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

Gruppe 12

Antal sider i alt: 17

Timeregnskab

For yderligere detaljer vedrørende timeregnskab se bilag 10.5.

Navn	Uge 21	Uge 22	Uge 23	Alle uger samlet	
	Totalt	Totalt	Totalt	Totalt	Snit pr. dag
Burhan Shafiq	20	39,75	42,25	102	6,8
Mathias Fager	19,5	45,75	41,34	106,59	7,106
Niklaes Jacobsen	19,5	65	45,5	130	8,666666667
Sebastian Sørensen	19,5	46,75	39	105,25	7,016666667
Simon Pedersen	20	47,42	43,5	110,92	7,394666667
Samlet	98,5	244,67	211,59	554,76	36,984

Brugervejledning

Programmet initialiseres ved at tilgå følgende hjemmeside: http://cdiofinal12.herokuapp.com

Her vil der baseret på ens formål være en række brugerscenarioer. Er formålet at afveje et nyt produkt batch komponent, skal der oprettes forbindelse til en fysisk vægt via ruter med statisk IP adresse. For at udføre afvejninger, trykkes der på feltet "VÆGT" øverst i højre hjørne, hvorefter der kan oprettes en forbindelse til en af de to givne vægte, via hhv. port 8000 eller port 8001. Dette vil starten vægten, og brugeren bedes verificere sit id overfor vægten, via dets tastatur. Herfra bedes brugeren følge vejledningen vist på vægtens display.

Er ens formål i stedet at administrere brugere, skal der i stedet trykkes på "ADMINISTRATOR" feltet, hvorefter en ny side vil blive vist. Her vil brugeren have muligheden for at oprette en ny bruger, finde og redigere en bruger samt få vist alle i systemet.

Ønsker en bruger at tilgå farmaceut relateret opgaver, trykkes der på feltet "FARMACEUT" i højre hjørne. Brugeren vil blive ført til en ny side, heri er det muligt at oprette elementer via opret feltet, få vist enkelte elementer via Find og få vist en liste over alt indhold af et specifikt element. Tilsvarende gøres gældende for tilfældet af produktionsleder relateret arbejde, her trykkes blot på "PRODUKTIONSLEDER", hvorefter en magen til menu som for farmaceut vises.

For en dybere vejledning af hjemmesiden se bilag 11.2.

Contents

Abstract

We have been assigned by a pharmaceutical company the task to create a piece of software for their weighing system. Through an iterativ work process and an relativly agile structure we were able to detect errors and correct them in time and thereby avoid building a program with bugs. The program is deployed on a webserver capable of running Java code and stores data on a remote database. Through the associated webpage the users are able to interact with the database, by retriving information about the elements, and establish connection to a desired weight. The weight is connected to a router with a static IP address, meaning we can communicate with it remotely and feed data to the database only via it and the router. The webpage allows user to enter different sites such as a weight, production manager, pharmacist and an admin page where a user can manage other users. These functions are part of the desired requirements among many others which we have had the need to revisit throughout the work process to assure the program is in accordance to the customers wishes. The report includes illustrations of class diagrams aswell as the system, an overview of how the program was implemented and a documentation of the testing. Furthermore the report includes a discussion part where our acts and choices are debated and a derived conclusion we could conclude at the end of this project.

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 igennem REST, og på den måde modificere og modtage data fra vores backend.

2 Krav specifikation

2.1 Funktionelle Krav

- Brugerne er defineret som følgende:

  Selve brugeren defineres ud fra et ID, navn, initialer.

  - ID skal være unikt og må gerne autogenereres.

- En superbruger skal kunne tilgå tre forskellige roller:

  Administrator, med følgende funktioner:

  - Oprette, redigere, vise og deaktivere/aktivere en bruger.

  Farmaceut, med følgende funktioner:

  - Farmaceut foretager administration af recepter og råvarer.

  Produktionsleder, med følgende funktioner:

  - Foretage administrationen af råvarebatches og produktbatches.

- En råvare defineres ud fra et ID, navn og leverandør.

  ID skal være unikt og skal ikke autogenereres.

- Et Produktbatch defineres ud fra et ProduktBatchID og et batch status.

  Produktbatch status kan være en af følgende: oprettet(0) / under produktion(1) / afsluttet(2).

  ID er unikt, og skal være nr. på den recept produktbatchen skal laves udfra.

- Recept defineres ud fra recept ID, navn, samt sekvens af receptkomponenter.

  ID er unikt og skal ikke auto-genereres.

- Receptkomponent defineres ved en råvare type, en mængde og en tolerance.

  Afvejningsresultaterne for de enkelte receptkomponenter skal gemmes som en produktbatchkomponent i produktbatch.

- Råvarebatch er defineret ud fra et ID.

  ID er unikt.

- Vægtens afvejningsprocedure skal være styret af en Afvejnings-styringsenheden (ASE).

2.2 Ikke Funktionelle Krav

- Krav til systemet:

  Systemet skal håndtere input-validering på vægt.

  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.

- Krav til Web applikationen:

  Forsiden skal indeholde tre knapper med valg af rolle.

  For 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.

2.3 Nedprioriterede krav

Herunder ses en liste over krav vi, som ikke vil implementere til at starte med.

- Krav til systemet:

  Et Produktbatch defineres ud fra et ProduktBatchID, oprettelses dato og et batch status
  - Her har vi nedprioriteret oprettelses dato

- Produktbatch status kan være en af følgende: oprettet / under produktion / afsluttet.
  - Her har vi simplificeret løsningen til blot et tal: "0 / 1 / 2".

