Technical University of Denmark

02324 VIDEREGÅENDE PROGRAMMERING

CDIO Final

GRUPPE 14

14. juni 2018

JOACHIM S. MORTENSEN
s175179

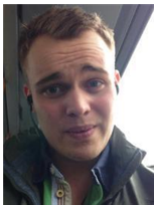
CHRISTOFFER VOIGT
s154308

KASPER B. NIELSEN
S175216

METTE L.B. ANDERSEN
s172840

THOMAS L. VESTERGAARD
S175219

PETER T. BLENDSTRUP
S175210T



NICOLAI KAMMERSGÅRD
S143780

1 Timeregnskab

Opgave del/navn	Analyse	Design	Implementering	Dokumentation	Diverse	Total
Joachim	10	12	42	4	8	76
Mette	9	11	50	5	12	87
Nicolai	9	14	38	7	7	75
Christoffer	8	13	33	8	13	75
Thomas	9	10	27	4	7	57
Kasper	10	11	39	7	6	73
Peter	8	12	43	4	40	107

2 Brugervejledning

For at kunne foretage en afvejning skal man have oprettet et produktbatch. For at kunne oprette et produktbatch skal man have oprettet en recept med tilhørende receptkomponenter. Receptkomponenterne kræver at der eksisterer råvarer. Samtidig skal der også findes en bruger så man kan logge ind på vægten. Selve afvejningen kræver råvarebatches som defineres ud fra råvarer.

Tilføjelser af råvarer, recepter og batches sker på hjemmesiden, hvor man kan navigere frem til de forskellige sider, som hver især står for oprettelsen af de forskellige objekter.

Når man har oprettet alt nødvendigt til at foretage sin afvejning, kan afvejningsprocessen startes ved først at tænde vægten, efterfølgende navigere til menuen på hjemmesiden og her kan der indtastes IP addresse i feltet og trykkes "forbind vægt". Herefter er vægten forbundet, og man kan fortsætte sin afvejningsproces fra vægten af.

Vægten beder dig først om at indtaste dit operatør ID. Du indtaster på vægtens tastatur og trykker på OK knappen.
Sørg for alt er fjernet fra vægten, og tryk på OK knappen.
Herefter placeres taraen på vægten og der trykkes OK.
Herefter indtastes productbatch IDet på det produktbatch der skal afvejes og der trykkes på OK knappen.
Nu afvejes råvaren. Afvej her det antal gram, som står på vægtens skærm og tryk på ENTER knappen.
Fjern alt fra vægten og tryk på OK knappen. Hvis flere råvare skal afvejes, spørger vægten dig om du vil afveje mere. Tryk på OK for at afveje næste råvare eller CANCEL for at afslutte afvejningen.
Hvis du trykker CANCEL eller der ikke er flere råvarer i produktbatchet, vil vægten spørge om systemet skal afsluttes. Tryk OK for at afslutte systemet eller CANCEL hvis det skal fortsættes.

Indhold

3 Indledning

Vi er blevet kontaktet af en medicinalvirksomhed, som ønsker et softwaresystem der kan anvendes til afvejning og dokumentering heraf. Systemet skal kunne varetage at der oprettes brugere, som skal kunne anvende systemet med forskellige rettigheder. Brugerne skal kunne varetage forskellige administrationsopgaver: receptadministration, receptkomponentadministration, råvareadministration, råvarebatchadministration og produktbatchadministration. Disse skal anvendes i en afvejningsproces, som opretter produktbatchkomponenter. Afvejningen foretages med en Mettler Toledo vægt.

4 Analyse

Vi har valgt i analyse fasen at opstille systemkravene vha. MOSCOW (must have, should have, won't have). Disse har vi omsat til nogle use cases for at skabe en prioriterings rækkefølge i den fortsatte design, implementering og test fase.

4.1 Kravspecifikation

Tabel 1: MOSCOW

Skal kunne	**Bør Kunne**	**Må godt kunne**
Oprette råvarer	Vise brugere	Håndtere roller
Oprette recepter	Vise råvare	Deaktivere brugere
Udskrive produktbatches	Vise råvarebatch	ID'e laborant på vægten
Oprette råvarebatches	Vise produktbatch	Udføre brutto-kontrol
Oprette produktbatches	Gemme produktbatchkomponenter	Ændre status i produktbatch
Oprette brugere		Lagerstyring
Have persistent datalag		
Virke med en vægt		

4.2 Use cases

Programmets funktionalitet / krav kan beskrives i 6 forskellige use-cases. Use-case 1-5 beskriver de forskellige funktionaliteter i web-applikationen. Den 6. use-case, beskriver afvejningsprocessen og giver os grundkravet til afvejning styrings enheden (ASE). De 6 use cases prioriteres og implementeres fra top-bottom og testes.

Proriteringsliste for use cases:

1. Råvare administration
2. Recept administration
3. Afvejnings-styringsenheden - ASE
4. Råvarebatch administration
5. Brugeradministration
6. produktionbatch administration / udskriv

Use Case 1 - Råvareadministrationen
Råvareadministrationen giver mulighed for at tilføje råvare, ændre i dem samt vise allerede oprettede råvare. Der kan ses et eksempel på flowet i bilag 1

Tabel 2: Use case 1 : Råvareadministration

Use case nr.	1
Primær Aktør	Farmaceut
Precondition	Er på råvareadministrationssiden.
Main Flow	1. Vis råvarer 2. Opret råvare 3. Ændr råvare

Use Case 2 - Recept administrationen
Receptadministrationen foregår meget lig råvareadministrationen, her er der dog også et behov for at definere de råvare som benyttes i recepterne, hvorfor det også er her at receptkomponenterne bliver oprettet. Altså kan vi både vise og oprette recepter. Der kan ses et eksempel på flowet i bilag 2

Tabel 3: Use case 2 - Receptadministration

Use case nr.	2
Primary actors	Farmaceut
Precondition	Er på receptadministrationssiden.
Main Flow	1. Vis recepter 2. Opret recept

Use Case 3: Afvejnings-styringsenheden - ASE
Afvejningsprocessen sørger for at vejlede brugeren igennem afvejningen af alle produktbatchkomponenter og gemme de forskellige vægte undervejs. Der kan ses et eksempel på flowet i bilag 3

