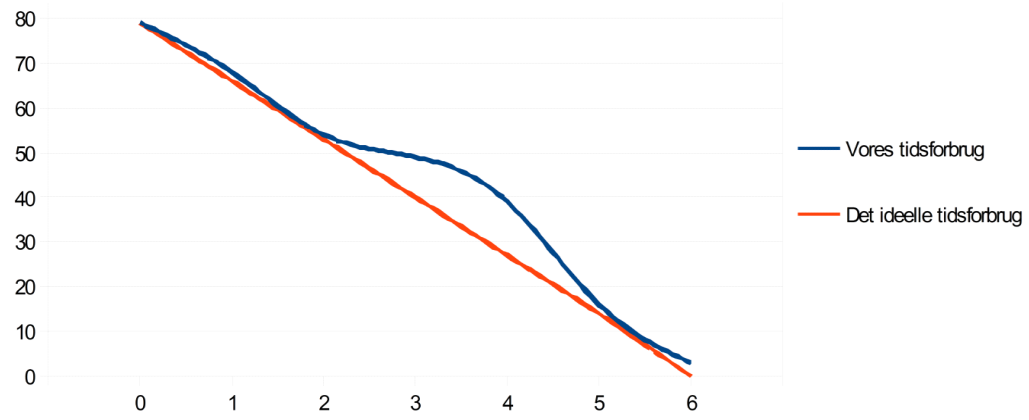
ID	Time Estimate
1	0,5
2	0,5
3	0,5
4	2
5	1
6	1
7	1
8	0,5
9	2
10	1
11	2
12	5
13	2

Burn down Chart – Uffe

Første Sprint:

X-aksen viser de antal dage der er i sprintet.

Y-aksen viser antal timer tilbage før sprintet er færdigt.

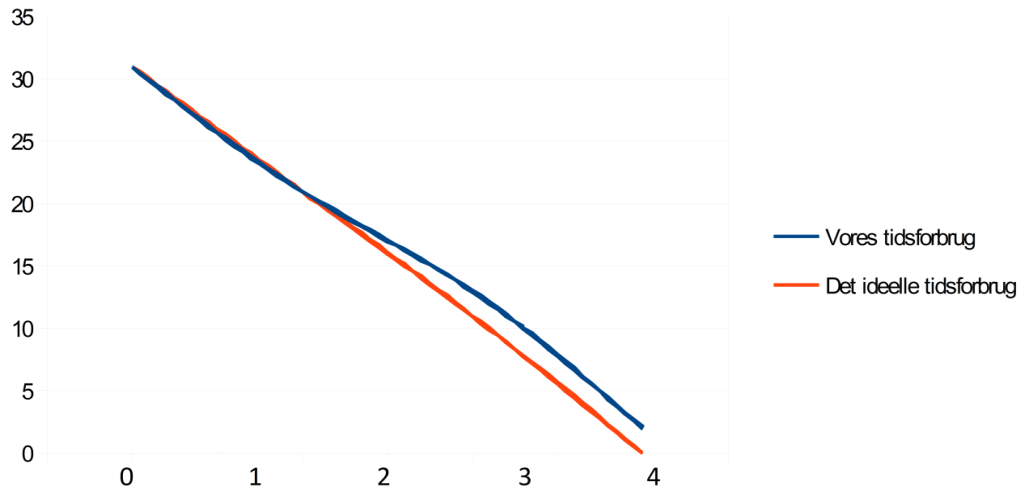


Figur 15 - Burn down Chart (Første sprint)

Andet Sprint:

X-aksen viser de antal dage der er i sprintet.

Y-aksen viser antal timer tilbage før sprintet er færdigt.



Figur 16 - Burn down Chart (Andet sprint)

På begge "Burn Down Charts" kan man se at vi har estimeret vores tid rimelig tæt på det endelige resultat. I begge tilfælde har vi dog været lidt længere tid om det, end først antaget. Både i sprint 1 og i sprint 2 starter vi med at være på linjen, men eftersom tiden går kommer vi længere og længere væk fra "det ideelle tidsforbrug", dog aldrig mere end et par timer. På sprint 1, mellem dag 2 og 3, har vi enten arbejdet langsommere end forventet, eller haft mere end forventet at arbejde med. Til gengæld bliver det udlignet imellem dag 4 og 5 hvor vi har estimeret for meget tid.

