02324 Videregående Programmering – CDIO Final

Projekt navn: CDIO-Final
Gruppe nr.: 12
Afleveringsfrist: Mandag den 24-06-2013
Denne rapport er afleveret via https://www.campusnet.dtu.dk (Derfor ikke underskrevet)

Rapport udarbejdet af:
Studie-nr., Fornavn(e), Efternavn

s010138, Fadi Abdul Halim 

s123185, Morten Lerkenfeld Henningsen 

s113733, Kristoffer Sørensen (Stedfortræder) 

s123162, Alexander Amer Hachach Sønderskov 

s123806, Jacob Honoré 

s124259, Jesper Vestergaard Mark (Formand) 

CDIO-final							
Timeregnskab							
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	Ialt
10-06-2013	Jesper	3	2	0,5			5,5
	Morten	2,5	1		1	1	5,5
	Jacob	3	1		1		5
	Alexander	3	2	0,5			5,5
	Fadi	2		1	1	1	5
	Kristoffer	3			2		5
11-06-2013	Jesper	1	3	2			6
	Morten	2	2	2			6
	Jacob	1	3	2		1	7
	Alexander	2	2,5	2			6,5
	Fadi	2	2	2		1	7
	Kristoffer	1	3	1	1		6
12-06-2013	Jesper		3			1	4
	Morten	1	3	2			6
	Jacob		2	2,5			4,5
	Alexander	1	4	2			7
	Fadi	1	3	2			6
	Kristoffer		2	2	2		6
13-06-2013	Jesper						0
	Morten		2	2	2		6
	Jacob		3	3			6
	Alexander		3,5	3			6,5
	Fadi	2	2	2			6
	Kristoffer		2	2	2	1	7
14-06-2013	Jesper	1	3		1		5
	Morten				3		3
	Jacob		4	2			6
	Alexander		4	2		0,5	6,5
	Fadi		4	2			6
	Kristoffer		2	2	2	0,5	6,5
17-06-2013	Jesper						0
	Morten		3	3			6
	Jacob		3	3			6
	Alexander		3	3,5			6,5
	Fadi		3	3			6
	Kristoffer		2	2	2		6
18-06-2013	Jesper	2	2	2		0,5	6,5
	Morten		2	2	2		6
	Jacob		3	3			6
	Alexander		3	3			6
	Fadi		3	2		0,5	5,5

	Kristoffer		2	2	2	0,5	6,5
19-06-2013	Jesper	1	2				3
	Morten		3		3		6
	Jacob		3	3			6
	Alexander	1	4	2			7
	Fadi		1	2	2	1	6
	Kristoffer		2	2	2		6
20-06-2013	Jesper		2	1	3		6
	Morten			1	5		6
	Jacob		3	3		0,5	6,5
	Alexander		3	3			6
	Fadi						0
	Kristoffer			1	5		6
21-06-2013	Jesper	1	1		1		3
	Morten				3	0,5	3,5
	Jacob		1	1	3		5
	Alexander		2,5	2			4,5
	Fadi			2	0,5	1	3,5
	Kristoffer			1	3	0,5	4,5
22-06-2013	Jesper				5,5		
	Morten				3,5		
	Jacob				4		
	Alexander				5		
	Fadi				4,5		
	Kristoffer				3		
23-06-2013	Jesper				3		
	Morten				3		
	Jacob				3		
	Alexander				3		
	Fadi				3		
	Kristoffer				3		
	Sum	36,5	124,5	96	98	12	323,5

Brugermanual

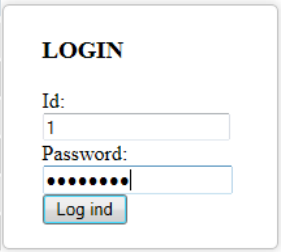
Projektet(.zip) skal importeres lokalt i Eclipse vha. "Import".

For at køre Afvejningsstyringsenheden (ASE), skal klassen 'Main' findes under pakken 'ase', og main() eksekveres som normalt. Herefter kan der enten testes vha. den tilhørende vægtsimulator fra pakken 'weight_simulator', som tilsvarende startes fra 'Main'. Fra ASE indtastes nu en tom linje som IP og 8000 som portnummer. Herefter er forbindelsen oprettet til vægtsimulatoren.
For at tilslutte den fysiske vægt indtastes vægtens IP-nummer og tilsvarende portnummer 8000 (ethernet-forbindelse er en forudsætning)

For at starte vores web-applikation findes klassen 'Servlet' under pakken 'web_interface' Denne klasse skal åbnes vha. 'Run As -> Run on Server' (Tomcat v7.0.5 Server forudsættes installeret)

For at komme ind i systemet kan bruges én af nedenstående login-oplysninger i LOGIN-skærmen

oprId operatør id	password password uden kryptering	aktoer 1 = admin / 2 = pharma / 3 = supervisor / 4 = operator
1	jespeR15	1
2	02324it!	2
3	02324it!	3
4	02324it!	4
13	Q4d_M+	3
15	8U8F.+	3



<u>Vores system består af følgende komponenter</u>
- Database (MySQL)
- Database Access Lag (java/JDBC)
- WEB applikation (Servlet)
- WEB GUI (JSP/html)
- Afvejningsstyringsenhed (ASE - java), som forbindes vha. TCP/IP med den fysiske Mettler-BBK vægt eller vægtsimulator
- TUI-baseret vægtsimulator (java)

For at tilgå backenden kan disse oplysninger om vores database bruges:
Host: maxgree.com
Brugernavn: 3ugers
Password: 43tBfUvQteyF5ytQ
Databasenavn: 3ugers
phpMyAdmin: http://maxgree.com/phpmyadmin

Poster

Technical University of Denmark

Developing a software system used for weighing drug batches with a web-based graphic user interface.

A fictive pharmaceutical company has requested this group to develop and update their current weighing system.
The pharmaceutical company currently uses several Mettler Toledo weight scales, which should, with interaction of a MySQL-based database, help the operators creating new drug batches. The company's employees are placed in different groups: administrators, pharmacists, supervisors and operators. They should have different rights in the web application.

Use Cases (Figure 1)

The diagram in Figure 1 shows which functions each kind of user has available. The different actors inherit the parent's rights shown with a generalization relationship.

From the beginning of the development process the group paid a lot of attention and discussion on the use cases, to be clear which different rights the users should have, and which complications these would give.

System architecture (Figure 2)

In this project the architecture of the system consists of several components. In figure 2 you will see an outline of the system and how it interacts between each core component. When looking at the database there are two different sub-systems, independent of each other, that share the database and use MySQL for queries. At your right you can see our weight control unit, which communicates with a weight simulator or a real physical weight. At your left we have our servlet, which controls our jsp-pages and creates our web-application. Additionally, we have specified with which network protocol each component communicates with each other. We drew the sketch of the overall architecture of the system in order to create a good overview of the system. Another intention with the sketch was to give the customer a good feeling and an idea of how the system could be structured.

Hierarchy

Operators should only be able to use the weight terminal to weigh new drug batches.
The supervisor inherits the operator's rights, and is as well able to create and show drug batches and basic material batches.
The pharmacist is able to manage basic materials and formulas and inherits the supervisor's rights
The administrator manages all the users of the system and inherits all the other rights, thus he is a super-user.

Snap of the code (Figure 3)

In figure 3 there is a section of our code from the class 'Sequences' which is a part of our weight control unit. The third sequence is where the operator enters the drug batch number and afterwards you see two `if`-statements. These checks are supposed to make sure that you enter a valid drug batch number. The second `if`-statement makes sure that the drug batch is not already processed, by checking that the status number is not 2, which would mean it is already processed.

Figure 4

Web interface (Figure 4)

In figure 4 you will see a screenshot from our web application interface. After logging in this will be the menu that you see from the system administrators view. If we had logged in as a pharmacist, there would have been fewer options but otherwise the same design. Compared to the administrator the pharmacists would have one less option, which would be the ability to manage users in this case. Continually throughout the program we have made sure that no matter which input you will give the system, the program will not crash, but instead give the user appropriate error messages.

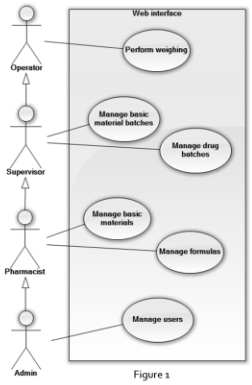


Figure 1

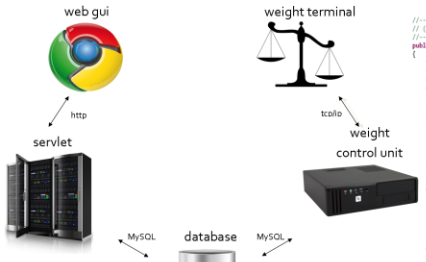


Figure 2

```
//-----------------------------------------------------------------
// (3)  Operatør indtaster produktbatchnummer
//-----------------------------------------------------------------
public void sequence3(BufferedReader inFromServer, DataOutputStream outToServer) throws IOException, DALException
{
    this.RPPrintOgRead(20, 8, "Indtast produktbatchnr.", inFromServer, outToServer);
    this.tjekHvisFejlOgRead(3, inFromServer, outToServer);
    checkForCancel(inFromServer, outToServer);
    data.setPbID(this.splitInt(inFromServer, outToServer));

    // Ikke eksisterende PbID.
    if (!func.testPbId(data.getPbID())){
        this.RPPrintOgRead(49, 2, "Dette produktbatch ID eksisterer ikke!", inFromServer, outToServer);
        this.tjekHvisFejlOgRead(3, inFromServer, outToServer);
        data.setServerInput(inFromServer.readLine());
        this.tjekHvisFejlOgRead(3, inFromServer, outToServer);
        this.sequence3(inFromServer, outToServer);
    }

    // Allerede afsluttet PB (status=2).
    else if(mPb.getProduktBatch(data.getPbID()).getStatus() == 2){
        this.RPPrintOgRead(49, 2, "Produktbatch allerede afsluttet!", inFromServer, outToServer);
        this.tjekHvisFejlOgRead(3, inFromServer, outToServer);
        data.setServerInput(inFromServer.readLine());
        this.tjekHvisFejlOgRead(3, inFromServer, outToServer);
        this.sequence3(inFromServer, outToServer);
    }
    else
        data.setReceptID(mPb.getProduktBatch(data.getPbID()).getReceptId());
    data.setListen(mReckComp.getReceptKompList(data.getReceptID()));
    this.sequence4(inFromServer, outToServer);
}
```

Figure 3

Gruppe 12
Kursus: 02324
Juni 2013

Jesper Mark, **s124259**
Alexander Sønderskov, **s123162**
Morten Henningsen, **s123185**

Kristoffer Sørensen, **s113733**
Jacob Honoré, **s123806**
Fadi Halim, **s010138**

Indholdsfortegnelse

Indledning

Denne arbejdsgruppe har modtaget en opgave fra en fiktiv medicinalvirksomhed om at udvikle og opdatere deres nuværende afvejningssystem med tilhørende web-interface, der både skal kunne bruges som personaleadministration samt lagerstyring.
Virksomheden har vedlagt en kravspecifikation, som gruppen har diskuteret og arbejdet ud fra. Fra start har det stået fast, at virksomheden har nogle klare krav og forventninger til hvad systemet skal ende som.
Specifikt har virksomheden nogle forretningsregler og en afvejningsprocedure, som skal overholdes i vores system. Derudover er der også et krav om, at operatørerne, som foretager selve afvejningsprocessen, ikke skal afvige fra deres normale rutine med dette nye opdaterede system.
Der er naturligvis også punkter og områder, hvor kravene er mere uspecifikke, hvor gruppen har lavet en vurdering og prioritering af, hvad tidsrummet tillader at implementere. Krav- og analysedelen udføres normalt i tæt samarbejde med kunden, men med en fiktiv kunde, har vi måttet ty til antagelser og interne diskussioner om nogle mål.
Vi har haft to obligatoriske milepæle i løbet af projektet, hvor vi, ved den første milepæl, præsenterede vores projektplan og dermed vores strategi for perioden for "udviklingschefen og sælgeren". Ved den anden milepæl præsenterede vi vores system for "slutbrugeren/kunden", hvor vi fik feedback på evt. forbedringer.

Projektplan

Prioritering af krav

Gruppen vil fokusere på kernen af systemet fra start, dvs. at der vil fokuseres på at oprette en database, så der kan kommunikeres med denne vha. MySQL-forespørgsler og anvendelse af Data Transfer Objects (DTO) og Data Access Objects (DAO).

Vi vil desuden fokusere på at få oprettet og testet, at de enkelte aktører har de relevante rettigheder og ligeledes, at der kan udføres en korrekt afvejningsproces jf. opgaveformuleringens use case 6.

Gennem udviklingsprocessen vil gruppen undervejs drøfte hvilke alternative flows, der skal lempes.

Som udgangspunkt vil vi se bort fra implementation af et GUI-baseret interface af vores vægtsimulator, da vi har vægtet de funktionelle krav højere. Derimod vil vi have stor fokus på at lave et brugervenligt TUI med gode og korte kommentarer samt relevante fejlmeddelelser.

Arbejdsfordeling i projektgruppen

Planen er at nedbryde opgaven i mindre delopgaver, så det er muligt at uddele de enkelte delopgaver til den enkelte eller i 2/3-mands grupper. Der sørges for, at den enkelte får forskellige og afvekslende opgaver gennem perioden, så man får prøvet forskellige arbejdsområder.

Vi vil hele tiden være i kommunikation med hinanden i gruppen, så vi kan få koordineret vores arbejde, og så vi kommer hurtigst muligt fremad i arbejdsprocessen uden at miste overblikket.

Overordnet strategi

Vores overordnede arbejdsstrategi vil være at mødes på DTU alle hverdage fra kl.9-16 i 325.117, hvor vi kort samler op på, hvor langt gruppen er nået med de enkelte delopgaver, og derefter tager en fælles beslutning om, hvad der i løbet af dagen skal arbejdes videre med.

I denne forbindelse er der oprettet et regneark, som er delt med alle vha. Google Docs, hvor vi har opstillet vores overordnede plan med de obligatoriske milepæle, men også med vores egne interne milepæle. Vi kan dermed hele tiden kan se, hvor langt vi er nået, i forhold til hvor langt vi har forventet at skulle være. Derudover har vi, fra start, oprettet en "To-do liste", hvor vi løbende har streget de enkelte delopgaver ud, når de er blevet gennemført og testet, så vi hele tiden har haft styr på, hvad der har manglet.

Ligesom i vores tidligere CDIO-delopgaver anvendes GitHub som versionsstyringsværktøj under implementeringen, hvilket vi har positive erfaringer med.

Der er oprettet en Facebook-gruppe, så vi har ét sted, hvor al intern kommunikation foregår, for på den måde at sikre os, at alle får generel information at vide. Derudover har vi anvendt Dropbox til fildeling.

