CDIO FINAL (Sommer 2018)

02324 Videregående programmering

15. Juni 2018

Mathias
Fager
s175182

Niklaes
Jacobsen
s160198

Sebastian
Stokkebro
s170423

Simon
Pedersen
s175195

Burhan
Shafiq
s175446

Contents

Tidsforbrug

Herunder ses vores tidsforbrug på projektet, opdelt pr. pers. angivet i hele timeantal

Navn	Uge 21	Uge 22	Uge 23	Alle uger samlet	
	Totalt	Totalt	Totalt	Totalt	Snit pr. dag
Burhan Shafiq	20	39,75	6,5	66,25	6,022727273
Mathias Fager	19,5	45,75	10,5	75,75	6,886363636
Niklaes Jacobsen	19,5	65	0	84,5	7,681818182
Sebastian Sørensen	19,5	46,75	8	74,25	6,75
Simon Pedersen	20	47,42	8	75,42	6,856363636
Samlet	98,5	244,67	33	376,17	34,19727273

1 Introduktion

I dette projekt vil vi beskæftige os med at udvikle et system, der skal håndtere afvejning af råvarer og produktionsprocessen af et produkt. Systemets primære prioritet ligger i at gemme data for afvejningerne og produktionerne, så de senere kan vises og printes. Frontend delen skal bestå af et webinterface, hvor brugeren har mulighed for at vælge sin rolle, og efter det kan udføre de tilhørende scenarier. Backend delen vil være en MySQL database, hvor data lagres, og kan ændres. Vi vil prioritere at køre systemet på en Heroku server, således det er let at tilgå alle steder fra.

Backend delen er REST-baseret og ved hjælp af HTTP forespørgsler, vil det være muligt af modificere data på serveren gennem webinterfacet.
Webinterfacet består af en række HTML-sider, der linkes til afhængigt af, hvad der navigeres til. Der bruges CSS til styling af siderne og Javascript (JQuery) til funktionaliteterne. Ved hjælp af Ajax vil vi kunne bruge Java-metoderne fra funktionalitetslaget og på den måde modificere og modtage data fra vores backend.

2 Analyse

2.1 Kravspecifikation

2.1.1 Backend

- Krav til brugere:

  En bruger defineres ud fra ID, navn, initialer.

  - Bruger-ID skal være unikt og må gerne autogenereres.

  Administratoren skal kunne:

  - Oprette, redigere, vise og fjerne (deaktivere) en bruger.
  - Må ikke kunne deaktiveres

  Farmaceut skal kunne:

  - Oprette, redigere og vise en råvare.
  - Farmaceut foretager administration af recepter (Oprette og vise recepter).

- En Produktionsleder skal kunne:
  - Foretage administrationen af råvarebatches og produktbatches.
- En råvare defineres ud fra et råvare ID, navn og leverandør.

  Råvare-ID skal være unikt og skal ikke autogenereres
- Et Produktbatch defineres ud fra et ProduktBatchNr, oprettelses dato og et batch status.

  Produktbatch statuser kan være en af følgende: oprettet / under produktion / afsluttet.

  Produktbatch ID er unikt, og skal være nr. på den recept produktbatchen skal laves udfra.
- Recept defineres ud fra recept nummer, navn, samt sekvens af receptkomponenter.

  Recept nummer er entydigt (unikt) og bruger defineret (Ikke auto-genereret)
- Receptkomponent defineres ved en råvare type, en mængde og en tolerance.
- Et råvarebatch er defineret ud fra:

  RåvarebatchNR, dette skal være unikt.
- Laboranten tilgår vægten via sit bruger ID.
- Afvejningsresultaterne for de enkelte receptkomponenter skal gemmes som en produktbatchkomponent i produktbatch.
- Produktbatchen har en status der afspejler receptens afvejningstilstand.
- Krav til systemet:

  Systemet skal håndtere input-validering

  Systemet skal sikre at afvejningsproceduren er intuitiv og fejltolerant

  Systemet tilgås ved at en laborant indtaster sit bruger ID.
- Krav til datalaget:

  Datalaget i applikationen kan f.eks. implementeres som en MySql database.

  Data access lag adskiller databasen fra andre komponenter i system arkitekturen.