SCRUM konklusion – Mads

Scrum var en meget vigtig del af projektet, da det var et virkelig godt værktøj at bruge til at finde ud af hvad der skulle laves, hvornår det skulle laves og hvor vigtigt det var at det blev lavet. Det var en meget god måde hvor holdet altid kunne have et overblik over hvor langt vi var i processen med alle tingene vi var i gang med, og hvor meget vi havde tilbage der stadig skulle laves inden vi var færdige med vores sprint og vores program.

Vi kunne også mærke hen mod slutningen af Scrum perioden, da vi ikke havde nogle 'user

stories' der kunne indbefatte vores behov til at få programmet pudset af, at det blev sværere at finde ud af hvor langt vi var med diverse processer og vores rapport skrivning.

Retrospektiv om første sprint forløb – Mads

I det første sprint, fik vi at vide at det var vigtigt at vi kunne oprette en ordre, og tjekke om ressourcerne ordren skulle bruge var ledige i den periode som ordren forløb over. Da vi havde estimeret det til at det ville tage 5 dage, og vi havde 7 dage i vores sprint valgte vi også at vi ville kunne oprette en kunde i sprintet, da det tog ca. 2 dage i forhold til vores estimering.

Vi havde lidt problemer med at få vores ressource check metode til at virke, men Søren fik det afklaret, og ellers var der ikke noget der voldede problemer, andet end nogle design problemer, som der skulle diskuteres. Vi klarede alt hvad vi havde sat os for at kunne klare i vores første sprint, så da vi viste vores produkt i til palle var han glad for hvad vi havde nået og lavet.

I det første sprint møde fik vi så at vide at vi manglede en user story; "Ændre en Ordre" som vi så oprettede på vores user stories og at det var vigtigt hvis vi kunne gå ind og knytte teams til vores ordre, oprette en ressource og hvis vi havde tid var det godt hvis vi kunne gå ind og udskrive en pakkeliste, da det var en task der var både lille, men også en task der viste at vores produkt rent faktisk kunne de ting vi havde brugt det første og det andet sprint på at lave.

Retroperspektiv om andet sprint forløb – Mads

I det andet sprint, var det vigtigt at vi fik afklaret at man kunne booke et team til en ordre, og tjekke om et team var ledigt i den givne periode som ordren skulle oprettes i. Vi valgte også at det var vigtigt at man kunne udskrive en pakkeliste, da det var en funktionalitet som var både kort men også var en funktionalitet som man kunne lave hurtigt.

Oprette en ressource var også en vigtig user story for os, da den indbefattede både at man kunne oprette et hold og at man kunne oprette et nyt materiale til stilladserne. Med de user stories, inklusiv vores user stories fra vores første sprint forløb, betød at i vores andet sprint havde vi et program som var funktionelt, og hvis ikke andet, havde vi i det mindste et program hvor en salgsmedarbejder kunne gøre ting, som var vigtigt for firmaet.

Retroperspektiv om tredje sprint forløb – Mads

I det tredje sprint forløb var det vigtigt at vi begyndte at kigge på de ting vi allerede havde lavet, og derefter begyndte at sige hvad der kunne forbedres og hvad der måske skulle laves om, for at gøre produktet både mere funktionelt men også mere brugbart. Der var også meget der stadig skulle finpudses i vores kode, og det var det der blev arbejdet mest på.

Da vi fik snakket med Product Owner, var han glad for hvor langt vi var kommet, selvom der var nogle små problemer med vores design valg for nogle af delene for vores GUI. Efter det tredje møde med product owner, valgte vi at det var vigtigt at vi fik afklaret de små problemer vi havde i både vores GUI og i vores kode og at vi derefter begyndte at arbejde på at finpudse vores program, så det ville være et fuldendt og gennemført program, der ville være nemt at bruge og installere.

Konklusion af sprint review – Mads

Det at have sprint reviews var meget vigtigt for vores iterative proces. Det var en god måde for os at komme videre i vores udviklings proces da det gav os et overblik over hvad vi manglede og hvad vi skulle have lagt mere fokus på i vores sprint og vores opkommende sprint.

