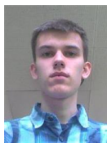
02324

Vidregående programmering

CDIO - Final

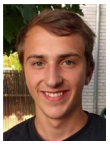
Gruppe 17

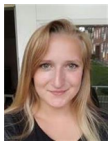
s185118
Aleksander Lægsgaard Jørgensen



s175561
Andreas Østergaard Schliemann

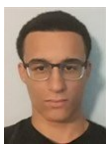
s185096
Jonatan Amtoft Dahl





s164177
Josephine Weirøse

s185103
Søren Hother Rasmussen



s185121
Theodor Peter Guttesen

17. juni 2019

DTU Compute
Institut for Matematik og Computer Science

DTU Diplom
Center for Diplomingeniøruddannelse

Kode og Git: https://github.com/aleksanderlj/17-CDIO_Final-2Sem

Abstract - Theodor

Indhold

1 Indledning - Theodor

I dette projekt har vi lavet en softwareløsing, der kan bruges til dokumentation af råvarebatch forbrug og afvejning af råvarer.

2 Timeregnskab

2.1 Forklaring - Theodor

Rapport: dækker over de diagrammer og den tekst der er og til opsætning af struktur i LaTeX. Hvem der har skrevet de forskellige afsnit eller lavet diagrammer fremgår af overskrifterne i rapportens sektioner.
Database: dækker over opsætning af database i MySQL og Java-JDBC laget i form af DAO'er og DTO'er
Webpage: Dækker over arbejde på hjemmesiden især HTML, CSS og JavaScript, men også REST i java.
Vægt: Dækker over arbejdet med WeightController og WeightConnector klasserne.
Projektplanlægning: Dækker over tidsplan og diskussion af proces.
Test: Dækker over unit-tests

Opgave	**Josephine**	**Aleksander**	**Søren**	**Jonatan**	**Andreas**	**Theodor**	**Sum**
Rapport	11	0.3	4	10	13	25	**63.3**
Webpage	0	49	35	0	0	0	**84**
Database	0	0	7	20	2	2	**31**
Vægt	0	0	0	10	0.5	14	**24.5**
	0	0	0	0	0	0	**0**
Projektorganisering	2	2	2	2	2	2	**12**
Test	0	0	0	8	1	3	**12**
Sum	**13**	**51.3**	**48**	**50**	**18.5**	**46**	**226.8**

3 Krav

3.1 Vision - Andreas

Medicin virksomheden, Panther Pharma, ønsker et softwaresystem, de kan bruge til afvejning af råvare og dokumentation af afvejningen og råvarebatch-forbrug. I dette system vil det være muligt at definere råvare og indskrive råvarebatches, som sendes fra leverandører og stilles på Panther Pharmas lager. Hvert råvarebatch vil indeholde en mængde råvarestof, der bruges af til produktion af produktbatches. Det vil også være muligt at definere recepter, som indeholder en liste af råvare og den mængde, der skal bruges til produktion. Disse recepter vil blive anvendt til at producere produktbatches, som indeholder information om, hvilken recept, der er blevet brugt, hvilke råvarebatches, som er blevet brugt og den aktuelle mængde af dem, der blev brugt. I systemet er fire roller, administrator, produktionsleder, farmaceut og laborant, som brugere kan have, der hver har forskellige rettigheder.

3.2 Navneordsanalyse - Aleksander og Andreas

Navneord	**Beskrivelse**
Råvare	En ingrediens, men på et konceptionnelt niveau.
Råvarebatch	En fysisk ingrediens.
Recept	En opskrift bestående af råvare.
Produktbatch	Et fysisk produkt lavet ud fra en recept og lavet af råvarebatches.
Administrator	En bruger, der, som den eneste, kan oprette og redigere bruger i systemet.
Farmaceut	En bruger, som kan oprette og redigere recepter og råvarer. Kan det samme som produktionsledere.
Produktionsleder	En bruger, der kan oprette og redigere råvarebatches og produktbatches. Kan der samme som laboranter.
Laborant	En bruger, som kun kan afveje råvarebatches.

3.3 Kravliste - Andreas og Josephine

ID	**Funktionelle krav**
Must have	
M01	Systemet skal kunne oprette og redigere råvarer
M02	Systemet skal kunne oprette og redigere råvarebatches
M03	Systemet skal kunne oprette og redigere produktbatches
M04	Systemet skal kunne oprette og redigere recepter
M05	Systemet skal kunne oprette og rediger bruger
M06	Systemet skal ikke kunne slette bruger men skal markerer dem som inaktive
M07	Systemet skal kunne afveje råvarer gennem en vejeterminal

M08	Systemet skal kun tillade de rigtige roller adgang til de rigtige funktioner
M09	Systemet skal kunne vise en brugerliste
M10	Systemet skal kræve at brugeren angiver rolle på forsiden
M11	Systemet skal garanterer at råvare, råvarerbatch, recepter, produktbatches og brugere alle har unikke ID
M12	Systemet skal holde styr på produktbatches status (Oprette/Under produktion/Afsluttet)
M13	Systemet skal kunne åbne og lukke forbindelse til enten en virtuel eller en virkelig vægt, og disse skal kunne snakke frem og tilbage mellem vægten og systemet
M14	Vægten skal kunne bruge kommandoerne S, T, D, DW, p111, RM20 8
M15	Vægten skal kunne ud fra operatørnummre finde operatorens navn
Should have	
S01	Systemet skal holde styr på den aktuelle mængde råvare der er blevet brugt i et givent produktbatch
S02	Systemet skal kunne holde styr på tilbageværende mængde af råvarerbatch
S03	Systemet skal sørger for at recepter har tolerance på alle råvarer
Could have	
C01	Systemet skal holde styr på produktbatches oprettelses dato
C02	Systemet skal lave en produktionsforskrift som kan printes
Won't have	
W01	Systemet skal kræve login fra bruger

