20 CDIO Final

02324 Advanced Programming
Project title: CDIO Final
Group number: 20
Deadline: 16. Jun. 2017 - 12:00
Version: 1

This report contains 57 pages, including this page.

Student nr., Surname, First name	**Signature**
S165248, Gadegaard, Theis 	_____
S165208, Højsgaard, Tobias André 	_____
S165209, Jønsson, Danny 	_____
S165227, Petersen, Gustav Hammershøi 	_____
S145005, von Scholten, Frederik 	_____

Work Distribution / Timetable

Participant	Analysis	Design	Impl.	Test	Doc.	Other	Total Hours
Gadegaard, Theis	5		35,5		5	0,5	**46**
Højsgaard, Tobias André	1		29,5	9,5	10	1	**51**
Jønsson, Danny	6		20,5	2	9	1	**38,5**
Petersen, Gustav Hammarshøi	2		35		9,5	0,5	**47**
von Scholten, Frederik	8	1	30,5	2	11,5	1	**54**
Total	**22**	**1**	**151**	**13,5**	**45**	**4**	**236,5**

Installation/Startup guide

Installation Guide

Java Development Kit
In order to develop applications in Java, you will need Java Development Kit on your computer. Therefore you are to download Java Developtment Kit 8 for the desired operating system at
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html .
Now run the .exe and follow the installation guidelines on the screen.

Eclipse IDE
Eclipse IDE (Integraded Development Environment), is used to compile, run and test your software. It's mostly used for Java but can be used for other programming languages with the need of plug-ins. We use it for our development, as we are using Java as our main language.

To download Eclipse, simply go to https://www.eclipse.org/downloads/ and download Eclipse Neon, which is the latest version available at this time.
Now run the .exe file and follow the guidelines.

REST with JAVA using Jersey
Download the Jersey distribution as zip file from https://jersey.java.net/download.html (the zip contains the Jersey implementation JAR and its core dependencies). Copy all JARs from your download into the project.

Apache Tomcat Installation
Download Apache Tomcat 8 from http://tomcat.apache.org/download-80.cgi (we are using Apache Tomcat version 8.5). Download the windows 64-core file. Unpack the file to a suitable place where you can find i easily (ex. c:/tomcat) - no further installation is needed you only need the file.

NOTE: Download the installation file corresponding the operation system you are running if you are running another OS. You can try installing Tomcat 9 but version 8.5 is advised as we know it is a working version.

Now you have to add the server-path in eclipse. Select the server-tab and choose Windows → Show View → Servers.
Add the server by using the link and right-click the the server-windows → New → server and

Now locate the Tomcat folder, choose Tomcat v.8.5 Server and click finish.
Congratulations! The development-environment is now ready.

Startup Guide

Running the Web Application

Runningen the application from Eclipse right click the project and chose *Run As* then *Run on Server*. If this is the first time running it then tick *Manually define a new server* select *Apache* and then *Tomcat v8.5 Server* click next and finish. By default eclipse will run the application in its build in browser. Copy/paste the url (http://localhost:8080/20cdioFinal/) to your browser fx. Chrome to be able to all of the features. Just login - the admin user and password will be auto filled.

NOTE! It is recommended to use a modern browser fx. Firefox or Chrome in a version that is not older than 2-3 years.

General usage

Web Application

Log in at http://localhost:8080/20cdioFinal/ in by pressing submit at the screen. Credentials should already have been filled in, but they can be changed if it desired to log in as a different user.

When logged in, press the radio buttons in the top left of the page to switch between different views. The name of the view is to the right of the corresponding button.

When in the desired view, use the button add to add a new element to the view or press the edit or update buttons to change the values of an already existing element. In both cases, the page will be redirected to a form in which the desired values can be filled in. Press submit to complete the operation. Some views are only designed to show the elements and have neither add nor edit buttons.

Weight

We haven't used a real weight for this project so we recommend using the simulator to reduce errors. Start using the weight by running the weightsimulator application (run the same program as the web application, but do not choose run on server. Instead choose java application) and connect to it using the program named "Putty" which is free to download from the internet. Insert "localhost" or your localhost ip into the "Host Name" input field and "8000" into the port field and set the connection type to RAW, now you should be able to connect to the database and get a console window. In the console window run the command "RM208" and the weighting process should begin. After this, just use the weight GUI.

Table of Contents

1 Introduction

1.1 Quick Project Overview

A medicinal company wishes to have a system developed that can take care of weighing commodities and registering the amount used. This system also needs a web-based graphical user interface with several administrative views. From these, it should be possible to administrate users and create new prescriptions, which is a recipe for a given product that the company could produce.

1.2 Project Plan

This project has three main parts.

- A web-based user interface with administrative views for various parts of the company, including users, prescriptions and commodities.
- A WCU (Weight Control Unit), for controlling the process of weighing an item and registering the usage of the given product.
- A database able to store all of this information, with a data access layer applying the indirection pattern.

Our plan is to split into three groups, one group works on the user interface, another on the WCU and the last on the database. Each group then works on their part of the project. As they finish, they will integrate their work into the work of one of the other groups and assist the members of this group with their work until all parts of the project have been merged into one unit and everybody works on testing.

The groups we decided on in the beginning of the project were:

- Danny Jønsson and Frederik von Scholten in charge of the web-based user interface
- Theis Gadegaard and Gustav Petersen in charge of the WCU
- Tobias Højsgaard in charge of the database and data access layer, later assisting the web-interface group when completing this task.

2 Analysis

2.1 Definitions

The system is the software that we have been tasked with developing, it encompasses all the three parts of the project described in section 1.2-project plan.

The company is the medicinal firm requesting the system.

A product (produkt) is what the company sells. It is a collection of specific ingredients in specific amounts dictated by a recipe.

Commodities (råvare/raavare) are the resources or ingredients that the company uses in their products.

A prescription (recept) is a recipe for a product that describes the ingredients and amounts of these ingredients that are necessary to make the product.

A user (bruger) is anybody who uses the system. It does not matter which function they have.

The database is the storage for all the information that the system should contain.

WCU (Weight Control Unit) / (ASE (Afvejnings Styrings Enhed)) - The weight control unit is the part of our system that can be used to weigh an amount of a commodity and register this weight to our database.

Views are the different administrative menus that we have implemented.

2.2 System Users / Roles

The system contains four roles which each has different privileges. One user must have at least one role. The roles in the system are as following:

Administrator
The administrator, known as user admin, has the privilege to create and edit users as he wishes. He can grant and revoke roles from users, though they must always have one role. And most important, he can set users as inactive. On the user's perspective it will be the same as a deletion, but the information of the user will remain in the database.

Pharmacist (Farmaceut)

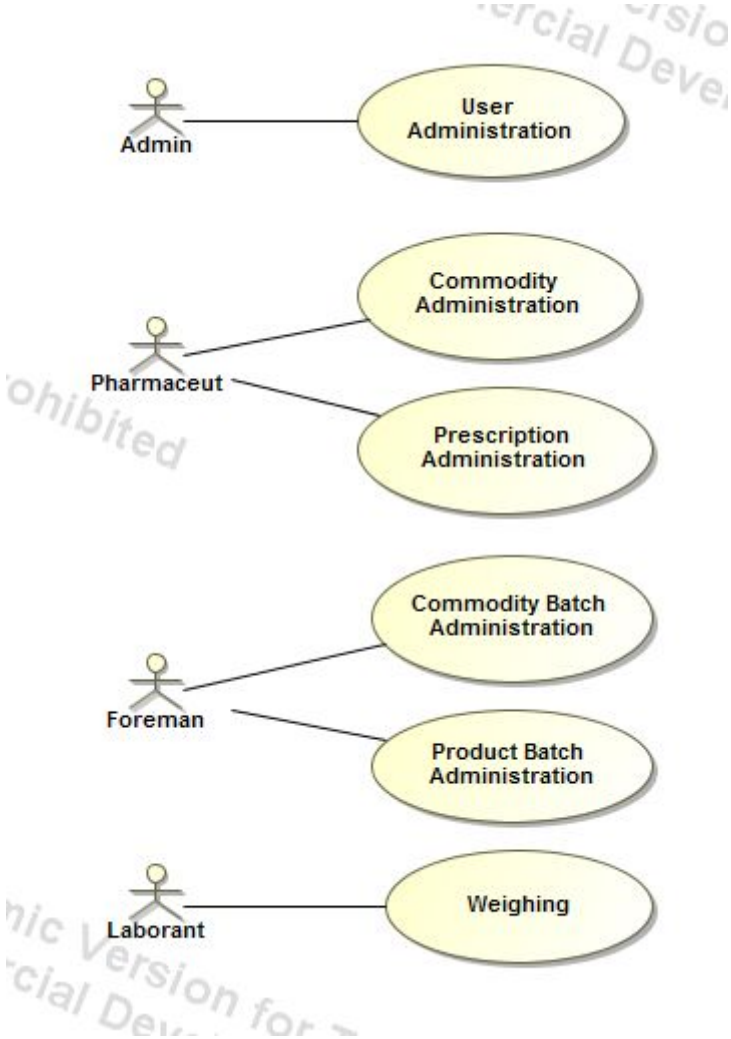
The pharmacist handles the administration of the commodities and prescriptions where they can add and edit the commodities and prescriptions as they wish. Also they work closely with the foreman and has the same rights as they do.

Foreman (Værkfører)
The foreman has the same rights as the pharmacists, but they handle the administration of the commodity batches and product batches, editing the amount of commodities, as well as the status of the prescription batches.

Laborant
The laborants has physical access to the WCU. It is the laborants responsibility to weigh and register each commodity for a given product batch.