Tabel 4: Use case 3 - Afvejning

Use case nr.	3
Primary actors	Laborant
Precondition	Bruger, ProduktBatch, IngredientBatch.
Postcondition	Ny ProduktBatchKomponent.
Main Flow	1. Laboranten indtaster sit operatør id og trykker OK 2. ASEen kvitterer med navnet på operatøren 3. Laboranten verificerer korrekt operatør 4. Laboranten indtaster produktbatch id 5. ASEen kvitterer med det tilhørende recept navn 6. Laboranten verificerer korrekt produktbatch. 7. ASEen forespørger at vægten er ubelastet 8. Laboranten kvitterer 9. ASEen forespørger TARA 10. Laboranten placerer tara og kvitterer 11. Laboranten indtaster råvarebatch id 12. ASEen forespørger specifik mængde. 13. Laboranten placerer mængde og kvitterer 14. ASEen forespørger at alt fjernes 15. Laboranten fjerner alt fra vægten og kvitterer 16. ASEen kvitterer med godkendt og gemt afvejning 17. Step 7-16 gentages indtil alle produkt batch komponenter er afvejet, eller operatøren vælger at afslutte.

Use case 4: Råvarebatch administration Råvarebatch administrationen giver os muligheden for at oprette nye batches modtaget fra leverandører. Disse sammenkobles med en råvare ved at specificere deres råvareId. Igen har vi også mulighed for at liste de allerede modtagne råvarebatches. Der kan ses et eksempel på flowet i bilag 4

Tabel 5: Use case 4 - Råvarebatch administration

Use case nr.	4
Primary actors	Produktionslederen
Precondition	Er på råvarebatchadministrationssiden.
Main Flow	1. Vis råvarebatches 2. Opret råvarebatch

Use case 5: Brugeradministration
Brugeradministrationen giver os mulighed for at oprette, ændre, vise samt deaktiverer brugere som benytter systemet. Der kan ses et eksempel på flowet i bilag 5

Tabel 6: Use case 5 - Brugeradministration

Use case nr.	5
Primary actors	Administrator
Precondition	Er på brugeradministrationssiden.
Main Flow	1. Vis brugere 2. Opret bruger 3. Ændr bruger 4. Deaktiver bruger

Use case 6: Produktbatchadministrationen
Produktbatch administrationen giver os vores overblik over produktionen. Vi har mulighed for at oprette nye produktbatches hvor vi kan specificere hvilken recept der bruges lige i dette produktbatch. Vi har også en status tilkoblet et produktbatch, denne sættes til under produktion når afvejningsprocessen igangsættes og til afsluttet når alle produktbatchkomponenter er afvejet. Der kan ses et eksempel på flowet i bilag 6

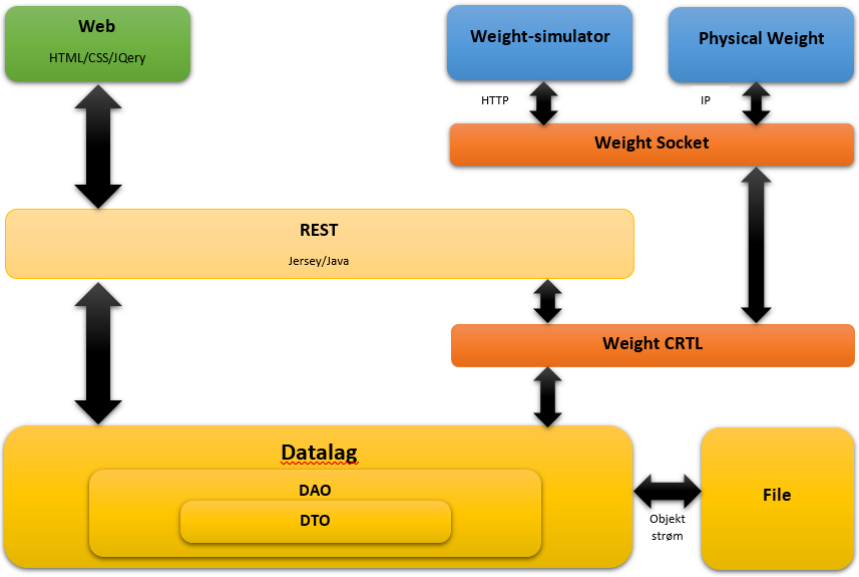
Tabel 7: Use case 5 - Produktbatch administration

Use case nr.	6
Primary actors	Produktionsleder
Precondition	Er på produktbatchadministrationssiden.
Main Flow	1. Vis produktbatches 2. Opret produktbatch

5 Design og Implementering

Programmet bygger på 3 forskellige dele samt brugerinteraktion. De tre dele er henholdsvis en Webapplikation hvor brugere kan oprette og redigere elementer i datalaget, et datalag med specifikke klasser og metoder for tilgang til data samt en ASE som holder styr på forbindelse og kommunikation mellem vejeterminalen og datalaget. Brugerne interagerer med vægten eller web-applikationen. Vi har et behov for at den data vi opretter i webapplikationen kan tilgås af vægten under afvejningsprocessen (af ASE-en). Dette klarer vores datalag, ved at gemme data i filer som begge kan tilgå. Programmet fungerer helt praktisk ved at vi har en webapplikation som brugeren kan interagere med. Applikationen kommunikerer med datalaget ved at bruge jQuery til at tilgå en RESTful service som bruger JSON til at kommunikere med datalaget.

Hvordan de forskellige systemer interagerer ses nedenfor i et systemarkitekturdiagram:



Figur 1: Systemarkitkturdiagram

5.1 Fravalg

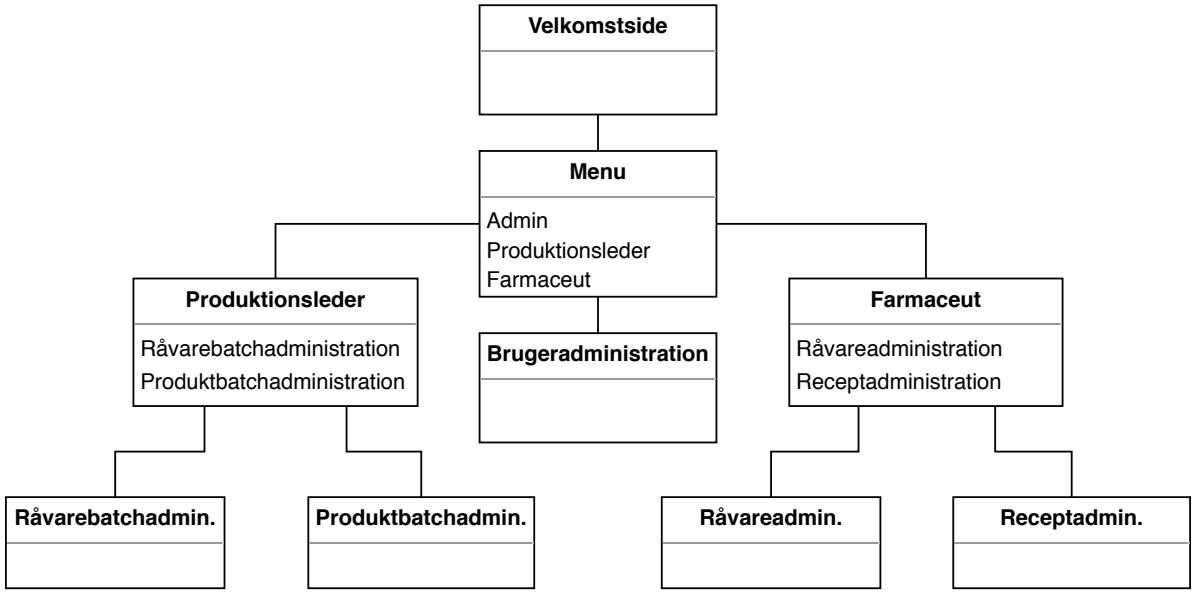
Leverandør
Da leverandør generelt ikke er synlig nogen steder så vi har nedprioriteret den helt væk.
Start/slutdato
Da den givne ProduktBatchDTO ikke inkluderede start- og slutdata, har vi fravalgt dette.
Fuldstændig simulator support.
Da vægtsimulatoren opfører sig markant anderledes end den fysiske vægt, har vi fravalgt at understøtte simulatoren.
Login / Session
Da vi ikke har erfaring med sessioner har vi ikke implementeret en login funktion.
Tjek af brugers rolle og status som aktiv
Da de fleste roller kan benytte vægten jvf. projektoplægget har vi fravalgt at tjekke roller på vægtens login.
Vægt viser ikke foreslag til næste råvare

Det virkede ikke vigtigt at foreslå råvarer til laboranten eftersom de har et udprint med alle receptkomponenterne listet.
Fratræk mængde fra råvarebatch når man bruger fra dem
Vi har nedprioriteret at mængden på lager bliver holdt opdateret imens man bruger det.
Opdeling af produktbatchkomponenter over flere råvarebatches
Vi har fokuseret på at man overhovedet kan oprette produktbatchkomponenter.

5.2 Web

Til almindelig brug af systemet er der bygget en webapplikation med sider lavet til at opfylde specifikke funktioner. Applikationen er lavet som en multipage applikation.

Side struktur Applikationen er bygget op sådan at man starter på en forside hvor man vælger hvilke type af bruger man er. Herefter bliver der loadet en ny side med oversigt over de funktioner som den pågældende bruger. Vælger man f.eks. produktionsleder får man muligheden for herefter at vælge om man vil administrere råvarebatches eller produktbatches. Vælger man en af disse sider kommer man så ind hvor man aktivt kan tilføje og redigere ting i datalaget, hvilket sker via jQuery. Bruger indtaster data i HTML felter som er tildelt en klasse eller et id. Data hives ud med JavaScript vha. jQuery, og sendes til server, som gemmer det i datalag. Vores system er baseret rundt om at brugeren har ansvaret for at vælge den del af systemet som er nødvendig for brugerens "use-case", fremfor at uddele rettigheder i et account-baseret system, hvor at disse rettigheder er uddelt til de forskellige medlemmer på forhånd.



Figur 2: Strukturen af hjemmesiden.

Special case: overførsel af mange entries
Når vi overfører receptkomponenterne, hvor der kan være mange af dem, vil vi gerne undgå at der sendes mange små beskeder med stort HTTP overhead. Derfor indtaster farmaceuten her alle receptkomponenterne i en tabel i sin browser, som pakkes ind i et JavaScript array og overføres i ét hug.

ID i URL
Da vi har valgt at lave vores sider som HTML filer med indlejret JavaScript er det ikke muligt at overføre en side udfyldt med data fra serveren, altså bliver svaret på GET request ikke påvirket af parameteren. For at f.eks. vise receptkomponenter, placerer vi en receptId parameter i URL links til recept display siden, som vi griber fat i med JavaScript og bruger til at hente de korrekte data fra serveren.

jQuery-Tomcat kommunikation
Services på serveren er sat op til at direkte modtage objekterne i vores DTO. Det gør den ved at Jackson oversætter JSON sendt gennem Agent.postJson i browseren. Services er bygget på en RESTful måde, hvor GET bruges til at hente data fra serveren, POST bruges til at oprette nye entries i vores datalag, UPDATE bruges til at redigere entries. DELETE bruges ikke. Services er stateless, klienter har ingen session med serveren.

Agent
Vi har lavet Agent for at forenkle vores AJAX kald i resten af JavaScript koden. Vi anvender 3 funktioner på agenten til RESTful web-service GET, POST og PUT. Agenten er et objekt hvor der defineres hjælpefunktioner, som tager imod 3-4 parametre. URL for en Service, Data som bliver lavet om til JSON-streng hvis det er POST eller PUT, Succes-callback, hvad skal der ske når det går godt, Error-callback, hvad skal der ske hvis der sker en fejl. Med agenten undgås der at skulle specificere HTTP header for hver restful web-service request. I hver HTTP request header, skal der defineres url, method, success, content-Type, error med GET request og DELETE request og med post og put skal der tilføjes data

```
var Agent = {}
Agent.getJson = function(url, success, error){
    $.ajax({
        url:url,
        method: 'GET',
        contentType:'application/json',
        success:success,
        error:error
    })
```

Figur 3: Agent findes under WebContent som en JavaScript fil

Overførsel af data
Data overføres i objekter hvis form svarer til vores DTOer. Det gør at vi kan oprette et JavaScript objekt og så længe det har de attributter som DTOen kræver, kan den overføres som JSON streng som Jersey opsnapper og laver til en instans af DTOen igennem Jackson.