  Ved fejl i data access laget kastes en exception der beskriver fejlen til de øvre lag.
- Vægtens afvejningsprocedure skal være styret af en Afvejnings-styringsenheden (*ASE*).
- Der skal tages udgangspunkt i den fysiske vægt

2.1.2 Frontend

- Krav til Web applikationen:

  Forsiden skal indeholde tre knapper med valg af rolle.

  Får produktionslederen har oprettet et nyt produktbatch skal outputtet vises så der er mulighed for at printe det.

  Web applikationen skal implementere bruger-, råvare-, recept-, råvarebatch og produktbatch-administraton.

  Alle form input felter skal valideres iht. gyldige områder og der skal vises passende fejlmeddelelser ved fejl.

2.2 Use-cases

- Administratoren:

  Opretter en bruger.

  Redigerer en bruger

  Deaktiverer/aktiverer en bruger.

- Farmaceuten:

  Opretter en recept/recept komponent

  Opretter en råvare

  Redigerer en recept/recept komponent

  Redigerer en råvare

  Sletter en recept/recept komponent

  Sletter en råvare

- Produktionslederen:

  Opretter et råvarebatch

  Opretter et produktbatch/produktbatch komponent

  Redigerer et råvarebatch

  Redigerer et produktbatch/produktbatch komponent

  Sletter et råvarebatch

  Sletter et produktbatch/produktbatch komponent

- Laboranten:

  Indtaster ID

  Indtaster produktbatch ID på det batch der skal produceres

  Afvejer tara

  Indtaster råvarebatch ID for afvejningen

  Afvejer råvarebatch mængde (overholder tolerance angivet i recept komponent)

3 Design

3.1 Overvejelser af design

Ud fra vores design overvejelser har vil lavet følgende oversigts-diagram. Diagrammet visualiserer tydeligt hvordan de forskellige lag i arkitekturen vil arbejde sammen mellem backend og frontend. Først og fremmest kan det ses, hvordan backend delen bl.a. skaber views og transactions fra databasen med SQL Queries, som bliver forbundet til Java-laget ved hjælp af JDBC-driveren. Efter der er opnået adgang til dataen, kan funktionaliteten benytte sig af denne. Det kan samtidig også ses at web-delen får adgang til funktionaliteten gennem HTTP og REST. Ligeledes kan man også se at vægten får adgang til funktionaliteten gennem TCP og en socket, der skabes i Java.

