



# Belman: Exam Project

Erhvervsakademi SydVest, Computer Science second semester exam project

---

**Featuring the crew from Duolingos Bizarre Adventure (Featuring David from Devil May Cry series):**

Grzegorz Charyszczak

David Kalatzis

Nedas Surkus

**Github repository:** <https://github.com/schemabuoi/Order-Tracking-Application>

---

**2019 June 03**

## Table Of Contents

---

<b>Introduction</b>	<b>3</b>
1.1 Background	3
1.2 Problem Statement	3
1.3 Product Vision	3
1.4 Strategic Analysis	4
<b>2. Scrum Process</b>	<b>7</b>
2.1 Pre Game	7
2.1.1 Project Organization	7
2.1.2 Overall Project Schedule	9
2.1.3 Initial Product Backlog	9
2.1.4 Architecture	12
2.2 Sprint 1	14
2.2.1 Sprint Planning	14
2.2.2 Daily Meetings	15
2.2.3 GUI	15
2.2.4 Data Model	16
2.2.5 Implementation	18
2.2.5.1 Code Examples	18
2.2.5.2 Design Patterns/Principles	21
2.2.6 Sprint Review	22
2.2.7 Sprint Retrospective	22
2.3 Sprint 2	23
2.3.1 Sprint Planning	23
2.3.2 Daily Meetings	23
2.3.3 GUI	23
2.3.4 Data Model	28
2.3.5 Implementation	29
2.3.5.1 Code Examples	30
2.3.5.2 Design Patterns/Principles	31
2.3.6 Sprint Review	31
2.3.7 Sprint Retrospective	32
<b>3. Post-Game</b>	<b>32</b>
<b>4. Conclusion</b>	<b>35</b>
<b>5. References</b>	<b>36</b>
<b>6. Appendices</b>	<b>37</b>

# 1. Introduction

## 1.1 Background

One of the core aspects to maintain any manufacturing facility is correctly tracking the progress and the completion of the orders. A single facility could have anywhere from 1 to 1000+ orders which can cause mistakes and delays during the production. To avoid loss in money and time the workers must be able to focus and maintain said focus for long periods of time on different orders happening at the same time with various amount of progress in each of them. However this could lead to information being unreliable since mistakes are likely to occur which might in the end cost a significant amount of money to fix.

In order to prevent such losses Belman A/S approached the development team to build a system which would be responsible for monitoring and managing the orders. This system should allow observing all of the existing tasks and orders and notifying about new orders. This system would allow for managers to know which orders are completed and which orders are still being processed, if they are delayed or/and behind schedule and it would also give information about which department is currently working on the order. It would help the workers in knowing which tasks they should prioritise on and which tasks are being held up in other departments in addition to allowing them to easily confirm the completion of the orders task in their department. This could potentially save a lot of time and money as well as allow the employees to work in a more stable and less stressful environment.

## 1.2 Problem Statement

With increased number of orders the client faces real risk of the order data going missing, which can result in time being wasted trying to recover that data and significant monetary loss. Furthermore employees are experiencing more stress due to the need to be constantly concentrated on the tasks and not on the work itself which can cause mistakes to be made that can be costly in the long run. Workers currently are also more generally confused as they do not know which department to call when an order is delayed.

## 1.3 Product Vision

In order to ensure the quality of work, a desktop application was envisioned which would give a view on a relevant tasks for all of the departments. The program would automatically display the current orders for the selected department and would operate on separate computers in the manufacturing facility on which any employee could access. The application is intended to be built on a trust system where each employee has to make their own decision when to complete a task without any manager interference. The primary goal of the program was to create a stable, reliable, readable and easily accessible program which would contain only functionality that was specified by the client since our main goal was to create high quality and intuitive application without unnecessary functions - the vision statement during whole development process was quality over quantity.

