Projektopgave forår 2015 – CDIO final
02324 - Videregående Programmering

Projektnavn: **19_CDIOFinal**

Gruppe nr: **19**

Afleveringsfrist: **19-06-2015 kl. 12:00**

Den rapport er afleveret via Campusnet (der skrives ikke under).

Denne rapport indeholder ***59*** sider inklusiv denne side (svarende til ca. 36 normalsider eksklusiv bilag).

S144847, Nielsen, Jon Tvermose

Kontaktperson(Projektleder)

S125015, Berthold, Ebbe Bjerregaard

S144855, Olsen, Camilla Braae

S144875, Jepsen, Jacob Worck

S144843, Mikkelsen, Christian Flygare Carlend

Timeregnskab

CDIO 15_final							
Time-regnskab							
Dato	**Deltager**	**Design**	**Impl.**	**Test**	**Dok.**	**Andet**	**Ialt**
	Jon	10,25	55,75	11	19,5	14,75	111,25
	Ebbe	5,5	47	11	15	4	82,5
	Christian	7,5	71,5	13,5	8,5	2	103
	Camilla	7	49,5	1	24,5	2	84
	Jacob	6	47,5	11	14	4	82,5
	Sum	**36,25**	**271,25**	**47,5**	**81,5**	**26,75**	**463,25**
Design	Analyse + design diagrammer						
Impl.	Kode systemet						
Test	Kode tests (junit m.m.) + udførelse af brugertest						
Dok.	Rapportskrivning						
Andet	Koordinering af arbejdsopgaver, projektledelse						

Figur 1: Timeregnskab

Projektlederens kommentar til timeregnskab

Design fasen (analysen) af projektet kunne overstås forholdsvis hurtigt idet alle gruppemedlemmer på forhånd havde gennemlæst opgaven inden projektstart. Herudover kunne store dele af kravs-analyserne delvist genbruges fra tidligere løste opgaver. Design fasen bestod derfor primært af at analysere huller i forhold til tidligere løste opgaver, samt hvordan de forskellige delopgaver skulle samles til en helhed.

Implementeringsfasen har taget langt størstedelen af projekttiden (over 50%). Specielt ASE har taget længere tid at tilrette en forventet og planlagt. Dette skyldes sandsynligvis at der er valgt en meget kodeteknisk løsning vha. Enums, samtidig med at ASE i en stor del af tiden har været en integreret del af GWT-projektet, hvilket har gjort det næsten umuligt at debugge. Dette blev dog ændret i den sidste uge af projektfasen, hvorefter udviklingen af ASE tog betydelig fart.

For GWT Web GUIen er der lagt mange timer i at gøre brugen heraf så intuitiv og brugervenlig som mulig. Herudover er der lagt fokus på at udvikle et lækkert design.

I sidste uge af projektet har den primære fokus været på test via JUnit og brugertest, samt dokumentation af systemet via udarbejdelse af denne rapport. ASE er blevet underlagt grundig test for eventuelle fejlsituationer ved at benytte den fysiske vægt.

Konfiguration

Afsnittet beskriver, hvad afleveringen indeholder, samt hvordan programmet startes og bruges. Det forudsættes, at der benyttes Eclipse Luna, og at der er installeret MySQL (5.5.42) samt GWT 2.6.1 på computeren. Herudover skal Eclipse være sat op til at benytte tegnsæt UTF-8. Projektet benytter Java 1.7, men TokenHandler.java der er stillet til rådighed af Ronnie Dalsgaard er først inkluderet i Java 1.8.

Afleveringen består af filen 19_CDIOFinal.zip. Filen indeholder et projekt, som kan importeres og køres i Eclipse. Herudover indeholder filen følgende:

- **19_rapport.pdf** : Denne rapport, som dokumenterer systemet
- **19_db.sql** : En række SQL statements, der opretter de nødvendige tabeller og testdata for at afvikle programmet

Opsætning af databasen

I klassen cdio.server.DAL.Connector.java bestemmes hvilken database, der skal benyttes. Der kan benyttes den centrale databaseserver eller en lokal beliggende database på computeren. Som udgangspunkt benyttes en central database (server ip 62.79.16.16).

For at benytte en lokal database skal man oprette en MySQL-database ved navn 19_db. Kør scriptet 19_db.sql på databasen vha. kommandoen ”source [filepath].19_db.sql”. Databasen bør nu være oprettet og indlæst med testdata. Udkommentér den relevante kode i cdio.server.DAL.Connector.java, så den lokale database benyttes.

Login Web GUI

Der kan logges på Web GUI med et bruger id og password fra databasen. Herunder er givet en oversigt over bruger id, password, samt hvilken rolle personen har i systemet. Bemærk – hver rolle har adgang til forskellige ting i applikationen, og en operatør har ikke adgang til at logge på Web GUI.

Bruger id	Password	Rolle
1	02324it!	Admin
2	02324it!	Farmaceut
3	02324it!	Værkfører
4	02324it!	Operatør

ASE

For at teste Afvejnings Styrings Enhed (ASE) kræves en vægt. Projektets ASE er specifikt designet til den fysiske vægt, hvorfor det kun kan lade sig gøre at benytte en vægtsimulator, hvis den har en fuldstændig identisk kommunikationsprotokol, og samme funktioner som den fysiske vægt.

ASE startes ved at køre klassen ASE.Main.java. Når programmet startes vil det selv afsøge en række IP-adresser og forsøge at oprette en forbindelse til disse, indtil programmet manuelt termineres. Såfremt det lykkes at oprette en forbindelse til vægten, vil displayskærm skifte til ”Tast bruger ID:”, og ASE vil skrive i Eclipse-konsollen at en forbindelse er oprettet. Herefter er ASE klar til at starte afvejningsproceduren. Bemærk - kun en bruger med rollen operatør kan benytte vægten. Opsætningen af de IP-adresser (samt port) der skal forsøges at oprette forbindelse til kan ske i *ASE.ServerASE.java*.

Indholdsfortegnelse

Indledning

Denne rapport er udarbejdet af projektgruppe 19 og indgår i 3-ugers eksamen til kursus 02324 – Videregående Programmering på Danmarks Tekniske Universitet, forårssemester 2015.

Rapporten dokumenterer et distribueret afvejningssystem, der består af et webinterface, en MySQL-database samt en vægt. Webinterfacet har back-end til bruger- og produktadministration. Datalaget er implementeret som en database i MySQL. Vægten styres af en afvejningsprocedure (ASE).

Lagt i rapporten er et timeregnskab, der viser, hvor meget tid projektgruppen har brugt på de forskellige dele af projektet, herunder design, implementering, test og dokumentation.

Projektplan

Tidligt i forløbet er der blevet udarbejdet en projektplan med udgangspunkt i tre af de fire faser fra CDIO (Concieve, Design, Implement, Operate).

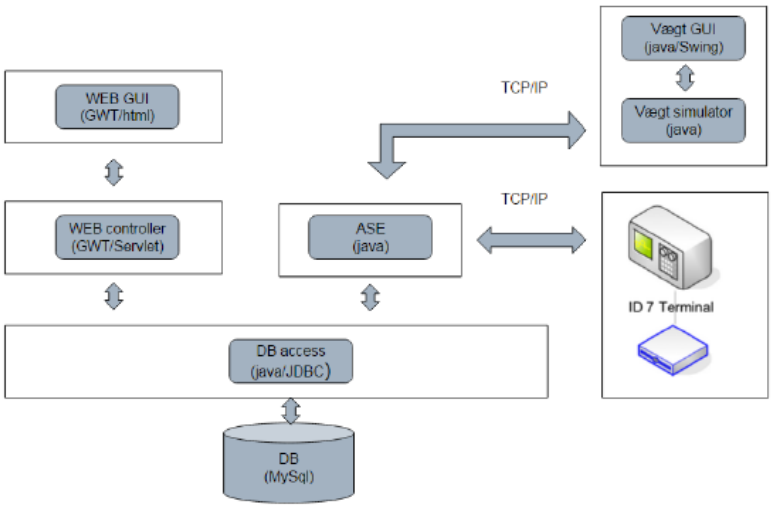
		Torsdag	Fredag	Lørdag	Søndag	Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Mandag	Tirsdag	Onsdag	Torsdag	Fredag
		04-06-15	05-06-15	06-06-15	07-06-15	08-06-15	09-06-15	10-06-15	11-06-15	12-06-15	13-06-15	14-06-15	15-06-15	16-06-15	17-06-15	18-06-15	19-06-15
Milestone 1							X										
Milestone 2														X			
Aflevering																	X
Delopgaver:	Ansvarlig																
Web GUI Design	Jon																
Database Design	Jacob																
ASE Design	Christian																
Samle delprojekter	Jon																
WEB GUI Implementering	Jon																
Database Implementering	Jacob																
ASE Implementering	Christian																
Test																	
Dokumentation																	

Figur 2: Projektplan

Der startes med en designfase (markeret med grøn), hvor opgaveoplægget analyseres og sammenholdes med tidligere afleverede opgaver. Der identificeres mangler og en plan samt indledende arbejdsfordeling fastlægges. Herudover gennemgår gruppen en prioriteringsøvelse mht. de identificerede opgaver.

Implementeringsfasen (markeret med gul + lys orange) består af at samle tidligere projekter samt tilrette og udbygge disse så de matcher kravene fra designfasen.

Slutteligt underkastes systemet en grundig test (mørk orange), og dokumentationen (blå) udarbejdes i form af denne rapport.



Figur 3: Systemets arkitektur

Ovenstående figur viser systemets arkitektur fra opgaveoplægget. Tidligt i forløbet har projektgruppen fastlagt sig på at der ønskes at udvikle en robust og brugervenlig prototype, der er optimeret mht. den fysiske vægt. Ud fra dette er der blevet udarbejdet en liste som indeholder hvilke elementer af opgaveoplægget der løses af projektgruppen (in scope), samt hvilke der aktivt fravælges (out of scope).

In scope

- Brugervenligt og intuitivt design – gælder for alle elementer
- Fungerende prototype indeholdende:
    - **GWT Web GUI**
        - Inputvalidering
        - Tokenbaseret login
        - Admin-funktioner
        - Farmaceut-funktioner
        - Værkfører-funktioner
    - **Database (MySQL)**
        - 3NF
        - Data Access Layer + Data Transfer Objects
    - **ASE (Afvejnings Styrings Enhed)**
        - Robust fejlhåndtering
        - Komplet afvejningsprocedure
- Test og dokumentation af system
    - JUnit test
    - Brugertest

Out of scope

- Vægt simulator + Vægt GUI
- Web GUI: Operatør-funktioner

Ud fra ovenstående opdelte projektgruppen hurtigt arbejdsopgaverne i tre hovedopgaver, bestående af **GWT Web GUI**, **Database** og **ASE**. De vil blive behandlet ligeså i denne rapport.

Analyse

Dette afsnit beskriver dele af analysefasen, herunder hvilke krav der er blevet identificeret til projektet. Afsnittets formål er at komme nærmere, hvilke elementer systemet skal indeholde, samt hvad disse elementer skal indeholde.

Kravspecifikation

Kravsspecifikationen er udarbejdet ud fra opgaveoplægget samt via samtaler med kunden. Kravsspecifikationen er dermed afgørende forskellig fra opgaveoplægget på diverse punkter, hvor oplægget har været uklart formuleret. F.eks. har en bruger med rollen admin ikke samme adgangsrettigheder som en bruger med rollen farmaceut. Der er dermed ikke tale om et hierarki af brugerne baseret på deres roller. Kravsspecifikationen er opdelt i tre kravsspecifikationer, en for hver delelement identificeret og beskrevet i foregående afsnit, Indledning.

GWT Web GUI

Funktionelle krav

- 4 roller i systemet (Admin, Farmaceut, Værkfører, Operatør)
- Forskellige funktioner mulige for hver rolle
- Administrator har følgende funktioner for rollerne Operatør, Farmaceut eller Værkfører:
    - Oprette
    - Rette
    - Slette (deaktiver bruger)
    - Vise
- En bruger må aldrig slettes
- **Farmaceut** skal kunne udføre følgende handlinger for **råvarer**:
    - Opret
    - Rette
    - Vise
    - Slet (såfremt råvaren ikke har været brugt)
- En råvare defineres ved et råvarenummer (brugervalgt og entydigt), navn samt leverandør
- **Farmaceut** skal kunne udføre følgende handlinger for **recepter**:
    - Opret
    - Vise
    - Slet (såfremt recepten ikke har været brugt)
- En recept defineres ved et receptnummer (brugervalgt og entydigt), navn samt sekvens af receptkomponenter
- En receptkomponent består af en råvare (type), mængde samt tolerance
- **Værkfører** skal kunne udføre følgende handlinger for **råvarebatches**:
    - Opret
    - Vise
- En råvarebatch defineres ved et råvarebatchnummer (brugervalgt og entydigt) samt mængde
- **Værkfører** skal kunne udføre følgende handlinger for **produktbatches**:
    - Opret
    - Vise
    - Slet (såfremt produktbatch ikke er påbegyndt)

- En produktbatch defineres ved et produktbatchnummer (entydigt), nummeret på den recept produktbatchen produceres ud fra, dato for oprettelse, status for batchen (oprettet, under produktion, afsluttet)
- Når værkfører har oprettet en ny produktbatch printes denne og tildeles en udvalgt operatør (der skal vises en side som matcher den i bilaget)
- Der skal være input validering på alle felter
- Det skal være brugervenligt
- Brugere skal autentificeres ved login. Baseret på brugers rolle tilpasses indhold af Web GUI
- Der skal vises passende fejlmeddelelser ved fejl

Nonfunktionelle krav:

- Web GUI skal implementeres vha. GWT

ASE

Funktionelle krav:

- Styre afvejningsproceduren
- Tage imod operatør ID og identificere operatør
- Svare med operatørens navn
- Tage imod produktbatchnummer og identificere produktbatch og recept
- Registrere og gemme vægt af beholder i produktbatchkomponenten
- Tage imod hvilken råvarebatch en råvare stammer fra
- Acceptere afvejning når denne er inden for receptens tolerance
- Afvejningen gemmes i produktbatchkomponenten
- Status ændres fra ”oprettet” til ”under produktion”
- Gentagelse for alle råvare i recepten
- Status ændres slutteligt til ”afsluttet”
- Ingen kontakt mellem ASE og GWT

Leverandørens afvejningsprocedure

Leverandøren har specificeret en afvejningsprocedure i opgaveoplægget.

1. Operatøren har modtaget en produktionsforskrift på papir fra værkføreren.
2. Operatøren vælger en afvejningsterminal.
3. Operatøren indtaster operatør nr.
4. Vægten svarer tilbage med operatørnavn som så godkendes.
5. Operatøren indtaster produktbatch nummer.
6. Vægten svarer tilbage med navn på recept der skal produceres (eks: saltvand med citron)
7. Operatøren kontrollerer at vægten er ubelastet og trykker ’ok’
8. Systemet sætter produktbatch nummerets status til ”Under produktion”.
9. Vægten tareres.
10. Vægten beder om første tara beholder.
11. Operatør placerer første tarabeholder og trykker ’ok’
12. Vægten af tarabeholder registreres
13. Vægten tareres.
14. Vægten beder om råvarebatch nummer på første råvare.
15. Operatøren afvejer op til den ønskede mængde og trykker ’ok’

16. Pkt. 7 – 14 gentages indtil alle råvarer er afvejet.
17. Systemet sætter produktbatch nummerets status til ”Afsluttet”.
18. Det kan herefter genoptages af en ny operatør.

Database

Funktionelle krav

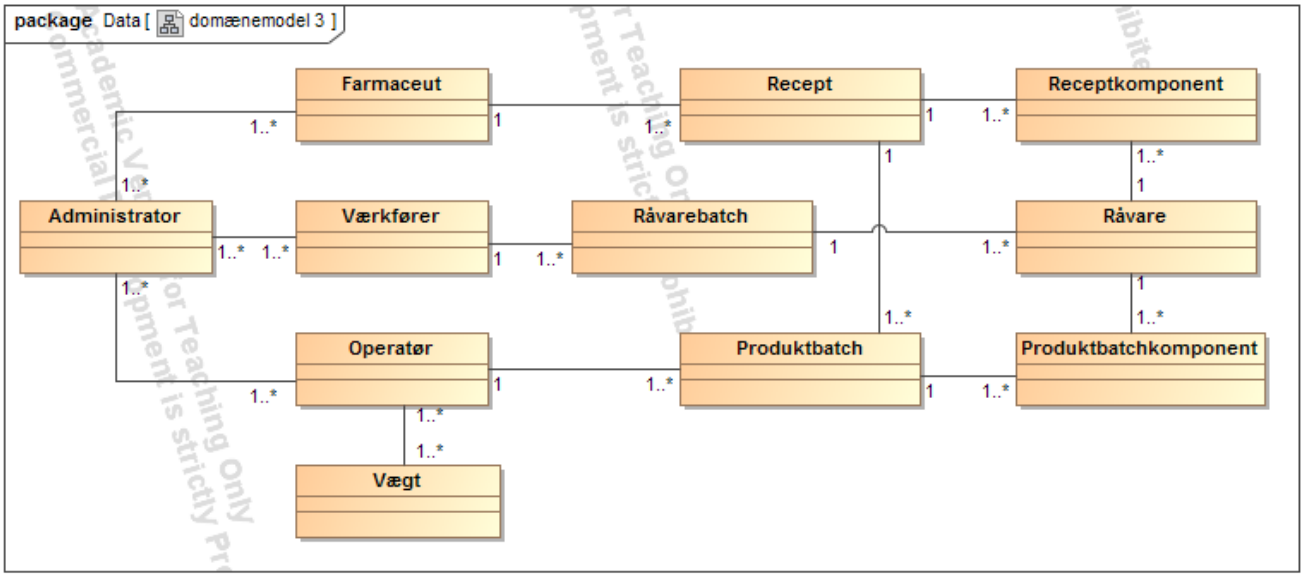
- I systemet skal det være muligt at definere råvarer, som hjemtages I råvarebatches.
- Det skal være muligt at oprette recepter.
- Systemet skal holde styr på den aktuelle mængde i alle råvarebatches.
- Systemet skal indeholde følgende aktører:
    - Administrator
    - Farmaceut
    - Værkfører
    - Operatør
- Systemet skal kunne håndtere følgende:
    - Oprettelse af aktører
    - Rettelse af aktører
    - Sletning af aktører
    - Visning af aktører
- En bruger med rollen operatør kan ikke slettes fra systemet.
- En råvare defineres ved følgende:
    - Et råvarenummer som er brugervalgt og entydigt
    - Navn
    - Leverandør
- En recept defineres ved følgende:
    - Et receptnummer (brugervalgt og entydigt)
    - Et navn
- En receptkomponent består af:
    - Recept id
    - Råvare id
    - Mængde (netto)
    - Tolerance.
- En råvarebatch defineres ved følgende:
    - Råvarebatch nummer som er brugervalgt og entydigt
    - Råvare id
    - En mængde
- En produktbatch defineres ved:
    - Produktbatchnummer som er entydigt
    - Receptnummer fra hvor produktbatchen er produceret
    - Dato for oprettelse
    - Oplysning for status: 0 ikke påbegyndt, 1 for under produktion, 2 for afsluttet
- En produktbatchkomponent defineres ved:
    - Produktbatch id
    - Råvarebatch id
    - Operatør id
    - Tara

  - Netto
- Der skal være exception-håndtering

Nonfunktionelle krav:

- Databasen skal implementeres som en MySQL database
- Databasen skal minimum være på tredje normalform.
- Der skal laves et Data Access Lag som adskiller databasen fra de andre komponenter

Domæne model



Figur 4: Domæne model

Ovenstående diagram viser domænemodellen for systemet. Domænemodellen illustrerer de fysiske elementer som indgår i problemdomænet, og deres indbyrdes forhold.