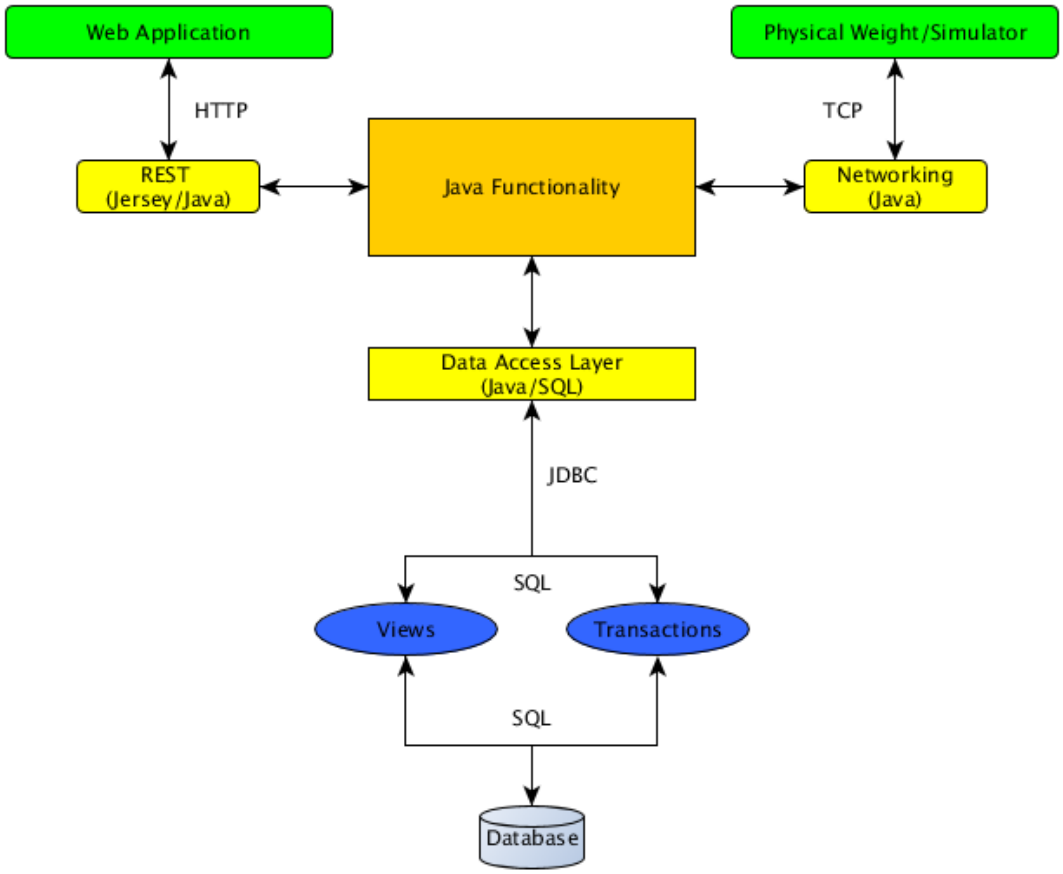


Figure 1: Oversigt over systemet

3.1.1 Programmets Arkitektur

Vi har lagt meget fokus på arkitekturen af vores program, således overskueligheden af programmet bliver optimalt mest muligt. For at opnå dette mål opbygges projektet efter 3-lags modellen for at kunne skabe 3 uafhængige dele af vores program. Dette vil hjælpe os med at få sikret et program, der har mindre risiko for fej og gøre det lettere for kunden evt. at udskifte/erstatte elementer af programmet med andet. For at sikre os at diverse elementer let kan udskiftes vil vi benytte os af interfaces, der vil kunne bruges som en slags 'opskrift' til hver del. Dette vil gøre det lettere at kunne udvikle en alternativ logik-klasse til programmet uden at skulle justere alle andre dele af programmet til at kunne være kompatible med den nye klasse. Derudover vil vi også benytte os af BCE til at uddelegere og fordele ansvaret i mellem klasserne og derved skabe en høj kohæsion blandt de forskellige klasser og dele af programmet.

3.1.2 GRASP

I et forsøg på at skabe en effektiv og overskuelig kode anvendes GRASP principperne. Principperne hjælper med at belyse diverse arbejdsområder for hver klasse, således ansvar kan uddelegeres derefter. Det kan ses i det følgende klasse-diagram, at klasser er afgrænset i en stream-lined manér, hvilket styrker klasser minimale interaktion med andre, der ikke er ønsket. I denne stream-lined opbygning er DTO'erne, altså informationen, isoleret, og bliver igennem vores DAO skabt. Metoderne heri bliver da anvendt i en række Controllers, der hver især står for at oversætte og kommunikere med REST'en. Envidere er et lag af interfaces brugt til at øge genbrugeligheden samt øge venligheden af opretholdelse af systemet. Ved at introducere disse, gives et blue-print på, hvilke metoder klasserne skal indeholde, som skaber overblik og giver let tilgængelighed. Derudover er en række overvejelser af opbygningen af diverse klasser gjort i et forsøg på at danne et stykke kode, hvori fokus har været på at bevare High Cohesion og Low Coupling igennem hele forløbet.

3.2 Designklassediagram

Nedenfor ses Klassediagrammet over systemet, som, grundet redundans af klasser, kun tager udgangspunkt i et generelt scenario.