Den overordnede og fælles strategi for at opnå det bedst mulige og mest robuste system, er at få testet det nyeste implementerede, så der ikke, når deadlinen nærmer sig, opstår fjollede taste- og tænke fejl, som kunne være undgået længere tid forinden.

Analyse

Ud fra opgavebeskrivelsen har vi, fra start af, kunne opskrive nedenstående funktionelle krav.

Funktionelle krav til vores web-applikation

- Brugeren skal kunne logge ind med id og password.
- Brugeren skal kunne oprette, slette, vise og opdatere andre brugere af systemet.
- Brugeren skal kunne oprette, vise og opdatere råvarer.
- Brugeren skal kunne oprette og vise recepter.
- Brugeren skal kunne oprette og vise råvarebatches.
- Brugeren skal kunne oprette og vise produktbatches.
- Systemet skal kunne give den enkelte bruger rettigheder afhængig af brugerens log-in.
- Systemet skal kunne holde styr på aktuelle mængder i alle råvarebatches.

Funktionelle krav til vores afvejnings styrings enhed (ASE)

- Brugeren skal kunne foretage en korrekt afvejningsprocedure jf. bilag 4 i opgavebeskrivelsen.
- Systemet skal kunne holde styr på aktuelle mængder i alle råvarebatches.

Non-funktionelle krav

- Pålidelighed: "Vores program skal ikke kunne crashe på noget tidspunkt."

Dette har vi sat, som et målbart krav for gøre vores system så robust, som muligt over for diverse fejl-inputs og dermed et fokus på inputvalidering.

- Brugervenlighed: "God og fornuftig interaktion mellem bruger og system."

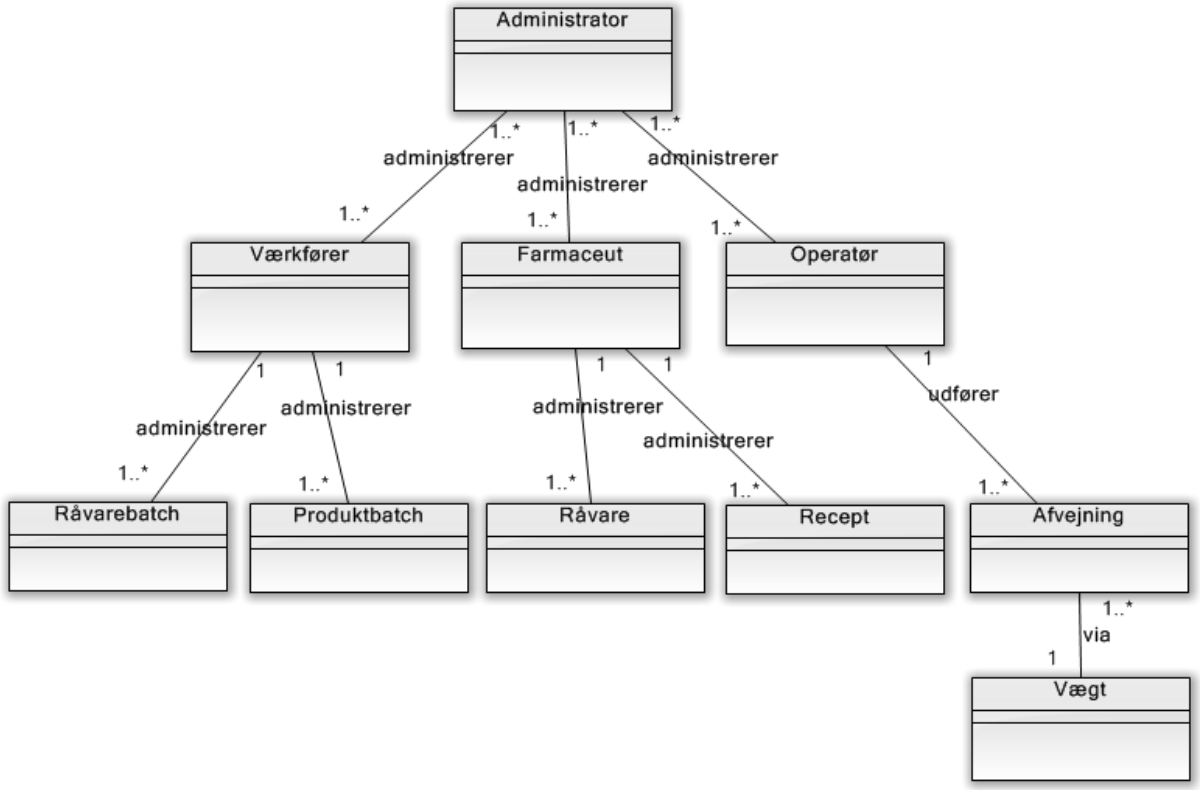
Dette vil vi have et stort fokus på, da der jf. opgavebeskrivelsen også er et stort fokus på brugbare fejlmeddelelser, hvis f.eks. brugeren indtaster noget forkert. Det er dog svært at gøre dette til et målbart krav, da folk har individuelle opfattelser af, hvad der er brugervenligt, men vi har diskuteret os frem til nedenstående fokuspunkter, for at gøre vores system så brugervenligt, som muligt:

Sikring af brugervenligt-interface

- Have en let forståelig dansk grænseflade
- Give god og forståelig respons på fejl, som brugeren kan bruge til noget
- Det skal være nemt at navigere rundt
- Brugeren skal altid kunne gå tilbage til hovedmenuen
- Vores tekst skal være opbygget af korte præcise ord/sætninger
- Der skal desuden kun vises de nødvendige informationer og valgmuligheder afhængig af aktør og situation

Domænemodel

Denne model har vi lavet for at skabe et overblik over alle fysiske komponenter dvs. indeholder navneordene i hele vores system.

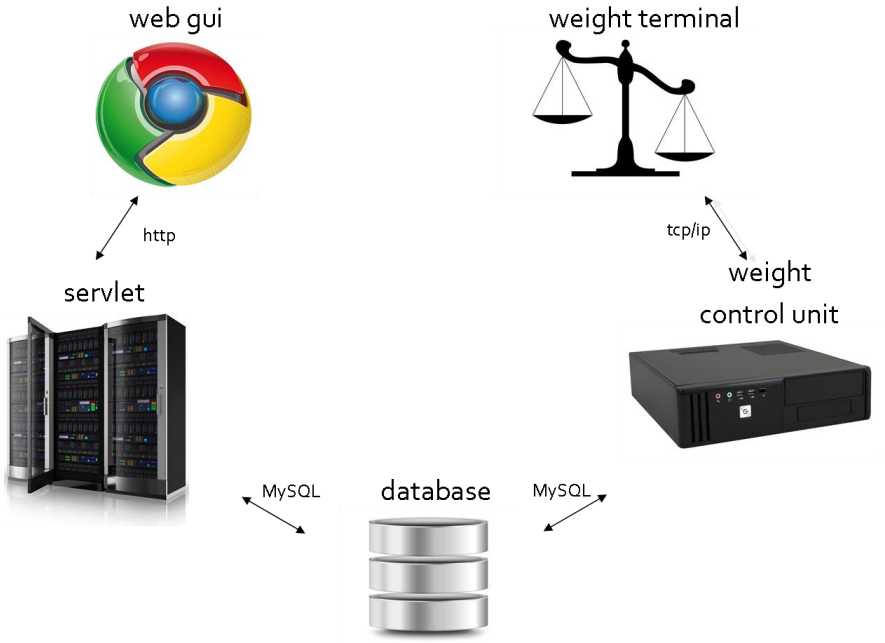


Figur 1 - Domænemodel

Ordet "administrerer" betyder, at en bruger i hvert fald kan vise og oprette samt i nogle af tilfældene slette og opdatere en specifik ting.

Vores figur viser ikke, at man f.eks. som værkfører også kan udføre en afvejning og at en farmaceut også kan administrere råvare- og produktbatch. Dette er muligt jf. opgaveformuleringens beskrivelse af systemets aktører, men vi har udeladt at vise dette i denne model for at bibeholde overblikket og ikke skabe mere forvirring med flere pile.

Rigt billede



Figur 2 - Rigt billede

Ovenstående billede viser, hvordan de enkelte komponenter i vores system hænger sammen.
Ud fra vores database er der ligesom 2 retninger af vores system.
Til højre er vores ASE, som kommunikere med enten vores egen vægtsimulator eller den fysiske vægt.
Til venstre er vores servlet, som styrer vores jsp-sider og danner vores web-applikation.
Derudover har vi angivet, hvordan de enkelte komponenter kommunikerer med hinanden.
Dette billede har vi lavet for at skabe et overblik over hele vores system, så kunden kan få en fornemmelse af, hvordan vores system helt overordnet er struktureret.

Aktørbeskrivelser

Her har vi listet de forskellige aktører og de muligheder, som den enkelte aktør skal kunne i vores system:
(Forudsætning: alle skal kunne logge ind)

- Operatør:
  - Foretage afvejning
- Værkfører:
  - (Det samme som ovenstående) +
  - Administrere råvarebatch
  - Administrere produktbatch
- Farmaceut:
  - (Det samme som ovenstående) +
  - Administrere råvare

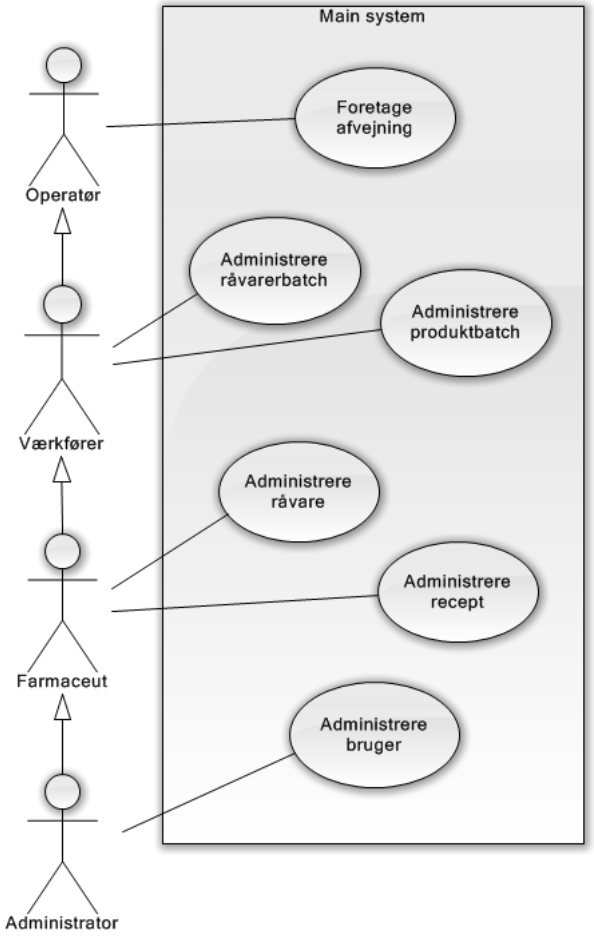
Administrere recept

- Administrator:

(Det samme som ovenstående) +
Administrere bruger

Ovenstående aktørliste leder os frem til nedenstående use case diagram.

Use case diagram



Figur 3 - Use-case diagram

På ovenstående billede illustreres, hvilke funktioner henholdsvis en operatør, værkfører, farmaceut og administrator hver især skal kunne i vores program.
Vi anvender nedarvning blandt aktørerne for ikke at skulle gentage de enkelte use cases, da f.eks. en administrator skal kunne alle de ovenstående funktioner samt oprette nye brugere af systemet.
I dette diagram viser vi ikke use casen, som alle brugeren skal igennem for at kunne anvende systemet, nemlig logge ind med bruger id og password.

Use case beskrivelser

Use case: UC1, Bruger administration
Scope: Afvejningssystem til medicinalvirksomhed
Primær aktør: Administrator (admin)
Stakeholders:

- Admin: Ønsker et brugervenligt og effektivt interface
- Admin: Ønsker et robust system ved forkert anvendelse
- Admin: Vil hellere have et solidt og anvendeligt system, end finurlige og grafisk effekter på de forskellige sider

Precondition: Admin er logget ind på systemet med sit ID og password gennem login-siden, og ser nu admin-forsiden.
Basic Flow:

1. Admin trykker på knappen 'Administrer brugere'
2. System præsenterer de fire CRUD-knapper
3. Admin trykker på 'Opret bruger'
4. System præsenterer 'Opret bruger' side, hvor admin skal indtaste en formular
5. Admin indtaster navn, initialer, cpr-nummer og angiver aktør-niveauet, og trykker 'Opret bruger'
6. System opretter bruger i databasen og viser den nye brugers password
7. Admin trykker "Tilbage til hovedmenu"
8. System præsenterer hovedmenuen "Admin side"
9. Admin trykker "Log af"
10. System logger brugeren af og præsenterer "Login" siden.

Precondition: Admin er logget ind på systemet med sit ID og password gennem login-siden, og ser nu admin-forsiden.
Alternative Flow 1:

1. Administrator trykker på knappen 'Ændr Password'
2. System præsenterer 'Ændr password' side
    a. Admin skriver gammelt password forkert
        i. System giver besked "Dit gamle password er forkert!"
    b. Nyt password ikke skrevet identisk to gange i træk
        i. System giver besked "De to nye password er ikke identiske!"
    c. Admin skriver gammelt password og nyt password korrekt
        i. System giver besked "Password er ændret!"
    d. Admin skriver for mange tegn eller opfylder på anden måde ikke password kravene

i. System giver besked "Dit nye password er ikke godkendt ud fra nedenstående regler!

e. Admin indtaster password i overensstemmelse med forretningslogikken
   i. System opretter det nye password i databasen.

Precondition: Admin er logget ind på systemet med sit ID og password gennem login-siden, og ser nu admin-forsiden.

