CDIO 4 - Final report

Final exam project with Java, Javascript and WebInterface.

By group 22

Peter El Habr
s165202

Simon Engquist
s143233

Arvid Langsø
s144265

Mikkel Lund
s165238

Jeppe Nielsen
s093905

Mads Stege
s165243

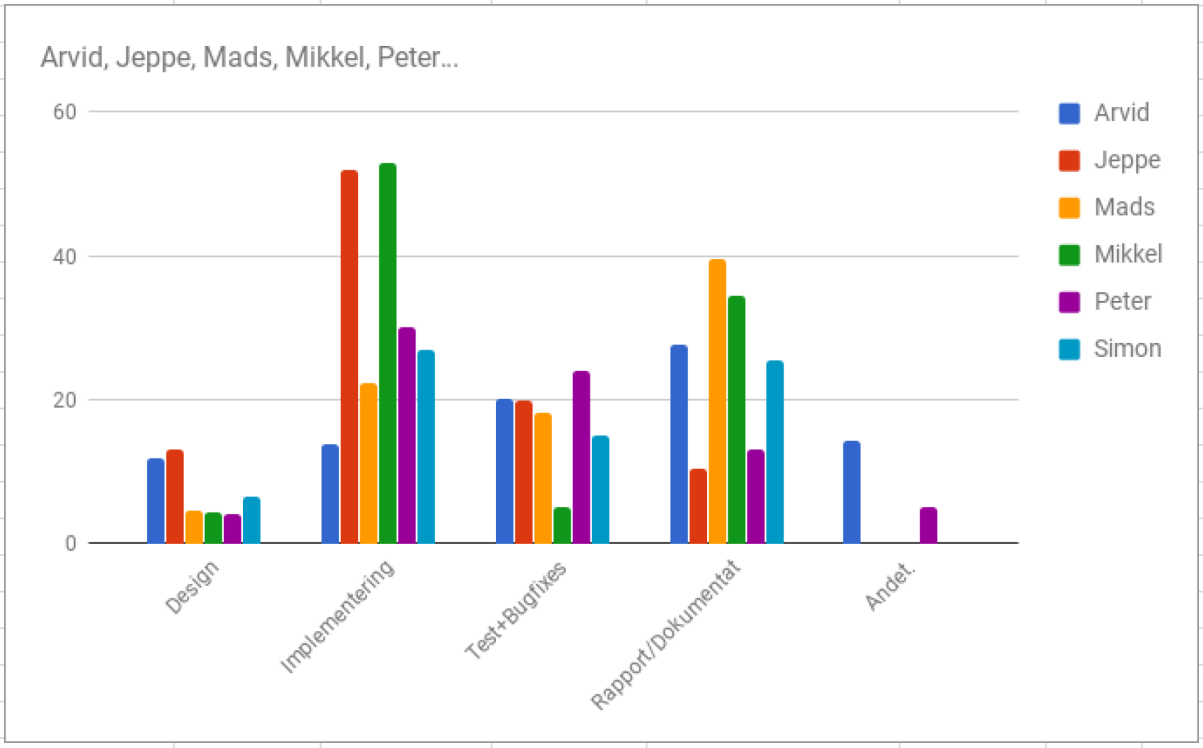
Technical University of Denmark
DTU Compute

16. of June, 2017 - 12:00 AM

Number of pages (including appendix): 72

Courses: 02324

	Design	Implementering	Test+Bugfixes	Rapport/Dokumentation	Andet.	Total
Arvid	11,91633333	13,74666667	20,08333333	27,66666667	14,33333333	87,74633
Jeppe	13	52	20	10,5	0	95,5
Mads	4,5	22,25	18,25	39,49	0	84,49
Mikkel	4,25	53	5	34,5	0	96,75
Peter	4	30	24	13	5	76
Simon	6,5	27	15	25,5	0	74
Total:	44,16633333	197,9966667	102,3333333	150,6566667	19,33333333	514,4863



A full time table for every person can be found in the last appendix. This has detailed information on every developer.

1 User Guide

The purpose of this guide is to help the IT-administration of a pharmaceutical company to set up, after purchase of this software solution. It contains a list of required software to run the system, a guide on how to setup and shutdown the system, as well as a easy-to-read overview of the website functionality.

1.1 Prerequisites

The following software needs to be pre-installed in order for the system to work.

- The lastest version of Java (the current version as of June 2017 is Java 8, build 131).
- The lastest version of Google Chrome 64-bit (the current version as of June 2017 is Google Chrome 64-bit v. 59.0.3071.86).
- The latest version of the IDE Eclipse with an Apache Tomcat server installed (the current version as of June 2017 is Eclipse Neon 3 with Apache Tomcat 8.5.11.)
- The lastest version of Windows (the current as of June 2017 is Windows 10).

When all of the prerequisites have been installed, the system is now ready for the start up process.

1.2 Folder setup

To allow the system to save its data correctly, it is necessary for the user to verify that the folder ”C:/Users/< USERNAME> /Documents” exists on the installed system. It is vital for the system to function properly, to create this location if it does not exist already.

1.3 System setup - step by step guide

1. Fetch the latest version of the software from GitHub: `github.com/ArvidLangsoe/22_CDIO4` or use the supplied .zip file.
2. Import the project into Eclipse.
3. Before continuing you need to create a Tomcat server in Eclipse.
4. Right-click on project. The default name is ”22_CDIO4”. Then select ”Run as”, and finish by clicking ”Run on server”.
5. Click ”Finish”. This is will start the program using the server you chose (Apache Tomcat 8.5.11).
6. Upon startup, the program will create its own WeightTable.txt. The file is located in your Documents folder, as listed in section 1.2. The file will contain a list of weights that the

program should connect to. The initial file contains a dummy-weight as a placeholder. Use the same setup when adding your own weights.

7. Having inserted your own weights into the WeightTable, you now need to close the program by clicking the red square in Eclipse.
8. Run the server again as you did in steps 4-5.
9. Your program and website is now running on your system. Go to your browser, and enter `localhost:8080/22_CDIO4` into the search bar. If the website does not open up a log-in page, try shift+F5 to ignore your locally cached content (if any).
10. You are now able to log onto the system with a certified user.
11. Upon a successful log-in, the user will now be able to manipulate items on the website, according to his or her role.

If for whatever reason the system does not start up properly, please try re-importing the project, and verify the installation of the prerequisite software.

1.4 System shutdown - step by step guide

If you want to shut down the system, for example because of maintenance, it is important to note the following: Even though the system is designed to receive and save valid information packages continuously, in case of an abrupt shutdown, non-saved information *will* be lost.

To avoid problems and data loss, announce maintenance breaks beforehand to avoid that any employees are using the system when you shut down the system.

Now you can safely shut down the system by killing the process running on the host system.

1.5 Web-client overview

In figure 3 below, an overview of the Web-clients front page is shown when a user has access to everything. How many of the menus to left are shown, depends on the the role assigned to the user upon creation.

Having finished the log-in procedure, the user will get access to the main page of the website, allowing for a broad and easy to overlook view of the possible actions.

Predefined data

You can find all the predefined data in section A Predefined data.

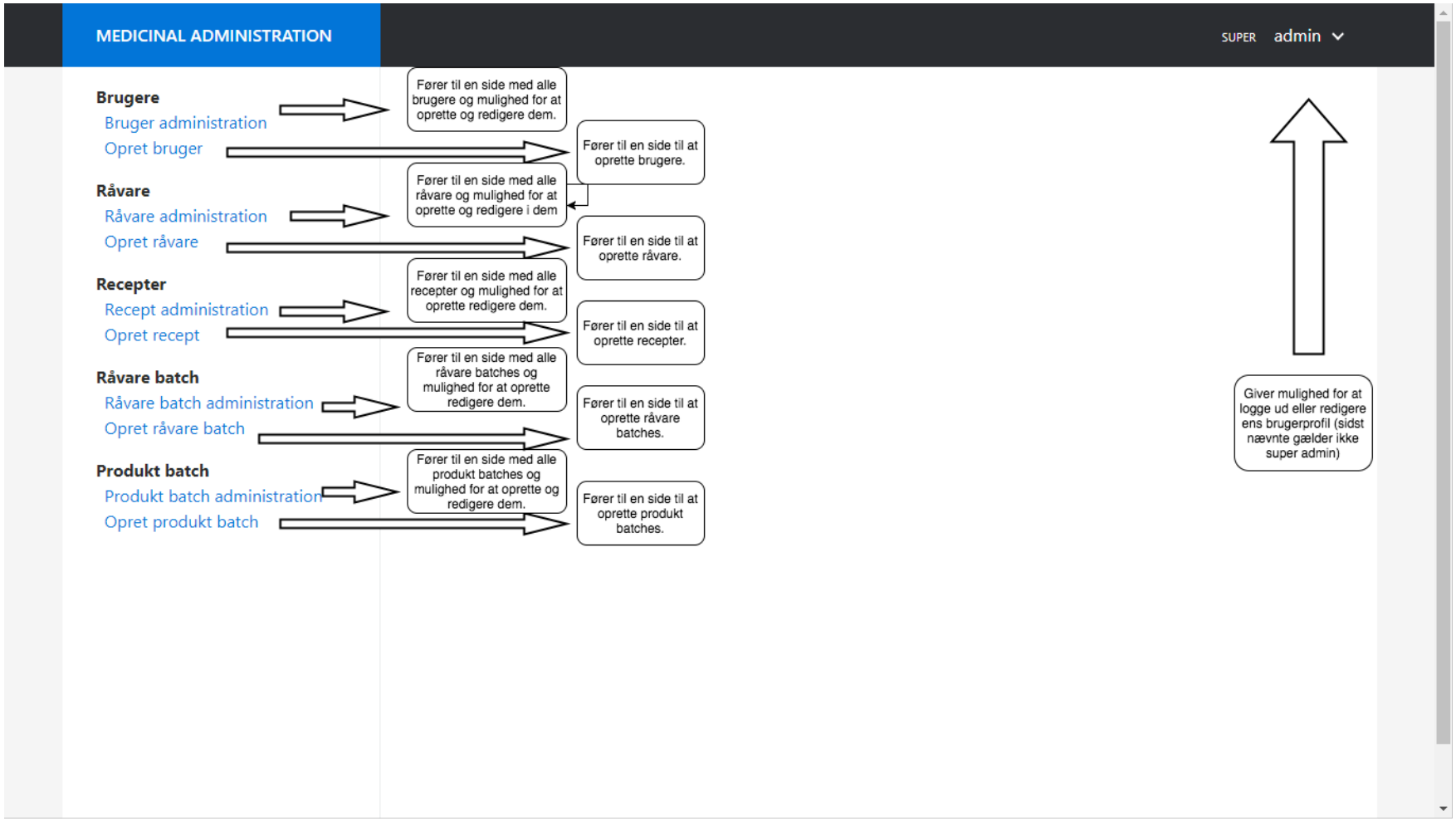


Figure 3: Explanation of functions and possibilities on the web page

2 Abstract

In this project a software solution to a pharmaceutical company has been made. The solution consists of the following three parts:

Firstly, an Industrial weight control unit (ASE) that can connect to the weights in the company. Furthermore it keeps track of the weighing procedure when making products in the company. In addition to that it saves the measurements in the data layer on the Web-server. Secondly, a website that is managing all of the company's data. This data includes information about the users of the system and information about the products that the company makes, how to create them and what they consists of. All the manageable data comes from the Web-server that the website is hosted on. Lastly, a data structure to save all of the data on the Web-server in. The data is accessed through the data access layer.

The website consists of a front-end which consists of HTML, CSS and JavaScripts. There has been used certain frameworks, libraries and APIs such Bootstrap, jQuery and Ajax. The front end is connected to the back end using Ajax to call the Rest layer in the Java part of the program. From there the data propagates down through the layers until it is saved in the data.

The data structure used to save the data in this project is files. When the data is underway from the front-end to the back-end it is checked several times, both in the Javascript and in the Java code.

In the industrial weight control unit (ASE) there has been used threads and sockets to establish that you can run and connect to multiple weight concurrently.

In conclusion the developing team has made a software solution ready to be tested in the pharmaceutical company. In future iterations the most important things to be addressed is the need of a database, security and user experience.

Content

3 Introduction

The objective of this assignment is to create a software system that aids a pharmaceutical company in their daily routines. These routines includes administration of users, raw materials, raw material batches, recipes and product batches and in addition to that, a weighing of product batch components procedure. It is expected that these routines do not change much from the old, previous routines. This allows for an easy transition. Furthermore, all users of the system needs to have an account which contain information about the user such as personal information, log-in information and accessibility information. The accessibility information includes the role of the user, which determines which parts of the system the user has access to.

The segment above mostly describes one part of the software system, namely the Web-client. In truth, the software solution is a distributed system which consists of a Web-server with a Web-client and an ASE-unit. The ASE-unit has the responsibility of controlling the different industrial weights in the pharmaceutical company and the measurement procedure of an product component. At last it saves the measured data on the Web-server. The last component, the Web-server, handles requests from the Web-clients. These request are then handled and relevant information is saved on the server.

The website (the web-client) must be accessible from most modern Web-browsers. Therefore the website is to be developed in HTML5 and JavaScript, which are tools used in modern Web-development.

To host the website, a web server is needed. The pharmaceutical company states that it should run on an Apache Tomcat-Server v8.5.11. The server runs on REST API, using the Maven framework. The server will be programmed to run on java version 8 or newer.

The software solution for the pharmaceutical company needs to be well-documented and tested to avoid misunderstandings and the most common errors in the system.

In the following section, a full of requirements to the software system requested by the pharmaceutical company can be found.

4 Software requirements

Together with the pharmaceutical company, the development team designed the requirements to the software system that matches the company's expectations.

Since the pharmaceutical company wants the software within short notice, the requirements will be implemented in a prioritised order. Below, one can find the list of requirements that was agreed upon. This list acts as a part of the contract between the development team and the pharmaceutical company.

The system is to consist of the following overall features:

ASE

ASE is the Industrial weight control unit. There is the following requirements to the ASE:

- Connects to predefined weight(s) from a list of available IP addresses.
- Controls the weighing procedure.
- Saves measurements made by the lab technician to the data structure chosen.

Data structure

A data structure which saves the pharmaceutical company's data. There is the following requirements to the data structure:

- **Safe storage** - The stored data needs to be safe from harm. It should not be deleted in case of a bug or corrupted files.
- **Availability** - The data needs to be available at all times. This is to ensure that the company's production is not halted because the data structure had to take a break. (Maintenance is of course an exception to this.)
- **Speed** - The data structure needs to be fast. Preferably so fast that the users wont notice any significant delay.
- **No invalid data** - The database should not contain any invalid data. This means that data that depends on other data can not be deleted. Other kinds of invalid data could be redundancy, which mean having duplicate data in the structure. This risks having different values for the same data. The system must protect itself from invalid inputs through thorough checks.

Web-client

The Web-client is the website where administration of all the company's data takes place. There is the following requirements to the Web-client:

- When the website is first opened, the users are asked to log in using their user id and password.
- Users with different roles can administrate or utilise different things. More on this in section: 4.1
- The website must be easy and intuitive to use, even for first-time users.
- The website must have a simple and user friendly design. Emphasis on ease of use, by using easily comprehensible wording and setup.
- The website must connect to the Web-server to retrieve the newest data available.