Retroperspektiv om tekniks gennemgang 1 – Mads

Til vores første møde med den ”Technically proficient product owner” havde vi vores diagrammer; Domæne model, E/R Diagram og Relationelle skema med. Vores domæne model var noget vi havde haft mange problemer med og havde taget meget længere end vi havde regnet med til at starte med, da vi havde nogle problemer med om vi skulle have ressourcen til ressource type og så til ordren, eller om vi skulle have ressource type til ressourcen og så til ordren. Det skabte også meget debat mellem lærerne, men til sidst så blev det valgt at vi ville lave det på den måde at vi har en ressource type først, og så en ressource under. Da vi fik snakket med Henrik, virkede han generelt glad for vores arbejde, men der var nogle småting der skulle rettes her og der.

Retroperspektiv om tekniks gennemgang 2 – Mads

Til vores andet møde med Henrik, havde vi nogle problemer med vores kode. Hvordan det skulle virke, og hvor det skulle stå. Efter at have snakket med Henrik stod det alt sammen

klart, for hvordan vi skulle arbejde med vores kode fremover. Vi fik også vist vores kode til Henrik, som han generelt var glad for. Vi fik også nogle vigtige ting at vide, ang. hvordan vi skulle bruge metoder.

Retroperspektiv om tekniks gennemgang 3 – Mads

Til det tredje tekniske review møde, fik vi også mere eller mindre det samme at vide fra Product Owners side tidligere, som var at vi kunne nu bruge tiden på at prøve at tilføje så mange features som vi ville eller finpudse programmet. Hvis vi syntes det var tilstrækkeligt nok materiale og opfyldte krav. Vi valgte at der måske skulle laves lidt flere features i vores program, men vi ville bruge det meste af tiden på, at få lavet vores program mere finpudset og at vi skulle i gang med vores proces rapport. Det var vigtigt for os at vi havde et program der kunne bruges. Det skulle også være nemt for nye brugere, at blive sat ind i hvordan system virkede. Hvis programmet var blevet afleveret på daværende tidspunkt, ville der være mange forstyrrende elementer i den grafiske brugerflade og ikke nok feedback til brugerne, hvis der var bestemte krav for oprettelse af diverse ting der ikke blev overholdt.

Konklusion af teknisk gennemgang – Mads

De tekniske reviews har hjulpet os med, at holde projektet på sporet og åbne op for evt. idéer vi var fast låste på. Samtidigt har det givet os, muligheden for revaluere evt. design valg i programmet.

Evaluering af gruppens samarbejde – Mads

Da vi brugte scrum i vores projekt, var det vigtigt at vi fik uddelt roller ordentligt i gruppen. Så personer der havde ekspertise indenfor et bestemt område også kunne arbejde på det område. Det var dog også vigtigt at vi alle fik at vide hvad det var alle sad og arbejde på og at alt blev forklaret efter det var blevet lavet. Hvis ikke vi havde haft vores sprint backlog, hvor vi definitivt sagde hvem der skulle arbejde på hvad, kunne der muligvis have været meget rod i hvem der skulle arbejde på hvad, og hvornår det skulle være klart til.

Det var også vigtigt for os at hvis der var en person der var god til noget, såsom sql eller java, ville det ikke kun være hans ansvar at få den del af projektet færdigt, men at arbejde sammen med andre, så andre i gruppen der var mindre gode til det, kunne lære noget om den del af

projektet. At vi alle fik lov til at sige hvad vi syntes var også en vigtig pointe i vores udvikling da det gav os muligheden hvis alles synspunkt blev forstået under udviklingsprocessen.

At alle mødte op så ofte så muligt i klassen hjalp os også med at komme igennem projektet, da vi altid havde en nem mulighed for at snakke med hinanden og snakke om hvad vi syntes der manglede og hvad vi syntes at vi skulle have lavet for at komme videre, og så var det også en god ressource at vi kunne snakke med lærerne hver eneste gang vi sad fast i vores projekt, eller vi havde en design proces som vi ikke alle var helt enige om hvordan skulle udføres. Det var også en stor nødvendighed for os at vi fik brugt disse ressourcer ordenligt da det virkelig var en måde vi kunne få et produkt produceret som var så godt og så veludviklet som vi kunne fremstille i de dage og uger vi havde til rådighed.