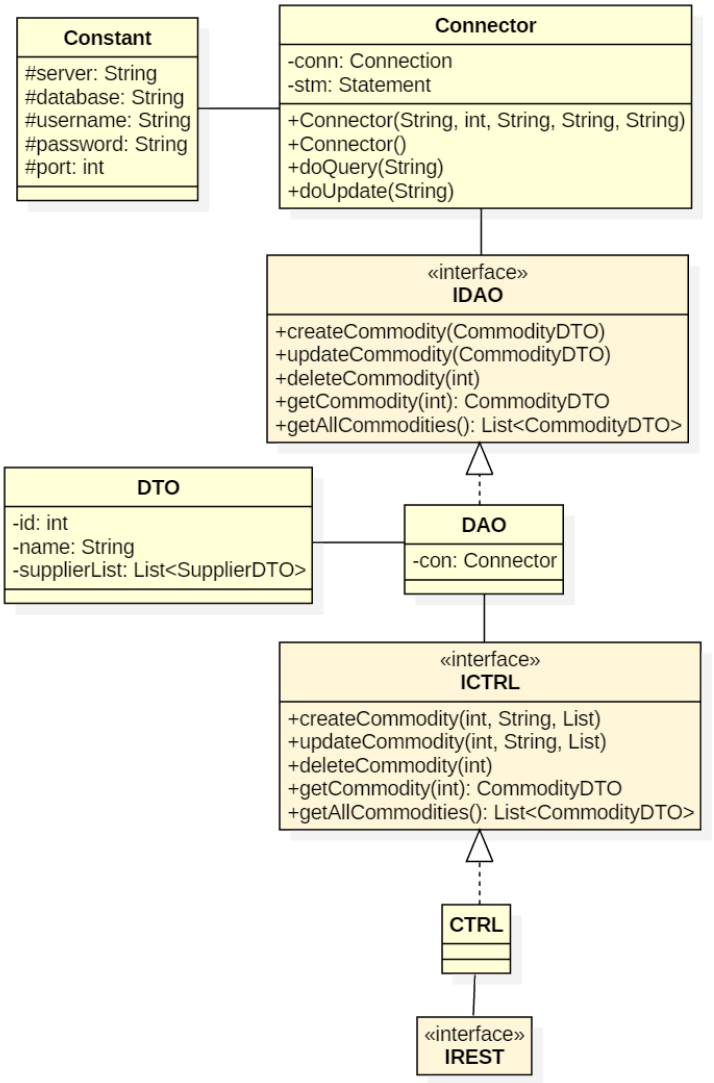


Figure 2: Designklassediagram

4 Implementation

4.1 Gennemgang af kode

4.2 Klasseopbygning / hvem laver hvad / hvilken model har vi brugt - Skal ikke være en fed tekst

Til opgavebeskrivelsen fulgte nogle foreslag til hvordan vi kunne opsætte databasen, samt opsætningen af disse DTO'er og DAO'er som gør programmet i stand til at danne objekter af de forskellige dele, samt kunne indsætte og udtrække dataen fra en database. Vi valgte at gemme data mellem sessioner i en SQL database, so skal kunne hostes på nettet og dette gør det muligt at programmet altid kan tilgå databasen og det ikke kun er lokalt. Udover de udleverede DTO'er DAO'er valgte vi at lave en DAO og DTO til leverandørne, så det er nemt at have flere leverandøer til en enkelt råvare i vores Java system. Derudover har vi haft stort fokus på at kunne have en ekstern database, koblet til en Java server som skal gøre det muligt at programmet kører via nettet, og sidst men ikke mindst har vi valgt at bruge et 4G modem til vægten, som smider den på nettet også. Dette gør det altså muligt at hele systemet er uafhængigt af hinanden, i den form at de 3 dele, vægten, programmet og databasen, ikke skal direkte forbindes via kabler men data kan overføres via nettet. Databasen og Java laget bliver forbundet med en JDBC (Java DataBase Connectivity), som gør det muligt at sende queries fra Java ned i databasen og eventuelt danne et ResultSet af dem, som kan bruges til at fører data, såsom integers og strings, ind i Java laget hvor vi kan bearbejde dataen. Java og web-siden kommunikere via REST, hvor vi sender JSON objekter fra Java til JavaScriptet og her er vi i stand til at bearbejde og fremvidst data, eksempeltvis vis en bruger skal