The points listed above is the requirements for the basic structure of the software system. Each of these components contain a multitude of sub-requirements, discussed in subsections below.

4.1 User roles & their purpose

The pharmaceutical company has given a list of roles there needs to be in the system. Furthermore, they have provided a list of requirements to each of the roles. The list of roles includes an *Admin*, a *Pharmacist*, a *Foreman* and a *Lab Technician*. For development purposes the development team has added an extra role, *Super Admin*. This role is used to test the system and will eventually be handed over to the client for future problem solving. The following use case diagram describes the different roles' use cases.

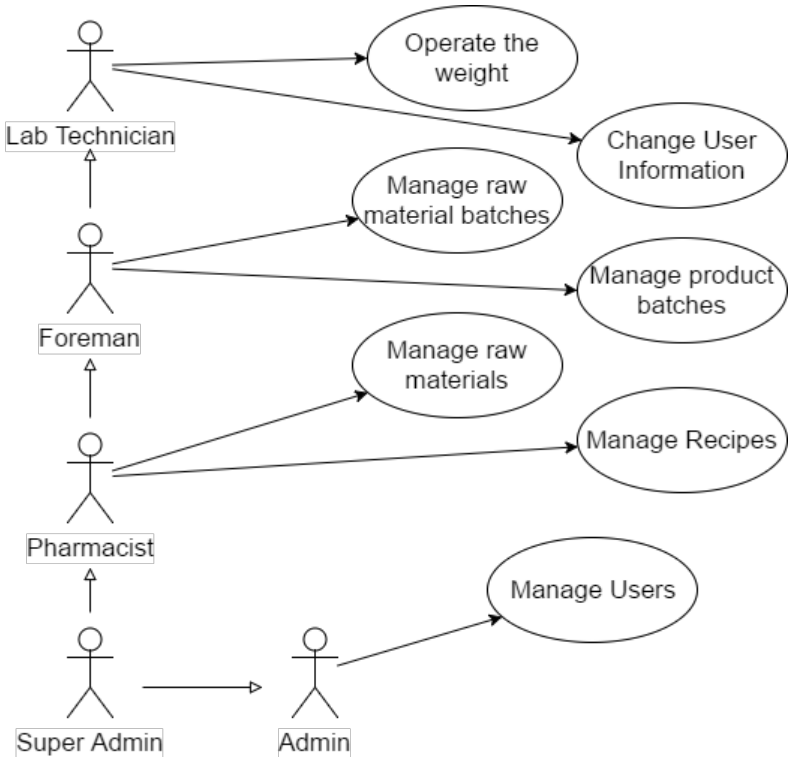


Figure 4: Use case diagram that shows what different users of the system can do.

As we can see on the use case diagram above, most of the roles inherit use cases from other roles. For instance the *Foreman* inherits use-cases from the *Lab Technician* which means that users with the *Foreman* as a role can both do the use cases of a *Foreman* and a *Lab Technician*. However a user with the role as a *Lab Technician* cannot perform the same tasks as a user with the role as a *Foreman*. The developing role *Super Admin* is special because it inherits the use cases of all the other roles in the system. This is usually something businesses want to avoid, since it gives too much power to a single person. For this reason the super admin should only be accessible by the developers.

The following list describes all the use cases of the different roles:

Lab Technician

The users with the *Lab Technician* role can operate the weights and record the measurements. They have the use cases

- Operate the weight - Use the industrial weight in the company to create product batches.
- Change User Information - User can change his own information, but can not change their role.

Foreman

The users with the *Foreman* role administrates raw material batches and product batches. Furthermore, they can do the same things as the users with the *Lab Technician* role. They have the use cases

- Manage Raw Material Batches - Create and update Raw material batches. Containing some amount of raw materials.
- Manage Product Batches - Create Product batches which are to be produced or has been produced.

Pharmacist

The users with the *Pharmacist* role administrates recipes and raw materials. Furthermore they can do the same things as the users with the *Foreman* role. They have the use cases

- Manage Recipes - Create and update recipes in the system and add ingredients to these recipes.
- Manage Raw materials - Create and update raw materials that are used as ingredients in the recipe.

Admin

The users with the *Admin* role administrates the users in the system. They have the use cases

- Manage Users - Create and update users in the system (including updating their own profile). This includes resetting passwords without seeing the password.

Super Admin

The *Super Admin* is a role created for the developers. It is not part of the data and cannot be assigned to an user. It does not contain any personal information. It has the use cases of all the other roles, which mean it has access to anything available on the website. The *Super Admin* is among other things used to test the software and help the client when they encounter problems.

4.2 Meta data - Company data description

The pharmaceutical company has also specified a list of requirements to the data that needs to be saved in system. The data attributes in the brackets () was not specifically mentioned in appendix 3 given by the pharmaceutical company. This appendix describes the data. But it has been indirectly suggested in other parts of the document the developers were given by the client. The list below is a description of each of the data attributes in each of the entities that needs to be saved. All id attributes are chosen by the user when the entity is created.

The user entity

All the roles which a user can be has the same amount of data attributes.

- **user id** - This is the id of the user. It is unique. It is an integer value in the range 1 to 99.999.999, defined by the client.
- **user name** - The name of the user. It is a string value with 2 to 20 characters.
- **Initials** - The initials of the user. It is a string value with 2 to 4 characters.
- **CPR-no** - The CPR-number of the user. Has to be a valid Danish CPR-number.
- **Role** - The role that a user can have. Has to one of the following: *Admin*, *Pharmacist*, *Foreman* or *Lab Technician*.
- **password** - The users password. It is a string value. The client wanted a password between 5-8 characters, but also wants it to follow the same rules as DTU's password rules. Since the 5-8 character limit is too insecure, it has been decided to only follow DTU rules.

The raw material entity

The raw materials that are used in the production of medicine.

- **raw material id** - This is the id of the raw material. It is unique. It is an integer value in range 1 to 99.999.999, defined by the client.

- **raw material name** - The name of the raw material. It is a string value between 2 and 20 characters.
- **supplier** - The name of the supplier. It is a string value between 2 and 20 characters.

The raw material Batch entity

The material batches that are delivered to the company.

- **raw material batch id** - This is the id of the raw materials batch. It is unique. It is an integer value in the range 1 to 99.999.999, defined by the client.
- **raw material id** - The id of the material that the batch contains. It is unique. This correspond to an existing raw material. It is an integer value in the range 1 to 99.999.999, defined by the client.
- **amount** - The amount of material delivered in the batch in kilogram (kg).

The recipe entity

The recipes used to create products.

- **recipe id** - This is the id of the recipe. It is unique. It is an integer value in the range 1 to 99.999.999, defined by the client.
- **recipe name** - Name of the recipe. It is a string value with 2 to 20 characters.
- **recipe component** - A recipe has n recipe components which contain the following data attributes
  - **raw material id** - The id of the material described in this. It is an unique integer value in range 1 to 99.999.999.
  - **nom netto** - This is the amount of the raw material in kilogram (kg) that should be added to the recipe. It is a Double value between 0,05 and 20.
  - **tolerance** - The maximum deviation from nom netto in percent. It is a double value between 0,1%-10%.

The product batch entity

The products that are in production or has finished production.

- **Product batch id** - This is the id Product batch. It is an unique integer in the range 1 to 99.999.999, defined by the client.
- **recipe id** - The id of the recipe being produced. It is an unique integer in the range 1 to 99.999.999, defined by the client.

- **status** - The status of the current production. There are 3 different options: Not begun = 0 / Under Production = 1 / finished = 2.
- **(start date)** - The date of the production start.
- **(end date)** - The date of the production start.
- **Product batch component** - A product has n product batch components, each of which contains the below data attributes. The product batch component correspond to a recipe component in the recipe being produced.
  - **user id** - The id of the user completing this part of the production. It is an unique integer in range 1 to 99.999.999, defined by the client.
  - **raw material batch id** - The batch being measured. It is an unique integer range 1 to 99.999.999, defined by the client.
  - **tara** - The weight in kilogram (kg) of the tara being used to measure the product.
  - **netto** - The weight in kilogram (kg) of the material being measured.

4.3 Industrial Weight Measurements

The ASE system, as previously mentioned, is the system that controls the industrial weights in the company. The system allows for multiple weights to connect to the server and save product batch components on the server. The company has a usual routine on how a measurement takes place and they want this to change as little as possible so that there can be an easy transition for the employees. Since there is a bit of discrepancy in the given documentation on how this takes place, a decision was made to accommodate the differing descriptions. The following is the order of how a measurement is completed:

1. The Lab technician receives a paper stating what they need to produce.
2. The Lab technician chooses an industrial weight, and enters their id.
3. The weight responds with the lab technicians name and the ask for confirmation.
4. The Lab technician confirms and enters the productionbatch id.
5. The weight respond with the recipe id and ask for confirmation.
6. The Lab technician now repeats the following pattern until the production is finished.
   (a) The Lab technician Enters the raw material batch id.
   (b) The weight respond with the raw material id and ask for confirmation.
   (c) The Lab technician clears the weigh, and confirms.
   (d) The Lab technician place the tara on weight, and confirms.
   (e) The Lab technician measures the product, and confirms.
   (f) The Lab technician is then asked to remove the product.

(g) The system performs a check to see if the measurement matches the recipe.

7. When all the measurements are done, the system set the product batch to be finished.

4.4 Unprioritised requirements

All of the above requirements, are requirements the development team wish to fulfil. However, it has been decided to not prioritise the following requirements due to the time constraints of the project:

Database

The development team has chosen not to deliver the initial project with a database. This is due to the requirements for the exam about online hosting, billing, availability and etc. Therefore, it was decided to focus on other aspects and materials utilised in the project. It was decided to have the program save data in files using serialising. More on this subject in section 7. The database has already been designed though. This design will be explained in section 14.5.

Weight Simulator

The development team has also chosen to not prioritise setting up the weight simulator. This is due to the man hours it would take to create a correctly working weight simulator that performs exactly like the industrial weight. Therefore, it was decided to focus on using the real weight, since the development team had easy access to this. The team also thinks that the weight simulator is only used for testing on the developer side, and is not needed for the company. Thus the team thinks it would be an unwise use of their time.

Printing

The development team has decided not to implement a printing option in the system. This is because it would take a lot of development time. Alternatives to a dedicated printing function already exists. For instance, one could just print the website, and have all of the needed information be available there.

Data attributes details

Due to some early confusion, the amount of each raw material batch is not updated after each product batch component has been made. In addition to, the dates and time stamps of when the product batches has been made, has not been added in the initial system.

Deactivation of users

The development team has decided to not focus on making deactivation of users possible. This is again due to the time it would take and because because of the relative ease a system administrator has restricting access to the system. This can be done easily by changing the users password. This could for example be needed if an employee had been fired, to prevent them from logging onto the website.

4.5 Work distribution

Due to the size of the project, the development team has been split up into three groups. One doing Web-development, the second taking care of the ASE (industrial weight control unit) and the last taking care of the rest, mainly the data layer.

- Web development: Jeppe and Mikkel.
- ASE: Simon, Mads and Arvid.
- Data layer: Peter.

The split up was necessary to ensure that the three parts of the software solution got the needed man hours.

5 Design of ASE - The Industrial weight control unit

This section describes how the Industrial weight control unit (ASE) is designed.

The ASE is in charge of connecting the weights in the company to the Web-server. Figure 5 below shows the class diagram for the ASE-system. It illustrates what the ASE-system consists of, which classes are related and what information they contain.

Description of the ASE-process

The whole process starts at the *WeightTable* file. It is a simple .txt file containing a list of all the information required to connect to the industrial weights in the company. The required information is the IP of the weight and its port number. The type of weights in this project have port number 8.000. If the system does not have the *WeightTable.txt* file, the program is designed to create the file itself. As a default, this file contains the IP 1.1.1.1 and port number 1.

How to add new weights to the system.

In order for a user to add to a new weight, the user simply needs to enter in the IPv4 and port number of the wished weight in the WeightTable.txt in the following manner:

```
1  IP xxx.xxx.xxx.xxx
2
3  PORT xxxxx
```

Repeat the above for the next weight.

The system itself follows a specific procedure. This is discussed in the next subsections.

Connecting to the weights

Whenever the ASE-system start, the *ConnectionReader* reads all available IPs and port numbers from the *WeightTable.txt* file, and verifies their integrity and validity.

Next up, the *ConnectionManager* reads the IPs and port numbers from the *ConnectionReader*, and attempts to establish a connection between the system and the weights.

This is done by trying to create a socket connection (TCP/IP) between the two. When the connection is establish a *WeightController* is created and run in a new thread. Afterwards the *ConnectionManager* gives the newly created socket to the *WeightController*. This allows the system to connect to and communicate with multiple weights concurrently.

Both the *ConnectionReader* and *ConnectionManager* should be designed to check if the IPs and port numbers in the *WeightTable.txt* file are read correctly.

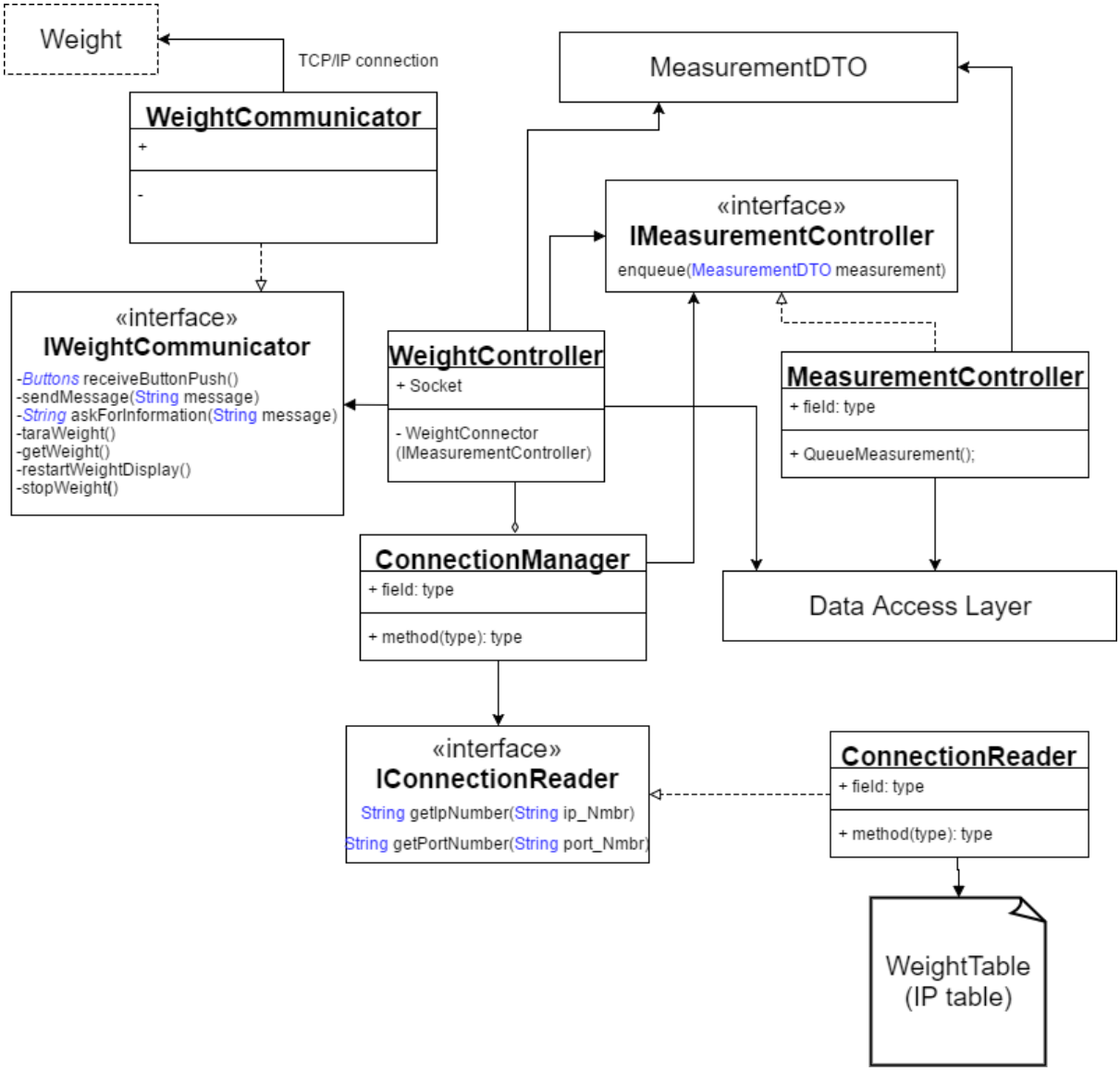


Figure 5: ASE - Class Diagram

Controlling the weights

The *WeightController* is responsible for controlling the weights connected to the system. It consist of two parts separated by an interface. The *WeightController* itself controls the logic. It decides what to show the user and when. It also make sure to handle any errors that might occur. The *WeightController* uses a *WeightCommunicator* as seen on figure 5. The socket connection from the *ConnectionManager* is passed on to this *WeightCommunicator* through the *WeightController*. The *WeightController* uses an interface, that allows for a potential replacement of the industrial weight.