Evaluering af brugbarheden af metoder, værktøjer og teknikker som blev brugt under projektet

SCRUM – Mads

Til det projekt brugte vi SCRUM, som er beskrevet som:" **Scrum** is an iterative and incremental agile software development framework for managing product development". Det var vigtigt for os at vi kom godt i gang med vores SCRUM og vi satte os derfor ned den første dag vi begyndte og fik lavet en god oversigt i Excel over vores user stories. Det var også vigtigt for os at vi fik en realistisk oversigt over hvor lang tid det ville tage at gennemføre hvert task, så vi nemmest kunne planlægge hvad vi skulle lave i vores sprint forløb. Den første dag af hvert vores sprints var meget vigtige i vores SCRUM forløb, da det var der hvor vi virkelig satte os ned og snakkede om hvor langt vi var kommet og hvor langt vi endnu havde tilbage for at få et produkt der både kunne fremvises til product owner men også var brugbart hvis du var en ny bruger. Det var også vigtigt for os at vi alle fik lov til at sige vores meninger om produktet og hvordan vi alle syntes at vores fremgang til den nye sprint skulle være.

Git – Stefan

Vi har valgt i projektet at benytte os af Git, som er et gratis opensource versionsstyrings værktøj. Til at hoste vores projekt har vi brugt GitHub, som er en hjemmeside som giver os muligheden til at have vores eget repository (projekt). Git giver os muligheden for at kunne samarbejde om vores fælles kode, dette inkludere at ændre, slette og tilføje nyt kode til

projektet. Git finder selv ud af om vi har lavet nogle ændringer i koden, som er forskellig fra den kode som ligger på GitHub. Samtidig vil Git også tjekke om koden som ligger på GitHub er nyere, end den kode som man selv har lavet ændringer i.

Dette har givet os muligheden til at kunne samarbejde om samme projekt, men arbejde i forskellige dele af koden. Dog har det betydet at vi i de fleste tilfælde har været tvunget til at kun arbejde i specifikke java klasser i selve koden. For at forhindre merge conflicts i Git.

Vi kunne have brugt Git bedre i vores projekt, hvis vi havde brugt mere tid på at lære værktøjet bedre at kende, samt dets mange funktioner. Dog skal det siges at Git er et værktøj som kan rigtig meget og, at lære alle dets funktioner vil tage meget mere tid end vi har til rådighed i projektet.

Vi ville kunne have forbedret vores arbejdsgang ved at benytte en funktion i Git som hedder Branching, det ville have givet os muligheden til at lave en hel kopi af projektet, som vi ville kunne arbejde på og implementere en ny funktion i koden. Sideløbende med vores hoved projekt. Når den nye funktion så var implementeret succesfuldt kunne man Merge det med vores hoved projekt. Dette ville man kunne have gjort for alle de nye funktioner vi implementeret i vores program.

Lucid Chart – Mads

Lucid Chart var et vigtigt program for os, da det hjalp os med nemt at kunne lave vores diagrammer på en måde hvor alle kunne være med til at redigere og nemt åbne dem hjemme fra og i skolen, på samme tid. Det var en god måde at få lavet vores vigtige diagrammer og få delt dem med hinanden.

3-lags arkitektur model – Mads

Tre lags modelen var modellen der blev brugt til at lave vores program, da det var den nemmeste og mest overskuelige, til et projekt af denne størrelse. Det betød at vi havde vores præsentations lag, med al vores GUI, vores data lag hvor alle vores data bliver gemt til vores database, og så vores logik lag hvor alle vores metoder lå.

jUnit test – Mads

Disse tests er med til at sikre metoderne i *OrderMapper* klassen gør som der forventes. Dette har været en stor hjælp i projektet især under implementeringen af samtidighed i programmet, det har flere gange været med til at spare tid på debugging i *OrderMapper* klassen, efter at en metode har været ændret.

