

21_CDIOFinal

Projektnavn: CDIOFinal

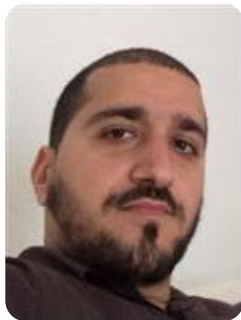
Gruppe nr.: 21

02324 Videregående programmering F16

Deltager:



Dannie Michael Fjordside
s114356



Haydar Tas
s136504



Abdi Shukri Aden Mohamud
s136615



Aslanbek Usamovitj Sultanov
s132993



Amer Ali
s145224



Mads Heidemann Justesen
s134844



Jonas Friis Kjær
s134840

Timeregnskab

Vores tidsregnskab kan ses for hver dato, samt summen i antal timer og i procent for hver gruppemedlem.

Indholdsfortegnelse

Timeregnskab	2
Indledning	5
Vision	7
Udviklingsredskaber	7
Begrænsninger	8
Arbejdsfordeling	8
Kravspecifikation for Web interface:.....	9
Funktionelle krav	9
Ikke-funktionelle krav	9
Kravspecifikation for Aktører:	10
Funktionelle krav	10
Ikke funktionelle krav	10
Kravspecifikation for Afvejningen:	10
Funktionelle krav	10
Ikke funktionelle krav	11
Krav til Data Access Layer (DAL) og Databasen.....	12
Funktionelle krav	12
Tre lags arkitektur	12
Usecases	13
Use case 1.....	14
Usecase 2.....	15
Use case 3.....	17
Usecase 4.....	18
Usecase 5.....	19
Usecase 6:	20
Designsekvensdiagram.....	21
Design sekvens diagram ASE	22
Database	25
Entity-forholds model (ER diagram)	25
Data Access Object (DAO).....	26
Normaliseringsanalyse	29
Design og organisering af tabeller i databasen	29

Operatoer.....	29
Raavare.....	30
Recept.....	30
Produktbatch	30
Raavarebatch.....	30
Data Access Layer (DAL).....	30
Data Transfer Object (DTO).....	31
Data Access Object (DAO).....	31
Google Web Toolkit (GWT).....	31
Vejledning til brug af ASE.....	37
Klassediagram ASE	37
Vægtsimulator.....	38
ClientController/CC:.....	38
IOController/IOC:.....	38
WeightController/WC:.....	38
Test.....	39
Google Web Toolkit (GWT).....	44
Design af GUI.....	44
ASE	46
Vejledning til brug af ASE.....	48
Vægtsimulator.....	49
ClientController/CC:.....	49
IOController/IOC:.....	49
WeightController/WC:.....	49
Konklusion	49
Bilag 1	51

Indledning

I dette CDIO projekt er der blevet taget udgangspunkt i selve kurset 02324, men der er også blevet inddraget især en del fra 02327 samt noget fra 62576 og 62577.

Opgaven går ud på at lave et software system til en medicinalvirksomhed. Det bruger vi bl.a. til afvejning af råvare. I systemet har vi en database, en afvejnings styrings enhed(ASE), en vægt simulator samt et GWT web interface. Via GUI'en som er vores web del læser og skriver vi til databasen. Der tages udgangspunkt i kundens krav i opgaven.

Brug af programmet.

Da alle aspekter af programmet ikke er funktionelt har vi forsøgt at vise de dele der virker. Vores GWT startes ved at køre i super development mode. Der er pre-indsat et validt Password: bruger: 4, Password: 4567Pass. GWT'en kommunikerer med vores MainController(), der desværre ikke kan få forbindelse med DataBasen, men opretter et mockObject, der udgør det for en Operatør (der er 4 brugere ma kan logge ind med:

1, 1234Pass

2, 2345Pass

3, 3456Pass

4, 4567Pass

4'eren er den eneste der giver adgang til alle knapper ved login (da han er administrator).

Ændring af Password er en funktion vi har fået implementeret - og den virker på de 4 førnævnte brugere.

Derefter kan der afprøves nogle funktioner via "Brugeradministration" - i opret bruger kan der prøves at oprette bruger med forkert Password (som jo selvfølgelig ikke er muligt), dog er selve oprettelsen ikke implementeret.

"Ret Bruger" er også implementeret med valideringer og popUp's, men søgningen returnerer dog altid bruger 4.

DataBasen, vi desværre ikke kunne connecte, kan afprøves ved at køre Main i Java (der ligger i cdio3.server.DB.test), den laver et par kald, kontrollerer den og viser dem på consollen.

ASE køres ved at forbinde vægten til computeren, derefter starter man klassen Run (der ligger i cdio3.server.ASE.controller) som så beder dig om at vælge hvilken vægt du vil bruge, når du har valgt det følger man de beskeder der bliver skrevet på vægten.

Alt efter hvilken fysisk vægt man bruge kan det være man skal ændre IP'en inden i ASE klassen.

Hvis du forbinder til vægt simulatoren skal du først starte simulatoren ved at åbne for cmd og derefter skrive java -jar CDIO2.jar. Du kan nu starte ASE og vælge Vægt simulator, derefter vil det foregå på samme måde som med den fysiske vægt.

Arbejdsfordeling

Navn	GWT	ASE	DB	Sammenbinding
Aslanbek				x
Haydar			x	
Amer			x	
Dannie	x			
Jonas	x			
Mads		x		
Abdi		x		

Tidsregnskab

Vores tidregnskab kan ses i bilag 1 for hver dato, samt summen i antal timer og i procent for hver gruppemedlem. Her er summen for hver person i antal timer og i procent.

	Abdi	Amer	Aslanbek	Dannie	Haydar	Jonas	Mads
Timer i alt	71	72	65	116	72	67	71
Procent	13.32	13.51	12.20	21.76	13.51	12.57	13.32

Analyse

Analysedelen er en af de vigtigste dele af projektet, da det er her vi beslutter os hvordan vores endelig program kommer til at være. I det her afsnit vil vi beskrive analysefasen i vores projekt, dvs. vi vil prøve vha. af tekst og diagrammer, at beskrive fra start til slut vores tanker bag det her projekt. En god ordning i analyse fasen øver også gangen i udviklingsfasen. Det er også vigtigt her at holde sproget så simpelt som muligt så kunden også kan følge med i projektets udvikling.

Vision

Som mål har vi at have en normaliseret database hvor vi har de 4 typer brugere (Operatør, Værkfører, Farmaceut og Administrator) med forskellige rettigheder og roller. Vi skal også have en færdiglavet ASE som styrer operatørens brug af vægten. Databasen skal kunne tilgås af ASE'en via et Data Access Lag (DAL) så man ved hjælp af ASE'en kan manipulere med databasen. Opgaven skal også overholde 3-lags modellen. Der skal kunne udføres blackbox test og selvfølgelig dokumentation til de forskellige ting.

Udviklingsredskaber

Til udarbejdelse af dette projekt har vi brugt følgende software redskaber:

- MagicDraw – Udarbejdelse af design diagrammer osv.
- Eclipse – Java IDE til udarbejdelse af softwaren
- Workbench – Håndtering af databasen
- Google Docs – Rapportskrivning
- GWT – Udvikling af webinterfaces
- Operativsystem – Windows 8, 8.1, 10 (32 og 64 bit)

Begrænsninger

Målet er jo at blive færdig med det hele men det betyder ikke at der ikke kan forekomme udfordringer, og derfor har vi nogle ting som vi gerne vil prioritere:

- Websidens design skal være nemt og overskueligt
- Databasen skal virke korrekt
- Forbindelse fra ASE til den fysiske vægt

Vi har valgt at prioritere netop disse dele, da vi i gruppen søger at få de enkelte dele færdige frem for en gennemløbende forbindelse i programmet. Vi vil dermed hellere have et program hvor vi igennem test kan vise at alle delene (databasen, GWT, controller, ASE osv.) virker korrekt og er komplette, frem for et program hvor alle delene har forbindelse til hinanden, og virker korrekt, men hvor der mangler funktionalitet i delene. Så vores begrænsning af opgaven ligger i, at vi vil vise, at vi kan lave programdelene korrekt, frem for forbindelse mellem de enkelte dele.

Arbejdsfordeling

Navn	GWT	ASE	DB	Sammenbinding
Aslanbek				x
Haydar			x	

Amer			x	
Dannie	x			
Jonas	x			
Mads		x		
Abdi		x		

Analyse

Analyse delen er en af de vigtigste dele af projektet, da det er her vi beslutter os hvordan vores endelig program kommer til at være. I det her afsnit vil vi beskrive analysefasen i vores projekt, dvs. vi vil prøver vha. af tekst og diagrammer, at beskrive fra start til slut vores tanker bag det her projekt. En god ordning i analyse fasen øver også gangen i udviklingsfasen. Det er også vigtigt her at holde sproget så simpelt som muligt så kunden også kan følge med i projektets udvikling.

Kravspecifikation for Web interface:

Funktionelle krav

- Web Interface skal være forbundet med en mysql database
- Det skal kunne være muligt for den at opretholde de oprettede brugers informationer i databasen
- Via webinterfacet skal det derfor være muligt at oprette, redigere, vise og aktivere eller deaktivere operatører via webinterfacet
- Det skal kunne være muligt for de oprettede bruger at ændre deres password

Ikke-funktionelle krav

- Ved brug af GWT programmeres der i Java
- Det relevante illustreres i UML diagrammer
- Design efter 3-lags modellen

Kravspecifikation for Aktører:

Funktionelle krav

- Programmet skal indeholde 4 forskellige aktører som er Administrator, Farmaceut, Værkfører og Operator
- Administratoren har de samme rettigheder som de andre fire, men derudover selvfølgelig også Admin rettigheder nemlig at administrere brugere.
- Farmaceuten skal kunne håndtere råvare og recepter, og har også værktøernes rettigheder.
- Værktøeren skal kunne håndtere raavarebatches og produktbatches og har derudover operatøernes rettigheder
- Operatøren sørger for afvejningen

Ikke funktionelle krav

- Ved brug af GWT programmeres der i Java
- Det relevante illustreres i UML diagrammer
- Design efter 3-lags modellen

Kravspecifikation for Afvejningen:

Funktionelle krav

- Ved hjælp af TCP skal vægten tilgås
- Gennem RM20 kommandoen skal der kunne være en kommunikation til operatøren
- Der skal ikke kunne ske en afbrydelse af afvejningen
- Afvejningen skal kunne afbrydes på et hvilket som helst tidspunkt