The industrial weight has a protocol with commands on how to communicate with it. The *WeightCommunicator* knows some of the commands of the specific weight used by the company. The *WeightCommunicator* sends and receives commands to and from the weight. The

developing team makes use of seven protocols seen below:

- **RM20** - Sends a message and ask the user to fill in information.
- **P111** - Sends a message and display it on the screen.
- **Tara** - Tara the weight.
- **Measurement** - Measure the current weight.
- **DisplayClean** - Clear the display and input stream.
- **Quit** - Stop the weight.
- **StartUp** - Start up the weight in K 3 mode.

The *WeightCommunicator* also has some commands to handle button inputs from the user through the weight. The specific protocol commands used by the weight can be found in section B

The *WeightController* has a specific flow as seen on figure 6 below.

- First the user enters their id to log in.
- Then they register the product.
- When the product is registered, a loop begins that measure different raw materials.
- First a material is registered.
- Then the raw material is weighted.
- Then the measurement is checked in the product recipe.
- If the measurements are wrong, the user will be told so and prompted to redo the measurement.
- If the measurement was successful, they can continue with the next raw material. When all the materials are measured they are saved in the system

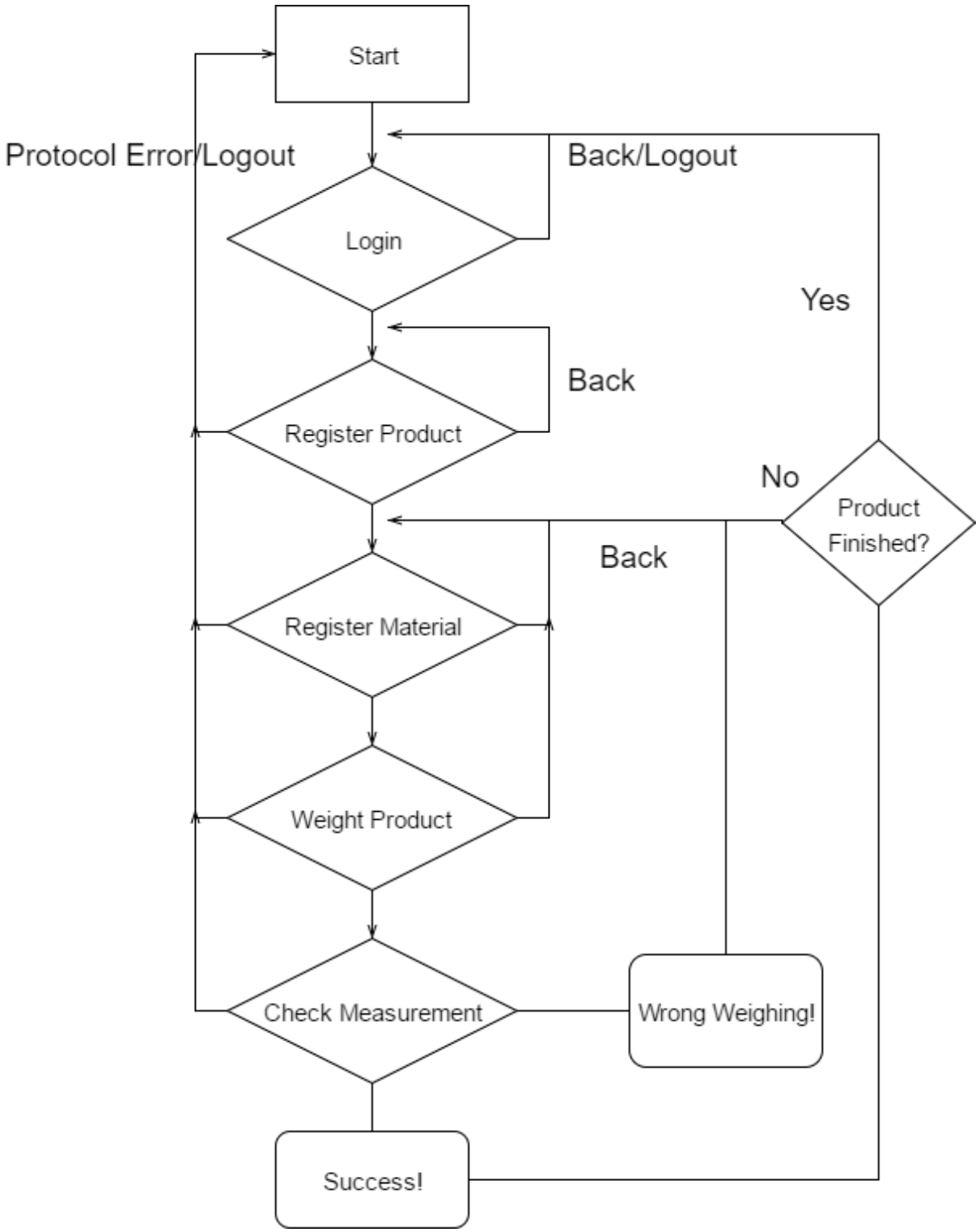


Figure 6: Weight control flow

Finally, the *WeightController* sends the measurements to the *MeasurementController* that receives them in the form of a *MeasurementDTO*. The *MeasurementController* then makes sure the data is put in the data structure using the data-access-layer.

ASE summary

The processes described above are summarised in a sequence diagram on figure 7 below. It illustrates when the different objects are used and who they communicate with. You can clearly see how reader first gets the IP addresses. Then the *ConnectionManager* construct the *WeightController*s corresponding to the number of IPs in the original files. The *WeightController* continuously communicate with the *WeightCommunicator* to control the weight. When a product batch is finished the *MeasurementController* receives a *MeasurementDTO*, and then the *MeasurementDTO* is send to the data access layer to be saved.

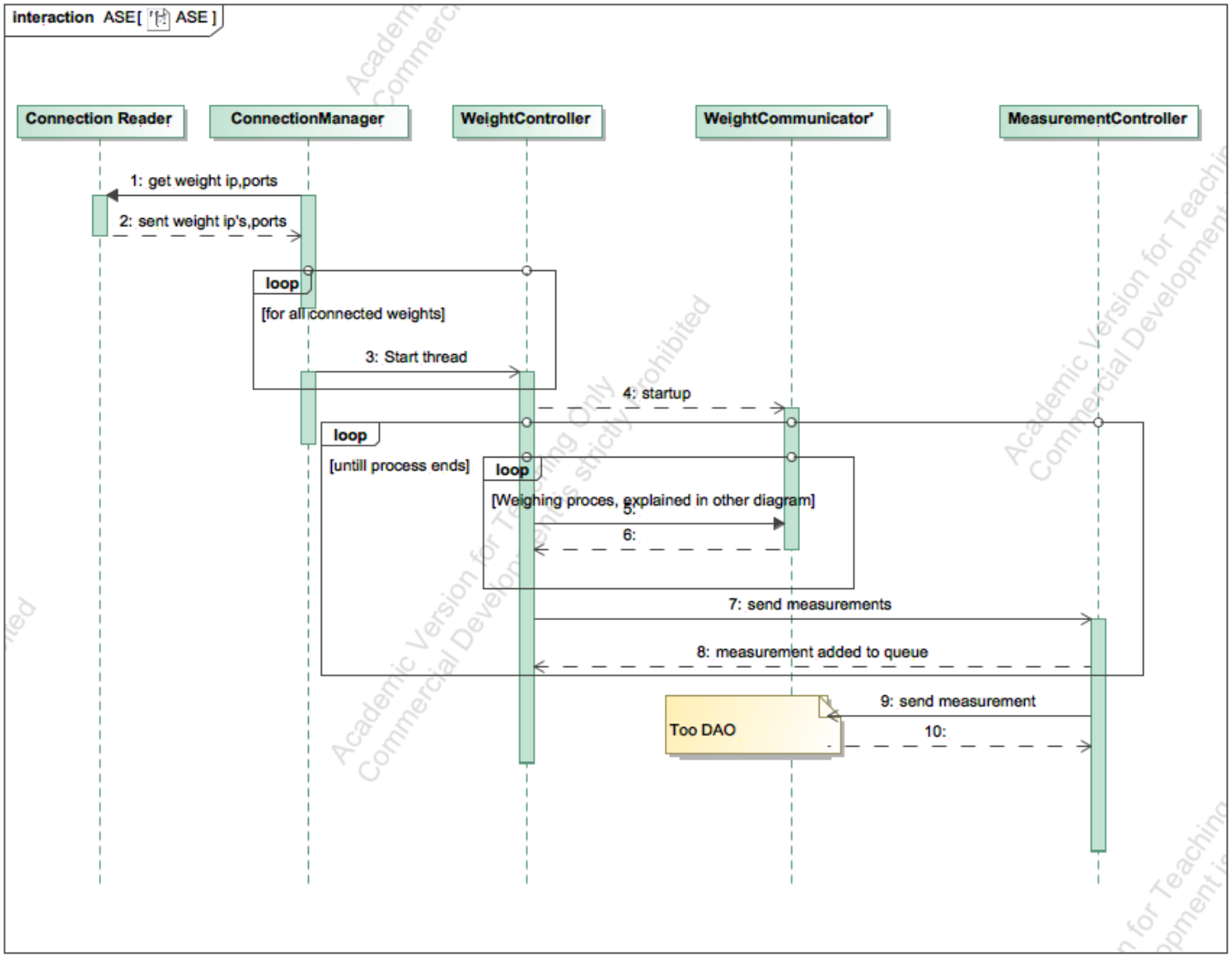


Figure 7: ASE-sequence diagram that shows the program flow for the ASE-system

6 Web-design

In the design phase of the website, one goal was to have a nice and easy to understand folder structure in the IDE used. This was to give a better overview of the many different type of files and their function.
Another goal was to have a great user experience when using the website. Therefore the layout are designed to be minimal and simplistic only containing what the user needs to do his job. This among other things mean that there is no commericials, advertisements or intranet related stuff on the website. The simplistic design is also why it was decided for all of the different pages to retain the same layout to provide the best user experience possible.

6.1 Folder Structure and Overview of Files

The files used for the website are located in different folders. This is done to achieve a better overview over of all the code and its functions. Below is an overview of the different folders and their purpose. All the folders are located in the *WebContent* folder.

- *dist* This folder contains the cascading style sheet and javascript file used for the website. These files are auto generated after running the Ant Build on the *build.xml* file, more about this in section 9.5.
- *lib* This folder contains all the library files used on the website, such as Bootstrap, jQuery, Mustache and more.
- *src* This folder contains all the source files. The sub folders and their content is described in the following section.

6.1.1 Source files in WebContent

JavaScript Files

The *js* folder contains all of the JavaScript (JS) files. Instead of having one JS file with all the JS code used in the project, the front-end development team decided to split it up into several files, which is then combined into one file with the Ant Build to reduce the number of HTTP requests on the website. All JS used for the users are located in *user.js* file, all JS used for the recipes are located in *recipe.js* file etc.

Cascading Style Sheet Files

The *css* folder contains the cascading style sheets (CSS) used on the website. All the styles are gathered in one *layout.css* file. The styles could also have been split into different files, to keep a better overview of the css styles, but since there is not as much CSS as JS, this was not needed and therefore not prioritised.

HTML Files

The *html* folder consists of several HTML files, which is used as templates and loaded in on the page with jQuery. This is done to achieve a single-page application where the website's content is loaded dynamically into one HTML *index.html* file.
Most of the HTML templates are rendered with the *Mustache* library, but more about this in section 9.2.

6.2 Layout

Figure 8 below displays the website's layout. The header of the website displays the logo (see 1) on the left and the logged in user on the right. The user dropdown (see 2) displays the user's role and name in the system. When dropped down, the dropdown contains a link to the user's profile, where they can edit their profile settings, and a link to log out.
The navigation menu (see 3) is a quick way to access the website's different pages. All the pages' content (see 4) is shown to the right of the navigation menu. All administration pages also includes a button (see 5), which functions as a link to the creation page of the appropriate item.

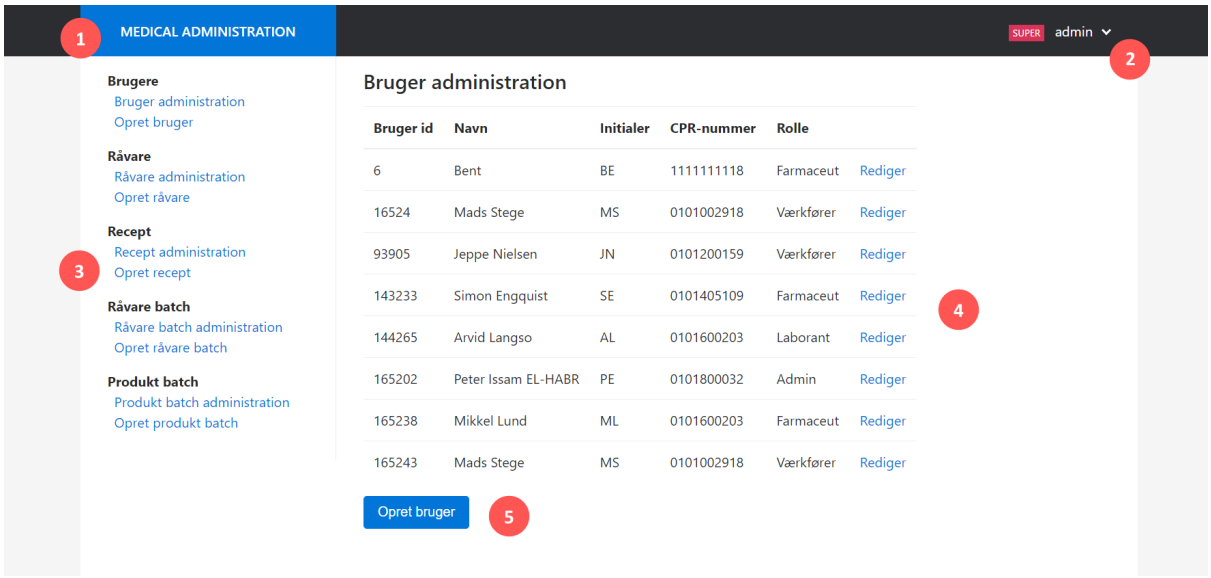


Figure 8: Website layout - User Adminstration

6.3 Forms

All the forms on the website are clear and simple to use, see figure 9 below. All the different fields are validated using jQuery Validation. The fields are validated to ensure that the correct data is sent to the data layer and to prevent fatal errors on the system, see section 9.4 for further information of how it has been implemented.