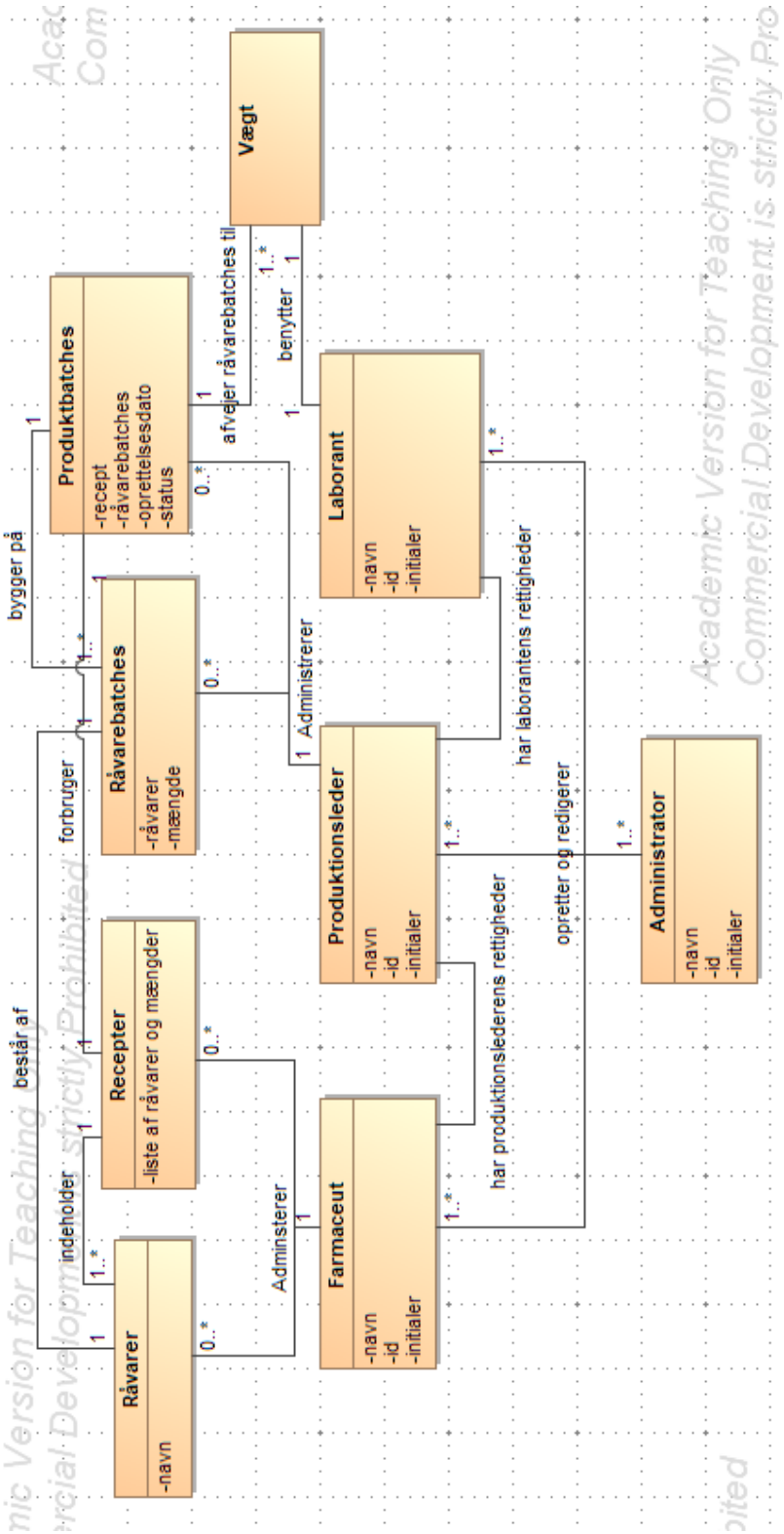
ID	**Ikke-funktionelle krav**
Must have	
IF01	Systemet skal være implementeret som en Single page application med en REST-backend og en HTML/CSS/JavaScript baseret frontend.
IF02	Data skal lagres i en persistent database mellem sessioner.
IF03	Der skal sendes til/fra Rest-backenden serialiseret på en måde (eks. JSON) så andre kan anvende servicen
IF04	Modulet skal kunne tilgås via. en webbrowser
IF05	Vægten skal tale med databasen gennem websidens REST API.

3.4 Domænemodel - Theodor

Vi har foretaget en navneordsanalyse og fundet alle navneord relevante for opgaven. Derefter har vi sorteret dem i klasse-kandidater og attribut-kandidater.

Klasser	**Attributter**
Råvarer	leverandører
Råvarebatches	mængde
Recepter	id
Produktbatches	navn
Farmaceut	initialer
Produktionsleder	tolerance
Laborant	dato
Administrator	status
Vægt	

Domænediagrammet viser klasserne og deres attributter, som de er i virkeligheden og beskriver forholdet mellem dem. Domænediagrammet ligger til grund for bl.a. designklassediagrammet.



Figur 1: Domænediagram

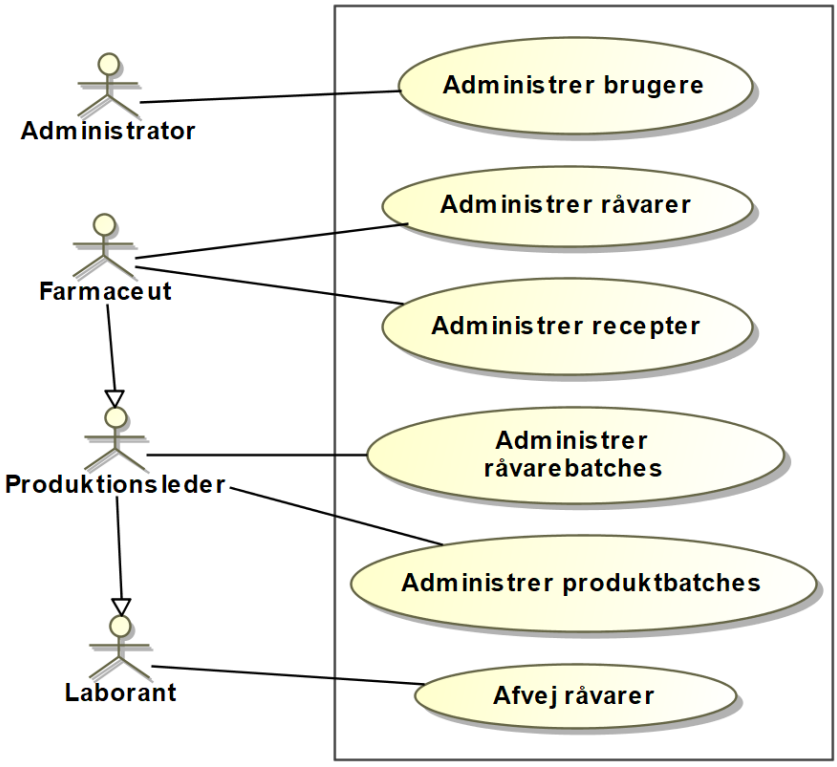
4 Analyse

4.1 Aktører - Jonatan

Aktørtype	Aktører
Primære	Administrator, farmaceut, produktionsleder og laborant
Supporterende	
Offstage	

4.1.1 Use case diagram - Andreas

Vi har i alt fire aktører, administrator, farmaceut, produktionsleder og laborant. Administratoren sørger for at oprette, redigere og deaktivere brugere i systemet. Farmaceuten kan oprette, redigere og deaktivere råvarer og recepter. Farmaceuten kan også det samme som produktionslederen, som kan bestille nye råvarebatches og oprette dem i systemet og igangsætte produktionsbatches. Produktionslederen kan også det samme som laboranten, der kun kan afveje råvare.



Figur 2: Usecase diagram

4.2 Use case - Aleksander

Use cases er taget fra opgavebeskrivelsen og er blevet gentegnet som "fully dressed"efter behov.

4.2.1 Brugeradministration

Den overordnede administration af brugere i systemet foretages af administrator aktøren. Denne skal kunne oprette, rette, fjerne og vise brugere i systemet. Ved oprettelsen angives brugerens ID (entydigt), navn, initialer.
En bruger, der én gang er oprettet i systemet, kan ikke slettes men kan inaktiveres.