2.3 Use Cases

Use Case Diagram



User-Administration

The user administration should allow users with the administrator role to

- Make new users
- Update existing users
- Remove users from the system
- Show the users in the system

Commodity-Administration

The commodity administration system should allow a user with the pharmaceut role to

- Add new commodities to the system
    - The commodity should have an identification number that is unambiguous and ideally chosen by the user.
- Show the existing commodities

Prescription-Administration

The prescription administration system should allow a user with the pharmaceut role to

- Add new prescriptions to the system
- Show the existing prescriptions

Commodity-Batch-Administration

The commodity batch administration system should allow a user with the foreman role to

- Add new commodity batches to the system
- Show the existing commodity batches

Product-Batch-Administration

The product batch administration system should allow a user with the foreman role to

- Add new product batches to the system
- Show existing product batches
- Get a page with all the information regarding a single product batch in a printable format

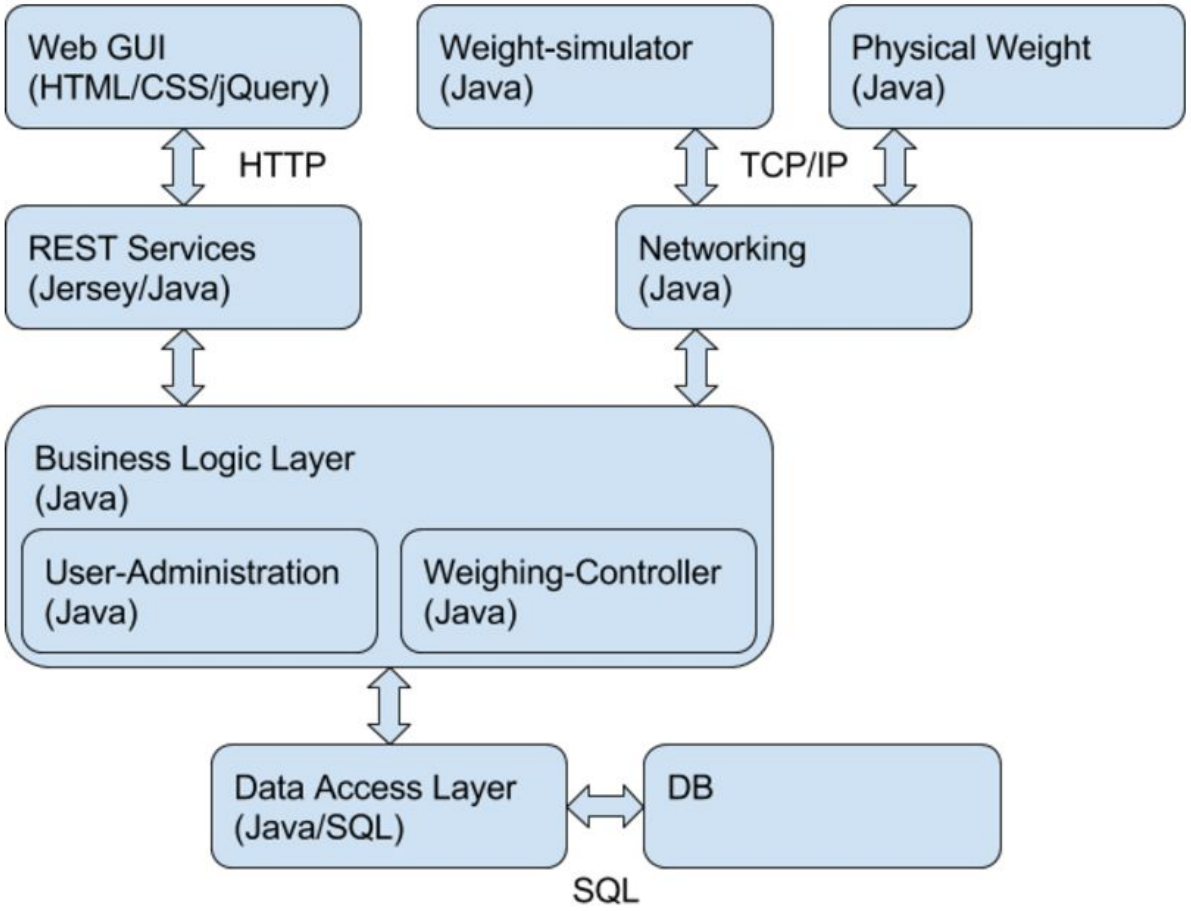
Weighing

The weight control unit should allow a user with the laborant role to

- Weigh each commodity in a product batch
- Register each product batch component in the database with the necessary information

2.2 Domain Diagram

This is a diagram of all the different parts of the system.



2.3 Requirements

2.3.1 Functional Requirements

R1.1	**SYSTEM FUNCTIONAL REQUIREMENTS**
R1.1.1	The system shall consist of following: a. Database b. WebApplication c. Weight-terminal d. Weight-Controller
R1.1.2	The system shall consist of 4 user-roles a. Administrator b. Pharmacist (Farmaceut) c. Foreman (værkfører)

	d. Laborant
R1.1.3	Every user-role has different privileges a. Administrator i. Access to the user administration with following actions 1. Create a new user 2. View current users 3. Edit a user 4. Set a user inactive b. Pharmacist i. Access to the commodity administration with following actions 1. Create a new commodity 2. View commodity 3. Edit commodity ii. Access to the prescription administration with following actions 1. Create prescriptions 2. View prescriptions c. Foreman i. Access to the prescription batches administration with following actions 1. Create a prescription batch 2. View prescription batches ii. Access to the product batch administration with following actions 1. Create a product batch 2. View product batches d. Laborant i. Access only to the weight and all its features
R1.2	**DATABASE FUNCTIONAL REQUIREMENTS**
R1.2.1	The database shall exist on an online server and be accessible through it
R1.3	**WEB APPLICATION**
R1.3.1	Shall implement the following administrations a. User administration (Bruger administration) b. Commodity administration (Raavare administration) c. Prescription administration (Recept administration) d. Commodity batch administration (Raavare batch administration) e. Product batch administration (Produktbatch administration)
R1.3.2	Shall run an authentification of the users in a login-session
R1.3.3	Based on the user's role(s), the permitted pages shall be shown
R1.3.4	All input shall be validated
R1.3.5	Errors shall be shown correctly with appropriate messages
R1.3.6	Shall have access to the database
R1.4	**WEIGHT-TERMINAL**

R1.4.1	Shall be connected through ETHERNET
R1.4.2	Shall communicate with the Weight-Controller
R1.5	**WEIGHT-CONTROLLER**
R1.5.1	The Weight-Controller shall communicate with the weight and the database
R1.5.2	Shall have the functions that are used in the weighing procedure

2.3.2 Usability Requirements

R2.1	The weighing procedure shall be intuitive
R2.2	The system shall be input error tolerant
R2.3	The GUI shall be user friendly

2.3.3 Reliability Requirements

	None

2.3.4 Performance Requirements

	None

2.3.5 Supportability Requirements

R5.1	The system shall support a *Single page application* with a REST-backend
R5.2	The system shall support a *Single page application* with a HTML/CSS/JavaScript based frontend

R5.3	The data layer shall be decoupled with an interface IUserDAO with definitions and statements for the necessary methods
R5.4	The data layer shall be reusable even if changed from REST/Jersey

2.3.6 Implementation Requirements

R6.1	Shall consist of the following administrations a. User administration b. Commodity administration c. Commodity Batch administration d. Product Batch administration e. Prescription administration
R6.2	The software package shall consist of: a. a weight simulator that works as the weight terminal b. a weighing controller unit c. a webapplikation d. a maven pom.xml allowing automatically fetching additional libraries
R6.3	The software package shall have a weighing controller
R6.4	The weighing procedure shall be implemented as it was in the *old* system

3 Design

3.1 Expanded Use Cases

User-Administration

Use case ID:	UC1
Use case name:	Create user
Description:	A function to create new users which in turn gets stored in the server database
Primary actors:	Administrator
Secondary actors:	Database
Preconditions:	The administrator must be logged in.
Postconditions:	A new user is successfully created in the database
Main flow:	1. Admin. initiates create user process 2. Admin. inputs required information into appropriate inputs and ends user creation process 3. Admin. completes the process
Alternate flows:	1. Admin inputs invalid or wrong syntax in inputs 2. An error is thrown and the appropriate error message is shown.

Use case ID:	UC2
Use case name:	Show users
Description:	The admin. has the ability to see all existing users in the server database.
Primary actors:	Administrator.
Secondary actors:	Database.
Preconditions:	The administrator must be logged in..

Postconditions:	All users currently in the server database is shown.
Main flow:	1. User requests for users to be shown by selecting User Administration in the menu.
Alternate flows:	None.

Use case ID:	UC3
Use case name:	Update user
Description:	Gives the admin. the ability to change data about any user.
Primary actors:	Administrator.
Secondary actors:	Database.
Preconditions:	The administrator must be logged in and one or more users exists in the database
Postconditions:	A field of data has been changed in the database
Main flow:	1. The admin. selects the “update user” option 2. The admin. selects the data that they wish to change 3. The admin. writes in what new data should be assigned to the given field
Alternate flows:	1. The input given by the user does not match the datatype of the selected data 2. An exception is thrown telling the user that their input is invalid 3. The user is asked to give a new input

Use case ID:	UC4
Use case name:	Set inactive
Description:	Gives an admin the ability to set a user in an inactive state. The user cannot be deleted but won’t have any access while the user is inactive.
Primary actors:	Administrator

Secondary actors:	Database
Preconditions:	The administrator must be logged in and two or more users must exist in the database
Postconditions:	The user is set as inactive and won't be able to access the system
Main flow:	1. Admin initiates update process 2. Admin changes the status field to inactive 3. Admin completes update process
Alternate flows:	None

Commodity-Administration

Use case ID:	UC5
Use case name:	Add new commodity
Description:	The pharmacist creates a new commodity which in turn gets stored in the database
Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	The pharmacist must be logged in
Postcondition:	The commodity gets stored in the database
Main flow:	1. Pharmacist initiates the create commodity process 2. Pharmacists fills out the required fields 3. Pharmacist completes the create commodity process
Alternative flows:	3. Pharmacist inputs invalid or wrong syntax in inputs 4. An error is thrown and the appropriate error message is shown.

Use case ID:	UC6
Use case name:	Update commodity
Description:	The pharmacist chooses to edit a commodity

Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	The pharmacist must be logged in and one or more commodities exists in the database
Postcondition:	The commodity has been edited in the database
Main flow:	1. Pharmacist initiates the edit process 2. Pharmacist changes the data he/she wishes to edit 3. Pharmacist completes the edit process
Alternative flows:	1. Pharmacist inputs invalid or wrong syntax in inputs 2. An error is thrown and the appropriate error messages is shown

Use case ID:	UC7
Use case name:	Show commodities
Description:	The pharmacists chooses to view all commodities in the database
Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	The pharmacist must be logged in.
Postcondition:	All commodities are shown to the pharmacist
Main flow:	1. The pharmacist requests for all commodities to be shown
Alternative flows:	None

Prescription-Administration

Use case ID:	UC8
Use case name:	Create prescription
Description:	The pharmacist creates a new prescription
Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	The pharmacist must be logged in

Postcondition:	The prescription has been stored in the database
Main flow:	1. Pharmacist initiates the create prescription process 2. Pharmacist fills out the required fields 3. Pharmacist completes the create prescription process
Alternative flows:	1. Pharmacist inputs invalid or wrong syntax in the input 2. Error is thrown and an appropriate error message is shown

Use case ID:	UC9
Use case name:	Show prescriptions
Description:	The pharmacists is able to view all prescriptions in the database
Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	The pharmacist must be logged in.
Postcondition:	All commodities are shown to the pharmacist
Main flow:	1. The pharmacist requests for all prescriptions to be shown.
Alternative flows:	None