Konklusion – Mads

Dette projekt har været en lærerig oplevelse. Det har været spændene at få lov til at arbejde som folk ude i den rigtige verden, inden for programudviklings-verdenen, ville arbejde med et projekt. Det har også været interessant at prøve at få lov til at arbejde med noget som SCRUM, da det var en meget interessant proces, som ingen af os havde prøvet før vi arbejde med den i det her forløb. Det var ret overraskende at prøve at arbejde på den her måde, og finde ud af hvor effektivt det egentlig var når man prøvede at finde ud af hvad alle ville arbejde på, hvor lang tid det ville tage, og så bruge den iterative proces at finde ud af hvor lang tid den næste del af processen vil tage, efter at have kigget tilbage på det tidligere forløb.

Der var visse ting der kunne gøres bedre, så som at planlægning måske var noget vi skulle have kigget mere på, så vi ikke løb ind i problemer senere hen i projektet, da der begyndte at opstå nogle problemer med programmet da vi skulle have lavet vores samtidighed, da vores program var blevet oversimplificeret til at starte med. Det betød at vi ikke kunne implementere vores samtidighed ordentligt, og der var nogle ting der skulle laves om.

Processen gik ellers godt, og det vi havde sat os for at lave i et sprint fik vi klaret, men vi fik desværre ikke opfyldt alle vores user stories, men vi fik lavet en basis for vores projekt hvor vi havde et program der kunne gennemføre en ordre og tjekke ressourcerne, osv. Hvilket virkelig var noget af det der var vigtigt at få lavet for os og så var det vigtigt at vi fik lavet et program som product owner blev glad for. Hver gang vi havde møde med product owner virkede han også glad for hvor langt vi var kommet med vores projekt og han virkede generelt glad for vores fremgang.

At have en ressource som en lære til at være der hver eneste dag til at hjælpe os var også en stor hjælp, da selv hvis vi ikke havde brug for det, vidste vi at der altid var en der kunne hjælpe os igennem de sværere opgaver, og der var altid en der kunne spørges om en mening,

om hvad vi havde lavet, og det er altid dejligt at få en mening fra en der ikke var med i gruppen, om hvad vi havde gang i og om vores fremgangsmåde var korrekt.

Hvis hele dette projekt skulle have været lavet uden noget som for eksempel scrum, kunne det have været en svær proces, da det at vi havde et ugentligt mål virkelig var en stor hjælp for at finde ud af hvor meget vi skulle nå inden for den tidsramme. Det kunne have virket meget uoverskueligt, og virkede det også, til at starte med hvis vi skulle til at lave det her program, over en måned, uden at få lov til at snakke med nogen og have nogen til at sige hvad de syntes om vores projekt indtil videre.

At få lov til at arbejde i en større gruppe var også en ny oplevelse, da det tidligere kun har været 2 personers grupper, og det betød at der var mange flere inputs og at vores produkt kunne laves større end vi tidligere har prøvet, hvilket også var en ny oplevelse, at få lov til at arbejde på et program af den størrelse og af den slags. Det har alt i alt været en meget interessant oplevelse at få lov til at arbejde i et team og at være en del af en iterativ proces som scrum har været, da det var en meget bedre måde at få lov til at prøve at lave et rigtigt produkt til en rigtig kunde. Det var dog ikke muligt at få hele programmet lavet færdigt i den tid vi havde til at arbejde på det, da der var for mange features vi havde tænkt os at have med i vores program. Vi kunne have valgt at lade vær med at finpudse vores projekt, og i stedet lagt flere features i det, men vi følte at det var mere vigtigt at få et program der havde lidt færre features, og i stedet ville vi have et program der var nemt at bruge, og der gav brugerne god feedback igennem vores GUI.

Alt i alt har det været en interessant proces og det har været en sjov proces at kunne få lov til at udvikle et program af denne størrelse, med de ressourcer, metoder og teknikker som vi har fået sat os til rådighed.