- Systemet skal håndtere input-validering på web.

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

3 Analyse

3.1 Use-cases

- Administratoren:
  - Administrer brugerer
    - Opret en bruger.
    - Rediger en bruger
    - Deaktiver / aktiver en bruger.
- Farmaceuten:
  - Administrer råvare
    - Opret en råvare
    - Rediger en råvare
  - Administrer recepter
    - Opret en recept / recept komponent
- Produktionslederen:
  - Administrer råvarebatch
    - Opret et råvarebatch
  - Administrer produktbatch
    - Opret et produktbatch / produktbatch komponent
- Laboranten:
  - Afvej

3.2 Casual beskrivelse af usecases

Laborant - Afvej
Laboranten vil først og fremmest blive spurgt om at indtaste sit operatør ID i vægten, hvorefter de vil modtage en velkomst besked fra vægten. Herefter vil Laboranten gå videre til at indtaste et produktbatch ID, for det produktbatch laboranten gerne vil redigere i. Laboranten vil derefter blive spurgt om at angive hvilken råvare batch han gerne vil ændre i ved at angive ID'et på det råvarebatch. Derefter vil laboranten blive spurgt om at afveje, diverse råvare i råvarebatchen, hvorefter de skal bekræfte at den registrerede vægt er korrekt. Efter laboranten er færdig med at afveje de forskellige råvare, vil laboranten afslutte processen og de forskellige ændringer der er vedtaget vil blive gemt i databasen.

4 Design

4.1 Overvejelser af design

Vi har gennem projektetsforløbet haft forskellige overvejelser omkring opsætning af programmet. Vi har haft mange forslag til opsætning af de forskellige controllers i programmet. En af vores første forslag var at opsætte en "super controller" der ville forbinde alle DAO'erne med REST, som derfra ville forbinde med hjemmesiden. Vi har også overvejet at lave en klasse der ville danne forbindelsen til databasen for alle DAO klasserne. Vi blev dog hurtigt enige om at disse forslag ville gøre programmet meget uoverskueligt og formindske kohæssionen i vores projekt. Vi blev derfor enige om at give hver DAO et flow der vil bearbejde det relevante information fra REST'en hele vejen ned til DAO'en og omvendt. Vi har også givet hver DAO sin egen instans af connectoren til at forbinde med databasen, så hver DAO har sin egen forbindelse.
Vi var allerede fra start enige om den helt overordnet opsætning af projektet samt, hvilke applikationer der skulle bruges til at forbinde de enkelte dele af projektet. Den opsætning, som vi havde i tankerne, benyttede vi os af, til at opsætte det nedenstående 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 bliver forbundet til Java-laget ved hjælp af JDBC-driveren. Efter der er opnået adgang til dataet, 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.

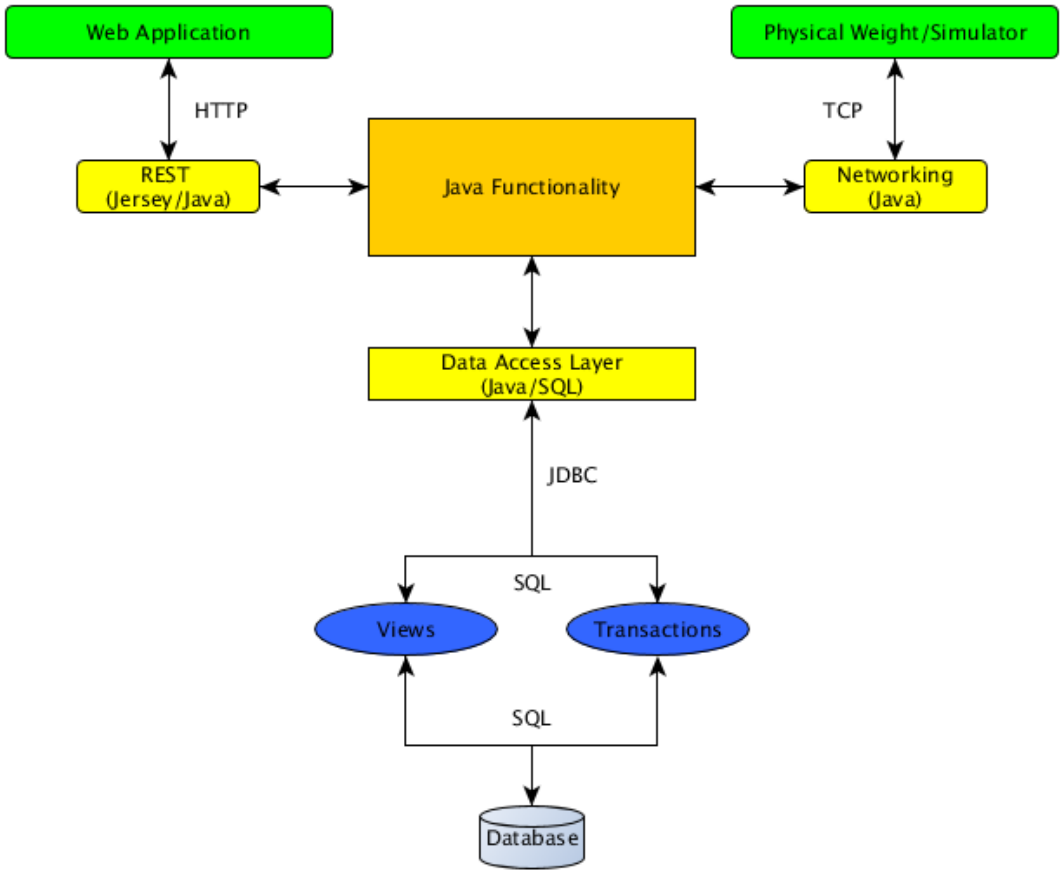


Figure 1: Oversigt over systemet

4.1.1 Programmets Arkitektur

Igennem projektets forløb er der i arkitekturen af programmet haft et fokus på 3-lags modellen, men i takt med dets udvikling, opstod et behov for at introducere flere lag. For at styrke programmets evne til let at blive vedligeholdt og genanvendt, blev brugergrænseflader adskilt fra funktionalitet såvel som data (Og ligeledes gældende for de heromtalte lag).