Alternative Flow 2:

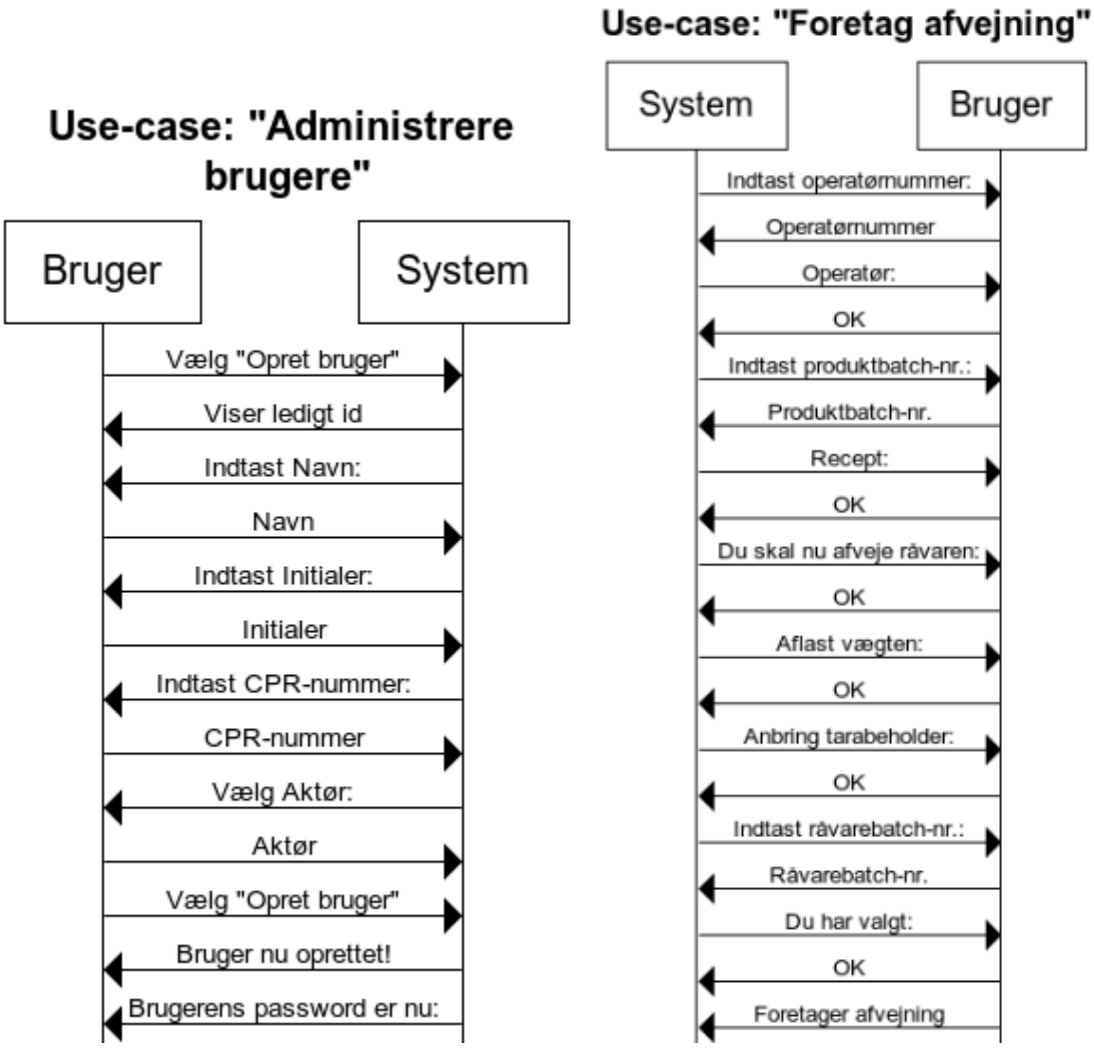
1. Admin trykker "Log af"
2. System logger brugeren af og præsenterer "Login" siden.

Precondition: Admin er logget ind på systemet med sit ID og password gennem login-siden, og ser nu admin-forsiden.

Alternative Flow 3:

1. Admin trykker på knappen 'Administrer råvarebatches'
2. System præsenterer 'Råvarebatch administration' side
3. Admin trykker på knappen 'Opret råvarebatch'
3. System præsenterer 'Opret råvarebatch' side
   a. Admin indtaster et allerede eksisterende Råvarebatch ID
      i. System giver besked "Råvarebatch ID findes i forvejen!"
   b. Admin glemmer at indtaste mængden og efterlader feltet tomt
      i. System giver besked "Fejl i råvarebatchid, råvareid eller mængden, muligvis specialtegn!"
   c. Admin indtaster Råvarebatch ID forkert med enten specialtegn eller lignende
      i. System giver besked "Fejl i råvarebatchid, råvareid eller mængden, muligvis specialtegn!"
   d. Admin indtaster et nyt Råvarebatch ID, et tilfældigt Råvare ID og en mængde af hvilken som helst størrelse
      i. System giver besked "Råvarebatch oprettet!"
   e. Admin indtaster et meget langt Råvarebatch ID
      i. System giver besked "Fejl i råvarebatchid, råvareid eller mængden, muligvis specialtegn!"
   f. Admin indtaster alle oplysninger i overensstemmelse med kravene
      i. System giver besked "Råvarebatch oprettet!"

Systemsekvensdiagrammer



Figur 4 og 5 - Systemsekvens-diagrammer

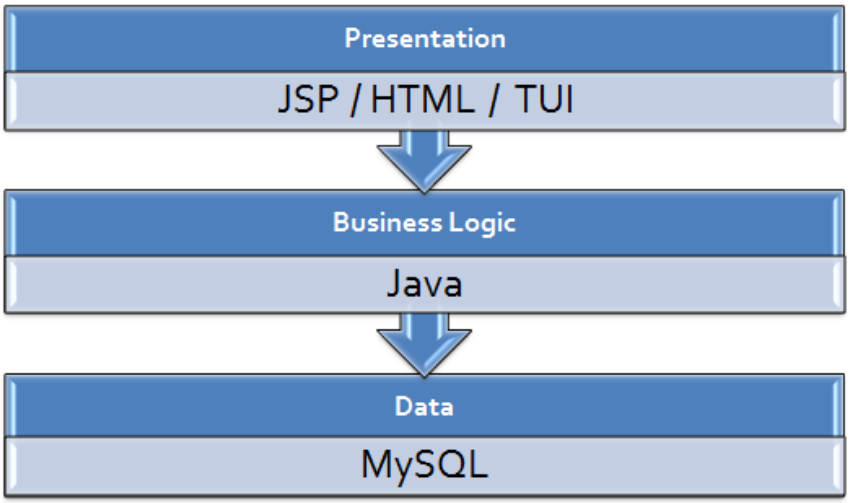
Ovenstående systemsekvensdiagram til venstre viser, hvordan interaktionen i vores webapplikation er mellem bruger og system ud fra use casen: "Opret bruger", som hører under den overordnet use case: "Administrere brugere" jf. vores use case diagram.

Vi har udarbejdet et andet systemsekvensdiagram til højre, som igen viser interaktionen mellem bruger og system, men nu ud fra use casen: "Foretag afvejning".
Afvejningsproceduren fortsættes på samme måde for alle de råvarer, som der indgår i den valgte produktbatch.

Design

I dette kursus har vi lært om fler-lags-arkitekturen (specifikt 3-lags-arkitekturen), hvorfor vi finder det naturligt at implementere et system fra denne. Det øverste lag, grænsefladen

3-lagsmodellen hjælper os med at adskille logikken fra data og grænseflade med interfaces, hvor fordelen er at koden vil afkobles, som dermed bliver lettere at vedligeholde og ændre i. Derudover øger det overskueligheden og skaber bedre mulighed for genbrug af kode i andre sammenhænge.



Figur 5 - 3-lags arkitektur

ASE

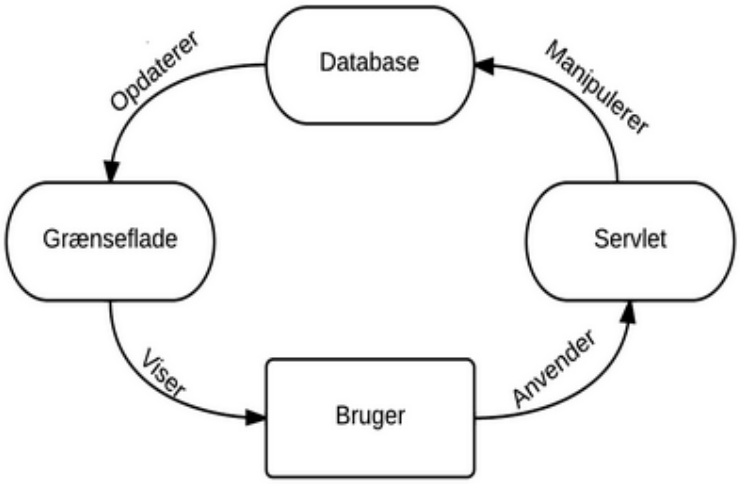
I designet af ASE valgte vi at tage udgangspunkt i CDIO delopgave 2 fra kurset Datakommunikation. Med inspiration fra den delopgave har vi her valgt at opdele programmet i sekvenser, sådan at der var fokus på en gennemgang step-by-step af hele processen.
I vores tidligere opgave havde vi i starten en del variabler. I starten havde vi blot disse variabler gennem hele programmet, men vi besluttede os for at lave det om til metoder og placere dem i en klasse for sig selv, kaldet "Data". De blev således kaldt i de enkelte relevante sekvenser i det forrige program.
Det er også sådan, at det fungerer i vores program nu, men vi har slået hjælpeklassen "data" og en anden hjælpeklassen "Other" sammen til "SequenceHelper", hvor vi har en række variabler og metoder etc. Grunden til denne ændring var, at det ville give mere mening og overblik. Princippet fra vores tidligere opgave kunne dermed genbruges og blev således et skelet for vores ASE.

Vores ASE består således af tre klasser: Main, Sequences og SequenceHelper, som alle ligger i pakken "ase". Vores main-klasse opretter forbindelse til vægten vha. en TCP-socket og derefter kaldes metoden sequence1, som er vores opstarts sekvens. Sequences-klassen indeholder metoder for hver sekvens jf. opgaveformuleringens afvejnings-successcenarie. Vi har delt

klassen op i sekvenser, da det har overskueliggjort afvejningsprocessen væsentligt. Derudover er der knyttet kommentarer til hver sekvens, så det specificeres, hvad der sker netop i denne metode.

Webarkitektur

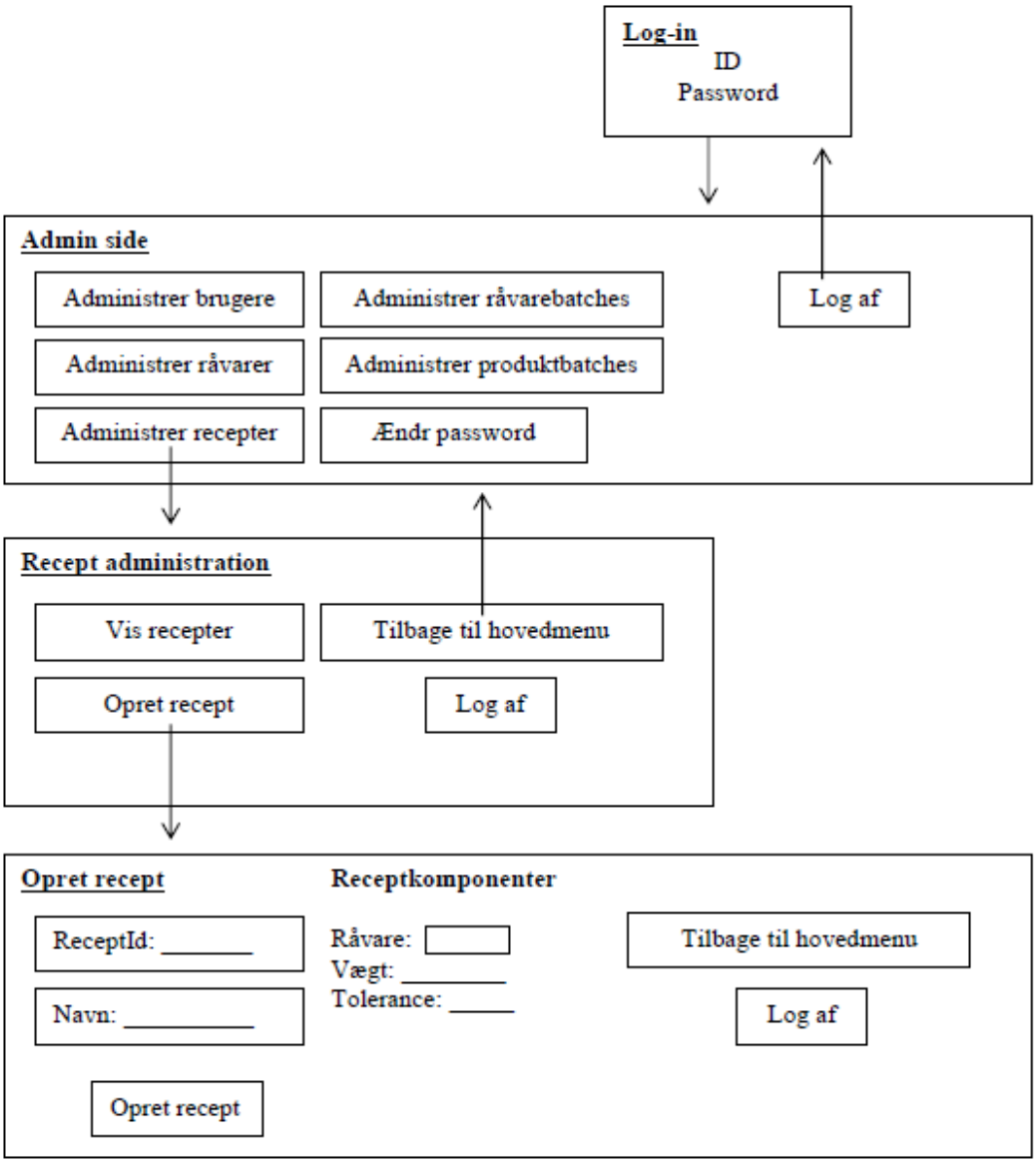
Vi har taget udgangspunkt i designprincipperne i 3-lags modellen, som under webprogrammering er nærmere defineret i en model 1 eller 2-arkitektur. Herunder ses den anvendte model 2 arkitektur.