Hvis modellen betragtes fra venstre mod højre, ses det altså at administrator har mange-til-mange forhold til farmaceut, værkfører og operatør. Det er illustreret på denne måde, idet administratorens opgave er at udføre brugeradministration på de tre typer af brugere i systemet.

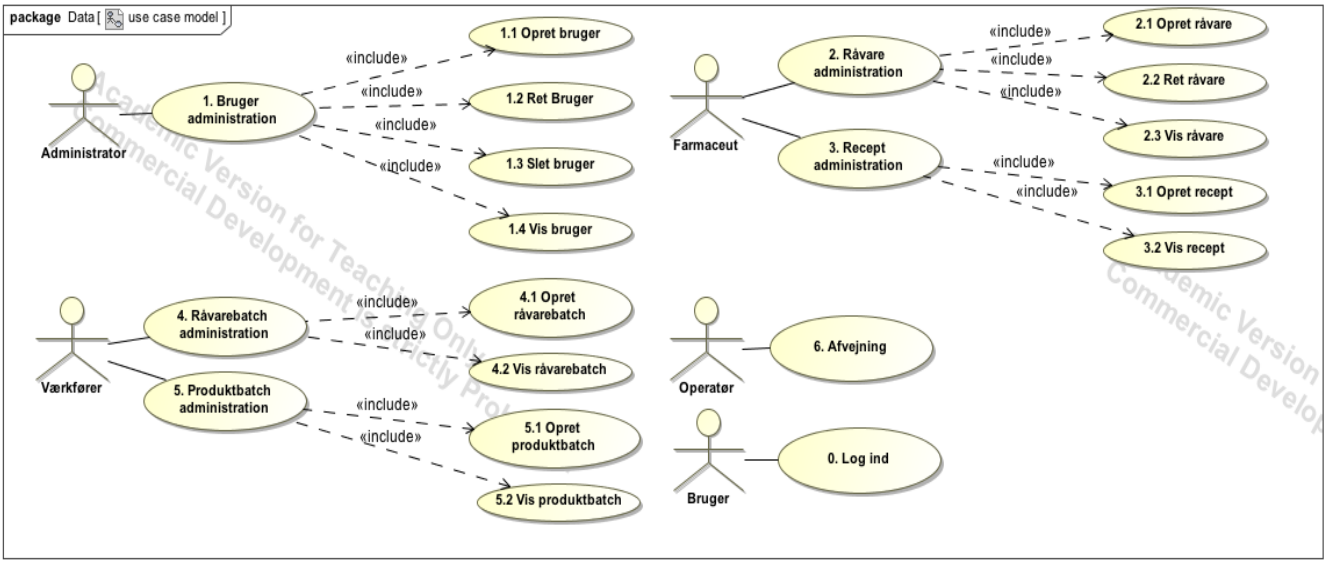
Længere henne ses det at farmaceuten har et en-til-mange forhold til recept. Dette skyldes at en farmaceut kan oprette flere recepter, men hver recept er oprettet af én farmaceut. Den samme type forhold gør sig gældende mellem værkfører og råvarebatch samt operatør og produktbatch. Forholdet er lidt anderledes mellem operatør og vægt idet en operatør kan benytte flere vægte, og der gennem tiden kan være flere operatører der benytter samme vægt.

Mellem recept og receptkomponent ses igen et en til mange forhold, som beskrevet ovenfor. Her forstås det på den måde, at en recept indeholder flere receptkomponenter, mens en receptkomponent hører til en bestemt recept. Igen ses samme forhold mellem produktbatch og produktbatchkomponent.

Til gengæld ses et en-til-en forhold mellem råvarebatchkomponent og råvare, da en råvarebatchkomponent indeholder en mængde af en råvare, men altså ikke flere forskellige råvarer.

Slutteligt er der en til mange forhold mellem råvare og receptkomponent, samt råvare og produktbatchkomponent. Dette er igen fordi at en receptkomponent kun indeholder en råvare, mens en råvare godt kan være flere receptkomponenter. Det samme er gældende for produktbatchkomponent.

Use case model



Figur 5: Use case model

Ovenstående diagram viser en oversigt over de overordnede syv use cases (UC 0-6) som projektgruppen fik opgivet i projektoplægget. Projektgruppen har valgt at dele disse use cases op i mindre use cases, med henblik på at illustrere hvad de reelt indebærer. Ud over dette er use case *0. log ind* tilføjet, hvori aktøren er *bruger*. Dette er gjort for at tydeliggøre at *log ind* er en generel use case for alle brugere af Web GUI. Dette medfører at denne use case dækker administrator, farmaceut og værkfører, men ikke operatør.

For hver af de overordnede use cases, er der lavet én use case specifikation, der dækker én af de inkluderede use cases. Ønskes det at læse disse, kan de findes i bilag. Ud over dette er der lavet en tabel, der illustrerer hvilke use cases, der dækker kravene for Web GUI. Der er fokuseret på at sammenligne disse to, idet størstedelen af brugerinteraktionen foregår via GUI, hvorfor det også er her use casene dækker kravene til funktionaliteten. Tabellen kan ligeledes findes i bilag.

Delkonklusion

Der er i analysen blevet udviklet en domæne model, et use case diagram med tilhørende use case beskrivelser, samt en kravsspecifikation. Samlet set giver disse elementer et godt overblik over hvordan systemet skal opføre sig, samt hvad det skal indeholde.

Herudover er der fra projektoplæggets side givet et bud på en systemarkitektur som giver yderligere indsigt i systemets opbygning.

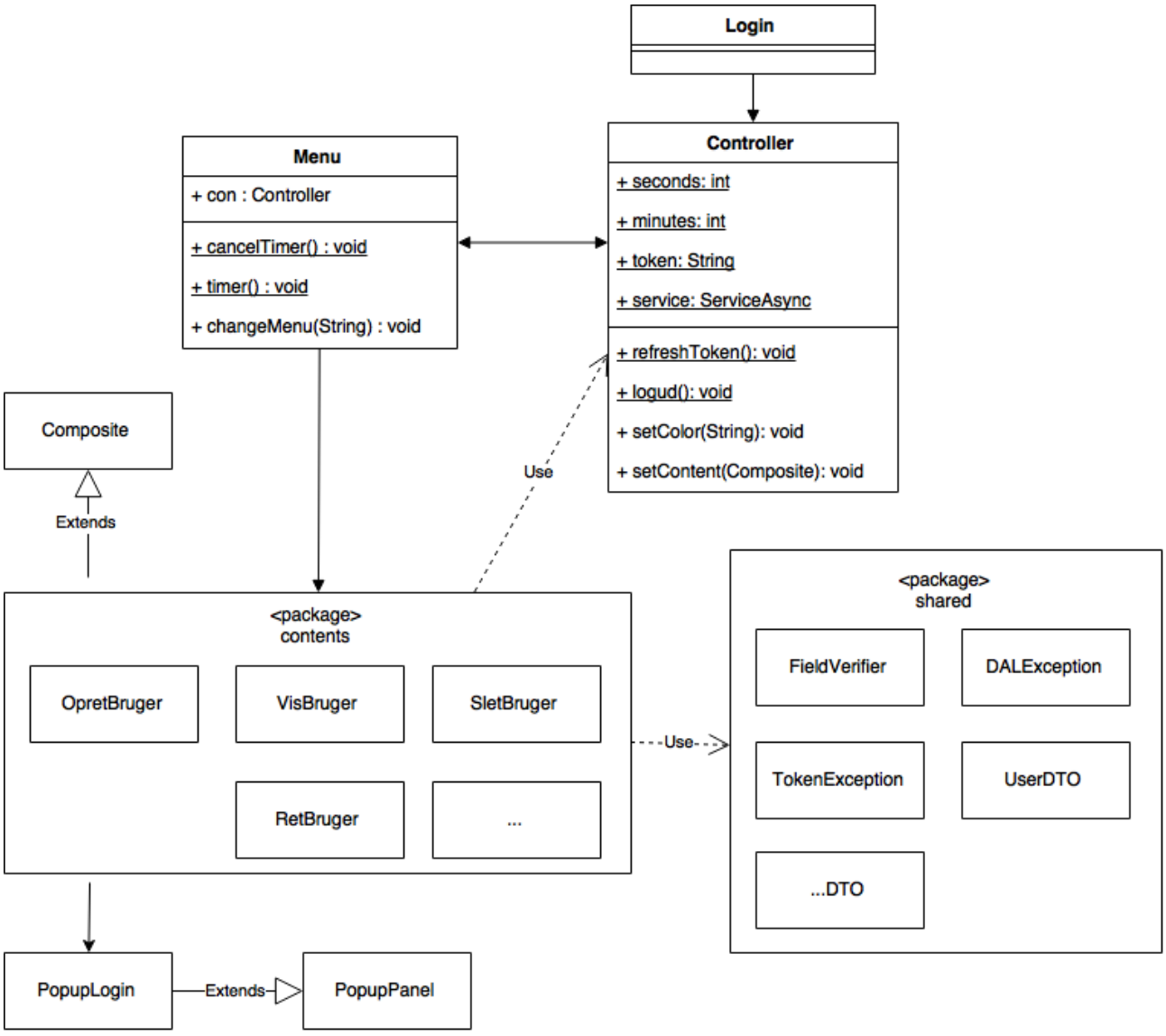
Design

Afsnittet dokumenterer det samlede system. Der er udvalgt en række centrale områder i systemet, der dokumenteres.

Klassediagram

Følgende afsnit vil illustrere en række design klassediagrammer for udvalgte dele af projektet. Klassediagrammerne vil fremstå i forsimplet udgave for overskuelighedens skyld. Formålet med afsnittet er dermed at skabe indsigt i, hvordan dele af programmets arkitektur er opbygget.

Klassediagram: GWT Client



Figur 6: Klassediagram GWT Client

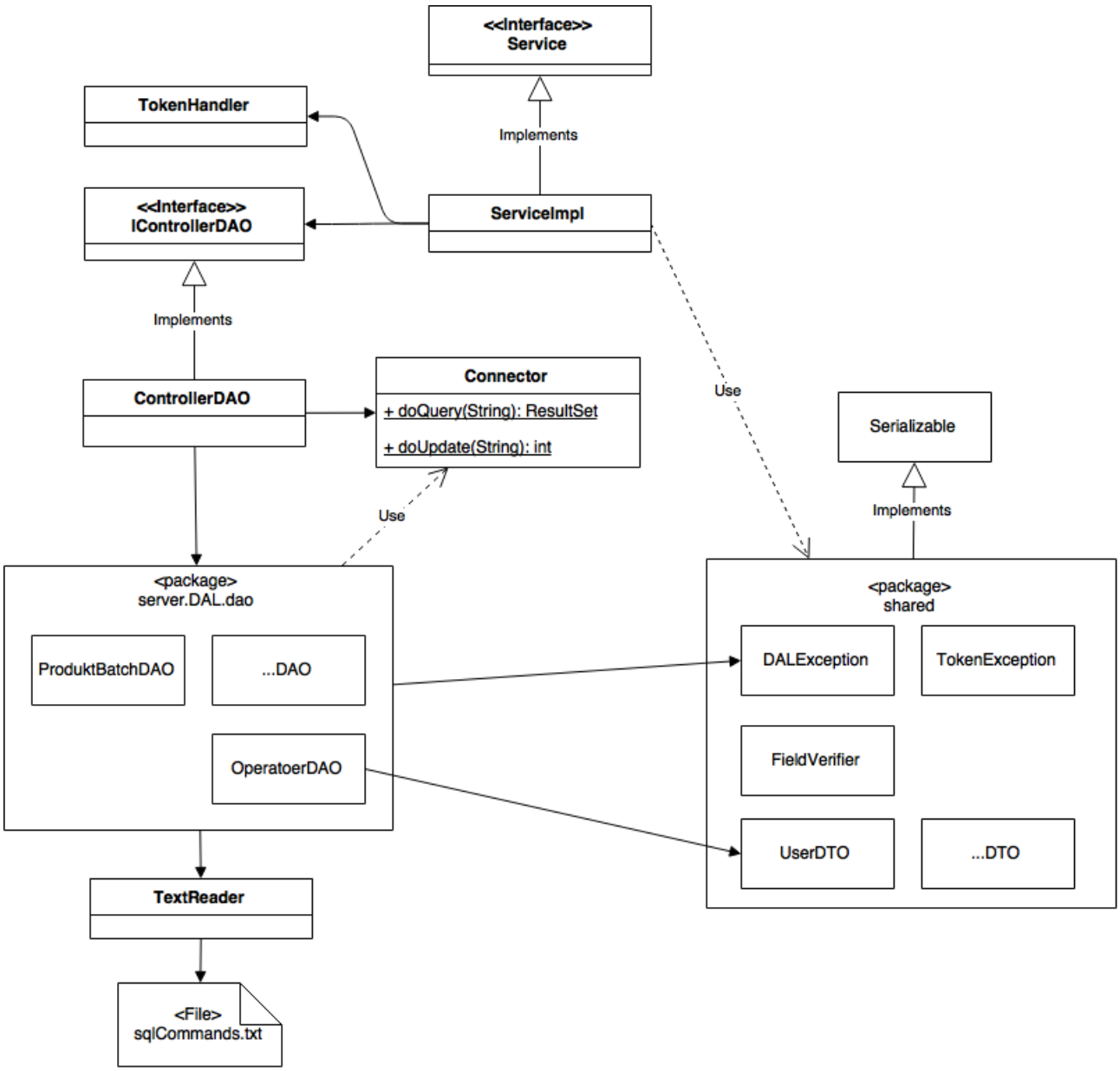
Ovenstående figur illustrerer et delvist forsimplet klassediagram for GWT klientdelen. Det ses, at programmet bl.a. består af en Controller, en Menu samt en række Content-klasser. Menuen har via Controller adgang til at ændre, hvilket Content (indhold), der skal vises.

Controller indeholder en række static variabler heriblandt service, som er forbindelsen til selve serveren, samt token, som er brugerens token, der skal bruges for at verificere en bruger, når serveren kaldes. Herudover har Controlleren de to static metoder refreshToken() samt logud(), som alle content-klasser også har adgang til. refreshToken() benyttes, når brugeren udfører en handling, som kræver, at brugerens token valideres. Metoden vil kontakte serveren med den nuværende token, og forsøge at få den opdateret og dermed forlænge levetiden (en token har en begrænset levetid, i dette projekt sat til 30 minutter).

Det ses, at alle content klasser kan lave et objekt af typen PopupLogin. Denne klasse benyttes, hvis en token er udløbet, og en bruger forsøger at udføre en handling, der kræver validering. Brugeren vil herefter blive præsenteret for en popup-skærm, hvor brugeren skal indtaste sit password igen. Gøres dette kan brugeren fortsætte, hvor han slap uden at miste noget data. Alternativt kan brugeren annullere og vil derefter blive ledt til den generelle login-side (Login.java).

Data sendes mellem klient og server vha. asynkrone kald og overføres som Data Transfer Objects (DTO). For at dette kan lade sig gøre, skal alle DTO implementere interfacet Serializable. Dette gør, at objekterne kan serialiseres som en bit-string eller en simpel string.

Klassediagram: GWT Server



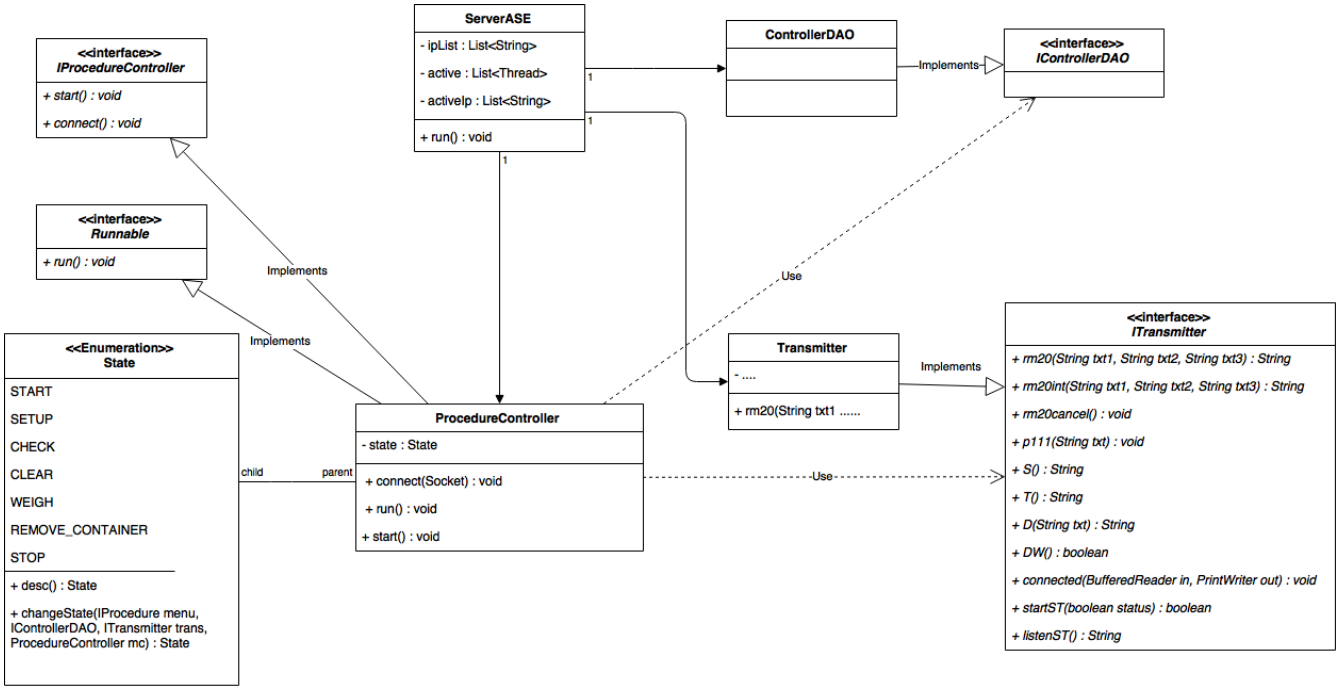
Figur 7: Klassediagram GWT Server

Ovenstående figur illustrerer et forsimplet klassediagram af GWT serverdelen, der er tilsluttet en MySQL database via Connector-klassen. Klassen ServiceImpl implementerer interfacet Service, som er defineret i Client-delen af GWT-projektet. Det implementerer dermed de metoder, som klienten kan benytte sig af, og fungerer som bindeled mellem klient og server.

