

---

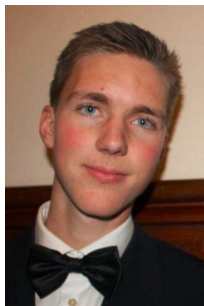
# CDIO Final

## Afvejningssystem Gruppe 15

---



(a) s153475  
Lukas Bek



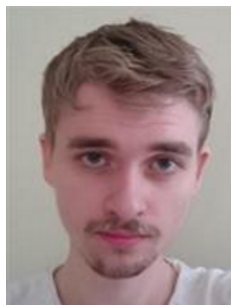
(b) s144850  
Lars Quaade



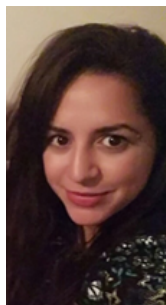
(c) s154306  
Sebastian Hoppe



(d) s153947  
Magnus Haakonson



(e) s153356  
William Wiberg



(f) s092044  
Mariam Saidi



(g) s090830  
Tanja Pajlina

Danmarks Tekniske Universitet  
Softwareteknologi Diplom  
Videregående programmering 02324  
Stig Høgh og Finn Gustafsson  
17. Juni 2016

## Time Regnskab

Navn	Dok.	Design	Impl.	Test	Andet	I alt
<i>Lukas Bek</i>	8	8	35	5	8	64
<i>Sebastian Hoppe</i>	12	9	35	4	6	66
<i>Magnus Haakonsson</i>	16	7	26	6	6	61
<i>Lars Quaade</i>	17	8	27	5	6	63
<i>William</i>	14	12	14	9	10	59
<i>Tanja Paglina</i>	13	6	12	20	9	60
<i>Mariam Saidi</i>	12	7	12	20	9	60
<i>Sum</i>	92	57	161	69	54	440

## Program Information

Herunder er de brugere, som eksisterer per default i databasen:

- **Brugernavn:** 1 - **Kode:** password - **Rettighed:** Administrator
- **Brugernavn:** 2 - **Kode:** password - **Rettighed:** Operatør
- **Brugernavn:** 3 - **Kode:** password - **Rettighed:** Farmaceut
- **Brugernavn:** 4 - **Kode:** password - **Rettighed:** Værksfører

## Kør GWT

For at kunne bruge GWT skal software, drivere og hardware stemme overens med det der er beskrevet i konfigurationsstyring. Når dette er opfyldt, kan projektet importeres til Eclipse som der står beskrevet under afsnittet "Konfiguration". Når projektet er korrekt importeret i Eclipse er programmet næsten klar til at blive kørt. Nu skal projektet findes i Eclipse's "Package Explorer". Efter at have lokaliseret projektet skal der højreklikkes på projektet, hvorefter musemarkøren skal bevæges hen over undermenuen "Run as", i "Run as" undermenuen, skal der klikkes på "5 Web Application (GWT Super Dev Mode)", eller "Web Application (GWT Super Dev Mode)". Denne proces tager et øjeblik. Efter den er færdig vil der komme et link frem i en undersektion kaldet "Development Mode". Her findes også en kort beskrivelse af hvordan dette link kan åbnes. Åben linket. Projektet kompiles nu.

Efter dette er færdigt vil der være en Log ind skærm med brugernavn og adgangskode som tekstfelter, og en overskrift der siger "System Administration". For at logge ind kan der bruges en af flere brugere, men det anbefales at bruge bruger med ID 1. Så der klikkes med musen på "Brugernavn" feltet og der skrives "1" i feltet, dernæst klikkes på "Adgangskode" feltet og der skrives "password" ind i dette. Nu kan der klikkes på "Log ind" knappen. Nu er programmets hovedmenu åben, med alle muligheder i Administrations menuen

åbne. For at finde flere brugere kan der klikkes på "Inspicer en operatør". Her kan der indtastes et andet ID og hvis ID'et findes, vil oplysningerne om denne bruger blive vist i rækkefølgen: Navn - Cpr. nummer - Adgangskode - Initialer - Administrations privilegier.

For at kunne logge ind som en af disse andre brugere, skal brugerens adgangskode og det anvendte bruger ID (Det der blev skrevet ind i tekst feltet) noteres til senere brug. Efter at adgangskoden er noteret, kan der klikkes "< – Tilbage" og dernæst "Log ud".

Nu kan log ind processen starte forfra som beskrevet tidligere, men dog indtastes bruger ID'et der blev noteret, sammen med den adgangskode der blev noteret. Nu kan der så klikkes på "Log ind" knappen, og der er nu logget ind med den bruger der blev noteret. Hvis brugeren der blev noteret kun har operatør privilegier kan der ikke logges ind i denne menu, men der kan prøves med en anden.

## **Kør ASE/Vægtsimulator**

Hvis der ønskes at køre ASE'n og benyttes vægtsimulatoren, skal vægtsimulatoren "Weight" først køres og herefter Main under client (i eclipse projektet). Herefter vil Vægten(simulatoren) nu instruere i hvad der skal ske.

## **System Komponenter**

Systemets komponenter kan ses i detalje, ved at logge ind som beskrevet tidligere med eksempelvis admin. Så kan alle knapper bruges til at se komponenterne der eksistere i databasen. Dette kan eksempelvis gøres ved at klikke på "Råvare" knappen, og derefter klikke på "Se Råvare". Dette vil resultere i at alle råvare bliver vist i en tabel. Dette kan også gøres med recepter, råvarebatch, produktbatch osv. Der kan også fra disse laves ny data i databasen. Eksempelvis hvis der skal tilføjes et helt nyt Råvarebatch, skal

der først tilføjes en ny råvare. Dette sker ved at gå ind i råvare menuen, og så klikke på "Tilføj Råvare". Her kan indtastes et nyt ID der ikke allerede findes, såsom 42, og et tilfældigt navn og leverandør. Til slut trykkes på "Opret", og man sendes nu tilbage til hovedmenuen. Herfra skal der klikkes på "Råvarebatch" og så "Tilføj Råvarebatch" og der indskrives et ID på det råvarebatch der ønskes oprettet. Dernæst ID'et på den råvare der lige er oprettet, altså 42, efterfulgt af en vægt. Der afsluttes ved at trykke på "Opret", hvorefter man returneres tilbage til hovedmenuen. Den ny oprettede råvarebatch kan nu ses ved at klikke på "Råvarebatch", efterfulgt af "Se Råvarebatches".

# Indhold

Program Information . . . . .	3
Kør GWT . . . . .	3
Kør ASE/Vægtsimulator . . . . .	4
System Komponenter . . . . .	4
Indledning . . . . .	9
Kravspecificering . . . . .	10
Funktionelle Krav . . . . .	10
Aktører . . . . .	11
Database specifikation . . . . .	11
ASE specifikation . . . . .	11
Web applikations specifikation . . . . .	12
Non Funktionelle krav . . . . .	13
Konfiguration . . . . .	13
Design . . . . .	16
Use-Case-Diagram . . . . .	17
Use-Case Eksempel . . . . .	17
SSD for ASE . . . . .	20
BCE model . . . . .	21
Klasse Diagrammer . . . . .	22
3-lags-modellen . . . . .	26
GWT . . . . .	26

ASE . . . . .	28
Afvejnings styring enhed . . . . .	29
ASE funktioner . . . . .	30
Vægtsimulatoren . . . . .	31
Databasen . . . . .	34
Data Access Laget (DAL) . . . . .	34
Test af database . . . . .	34
Normalisering . . . . .	35
Transaktionslogik . . . . .	37
WebInterface . . . . .	38
GWT . . . . .	38
Service-klasser . . . . .	38
Service.java . . . . .	38
ServiceAsync.java . . . . .	39
ServiceClientImpl.java . . . . .	39
ServiceImpl.java . . . . .	41
Klient . . . . .	41
Server . . . . .	42
Shared . . . . .	42
Implementering . . . . .	43
Generel implementering . . . . .	43
Test . . . . .	45
Fysiske tests . . . . .	45
Manuel tests . . . . .	45
ASE test . . . . .	48
Problemer . . . . .	48
GWT . . . . .	48
ASE/Vægt . . . . .	50
Viderebygning . . . . .	50

GUI . . . . .	50
Program Logik . . . . .	51
Database . . . . .	51
Konklusion . . . . .	53
Bilag . . . . .	54
Use Cases beskrivelser . . . . .	54
Klasse Diagrammer . . . . .	89
EER Diagram . . . . .	93



## Indledning

Denne rapport har til formål at give et endeligt overblik over afslutningen og sammenfletningen af de tidligere delprojekter i blandt andet faget 02324 Videregående programmering og dermed give et endeligt resultat over, hvad vi er kommet frem til, og hvad vi har udviklet ud fra opgavebeskrivelsen.

Til dette projekt laver vi et system, der skal bruges til afvejning af råvarer, disse skal kunne bruges til produktionen af forskellige recepter i en medicinal-virksomhed. Systemet skal desuden kunne lave dokumentation af de afvejninger, der bliver foretaget, herunder registrere hvilke råvare der bliver benyttet. Systemet skal også benyttes til lagerstyring af produkter og råvarer. Selve systemet skal udvikles, så det er muligt at lave, definere og ændre information for råvarer og recepter. Der skal implementeres leverandører til de forskellige råvarer, så disse kan hjemtages til firmaets lager, i såkaldte råvarebatches, hvor man her kan se mængden af den pågældende råvare. Selve recepterne skal være en liste over de råvarer, som skal benyttes, og de rigtige doseringer for at kunne udvikle/realisere et produkt. Recepterne skal kunne benyttes til et vilkårligt antal produktionsbatches, som er selve de konkrete produkter, der skal udvikles. I disse produktionsbatches skal man kunne se information omkring hvilken recept, der bliver benyttet, samt hvordan disse er oprettede ud fra hvilke recepter. Det skal også være muligt i systemet at holde styr på mængden i lagerbeholdningen ifm. råvarerne der bliver brugt i produktionen.

Måden vi har tænkt os, at etablere dette system på, er ved at oprette en database som kan lagre, opretholde og gemme data omkring: råvarer, recepter, leverandører og aktør-typer. Aktør-typer er brugere, der benytter systemet eller opretholder, opdaterer råvarer og lagerbeholdningen. Denne database skal desuden udvikles sådan, at det vil være nemt at tilgå data

og information omkring råvarer, råvarebatches, recepter, produktbatches, lagerbeholdningen (mængden) og leverandørerne som leverer råvarerne. For at medarbejderne i virksomheden har en mulighed for at benytte og se informationerne og data i databasen, bliver vi desuden nødt til at udvikle et dynamisk web-interface, hvilket vi gør med Google Web Toolkit. Vi skal herudover udvikle en vægt simulator, som har de samme funktioner som den fysiske vægt, som medarbejderne har til rådighed på virksomheden. Dette gøres i Java, hvor vi skal have vægten til at kommunikere med vores program gennem en socket-forbindelse. For at få alle disse instanser til at kommunikere og fungere sammen, vil vi udvikle de nødvendige klasser i java, som får hele systemet til at lave et samspil.

Vores gruppe har mødtes hver dag i lokale 009 hvorefter vi sammen har udarbejdet dette projekt. Vi har fordelt opgaverne således at alle altid har haft noget at lave. Vi sørgede for at snakke sammen om hvilke klasser vi arbejdede i, så der ikke opstod nogle konflikter mellem klasserne når der skulle pushes til vores Git repository. Vi har prøvet at inkludere alle i hver del af projektet, så det fx ikke var den samme person der stod for alt med GWT. Det havde til formål at alle ville danne sig viden inden for alle emner, så det vil være bedre for os når vi skal forberede os til eksamen.

## Kravspecificering

### Funktionelle krav:

- Database med oplysninger om operatør, råvare, råvarebatches, recepter og produktbatches.
- Programmet skal have 4 aktører med forskellige rettigheder: Administratorer, Farmaceut, Værkfører, Operatør
- Programmet skal have et webinterface hvor aktørerne kan udføre deres

respektive roller

- Der skal bruges en enten fysisk vægt eller en simulator af en vægt
- Der skal laves et program (ASE), der styrer alle vejeterminalerne og gemmer data i databasen.
- ...

### **Aktører**

- Administrator: Er superbruger, som har alle funktionaliteter og kan oprette brugere i systemet
- Farmaceut: Administrerer råvarer og recepter. Har derudover også samme rettigheder som værkfører.
- Værkfører: Administrerer råvarebatches og produktbatches. Har derudover også operatørrettigheder
- Operatør: Administrerer afvejning

### **Database specifikation**

- Skal have data access lag, som mellemlid mellem database og system arkitekturen
- Fejl i data access lag skal give en exception, som effektivt beskriver fejlen.
- Skal som minimum være på 3. normalform

### **ASE (Afvejnings Styring Enhed) specifikation**

- Tilgå vægten gennem TCP

- Kommunikation skal ske via RM20-Kommandoen
- Det skal være muligt at afbryde afvejnings-proceduren
- Afvejningen skal følge afvejnings-proceduren som står beskrevet i bilag 4.