4.2.2 Råvareadministration

Administrationen af råvarer i systemet foretages af farmaceut aktøren. Denne skal kunne oprette, rette samt vise råvarer i systemet.
En råvare defineres ved et råvareNr (brugervalgt og entydigt) og navn. (d.v.s. råvareNr skal IKKE autogenereres.)

4.2.3 Receptadministration

Administrationen af recepter foretages af farmaceut aktøren. Denne skal kunne oprette samt vise recepter i systemet.
En recept defineres ved et receptNr (brugervalgt og entydigt), navn samt en sekvens af receptkomponenter.
En receptkomponent består af en råvare (type), en mængde samt en tolerance.

4.2.4 Råvarebatchadministration

Administrationen af råvarebatches i systemet foretages af produktionslederen. Denne skal kunne oprette samt vise råvarebatches i systemet.
En råvarebatch defineres ved et råvarebatchNr (brugervalgt og entydigt) samt mængde og leverandør.

4.3 Casual og fully-dressed use cases - Theodor

Use case: Administrere produktbatches
ID: UC5
Kort beskrivelse: Produktionslederen opretter produktbatches i systemet
Primære aktøre: Produktionslederen
Preconditions: Råvarerne og råvarebatches er oprettet i systemet
Main flow: 1. Produktionslederen definerer produktBatchNr, receptNr, dato for oprettelse, status for batchen hhv. oprettet/under produktion/ afsluttet. 2. Afvejningsresultater for enkelte receptkomponenter gemmes som en produktbatchkomponenent, i takt med produktet fremstilles. 3.Når produktbatchet er oprettet udprintes det på papir og uddeles til en udvalgt laborant, der herefter har ansvaret for produktionen.
Postconditions: Når produktbatchet er oprettet kan laboranten gå i gang med at producere det.
Alternative flows:

Fully dressed

I arbejdet med use cases kan der benyttes tre forskellige grader af beskrivelse. Her er der tale om "brief", "casual"og "fully-dressed". Brief er den mindst omfattende og fully-dressed er den mest omfattende.

Use case: Afveje råvarer
ID: UC6
Kort beskrivelse: Laboranten afvejer råvarer fra råvarebatches til fremstilling af produktbatches
Primære aktøre: Laborant
Preconditions: Produktionslederen har defineret et nyt produktbatch, der er blevet udprintet og uddelt til laboranten
Main flow: 1. Laboranten indtaster sit laborantNr i vejeterminalen. 2. Vejeterminalen svarer tilbage med laborantens navn. 3. Laboranten indtaster nummeret på produktbatchet. Produktbatchets status ændres fra 'oprettet' til 'under produktion'. 4. Vejeterminalen beder laboranten om at placere en beholder. 5. Beholderens vægt registreres. 6. Vægten tareres og tarabelastningen gemmes. 7. Vejeterminalen oplyser, hvilken råvare der skal afvejes. 8. Laboranten henter råvaren og oplyser råvarens batchnummer. 9. Laboranten afvejer råvaren indenfor tolerancen. 10. Vægten udfører brutto-kontrol og registrerer afvejningsresultatet i *produktbatchkomponenenten*. Trin 7-10 gentages for alle råvare i produktbatchet.
Postconditions: Produktbatchets status ændres fra 'under produktion' til 'afsluttet'
Alternative flows: 2.1 laborantNr eksisterer ikke og terminalen giver en fejlmeddelelse og beder om laborantNr igen. 3.1 Produktbatchet findes ikke eller er allerede afsluttet, så vejeterminalen giver fejlmeddelelse og beder om produktbatch nummeret igen. 8.1 Batchnummeret er ukendt eller batchet er opbrugt. så vejeterminalen giver fejlmeddelelse og beder om batchnummeret igen. 10.1 Brutto-kontrollen giver ikke det ønskede resultat. Laboranten kontrollerer for fejl.

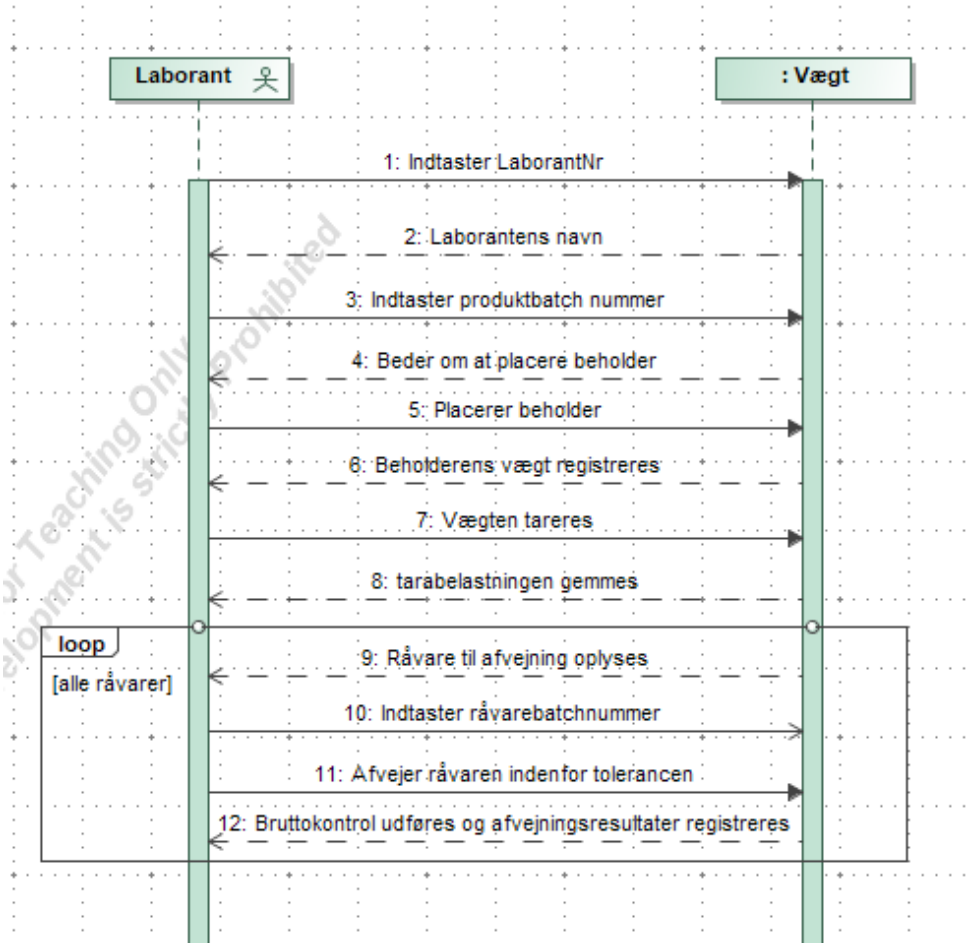
4.4 Traceability matrix: Use cases og krav - Josephine

I følgende tabel ses sammenhængen mellem use cases og krav, de grønne er dem, der er implementeret, de røde er endnu ikke implementeret.