Special case: print ProduktBatch
For at printe produktbatches er der data fra mange DTOer der skal samles. Derfor har ProductBatchPrint-Service sin egen private DTO hvori den samler alle de oplysninger en række output skal have. Det gør vi for at undgå JavaScript der asynkront skal hente data fra alle forskellige services.

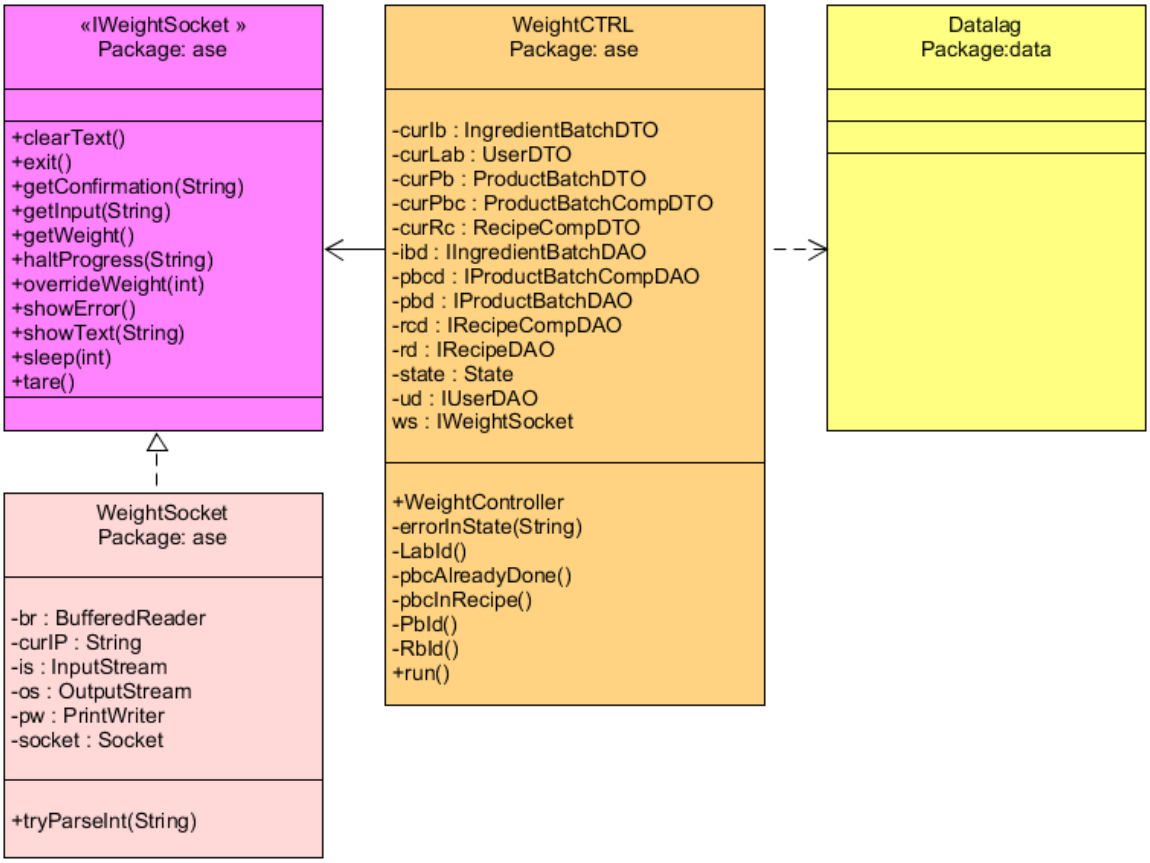
WebDAOException
For at sende svar tilbage til browseren når en request fejler har vi lavet en WebDAOException med tilhørende ExceptionMapper, som automatisk sender en 500 Bad Request response der indeholder et objekt med en *message* besked. Det gør at vi kan vise fejlbeskeder til brugeren uden at skulle gætte på hvad fejlen var på serveren. WebDAOExceptions kastes kun af vores services, generelt når de får en DALException fra en DAO, f.eks. hvis man prøver at oprette en ingrediens med et ID der allerede er i brug.

Thread
Vi har valgt at starte vores ASE på forsiden af vores website. Dette gøres ved at angive vægtens IP og trykke forbind vægt. Dette starter en ny tråd hvor ASEen vil blive kørt på. Vi sørger for at beholde referencen til vores tråd objekt, ved at have en WeightThreadController klasse, som er en singleton. Denne klasse sørger for at holde tråd objektet, samt implementerer en metode der kan give os en status på om tråden stadigvæk er i live eller startet. Dette sætter selvfølgelig et krav til vores WeightController om at den implementerer 'Runnable'. Dette betyder blot at vi skal sikre os at overskrive run() metoden fra interfacet Runnable, da det er denne metode som bliver kørt ved tråd start.

5.3 ASE

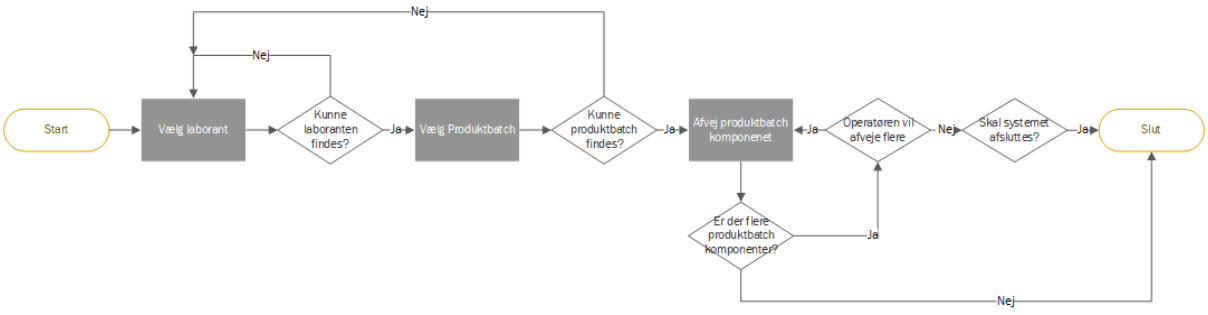
ASE'en består af to komponenter, en WeightSocket der sørger for at oprette og holde forbindelsen til vægten samt indeholder de kommandoer der skal kunne sendes til vægten, WeightController sørger for at disse kommandoer bliver kaldt i den rigtige sekvens, og sørger for at dataen der bliver hevet ud af vægten bliver gemt de rigtige steder.