### **Web applikations specifikation**

- Web Applikationen skal implementeres med Google Web Toolkit
- Webinterfacet skal implementeres med en underliggende MySQL database
- Webinterfacet skal indeholde et login system, hvor MySQL databasen opretholder alle informationer om de oprettede brugere
- Man skal kunne oprette, redigere og vise brugere gennem web interfacet
- Ved oprettelse af brugere angives et bruger ID (entydigt), navn, initialer, password samt brugernes rolle
- En bruger der én gang er oprettet, kan ikke slettes
- Passwords skal ændres / oprettes i overensstemmelse med DTU's password regler
- Alle input felter skal valideres iht. gyldige områder og der skal vises passende fejlmeddelelser ved fejl
- Der skal foretages autentikation af brugerne i en login session
- Der skal vises sider med menuer baseret på brugerens rettigheder

## Non Funktionelle krav:

- Opbygges efter 3-lags modellen
- Kodes i Java
- UML benyttes til at beskrive programmet
- – – – > Java Version 8 update 73 < – – –
- Skal fungere i et Eclipse 4.4 (Luna) miljø
- GWT versionen som skal anvendes er GWT SDK v. 2.6.1

## Konfiguration

Udviklings- og produktionsplatformen brugt til at udvikle dette projekt er næsten ens. Den første liste kan beskrives som en samlet platform og den efterfølgende er kun en del af udviklingsplatformen. De inkluderer følgende:

- OS: Windows 10
- Java Version 8 update 91
- Eclipse Version: Mars 2 Release (4.5.2)
- GWT plugin SDK 2.7
- JDBC Driveren - mysql-connector-java-5.1.38-bin.jar

**Konfiguration for GWT** For at kunne benytte GWT er det en forudsætning at man allerede har følgende installeret:

- JDK 1.6 eller nyere opdatering
- Java

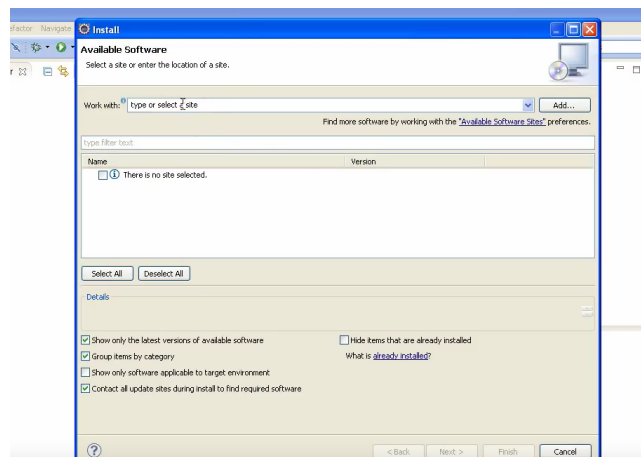
- Eclipse

Når disse installeringer er på plads, skal man så installere GWT SDK og plugin for Eclipse. Dette kan man gøre på Google Web Toolkit's hjemmeside <http://www.gwtproject.org/download.html>. Her skal man downloade en samlet pakke som indeholder:

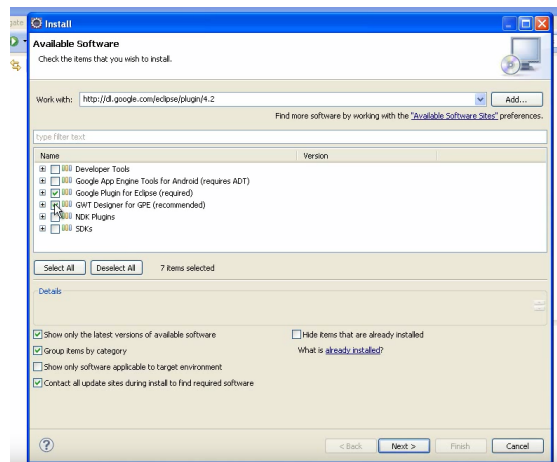
- GWT SDK 2.4 (eller højere)
- Plugin for Eclipse

Når disse plugin for Eclipse og GWT SDK er installeret, skal man åbne Eclipse og installere vores plugin. Dette gør vi ved følgende:

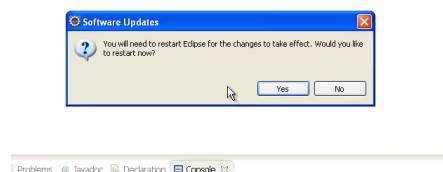
- Gå ind på GWT's hjemmeside og find URL for plugins til den benyttede Eclipse version
- Klik på "help" oppe i menubaren og find "Install new Software"
- I feltet "work with" kopierer man URL'en ind og trykker 'enter'



- vælg herefter "Google plugin for Eclipse", "GWT Designer for GPE" og "SDK's"



- Tryk næste, accepter betingelserne og installer softwaren.
- Herefter vil programmet bede om accept til at genstarte Eclipse for at kunne benytte sig af den nye software. Dette accepteres og Eclipse åbnes igen



- Herefter vil man kunne importere et GWT projekt eller oprette et nyt

## Udviklingsplatform eksklusivt

- MySQL Workbench 6.3

Produktet kan ses på nedenstående link:

[https://github.com/SoftwareCDIO/15\\_Final\\_GWT.git](https://github.com/SoftwareCDIO/15_Final_GWT.git) ASE'n, GWT og vægtsimulator er alle versioneret i tre forskellige repositories:

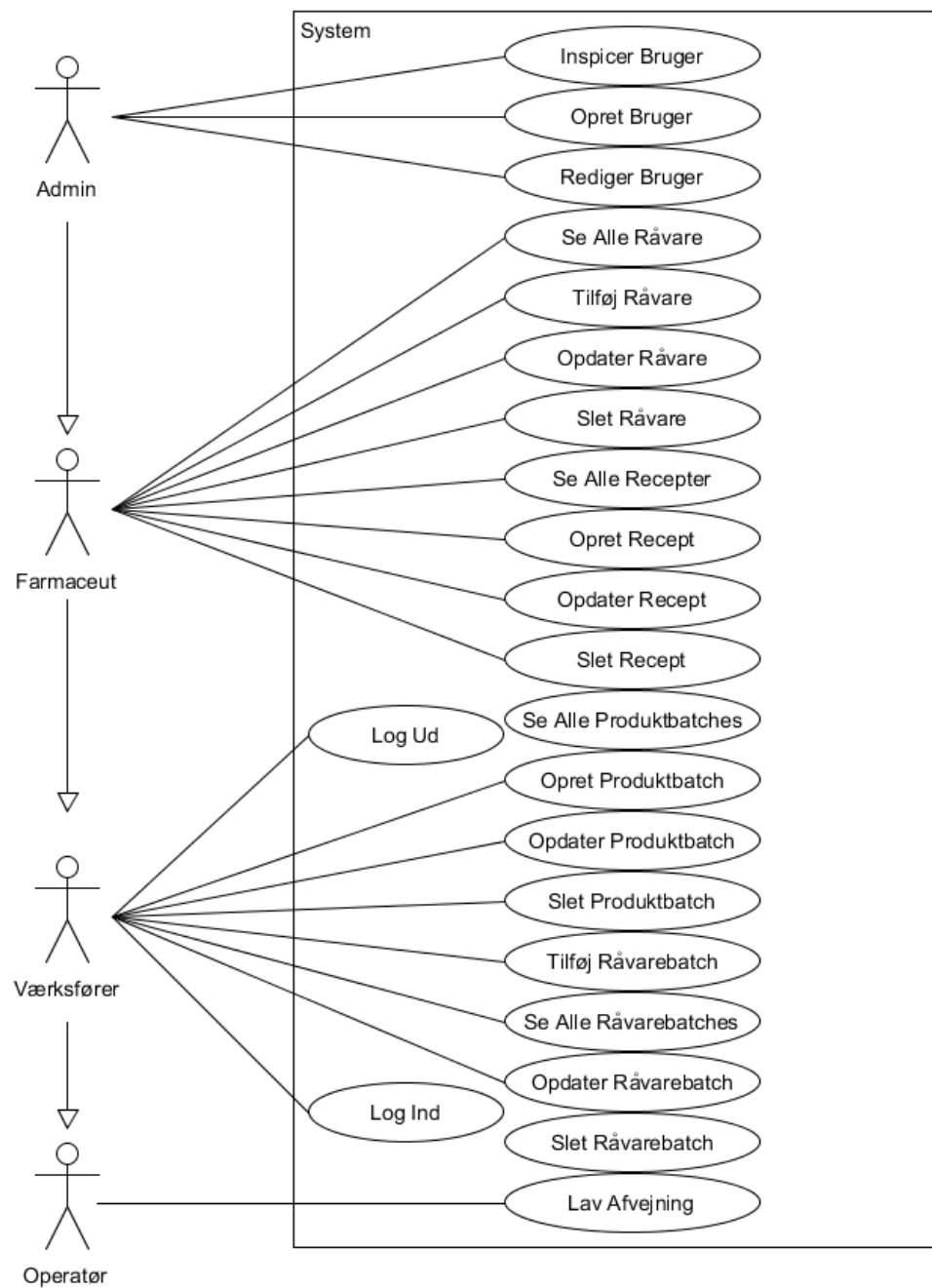
GWT: [https://github.com/SoftwareCDIO/15\\_Final\\_GWT](https://github.com/SoftwareCDIO/15_Final_GWT)

ASE: <https://github.com/LukasBek/CDIOFinalASE>

Vægtsimulator: [https://github.com/LukasBek/15\\_CDIO\\_D2](https://github.com/LukasBek/15_CDIO_D2)

# Design

## Use-Case-Diagram





Ovenfor ses et Use Case Diagram. Diagrammet viser de 4 aktører (Admin, Farmaceut, Værksfører og Operatør) og de use cases de udfører, indkapslet af programmet. Aktørerne arver use cases fra de aktører der er under dem, ud over deres egne use cases, der er markeret med linjer, der går over til deres use cases. Værksføreren kan også det som Operatøren kan, Farmaceuten det samme som værksføreren og Admin det samme som Farmaceuten. Arvingen fortsætter ned igennem systemet, så når Admin arver use cases fra Farmaceuten, inkluderer det også alt hvad Farmaceuten arvede fra Operatøren og Værksføreren. Disse arvinger er vist i diagrammet, i form af pile, der peger ned på den aktør, der arver use cases fra.

De forskellige use cases kan ses i detaljer i bilaget.

## Use-Case Eksempel

Et eksempel på en use case, som ses forinden, er "Inspicer Bruger". Her er kun Admin den primære aktør, da det kun er ham der adgang til denne use case. Det primære flow viser, hvordan brugeren bliver vist og det alternative flow viser en fejlmeddelelse, hvis ID'et ikke findes i systemet.

### Inspicer Bruger

**ID: 07**

**Kort Beskrivelse:** Admin inspicerer en bruger

**Primær Aktør:** Admin

**Sekundære Aktører:** Database

**Forudsættelser:** Admin har bruger ID på brugeren, og brugeren er oprettet i databasen.

#### Primært flow:

1. Admin logger ind

2. Admin klikker på “Inspicer en operatør” knappen
3. Admin indtaster bruger ID for den operatør der skal inspiceres
4. Admin klikker på “Inspicér”
5. Brugerens brugeroplysninger for brugeren med det indtastede bruger ID vises på skærmen
6. Admin afslutter ved at klikke “Tilbage” og dernæst “Log ud”

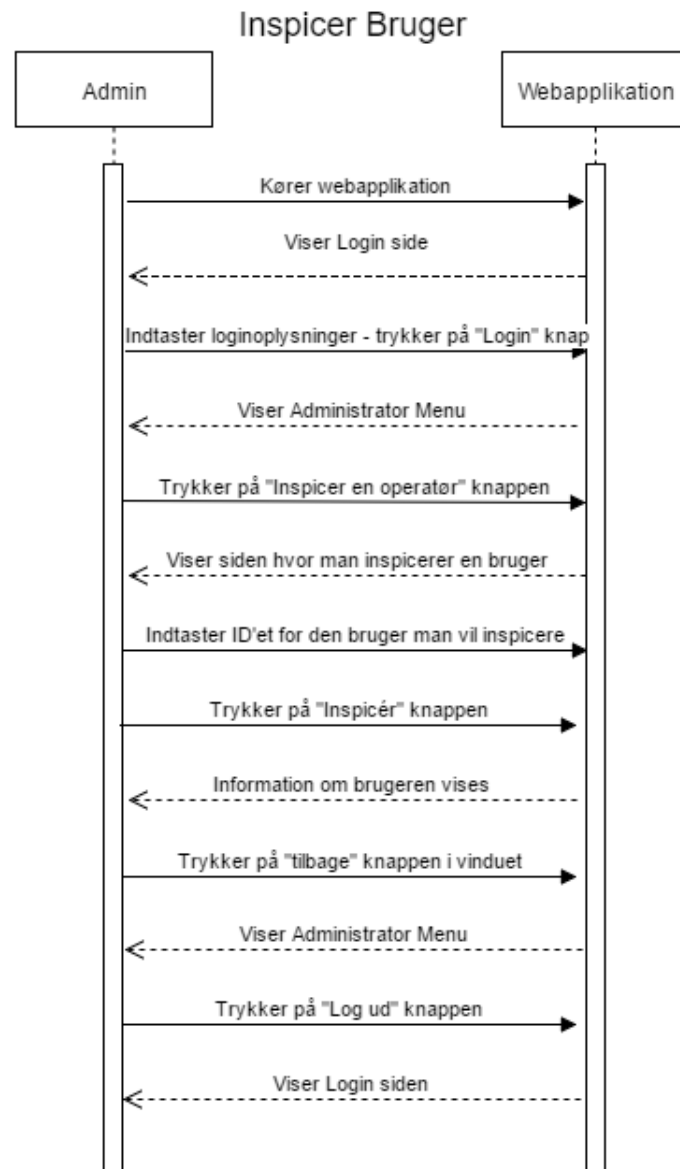
**Efterfølge:** Information om en bruger er blevet vist og Admin er nu logget ud.

#### **Alternative Flows:**

1. Admin logger ind
2. Admin klikker på “Inspicer en operatør” knappen
3. Admin indtaster et bruger ID der ikke eksistere i databasen
4. Admin klikker på “Inspicér”
5. Fejlmeddelelsen: Ingen operatør med ID: >indtastet bruger ID< blev fundet. Indtast venligst et gyldigt ID.

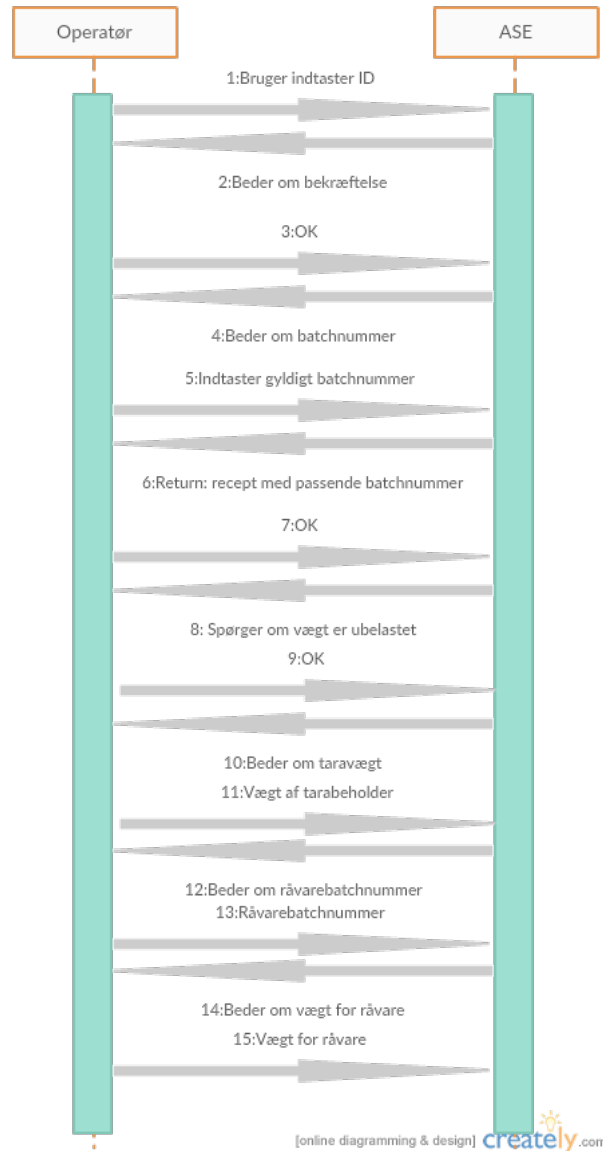
**Efterfølge:** Brugeren kan vælge at indtaste et nyt ID eller at navigere ud af systemet.