Krav \ Use Case	UC01	UC02	UC03	UC04	UC05	UC06
M01		x				
M02				x		
M03					x	
M04			x			
M05	x					
M06	x					
M07						x
M08	x	x	x	x	x	x
M09	x					
M10	x					
M11	x	x	x	x	x	x
M12					x	
M13						x
M14						x
M15						x
S01		x				
S02				x		
S03			x			
C01					x	
C02					x	

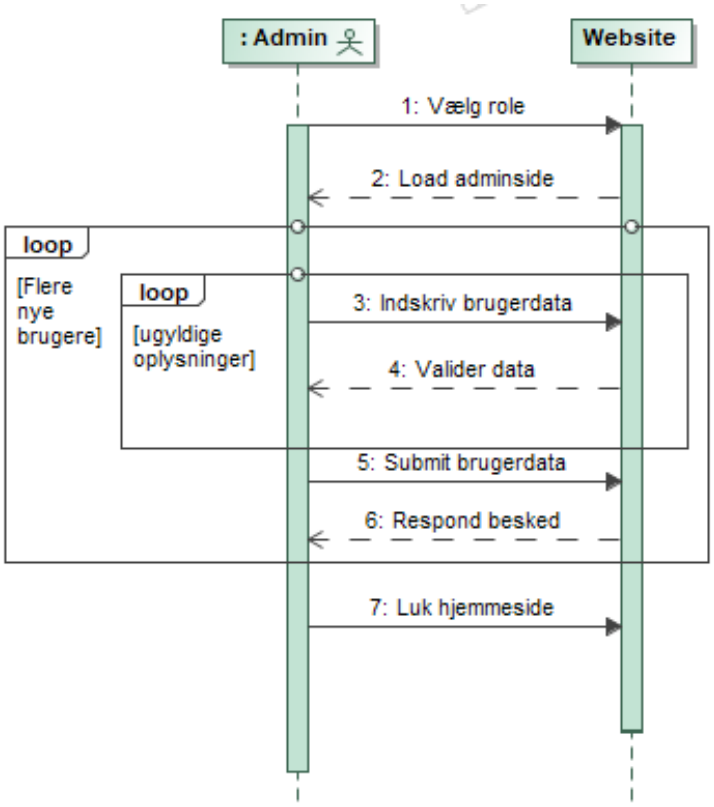
4.5 System sekvensdiagram - Theodor, Søren, Jonatan og Andreas

Dette systemsekvensdiagram viser, hvordan en laborant interagere med vægten. Først vil laboranten indtaste sit id, hvorefter vægten viser laborantens navn. Derefter indtastes produktbatch id, hvor vægten så beder om, at en beholder stilles på vægten. Når en beholder er stillet på vægten gemmes beholders vægt og vægten tarares. Herefter indskrives den råvare, der skal afvejes ved at indtaste råvarebatchid. Så afvejes råvaren indefor tolerancen, bruttokontrol udføres og vægten af råvaren registreres. Gentag for alle råvare i recepten.



Figur 3: Vægt systemsekvens

I dette systemsekvensdiagram vises en administrators interaktion med websiden. Først vælger brugeren at være en administrator, hvorefter administrator siden vises. Her indtaster administrator brugerdata for en ny bruger. Derefter validere websiden den indtastede data, hvis den ikke er valid, skal administratoren prøve igen. Hvis dataen er valid kan administratoren gemme den, hvorefter websiden retunere en om besked om, at den er blevet gemt. Gentag for eventuelle andre nye brugere. Administratoren lukker websiden, når de er færdig.



Figur 4: Admin opretter bruger

5 Design

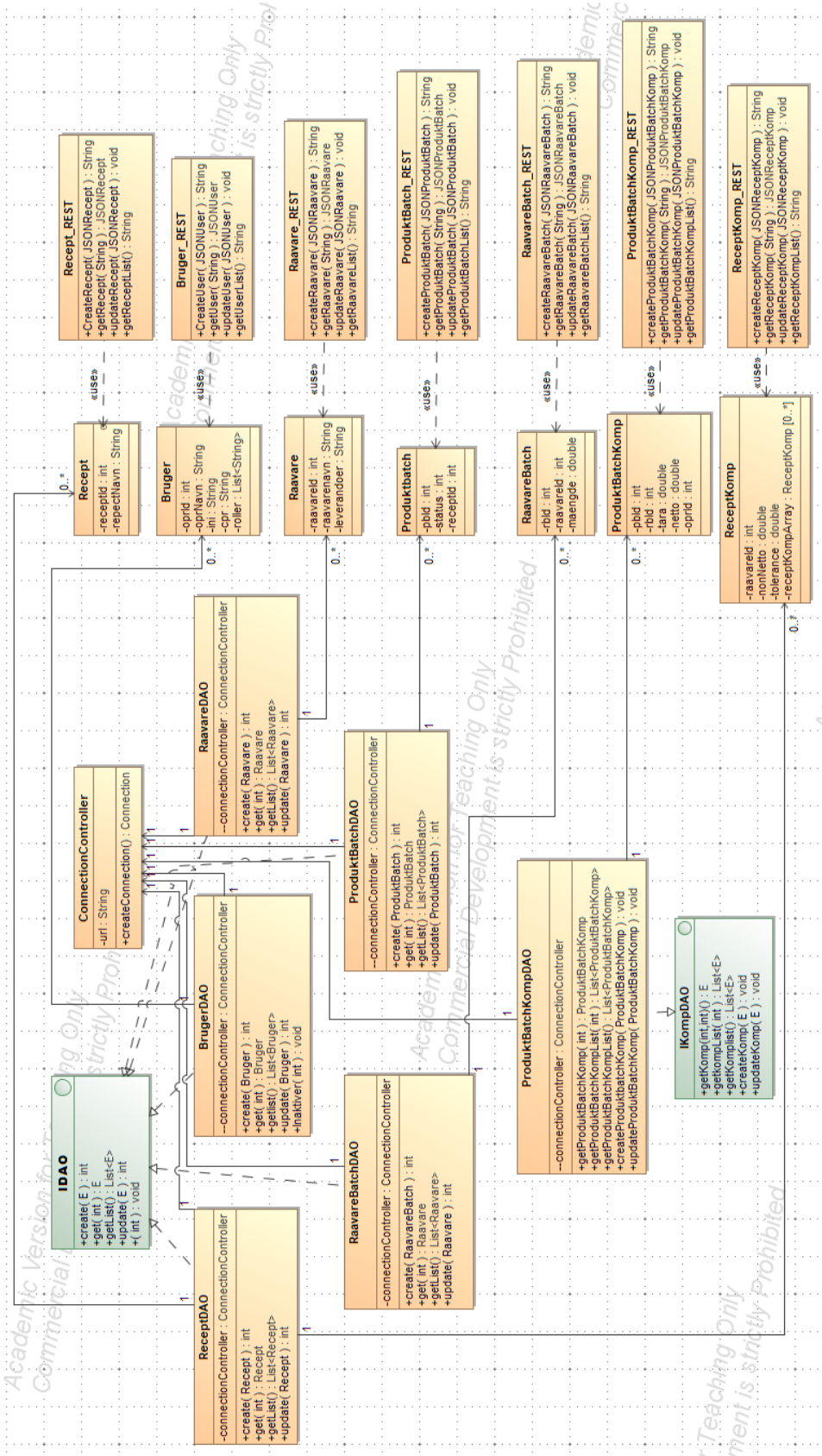
5.1 Hjemmeside - Theodor

Vi har nøje overvejet brugervenlighed i vores design af hjemmesiden. Vores mål har været, at gøre det nemt at få fat i de oprettelses og redirengsmenuer, hver rolle må bruge. Derfor har vi brugt en singlepage application.