Teori:		Praksis:
Model	=	Database
View	=	JSP sider
Controller	=	Servlet

Figur 6 - Model 2 arkitektur

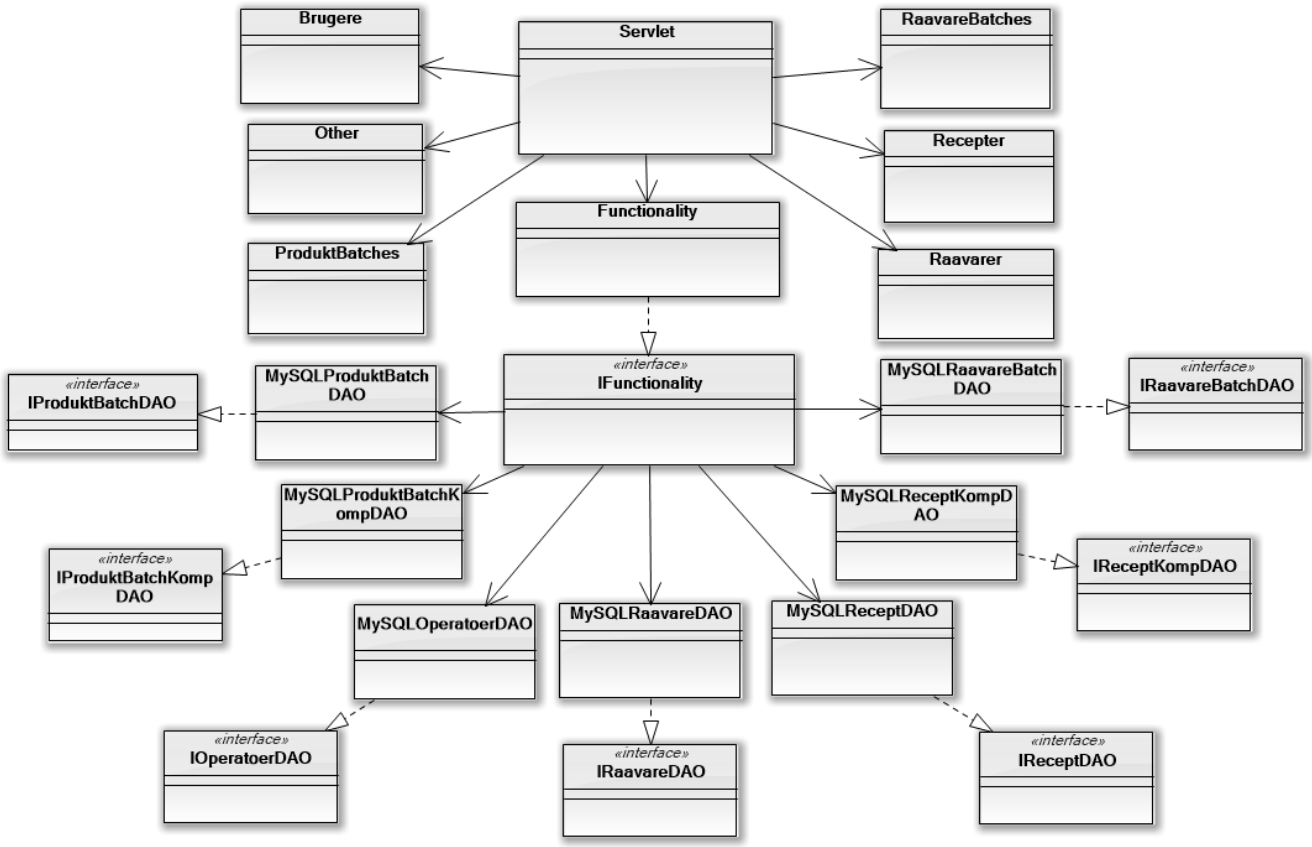
Vi har valgt at lave web-programmeringen ud fra en model 2-arkitektur, hvor vi har java-koden adskilt fra HTML i vores jsp-sider. Al afvikling, forretningslogik og databasestyring vil ske i back-enden, og præsentationen vil foregå i front-enden i .jsp-siderne.
Dette overskueliggør således programmet og er lettere at danne et overblik over og senere vedligeholde, i forhold til model 1-arkitekturen. Model 2 er også kendt som "Model-View-Controller"-arkitekturen, som netop er et designmønster hvor man bruger en centraliseret styring gennem en controller. Denne controller er vores servlet i web-applikationen og den håndterer, hvilke jsp-sider der vises ud fra brugerens valg. En servlet øger i høj grad sikkerheden i ens program, da der bliver sørget for at alt input valideres i vore servlet, hvorefter der så tages stilling til, hvad der efterfølgende skal ske. Dette har vi set, som en stor fordel at implementere i vores system.



Figur 7 - webarkitektur

Ovenstående billeder viser et konkret eksempel på, hvordan vores web-applikation med jsp-sider er bygget op. Som bruger skal man altid logge korrekt ind først og derefter afhænger ens valgmuligheder af ens rettigheder, som er defineret ud fra ens rolle i databasen.
Nedenstående eksempel viser, når man er logget ind som administrator og har alle valgmuligheder, hvor der er derefter er valgt administrering af recepter og derefter oprettelse af recept.
Pilene indikerer, hvor man som bruger, bliver dirigeret hen , når man vælger den pågældende knap.
Dette eksempel viser vores generelle opbygning af jsp-siderne, da vores web-applikation er bygget op ud fra samme struktur.

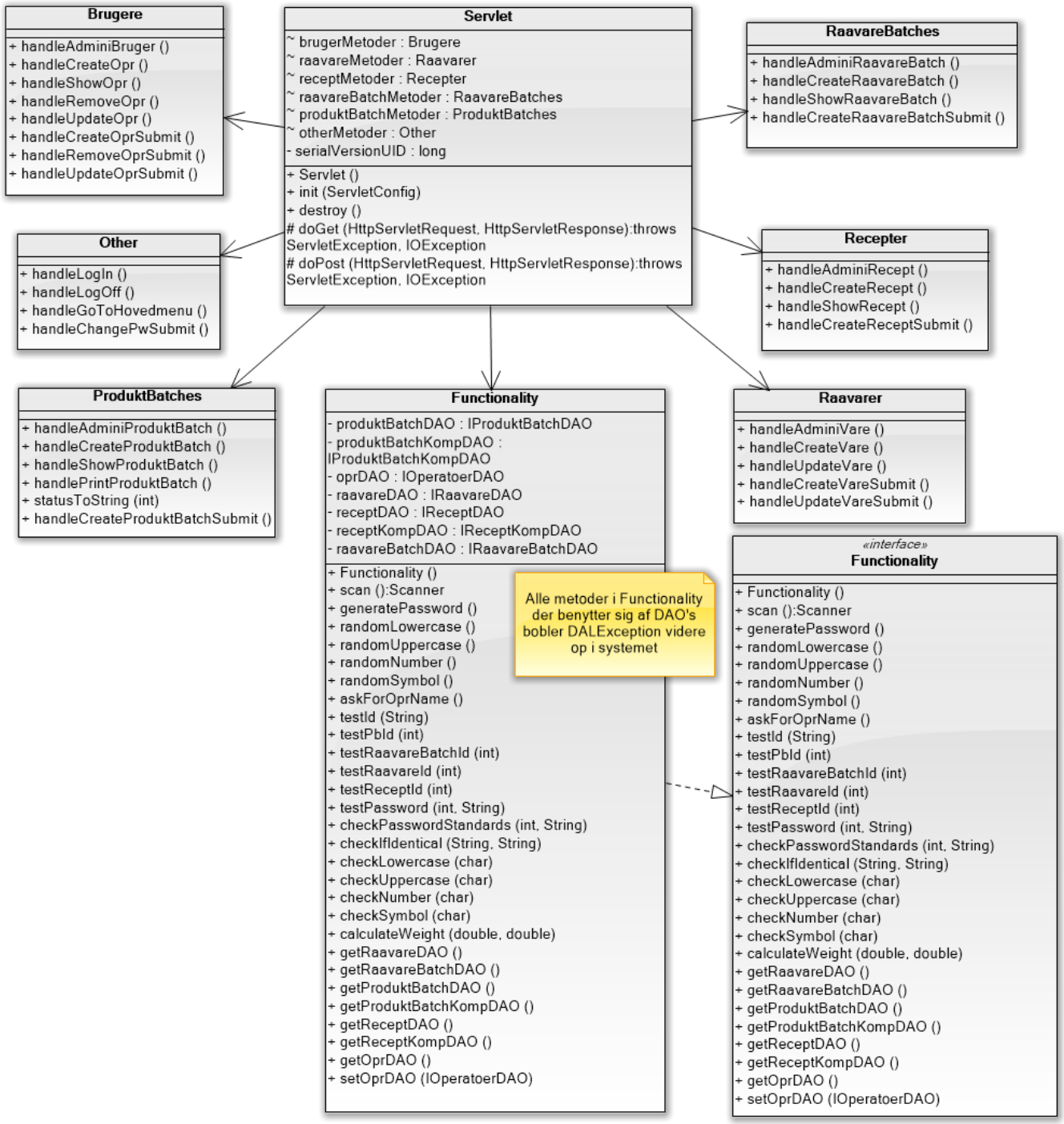
Designklassediagram af Servlet - overview



Figur 8 - Designklassediagram Servlet overview

Ovenstående diagram viser helt overordnet, hvordan vores servlet er helt central i vores web-applikation og fungerer, som vores controller. Vores servlet anvender diverse hjælpeklasser, som f.eks. klassen "Brugere" og klassen "Recepter", som indeholder relevante metoder afhængig af brugerens situation og valg.
Derudover anvender vores servlet klassen "Functionality", som implementerer interfacet "IFunctionality" med diverse tomme hjælpemetoder, som så defineres og anvendes i "Functionality".
Det ses desuden, hvordan interfacet "IFunctionality" anvender vores forskellige DAO'er, hvor vores SQL-kald til databasen foretages. Disse DAO klasser implementerer desuden også hvert sit interface, som det ses i diagrammet.

Designklassediagram af Servlet - in-depth



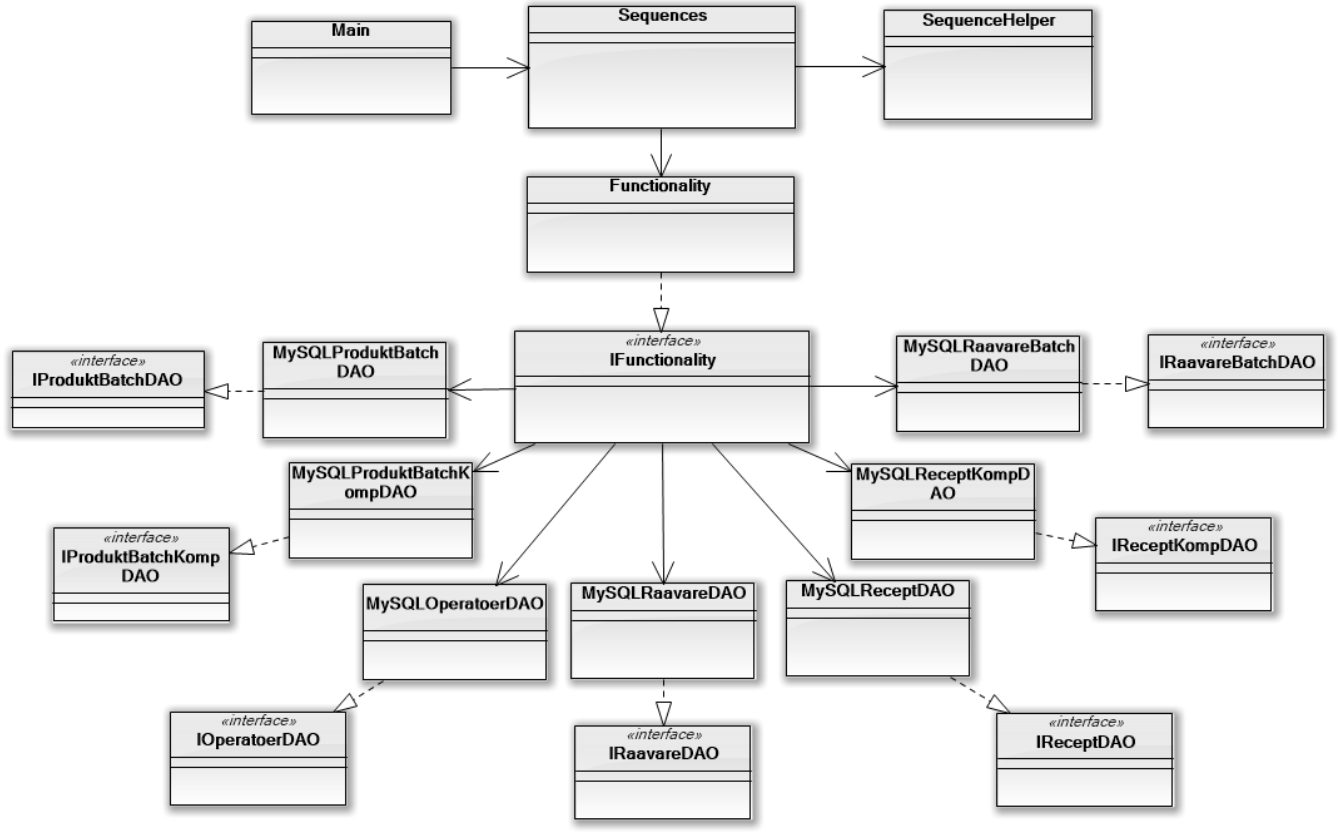
Figur 9 - Designklassediagram servlet in-depth

På ovenstående diagram ser vi nu nærmere på de væsentligste klasser i vores web-applikation, hvor vi både kan se klassernes attributter og metoder. Det kan tydeligt ses, at vores

funktionalitets-klasse indeholder de fleste metoder, hvilket giver god mening, da det er her, hvor al logikken foretages.

Afvejnings Styrings Enhed (ASE)

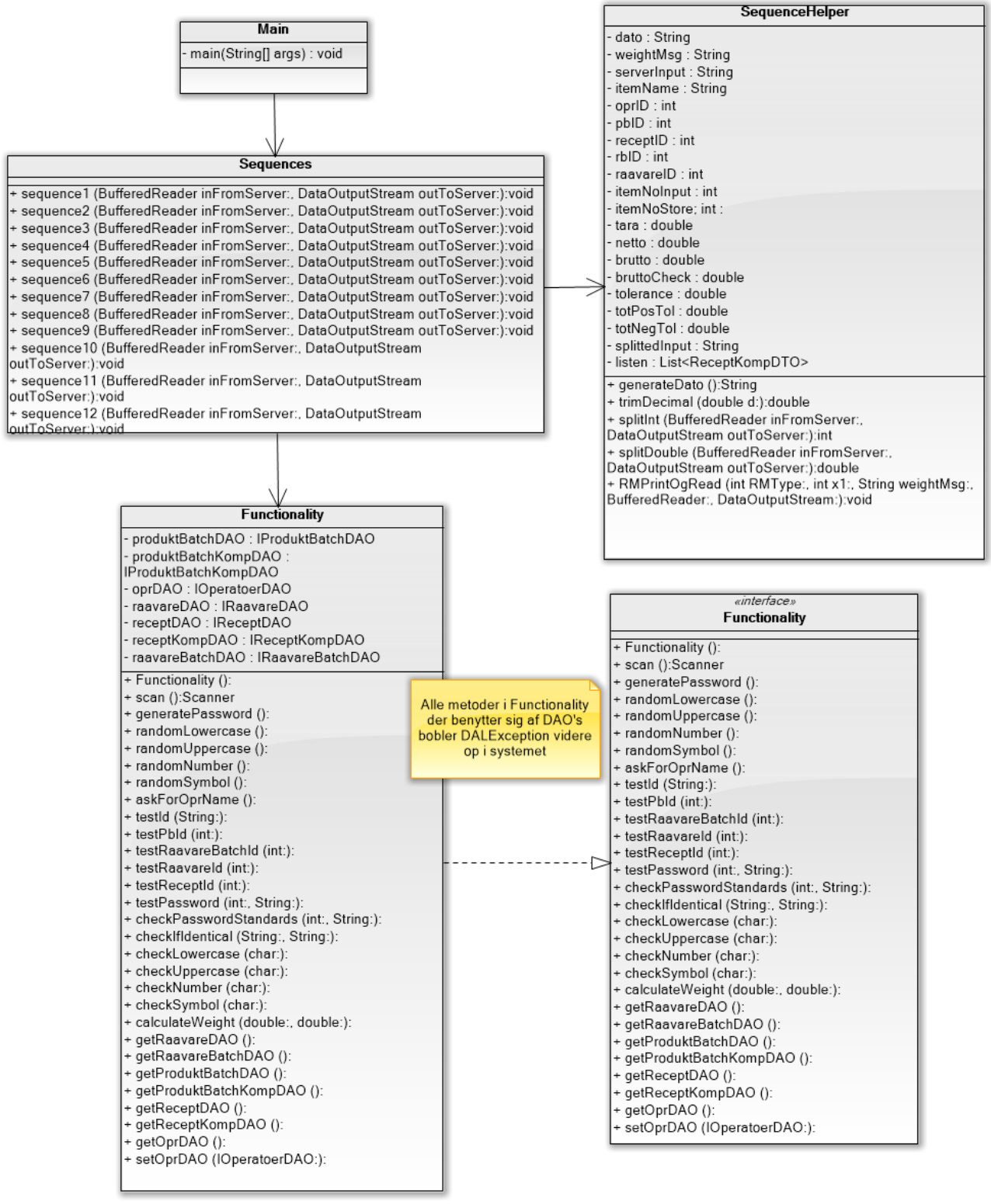
Designklassediagram af ASE - overview



Figur 10 - Designklassediagram ASE overview

Ovenstående diagram viser et overordnet overblik over, hvordan vores ASE er designet og hvilke klasser, som den hænger sammen med. Vores centrale klasser er her "Sequences", som anvender metoderne i hjælpeklassen med det passende navn "SequenceHelper". Derudover ses det, at "Sequences", ligesom vores servlet, anvender metoder fra klassen "Functionality", som stadig implementerer interfacet "IFunctionality" osv. ligesom beskrevet i vores tidligere designklassediagram over vores servlet.