ServiceImpl har adgang til ControllerDAO, som fungerer som systemets Data Access Object/Layer. Dvs. via denne klasse har ServiceImpl indirekte adgang til at udføre og hente information fra databasen. Herudover har ServiceImpl adgang til de forskellige Data Transfer Objects (DTO), som overføres mellem klient og server. Disse ligger i package shared. Slutteligt har ServiceImpl også adgang til klassen TokenHandler, som benyttes til at oprette og validere tokens.

Det ses, at de forskellige DAO-implementeringer i package server.DAL.dao benytter de klasser, der findes i shared package. Disse DAO-implementeringer oversætter et ResultSet returneret fra databasen (via Connectors statiske metode *doQuery()*) til det korrekte Data Transfer Object. Såfremt en fejl opstår i dette lag, kastes en DALException med en passende besked, som via ServiceImpl kastes videre til klienten og endeligt vises for brugeren.

Klassediagram: ASE



Figur 8: ASE klassediagram

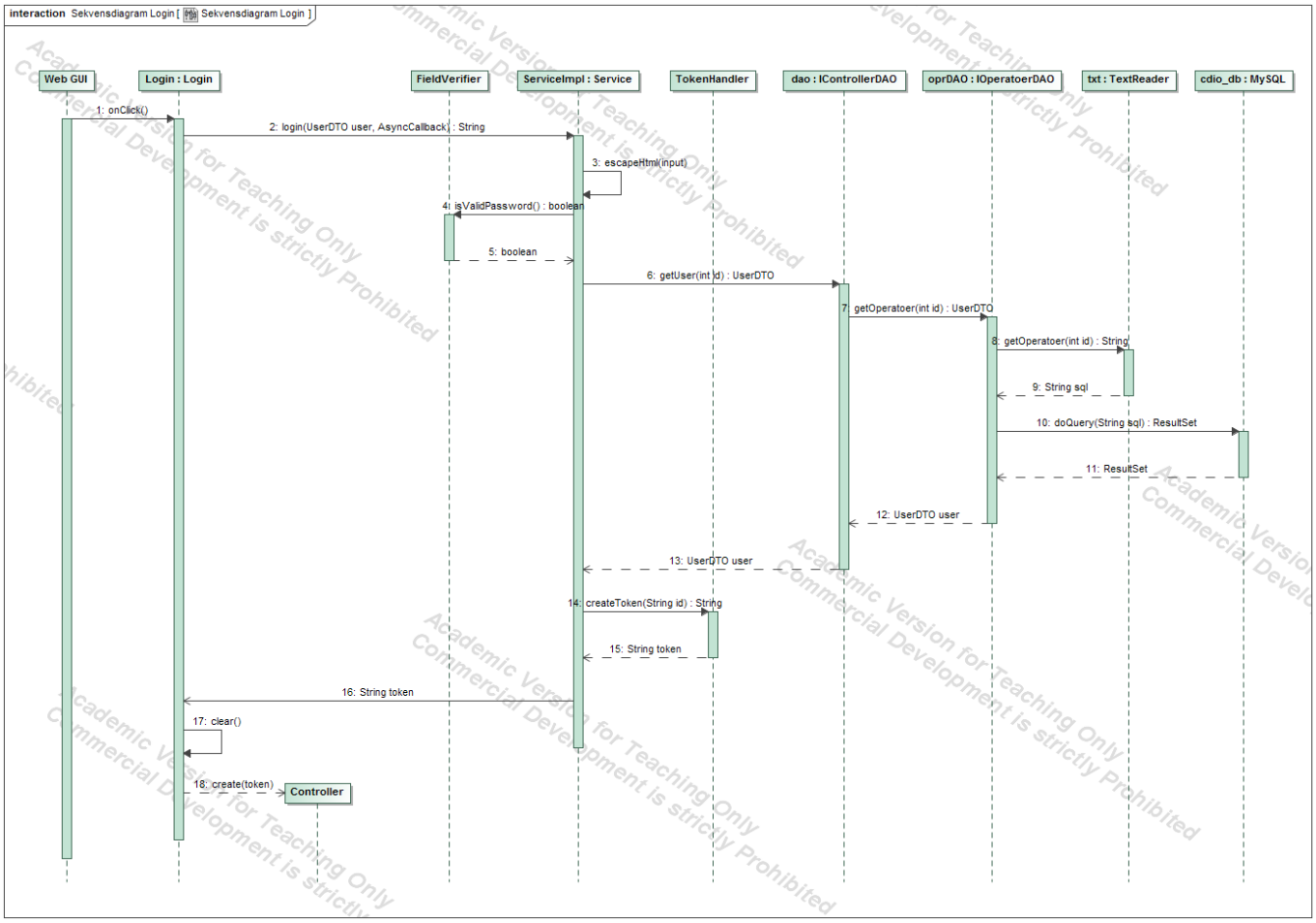
Ovenstående figur illustrerer et delvist klassediagram af ASE. Det er for overskuelighedens skyld valgt at udelade en række attributter og metoder. I figuren ses at ServerASE kan oprette objekter af typerne ProcedureController, Transmitter og ControllerDAO. ProcedureControlleren benytter en reference til IControllerDAO og ITransmitter. Herudover benytter ProcedureController enums kaldet State. Enums gennemgås i senere afsnit i rapporten, men på klassediagrammet ses de forskellige værdier State kan tage, samt hvilke metoder hver enkelt værdi skal implementere (desc + changeState).

Det ses også at ProcedureController implementerer to interfaces, heriblandt Runnable som er nødvendigt for at kunne starte mere end en enkelt afvejningsprocedure af gangen vha. tråde. Tråde gennemgås ligeledes senere i rapporten.

Sekvensdiagram

Sekvensdiagrammer viser et bestemt program flow, med udgangspunkt i en use case specifikation.

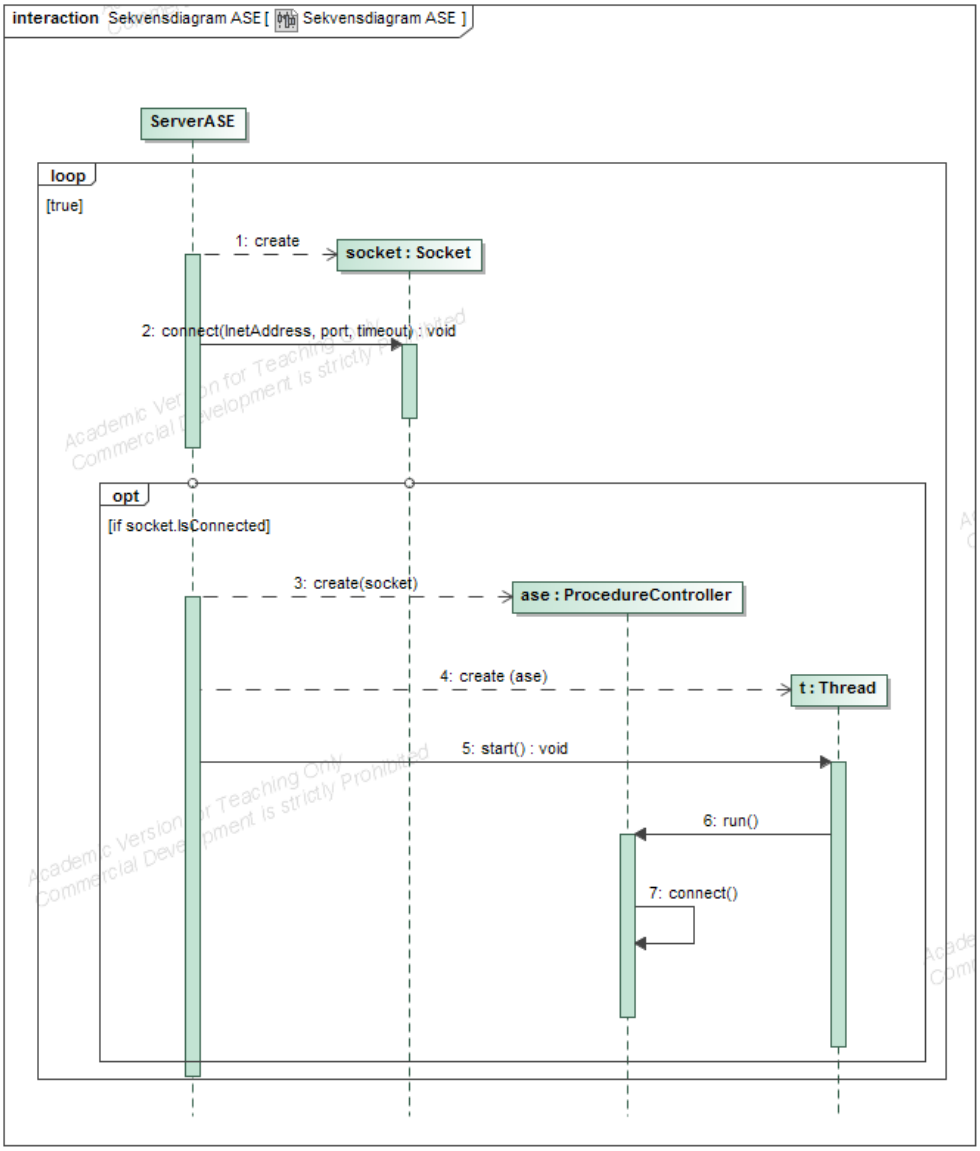
GWT Login



Figur 9: Sekvensdiagram Login

Ovenstående figur illustrerer et sekvensdiagram for Login-funktionen som refererer til use case 0 – Login. Sekvensdiagrammet viser at klienten (Web GUI + Login) kommunikerer med serveren (ServiceImpl). Serveren kontrollerer det data (bruger id + password) den får tilsendt af klienten, op mod data der hentes fra MySQL databasen via Data Access Layer objekterne dao og oprDAO. Hvis bruger id og password fra klienten stemmer overens med databasen, laver serveren en token (via TokenHandler) og returnerer denne til klienten. Herefter opretter klienten et Controller-objekt, og dette objekt overtager herefter styringen af klienten. Brugeren er dermed logget ind.

ASE opstart



Figur 10: Sekvensdiagram ASE

Ovenstående sekvensdiagram viser hvordan en afvejningsprocedure startes når en vægt bliver tændt. Sekvensdiagrammet relaterer til use case 6 – afvejning (specifikation kan findes i bilag). Diagrammet er simplificeret, og viser at ServerASE vil forsøge at oprette en forbindelse til en given IP-adresse og port, med en fastsat timeout på 1 sekund. Lykkes det at oprette forbindelsen oprettes en ProcedureController samt en ny tråd til denne, og tråden startes. Herefter kører afvejningsproceduren på den pågældende IP-adresse/vægt.

GWT

Afsnittet beskriver, hvordan udvalgte dele af GWT applikationen er designet.

GWT applikation i Eclipse

For at oprette et nyt GWT projekt, benyttes det GWT plugin som er installeret i Eclipse. I forbindelse med oprettelse af projektet, genereres en række filer, som er nødvendige for at køre projektet. Herunder de tre service-klasser, til at foretage RPC, som er beskrevet senere i rapporten.

Ud over dette oprettes også en xml fil, en html fil og en css fil.

```
  <!-- Servlets -->
  <servlet>
    <servlet-name>greetServlet</servlet-name>
    <servlet-class>cdio.server.ServiceImpl</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>greetServlet</servlet-name>
    <url-pattern>/_9_cdiofinal/greet</url-pattern>
  </servlet-mapping>

  <!-- Default page to serve -->
  <welcome-file-list>
    <welcome-file>_9_CDIOFinal.html</welcome-file>
  </welcome-file-list>

</web-app>
```

Figur 11: _9_CDIOFinal.gwt.xml

Xml-filen indeholder selve definitionen af GWT modulet, blandt andet hvilken klasse, som er den ønskede entry-point klasse, og hvad navnet er på den html fil, som projektet kører på. Oven for ses et screenshot af projektets XML fil. Som det ses er entry-point klassen, såvel som html filen erklæret.

```
 */
public class _9_CDIOFinal implements EntryPoint {

    /**
     * Create a remote service proxy to talk to the server-side Greeting service.
     */
    private final ServiceAsync service = GWT.create(Service.class);

    /**
     * This is the entry point method.
     */
    public void onModuleLoad() {
        RootPanel.get().add(new Login(service));
    }
```

Figur 12: client._9_CDIOFinal.java

I selve entry-point klassen er GWT interfacet Entrypoint implementeret. I denne er metoden OnModuleLoad() integreret, som er den metode der køres, når programmet initieres. Ovenfor ses hvordan

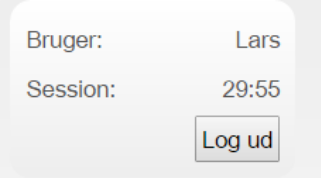
entry-point klassen ser ud, for dette projekt. Som det ses tilføjes en instans af klassen Login til Root Panel, hvilken passer med, at det første der ses, når projektet køres, er login siden.

Når GWT projektet compiles, oversættes det til browser-specifik JavaScript. Det er derfor nødvendigt at have JavaScript slået til når Web GUIen tilgås.

Web GUI features

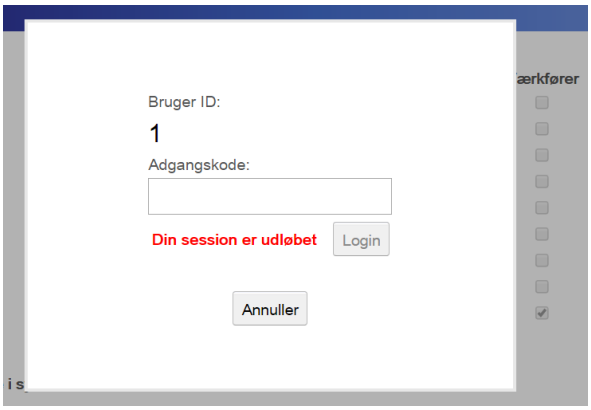
Web GUIen er designet med tanke om at gøre brugen heraf så brugervenlig og intuitiv som muligt. Der er lagt vægt på at gøre processerne (use cases) overskuelige for brugeren, igennem simple valg for brugen på hjemmesiden. Alle menupunkter er givet et sigende navn som beskriver deres funktion, og brugeren kan kun se de menupunkter som han har adgang til at bruge.

Sessionstid



Figur 13: Sessionstid

På hjemmesiden er der indført en nedtælling (resterende sessionstid), for at give kunden en klar fornemmelse for, hvor lang tid de har tilbage af deres tid på hjemmesiden, før de skal logge på igen. Sessionstiden fornys når kunden tilgår systemets server og benytter sin token. Dette beskrives nærmere i afsnittet ”Token baseret Login”.



Figur 14: Login ved udløbet session

Hvis sessionstiden udløber, vil en pop-up aktiveres og brugeren vil blive bedt om at indtaste deres password igen, for at forny deres sessionstid. Selve sessionstiden er pt. sat til 30 minutter. Hvis kunden ikke ønsker en sessionstid på 30 minutter, kan dette nemt ændres i systemet (i *server.TokenHandler* samt *client.Controller.refreshToken()*).

Farvevalg

Figur 15: Web GUI farvevalg

Der er også implementeret et farvevalg til hjemmesiden, hvor det er muligt for kunden at vælge forskellige farver i den nuværende session alt afhængigt kundens præferencer. Dette er implementeret via CSS. Farvevalget gemmes ikke i databasen.

Inputvalidering

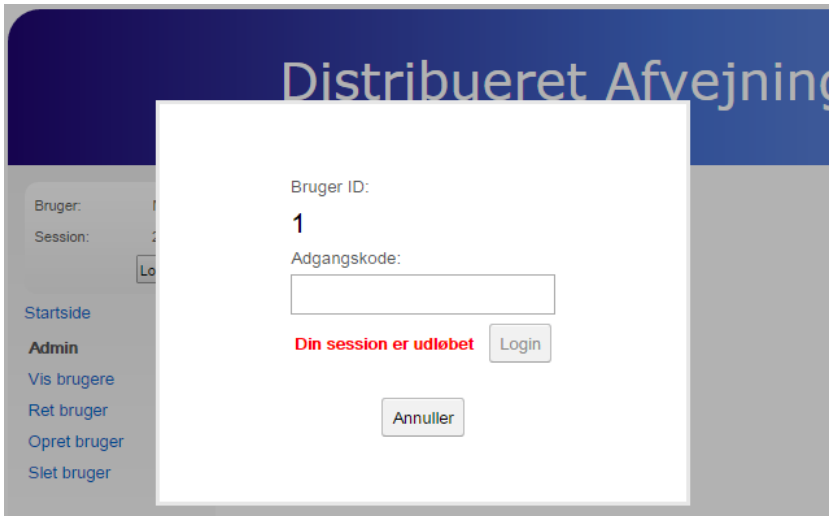
Der er i klienten tilføjet inputvalidering på samtlige felter som synliggøre når en bruger har indtastet et ugyldigt input. I nogle tilfælde er det nødvendigt at brugeren sender det indtastede information til serveren, hvorefter brugeren vil få en passende fejlmeddelelse vist såfremt et felt indeholder ugyldigt input. Inputvalidering på server-siden beskrives nærmere i afsnittet "Client-server principper".

Token-baseret Login

For at logge på Web klienten er det nødvendigt at autentificere sig med et bruger id og et password. Til projektet er det valgt at benytte et token-baseret loginsystem. Dvs. at en bruger kun sender sit id og password til serveren én gang og ud fra dette får udstedt en token fra serveren, som derefter benyttes. Brugeren behøver dermed ikke indtaste sit password igen, før denne token er udløbet. Processen for det implementerede token-baserede loginsystem er som følger:

1. Bruger angiver ID + Password på klienten. Dette sendes til serveren.
2. Server modtager ID + Password og tjekker dette op mod databasen. Hvis ID + Password er korrekt, laver serveren en token ved at kryptere bruger id, timestamp samt en hemmelig sætning. Token sendes retur til klienten, og serveren har ikke nogen reference til den pågældende token liggende.
3. Klienten modtager token fra serveren og holder på denne. Når klienten beder serveren om information, som er beskyttet, medsendes denne token.
4. Serveren modtager token fra klienten og validerer den. Via den hemmelige sætning kan serveren kontrollere, at den pågældende token er lavet af serveren, og via timestamp kan serveren kontrollere at token ikke er udløbet.
5. Hvis tjekket i step 4 er gået godt, udfører serveren den ønskede handling og returnerer resultatet.

I projektet er det valgt at forlænge levetiden på en brugers token, så længe han er aktiv. Dette er implementeret på den måde, at når en bruger har fået et succesfuldt svar fra serveren, beder klienten serveren om at få opdateret timestamp på brugerens token. Serveren gør dette ved at udstede en ny token til den pågældende bruger og returnerer denne, som klienten nu gemmer som sin nye token. Specifikt benyttes den statiske metode *Controller.refreshToken()* til dette.