5.2 Klassediagram - Theodor

Design klassegrammet er udgangspunktet for programmet. Designklassediagrammet er udarbejdet ud fra domænemodellen og use cases. Det er et statisk UML diagram. Vi har inkluderet attributter og metoder, såvel som klassens navn og multiplicitet i vores klassediagram. Desuden viser designklassediagrammet relationerne klasserne imellem.

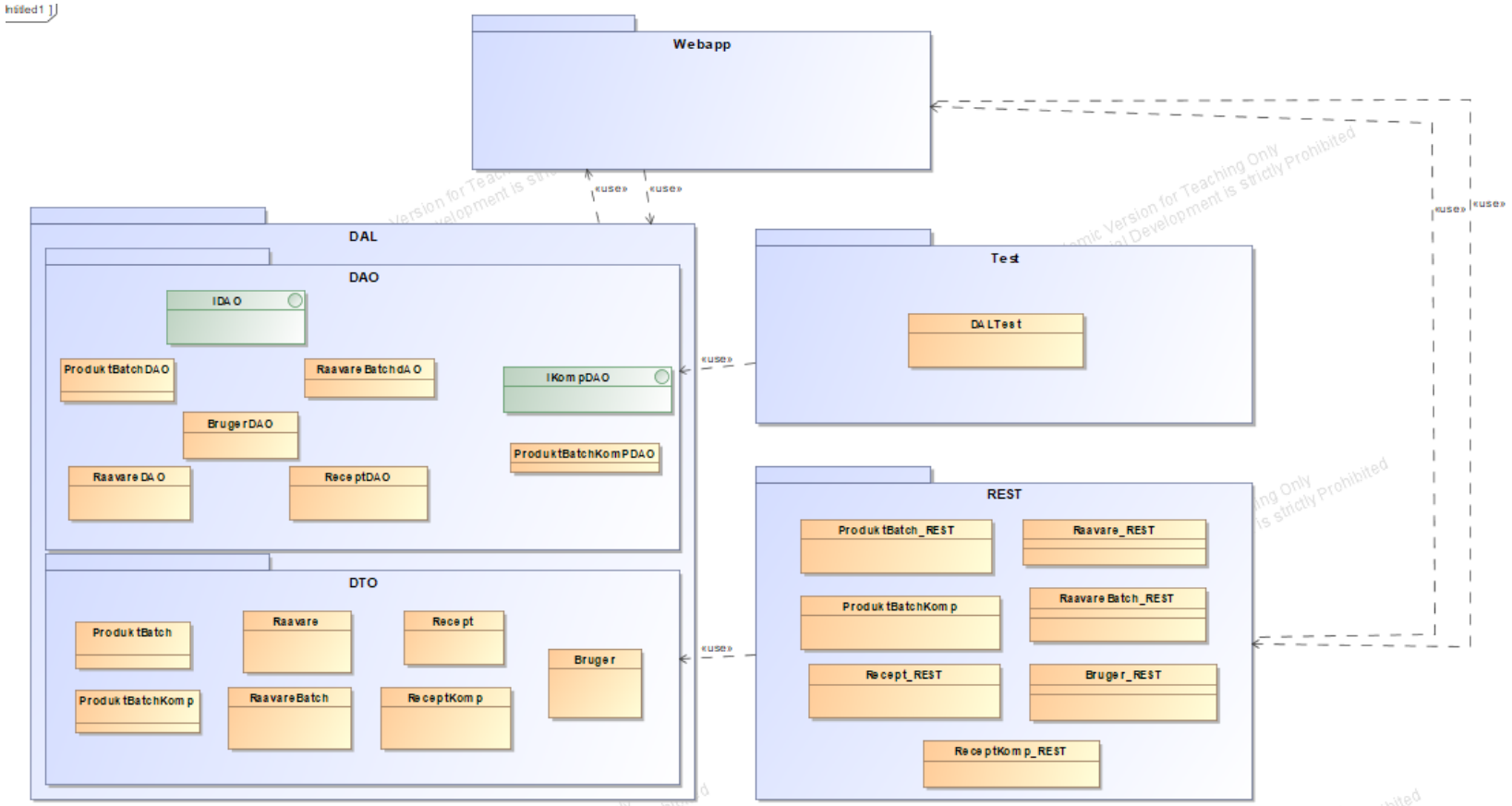
Programmet kan deles op i DAO'er, DTO'er, REST og nogle hjælpeklasser. DAO'erne overholder interfaces hhv. IDAO og IKompDAO. DAO'erne får deres connections til databasen fra en ConnectionController klasse. DTO'erne bruges også af REST klasserne til at få data fra.