Designklassediagram af ASE - in-depth

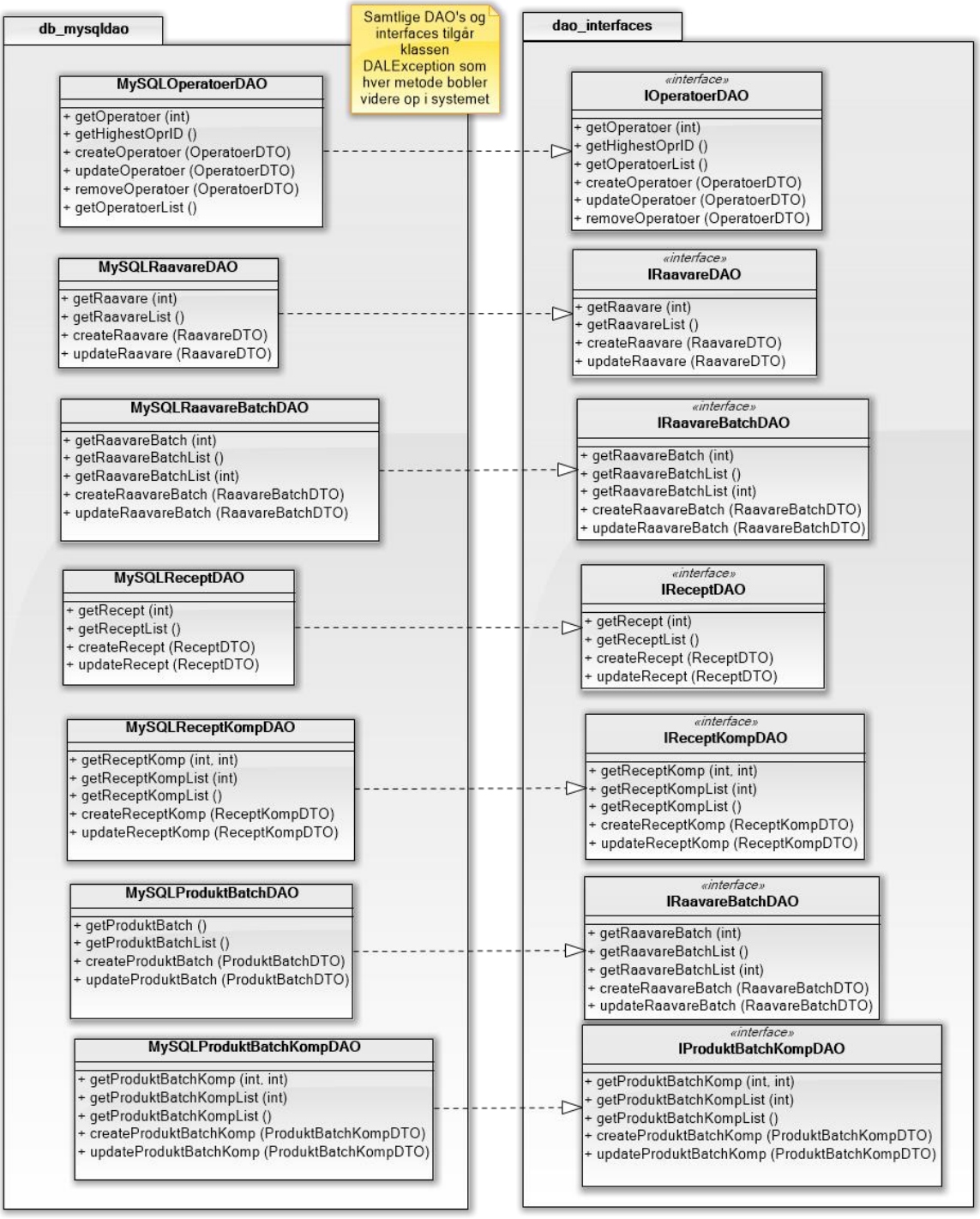


Figur 11 - Designklassediagram ASE in-depth

Her har vi igen fokuseret ind på de væsentligste klasser og listet deres attributter og metoder. Det kan ses, at klassen "Sequences" indeholder forskellige metoder afhængig af brugerens situation, som vi har delt op i sekvenser på samme, som man har angivet det i bilag 4 jf. opgavebeskrivelsen. Dette er med til at skabe et bedre overblik i klassen.

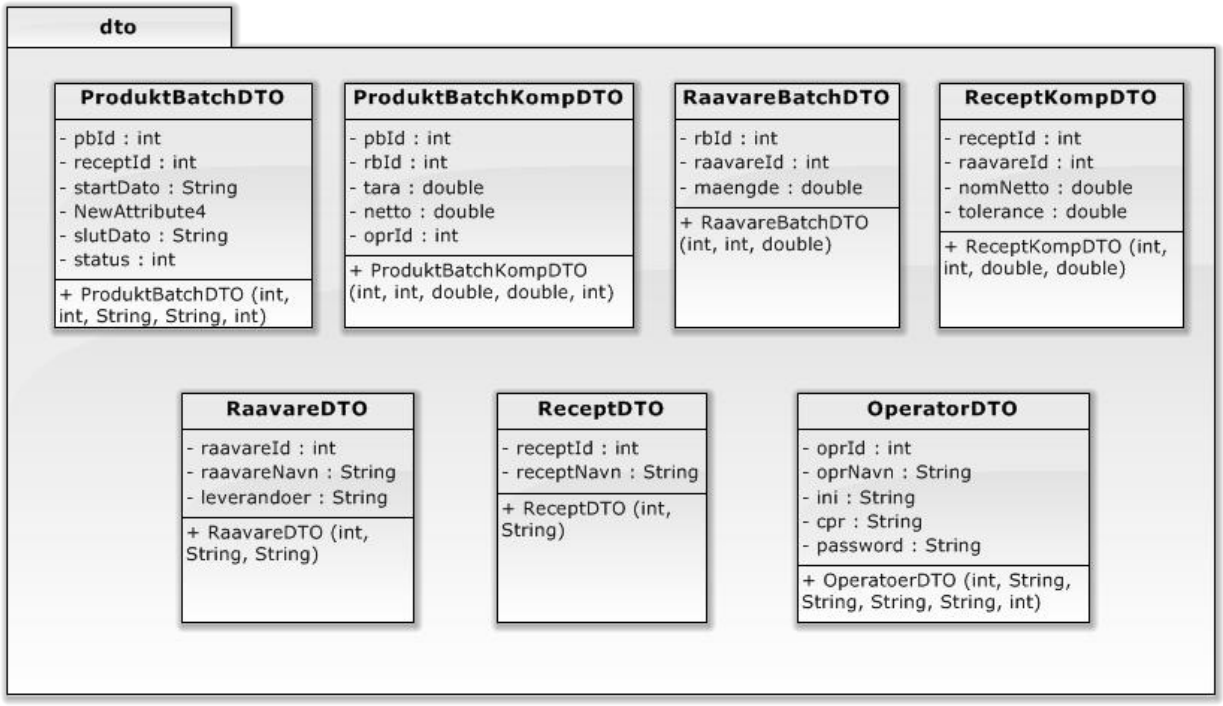
Datalag - herunder database, DTOs og DAOs

DTO's og DAO's



Figur 12 - DTO og DAO

Ovenstående klasse-diagram viser alle Data Access Objekterne(DAO), der hver især bruges til at tilgå de respektive Data Transfer Objekter. Ved hjælp af interfaces afkobles systemet, og for hver gang en controller-klasse skal tilgå et DTO, bliver metoden kaldt gennem et af de forskellige interfaces. I noten står desuden at hver metode, vha. throws bobler DALExceptions videre op i systemet, hvor disse så vil blive håndteret. Alle DAO-klasserners metoder indeholder forespørgsler direkte til MySQL-databasen.

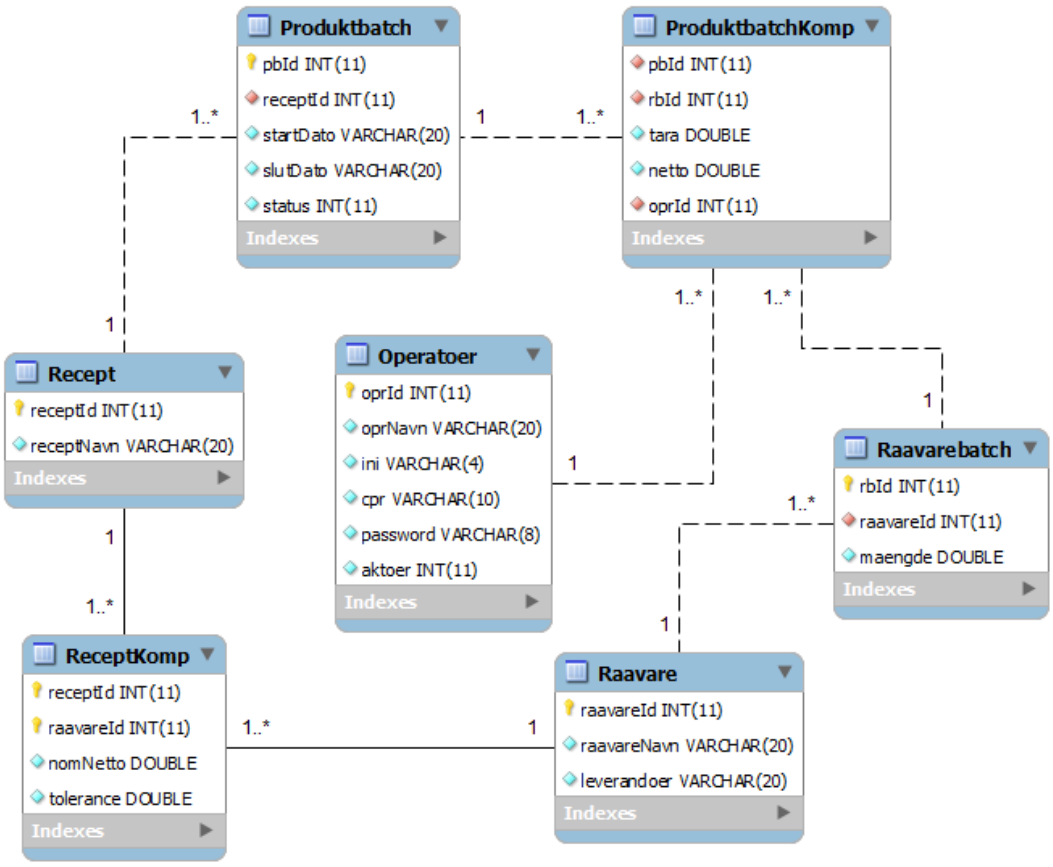


Figur 13 - DTO

Her ses alle DTO'erne med hver deres attributter og konstruktør.

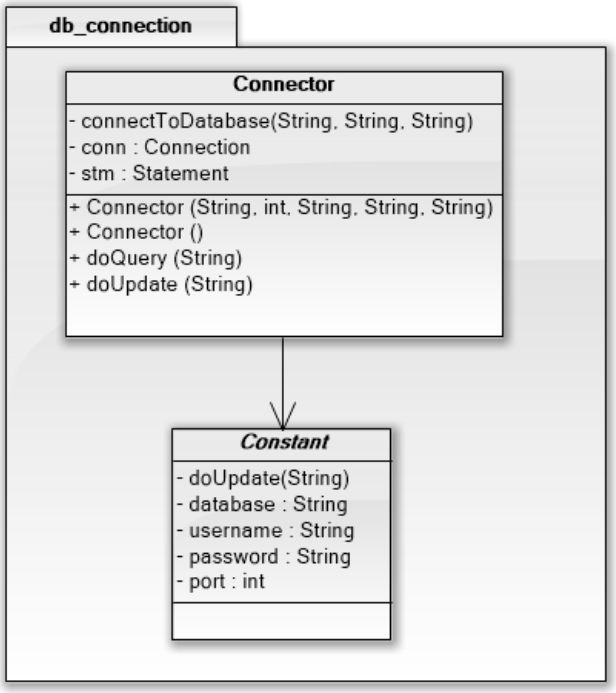
Database

Tabellerne jf.bilag 3 i opgavebeskrivelsen vil det ikke give nogen mening at opdele yderligere i flere og dermed mindre tabeller. Dog har vi ved en mange-til-mange relation ved 2 tabeller været nødt til at lave en 3.tabel for at kunne forbinde dem ved anvendelse af primærnøgler og fremmednøgler. Dette har vi gjort ved tabellerne "Recept" og "Produktbatch", hvor vi har oprettet én ekstra tabel ved dem hver især, som hedder "Receptkomponent" og "Produktbatchkomponent".