fremvises og rettes. Vi har til dette program sigtet efter at bruge 3-lags modellen, som går ud på at have et data lag i bunden som sørger for at opbevare data og danne objekterne af de ting vi har med at gøre. Ovenpå data laget lægger logiklaget (funktionalitetslag/controller lag) som sørger for at udveklse og bearbejde den data som sendes til brugergrænsefladen som er det lag som bliver præsenteret til brugeren af systemet. Af andre ting vi har gjort som gør vores program smart, er at vi har lavet en specifik klasse med metoder som gør det meget nemt at kommunikere med vægten. Når en besked sendes til vægten, skal denne besked sendes med en form for kode der gør at vægten ved hvordan den skal håndtere denne besked. Om den bare skal vises i en given tid, om der skal et svar tilbage osv., vægten har nemlig mange funktioner som vi kan gøre brug af i vore system. Eksempeltvis så hvis en besked skal sendes til vægten, og der forventes at svar, kræver det at vægten modtager kommandoen "RM20 3 "" "" "" ", hvor de 3 strings vises forskelligt på vægten. Vi har derfor lavet en metode, getMessageWithInput, som kaldes og de 3 strings er så parametrene, så sørger metoden selv for at sende beskeden rigtigt afsted, samt håndtere det svar som kommer tilbage, da disse også kan være forskellige alt efter om det gik godt, hvad input er osv.
I tilfælde af at en fejl sker i programmet og der returneres en exception, har vi sørget for at brugeren af vægten for en passende besked så brugeren ved hvad der er gået galt. Eksempeltvis så hvis en bruger indtaster et forkert ID ét af de steder, hvor vi bruger ID, så modtager brugeren af vægten beskeden "Forkert input prøv igen", hvorefter brugeren bliver ført tilbage til beskeden hvor han skal indtaste ID.