Commodity-Batch-Administration

Use case ID:	UC10
Use case name:	Create Commodity batch
Description:	Foreman chooses to create a commodity batch
Primary actors:	Foreman
Secondary actors:	Database
Preconditions:	Foreman must be logged in
Postcondition:	The commodity batch has been stored in the database
Main flow:	1. Foreman initiates the create commodity batch process 2. Foreman fills out the required fields 3. Foreman completes the process
Alternative flows:	1. Foreman inputs invalid or wrong syntax in the input field

	2. An error is thrown and an appropriate message is shown

Use case ID:	UC11
Use case name:	View commodity batch
Description:	The foreman requests to view all commodity batches in the database
Primary actors:	Foreman
Secondary actors:	Database
Preconditions:	Foreman must be logged in
Postcondition:	All commodities are shown to the Foreman
Main flow:	1. Foreman requests to view all commodity batches
Alternative flows:	None

Product-Batch-Administration

Use case ID:	UC12
Use case name:	Create new product batch
Description:	The foreman creates a new product batch
Primary actors:	Foreman
Secondary actors:	Database
Preconditions:	Foreman must be logged in
Postcondition:	1. The product batch gets stored in the database 2. The product batch is printed
Main flow:	1. Foreman initiates the create new product batch process 2. Foreman fills out the necessary fields 3. Foreman completes the product batch process
Alternative flows:	1. Foreman inputs invalid or wrong syntax in the input field 2. And error is thrown and an appropriate messages is shown

Use case ID:	UC13
Use case name:	View product batches
Description:	The Foreman has the ability to see all existing product batches in the server database.
Primary actors:	Foreman.
Secondary actors:	Database.
Preconditions:	The Foreman must be logged in..
Postconditions:	All product batches currently in the server database is shown.
Main flow:	2. Foreman requests for product batches to be shown.
Alternate flows:	None.

Weighing

Use case ID:	UC14
Use case name:	Weighing procedure
Description:	The procedure for weighing each product batch component for a given product batch and submitting it to the database.
Primary actors:	Laborant
Secondary actors:	None
Preconditions:	A WCU is connected with an ethernet cable to a computer running our program. The computer has an internet connection. The laborant knows the id for the product batch he is about to weigh
Postcondition:	The status of the product batch has changed to 2 (finished) Each product batch component for the batch has been created with valid information
Main flow:	1. The user initiates the RM208 command 2. The WCU asks for a operator ID 3. The user enters his or hers operator ID and the database shows the corresponding name in the display 4. The WCU asks for a product batch ID 5. The WCU confirms that the ID is valid and instructs the user

	to place an empty container on the WCU and tara 6. The user places container and presses tara 7. The WCU confirms and instructs the user to bring the next commodity for the product batch, and enter the ID of the used commodity batch 8. The user enters the ID and is instructed to weigh the commodity 9. The user weighs the commodity 10. The WCU informs the user that he is about to commit an product batch component to the database. The user confirms 11. Points 5-10 are repeated for each product batch component
Alternative flows:	None

3.2 System Architecture

3.2.1 Database

3.2.1.1 Entity/Relation Diagram

The E/R diagram shows the entity relations in the database. Each entity (table) symbolised by a rectangle consist of some attributes shown as ellipses and underscoring the primary keys. The relations between the entities is represented by diamonds with the foreign keys. The cardinality is shown with crow's-foot notation fx. óne operator (user) has one or more product batches - this is a one to many relation. Following E/R diagram is a draft of the database.

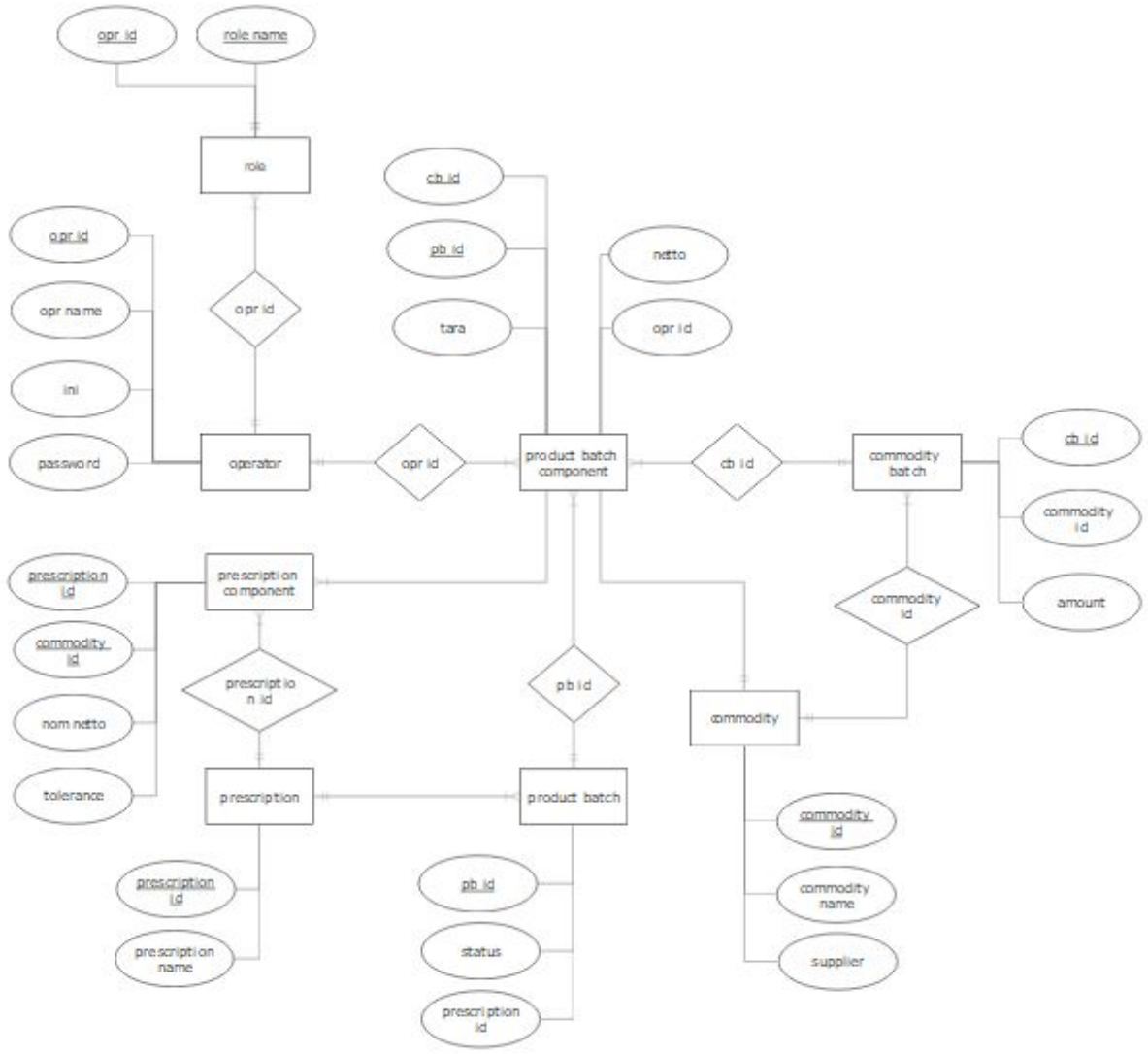


Figure above: E/R diagram.

3.2.1.2 Database design decisions

Most of our database is following the relations outlined in the project description, but we have made a few changes.

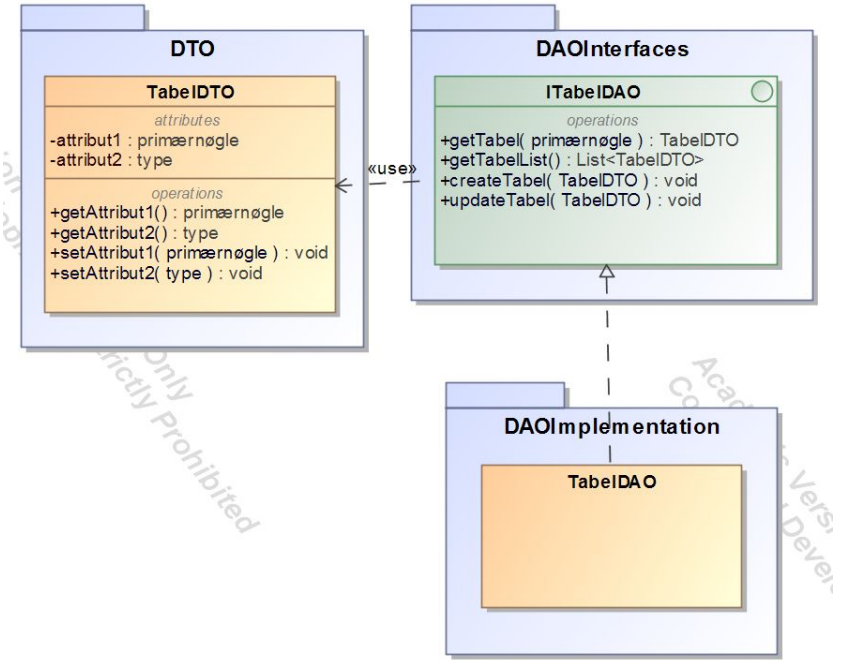
The first change is that we have named the user table operatoer instead of laborant, and we have removed the CPR value from the relation. The CPR value could potentially be used to identify a single user like an id, and this would break the principles of the 3rd normal form that there should be no way to identify a single row other than the primary key.

The second change is that we have created a role table for the operatoer, that uses both operatoer id and a role string as its primary key. The roles of an operatoer can be found by accessing this table using only the opr_id field, which will return a list of all the roles that are associated with the operatoer. This way we can keep the roles of the operatoer in the database without breaking the normal form principles.

3.2.2 Web Application

3.2.2.1 System Package Diagram

This generalised diagram shows the principles of the packages and classes of the web application. Because the web applications classes is quite similar.



We have setup our web application system with packages, each having their own responsibility, ensuring high cohesion. The packages are as following.

Datalayer
The data layer implements the data layer interface, but is where the actual magic happens. Every class within this package are the only java classes within the web application that connects to the database with the logic to receive and insert data through their own public methods. It accepts data transfer objects in the methods to avoid multiple request being made within a short time.

Datalayer interface
A data layer interface sets rules on the data access for different applications, by providing operations without exposing details of the database. Within the interfaces, we use objects called Data Access Objects (DAO), that only has the specific operations that we need.
For each table in the database, we have an interface with the few operations that are needed to use. Most common operations in this project would be to create and update an object as well as reaching for the entire list that the object has.

Data Transfer Objects

The main purpose with the DTO is to transfer data throughout the system while achieving the smallest performance cost. For that reason, it's best that we transfer the data as an object at every request with as small amount of information as possible to avoid repeatings.

Therefore the simplest solution is to create a DTO for every table and view in the database for which we would like to work with, as well as variables to save every attribute.