Dette kan ses på opbyggelsen af vores hjemmeside, der i sig selv har 3-lag i form af et view (HTML), controller (JavaScript) og en boundary (AJAX). Dette vil sige at vores program er opbygget efter 'n-lag', hvor vi har flere forskellige applikationer der har forskellige lag.
I selve java programmet, har vi benyttet os af interfaces for at kunne gøre hver klasse let udskiftlig, så hvis kunden på et tidspunkt skulle have lyst til at udskifte en del i vores program ville de kunne bruge interfacet som en 'kontrakt' til, hvordan den nye del skal se ud. På denne måde vil en anden programmør også kunne let udskifte de forskellige dele af programmet uden at skulle ændre alt muligt i programmet for at gøre det kompatibelt med hans ændringer.
For at kunne skabe forbindelse mellem databasen, hjemmesiden, vægten og selve programmet, har vi benyttet os af forskellige applikationer. For eksempel bliver AJAX og REST brugt som boundaries til hjemmesiden og programmet. På grund af disse to boundaries kan vi let sende information mellem controllerne fra vores hjemmeside og program, frem og tilbage. Samme måde har vi også brugt andre applikationer som boundaries til at forbinde vores program med databasen i form af en JDBC og en socket til at forbinde programmet med vægten.
På denne måde får vi også forsikret os at alt information kører igennem programmet og bliver bearbejdet ordentligt før det bliver sendt videre til en anden del af projektet.

4.1.2 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 og pakker. Vi har opdelt vores kode i såsom denne adskillelse af data og controllers, gjort i et forsøg på at danne et stykke kode, hvori fokus har været på at bevare høj kohæsion og lav kobling igennem hele forløbet.

Det kan også ses i form af at funktionaliteten og REST-delen er adskilt. Altså i stedet for at implementere rest direkte på controller klasserne, er der udviklet interfaces til et REST lag som benytter sig af sin egen controller.

