SDU
University of
Southern Denmark

Project Report

University of Southern Denmark

Faculty of Engineering

Maersk Mc-Kinney Moller Institute

3rd Semester project

Course code - ST3-PRO

Project period: September 1st - December 30th

Group ST01

Mathias Jeppesen Engmark	maeng20@student.sdu.dk
Oliver Heine	olhei20@student.sdu.dk
Anton Valdemar Dahlin Irvold	anirv20@student.sdu.dk
Nicklas Bruun Jensen	nickj20@student.sdu.dk
Christoffer Schurmann Krath	chkra19@student.sdu.dk
Kasper Stokholm	kasto16@student.sdu.dk

Supervisor

Eun-Young Kang	eyk@mmmi.sdu.dk

II Title page

University of Southern Denmark

Faculty of Engineering

Maersk Mc-Kinney Moller Institute

3rd Semester project software

Course code - ST3-PRO

Project period: September 1st - December 30th

Authors

Mathias J. Engmark _____

Oliver Heine _____

Anton V. D. Irvold _____

Nicklas B. Jensen _____

Christoffer S. Krath _____

Kasper Stokholm _____

III Summary

At the start of the project a case from Refslevbæk Bryghus A/S was received addressing an issue they encountered and wished to be solved. The issue was the expansion of their production line and their current situation with the production of beer relying on manual oversight. Therefore, Refslevbæk Bryghus A/S bought a new and optimized beer production machine, though they still need a software system that can connect to the beer machine and add further functionalities for automation. Based on the issue at hand the following problem definition was written; "*This project aims to develop a prototype system to control, maintain and optimize a physical simulation of a beer brewing machine.*" The problem definition was further specified in Chapter 1.

The project's structure and planning was controlled through Scrum where Jira was used as a tool to visualize the planning of the project. Requirements specifying the systems functionality was further developed and prioritized through the MoSCoW method. An analysis of the detailed requirements for the critical use-cases was developed in order to understand the requirements at a deeper level. The critical use-cases were also used to make an analysis class diagram describing the system's structure and the relationships between classes for those particular use-cases.

Tests were conducted in order to verify and validate the developed project in relation to the developed requirements and the implementation of said requirements. The tests showed that the implementation of the system's must-have requirements were successfully implemented though improvements are still needed.

IV Preface

This report is written as part of a 10 ECTS course at the Maersk Mc-Kinney Moller Institute in conjunction with the 3rd semester for Software Technology.
The report is written as documentation of a proposed solution to Refslevbæk Bryghus A/S´ production issue. The report describes the work towards the proposed solution carried out by the project group.

The report is written in a way that assumes the reader has at least the same knowledge as the authors have obtained throughout the third and previous semesters. The audience is mainly intended to be the supervisor and censor due to the project being a semester project that will be examined.

The project group would like to thank Eun-Young Kang for excellent and competent supervision throughout the project which has raised the professional aspect in relation to the development of the project. Furthermore, B&R shall receive great thanks for making the physical beer machine available for testing purposes. Lastly, every test person who helped with the validation & verification chapter should receive a great thanks.

V Table of contents

VI List of figures

This report is structured chronologically and therefore, is meant to be read as such. Though it might be possible to read abstractions and still be able to understand the subject. It is expected that the reader has basic knowledge of the programming languages Java, PHP and mySQL and lastly the physical machine as well as the machine communication protocol OPC UA.

Chapter 1 - Introduction, gives an overall background of the project and specifies the problem definition and stakeholders.

Chapter 2 - Theory & methods, includes a description of how the project is planned and the methods & technologies used throughout the project.

Chapter 3 - Requirements specification, defines the requirements needed based on the stakeholders' wishes and the requirements formed by the project group. Furthermore, the chapter defines two detailed use-cases.

Chapter 4 - Analysis, identifies the structure of and the relationships between classes in the backend for the chosen use-cases.

Chapter 5 - Architecture, specifies the system's architectural structure and usages of certain technologies.

Chapter 6 - Design, explains the design choices for each layer.

Chapter 7 - Implementation, describes and explains the implementation for key use-cases.

Chapter 8 - Validation & verification - Validates and verifies the functionality of the developed system through unit, use-case and user tests as well as a performance test for the web-application

Chapter 9 - Discussion, discusses the results from chapter 8 in relation to the problem definition

Chapter 10 - Conclusion, concludes on the project and its results in relation to the problem definition

Chapter 11 - Perspectives, takes the reader through a journey on what could have been done differently or might be added to the system in a later development phase

VII Editorial

This report is written as documentation of the project and its results where each chapter is divided between group members. The table below shows how each chapter is divided between members where the percentages column represents the workload each author has contributed in relation to that specific chapter. The percentages in the table below do not necessarily correlate with the workload during the project in regards to development.

Chapter	**Written by**
Summary	Oliver (100%)
Preface	Oliver (100%)
Introduction	Whole group (16.6% each)
Theory & methods	Anton (40%), Christoffer (20%), Oliver (20%), Mathias (20%)
Requirements specification	Whole group (16.6% each)
Analysis	Kasper (50%), Oliver (50%)
Architecture	Anton (100%)
Design	Kasper (28.6%), Oliver (16.6%), Nicklas (16.6%), Anton (16.6%), Christoffer (16.6%), Mathias (5%)
Implementation	Whole group (16.6%)
Validation & verification	Anton (40%), Nicklas (40%), Mathias (20%)
Discussion	Christoffer (30%), Nicklas (30%), Oliver (25%), Anton (15%)
Conclusion	Kasper (60%), Oliver (30%), Mathias(10%)
Perspectives	Kasper (100%)
Video	Christoffer (50%), Mathias (50%)

Table 1 Editorial for project contents

Chapter 1

Introduction

The brewing company Refslevbæk Bryghus A/S needs an optimized production line for their beer brewing industry, consisting of two parts: hardware and software. B&R is producing the hardware to Refslevbæk Bryghus A/S but to use the hardware in combination with the company some new software is required. This project will provide a prototype solution to how the software could be developed.

1.1 Background & motivation

The B&R company needs a software system (Manufacturing Execution System) to control and manage the beer production machine for the brewing company Refslevbæk Bryghus A/S, in an efficient manner that takes environmental variables such as temperature, humidity, and vibrations into account. The system should also use an algorithm to optimize the beer production process based on collected data.
The group will complete this task with a web application that interacts with the hardware simulation through an OPC-UA connection, uses regression and other mathematical methods to optimize the process, and dynamically responds to the environmental variables to minimize the discarded products.
This system will minimize the workload for the production manager at the brewery because it eases the maintenance and refilling process, optimizes the process based on previous data, and it automatically reacts and adapts so the beer production machine produces more beer faster and with less waste and manual oversight.
The changes will be a step up from the current process which takes place manually on the machine or through a basic web UI that only acts as a remote interface to the physical buttons on the machine.

1.2 Problem definition

This project aims to develop a prototype system to control, maintain and optimize a physical simulation of a beer brewing machine.
A system with the purpose of easing the manual workload for the production manager of the beer brewing hardware, and providing further value to the system through optimization of the production

- A web application made with a JavaScript frontend as a user interface to the system hardware
- A web application with a Java backend and communication with the hardware through the OPC-UA protocol
- A deployed web application, e.g., using Docker and network technologies

1.2.1 Stakeholders

Refslevbæk Bryghus A/S is the company for which B&R is producing a beer brewing machine. The company needs to optimize its beer production process as its current workflow cannot keep up with the demand.
B&R is producing the hardware for a beer brewing machine for Refslevbæk Bryghus A/S that can communicate with other systems to automate the process and keep the cost down for Refslevbæk Bryghus A/S.
The internal product owner is the project group member decided by the group who has the final decision on the prioritization of the project requirements. The product owner is a role defined by the agile work method Scrum.
Since there is no direct contact with the stakeholders during the project, the group is forced to self-analyze decisions and find relevant people for software testing.

1.2.2 Ideas for solutions

Ideas that can be used to narrow how the aim and objectives will get implemented. This is the group's initial thought process before starting the development process, in order to narrow down the scope of the problem.

- To use a Raspberry Pi as a server to host the system simulation and web application
- Web-application to act as an interface between the manager of the beer brewing machine and the beer brewing machine
- The system should optimize the various kinds of beer production based on mathematical principles and algorithms

- The system should provide graphs showing the beer production over time when using the different tuning mechanisms
  - Efficiency
  - Resource usages
  - Production time per bottle of beer
  - Temperature, vibrations, and humidity
- The system should provide a dynamic solution to problems in the machine's environment to prevent unnecessary discard of beer
- The brewing machine's sensors should constantly measure vibrations, temperature, and humidity and if any of these parameters show abnormal values the brewing machine should abort.
- A database to provide login credentials and stored information about the beer production system
- The ability to show previous batches on the web application based on stored beer production data
- A queue system for queuing more production batches with equal or different production parameters such as type of beer, production speed, and amount to produce. This will allow the brewing machine to produce continuously and minimize downtime.

Chapter 2

Theory & methods

This chapter touches on some key aspects of the earlier parts of the project such as planning, project structuring and different technologies used throughout the process.

2.1 Planning of project

For the planning of the project, the group decided to follow a Scrum workflow with Jira as the preferred tool to plan out sprints. The project was divided into three separate sprints each spanning 3 weeks. During each of the group's work sessions, a quick Scrum meeting was held with the chosen Scrum master following up on previous tasks and future work to be done. Issues in the backlog were divided evenly among the group members to get a collective effort and to prevent a single group member from doing a majority of the work.

2.2 Methods & technologies

The project has used a variety of technologies and methods which relate to each course from the third semester. The used technologies will be described and relation to their course.

2.2.1 Web technologies

In the Webtech course, the group was introduced to a development framework called Laravel this framework is used for the frontend part of the software for the project. The Laravel framework is built with the model-view-controller pattern in mind and is based on core PHP functionality. Although it has to be said, Laravel is a frontend and a backend framework that makes development much easier, however, the group gained the knowledge of the Eclipse Milo library; an open-source implementation of OPC UA[1]

which can be used to connect to the physical machine. Therefore, the group chose to develop the backend through Java and the frontend through Laravel.
One benefit to using Laravel is the native PHP template engine extension *Blade*[2] that allows the programmer to use the Blade-notation which is designed with hierarchical blocks and layouts with predefined blocks instead of writing *"regular"* PHP. The group uses AJAX to update parts of an HTML page without reloading all the content. These technologies combined make up the web technology aspect of the project.

2.2.2 Calculus and Linear Algebra

The group has gathered data from the actual hardware simulation to calculate the optimal speed for each beer type. The group used regression to determine the produced *good count per minute* as a function of the speed, $f(x)$. This function was differentiated, optimized and maximized using methods from the Calculus and Linear Algebra class (as well as the 1st-semester class Statistical Data Analysis) to determine the optimal speed. The calculations for this can be seen in AppendixI.

2.2.3 Industrial cyber-physical systems

The system is the beginning of an MES system, that can be further developed into a full functioning MES system. It has MES-like functionality such as calculating the OEE and storing the performance in reports in the database, providing production orders with manual prioritization or a queue-like system, batch creation, and easier execution of the system, measures production costs and optimizes the variables to improve the efficiency of the system. The system also uses the Eclipse Milo Java library, which is an implementation of the OPC-UA protocol which is considered an Industrial Internet of Things protocol, to make inter-machine-operability easier.

2.2.4 Operating systems and distributed systems

As this system consists of a few different services, it would benefit greatly from being containerized for easier deployment. This project did not focus a lot on the deployment of the system, so the containerization part will not be greatly valued. However, the MySQL database along with a Phpmyadmin installation will be containerized, so no local installation of the MySQL database is necessary. This means the developer of the system just runs the docker-compose file in the repository and a proper database is set up and exposed on port 3306.

2.3 Related research

As mentioned earlier in Section 2.2 this project will handle a frontend developed with Laravel and a backend in Java that communicates with the physical machine. In order to create a connection between the frontend and backend, Spring Boot was chosen. Spring Boot is an open-source Java-based framework that provides the possibilities to create and manage REST endpoints[3]. Spring Boot in this project is used the following way:

- To create and manage the database through auto-configuration
- To provide a connection between frontend & backend

The use of Spring Boot will be further described in Section 6.2.

In order to measure a websites' performance, a variety of tools can be used. One of the more popular ones is Google's Lighthouse tool, which provides insight into how a website can be optimized further. Some of the aspects the Lighthouse tool analyzes are loading time, interactivity, and visual stability. These three aspects in particular determine the general performance of any website. The use of the tool is explained more in-depth in Section 8.4.

2.4 Concept definition

- Manager
  - The person who is in charge and control the beer production
- Hardware simulation
  - The simulation with the physical beer brewing machine provided by B&R
- Software simulation
  - The virtual simulation of the brewing machine provided by B&R

Chapter 3

Requirements specification

The requirements specification is used to get a clear picture of the project, its requirements, constraints and architectural shape. The problem definition and ideas for solution in a combination with the semester-info have shaped the outcome of the requirements specification.

3.1 Requirements

This section will present the functional requirements as well as the non-functional requirements.

3.1.1 Functional requirements

The following table defines the systems requirements. The functionality of the system. Each requirement has been prioritized in accordance with MoSCoW.