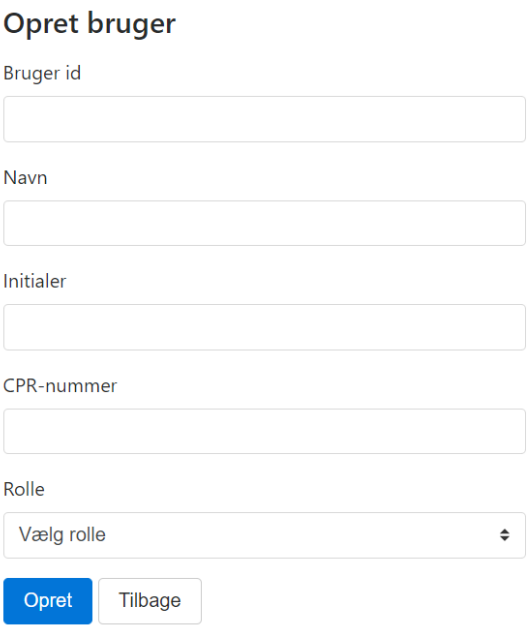


Figure 9: User Creation Form

6.4 Alert Windows

When the user submits forms, the user is given feedback from the system whether his submit was successful or if an error occurred. Figure 10 shows a successful alert window with the message when a administrator resets one of the user's password.

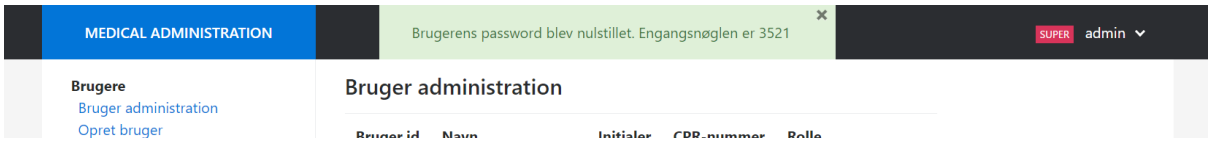


Figure 10: User Password Reset Alert Window

7 Design of the data structure used in the data Layer

There are four possible solutions for a Data Layer design. An external MySQL database, an internal database with a .db file using SQLite3, a temporary ArrayList, or a permanent storage. The last solution would have one or more files converted to a Java object with the FileStream using serializing/deserializing.

After analysing the project, it was decided to first use a dynamic list and then implement the files with the .data extension of Java objects. The idea of an external MySQL database has been abandoned to avoid the possibility of loss of connection during the exam and a lack of hosting options as well as reasons explained earlier. SQLite3 was not chosen because it does not include Stored Procedures, and was deemed a necessary feature, to prevent SQL-injections.

Further ruminations on database design is explained in section 14.5.

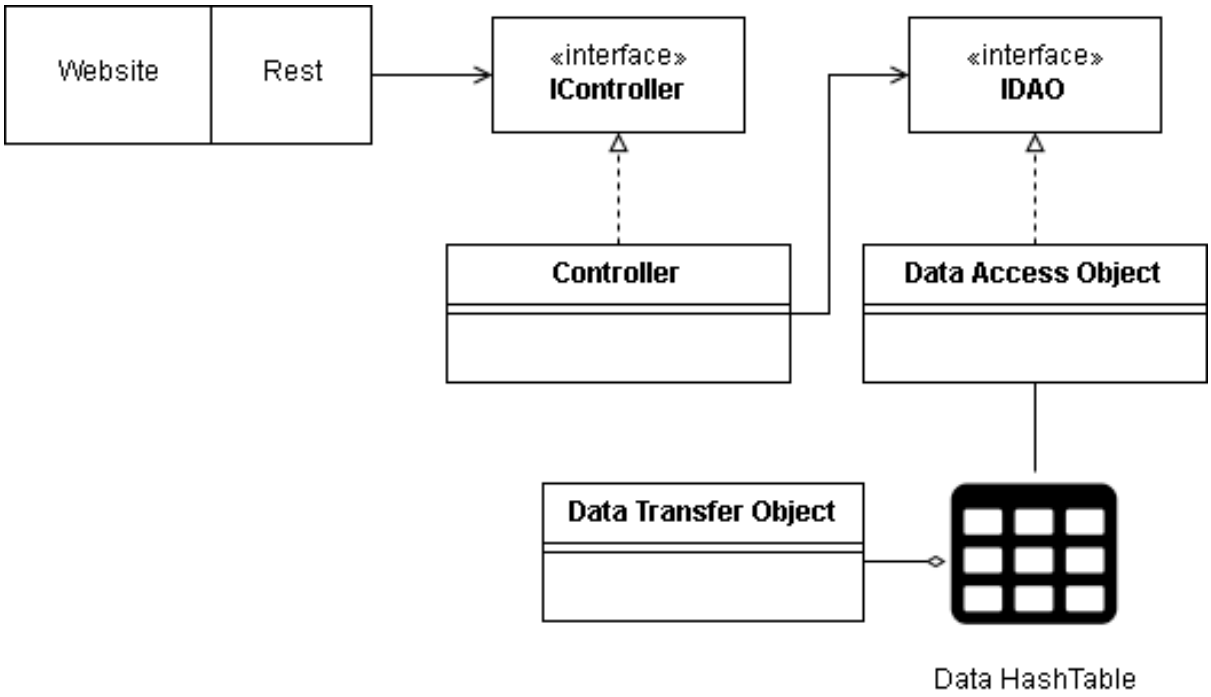


Figure 11: This domain model shows how the data and control layer is set up on the web server.

Figure 11 above describes how elements (entities, controllers, interfaces) of the system are organized. There are six different data types: *ProductBatchComp*, *ProductBatch*, *RawMaterialBatch*, *RawMaterial*, *RecipeComp*, *Recipe* and *User*. Each of the different data types has a corresponding DTO, DAO, controller and REST (CRUD) and interfaces for those. The LoginController also exists but is represented in another way because it is not stored permanently. The LoginController, however does use the User from the data layer. Therefore you could say it follows a similar pattern.

Data Transfer Objects

DTO (Data Transfer Objects) classes are the data that must flow through the the system. For example, if the administrator wants to create a new user, the system will need to use the UserDTO to store the data for these attributes. Then the instance of the UserDTO object will traverse through the system from the Javascript present on the Website to the DAO (Data Access Object). This is done via the appropriate Rest and Controller used in the scenario of creating a new user. In the figure 12 below an example of how the DTOs are structured is shown.

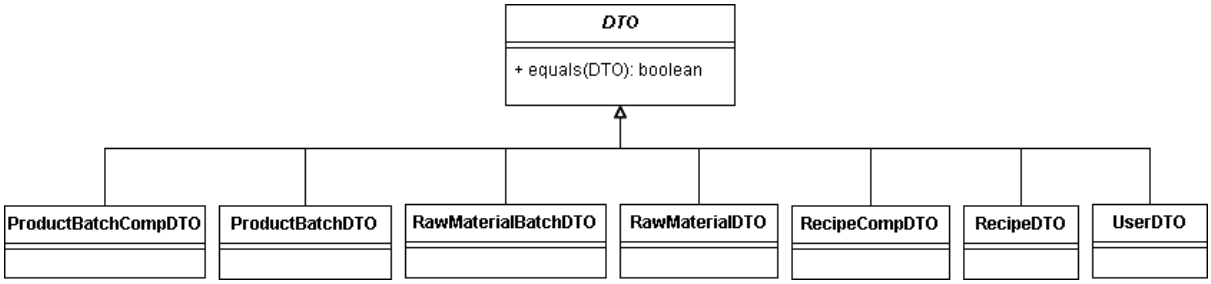


Figure 12: DTO Class Diagram

Data Access Object

DAO (Data Access Object) are the classes that will receive data transmitted from the system and put it in the data layer. In the framework of the project, these will allow one to store information in a HashTable, using the ID of the object or ArrayList if there are several IDs to be handled.

Plain Old Java Object

POJO (Plain Old Java Object) are very simple objects containing only some of attributes in the original DTO. These POJO classes other than a few attributes only have constructors, getters and setters.

Controllers

The Controllers are used to direct the system, linking the Rest and the Data Layer. They also check certain parameters before using the DAOs like Input Exceptions that are also detectable with the Validator class. There is a Controller for each DAO / DTO / Data type.

8 Security design

To secure the data, it was decided to include a user password encryption tool. This tool is the Non-Keyed Cryptographic Hash Function called SHA-512. This hash function has been chosen because its not decryptable, and is the recommended hashing standard. For more security, the function is used as: H(password + userID), the purpose of setting the UserID is to guarantee a unique Hashed Key in the data. Then, in incase of data leak, the data retrieved from the file USER.data, does not contain any hashes that could be easily decrypted.

8.1 Reset Password

To allow *Admins* on the website to be able to reset the password of the different users without being able to see it, an Admin Key system was set up. When an Admin presses the corresponding button, a message is sent to him containing the message that the password has been reset and the generated key. This key shall be given to the user in some way. The user can then use the key as a one time password. This allows them to be able to define his / her password. By default, when creating or resetting passwords, the password is automatically generated and can not be used because only the Admin key allows it to be redefined.

If you for some reason want to change your password you can do that yourself without the help of an *Admin*. Go to your user profile, which can be found in the top right corner. Now follow the instructions there, to change your password. When changing your password yourself, the admin key is not needed as long as you know your old password.

9 Implementation - Web development

Many different libraries and frameworks have been used for the website. These are all explained below with code examples.

9.1 jQuery & JavaScript

All of the website's functionality is implemented, using jQuery and JavaScript.
jQuery is a fast, small, and feature-rich JavaScript library, which makes things like HTML document manipulation, event handling, animation, and Ajax much simpler.
In the project the developers bind the different events, such as submitting forms and clicking links with jQuery. Below is a code-snippet of submitting the login form.

```
16    $(document).on("submit", "#login_form", function(event) {
17      event.preventDefault();
18
19      userLogin($(this).serializeJSON()).done(function(data) {
20        onLoginShowPage(data);
21      }).fail(function(data) {
22        console.log("Fejl i Login REST");
23      });
24    });
```

When submitting the login form, the userLogin(form) function is called. The function takes the form's data serialised to JSON as parameter. The function then makes a POST Ajax request to the REST layer, and if the Ajax request succeeds, the onLoginShowPage(data) is then called. Look below to see the onLoginShowPage(data) function code.

```
83  function onLoginShowPage(data) {
84    var splitData = data.split(": ");
85    var userId = $("#login_form input[name=\"id\"]").val();
86    $("#login .alert").remove();
87    switch(splitData[0]) {
88    case "super_login":
89      $.get("src/html/master.html", function(template) {
90        $("body").html(template);
91        $("#user_dropdown_menu .user_edit_link").hide();
92        $(".top_nav_role").addClass("role_super");
93        $(".role_super").text("Super");
94        $(".top_nav_name").text("admin");
95        showStartPage();
96
97        getRoleTemplate("src/html/role_privilege/admin_privilege.html").done(
              function() {
98          getRoleTemplate("src/html/role_privilege/pharmacist_privilege.html"
                ).done(function() {
99            getRoleTemplate("src/html/role_privilege/foreman_privilege.html")
                  .done(function() {
100             getRoleTemplate("src/html/role_privilege/labtech_privilege.html
                    ");
101         });
```

```
102             });
103             });
104         });
105         break;
106     case "new_login":
107         showNewLoginPage(userId);
108         break;
109     case "true_login":
110         showFullPage(userId);
111         break;
112     default: // not logged in
113         $("#login").find(".form-group:last").prepend("<div class=\"alert alert-
            danger\" role=\"alert\">" + splitData[1] + "</div>");
114     }
115 }
```

The onLoginShowPage(data) takes the REST response as parameter and depending on the response, which is the type of login, different events happen. The login types are *super_login*, *new_login*, *true_login*, and *not_login*.

The *super_login* only happens if the super admin is logging in. The *Super Admin* data is then displayed. More on what he can in section 4.1

The *new_login* happens if a user's password has been reset by an *Admin*, and when the user login using the temporary login key, the user is then forced to make a new password.

The *true_login* happens if the user id and password matches in the data layer, and the user is logged in normally.

The *not_login* happens if the user id and password does not match, and an error message is shown.

9.2 Mustache

Mustache is a logic-less template syntax and can be used for HTML files. Logic-less meaning there are no if statements, else clauses, or for loops. Instead only tags are viable.
It works by expanding tags in a template using values provided in a hash or object.
In the project, Mustache is used to render most of the HTML templates. The Mustache render is provided with a JSON object as data, and when it renders the template, the variables surrounded with the Mustache tags {{}} will be replaced with the data in the object. Below is an code snippet of the edit-user-Mustache-template. This way the edit-user form is filled with the user's data when the user's profile page is displayed.

```
1 <div class="form-group">
2     <label>Navn</label>
3     <input type="text" class="form-control" name="name" value="{{name}}">
4 </div>
```

And to insert the rendered template into the website's content element, we use the following code.

```
1 $.get("src/html/product_batch/product_batch_edit.html", function(template)
    {
2 $("#content").html(Mustache.render(template, data));
3 });
```

9.3 Bootstrap

The website layout is build upon the Bootstrap framework. Bootstrap is used to easier apply CSS styles and functionality to the different HTML elements. The website uses the newest bootstrap version, v4.

9.4 jQuery Validation

All of the website's forms are validated using the jQuery Validation library. The library makes it super easy and fast to validate the user's input on the client side, for a responsive user experience, see example below for the implementation. The library also allows the implementation of your own methods. We have used this to validate a user's CPR-number, Password and more. While validating the user input on the client side is effective, validating the user's input on the back-end is also important. This is important if the user tries to bypass the validation by disabling JavaScript for example. User input error handling is discussed in section 11.1.

```
261 function validateProductBatch(form) {
262   $(form).validate({
263     rules : {
264       pbId: {
265         required: true,
266         validID: true
267       },
268       recipeId: {
269         required: true,
270         validID: true
271       }
272     }
273   });
274 }
```

9.5 Combining the JavaScript Files

Since all the JS code are split into several files, the development team decided to combine all the code from the different files into one, to reduce the number of HTTP requests on the website. This is achieved by using *Ant*, which is a Java-based build tool. The Ant build tool is run by the script *build.xml* located at the root of the *WebContent* folder.
The script is also using the Yahoo! JavaScript and CSS Compressor (YUI Compressor) to minify the JS and CSS. Minifying meaning removing white spaces and comments to reduce the file size. The minified files are not loaded on the website to make debugging easier, and the YUI Compressor is only used to experiment with the Ant build tool.