IDAO
+create(E) : int
+get(int) : E
+getList() : List<E>
+update(E) : int
+(int) : void
ConnectionController
-url : String
+createConnection() : Connection
ReceptDAO
-connectionController : ConnectionController
+create(Recept) : int
+get(int) : Recept
+getList() : List<Recept>
+update(Recept) : int
RaavareBatchDAO
-connectionController : ConnectionController
+create(RaavareBatch) : int
+get(int) : Raavare
+getList() : List<Raavare>
+update(Raavare) : int
BrugerDAO
--connectionController : ConnectionController
+create(Bruger) : int
+get(int) : Bruger
+getList() : List<Bruger>
+updateI(Bruger) : int
+inaktiver(int) : void
RaavareDAO
--connectionController : ConnectionController
+create(Raavare) : int
+get(int) : Raavare
+getList() : List<Raavare>
+updateI(Raavare) : int
ProduktBatchDAO
--connectionController : ConnectionController
+create(ProduktBatch) : int
+get(int) : ProduktBatch
+getList() : List<ProduktBatch>
+update(ProduktBatch) : int
ProduktBatchKompDAO
--connectionController : ConnectionController
+getProduktBatchKomp(int) : ProduktBatchKomp
+getProduktBatchKompList(int) : List<ProduktBatchKomp>
+getProduktBatchKompList() : List<ProduktBatchKomp>
+createProduktbatchKomp(ProduktBatchKomp) : void
+updateProduktBatchKomp(ProduktBatchKomp) : void
IKompDAO
+getKomp(int,int)() : E
+getKompList(int) : List<E>
+getKompList() : List<E>
+createKomp(E) : void
+updateKomp(E) : void
Recept
-receptId : int
-receptNavn : String
Bruger
-oprId : int
-oprNavn : String
-ini : String
-cpr : String
-roller : List<String>
Raavare
-raavareId : int
-raavarenavn : String
-leverandoer : String
Produktbatch
-pbId : int
-status : int
-receptId : int
RaavareBatch
-rbId : int
-raavareId : int
-maengde : double
ProduktBatchKomp
-pbId : int
-rbId : int
-tara : double
-netto : double
-oprId : int
ReceptKomp
-raavareId : int
-nonNetto : double
-tolerance : double
-receptKompArray : ReceptKomp [0..*]
Recept_REST
+CreateRecept(JSONRecept) : String
+getRecept(String) : JSONRecept
+updateRecept(JSONRecept) : void
+getReceptList() : String
Bruger_REST
+CreateUser(JSONUser) : String
+getUser(String) : JSONUser
+updateUser(JSONUser) : void
+getUserList() : String
Raavare_REST
+createRaavare(JSONRaavare) : String
+getRaavare(String) : JSONRaavare
+updateRaavare(JSONRaavare) : void
+getRaavareList() : String
ProduktBatch_REST
+createProduktBatch(JSONProduktBatch) : String
+getProduktBatch(String) : JSONProduktBatch
+updateProduktBatch(JSONProduktBatch) : void
+getProduktBatchList() : String
RaavareBatch_REST
+createRaavareBatch(JSONRaavareBatch) : String
+getRaavareBatch(String) : JSONRaavareBatch
+updateRaavareBatch(JSONRaavareBatch) : void
+getRaavareBatchList() : String
ProduktBatchKomp_REST
+createProduktBatchKomp(JSONProduktBatchKomp) : String
+getProduktBatchKomp(String) : JSONProduktBatchKomp
+updateProduktBatchKomp(JSONProduktBatchKomp) : void
+getProduktBatchKompList() : String
ReceptKomp_REST
+createReceptKomp(JSONReceptKomp) : String
+getReceptKomp(String) : JSONReceptKomp
+updateReceptKomp(JSONReceptKomp) : void
+getReceptKompList() : String
«use»
0..*
1
Academic Version for Teaching Only
Commercial Development is strictly Prohibited

Figur 5

5.3 Pakkediagram(Midlertidigt) - Theodor

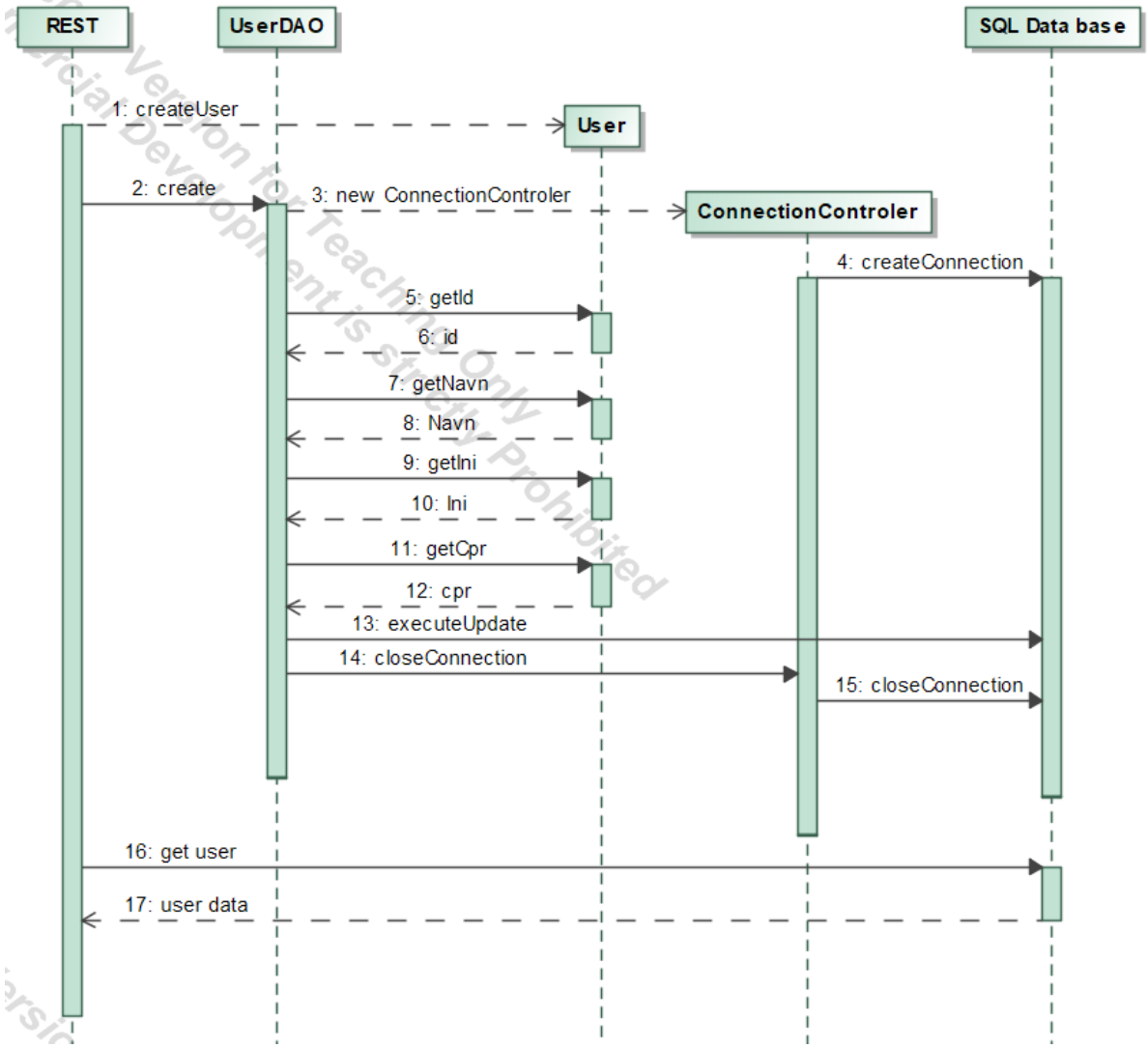
Pakkediagrammet giver overblik over de forskellige pakker i programmet og hvilke pakker der har med hinanden at gøre.



Figur 6

5.4 Sekvensdiagram - Andreas

Dette sekvensdiagram viser flowet i den del af programmet, hvor en bruger bliver oprettet. Først bliver alle indskrevne data fra hjemmesiden sendt til Java delen af programmet, hvor et User objekt bliver oprettet ud fra de indskrevne data. Herefter bliver create metoden i UserDAO kaldt, som opretter et ConnectionControler objekt, der så opretter en forbindelse til SQL databasen. Derefter kalder create metoden gettere, i User klassen, for id, navn, ini og cpr, som alle returnere de forskellige data. Efter dette overføres dataen til SQL databasen. Tilsidst henter hjemmesiden dataen fra SQL databasen.