## SSD for GWT



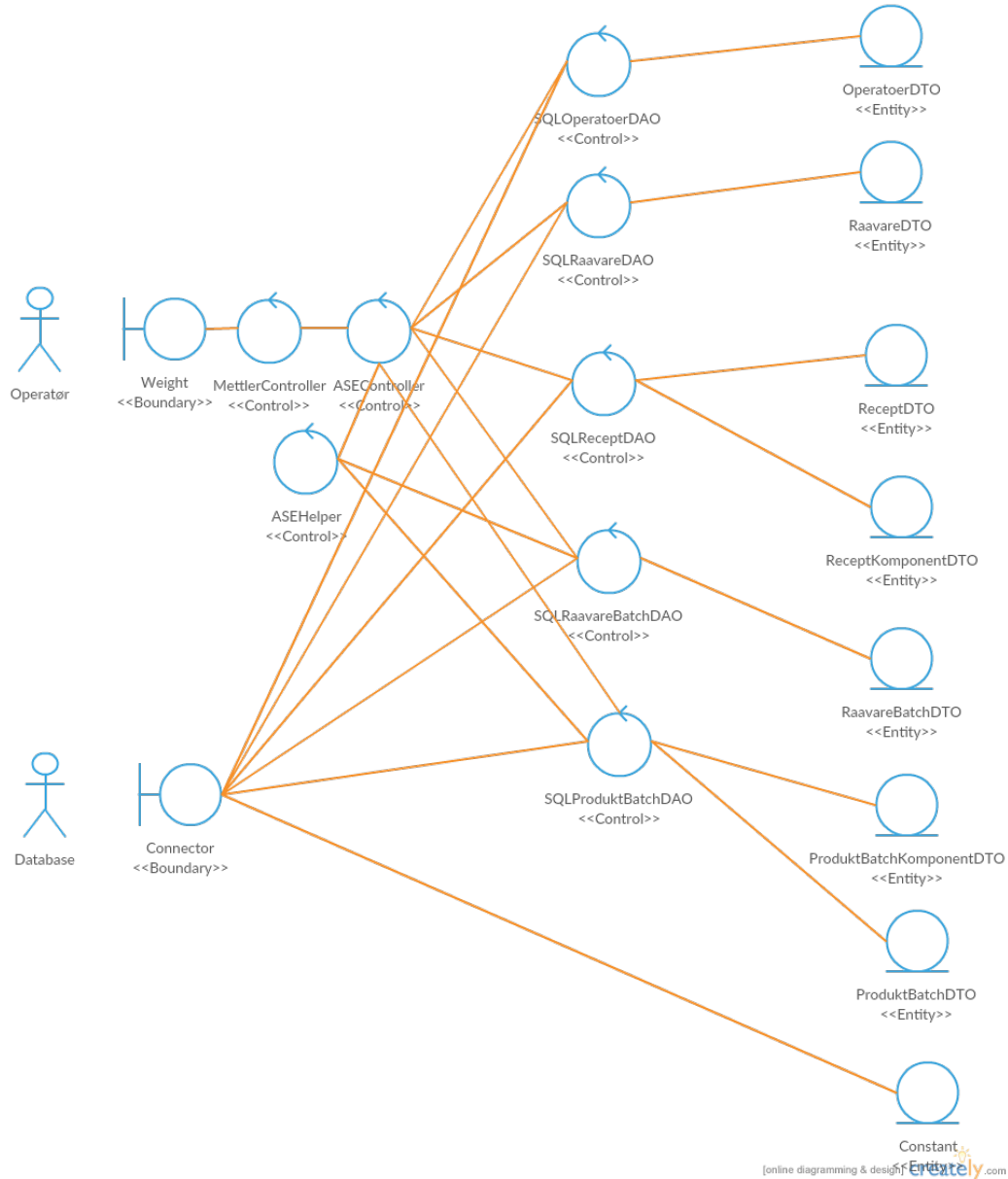
Ovenfor ses et System Sekvens Diagram over use casen "Inspicer Bruger". Det bruges for at danne et overblik over, hvilken udveksling der sker mellem brugeren (administratoren i dette tilfælde) og webapplikationen, for at se et respons fra applikationen, ud fra brugerens handlinger fra use casen.

## SSD for vægtsimulator/ASE



Dette SSD, beskriver en sekvens i vægtsimulatoren, hvor en bruger logger sig på vægten og foretager en vejning. Den beskriver step-by-step, hvordan ASE kommunikerer med brugeren, som sidder med vægten. SSD'en ender, når en bruger er færdig med at oprette en vejning.

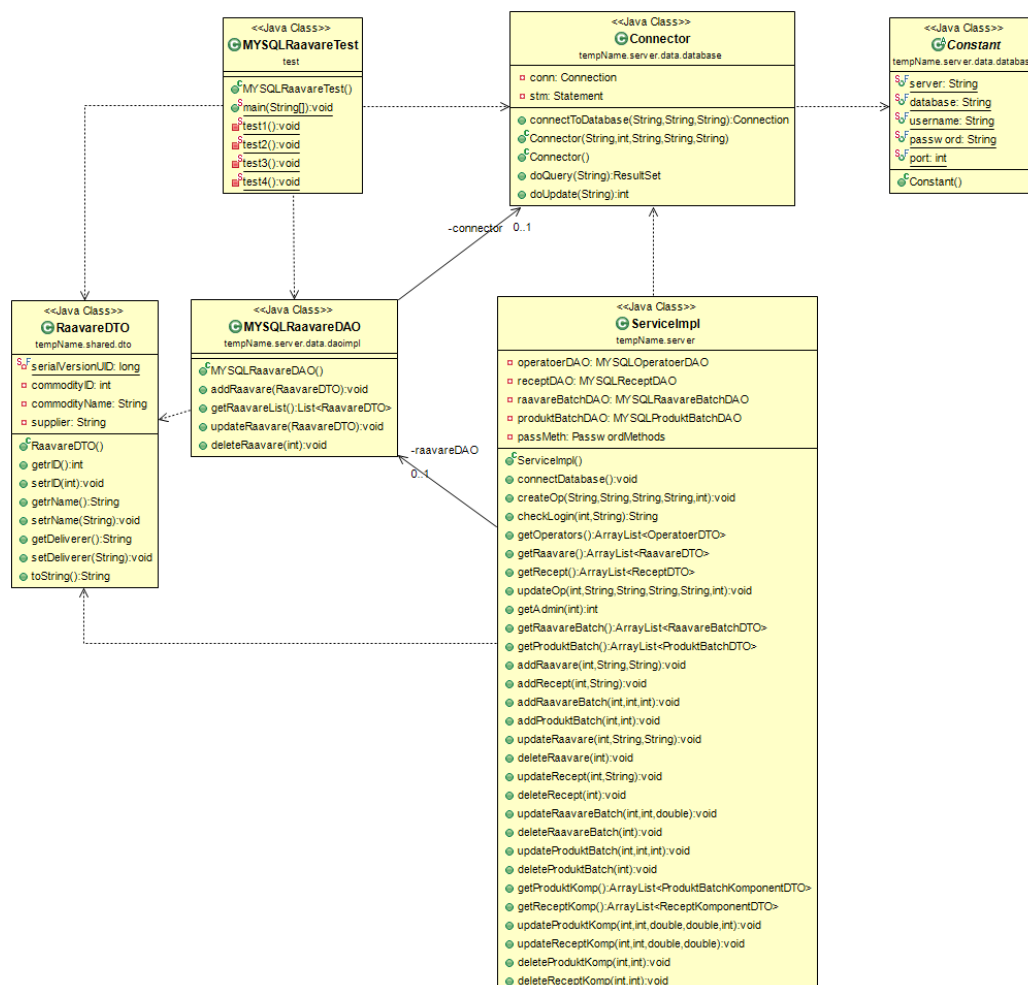
## BCE model



Ovenover ses en BCE-model af vores ASE, samt vægten. Den viser at selve vægten er boundary for operatøren, hvilket vil sige, at det er den grænseflade brugeren får vist. Bagefter er der tre controllere, der håndterer inputs fra vægten, fortolker dem og sender dem videre til DAO-klasserne, som også er controllers. Disse controllers har adgang til DTO'erne, som entities. Samtidig

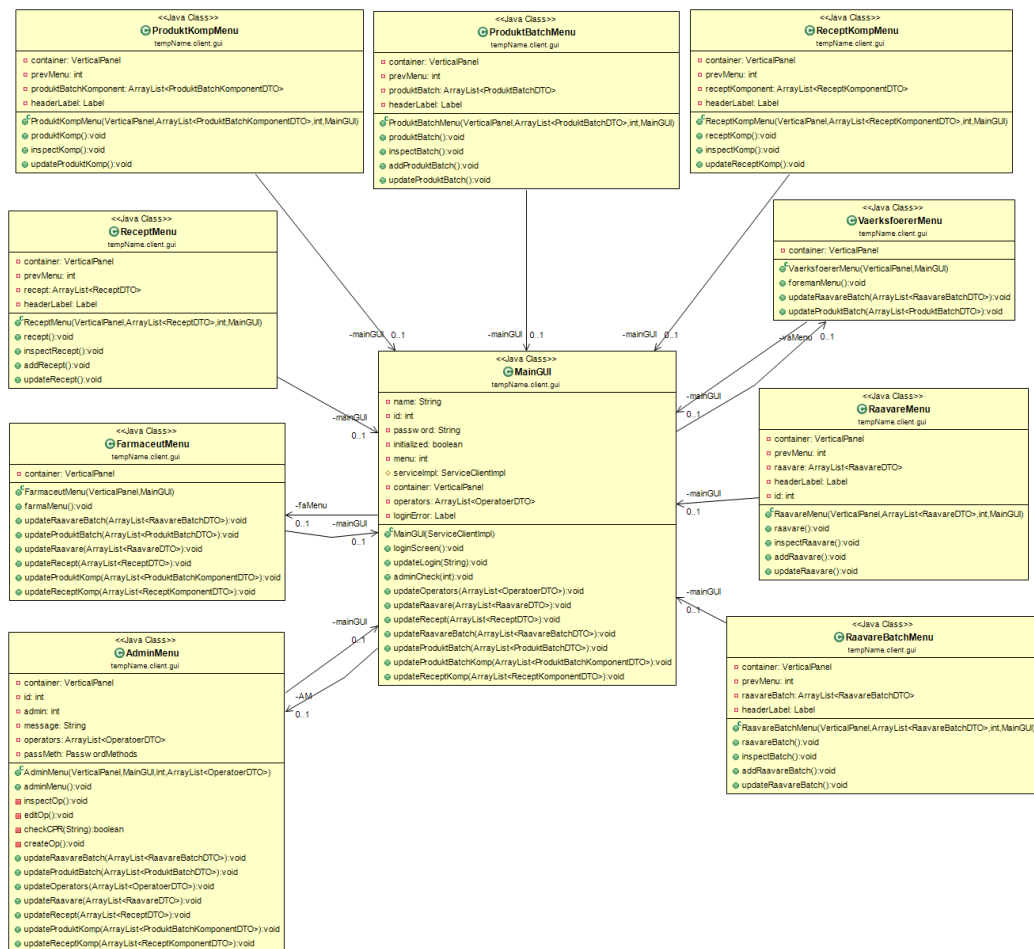
er de også forbundet til connectoren, som fungerer som boundary for databasen, som er en sekundær aktør. Connectoren er også forbundet til Constant, som indeholder data om, hvordan man tilslutter til databasen.

## Klasse Diagrammer



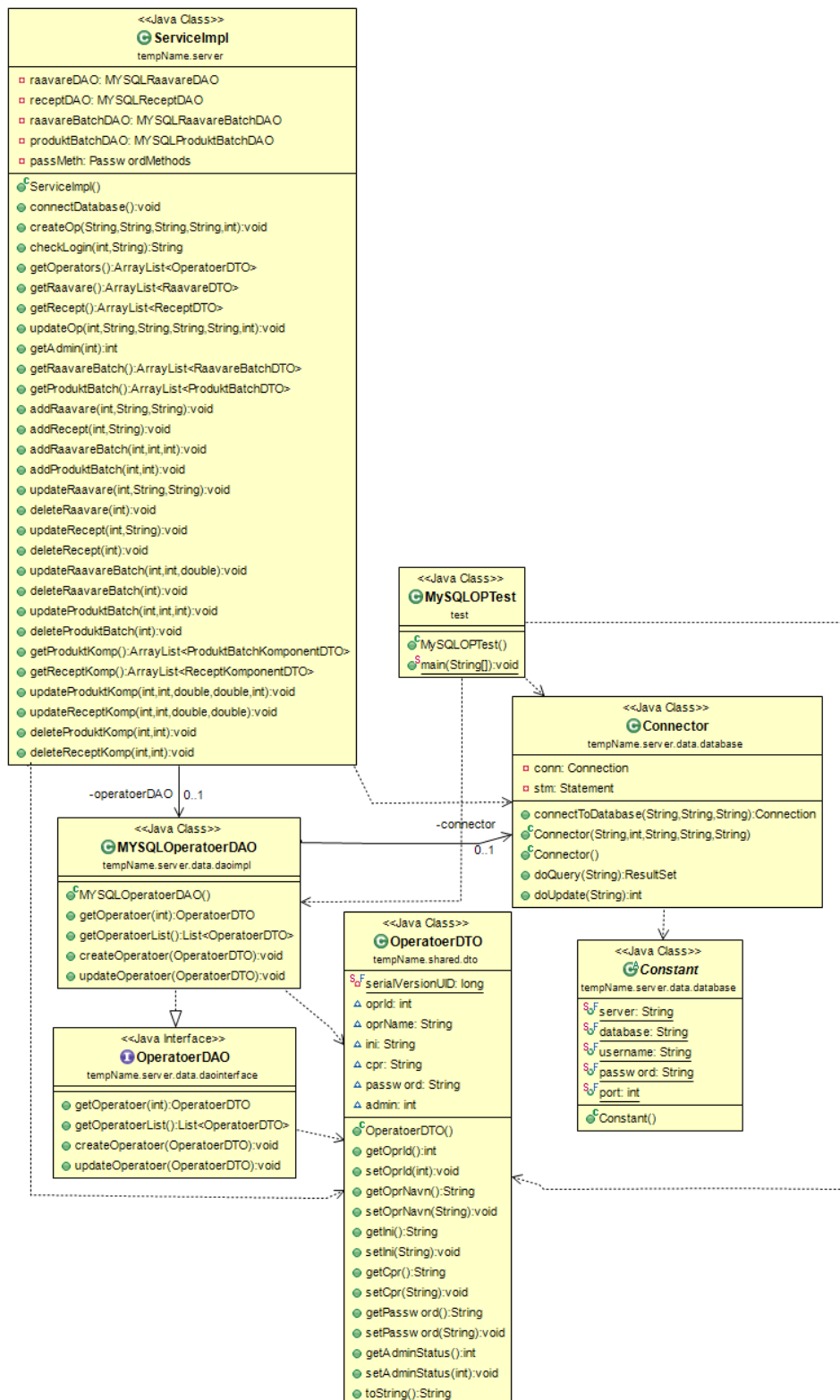
Dette klasse diagram giver et detaljeret overblik over klasserne RaavareDTO, MySQLRaavareTest, MySQLRaavareDAO, Connector, Constant og ServiceImpl. Klassen MySQLRaavareDAO kender til Connector klassen, det ses ved at der går en envejs associering til Connector klassen, denne associering indeholder et rolle navn og en multiplicitets værdi for den kendte

[illegible]



Ovenover ses et klassediagram over opstillingen af GWT's GUI-klasser. Der ses tydeligt på dette diagram, at alle GUI-klasser har en instans af MainGUI. Det har de, fordi de skal bruge den, til at lave kald til serveren. Det ses også hurtigt, at MainGUI kun har instanser af Admin, Farmaceut og Værksfører-menuerne, men ikke af nogle af de andre. Dette er grundet, at de sidste menuer, bliver skabt i de tre operatørmenuer, hvilket betyder, at det eneste de klasser bruger MainGUI til, er at lave servicekald.