Ikke funktionelle krav

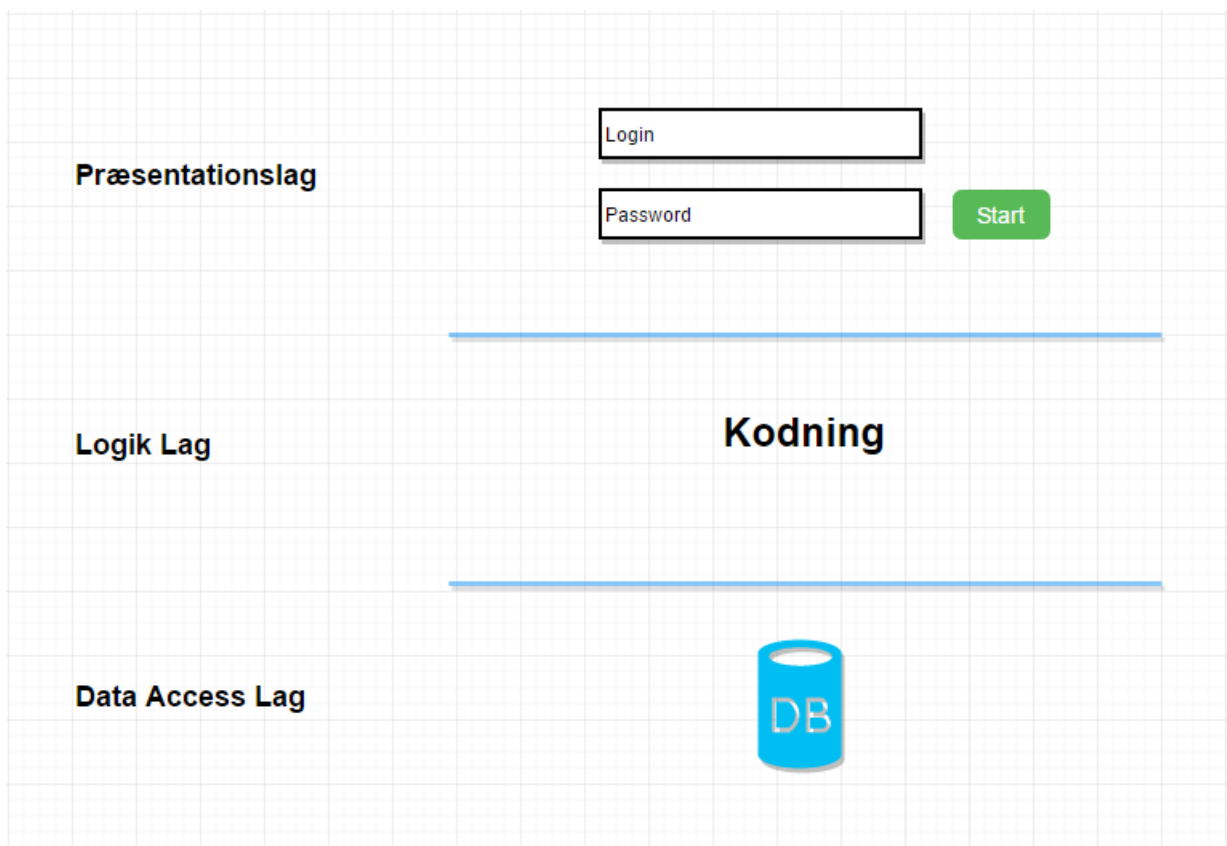
- Brugeren skal kunne få en fejlmeddelelse når der forekommer en fejl

Krav til Data Access Layer (DAL) og Databasen

Funktionelle krav

- Aktører skal ikke kunne slettes med derimod sættes som inaktive

Tre lags arkitektur

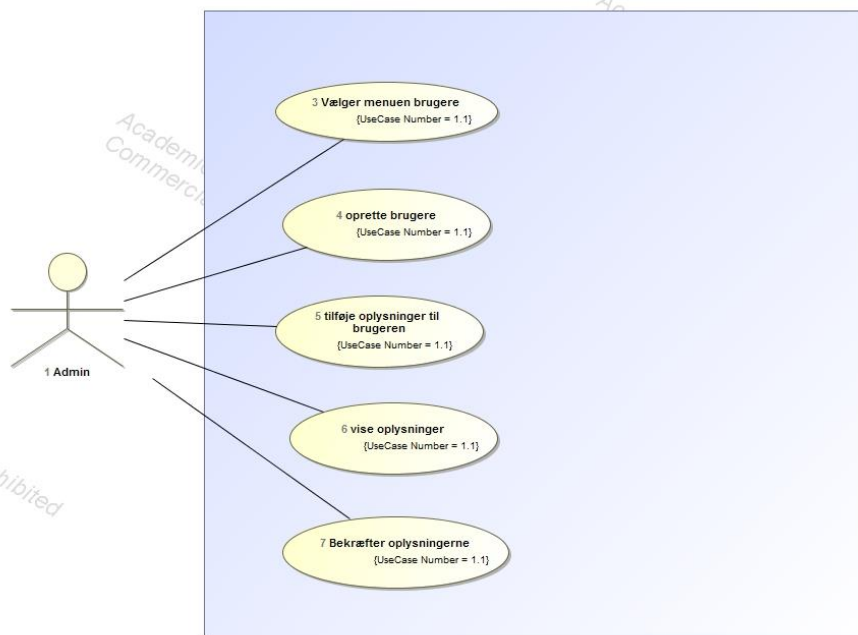


Vi har overholdt 3-lags modellen i implementation af vores projekt. View optræder som grænsefladen, hvor vi har brugt Google Web Tool (GWT). Alt vores logik er placeret i funktionslaget, som er vores controllere der udelukkende har med funktionaliteten at gøre. DTO og deres tilhørende interface DAO omfatter datalaget, hvor vi i DAO har MySQL sætninger der kan hente oplysninger fra databasen, som udtrykkes af en bruger i GWT.

Det samme gælder for ASE'en hvor vores grænseflade er vægten eller simulatoren, funktionslaget er vores ASE og controller. Datalaget er selvfølgelig vores database.

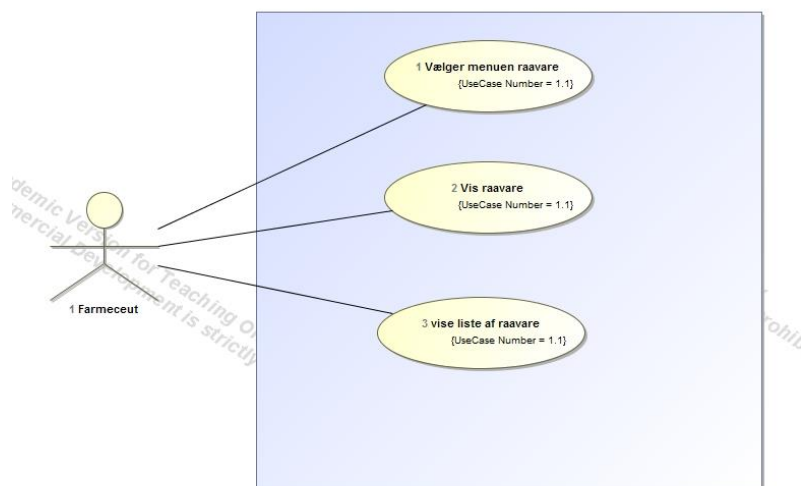
Usecases

Her præsenterer vi usecases baseret på ovenstående kravspecifikationer og opgaveoplægget. Usecasener beskriver aktørens mulige interaktioner med systemet.



Use case 1

Use Case: Bruger Administration.
ID: UC1
Kort beskrivelse: Administrator opretter en ny Bruger.
Primær bruger: Administrator (A).
Forudsætninger: Administratoren er logget ind.
<p>Main flow:</p> <ol style="list-style-type: none">1. A vælger menuen "Brugere".2. A vælger at oprette ny Bruger.3. A skriver fornavn, efternavn, cpr. nr. og stilling på den nye Bruger.4. A får vist den nye Brusers oplysninger, inkl. autogenerated Password og ID.5. A bekræfter at oplysningerne er korrekte og returneres til bruger-menuen.
Postconditions: A kan nu afslutte programmet eller vælge videre fra bruger-menuen.
<p>Alternativt flow:</p> <ol style="list-style-type: none">1. A indtaster et ugyldigt input.<ol style="list-style-type: none">1.1. A bliver bedt om at ændre inputtet.2. A indtaster et cpr. nr. der allerede eksisterer.<ol style="list-style-type: none">2.1. A får af vide at cpr. nr. allerede er i systemet.2.2. A kan vælge at det er en gammel bruger der er blevet genansat, eller afbryde oprettelsen.



Figur 1 - UC1 Beskrivelse

Denne usecase begynder, når en Administrator vil oprette en ny Bruger i systemet. Efter udførelsen af usecasen er der oprettet en Bruger.

Usecase 2

Usecase: Råvare Administration
ID: UC2
Kort beskrivelse: Farmaceut ser alle råvarer i systemet
Primær bruger: Farmaceut (F).
Forudsætninger: Farmaceuten er logget ind.
Main flow: <ol style="list-style-type: none"> 1. F vælger menuen "Råvarer". 2. F vælger menuen "Vis råvarer". 3. F får vist en liste over alle råvarer inkl. deres ID.
Postconditions: F kan nu afslutte programmet eller vælge videre fra råvarer-menuen.
Alternativt flow:

1. Et jordskælv i det Indiske Ocean skaber en tsunami.

1.1. Computeren bliver oversvømmet.

1.2. Programmet termineres.

Denne usecase begynder, når en Farmaceut vil se alle råvarerne i systemet. Efter udførelsen af usecasen er der ikke sket en ændring i programmet ud over at råvarer-menuen er åbnet.

Use case 3

Usecase: Recept Administration.
ID: UC3
Kort beskrivelse: Farmaceut opretter en recept.
Primær bruger: Farmaceut (F).
Forudsætninger: Farmaceuten er logget ind.
<p>Main flow:</p> <ol style="list-style-type: none">1. F vælger menuen "Recept".2. F vælger menuen "Opret Recept".3. F skriver navn, ID og en sekvens af receptkomponenter.4. F får vist den nye Recpts oplysninger.5. F bekræfter at oplysningerne er korrekte og returneres til recept-menuen.
Postconditions: F kan nu afslutte programmet eller vælge videre fra recept-menuen.
<p>Alternativt flow:</p> <ol style="list-style-type: none">1. F indtaster et ugyldigt input.<ol style="list-style-type: none">1.1. A bliver bedt om at ændre inputtet.2. A indtaster en receptkomponent der ikke eksisterer.<ol style="list-style-type: none">2.1. A får af vide at receptkomponent ikke er i systemet.2.2. A kan skrive en ny receptkomponent og fortsætte oprettelsen.

Denne usecase begynder, når en Farmaceut vil oprette en ny recept i systemet. Efter udførelsen af usecasen er der oprettet en Recept.