Figur 16: Skærmdump fra klient når Token er udløbet

I klienten benyttes en GWT Timer til at holde styr på levetiden for brugerens token. Hver gang brugerens token refreshes, nulstilles timeren. Skulle timeren udløbe, indikerer dette at brugerens token også er udløbet. Dette medfører at klienten vil vise en popup (illustreret i ovenstående figur), hvor brugeren bedes indtaste sit password igen for at forny sin token/forlænge sin session. Gøres dette kan brugeren fortsætte hvor han var, og intet data/indtastning vil være mistet. Hvis han har forladt computeren og en anden bruger skal benytte klienten, kan denne bruger klikke på Annuller, og brugeren vil blive sendt til Login-siden hvor han kan angive sit eget bruger id og password. I bilaget Menu.timer() findes koden for hvordan nedtællingen fungerer.

Den implementerede proces for token-baseret login i projektet er desuden dokumenteret i sekvensdiagrammet ”Login”. Bemærk at TokenHandler.java ikke er udviklet af projektgruppen, men er stillet til rådighed af Ronnie Dalsgaard.

Client – Server principper

I udviklingen af GWT applikationen har projektgruppen fastlagt sig på en række klient-server principper, som skal overholdes i al kommunikation mellem klientdelen og serverdelen. Principperne er designet ud fra, at der ikke ønskes at gemmes korrupt data i databasen, samt at sikre adgangsbegrænsning til funktioner baseret på en brugers rolle. Der er dermed forsøgt taget højde for et eventuelt ondsindet angreb på databasen. Principperne er:

1. Al input, der bruges på serveren, valideres vha. metoder i FieldVerifier både i klienten og på serveren.
2. Når en klient requester information fra serveren, medsendes klientens Token. Det er serverens ansvar at tjekke, at Token er valid, og at brugeren bag den pågældende token har adgang til den information, der requestes (dvs. har korrekte rolle).
3. Hvis der opstår en Exception på serveren, kastes den specifikke Exception til klienten.
4. Hvis klienten får kastet en TokenException, skal brugeren autentificere sig igen ved at indtaste sit password (det sker typisk, hvis token er udløbet).
5. På klienten skal samtlige felter, hvor der kan indtastes, sikres mod crosssite scripting vha. metoder i FieldVerifier.

Samlet set er principperne udviklet for at øge brugervenligheden på klientsiden og øge sikkerheden på serversiden. På klienten får man besked, hvis man har indtastet et ulovligt input, inden dette sendes til serveren. På serveren valideres alt input, samt brugerens adgang til den pågældende funktion, inden funktionen udføres. Dette betyder, at der er en betydelig sikring mod at behandle ondsindet og/eller korrupt data og potentielt sende denne til databasen.

GWT Exception håndtering

I systemet benyttes forskellige funktioner som kan kaste en Exception. Disse exceptions bliver i programmet så vidt muligt håndteret hvor de kan opstå.

```
@Override
public List<UserDTO> getOprList(String token) throws TokenException, DALException {
    // Tjek om brugeren bag token har adgang til informationen
    if (!getRole(token).equalsIgnoreCase("Admin"))
        throw new TokenException("Adgang nægtet");

    return dao.getOprList();
}
```

Figur 17: Eksempel på metode der kaster TokenException og DALException

Ovenstående figur viser hvordan det håndteres når listen over brugere skal hentes fra databasen. Først valideres brugeren der har foretaget kaldet – i dette tilfælde en Admin. Hvis dette ikke er tilfældet vil metoden kaste en TokenException der fanges nede i if-sætningen med en besked ”Adgang nægtet”. Denne besked bliver kastet videre op i kaldet, hvor den bruges i en if-sætning i onFailure:

```
@Override
public void onFailure(Throwable caught) {
    if (caught instanceof TokenException){
        final PopupLogin pop = new PopupLogin();
        pop.setPopupPositionAndShow(new PopupPanel.PositionCallback() {
            public void setPosition(int offsetWidth, int offsetHeight) {
                int left = (Window.getClientWidth() - offsetWidth) / 3;
                int top = (Window.getClientHeight() - offsetHeight) / 3;
                pop.setPopupPosition(left, top);
```

Figur 18: Eksempel på onFailure-metode med specifik exception håndtering

Der er også implementeret en DALException i kaldet, som primært bruges i alle de kald der går igennem acces-laget til databasen. DALExceptionen brugers til at kommunikere forbindelsesproblemer til databasen, eller retur-besked fra databasen ved forkert input/forsøg på at hente information der ikke eksisterer.

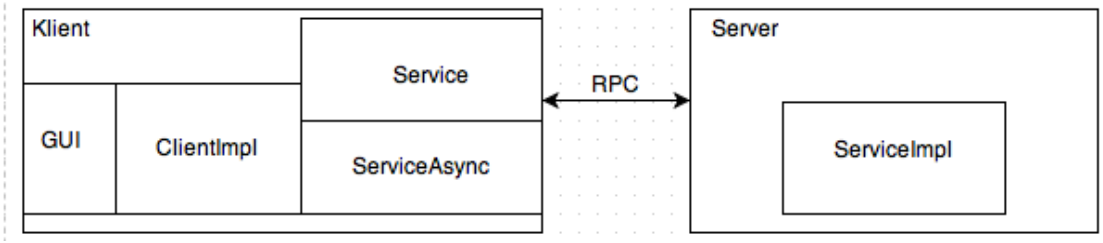
Med Remote Procedure kald, er der to forskellige stadier af exceptions: ukendte og kendte exceptions. Ukendte exceptions er exceptions som programmet ikke specifikt kender til, det kunne f.eks. være at et kald til en database fejler og kaster en IOException. Hvis exceptionen ikke ”kendes” i programmet, vil programmet ikke kunne bruge exceptionen til noget og fejlen vil i bedste fald blive smidt ud som en fejlbesked via onFailure.

Hvis fejlen til gengæld er en kendt exception (f.eks. TokenException eller DALException), vil det være muligt at bruge den exception til at udføre forskellige logiske funktioner i programmet. Eksemplet i ovenstående figur er f.eks. en måde at bruge en kendt exception på.

Asynkrone kald / Remote Procedure Call

RPC, eller remote procedure calls, benyttes mellem klientsiden og serversiden. Overordnet set kan et RPC forstås som et metodekald til serveren.

Normaltvis når en metode kaldes, gøres dette synkront. Det vil sige at der laves en forespørgsel, og der afventes indtil man modtager svar på forespørgslen. Den kode der afvikles i browseren er single-trådet, hvilket betyder at ens browser vil hænge, hvis man udfører RPC synkront, idet den vil stå og afvente svar fra serveren. Løsningen på dette er at gøre kaldene asynkrone i stedet, hvilket betyder at browseren ikke hænger, mens den afventer svar fra serveren, når en metode kaldes. I stedet kaldes metoden blot, og når serveren har udført den påkaldte metode, sendes svaret retur til klientsiden.



Figur 19: Opbygning af RCP interface i GWT

For at dette kan udføres, deklareres et RPC interface. Dette har essentielt tre komponenter:

- Et serviceInterface (i vores tilfælde Service) som extender RemoteService
- En implementationsklasse (i vores tilfælde ServiceImpl) som implementerer det ovenstående interface
- Et asynkront interface (i vores tilfælde ServiceAsync), indeholdende de samme metoder som serviceInterfacet, som kaldes fra koden på klientsiden.

For at kunne benytte de asynkrone kald, skal disse inddrages i metoderne. Et eksempel på dette ser ud på følgende måde:

```
void getRaavareList(String token, AsyncCallback<List<RaavareDTO>> asyncCallback);
```

Som det fremgår af kodeudsnittet, fra ServiceAsync, indgår AsyncCallback altså som parameter i metoden. Idet metodekaldet er asynkront, og det altså ikke i første omgang returnerer noget, er denne form for metoder altid deklareret med void. Man kan dog se det reelle retur parematiseret i ovenstående tilfælde som <List<RaavareDTO>>. Der returneres dermed en liste indeholdende objekter af typen RaavareDTO.

```
Controller.service.getRaavareList(Controller.token, new AsyncCallback<List<RaavareDTO>

public void onFailure(Throwable caught) {
    if (caught instanceof TokenException){
        final PopupLogin pop = new PopupLogin();
        pop.setPopupPositionAndShow(new PopupPanel.PositionCallback() {

public void onSuccess(List<RaavareDTO> result) {
    Controller.refreshToken();
    vPane.clear();
    visRaavarer = new Label("Vis råvarer");
```

Figur 20: Eksempel på onFailure og onSuccess

Til gengæld erklærer man, ved hjælp af onSucces og onFailure metoderne, som ses ovenfor, hvordan resultatet skal behandles, når der kommer et svar fra serveren.

CSS / Web GUI opbygning

Web GUIen er stylet vha. Cascading Style Sheets (CSS). Via CSS kan man styre Web GUIens opbygning af elementer.

Figur 21: Web GUI opbygning i elementer

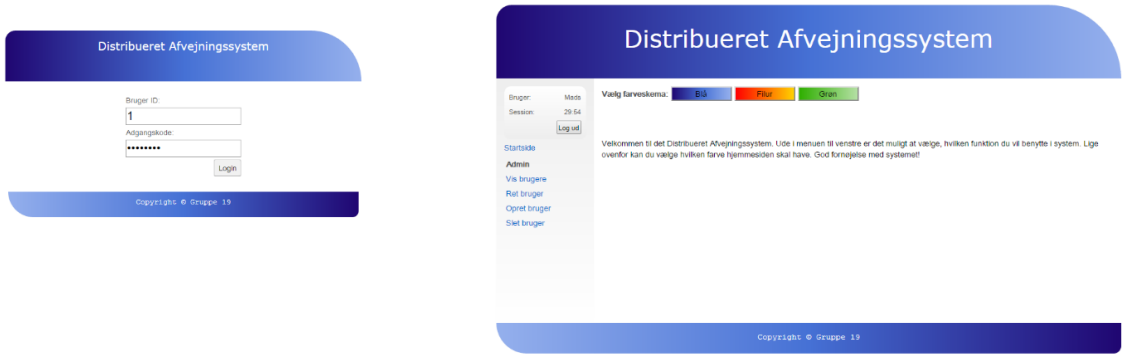
Ovenstående figur viser netop hvordan Web GUIen er opdelt i de forskellige elementer, header, menu, content og footer. På hjemmesiden benyttes en række CSS-kode som ikke fungerer på alle browsere (f.eks. ældre Safari-versioner). Det er valgt at optimere hjemmesiden til Google Chrome (nyere version).

```
75  .Header {
76      border-radius: 25px 120px 0px 0px;
77      background: linear-gradient(to right, #200772 , #4671D5, #95B0ED);
78      height: 120px;
79      padding: 30px;
80      width: 1024px;
81  }
```

Figur 22: Udsnit fra _9_CDIOFinal.css

Ovenfor ses hvordan Header-elementet er stylet (med blå farvevalg). Det ses at de runde kanter på hjemmesiden er sat vha. egenskaben "*border-radius*", og at den graduerede farve sættes vha. "*background: linear-gradient(retning, farve#1, farve#2, farve#3)*". Herudover defineres en højde, bredde samt en afstand fra kanten af elementet til indholdet af elementet (padding).

Det bemærkes at positionen på Header-elementet ikke er bestemt via CSS, og det skyldes et designvalg fra projektgruppen. Gruppen har ønsket en fuldstændig dynamisk Web GUI, som blandt andet betyder at man præsenteres for en Login-skærm der adskiller sig fra resten af Web GUIen.



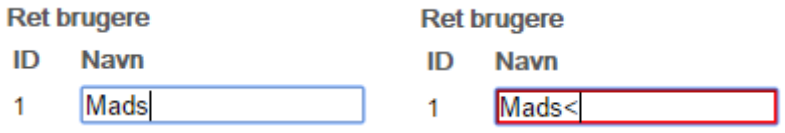
Figur 23: Loginskærm vs. Startpage

For at gøre dette er elementernes (header, content, menu, footer) position defineret i Java-delen af programmet, vha. VerticalPanel og HorizontalPanel.

```
35          // Lav de indledende elementer
36          header = new Header();
37          footer = new Footer();
38          menu = new Menu(this);
39          content = new VerticalPanel();
40          content.add(new StartPage(this));
41
42          // Sæt style på hver element (bestemmer delvis placering)
43          header.setStyleName("Header");
44          footer.setStyleName("Footer");
45          menu.setStyleName("Menu");
46          content.setStyleName("Content");
47
48          // Tilføj basis-elementerne til controlleren
49          vPane.add(header);
50          hPane.add(menu);
51          hPane.add(content);
52          vPane.add(hPane);
53          vPane.add(footer);
```

Figur 24: Udsnit af client.Controller.java

Ovenstående viser hvordan de forskellige elementer oprettes og tilføjes VerticalPanel vPane når der er logget ind, som indeholder samtlige elementer der vises. De forskellige del-elementer tildeles en specifik CSS-style vha. metoden *Element.setStyleName("Style-Name"),* f.eks. som i linje 43 hvor Header-elementet defineres.



Figur 25: Inputvalidering på Ret brugere

Ovenstående viser hvordan inputvalidering synliggøres på Web GUI vha. farvemarkering på det/de relevante felter. Denne styling er også lavet via CSS, og gøres ved at ændre setStyleName på det

pågældende element når en række betingelser gør sig gældende. Betingelserne tjekkes typisk på hvert KeyUpEvent eller OnClick.

```
@media print {
    .Header {
    display: none;
    }
    .Footer {
    display: none;
    }
    .Menu {
    display: none;
    }
    .Button-Ret {
    display: none;
    }
}
```

Figur 26: Udsnit fra .CSS

Ovenstående figur viser hvordan det håndteres på Web GUIen når en bruger med rollen værkfører ønsker at printe en Produktbatch. CSS-koden skjuler ovenstående elementer når der detekteres at browseren er i "print-mode". Resultatet er at værkføreren kun printer det element med style Content, som netop er produktbatchen.

ASE

Afvejnings Styrings Enhed (ASE) er udviklet til at sikre en central styring og ensartethed af afvejningsprocedurer. ASE bygger på den eksisterende afvejningsprocedure (oplyst i projektoplægget) med passende ændringer foretaget i dialog med kunden.

Procedure

Den implementerede afvejningsprocedure er baseret på den eksisterende procedure, men afviger på nogle punkter. Nedenstående skema sammenholder kundens tidligere procedure og den i ASE implementerede.

	Kundens tidligere procedure	ASE implementeret procedure
1	Operatøren har modtaget en produktionsforskrift på papir fra værkføreren	Operatøren har modtaget en produktionsforskrift på papir fra værkføreren
2	Operatøren vælger en afvejningsterminal	Operatøren vælger en afvejningsterminal
3	Operatøren indtaster operatør nr	Operatøren indtaster operatør nr
4	Vægten svarer tilbage med operatørnavn som så godkendes	Vægten svarer tilbage med operatørnavn som så godkendes
5	Operatøren indtaster produktbatch nummer	Operatøren indtaster produktbatch nummer
6	Vægten svarer tilbage med navn på recept der skal produceres (eks: saltvand med citron)	Systemet undersøger om produktbatch er 'under produktion' fortsætter med det manglende
7	Operatøren kontrollerer at vægten er ubelastet og trykker 'ok'	Vægten svarer tilbage med navn på recept der skal produceres (eks: saltvand med citron)
8	Systemet sætter produktbatch nummerets status til "Under produktion"	Systemet sætter produktbatch nummerets status til "Under produktion"

9	Vægten tareres	Operatør placerer første tarabeholder og trykker 'ok'
10	Vægten beder om første tara beholder	Vægten tareres
11	Operatør placerer første tarabeholder og trykker 'ok'	Vægten beder om raavarebatch nummer på første råvare
12	Vægten af tarabeholder registreres	Operatøren afvejer op til den ønskede mængde og trykker 'ok'
13	Vægten tareres	Operatøren fjerner tarabeholder og trykker 'ok'
14	Vægten beder om raavarebatch nummer på første råvare	Systemet registrerer afvejningen med relevant data i databasen
15	Operatøren afvejer op til den ønskede mængde og trykker 'ok'	Pkt. 7 – 14 gentages indtil alle råvarer er afvejet
16	Pkt. 7 – 14 gentages indtil alle råvarer er afvejet	Systemet sætter produktbatch nummerets status til "Afsluttet"
17	Systemet sætter produktbatch nummerets status til "Afsluttet"	Vægten er klar til ny afvejning
18	Det kan herefter genoptages af en ny operatør	

De to procedurer er bevidst opbygget meget ens, dog er punkt 7 i den tidligere procedure (ubelastet vægt) i den nye procedure lagt oven i punkt 9 (påsat tarabeholder). Den endelige implementering fremstår mere strømlinet end ovenstående beskrivelse, da operatøren og vægten udgør en synergetisk enhed, hvilket favoriserer den holistisk udviklede procedure.

Robusthed og brugervenlighed i afvejningsproceduren er bestemt af den implementerede fejlhåndtering. Således afgrænses brugeren så vidt muligt fra at kunne foretage valg eller aktioner der kan resultere i procedure-terminering. Nedenstående skema viser en kortfattet oversigt over de vigtigste fejlhåndteringer proceduren tager højde for.

	ASE implementeret procedure	Fejlhåndtering
1	Operatøren har modtaget en produktionsforskrift på papir fra værkføreren	
2	Operatøren vælger en afvejningsterminal	
3	Operatøren indtaster operatør nr	- Input låst til at acceptere tal - Maksimalt 4 tegn - Rolle kontrol = farmaceut - Cancel = retur til start
4	Vægten svarer tilbage med operatørnavn som så godkendes	- Identificeret navn retur - Cancel = retur til start - 'Q' = retur til start
5	Operatøren indtaster produktbatch nummer	- Input låst til at acceptere tal - Maksimalt 4 tegn - Produktbatch status != 2 - Cancel = retur til start
6	Systemet undersøger om produktbatch er 'under produktion' fortsætter med det manglende	- Produktbatch status = 1 - Kun ikke-allerede afvejede komponenter

7	Vægten svarer tilbage med navn på recept der skal produceres (eks: saltvand med citron) som så godkendes	- Identificeret receptnavn retur - Vilkårligt = nyt produktbatch ID - Cancel = retur til start - 'Q' = retur til start
8	Systemet sætter produktbatch nummerets status til "Under produktion" og registrerer påbegyndt timestamp	- Produktbatch status != 1
9	Vægten beder om raavarebatch nummer på første råvare	- Input låst til at acceptere tal - Maksimalt 4 tegn - Cancel = retur til start
10	Vægten svarer tilbage med råvarenavn som så godkendes	- Identificeret receptnavn retur - Vilkårligt = nyt råvarebatch ID - Cancel = retur til start - 'Q' = retur til start
11	Operatør placerer første tarabeholder og trykker 'ok'	- "OK" retur - Vilkårligt = nyt råvarebatch ID - Cancel = retur til start - 'Q' = retur til start
12	Vægten tareres	
13	Operatøren afvejer op til den ønskede mængde og kvitterer	- Afvejet mængde indenfor tolerance
14	Operatøren fjerner tarabeholder og trykker 'ok'	- "OK" retur - Vilkårligt = nyt råvarebatch ID - Cancel = retur til start - 'Q' = retur til start
15	Systemet registrerer afvejningen med relevant data i databasen	
16	Pkt. 7 – 14 gentages indtil alle råvarer er afvejet	
17	Systemet sætter produktbatch nummerets status til "Afsluttet"	
18	Vægten er klar til ny afvejning	

Til at styre den identificerede procedure er der blevet defineret en variabel (state) af den specielle datatype enum (State). Denne variabel kan udelukkende antage prædefinerede værdier og hver værdi er valgt til i store træk, at svare til et punkt i proceduren. Værdien STOP benyttes til at kunne afslutte en procedureafvejning. Operatøren har ikke adgang til STOP da procedureprogrammet ikke skal kunne stoppes fra vægten, svarende til at en klient ikke kan stoppe et serverprogram. Værdierne med tilhørende procedurehåndteringer er:

- ***START*** (OperatørID, bekræft opreratør navn)
- ***SETUP*** (ProduktBatchID, bekræft recept navn)
- ***CHECK*** (RåvareBatchID til råvare, status kontrol, mængde kontrol)
- ***CLEAR*** (Beholder påsætning, tara registrering)
- ***WEIGH*** (Afvejningsregistrering, tolerance kontrol)

- ***REMOVE_CONTAINER*** (Beholder aftagning, komponent registrering, mængde opdatering)
- ***STOP*** (Procedure stop)

Således kan proceduren styres ved at springe mellem enum'er for at opnå den ønskede funktionalitet. Dette kan lade sig gøre da en while-løkke kører så længe *state* ikke antager værdien "STOP" og i denne løkke kaldes changestate metoden på den senest gemte enum. Changestate metoden er defineret abstrakt i den indlejrede enum-klasse og er derfor implementeret i alle enum-værdier. Enum datatypen gennemgås nærmere i det følgende enum afsnit.