Ovenover ses et klassediagram over relationerne, mellem de relevante klasser, på serversiden af GWT-projektet, til at manipulere med operatørdata, samt test-klassen, som også benytter sig af operatør-klasserne. Den viser fint, hvordan ServiceImpl klassen benytter DAO'en, som bruger connectoren til at få forbindelse til databasen. Dette beskriver også, hvordan tingene fungerer for de andre DAO-klasser på serveren, da disse fungerer på helt samme måde, som denne.

ASE klasse diagrammet kan ses her:<sup>1</sup> Det kan ses at ASEControlleren skaber et objekt af både ASEHelper og MettlerController. Det ses også at Main er afhængig af ASEControlleren. Det kan til dette diagram knyttes at MettlerControlleren har en forbindelse til den Vægtsimulator der ligger i et helt andet projekt, men dette er en internetforbindelse, som det kan ses ved at det bla. har clientSocket af typen Socket. Derfor hører vægtsimulatoren ikke til i dette diagram.

## 3-lags-modellen

### GWT

#### 1. lag - Grænsefladen

1. lag i trelagsmodellen er selve grænselaget. Det som brugeren bruger og ser. I dette projekt er det selve HTML koden der er skrevet, da denne bestemmer hvordan selve siden ser ud og hvordan man interagerer med selve koden. Dertil kommer også GWT (Google Web Toolkit), da denne sammen med HTML giver de knapper og tekstfelter som der kommer frem på Websiden. Ydermere er der også CSS (Cascading Style Sheets), der giver udseendet af elementerne på Web siden, knapper, tabeller, tekst og baggrunde.

---

<sup>1</sup>Billag side 89

## **2. lag - Funktionaliteten**

Ligesom at GWT i det første lag stod for at skabe de forskellige widgets (knapper, tekstbokse og tabeller), er GWT også en del af 2. lag, men her gør det noget lidt andet. Her står den for at give knapper og tekstbokse funktion, altså at man kan klikke på den eller skrive noget i dem. Sammen med Java kode og GWT kommer der funktion ind i Websiden. GWT giver funktionalitet til knapper og felter og Java koden bestemmer hvad der skal ske, når der trykkes på en knap, eller der skrives i et felt. Java koden er så ansvarlig for at de rette ting sker når en knap trykkes. Eksempelvis at data skal hentes fra databasen og derefter tjekkes med noget der er skrevet ind i et felt.

## **3. lag - Dataen**

I modsætning til tidligere projekter, hvor der som data lag er blevet anvendt lokale data-klasser, hvorfra data kunne hentes, var det i dette projekt et krav at der skulle anvendes en ikke lokal database. Til dette blev vores gruppe tildelt en database, der kunne tilgås på en URL adresse. Denne database er en MySQL database, der er en afart af SQL sproget.

At data ligger på en database har visse fordele. Heriblandt ligger dataen ikke lokalt, så hvis en bruger ændrer på en af de tabeller der ligger i databasen, kan en anden bruger se ændringen på sin maskine med det samme. Altså uden at den ene bruger skulle ”committe”noget og den anden ”pulle”, da data som begge brugere henter er den samme, og de begge kan ændre i den samme data. At al data ligger i en database vi ikke har fysisk adgang til kan dog give problemer.

## **ASE**

### **1. lag**

Det første lag i ASE-projektet er i vægtsimulatoren, da det er denne, som kan håndtere modtagelse og præsentation af de data som ASE'en efterspørger. Gennem Mettler Controlleren, som er en del af logikken, kan vægtsimulatoren modtage forespørgsel på data, som vægtsimulatoren så præsenterer for brugeren. Når brugeren har indtastet det efterspurgte, som evt. kan ses på vægtsimulatorens display, vil vægtsimulatoren returnere denne til Mettler Controlleren. Afslutningsvis vil Mettler Controlleren returnere denne til ASE-Controlleren. Udvekslingen sker ved at Mettler Controller sender en RM kommando til Vægtsimulatoren. Vægtsimulatoren bekræfter efterspørgslen med en RM20A-kommando og sender herefter data tilbage med en RM20B-kommando sammen med klient-inputtet. Det er altså gennem vægtsimulatoren at ASE-programmet kommer tættere på brugeren og derved er vægten en ”GUI”.

### **2. lag**

Det 2. lag i trelagsarkitekturen er logiklaget. Logiklaget udgør de objekter som udveksler data mellem præsentationslaget og datalaget. I ASE programets tilfælde, er dette både ASE Controller og Mettler Controller som udgør dette lag. Mettler Controller udveksler data mellem vores vægtsimulator, som er i præsentationslaget og giver disse videre til vores ASE. ASE sender derefter data videre til DTO'erne, som udgør datalaget. ASE Controller og Mettler Controller arbejder derfor sammen, om at udveksle data mellem præsentationslaget og datalaget. ASE indeholder en konstruktør, som opretter et nyt Mettler Controller objekt, for hver gang vores ASE bliver kørt og giver port nummeret videre til Mettler Controller.

### 3. lag

Vores datalag er det lag som opbevarer og håndterer vores data. Det er derfor at vores tredje lag bliver udgjort af vores DAO'er, DTO'er og selve databasen. Vores DAO'er håndterer vores data, vores DTO'er opbevarer vores data og vores database gemmer al vores data.

## ASE

Afvejnings styrings enheden (ASE) er en applikation, som skulle styre afvejningsprocessen. Applikationen er en viderebygning af det tidligere CDIO\_DEL2 projekt, så den er tilpasset til de ønskede funktioner i dette projekt.

ASE'n er opbygget af 2 hoveddele, ASE Controller (AC), Mettler Controller (MC). Hertil benyttes også en vægtsimulator (VS). AC'en har forbindelse til VS'en igennem MC'en. Dette sker ved at AC'en kalder et antal metoder i MC'en, hvorefter den laver et kald til VS'en, som opfanger dette kald og gør hvad den er programmeret til at gøre med dette kald. Eksempelvis skrive noget på skærmen, eller at tage en afvejning. Dette varierer alt efter hvilken metode AC'en kalder i MC'en.

Hvis AC'en skulle have information fra VS'en, såsom et navn eller et ID., ville der blive sendt kommandoen "RM" til MC'en, som ville tolke denne på en bestemt måde, og sende "RM" videre til VS'en. Dette ville gøre, at noget blev skrevet på skærmen på vægten, såsom "Indtast deres ID", efter der er blevet skrevet på skærmen, vil "RM"ordren stå og vente på et input fra brugeren. Efter brugeren har indtastet sit input, skal brugeren trykke på en forudbestemt tast, hvor indrettet fra brugeren vil blive sendt fra VS'en tilbage til MC'en og så videre til AC'en, hvor AC'en kan anvende denne nye information til eksempelvis at finde brugeren i en database. Herefter kan der laves en anden ordre på MC'en. Dette fortsætter så indtil hele afvejningspro-

cessen er færdig. Denne proces er beskrevet i opgavebeskrivelsen.

Som nævnt benytter ASE'en en simulator, som er den samme vægtsimulator, som blev brugt i projekt CDIO\_D2. Da opgavebeskrivelsen lød at ASE'en skulle bruge enten en vægt eller en vægtsimulator, blev vægtsimulatoren valgt.<sup>2</sup>

En sekvens i ASE'en kan både ses i usecases og sekvensdiagrammet i designdelen, men for kort at nævne hvordan den er designet, kører en normal sekvens således:

Bruger taster et ID og bekræfter sin identitet. Herefter kan der indtastes produktbatch, hvor systemet finder ud af hvad der skal afvejes i dette. Efter en korrekt afvejning vil systemet registrere det i databasen, hvorved operatøren nu kan gå igang med næste afvejning i samme produktbatch. Er produktbatchen færdigt, vil dets status ændres til 2 i databasen og operatøren bliver smidt tilbage til at skulle identificere sig overfor for vægten igen. Skal han herefter måle flere produktbatches, kan processen gennænføres igen.

### **ASE funktioner**

I CDIO\_DEL2 skulle der laves en applikation, som kunne snakke sammen med en MettlerBBK vægt. De kommandoer, som blev brugt, var kommandoer defineret i SISC protokollen, dog med undtagelser af et par kommandoer, der var nødsaget at have med da det er en simulator.

De funktioner som virker er følgende:

RM - laver RM20 ordren

---

<sup>2</sup>Taget fra side 5 i opgavebeskrivelsen: "Afvejnings styrings enheden (ASE) er en applikation, som kan styre afvejningsproceduren på vægten. Applikationen styrer vægten (eller vægtsimulatoren)"

T - Tarerer vægten D - Skriver noget i displayet

DW - Sletter det, der står i displayet

B - Sætter et objekt på vægten (findes ikke på en normal vægt)

Q - lukker ned for vægten/ASE'n (findes ikke på en normal vægt)

Det skal nævnes at de ovenstående kommandoer virker, men kan ikke bruges af brugeren. Dette skyldes at når ASE programmet bliver kørt, vil afvejningsproceduren gå i gang og brugeren skal derfor følge vejledningen, som står på displayet. Et af kravene til ASE'n var også at den skulle kunne afslutte afvejningsproceduren når som helst (med en Q kommando), dette er dog ikke implementeret, da deadline til slut pressede på og det blev prioriteret ud.

Det portnummer som ASE'n og vægtsimulatoren lytter på, kan ændres ved hjælp af følgende (kræver du har eksporteret ASE'n og vægtsimulatoren til en jar fil):

- Åben kommandoprompten
- `cd desktop // Stien til filen, hvis den eksporterede .jar fil ligger på desktop`
- `java -jar *filnavn*.jar *portnummer* // fx java -jar ServerD2.jar 3000`

## Vægtsimulatoren

Vægtsimulatoren er genbrugt fra CDIO2 projektet og er blevet tilpasset efter kravene til dette projekt. Mettler Controlleren opretter forbindelse til vægtsimulatoren, når Mettler Controlleren bliver kørt. Det er også muligt at ændre portnummeret som programmet bliver kørt på fra vægtsimulatoren. Det virker dog kun, hvis portnummeret også bliver ændret på Mettler Controller siden. Udover at oprette forbindelse til Mettler Controller, er det her

menuen på vægtsimulatoren bliver printet.

\*\*\*\*\*

Netto: 0.0 kg

Instruktions display: indtast id

\*\*\*\*\*

Debug info:

Hooked up to /127.0.0.1

Brutto: 0.0 kg

Streng modtaget: RM INDTAST ID

For at taste 'ok' tryk enter

Denne vægt simulator lytter på ordrene

S, T, D 'TEST', DW, RM20, B og Q

på kommunikation sporten.

\*\*\*\*\*

Tast her:

Printet af menuen sker igennem en while-lykke, som lytter til klient-inputtet fra ASE og bliver ved med at køre indtil kommandoen Q bliver modtaget, som er den kommando som afslutter programmet. Via instream, outstream og indtDisp bliver input registreret, opdateret og sendt ud på vægtsimulatoren display.

Hvis vægtsimulatoren modtager en RM kommando, sender den med det samme RM20A tilbage, vægtsimulatoren skriver noget på skærmen, typisk bedes der om bruger input. Når dette input bliver registreret gennem en scanner, vil den returnere den som en RM20B sammen med dette input. ASE'en har



ikke direkte kommunikation til vægtsimulatoren. Det er Mettler controlleren som kommunikerer med ASE'en hvorefter Mettler controlleren giver besked videre til vægtsimulatoren.

## Databasen

### Data Access Laget (DAL)

#### DAO (Data Access Objekt)

For at kunne bruge databasen, har der været behov for at kunne lave CR-UD operationer på databasen, altså create, read, update og delete. Dette har vi gjort ved at lave vores egne DAO klasser, som er oprettet i pakken `tempName.Server.data.daimpl` (i GWT-projektet). Der er oprettet tilhørende interfaces for disse.

#### DTO (Data Transfer Object)

Et DTO er et Java objekt, der indeholder set og get metoder til at sætte, rette og hente data i disse objekter, der er oprettet en DTO for hver tabel i databasen. DTO'erne hænger sammen med DAO'erne til opbevaring af deres respektive områders data. De indeholder kun funktionalitet til opbevaring af en tabels række data, så én post per DTO. For at minimere antallet af metodekald, for returnering af værdier, benytter vi os af DTO'er for vores database afhængige objekter. Disse klasser er serializable, så vi kan sikre os at værdierne bliver returneret i orden og uden problemer.

### Test af database

For at sikre os at samspillet mellem vores metoder i DAO-klasserne og databasen fungerer, har vi lavet en testklasse til at sikre os dette. Dette har vi gjort under pakken "test", og det har lykkedes os at få vores CRUD funktioner til at virke.

Vi har et EER Diagram over vores database, som ligger i bilagene<sup>3</sup>

## Normalisering

Normalisering af vores database er vigtigt, da det beskytter vores data og gør databasen mere fleksibel. I denne proces fjerner vi overflødige data og inkonsistent afhængighed. Det gør det også nemmere, når vi skal lave ændringer i databasen, da elementer kun skal ændres i en af tabellerne, frem for at den samme værdi kommer igen og skal ændres i flere forskellige tabeller. Det skal også være let at finde stien til den data som skal bruges. Det er derfor vigtigt at tabellerne er konsistente, altså at der fx ikke står en operatørs CPR nr. i tabellen med recepter. Da operatørens CPR nr er relateret til operatøren, hører denne til i operatør tabellen. Vores database er på 3. normalform, som vil sige at den overholder reglerne for 1., 2. og 3. normalform. Der findes mange niveauer af normalisering, men for de fleste anvendelser anses 3. normalform for at være det højeste nødvendige niveau, som også er den grad vi har normaliseret vores database til.

### 1. Normalform

- Fjerne gentagende grupper i individuelle tabeller.<sup>4</sup>
- Opret en særskilt tabel for hvert sæt af relateret data.
- Identificer hvert sæt af relateret data med en primærnøgle.