The objects has get- and set-methods to every variable and the variables themselves are private.

Rest
Rest (Representational state transfer) uses a stateless protocol to handle operations for every client as fast as possible, ensuring high performance. It's also limited to little data being sent forth and back between the server and the client compared to other web services.

The system uses the rest service to communicate with the client through http requests, while also communicating with the data access object (DAO) and data transfer object (DTO) using java language. Every services also has little responsibility, for instance, userservice handles every operation that takes place within the user administration page.

SQL Connector
Since the database is the core of all data within this entire project, we need an SQL connector within the web application. This package has three classes; Connector, Constant and a SQLMapper that handles everything regarding the connection and communication between the web application and the database.

3.2.2.2 System Class Diagram: User Administration

The following SCD diagram only shows the packages and classes connected to the *User Administration* function for simplicity. The fully detailed SCD is shown in appendix 8.1.1.

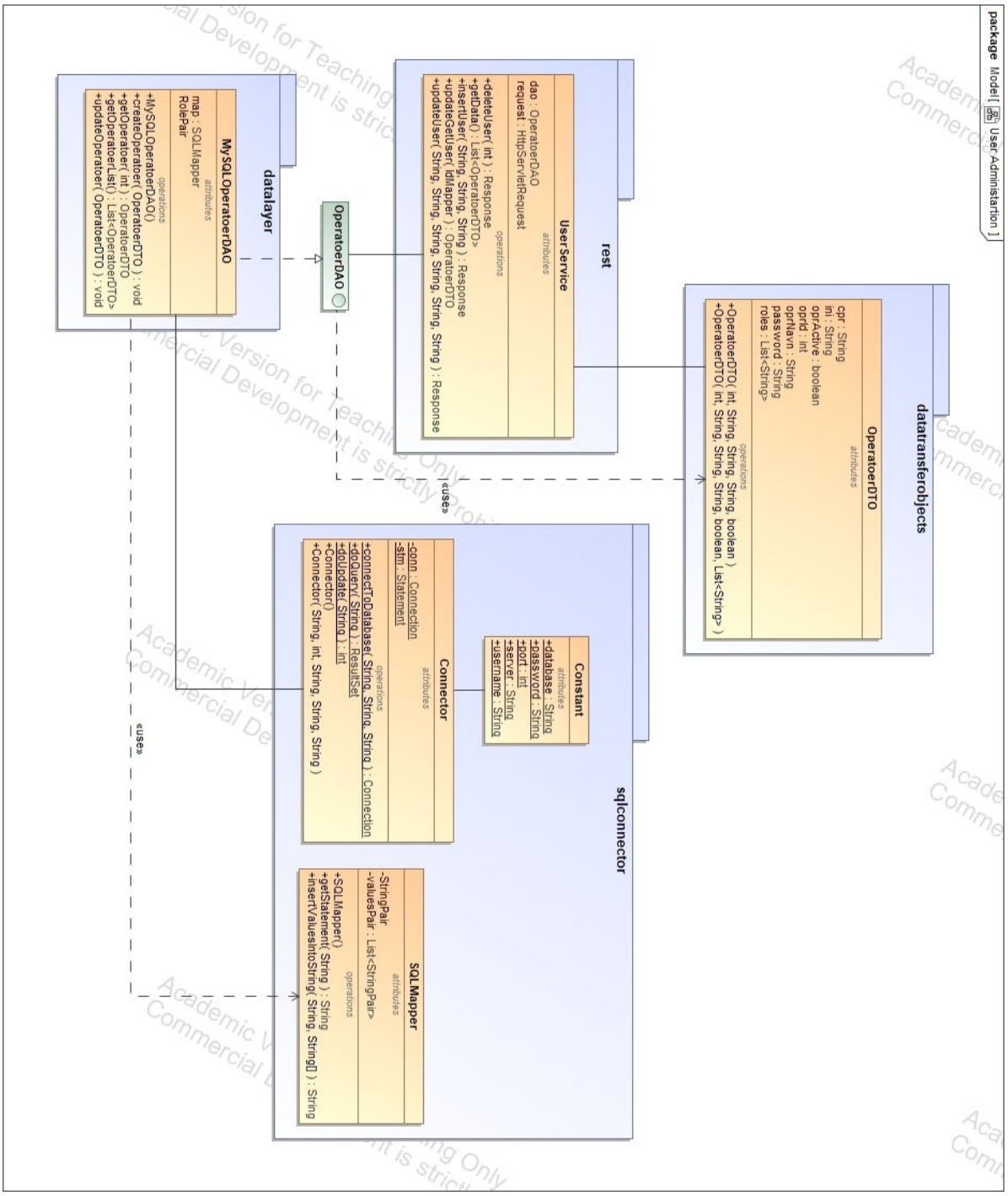
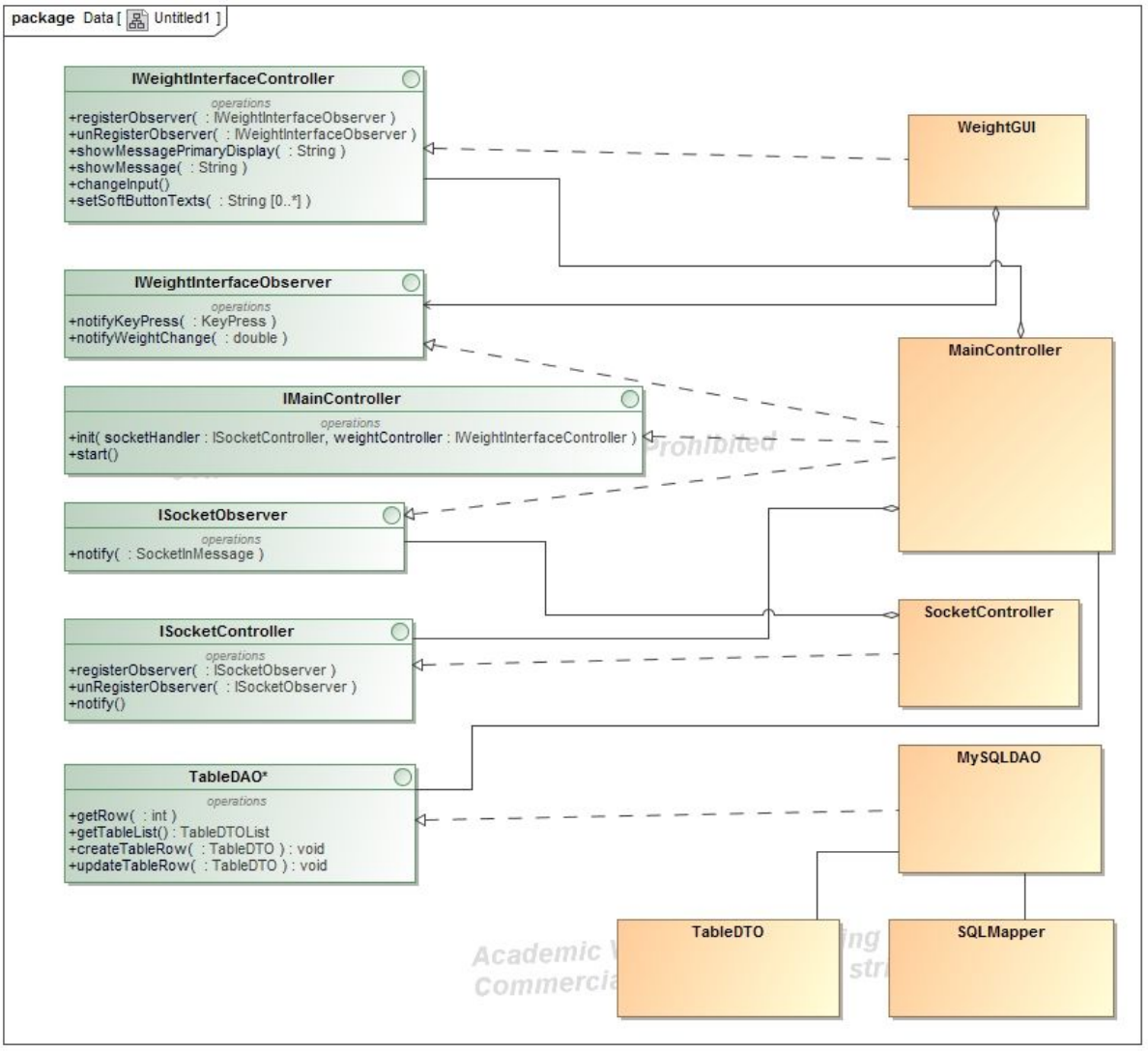


Figure above: This SCD diagram only shows the User Administration relevant classes.

3.2.3 Weight Simulator

3.2.3.1 System Class Diagram



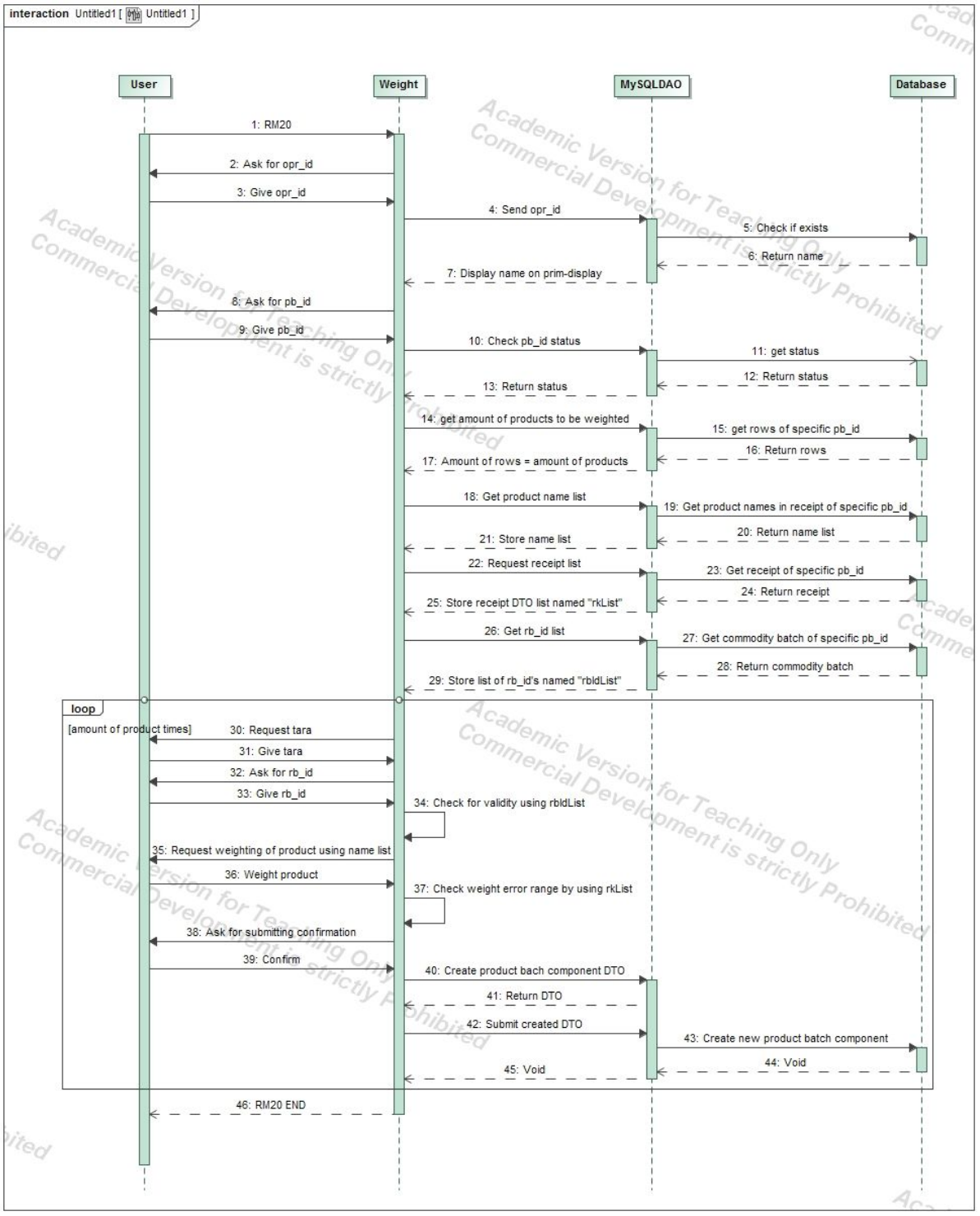
WeightGUI, MainController, SocketController and MySQLDAO all implements an interface. The WeightGUI is the class which controlls the graphic user interface of the weight. The SocketController is the class which handles our wireless connection between the weight and a device trying to connect and MainController is the Controller class which uses both the SocketController and WeightGUI and manages the functionalities of the weight.