ID	**Name**	**Description**	**Prioritization (MoSCoW)**
U01	Set a production batch	A user can set a production batch with the following parameters: • Amount • Speed • Type The system will allocate a unique batch ID	M
U02	Start the machine	A user can start the machine manually and begin producing a batch	M
U03	Stop the machine	A user can stop the machine manually and stop production of batch defined in U01	M

U04	Reset the machine	A user can reset the machine to ready it for production. After resetting the machine will enter an idle state of 4.	M
U05	Do maintenance	When maintenance is required, it should be possible to activate the process through the systems interface. This process could be started manually or done automatically	M
U06	Refill the ingredients	When ingredients are missing, it should be possible to activate the refilling process through the systems interface. This process could be started manually or done automatically	M
U07	Produce batch report	• Batch ID, product type, production speed and OEE • Number of products (total, defect and acceptable) • Amount of time used in the different states • Logging of temperature, humidity and vibrations over the production time	M
U08	Show live data with visualization for current batch	Graphs updating dynamically based on variables such as temperature and vibration	M
U09	Calculate Overall Equipment Effectiveness, OEE	Following the OEE formula to calculate the systems overall effectiveness the system should be able to display said effectiveness for the current batch	S
U10	Calculate optimal production speed and batch amount	Given the desired batch requirements in U01, the system should calculate the optimal speed for production to increase effectiveness and reduce errors. This refers to U08 and U09	S
U11	Emergency-stop	When a set of parameters are met the system must issue an emergency stop	S
U12	Export batch report data to a file	Refer to U06	C

U13	Dynamic response to environment	The machine should stop the batch and update the parameters to meet the updated environment variables	C
U14	User credentials	A user should input user credentials before accessing the interface	C

Table 3.1 Use-cases with prioritization

Requirements that will not be implemented

The functional requirements *U05 Do maintenance* and *U06 Refill the ingredients* were later on in the implementation process identified to be impossible to implement as the beer machine simulation has certain read and write restrictions that prevent this. They have been kept in the diagram to show the thought process of the group.

3.1.2 Non-functional requirements

The following table defines a set of requirements the system's functionality must uphold.

ID	Name	Type	Description	Prioritization (MoSCoW)
NF01	Response time	Performance	The web server should have a response time of 5 seconds using the Google Chrome Lighthouse extension to estimate	M
NF02	Manufacturing productivity	Performance	The manufacturing productivity of the brewing machine system should achieve an OEE score of at least 70%	M
NF03	Emergency Stop	Safety	When an emergency-stop is issued the system must shut down in accordance with environmental & staff safety	S
NF04	User-friendly graphical user interface (GUI)	Usability	The system should have a user-friendly and intuitive GUI. The GUI should minimize the amount of user errors when using the system. The user should also be able to efficiently navigate the GUI with as few inputs as possible	S
NF05	User credentials	Security	The system should only be accessible to authorized personnel	S

Table 3.2 Non-functional requirements with prioritization

3.2 Detailed requirements

In order to understand the identified requirements at a deeper level a detailed use-case description was developed on use-case U01 & U02 shown in Table 3.3 & 3.4. Use-case U01 & U02 will also be used throughout the report as examples as both are seen as critical use-cases based on their prioritizing in Table 3.1.

Use-case: Create production batch
ID: U01
Primary actors: Manager
Brief description: The manager can set a production batch with the following parameters: Amount, Type & Speed. The system will allocate a unique batch ID.
Main flow: 1. The use-case starts when the actor wishes to set a production batch 2. The actor presses the “Make new batch" button 3. The system sets a predefined speed based on the calculated error function «include U10» 4. The actor sets the following parameters: 4.1. Amount 4.2. Type 4.3. Speed 5. The actor presses the button “submit” 6. The system generates a unique batch id for the current batch 7. The system saves the batch in the system 8. The system creates a unique batch report for the new batch «includes 07» 9. The system returns the actor the batch page
Alternative flow: • The actor can cancel the use-case between 2-7. – If the use-case is canceled between 2-7 nothing new will be saved to the system
Post conditions: • The batch has been saved in the system • A unique batch report is saved to the system
Cross References: • U07 • U10

Table 3.3 Detailed Use-case - Create production batch

Use-case: Start production batch
ID: U02
Primary actors: Manager
Brief description: The manager can start the machine manually and begin production of the chosen batch.

Pre-conditions:
- The beer brewing machine must be in the idle state (State 4)
- There must be at least one batch in the system

Main flow:
1. The use-case starts when the actor wishes to start a production batch
2. The actor can press the button “Start batch” on the "Home" page
   2.1. The system will then start the next batch from the database table
   OR
3. The actor can press the button "batches"
   3.1. The system will show a list of all batches
   3.2. The actor can now choose a specific batch from the list
   3.3. The system will start the chosen batch
   3.4. The system returns the actor back to the "Home" page
4. The system starts the live data «includes U08»

Alternative flow:
- The actor can cancel the use-case at point 2 or between 3 and 3.3.

Post conditions:
- The system has started a production batch

Cross References:
- U08

Table 3.4 Detailed Use-case - Start Production Batch

Chapter 4

Analysis

The static analysis is used to identify classes and structure for the systems backend through a noun-analysis of the detailed use-cases described in Section 3.2. The identification will lead to the development of an analysis class diagram which contains the classes and relationship between them to obtain the functionality for the chosen use-cases.

The dynamic analysis will develop a use-case realization based on the use-cases from Section 3.2. The use-case realization will cover the development of a system operations contract describing and identifying the systems operations which will lead to a sequence diagram explaining the interaction between the user and the system. Finally an updated analysis class diagram based on the one from Section 4.1 with relevant methods and attributes.

4.1 Static analysis

The identification will be made on the detailed use-case U01 & U02 where all nouns will be marked with red, though, if the same word is repeated multiple times it will not be marked again. Both noun analyses are included in the appendix Section A Table 1 & 2.

Based on the noun analysis every word was considered as a possible class or attribute which led to the final development of an analysis class diagram showing the relationship and structure between classes in relation to use-case U01 and U02. Though, some attributes included in the diagram were requested by the stakeholders described in Section 1.2.1. The analysis class diagram can be seen in Figure 4.1.

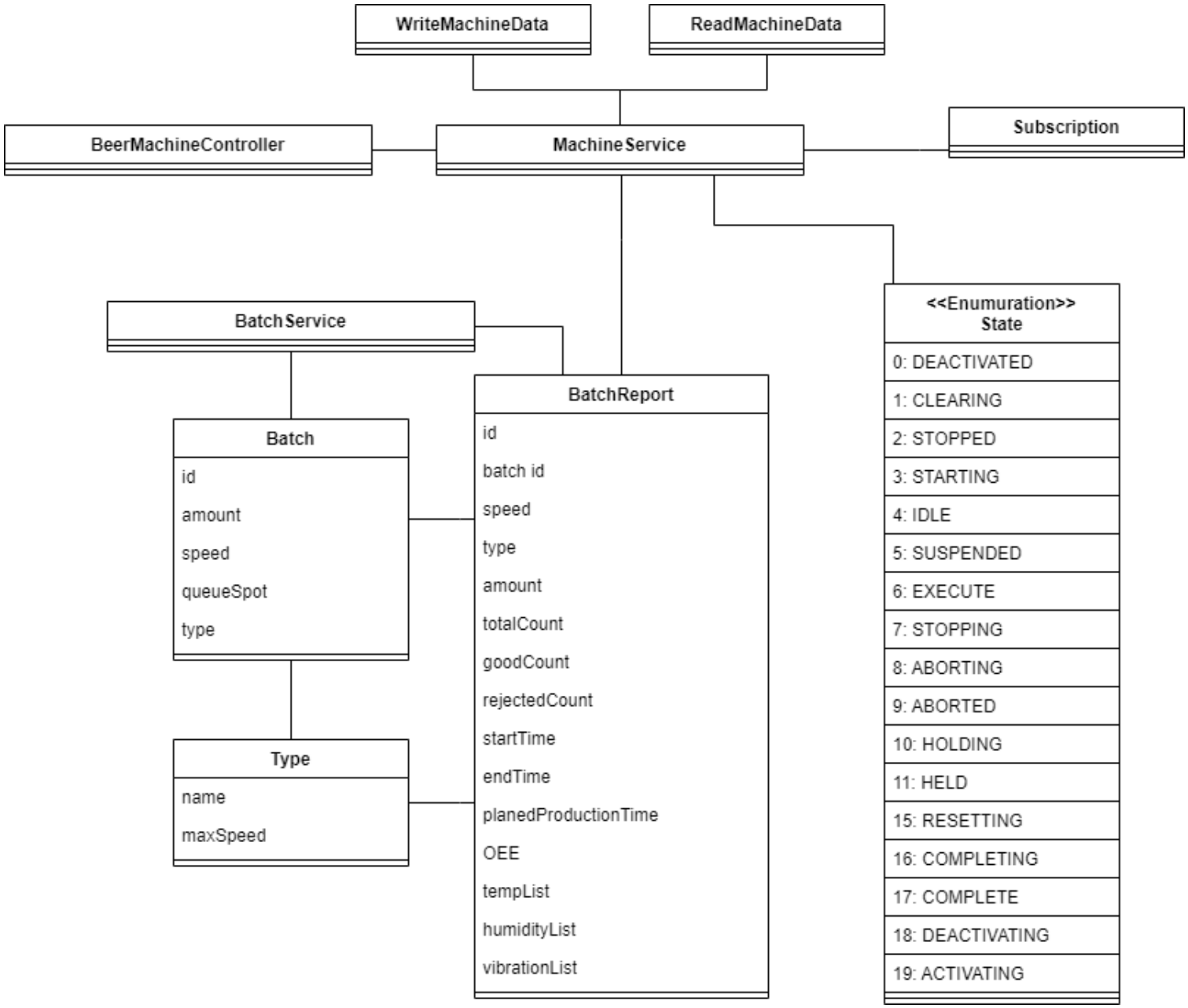

Figure 4.1 Analysis Class Diagram - First edition based on the noun-analysis

4.2 Dynamic analysis

The use-case realization will be made on the detailed use-case U01 where the use-case realization for U02 can be seen in Appendix B Table 3 & Figure 1. The use-case realization will be conducted using a *happy day* scenario.

System operations contract
Table 4.1 shows the system operations contract for use-case U01. The contract includes the three main responsibilities.

Contract	
Operation:	createProductionBatch(amount, typeId, speed))
Cross reference:	Use-Case 07 & U10
Responsibility	• To send the following user-entered information to the REST API: – Amount of beer to be produced – The type of beer – An id for the batch • To create a new production batch with send parameters & a batch report related to the new batch • To persist the created production batch & batch report • To return a message to the user if successful
Output	• Batch with unique id and required parameters mentioned above • A success message shown to the user
Preconditions	There must exist at least one type in the system
Postconditions	If the conditions for creating a new production batch are true: • A production batch has been created • The id has been associated with the batch • A batch report related to the batch has been created • The production batch has been persisted • view has been updated to notify the actor of said changes

Table 4.1 System operation contract - U01 Create production Batch

Sequence diagram
Figure 4.2 presents a sequence diagram of the use-case U01 which displays the flow of the use-case from an actor's perspective. It is worthy to note that pseudo-code is used in

the diagram as this was created before any coding happened and that routing has been left out of the diagram.

Analysis class diagram

Based on the analysis class diagram from Section 4.1 Figure 4.1 an updated diagram is presented in Figure 4.3 which contains methods relevant to use-case U01 & U02 found through the use-case realization