4.2 Designklassediagram

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

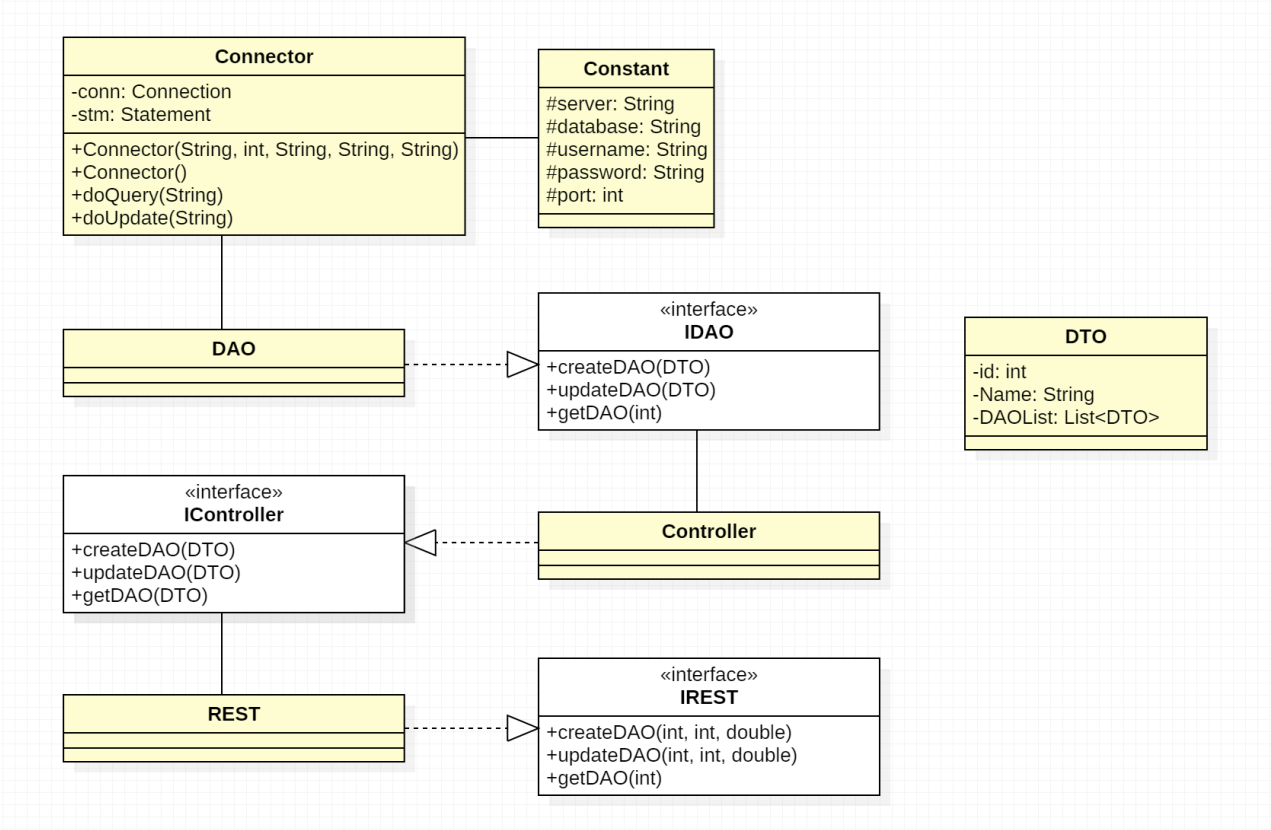


Figure 2: Designklassediagram

Man kan altså se hvordan opbygningen af programmet ser ud ved hjælp af diagrammet. Det kan fx ses i brugeradministrationen at denne struktur bruges. Hvis der tages udgangspunkt i klassen `UserREST`, kan man se at der skabes en instans af `UserController` i denne. Der er nu adgang til controlleren og de metoder der hører til for brugeradministrationen. I `UserController` kan man da se at der nu oprettes en instans af `UserDAO`, som giver adgang til data i databasen.

4.3 Ændring i databaseopsætning

En ændring i databasen har været nødvendig for at opfylde en højre normalform, samt at undgå at have to forskellige identifikations-numre til hver laborant. I f.eks. råvare-tabellen har vi ønsket at trække leverandøren ud og indsætte i en seperat tabel med navn og id, hvor id er fremmednøgle i råvare-tabellen. Brugeradministration er dem der skiller sig mest ud fra de andre klasser i forhold til at man skal kunne aktivere og deaktivere en bruger. De fleste klasser består kun af en række opret, rediger og vis metoder, hvilket er hvorfor der måtte bruges ovenstående metoder til afspejlingen af systemet i vores designklasse-diagram.
En anden vigtig del er at skabes ud fra et interface. Når det gøres er det let at udskifte komponenterne i klasserne.

5 Implementering

5.1 Programopbygning

Til opgavebeskrivelsen fulgte nogle foreslag til hvordan vi kunne opsætte databasen, samt opsætningen af disse DTO'er og DAO'er. Dette gør programmet i stand til at danne objekter af de forskellige dele, samt kunne indsætte og udtrække dataet fra en database. Vi valgte at gemme dataet mellem sessioner i en SQL database, som skal kunne hostes på nettet og dette gør det muligt at programmet altid kan tilgå databasen og det ikke kun er lokalt. Udover de fastlagte DTO'er og DAO'er valgte vi at lave en DAO og DTO til leverandørene, så det er nemt at have flere leverandører til en enkelt råvare i vores Java system. 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. Vi valgt at bruge et 4G modem til vægten, som har en statisk IP-adresse. Dette gør det altså muligt at de enkelte dele af systemet er uafhængigt af hinanden. Vægten, programmet og databasen, skal altå ikke direkte forbindes via kabler men dataet kan overføres via nettet. Databasen og Java laget bliver forbundet med en JDBC (Java DataBase Connectivity) driver, som gør det muligt at sende queries fra Java ned i databasen. Det er da muligt at danne ResultSet af dem, som kan bruges til at føre data, såsom integers og strings, ind i Java laget hvor programmet kan bearbejde dataet.

Java og web-siden kommunikerer via REST, hvor programmet sender JSON objekter fra Java til JavaScriptet. Her er JavaScriptet i stand til at bearbejde og fremvise dataet, eksempelvis vis en bruger skal fremvises og rettes. Programmet benytter sig også af AJAX til at kunne fortælle JavaScriptet hvor den kan få fat på de forskellige metoder, som den gerne vil benytte sig af fra Java laget. Dette gøres ved at benytte sig at @Path i REST og dermed skabe en stige som AJAX kan definere i JavaScriptet.

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 vores system. Eksempeltvis så hvis en besked skal sendes til vægten, og der forventes et 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(WeightException), har vi sørget for at brugeren af vægten får en passende besked så brugeren ved hvad der er gået galt. Eksempelvis 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 vedkommende skal indtaste ID.

Nedenstående er et eksempel på hvordan exceptions og beskeder til brugeren i browseren håndteres. Koden ser således ud.

```
JSONObject returnMessage = new JSONObject();
try
{
   if(name.equals("") || ini.equals("") || active < 0 && active > 1)
   {
      returnMessage.put("message", "Fejl i inputtet!");
      return returnMessage.toString();
   }
   else
   {
      uc.createUser(name, ini, active);
      returnMessage.put("message", "Brugeren " + name + " er oprettet.");
      return returnMessage.toString();
   }
}
catch (DALException e)
{
   returnMessage.put("message", e.getMessage());
   return returnMessage.toString();
}
```

Ideen med at sende et JSON string object videre, er at det skal læses i JavaScript og videregives som en `alert()` besked til brugeren. Der oprettes altså i starten af metoden et JSON objekt. Beskeden brugeren skal se bliver indsat i et JSON objektet med en tilhørende hashkey så man i JavaScript kan referere til beskeden ved hjælp af denne key.

Når dette håndteres i JavaScript ser det således ud.

```
$(document).ready(function() {

   $("#creatingUser").click(function() {
      $.ajax({
         url: "/rest/user/createUser",
         data: $('#createUserForm').serialize(),
         dataType: "json",
         contenttype: "application/x-ww-form-urlencoded",
         method: "POST",
         error: function(xhr) {
            console.log(xhr.responseText);
            alert(xhr.status);
         },
         success: function(data) {
            alert(data.message);
         }
      });
      return false;
   });
});
```

Hvis metoden ender i en succes, altså at den får sendt data afsted i form af et JSON string objekt, vil der blive kaldt en alert med beskeden i. Beskeden fås blot ved at skrive `data.message`, hvor "message" er nøglen til beskeden. Bliver der ikke parset noget data, vil programmet hoppe ned i `error` funktionen som alerter fejlstatussen og logger respons teksten til fejlen.