Usecase 4

Usecase: Råvarebatch Administration
ID: UC4
Kort beskrivelse: Værkfører opretter en råvarebatch.
Primær bruger: Værkfører (V).
Forudsætninger: Værkføreren er logget ind.
<p>Main flow:</p> <ol style="list-style-type: none">1. V vælger menuen "Lagerstyring".2. V vælger menuen "Opret Råvarebatch".3. V skriver navn, ID og mængde på Råvarebatchen.4. V får vist den nye Råvarebatch' oplysninger.5. V bekræfter at oplysningerne er korrekte og returneres til råvarebatch-menuen.
Postconditions: V kan nu afslutte programmet eller vælge videre fra råvarerbatch-menuen.
<p>Alternativt flow:</p> <ol style="list-style-type: none">1. V indtaster et ugyldigt input.<ol style="list-style-type: none">1.1. V bliver bedt om at ændre inputtet.2. V indtaster et Råvarebatch der allerede eksisterer.<ol style="list-style-type: none">2.1. V får af vide at Råvarebatchen allerede er i systemet.2.2. V kan vælge at rette oplysningerne eller returnere til råvarebatch-menuen.

Denne usecase begynder, når en Værkfører vil oprette en ny råvarebatch i systemet. Efter udførelsen af usecasen er der oprettet en råvarebatch.

Usecase 5

Usecase: Produktbatch Administration
ID: UC5
Kort beskrivelse: Værkfører printer en ordre til en Operatør.
Primær bruger: Værkfører (V).
Forudsætninger: Værkføreren er logget ind.
<p>Main flow:</p> <ol style="list-style-type: none">1. V vælger menuen "Lagerstyring".2. V vælger menuen "Opret Produktbatch".3. V skriver ID og recept-id på Produktbatch'en.4. V får vist den nye Produktbatch' oplysninger.5. V bekræfter at oplysningerne er korrekte.6. V printer Produktbatch'en og returneres til produktbatch-menuen.
Postconditions: Operatøren kan nu begynde ordren.
<p>Alternativt flow:</p> <ol style="list-style-type: none">1. V indtaster et ugyldigt input.<ol style="list-style-type: none">1.1. V bliver bedt om at ændre inputtet.2. V indtaster et Produktbatch-id der allerede eksisterer.<ol style="list-style-type: none">2.1. V får af vide at Produktbatchen allerede er i systemet.2.2. V kan vælge at vise/printe oplysningerne eller returnere til produktbatch-menuen.

Usecase 6:

Usecase: Afvejning
ID: UC6
Kort beskrivelse: Operatør vejer en råvare.
Primær bruger: Operatør (O).
Forudsætninger: Operatøren er blevet tildelt en udprintet Produktbatch af Værkføreren.
<p>Main flow:</p> <ol style="list-style-type: none">1. O logger ind i en veje-terminal, ved indtastning af sit ID.2. O bliver valideret af terminalen (hans navn bliver vist).3. O indtaster ID fra den udprintet Produktbatch.4. Terminalen angiver at der skal anbringes en beholder på vægten.5. O nulstiller vægten og placerer beholderen på den.6. O tarerer vægten.7. Terminalen angiver hvilket råvare-id Operatøren skal afveje.8. O henter råvaren og indtaster dens råvarebatch-id.9. O placerer råvaren på vægten.10. Terminalen angiver at afvejningen er gennemført.
Postkonditions: Operatøren kan viderebehandle ordren.
<p>Alternativt flow:</p> <ol style="list-style-type: none">1. O indtaster et ugyldigt input.<ol style="list-style-type: none">1.1. O bliver bedt om at ændre inputtet.2. O indtaster et Produktbatch-id der ikke eksisterer.<ol style="list-style-type: none">2.1. O får af vide at Produktbatchen ikke er i systemet.2.2. O returnerer til pkt. 3 i Main flow.

3. O indtaster et råvarebatch-id der ikke svarer til den pågældende vare eller ikke eksisterer i systemet.

3.1 O får af vide at Råvarebatch'en ikke er i systemet.

4. Råvarens vægt er ikke inde for tolerancen.

4.1 O justerer mængden af råvaren og vejer den igen, iterativt indtil tolerancen er acceptabel.

4.2. O returnerer til pkt. 10 i Main flow.

5. Ved endt vejning er brutto-kontrollen ikke korrekt.

5.1 O skal starte forfra ved pkt. 3 i Main flow.

Designsekvensdiagram

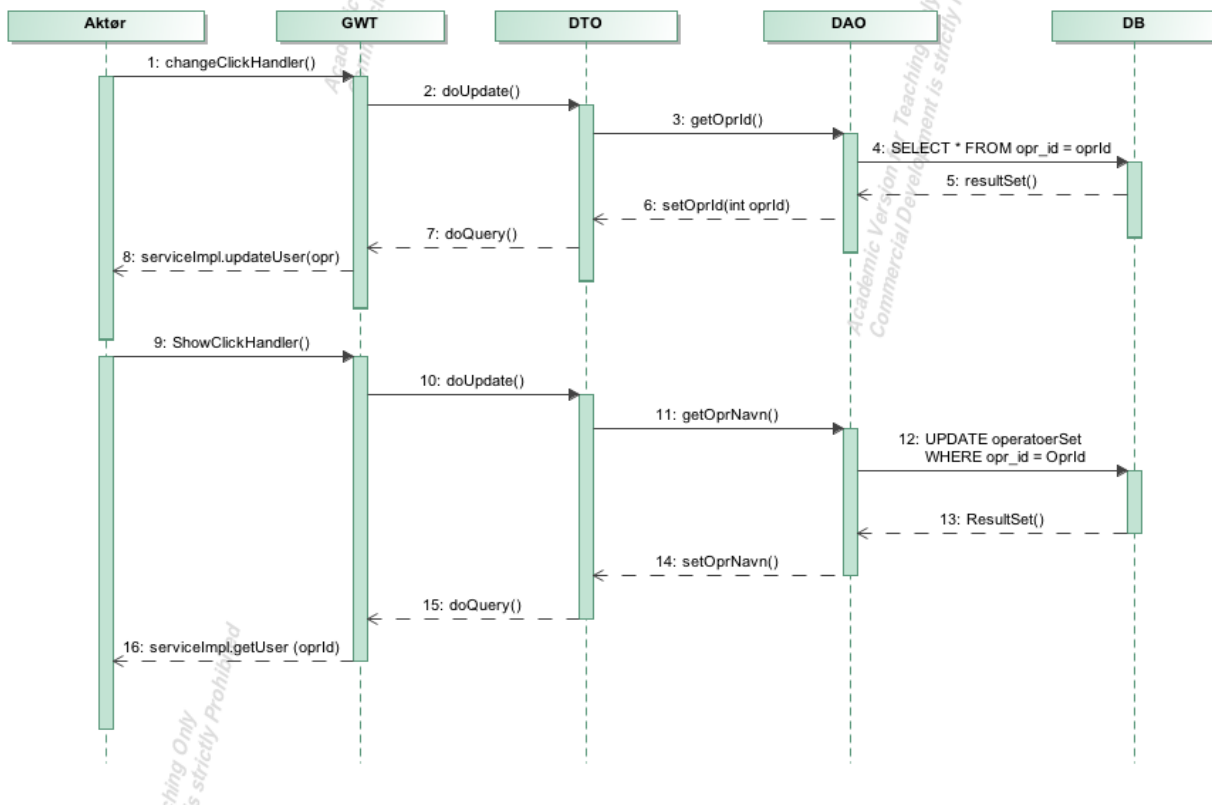
Vores DSD giver et simpelt overblik over forløbet, når en administrator skal foretage en ændring i én af operatørernes oplysninger.

Administratoren starter med kalde på oplysninger for en bestemt operatør, og disse oplysninger bliver gemt i `doUpdate()`, når der hentes fra databasen via datalaget.

Efterfølgende bruges metoden `doQuery()` til at generere det endelige output, via web-serveren til vores GWT, og sendes til administratoren. Når administratoren har fået de oplysninger om operatøren, og har foretaget en ændring, benyttes `changeClickHandler` til at opdatere brugeren gennem web interfacet, og dermed bliver sendt med Clickhandler efter opdatering af oplysningerne i inputfelterne.

Oplysningerne bliver hentet fra databasen, som så vil blive indsat i vores `OperatoerDTO` objekt. Ændringer som administratoren har foretaget vil blive gemt ved at kalde `getOprNavn()` så vores `OperatoerDAO`, som så laver oplysningerne om til SQL

sætninger, der så sendes til databasen.



Design sekvens diagram ASE

ASEControlleren starter med at forbinde med vægten igennem WeightController (WC) og så derefter databasen igennem DBController (DBC).

Nu spørger ASE WC om ID på operatøren, derefter sender den ID'en videre til DBC for at få navnet på operatøren, navnet sendes til vægten hvor brugeren godtager den.

Nu beder ASE om at få PBID fra WC og ASE bruger så PBID til at få fat i navnet på recepten fra DBC, hvilket den så sender til WC igen.

Nu går programmet ind i en løkke hvor selve afvejningen sker.

Først beder ASE om at WC tømmer vægten, derefter ændre den status på PB og opdater i DBC.

ASE beder WC om at få fat i vægten og bagefter beder den om at Tara vægten.

Tara gemmes af DBC og derefter for man fat i navnet på det næste råvare.