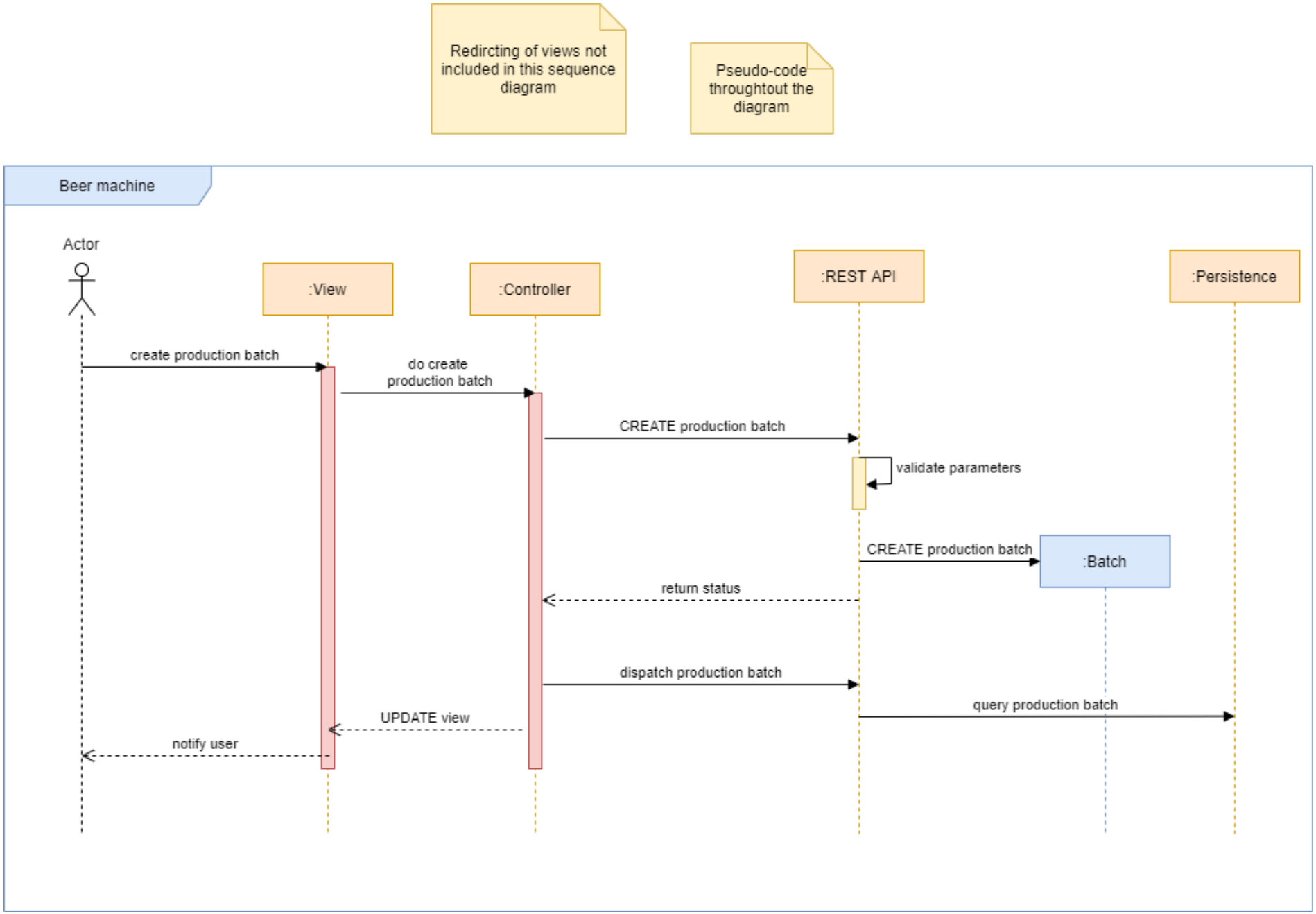

Figure 4.2 Sequence-diagram of use-case U01 createProductionBatch

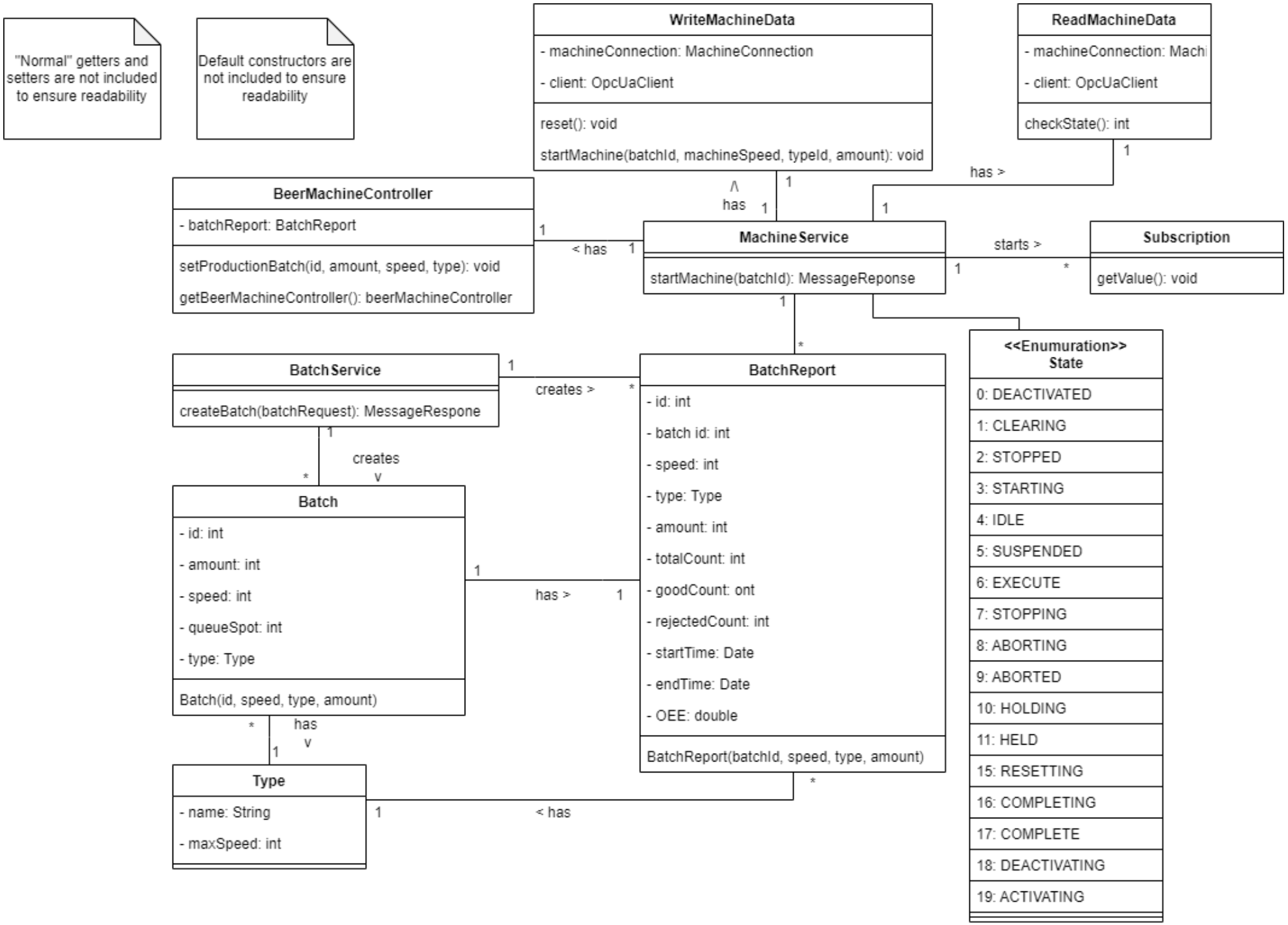

Figure 4.3 Dynamic version of the analysis class diagram.

Chapter 5

Architecture

In this chapter, the thoughts on the system's architectural structure of the system will be described; the multi-tier architecture used, the MVC used in the backend, the backend's communication with the external hardware, and the backend's communication with the frontend and database.

5.1 Multi-tier architecture

The system has been designed with a multi-tier architecture in mind. A backend Rest API where the frontend can get the necessary information and post the changes to the system through the HTTP protocol, as well as communicate with the beer machine simulation. A database made in MySQL is hosted in a docker container with a Phpmyadmin database explorer interface for system administration purposes. A web frontend was made using the PHP Laravel framework. The architecture is shown in the following diagram, Fig. 5.1

5.1.1 Frontend

During the course of this semester, the group has realized that the Laravel framework is redundant, as a second backend server is not necessary. However, this was the technology the group had learned at the point of creating the frontend. Instead, a pure Javascript frontend would have sufficed, using AngularJS, Vue.js, React, or similar frameworks. This does not mean that it is not a decent solution, but it takes up unnecessary server space. The frontend uses the backend Rest API and displays the information to the user on a dynamic webpage, which updates the appropriate HTML classes in real-time with AJAX.

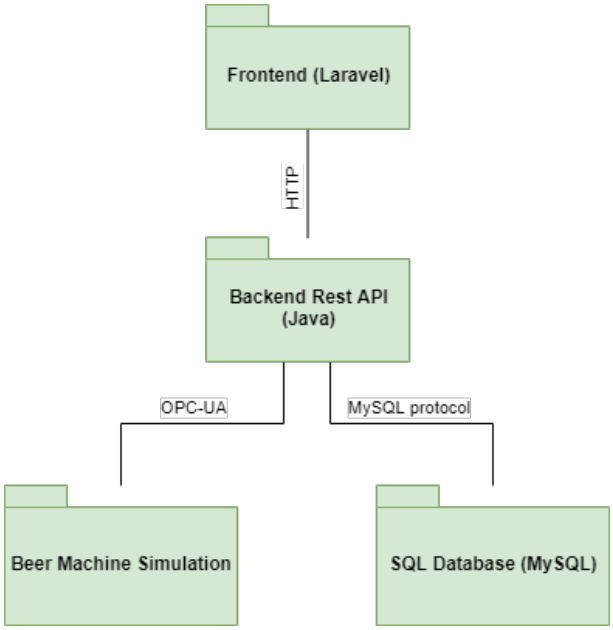

Figure 5.1 Simple package diagram of the system architecture

5.1.2 Backend API

The backend is a web service that acts as the core of the system. It is made in Java using the Spring framework. The backend subscribes to nodes from the beer machine simulation using the OPC-UA protocol through the Java Milo framework. In this way, it is possible to collect data in real-time. The backend also uses the nodes to read and write data. This data is then persisted and made available through the API endpoints on an exposed port.

5.1.3 Database

The database is hosted in a docker container and accessed through the Spring MVC. Thus the data is persisted through the MySQL database directly as the Java objects. The Spring backend and the MySQL database communicate through the MySQL data transfer protocol.

5.1.4 MVC

The backend uses a Model-view-controller to separate the persistence, logic, and view easily. This is done through the Java Spring Boot framework. The framework persists the Java objects to the MySQL database, performs the necessary logic and makes the models available through a Rest API.

Chapter 6

Design

The project is developed with the use of Scrum to structure and manage the project's timeline, see Section 2.1, which also means that the defined requirements are subject to change. To mitigate the impact of such change the system is designed based on the three-layered architecture, to allow for more flexibility during development. The chief benefit of the three-layered architecture is that each layer can be developed, updated, or even scaled simultaneously without impacting other layers[4]. The design is based on the earlier developed analysis of the system described in Chapter 4.

6.1 Design of frontend

The frontend is designed using the Laravel-framework but disregarding the use of Models in this framework. The controller classes are responsible for communication with the backend through HTTP requests in the form of GET, POST, and DELETE through JSON objects. Forms are used to retrieve input from a user which is then transferred to the backend and later persist to a database. Information from the backend is presented in a number of *views* which are written with the Blade extension. A list of *routes* were developed for showing views and handling the requests to the backend - requests such as creating a new batch, starting or stopping the machine. Views are supplemented with Bootstrap to simplify the development of the web pages keeping in mind the ease of use of the web pages as well as the presentation of buttons, variables, and typography. Several JavaScript files are also developed to handle tasks such as showing live data without having to refresh the web page using Ajax and for updating values in charts depending on the chosen type of beer.

6.2 Design of backend

The backend is implemented with the use of Spring Boot as a tool to ease the development of connecting the front- and backend. The backend is split up into three main packages, *data*, *web*, and *service*. Data contains everything related to what needs to be persisted and the objects that need to be supported throughout the system. Due to the use of Spring Boot this package - or more specifically the models - define the structure and design of the database implementation and Spring Boot manages the creation of tables and SQL statements. The *web* package acts as a REST API and the frontend uses it to access the functionality of the backend. The *service* package handles the functionality needed to process the API-requests from the web application. The backend also ensures communication to the beer brewing machine which is done through Milo, an open-source implementation of OPC UA, See Section 2.2. The main functionality of the implemented communication takes place in the three classes *Read*, *Subscription*, and *Write*. The Read-class takes care of opening a connection to read a specific value once. The Subscription-class takes care of opening a connection to constantly monitor the value of a specific parameter and adds the support for live data. The Write-class takes care of sending commands to the machine where any parameters, if needed, are received from the frontend through the REST API. To ensure a more stable flow of data through the system the class BeerMachineController was designed with a singleton design pattern. This prevents the system from getting unreliable data values from several instances of the BeerMachineController class and in turn also prevents the risk of data loss.

6.3 Design of database

Part of the requirements set up during the beginning of the project refers to producing a batch report, Chapter 3 U07. Part of this is designing a database to store the batch reports and related data to show on the application. Every table is fitted with a SERIAL datatype as a unique identifier to the values. Looking at the overall design of the database Figure 6.1, the database is centered around *batchReports* and batches with cardinalities from said tables to the rest of the database which allows the system to find information regarding a single batch based on either the batch or the batch report ID. The database has been normalized to BCNF to make sure the data was readable and minimize redundancy. The database persists the Java Spring Boot models and all have an equal model in the backend subsystem. The database itself is constructed with the help of Spring Boot through the Java backend where everything related to the creation of tables is located in the *data* package described in Section 6.2.

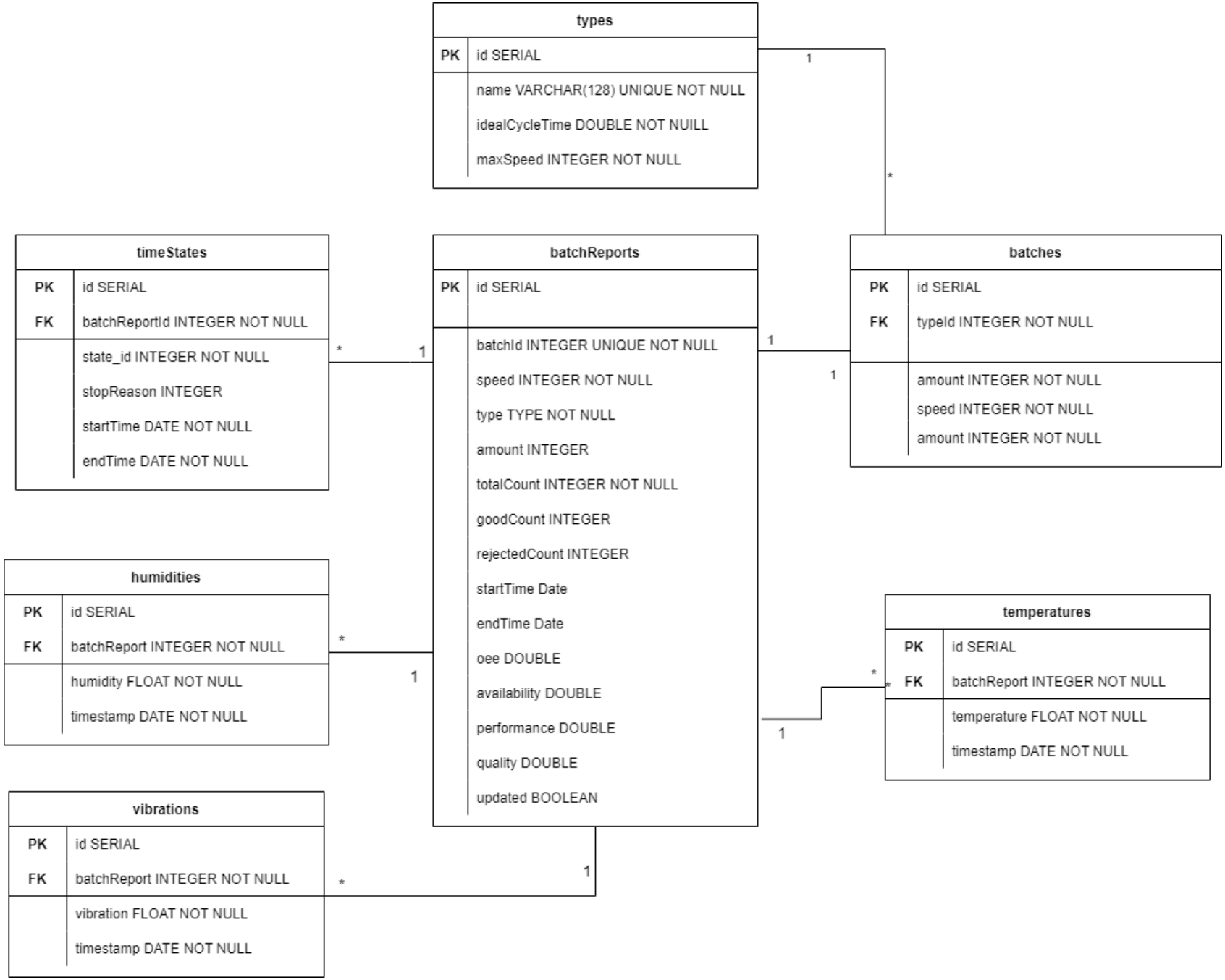

Figure 6.1 Database design diagram

Chapter 7

Implementation

If one takes a look at the problem definition, Section 1.2, one will see that the problem defined was to develop a system to control, maintain and optimize a beer brewing machine with a web application as a user interface and a Java backend that handles the communication to the hardware. The system's web application, the frontend, was developed with the use of Laravel whereas the backend and database were developed with the use of Spring Boot and Java. Use-case U01 and U02 will receive a closer look in relation to the implementation from the frontend all the way to the backend.