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 Test

Der har benyttet sig af forskellige test metoder til at afprøve vores program for at finde fejl i programmet, hvorefter vi har justeret i vores kode efter de diverse fejl vi opdagede. Nedenstående testmetoder har været brugt til afprøvning af vores program:

7.1 JUnit test

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

7.2 Brugertest

Vi har ved hjælp af brugertest testet programmet ud fra vores use-cases for at sikre os at slutbrugeren ikke løbe ind i problemer vi ikke har taget hånd om. Vores bruger test blev udført af en tilfældig bruger, der på ingen måde var tilknyttet til udviklingen af programmet. Vi fik følgende bruger til at gå igennem de forskellige use-cases på vores hjemmeside og vægt. Årsagen bag valget af en tilfældig bruger er at vores slutbruger højst sandsynligt ikke kommer til at have kendskab til selve koden i programmet. Derfor ville en tilfældig bruger kunne give os en mere akkurat repræsentation af, hvordan slut brugeren ville bruge programmet.
Under brugertesten ville vi notere nogle fejl eller bemærkninger brugeren havde om programmet og derefter se, hvad vi kunne gøre for at forbedre programmet i forhold til testbrugerens bemærkninger og de fejl som blev fundet under brugertesten.

7.3 Test cases (brugertest)

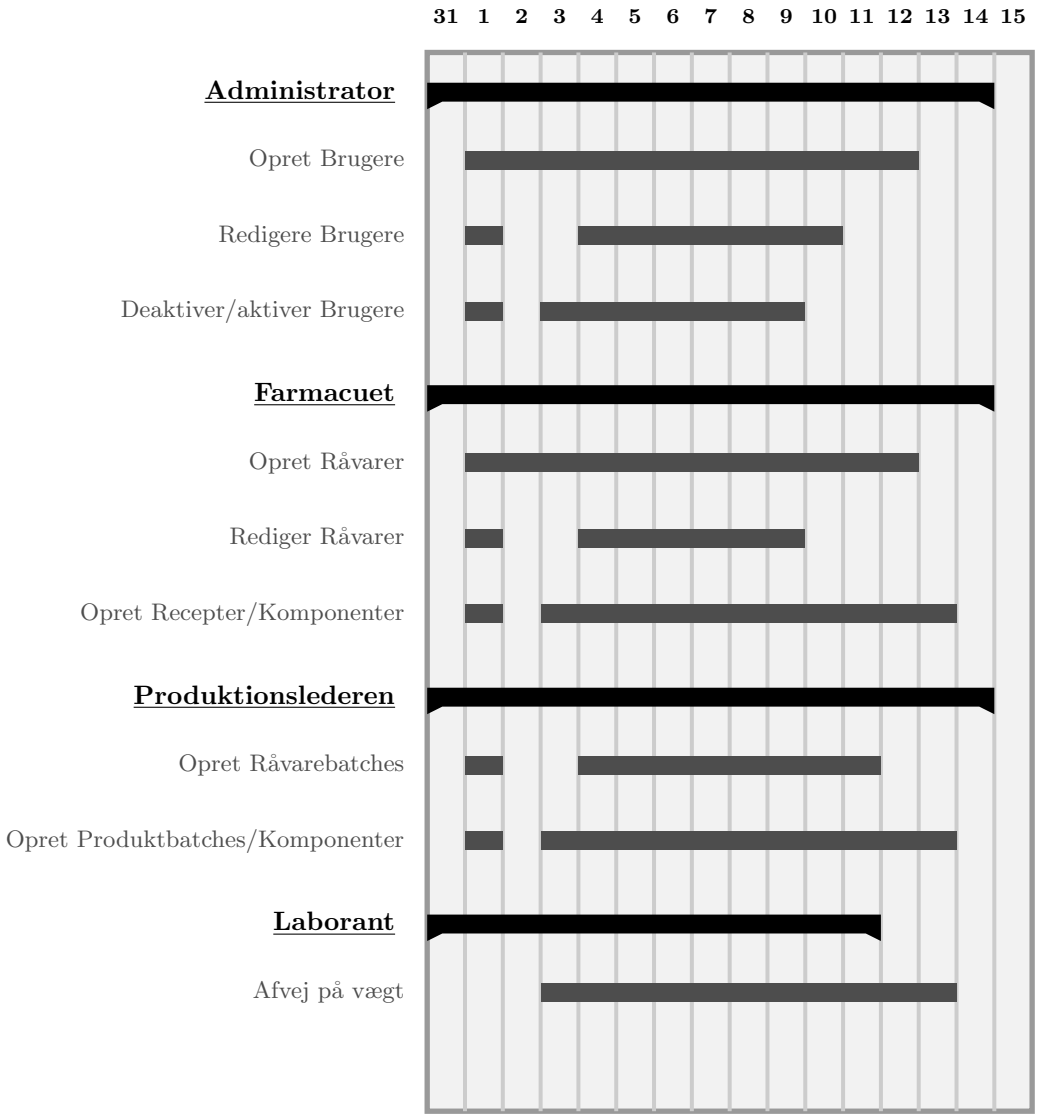
Test ID	Beskrivelse	Forventet Resultat	Opnået Resultat	Noter
T1	Opret bruger	En bruger bliver oprettet	En bruger var oprettet	Success
T2	Find en eksisterende recept komponent.	En recept komponent bliver vist.	Brugeren modtager en fejlbesked	Tjek JS for denne metode
T3	Afvej en råvare fra en råvarebatch	Afvejningen er registreret i databasen	Vægten er registreret i databasen	Success

7.4 Destruktiv testing

For at sikre at programmet fungerer som forventet, er det tilsvarende nødvendigt at undersøge om fejl er håndteret korrekt. Ved at teste programmet uden for dets arbejdsområde, ved at teste input omkring grænsepunkterne, er det muligt at bestemme, om fejl-input resulterer i, det vi forventer. Denne tilgangsmetode er fx blevet anvendt i forbindelse med test af vægt og inaktive brugere. Her er formålet at benægte inaktive brugere adgang til at udføre afvejning på vægten, hvilket er opnået ved at returnere en fejlmeddelelse, når brugerens status ikke er lig 1.