MySQLDAO and its interface TableDAO has been very generalised since there is a lot of classes and corresponding interfaces which pretty much functions the same but for different SQL tables in our database.
The proper name for our TableDAO and MySQLDAO for each of our classes can be found by replacing “Table” in the interface with the desired SQL Table and in our class by putting in the desired table name between the “MySQL” and “DAO” part. Depending on the Table

associated with the interface then the interface might also have an extra function which returns a list of DTO’s specified by a given id submitted in the methods parameters

MySQLDAO classes have the abilities to get a row from a table in our database by some id input, get a full list of the table by using DTO’s (classes which stores multiple values of data), create a table row to insert and to update a table in our database with fx. a tablerow you’ve made. The SQLMapper class is a class utilized by our MySQLDAO and manages the connection to our database.

3.2.3.2 System sequence diagram



The sequence diagram, there is missing a few loops/if statements in the top end since its checking alot for validity.

3.2.3.3 Threads

We run 2 threads in our weight system in which the first thread manages our WeightGUI inputs and the other thread manages our inputs through our socket connection. The two threads gives us flexibility and makes it possible for us to both run a command like RM20 which guides you through a linear process and providing that process with inputs from the WeightGUI.

3.2.3.4 RM20

RM20 is the bread and butter command for our weighting process, RM20 is the command which starts the weighting process for a complete productbatchcomponent which we require multiple of to complete a productbatch.

We have decided to assume beforehand that creating a productbatch is a process with no breaks inbetween meaning its a process which is being done in one session from start to finish, this gives us some leeway in how we've decided to make the RM20 procedure.

The RM20 process is as followed: You connect to the Weight and run the RM20 command, the Weight will request for you to input your user id and if it exists in our database it will confirm you by showing your name on its primary screen, the weight will now ask you which product batch you want to start on - you will not be able to choose a product batch with the status of 1 or 2 in the database as these product batches are being worked on by someone else and since we assume a product batch is done in one session from start to finish then you cant continue the work of a product batch with status 1. When you have chosen a product batch which is valid the weighting procedure will start for real, there is a receipt connected to each product batch and said receipt will list some stuff which needs to be weighted, the RM20 procedure will not finish until every object on the receipt has been weighted. The procedure will loop you through first weighting and tara a container, input a commodity batch id and having you weight the object it requests on its secondary display, the weight will not accept how much you weight unless its within a 10% fail margin of an ideal weight of the object stored in the database.

We have stored your operator id, the valid product batch number (as well as changed its status from 0 to 1 in the database), the tara, the commodity batch id and the weighting results and by the end of each loop we then be able to create a product batch component in the database and when the user has been looped through every object to be weighted the RM20 process will end and you will have to repeat the command if you want to start it anew.

In our RM20 procedure we have not take height of storage management as there haven't really been given that many specifications if any regarding the matter in our given tasks. If we did choose to implement it we would have also run into problems like what is a unit of a product as it is listed in the database in the commodity batch's "maengde" row, is it meassured in weight or units? we wouldnt know.

4 Implementation

4.1 Database & Data Access Layer

4.1.1 Basic Functionality

We started with signing up for a service that provides free online web hosting and allows us to keep a MySQL database. The only problem with this is that we only have one user to work with, which means that we have a maximum of 10 simultaneous connections. If another database with the option for adding more users was added, it could be an option to make a data access layer that allows the users to log into the MySQL database with different credentials instead of using the same single MySQL user for each new connection.

In other words, we have a database active that follows our database design, but only 10 connections can be active at a time because we only have a single user for this database to edit it with. This is due to the nature of the web service. It is free, but very limited.

We have created a new interface and data access object for each table in our database, except roles, which we consider to be a part of the data access object that handles users. We also have a view data access object for the situations where we need to get something from a view.

We have data transfer objects for everything except roles, and multiple different data transfer objects for the view dao, because we retrieve data from several different tables in this.

4.1.2 Getting and executing the SQL queries

We use a class we call SQLMapper to store all the queries we need. We can then get a statement from it by referring to its name with the method “getStatement”. In practice, the code for getting a statement looks like this:

```
SQLMapper map = new SQLMapper();

String statement = map.getStatement("opr_SELECT");
```

These two lines of code are a bit farther apart in the actual implementation as we use the same SQLMapper for all methods in a dao implementation, but the basic principle remains. The above code would get the string:

```
"SELECT * FROM operatoer WHERE opr_id = ?;”
```

We use the method “insertValuesIntoString” to insert our own values instead of the question marks. These are inserted in order, so it is important to understand which values should go

where in the string. When there is only 1 value, it is simple, but when the number rises to 6 or 7, which it does in some of our update statements, then it becomes more complicated.

When the statement is made, we use a Connector class, which establishes a connection to our database, and can execute queries and updates.

```
ResultSet rs = Connector.doQuery(statement);
```

The result is a resultset, from which we can get the values in the database. We can use while(rs.next()) or if(!rs.first()) to either run through all the values in the resultset or to check if we are on the first element in the result set (if there even is one), which allows us to run our code for only one element in the resultset, which is the first element.

```
if (!rs.first()) throw new DALException("Operatoeren " + oprId + " findes ikke");
temp = new OperatoerDTO (rs.getInt("opr_id"), rs.getString("opr_navn"),
rs.getString("ini"), rs.getString("password"), rs.getBoolean("opr_active"));
```

The above code would throw an exception if there are no elements in the resultset. Otherwise it will set the value of our OperatoerDTO temp to be a new OperatoerDTO based on the values in the database.

This way we can return single OperatoerDTO's or a list of these to work with further up in the system.

4.2 Web Application

4.2.1 REST

The web application uses REST with Jersey (Jax-Rs). REST (REpresentational State Transfer) mainly uses the methods GET, POST, PUT and other methods in the HTTP protocol to send data between the client and the server.

The system was created as a dynamic web maven project in order to allow for this, and we added dependencies for the Jax-Rs libraries. This way they will automatically be downloaded when the project is updated.

After the project setup, we needed to first specify a path for our application. REST works in a way that is very similar to clicking a link and getting a web page. When targeting a specific URL, instead of getting a html document, the user instead arrives at a service in form of a piece of java code. But in order to get there, we need to specify where the services are. The @ApplicationPath annotation allows us to specify part of the path to our services. In this case, we chose the name "/rest" for the path, so every service we have will have a path beginning with "/rest".

```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/rest")

public class AppConfig extends Application{

}
```

The above class is also necessary. The Application extension allows the jax-rs library to work.

The above code allows us to start creating services for the REST backend, each with their own path.

```
21  @Path("/login")
22  @Consumes(MediaType.APPLICATION_JSON)
23  @Produces(MediaType.APPLICATION_JSON)
24  public class LoginVerification {
25          @Context
26          HttpServletRequest request;
27
28          OperatoerDAO dao = new MySQLOperatoerDAO();
```

This service is the beginning of our login verification service. This service has the path “/login”, meaning one could use POST or GET on “/rest/login” and potentially run a piece of java code if no other paths are specified for the individual methods.

```
30      @POST
31      @Path("/verify")
32      @Consumes("application/x-www-form-urlencoded")
33      public Response verifyUser(
34                  @FormParam("username") String userName,
35                  @FormParam("password") String password
36                  ) throws DALException, URISyntaxException{
37
38
39              List<OperatoerDTO> users = dao.getOperatoerList();
40
41              for(OperatoerDTO opr:users){
42                      if(opr.getOprNavn().equals(userName) && opr.getPassword().equals(password)){
43                              request.getSession().setAttribute("user", opr); //Session attribute
44                              return Response.temporaryRedirect(new java.net.URI("../menu.html")).build();
45                      }
46              }
47          return Response.temporaryRedirect(new java.net.URI("../")).build();
48
49      }
```

This method is run when POST is used on “/rest/login/verify”. We use it as soon as a user tries to log in. Through our datalayer, we get a list of all the users in our system. This methods then checks through each one. If it finds a user where the username and password matches the values that were typed into the login form (which this method takes as parameters using the @FormParam annotation) then it will get the session of the current HttpRequest and set a session attribute in form of this OperatoerDTO.

In other words, the server checks if the login credentials are correct and sets a cookie with the user’s information if they are.

The user does not have access to this cookie, so it is just as secure as other alternatives like JWT Token security, but it is a bit more resource consuming, as each user gets their own cookie on the server instead of having an encrypted token.

4.2.2 AJAX

On our webpages, we use AJAX (Asynchronous Java and XML) to fetch data with help from our REST backend. We use JQuery to make these AJAX calls simpler.

```
9   <script>
10  $( document ).ready(function() {
11          $.ajax({
12                  url : "rest/userservice/getusers",
13                  data : $('#form').serializeJSON(),
14                  contentType: "application/json",
15                  method: 'POST',
16                  success : function(response){
17                          console.log(response);
```

When the document is loaded, the above code is run. It targets a specific service with the method ‘POST’ in this case. If it is successful (i.e. it gets a response), then we have a javaScript function that is run with the response as the parameter. Otherwise we get an error. The response has any data that the service sends along.