7.1 Persistence

In order to persist data received from the machine and user a mySQL database was set up and connected to with the help of Java Spring Boot, see Listing 7.1. A database managed through Java Spring Boot was chosen due to the native support offered by Spring Boot and Java classes.

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/beerMachine?
    useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false
2 spring.datasource.username=root
3 spring.datasource.password=secret
4
5 server.port=8081
```

Listing 7.1: Code-snippet from application.properties of Spring Boot connecting to database

Once a successful connection has been established the project utilizes auto-configuration from Spring Boot to create tables and relations. Listing 7.2 shows part of the `BatchReport.java` class that acts as a template for the creation of tables in the database where Line 8 to 10 define the class as persistable to Spring Boot. The structure of the table itself

is done through Java and "SQL-like" annotations seen from Line 11 to 24. The `@Id` and `@GeneratedValue`, Line 12 and 13, specify the primary key and the generations sequence where line 14 specifies the data type and name of the column. The last important annotation in this class is the `@ManyToOne`, Line 23, which specifies how the table is related to the table *type*.

```
8  @Entity
9  @SequenceGenerator(name = "seqBatchReport", allocationSize = 0)
10 @Table(name="batchReports")
11 public class BatchReport {
12     @Id
13     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator
           = "seqBatchReport")
14     private Integer id;
15
16     @NotNull
17     private Integer batchId;
18
19     @NotNull
20     private Integer speed;
21
22     @NotNull
23     @ManyToOne
24     private Type type;
```

Listing 7.2: Code-snippet from BatchReport.Java creating the tables for batchreports

In order to access persisted data, an interface for each table has to be made. Listing 7.3 shows how the interface for the `BatchReport` table is structured. The annotation `@Repository`, Line 7, and the `JpaRepositiry`, Line 8, specify to Spring Boot the connection between the interface and the table.

```
7 @Repository
8 public interface BatchReportRepository extends JpaRepository<
      BatchReport, Integer> {
9 }
```

Listing 7.3: Code-snippet from BatchReportRepository setting the class as database repository

The mySQL database is containerized in a Docker container as an attempt to make the database easier to run. The service is made in a docker-compose.yml file shown in Listing 7.4. The service called beerMachine-database uses the Docker image mysql:5.7, sets the

database-name to beerMachine and the password to secret, Line 3-7. For more security and in a production environment, the password would be set in a .env file and referenced in the docker-compose file. Lastly the port 3306 from the docker container, is exposed on port 3306 on the host machine, Line 8 - 9.

```
3  version: "3.7"
4  services:
5      beerMachine-database:
6          image: mysql:5.7
7          environment:
8              MYSQL_DATABASE: beerMachine
9              MYSQL_ROOT_PASSWORD: secret
10         ports:
11             - "3306:3306"
```

Listing 7.4: Code-snippet of Database service being created in a docker container

Another service, database-explorer, is also created, See Appendix C, which is a web phpMyAdmin GUI, that enables tools for development purposes.

7.2 Backend

This section about the implementation of the backend will take a closer look at how the backend logic behind the use-cases *U01 Create a production batch* and *U02 Start a production batch* was implemented.

7.2.1 U01 - Create a production batch

In order to create a new batch, the web application has to call a specific route from the `BatchController` class specified at Line 38 in Listing 7.5, passing information related to a new batch via a `post-mapping`. From there the controller class' `addBatch` method sends the obtained information to the `BatchService` class through the method `createBatch` seen at Line 40 in Listing 7.5.

```
37      @CrossOrigin
38      @PostMapping("/add")
39      public ResponseEntity<MessageResponse> addBatch(@RequestBody
            BatchRequest batch) {
40          MessageResponse newBatch = batchService.createBatch(batch)
                ;
41          return new ResponseEntity<>(newBatch, HttpStatus.CREATED);
```

```
42      }
```

Listing 7.5: Code-snippit of the route to create a new batch, placed in the BatchController class

The information passed to the `createBatch` method is then inserted to an instance of Batch which is created at Line 33 Listing 7.10. Using set-methods the instance of batch, `newBatch`, gets the information passed from the web application, Line 35 to 44. At Line 45 the method `save` is called on the `batchRepository` instance which saves the instance of batch to the database. The `batchRepository` is a part of and managed through Spring Boot also discussed in Section 7.1. Last but not least an instance of `BatchReport` is created, Line 46 & 47 which contains information related to the batch. The `batchReport` is also persisted to the database through the `save` method though called on the instance of `batchReportRepository` at Line 48

```
31      @Override
32      public MessageResponse createBatch(BatchRequest batchRequest)
            {
33          Batch newBatch = new Batch();
34          //The calculateAmountToProduce method is used to get a
                more precise estimate for the end result of good count
35          newBatch.setAmount((int) calculateAmountToProduce(
                batchRequest.getType(typeRepository), batchRequest.
                getAmount(), batchRequest.getSpeed()));
36          newBatch.setType(batchRequest.getType(typeRepository));
37          if (queueService.getQueue() != null & queueService.
                getQueue().size() > 0) {
38              Integer lastQueueIndex = queueService.getQueue().size
                    ();
39              Integer lastQueueSpot = queueService.getQueue().get(
                    lastQueueIndex-1).getQueueSpot();
40              newBatch.setQueueSpot(lastQueueSpot + 1);
41          } else {
42              newBatch.setQueueSpot(1);
43          }
44          newBatch.setSpeed(batchRequest.getSpeed());
45          batchRepository.save(newBatch);
46          BatchReport newBatchReport = new BatchReport(newBatch.
                getId(), newBatch.getSpeed(),
47                   newBatch.getType(), newBatch.getAmount());
48          batchReportRepository.save(newBatchReport);
49          return new MessageResponse("New Batch created successfully
```

```
                with a corresponding batch report");
50      }
```

Listing 7.6: Code-snippit of the createBatch method placed in the BatchServiceImpl class

7.2.2 U02 - Start a production batch

In order to start a production batch the web application calls a route from the `MachineStateController` shown at Line 26 in Listing 7.7, which passes the batch id of the started batch via a `post-mapping`. The `startMachine` method then sends the received batch id to the `startMachine` method of the `MachineService` class, Line 28. On Line 29 and 30 an internal server error is returned if the `startMachine` method doesn't successfully start the machine.

```
26      @PostMapping("/start/{batchId}")
27      public ResponseEntity<MessageResponse> startMachine(
            @PathVariable Integer batchId) {
28          MessageResponse startMachine = machineService.startMachine
                (batchId);
29          if (startMachine.getMessage().equals("Machine didn't start
                ...")){
30              return new ResponseEntity<>(startMachine, HttpStatus.
                    INTERNAL_SERVER_ERROR);
31          }
32          return new ResponseEntity<>(startMachine, HttpStatus.OK);
33      }
```

Listing 7.7: Code-snippit of the route to start a batch, placed in the MachineStateController class

The `startMachine` method from the `MachineServiceImpl` class is shown in Listing 7.8. On Line 49-51 the method checks the current machine state and resets the machine if the machine isn't in the idle state of 4. The implementation of the `checkState` and `reset` methods used here can be found in Appendix E Listing 3 and 4. On Line 52 the method uses the received batch id to find the related batch report in the database via the `BatchReportRepository` class. On Line 54-55 the method sends the received data from the found batch report to the `startBatch` method in the `Write` class. On Line 56-57 the method sets the start time of the batch report and saves the updated batch report to the database via the `save` method. On Line 60-63 the `BeerMachineController` class sets the current batch and batch report. This method is enclosed in a try catch block returning either depending on the success outcome

```
46      @Override
```

```
47      public MessageResponse startMachine(Integer batchId) {
48          try {
49              if(read.checkState() != 4) {
50                  write.reset();
51              }
52              BatchReport batchReport = batchReportRepository.
                    findById(batchId).get();
53              //The subtraction of 1 from the type_id is used
                    because of different indexing methods (0index!=1
                    index)
54              write.startBatch(batchReport.getBatchId().floatValue()
                    , batchReport.getSpeed(),
55                      batchReport.getType().getId()-1, batchReport.
                            getAmount());
56              batchReport.setStartTime(Date.from(Instant.now()));
57              batchReportRepository.save(batchReport);
58
59              //Saves the current batch and related data in th beer
                    machine controller
60              BeerMachineController.getBeerMachineController().
                    setProductionBatch(batchId, batchReport.getAmount()
                    ,
61                      batchReport.getSpeed(), batchReport.getType())
                            ;
62              //Saves the current batchReport to the beer machine
                    controller
63              BeerMachineController.getBeerMachineController().
                    setCurrentBatchReport(batchReport);
64          } catch (Exception e) {
65              System.out.println(e);
66              return new MessageResponse("Machine didn't start...");
67          }
68          return new MessageResponse("Machine started...");
69      }
```

Listing 7.8: Code-snippit of the startMachine method, placed in the MachineServiceImpl class

The `startBatch` method from the `Write` class is shown in listing 7.9. Firstly, the method creates a connection and connects a client to send the received data to the machine, Line 33-35. On Line 37-49 a node that references each passed value in the machine is created and the client uses the created node to send the value received to the machine itself.

After the values have been sent to the machine the method sends a command to start the machine and a request to change the command which executes the start command, Line 51-55. Finally, after all the values and commands have been sent to the machine the client disconnects from the machine which prevents the system from running out of available connections, Line 56.

```
31      public void startBatch(Float batchId, float machine_speed, int
            type_id, float amount) {
32          try {
33              machineConnection = new MachineConnection();
34              machineConnection.connect();
35              client = machineConnection.getClient();
36
37              NodeId mach_speed_node = NodeId.parse("ns=6;s=::
                    Program:Cube.Command.MachSpeed");
38              client.writeValue(mach_speed_node, DataValue.valueOnly
                    (new Variant(machine_speed))).get();
39
40              NodeId batchIdNode = NodeId.parse("ns=6;s=::Program:
                    Cube.Command.Parameter[0].Value");
41              client.writeValue(batchIdNode, DataValue.valueOnly(new
                     Variant(batchId))).get();
42
43
44              NodeId type_id_node = NodeId.parse("ns=6;s=::Program:
                    Cube.Command.Parameter[1].Value");
45              client.writeValue(type_id_node, DataValue.valueOnly(
                    new Variant(type_id))).get();
46
47              NodeId amount_node = NodeId.parse("ns=6;s=::Program:
                    Cube.Command.Parameter[2].Value");
48              client.writeValue(amount_node, DataValue.valueOnly(new
                     Variant(amount))).get();
49
50
51              NodeId command = NodeId.parse("ns=6;s=::Program:Cube.
                    Command.CntrlCmd");
52              client.writeValue(command, DataValue.valueOnly(new
                    Variant(2))).get();
53
54              NodeId change_request = NodeId.parse("ns=6;s=::Program
```

```
                :Cube.Command.CmdChangeRequest");
55              client.writeValue(change_request, DataValue.valueOnly(
                    new Variant(true))).get();
56              client.disconnect();
57          }
58          catch(Throwable ex) {
59              ex.printStackTrace();
60          }
61      }
```

Listing 7.9: Code-snippit of the startBatch method placed in the Write class

Listing 7.10: Code-snippit of the createBatch method placed in the BatchServiceImpl class

7.3 Frontend

In this subsection, the implementation of use-case *U01 Create Production Batch* & *U02 Start Production Batch* is described along with some of the decisions regarding the implementation. For illustrative purposes watch the video made by the group, which is attached in the hand-in.

7.3.1 U01 - Create Production Batch

In order to *create* a new production batch the user navigates to the batch creation form. The user can then choose the parameters which are listed in the use-case, see Section 3.2.