På 1. normalform må der ikke bruges flere felter i en tabel, til at gemme data som er ens. Fx hvis vi har en råvare som bliver brugt i flere forskellige recepter, så skal vi ikke bare tilføje recepten til råvarens tabel. Det er ikke

---

<sup>3</sup>Side 92

<sup>4</sup>Punkterne i hver normalform er taget fra Microsofts hjemmeside: <https://support.microsoft.com/da-dk/kb/283878>

nemt at tilpasse databasen til et dynamisk antal råvare, da dette vil kræve tabelændringer, som giver problemer i programmet. I stedet giver vi råvare en nøgle og deler tabellen op, så vi har en tabel med recepterne og en tabel med råvare, som så er kædet sammen.

## 2. Normalform

- Opret særskilte tabeller til værdisæt, som kan anvendes til flere poster
- Relater disse med en fremmednøgle

Nogle informationer kan være relevante i flere forskellige tabeller så på 2. normalform kontrollerer vi netop at disse har deres egen tabel i stedet for at gemme informationerne i hver tabel. Fx er en operatørs initialer relevant flere steder. Operatør tabellen indeholder informationerne om hver operatør, hvor hver operatør er tildelt et ID. Ved hjælp af dette ID kan man finde operatøren i tabellen 'operatorer' og finde de relevante informationer. Det er derfor vigtigt at alt indholdet i tabellen er afhængigt af tabellens primærnøgle, som er operatørens ID.

## 3. Normalform

- Fjern de felter, der ikke afhænger af nøglen

På 3. normalform finder vi de poster som ikke er en del af tabellens nøgle, da disse ikke hører til i tabellen, men i stedet i en tabel for sig selv. Fx hvis vi skal bruge leverandøren til en bestemt recept. Så vil der være en tabel for sig selv med råvare og deres leverandør og denne ville så være kædet sammen med recepterne, med en nøgle så der ikke opstår flere instanser af den samme recept, fordi der fx kan findes forskellige leverandører til den samme vare.

## Transaktionslogik

I dette projekt er der anvendt en database. I denne database sker der opdateringer visse steder, såsom opdateringer, tilføjelser og sletninger. Lige så snart der enten skal skrives til databasen eller at der skal slettes noget fra den, kan det give mening at anvende transaktionslogik. Transaktionslogik er en måde at ændre data i en database på en sådan måde at databasen ikke kan komme til at være ufuldstændig. I det ligger der, at hvis der skal sættes data ind i flere tabeller på i samme omgang, altså det tager lidt tid fra der startes til det hele er færdigt, så kan databasen crashe eller andre faktorer kan gøre at forbindelsen forsvinder. Hvis dette sker, kan det være at noget af dataen er kommet ind. Her er nøgleordet ”noget” da man ikke kan være sikker på om det hele eller ingen data er kommet igennem. Dermed kan der være steder i databasen, hvor data ikke er kommet helt ind, altså der er et sted hvor der mangler noget data.

Transaktionslogik vil her være kode der sikre sig at al data er kommet igennem, eller at ingen data er. Dette kan opnås ved at anvende en SQL buffer, sammen med ”commits” eller ”rollbacks”. I vores projekt ville transaktionslogik være ideel i det tilfælde at der for eksempel skal slettes én recept, da dette kræver sletning af mange tupler i mange andre tabeller, da disse ikke kan eksistere uden recepten, men recepten kan eksistere uden dem. Så her ville transaktionslogik sammen med stored procedures give rigtig god mening.

Dog er dette blevet nedprioriteret i udviklingen af det færdige produkt, og er dermed ikke blevet implementeret. Men til viderebygning af systemet ville dette være et sted at sætte ind.

# WebInterface

## GWT

GWT (Google Web Toolkit) er en gruppe af open source værktøjer, som blev udviklet af Google. Disse værktøjer er designet til at give brugeren en mulighed for at udvikle en FrontEnd applikation i Java. Der findes forskellige open source plugins, som gør det nemmere at arbejde med GWT i forskellige IDE'er. I vores tilfælde bruger vi et plugin til Eclipse, som kan hentes fra [gwtproject.org](http://gwtproject.org). Vi bruger GWT til at udvikle vores web applikation, som gør det muligt for de forskellige brugere at løse deres respektive opgaver. I GWT benytter vi os af RemoteProcedureCalls(RPC) til at udveksle objekter, eller data i mellem client og server. Vores Service (metoder vi har lavet i ServiceImpl) kører på serverside. Clientside laver så forespørgsler til de metoder via asynkrone kald. Disse kald opsnapper så, via callback adresser, metoder specificeret i ServiceClientImpl, svaret fra serveren når det kommer og afhængigt af hvilken type datacontainer der modtages i callback aktiveres den korrekte modtagermetode i maingui.

## Service-klasser

Vi benytter os af i alt 5 Service-klasser i dette GWT projekt. 4 i klienten og 1 på serveren.

### Service.java

Service.java, er et simpel interface på vores klient, som definerer de metoder, der kan blive kaldt til serveren.

Der er en linje i toppen der hedder:

```
@RemoteServiceRelativePath("greet")
```

Denne linje definerer stien der skal til for at få forbindelse til serveren, som i dette tilfælde er "greet". Klassen skal også extende RemoteService, som er en RPC interface.

Herefter bliver alle metoderne defineret, som de ville blive i en interface.

### **ServiceAsync.java**

ServiceAsync.java er Service.java's Asynkrone modpart. Det vil sige, at den har de samme metoder, som Service.java har, men her med et tilføjet Async-Callback.

Dette betyder, at den sørger for, at klienten ikke behøver at vente på serverens svar, når den sender en forespørgsel. Dette vil blive forklaret nærmere, i ServiceClientImpl.java sektionen.

### **ServiceClientImpl.java**

ServiceClientImpl.java er nok den mest spændende service-klasse hos klienten.

I konstruktøren bliver en ServiceAsync service, defineret med kommandoen:

```
GWT.create(Service.class);
```

Som definerer, at det bliver en asynkron klasse med interfacet Service.java. Herefter bliver ServiceDefTarget endpoint defineret, som slut-URL'en, hvilket er den linje der blev omtalt i Service.java sektionen. Efterfølgende bliver endpointets url sat til at være det, som den får, når ServiceClientImpl.java objektet bliver kreeret. Det er også herfra, GUI'en bliver skabt. GUI'en får så ServiceClientImpl med i sin parametreliste, som den bruger til at lave kald til serveren. Det er altså her i ServiceClientImpl.java, at kaldet starter (udover fra GUI'en). Denne klasse har så metoder, der sender dem videre til Servicen. Nemlig ServiceAsync.java, for at sørger for at de bliver kørt asynkront.

Vi har nogle Inner-classes her, som er vores callbacks. Disse callbacks er klasserne, der tillader at alt dette kan køre asynkront. For når serveren er færdig med at fortage de handlinger, den er blevet bedt om, sender den noget tilbage til klienten, som bliver håndteret med disse callbacks. Callbacksne implementerer interfacet `AsyncCallback`, som indeholder to metoder. `onFailure()` og `onSuccess()`. Disse to metoder er rimelig selvforklarende, hvor `onFailure()` kører, når serveren fejlede i at gøre det den blev bedt om, bliver `onSuccess()` kørt, når serveren lykkedes.

I vores `onFailure()` metoder, tjekker vi hvilken fejlmeddelelse der er blevet givet fra serveren og fortolker det, samt giver brugeren besked i form af fortolkningen.

I vores `onSuccess()` metoder, tjekker vi hvad vi har fået tilbage fra serveren, hvorefter metoder i GUI'en bliver kaldt, som passer til det respons der kom fra serveren. Det er her GUI'en får besked om, at nu har vi modtaget et svar fra serveren, så nu kan den bruge dette svar.

Vi har fundet frem til, at skulle benyttes 3 forskellige callbacks. Én til get-metoder, som tjekker hvilken metode den skal kalde, alt efter hvad der er hentet fra databasen. Én til metoder, der tilføjer ting i databasen, som giver fejlbeskeder, hvis man prøver at indsætte noget i databasen den ikke tillader. Dette kunne være at man vil indsætte noget i databasen, som har en primærnøgle, der allerede eksisterer.

Vi har også én til operatør oprettelse og redigering, bare for at specificere, hvis man indtaster et password eller navn, som er for langt i forhold til hvad databasen tillader.

`ServiceClientImpl.java` har interfacet `ServiceClientInt.java`, der fungerer som et helt basalt interface.



## ServiceImpl.java

ServiceImpl.java er den ene service klasse, som ligger på serveren. Det er denne klasse, som modtager kaldene fra service-klasserne på klient-siden.

Det eneste denne klasse gør, er at sende et nyt kald til vores DAO-klasser, som manipulerer med data i databasen, samt sende et respons tilbage til klienten. Denne klasse fungerer, altså som et mellemlid, mellem klient og database.

## Klient

Klienten er den del af GWT-projektet, som brugeren kommer til at benytte. Det er denne del, der skaber brugergrænsefladen, samt behandler brugerens input og sender det videre.

Koden til klienten bliver skrevet i Java, men da java ikke bruges som web-sprog, konverterer GWT koden til Javascript, som er et almindeligt anvendeligt sprog til skabelse af web baserede sider.

I projektet er der én html fil og én associeret css fil. Der er i html-filen en div, defineret som "main". Dette er vores panel i programmets RootPanel.

I java-koden kan vi ved hjælp af denne kommando:

```
RootPanel.get("main").add("tilføjelse");
```

tilføje genstande til vores webside.

Vil man gerne tilføje et stykke text, ville man for eksempel kunne skrive følgende i en given metode:

```
Label labelName = new Label("Dette er en GWT webside");  
RootPanel.get("main").add(labelName);
```

Bliver ovenstående kørt i programmet. Vil websiden vise et stykke tekst, hvor der står: "Dette er en GWT webside". I vores program, har vi valgt at oprette én container i form af et VerticalPanel, som det eneste der bliver tilføjet til

RootPanel.

Dette betyder nemlig, at når vi vil tilføje noget til websiden, skal vi blot bruge udtrykket:

```
container.add("tilføjelse");
```

Hvor, hvis man vil fjerne det der er på websiden bruges:

```
container.clear();
```

Dette gør det hurtigt og simpelt, at forme sin webside efter hvad der skal være på den, som for eksempel, når man skifter imellem at være på forskellige menuer. Ulempen ved dette er, at det begrænser hvor meget der kan designes på projektet, i og med vi kun benytter ét panel. Men da menuerne, der skal eksisterer i dette projekt er forholdsvis simple, har vi valgt at gøre det på den her måde. Mere om dette kan ses i sektionen om problemer.

## Server

Serverens eneste reelle opgave er, at fungere som et mellemlid, mellem database og klient. Serveren består, ud over sin ServiceImpl klasse, kun af tre pakker. Den ene er database pakken, hvor der er klasser og metoder, der bruges til at oprette forbindelse til MySQL-databasen. Den anden er daoimpl pakken, der indeholder vores DAO(Data Access Object)-klasser, som er dem, der laver kald til MySQL-databasen. Den sidste består af Interfaces for DAO-klasserne, samt en klasse til at håndtere DALEExceptions.

## Shared

I shared pakken, har vi de filer, som både server og klient benytter sig af. Dette er for eksempel vores DTO(Data Transfer Objekt)-klasser, som om-danner data, fra databasen og transformere dem til objekter, vi kan bruge i java-koden.

Herudover er der også nogle klasser, som indeholder metoder til tjek af kodeord. Disse kan benyttes, når man tjekker adgangskoden, som en bruger har skrevet for at logge ind, eller når en administrator vil oprette en ny bruger.

## Implementering

### Generel implementering

En vigtig ting at nævne ved implementeringen er måden projektet er delt op på. Ud fra opgavebeskrivelsen, ses opgaven som værende delt op i 3: GWT, ASE og vægtsimulator. Projektet er udviklet sådan, så GWT styrer administrator, farmaceut og værksførers funktioner. ASE'n ligger for sig selv og skal køres hver gang et produktbatch skal laves. ASE'n benytter derved vægten(simulatoren) og på den måde hænger de to sammen. Det vil dog sige at GWT og ASE'n ikke har andet med hinanden at gøre, end at de bruger samme database.

### Password

I webapplikationen bruges der ofte nogle passwords i forbindelse med at logge ind, eller oprette en bruger og der er i denne forbindelse behov for noget logik vedrørende passwords, da det skal overholde DTU's standard (jf. kravsspecifikationen).

Passwordklasserne består af en simpel interface klasse, der indeholder en metode til password data klassen, en PasswordData klasse og en Password-Methods klasse.

I password dataklassen bliver der oprettet en arraylist, som indeholder alle de karakterer der er tilladte i de passwords der skal bruges. Arraylisten bliver fyldt vha. nogle for-loops der tæller nogle indexes op og tilføjer et tal, eller et bogstav til arraylisten, hver gang loopet bliver inkrementeret. Til sidst bliver

tegnene føjet til arraylisten "manuelt" og arraylisten ender med at bestå af alle små og store bogstaver, fra a til z, alle tal fra 0-9 og nogle tegn. Denne arraylist bliver så brugt i en "get metode", der returnerer arraylisten med informationerne, så den kan bruges i PasswordMethods klassen.

I PasswordMethods klassen er der en række forskellige metoder til at tjekke om et password er gyldigt, der er brugt andre steder i koden til conditions. Metoden "checkPassLength" tager en streng, som parameter og tjekker om passwordet er mindre end 6, eller større, eller lig med 6 og returnerer true, hvis passwordet er større, eller lig med 6. Metoden "checkPass" tager en streng som parameter og tjekker om strengen indeholder mindst 3 ud af de 4 grupper; tal, små bogstaver, store bogstaver og tegn, og returnerer true, hvis det er tilfældet og ellers false. Arraylisten fra PasswordData klassen bliver i denne metode brugt til at lave nogle for-loops, med nogle intervaller der indeholder grupperne af tal, små bogstaver, store bogstaver og tegn i arraylisten, og til at skelne mellem de forskellige karakterer og bruge disse loops, til at lave nogle tjek, for hvorvidt et password indeholder en del af nogle af intervallerne. Yderligere er der metoden "samePass", som tager 2 strenge, som parametre og sammenligner dem. Hvis de er ens returnerer metoden true og ellers false. Den sidste metode, "correctUserPassword", i klassen tager en streng password, en List med objekter af "OperatoerDTO" klassen og en integer id, som parametre og består af et for-loop, med index fra 0 til størrelsen af listen og sammenligner strengen, som var den første parameter, og id'et, med alle passwords og id'er i listens objekter og returnerer true hvis de begge matcher.

# Test

## Fysiske tests

I forbindelse med GWT blev der lavet mange fysiske tests, for at afprøve alle funktioner og komme ud i alle hjørner af systemet. Alle knapper blev testet og der blev sammenlignet i databasen før og efter en test, for at se om der skete det der burde.