We use this to display our users. The userservice/getusers service produces a JSON string and sends this along to our AJAX code. This JSON contains information on all the users that the user who is logged in is allowed to see. We can then append the data to any element in our HTML document.

```
16                  success : function(response){
17                          console.log(response);
18                          console.log(response.userId);
19                          for(var i = 0; i < response.length; i++){
20                                  $('#usertable').append("<tr>");
21                                  $('#usertable').append("<td><button class=\"small\" id=\""
22                                                  + response[i].oprId
23                                                  + "\" type=\"submit\" form=\"updateUser\" value=\""
24                                                  + response[i].oprId
25                                                  + "\" name=\"submit\">EDIT</button></td>");
26                                  $('#usertable').append("<td>" + response[i].oprId + "</td>");
27                                  $('#usertable').append("<td>" + response[i].oprNavn + "</td>");
28                                  $('#usertable').append("<td>" + response[i].ini + "</td>");
29                                  $('#usertable').append("<td>" + response[i].roles + "</td>");
30                                  $('#usertable').append("<td>" + response[i].oprActive + "</td>");
```

In this case we append strings of HTML code to the element with the id ‘usertable’ and add some data from the response while doing so. This results in something that looks like this:

Add user

	ID	User Name	Initials	Roles	Active	
EDIT	0	Duran Duran	DD	laborant	true	X
EDIT	1	Angelo A	AA	admin,foreman	true	X
EDIT	2	Antonella Barbarossa	AB	foreman	true	X
EDIT	3	Luigi C	LC	laborant,pharmaceut	false	X
EDIT	4	admin	a	admin,foreman,laborant,pharmaceut	true	X

The buttons have an id that matches the user they belong to, meaning we can use them to send the id as data to other services. This is what we use when updating a user or setting a user inactive.

4.2.3 Web content - HTML & CSS

The web application pages itself is written using HTML and styled in CSS. While we’re not creating a ‘single-page’ website, we make authorizations check on a user’s session, when entering a new page. The application is using scripts to create and run the desired functions fx. in the User Administration.

HTML
The Web content consists of several .html text files / pages each fulfilling a function fx. view, add or update etc.

The user enters the index.html, the login page when running the web application.
When logged in the user is redirected to the menu.html page which consists of a menu and an iframe. The user never actually leaves the menu.html page while logged in.

When selecting a page in the top menu the user won’t be redirected to that .html page instead it is the content of the iframe which changes to the desired page. *HTML iframe*

represents a nested browsing context, a way to embedding another HTML page into the current page, showing the content of that page in the iframe. Selecting another page in the menu will redirect the page in the iframe.

CSS

To create the layout for our pages, we have two .css files.
One to make every page consistent in the design and layout, and another to create a print friendly layout when printing the product batches.

mainStyle.css is the main stylesheet in our project. We use percents when designing the boxes and tables, while we use em to define the size of margins, paddings and fonts. The reason for em instead of pixels are that, if a user should choose to resize the font size on his/her computer, the boxes scales accordingly.

print.css is the stylesheet for printing the product batches. While mainStyle.css is scaling on the users choice of increasing/decreasing font size, the print page remains fixed to fit a paper.
It’s layout is also created with boxes within boxes to create the correct margins as if it was a real page. On a side note - what you see on the html. file, is what would be printed on a machine.

4.3 Weight Simulator

Features

As of yet everything required for the weighing process is implemented and working. The RM20 command has been specified in an earlier section of our report and works as intended with some minor errors which will be covered in the next section. Our connection with the database is working as intended and our program can update and view it when required but without the user directly doing it to prevent unnecessary database problem. Our WCU inputs are also working as intended.

Problems

We have a problem regarding our weighing process. When the WCU assigns what should be weighted in a commodity batch then the SQL Query we use can and will list every object which exists in the receipt corresponding to the commodity batch id submitted. By listing everything we run into the problem that if a product has more than one commodity batches it will be listed multiple times.

pb_id	recept_id	rb_id	raavare_id	raavare_navn
5	3	1	1	dej
5	3	4	5	ost
5	3	5	5	ost
5	3	6	6	skinke
5	3	7	7	champignon

As seen above the product “ost” has multiple instances since it has multiple commodity batch id’s, the receipt is also missing the product “tomat” which has a product id of 4 but thats because the product hasn’t been assigned as a commodity batch and is not really a mistake on the weight end but rather a mistake on the database end. This problem has most likely occured as we are using NATURAL JOINs for every join:

```
SELECT pb_id, recept_id, rb_id, raavare_id, raavare_navn
FROM produktbatch
        NATURAL JOIN receptkomponent
        NATURAL JOIN raavarebatch
        NATURAL JOIN raavare
WHERE pb_id = 5;
```

And can most likely be fixed by changing the JOIN regarding our “raavarebatch” table.

Our second problem comes from the assumption that a product batch component is being made throughout one session from start to finish. This assumption makes us not care about continuing a weighting procedure which has been stopped midway (status 1) since we assume it is being worked on until it is finished (status 2). When a user starts a weighting procedure the status for the chosen product batch will change from 0 to 1 but if by chance the procedure is stopped midway there will be no way to continue the procedure, and it will remain unaccessible from that point on (since you can’t start a procedure with a product batch having a status of ½), unless someone manually resets its status in the database and that sums up the problem. Closing in the middle of a session can be done by hitting any close button including the exit button on the weight.

5 Test

5.1 Unit Tests

5.1.1 Database & Data Access Layer

5.1.1.1 SQL Queries

We have tested our database by running all the SQL statements we wanted to use on it before implementing them in our java program (specifically the class SQLMapper) in order to make sure they were producing the correct result.

We did not insert a new SQL Query into the SQLMapper before having run it at least once.

5.1.1.2 Data Access Layer

We have created unit tests for SQLMapper and various data access object implementations in the class **TestDAO**. Many of our data access objects have already been tested in previous projects, so we mainly focused on the dao's that were modified in this project. This is mainly the MySQLOperatoerDAO, which we have added roles to, and the new MySQLViewDAO, which we use to fetch data from our views. The unit tests mainly test if we get a datalayer exception, and fail if this is the case. This includes SQLExceptions, meaning if something is wrong with a query or we violate integrity constraints, the test will fail. The exception to this is the insert command, which will almost always give an error in the test because it tries to insert an operatoer with a primary key which already exists in the table.

Our conclusions from these unit tests were that the data access layer connects to the database properly and can insert and fetch data in the correct way.

5.1.2 Web Application

5.1.2.1 Website User Interface

The website user interface is practically impossible to do a unit test with without deconstructing the entire program, which would take too long. As such we have not created methods for this in a unit test, but we have however gone through most of the button presses we could make in a given administrative view.

There are edge cases remaining that we have not tested, but most buttons and methods seems to work.

5.1.3 Weight Simulator

The weight simulator is the easiest way to test our system while implementing, without needing to have access to a physical WCU. It is therefore essential that the necessary functions in the weight simulator works correctly. The weight simulator is not capable of every function available in the real WCU, but everything needed for this particular project works correctly. The weight simulator has a collection of buttons; “Exit”, “Zero”, “Tara”, “[->”(send), “Shift”, “C” and the number pad. Each of these apart from “C” works as intended. “C” was supposed to clear the last entered character, but has not been implemented. “C” would be convenient to have, but is not needed because the user can always press “Zero” to clear everything and start over.

5.2 Use-Case Tests

User-Administration

Test case ID	TC1
Summary	Tests to see if creating a user, actually creates a user with the correct data.
Requirements	R1.1.3.a.i.1
Preconditions	The administrator must be logged in.
Test Procedure	1. Admin. initiates create user process 2. Admin. inputs required information into appropriate inputs and ends user creation process 3. Admin. completes the process
Test data	1. Username a. Testcase1 2. Password a. 1234 3. Initials a. TC1 4. Role a. admin
Expected result	a. User added to database
Actual result	a. A new user was added
Status	Passed
Tested by	F & D

Date	15/06/2017

Test case ID	TC2
Summary	Tests to see if the admin. has the ability to see all existing users in the server database.
Requirements	R1.1.3.a.i.2
Preconditions	The administrator must be logged in.
Test Procedure	1. User requests for users to be shown by selecting User Administration in the menu.
Test data	None
Expected result	All users in the database is shown
Actual result	All of the actual results are as the expected results
Status	Passed
Tested by	Frederik
Date	15/06/2017

Test case ID	TC3
Summary	Tests to see if editing a user will actually edit a user
Requirements	R1.1.3.a.i.3
Preconditions	Must be logged in as an admin
Test Procedure	Clicks on edit user ‘Danny’
Test data	Replaces ‘Danny’ with ‘DANNY’
Expected result	The user’s name should be DANNY
Actual result	The user’s name is now DANNY
Status	Passed
Tested by	Danny
Date	15/06/2017

Test case ID	TC4

Summary	Testing the edit status function to see if a person's status changes
Requirements	R1.1.3.a.i.4
Preconditions	Must be logged in as an admin
Test Procedure	1. Admin clicks on edit to the left of 'DANNY's name a. Admin clicks the drop down menu and puts the status to inactive. 2. Admin clicks on edit to the left of 'DANNY's name a. Admin clicks the drop down menu and puts the status back to active 3. Admin clicks on the X to the right of 'DANNY's name
Test data	1. No data needed 2. No data needed 3. No data needed
Expected result	1. The user gets redirected to the user view table and 'DANNY's status is set as inactive 2. The user gets redirected to the user view table, and 'DANNY's status is set as active. 3. The page is refreshed and the status is now inactive again.
Actual result	The status changed correspondingly to the expected result
Status	Passed
Tested by	Danny
Date	15/06/2017

Note: The user with the name "admin" cannot have their status changed or any other field for that matter. This is a way to ensure that the user "admin" always has the privileges they need to access all parts of the system.