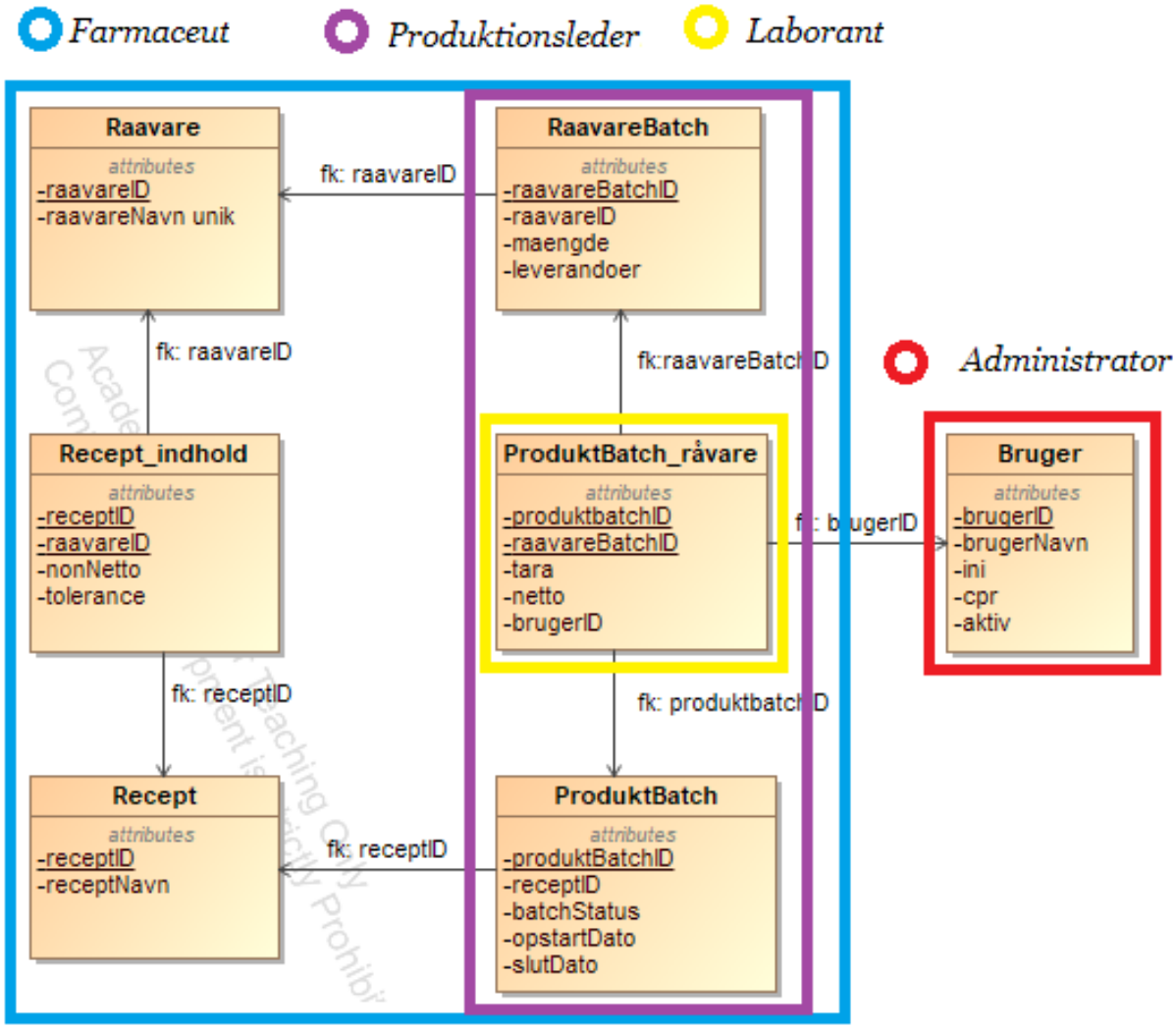
Figur 7

5.5 Database design - Søren

For at lave et fornuftigt design af databasen, er der undersøgt hvad der skal administreres. Dette er brugere, råvarer, recepter, råvarebatchs og produktionsbatchs, derfor er disse 5 oprettet som tabeller med relevante tabelnavne. Hver recept har dog flere råvare for hver recept, så der er oprettet endnu en

tabel for at holde styr på dette kaldt Recept_indhold. Det samme gælder produktbatchs og råvarebatchs, så derfor er der oprettet tabellen Produktbatch_indhold (navnet er rettet efter billedet)

Hver af de 5 ting der skal administreres, benytter kun en primary key, som kandidat nøgle, så det er let at



Figur 8: MIDLERTIDIGT SKEMADIAGRAM! UDSKIFT MED DET UDEN FARVER EFTER MIL 1

6 Implementering

6.1 Overblik - Theodor

Vi har fulgt opgaveformuleringens forslag og lavet DTO(Data Transfer Object) og DAO(Data Access Object) klasser. DTO og DAO er i hver sine packages med samme navne, de packages er i en DAL(Data access layer) package. Alle DAO'erne undtagen ProduktBatchKompDAO overholder IDAO interfacet. ProduktBatchKompDAO overholder interfacet IKompDAO, fordi IKompDAO bruger method overloading, dvs. indeholder to getList metoder, som IDAO ikke gør. IDAO og IKompDAO gør brug at generics, så vi kan bruge de samme interfaces med forskellige klasser. DAO'erne indeholder den kode, der skal til for at hente ting fra databasen ind i java og omvendt. De metoder er create, get, getList, update og delete. DTO'erne definerer de java objekter, vi har brug for, for at modellere de objekter, der findes i medicinproduktionen.

Så har vi en hjemmeside, der udover DAL består af en REST package med rest klasser for hver DTO klasse. Vores løsning bruger en persistent database i form af s185103 MySQL database. Vi har en test package med DAO tests. Der er en webapp mappe med HTML filer, CSS filer og javascript filer, der udgør vores hjemmeside.

Vi har også et afvejningsprogram, der udover DAL består af en weight package.

6.1.1 Hjemmeside - Theodor

Vi har valgt at benytte single-page applications arkitektur, det betyder at alle html filer loades ind i index.html når de skal bruges. Vi har brugt CSS til at ændre html grafiske udtryk og øge brugervenligheden. Vi har brugt java-script til at lave funktionalitet og få den rette data ind på siden og ind i databasen. På brugersiden er der i oprettelsesfelt

6.1.2 Afvejning - Theodor

Afvejningsprogrammet består, lige som hjemmesiden, af en DAL pakke med DTO'er og DAO'er, der er identiske med hjemmesidens. Forskellen er en weight package, hver i der er en WeightConnector klasse og en WeightController klasse. WeightConnector klassen har en konstruktør der opretter en socket til localhost på port 8000. Den indeholder metoden callWeight, der laver PrintWriter, der kommunikerer med vægten og en BufferedReader, der modtager en inputStream fra vægten. Derudover udskriver den kommandoerne i konsollen. Fordi vægten ikke er som simulatoren og laver for os overflødig kommunikation, har vi lavet en listen metode, der benytter et while loop og et if-statement med en contains metode til at finde den string der skal bruges. Der er også en modificeret callWeight metode kaldet SCallWeight. Den findes fordi commandS() bruger den, fordi den skal returnere en double og dermed skal bruge input fra vægten, så hvor der i callWeight() er en in.readLine() er den flyttet ned i commandS(). Resten af klassen består af metoden, der bruger callWeight til at udføre en af kommandoerne beskrevet i https://docs.google.com/presentation/d/1Z0ELvdqur66q1fYCKN6YDwwoqr2ISwxB5B_7iL9zVYM/edit

Den anden klasse er WeightController, der indeholder den store metode afvejning(). Derudover er der 4 hjælpemetoder. 2 af dem laver vægtens kommunikation om til brugbare variable og de andre 2 tjekker om en DTO findes i databasen. Afvejning() deklarere alle simple variable og objekter i toppen af metoden. Det første der sker i metoden er at brugen bliver bedt om at indtaste sit ID, så tjekkes det om ID matcher med et ID i databasen. Så findes navnet der passer med ID'et frem. Så skal laboranten be- eller afkræfte om der er det rette navn. Hvis ikke starter hele processen om igen, ved hjælp af et do while loop.