Ud over det blev der også lavet negative tests, som fx at redigere en bruger med et ID, som ikke fandtes, for at se om der opstod nogle problemer, eller om der manglede de relevante fejlmeddelelser. Der blev yderligere også lavet tests på de forskellige input lokationer, for at se om alle exceptions blev behandlet og om der kom de rigtige fejlmeddelelser op. Et eksempel kunne være at sætte et forkert antal cifre ind på CPR-nummer feltet, eller tjekke at den rigtige fejlmeddelelse dukkede op, hvis man skrev bogstaver et sted, hvor der kun må accepteres tal, som gyldigt input.

## Manuel tests af MYSQL DAO klasser

I denne opgave har vi valgt at udfører manuel test på klasserne MYSQLRaavareBatchDAO, MYSQLRaavareDAO, MYSQLProduktBatchDAO, MYSQLReceptDAO og MYSQLOperatoerDAO. For at sikre os at metoderne til manipulation af data i databasen vha. SQL forespørgelser som "SELECT \* FROM", "INSERT INTO", "UPDATE" og "DELETE" fungerer som ønsket.

Klassen "RaavareDAO" indeholder fire metoder som alle ønskes testet- *addRaavare()*, *getRaavareList()*, *updateRaavare()* og metoden *deleteRaavare()*. Ser vi nærmere på metoden *getRaavareList()* fra klassen "MYSQLRaavareDAO" forventes det at metoden udskriver en liste over alle de råvarer database tabellen "raavare" indeholder.

```

public List<RaavareDTO> getRaavareList() throws DALException {
    List<RaavareDTO> list = new ArrayList<RaavareDTO>();
    ResultSet rs = connector.doQuery("SELECT*FROM raavare");
    try{
        while (rs.next()){
            RaavareDTO raavareDTO = new RaavareDTO();
            raavareDTO.setrID(rs.getInt("raavare_id"));
            raavareDTO.setrName(rs.getString("raavare_navn"));
            raavareDTO.setDeliverer(rs.getString("leverandoer"));
            list.add(raavareDTO);
        }
    } catch(SQLException e){
        throw new DALException(e.getMessage());
    }
    return list;
}

```

Figur 1: Udsnit der illustrere metoden *getRaavareList()* fra klassen MySQL-RaavareDAO

Det har vi ønsket at teste, derfor er en test klasse oprettet ” MySQLRaavareTest”, hvor metodens output testes og sammenlignes med det forventede output. Opstår der en fejl ved eksekvering af metoden udskrives en Exception e.



```

private static void test2() throws DALException {
    MySQLRaavareDAO dao = new MySQLRaavareDAO();
    try {
        System.out.println("Test 2: Alle raavare vises -->  "+ dao.getRaavareList());
    }
    catch (DALException e) {
        System.out.println(e.getMessage());
    }
}

```

Figur 2: Udsnit der illustrere *test2()* fra klassen MySQLRaavareTest, her er det metoden *getRaavareList()* der testes

Gennemføres metoden uden fejl forventer vi en udførlig råvareliste med raavare\_id, raavare\_navn, leverandoer som er identisk med den råvareliste database tabellen ”raavare” indeholder.

Result Grid   Filter Rows: <input type="text"/>			
	raavare_id	raavare_navn	leverandoer
▶	1	vand	Vand A/S
	2	magnesium	Miner
	3	jernnn	Minerrr
	4	salt	Haribo
	5	gelatine	Haribo
	6	c-vitamin	Vitamin A/S
	7	glukose	KemiFolket
	9	k	kk
★	NULL	NULL	NULL

Figur 3: Udsnit der illustrere database tabellen "raavare" som er det forventede output ved eksekvering af *test2()*

```
<terminated> MySQLRaavareTest [Java Application] C:\Program Files (x86)\Java\jre1.8.0_77\bin\javaw.exe (16/06/2016 10.20.48)
Test 2: Alle raavare vises --> [1 vand Vand A/S, 2 magnesium Miner, 3 jernnn Minerrr, 4 salt Haribo,
```

Figur 4: Udsnit der illustrere aktuelt output ved eksekvering af *test2()* fra klassen MySQLRaavareTest, dette output sammenlignes med Figur: 3

Samme fremgangsmåde er benyttet til at teste de resterende klasser, til klassen MySQLRaavareBatchDAO er der oprettet en test klasse MySQLRaavareBatchTest, klassen MySQLProduktBatchDAO har en test klasse MySQLProduktBatchTest, til klassen MySQLReceptDAO er test klassen MySQLReceptTest oprettet og klassen MySQLOperatoerDAO har test klassen MySQLOperatoerTest og de dertilhørende metoder er på samme måde testet.

## ASE test

Til dette projekt er der ikke direkte testdokumentation af ASE'n. Skyldet tidspres kunne der ikke oprettes test til ASE'n, men der er blevet kørt grundige manuelle blackbox test, herunder rigtig mange negative test af systemet.

## Problemer

Nogle gange i længere tid, ville databasen blive utilgængelig af uransagelig grunde. Dette gjorde det at i disse perioder var det umuligt at teste programmets normale funktioner, da databasen i dette projekt holder al data, heriblandt data om brugere og deres rettigheder. Når databasen er nede, kan programmet ikke hente oplysningerne om det login man har tastet i programmet. For slet ikke at tale om anden data som råvare og recepter, der ikke kan tilgås selvom man var inde i programmet.

Vi ville gerne have haft de to dele i projektet (GWT og ASE/vægt) til at hænge lidt mere sammen end bare med databasen. Dette havde vi udtænkt kunne gøres, ved at have en knap i GWT-projektet, som startede ASEn op. Dette viste sig dog ikke at være let at gøre, da man ikke umiddelbart kan starte et andet program op, med et tryk på en knap. Altså er det færdige produkt blevet to forskellige filer, der skal startes hver for sig.

## GWT

Da vi ikke har brugt UiBinder i dette projekt, har det resulteret i, at det visuelle design ikke er synderligt imponerende. Vi valgte ikke at benytte os af dette, da vi mente, at programmet var simpelt nok, til at det ikke var nødvendigt at koncentrere sig så meget om layoutet. Dette har givet nogle problemer, når vi for eksempel har lavet en log ud knap. Vi ville gerne have



denne knap til at sidde væk fra de andre knapper, fortrækkelsesvis nede i venstre hjørne af siden, men dette kunne vi ikke få til at fungerer, grundet manglende planlægning af, hvordan layoutet skulle sættes om.

Skulle vi lave projektet om, havde vi benyttet os af UiBinder, til det visuelle design.

Vi havde nogle problemer med vores tabeller, da vi ikke helt havde styr på, hvordan de skulle sættes op i GWT. Vores løsning blev et CellTable, som vi giver en fast max størrelse på 500, hvilket vil give problemer, hvis man skulle gå hen at have en database, med mere end 500 elementer i en tabel.

Selvom dette måske ikke er så sandsynligt i dette projekt, kunne det være et problem i det virkelige liv, hvor man måske hellere ville have en dynamisk tabel, der har en scrollbar i siden. Dette var dog ikke lige til at få til at fungere. Vi prøvede ting, som at sætte CellTablen ind i et ScrollPanel, men dette gjorde hele siden scrollbar, ikke tabellen.

Vi prøvede også med et DataGrid, men det viste sig, at være for besværligt til at det var værd at bruge tid og besvær på.

Nogle klasser havde vi i CDIO-D3 lagt på server-siden, men i dette projekt har vi skulle bruge dem på klientsiden også. Dette har resulteret i, at vi skulle lægge klasserne i Shared pakken. Dette gav nogle problemer, da nogle af metoder benyttede objekter fra andre serverklasser. Disse objekter prøvede klienten at kompilere, hvilket gav fejl, da filer der ligger i Server pakken er usynlig for filer i Client pakken. Altså måtte disse metoder omskrives, så de ikke længere skulle bruge objekter fra serverklasser.

Vores GWT-projekt er bygget op således, at når man er ude i en operatørmenu, f.eks. admin menuen, og derefter klikker ind på f.eks. råvaremenu, bliver data om råvarer hentet her. Det betyder, at når man så laver nogle

ændringer til databasen i råvaremenuen, er man nødt til at skulle ud i adminmenuen igen, for så at gå tilbage til råvaremenuen, for at ændringerne kan ses. Løsningen er simpel idet, at programmet automatisk sender brugeren tilbage til adminmenuen, efter man har foretaget en ændring.

Da denne funktion ikke virker unaturlig eller forvirrende i praksis, har vi valgt at beholde det i denne tilstand, selvom det godt kunne ændres.

## **ASE/Vægt**

Der var nogle problemer med forståelsen af, hvordan vægten og ASE skulle hænge sammen. Grundet vores misforståelse, blev ASE'n udviklet sammen med vægten men kommunikationen imellem dem og især det forhold de havde til hinanden var forkert. Derfor blev det hele lavet om, så det nu alt sammen var kodet i vægten (så ASE'n og vægten var én og samme ting). Der blev herefter fundet ud af at dette også var forkert, så for 3. gang måtte ASE'n nu laves. Dette kostede rigtig meget tid, der egentlig var sat af til rapporten og er også den største grund til at ASE'n ikke er implementeret så den fungerer med en rigtig vægt.

## **Viderebygning**

### **GUI**

Den udarbejdede webapplikation har et meget simpelt design. Der kunne i en viderebygning af systemet, konstrueres et mere attraktivt design, der også ville få webapplikationen til at fremstå mere professionelt. Dette kan gøres med hjælp fra UIBinder. Der kunne fx indføres en navigations bar i toppen af siden, hvor de forskellige emner lå. Det ville gøre hjemmesiden meget mere overskuelig, så man har lettere ved at finde frem til det man skal. Generelt ville et app ligende interface, hvor knapper har farver give hjemmesiden et

skarpt look.

Det kunne også være smart at gøre tabellerne i webapplikationen scroll-bare, så man kan bruge mussehjulet til at scrolle igennem f.eks. ens råvarer.

## **Program Logik**

Som beskrevet i Problem sektionen, sender programmet brugeren tilbage til hovedmenuen, efter man har foretaget en ændring i databasen. Dette kunne godt ændres. Det ville kræve at metoderne kaldte deres update metode til serveren, i stedet for bare at vende tilbage til operaørmenuen. Hvis man vælger metoden i RaavareMenu.java, som hedder addRaavare(), ville man udskifte linjen:

```
mainGUI.adminCheck(prevMenu);
```

Med:

```
mainGUI.serviceImpl.getRaavare();
```

Dette sørger for, at den bare returnere til råvaremenuen i stedet for f.eks. adminmenuen, hvis det er det man er logget ind som.

## **Database**

Til dette projekt haves ikke noget transaktionslogik til databasen. Dette er uheldigt, da vi ofte manipulerer med data i databasen fra flere forskellige tabeller, samtidig med, at der kan risikeres at ændres i flere tabeller på én gang.

Transaktionslogik der sørger for, at intet data bliver delvist indsat i databasen, ville være en rigtig god idé for dette projekt, da langt de fleste funktioner i programmerne omhandler indsættelse eller redigering af data i databasen. Vi har også foreign keys flere steder i tabellerne, hvilket også giver en grund

til at indføre noget transaktionslogik.

Ud over dette, kunne nogle views blive etableret til bedre at illustrere databasen og hvordan sammenhængen er mellem tabellerne.

## Konklusion

Det har lykkedes os at lave et system, som kan bruges til afvejning af råvarer og lave dokumentation af de afvejninger der bliver foretaget, samt registrere hvilke råvarer som bliver benyttet. Systemet kan også benyttes til lagerstyring af produkter og råvarer. Det er muligt at lave, definere og ændre information for råvarer og recepter. Herudover har det også lykkedes os at lave systemet således, at der kan implementeres leverandører til de forskellige råvarer, så disse kan hjemtages til firmaets lager i råvarebatches, hvor man kan have mulighed for at se mængden af den pågældende råvare. Recepterne i systemet er udviklet således at de indeholder de forskellige råvarer som skal benyttes, samt doseringer af disse for at kunne udvikle et produkt. Man kan også benytte recepterne til produktbatches.

For at få systemets funktioner til at virke, har vi oprettet en database som kan lagre, opretholde og gemme de data der er behov for. Systemet kan tilgås fra klienten via et dynamisk webinterface vi har udviklet med Google Web Toolkit, som gør det muligt at logge på systemet fra en browser, hvor de forskellige aktører har deres egen tilpassede menu med de funktioner de har adgang til. Til afvejningsdelen har vi formået at tilkoble vores vægtsimulator fra CDIO2 til vores system, så man kan indtaste afvejninger uden en fysisk vægt.

Som nævnt i afsnittet ASE har vi ikke fået nået at implementere at man kan afslutte afvejningsproceduren på et hvilket som helst tidspunkt, grundet tidspres og prioritering.

Vi kan hermed konkludere at vi har udviklet et fungerende program, hvor samspillet mellem de forskellige dele virker, hvormed hele kravspecifikationen, med undtagelse af det ovennævnte, synes besvaret.

# Bilag

## Use Case beskrivelser

### Tilføj Råvare

**ID:** 01

**Kort Beskrivelse:** Bruger tilføjer en råvare

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Bruger har den nødvendige info om råvaren.

#### Primært flow:

1. Bruger logger ind
2. Bruger klikker på “Råvarer” knappen
3. Bruger klikker på “Tilføj Råvare” knappen
4. Bruger indtaster følgende:
  - Råvare ID
  - Råvare navn
  - Råvare leverandør
5. Brugeren klikker på “Opret”
6. Besked om succesfuldt opretelse kommer frem

**Efterfølge:** Råvaren er nu oprettet

#### Alternative Flows:

1. Bruger logger ind
2. Bruger klikker på “Råvarer” knappen
3. Bruger klikker på “Tilføj Råvare” knappen
4. Bruger indtaster følgende:
  - Råvares ID
  - Råvares navn
  - Råvares leverandør
5. Brugeren klikker på “Opret”
6. Fejlmeddelelse kommer frem ”Kunne ikke oprettes i databasen, grundet valgte ID allerede eksisterer

## Opdater Råvare

**ID: 02**

**Kort Beskrivelse:** Bruger opdaterer en råvare

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Bruger har nødvendig info om råvaren og råvaren er oprettet i databasen.