9.6 REST

To make a connection between the front end and the back end the development team is using the Representational state transfer (REST) or RESTful API. Below is a code-snippet of the REST method createUser.

```
64    @Path("create")
65    @POST
66    @Consumes(MediaType.APPLICATION_JSON)
67    public String createUser(UserDTO user) {
68      try {
69        int key = Initializer.getLoginController().generateAdminKey(user.
              getId());
70        Initializer.getUserController().createUser(user);
71        return "success: Brugeren blev oprettet med id " + user.getId() + "
              og engangsnøglen " + key + ".";
72      } catch (InputException e) {
73        return "input-error: Det indtastede er ugyldigt.";
74      } catch (CollisionException e) {
75        return "collision-error: Der eksisterer allerede en bruger med det
              indtastede id.";
76      } catch (DALException e) {
77        System.out.println(e);
78        return "system-error: Der skete en fejl i systemet.";
79      }
80    }
```

The method is called by the JavaScript in the code below. The function makes an Ajax POST request to the url *rest/user/create*, and posts the form data to the REST layer, which then converts the form data (JSON object) to a UserDTO. The createUser method can then use the UserDTO to update the data layer with the new user.

```
201 function createUser(form) {
202   return $.ajax({
203     url : 'rest/user/create',
204     type : 'POST',
205     contentType : "application/json",
206     data : form
207   });
208 }
```

10 Implementation - ASE

The ASE works as a seperate entity to the rest of the system. The ASE structure and functionality is described in rough terms in section 5 of this report. The ASE consists of a bunch of different parts that make up the ASE system. These parts are described below.

10.1 Controlling the weights

For each weight, the system constructs a WeightController. The WeightController is responsible for controlling the weight so that the user can measure products. The WeightController uses the WeightCommunicator to communicate with the weight. More on this in section 10.2.

In order to facilitate multiple weight, the system had to make use of multithreading. This means that the weight controller implements the Runnable interface. This way, it is able to run multiple weight controllers simultaneously. This means there will be one socket connection for every weight in the system.

The Weight Controller has a run method. This method controls the measurements. The code below is the primary control of the weight.

```
112     while (true) {
113       try {
114         // Prompts the user to login.
115         login();
116         // Prompts the user to choose a product batch.
117         registerProduct();
118         // Starts measuring materials.
119         while (true) {
120           // Ask to enter Raw Material id.
121           registerRawMaterial();
122           // Measure materials.
123           ProductBatchCompDTO measurement = measureRawMaterial();

135           if (remainingReceptComp.isEmpty()) {
136             break;
137           }
```

Some of the code is missing, but is not needed for the explanation. First, the user is asked to log in. Then, the product number is entered. Finally, the loop keeps registering materials and measuring them, until all the materials have been measured. The measurements are then stored and the code above is repeated forever, until the system shuts down, or system breaking error occurs.

10.1.1 Extra features

It was expected for the user to be able to log out and go back in the system. This would allow the user to stop what they were currently doing if something went wrong etc. The two types of events have been handled in two different ways. For the log out option, a LogOutException has

been implemented. This exception is thrown whenever the user press log out on the weight. The log out button has been chosen to be the *On/Off* button on the weight. The LogOutException is only caught in the main run method. Basically, it restarts the program back at the start point. The code below shows the catch of the LogOutException:

```
145         catch (LogOutException e) {
146           try {
147             sendMessageAndConfirm("Du er nu logget ud.");
148           } catch (ProtocolErrorException | LogOutException e1) {
```

The above is an example of the exception being caught so that the system can continue. The back button, which has been chosen as the (>0<) button, goes back to what the developers thought was the most sensible part of the measurements. This is handled in different ways. For instance, when the user registers a material, if one clicks back, the system should redo the ”register material” part again. This split up code illustrates how this process works:

```
263       do {
264         try {
265           buttonConfirmation = getDTOAndConfirm(rbDTO, rbDAO, "rb id", "rå
              vare id");
266           if (buttonConfirmation == Buttons.BACK)
267             continue;

282       } while (buttonConfirmation != Buttons.CONFIRM);
```

As shown in the example above, there is a do-while loop that waits for the confirmation button to be pressed. If the button is a back button, the code immediately continues the loop. In other parts of the code, this is handled differently to achieve different goals.

10.1.2 Error Handling

In order to have system errors thrown, the code makes use of two different exceptions. The *InvalidReturnMessageException* and the *ProtocolErrorException*. The *InvalidReturnMessage-Exception* is used when the system is running specific parts of the code, for instance when it is attempting to enter a product id and some unexpected message is sent from the weight. To handle this, the *InvalidReturnMessageException* is caught and the last protocol message is printed to the console too see if the user pressed a wrong button. If the button is what caused the exception being thrown, the appropriate action is taken according to what button is pressed. Here is an example:

```
426       } catch (InvalidReturnMessageException e) {
427         return weightCommunication.receiveButtonPush();
428       }
```

The exception is caught, and the function checks if a button was pressed.

If it wasn't a button press, the *ProtocolErrorException* will be thrown. This means that the program has received either an unknown command, or a command that was out of sync. When this happens, the code restarts the *WeightController* in order to remove any potential leftover errors. The user can then begin the login again after receiving an error message. Here is the exception caught in the run method.

```
154         catch (ProtocolErrorException e) {
155           weightCommunication.restartWeightDisplay();
156
157           try {
158             sendMessageAndConfirm("Systemfejl!");
```

After catching the exception, the weight display is reset and a system failure is sent to the user.

10.2 The WeightCommunicator

The *WeightCommunicator* knows some of the commands used by the industrial weight. Its task is to translate the messages between the *WeightController* and the weight itself. When the *WeightCommunicator* receives a command to send a message and wait for user response, it wraps the message into a RM20 protocol.

```
53    public void sendProtocol(Protocol protocol, String message) {
54      try {
55        switch (protocol) {

59        case RM20:
60          outToWeight.writeBytes("RM20 8 \"" + message + "\" \"\" \"&3\"" + "
              \n");
```

The Java socket library wraps the message in a TCP header, and then in a IPv4 header to send it to the weight.

```
42      outToWeight = new DataOutputStream(mySocket.getOutputStream());
```

When the *WeightCommunicator* receives the package from the weight, it checks if it is an acknowledgement. Then it checks if the message matches the expected acknowledgement according to the RM20 command specification. If it does not recognise the command, it throws an exception, and saves the message to resolve the problem later.

```
164     splitAnswer = answerReceived.split(" ");
165     splitAnswer[1]=String.valueOf(splitAnswer[1].charAt(0));
166     //Checks for acknowledgement
167     try {
168       if(splitAnswer[0].contains("RM20")&&splitAnswer[1].contains("B")){

173       else{
174         previousMessageRecived=answerReceived;
175         throw new ProtocolErrorException(answerReceived);
```

After the first acknowledgement, the next package should contain the answer. If this is not the case, the RM20 function handler in *WeightCommunicator*, will throw an exception. The other commands have similar checks and rules, some of the functions however, only expects an acknowledgement.

10.3 MeasurementController

This class runs a single separate thread to ensure that every *WeightController* can save the measurements without having collision problems in the DAO layer due to threads accessing memory simultaneously. This class implements the *IMeasurementController* interface. This interface is used by the controller to enqueue measurements, so that they can be put into storage. It then continuously run the dequeue method in order to add measurements to the data access layer. The code below is the part of the queue that adds all the recipe components to the data access layer.

```
44        temp = measurements.remove();
45        for (ProductBatchCompDTO comp : temp.getMeasurements())
46          try {
47            productBatchCompDAO.createProductBatchComp(comp);
48            System.out.println(comp);
49          } catch (DALException e) {
50            System.out.println("Could not add ProductBatchComp: " + comp);
51            e.printStackTrace();
```

10.4 Connecting the weights

ConnectionReader

The ConnectionReader is responsible for reading and verifying the IP addresses and port numbers on all of the listed weights in the system.

The method *getWeightIPs* attempts to read the *WeightTable.txt*, and extract useful information from it. The *ConnectionReader* scans for information for as long as the text file has information.

The key component in *ConnectionReader* can be seen below.

```
61            if (tokenCheck.equals("IP")) {
62              weightIP = weightScanner.nextLine().trim();
63              weightScanner.skip("PORT");
64              String weightPort = weightScanner.nextLine().trim();
```

As seen in the code above, the *ConnectionReader* relies heavily upon a set of checks and for the proper syntax to be used. Firstly, it tries to see if the information is set up in the correct way, as discussed in section 5. If it is, it begins separating the information bits into useful pieces.

```
70              if (validateIP(weightIP)) {
71                if (validatePORT(weightPort)) {
72                  allIPAddresses.add(weightIP);
73                  allPortNumbers.add(weightPort);
```

These data bits are passed onto the *validateIP* and *validatePORT* methods, verifying the IP and port number respectively. If these checks are passed, the IP and port number will be added to two separate Arraylists for later use. The program look for the following typical errors:

- Wrong amount of periods (.) in the IP.

- Invalid characters (letters or special characters) entered into the IP or port number.
- Invalid IP address (0.0.0.0) entered into the IP.
- Numbers in the IP address and port number are too small (less than zero) or too big (greater than 255 or 65.535, for either IP or port respectively).

If the *Reader* encounters an error in either of the checks, it prints out the offending weight information, as well as what type of error occurred.

The *ConnectionReader* is also equipped to handle missing files. If the WeightTable.txt does not exist in the specified folder, the program will attempt to read a similarly named file in the default folder (The default folder is the user's Documents-folder). Should the Documents folder not contain a *WeightTable* file either, the program makes its own text file, names it *WeightTable*, clean it of left over text - if any - and inserts an example of a weight.

ConnectionManager

The *ConnectionManager* is responsible for creating a socket and thread for each IP and port number received from the *ConnectionReader*. The procedure is relatively simple to make. For each IP found, create a new socket. On each socket connection, create a thread, so that the system can communicate with multiple weights simultaneously.

```
50  public void threadStarter() {
51    try {
52      connectionReader.getWeightIPs();
```

The *ConnectionManager* first tries to establish a connection to the *ConnectionReader*. Upon a successful connection, the program begins constructing new sockets using IPs and ports retrieved from the *ConnectionReader*.

```
54    for (int i = 0; i < connectionReader.getAllIPAddresses().size(); i++) {
55      try {
56        weightSocket = new Socket(connectionReader.getIPString(i),
              connectionReader.getPortInt(i));
57        weightController[i] = new WeightController(measurementController,
              weightSocket);
58        (new Thread(weightController[i])).start();
```

Having created these sockets, the *ConnectionManager* begins starting creating *WeightControllers* in new threads for each socket.

11 Implementation of the Web Controller

11.1 Error-handling

To keep the system running properly in the event of errors, it was neccessary to include good, solid coding, despite the potential erros that the system could generate. For this reason, in the Controllers and Data section, almost all methods include Thrown Exceptions. This makes it possible to clearly distinguish between the different kind of errors in the system. The Thrown Exceptions also allows for the system to utilise Try/Catch statements for better error messages back to the user. Some errors, such as those from a bad user input, will be printed on the user screen.

DAL Exception

DAL (Data Access Layer) Exception is a system error that affects only the part responsible for data management. This error comes from the DAO classes, and for most of the time, indicate that data could not be found or written.

Input Exception

This type of exception is an error caused by a user input such as, for example, an invalid password. It is sent by the Controllers and the *Validator* class with static methods used to test the input values. The Input Exception is an extension of the DAL Exception.

Collision Exception

The collision error occurs when creating data with one or more unique ids, which defines a specific element that already exists. For example, it is not possible to create two users with the same id. The second attempt to create such a user will result in a Collision Exception. The Collision Exception is also an extension of the DAL Exception.

11.2 Login Controller

The LoginController is quite important in the system, because it allows users to log in, but also to reset their password.

The method *checkLogin* takes in the parameter *LoginPOJO* containing a user id and a password. Using the method present in the *LoginPOJO isSuperAdmin* which is a method with a simple if statement returning a boolean, it can decide if the user is a *Super Admin*. If it is true, then this one will return a *LoginState.SUPER*.

Listing 1: Java code example of LoginState enum

```
    public enum LoginState{
        TRUE,
        FALSE,
        SUPER,
        NEW
    }
```

In other cases, the Controller must check if the user connects by entering an admin key. This key is present when creating a new user or resetting the password. It is generated by the *generateAdminKey* method which will insert a value between 1.000 and 11.000 in a HashTable with the user id as the key and will return it as an integer. To verify that the entered password is a key, the method will try to convert the password to an integer number. If it is received and the key corresponds to the table, then the user tries to access the website via a new admin key and the method returns *LoginState.NEW*. If it does not, the Controller will call the *UserDAO* to try and match the entered password with the user data. To do this, it takes the password followed by the user id and hash it with the *SHA512* function such as *Encryption.sha512* (password + user id). If the hashed result is equal to the value retrieved in the data, then the password is valid and the method returns *LoginState.TRUE*, else *LoginState.FALSE*.

Listing 2: Java code checkLogin method

```
try {
            if (user.isSuperAdmin())
                return LoginState.SUPER;
            else if (adminKeyTable.remove(Integer.parseInt(user.getid())).
                intValue() == Integer.parseInt(user.getPassword()))
                return LoginState.NEW;
            else
                return LoginState.FALSE;
        } catch (Exception e) {
            try {
                if (dao.getUser(Integer.parseInt(user.getid())).getPassword
                    ().equals(Encryption.sha512(user.getPassword() + user.
                    getid())))
                    return LoginState.TRUE;
                else
                    return LoginState.FALSE;
            } catch (Exception e2) {
                return LoginState.FALSE;
            }

        }
```

When a new user is created by the *UserController*, its password is generated randomly according to the rules of the generatePassword method. The method creates an unusable password starting with “XX”, plus a number between 1 and 1.000.000.000 and hashes it with the id. This password is not unusable because it is not shown to the Admin who creates/updates the user. To allow the user to obtain a valid password, use the *setNewPassword* which is the only way to update it correctly.

The system also allows for users to reset the password with the *resetPassword* method using the resources previously seen.

11.3 Initializer

The *Initialiser* class that implements the *ServletContextListener* interface creates the various objects that the system will need to function such as the different Controllers and DAOs. It also allows several operations to be performed when the server is started and closed. When it is opened, the DAOs will read the various .data files to retrieve the stored data and write the logs in real time to the server console.