5.3.1 Designklassediagram af ASE



Figur 4: Strukturen af ASE.

5.3.2 Flow chart-diagram af ASE



Figur 5: Flowet af ASE.

5.3.3 WeightSocket

Efter nogle overvejelser af hvilken funktionalitet der var nødvendig på vægten for at kunne udfylde use-case nr. 6's krav og følge dens forløb blev denne liste udarbejdet.

- Få belastningen i gram på vægten
- Tarere vægten
- Vise tekst på displayet
- Rense al tekst fra displayet
- Få tal-input fra brugeren
- Få ja/nej input fra brugeren
- Sætte processen på pause indtil brugeren kvitterer
- Vise en fejl til brugeren

Denne liste blev grundlaget for implementationen af WeightSocket klassen.
Forbindelsen består af et Socket der opretter en TCP-forbindelse til vægten, en OutputStream og en InputStream. Der skrives og læses til og fra disse to streams ved hjælp af en BufferedReader til at læse, og en PrintWriter til at skrive. Writer og Reader objekterne bruges så i de separate funktioner til at henholdsvis sende SICS-kommandoer og behandle deres respons.

5.3.4 WeightController

WeightController klassen benytter WeightSocket og sørger for at forløbet i afvejnings-use-casen følges. Den implementerer desuden Runnable, så den kan køres på en tråd for sig selv. Systemsekvensdiagrammet kan findes i bilag 3.

5.4 Datalag

Datalaget består af 4 komponenter: DTO (Data Transfer Object), og DAO (Data Access Object) som nedarver StorageDAO og implementerer IDAO (Interface Data Access Object).

Data Transfer Object
DTO skaber muligheden for at holde flere detaljer / variable i et objekt. Det vil eksempelvis være en Bruger, hvor vi har behov for en Integer i form af Id, en String i form af Navn, osv. Altså får vi mulighed for at relaterer til en bruger som et objekt.

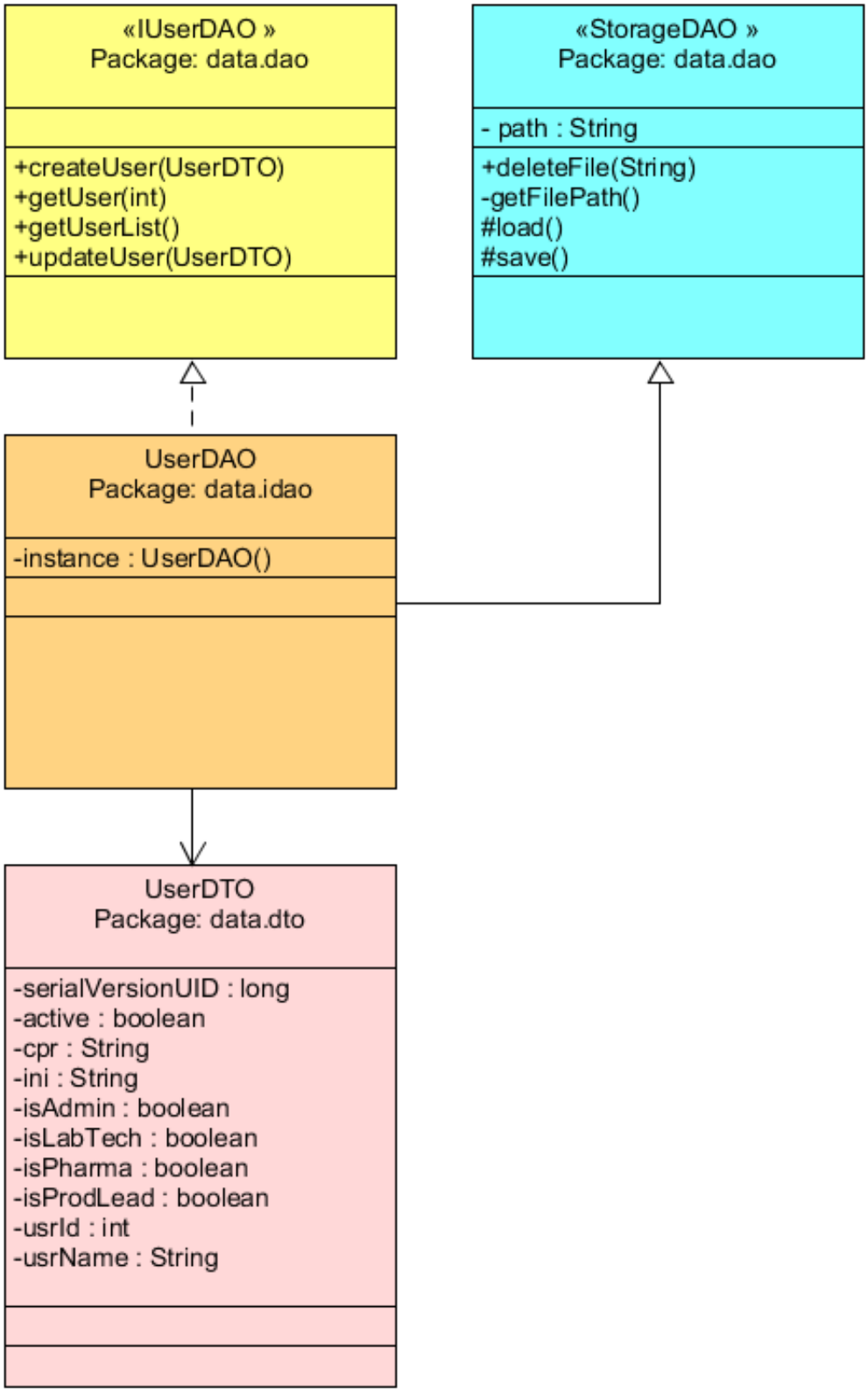
Interface Data Access Object
IDAO er et interface der definerer de funktioner og krav vi har i og til vores DAO.

Data Access Object
DAO indeholder de funktioner som bliver brugt til at tilgå datalaget. DAOerne indeholder validerings funktionalitet, der checker om de objekter vi prøver at oprette allerede eksisterer og hvis de eksisterer kaster den en DALException. DAOerne er også ansvarlig for at loade og gemme vores lokale datafil sådan at vi kan tilgå dataen og gemme den. Se afsnittet om StorageDAO for flere detaljer om gem og load funktionaliteten.