Enums

Enum er en datatype der defineres med en række værdier, som begrænser en variabel af denne enum-type til kun at kunne antage disse definerede værdier. Det noteres at en enum variabel også kan antage værdien "null", dette ses bort fra i den følgende forklaring. Et forklarende eksempel på enum kan være en variabel kaldet "årstid". Denne defineres med værdierne "FORÅR", "SOMMER", "EFTERÅR" og "VINTER". Værdierne skrives med kapitaler da disse er konstante. Herefter kan "årstid" kun antage én af disse fire værdier, på samme måde som en variabel af typen boolean kun kan antage værdierne "null", "true" og "false". Enum'er kan helt simpelt bruges til en switch forgrening eller indeholde primitive værdier, men deres versatilitet gør dem mere anvendelige end dette. En enum deklarering er i realiteten en nested klasse og de definerede værdier fungerer (men er ikke) nedarvede nestede klasser heri. Således kan der deklareres abstrakte metoder i enum, som herefter skal implementeres i de definerede værdier. Dette giver mulighed for at traversere en række enum'er af samme type i en for-each løkke og kalde en enkelt metode, med mulighed for forskelligt indhold, på alle de traverserede værdier.

Med den store versatilitet, er enum ideelt at bruge til at konstruere state-maskiner. Derfor er denne egenskab udnyttet i den implementerede procedurestyring. I enum-klassen state er der defineret to abstrakte metoder desc() og changeState(…), som alle de deklarerede enum-værdier skal implementere i stil med nedarvning fra en abstrakt klasse med metoder. Desc() bruges til at udskrive navnet på den pågældende værdi som debug i konsollen. ChangeState(…) indeholder logikken bag hvert skridt i proceduren repræsenteret af de individuelle værdier. Således kan changeState(…) kaldes på variablen state og alt efter hvilken værdi state har bliver den korresponderende changeState(…) metode kaldt. På denne måde kan der nemt skiftes frit mellem de forskellige skridt i proceduren ved blot at have en enkelt linje kode i et loop, frem for en mindre elegant løsning med mere simple datatyper. I den implementerede løsning er changeState(…) deklareret til at returnere en enum af type state. Således holder procedure-klassen en aktuel state-variabel der bliver opdateret når den seneste changeState(…) returneres. I de individuelle changeState(…) kan det afgøres hvorledes proceduren skal fortsætte, skal der startes forfra grundet et fejlagtigt input, skal det aktive skridt gentages eller kan proceduren fortsætte til næste skridt. Dette giver en dynamisk implementering der kan tilpasses koncist til kundens ønske.

Socket programmering

En socket er et endepunkt i en to-vejs kommunikation mellem to programmer på et netværk. Hver socket er en forbindelse mellem transportlaget og applikationslaget (OSI model). En socket defineres bl.a. vha. en port som bliver bundet til socketen, når den oprettes. I dette system kan man redigere i klassen ServerASE.java for at tilføje yderligere IP-adresser samt ændre hvilken port der forsøges at oprettes forbindelse til vægtene.

```
50    @Override
51    public void connect(String host, int port){
52        try (Socket socket = new Socket(host, port);
53                PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
54                BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));){
```

Figur 27: Udklip fra ProcedureKontroller.java - oprettelse af en klient socket

I dette program er der udelukkende programmet klient-sockets, idet den fysiske vægt fungerer som en server. I ovenstående figur vises hvordan der oprettes en socket til en givet host og port. Herefter findes socketens outputstream og inputstream. Disse bindes til et PrintWriter og BufferedReader-objekt som derefter kan benyttes til at læse og sende data over socketen.

Threads

Systemet er designet så det kan benyttes i den virkelige verden, dvs. at man kunne forestille sig en virksomhed havde mere end én vægt, og gerne ville kunne foretage afvejning på mere end én vægt af gangen. For at håndtere dette, benytter ASE sig af tråde (threads). Klassen ServerASE.java er lavet netop til dette formål. Klassen fungerer som en slags server, der konstant forsøger at oprette forbindelse til en række foruddefineret IP-adresser på port 8000. Hvis en forbindelse oprettes, betyder dette at vægten med den pågældende IP-adresse er blevet tændt, og ServerASE vil starte en ny ProcedureController (ASE) i en separat tråd. IP-adressen fjernes herefter fra listen af IP-adresser der forsøges at oprette forbindelse til.

```
70    IProcedureController ase = new ProcedureController(menu, dao , ip.get(i), port, trans);
71    Thread t = new Thread((Runnable) ase);
72    t.start();
```

Figur 28: Udklip fra ServerASE.java

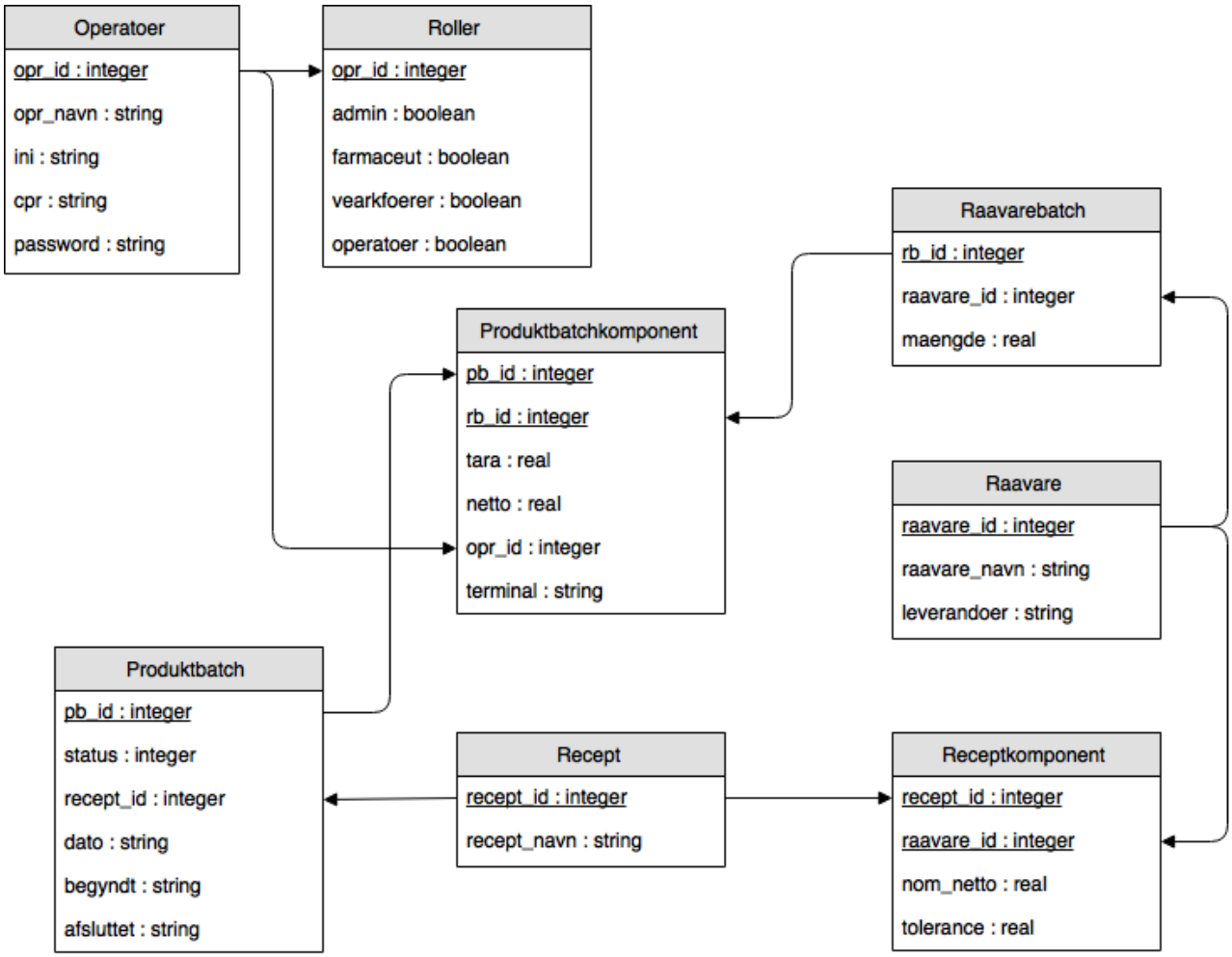
I ovenstående figur vises netop hvordan en ny tråd startes (inden da er det kontrolleret at vægten der forsøges oprettet en forbindelse til er tændt). Tråden startes ved at bruge konstruktøren i Thread, som tager imod en reference til et Runnable objekt. Objektet ase er af typen ProcedureController, men implementerer også Runnable. Derfor kan polymorfisme benyttes til at caste objektet til Runnable. Herefter kaldes t.start() som starter metoden ase.run() i en separat tråd fra Main tråden. Efter oprettelsen af denne nye tråd kører der dermed mindst 3 tråde i programmet (Main, t, garbage).

Der er forsøgt at indbygge en logik til at fange når en vægt slukkes. Dette gøres ved at ServerASE beholder en reference til den startede tråd, og løbende tjekker om den pågældende tråd fortsat er aktiv (Thread.isAlive()). Hvis tråden ikke er aktiv, kan det konkluderes at afvejningsproceduren på vægten med den IP-adresse der korresponderer til den pågældende tråd er afsluttet. Dette kan kun være tilfældet hvis vægten er blevet slukket. I så fald vil ServerASE tilføje vægtens IP-adresse til listen over IP-adresser den skal forsøge at oprette forbindelse til. For at dette fungerer vil det dog være nødvendigt at sætte Socket.setSoTimeout() så metoderne der læser data fra vægten (via BufferedReader.readLine()) ikke blokerer for evigt. Da projektgruppen ikke kender til kundens produktionsdetaljer på dette punkt, og derfor ikke kan bestemme et passende timeout interval, er det valgt ikke at implementere denne funktion. Det betyder at såfremt en fysisk vægt slukkes, skal ServerASE genstartes før der oprettes forbindelse til vægten igen.

Database(MySQL)

Afsnittet vil dokumentere udvalgte elementer af databasedesignet, samt beskrive nogle af de centrale termer der benyttes i forbindelse med databasen.

Logisk Skema (Relations model)



Figur 29: Logisk Skema (MySQL)

Ovenstående figur viser det logiske skema (relations skemaet) for databasen. Skemaet viser opbygningen af databasen, dvs. hvordan de forskellige tabeller er sammensat af attributter, primærnøgler og fremmednøgler. Primærnøgler identificeres vha. understregning, og fremmednøgler er indikeret ved pile. F.eks. er attributten recept_id en fremmednøgle i tabellen Produktbatch. Yderligere afhængigheder er ikke markeret i figuren.

Databasen er videreudviklet ud fra en opgave afleveret i kursus 02327.

JDBC

JDBC (Java-Database-Connectivity) er et Java-API, der gør det muligt for et Java program, at arbejde sammen med en database vha. SQL. For at dette kan lade sig gøre, skal JDBC driveren installeres og lokaliseres i ”ext” mappen i Java mappen på C: drevet.
Kommunikationen mellem Eclipse og MySQL muliggøres via klassen DriverManager, som er en integreret

del af Java-API'et. Et objekt af driveren er påkrævet, hvorefter det skal registreres. Alt dette gøres ved brug af metoden forName("com.mysql.jdbc.Driver").newInstance(). Når driveren er registreret, skal forbindelsen oprettes. Også til dette bruges DriverManager.

```
public static Connection connectToDatabase(String url, String username, String password)
        throws InstantiationException, IllegalAccessException,
                ClassNotFoundException, SQLException
{
    // call the driver class' no argument constructor
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    // get Connection-object via DriverManager
    return (Connection) DriverManager.getConnection(url, username, password);
}
```

Figur 30: Initiering af JDBC driveren

Ovenstående billede er et udsnit fra det omtalte. Der ses både registreringen af driveren, samt oprettelsen af forbindelsen.

I projektet udføres der SQL-operationer for både at hive information ud af databasen samt at ændre i databasen, og dette muliggøres ved, at oprette et objekt af klassen Statement.

```
conn    = connectToDatabase("jdbc:mysql://"+server+":"+port+"/"+database,
            username, password);
stm     = conn.createStatement();
```

Figur 31: Oprettelse af stm

Fra kode udsnittet på ovenstående billede ses det, at "Statement stm = conn.createStatement();" er afhængig af forbindelses objektet "conn". Der benyttes to metoder i Statement klassen. Det er henholdsvis executeUpdate samt executeQuery.

```
public static int doUpdate(String cmd) throws DALException
{
    try { return stm.executeUpdate(cmd); }
    catch (SQLException e) { throw new DALException(e); }
}
```

Figur 32: statisk metode til at opdatere databasen

Til opdateringer af databasen (ændringer) bruges executeUpdate()-metoden. Det ses på billedet som "stm.executeUpdate(cmd)". Returtypen er en Int, som returnerer alt lige fra 0 og op efter. Tallet indikerer hvor mange rækker der er blevet påvirket af SQL-operationen.

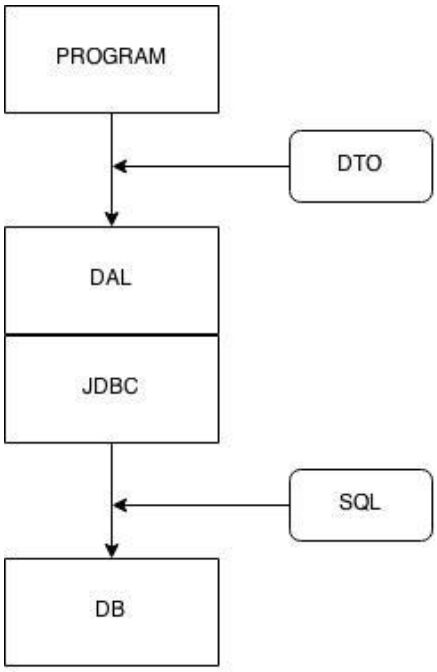
```
public static ResultSet doQuery(String cmd) throws DALException
{
    try { return stm.executeQuery(cmd); }
    catch (SQLException e) { throw new DALException(e); }
}
```

Figur 33: Metode til at hente information fra databasen

Til forespørgsler af databasen (SELECT-sætninger i SQL) bruges executeQuery()-metoden. I denne metode er returtypen ResultSet, som returneres i et objekt/tabel. Generelt vil der ved en forespørgsel blive returneret et svar indeholdende flere rækker. For at alt data bliver gennemgået, benyttes next()-metoden. Next()-metoden tjekker, om den næste række findes. Antag at denne SQL-operation blev foretaget i databasen: ”SELECT * FROM operatoer”. Der vil ResultSet indeholde hele operatoer tabellen. Next()-metoden sørger så for, at alle rækker blive løbet igennem, så den fundne data kan udskrives eller på anden vis anvendes.

Data Access Layer / Data Transfer Object

For at sikre lav kobling imellem selve Java-programmet og databasen, benyttes ét specifikt lag, til at kalde metoder til databasen. Ud over at de to dele i systemet i høj grad holdes adskilt, gør denne struktur det også nemmere, hvis det fremtidigt ønskes at opdatere logikken i enten databasen eller i Java-programmet. I tilfælde af at dette sker, er der et minimalt antal klasser, som skal rettes til for at programmet kan køre.



Figur 34: Sammenhæng mellem Database, JDBC, DAL, DTO og Javaprogram

Som det fremgår af ovenstående figur, ligger selve Data Access laget sammen med JDBC i mellem program og databasen. Til sammen udgør de to et bindeled mellem program og database, og det er herigennem metoderne kaldes og data overføres imellem program og database.

Til formålet deklareres to typer af klasser: DTO (Data Transfer Object) og DAO (Data Access Object). DTOen er et objekt der indeholder data som skal overføres i mellem processer. I dette tilfælde fungerer det således at DTOen ud pakkes i DAL, hvorefter dens data benyttes til metoder, der kaldes videre ned til

databasen. DTOen indeholder ikke nogen logik, men blot den information som skal overføres til og fra databasen.

Nedenfor ses et eksempel fra projektet der illustrerer, hvordan en DTO kan se ud:

```
public class ReceptKompDTO implements Serializable
{
    private int receptId;                   // auto genereres fra 1..n
    private int raavareId;                 // i omraadet 1-99999999
    private double nomNetto;                // skal vaere positiv og passende stor
    private double tolerance;               // skal vaere positiv og passende stor


    public ReceptKompDTO(){
```

Figur 35: Udsnit af ReceptKompDTO

Som det ses, indeholder klassen ikke nogen logik, men blot simpel data.