```
18      <form action="{{route("batch.store")}}" method="post">
19          @csrf
20          <label for="amount" class="fw-bold">Amount</label>
21          <input type="number" id="amount" name="amount" class="
                form-control" required>
22          <label for="beerType" class="fw-bold">Type of Beer</label>
23          <select id="beerType" name="type" class="form-control
                dropdown" onchange="chartScript()">
24              @foreach($types as $type)
25                  <option value="{{$type['id']}}">{{$type['name']}}<
                        /option>
26              @endforeach
27          </select>
```

```
28          <label class="fw-bold">Speed</label>
29          <input type="number" id="speed" name="speed" class="
                form-control" placeholder="460" required>
30          <br>
31          <button type="submit" class="form-control">Submit</button>
32      </form>
```

Listing 7.11: Code snippet of the create-form

Listing 7.11shows the HTML structure of the create form. It is important to note, that the form-action is not ordinary. Hence the group is using the Blade notation, it is, therefore, possible to use the shorthand-notation **route("batch.store")** which refers to the route-name that is defined in the **web.php** file. See Appendix F. The Blade notation is also used at line 25 where the **$type['id']** refers to the data sent by the controller to the view and sets that value as the id of the `<option>` element. The same is done for the name of that element.

When the user is satisfied with the new production batch, the button "Submit" is then pressed and the user is redirected to the batch list. This new batch is not started yet, though is placed in the queue and can be started at a later point in time. This is further explained in the Subsection 7.3.2.

After submission of the form, the web route makes a *post* call to the BatchController which then validates the inputs and sends the data to the REST side of the application which was described in Section 7.2.

7.3.2 U02 - Start production batch

In order to *start* a production batch the user can - on the main page - choose to start the first production batch in the queue which will run the production and present live data from the machine on the main page. Alternatively, a user can choose to start a specific batch from a list of batches not yet started.

Implementation of starting a batch from the frontend is a route where an id from the list of batches is provided and afterward, the batch is deleted. These functions are called `start(Request)` and `destroy(Request)` as seen per Listing 7.12 & Appendix D.

```
118     public function start(Request $request)
119     {
120         $id = $request->id;
```

```
121         Http::post("http://localhost:8081/machine/start/$id");
122         return redirect()->route("batch.destroy", $request->id);
123     }
```

Listing 7.12: Codesnippet from BatchController.php of start(Request)

When starting a batch, it was important that live data of the beer machine was available to the user at all times and without having to refresh the website to get updated values for variables such as humidity level or ingredients level. This was achieved with the use of *Asynchronous JavaScript and XML* (Ajax) since it provides the technology to develop asynchronous web applications and therefore can update values in real-time. The function `getHumidty()` requests a value from a backend endpoint as a JSON format and if successful will continue to execute `humidity()` and update the HTML every 0.5 seconds.

```
118     var getHumidity = function() {
119         $.ajax({
120             url: "http://localhost:8081/machine/getHumidity",
121             cache: false,
122             type: 'GET',
123             dataType: 'json',
124             headers: {
125                 'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr(
                        'content')
126             },
127             success: humidity
128         });
129     }
130     var humidity = function(data) {
131         $(".humidity").html(data);
132         setTimeout(getHumidity, 500);
133     }
134     getHumidity();
```

Listing 7.13: Codesnippet from machineVariables.js of Ajax function updating value in real-time: Humidity

Chapter 8

Validation & verification

In order to test the developed system and see if it has the functionality required to achieve the requirements from Chapter 3 Unit testing, use-case testing, user testing and frontend performance testing was performed.

8.1 Unit testing

The appropriate backend functionality described in use-case U01 and U02 have been tested using JUnit and the Spring test framework. The functionality is sorely tested in the Java backend, software simulation and mock databases, while the classes have been created to test the functionality of the primary methods in the use cases, `MachineService.startMachine()` and `BatchService.createBatch()`. Testing that these two methods work should verify the functionality of the use cases, as these are the methods being invoked when the appropriate routes are called. A sanity check for testing that the system boots up normally is also implemented.

8.1.1 BatchServiceTest

In this class the use case U02 is tested, as described in Table 3.3. This is done by invoking the method createBatch(), and testing the following main flow points in the detailed use-case description; 6, 7 and 8, in the corresponding method tests; `assertThatBatchIdIsUnique()`, `givenBatchRequestShouldAddBatchToRepository()` and `assertThatBatchReportIsCreated()` as seen in Figure 8.1. The rest of the main flow is done in the frontend and tested in other ways. The following code snippet Listing 8.1shows the implementation of the unit test. What is not shown is the creation of the batchRequest, however this is simply a JSON object with an amount, speed and typeId, all integers. The code counts the batchRepository and puts all the current batchId's in a list, before adding the new batch.

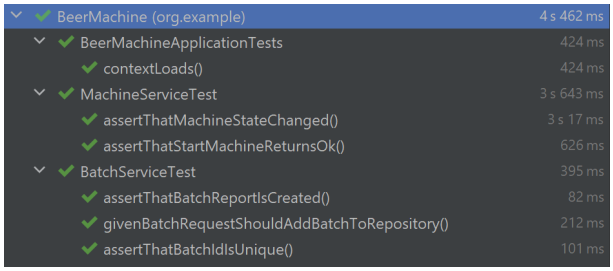

Figure 8.1 Overview of the completed unit tests and the elapsed time.

The message response is saved when invoking the createBatch() method, and tested that it is the correct response, indicating that the method worked properly. It is also asserted that the number of batches in the repository has gone up by 1.

```
48 @Test
49     void givenBatchRequestShouldAddBatchToRepository(){
50         long count = batchRepository.count();
51         batchRepository.findAll().forEach((n) -> ids.add(n.getId()
               ));
52         MessageResponse msg = batchService.createBatch(
               batchRequest);
53         assertThat(msg.getMessage()).isEqualTo("New Batch created
               successfully with a corresponding batch report");
54         assertThat(batchRepository.count()).isEqualTo(count+1);
55     }
```

Listing 8.1: Code-snippet from BatchServiceTest testing the U01 main flow point 7

The other methods work in a similar way. The `assertThatBatchIdIsUnique()` makes sure that the new batchId is not in the id-list saved in the above code snippet, and the `assertThatBatchReportIsCreated()` checks that there exists a new batch report in the BatchReportRepository for the new batch that is not null.

8.1.2 MachineServiceTest

The MachineServiceTest-class is created to test use-case U02, as described in Table 3.4. It is tested that the method startMachine returns the correct MessageResponse in method `assertThatStartMachineReturnsOk()` and that the machine state in the simulation changes to state 6 in `assertThatMachineStateChanged()`. The latter can be seen in the following code-snippet, 8.2.