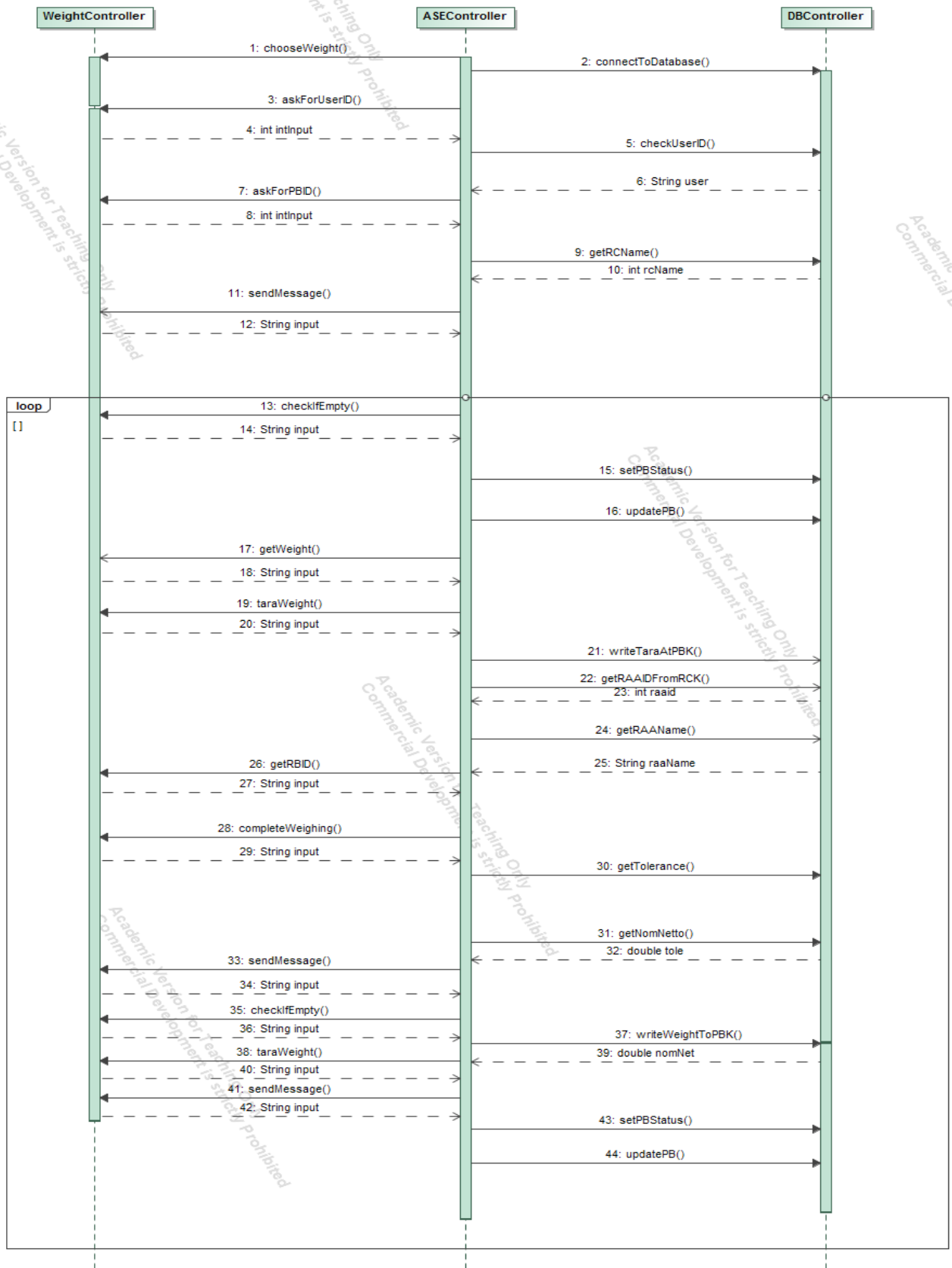
Selve afvejningen sker nu. Efter afvejning for ASE fat i tolerance og nomNetto fra DBC som bruges til at tjekke om vægten er acceptable.

ASE beder nu WC om at tømme vægten og så skriver den til DBC for at ændre PBK.

Til sidst i løkken beder ASE WC om at tara vægten.

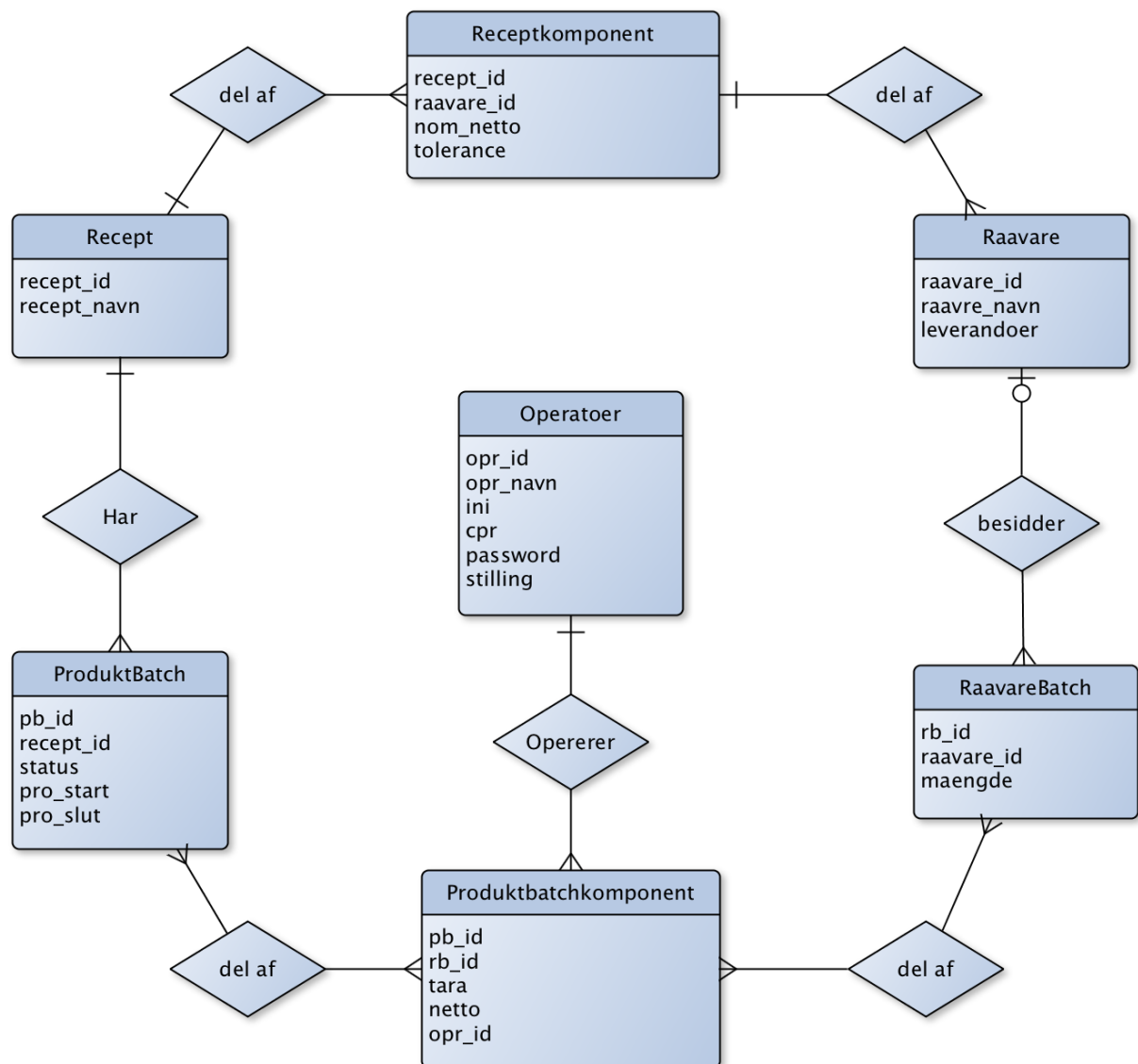
Når løkken er gentaget for alle ingredienser beder ASE DBC om at opdatere PB.

Programmet starter nu forfra og giver mulighed for det næste operatør for at begynde deres produktion.



Database

Entity-forholds model (ER diagram)



På den ovenstående billede har vi vores database design. Dette database design viser alle de tabeller vi har i MySQL database, som består af raavare, raavarebatch, produktbatchkomponent, operatoer, produktbatch, recept og receptkomponent. Vores nuværende database af bygget op af databasen vi brugte i kurset 02327, med en del ændringer i DDL og DML for at opfylde kundens krav.

Noget af det vigtige vi har tilføjet er de rettigheder, som de forskellige operatører har, som afhænger af deres stilling. I dette tilfælde har vi bestemt en stilling for hver operatør. Stillingen er en restriktion for bl.a. at farmaceuter ikke har administrationsrettigheder, og således kan man undgå misbrug af systemet. Derudover er det også muligt at finde en tidligere ansat operatører, eller hvis operatøren er sendt til et andet firma i et andet land, vil denne operatør også være inaktiv af sikkerhedsmæssige grunde, og dette kan ses ved at operatøren har stilling 0. Vi anvender vores ER diagram til at illustrere relationer mellem de forskellige tabeller. Vi anvender ER diagrammet til at oprette relations databasen. Da dette vil være nemmere at se hvilken vej relationerne skal trækkes, samt de relations typer der skal oprettes. I diagrammet viser vi hvilken type af entitet, vi har med at gøre, og relationerne indbyrdes er antallet af forekomster af entiteter (cardinality). Et eksempel på cardinality, kan vi se at Recept indeholder mange Raavarer, og mange Raavarer indeholder mange Recepter. Dvs. vi har med at gøre med en mange til mange syntaks, hvor vi har en binær relation.

Data Access Object (DAO)

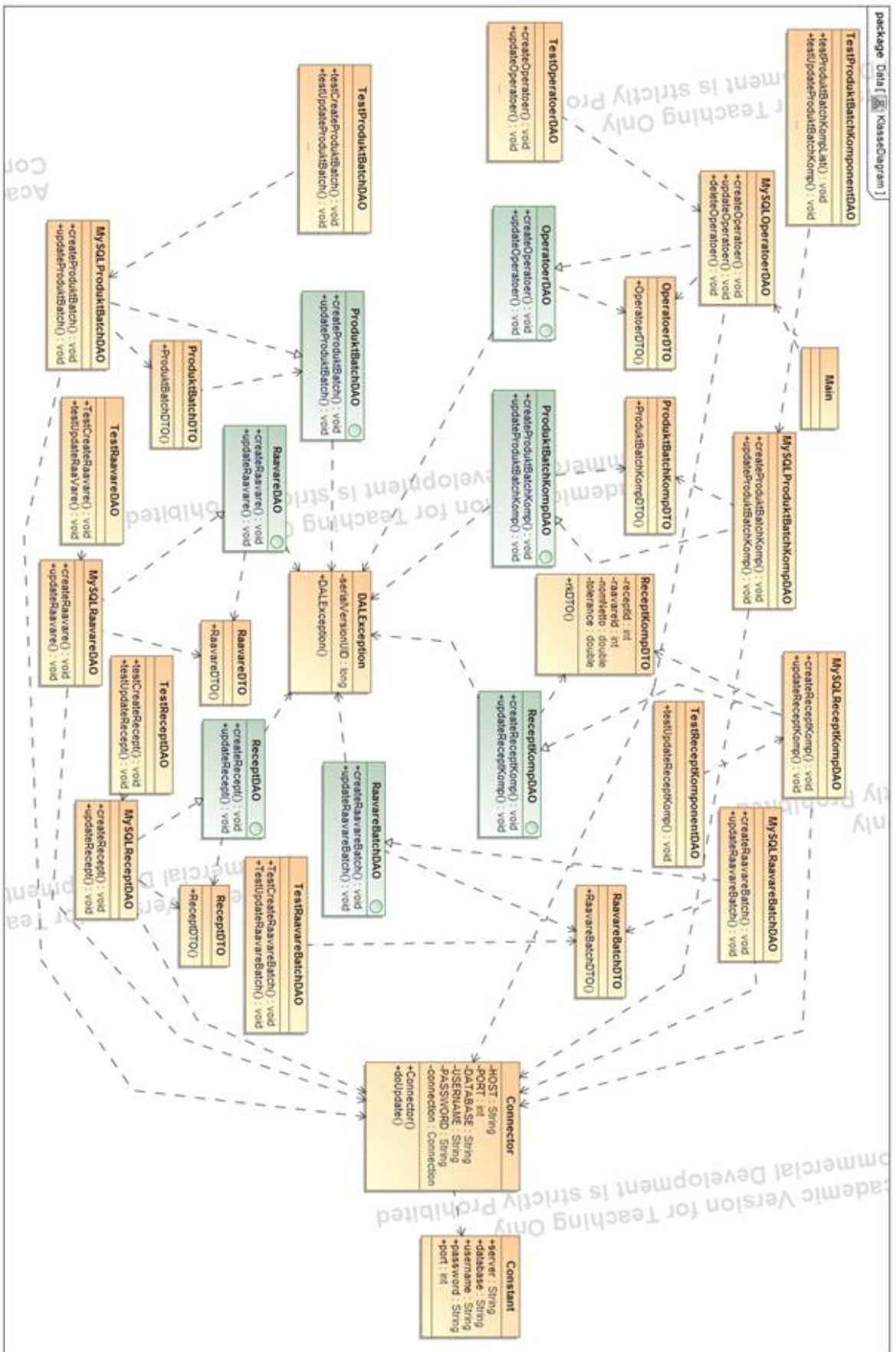
DAOernes funktion er at skabe adgang til de informationer, som er gemt i databasen vha. SQL forespørgsler for brugerens perspektiv. DAOerne har til formål at hente tabeller og tupler til visning for brugeren, samt opdatere tupler, hvor der er felter der skal ændres, slettes og laves nye tupler i tabellerne.