StorageDAO
For at gøre vores datalag persistent samt tilgængeligt for både ASE og Web-applikation, Har vi valgt at introducerer en super klasse til alle vores DAOer. Denne superklasse, StorageDAO, giver 2 metoder til alle underklasserne: save() og load(). save() giver os muligheden for at gemme et objekt til en fil. Her Gemmer vi alle vores DAO-HashMaps indeholdene vores DTOer i hver sin fil. load() giver os muligheden for at hente disse Hashmaps ind igen fra filen. Dette betyder at vi ikke holder vores HashMaps i hukommelsen i hele instansens levetid, men i stedet at vi henter fra fil hver gang vi har behov for data i vores metoder, samt at vi gemmer til fil med det samme hver gang vi modtager / opretter en ny DTO. Lokationen hvor vi gemmer dataen bliver en undermappe ved navn .weightdata i ens brugermappe. Valget herpå faldt da vi havde behov for en kendt path for både Linux og Windows maskiner. Dette kunne Java hjælpe os med vha. System.getProperty("user.home").

Nedenfor ses et eksempel af sammenhængen imellem IDAO, DAO, DTO og StorageDAO for en enkelt af vores data typer. Denne tilgang er generel og benyttes på tværs af vores 7 forskellige typer: Recept, Receptkomponent, Råvare, Råvarebatch, Produktbatch, Produktbatchkomponent og Bruger.

5.4.1 Designklassediagram af Datalag



Figur 6: Strukturen af datalaget.

5.4.2 Synchronized

Da services kører på flere tråde i Tomcat, DAOerne er singletons, og DAO 'transaktioner' ikke er atomiske, har vi synchronized alle DAO metoder som gemmer til og læser fra fil. På den måde undgår vi f.eks. at to service-tråde prøver at tilføje en recept, hvor den sidste recept overskriver den første men begge services returnerer succes. Synchronized tillader kun at et synchronized metodekald køres i samme objekt, hvilket kan sløve serveren ned hvis der er mange data der skal tilføjes.

6 Test

Til test-brug af programmet har vi dummy-data der kan nulstilles vha. en knap på velkomstsiden. Knappen vil sætte datalaget tilbage til fabrikstilstand.

Automatiske test af ASEen og vores website er ikke oplagt, da meget af indholdet i begge er afhængige af at en bruger kommer og interagerer hermed. Brugertesten er udført af medlemmer af gruppen.

6.1 Web

Opret recept
Vi har lavet en brugertest på oprettelse af recept. Denne kan ses trin for trin i bilag 7.

Udskriv produktbatch
Vi har lavet en brugertest på udskrift af produktbatch. Denne kan ses trin for trin i bilag 8.

6.2 ASE

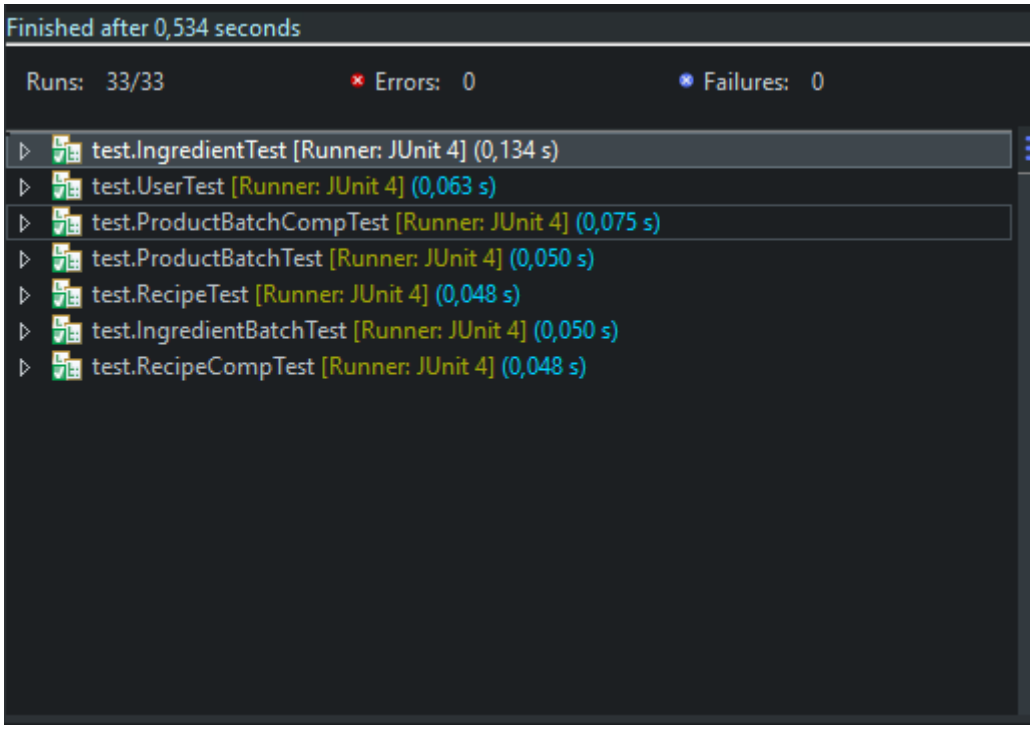
Vi har udført en brugertest jvf. Bilag 9, og den var succesrig. Vi har ud fra denne konkluderet at vores system lever op til kundens krav, og repræsenterer den use-case vi har opstillet.

6.3 Datalag

Datalaget vi benytter os af skal naturligvis testes. Automatiserede tests er oplagt til dette formål, da det handler om at lagre noget data, og at se om lageret opfører sig korrekt og som forventet.

JUnit
Vi tester alle vores Data Access Object klasser ved hjælp at JUnit tests, her testes de forskellige metoder, som anvendes af REST Services. Herunder tester vi generering af keystrings, oprettelse af objekter, opdatering af objekter og hentning af objekter samt lister med alle objekter af en DTO. Disse testes både positivt og negativt. Formålet med dette er at teste at vores datalager fungerer som ønsket.
Vi tester alle metoderne igennem positivt, f.eks. ved at teste at et objekt oprettes med foranstaltninger om at objektet oprettes uden fejl og at objektet faktisk indeholder det som vi forventer.
Vi tester også negativt, her vender vi foranstaltningerne om, og forventer at en fejl opstår og gribes, f.eks. ved at prøve at skabe et objekt med samme unikke ID, som et allerede eksisterende objekt.