DAOen fungerer som et interface til databasen. Denne indeholder de metodekald, som kaldes til databasen.

```
public ReceptKompDAO(TextReader txt) throws FileNotFoundException{
    this.txt = txt;
}

@Override
public ReceptKompDTO getReceptKomp(int receptId, int raavareId) throws DALException {
    ResultSet rs = Connector.doQuery(txt.getReceptKomp(receptId, raavareId));
    try {
        if (!rs.first()) throw new DALException("ReceptKomponent " + receptId + ", " + raavareId + " findes ikke");
        return new ReceptKompDTO (rs.getInt("recept_id"), rs.getInt("raavare_id"), rs.getDouble("nom_netto"),rs.getDoubl
    }
    catch (SQLException e) {throw new DALException(e); }
}
```

Figur 36: Udsnit af ReceptKompDAO

Ovenfor ses et eksempel på selve DAOen. Som det fremgår af koden, udføres en doQuery, som beskrevet i ovenstående afsnit. Denne tager som input en ny metode fra TextReader klassen, som indeholder logikken til at indsætte data fra Java-programmet i SQL-forespørgslen.

```
public String getReceptKomp(int receptId, int raavareId){
    String output = sqlCommands[18];
    output = output.replaceFirst(illegalString + "1", Integer.toString(receptId));
    output = output.replaceFirst(illegalString + "2", Integer.toString(raavareId));
    return output;
}
```

Figur 37: Metode i TextReader.java

Metoden fra TextReader klassen ses i ovenstående billede. Som det ses er der ikke eksplicit sql kode i metoden, men i stedet er sqlCommands[] en reference til en fil (war/WEB-INF/sqlCommands.txt), der er blevet indlæst og lagt i et array af Strings.

```
SELECT * FROM receptkomponent WHERE recept_id = #1 AND raavare_id = #2
```

Ovenstående er netop den SQL kode, der ligger i sqlCommands[18]. Som det ses sætter getReceptKomp output til at være SQL koden. Denne returneres så til getReceptKomp() i receptKompDao, hvor SQL forespørgslen, via DoQuery, bliver sendt til databasen. Når getReceptKomp modtager svar fra databasen,

behandles dette, og enten returneres den forespurgte receptkomponent, ellers returneres en passende exception.

Slet råvarer

I projektet blev det ønsket at man skulle kunne slette råvarer, råvarebatches, recepter, m.m. forudsat de pågældende elementer endnu ikke var brugt til at producere en produktbatchkomponent. For at gøre dette benyttes MySQL afhængigheden ON DELETE CASCADE.

ON DELETE CASCADE er en funktion i MySQL der muliggør direkte sletning i en parent tabel, så man undgår at skulle slette fremmednøglerne først, i alle child tabellerne.

I projektet er dette implementeret i databasen på tabel 'raavare' samt 'recept'. Idet ON DELETE CASCADE er implementeret på samme måde på såvel 'raavare' samt 'recept' vil implementeringen kun blive forklaret med udgangspunkt i 'raavare', da de begge virker på samme måde.

```
CREATE TABLE raavare(raavare_id INT PRIMARY KEY, raavare_navn TEXT, leverandoer TEXT) ENGINE=innoDB;

CREATE TABLE raavarebatch(rb_id INT PRIMARY KEY, raavare_id INT, maengde REAL, CONSTRAINT raavarebatch
   FOREIGN KEY (raavare_id) REFERENCES raavare(raavare_id) ON DELETE CASCADE) ENGINE=innoDB;
```

Figur 38: Udsnit af 19_db.sql

Ovenstående billede er et udsnit fra database scriptet. Det bemærkes at ON DELETE CASCADE ikke defineres på parent tabellen, men derimod på child tabellen 'raavarebatch'. Definitions syntaksen er til fordel for programmøren, men inden videre gennemgang af hvorfor, skal det påpeges, at 'raavare' ikke kun har en fremmednøgle i 'raavarebatch, men også i 'receptkomponent'. Desuden må en råvare ikke slettes hvis denne bliver brugt af en recept. Af netop denne grund, er ON DELETE CASCADE ikke implementeret på 'receptkomponent', da der ved forsøg på at slette en råvare der indgår i en recept, vil blive smidt en SQLException, som håndteres. Denne SQLException indikerer at råvaren, der ønskes slettet, eksisterer i en recept, da receptkomponent ikke har implementeringen af ON DELETE CASCADE, og vil derfor bryde reglen om, at fremmednøgler skal slettes før primærnøgler. Da man ved at der vil opstå en SQLException, hvis råvaren bliver brugt i en recept, kan man på denne måde bruge Exception håndteringen til at måle på dette, og dermed sikre at man ikke sletter en råvare, som er brugt i en produktbatchkomponent.

Interfaces

Et interface er en grænseflade for en anden klasse i et program. Interfaces bliver brugt til at skabe kobling imellem klasser der snakker indbyrdes med hinanden. Med interfaces er det muligt at skifte en klasse med et dertilhørende interface ud, med en anden klasse, for så stadigvæk at bruge det samme interface. Dette kan lade sig gøre fordi klassen implementerer interfacet. Dette kan ske for flere klasser en bare en klasse, dvs. at Klasse A og Klasse B kan implementere det samme interface i et program.

```
22 public interface IControllerDAO {
23
24     IOperatoerDAO getOprDAO();
25
26     IProduktBatchDAO getPbDAO();
27
28     IProduktBatchKompDAO getPbKompDAO();
29
30     IReceptDAO getReceptDAO();
```

Figur 39: Udsnit af interfacet IControllerDAO

Ovenstående figur viser et udsnit fra interfacet IControllerDAO. Interfacet består af deklarationen af en række metoder. Alle metoder i et interface er som udgangspunkt *abstract public*, også selvom dette ikke fremgår af ovenstående figur.

```
34 public class ControllerDAO implements IControllerDAO {

69     public IOperatoerDAO getOprDAO(){
70         return oprDAO;
71     }
72
73     public IProduktBatchDAO getPbDAO(){
74         return pbDAO;
75     }
76
77     public IProduktBatchKompDAO getPbKompDAO(){
78         return pbkompDAO;
79     }
```

Figur 40: Udsnit af ControllerDAO

I ovenstående figur ses et eksempel på en implementering af interfacet. Her er det ControllerDAO der implementerer IControllerDAO, som er et interface. Det ses at metoderne der er deklareret i IControllerDAO er implementeret i ControllerDAO.

Interfaces er meget populære at benytte i en såkaldt 3-lagsmodel, hvor der benyttes en grænseflade, funktionalitetslag og et Data lag, med hvert deres interface(s). Dette betyder at man nemt kan udskifte de forskellige lag uden af programmet ”går i stykker”. Man kunne for eksempel forestille sig at et program havde to forskellige grænseflader, en GUI og en textbaseret UI. Hvis programmet benytter et interface, kan man meget nemt skifte mellem de to.

Delkonklusion Design

Designafsnittet er, som så meget andet i rapporten, opdelt i de tre delelementer i programmet. Således lægges der i afsnittet ud med at gennemgå to overordnede klassediagrammer, ét for GWT og et for ASE. Deres formål er at give et godt overblik over hvordan systemet er opbygget. Efterfølgende er der givet et sekvensdiagrammer, som viser hvorledes login situationen er implementeret i GWT, samt et sekvensdiagram for ASE som viser hvordan en afvejning startes. Sekvensdiagrammernes formål at er at give et dybere, mere konkret indblik i hvordan opgaven reelt er løst ud fra de to eksempler.

Efter det følger en gennemgang af hvordan GUIen er lavet ved hjælp af GWT. Idet GWT har været nyt for projektgruppen, er der her også gennemgået flere aspekter af hvordan GWT virker, ved hjælp af eksempelvis asynkrone kald.

ASE-programmet er opbygget ved hjælp af enums, som giver hvert step i en afvejningsprocedure hver sin state. Denne løsningsmetode er også gennemgået, samt hvordan afvejningsproceduren er tolket og implementeret.

Til sidst gennemgås hvordan databasen er opbygget. Her lægges ud med et logisk skema, som har til formål at give et godt overblik over hvad databasen indeholder. Ydermere er der gennemgået hvordan databasen er implementeret, og hvordan DAL benyttes til at forbinde database og program med hinanden.

Test

Formålet med at teste er, at kvalitets- og kvantitetssikre alle krav, som er blevet stillet fra kunden i starten af projektet. Kravspecifikationerne er blevet brugt til at designe brugertests, der er blevet udført. Dette er sket gennem en iterativ proces, hvor brugertest har været i centrum, da en stor del af projektet består af en WEB GUI, hvor JUnit-testing ikke er praktisk. Der er udført JUnit-tests på de dele af serveren, som kan JUnit-testes, dvs. DAO-objekterne.

Brugertest

I projektet er det valgt at fokusere på brugertest og JUnit test, for at opnå størst mulig kvalitetskontrol. Der er lavet tre GWT brugertest – en for hver af de tre roller der kan benytte WEB GUIen. Hver brugertest er forskellig, da der lægges vægt på at vise så stor en del af systemet som muligt. De tre brugertest er vedlagt i bilag under navnene: Opret bruger, Ret råvare og Opret produktbatch.

For ASE er der også udført en brugertest, der viser en komplet afvejningsprocedure fra Operatørens synspunkt. Denne findes i bilaget ASE afvejning.

Server svartid

For at simulere scenariet, hvor serveren er lang tid om at svare, er der udført en brugertest, hvor serverens responstid er sat til 2 sekunder på alle operationer. Testen medførte, at flere dele af klienten er blevet tilpasset, så der tages passende højde for serverens svartid. F.eks. bliver en knap nu inaktiv, når der trykkes på den, indtil der modtages et svar fra serveren. Dette gør klienten betydeligt mere brugervenlig i tilfælde af, at serveren har en længere svartid end normalt. Ønsker man bevidst at 'ødelægge' klienten, kan dette dog stadig lade sig gøre.

Testen er udført ved at ændre variablen *TEST_DELAY* (boolean) i ServiceImpl til true.

```
if (TEST_DELAY)
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e1) {}
```

Figur 41: Udsnit af testkode fra ServiceImpl.java

Hver enkelt metode i ServiceImpl har ovenstående kode som første del af deres metode.

JUnit test

For at sandsynliggøre at systemet opfører sig som forventet, og ikke har nogen fejl, er der lavet et antal JUnit tests. JUnit tests tester den forventede opførsel for en specifik metode i programmet. Dette betyder at koden bliver brudt op i komponenter og testet særskilt. Hver DAO klasse har således gennem JUnit fået testet dennes metoder, som er fundet relevant at teste. Herunder metoder som get, update og create. De udførte JUnit tests kan findes i mappen 'test' under pakken 'cdio'.

cdio.client	0,0 %	0	2.134	2.134
cdio.server.ASE	0,0 %	0	1.825	1.825
cdio.server	0,0 %	0	1.283	1.283
cdio.server.DAL	55,1 %	1.463	1.193	2.656
TextReader.java	72,8 %	1.199	449	1.648
ControllerDAO.java	0,0 %	0	421	421
ScriptRunner.java	50,4 %	199	196	395
DTO.java	0,0 %	0	104	104
Connector.java	73,9 %	65	23	88
cdio.shared	18,6 %	214	934	1.148
cdio.server.DAL.dao	52,4 %	707	641	1.348
ProduktBatchDAO.java	40,1 %	127	190	317
OperatoerDAO.java	54,9 %	195	160	355
ReceptKompDAO.java	49,0 %	74	77	151
ProduktBatchKompDAO.java	55,0 %	93	76	169
RaavareBatchDAO.java	54,3 %	76	64	140
RaavareDAO.java	60,2 %	74	49	123
ReceptDAO.java	73,1 %	68	25	93
test	88,3 %	1.568	207	1.775

Figur 42: Code coverage for JUnit tests

Ovenstående billede viser Code coverage når programmets JUnit tests er afviklet, fordelt på hver package i projektet. Det ses at JUnit tester på DAO implementeringen (52,4%) og DAL (55,1%).

Til JUnit testene blev der hentet en ekstern klasse 'JDBC ScriptRunner' grundet problemer med databasen. Utalligt afviklede JUnit test fik databasen til at opføre sig ukorrekt og sende DALExceptions. Af denne grund blev JDBC ScriptRunner hentet og lagt i DAL package, da den skal have et Connection-objekt, hvorved det muliggøres at eksekvere database-scriptet via Eclipse således, at databasen renses efter hver JUnit test.

Konklusion

I projektet er der blevet lagt stor vægt på adskillelse af delelementer i systemet. F.eks. er der udarbejdet tre forskellige kravspecifikationer til hhv. database, WEB GUI og ASE-program. Dette har betydet at overskueligheden i projektet har været ligetil og det har især fremmet overskueligheden i arbejdsfordelingen igennem projektforløbet. Der er blevet lagt vægt på at skabe gode rammer for den kreative proces i forløbet, ved at have fokus på de to obligatoriske milestones, samt jævn fordeling af arbejdsopgaver på projektgruppens medlemmer. Der er aktivt blevet taget stilling til, hvorvidt nye identificerede opgaver har været in-scope eller out-of-scope.

Alle use cases er gennemtjekket for hvilke krav de dækker, og dette har ledt til en høj kvalitet i kontrollen af krav-dækning i projektet. Ydermere er use case-modellen i analysen simplificeret for at give et nemt overblik over hvilke use cases der er lagt vægt på.

I projektets design-del er klassediagrammerne blevet simplificeret for at give overskuelighed til modtager. Dette resulterer i en nem forståelig tilgang til hvordan systemet fungerer og et godt overblik over del-elementets arkitektur. Oven i dette er der også lagt vægt på simplificeringen af det udvalgte sekvensdiagram, hvor man nemt kan overskue hvad der sker i login-processen.

For at give et godt overblik af systemet, er der i rapporten lagt vægt på at fortæller om indviklede og komplicerede funktioner i systemet. Især funktionerne imellem klient og server bliver forklaret, samt diverse ASE- og JDBC-funktioner, hvor flere af indlæringsmålene bag funktionerne er forklaret i dybden.

I test-fasen af systemet er WEB GUIen blevet brugertestet, hvor resten af systemet er blevet testet igennem JUnit test. F.eks. er der blevet testet med en 2 sekunders svar-tid på serverdelen, for at komme så tæt på 'rigtige' forhold som muligt. Tilsammen har dette givet en høj kvalitetskontrol af især WEB GUIen, hvor størstedelen af alle fejl har ligget.

Fremtidigt arbejde

Der er i projektet fokuseret på at udvikle en velfungerende og robust prototype. Samtidig har der været et stort fokus på at gøre afvejningsproceduren og bruger-klienten så brugervenlig og intuitiv som muligt. Med det nævnt har projektgruppen identificeret følgende opgaver som kunne viderearbejdes på såfremt prototypen skal videreudvikles:

- Web GUI kan gøres yderligere robust mht. forsinkelser i kommunikationen med serveren og databasen.
- Der kunne udvikles et operatør-view til Web GUI hvor en operatør kunne se hvad han har afvejet i en given periode.
- Der kunne tilføjes søge- og sorteringsmuligheder til diverse Web GUI funktioner.
- ASE kunne gøres mere robust, og muligvis simplificeres så den er nemmere at debugge.
- Der kunne defineres en passende timeout på forbindelsen mellem ASE og en vægt, så processen nulstilles efter et vist antal minutter uden aktivitet.

Bilag

Kildehenvisninger

www.wikipedia.com

www.w3schools.com

www.tutorialspoint.com

www.stackoverflow.com

www.youtube.com (GWT tutorials)

http://pastebin.com/f10584951 (JDBC Scriprunner)

docs.oracle.com/en/java

www.gwtproject.org

Krav vs. Use cases

Krav/UC	0	1.1	1.2	1.3	1.4	2.1	2.2	2.3	3.1	3.2	4.1	4.2	5.1	5.2
4 roller i systemet (Admin, Farmaceut, Værkfører, Operatør)	X													
Forskellige funktioner mulige for hver rolle														
Administrator kan oprette operatører		X												
Administrator kan rette operatører			X											
Administrator kan 'slette' operatører				X										
Administrator kan vise operatører					X									
Administrator kan oprette farmaceuter		X												
Administrator kan rette farmaceuter			X											
Administrator kan slette farmaceuter				X										
Administrator kan vise farmaceuter					X									
Administrator kan oprette værkførere		X												
Administrator kan rette værkførere			X											
Administrator kan slette værkførere				X										
Administrator kan vise værkførere					X									
En bruger med rollen Operatør må aldrig slettes				X										
Farmaceut kan oprette råvarer						X								
Farmaceut kan rette råvarer							X							
Farmaceut kan vise råvarer								X						
En råvare defineres ved et ravarenummer, navn samt leverandør						X								
Farmaceut kan oprette recepter									X					
Farmaceut kan vise recepter										X				
En recept defineres ved: receptnummer, navn og receptkomponenter									X					
En receptkomponent består af: råvarem mængde, type									X					
Værkfører kan oprette råvarebatches											X			
Værkfører kan vise råvarer												X		
En råvarebatch defineres ved: rbbatchnummer og mængde											X			
Værkfører kan oprette produktbatches													X	
Værkfører kan vise produktbacthes														X
En produktbatch defineres ved: pbnummer, receptnr, dato, status													X	
En oprettet produktbatch skal udprintes													X	
Der skal være input validering på alle felter	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Brugere skal autentificeres ved login.	X													

Figur 43: Oversigt over hvilke krav der dækkes af use cases

Menu.timer()

```
339     public static void timer(){
340         sec = new Timer(){
341             @Override
342             public void run() {
343                 if (Controller.seconds < 10)
344                     time.setText(Integer.toString(Controller.minutes) + ":0" + Controller.seconds);
345                 else
346                     time.setText(Integer.toString(Controller.minutes) + ":" + Controller.seconds);
347                 Controller.seconds--;
348                 if (Controller.seconds == -1){
349                     Controller.seconds = 59;
350                     Controller.minutes--;
351                     if (Controller.minutes < 5){
352                         time.setStyleName("Time-Error");
353                         if (Controller.minutes == -1){
354                             cancelTimer();
355                             time.setText("00:00");
356                             PopupLogin pop = new PopupLogin();
357                             pop.center();
358                         }
359                     }
360                     else
361                         time.setStyleName("TextLabel-Logud");
362                 }
363             }
364         };
365         sec.scheduleRepeating(1000); // Opdater hvert sekund
366     }
```

Figur 44: Udklip fra Menu.java i package cdio.client

Use case specifikationer

Use case: Login

ID: 0

Kort beskrivelse:

Eksempel på situation hvor en farmaceut logger ind på WEB Gui

Primære aktører:

Farmaceut

Sekundære aktører:

-

Preconditions:

- Farmaceuten er oprettet som bruger i systemet
- Farmaceutens Web GUI er tilsluttet serveren og databasen

Main flow:

- Farmaceut åbner program via browser
- Farmaceut indtaster sit brugerid og sit password
- Farmaceut trykker 'ok'

Postconditions:

Farmaceut kan nu se en passende menu som indeholder mulighederne for at vise, rette samt slette råvarer og recepter.