For at kunne tilgå og bruge databasen, har vi haft behov for at kunne lave CRUD operationer på databasen, som er create, read, update og delete.

Dette har vi gjort ved at lave vores egne MySQLDAO klasser, som er oprettet i pakken `cdio3.server.DB.DAO`. Der er oprettet et DAO interface for alle vores database tabeller med tilhørende implementation af disse.

Vores create, update og get metoder er det der kan ses i MySQLDAO. Til disse fanger vi også to forskellige exceptions med `DAException` og `SQLException`.

Klassediagrammet til databasen



Dette klassediagram viser de forskellige klasser og interfaces som er en del af vores java program som kan tilgå databasen. Vi har forskellige klasser som fx de DAO'er, DTO'er. De forskellige attributter er selvfølgelig private da de bruges i konstruktørerne. For hver DAO har vi lavet test for at være sikre på at der er god interaktion med databasen

Normaliseringsanalyse

For at sikre at der kun er én entydig måde at tilgå en information i databasen (altså, ingen redundans), sikrer vi at hver tabel opfylder betingelserne for at være i 3. normalform. Dvs. at hver tabel SKAL og KUN kan finde en af dets attributter vha. den supernøgle som den pågældende tabel har. Normalt vil 3. normalform give et godt design med minimal redundans og rimelig sikkerhed for konsistens.

1. Normalform (1NF)

- Tabellen har en primærnøgle og ingen repeterende grupper (felter).

2. Normalform (2NF)

- Tabellen er på 1NF og alle “ikke nøgle attributter”, afhænger af alle kandidatnøgler der er i tabellen.

3. Normalform (3NF)

- Tabellen er på 2NF og “ikke nøgle attributter” er IKKE transitivt afhængige af en kandidatnøgle.

Design og organisering af tabeller i databasen

Primærnøglen er vores unikke nøgle, som kan finde frem til de data der står i samme række vha. primærnøglen. Det er vores unikke id, som ikke kan ændres.

Operatoer

I denne tabel ses alle de værdier en operatoer til vores system skal indeholde. Et operatoer_navn, operatoer_id, initialer, cprnummer, password og stilling. Ved hver variabel er der blevet givet forskellige datatyper, med forskellige værdier. Da den respektive operatør id er helt unikt, har vi valgt at gøre denne til primærnøgle.

Raavare

Tabellen angiver de oplysninger vi har brug for på hver en raavare. Disse oplysninger er raavare id, raavare_navn og en leverandør for raavaren. Vi har valgt at bruge raavare id'et, som vores primærnøgle, da det er unik værdi for raavarene.

Recept

Vores recept tabel indeholder et receptid og et receptnavn. Id'et har vi gjort til vores primærnøgle som er unikt.

Produktbatch

Vores produktbatchtabel består af pb_id som er primærnøgle og recept_id som fremmednøgle der peger på recept tabellens primærnøgle recept_id. Attributten status viser om en produktion ikke er sat i gang ved 0, produktionen er i gang ved 1 og færdig produkt ved 2. Endvidere har vi starttid og startdato samt sluttid og slutdato for produktionen, som udtrykkes ved attributterne pro_start og pro_slut.

Raavarebatch

Vores raavarebatch tabel består af et raavarebatch_id, et raavare_id og en mængde. Raavareid'et er der behov for, så man via dette kan finde frem til de forskellige raavarer, som bruges i vores raavarebatch.

Mængden har fået datatypen double, da værdien skal være et decimaltal.

Vores primærnøgle er vores raavarebatch_id og vores fremmednøgle er raavare_id, da den peger på primærnøglen raavare_id i raavaretabellen.

Data Access Layer (DAL)

Data Access Layer (DAL), er laget mellem applikationen og databasen. Laget består af Data Access Objekt (DAO) og en Data Transfer Objekt (DTO). DAL har til opgave at adskille applikationen og databasen, og have alt logik placeret i dette lag.

Data Transfer Object (DTO)

Et java objekt som bruges til at meddele informationer mellem systemerne. Der eksisterer DTO for hver tabel. Det indeholder ikke nogen system relateret logik, som ville kræve testning.

En DTO er et java objekt der udover en konstruktør til at definere dataindhold også indeholder set og get metoder til at rette og hente data i disse objekter, der er oprettet en DTO for hver tabel i databasen. DTO'erne har forbindelse til DAO'erne til opbevaring af data i de respektive områder. De indeholder kun funktionalitet til opbevaring af en tabels række data, hvad vil sige en post per DTO.

Data Access Object (DAO)

DAOernes funktion er at skabe adgang til de informationer, som er gemt i databasen vha. SQL forespørgsler for brugerens perspektiv. DAOerne har til formål at hente tabeller og tupler til visning for brugeren, samt opdatere tupler, hvor der er felter der skal ændres, slettes og laves nye tupler i tabellerne.

For at kunne tilgå og bruge databasen, har vi haft behov for at kunne lave CRUD operationer på databasen, som er create, read, update og delete.

Dette har vi gjort ved at lave vores egne MySQLDAO klasser, som er oprettet i pakken `cdio3.server.DB.DAO`. Der er oprettet et DAO interface for alle vores database tabeller med tilhørende implementation af disse.

Vores create, update og get metoder er det der kan ses i MySQLDAO. Til disse fanger vi også to forskellige exceptions med `DALException` og `SQLException`.

Google Web Toolkit (GWT)

GWT er et open source sæt værktøjer som blev udarbejdet af Google i perioden fra 2006 til 2014 fordi Google ansår det nødvendigt, at lave et bro mellem Java og web-udviklingen. Hvilket vil sige at vha. GWT kan webudviklere skabe og vedligeholde komplekse JavaScript front-end applikationer i Java.

I GWT findes der forskellige plugins som hjælpe en på vej i de forskellige IDEer. I vores projekt har vi brugt Eclipse, og derfor fik installeret et plugin fra. Der findes forskellige open source plugins som gør det nemmere at arbejde med GWT i forskellige IDEs. I vores tilfælde bruger vi et plugin til Eclipse som kan hentes fra <https://dl.google.com/eclipse/plugin/4.4> og installeres gennem Eclipses Software Installering

Vi bruger GWT til at udvikle vores web applikation som gør det muligt for de forskellige brugere at udføre deres respektive opgaver.

Design af GUI

I CDIO 3 har vi arbejdet med at lave en GUI i GWT. Hertil har vi lavet et nyt design, da designet af GUI'en i CDIO3 er bygget op efter et mindre program fra en udleveret GWT tutorial. Designet skal nu være med en klar opdeling af funktionalitet mellem en Menu klasse og en View klasse. De enkelte menuer i GUI'en hænger sammen vha. en Menuklasse og en Viewklasse. Tidligere har Menu klassen stået for forbindelser til og fra de to klasser, og View har stået for funktionalitet, clickhandlers og opsætningen af menuen. Hertil er det vores ide at holde Viewklasserne til kun at kunne opsætte den grafiske del af menuen, hvorimod Menuklasser skal indeholde clickhandlers, funktionalitet og forbindelser.

Yderligere vil vi lave en tempplate, til opsætning af klasser, så der ikke oprettes en Viewklasse for hver Menuklasse, men at der i stedet kan hentes dele ned fra en mere overordnet Viewklasse til at hænge sammen med en Menuklasse, hertil vil det være mere overskueligt at implementere flere Viewklasser end en enkelt kæmpe Viewklasse fyldt med alt hvad vi skal bruge af GWT metoder til design af GUI'en. Tempplaten skal også have nogle grafiske indstillinger, som standardiserer vores overordnede design igennem hele GUI'en.

Vi har designet hele GUI'en, såsom vi mente var mest brugervenlig, og samtidig designet enkelte dele intuitivt. F.eks. når en administrator vil have vist programmets brugerliste, udvælger administratoren brugere ud fra navn og rolle i systemet fra en

fold ud liste, der indeholder alle brugere. Hvorefter administratoren kan vælge at se detaljer om brugeren, hvorefter alt information på den udvalgte bruger vil blive vist i et popupvindue. Her bruger vi forskellige design dele til let at udvælge og se detaljer omkring den enkelte bruger i systemet. Evt. kunne man vælge brugerens rolle fx om brugeren var operatør, og hertil få vist en liste med navne på alle operatørerne, så det var nemmere for brugeren at udvælge den rigtige bruger. Men hertil skal det siges at vores mockobject database ikke indeholder mere end en håndfuld brugere, så det vil i vores program ikke være nemmere for brugeren.

Angående sikkerhed i vores program, så er det vigtigt at en operatør fx ikke kan udføre administratoropgaver. Igennem passwordvalideringen, som valideres gennem controlleren i vores mockobject database, så opsættes GUI'en efter hvilken rolle brugeren har. Så administratormenuen vil aldrig være til stede i en operatørs programvindue af GUI'en.