Listing 3: Java code 2 interface methods

```
    @Override
  public void contextInitialized(ServletContextEvent sce) {
  System.out.println("Listener initialized");}

    @Override
  public void contextDestroyed(ServletContextEvent sce) {
    System.out.println("Listener destroyed.\n");
  }

\section{Data Access Layer}
    \subsection{Data Access Object:}
For the DAO implementation, one of two data structures is used as static
   storage, which means that it is the same for each DAO instance. All DTOs
    stored in DAO structures are copies to prevent unwanted modification by
    pointers. This has been implemented like this in DTO:
 \begin{lstlisting}[language=Java, caption=Java code example of copy method
     in UserDTO]
  public UserDTO copy() {return new UserDTO(id, name, ini, cpr, password,
     role);}
```

All access classes have public methods get, getList (overload if it is with id in parameters), update and create. These methods provide the data management commands that only the Controllers can use through the Initializer which gives them the corresponding DAOs. When the constructor is called, it will retrieve the data stored in the assigned file, create it if it does not exist, and replace the dynamic storage with the file that provides permanent storage to the system using the static class *FileManagement*.

Listing 4: Java code example of UserDAO's constructor

```
  /*
   * The warning "unchecked" is there because Java can not define if the
      file we try to convert
   * to a HashTable is associated to this class.
   * We decided to ignore this warning in all our DAO.
   */
  @SuppressWarnings("unchecked")
  public UserDAO(){
    try{
      System.out.println("Retrieving User Data...");
      userList = (Hashtable<Integer, UserDTO>) FileManagement.retrieveData(
         TypeOfData.USER);
      System.out.println("Done.");
```

```
12
13      }catch(Exception e){ //if we can not read the file then it is missing
14        System.out.println(e);
15        System.out.println("Trying to create the saving file...");
16        FileManagement.writeData(userList, TypeOfData.USER);
17        System.out.println("Done.");
18      }
19    }
```

The methods change a lot compared to the two types of structures used, which is the *HashTable* and the *ArrayList*. Here is a comparison between the two with the get:

Listing 5: Java code example of UserDAO's getUser method

```
1
2   public UserDTO getUser(int id) throws DALException {
3
4     if(userList.get(id) != null)
5       return userList.get(id).copy();
6
7     else
8       throw new DALException("Unknown Userid: " + id);
9
10  }
```

Listing 6: Java code example of ProductBatchCompDAO's getProductBatchComp method

```
1   public ProductBatchCompDTO getProductBatchComp(int pbid, int rbid) throws
        DALException {
2
3     for(ProductBatchCompDTO productbatchcomp : productBatchCompList){
4
5       if(productbatchcomp.getPbid() == pbid && productbatchcomp.getRbid()
            == rbid)
6         return productbatchcomp.copy();
7     }
8     throw new DALException("Unknown Product Batch Comp id: " + pbid + " " +
          rbid);
9   }
```

When using an *ArrayList*, it is necessary to browse all the data before finding the one that matches both ids. This is why it was decided to make *HashTable* implementation a high priority.

11.4 FileStream

The default file path, as mentioned before, it is necessary to have the different folders to get the permanent data stored safely onto the system. This however poses a problem for a DTU databar computer. This is caused by the fact that there is no Documents folder. It has to be made manually.

Listing 7: Java code example of PATH variable

```
final static String PATH = "C:\\Users\\"+System.getProperty("user.name")+"
    \\Documents\\";
```

The *FileManagement* class is used to manage the various files necessary for the functioning of the system. To do this, it uses Input / Ouput of the *FileStream*.

When writing a file, *FileOutputStream* is used with an instance of the class *ObjectOutputStream*. This allows the system to write a java object of any data type into a file that will be named by the extension .data if it comes from DTOs lists. The different types of data must be inserted in the parameters of the method with an *Enum*.

Listing 8: Java code example of TypeOfData enum

```
    public enum TypeOfData{
        PRODUCTBATCHCOMP,
        PRODUCTBATCH,
        RAWMATERIALBATCH,
        RAWMATERIAL,
        RECIPECOMP,
        RECIPE,
        USER,
        TXT
    }
```

Listing 9: Java code example of writeData method in FileManagement

```
            f = new FileOutputStream(new File(PATH+type.toString()+".data")
                );
            o = new ObjectOutputStream(f);
            o.writeObject(dto);
```

TXT is inserted to write the *WeightTable.txt* file. The variable DTO is the object that shall be written to the disk, in the program it corresponds to a *HashTable* or an *ArrayList*. Type is the *Enum TypeOfData*.

During the file reading, the *FileInputStream* and *ObjectInputStream* classes are used to access the desired files and convert them into objects.

Listing 10: Java code example of retrieveData method in FileManagement

```
            fi = new FileInputStream(new File(PATH+type.toString()+".data")
                );
            oi = new ObjectInputStream(fi);
            Object dto = (Object) oi.readObject();
            return dto;
```

12 Data Structures

Hash table

A *HashTable* is a data table defined by a key and an element, written as (Java):

Listing 11: Java code example of UserList implementation as a static HashTable

```
static Hashtable<Integer, UserDTO> userList = new Hashtable<Integer, <
    UserDTO>();
```

The key is on the left side (Integer) and the right side contains the data element DTO.

It was decided to take this solution as a structure for all data not including *RecipeComponent* and *ProductBatchComponent*. This makes it possible to store the different DTOs in a table with the index equal to theirs id and to optimise the running time for the different CRU (Create, Read and Update) methods. If a user is inserted with id 45, their key will be 45 and it will be stored in the table. It is also possible to know if the location is already used or not to avoid replacing the current data with unwanted data. Because of incompatibility with the rest of the system, this solution is only used to store the data and not to process them.

ArrayList

An *ArrayList* is a dynamic list of objects from a specific class.

Listing 12: Java code example of ProductBatchCompList implementation as a static ArrayList

```
static List<ProductBatchCompDTO> productBatchCompList = new ArrayList<
    ProductBatchCompDTO>();
```

In the *RecipeComponent* and *ProductBatchComponent* examples, two ids are used to retrieve unique item. It was therefore impossible to use a *HashTable* because it uses only one id as a key. The advantages of the *ArrayList* is that it is easy to use and it has a good coupling with a big part of the system used in *Database Programming course*. The downside is the running time for different algorithms that takes longer. For example, it is necessary to browse the list to find the correct index assigned to an object. The *ArrayList* is used by the vast majority of the system and is an extension of the *List* class.

13 The different test

13.1 ASE-system test

In order to test the ASE system, a variety of different tests were needed. The classes that was fairly simple and easy to test has all been tested using JUnit, to make sure the class performs its task satisfactory on its own. These classes include:

- WeightCommunicator
- WeightController
- ConnectionReader
- MeasurementController

All of these unit test can be found as JUnit tests in the code. Most of these has been white-box tested, where every single possible branch in a function was tested. However due to the nature of *WeightController*, this has been black-box tested. Since the weight controller is self governing, no outside class calls it methods. It has therefore been given some information, and as well as a place to give output using Class stubs. Finally, the program checks to see if it gets the expected output given some input. The *WeightController* has not had a full unit test where every possible branch is tested due to time constraints.

The testing team has also performed a variety of different integration test. For instance the *ConnectionManager* is tested using the *ConnectionReader* since these are very closely integrated. The team has also tested the *WeightComunicator* and *WeightController* together. This was done by setting these up on one PC, and then connecting to *WeightCommunicator* from a different PC. The team can then start mimicking proper weight responses, and send these to the *WeightCommunicator*. This way, the team could test that the controller acted appropriately to the different weight responses. This was used to fix many smaller bugs in the protocol handling that had not been anticipated initially.

At last, the ASE-system was put together in order to perform a system test including the weight. The team ran the system on a computer connected to weight. Then a large amount of tests were run to ensure that the system worked as intended.

The team has tried to test different test cases in all layers of testing. Some of these are tested across different classes. A list of the test cases can be found in appendix C.

In general, the test succeeded and the team has only encountered a few bugs in the ASE-system. One very serious bug crashed the system. This however only happens if a user types nothing into an RM20 input. The team do not expect this to happen much, but the error should still be corrected in later version of the software before final shipping to the customer.

13.2 Web-server test

By Web-server, the team is referring to the lower layers of the system. In order to do the testing on the layer, unit tests have been performed on some of the classes affiliated. Because of how

the system is set up, most of the DAO's are very similar. Therefore, the time used unit testing can be cut down, by only testing two of them. These two represent the two different kinds of DAO's as described earlier. Specifically, the team unit tested the *ProductBatchComponentDAO* and the *UserDAO*. A variety of negative and positive test have been performed on the system to make sure the correct errors are thrown when the system catches them.

The controller and their functions have also tested. This was to ensure that the CRUD functions work as expected. Checking that names, id and other information follow the rules that have been set. Three different controllers have been tested, that represent the three different types, using unit testing. These worked as expected.

The entire Web-server has been tested during the system test. The team has seen that the system performs as expected, and the team has not been a crash the system. Errors do occur, but the system should handle them and continue functioning regardless.

13.3 Website test

Because of the difficulty in unit testing a website and its components, extensive user- and system-tests were conducted to ensure stability and functionality. All of the needed requirements made by the customer has been tested thoroughly, to verify that they were implemented properly.

A lot of internal testing on the website were also made during the creation process. This was done by religiously testing new features whenever these were implemented onto the website. Along the way, some features and components have been noticed which would have been desirable to have had implemented. The team also discovered some bugs in the system. One bug one could consider is that an admin can reset their own password. This can be used both maliciously, but it can also cause the admin to accidentally deny themselves access to the system.

User tests were organised to test the website using people with very limited knowledge of the website and the company's requirements and business. User friendliness had also been a primary selling point of the website, and as such, was of great interest to have tested.

It was decided to make user tests by using two groups of people. Young people, aged 18 to 25, and seniors, aged 50 plus. This was done to test if both age groups could navigate and use the website efficiently.

All the user-tests can be found in the appendix in section D. Looking at the test results, it is safe to say that the users, with little to no knowledge on the system, were able to perform most of the tasks in the system. One of the tasks failed though, due to a bug in the system. Another test was only partial successful because one user did not notice a pop up message when creating a new account.

Another advantage to the user tests were that previously unnoticed things, such as a lack of weighing units had not been inserted. While these units might come natural to a system designer, some users found it to be a bit confusing. This lack has been fixed in the final version of the system.

Overall, the website appears to be very user friendly, given both the test results and test user

statements. The users completed the given tests with minimal help needed from the instructors. These tests also help prove the website requirements are fulfilled satisfactory.

14 Improvements

Throughtout this project, a number of different ways to improve the program has been discovered. The improvements can either be at the back end, or at the front end of the software.

14.1 Improved security - Both front end and back end

During the project, the team found a few security holes, such as being able to change user information through most of the browsers with the Inspect Element function. When submitting the new information, the server will attempt to handle the new information correctly. This is a very serious security hole, which should be patched before implementation to the public.

Another security hole is that the *ConnectionReader* will read *any* IP address and port number listed in the WeightTable text file. This means that it is technically possible for a malicious person to add a wrong IP address and port number, opening a socket connection between an unsecured IP and system.

However, due to the architecture of the system and the build in weight margins, the effect of this is limited. A wrong IP would at best only send wrong weight measurements back to the user. Since it is the system connecting to the weight, and not the other way around, only a limited range of messages are sent back to the user, and verified there, before any further action is taken. This is however, a security hole which will need to be worked around somehow.

Finally, a token based login system would be great in ensuring that the users of the system would not have to confirm their identity. A token based security system works by assigning a unique token, which can then be used by the user for a short period as a way of confirming their identity.

14.2 More tools and functions

In a system made out of possibly hundreds, if not thousands of ingredients and recipes, a search function would be a godsend in terms of productivity. This is just one thing among many, that could be added to the system to make the user experience better when the amount of data on the website grows.

14.3 ASE Improvements

There are multiple things that can be improved in the ASE system.

During a measurement procedure, a user can not log out during a production, because the user would loose all the progress, they had made on that production. One solution is to store the data on the server when a user logs out.

There are no records of the RawMaterielBatch storage amount changes. This was a requirement but has not been implemented. This should be added in a later edition.

14.4 Optimisation

A system as vast as this one will inevitably contain redundant or unused pieces of codes. These fragments might be the remainders of now untapped resources, or tools. Furthermore, since the system use old pieces of code from CDIO 2 and CDIO 3, some libraries and dependencies might very well be unnecessary.

A thorough run through of all of the resources and system would prove beneficial, albeit very very time consuming. Another all too often problem with optimisation is the risk of ruining legacy code. The main difficulty with working around legacy code is that the developer run the risk of downright breaking completely unrelated functions and classes, if they rely on the removed pieces of code in some obscure way.

14.5 Database

As previously been stated, the team decided against having a database for the initial system. This was mostly due to exam requirements. A database would definitely have been the best data structure for our system. It can perform all the criteria mentioned in the requirements section. A design for the database have however been discussed and the team expect it to look something like the one shown on figure 13.

As shown, the DTOs we have previously mentioned match the tables in the database. This is due to that fact that we designed the system, so that it would be easy to replace the current format with a working database. One could easily use the DAO interfaces to implement a database.

In order to use the database, we would implement views, transactions and stored procedures, in order to make the database more secure. Using views would limit the visibility of different tables, Stored Procedures for additional functionality with Transactions and the possibility of Rollback if a transfer error occurred. A database is also a good way to avoid redundancy in the data, if maintained properly. With the above design, redundancy can be avoided completely.

Databases are optimised for data storage, so they are usually a lot faster. They can also handle multiple requests at the same time. All of this makes a database the ideal choice for the system, and it should be added in the future.