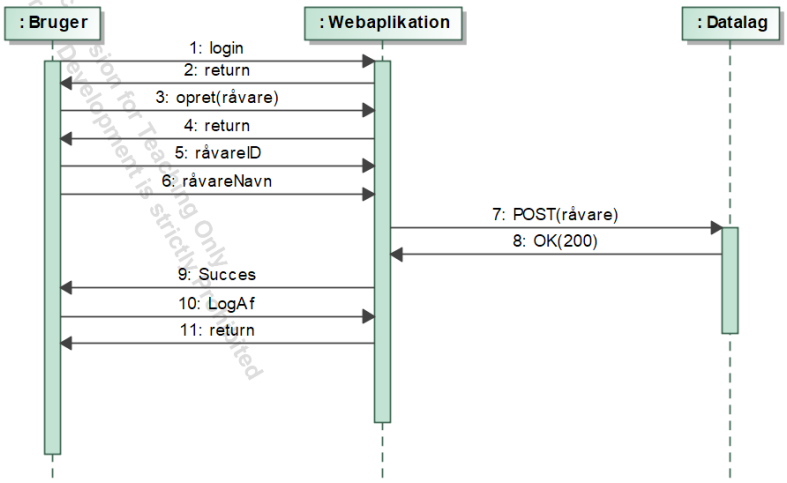
Nedenfor ses en oversigt over vores tests, og at de alle er bestået uden fejl.



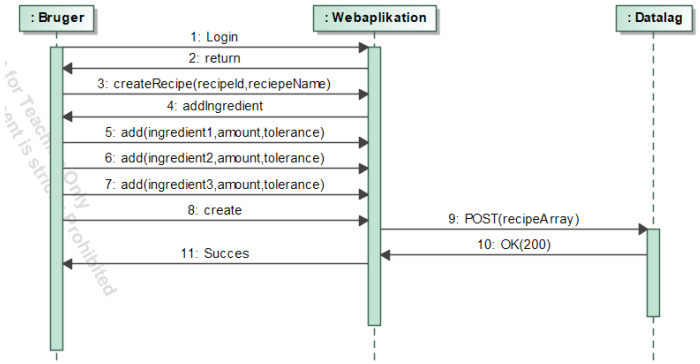
Figur 7: JUnit tests.

7 Bilag

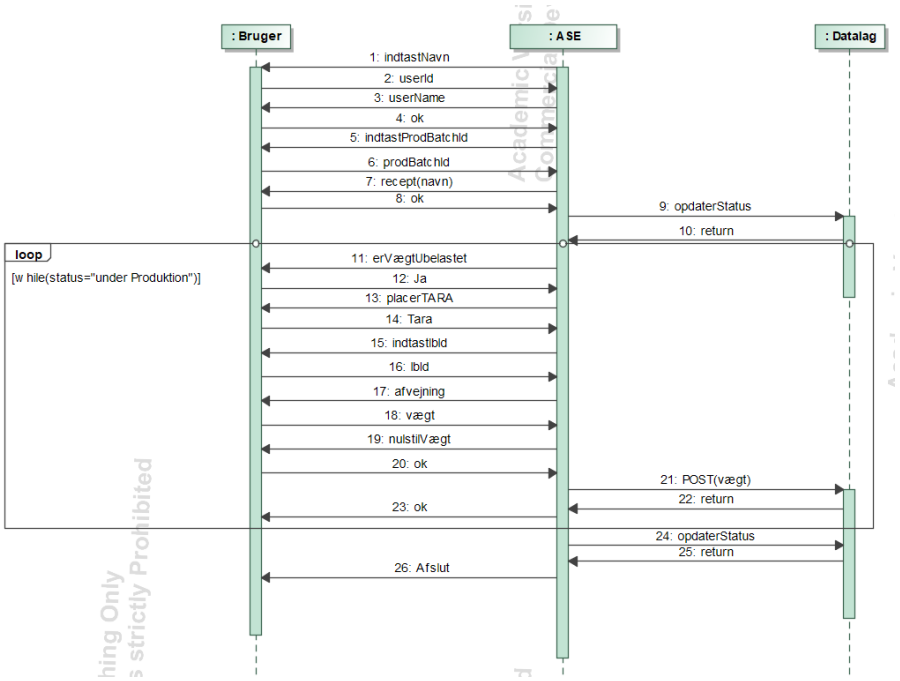
7.1 Bilag 1 : Råvare administration - SSD



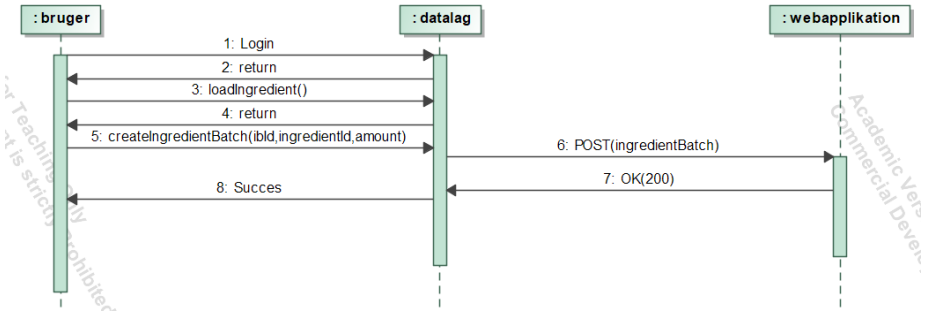
7.2 Bilag 2 : Recept administration - SSD