Klassen MainView er menuklassen og MenuView er viewklassen for deres sammenhæng. Igennem loginklasserne valideres passwordet vha. integer clearanceLvl fra operatørDTO'en, som henter information på den enkelte bruger fra vores mockobject database, og integeren bruges i klassen MenuView. Udfra clearance level 1 - 4 opsættes GUI'en for hver enkelt rolle i MenuView klassen. Denne klasse opsætter for hvert level en button, som åbner en ny menu når der klikkes. Der er altså 4 buttons i GUI'en hvis en administrator logger ind, og en enkelt button i GUI'en hvis en operatør logger ind, alt sammen indstansieret igennem klasserne MenuView og MainView. Fordelen ved denne løsning er, at vi kun bruger de samme to klasser uanset hvem der logger ind til opsætning af GUI'en, og da en operatør aldrig har menuen til fx at udføre administratoropgaver, er det ikke nødvendigt at have sikkerhed og clearance andre steder i programmet såsom sikkerhedstjek i controlleren for hver gang denne anvendes. Der er dermed et sikkerhedstjek i login, og efterfølgende er programmet ens for alle, men der indstansieres forskellige menuer i GUI'en an på brugerens rolle, som giver adgang til forskellige dele af programmet. Dette vil dog ikke sige at programmet ikke kan hackes, for når der kun

er et enkelt sikkerhedstjek, er programmet ikke særligt sikkert. Løsningen med validering flere steder i controlleren, vil sikkerhedsmæssigt være en overlegen løsning, men vi har valgt denne i stedet, da det implementeringsmæssigt var let overkommeligt, samtidig med at vi har sikkerhed implementeret, og den giver overskuelighed i vores GUI.

Implementering af GWT

Grundet tidspres har vi gået tilbage til den GWT tutorial vi anvendte til CDIO3, men dette betyder også at programmet kunne gøres mindre vha. Templates. Vi har ligeledes ikke den opdeling af forbindelse, funktionalitet og opsætning af menuer, som vi havde planlagt. Løsningen med et enkelt passwordvalidering passer her godt, da det implementeringsmæssigt var let overkommeligt. Havde vi valgt at have databasen fyldt med validering ville vi have haft meget travlt med at få programmet på plads. Vi har holdt fast i den brugervenlige GUI, og lagt arbejde i denne.

GWT'en mangler sammenkobling mellem controller og database. GUI'en er sat op, men den mangler delmenuer. Vi har holdt fast i vores brugervenlige design, som kan ses i administratormenuen, som er delvist opsat med funktionslaget, såsom i ret bruger hvor alt er implementeret, lige med undtagelse af at kaldet kun går fra viewet til mainview og tilbage igen, dvs. Uden information. Noget er altså sat op uden kun visuelt, og andet er ikke sat op mens dele af GUI'en har forbindelse til mackobject database, og controlleren og virker korrekt.

ASE

ASE'en er designet til at skabe kommunikation mellem vægt og database, der ud over checker den også at den information der bliver sendt er rigtigt.

ASE'en starter med at skabe en forbindelse til databasen, derefter vælger man om man skal bruge en normal vægt eller simulator.

Når man har valgt hvad man vil forbinde med skaber ASE'en en forbindelse.

ASE'en bruges til at fortolke den information der bliver sendt fra vægten, så det kan sammenlignes med databasen information. fx hvis vi sender en "S\r\n" til vægt vil den returnere "S S vægten i decimal \r\n" siden vægten er i en String form kan vi ikke sammenligne den med den double vi for tilbage fra databasen, så det vi gør er at fjerne "S S " og "\r\n" og så parser vi resten af String til en double.

Nogle gange har vægten sendt en fejlbesked som vi ikke har kunne finde ud af hvorfor, for at komme uden om dette problem har vi lavet et rekursive kald som bliver brugt hvis der bliver sendt en fejlbesked.

Det recursive kald er ikke en særlig god løsning da den endelig ikke håndtere den fejl der bliver sendt, men bare brute force sig igennem programmet.

Den samme løsning er blevet brugt i TaraWeight metoden af samme oversag. En ting vi har lagt mærke til angående TaraWeight er at vægten er så følsom at tryk på bordet kan få vægten til at svinge, hvilket er en af årsagerne til at Tara af vægt fejler, da programmet ikke kan finde ud af den svingende vægt.

Når ASE'en har forbindelse til vægten vil den fungerer som en guide til hvad operatøren skal gøre, dette fungerer som et double check af recept da operatøren står med en printet udgave af recepten.

Hvis recepterne ikke stemmer over ens er der en fejl i systemet.

Hvis et svar bliver sendt fra vægten som ikke er acceptable, i forhold til den information der er i databasen, vil ASE'en skrive en besked til vægten og bede operatøren om at gentage fx afvejning af produkt x.

ASE klassen er vores hoved klassen som kalder til de to andre klasser, henholdsvis DBController og WeightController, dette gøres ved brug af interfaces.

Både DBControlleren og WeightControlleren er designet til at fungere sammen med vores nuværende database og Mettler vægten, hvis en af disse to ønskes udskiftet, fx til en anden vægt, skal man også skrive en ny

WeightController. På grund af opbygningen skal man dog ikke ændre hverken DBController eller ASE. Man skal dog huske at ændre IP'en inden i ASE klassen når man forbinder til en ny vægt.

Det er desværre ikke lykkedes os at implementere alt til ASE'en

Mangler:

Vores vægt sender en RM20 besked og beder brugeren om at afveje og så derefter trykke ok.

RM20 beskeden blokerer desværre for at vægten kan vise hvor en given genstand vejer.

Dette blev realiseret for sendt til at vi kunne nå at ændre det.

Vi kunne ikke få "D text \r\n" eller "P111 text \r\n" til at virke og lave derfor nogle ekstra RM20 kald som overtog funktion, som fx at skrive navnet på recepten til operatøren.

numberOfIngred metoden som skulle give os antallet af loops til vores for løkke for hver recept bruger desværre også et forkert kald, og er derfor blevet ukommenteret.

I stedet har vi sat variabelen til tre da ingen af vores receptor har flere råvarer, dette er gjort udelukket for at kunne arbejde videre og burde ikke være en del af den endelige kode.

Programmet stopper desværre når du når til afvejning af råvarer nummer to. Vi har ved hjælp af debug fundet frem til at RM20 beskeden bliver sendt, den bliver dog ikke vist på vægten display,

vægten sender et svar tilbage, men svar nummer to som burde komme på en RM20 besked kommer aldrig. det er desværre ikke lykkedes os at finde ud af hvorfor dette sker.

Dataen der bliver brugt i løbet af proceduren mangler at bliver logget ordentligt, dette nåede vi aldrig til at vi sad fast i at prøve på at få vægten til at afveje råvarer nummer to.

Vejledning til brug af ASE

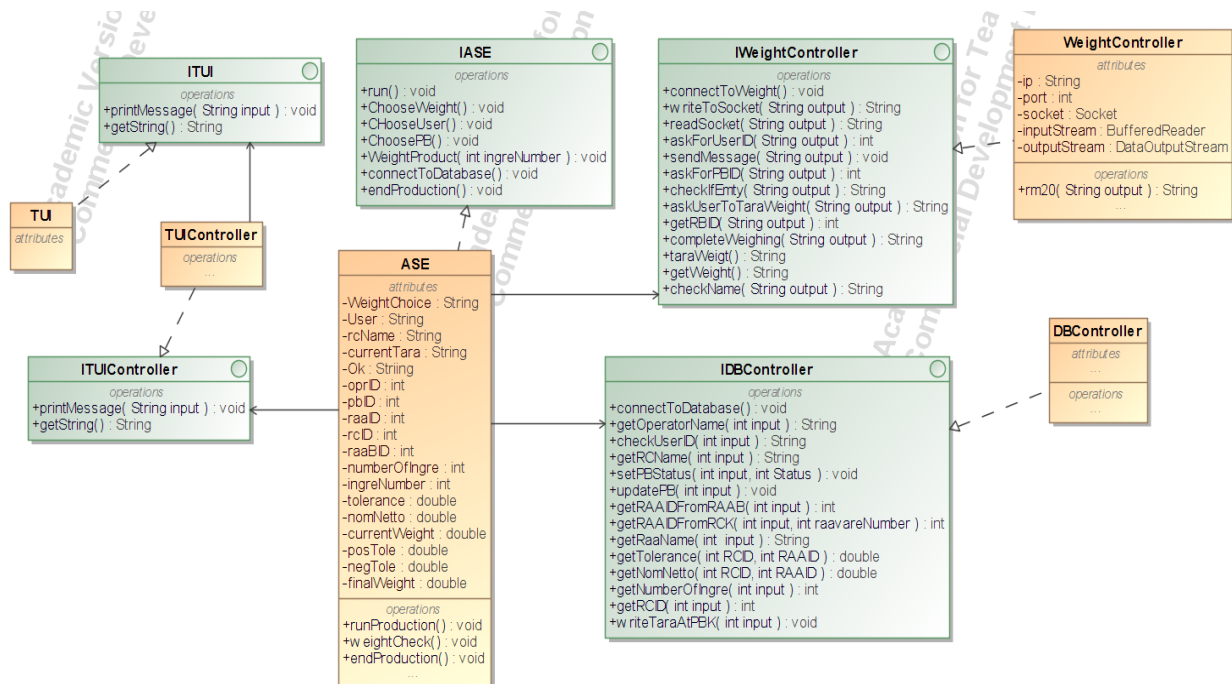
Det første man gør er at forbinde vægten til computeren, derefter starter man klassen Run som så beder dig om at vælge hvilken vægt du vil bruge, når du har valgt det følger man de beskeder der bliver skrevet på vægten.

Alt efter hvilken fysisk vægt man bruge kan det være man skal ændre IP'en inden i ASE klassen.

Hvis du forbinder til vægt simulatoren skal du først starte simulatoren ved at åbne for cmd og derefter skrive java-jar CDIO2.jar. Du kan nu starte ASE og vælge Vægt simulator, derefter vil det foregå på samme måde som med den fysiske vægt.

Klassediagram ASE

I nedenstående kan vi se vores ASE klassediagram, som giver os et overblik over de forskellige klasser og deres interfaces, derudover kan vi også se hvordan de kommunikere med hinanden. Hvis man ser på ASEcontroller kan man finde ud af, at den kender alle de andre klasser derfor opstår der en høj kobling, dvs. at alt bliver sat i gang i ASEcontroller derfor er den afhængig af de andre klasser.



Vægtsimulator

ClientController/CC:

Dette er klassen der styrer vægtens menu. Klassen kommunikere med tui og giver de forskellige værdier som bliver skrevet ind af brugeren. Klassen kommunikere også men WeightControlleren så CC kan skrive til putty brugeren.

IOController/IOC:

Denne klasse står for kontrol af fjernstyring. Klassen kommunikere med WeightData og WeightController.

WeightController/WC:

Denne klasse er den færdig skrevet skelet som vi fik udleveret. Klassen venter på et instream fra putty klienten og skriver derefter tilbage.

Der ud over har vi en TUI klassen som står for at skrive TUIen og så er der en weightData klassen som holder styr på vægten og andet information i programmet.

Test

I vores brug af test har vi valgt at fokusere på at vise, at vores korrekthed i implementationen virker efter hensigten. Vi har valgt at teste de vigtigste metoder med JUnit, der har en afgørende rolle for funktionalitet i vores program, for at kontrollere fremgangsmåden i de forskellige funktionaliteter.