Oprettelse af database – Bilag

Bilag 1.1

```
DROP TABLE TeamOrderJoined;

DROP TABLE Teams;
DROP TABLE Orderdetails;
DROP TABLE Orders;
DROP TABLE Ressources;
DROP TABLE Storages;
DROP TABLE RessourceTypes;
DROP TABLE salesmen;
DROP TABLE Customers;
DROP TABLE zipCodes;


drop sequence orderseq;
create sequence orderseq start with 10;


CREATE TABLE zipCodes(

ZIP int PRIMARY KEY,
City varchar2(50)

);

CREATE TABLE Customers(

CustomerID int PRIMARY KEY,
FullName varchar2(50),
companyName varchar2(50),
adress varchar2(50),
ZIP int REFERENCES Zipcodes,
phoneNumber int

);


CREATE TABLE salesmen(

SalesID int PRIMARY KEY,
firstName varchar2(30),
```

```
lastName varchar2(30),
phoneNumber int

);

CREATE TABLE RessourceTypes(

RessourceTypeID int PRIMARY KEY,
typename varchar2(30)

);

CREATE TABLE Storages(

StorageID int PRIMARY KEY,
Adress varchar2(50),
ZIP int REFERENCES Zipcodes

);

CREATE TABLE Ressources(

RessourceTypeID int REFERENCES RessourceTypes,
StorageID int REFERENCES Storages,
qty int,
primary key (RessourceTypeID, StorageID)

);


CREATE TABLE Orders(
OrderID int PRIMARY KEY,
CustomerID int REFERENCES customers,
SalesID int REFERENCES salesmen,
Confirmed int, --BOOL 0/1
StartDate date,
EndDate date,
Price float,
Ver int

);

CREATE TABLE Teams(
TeamID int PRIMARY KEY,
```

```
Fitters int,
Trucks int
);



CREATE TABLE TeamOrderJoined(

OrderID int REFERENCES Orders ON DELETE CASCADE,
TeamID int REFERENCES Teams ON DELETE CASCADE,
PRIMARY KEY (OrderID, TeamID)

);

CREATE TABLE Orderdetails(

OrderID int REFERENCES Orders ON DELETE CASCADE,
RessourceTypeID int,
StorageID int,
qty int,
Ver int,
FOREIGN KEY( RessourceTypeID, StorageID )
  REFERENCES Ressources(RessourceTypeID, StorageID),
PRIMARY KEY (OrderID, RessourceTypeID, StorageID)
);



commit;
```

Bilag 1.2

CustomerMapper

```
"INSERT INTO Customers "
        + "VALUES (?,?,?,?,null,?)"

"SELECT * "
        + "FROM Customers "
        + "WHERE CustomerID = ?"

"SELECT * "
        + "FROM Customers "
```

DatabaseQueries

```
"SELECT sum(qty) FROM Orderdetails WHERE OrderID IN ("
        + "     SELECT orderID FROM Orders WHERE ("
        + "     (startDate>=? AND endDate<=?) or"
        + "     (endDate>=? AND endDate>=?) OR"
        + "     (startDate>=?) OR"
        + "     (startDate<=? AND endDate>=? AND startDate<=? AND endDate>=? )"
        + "     ) "
        + "     AND confirmed=1)"
        + "     AND RessourceTypeID = ?"

"SELECT sum(qty) FROM Ressources WHERE ressourceTypeID=?"

"SELECT sum(qty) FROM Orderdetails WHERE OrderID IN ("
        + "     SELECT orderID FROM Orders WHERE ("
        + "     (startDate>=? AND endDate<=?) or"
        + "     (endDate>=? AND endDate>=?) OR"
        + "     (startDate>=?) OR"
        + "     (startDate<=? AND endDate>=? AND startDate<=? AND endDate>=? )"
        + "     ) "
        + "     AND confirmed=1)"
        + "     AND RessourceTypeID = ? AND StorageID= ?"
"SELECT sum(qty) FROM Ressources WHERE ressourceTypeID=? AND storageID = ?"

"SELECT sum(qty), storageID, ressourceTypeID FROM Orderdetails WHERE OrderID IN ("
        + "     SELECT orderID FROM Orders WHERE ("
```

```
        + "     (startDate>=? AND endDate<=?) or"
        + "     (endDate>=? AND endDate>=?) OR"
        + "     (startDate>=?) OR"
        + "     (startDate<=? AND endDate>=? AND startDate<=? AND endDate>=? )"
        + "     ) "
        + "     AND confirmed=1)"
        + "     AND RessourceTypeID =? GROUP BY StorageID, ressourceTypeID ORDER BY
STORAGEID ASC"

"SELECT QTY, StorageID, RessourceTypeID FROM Ressources Where ressourceTypeID=?
GROUP BY StorageID, ressourceTypeID, QTY ORDER BY STORAGEID ASC"

"SELECT * FROM teams "
        + "WHERE Teams.Teamid NOT IN "
        + "(SELECT teamid from teamorderjoined WHERE Teamorderjoined.Orderid IN "
        + "(SELECT orderid from Orders where (startDate=? OR endDate=?)))"
"DELETE FROM teamorderjoined "
        + "WHERE orderid = ?"

"INSERT INTO Orders "
        + "VALUES (?,?,?,?,?,?,?,?)"

"UPDATE Orders "
        + " SET CustomerID = ?, SalesID = ?, Confirmed = ?, StartDate = ?, EndDate = ?, Price =
?, Ver = ? "
        + " WHERE OrderID = ? AND ver = ?"


"INSERT INTO Orderdetails "
        + "VALUES (?,?,?,?,?)"

"UPDATE Orderdetails"
        + " SET qty = ?, ver = ?"
        + " WHERE OrderID = ? AND ressourcetypeid = ?"
        + " AND storageid = ? AND ver = ?"

"SELECT * "
        + "FROM Orders "
        + "WHERE OrderID = ?"
```