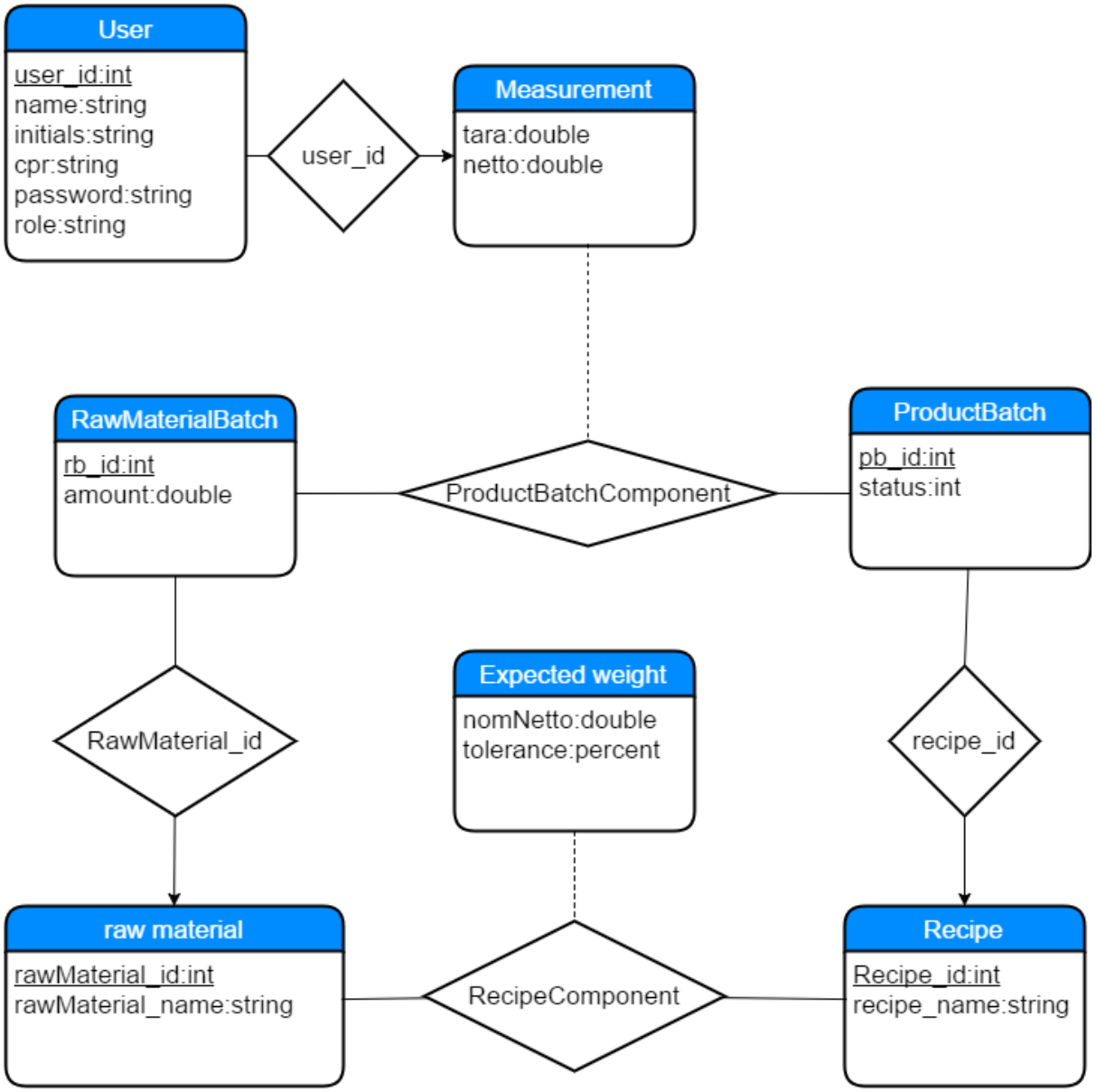


Figure 13: ER diagram

15 Conclusion

In this project a software system has been created to aid a pharmaceutical company in their daily routine. The company wanted to have these routines implemented in a software solution. One requirement was that the routines should change as little as possible from the old routines. The software has been developed with this in mind, and the routines have been altered as little as possible.

Every role in the company has been granted access to the proper features on the website. They have separate accounts, that they can login to, and gain access to all the information they need depending on their role. The website is primarily designed for the Chrome browser. However, the website should run on most modern web browsers.

In order to facilitate this website, the team had to develop a web server. This web server is responsible for storing data about the company. It is also responsible for ensuring that the data that is stored is correct, and does not conflict with any rules set by the company.

The company also use a number of industrial weights for their production. They wanted these to be a part of the software solution. In order to do this, a system (ASE) had to be developed. The system had to connect to all the industrial weights in the company, and control them. This way information could be stored automatically on the server, reducing some of the work time for the employees. The ASE-system is built to control multiple weights simultaneously, without any significant wait times. No more than 0,5 seconds delay you be expected on average.

All of the above mentioned system parts have been put together into one software solution that can be run on a server inside the company. This allows for easy access to weight connections, while also allowing employees from outside the network to connect to the web server.

The system has been tested on an Apache Tomcat server v. 8.5.11, which is easily available online. The system parts have been tested throughout, and no system crashing bugs was found. The system contains some bugs, that should be dealt with in future releases.

The system has been successfully created, and is ready for initial testing within the company. The system however lacks some major features like an external database and security features. The system can run without these, but these components will provide much better usability for the customer.

Overall the prioritised parts of the project has been implemented and are working as expected.

16 References

16.1 Books:

[1] Lewis & Loftus, *Java Software Solutions - Foundations of Program Design*, Addison-Wesley, USA, 7th edition, ISBN 10: 0-13-214918-4, 2012.

16.2 Webpages:

[2] Samual Santos, Samaxes, *"Combine and minimize JavaScript and CSS files for faster loading"*, uploaded May 2009, used June 2017, https://goo.gl/UT9YVH.

[3] M. Otto & J. Thornton, *"Boostrap 4"*, updated July 2016, used June 2017, https://goo.gl/hT9gAA.

[4] macek, *"jQuery library used in project"*, updated July 2012, used June 2017, https://goo.gl/hT9gAA.

[5] Librelist, *"Mustache - Template System"*, updated January 2017, used June 2017, http://mustache.github.io/.

Appendix

A Predefined data

Predefined users

Table 1 contains the data of all the predefined users, except for the super admin, which has access to everything. The super admin has the credentials id 'admin' and password 'root'.

ID	Navn	Initaler	CPR	Password	Rolle
6	Bent	BE	1111111118	null	Farmaceut
165202	Peter I EL-HABR	PE	0101800032	Peterpeter1	Admin
143233	Simon Engquist	SE	0101405109	Simonsimon1	Farmaceut
144265	Arvid Langso	AL	0101600203	Arvidarvid1	Laborant
165238	Mikkel Lund	ML	0101600203	Mikkelmikkel1	Farmaceut
93905	Jeppe Nielsen	JN	0101200159	Jeppejeppe1	Værkfører
165243	Mads Stege	MS	0101002918	Madsmads1	Værkfører

Table 1: Predefined user data.

Predefined Raw materials

Table 2 contains the data of all the predefined raw materials.

ID	Name	Supplier
1	Vand	Vand Co
2	Salt	Salt Co

Table 2: Predefined Raw materials.

Recipes

Table 3 contains the data of all the predefined recipes.

ID	Name
1	Salt vand

Table 3: Recipe.

Raw Material Batch

Table 4 contains the data of all the predefined raw material batches.

Batch ID	Raw material ID	Amount (kg)
1	1	8,6
2	2	10,7

Table 4: raw material batches.

Product Batches

Table 5 contains the data of all the predefined product batches.

Pb ID	status	recipe id
2	2	1
1	1	1
3	0	1

Table 5: Product Batches.

Product Batch Component

Table 6 contains the data of all the predefined product batch components.

Pb ID	Rb ID	Tara	Netto	Opr ID
1	1	0,1	0,151	165202
2	1	0,1	0,150	165202
2	1	0,1	0,149	165202

Table 6: Product Batch Component.

Recipe Component

Table 7 contains the data of all the predefined recipe components.

Recipe ID	RawMaterial ID	Nominielle Netto	tolerance
1	1	0,15	10
1	2	0,15	10

Table 7: Recipe Component

B Weight protocols

RM20

- To weight
  - 8, send a message that requires a user response.
  - 0, To stop any RM20 action on the weight.
- From weight
  - I, the message is understood and legal, but could not be preformed ATM.
  - L, the message is understood but does not have legal parameters
  - C, if the user presses cancel during user input.
  - A, contains a user input and is done with the RM20
  - B, RM20 is received, and another RM20 is coming after user input

P111

- To weight
  - send a message that will be shown on the weight
- From weight
  - L, message is understood but is too long
  - I, message is understood, but can not be preformed ATM.
  - A, message is preformed.

S

- To weight
  - send request for stable weight value
- From weight
  - S, Stable weight value done and sent in this message
  - I, message is understood, but can not be preformed ATM.
  - -, weight is underloaded
  - +, weight is overloaded

T

- To weight
  - send request for stable zero weight
- From weight
  - S, Stable zero the weight
  - I, message is understood, but can not be preformed ATM.
  - -, weight is underloaded
  - +, weight is overloaded

DW

- To weight
  - whipe the display for text
- From weight
  - A, done
  - I, message is understood, but can not be preformed ATM.

K

- To weight
  - 3, start K 3 mode
- From weight
  - A-, done

Q

- To weight
  - Quit weight K 3 mode

C ASE system test

1. **The system can connect to a weight.**
   Given a file containing the IP address and port number of a weight, can we expect the system to connect to said weight?

   Result: The system Successfully connects to the weight.

---

2. **Users can login to the weight**
   User that are registered in the system should be able to login with their id.

   Result: Success - All registered user we have tested can log in to the system.

---

3. **Only registered user can log in**
   User that are not registered in the system can not login.

   Result: Success - Entering unknown id gives a proper error message.

---

4. **User can enter product batches**
   Ensure that users can enter product batches, and that the system can find product batches that exist.

   Result: Success - All product batches that exist in the system gives the correct confirmation message. Unknown id are given an error message.

---

5. **User can enter raw material batches**
   Ensure that users can enter raw material batches, and that the system can find raw material batches that exist.

   Result: Success - All raw material batches that exist in the system gives the correct confirmation message. Unknown id are given an error message.

---

6. **User can measure products**
   Users can use the weight to measure materials that need to be produced. Also check if these are stored in the system.

   Result: Success - The user can measure materials and these are stored in the system.

---

7. **The system does not save invalid measurements.**
   The client has some rules for the measurement. It has to be close to a specific amount +- some tolerance. Therefore invalid measurements should not be saved in the system.

   Result: Success - Invalid measurements are not saved in the system. And the user is asked to repeat the measurement.

---

8. **Back Button**
   The user can use the back button to go back during a measurement, cancelling what he was doing.

   Result: Success - We tested the back button at all location, and it goes back to where we expect in all cases.

---

9. **Logout Button**
   The user can use the logout button to logout at any point after login in.

   Result: Success - The log out button logs the user out.

---

10. **Register id - negative test**
    Type in nothing in the RM20 fields.

    Result: failure - The weightcontroller crashes and the weight becomes unusable.

D Website user test

This test was conducted on the 14. of June 2017.

1. **Login as super admin**
   When you open the website the super admin username and password are already typed in. The tester just have to press login.

   Result: Success - The users logged in successfully.

   Comments:

   ---

2. **Logout**
   Ask the user to log off the website.

   Result: Success - The users logged out successfully.

   Comments: None

   ---

3. **Login as a normal user.**
   Ask the user to log in to the website using a specific id and password.

   Result: Success - The users logged in successfully.

   Comments:

   ---

4. **Edit personal information**
   Ask the user to change the personal information (not password) of their account.

   Result: Fail - The users are sent to the wrong side after editing their information.

   Comments:

   ---

5. **Edit password**
   Ask the user to change their password and use it to log back in.

   Result: Success - The users completed the task.

   Comments:

   ---

6. **Reset password**
   Ask the users to reset a password.

   Result: Partial Success - Some user completed the task.

   Comments: Sometimes a key is generated that is too short for the system.

   ---

7. **Go back to frontpage**
   Ask the users to go back to the front page. This should be done by clicking the top left corner.

   Result: Failure - Function not implemented.

   Comments: Sometimes a key is generated that is too short for the system.

---

8. **Find the menu that allows you to administrate users.**
   The user needs to find the button that leads them to page that shows them all the users in the system. This test should be done on a administrator account or super admin.

   Result: Success - The users found the page.

   Comments:

---

9. **Create a new user**
   Create a new user on website.

   Result: Partial success - The users successfully added the new user, but didn't see the pop that gave the the temporary login code.

   Comments: The pop up is shown at the top of the screen, but if the page is scrolled down, you won't see the pop up. User id is approved by the site but the user id already exist and therefore it gives an error.

---

10. **Edit user**
    Update an existing user.

    Result: Success - The users had no problem editing information.

    Comments: Initials can contains numbers and symbols. Fields that are locked will still be approved.

---

11. **Deactivate user**
    Deactivate an user.

    Result: Failure - This function is not implemented. **Comments:**

---

12. **Find the menu that display raw materials.**
    The user needs to find the button that displays the list of rawmaterials.

    Result: Success - The users found a way to show the raw materials.

    Comments: One user first tried to click the title which is not a link.

---

13. **Create a raw material**
    Create new raw materials in the system.

Result: Success - The users successfully completed the task.

Comments: Id is approved even tho its already taken.

---

14. **Edit raw materials**
    Update an existing raw material in the system.

    Result: Success - The users successfully completed the task.

    Comments:

---

15. **Find the menu that display raw materials batches.**
    The user needs to find the button that leads them to page that shows them raw material batches.

    Result: Success - The users found a way to show the raw material batches.

    Comments:

---

16. **Create a raw material batches**
    Create new raw materials batches on the website.

    Result: Success - The users successfully completed the task.

    Comments: Negative values are allowed for amounts.

---

17. **Edit raw materials batches**
    Update an existing raw material batch in the system.

    Result: Success - The users successfully completed the task.

    Comments:

---

18. **Find the menu that display recipes.**
    The user needs to find the button that leads them to page that shows them recipes.

    Result: Success - The users found a way to show the recipes.

    Comments:

---

19. **Create a recipe**
    Create new recipes on the website.

    Result: Success - The users successfully completed the task.

    Comments: It's unclear what recipes have already been taken. mangler enheder på netto og tolerance.

---

20. **Edit recipes**
    Update an existing recipe in the system.

    Result: Success - The users successfully completed the task.

    Comments: Recipes are not sorted after raw material id.

---

21. **Find the menu that display product batches.**
    The user needs to find the button that leads them to page that shows them product batches.

    Result: Success - The users found a way to show the product batches.

    Comments: Netto og tara mangler enheder.

---

22. **Create a product batch**
    Create new product batches on the website.

    Result: Success - The users successfully completed the task.

    Comments:

---

23. **Edit product batches**
    Update an existing product batche in the system.

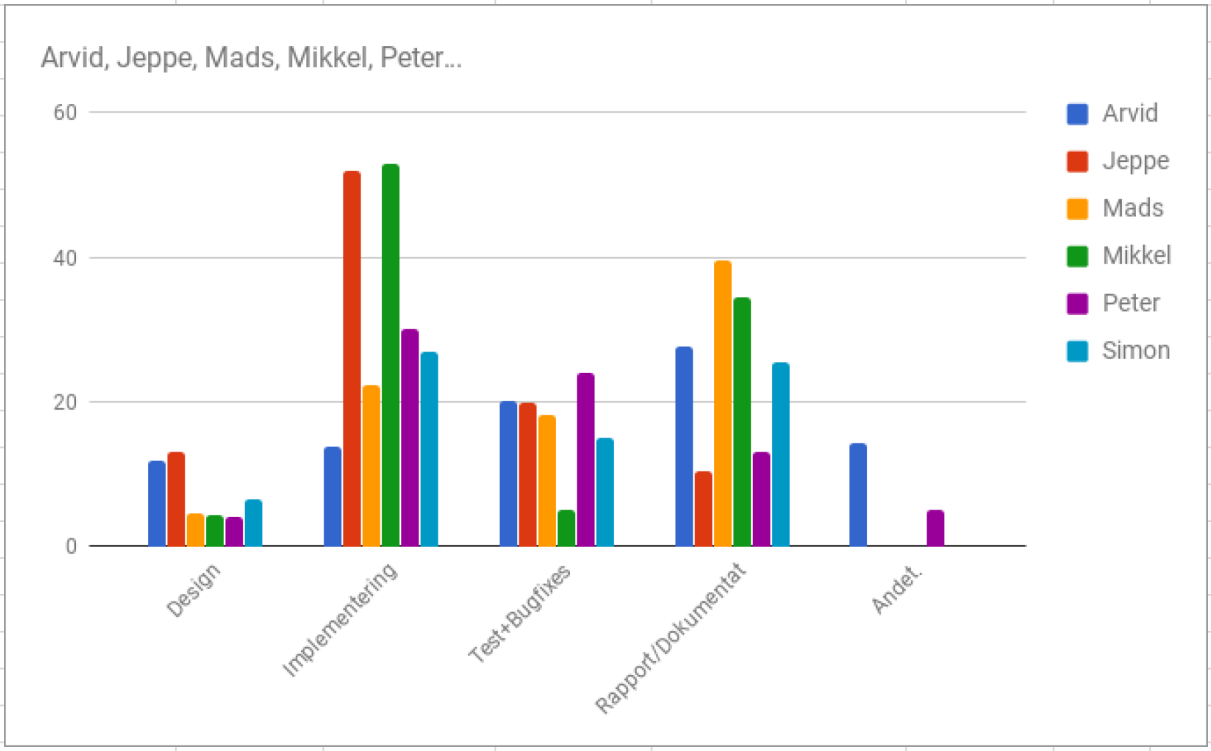
    Result: Success - The users successfully completed the task.