```
59 @Test
60     void assertThatMachineStateChanged() {
```

```
61          try {
62              Thread.sleep(2000);
63          } catch (InterruptedException exception) {
64              exception.printStackTrace();
65          }
66          assertThat(BeerMachineController.getBeerMachineController
                ().getMachineState().getStateSub().getMachineState()).
                isEqualTo(6);
67      }
```

Listing 8.2: Code-snippet from MachineServiceTest testing the U02 main flow point 2.1

The startMachine() method is called in the @BeforeEach setup() function, and a short delay is inserted in the test, giving the simulation and the subscriptions time to update. It is then tested that the machine state subscription has changed to 6.

8.2 Use-case testing

The use-cases U01 and U02 will be tested by running through the steps of the main flow in the use-cases. Additionally, the testing will also include the pre-conditions, post-conditions, and cross-referenced use-cases.

8.2.1 Use-case testing of U01 - Create production batch

The first two steps of the main flow in the use-case U01 revolves around initiating the creation of a new production batch. This is done by pressing the *Make new Batch* button on the page *Batches* shown in Appendix G Figure 2. The next three steps 3, 4, and 5 are focused on the features of the creation page shown in Appendix G Figure 3. At step 3 the system sets a predefined optimal speed of the production based on the error function graphed at the bottom of the page. In this instance, the predefined speed for the type Pilsner is 460. At step 4 the actor is prompted to set the parameters amount, type, and speed, which in this case is set to 25.000, Pilsner and the predefined 460. At step 5 the actor presses the *Submit* button. The last four steps 6, 7, 8 and 9 focus on saving the batch to the system with the correct parameters, creating a batch report for the new batch and returning the actor to the page *Batches*. This is shown in Appendix G Figure 4 where the new batch can be seen at the bottom of the list of batches with the unique ID of 13. Furthermore, the batch report related to the batch has been created and linked to under the *Batch Report* column underlined and colored blue.

8.2.2 Use-case testing of U02 - Start production batch

The use-case U02 has two pre-conditions that specify that the brewing machine must be in the idle state 4 and must have at least one batch in the system, Appendix G Figure 5 & in Appendix G Figure 4. The second step of the main flow shows the first option of starting the production of a batch. This is done by pressing the *Start batch* button on the *Home* page, Appendix G Figure 5, which in turn starts the next batch from the database. The third step shows the second option of starting a production batch, by navigating to the page *Batches*, which is done by selecting the *Batches* item in the navbar menu, Appendix G Figure 6. The steps 3.1 and 3.2, 7Appendix G Figure 7, makes the system list all the available batches that the actor can start and the actor then chooses a specific batch. The steps 3.3, 3.4 and 4 starts the chosen batch, returns the actor to the *Home* page and starts the live data. The post-condition and the cross reference are visualized in Appendix G Figure 8 where the *Home* page shows that the chosen *batch 13* has been started and is running based on the machine state 6 (executing). The figure also shows the live data where e.g the yeast & hops values decrease as the ingredients are being used.

8.3 User testing

The test person was given the following tasks to complete within the Beer Brewing Program

- Enter a machine state of 4 (idle) by resetting the machine
- Start the production of a batch on the *Home* page
- Stop the machine on the *Home* page
- Reset the machine to enter a machine state of 4 (idle)
- Navigate to the *Batches* page and create a new batch
  - Fill out the 3 parameters of the batch as it makes sense to you
- Locate and start your newly created batch. (Take note of the batch id)
- Navigate to the *Reports* page and find the batch report for your batch (Find by id)
- Navigate to the *Home* page and stop the machine

The feedback questions shown in Appendix H Table 4 were given to the test person in order to receive feedback on the usability of the system. The general feedback of the test was that the system is simplistic and easy to navigate, however, this simplicity also causes the user experience of a completely new user to be quite poor. Examples of this would be that the values displayed in the interface are lacking units which make them

ambiguous to the user. The user was also allowed to input invalid values when creating a batch which resulted in unrelated values being displayed in the batch report. The system also occasionally failed to reload and show the correct machine state which also caused confusion to the first-time user.

8.4 Frontend Performance

Regarding the frontend, a requirement for the general performance has been made. This requirement states that the webserver should have a maximum response time of 5 seconds. This can be measured with a variety of tools, one of these can be Google Chrome's developer tool, Lighthouse. The purpose of testing the web-servers performance is to determine which resources are inefficient and take a long time to load.

This test results in the following score: 87 for Performance, 100 for Accessibility and lastly 100 for Best Practices. Form the LigthHouse test, it is possible to identity for more time demanding resources, which ended up being the two files: *app.css* and *app.js*. These files are default layout and script files in the Laravel framework, which - according to LightHouse - can reduce the response time by 690ms and 1.050 ms respectively. See Figure 8.2. This indicates, as we knew, that the use of a redundant webserver slows the system down.

URL	Transfer Size	Potential Savings
/css/app.css (localhost)	213.4 KiB	690 ms
...bootstrap/bootstrap4-toggle.min.css (localhost)	2.9 KiB	80 ms
/js/app.js (localhost)	499.0 KiB	1,050 ms
...jquery/jquery-3.6.0.min.js (localhost)	135.9 KiB	560 ms
...jquery/jQueryScript.js (localhost)	87.5 KiB	400 ms
/js/bootstrap.min.js (localhost)	56.9 KiB	320 ms

Figure 8.2 Lighthouse time savings

Chapter 9

Discussion

The goal of this project is to develop a software system that can control, maintain and optimize the beer production for Refslevbæk Bryghus A/S while easing the manual workload for the production manager as described in Chapter 1. The system was always meant to be seen as a prototype, which among other functions has the ability to connect to a physical simulation of a beer brewing machine - With this aspect in mind, the project is considered a success.

The group has set up a series of requirements as seen in Chapter 3.1 which further defines the functionalities the system should offer. These requirements were prioritized through the MoSCoW method. Based on the prioritization a series of unit tests, use case tests, and user tests were developed along with a performance test of the developed web application, Chapter 8, with one goal in mind; to validate the project's requirements. Based on the performed tests the project's *must-haves* requirements defied in Figure 3.1 are considered successfully implemented. Though, this does not mean that the implementation of the project is perfect on the contrary there is room for improvement.

The user test was only conducted on one person but it was still useful as it showed how a first-time user experiences the interface of the system. The general results of the test showed that the first-time user found the interface to be simplistic and easy to navigate. This simplicity also caused some confusion for the user as several values on the interface were lacking units. As a result of this, the user only had a general idea of what the live data was showing with the remaining ingredients and maintenance status. There were several missing units throughout the system and therefore the values were only identifiable by the name of the label above the value. The missing units also caused user errors as the user entered invalid input values when trying to create a production batch. This led to a butterfly effect in the system that caused the related batch report to have values

that did not correlate with the batch that was created.
All in all the user found the system to be useful as it worked mostly as intended. The UI is generally user-friendly and intuitive but the UX leans more toward an experienced user rather than a user with no prior knowledge of the system or the production it is based on. User-friendliness is also one of the non-functional requirements described in Chapter 3.1.

The group decided to make use of the Laravel web framework for the frontend subsystem, even though it was unnecessary. This was done since this was the web framework taught in class. The use of a Javascript frontend framework would in hindsight be more optimal. The deployment of the system could be optimized with further use of Docker and Docker-Compose or Kubernetes to allow for possibly easier scalability and faster development. The Java backend and the web frontend could each be containerized and hosted on a server, making it much easier to access.

Development of the system has happened partly based on the cases provided for the project described in Chapter 1 and in turn requirements and analysis in Chapters 3.1 & 4. The group might have been able to further develop and possibly achieve better results if the system was tested additionally along with development with more specified user tests and feedback. Additional ideas for further development will be specified in Chapter 11.

Chapter 10

Conclusion

The developed system is a solution for Refslevbæk Bryghus A/S to produce beer and optimize the beer brewing production with automation in mind. The software system provides the necessary functionality to communicate to the physical brewing machine for Refslevbæk Bryghus A/S. The optimization is provided through several implementation points; the web application developed to give the end-user a structured overview of data from the machine, the ability to control the machine through the same software as well as the calculation for the optimal speed and the Overall Equipment Efficiency.

The defined requirements, the developed analysis as well as the design of the system created the foundation for the development of a prototype of a potentially more elaborate system that could be used by Refslevbæk Bryghus A/S in order to automate their production line. Based on the performed tests it can be concluded that the system meets the expected requirement thus providing software that can communicate with the physical machine and reduce the manual work done by the company. That said, the developed system has room for improvements which the tests also showed in form of the performed user test.

The developed prototype might provide building blocks for a more advanced system and might benefit from dockerizing the system to allow for faster development and easier scalability.

It can further be concluded that the systems functionality to calculate the overall equipment efficiency as well as storing the performance in reports in the database can be categorized as parts of functionality needed for an MES system. Though, an entire MES system has not been developed.

Chapter 11

Perspectives

The Beer Machine system, which goal is to produce a list of different types of beer while optimizing said production, could see use in a broad array of breweries and is not specifically limited to the brewing company Refslevbæk Bryghus A/S. The MES prototype that was built during this project can be used as a base and building block for creating a more elaborate brewing system in the future.

Much needed meetings directly with the customer, Refslevbæk Bryghus A/S, as it would have provided feedback from the client who, in the end, will be the main user of the system. If the project had included customer meetings after each sprint to show the developed product, more specific requirements might have been reached and thus providing a better result for the client. Especially when the requirement specification is crucial for further development of the system.

If this project was to be developed further, the requirements from Chapter 3.1 which were not yet implemented could be looked at. The implementation of these requirements would be combined with progressive feedback from end-users preferably the client themselves, Refslevbæk Bryghus A/S - specifically what an end-user would want to do in such a system but also what data they would like to pull from the system. A way of intercommunication between stakeholders of the system could also be interesting to look at. It would also be good to look again at the user-test described in Appendix H and develop further based on the feedback given.

Bibliography

[1] Ecclipse Foundation. *Eclipse Milo*. URL: https://github.com/eclipse/milo. (accessed: 16.12-2021).

[2] Laravel. *Laravel Blade template engine extention*. URL: https://laravel.com/docs/8.x/blade. (accessed: 20.12-2021).

[3] *Spring Boot - Introduction*. URL: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm.

[4] IBM Cloud Education. *Three-Tier Architecture*. URL: https://www.ibm.com/dk-en/cloud/learn/three-tier-architecture. (accessed: 16.12-2021).

Appendix

A Noun analysis U01 & U02

Use-case: Create production batch
ID: U01
Primary actors: Manager
Brief description: The manager can set a production batch with the following parameters: Amount, Type & Speed. The system will allocate a unique batch ID.
Main flow: 1. The use-case starts when the actor wishes to set a production batch 2. The actor presses the "Make new batch" button 3. The system sets a predefined speed based on the calculated error function «include U10» 4. The actor sets the following parameters: 4.1. Amount 4.2. Type 4.3. Speed 5. The actor presses the button "submit" 6. The system generates a unique batch id for the current batch 7. The system saves the batch in the system 8. The system creates a unique batch report for the new batch «includes 07> 9. The system returns the actor the batch page
Alternative flow: • The actor can cancel the use-case between 2-7. – If the use-case is canceled between 2-7 nothing new will be saved to the system
Post conditions: • The batch has been saved in the system • A unique batch report is saved to the system
Cross References: • U07 • U10

Table 1 Noun analysis of detailed Use-case - Create Production Batch

Use-case: Start production batch
ID: U02
Primary actors: Manager
Brief description: The manager can start the machine manually and begin production of the chosen batch.
Pre-conditions: • The beer brewing machine must be in the idle state (State 4) • There must be at least one batch in the system

Main flow: 1. The use-case starts when the actor wishes to start a production batch 2. The actor can press the button “Start batch” on the "Home" page 2.1. The system will then start the next batch from the database table *OR* 3. The actor can press the button "batches" 3.1. The system will show a list of all batches 3.2. The actor can now choose a specific batch from the list 3.3. The system will started the chosen batch 3.4. The system returns the actor back to the "Home" page 4. The system starts the live data «includes U08»
Alternative flow: • The actor can cancel the use-case at 2 or between 3 - 3.2.
Post conditions: • The system has started a production batch
Cross References: • U08

Table 2 Noun analysis of detailed Use-case - Start Production Batch

B Use-case realization U02

Contract	
Operation:	startMachine(batchId)))
Cross reference:	None
Responsibility	• To send following user-entered information to the REST API: – batchID • To update the values of the system to match the batch-values • To start the production of batch with batchID • To return a message to the user is successful
Output	• The machine has been started • A success message shown to the user
Preconditions	• Machine State has to be (4) • A batch with batchID has to exist in the system

Postconditions	If the conditions of starting the machine are true: • The values of the system has been updated according to batch • A batch with batchID has been started • View had been updated to notify the actor that the machine has started

Table 3 System operation contract - U02 Start the machine

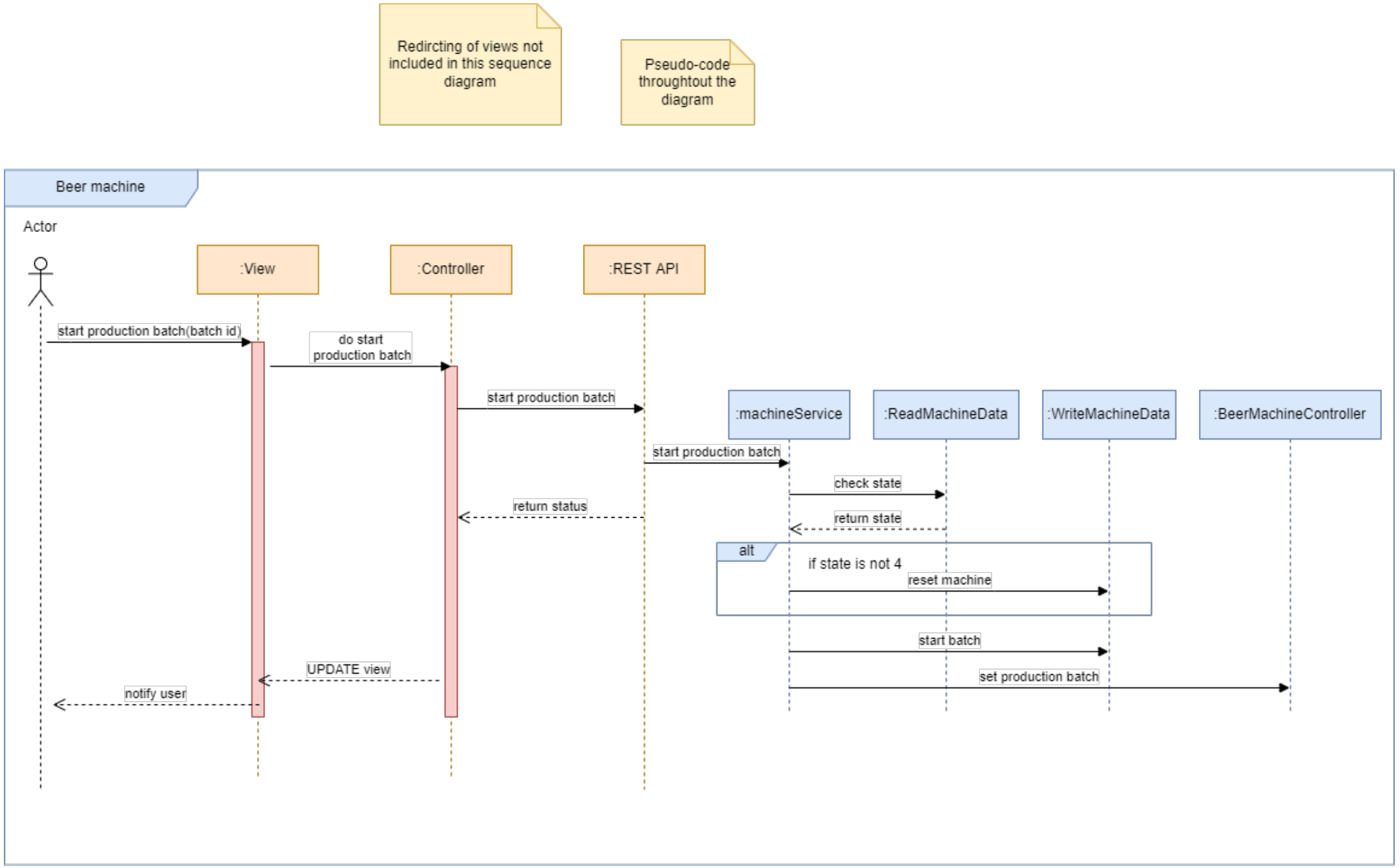

Figure 1 Sequence-diagram of use-case U01 createProductionBatch

C Database Dockerfile

```
version: "3.7"
services:
    beerMachine-database:
        image: mysql:5.7
        environment:
            MYSQL_DATABASE: beerMachine
            MYSQL_ROOT_PASSWORD: secret
        ports:
            - "3306:3306"
    database-explorer:
        image: phpmyadmin
        restart: always
        ports:
            - 8036:80
        environment:
            PMA_ARBITRARY: 1
            PMA_HOST: beerMachine-database
            PMA_PORT: 3306
            PMA_USER: root
            PMA_PASSWORD: secret
```

Listing 1: Entire docker-compose.yml file for the database

D Frontend implementation

```
    public function destroy(Request $request)
    {
        $id = $request->id;

        Http::delete("http://localhost:8081/batch/delete/$id");
        return redirect("/")->with('message', "Batch $id has been
            started");
    }
```

Listing 2: Codesnippet from BatchController.php of destroy(Request)

E Backend implementation

```
    public int checkState() {
```

```
45          try {
46              machineConnection = new MachineConnection();
47              machineConnection.connect();
48              client = machineConnection.getClient();
49
50              NodeId nodeId = NodeId.parse("ns=6;s=::Program:Cube.
                    Status.StateCurrent");
51
52              DataValue dataValue = client.readValue(0,
                    TimestampsToReturn.Both, nodeId).get();
53              Variant variant = dataValue.getValue();
54              stopReason = (int) variant.getValue();
55              client.disconnect();
56
57          } catch (Throwable ex) {
58              ex.printStackTrace();
59          }
60          return stopReason;
61      }
```

Listing 3: Code-snippit of the checkState method placed in the Read class

```
28      public void reset() {
29          try {
30              machineConnection = new MachineConnection();
31              machineConnection.connect();
32              client = machineConnection.getClient();
33
34              NodeId change_request = NodeId.parse("ns=6;s=::Program
                    :Cube.Command.CmdChangeRequest");
35              NodeId command = NodeId.parse("ns=6;s=::Program:Cube.
                    Command.CntrlCmd");
36              client.writeValue(command, DataValue.valueOnly(new
                    Variant(1))).get();
37              client.writeValue(change_request, DataValue.valueOnly(
                    new Variant(true))).get();
38              client.disconnect();
39          }
40          catch(Throwable ex) {
41              ex.printStackTrace();
42          }
```

```
43      }
```

Listing 4: Code-snippit of the reset method placed in the Write class

F PHP routes