Alternative flows:

- Farmaceut indtaster forkert password og bliver nægtet adgang
- Farmaceut indtaster forkert bruger id og bliver nægtet adgang

Use case: Bruger administration

ID: 1.1

Kort beskrivelse:

Eksempel på situation hvor en administrator opretter en ny farmacaut

Primære aktører:

Administrator

Sekundære aktører:

-

Preconditions:

Farmaceuten er ikke oprettet i systemet i forvejen

Main flow:

- Administrator logger ind på web gui
- Administrator vælger ”opret bruger”
- Administrator udfylder passende information om farmaceuten
- Administrator afslutter med ”opret”

Postconditions:

Farmaceuten er nu oprettet i systemet

Alternative flows:

Administrator indtaster ugyldigt input og får vist en fejl

Use case: Råvare administration

ID: 2.2

Kort beskrivelse:

Eksempel på situation hvor en farmaceut retter en råvare

Primære aktører:

Farmaceut

Sekundære aktører:

-

Preconditions:

- Råvaren er oprettet i systemet
- Råvaren er ikke blevet benyttet i en produktbatchkomponent

Main flow:

- Farmaceut logger ind på web gui
- Farmaceut vælger ”ret råvarer”
- Farmaceut ændrer leverandøren af en råvare
- Farmaceut afslutter med ”ret”

Postconditions:

Leverandøren af råvaren er nu rettet

Alternative flows:

-

Use case: Recept administration

ID: 3.2

Kort beskrivelse:

Eksempel på situation hvor en farmaceut ser en recept

Primære aktører:

Farmaceut

Sekundære aktører:

-

Preconditions:

Der findes mindst en recept i systemet

Main flow:

- Farmaceut logger ind på web gui
- Farmaceut vælger ”Vis recept”
- Farmaceut ser recepten

Postconditions:

-

Alternative flows:

-

Use case: Råvarebatch administration

ID: 4.1

Kort beskrivelse:

Eksempel på situation hvor en værkfører opretter en råvarebatch

Primære aktører:

Værkfører

Sekundære aktører:

-

Preconditions:

-

Main flow:

- Værkfører logger ind på web gui
- Værkfører vælger ”opret råvarebatch”
- Værkfører noterer hvilken råvare det drejer sig om, samt dennes unikke nummer
- Værkfører afslutter med ”opret”

Postconditions:

Råvarebatch er nu oprettet

Alternative flows:

Værkføreren vælger et råvarenummer der er optaget, og får vist en fejl.

Use case: Produktbatch administration

ID: 5.1

Kort beskrivelse:

Eksempel på situation hvor en værkfører opretter en produktbatch

Primære aktører:

Værkfører

Sekundære aktører:

-

Preconditions:

Der er mindst en recept med tilhørende receptkomponenter i systemet

Main flow:

- Værkfører logger ind på web gui
- Værkfører vælger ”opret produktbatch”
- Værkfører vælger nummer på den recept produktbatchen er produceret ud fra.
- Produktbatchen udprintes og uddeles til en udvalgt operatør

Postconditions:

Produktbatch er nu oprettet og har statuskode 0.

Alternative flows:

-

Use case: Afvejning

ID: 6

Kort beskrivelse:

Eksempel på situation hvor en operatør afvejer en produktbatchkomponent

Primære aktører:

Operatør

Sekundære aktører:

-

Preconditions:

- Vægten er tændt og har forbindelse til ASE
- Operatør er logget ind på vægt og udfyldt hvilken produktbatch der skal produceres

Main flow:

- Vejeterminal beder operatør placere en beholder på vejeterminalen
- Beholderens vægt registreres i produktbatchkomponenten
- Vægten tareres og tara-belastningen gemmes
- Vejeterminalen oplyser hvilket råvarenummer der skal afvejes
- Operatør oplyser nummeret på den batch råvaren kommer fra
- Operatør foretager afvejning af vare
- Afvejning ligger inden for tolerance
- Operatør fjerner beholder
- Produktbatchkomponenten gemmes

Postconditions:

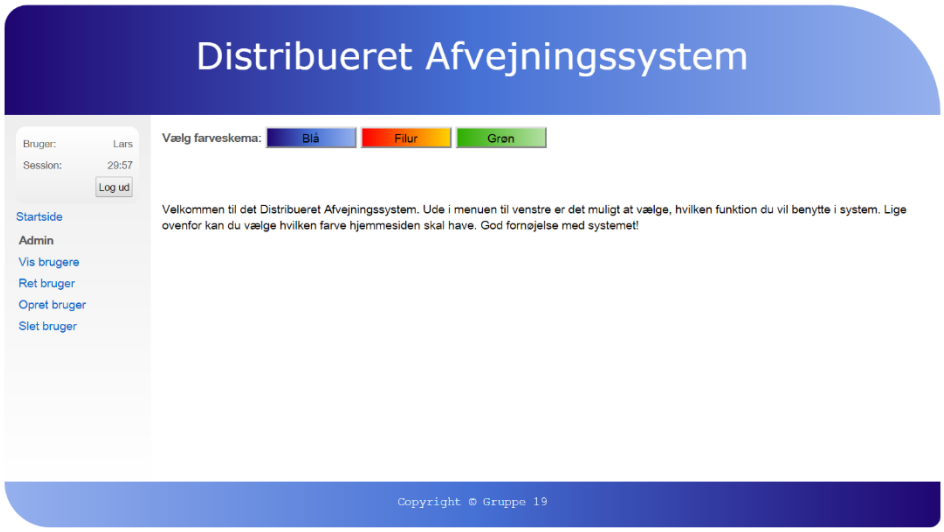
- Produktbatchkomponenten er nu påbegyndt og har statuskode 1
- Operatør afvejer næste produktbatchkomponent

Alternative flows:

-

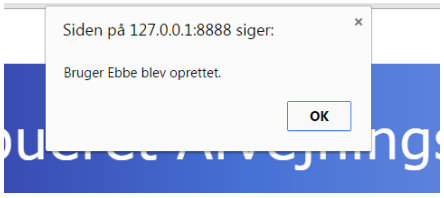
Brugertest

Brugertest 1: Opret bruger



Når der logges ind på hjemmesiden, ses ovenstående figur som det første. Ude til venstre ses menuen for de rettigheder som administratoren har adgang til.

På ovenstående figur ses Opret bruger siden, efter den er valgt ude i menuen. Her kan der oprettes nye brugere i systemet. I denne figur er alle informationer indtaste og dette påvirker "Opret"-knappen, som bliver aktiv når det er muligt at oprette en bruger.



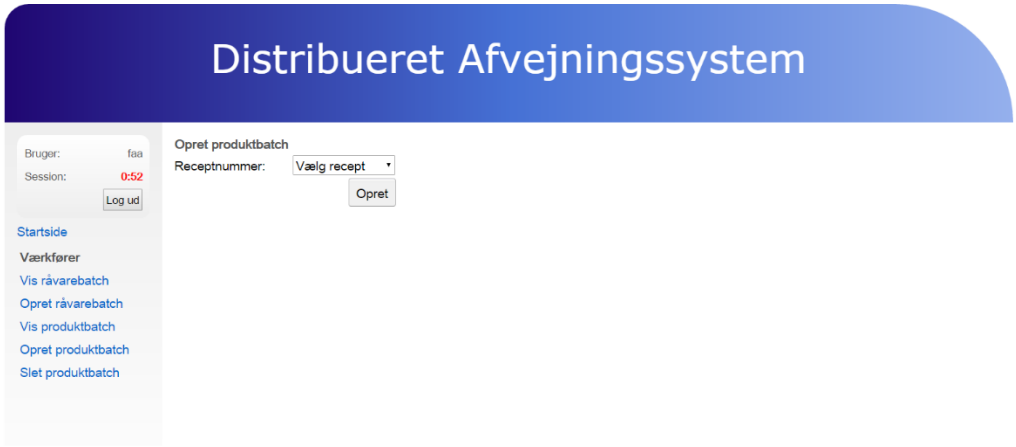
I ovenstående figur ses den popup-meddelelse der sker, når en bruger oprettes korrekt i systemet.

Slet brugere

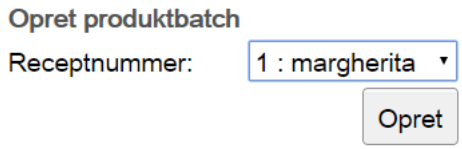
ID	Navn	Initialer	Cpr nr	Password	Admin	Farmaceut	Værkfører	Operatør	
1	Lars	LA	070770-7007	02324it!	☑	☐	☐	☐	Slet
2	Stig	ST	080880-8008	02324it!	☐	☑	☐	☐	Slet
3	Daniel	DA	090990-9009	02324it!	☐	☐	☑	☐	Slet
4	Ronnie	RN	123456-1234	02324it!	☐	☐	☐	☑	Slet
7	Ebbe	EB	123456-1234	1234	☑	☐	☐	☐	Slet

Herefter er det muligt at se den oprettede bruger i ”Vis brugere” menuvalget.

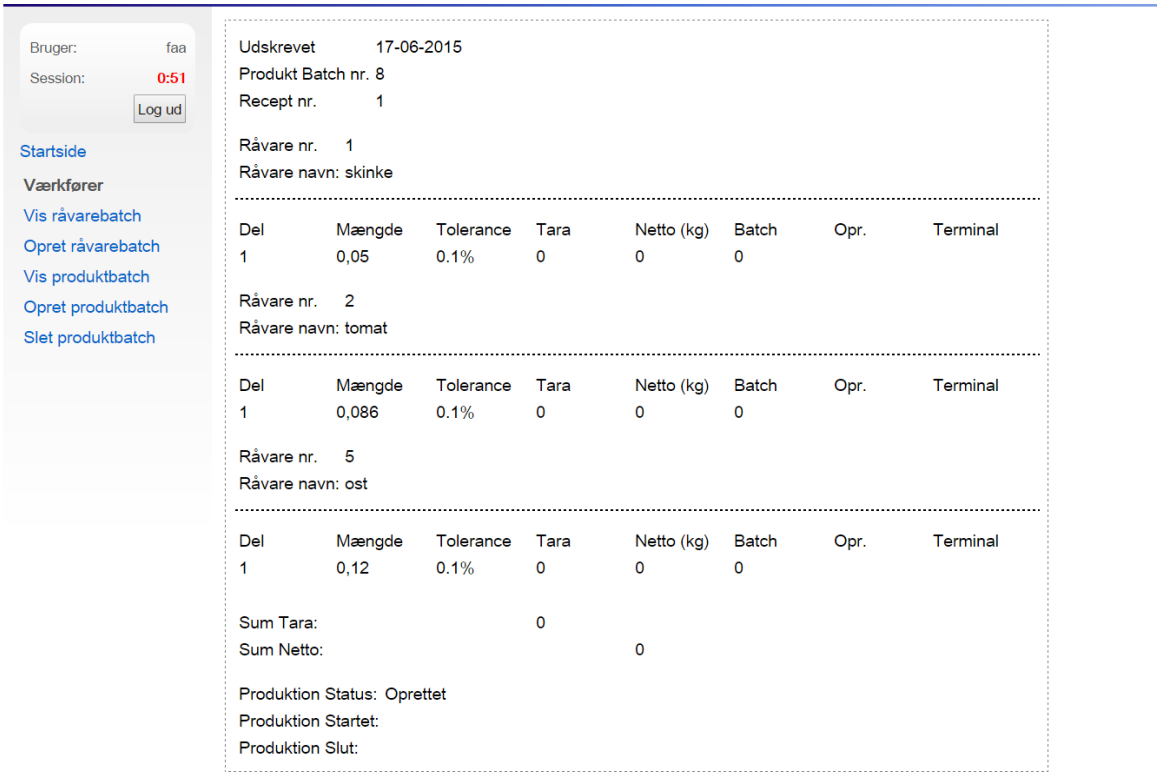
Brugertest 2: Opret produktbatch



Overstående figur viser ”Opret produktbatch”-siden, hvor det er muligt at oprettet en produktbatch.



Ved at vælge i dropdown-menuen kan man vælge en recept, man gerne vil lave en produktbatch ud fra.



Herefter oprettes en udprints-venlig side, med de informationer der skal være tilgængelige til operatøren.

Brugertest 3: Ret råvare

Her ses "Ret råvare"-siden", hvor det er muligt at ændre i de forskellige råvarer der er registreret i systemet.

Ret råvarer

ID	Navn	Leverandør			
1	skinke	Wawelka	Ret		
2	tomat	Knoor	Ok	Cancel	
3	tomat	Veaubais	Ret		
4	tomat	Franz	Ret		
5	ost	Ost og Skinke A/S	Ret		
6	skinke	Ost og Skinke A/S	Ret		
7	champignon	Igloo Frostvarer	Ret		
8	tomat	Heinz	Ret		

Når man trykker på ret, får man muligheden for at ændre i råvaren. Råvaren med råvare ID 2 vælges som forsøg.

Ret råvarer

ID	Navn	Leverandør		
1	skinke	Wawelka	Ret	
2	tomatketchup	Heinz	Ok	Cancel
3	tomat	Veaubais	Ret	
4	tomat	Franz	Ret	
5	ost	Ost og Skinke A/S	Ret	
6	skinke	Ost og Skinke A/S	Ret	
7	champignon	Igloo Frostvarer	Ret	
8	tomat	Heinz	Ret	

Der ændres i informationerne i den valgte råvare og der trykkes herefter ”Ok”.

Systemet fortæller herefter at operationen er gået gået godt og at råvaren nu er ændret.

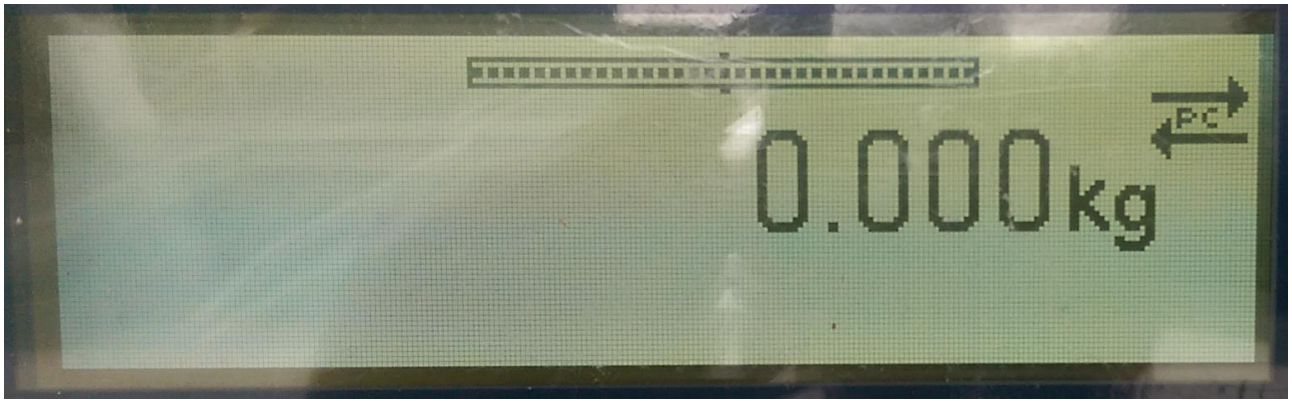
Ret råvarer

ID	Navn	Leverandør	
1	skinke	Wawelka	Ret
2	tomatketchup	Heinz	Ret
3	tomat	Veaubais	Ret
4	tomat	Franz	Ret
5	ost	Ost og Skinke A/S	Ret
6	skinke	Ost og Skinke A/S	Ret
7	champignon	Igloo Frostvarer	Ret
8	tomat	Heinz	Ret

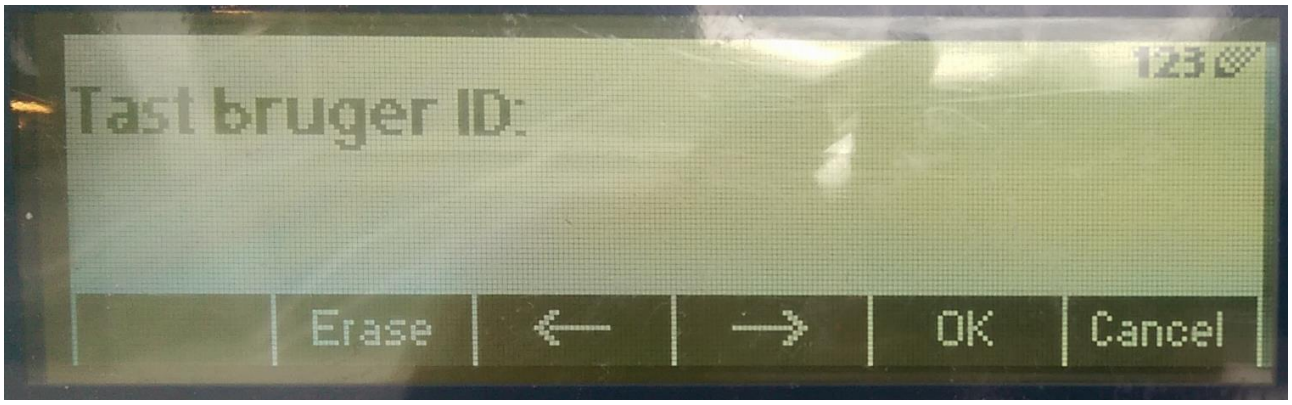
Det kan nu ses at råvaren med råvare ID 2 har fået et nyt navn og leverandør.

ASE afvejning

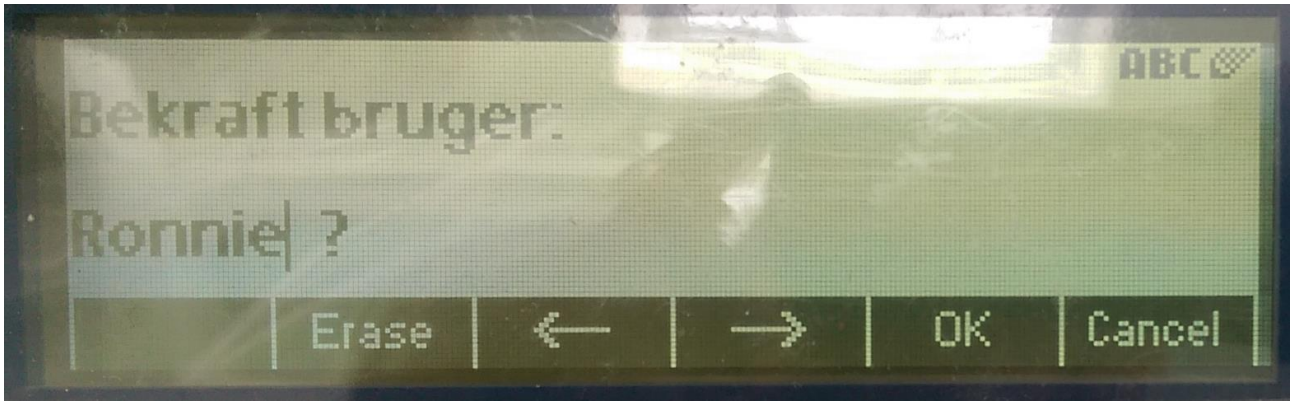
Bilaget dokumenterer en afvejningsprocedure på en fysisk vægt vha. ASE.