Samme struktur går igen for prodkuktBatchID'et.

7 Test

7.1 Positiv test

Funktionelle krav	Positiv test	Bestået
M01	Systemet skal kunne oprette og redigere råvarer	
M02	Systemet skal kunne oprette og redigere råvarebatches	
M03	Systemet skal kunne oprette og redigere produktbatches	
M04	Systemet skal kunne oprette og redigere recepter	
M05	Systemet skal kunne oprette og rediger bruger	
M06	Systemet skal ikke kunne slette bruger men skal markerer dem som inaktive	
M07	Systemet skal kunne afveje råvarer gennem en vejeterminal	
M08	Systemet skal kun tillade de rigtige roller adgang til de rigtige funktioner	Bestået
M09	Systemet skal kunne vise en brugerliste	
M10	Systemet skal kræve at brugeren angiver rolle på forsiden	Bestået
M11	Systemet skal garanterer at råvare, råvarerbatch, recepter, produktbatches og brugere alle har unikke ID	
M12	Systemet skal holde styr på produktbatches status (Oprette/Under produktion/Afsluttet)	Bestået
M13	Systemet skal kunne åbne og lukke forbindelse til enten en virtuel eller en virkelig vægt, og disse skal kunne snakke frem og tilbage mellem vægten og systemet	Bestået
M14	Vægten skal kunne bruge kommandoerne S, T, D, DW, p111, RM20 8	
M15	Vægten skal kunne ud fra operatørnummre finde operatorens navn	
S01	Systemet skal holde styr på den aktuelle mængde råvare der er blevet brugt i et givent produktbatch	
S02	Systemet skal kunne holde styr på tilbageværende mængde af råvarerbatch	
S03	Systemet skal sørger for at recepter har tolerance på alle råvarer	
C01	Systemet skal holde styr på produktbatches oprettelses dato	
C02	Systemet skal lave en produktionsforskrift som kan printes	

7.2 Negative test

7.3 Unit-test - Theodor

Unit-tests er automatiske tests. Vi har brugt JUnit4 i intelliJ, til at lave vores unit-tests. En typisk test bruger en assertEquals metode, for at se om metodens output er det samme som forventet output. Hvis det virkelige resultat og det forventede stemmer overens er testen bestået. Unit-tests kan findes i mappen Test under src i programmet.

Figur 9: Coverage Test

Element	Class	Method	Line
	%	%	%
	%	%	%
	%	%	%
	%	%	%
	%	%	%

Tabel 6: Coverage Test

Vi har lavet forskellige unit-tests, blandt andet en for hver DAO klasse, for at teste create, get, getlist, update og delete metoderne.

Test Case ID	**Beskrivelse**	**Status**
TC01	Test af UserDAO	Bestået
TC02	Test af ReceptDAO	Bestået
TC03	Test af RaavareDAO	Bestået
TC04	Test af RaavareBatchDAO	Bestået
TC05	Test af ProduktBatchKompDAO	Bestået
TC06	Test af ProduktBatchDAO	Bestået

7.4 Coverage test

8 Projektplanlægning

8.1 Planlagt forløb - Josephine og Theodor

I dette projekt har vi besluttet os for en fremgangsmåde som gerne skulle gøre processen mere flydende, end hvad vi tidligerer har haft. Vi vil fra start af i projektet gennemgå og diskuterer diagrammerne, der skal indgå i vores rapport. Desuden vil vi for at opnår en god rapport, gode diagrammer og undgå waterfall metoden, starte med at lave analyse afnittet derefter design afsnittet, før vi begynder at lave koden. Desuden vil vi undgå solokodning(at en bruger dag og nat og får skrevet en masse kode i et hug). Vi vil arbejde iterativt og prioritere de vigtigste use cases og krav og lægge dette ind i vores tidsplan. Vi vil så lave vores Master og Develoment branch i intelliJ, og undlade og pushe kode der giver errors til dem. Lave kravliste med færre "must have"krav end tidligere. Ud fra krav opstilles tests tidligt, så vi løbende kan teste vores program.
For at sikre kvaliteten af vores diagrammer har vi planlagt at gøre som følger:
Designklassediagram før og efter implementering med attributter, metoder, multiplicitet og forklarende tekst. Package diagram for at give overblik og designklasse for at uddybe. Package diagram indeholder MVC og Test osv. Systemsekvensdiagram for en central vej igennem programmet og ligeledes for sekvensdiagrammet.

Til milestone 1 vil vi have følgende klar: Skemadiagram, use cases, krav, midlertidigt design af hjemmesiden, domæne model, flowchart over web processen, tegning af frontend, frontend hjemmeside med felter semi implementeret, design klasse diagram, systemsekvens diagram, dele af rapport.

Til milestone 2 vil vi have følgende klar: Database, DAO'er, DTO'er, frontend, Junit begyndt, vægt, design klasse diagram, java Script.

Før afleveringsfrist: Test og finpudsning, JUnit test færdig, Konklusion.

8.2 Reelt forløb - Jonatan IKKE FÆRDIGT!!!

Starten af vores projekt gik præcis som planlagt, vi startede med at lave analyse delen og derefter designdelen. I den periode begyndte folk at splitte af rapporten, så koden begyndte at blive af højere og højere prioritet. Med koden begyndte nogle at starte med frontend primært Aleksander, mens resten startede med database kodning og lave java objekterne. Da DTO'erne og DAO'erne var begyndt at tage form, så startede vi allerede der, med at lave UnitTest til DAO'erne. I perioden her, fik vi også sat hele databasen op, med tabeller, variable, keys og constraints. Ved dette punkt kom Milestone 1 mødet. Med dette møde fik vi en bedre ide om hvordan projektet stod, og hvor godt med vi var.

Efter dette møde har vi fået arbejdet videre med hele frontenden, og vi har dermed nogle af hjemmesiderne, til at kunne snakke med java koden, og dermed også med databasen. Vi har fokuseret på brugervenlighed og overskuelighed, og havde en grundlæggende ide om hvordan hjemmesiderne skulle se ud, før vi startede med at lave dem. Derudover har vi fået lavet UnitTest til alle vores DAO'er, så vi ved at linket mellem java og databasen, ikke kan give os nogle fejl længere henne.

9 Konklusion

10 Bilag