```
18      //index
19      Route::get('/', [BatchController::class, "index"])->name("
            index");
20      //configuration
21      Route::get('/batches', [BatchController::class, "list"])->name
            ("batch.list");
22      //batches
23      Route::get('batch/create', [BatchController::class, "create"])
            ->name("batch.create");
24      Route::post('batch/create', [BatchController::class, "store"])
            ->name("batch.store");
25      Route::post('batch/start', [BatchController::class, "start"])
            ->name("batch.start");
26      Route::post('batch/stop', [BatchController::class, "stop"])->
            name("batch.stop");
27      Route::post('batch/abort', [BatchController::class, "abort"])
            ->name("batch.abort");
28      Route::post('batch/reset', [BatchController::class, "reset"])
            ->name("batch.reset");
29      Route::post('batch/maintenance', [BatchController::class, "
            maintenance"])->name("batch.maintenance");
30      Route::get('batch/destroy/{id}', [BatchController::class, "
            destroy"])->name("batch.destroy");
31      //queue
32      Route::post('queue/up/{id}', [BatchController::class, "queueUp
            "])->name("queue.up");
33      Route::post('queue/down/{id}', [BatchController::class, "
            queueDown"])->name("queue.down");
34      //reports
35      Route::get('/report', [ReportController::class, "
            showReportWithId"])->name("reportWithId.show");
36      Route::get('/reports', [ReportController::class, "reportList"
            ])->name("report.list");
```

Listing 5: List of web-routes

G Use-case testing of U01 and U02

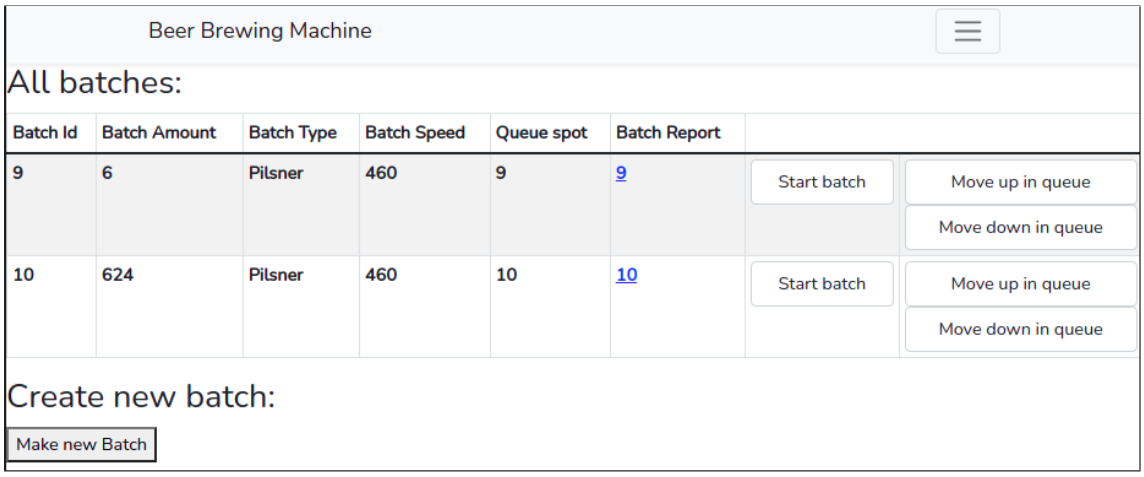

Figure 2 Step 1 and 2 in U01 from Table 3.3

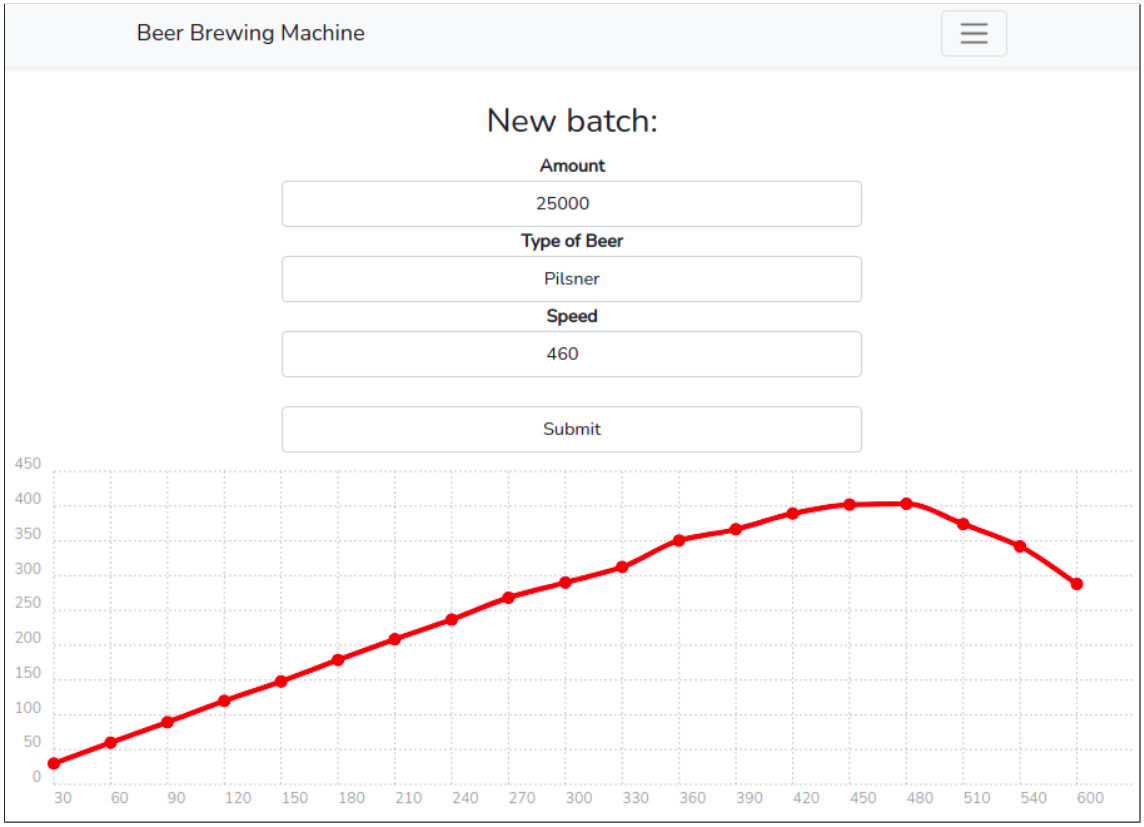

Figure 3 Step 3 to 5 in U01 from Table 3.3

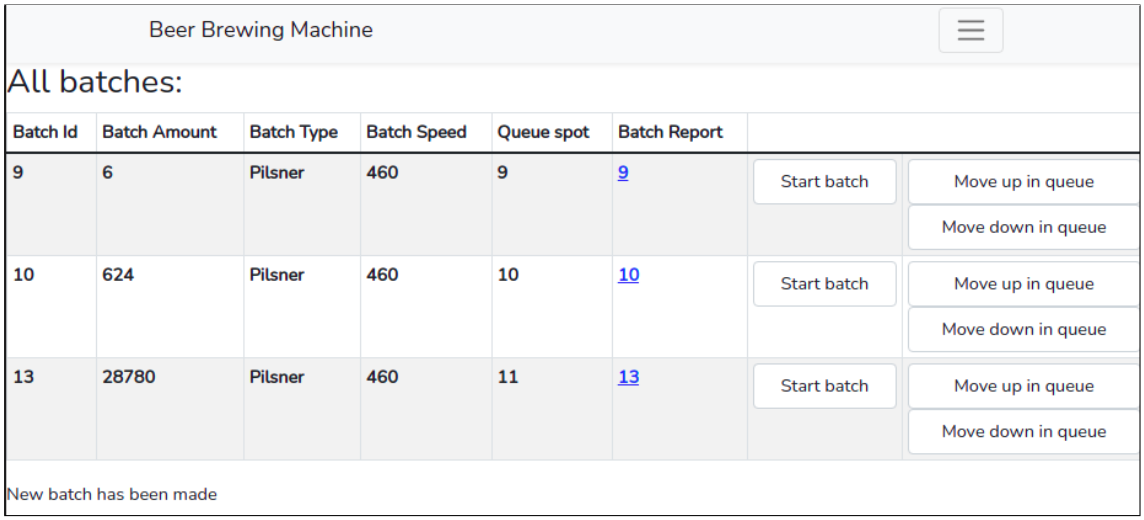

Figure 4 Step 6 to 9 in U01 from Table 3.3

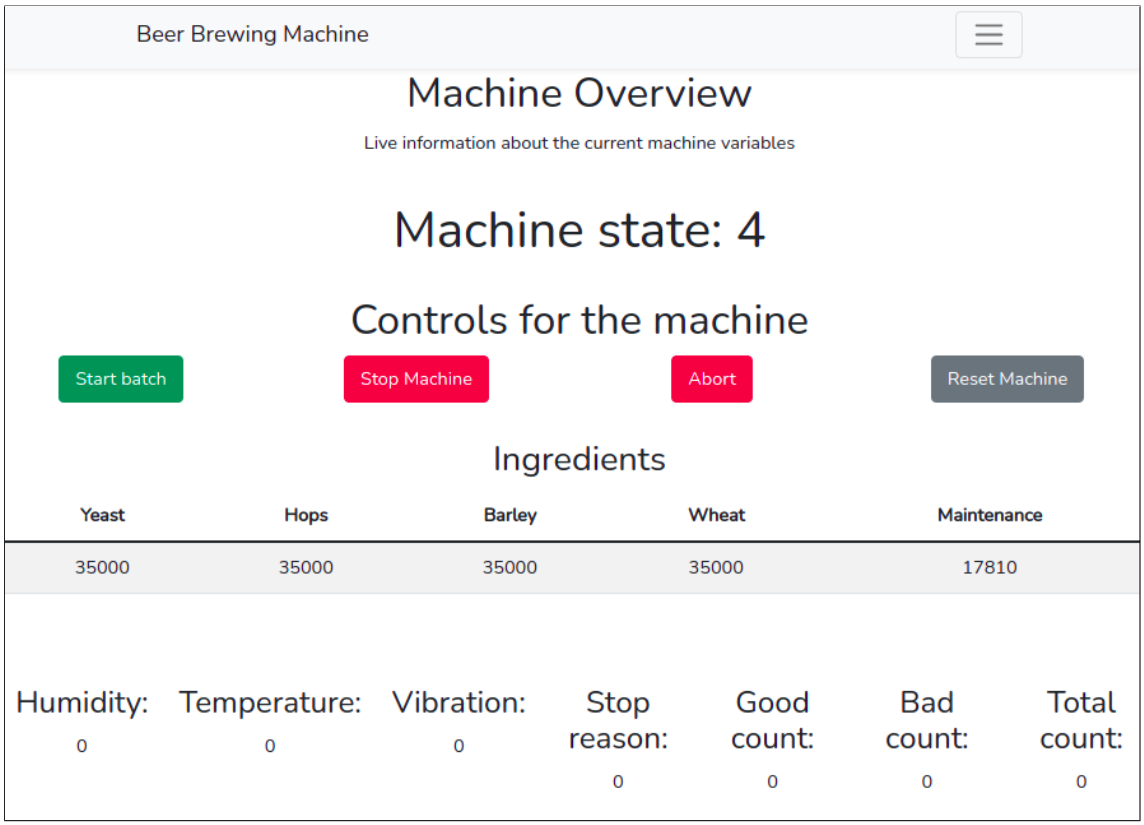

Figure 5 Step 2 in U02 from Table 3.4

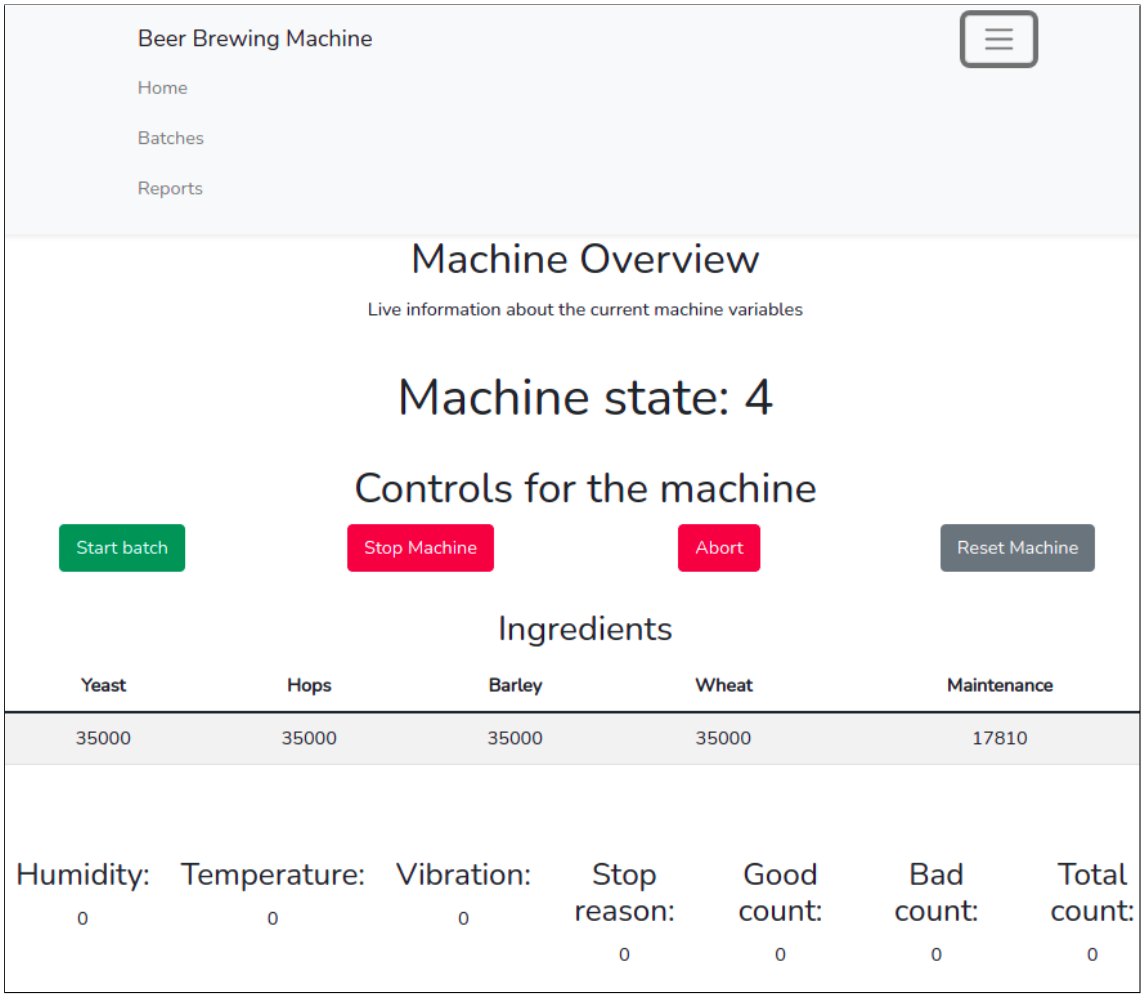

Figure 6 Step 3 in U02 from Table 3.4

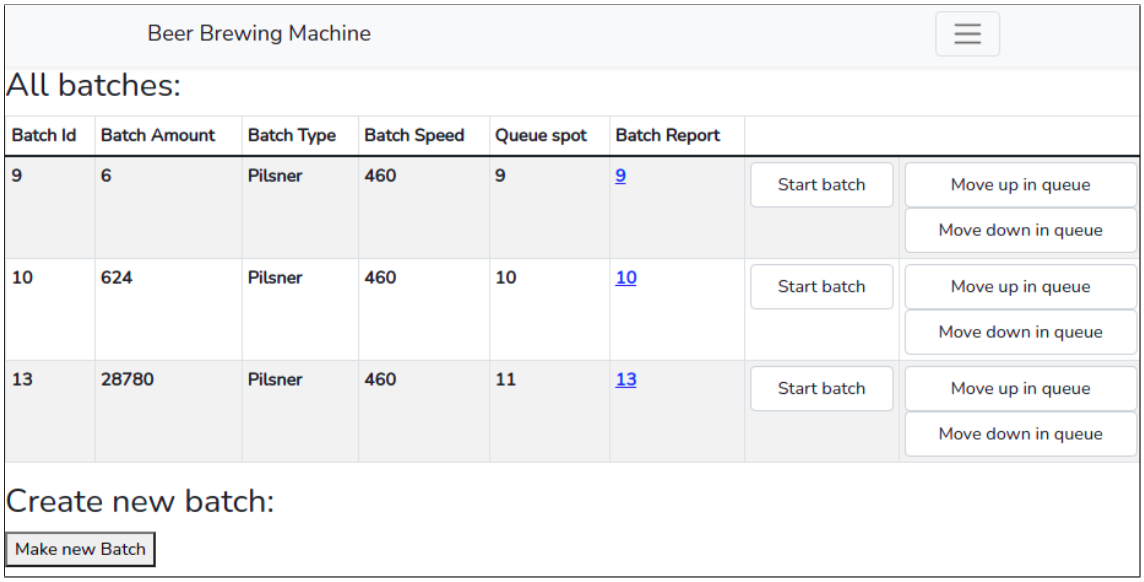

Figure 7 Step 3.1 and 3.2 in U02 from Table 3.4

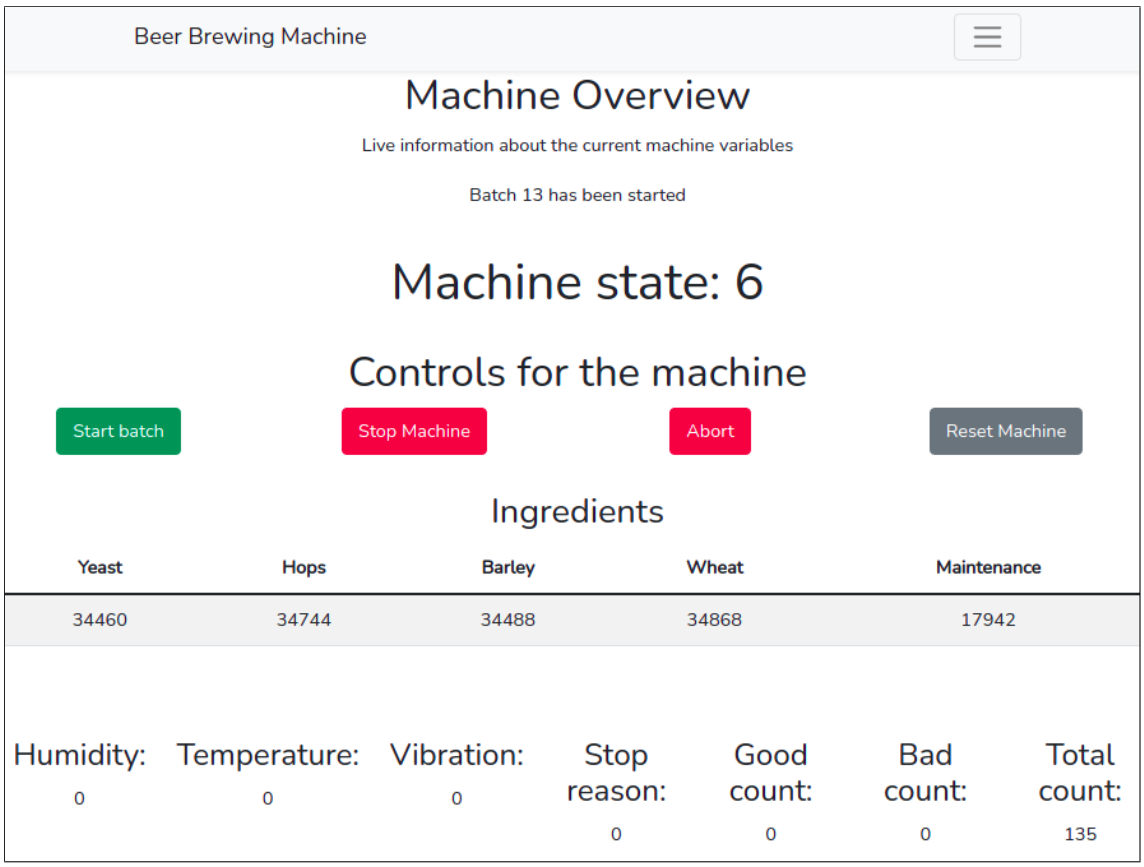

Figure 8 Step 3.3, 3.4 and 4 in U02 from Table 3.4

H User test

Questions	Answers
What was your experience like with the feedback received from the application when changing the machine states? How could the feedback be improved?	The feedback failed to update on its own, and a manual update was not prompted. The Machine state could be described further, though I would expect as much with regular use. The current function of the machine (Idle, stopped, started) could be shown more clearly by increasing font size, though I believe the machine state produce the same information if detailed for the user.
What is your understanding of the graph shown when creating a new batch? Try changing the type of beer to see a change in the graph. How could the graph be improved?	The graph shows a developing process, presumably of the beer production, though no labels are given for the axes. Though the graph does not seem to change based on input given in the fields above. (This, however, becomes visible when changing between beer types, yet not something intuitive)
What was your experience of creating a new batch? Did it make sense to you what values to input? How could the creation be improved?	The interface was simple and easily accessible, but the lack of units for given values created some confusion as to the exact nature of the interface. I assumed the volume input would be liters, while the time input, I would expect to be seconds. The interface and values might be intuitive to someone familiar with brewing, though a first-time user might have a harder time understanding the exact nature of the interface and the values, however, the simplicity works in the favor of presenting it to new users.
What was your experience like finding your newly created batch and starting it? Was it easy to find based on the parameters, ID, and order of the batches? Was the feedback displayed on the main page after starting the batch sufficient? How could the process or feedback be improved?	The interface was in general easy to navigate and rather intuitive. The presentation of the batches was easy to navigate and gave a good overview, though there seems to be no way to either delete or archive the batches already produced, so that those in progress or awaiting production can be more easily found. The feedback about starting and stopping the process was clear and direct, however, it could be presented more visibly, perhaps by increasing font size.