OrderMapper

```
"SELECT DISTINCT od.RessourceTypeID, od.StorageID, od.qty, rs.qty, rst.typeName, od.ver "
        + "FROM Ressources rs, OrderDetails od, RessourceTypes rst "
        + "WHERE od.OrderID = ? AND od.RessourceTypeID = rst.RessourceTypeID and
Od.Storageid = rs.storageID and od.Ressourcetypeid= Rs.Ressourcetypeid"

"SELECT * "
        + "FROM Customers "
        + "WHERE CustomerID = ?"

"SELECT * "
        + "FROM Orders "

"select orderseq.nextval  " + "from dual"
```

ResourceMapper

```
"SELECT sum(qty), Rst.Typename, Rst.Ressourcetypeid FROM ressources rs, ressourceTypes
rst "
        + "WHERE rs.ressourceTypeID = rst.ressourceTypeID "
        + "GROUP BY Rst.Typename, Rst.Ressourcetypeid "
        + "ORDER BY Rst.Ressourcetypeid"

"SELECT sum(qty), Rst.Typename, Rst.Ressourcetypeid FROM ressources rs, ressourceTypes
rst "
        + "WHERE rst.ressourceTypeID = ? AND rs.ressourceTypeID = rst.ressourceTypeID "
        + "GROUP BY Rst.Typename, Rst.Ressourcetypeid "
        + "ORDER BY Rst.Ressourcetypeid"

"SELECT Rs.Ressourcetypeid, Rst.Typename,rs.StorageID, rs.qty   "
        + "FROM ressources rs, ressourceTypes rst WHERE Rst.Ressourcetypeid =
Rs.Ressourcetypeid ORDER BY Rs.Ressourcetypeid"

"SELECT Rs.Ressourcetypeid, Rst.Typename,rs.StorageID, rs.qty   "
        + "FROM ressources rs, ressourceTypes rst WHERE Rst.Ressourcetypeid=? AND
Rst.Ressourcetypeid = Rs.Ressourcetypeid ORDER BY Rs.Ressourcetypeid"

"INSERT INTO ressources "
        + "VALUES(?, ?, ?)"
```

TeamMapper

```
"SELECT * "
        + "FROM TEAMS "
        + "WHERE teamID = ?"

"SELECT * "
        + "FROM Teams "

"INSERT INTO TeamOrderJoined VALUES(?,?)"

"SELECT * FROM teams, teamorderjoined toj WHERE toj.teamid = teams.teamid AND
Toj.Orderid=?"
```

UnitofWorkProcessOrder

```
"LOCK TABLE " + tableName + " in exclusive mode"
```