5 Dokumentation

I programmet bruges Java-doc til at dokumentere hvad de forskellige metoder gør, dette hjælper kundens fremtidige programmører, og ikke mindst os selv, til at huske og finde ud af hvordan programmets metoder er sat sammen. Til alle metoderne følger derfor en lille besked som fortæller hvordan metoden bruges, samt hvad eventuelle parametre betyder, akkurat ligesom Java udviklerne har gjort med de integreret metoder i Java.

6 Konfiguration

Vi har brugt følgende software til at programmere, samt planlægge, vores software:

- Styresystemer
  - Windows 10
  - Ubuntu (Linux-kerne)
  - Mac OS
- Programmering
  - Eclipse JEE
  - GitHub
  - Maven
  - JDK 1.8
- Diagrammer
  - yEd
  - StarUML
- Rapport
  - shareLatex
- Planlægning
  - Asana
  - Google Sheets
- Opbevaring / server
  - Google Drive
  - Heroku

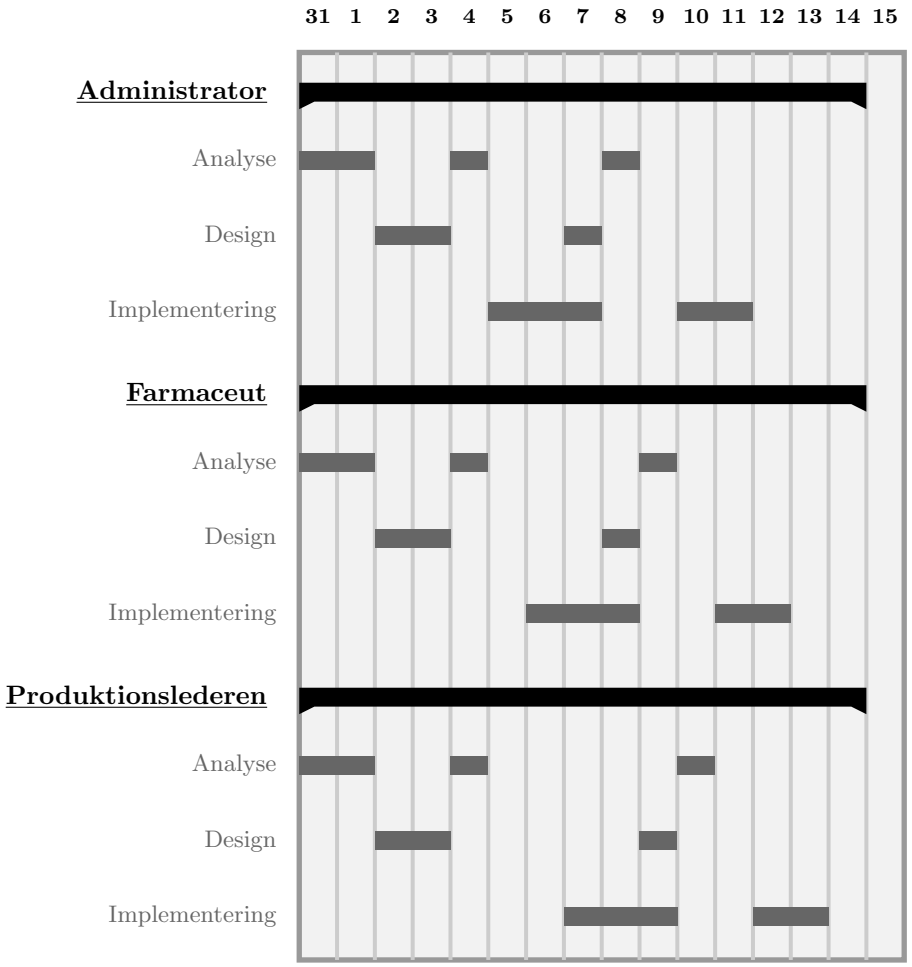
6.1 Kørsel af program

For at få kørsel af programmet til at køre mere gnidningsfrit, bliver vores projekt kørt på en ekstern server. Denne server (https://www.heroku.com) kan køre java-kode, og kan også få forbindelse med vores SQL server. Behandling af data sker over hjemmesiden, der altid vil kunne tilgås via linket:

https://cdiofinal12.herokuapp.com

7 Projektplanlægning & Prioritering - pls no waterfall

Vi har lavet vores planlægning for projektet med hjemmesiden asana.com, og har derefter taget et usecase og fået det implementeret. Vi startede med det grundlæggende i programmet, som var databasen. Vi lavede et udkast til den, så vi vidste hvor dataen skulle hentes fra når du lavede DAO'erne. DTO'erne var mere eller mindre givet til os fra kunden, så mens databasen blev designet og oprettet, blev DTO'erne lyn hurtigt også oprettet. Da vi havde nogle objekter at danne, gik vi i gang med DAO'erne, som forbinder databasen med DTO'erne. Databasen blev lavet om nogle gange, da den ikke gjorde hvad vi ønskede. Dernæst splittede vi gruppen, hvor 1 gik i gang med tests af WeightTranslation, DAO'er osv. 2 gik i gang med WeightController som står for flowet og de 2 sidste gik i gang med REST, HTML, JavaScript og så videre. Det var vigtigt for os at have alt den data til rådighed som blev brugt længere oppe i programmet, derfor startede vi fra bunden og byggede os op. Vi har taget udgangspunkt i usecases i diagrammet herunder.



8 Test - Mangler mere tekst

8.1 JUnit test

For at sikre os at programmet ikke pludseligt gør noget uforventet, har vi ved hjælp af JUnit tests sikret os at tingene fungere som forventet.
Dette betyder blandt andet at klassen som komminukerer med vægten er blevet testet så vi er sten sikre på at metoderne gør som forventet. Derudover har vi testet alle DAO'erne da det er vigtigt at de objekter vi danner ud fra data fra databasen, er rigtigt konstrueret så ikke det forkerte produktbatch får den forkerte varer osv.