What is your understanding of all the updating values on the "Home" page? (This test runs with the simulated machine, hence why several data points won't be updating). How could the display of values be improved?	The values lack units, making it more accessible for new users and easier to work with, perhaps in a rush, even for experienced users, which could help avoid production mistakes. My first guess would be that all the ingredients were given in tons, while maintenance is something I would normally presume to be given in years, without another prompt or unit than the numbers themselves, however, it seems to be months rather than years.
What was your experience like looking at the values of the batch report? Did they make sense to you? Did they correlate with the values you entered earlier? How could the understandability or the structure be improved?	Several values were missing, as the values given for 'amount' prior to the production, seems to not have been recognized by the program. As someone with little to no prior knowledge of the process beforehand, the output did not tell me much other than what the labels for each value meant, however, an experienced brewer would presumably have a better understanding of this.
Do you have any general comments to add?	The simplicity works in favor of the experience, though it also limits the user-friendliness regarding new users. An introduction manual would be needed to fully comprehend how the interface and other functions work. The missing updates of the home page, or the missing prompt to update seemed to be the biggest bump for the user interface. If its new users were to be targeted, tool-tips when hovering over labels, could help make it more accessible.

Table 4 Feedback questions from the user test

I Optimization calculations

The data that was collected from the machine can be seen in Table **??**. "type" descibes the beer type that was used, "speed" defines the speed that was used for that data point, "amount" the amount of beers that was set to produce, "time" the time in minutes for the batch to finish, "bad count" the amount of defect beers out of the total amount and "good count" the amount of good beers in the batch. "goodCount/min" is the dependent variable that will be plotted, and "speed" is the independent variable. The group find that without the max speed domain for the type, the speed functions are similar. This means that the only type where the speed function is not linear is type 0.
The 20 data points are plotted in Figure 9, where a 3rd degree polynomial is created using regression in Excel. A 3rd degree polynomial is used both because it fits the data well,

and because any higher degree would be unsolvable (or very hard to solve analytically) when finding the roots. To find the max point on this polynomial, we first find the critical point using the differentiated function and finding the roots, and then using the acceleration of the plot to confirm whether the point is a maximum or minimum. The function to differentiate is seen in Equation 1.

$$f(x) = -5.65 \cdot 10^{-6} \cdot x^3 + 3.48 \cdot 10^{-3} \cdot x^2 + 0.387 \cdot x + 25.2 \tag{1}$$

The $R^2 = 0.995$ value means the function describes the data points very well. The derived function is the following: 2 .

$$f'(x) = -1.695 \cdot 10^{-5} \cdot x^2 + 6.96 \cdot 10^{-3} \cdot x + 0.387 \tag{2}$$

The 2nd derivative describes the acceleration and is used to find whether the critical point is a maximum or minimum.

$$f''(x) = -3.39 \cdot 10^{-5} \cdot x - 6.96 \cdot 10^{-3} \tag{3}$$

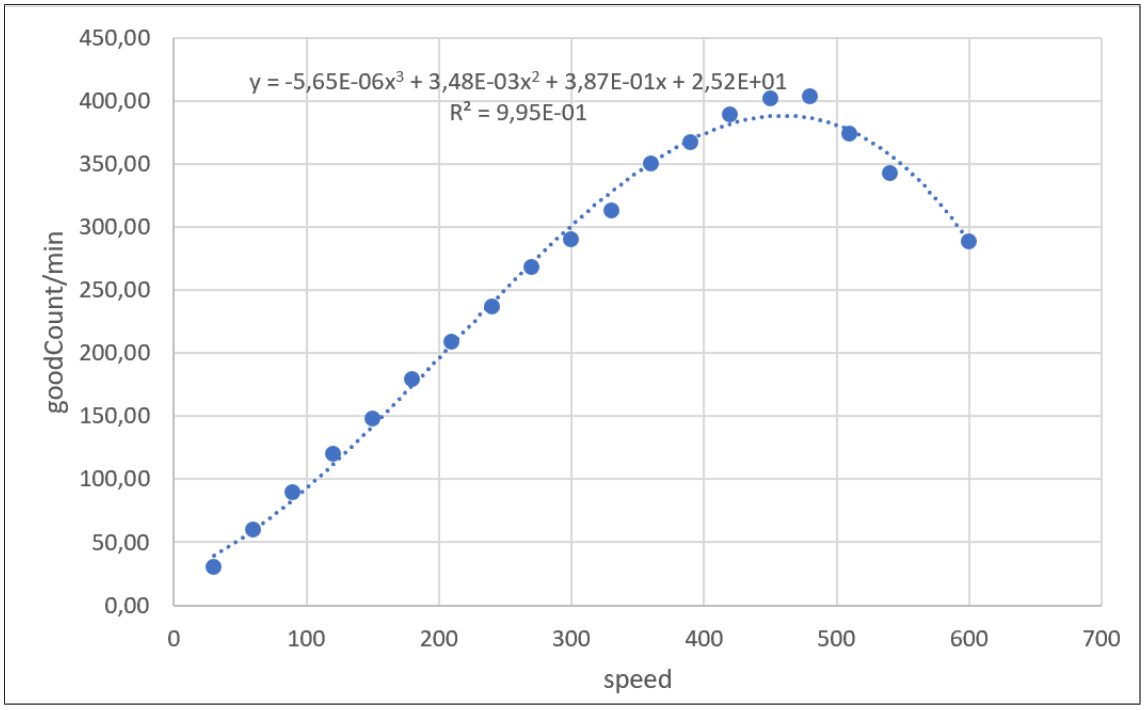

Figure 9 The speed plot based on type 0.

In order to find the critical point, we find the roots of the function Eq. 2. This is done

type	speed [amount/min]	amount	time	badCount	goodCount	goodCount/min
0	30	150	5,00	0	150	30,00
0	60	150	2,50	0	150	60,00
0	90	150	1,67	1	149	89,40
0	120	150	1,25	0	150	120,00
0	150	150	1,00	2	148	148,00
0	180	150	0,83	1	149	178,80
0	210	150	0,71	1	149	208,60
0	240	150	0,62	2	148	236,80
0	270	150	0,56	1	149	268,20
0	300	150	0,50	5	145	290,00
0	330	150	0,45	8	142	312,40
0	360	150	0,42	4	146	350,40
0	390	150	0,38	9	141	366,60
0	420	150	0,36	11	139	389,20
0	450	150	0,33	16	134	402,00
0	480	150	0,31	24	126	403,20
0	510	150	0,29	40	110	374,00
0	540	150	0,28	55	95	342,00
0	600	150	0,25	78	72	288,00

using the Quadratic Equation. We find that the solutions are the following, as seen in Eq. 4

$$x = -50 \ \vee \ x = 460 \tag{4}$$

However the solution has to be in the interval $1 \leq x \leq 600$, thus the critical point is $x = 460$. To verify whether this point is a maximum or minimum, we use the acceleration function, Eq. 3 and insert the x value, see Eq. 5

$$f''(460) = -3.39 \cdot 10^{-5} \cdot 460 - 6.96 \cdot 10^{-3} = -0.0226 \tag{5}$$

Since $f''(460) < 0$ the acceleration is negative, thus it is a maximum point. Since this maximum point is greather than the domain of the speed of the other type-functions, the maximum speed for the other beer types are their respective max values.