Vores ønskede test vil se således ud:

Test case

Preconditions:

1. Opret operatør
2. Opdater navn
3. Hent operatør
4. Hent operatørliste

Test:

1. Opretter en operatør
2. Eksisterer der en operatørliste

Postconditions:

1. Kontroller operatør
2. Kontroller operatørliste

```
@Before
public void connect()
{
    try {
        new Connector();
    }
```

```
    } catch (Exception e) {
```

```
@Test
```

```
    public void TestGetOperatoer() throws DALError {
```

```
        OperatoerDTO testOperatoer = null;
```

```
        List<OperatoerDTO> operatoerList = opDAO.getOperatoerList();
```

```
        int ID = operatoerList.get(0).getOprId();
```

```
        testOperatoer = opDAO.getOperatoer(ID);
```

```
        OperatoerDTO actual = testOperatoer;
```

```
        OperatoerDTO expected = operatoerList.get(0);
```

```
        boolean theSame = true;
```

```
        if (actual.getOprId()    != expected.getOprId())
```

```
            theSame = false;
```

```
            if (!actual.getOprNavn().equals(expected.getOprNavn()))
```

```
                theSame = false;
```

```
            if (!actual.getCpr().equals(expected.getCpr()))
```

```
                theSame = false;
```

```
            if (!actual.getIni().equals(expected.getIni()))
```

```
                theSame = false;
```

```
            if (!actual.getPassword().equals(expected.getPassword()))
```

```
                theSame = false;
```

```
        assertTrue(theSame);
```



```
}
```

```
@Test
```

```
public void createOperator() throws DALException{
```

```
    List<OperatorDTO> list = opDAO.getOperatorList();
```

```
    int currentHighestID = list.get(list.size()-1).getOprId();
```

```
    int expected =
```

```
opDAO.getOperatorList().size()+1 opDAO.createOperator(new Operator  
DTO(currentHighestID+1, "Kim Larsen", "KL", "090554-1911", "Ss123sS", 1));
```

```
    int actual = opDAO.getOperatorList().size();
```

```
}
```

```
}
```

```
@Test
```

```
public void createOperator() throws DALException{
```

```
    List<OperatorDTO> list = opDAO.getOperatorList();
```

```
    int currentHighestID = list.get(list.size()-1).getOprId();
```

```
    int expected = opDAO.getOperatorList().size()+1;
```

```
    opDAO.createOperator(new
```

```
OperatorDTO(currentHighestID+1, "Kim Larsen", "KL", "090554-1911", "Ss123sS",  
1));
```

```
    int actual = opDAO.getOperatorList().size();
```

```
@Test
```

```
public void updateOperator() throws DALException {
```

```
    OperatorDTO opDTO = null;
```

```
    String expected = "Kim Larsen";
```

```

        try {
            opDTO = opDAO.getOperatoerList().get(0);
            opDTO.setOprNavn(expected);
            opDAO.updateOperatoer(opDTO);
        } catch (DALErrorException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw new DALErrorException(e);
        }
        String actual = opDTO.getOprNavn();
        assertEquals(expected, actual);
    }
}

```

I det nedenstående screenshot har vi et eksempel på updateOperatoer() metoden, hvor vi deri forventer (expected) et navn vi har angivet som “Kim Larsen”, og sætter (setOprNavn()) med expected som funktion af metoden. Som resultat sammenligner vi (assertEquals) det forventede og den aktuelle værdi, og således vil vi se om vores program har sat navnet ved en update.



```

@Test
public void updateOperatoer() throws DALErrorException {
    OperatoerDTO opDTO = null;
    String expected = "Kim Larsen";






    try {
        opDTO = opDAO.getOperatoerList().get(0);
        opDTO.setOprNavn(expected);
        opDAO.updateOperatoer(opDTO);
    } catch (DALErrorException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        throw new DALErrorException(e);
    }
    String actual = opDTO.getOprNavn();
    assertEquals(expected, actual);
}

```

Her er resultatet af vores test, hvor alt virker efter hensigten.

Runs: 4/4  Errors: 0  Failures: 0



- ▼  test01917.TestOperatoerDAO [Runner:
 -  updateOperatoer (0,197 s)
 -  getOperatoerList (0,006 s)
 -  createOperatoer (0,018 s)
 -  TestGetOperatoer (0,009 s)

Google Web Toolkit (GWT)

GWT er et open source sæt værktøjer som blev udarbejdet af Google i perioden fra 2006 til 2014 fordi Google ansår det nødvendigt, at lave et bro mellem Java og web-udviklingen. Hvilket vil sige at vha. GWT kan webudviklere skabe og vedligeholde komplekse JavaScript front-end applikationer i Java.

I GWT findes der forskellige plugins som hjælpe en på vej i de forskellige IDEer. I vores projekt har vi brugt Eclipse, og derfor fik installeret et plugin fra. Der findes forskellige open source plugins som gør det nemmere at arbejde med GWT i forskellige IDEs. I vores tilfælde bruger vi et plugin til Eclipse som kan hentes fra <https://dl.google.com/eclipse/plugin/4.4> og installeres gennem Eclipses Software Installer

Vi bruger GWT til at udvikle vores web applikation som gør det muligt for de forskellige brugere at udføre deres respektive opgaver.

Design af GUI

I CDIO 3 har vi arbejdet med at lave en GUI i GWT. Hertil har vi lavet et nyt design, da designet af GUI'en i CDIO3 er bygget op efter et mindre program fra en udleveret GWT tutorial. Designet skal nu være med en klar opdeling af funktionalitet mellem en Menu klasse og en View klasse. De enkelte menuer i GUI'en hænger sammen vha. en Menuklasse og en Viewklasse. Tidligere har Menu klassen stået for forbindelser til og fra de to klasser, og View har stået for funktionalitet, clickhandlers og opsætningen af menuen. Hertil er det vores ide at holde Viewklasserne til kun at kunne opsætte den grafiske del af menuen, hvorimod Menuklasser skal indeholde clickhandlers, funktionalitet og forbindelser.

Yderligere vil vi lave en template, til opsætning af klasser, så der ikke oprettes en Viewklasse for hver Menuklasse, men at der i stedet kan hentes dele ned fra en mere overordnet Viewklasse til at hænge sammen med en Menuklasse, hertil vil det være

mere overskueligt at implementere flere Viewklasser end en enkelt kæmpe Viewklasse fyldt med alt hvad vi skal bruge af GWT metoder til design af GUI'en. Tempplaten skal også have nogle grafiske indstillinger, som standardiserer vores overordnede design igennem hele GUI'en.

Vi har designet hele GUI'en, såsom vi mente var mest brugervenlig, og samtidig designet enkelte dele intuitivt. F.eks. når en administrator vil have vist programmets brugerliste, udvælger administratoren brugere ud fra navn og rolle i systemet fra en fold ud liste, der indeholder alle brugere. Hvorefter administratoren kan vælge at se detaljer om brugeren, hvorefter alt information på den udvalgte bruger vil blive vist i et popupvindue. Her bruger vi forskellige design dele til let at udvælge og se detaljer omkring den enkelte bruger i systemet. Evt. kunne man vælge brugerens rolle fx om brugeren var operatør, og hertil få vist en liste med navne på alle operatørerne, så det var nemmere for brugeren at udvælge den rigtige bruger. Men hertil skal det siges at vores mockobject database ikke indeholder mere end en håndfuld brugere, så det vil i vores program ikke være nemmere for brugeren.

Angående sikkerhed i vores program, så en operatør ikke kan udføre administratoropgaver, så ligger dette delvist i GUI'en. Igennem passwordvalideringen, som valideres i vores mockobject database, så opsættes GUI'en efter hvilken rolle brugeren har. Så administratormenuen vil aldrig være til stede i en operatørs programvindue af GUI'en. Klassen MainView er menuklassen og MenuView er viewklassen i deres sammenhæng. Igennem loginklasserne valideres passwordet vha. integer clearanceLvl fra operatørDTO'en, som henter information på den enkelte bruger fra vores mockobject database, og integeren bruges i klassen MenuView.

Ud fra clearance level 1 - 4 opsættes GUI'en for hver enkelt rolle i MenuView klassen, som opsætter en button, som åbner en ny menu når der klikkes. Der er altså 4 buttons i GUI'en hvis en administrator logger ind, og en enkelt button i GUI'en hvis en operatør logger ind, alt sammen indstansieret igennem klasserne MenuView og MainView. Dette er smart idet vi kun bruger de samme to klasser uanset hvem der logger ind til opsætning af GUI'en, og da en operatør aldrig har menuen til fx at udføre administratoropgaver, er det ikke nødvendigt at have sikkerhed og clearance andre steder i programmet såsom databasen. Der er dermed et sikkerhedstjek i login, og efterfølgende er programmet ens for alle, men der indstansieres forskellige menuer i

GUI'en an på brugerens rolle, som giver adgang til forskellige dele af programmet. Dette vil dog ikke sige at programmet ikke kan hackes, for når der kun er et enkelt sikkerhedstjek, er programmet ikke særligt sikkert. Løsningen med validering flere steder i harddisken, vil sikkerhedsmæssigt være en overlegen løsning, men vi har valgt denne i stedet, da det implementeringsmæssigt var let overkommeligt, samtidig med at vi har sikkerhed implementeret.

Grundet tidspres har vi gået tilbage til den GWT tutorial vi anvendte til CDIO3, men dette betyder også at programmet kunne gøres mindre vha. Templates. Vi har ligeledes ikke den opdeling af forbindelse, funktionalitet og opsætning af menuer, som vi havde planlagt. Løsningen med et enkelt passwordvalidering passer her godt, da det implementeringsmæssigt var let overkommeligt. Havde vi valgt at have databasen fyldt med validering ville vi have haft meget travlt med at få programmet på plads. Vi har holdt fast i den brugervenlige GUI, og lagt arbejde i denne.

ASE

ASE'en er designet til at skabe kommunikation mellem vægt og database, der ud over checker den også at den information der bliver sendt er rigtigt.

ASE'en starter med at skabe en forbindelse til databasen, derefter vælger man om man skal bruge en normal vægt eller simulator.

Når man har valgt hvad man vil forbinde med skaber ASE'en en forbindelse.

ASE'en bruges til at fortolke den information der bliver sendt fra vægten, så det kan sammenlignes med databasen information. fx hvis vi sender en "S\r\n" til vægt vil den returnere "S S vægten i decimal \r\n" siden vægten er i en String form kan vi ikke sammenligne den med den double vi for tilbage fra databasen, så det vi gør er at fjerne "S S " og "\r\n" og så parser vi resten af String til en double.