8 Projektplanlægning & Prioritering

Vi har lavet vores planlægning for projektet med hjemmesiden *asana.com*, og har derefter taget udgangspunkt i enkelte use-cases og fået implementeret dem. Vi startede med det grundlæggende i programmet, som var databasen. Vi lavede et udkast til denne, så vi vidste hvor dataet skulle hentes fra, idet vi påbegyndte arbejdet på DAO'erne. De dertilværende DTO'er var på forhånd givet, men grundet overvejelser i forhold til normalformerne blev databasen ændret, og derved var der også et behov for at ændre DTO'erne.
Efter et stort fokus på database arbejde, blev gruppen fordelt på to, og hver tildelt et stykke arbejde. Arbejdet gik efterfølgende på at skabe et funktionalitetslag, der kunne skabe en kommunikation med hhv. hjemmesiden og vægten. Fokusset på den ene gruppen var hjemmeside opsætning og logik, og den anden var kommunikationen med vægten.
Nedenstående diagram illustrerer, hvor gruppen har haft arbejdsfokus. Diagrammet er use-case baseret, og tager højde for design, analyse såvel implementeringen af disse.



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. Det eneste hardware der skal opereres er vægten selv, af en laborant.
Vi har taget udgangspunkt i den fysiske vægt, og har derfor ikke support på den virtuelle vægt.
Flowet med vægten fungerer som forventet, hvor brugervenligheden er så intuitiv, som muligt på den måde. Hvis en bruger skulle komme til at taste noget forkert ind på vægten, så er systemet dynamisk i form af brugeren kan gå tilbage til et tidligere trin, uden at skulle starte forfra. Der er udført både bruger tests og JUnit tests på programmet, men grundet redigering af programmet, så er det ikke alle JUnit tests som fungerer, som de gjorde før redigeringen. Programmet har diverse mangler som med mere tid kunne klares, heriblandt er der nogle bugs med systemet. Vi kunne have sparet en del tid med en bedre analyse og forståelse for, hvordan databasen skulle sættes op.
At have Java laget kørende på en web server, har desværre forvoldet nogle problemer, vi ikke har været i stand til at takle. Det er derfor nogle gange nødvendigt, at genstarte vægten og modem, inden programmet kan køres på Heroku-serveren.

9.1 Perspektivering

Java
Den dynamiske del af vægt-flowet, det der gør at laboranten kan gå tilbage i systemet, har medført nogle fejl som ikke er blevet lappet. Heriblandt, hvis en råvare vejes og trækkes fra i lageret, hvorefter laboranten trykker cancel for at gå tilbage, så lægges den råvare der er taget, ikke tilbage igen i på lageret. Dette betyder jo så at råvaren er 'brugt' selvom det teknisk set stadig er på lager. Yderligere når et produkt batch startes og status ændres til 1, så ændres den ikke tilbage til 0, hvis brugeren vælger at gå tilbage igen. Altså hos produktadministrationen ser det ud som om at dette batch er under produktion, selvom det ikke er.
Vi har også haft et ønske om at ændre exception-håndtering i klassen der taler med vægten, WeightTranslation. Der bliver allerede pakket alle fejl ind i en WeightException, men reaktionen på de fejlbeskeder vægten sender, er ikke håndteret helt korrekt. Dette ville også have hjulpet på vores håndtering af fejl i WeightControlleren. Derudover ville vi også gerne have en generelt bedre håndtering af Exceptions i alle klasser.
WeightCommodities metoden i WeightController har et meget stort ansvar, som vi gerne ville have haft ud flere under-metoder. Dels fordi den er svær at teste, men også fordi vi ikke kan reagere lige så præcist på alle Exceptions, og så skal hele afvejning tages forfra. Vi ønsker at kunne have en form for checkpoint der gør, at brugeren ikke skal lave for meget om igen ved fejl.

HTML og CSS, JavaScript
Vi ville også gerne have haft brugt mere tid på JS, CSS og plain HTML, for at få hjemmesiden til at se mere indbydende ud. JavaScriptet kunne vi godt tænke os var mere overskuelig, så det ikke er så meget på én gang at tage ind. Bedre optimering kunne også være dejligt, så selve hjemmesiden har en hurtigere repsonstid.