Figur 14 - ER-model

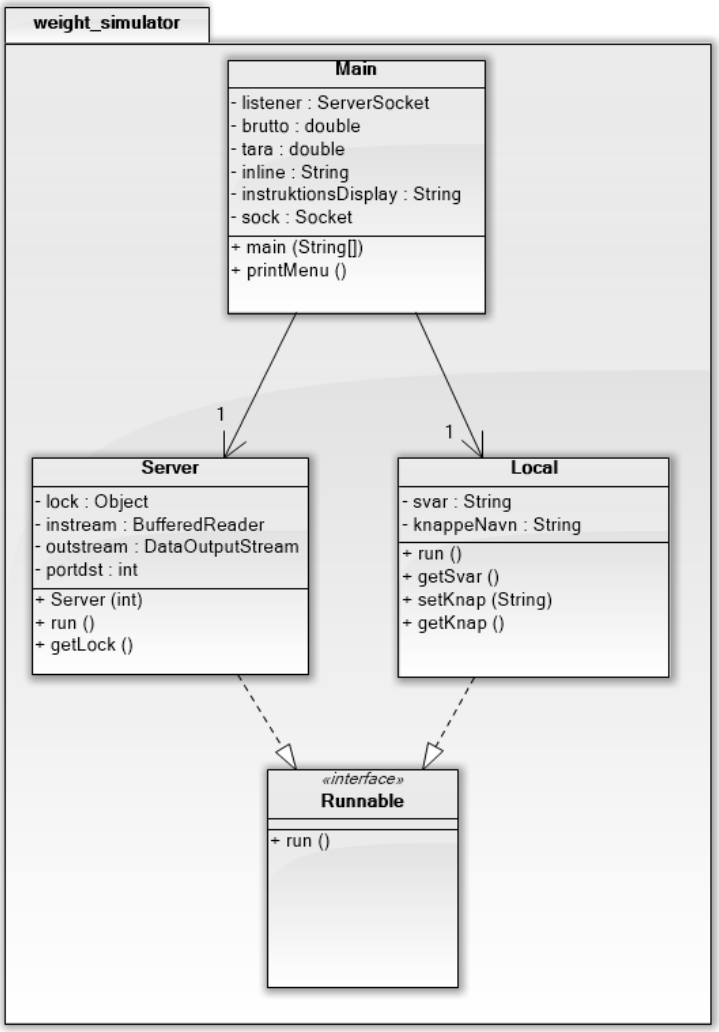
Ovenstående entity-relationship model (ER-model) viser vores tabeller i databasen og deres indbyrdes relationer og sammenhænge.
De gule nøgler indikerer, hvilken attribut, der er primærnøglen i den enkelte tabel, mens de røde romber indikerer fremmednøgler.
De stiplede linjer betyder, at der er en fremmednøgle, som peger på en primærnøgle i en anden tabel.
De optegnede linjer betyder, at der er en attribut, som både er en primærnøgle og fremmednøgle, som dermed refererer til en primærnøgle i en anden tabel.
F.eks. er receptId i tabellen "ReceptKomp" både en sammensat primærnøgle i denne tabel, men samtidig også en fremmednøgle, som refererer til receptId i tabellen "Recept".



Figur 15 – db_connection pakke

Her ses pakken med de to klasser der sørger for etablering af forbindelse til databasen. Connector-klassen sørger for forbindelsen ved brug af Constant-klassen som indeholder domæne- og brugernavn samt password og port til sql-serveren.

Vægtsimulator



Figur 16 - Vægtsimulator

Sådan ser designet for vores vægtsimulator ud, en lille program der skal simulere den fysiske Mettler vægt. Der køres to parallelle tråde foruden main-tråden, som tager imod hhv. TCP/IP-input gennem telnet og konsol-input gennem en lokal Scanner klasse.

Implementering

Web applikation

I denne opgave skulle vi også lave en web applikation som generelt skulle stå for administrering af de forskellige elementer, samt autentificering af brugere. Dette er implementeret med en række jsp-sider som alle ligger i mappen WebContent → WEB-INF. I alt har vi 23 jsp-sider. som vi har implementeret som en servlet, ved hjælp af både HTML og Java. Vores web_interface package indeholder 7 klasser, som sammen med dem i WEB_INF udgør hele web applikationen.
Vi startede med at lave det meste af vores servlet i en klasse, men som den blev større og større, indså vi, at den skulle opdeles i flere klasser for at skabe bedre overblik. Derfor endte det så med opdeling, som ses i det færdige produkt.

Vi har som tidligere nævnt anvendt model 2- eller "Model-View-Controller" arkitekturen, hvor vi netop har en model, et view og en controller i form af vores database, grænseflade og servlet.
Generelt er idéen med denne opbygning, at det hele kører i den her cirkel, som vist tidligere i rapporten under design-afsnittet. Den kommer dermed igennem samtlige lag, hvor det er vores servlets opgave at afgøre, hvilke jsp-sider der vises. Herunder ses en række af de else if-sætninger i vores servlet der håndtere, hvilke sider der vises afhængig af brugerens valg:

```
43  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
44      Functionality funktionalitetsLaget = new Functionality();
45      try{
46
47          if (request.getParameter("login") != null && request.getParameter("login").equals("Log ind")) {
48              otherMetoder.handleLogIn(request, response, funktionalitetsLaget);
49          }
50          else if (request.getParameter("logoff") != null && request.getParameter("logoff").equals("Log af"))
51              otherMetoder.handleLogOff(request, response);
52          else if (request.getParameter("hovedmenu") != null && request.getParameter("hovedmenu").equals("Tilbage til hovedmenu"))
53              otherMetoder.handleGoToHovedmenu(request, response);
54          else if (request.getParameter("adminibruger") != null && request.getParameter("adminibruger").equals("Administrer brugere"))
55              request.getRequestDispatcher("/WEB-INF/admin/bruger/adminibruger.jsp").forward(request, response);
56          else if (request.getParameter("createopr") != null && request.getParameter("createopr").equals("Opret bruger")) {
57              int oprID = 0;
58              oprID = funktionalitetsLaget.getOprDAO().getHighestOprID().getOprId()+1;
59              request.setAttribute("autoid", oprID);
60              request.getRequestDispatcher("/WEB-INF/admin/bruger/createopr.jsp").forward(request, response);}
61          else if (request.getParameter("showopr") != null && request.getParameter("showopr").equals("Vis brugere"))
62              brugerMetoder.handleShowOpr(request, response, funktionalitetsLaget);
63          else if (request.getParameter("removeopr") != null && request.getParameter("removeopr").equals("Slet bruger"))
64              brugerMetoder.handleRemoveOpr(request, response, funktionalitetsLaget);
65          else if (request.getParameter("updateopr") != null && request.getParameter("updateopr").equals("Opdatér bruger"))
66              brugerMetoder.handleUpdateOpr(request, response, funktionalitetsLaget);
67          else if (request.getParameter("adminivare") != null && request.getParameter("adminivare").equals("Administrer råvarer"))
68              raavareMetoder.handleAdminiVare(request, response, funktionalitetsLaget);
```

Figur 17 - Printscreen af kode 1

Ovenstående viser begyndelsen af vores doPost-metode. Det er netop denne metode der "styrer" alt mht. visning af jsp-siderne ved anvendelse af forward, hvor vi i CDIO-del 4 anvendte redirect fra en jsp til en anden.
Under hver if og else if-sætning ses det, at vi har en række andre metoder for hver enkelt jsp-side. De forskellige "handle" metoder er placeret i web_interface pakken i de relevante klasser. Det der ses herover er selvfølgelig kun starten af disse else if-sætninger, da de i alt fortsætter ned til linje 121, hvor metoden dermed også slutter.

Ligesom i CDIO del 4, bruger vi ikke GET metoder til at sende informationer videre. Dette sikre at man ikke kan se i URL'en hvilke parametre bliver sendes videre til næste JSP side. På den måde får vi øget sikkerheden. Vi bruger i stedet POST metoder i alle vores forms, da dette sikrer at brugeren ikke kan ændre på værdierne igennem URL'en.
Når der er behov for det bruger vi hidden input felter, når der skal sendes information videre fra en JSP side som ikke nødvendigvis skal ses af brugeren. Et eksempel er i createopr.jsp, hvor vi her sender ID'et videre på den bruger man er ved at oprette.

```
12          <input type="hidden" name="id" value="${autoid}">
```

Da vi nåede de afsluttende faser af implementering, begyndte vi at se en smule finpudsning af den visuelle del af vores web applikation. Det indebar blandt andet, at vi anvendte CSS til vores login side, for at få login-boksen til at være i midten af skærmen og derved give nemmere overblik over at finde ud af hvor brugeren skal indtaste sine login oplysninger. Vi har generelt ikke anvendt CSS i vores web-applikation, da det var langt nede på vores prioritetsliste. Vi har istedet valgt at holde designet meget simpelt for at øge brugervenligheden og derved ikke forstyrre brugeren ved unødvendigt design. Dette er også en af grundene til at vi gennem alle vores JSP sider blot har valgt at benytte os af submit knapper. For os var det vigtigste funktionaliteten og brugervenligheden. Dertil skal det siges, at alle websiderne er valideret i forhold til W3C standarderne.
Herunder ses vores CSS i login jsp-siden:

```
15      background-color: white;
16  -webkit-border-radius: 5px;
17  border-radius: 5px;
18  -webkit-box-shadow: 0px 0px 15px 2px rgba(0, 0, 0, 0.13);
19  box-shadow: 0px 0px 15px 2px rgba(0, 0, 0, 0.13);
20  padding: 30px;
21  padding-top: 5px;
22  margin-left: auto;
23  margin-right: auto;
24  margin-top: 160px;
25  border-style: solid;
26  border-color: #c0c0c0;
27  border-width: 1px;
28  text-align: left;">
```

Figur 18 - Printscreen af kode 2

Det som vi ville her, var at få placeret vores login boks på midten af siden, da vi tænkte det ville se bedre ud. Som det er nu er boksen i midten og når man ændrer på vinduet i bredden vil boksen forblive i midten. Den vil dog ikke forblive i midten, når man ændrer på vinduets højde, da vi har lavet en margin på 160 pixels op til toppen. Vi har bevidst prøvet at holde det så simpelt så muligt, derfor holdt vi os til baggrunds farven hvid. Efter lidt afprøvning besluttede vi, at dette ville give den bedste brugeroplevelse, da der hurtigt kan komme meget forstyrrelse ved at have en baggrundsfarve eller et baggrundsbillede..

Når du logger ind ser du en række forskellige muligheder, alt efter hvilken aktør du er og hvad dine rettigheder er sat til. F.eks. vil en administrator have en række flere muligheder end en

operatør, der kun kan foretage en afvejning. Herunder ses vores hovedmenu fra en administrators perspektiv. En administrator har alle muligheder og derved kan vi på dette billede se de implementerede funktioner.

Figur 19 - Printscreen af interface 1

En funktion, der er helt unik for netop administratoren er "Administrer brugere". Denne funktion har hverken farmaceuter, værkfører eller operatører. Denne funktions undermenu ses herunder:

Figur 20 - Printscreen af interface 2

Herunder ses en af metoderne, der skal håndtere når vi vil oprette en ny bruger.

```
79    public void handleCreateOprSubmit(HttpServletRequest request, HttpServletResponse response,
80            IFunctionality funktionalitetsLaget) throws ServletException, IOException, DALException {
81        int id = Integer.parseInt(request.getParameter("id"));
82        String navn = request.getParameter("navn");
83        String init = request.getParameter("init");
84        String cpr = request.getParameter("cpr");
85        cpr = cpr.replace("-", "");
86        String newPw = funktionalitetsLaget.generatePassword();
87        int aktoer = Integer.parseInt(request.getParameter("aktoer"));
88        int oprID = 0;
89        try {
90            oprID = funktionalitetsLaget.getOprDAO().getHighestOprID().getOprId()+1;
91        } catch (DALException e) {
92            e.printStackTrace();
93        }
94        request.setAttribute("autoid", oprID);
95        String errorMsg = "";
96        if (navn.length() == 0)
97            errorMsg = "Navn er ikke defineret! ";
98        if (init.length() == 0)
99            errorMsg += "Initialer er ikke defineret! ";
100       if (cpr.length() == 0)
101           errorMsg += "CPR-nummeret er ikke defineret! ";
102       if (cpr.length() != 10)
103           errorMsg += "Et CPR nummer er 10 cifre!";
104       if (errorMsg.length() == 0) {
105           funktionalitetsLaget.getOprDAO().createOperatoer(new OperatoerDTO(id, navn, init, cpr, newPw, aktoer));
106           request.setAttribute("password", "<br>Brugerens password er: " + newPw);
107           request.setAttribute("succes", "Bruger oprettet!");
108       }
109       else
110           request.setAttribute("fail", errorMsg);
111       request.getRequestDispatcher("/WEB-INF/admin/bruger/createopr.jsp").forward(request, response);
112   }
```

Figur 21 - Printscreen af kode 3

Dette er vores "handleCreateOprSubmit"-metode i vores brugere.java klasse. Den står netop for at oprette en ny bruger og ikke kun operatør, som metodens navn med Opr kunne indikere.
Vi kan blandt andet se en række if-statements, disse sørger for at teste, om brugeren har tastet informationerne korrekt ind. Det bemærkes blandt andet, at vi tjekker og sørger for, at cpr-nummeret skal være 10 cifre. Derudover laves der en replace på bindestreg. Dette gør, at det er ligemeget, om administratoren har skrevet bindestreg i CPR nummeret, bindestregen bliver blot fjernet inden CPR nummeret gemmes i databasen. Brugeren oprettes med createOperatoer metoden inde i vores OperatoerDAO klasse, denne metode tager et OperatoerDTO som argument. Hvis man opfylder kravene og længden af fejlbeskeden er lig 0, og der derved ikke er nogen fejl i de indtastede informationer, så bliver brugeren oprettet og gemt korrekt. Vi bruger setAttribute metoden i HttpServletRequest til at sende informationer fra servletten til JSP siderne. Dette gør at vi kan få dynamisk indhold på vores JSP siderne men alligevel adskille java og HTML.

Figur 22 - Printscreen af interface 3

JSP-siden ses ovenfor, dette er altså det, som brugeren indtaster og vælger, hvor brugeren oprettes, når der klikkes "Opret bruger" og de indtastede parametre gemmes i databasen.

Enhver aktør kan ændre sit eget password til et andet, ved at gå til siden 'Ændr Password'. Logikken er implementeret således at et nyt password skal overholde DTU's regler for ændring af password.[1]

ASE

Vores ASE tager udgangspunkt i afvejningsproceduren jf. bilag 4 i opgavebeskrivelsen. Vores ASE er derfor delt op i forskellige sekvenser, som hver især skal håndtere de enkelte opgaver under afvejningsprocessen. Med adskillige sekvenser, dvs. flere metoder, kan fejl lettere håndteres, da der dermed ikke skal startes forfra. Det er dermed muligt at starte fra en ønsket sekvens, hvis noget går galt.