Nogle gange har vægten sendt en fejlbesked som vi ikke har kunne finde ud af hvorfor, for at komme uden om dette problem har vi lavet et rekursive kald som bliver brugt hvis der bliver sendt en fejlbesked.

Det rekursive kald er ikke en særlig god løsning da den endelig ikke håndterer den fejl der bliver sendt, men bare brute force sig igennem programmet.

Den samme løsning er blevet brugt i TaraWeight metoden af samme oversag. En ting vi har lagt mærke til angående TaraWeight er at vægten er så følsom at tryk på bordet kan få vægten til at svinge, hvilket er en af årsagerne til at Tara af vægt fejler, da programmet ikke kan finde ud af den svingende vægt.

Når ASE'en har forbindelse til vægten vil den fungerer som en guide til hvad operatøren skal gøre, dette fungerer som et double check af recept da operatøren står med en printet udgave af recepten.

Hvis recepterne ikke stemmer over ens er der en fejl i systemet.

Hvis et svar bliver sendt fra vægten som ikke er acceptable, i forhold til den information der er i databasen, vil ASE'en skrive en besked til vægten og bede operatøren om at gentage fx afvejning af produkt x.

ASE klassen er vores hoved klassen som kalder til de to andre klasser, henholdsvis DBController og WeightController, dette gøres ved brug af interfaces.

Både DBControlleren og WeightControlleren er designet til at fungere sammen med vores nuværende database og Mettler vægten, hvis en af disse to ønskes udskiftet, fx til en anden vægt, skal man også skrive en ny

WeightController. På grund af opbygningen skal man dog ikke ændre hverken DBController eller ASE. Man skal dog huske at ændre IP'en inden i ASE klassen når man forbinder til en ny vægt.

Det er desværre ikke lykkedes os at implementere alt til ASE'en

Mangler:

Vores vægt sender en RM20 besked og beder brugeren om at afveje og så derefter trykke ok.

RM20 beskeden blokerer desværre for at vægten kan vise hvor en given genstand vejer. Dette blev realiseret for sendt til at vi kunne nå at ændre det.

Vi kunne ikke få "D text \r\n" eller "P111 text \r\n" til at virke og lave derfor nogle ekstra RM20 kald som overtog funktion, som fx at skrive navnet på recepten til operatøren.

numberOfIngred metoden som skulle give os antallet af loops til vores for løkke for hver recept bruger desværre også et forkert kald, og er derfor blevet ukommenteret.

I stedet har vi sat variablen til tre da ingen af vores receptor har flere råvare, dette er gjort udelukket for at kunne arbejde videre og burde ikke være en del af den endelige kode.

Programmet stopper desværre når du når til afvejning af råvarer nummer to. Vi har ved hjælp af debug fundet frem til at RM20 beskeden bliver sendt, den bliver dog ikke vist på vægten display, vægten sender et svar tilbage, men svar nummer to som burde komme på en RM20 besked kommer aldrig. Det er desværre ikke lykkedes os at finde ud af hvorfor dette sker.

Dataen der bliver brugt i løbet af proceduren mangler at bliver logget ordentligt, dette nåede vi aldrig til at vi sad fast i at prøve på at få vægten til at afveje råvare nummer to.

Vejledning til brug af ASE

Det første man gør er at forbinde vægten til computeren, derefter starter man klassen Run som så beder dig om at vælge hvilken vægt du vil bruge, når du har valgt det følger man de beskeder der bliver skrevet på vægten.

Alt efter hvilken fysisk vægt man bruge kan det være man skal ændre IP'en inden i ASE klassen.

Hvis du forbinder til vægt simulatoren skal du først starte simulatoren ved at åbne for cmd og derefter skrive java-jar CDIO2.jar. Du kan nu starte ASE og vælge Vægt simulator, derefter vil det foregå på samme måde som med den fysiske vægt.

Vægtsimulator

ClientController/CC:

Dette er klassen der styrer vægtens menu. Klassen kommunikerer med tui og giver de forskellige værdier som bliver skrevet ind af brugeren. Klassen kommunikerer også med WeightControlleren så CC kan skrive til putty brugeren.

IOController/IOC:

Denne klasse står for kontrol af fjernstyring. Klassen kommunikerer med WeightData og WeightController.

WeightController/WC:

Denne klasse er den færdig skrevet skelet som vi fik udleveret. Klassen venter på et instream fra putty klienten og skriver derefter tilbage.

Derudover har vi en TUI klassen som står for at skrive TUIen og så er der en weightData klassen som holder styr på vægten og andet information i programmet.

Konklusion

ASE kan forbinde til både vægt og database, men har nogle problemer med at sende nogle specifikke kommandoer til vægten. Derudover laver vi nogle semi optimale løsninger for at omgå nogle problemer med svar fra vægten. Generelt fungerer det essentielle, men det stopper med at virke efter at programmet har kørt lidt tid.

På baggrund af vores analyse og krav til opgaven, samt det udleverede kode har vi formået at implementere de nødvendige interfaces. Vi har fået en stabil forbindelse med vores database, hvor vi har sammensmeltet SQL sætninger med Java kode til alle de eksisterende tabeller, vi har i vores database. Vi har brugt localhost til at connecte vores Java kode i eclipse med databasen. Vi har sørget for at håndtere de forskellige exceptions, så fejlen kan spores og findes korrekt. Vi har konstrueret JUnit test på de

vigtigste områder, som viser at vores Java kode og SQL forespørgsler virker efter hensigten, og giver de ønskede resultater.

GWT'en mangler sammenkobling mellem controller og database. GUI'en er sat op, men den mangler delmenuer. Vi har holdt fast i vores brugervenlige design, som kan ses i administratormenuen, som er delvist opsat med funktionslaget, såsom i ret bruger hvor alt er implementeret, lige med undtagelse af at kaldet kun går fra viewet til mainview og tilbage igen, dvs. Uden information. Noget er altså sat op uden kun visuelt, og andet er ikke sat op mens dele af GUI'en har forbindelse til mackobject database, og controlleren og virker korrekt

Bilag 1

Dato	Deltager	Analyse	Design	Impl.	Andet	I alt	Bemærkninger
03.06.2016	Abdishukri	3	3			6	Fordelling
03.06.2016	Amer	2			4	6	Fordelling
03.06.2016	Aslanbek			3	4	7	Fordelling
03.06.2016	Dannie				4	4	Fordelling
03.06.2016	Haydar	2			4	6	Fordelling
03.06.2016	Jonas				4	4	Fordelling
03.06.2016	Mads				4	4	Fordelling
04.06.2016	Abdishukri	2	4			6	
04.06.2016	Amer	4				4	
04.06.2016	Aslanbek	4				4	
04.06.2016	Dannie	8				8	
04.06.2016	Haydar	4				4	
04.06.2016	Jonas						
04.06.2016	Mads						
05.06.2016	Abdishukri						
05.06.2016	Amer				3	3	
05.06.2016	Aslanbek						
05.06.2016	Dannie						
05.06.2016	Haydar						
05.06.2016	Jonas						
05.06.2016	Mads						
06.06.2016	Abdishukri	4			2	6	
06.06.2016	Amer	2			4	6	
06.06.2016	Aslanbek	2			2	4	

06.06.2016	Dannie	2			4	6	
06.06.2016	Haydar	2			4	6	
06.06.2016	Jonas	2			2	4	
06.06.2016	Mads	2			2	4	
07.06.2016	Abdishukri			5		5	
07.06.2016	Amer	1			5	6	
07.06.2016	Aslanbek	7				7	
07.06.2016	Dannie	3			3	6	
07.06.2016	Haydar	1			5	6	
07.06.2016	Jonas	7				7	
07.06.2016	Mads	7				7	
08.06.2016	Abdishukri						
08.06.2016	Amer	3		1		4	
08.06.2016	Aslanbek		4			4	
08.06.2016	Dannie	1		1		2	
08.06.2016	Haydar	3		1		4	
08.06.2016	Jonas		1			1	
08.06.2016	Mads		4			4	
09.06.2016	Abdishukri						
09.06.2016	Amer			4		4	
09.06.2016	Aslanbek		6			6	
09.06.2016	Dannie		2	11		13	
09.06.2016	Haydar			4		4	
09.06.2016	Jonas		5	5		10	
09.06.2016	Mads		6			6	
10.06.2016	Abdishukri			8		8	

10.06.2016	Amer			6	2	8	
10.06.2016	Aslanbek		2		6	8	
10.06.2016	Dannie			2		2	
10.06.2016	Haydar			4	4	8	
10.06.2016	Jonas			2	6	8	
10.06.2016	Mads			4	4	8	
11.06.2016	Abdishukri			4		4	
11.06.2016	Amer			4		4	
11.06.2016	Aslanbek			4		4	
11.06.2016	Dannie			11		11	
11.06.2016	Haydar			4		4	
11.06.2016	Jonas						
11.06.2016	Mads						
12.06.2016	Abdishukri			6		6	
12.06.2016	Amer	2			4		
12.06.2016	Aslanbek						
12.06.2016	Dannie			12		12	
12.06.2016	Haydar		4	2		6	
12.06.2016	Jonas						
12.06.2016	Mads						
13.06.2016	Abdishukri		1	4		5	
13.06.2016	Amer		4			4	
13.06.2016	Aslanbek						
13.06.2016	Dannie			10		10	
13.06.2016	Haydar		4			4	
13.06.2016	Jonas			5		5	
13.06.2016	Mads			5		5	

14.06.2016	Abdishukri						
14.06.2016	Amer		3		3	6	
14.06.2016	Aslanbek		4			4	
14.06.2016	Dannie			9	3	12	
14.06.2016	Haydar		3		3	6	
14.06.2016	Jonas			4	4	8	
14.06.2016	Mads			8		8	
15.06.2016	Abdishukri		2	5		7	
15.06.2016	Amer		2	3	1	6	
15.06.2016	Aslanbek						
15.06.2016	Dannie			7,5		7,5	
15.06.2016	Haydar		3	3		6	
15.06.2016	Jonas		2	7		9	
15.06.2016	Mads			9		9	
16.06.2016	Abdishukri		8			8	
16.06.2016	Amer		8			8	
16.06.2016	Aslanbek		8			8	
16.06.2016	Dannie			11,5		11,5	
16.06.2016	Haydar		8			8	
16.06.2016	Jonas			8		8	
16.06.2016	Mads					8	
17.06.2016	Abdishukri		3			3	
17.06.2016	Amer		2			2	
17.06.2016	Aslanbek		3			3	
17.06.2016	Dannie		3			3	
17.06.2016	Haydar		3			3	
17.06.2016	Jonas		3			3	

17.06.2016	Mads		4		4	
------------	------	--	---	--	---	--

	Abdi	Amer	Aslanbek	Dannie	Haydar	Jonas	Mads
Timer i alt	71	72	65	116	72	67	71
Procent	13.32	13.51	12.20	21.76	13.51	12.57	13.32