### **Primært flow:**

1. Bruger logger ind
2. Bruger klikker på “Råvarer” knappen
3. Bruger klikker på “Redigerer Råvare” knappen
4. Bruger vælger en råvare udfra dropdown menuen og kan derefter redigere i:
  - Råvare navn
  - Råvare leverandør
5. Brugeren klikker på “Opdater”

**Efterfølge:** Råvaren er nu opdateret



## Slet Råvare

**ID: 03**

**Kort Beskrivelse:** En bruger sletter en råvare

**Primær Aktør:** Admin / Farmaceut / Værksfører

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at slette råvare og den råvare der ønskes slettet er ikke i brug

### Primært flow:

1. Bruger logger ind
2. Bruger trykker på knappen "Råvare"
3. Bruger trykker på knappen "Rediger Råvare"
4. Bruger vælger en råvare ud fra ID'erne i drop-down listen
5. Bruger trykker på knappen "Slet" og bliver sendt tilbage til brugerens menu
6. Bruger trykker på knappen "Log ud"

**Efterfølge:** Råvaren er nu slettet og brugeren er logget ud

### Alternative Flows:

#### Råvare ID i brug i en anden tabel

1. Bruger logger ind
2. Bruger trykker på knappen "Råvare"
3. Bruger trykker på knappen "Rediger Råvare"

4. Bruger vælger en Råvare ud fra ID'erne i drop-down listen
5. Bruger trykker på knappen "Slet"
6. Bruger får en fejlbesked om at Råvaren der ønskes slettet, er i brug i en anden tabel og kan derfor ikke slettes

**Efterfølge:** Brugeren kan slette en anden Råvare der ikke er i brug eller navigere ud af systemet

## **Se Alle Råvare**

**ID:** 04

**Kort Beskrivelse:** Bruger vil se en liste over alle råvare

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Databasen indeholder råvarer

### **Primært flow:**

1. Bruger logger ind
2. Bruger klikker på “Råvarer” knappen
3. Bruger klikker på “Se Råvare” knappen

**Efterfølge:** Alle råvare vises i en tabel

## Opret Bruger

**ID:** 05

**Kort Beskrivelse:** Admin opretter en ny bruger

**Primær Aktør:** Admin

**Sekundære Aktører:** Database

**Forudsættelser:** Admin har nødvendig info om brugeren

### Primært flow:

1. En admin logger ind
2. En admin klikker på “Opret ny operatør” knappen
3. En admin indtaster oplysningerne på den nye bruger:
  - Navn
  - Initialer
  - Cpr-nummer
  - Kodeord
  - Bruger type
4. En admin klikker på “Opret”
5. Meddelelse om korrekt oprettelse kommer frem på skærmen
6. Admin trykker “ok” på meddelelsen

**Efterfølge:** Admin bliver returneret til Administrator Menuen og brugeren er nu oprettet med de ønskede attributter

### Alternative Flows:

### **For langt Cpr-nummer**

1. En admin logger ind
2. En admin klikker på “Opret ny operatør” knappen
3. En admin indtaster oplysningerne på den nye bruger:
  - Navn
  - Initialer
  - For langt : Cpr-nummer
  - Kodeord
  - Bruger type
4. En admin klikker på “Opret”
5. Meddelelse om korrekt længde af et Cpr. nummer kommer frem

### **For kort Cpr-nummer**

1. En admin logger ind
2. En admin klikker på “Opret ny operatør” knappen
3. En admin indtaster oplysningerne på den nye bruger:
  - Navn
  - Initialer
  - For kort : Cpr-nummer
  - Kodeord
  - Bruger type
4. En admin klikker på “Opret”
5. Meddelelse om korrekt længde af et Cpr. nummer kommer frem

### **For mange initialer**

1. En admin logger ind
2. En admin klikker på “Opret ny operatør” knappen
3. En admin indtaster oplysningerne på den nye bruger:
  - Navn
  - For mange : Initialer
  - Cpr-nummer
  - Kodeord
  - Bruger type
4. En admin klikker på “Opret”
5. Meddelelse om at initialer skal være maksimalt 3 karaktere

### **For få initialer**

1. En admin logger ind
2. En admin klikker på “Opret ny operatør” knappen
3. En admin indtaster oplysningerne på den nye bruger:
  - Navn
  - For få : Initialer
  - Cpr-nummer
  - Kodeord
  - Bruger type
4. En admin klikker på “Opret”
5. Meddelelse om at initialer skal være maksimalt 3 karaktere

## Rediger Bruger

**ID:** 06

**Kort Beskrivelse:** Admin ændre på en bruger

**Primær Aktør:** Admin

**Sekundære Aktører:** Database

**Forudsættelser:** Admin har nødvendig info om brugeren og brugeren er oprettet i databasen.

### Primært flow:

1. Admin logger ind
2. Admin klikker på “Ændre attribut for operatør” knappen
3. Admin indtaster bruger ID for den operatør der skal ændres, samt det nye:
  - Brugernavn
  - Initialer
  - Cpr-nummer
  - Kodeord
  - Bruger type
4. Admin klikker på “Opdater” og får bekræftet at opdateringen var succesfuld

**Efterfølge:** Operatøren er nu opdateret, med nye attributter

### Alternative Flows:

1. En admin logger ind

2. En admin klikker på “Ændre attribut for operatør” knappen
3. En admin indtaster bruger ID for den operatør der skal ændres, samt det nye:
  - Brugernavn
  - Initialer
  - Cpr-nummer
  - Kodeord
  - Bruger type
4. Admin klikker på “Opdater”
5. Fejlbesked kommer frem med teksten “Indtast venligst et gyldigt ID for en eksisterende operatør”



## Inspicer Bruger

**ID: 07**

**Kort Beskrivelse:** Admin inspicerer en bruger

**Primær Aktør:** Admin

**Sekundære Aktører:** Database

**Forudsættelser:** Admin har bruger ID på brugeren, og brugeren er oprettet i databasen.

### Primært flow:

1. Admin logger ind
2. Admin klikker på "Inspicer en operatør" knappen
3. Admin indtaster bruger ID for den operatør der skal inspiceres
4. Admin klikker på "Inspicér"
5. Brugerens brugeroplysninger for brugeren med det indtastede bruger ID vises på skærmen
6. Admin afslutter ved at klikke "Tilbage" og dernæst "Log ud"

**Efterfølge:** Information om en bruger er blevet vist og Admin er nu logget ud.

### Alternative Flows:

1. Admin logger ind
2. Admin klikker på "Inspicer en operatør" knappen
3. Admin indtaster et bruger ID der ikke eksistere i databasen
4. Admin klikker på "Inspicér"

5. Fejlmeddelelsen: Ingen operatør med ID: >indtastet bruger ID< blev fundet. Indtast venligst et gyldigt ID.

**Efterfølge:** Brugeren kan vælge at indtaste et nyt ID eller at navigere ud af systemet.

## Tilføj Råvarebatch

**ID:** 08

**Kort Beskrivelse:** Bruger tilføjer en råvarebatch

**Primær Aktør:** Admin / Farmaceut / Værksfører

**Sekundære Aktører:** Database

**Forudsættelser:** Bruger har den nødvendige info om råvaren.

### Primært flow:

1. Bruger logger ind
2. Bruger klikker på “Råvarebatch” knappen
3. Bruger klikker på “Tilføj Råvarebatch” knappen
4. Bruger indtaster følgende:
  - Råvarebatches ID
  - Råvare ID
  - Råvare mængde
5. Brugeren klikker på “Opret”
6. Besked om succesfuld oprettelse fremkommer

**Efterfølge:** Råvarebatch er nu oprettet

### Alternative Flows:

1. Bruger logger ind
2. Bruger klikker på “Råvarebatch” knappen
3. Bruger klikker på “Tilføj Råvarebatch” knappen

4. Bruger indtaster følgende:

- Råvarebatches ID
- Råvare ID
- Råvare mængde

5. Brugeren klikker på “Opret”

6. Fejl besked kommer frem ”Kunne ikke oprettes i databasen, grundt valgte ID allerede eksisterer”

## Opdater Råvarebatch

**ID:** 09

**Kort Beskrivelse:** Bruger opdaterer et råvarebatch

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Bruger har nødvendig info om råvaren og råvarebatch og disse skal være oprettet i databasen

### Primært flow:

1. Bruger logger ind
2. Bruger klikker på “Råvarebatch” knappen
3. Bruger klikker på “Redigerer Råvarebatch” knappen
4. Bruger vælger et råvarebatch ID udfra dropdown menuen og kan derefter redigere i:
  - Råvare mængde
5. Brugeren klikker på “Opdater”

**Efterfølge:** Råvarebatch er nu opdateret

## **Slet RåvareBatch**

**ID: 10**

**Kort Beskrivelse:** En bruger sletter en RåvareBatch

**Primær Aktør:** Admin / Farmaceut / Værksfører

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at slette RåvareBatch og den RåvareBatch der ønskes slettet er ikke i brug

### **Primært flow:**

1. Bruger logger ind
2. Bruger trykker på knappen "RåvareBatch"
3. Bruger trykker på knappen "Rediger RåvareBatch"
4. Bruger vælger en RåvareBatch ud fra ID'erne i drop-down listen
5. Bruger trykker på knappen "Slet" og bliver sendt tilbage til brugerens menu
6. Bruger trykker på knappen "Log ud"

**Efterfølge:** RåvareBatch er nu slettet og brugeren er logget ud

### **Alternative Flows:**

#### **råvare ID i brug i en anden tabel**

1. Bruger logger ind
2. Bruger trykker på knappen "RåvareBatch"
3. Bruger trykker på knappen "Rediger RåvareBatch"

4. Bruger vælger en RåvareBatch ud fra ID'erne i drop-down listen
5. Bruger trykker på knappen "Slet"
6. Bruger får en fejlbesked om at RåvareBatch der ønskes slettet, er i brug i en anden tabel og kan derfor ikke slettes

**Efterfølge:** Brugeren kan slette en anden RåvareBatch der ikke er i brug eller navigere ud af systemet

## **Se Alle Råvarebatches**

**ID:** 11

**Kort Beskrivelse:** Bruger vil se en liste over alle råvarebatches

**Primær Aktør:** Admin / Farmaceut / Operatør

**Sekundære Aktører:** Database

**Forudsættelser:** Databasen indeholder råvarer og råvarebatches

### **Primært flow:**

1. Bruger logger ind
2. Bruger klikker på “Råvarerbatch” knappen
3. Bruger klikker på “Se Råvarebatches” knappen

**Efterfølge:** Alle råvarebatches vises i en tabel



## **Opret Recept**

**ID: 12**

**Kort Beskrivelse:** En bruger opretter en ny recept

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at oprette en recept

### **Primært flow:**

1. Bruger logger ind
2. Bruger trykker på knappen "Recepter"
3. Bruger trykker på knappen "Tilføj Recept"
4. Bruger indtaster Recept ID og recept navn
5. Bruger trykker på knappen "Opret" og bliver sendt tilbage til brugerens menu
6. Bruger trykker på knappen "Log ud"

**Efterfølge:** En ny recept er nu oprettet og brugeren er logget ud.

### **Alternative Flows:**

#### **Receptens ID eksistere allerede**

1. Bruger logger ind
2. Bruger trykker på knappen "Recepter"
3. Bruger trykker på knappen "Tilføj Recept"
4. Bruger indtaster et allerede eksisterende recept ID og et recept navn

5. Bruger trykker på knappen "Opret"
6. Bruger får en fejlmeddelelse om at det indtastede recept ID allerede er oprettet

**Efterfølge:** Brugeren kan i stedet oprette en recept med et tilgængeligt ID eller logge ud.

#### **Indtastet recept ID er ikke et tal**

1. Bruger logger ind
2. Bruger trykker på knappen "Recepter"
3. Bruger trykker på knappen "Tilføj Recept"
4. Bruger indtaster et recept ID der indeholder bogstaver og et recept navn
5. Bruger trykker på knappen "Opret"
6. Bruger får en fejlmeddelelse om at det indtastede recept ID skal være et tal.

**Efterfølge:** Brugeren kan skrive et gyldigt ID, som er et tal, eller logge ud

## **Opdater Recept**

**ID: 13**

**Kort Beskrivelse:** En bruger opdaterer en recept

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at opdatere en recept

### **Primært flow:**

1. Bruger logger ind
2. Bruger trykker på knappen "Recepter"
3. Bruger trykker på knappen "Rediger Recept"
4. Bruger vælger den recept der skal opdateres fra drop-down listen
5. Bruger indtaster et nyt navn for recepten
6. Bruger trykker på knappen "Opdater" og bliver sendt tilbage til brugers menu
7. Bruger trykker på knappen "Log ud"

**Efterfølge:** Recepten er nu opdateret i systemet

## Slet Recept

**ID:** 14

**Kort Beskrivelse:** Bruger sletter en recept

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at slette en recept og den recept der ønskes slettet er ikke i brug i andre tabeller

### Primært flow:

1. Bruger logger ind
2. Bruger trykker på knappen "Recepter"
3. Bruger trykker på knappen "Rediger Recept"
4. Bruger vælger den recept der skal slettes fra drop-down listen
5. Bruger trykker på knappen "Slet" og bliver sendt tilbage til brugerens menu
6. Bruger trykker på knappen "Log ud"

**Efterfølge:** Recepten er nu slettet fra systemet og brugeren er logget ud

### Alternative Flows:

#### recepten er i brug

1. Bruger logger ind
2. Bruger trykker på knappen "Recepter"
3. Bruger trykker på knappen "Rediger Recept"

4. Bruger vælger den recept der skal slettes fra drop-down listen
5. Bruger trykker på knappen "Slet"
6. Bruger får en fejlbesked om at recepten der ønskes slettet, er i brug i en anden tabel og kan derfor ikke slettes

**Efterfølge:** Recepten er ikke slettet, men brugeren kan vælge at slette en anden recept der ikke er i brug eller navigere ud af systemet

## **Se Alle Recepter**

**ID:** 15

**Kort Beskrivelse:** Bruger tjekker Recepter

**Primær Aktør:** Admin / Farmaceut

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at se recepter

### **Primært flow:**

1. Bruger logger ind
2. Bruger trykker på knappen "Recepter"
3. Bruger trykker på knappen "Se Recept"

**Efterfølge:** Tabellen med recepter vises

## **Tilføj Produktbatch**

**ID:** 16

**Kort Beskrivelse:** En bruger tilføjer en produktbatch

**Primær Aktør:** Admin / Farmaceut / Værksfører

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at oprette produktbatches

### **Primært flow:**

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Tilføj Produktbatch"
4. Bruger indtaster recept ID
5. Bruger trykker på knappen "Opret" og bliver sendt tilbage til brugerens menu
6. Bruger trykker på knappen "Log ud"

**Efterfølge:** En ny produktbatch er oprettet og brugeren er logget ud.

### **Alternative Flows:**

#### **Ikke eksisterende recept ID**

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Tilføj Produktbatch"
4. Bruger indtaster et recept ID som ikke eksisterer

5. Bruger trykker på knappen "Opret"
6. Bruger får en fejlbesked om at ID'et ikke eksisterer og bliver rykket tilbage til hans menu

**Efterfølge:** Brugeren kan vælge prøve igen fra menuen eller logge ud.

### **Indtastning af ID der ikke er kun et tal**

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Tilføj Produktbatch"
4. Bruger indtaster et recept ID som ikke er et tal
5. Bruger trykker på knappen "Opret"
6. Bruger får en fejlbesked om at ID'et skal være et tal