10 Bilag

10.1 Afvejning på vægten

Selve det flow som gør at en laborant kan afveje på vægten, startes på en knap på hjemmesiden, hvorefter at flowet kører i et loop, så laboranten kan blive ved med at afveje produkter. Selve flowet, efter der er trykket på knappen er:

- Vægten beder laboranten om at indtaste sig operatør-id

  Operatør kan taste sit id forkert og få en fejl besked, hvorefter der kan forsøges igen

  Operatør kan taste sit id rigtigt og komme videre til næste besked

- Vægten byder operatøren velkommen med operatørens navn

  Operatøren kan taste cancel, for at komme tilbage og indtaste sit id igen

  Operatøren kan trykke ok og komme videre til næste stadie

- Vægten beder operatøren om at indtaste det produktbatch ID som der er opskriften, med de komponenter der skal bruges

  Operatøren kan taste cancel for at gå tilbage og få vist operatør navnet igen

  Operatøren kan taste produktbatch id forkert og få en fejlbesked hvorefter der kan forsøges igen

  Operatøren kan taste produktbatch id rigtigt og komme videre til næste besked

- Vægten beder nu operatøren om at placer den beholder som skal taras

  Operatøren kan taste cancel for at komme tilbage til produktbatch beskeden

  Operatøren kan placer beholderen og trykke ok, hvorefter systemet til tara vægten

- Vægten beder nu om råvarebatch id

  Operatør kan taste cancel, for at komme tilbage til tara stadiet

  Operatøren kan taste forkert id, hvorefter der kommer en fejl besked, hvorefter der kan forsøges igen

  Operatøren kan taste rigtigt id, hvorefter operatøren føres videre i flowet.

- Vægten fortæller operatøren navnet på den råvare der skal vejes og den mængde som råvaren skal veje

  Operatøren kan taste cancel for at komme tilbage til råvarebatch id

  Operatøren kan fortsætte ved at trykker ok

- Operatøren får at vide om råvaren vejer for lidt eller for meget

- Når råvaren vejer den ønskede mængde, spørger vægten om operatøren vedtager vægten

  Operatøren kan sige nej, hvorefter operatøren har 2 sekunder til at opnå den ønskede vægt, hvorefter sidste stadie spørges igen

  Vedtager operatøren vægten, sørger systemet for at ændre på databasen, hvorefter næste råvare skrives på displayet

- Alle råvare for recepten gennemgås og vægten slutter af med at spørge om operatøren er færdig

  Svarer operatøren ja, slukker vægten

  Svarer operatøren nej, vil hele flowet køres igen

10.2 Brug af hjemmesiden

Når hjemmesiden tilgås (www.cdiofinal12.herokuapp.com), mødes brugeren med en side hvor kundens logo står. Oppe til højre er der 4 menuer, hvor de forskellige roller kan tilgå deres tildelte ansvar. Den første knap fra venstre, hvor der står "vægt" giver brugeren mulighed for at starte vægtens flow. Brugeren bliver spurgt om den port der skal forbindes til, hvor brugeren kan indtaste 8000 for den vægt som har IP: 169.254.2.2 og 8001 for den vægt som har IP: 169.254.2.3. Til højre for den knap kan bruger administratoren komme ind og udføre sit arbejde. Til højre for administrator knappen kan farmaceuten klikke på sin knap og komme ind og administrere råvare, recepter, recept komponenter og leverandører. Til sidst kan produktionslederen klikke på sin knap, hvor der kan administreres råvarebatches, produktbatches og produktbatch komponenter. Til enhver tid kan rollerne få lov til at printe hvad der ønskes fra hjemmesiden med genvejen ctrl+p. Alternativt kan et snip fra hjemmesiden tages med programmet snipping tool, som er installeret på alle windows maskiner.

10.3 Heroku opsætning

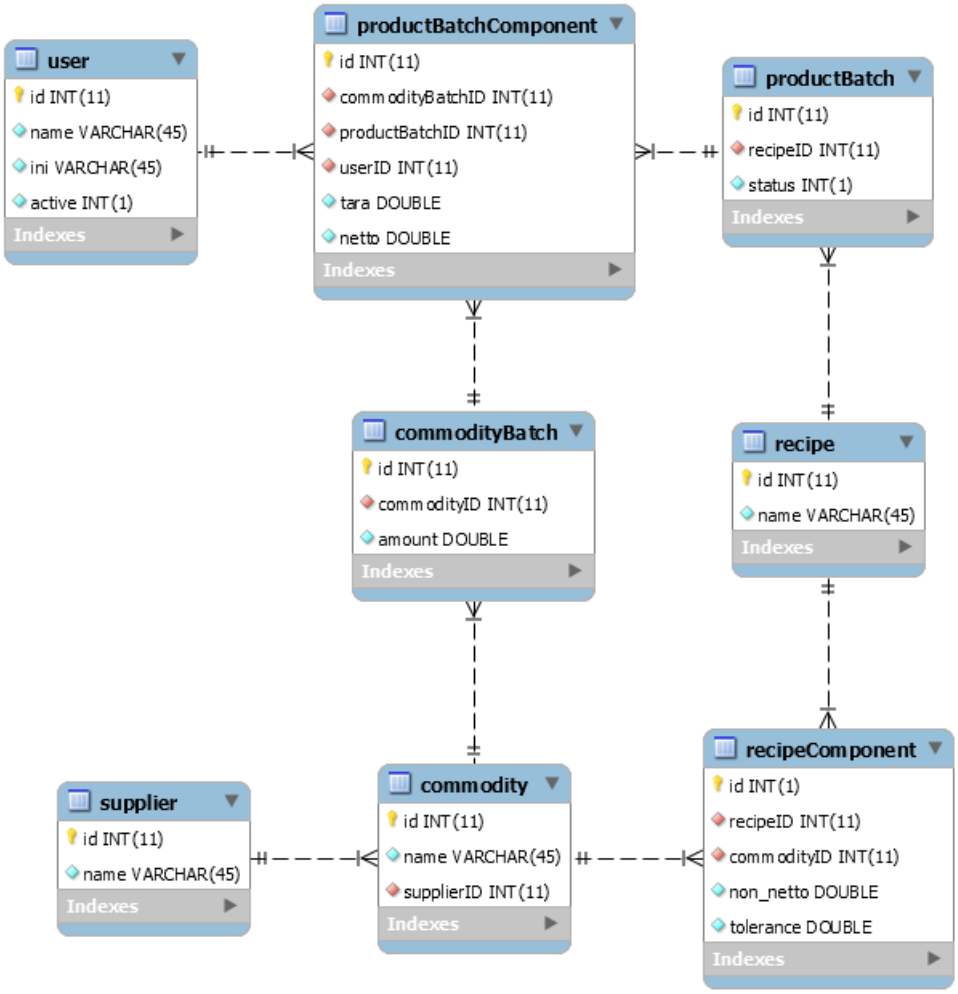
Til opsætning af heroku (projektest deployment-løsning) har vi oprettet en heroku-konto, og en gmail, for at have en e-mail til brug på heroku.
Heroku-kontoen kompilerer koden og henter de repositories der ligger i projektets maven-fil.