[1] https://password.dtu.dk/admin/change_password.aspx - 23-06-2013

```
339    public void tjekMsgFejlOgRead(int sekvensVedFejl, BufferedReader inFromServer, DataOutputStream outToServer)
340            throws IOException{
341        try{
342            if (seqH.getServerInput().contains("T S ") || seqH.getServerInput().contains("S S ")
343                    || seqH.getServerInput().contains("RM30 B") || seqH.getServerInput().contains("DW A")
344                    || seqH.getServerInput().contains("P121 A") || seqH.getServerInput().contains("RM39 A")){
345                return;
346            }
347            else if(seqH.getServerInput().contains(" B") || seqH.getServerInput().contains(" A")){
348                seqH.setServerInput(inFromServer.readLine());
349                return;
350            }
351            else if(seqH.getServerInput().contains(" I") || seqH.getServerInput().contains(" L")
352                    || seqH.getServerInput().contains("T +") || seqH.getServerInput().contains("T -")
353                    || seqH.getServerInput().contains("S +") || seqH.getServerInput().contains("S -")){
354                switch(sekvensVedFejl){
355                case 2: this.sequence2(inFromServer, outToServer); break;
356                case 3: this.sequence3(inFromServer, outToServer); break;
357                case 4: this.sequence4(inFromServer, outToServer); break;
358                case 5: this.sequence5(inFromServer, outToServer); break;
359                case 6: this.sequence6(inFromServer, outToServer); break;
360                case 7: this.sequence7(inFromServer, outToServer); break;
361                case 8: this.sequence8(inFromServer, outToServer); break;
362                case 9: this.sequence9(inFromServer, outToServer); break;
363                case 10: this.sequence10(inFromServer, outToServer); break;
364                case 11: this.sequence11(inFromServer, outToServer); break;
365                case 12: this.sequence12(inFromServer, outToServer); break;
366                }
367            }
368        }
369        catch (DALException e) {
370            e.printStackTrace();
371        }
372    }
```

Figur 23 - Printscreen af kode 4

På ovenstående billede ses et udsnit af klassen sequence.java. Metoden tjekMsgFejlOgRead blev oprettet for at reducere de mange linjer med if og else, der opstod, idet alle vægtens svar på diverse kommandoer skulle håndteres. I stedet køres denne metode hver gang, vægten returnerer et svar på en kommando. Metoden får altid en integer med som parameter, der, hvis vægten sender en fejlbesked, videresender brugeren til en forrig sekvens, men dog oftest sekvensen, hvor noget gik galt. Dermed sendes kommandoen igen og forsøges at udføres endnu engang.

```
253    public void sequence11(BufferedReader inFromServer, DataOutputStream outToServer)
254            throws IOException, DALException
255    {
256        // Skifter til vejedisplay.
257        outToServer.writeBytes("DW" + "\r\n");
258        seqH.setServerInput(inFromServer.readLine());
259        this.tjekMsgFejlOgRead(11, inFromServer, outToServer);
260
261        // Tilføjer tolerance-pil vha. udregnet tolerance.
262        outToServer.writeBytes("P121 " + mRecKomp.getReceptKomp(seqH.getReceptID(),
263                seqH.getRaavareID()).getNomNetto() + " kg " + seqH.getTolerance() +
264                " kg " + seqH.getTolerance() + " kg " + "\r\n");
265        seqH.setServerInput(inFromServer.readLine());
266        this.tjekMsgFejlOgRead(11, inFromServer, outToServer);
267
268        // Opretter en softknap til vejedisplay
269        outToServer.writeBytes("RM30 \"" + "Afvej" + "\"\r\n");
270        seqH.setServerInput(inFromServer.readLine());
271        this.tjekMsgFejlOgRead(11, inFromServer, outToServer);
272
273        // Anbringer den oprettede softkey.
274        outToServer.writeBytes("RM39 1" + "\r\n");
275        seqH.setServerInput(inFromServer.readLine());
276        this.tjekMsgFejlOgRead(11, inFromServer, outToServer);
277
278        // Venter på, at der trykkes AFVEJ.
279        seqH.setServerInput(inFromServer.readLine());
280        outToServer.writeBytes("S\r\n");
281        seqH.setServerInput(inFromServer.readLine());
282        this.tjekMsgFejlOgRead(9, inFromServer, outToServer);
283        seqH.setNetto(seqH.splitDouble(inFromServer, outToServer));
```

Figur 24 - Printscreen af kode 5

Ovenstående er et uddrag af sequence11. Der bruges her adskillige kommandoer, da det er i denne metode, selve vejningen skal foregå. I første omgang skal vægten skifte over til vejedisplay'et, der dermed kan vise operatøren, hvor meget, der vejes af. Dette gøres ved at sende kommandoen DW til vægten. ServerInput sættes dernæst til vægtens svar på kommandoen så vi dernæst, vha. førnævnte tjekMsgFejlOgRead-metoden, kan tjekke svaret gennem for fejl og, hvis der dermed opstår en fejl, kan starte sekvensen forfra ud fra den angivne integer i metodens parameter.
Dernæst bruges kommandoen P121 for at placere tolerancepilen på vægten til at hjælpe operatøren med at vide, hvor meget vedkommende skal afveje. Denne tager adskillige parametre som består af den angivne nominelle nettovægt samt den positive og negative tolerance-grænse. Alt dette hentes fra de tilhørende set-metoder.
En softknap oprettes og placeres alt imens tjekMsgFejlOgRead-metoden tjekker for svar-fejl fra vægtens side.
Til sidst trykkes der på den oprettede softknap, når vægten skal afveje.

```
285         if(seqH.getNetto() >= seqH.getTotNegTol() && seqH.getNetto() <= seqH.getTotPosTol()){
286             RaavareBatchDTO raavareBatch = mRaaB.getRaavareBatch(seqH.getRbID());
287             raavareBatch.setMaengde(raavareBatch.getMaengde() - seqH.getNetto());
288             mRaaB.updateRaavareBatch(raavareBatch);
289 
290             mPbKomp.createProduktBatchKomp(new ProduktBatchKompDTO(seqH.getPbID(),
291                     seqH.getRbID(), seqH.getTara(), seqH.getNetto(),
292                     seqH.getOprID()));
293             ProduktBatchDTO produktBatch = mPb.getProduktBatch(seqH.getPbID());
294             produktBatch.setStatus(1);
295             mPb.updateProduktBatch(produktBatch);
296             this.sequence5(inFromServer, outToServer);
297         }
298 
299         else{
300             String weightMsg = "Ugyldig vejning. Den nominelle nettovaegt skal vaere mellem "
301         + seqH.getTotNegTol() + " kg og " + seqH.getTotPosTol() + " kg ifoelge databasens " +
302                 "tolerancevaerdier. Undgaa yderligere, at vaegten er i overbelastning eller "
303         + "underbelastning.";
304             seqH.RMPrintOgRead(49, 2, weightMsg, inFromServer, outToServer);
305             seqH.setServerInput(inFromServer.readLine());
306             this.sequence11(inFromServer, outToServer);
307         }
308     }
```

Figur 25 - Printscreen af kode 6

I ovenstående fortsættelse af sequence11 tjekkes der nu, om den vejede mængde er inden for databasens toleranceværdier for råvaren. I så fald opdateres RaavareBatchen, så denne får trukket den vejede mængde fra sin lagerbeholdningsmængde. Dernæst oprettes der en ProduktBatchKomp med de nye oplysninger. Imidlertid opdateres ProduktBatch'en med status sat til 1 (under produktion). Dernæst fortsættes der til metode 5, hvor den næste RaavareBatch vælges.
Hvis vejningen dog ikke stemmer ens, modtages en fejlmeddelelse og man skal igen afveje den korrekte mængde inden for tolerancen.

Fysisk vægt i forhold til simulator

I den indledende fase af ASE implementeringen testede vi med "Scale.exe" vægt-simulatoren som var givet fra CDIO opgave 1/2 i datakommunikation. Dette lettede processen med at opdage fejl, men da det kom til at programmet skulle virke med den fysiske vægt, var der nogle problemer. Et af disse problemer bestod af at antallet af readlines. Readline er en metode som er en default java "pakke", som vi har importeret. Den modtager input fra serveren og er dermed med til at kommunikere med vægten og modtage de forskellige svar fra vores RM ordre. Når vi sender en RM20 8 ordre til den fysiske vægt, modtager vi en RM20 B efterfulgt af RM20 A, hvis det er noget bruger-input vi skal modtage. I mange tilfælde rundt omkring i programmet er det tilfældet at vi har været nødt til at ændre på mængden af readlines for at få det til at fungere.

Det var relativt tidligt, at vi gik over til at teste på den rigtige vægt så meget vi kunne, da det virkede som det mest effektive at gøre. På den måde ville vi undgå enhver slags "omskrivning" af koden og samtidig sikrede vi os, at sekvenserne som vi implementerede også virkede på

vægten. I starten af uge 2 fik vi opdateret vores egen vægt-simulator, som vi lavede i CDIO opgaverne i videregående programmering. Dermed gik vi over til at bruge både den fysiske vægt og vores egen vægtsimulator, når vi ikke havde mulighed for at bruge den rigtige vægt.

Vi havde brugt en del tid og haft noget besvær med at få teksten til at passe på den fysiske vægts display. Godt inde i opgaven opdagede vi, at nogle andre grupper havde fundet en måde at omgå RM20 ordrens begrænsede 24 karaktere. Det var med kommandoen RM49 i stedet, der kan håndtere op til 240 karaktere i stedet. Vi valgte dermed at implementere denne ordre i stedet for RM20, i hvert fald langt de fleste steder i vores program. Til trods for at have alt den plads tilgængeligt, havde vi stadig brugervenlighed og overskuelighed i fokus og prøvede derved stadig at finde frem til de korteste og mest præcise formuleringer. Dette betød samtidig at vi måtte implementere flere kommandoer i vores vægt simulator for at få den til at virke sammen med vores ASE.

Database

Vi valgte at sætte vores egen MySQL server op på en server placeret i Tyskland frem for at bruge DTU's eller en anden ekstern udbyder. Dette har gjort at vi har haft fuld kontrol over serveren og databasen. Vi valgte at bruge OpenSource-værktøjet phpMyAdmin. phpMyAdmin tilbyder et webinterface hvor man kan se samtlige tabeller og databaser på MySQL serveren. Dette giver en god brugervenlighed og overskuelighed, samt sparer en masse tid og unødvendige tastetryk. Vi kunne også have valgt at køre en lokal MySQL server, men det har været en stor fordel at kunne bruge samme MySQL-database fra flere computere på samme tid. Dette har gjort at vi har kunnet udvikle vores program på tværs af platforme og lokationer - der skulle blot være internet til at kunne forbinde til MySQL-databasen.

I vores ASE og Servlet har vi valgt at bruge de samme DAO (Data Access Object) klasser til at forbinde til databasen med. Vi har en DAO klasse til at forbinde til hver relation. Hver DAO er opbygget på samme måde for at skabe overskuelighed, de består af de grundlæggende metoder get, create, update, remove og getList. Get metoderne står for at returnere et DTO (Data Transfer Object) ud fra et ID givet ud fra argument i get metoden. Denne laver altså et SELECT statement der henter fra den relation der passer sammen med DAO'en. Create, update og remove metoderne tager et DTO som argument og indsætter/opdaterer eller sletter en tuppel i relationen ud fra informationerne i DTO'et. getList metoderne returnerer blot en fuld liste over relationen i form af en liste af DTO'er.

Test

Hovedformålet med at teste er at finde og dermed kunne forebygge fejl, så tidligt som muligt i processen. Dermed øges systemets kvalitet og vi sandsynliggør, at vores krav bliver opfyldt. Vi får desuden et overblik over vores mangler og udestående i systemet ved at teste de enkelte dele.
I vores ASE har vi optimeret programmet til at kunne debugge ved at have system out lines i både vores setWeightMsg funktion og setServerInput funktion. De står mere eller mindre for alt vores kommunikation med vægten og ved at have disse system outs kan man følge med i kommunikationen i konsollen. Dette overskueliggør programmet væsentligt, idet man rent faktisk kan se, hvad der sker. Dermed er fejl meget lettere at opdage da man kan se præcist hvad der skete sidst, før det gik galt.

I dette program skulle alle komponenter testes og dette fik vi gjort løbende som programmet udviklede sig. Udover at teste alle komponenter testede vi også løbende, at vores program stadig virkede med "kundens pc'er" som i dette tilfælde er DTU's maskiner. Det forløb som regel godt og når det ikke gjorde fik vi således udbedret fejlene, så det virkede. Vi følte at denne form for testing var meget væsentligt at få gjort undervejs, frem for bare til sidst. Derudover sørgede vi også for, at vi fik testet alle vores komponenter undervejs på en række forskellige måder, som ses herunder.

Usability-test

Vi har fået testet vores program af nogle udefrakommende, som har forestillet at være en af vores aktører. Baseret på deres erfaringer og kommentarer har vi kunnet overveje og forbedre vores interface og dermed interaktionen mellem bruger og system. Mange gange har det givet utrolig meget information om vores program, når nogen, der ikke er en del af projektet, prøver programmet af. Generelt kan man, ved selv at teste gentagne gange, finde adskillige fejl. En udefrakommende ved heller ikke, hvordan koden bagved fungerer. Dermed tester de på en unik måde kun ud fra output og det, de ser på skærmen. En af de "brugere", der blev testet, påpegede blandt andet nogle grammatiske fejl og tastefejl, som vi fik fjernet derefter. Udover de par småting, der blev rettet, blev vi også opmærksomme på manglende funktionalitet, som vi måske ellers ikke havde tænkt over.

Brugerinput/integrations-test

Igennem hele forløbet har vi desuden anvendt diverse brugerinput- og integrationstest, hvor vi har prøvet forskellige "ikke-mulige" brugerinputs, hvor vi så har fundet ud af, om vores fejl-meddelelsesbokse virker de relevante steder. Vha. integrationstest har vi kunne finde fejl i vores brugergrænseflade, som vi løbende har rettet til, så der selvfølgelig sker de ting, som skal ske de rigtige steder.

Stress-test

Vi udførte diverse stress-tests på vores web-applikation, hvor vi indtastede alt muligt i forskellige felter rundt omkring i vores system for at prøve at crashe den. Vi prøvede med bogstaver og tal blandet, specialtegn, meget lange inputs osv. Vi prøvede dermed med næsten alt man kan forestille sig og fik programmet til at crashe et par gange. Dette resulterede i, at vi fik udbedret vores web-applikation. Generelt set forventer vi ikke, at en bruger formår at taste så meget forkert, som vi gjorde, men vi føler alligevel, at det var godt at håndtere disse fejl. Vi udførte også lidt stress-testing på vores ASE, men ikke i så høj grad, som ved vores web-applikation.

Use case test

Vi har opstillet nedenstående testskema, hvor vi gennemgår nogle test-scenarier på input og forventet output ud fra nogle af vores use cases.