**Efterfølge:** Brugeren kan prøve igen med et rigtigt ID eller gå tilbage.



## Opdater ProduktBatch

**ID: 17**

**Kort Beskrivelse:** En bruger opdaterer en produktbatch

**Primær Aktør:** Admin / Farmaceut / Værksfører

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at oprette produktbatches

### Primært flow:

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Rediger Produktbatch"
4. Bruger vælger en produktbatch ud fra ID'erne i drop-down listen
5. Bruger indtaster det recept ID, som det gamle recept ID skal ændres til
6. Bruger trykker på knappen "Opret" og bliver sendt tilbage til brugerens menu
7. Bruger trykker på knappen "Log ud"

**Efterfølge:** En ny produktbatch er oprettet og brugeren er logget ud

### Alternative Flows:

#### Ikke eksisterende recept ID

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"

3. Bruger trykker på knappen "Tilføj Produktbatch
4. Bruger indtaster et recept ID som ikke eksisterer
5. Bruger trykker på knappen "Opret"
6. Bruger får en fejlbesked om at ID'et ikke eksisterer og bliver rykket tilbage til hans menu

**Efterfølge:** Brugeren kan prøve igen med et gyldigt ID eller logge ud.

#### **Indtastning af ID der ikke kun er tal**

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Tilføj Produktbatch
4. Bruger indtaster et recept ID som ikke er et tal
5. Bruger trykker på knappen "Opret"
6. Bruger får en fejlbesked om at ID'et skal være et tal

**Efterfølge:** Brugeren kan prøve igen med et rigtigt ID eller gå tilbage

## Slet ProduktBatch

**ID: 18**

**Kort Beskrivelse:** En bruger sletter en produktbatch

**Primær Aktør:** Admin / Farmaceut / Værksfører

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at slette produktbatches og den produktbatch der ønskes slettet er ikke i brug

### Primært flow:

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Rediger Produktbatch"
4. Bruger vælger en produktbatch ud fra ID'erne i drop-down listen
5. Bruger trykker på knappen "Slet" og bliver sendt tilbage til brugerens menu
6. Bruger trykker på knappen "Log ud"

**Efterfølge:** Produktbatchen er nu slettet og brugeren er logget ud

### Alternative Flows:

#### recept ID i brug i en anden tabel

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Rediger Produktbatch"

4. Bruger vælger en produktbatch ud fra ID'erne i drop-down listen
5. Bruger trykker på knappen "Slet"
6. Bruger får en fejlbesked om at produktbatchen der ønskes slettet, er i brug i en anden tabel og kan derfor ikke slettes

**Efterfølge:** Brugeren kan slette en anden produktbatch der ikke er i brug eller navigere ud af systemet

## **Se Alle Produktbatches**

**ID:** 19

**Kort Beskrivelse:** En bruger tjekker produktbatches

**Primær Aktør:** Admin/Farmaceut/Værksfører

**Sekundære Aktører:** Database

**Forudsættelser:** Brugeren har rettigheder til at tjekke produktbatches

### **Primært flow:**

1. Bruger logger ind
2. Bruger trykker på knappen "Produktbatch"
3. Bruger trykker på knappen "Se Produktbatches"

**Efterfølge:** Tabellen med produktbatches vises

## **Log Ud**

**ID:** 20

**Kort Beskrivelse:** En bruger logger ud

**Primær Aktør:** Admin / Farmaceut / Værksføre

**Sekundære Aktører:**

**Forudsættelser:** Bruger er logget ind

**Primært flow:**

1. Bruger klikker på ”Log ud”knappen

**Efterfølge:** Brugeren er nu logget ud

**Alternative Flows:**

## **Log Ind**

**ID: 21**

**Kort Beskrivelse:** En bruger logger ind

**Primær Aktør:** Admin / Farmaceut / Værksføre

**Sekundære Aktører:** Database

**Forudsættelser:** Bruger har et bruger ID og en godkendt kode, som er gemt i databasen

### **Primært flow:**

1. Bruger indtaster sit bruger ID
2. Bruger indtaster sin kode

**Efterfølge:** Brugeren er nu logget ind

## **Lav Afvejning**

**ID: 22**

**Kort Beskrivelse:** Bruger laver en afvejning

**Primær Aktør:** Admin / Farmaceut / Værksfører / Operatør

**Sekundære Aktører:** Database

**Forudsættelser:** Der er oprettet et produktbatch

### **Primært flow:**

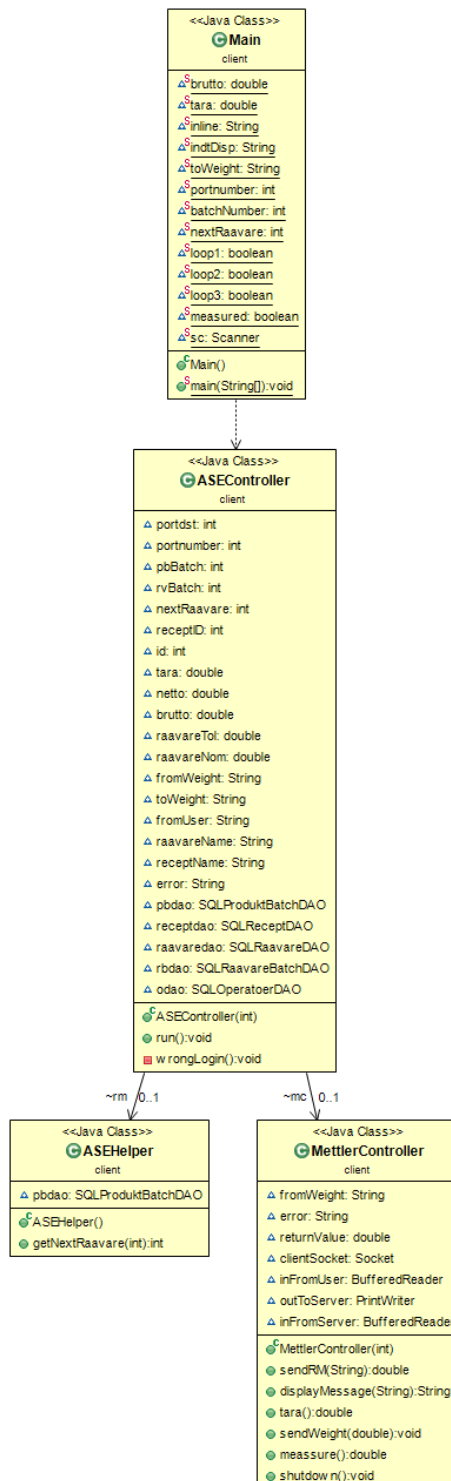
1. Bruger identificerer sig over for vægt
2. Bruger indtaster produktbatchnummer
3. Bruger tjekker at der ikke står noget på vægten
4. Bruger sætter en tarabeholder på vægten (indtaster i konsollen)
5. Bruger indtaster råvarebatchnummer
6. Bruger passende mængde af den pågældene råvare på vægten (indtaster i konsollen)
7. Bruger får at vide at vægten er registreret

**Efterfølge:** Den første receptkomponent i recepten for produktbatchen er nu registreret

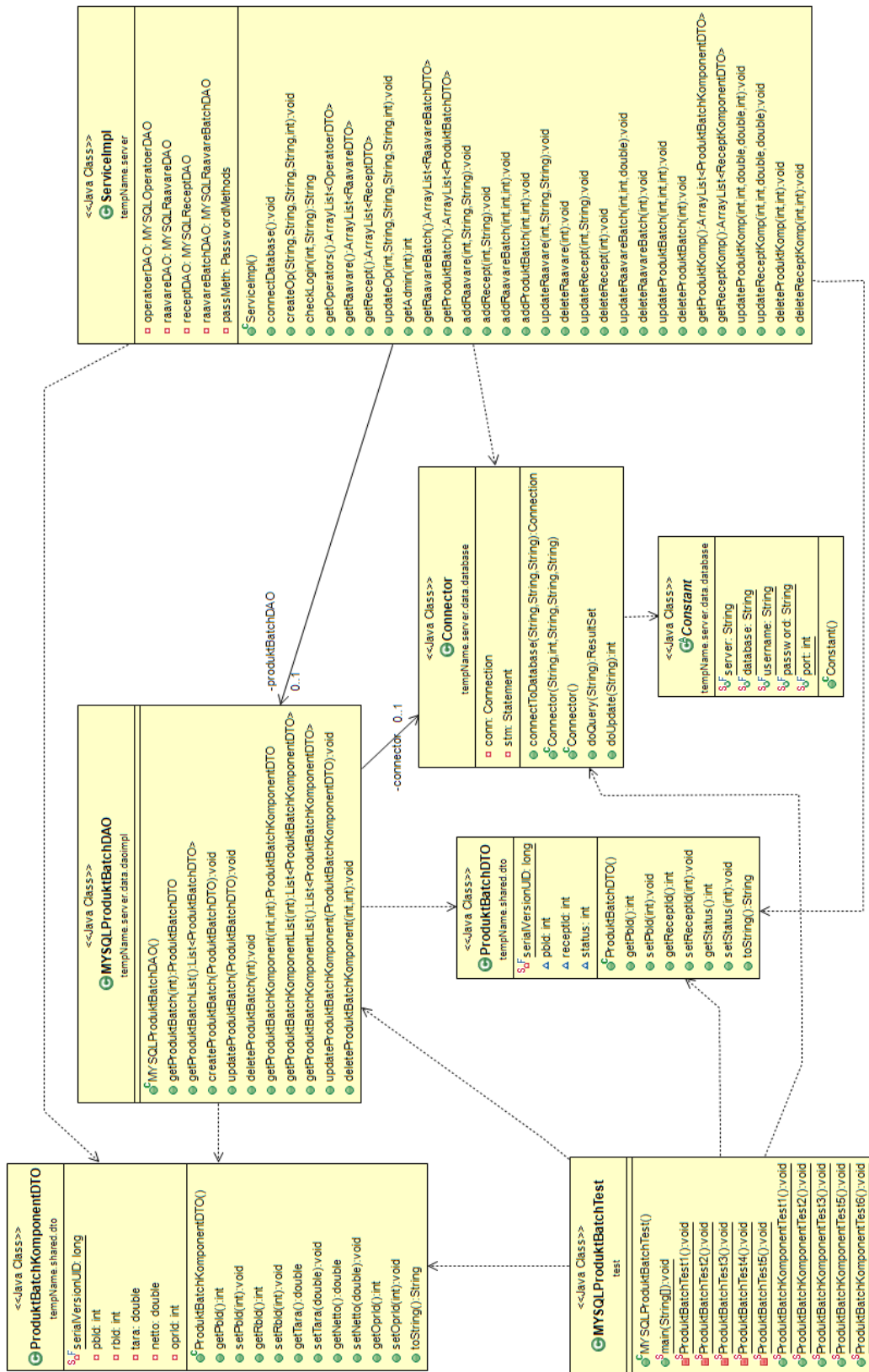
### **Alternative Flows:**

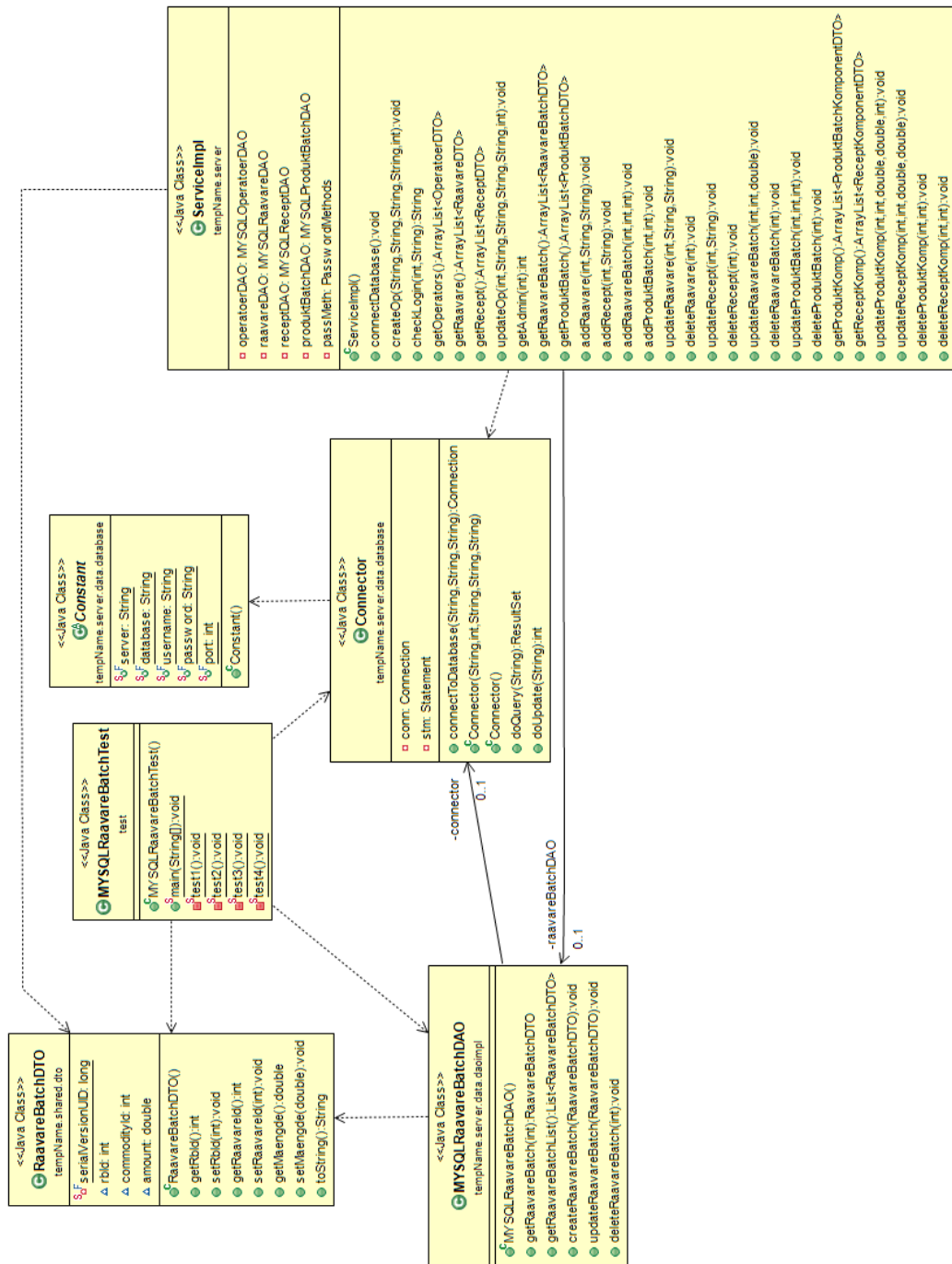
1. Bruger indtaster noget forkert/et batchnummer der ikke stemmer overens/en vægt der ikke ligger inden for tolerance
2. Bruger bliver bedt om at indtaster den forkerte indtastning igen





ASE Klassediagram







## EER Diagram