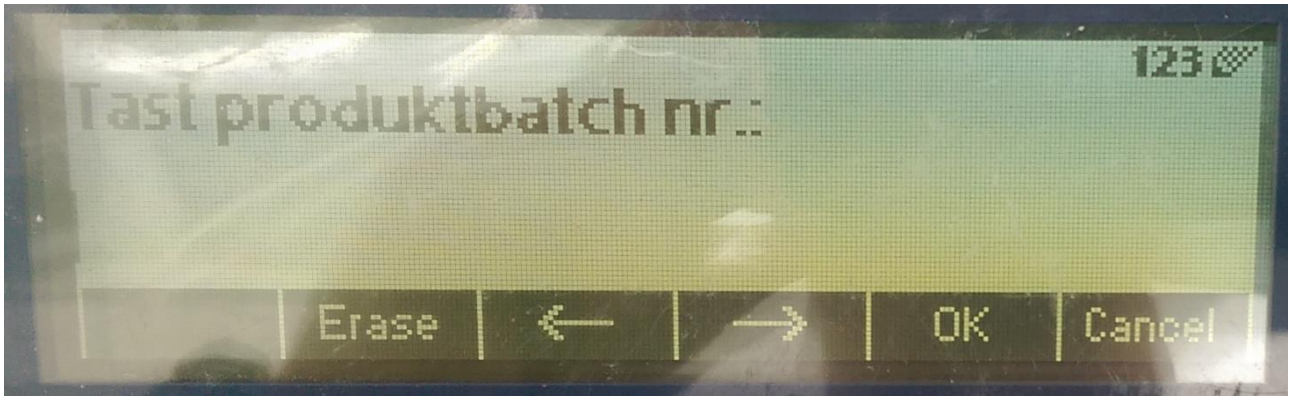
Her ses startmenuen på vægten, inden ASE er tilsluttet.



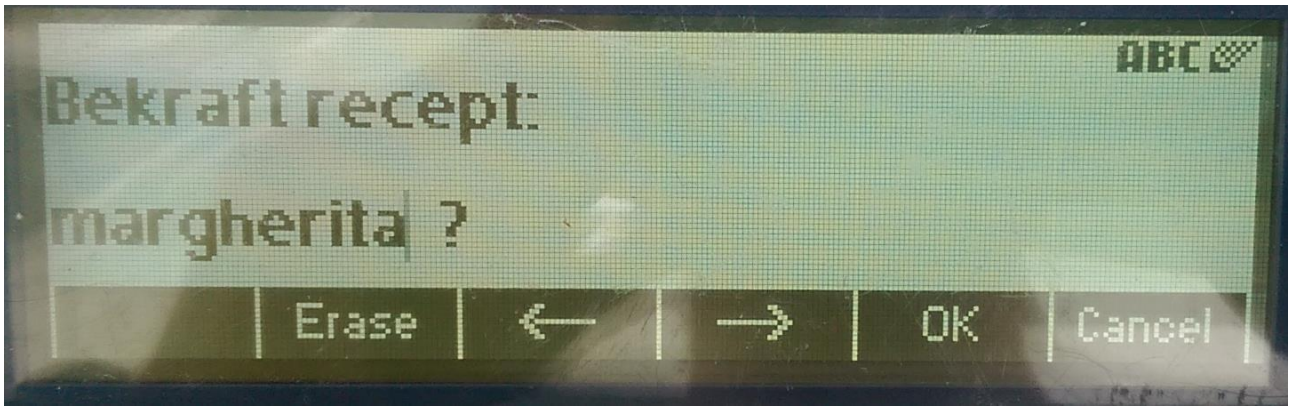
Når en produktbatchs komponenter skal afvejes starter vægten med at spørge efter et bruger ID.



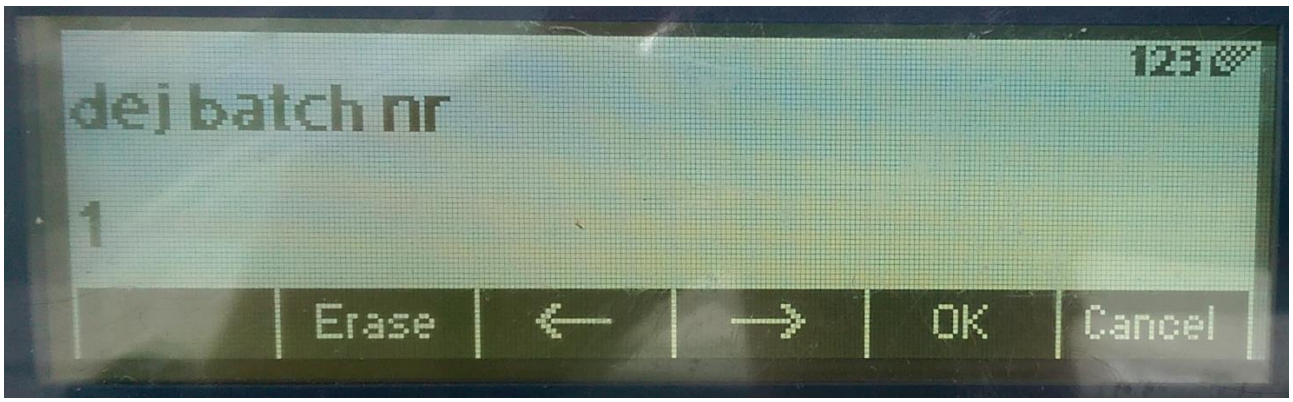
Bruger ID'et indtastes og vægten spørger efter bekræftelse.



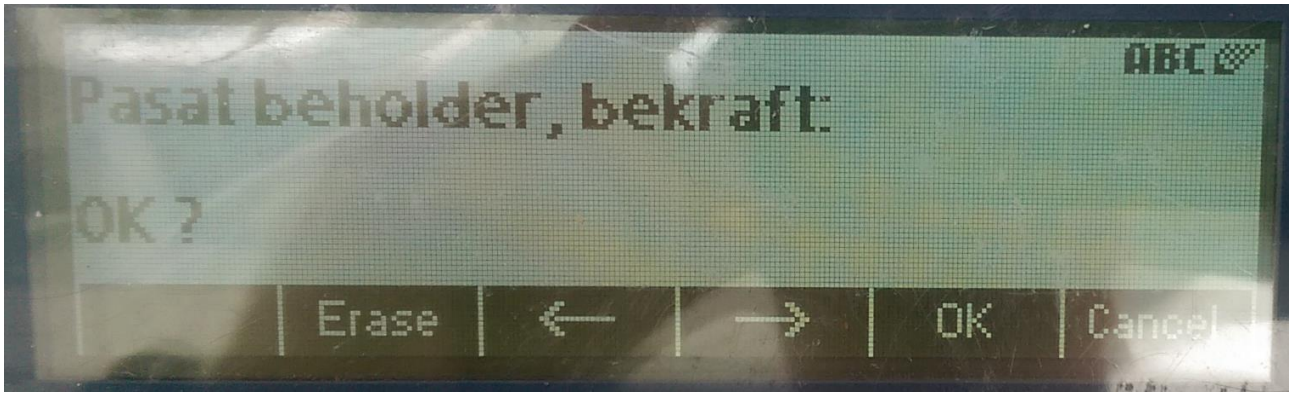
Herefter skal produktbatchens nr indtastes for at sikre at den rigtige komponent bliver afvejet.



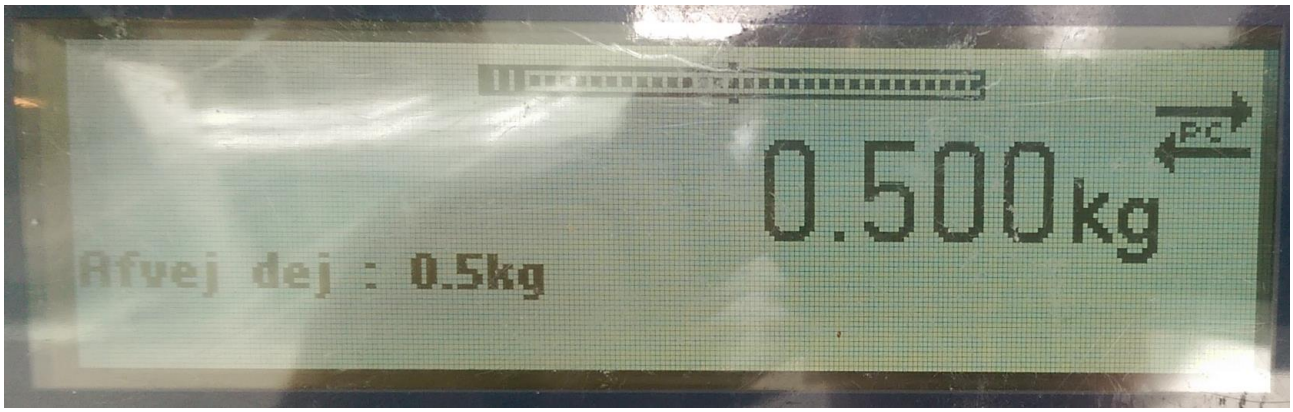
Recepten for produktbatchen skal derefter godkendes ved bekræftelse.



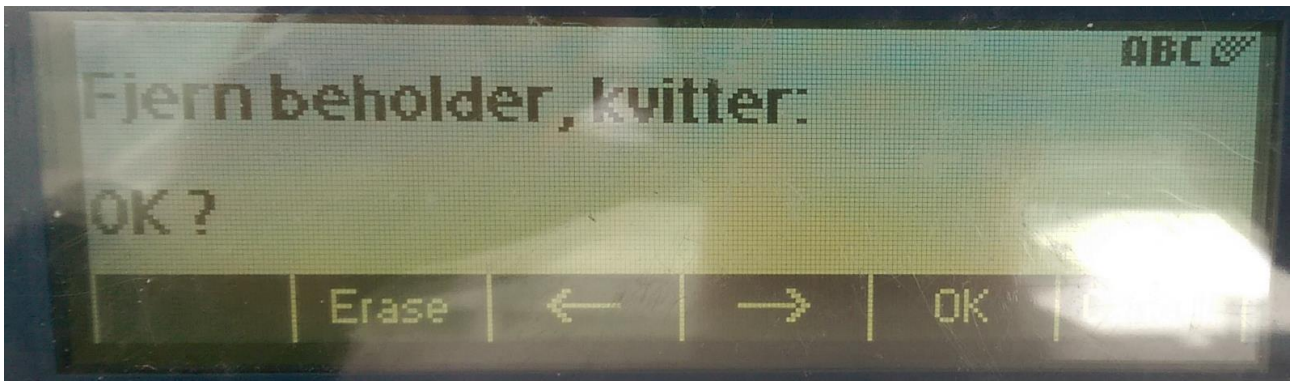
Herefter vises råvarebatchen som afvejes til produktbatchen.



En beholder påsættes og derefter kan vægten tarere.



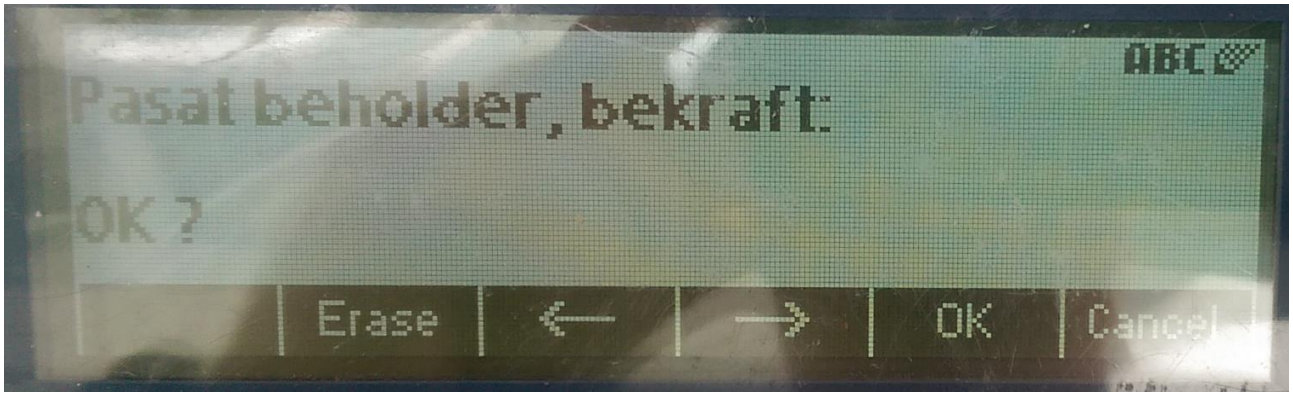
Råvaren afvejes inden for differencen.



Beholderen med råvaren fjernes og næste afvejning kan begynde.

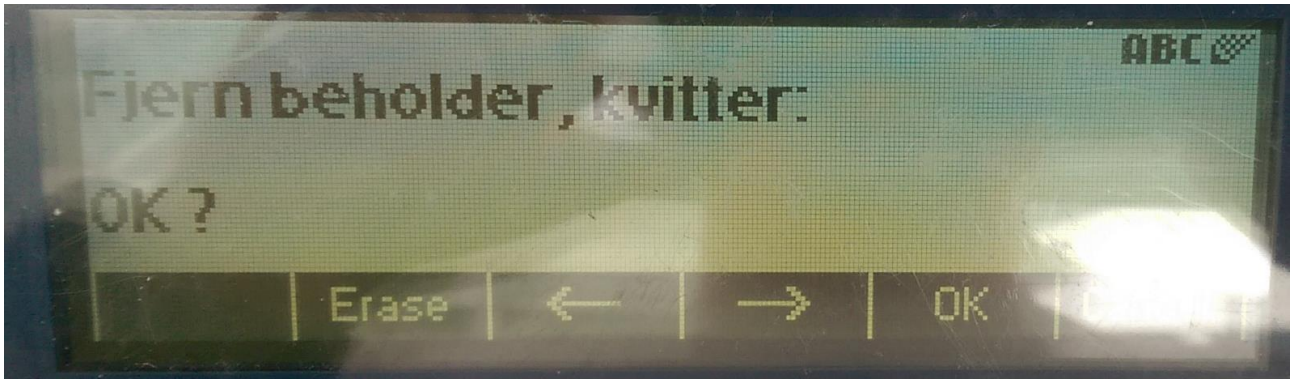
Billede mangler

Herefter vises råvarebatchen som afvejes til produktbatchen.

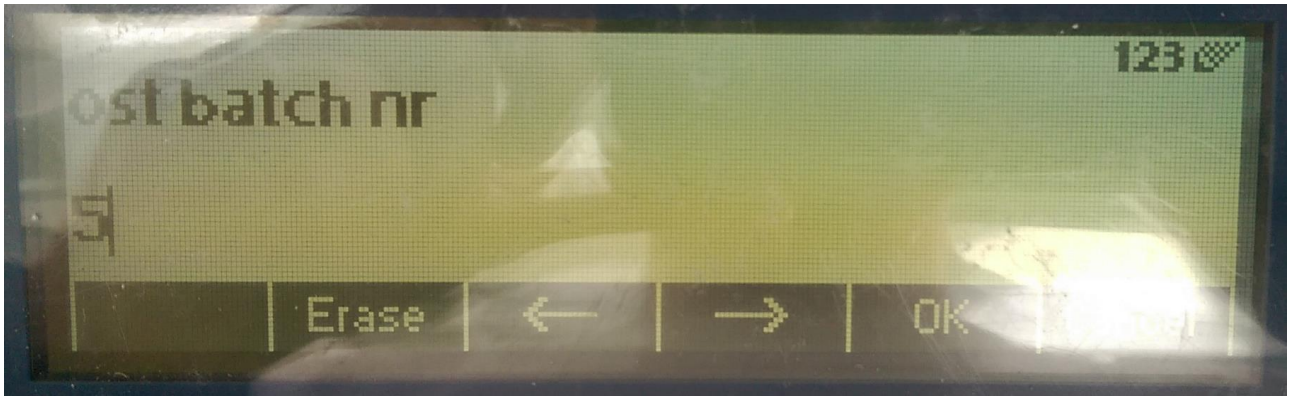


En beholder påsættes og derefter kan vægten tarere.

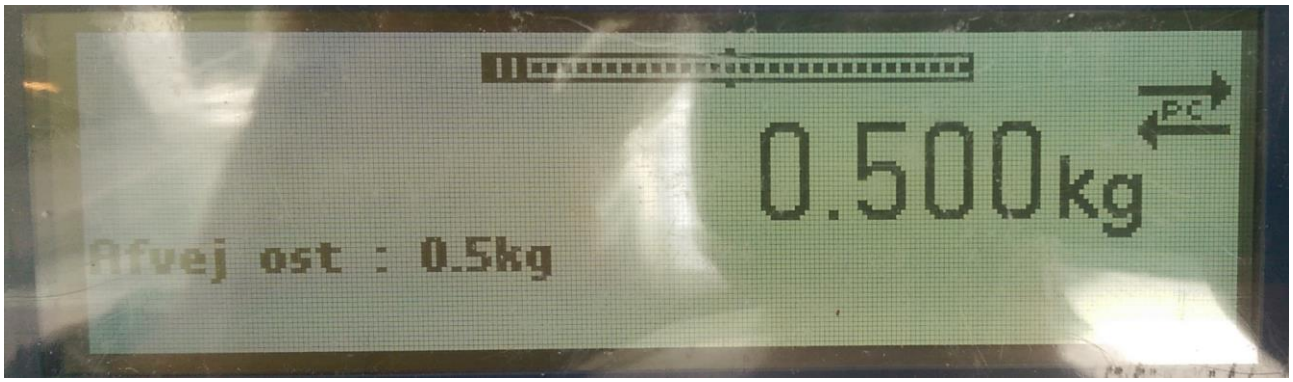
Råvaren afvejes inden for differencen.



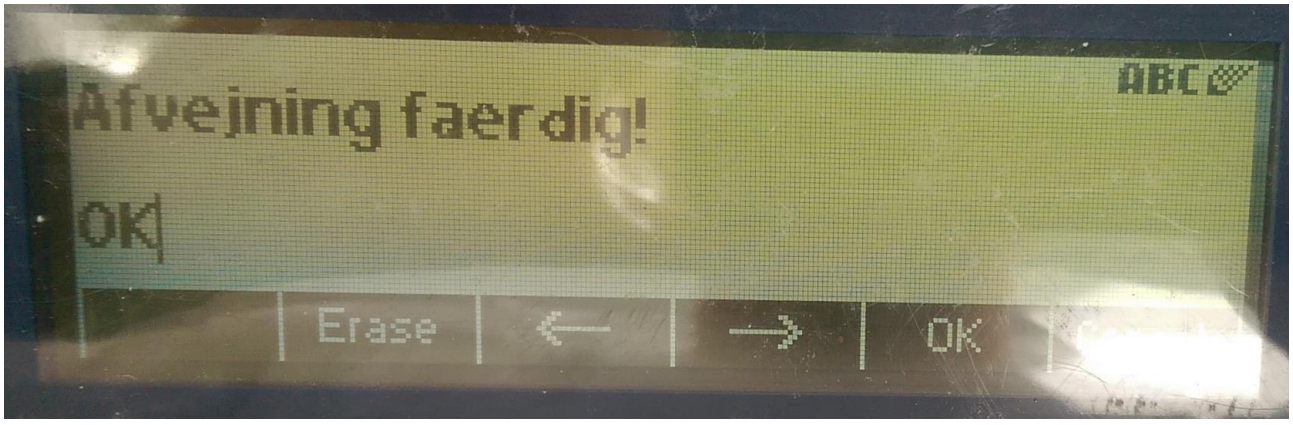
Beholderen med råvaren fjernes og næste afvejning kan begynde.



Herefter vises næste råvarebatch som afvejes til produktbatchen.



Råvaren afvejes inden for differencen.



Når afvejningen for alle produktbatchkomponenter er slut, prompter vægten med en afsluttet status og produktbatchen er nu helt færdig.