Heroku-kontoen er forbundet til vores github-repository, hvor koden kompileres automatisk:
https://github.com/OneGruppe/cdioFinal

Herunder ses log-in detaljer for både heroku-konto
https://dashboard.heroku.com
username: grp12cdiofinal@gmail.com
password: cdioFinal2018!

Herunder ses log-in detaljer for både gmail
https://mail.google.com
username: grp12cdiofinal@gmail.com
password: cdioFinal2018!

10.4 ER-diagram



10.5 Detaljeret timeregnskab

Deadlines										CDIO FINAL START												Ekstra timer		Totalt
22	28/05/18			29/05/18			30/05/18			31/05/18			01/06/18			02/06/18			03/06/18					
	Mandag		Work	Tirsdag		Work	Onsdag		Work	Torsdag		Work	Fredag		Work	Lørdag		Work	Søndag		Work		Work	
Burhan Shafiq										10	16	Analyse	10	16,5	Design	9	16,5	Des., M1, Impl.						20
Mathias Fager										10	16	Analyse	10,5	16,5	Design	9	16,5	Des., M1, Impl.						19,5
Niklaes Jacobsen										10	16	Analyse	10,5	16,5	Design	9	16,5	**Des., M1, Impl.**						19,5
Sebastian Sørensen										10	16	Analyse	10,5	16,5	Design	9	16,5	**Des., M1, Impl.**						19,5
Simon Pedersen										10	16	Analyse	10	16,5	Design	9	16,5	Des., M1, Impl.						20

Deadlines							M1(igen)															Ekstra timer		Totalt
23	04/06/18			05/06/18			06/06/18			07/06/18			08/06/18			09/06/18			10/06/18					
	Mandag		Work	Tirsdag		Work	Onsdag		Work	Torsdag		Work	Fredag		Work	Lørdag		Work	Søndag		Work		Work	
Burhan Shafiq	9,25	16,5	Implementering	10	13	Implementering	9	17,75	Impl + Design	10	18,75	Impl + Design	10,5	16	implementering							6,5	Implementering	39,75
Mathias Fager	9,25	17	Implementering	10	17,25	Implementering	9	18,25	Impl + Design	9	22,5	Impl + Design	10	16	Implementering				12	13,5	Rapport	0,5	Implementering	45,75
Niklaes Jacobsen	9,25	17	Implementering	10	17,5	Implementering	9	18,25	Impl + Design	9	22	Impl + Design	9,5	17	Implementering	16	24	impl + deploym.	0	2	impl + deploym.	10	Impl + design	65
Sebastian Sørensen	9,25	16,75	Implementering	10	17	Implementering	9	16,75	Impl + Design	8	20,5	Impl + Design	10,5	17	Implementering							5,5	Implementering	46,75
Simon Pedersen	9	17	Implementering	10	17	Implementering	9	18	Impl + Design	9	22	Impl + Design	9,33	17,75	Implementering							2	Implementering	47,42

Deadlines				M2 - 9:00									Aflevering - 12:00									Ekstra timer		Totalt
24	11/06/18			12/06/18			13/06/18			14/06/18			15/06/18			16/06/18			17/06/18					
	Mandag		Work	Tirsdag		Work	Onsdag		Work	Torsdag		Work	Fredag		Work	Lørdag		Work	Søndag		Work		Work	
Burhan Shafiq	9,5	16	Impl. + Design + Analyse	10	18,5	Implementering	9,5	17	Implementation	9,5	20,75	Impl + Rapport										8,5	Impl. + Design + Analyse	42,25
Mathias Fager	9,5	17	Impl. + Design + Analyse	10	18,5	Implementering	9,75	17,25	Implementation	9,16	21	Impl + Rapport										6	Impl. + Design + Analyse	41,34
Niklaes Jacobsen	9,5	17	Impl. + Design + Analyse	10	18	Implement + deploym.	9,5	16,5	Implementation	9	21	Impl + Rapport										11	Impl. + Design + Analyse + deplo	45,5
Sebastian Sørensen	9	17	Impl. + Design + Analyse	10	18,5	Implement + deploym.	9,5	16,5	Implementation	9	20,5	Impl + Rapport										4	Impl. + Design + Analyse	39
Simon Pedersen	9	17	Impl. + Design + Analyse	10	18,5	Implementering	9	17	Implementation	9	21	Impl + Rapport										7	Impl. + Design + Analyse	43,5

Link til et Google Spreadsheet kan findes her:
https://drive.google.com/open?id=1FEbC-c2m5acl0UMRjZWrmY2RMhw8CV0tgy4n_vwb__k