Test case ID	TC:A1
Summary	Tests to see if the created user can log in.
Requirements	R1.3.2
Preconditions	The user must not yet be logged in
Test Procedure	1. User enters the required data and presses log in
Test data	1. Username: Testcase1

	2. Password: 1234
Expected result	The user should be redirected to the main page (menu.html)
Actual result	The user is now redirected to the main page.
Status	Passed
Tested by	Danny
Date	/06/2017

Commodity-Administration

Test Case ID:	TC5
Test Case name:	Add new commodity
Description:	The pharmacist creates a new commodity which in turn gets stored in the database
Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	The pharmacist must be logged in
Test Data	Commodity name = “ananas” Commodity distributer = “Hawaii.co”
Test Procedure	1. Pharmacist initiates the create commodity process, press add 2. Pharmacists fills out the required fields 3. Pharmacist completes the create commodity process, press submit
Expected Result	The commodity gets stored in the database
Actual Result	The commodity is stored correctly and is shown as soon as the user is redirected to the commodity view.
Status	Passed
Tested By	Tobias Højsgaard

Date DD/MM/YY	15/06/2017

Test Case ID:	TC6
Test Case name:	Update commodity
Description:	The pharmacist updates an existing commodity. The new values get stored in the database
Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	The pharmacist must be logged in At least one commodity must exist already
Test Data	Commodity name = “ananas” CHANGE TO “ANANAS” Commodity distributer = “Hawaii.co”
Test Procedure	1. Pharmacist initiates the edit process, by pressing the EDIT button 2. Pharmacist changes the data he/she wishes to edit 3. Pharmacist completes the edit process, by pressing the SUBMIT button
Expected Result	The commodity gets stored in the database
Actual Result	The update is completed successfully, but at one point there was an error. It was a nullpointer exception. This is though the commodity existed at the time. The error did not persist though, so the test passes in the end.
Status	Passed
Tested By	Tobias Højsgaard
Date DD/MM/YY	15/06/2017

Test case ID	TC7
Summary	Test if the Pharmasist can show commodities
Requirements	R1.1.3.b.i.2
Preconditions	Pharmacist must be logged in
Test Procedure	1. The pharmacist requests for all commodities to be shown by

	selecting Commodity Administration in the menu
Test data	None
Expected result	All commodities are shown to the pharmacist
Actual result	All commodities are shown to the pharmacist
Status	Passed
Tested by	Frederik
Date	15/06/2017

Prescription-Administration

Test Case ID:	TC8
Test Case name:	Create prescription
Description:	Tests the add new prescription function
Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	Pharmacist is logged in
Test Data	“Burger
Test Procedure	Pharmacist clicks on add prescription and types in the test data
Expected Result	Burger should be part of the prescription list
Actual Result	Burger is on the list
Status	Passed
Tested By	Danny
Date DD/MM/YY	15/06/2017

Test Case ID:	TC9
Test Case name:	Show prescriptions
Description:	Tests to see if pharmacist can see the current prescriptions in the database

Primary actors:	Pharmacist
Secondary actors:	Database
Preconditions:	Pharmacist is logged in
Test Data	None
Test Procedure	Pharmacists enters Prescription administration
Expected Result	Pharmacist gets a list of all the prescriptions
Actual Result	The list is shown to the pharmacist
Status	Passed
Tested By	Danny
Date DD/MM/YY	15/06/2017

Commodity-Batch-Administration

Test case ID	TC10
Summary	Test creating a commodity batch
Requirements	
Preconditions	Foreman must be logged in
Test Procedure	1. Foreman initiates the create commodity batch process 2. Foreman fills out the required fields 3. Foreman completes the process
Test data	1. Commodity ID a. 0 b. 10 c. 0 2. Amount a. 1000000 b. 1 c. 1
Expected result	a. The commodity batch has been stored in the database b. The commodity batch has been stored in the database c. The commodity batch has been stored in the database
Actual result	a. The commodity batch has been stored in the database

	b. The commodity batch has been stored in the database c. The commodity batch has been stored in the database
Status	Passed
Tested by	Frederik
Date	15/06/2017

Product-Batch-Administration

Test case ID	TC11
Summary	Test if the foreman can view view all commodity batches in the database.
Requirements	R1.1.3.c.ii.2
Preconditions	Foreman must be logged in
Test Procedure	2. Foreman requests to view all commodity batches by selecting Product batch Administration in the menu
Test data	None
Expected result	All commodities batches are shown to the Foreman
Actual result	All commodities batches are shown to the Foreman
Status	Passed
Tested by	Frederik
Date	15/06/2017

Test case ID	TC12
Summary	Test creating new product batch
Requirements	R1.1.3.c.ii.1
Preconditions	Foreman must be logged in
Test Procedure	1. Foreman initiates the create new product batch process 2. Foreman fills out the necessary fields 3. Foreman completes the product batch process

Test data	1. Prescription ID a. 4 2. Status a. 0
Expected result	1. The product batch gets stored in the database 2. The product batch is printed
Actual result	1. The product batch gets stored in the database 2. The product batch is not printed and the print function does not work as intended
Status	Passed (1/2)
Tested by	Frederik
Date	15/06/2017

Note: The print function was not implemented as a response to creating a new product batch but as a button - we failed to implement a fully functional printing function.

Weighing

Test case ID	TC13
Summary	Test to see if the weighing procedure works as intended and is sufficiently intuitive
Requirements	R1.4.1, R1.4.2, R1.5.1, R1.5.2, R2.1, R2.2, R2.3
Preconditions	The WCU is connected via an ethernet cable to a computer with internet. The database server is running A user ID is available for use A product batch with status 0 (not started) is available
Test Procedure	A complete weighing procedure runthrough is made
Test data	User ID: 1 Product batch ID: 5
Expected result	The status for the product batch ID 5 will have changed to 1 (in progress) when the ID is entered on the WCU The status for the product batch ID 5 will have changed to 2 (finished) when the last product batch component has been entered and accepted Five product batch components with product batch ID 5 have been created in the database. Each with acceptable values.
Actual result	The status for the product batch ID 5 updated correctly both times.

	Five product batch components with product batch ID 5 have been created in the database. All except one have correct values. There is an error in the structure of the database that courses one commodity ‘tomat’ to be replaced with ‘ost’. This is a very serious error, but not one caused by the weighing procedure
Status	Passed
Tested by	Theis
Date	15/06/2017

Result Matrix

Testcase Matrix

Testcase / Usecase	**Requirements**	**Status**
TC1 / UC1	R1.1.3.a.i.1	Passed
TC2 / UC2	R1.1.3.a.i.2	Passed
TC3 / UC3	R1.1.3.a.i.3	Passed
TC4 / UC4	R1.1.3.a.i.4	Passed
TC5 / UC5	R1.1.3.b.i.1	Passed
TC6 / UC6	R1.1.3.b.i.3	Passed
TC7 / UC7	R1.1.3.b.i.2	Passed
TC8 / UC8	R1.1.3.b.ii.1	Passed
TC9 / UC9	R1.1.3.b.ii.2	Passed
TC10 / UC10		Passed
TC11 / UC11	R1.1.3.c.ii.2	Passed
TC12 / UC12	R1.1.3.c.ii.1	Partially passed
TC13 / UC13	R1.4, R1.5, R2	Passed
TC:A1	R1.3.2	Passed

5.3 Integration Test

The integration test focused mainly on whether switching between different views/use cases was a smooth experience. It was mainly a test of the menu.html page that has an iframe that shows the different views we have created, with buttons that change the target of this iframe.

When pressing the buttons, the view switches. Sometimes, when pressing a button before a view has loaded, the page will alert with the 500 error, but this is simply because a process was interrupted in the java code in the REST backend and the new view that the user switched to will usually load right afterwards, just like switching normally.

Every view functions normally, even while inside the iframe, so if a given view passed all of its use case tests, it will still work in the same way inside the menu page. In some ways, the system is better at dealing with errors in the views that do not work well, because the buttons in the menu page provide an easy way to reset the views.

The logout button also works as intended and can always return a user to the start screen and sets their session data to null in the process.

5.4 Assessment

5.1 What worked well

We have implemented all the views given in the project description. A few of these have issues, but the majority work reliably.

The user administration menu is the most robust of these and accounts for most things a user could do that could otherwise pose a problem to the system. For instance, originally a single administrator could in theory remove the roles of all other users or set every user inactive, but we have implemented a super user “admin”, which cannot be changed, even by itself, meaning there is always a way to restore the user’s again, though it would take a lot of time. The users without the admin role can also only edit themselves, and cannot change their roles.

We also managed to implement both the weight control unit and the weight simulator, so overall the parts of the system given in the project description are all there, though we had to leave out some functionalities we would have liked to implement.

5.2 What did not work well

While we managed work most of the features we wanted into our system, a few features still didn’t get implemented. A few of them were partially implemented and are still in the system, but do not work properly.

One of the features we did not succeed in creating is a prescription that also has prescription components. We ran into problems with inserting values into the database with regards to the components and did not have enough time to fix the issue.

Next is the printing of the product batches. A template “view_print.html” file has been made, with a layout matching the one given in the project description, but we had issues with inserting data into this template. Other parts of the project had to be prioritized, and therefore we did not end up implementing this print functionality.

6 Conclusion

Overall the system works with most features successfully implemented both the WCU and the web application. We have implemented all the views with a few additions - a few of these have issues, but the majority works as intended. The following requirements were either not implemented or only partially implemented.

Input validation in most of the forms was not implemented. This makes the site open to cross site scripting. We also did not work with the actual weight, but by using PUtty will make the controller work with the weight. The weight works as intended except for two very minor issues.

The work distribution and process worked as intended. Some group members did invest more time and work on the project than other but in general everyone did their part.

7 Literature

7.1 Books

John Lewis, William Loftus: *Java software solutions, Seventh Edition*
Published 2012
Pearson Education, Inc., publishing as Addison-Wesley, Boston, Massachusetts
ISBN 13: 978-0-13-214918-1

Craig Larman: *Applying UML and Patterns: An introductions to object-oriented analysis and design and iterative development*
Published: 30. October 2004
Pearson Education, Inc. publishing as John Wait
ISBN: 0-13-148906-2

7.2 Web pages

MySQL v5.5 reference document; By Oracle Corporation and/or its affiliates; © 2017
https://dev.mysql.com/doc/refman/5.5/en/

REST with Java (JAX-RS) using Jersey - Tutorial; By Lars Vogel; © 2009
http://www.vogella.com/tutorials/REST/article.html