    Comments:

E Detailed time tables

RapportTidsplan - Total tidsplan

	Design	Implementering	Test+Bugfixes	Rapport/Dokumentation	Andet.	Total
Arvid	11,91633333	13,74666667	20,08333333	27,66666667	14,33333333	87,74633
Jeppe	13	52	20	10,5	0	95,5
Mads	4,5	22,25	18,25	39,49	0	84,49
Mikkel	4,25	53	5	34,5	0	96,75
Peter	4	30	24	13	5	76
Simon	6,5	27	15	25,5	0	74
Total:	44,16633333	197,9966667	102,3333333	150,6566667	19,33333333	514,4863



A full time table for every person can be found in the last appendix. This has detailed information on every developer.

RapportTidsplan - Arvid

	Design	Implementering	Test+Bugfixes	Rapport/Dokume	Andet.	Total	Kommentarer
Dato:	11,91633333	13,74666667	20,08333333	27,66666667	14,33333333	87,74633333	
01/06/2017	4					4	Lagde en plan for projektet og overordnet design.
01/06/2017		1,5				1,5	Opsatte projektet i ecllipse, og ryttede op i gamle filer
02/06/2017		0,25				0,25	Password check og cpr check funktion påbegyndt. I toget
02/06/2017		0,33				0,33	Færdigjorde overstående og lavede CPR generator.
02/06/2017			1			1	Test af overstående funktioner
02/06/2017					3,25	3,25	Hjalp andre med problemer.
02/06/2017		2,5				2,5	Begyndte på WeightController
05/06/2017	5,333					5,333	Udpensling af Design + M1
05/06/2017					0,5	0,5	Undersøg mulighed for opstart af projekt.
05/06/2017					0,25	0,25	Diskussion med anden gruppe om mulige løsninger.
05/06/2017	0,25	1				1,25	Hjemme/Tog: Design og reimplementering af WeightController
06/06/2017					2,083333333	2,083333333	Hjælp andre+ frokost
06/06/2017	0,1666666667	0,5				0,6666666667	tog/hjemme: Weightcontroller implementering og design
06/06/2017					0,5	0,5	Git problem fix
06/06/2017		1	0,75			1,75	Weight controller implementerin og test
06/06/2017					0,5833333333	0,5833333333	Samtale med finn..
06/06/2017		1			1	2	Omnavngiving af projektfiler
07/06/2017			0,4166666667			0,4166666667	Tog: test af weight control
07/06/2017		1,5	2,416666667			3,916666667	Implementering og test af weightcontrol.
07/06/2017	1,083333333			0,25		1,333333333	Yderligere design og rapport struktur.
07/06/2017					2	2	Hjalp andre med problemer + frokost.
08/06/2017			0,3333333333			0,3333333333	Test af weightcontroller (tog)
08/06			0,8333333333			0,8333333333	Test af weightController
08/06/2017				0,3333333333		0,3333333333	Gennemlæsning af krav
08/06/2017					0,8333333333	0,8333333333	Frokost
08/06/2017	0,5	3,166666667				3,666666667	Redesign af weightkontrol grundet strammere krav.
09/06		0,4166666667	0,5			0,9166666667	Measurement Controller + test.
09/06/2017			2,166666667			2,166666667	WeightController test
09/06/2017			2,75			2,75	Weightcommunicator test med vægt og andre test.
09/06/2017					1,166666667	1,166666667	Frokost
12/06/2017		0,5833333333	6,916666667			7,5	Test og implementering af ASE system
12/06/2017					0,25	0,25	Gennemlæst og færdiggjort M2
13/06/2017			1,333333333		1,166666667	2,5	Weight COntrol test + stig gennemgang.
13/06/2017					0,75	0,75	Frokost
13/06/2017	0,5833333333					0,5833333333	Design af comparator

13/06/2017				4,166666667		4,166666667	Rapport - Requirments og Introduction
14/06/2017			0,6666666667	7,416666667		8,083333333	Hjemme: Bruger test med en bruger ukendt af system. + rapport: Design, Implementering, Test.
15/06/2017				13,5		13,5	Dokumentation
16/06/2017				2		2	

RapportTidsplan - Jeppe

	Design	Implementering	Test+Bugfixes	Rapport/Dokume	Andet.	Total	Kommentarer
Dato:	13	52	20	10,5	0	95,5	
01/06/2017	5					5	Planlægning af design af web app
02/06/2017	4					4	Design af web app og lavede build.xml til Ant build (samling og minificering af javascripts/css)
02/06/2017		3				3	Oprettede HTML sider til råvarebatch og produktbatch HTML
05/06/2017		4				4	REST og javascript til login siden
05/06/2017			2			2	Oprettelse af test stubbe controllers, til at teste login-siden
05/06/2017			2			2	Opsætning af mustache templates og test af dette (hjemme)
06/06/2017		4				4	Opdatering af user rest og login rest og deres tilhørende javascript
06/06/2017			3,5			3,5	Test af login og user rest og deres javascript
06/06/2017			1			1	jQuery ajax fix for funktioner (hjemme)
07/06/2017		2	4			6	Arbejdede videre på user javascript og rest og tests af dette, samt startede på råvarebatch javascript
07/06/2017	4					4	Design af web app (Photoshop -> HTML & CSS) (hjemme)
08/06/2017		7				7	Fixes fra igår + productbatch og råvarebatch javascript/rest
08/06/2017			0,5			0,5	Fixede recept templates og javascript (hjemme)
09/06/2017		6				6	Færdiggjorde produktbatch og råvarebatch templates + javascript
10/06/2017		2,5				2,5	Lavede function og template til at vise en besked når der gemmes data i datalaget. Funktionen er implementeret i råvarebatch. (hjemme)
10/06/2017			1,5			1,5	Test af ovenstående funktion og besked-template. (hjemme)
11/06/2017		1				1	Implementerede jquery validation plugin. (hjemme)
12/06/2017		6				6	Design af login side, samt bug fixes i javascript filerne
12/06/2017			1,5			1,5	Kommentarer og implementering af den generelle "rest besked" funktion, samt testing af dettet (hjemme)
13/06/2017		7,5				7,5	Implementering af admin reset password og bug fixes
14/06/2017		7				7	Implementering af bruger ændring af password og validering af html forms
14/06/2017		2				2	Implementering af de resterende form valideringer og implementering af select felter til html templates (hjemme)
15/06/2017			4	3		7	Sidste bug fixes i javascript filer. Rapport skrivning 9-16
15/06/2017				3		3	Rapport skrivning 16-19

15/06/2017				4		4	Rapport skrivning 20-24
16/06/2017				0,5		0,5	Rapport rettelser 30min (hjemme)

RapportTidsplan - Mikkel

	Design	Implementering	Test+Bugfixes	Rapport/Dokume	Andet.	Total	Kommentarer
Dato:	4,25	53	5	34,5	0	96,75	
1/6/2017	2	0	0	0	0	2	Design og planlægning af projekt
1/6	0	3,75	0	0	0	3,75	HTML design
2/6/2017	0	7	0	0	0	7	Jeg har lavet html sider til rawMaterialAdministration, recipeAdministration og userAdministration.
5/6/2017	2	0	0	0	0	2	M1 - webside design diagram og diskussion
5/6/2017	0	3	0	0	0	3	Diskussion af hvordan websiderne skulle implementeres
5/6/2017	0	2	0	0	0	2	Implementering af start login og rawMaterialCRUD
6/6/2017	0	3	0	0	0	3	RawMaterialCRUD og RawMaterialBatchCRUD
6/6/2017	0	3	0	0	0	3	Oprydning og korrektion af alle CRUDS pånær UserCRUD.
6/6/2017	0	0	1,5	0	0	1,5	Renaming Dansk til engelsk. Rettelser og lignende.
7/6/2017	0,25	0	0	0	0	0,25	design og planlægning af rapport.
7/6/2017	0	6,75	0	0	0	6,75	Raw material javascript og design heraf.
8/6/2017	0	7,5	0	0	0	7,5	Alt omkring recipe. + ændringer fra gårdsdagens.
9/6/2017	0	7,5	0	0	0	7,5	Videre med recipe.
12/6/2017	0	7,5	0	0	0	7,5	Skrevet TODO liste til webdel, samt færdiggjort recipe. Snakket webdesign mv.
13/6/2017	0	0	0	3	0	3	Rapport skrivning, rettelser og introduktion
13/6/2017	0	2	0	0	0	2	Lavet sidste med Recipe plus rettelser.
13/6/2017	0	0	3,5	0	0	3,5	Kommentering og rettelser af CRUD klasser.
14/6/2017	0	0	0	8	0	8	Retning af introduktion, user guide og tildels requirements. Hjælp med appendix og andre ting.
15/7/2017	0	0	0	16	0	16	lidt for grunding retning af rapport mm.
16/7/2017	0	0	0	7,5	0	7,5	Mere af ovenstående.

RapportTidsplan - Mads

	Design	Implementering	Test+Bugfixes	Rapport/Dokume	Andet.	Total	Kommentarer
Dato:	4,5	22,25	18,25	39,49	0	84,49	
01/06-17	3	1,5	0	0	0	4,5	Diskussion omkring design og struktur, samt opsætning af ASE.
02/06-17		7				7	Implementering af ConnectionReader til ASE, samt opstart på ConnectionManager.
04/06-17		1				1	Implementerede mindre getmethoder og arbejdede på ConnectionManager (hjemme).
05/06-17	1,5					1,5	Omstrukturering af ASE model.
05/06-17		2,25				2,25	Endelig implementering af ConnectionReader, samt dokumentation af ConnectionReader
05/06-17		1,5	1,25	1,25		4	Videre arbejde med ConnectionManager, samt JUnit tests af ConnectionReader. Fortsætter imorgen.
06/06-17		3				3	Implementation af nye funktioner og metoder i ConnectionManager og ConnectionReader.
06/06-17		3				3	Tests på ConnectionManager og ConnectionReader med nye funktioner.
06/06-17		1				1	Videre arbejde med Manager og Reader.
07/06-17			5,5			5,5	Arbejde med tests af ConnectionManager. Fejl fundet, og rettet. Nye fejlbeskeder og håndtering implementeret.
07/06-17				1		1	Opsætning og diskussion af rapportindhold.
08-06/17		1	0,5			1,5	Korrigering af ConnectionManager, nye tests og tilføjelse af Java Doc til WeightCommunicator.
08/06-17				5,5		5,5	Rapportskrivning af Introduction, Problem Description, Requirements, User Roles og rapport opsætning.
09/06-17				1,16		1,16	Rapport - Optimisation færdiggjort.
09/06-17				1,33		1,33	Indtegning af Workflow diagram for system.
09/06-17			4			4	Gennemlæsning af JUnit tests, og test af vægt.
12/06-17			3	3½		3	Rapport og test af vægt/gennemcheck af klasser.
12/06-17		1				1	Færdiggørrelse af M2 (hjemme)
13/06-17				7,5		7,5	Rapportskrivning, fejlrettelser og redigering.

14/06-17				6		6	Rapportskrivning, korrekturlæsning og dokumentation
14/06-17			1			1	Omstrukturering af ConnectionReader og tests. ConnectionManager kræver gensyn
14/06-17			1,5			1,5	Brugertest (Hjemme)
15/06-17			1,5			1,5	Indskrivning og diskussion af User tests
15/06-17				10,5		10,5	Rapportskrivning og fejlrettelser.
16/06-17				3		3	Fejlrettelser
16/06-17				2,25		2,25	Endelige gennemlæsninger

RapportTidsplan - Peter

	Design	Implementering	Test+Bugfixes	Rapport/Dokume	Andet.	Total	Kommentarer
Dato:	4	30	24	13	5	76	
01-06-2017	4	1,5	0	0	0	5,5	Planing & FileManagement class implemented
02-06-2017		6	2			8	Login and User system implemented & tested (changing to local static HashTable Data for all DAO)
04-06-2017		2				2	Data Layer : ProductBatchDAO, ProductBatchCompDAO, RawMaterialBatchDAO and RawMaterialDAO implemented
05-06-2017		3	1		1	5	Data layer finished, bugfixes, global updates. Other : Help
06-06-2017		6	1		1	8	Controller layer (& interfaces) implemented, critical bugfixes, help others and Home : Bugfix
07-06-2017		2	3		1	6	Initializer and DTO abstract class implemented. Controllers DAO and DTO updated. Massive Bugfixes with tests. Help others and planning the report.
08-06-2017		4	2		1	7	FileManagement works, all the Data is now saved on files. Big updates on all DTO, DAO and Controllers. Critical Bug Fixes & tests. Other (also at home) : help + adding comments and Java docs
09-06-2017		2	4			6	Choosing and implementing Encryption solution. More tests, updates and BugFixes over Controller and Data layers.
12-06-2017		1	3		1	5	Help others with some bugs in the system and implementation of the hashed password. Home : Reading about Iterrors and ConcurrentModificationException
13-06-2017		1,5	4	2		7,5	Implementation of DTO sorting to ArrayList, User tests & Critical BugFixes and Report.
14-06-2017		1	1	5		7	Small priority updates, bugfixes and writing report
15-06-2017			3	6		9	BugFixes and report

RapportTidsplan - Simon

	Design	Implementering	Test+Bugfixes	Rapport/Dokume	Andet.	Total	Kommentarer
Dato:	6,5	27	15	25,5	0	74	
01/06-2017	4,5	1	0	0	0	5,5	design, og kort implementering
02/06-2017	2	5				7	Design af ASE og Implementering af Measurement controller + test
06/06-2017		2	1			3	Implementering af CRUD's + lidt af hver med ASE.
07/06-2017		5				5	Færdig gjorde measurement controller og testede
07/06-2017		2				2	Påbegyndte ConnectionManeger
08/06-2017		7				7	arbejdede vidre på ConnectionManeger
09/06-2017		5	2			7	Implement and test weightcom
12/06-2017			4			4	fix and test weightcom
12/06-2017			3			3	Test af det fulde ASE med vægt, ft Arvid, med fixes.
13/06-2017			4			4	test, fremvisning af system med vægt
13/06-2017				3		3	protocoler, og ase sequence diagram
14/06-2017				7		7	Rapport
14/06-2017			1	1		2	test rettelser og rapport skrivning
15/06-2017				6		6	9-15
15/06-2017				2		2	16-18
15/06-2017				6,5		6,5	19-02:00