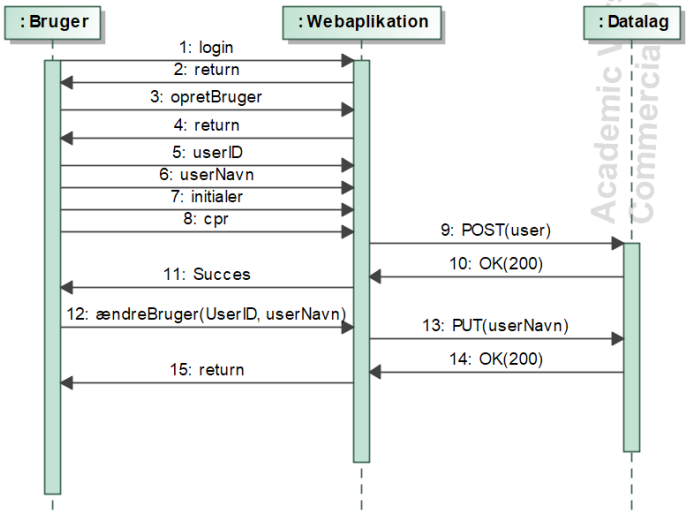
7.3 Bilag 3 : Afvejnings-styringsenheden - ASE - SSD



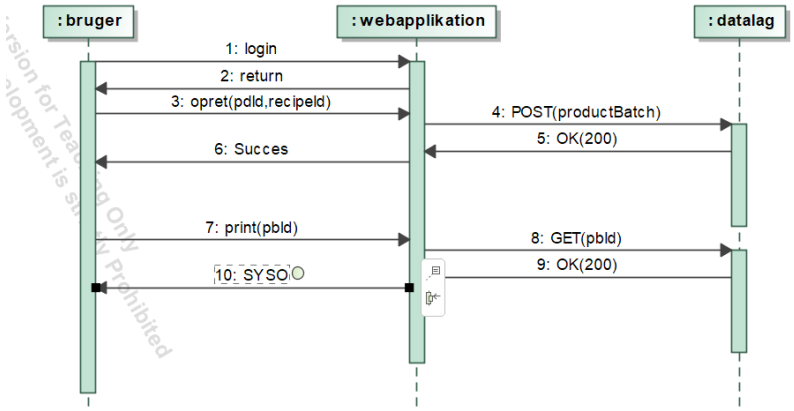
7.4 Bilag 4 : Råvarebatch administration - SSD



7.5 Bilag 5 : Brugeradministration - SSD



7.6 Bilag 6 : produktionbatch administration / udskriv - SSD



Bilag 7 : Brugertest af receptopretning

Test #	Description					
#3		**Step #**	**Steps**	**Test Data**	**Expected Results**	**Actual Results**
		Pre-conditions :	Råvare data der kan gribes fat i ved oprettelse af ny reccept			
		1	Åben browser og vælg farmaceut		Får Farmaceut menu	Får Farmaceut menu
	Opret ny recept med tilhørende receptkomponenter	2	Vælg receptadminstration		Liste med recepter og oprettelse vises	Liste med recepter og oprettelse vises
		3	Indtast recept Id og navn og tryk opret	13, Panodiler	Recept oprettes og der videredirigeres til receptkomponent side	Recept oprettes og der videredirigeres til receptkomponent side
		4	Råvare vælges, mængde og tolerance indtastes	Laktose, 400, 2	Receptkomponentet bliver tilføjet til tabellen	Receptkomponentet bliver tilføjet til tabellen
		5	Råvare vælges, mængde og tolerance indtastes	Smertestillende, 100,	Receptkomponentet bliver tilføjet til tabellen	Receptkomponentet bliver tilføjet til tabellen
		6	Tryk på færdig		Samtlige komponenter sendes til serveren	Samtlige komponenter sendes til serveren
		7	Tryk 'vis' ud for det nye Recept	13	Detaljeret indhold af den valgte recept vist	Detaljeret indhold af den valgte recept vist

Bilag 8 : Brugertest af produktbatchopretning

Test #	Description					
#1		**Step #**	**Steps**	**Test Data**	**Expected Results**	**Actual Results**
	Start nyt produktbatch	1	Åben browser og gå ind under produktbatchadmin		Liste over produktbatches	Liste over produktbatches
		2	Indtast pbId og receptId, tryk opret		Nyt produktbatch oprettes og tilføjes til den allerede synlige liste	Nyt produktbatch oprettes og tilføjes til den allerede synlige liste
		3	Tryk 'vis' ud for det nye produktbatch	99	Detaljeret indhold af produktbatch som matcher det givne eksempel	Detaljeret indhold af produktbatch der nogenlunde matcher eksemplet.

Bilag 9 : Brugertest af afvejningsproces

Test #	Description					
#2		Step #	Steps	Test Data	Expected Results	Actual Results
		1	Start the program and the simulator			
		2	Enter UserID and click OK	1	Greeted by program	Greeted by program
		3	Confirm correct UserID	y	Continue with program	Continued with program
		4	Enter productbatchID	1001	Continue with program	Continued with program
		5	Confirm correct productbatchID	y	Continue with program	Continued with program
	Afvej råvarebatch	6	Confirm the weight isn't in use	y	Continue with program	Continued with program
		7	Enter tare	0.100 kg	Weight sets 0.100 kg to 0	Weight set 0.100 kg to 0
		8	Enter ingredientbatchID	100	Continue with program	Continued with program
		9	Weigh off 500g of chosen ingredient	0.500 kg	Weight responds with "fjern alt"	Weight responds with "fjern alt"
		10	Remove the full weight inc tare	-0.600 kg	Weight responds with "Godkendt. Afvejning gemt"	Weight responds with "Godkendt. Afvejning gemt"
		11	Confirm another weighing	y	Goes to step 6 again	Went to step 6 again
		12	Repeat steps 6 to 10 for ingredientbatchID "101"	Same as 6-10	Same response as 6-10	Same response as 6-10
		12	Enter UserID and productbatchID again	1, y and 1001	Program to tell us "Produktbatch er afsluttet"	Program tells us "Produktbatch er afsluttet"

8 Konklusion

Projektet er blevet lavet så det lever op til de krav der er blevet stillet af kunden. Projektet er blevet analyseret og der er blevet opstillet kravliste og use cases til design af programmet. Den endelige løsning er en webapplikation hvor brugerene kan tilgå de funktioner de skal bruge for at kunne udføre deres arbejdsopgaver. Alt data som bliver tilføjet til systemet bliver gemt i en lokal løsning som tilgås via et Data Access Object. Til systemet er der også blevet implementeret muligheden for at tilslutte vejeterminaller som har adgang til komponenter i datalaget så man nemt kan finde ud af hvilke råvare der er blevet brugt til fremstilling af produkter. Systemet er lavet på sådan en måde at der nemt kan arbejdes videre med det såfremt kunden ønsker det, f.eks. i tilfælde af at der ønskes bruger login.