Use case navn (testet)	**Inddata/valg**	**Forventet output**	**Fejl (hvad skete der)**
Log ind	Id: 10 Password: jespeR15	Systemet kender ikke id'et, da det ikke ligger i databasen	Systemet lader ikke brugeren komme ind i systemet, hvilket er korrekt i dette tilfælde. Der sendes passende fejlmeddelse
Opret operatør (logget ind, som administrator)	Vælger "Administrere brugere" → "Opret bruger" og indtaster/vælger oplysninger. Id: 30 (autogenereret) Navn: Jens Hansen Initialer: JH CPR: 2345343565 Aktør: administrator	Den nye operatør bliver gemt i databasen og får tildelt et autogenereret password	Systemet gemmer korrekt operatøren i databasen med de korrekt-indtastede oplysninger
Vis recepter (logget ind, som farmaceut)	Vælger "Administrere recepter" → "Vis recepter"	En liste af alle vores oprettede recepter i vores database bliver vist	Systemet viser korrekt en liste over alle vores recepter
Opret produktbatch	Vælger "Administrere	En besked om, at der	Systemet gemmer

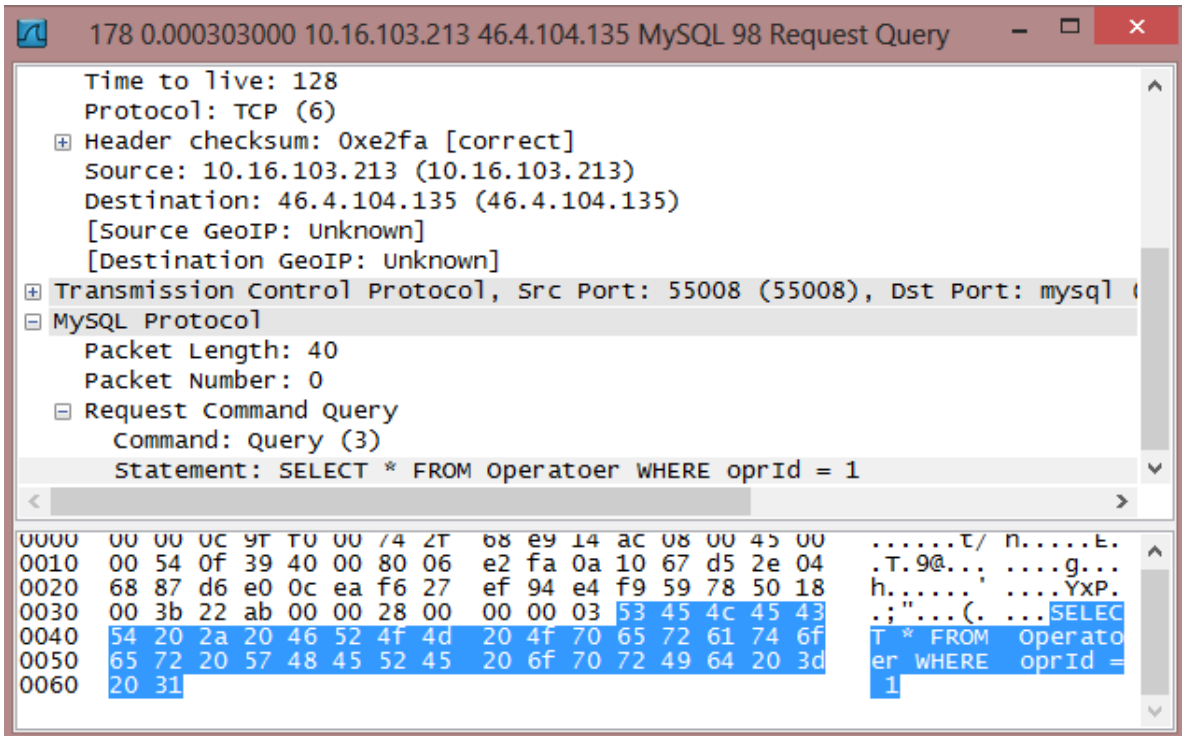
(logget ind, som supervisior)	produktbatches" → "Opret produktbatch" og indtaster/vælger oplysninger. Produktbatch id: 6 Recept id: 50 Status: Startet	nu er oprettet et produktbatch, som nu er gemt i databasen	korrekt den nye produktbatch i databasen og skriver, at den nu er oprettet
Ændre password	Administrator med ID 1 vælger "Ændr password" i hovedmenu og indtaster/vælger oplysninger. Gammelt password: jespeR15 Nyt password: 1æåø23! Nyt password(igen): 1æåø23!	Dit password er nu ændret! (og dermed at det nye password nu er 1æåø23!)	Korrekt output besked om at password er ændret, men fejl ved at password bliver ændret til 123! i stedet for 1æåø23!. Systemet kan således ikke håndtere bogstaverne åæø i denne sammenhæng.

Test af kommunikation med "Wireshark"

Vi har anvendt Wireshark til at se kommunikationen imellem vores forskellige komponenter. Vi har lavet to målinger med Wireshark. Første måling var af kommunikationen imellem vores klient og databasen. Vi har set på nogle af de steder af målingen hvor MySQL protokol anvendes til at hente vores oplysninger fra databasen. Vi ser herunder en forespørgsel og et svar mellem klient og database:

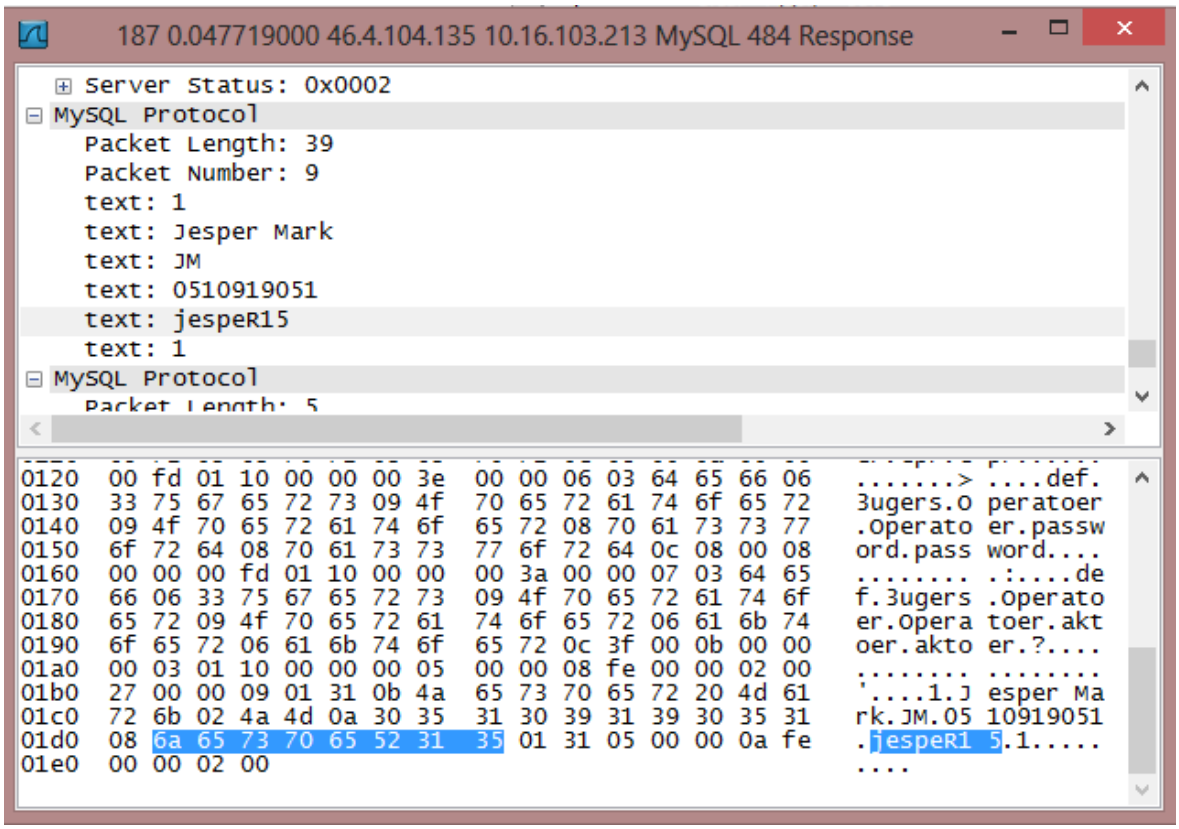
No.	Time	Source	Destination	Protocol	Length	Info
178	3.936099000	10.16.103.213	46.4.104.135	MySQL	98	Request Query
179	3.984531000	46.4.104.135	10.16.103.213	MySQL	484	Response

I nummer 178 sendes vores forespørgsel, som i dette tilfælde indeholder login "scenariet" hvor brugeren har indtastet sit ID, som i dette tilfælde er '1'.



Figur 26 - Printscreen af wireshark 1

Her kan kommandoen som afsendes af vores klients mysql dao aflæses: "SELECT * FROM Operatoer WHERE oprId = 1". Herefter modtager vi et svar på forespørgslen fra vores database.



Figur 27 - Printscreen af wireshark 2

Ovenstående screenshot er netop det svar, som vi får fra vores database. Her opfanges databasens response med al dataindhold i klartekst. Altså operatørens oplysninger som aktør, initialer, cpr, password osv. Dette et eksempel på kommunikationen mellem Servlet og DB.

Mulige forbedringer og udbyggelse

ASE

- En operatør der er i gang med en afvejning, skal på et hvilket som helst tidspunkt kunne afbryde afvejningen, for på et senere tidspunkt at kunne genoptage denne.
- En operatør skal til enhver tid kunne påbegynde en anden operatørs produktbatch, hvis denne er ufuldendt.
- Vores ASE skal kunne håndtere flere vægt-terminaler på én gang vha. tråde, således at flere operatører kan afveje flere produktbatches parallelt.
- Vores vægtsimulator skal være GUI-baseret og implementeres vha. Java's JFC-API (Swing)
- Vores vægtsimulator skal ligge tættere op ad den fysiske vægt, således at den har et større kartotek af fejlmeddelelser og kendte kommandoer.
- ASE skal kunne håndtere at en negativ vægt bliver afvejet, da der på nuværende tidspunkt vil kastes en exception, og lukke programmet.

Web-interface

- Samtlige requests/responses mellem Servlet, jsp-sider og database bliver sendt ukrypteret. Dette medfører, at samtlige forbindelser er uden yderligere sikkerhed, og systemer er sårbart overfor bl.a. man-in-the-middle-attacks. En simpel wireshark måling mellem vores servlet og klienten kan også opfange den indtastedes password, cpr-nummer og lignende information i klartekst.
- En administrator skal kunne slette råvarer og recepter. Dette er et krav, som ikke står beskrevet i opgavebeskrivelsen, men som vi tænker kunne være meget brugbart i et fremtidigt system.
- En administrator skal ikke kunne slette en operatør, som allerede er blevet oprettet én gang i systemet jf. use case 1 i opgavebeskrivelsen. Her afviger vores nuværende system, da man godt kan slette enhver operatør, men man kan til gengæld ikke slette en administrator.
- Når man ændrer password og bruger bogstaverne æøå i sit nye password vil systemet lave passwordet om, minus disse bogstaver hvis de indgår i det nye password. Altså vil det indtastede æå343 blive til det nye password "343" automatisk.

Andet

- Man skal ikke kunne dele initialer, da hver persons initialer burde være unikt.
- Virksomhedens webserver skal have implementeret et email-system, således at hver gang en ny bruger i systemet bliver oprettet, bliver der automatisk genereret en ny, unik mail-adresse til denne, ud fra brugerens initialer.
- Der skal laves et javadoc der dokumenterer koden og alle metoder, frem for, i øjeblikket, almindelige kode kommentarer.

Konklusion

Vi har implementeret alle de enkelte komponenter jf. opgavebeskrivelsens bilag 1 - System arkitektur diagram, som vi også har nævnt under vores brugermanual. Disse komponenter er sat sammen og udgør det færdige system. Vi har ikke konstrueret en GUI-baseret vægtsimulator, da det var nedprioriteret. Vi har prøvet at overholde og designe vores program efter designprincipperne bag 3-lagsmodellen. Nærmere bestemt har vi fulgt model 2-arkitekturen for webprogrammering, da vi har implementeret en controller i form af vores servlet og fået adskilt java og HTML-kode i vores jsp-sider.
Vi har etableret en database og kan kommunikere med denne via et DB access-lag, hvor vi dermed kan hente og gemme information.
Alle systemets aktører har mulighed for at udføre det som deres respektive use cases beskriver, og gruppen har - uden at det stod beskrevet i opgaveformuleringen - implementeret at samtlige aktører til enhver tid kan ændre deres passwords.

Figurfortegnelse

Anvendte værktøjer

- Eclipse Juno version 4.2
- GitHub (versionsstyringsværktøj)
- phpMyAdmin version 3.3.7deb7 (webinterface til database)
- Software Ideas Modeler version 5.40.4586.39098 af Dušan Rodina
- MySQL Workbench 5.2 CE
- Lucidchart
- Google docs
- Dropbox version 2.0.22
- Telnet Client (Update: 03/24-2010)
- Wireshark version 1.8.5
- Microsoft Publisher 2010
- MySQL version 5.1.66
- http://www.websequencediagrams.com/#

Litteraturliste

Hjemmesider

http://www.medicines.org.uk/emc/BrowseDocuments/a (10-06-13)
http://javabog.dk/JSP/kapitel10.jsp (10-06-13)
http://www.europages.dk/ (12-06-13)
http://www.w3schools.com/sql/sql_foreignkey.asp (13-06-13)
http://www.javadb.com/get-and-set-session-variables-in-a-servlet (13-06-13)
http://www.websequencediagrams.com/# (20-06-13)
http://stackoverflow.com/questions/8486878/how-to-cut-off-decimal-in-java-without-rounding(21-06-13)
https://password.dtu.dk/admin/change_password.aspx (23-06-13)

Billeder i poster

http://www.digitaltrends.com/wp-content/uploads/2011/01/google-chrome-logo-1000.jpg
(21-06-13)
http://www.thebrazostech.com/wp-content/uploads/2012/12/Server.png (21-06-13)
http://3.bp.blogspot.com/-
8_PmEa5zOOo/UQEmyJB62lI/AAAAAAAAAAU/ySJICkQlpTg/s200/silver-database-icon.jpg
(21-06-13)
http://www.aone.zen.co.uk/imagesproduct/cs/large/csen1501bnopsu.jpg (22-06-13)
http://www.philgalfond.com/wp-content/uploads/ethics-scale.jpg (21-06-13)

Dokumenter

WEB arkitektur.pdf (anvendt i 12.lektion i 13-ugers undervisningsperiode)
servlets.pdf (anvendt i 10.lektion i 13-ugers undervisningsperiode)
Introduktion til test DTU_slides_final.pdf