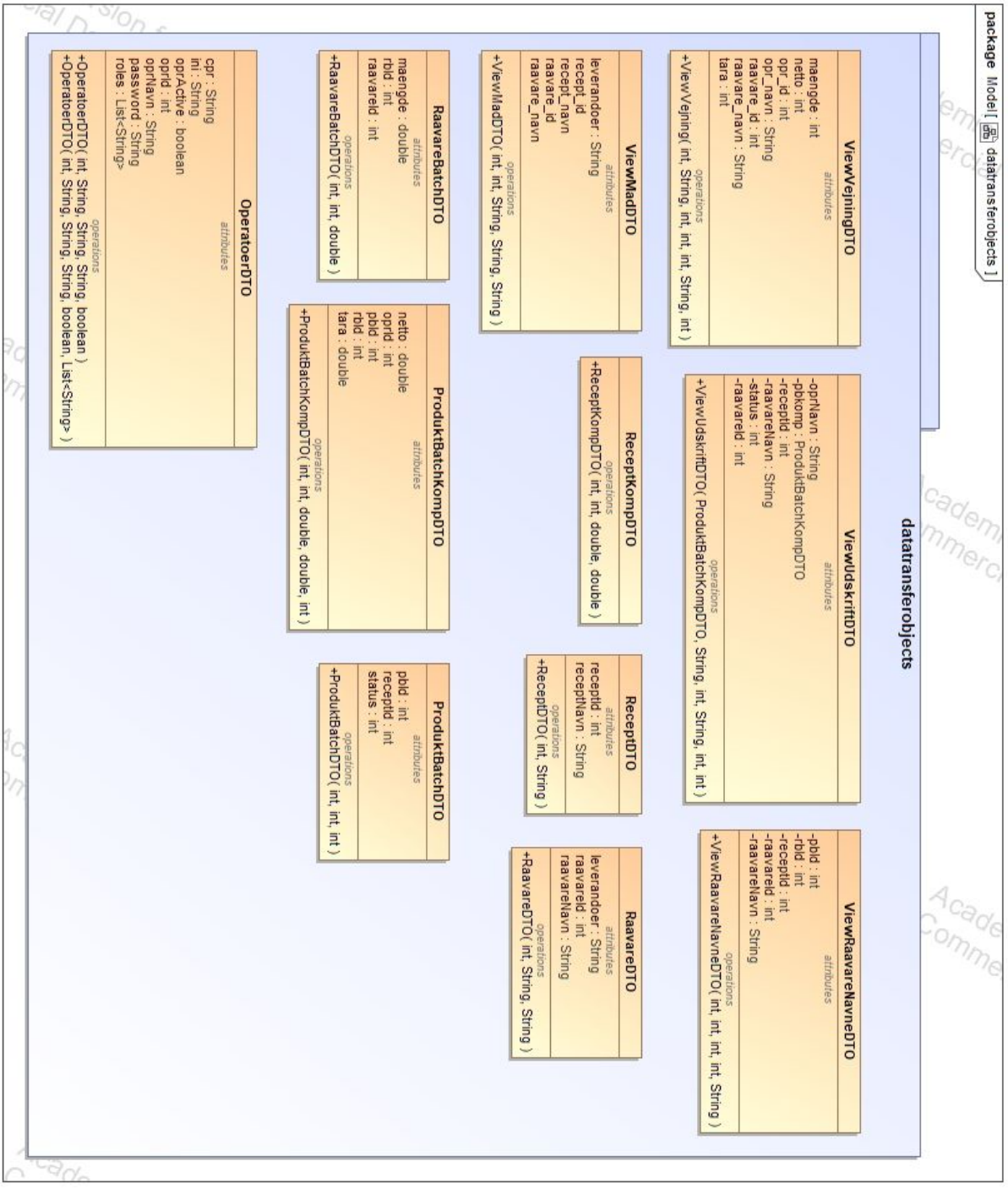
W3Schools - HTML Tutorial; By Refsnes Data; © 1999-2017
https://www.w3schools.com/html/default.asp

8 Appendix

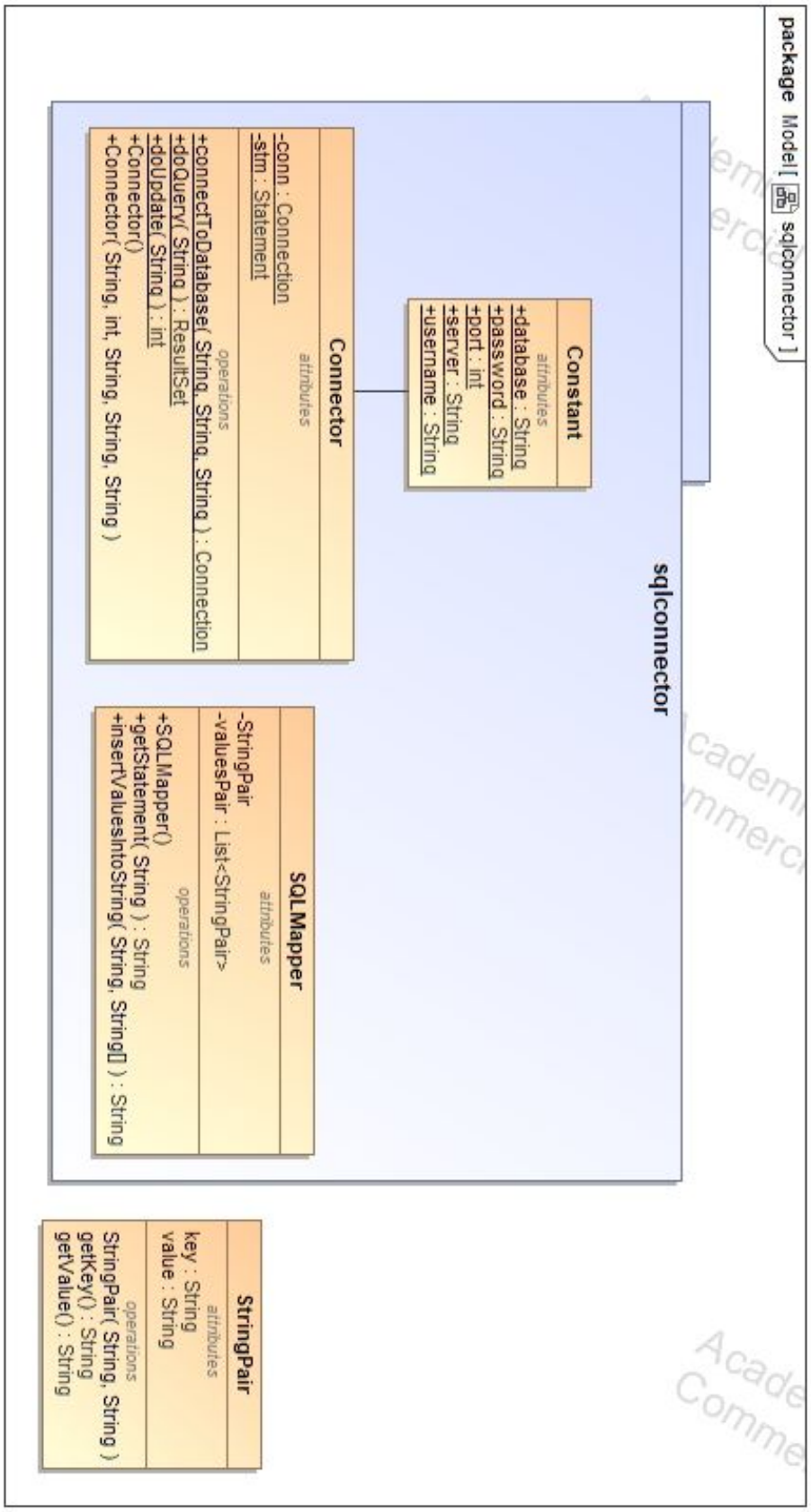
8.1 Additional Diagrams

8.1.1 System Class Diagram

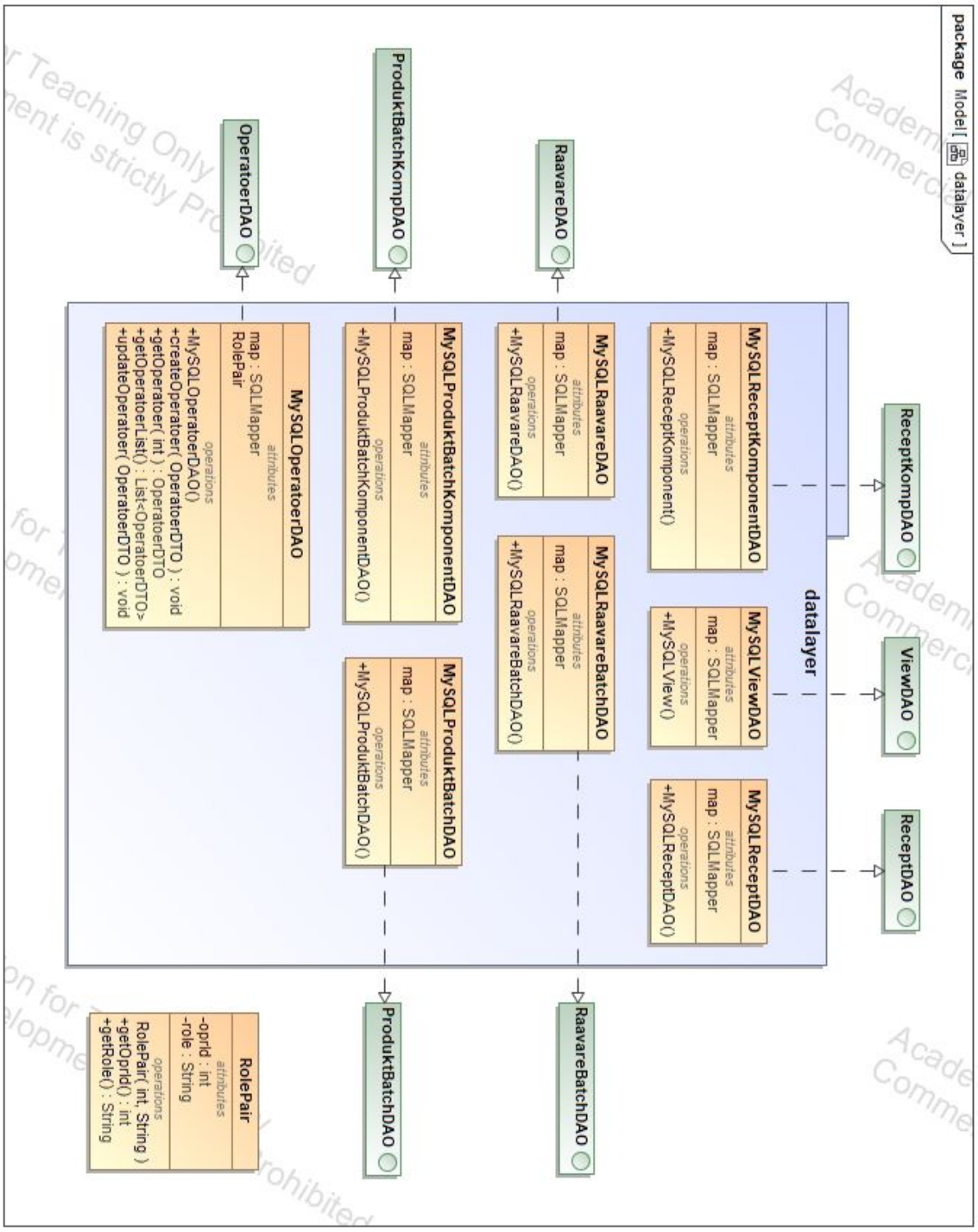
8.1.1.1 SCD datatransfereobjects



8.1.1.2 SCD sqlconnector



8.1.1.3 SCD datalayer



8.1.1.4 SCD datainterfaces