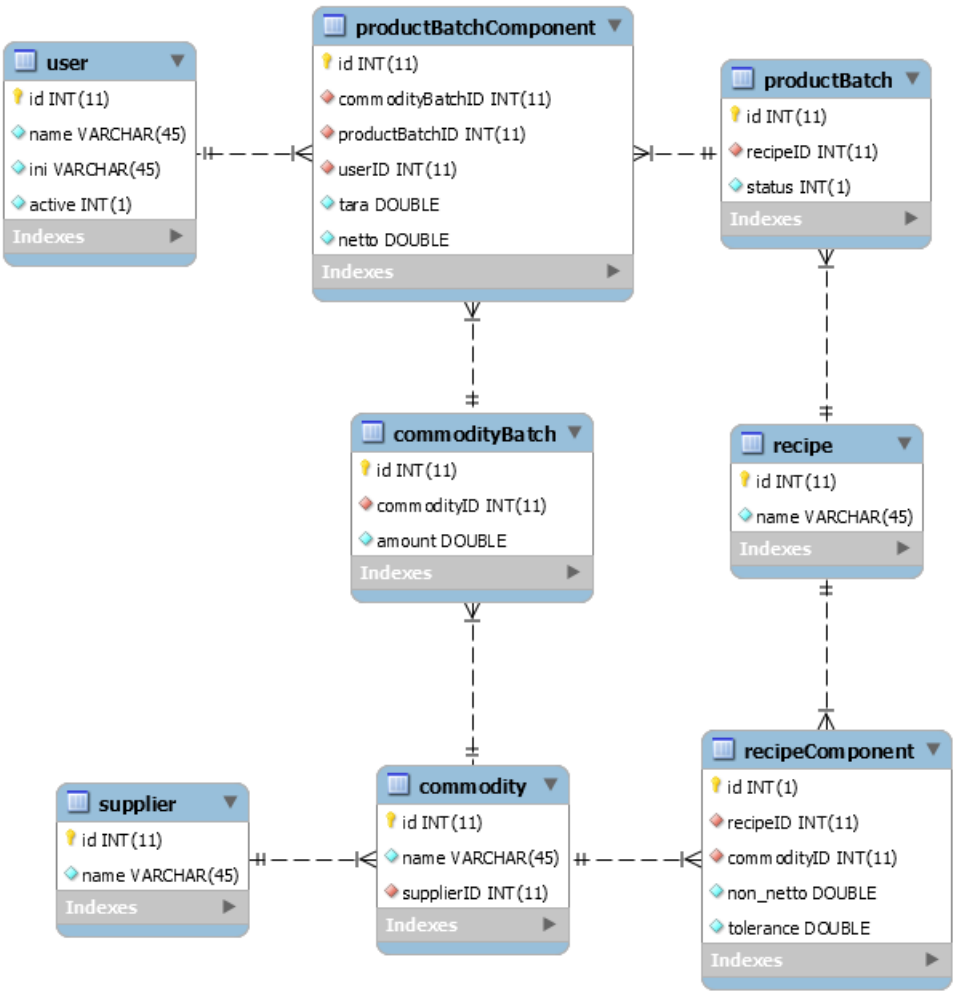
8.2 Brugertest

Vi har bruger testet programmet ud fra vores use-cases for at sikre os at slutbrugeren ikke løber ind i problemer vi ikke har taget hånd om. Det er vigtigt at brugeren kan udføre det flow som kunden har beskrevet, samt at hvis der skulle ske en fejl, så er slutbrugeren eller programmet i stand til at komme ind på rette spor igen. Dette betyder blandt andet at vi har sørget for at flowet er dynamisk, i form at brugeren kan gå tilbage til tidligere skridt og ændre på den data der er blevet indtastet.

9 Konklusion

Vi kan hermed konkludere at vi har lavet et program som kan implementeres ude hos kunden, efter de kravspecifikationer som kunden har ønsket. Vi har udover de grundlæggende krav bygget ovenpå med en online SQL server, samt smidt både Java delen og vægten på nettet, hvilket betyder det hele er cloud baseret og det eneste hardware der skal opereres er vægten selv, af en laborant. Til programmet fører en hjemmeside, hvor der kan logges på som den rolle en person har, og derefter få muligheder til at administrerer brugere, hvis du er bruger administrator, eller administrerer produkter, recepter osv., hvis du er produktionsleder m.m. Flowet med vægten fungerer som aftalt, hvor brugervenligheden er helt i top i den facon at, hvis en bruger skulle komme til at taste noget forkert ind på vægten, så er systemet dynamisk i form at brugeren kan gå tilbage til et tidligere step, uden at skulle starte forfra. Vi har desuden testet det meste af programmet, både med brugertests og JUnit tests for at vi sikre os at programmet fungerer som planlagt, og vi ikke har misforstået hvordan vi skal kommunikere med vægten.

10 Bilag


user
id INT(11)
name VARCHAR(45)
ini VARCHAR(45)
active INT(1)
Indexes
productBatchComponent
id INT(11)
commodityBatchID INT(11)
productBatchID INT(11)
userID INT(11)
tara DOUBLE
netto DOUBLE
Indexes
productBatch
id INT(11)
recipeID INT(11)
status INT(1)
Indexes
commodityBatch
id INT(11)
commodityID INT(11)
amount DOUBLE
Indexes
recipe
id INT(11)
name VARCHAR(45)
Indexes
supplier
id INT(11)
name VARCHAR(45)
Indexes
commodity
id INT(11)
name VARCHAR(45)
supplierID INT(11)
Indexes
recipeComponent
id INT(1)
recipeID INT(11)
commodityID INT(11)
non_netto DOUBLE
tolerance DOUBLE
Indexes