Using the vision board, the development team came to this product vision:

*“For workers who need to know the concurrent status of the orders in which they are assigned to work on, Order Tracking System (OTS) is an easily accessible and readable application, which guarantees a trustworthy source information for every order, in each department of the company.”*

For full vision board see **Appendix D**

Large amount of order management is often a very confusing and difficult task. Thus there are significant improvements in having an order tracking system which include :

1. **Clear understanding of the order:** If an employee receives an order then they have a clear and concrete understanding of it, with a visible indication of what type of order it is, including information which displays the department and the customer of the order. Thanks to this there is little to no confusion between the workers, which helps in avoiding big losses in terms of money, time and delays.
2. **Prioritising importance:** In some cases orders can be delayed which could potentially be very costly for the company. With that in mind the benefit in having an application which tracks all the orders and their delivery time is that it reduces the chance of a delay and prioritises the tasks which are delayed to be completed first. Thus saving money while also having lower stress potential for the other department workers as they know exactly where the delayed order is stuck and who to call.
3. **Ease of task completion:** Once an order is completed it is possible with a single click of a button to send the task to next department in queue. This helps in saving time and money as previously the employees would have to contact the following department in order to get information of what is happening, and that could cause unnecessary delays. But with a tracking system it means that employees would now be more focused on other tasks and less focused on calls or talks between departments.

## 1.4 Strategic Analysis

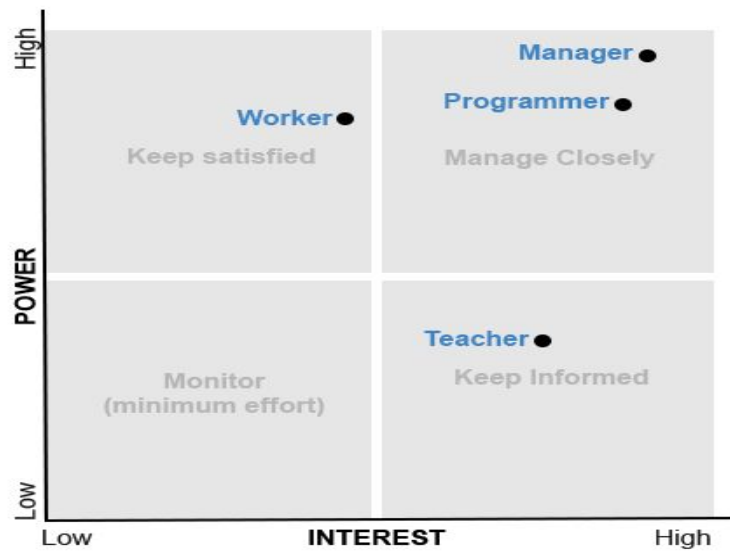
### Stakeholder Analysis

With all that being said, we realized early on the danger that could derive from not assessing risks involved in this project. The highest possible risk could be the mismanagement or planning issues coupled with misunderstanding the goal of the project or what the product owner requires. Lower risks are natural and environmental disasters which have the potential to impact us, however it is very unlikely to happen. Having this in mind we established our stakeholders early on.

- Internal Stakeholders : Programmers
- External Stakeholders: Teachers, Workers, Managers.

A power-vs-interest matrix was drawn (**Figure 1**) in the hopes of creating a clear understanding of where each stakeholder sits in the project.

**Figure 1.** Power-Vs-Interest Stakeholder Matrix



The stakeholders that clearly classified as the most important people to satisfy were the workers. This was decided once it became clear that the main clients and the users will be the workers themselves as their whole workflow will be based around this system. Also it was clear that they are currently using a completely different system than the one that is going to be developed, with regards to this fact it became evident that one of the main priorities should be ensuring that it is easy to grasp and use the program for any worker.

Other stakeholders do play a big role and are integral to the system itself, however the managers are identified with the most power over the future of this system.

### Risk Analysis

A risk analysis was also conducted along side the stakeholder analysis. This can be seen at **Table 1** which also shows the high risks of our project. These risks include the loss of our project's data, the loss of the client and the risk of wasting too much valuable time in one task because of team mismanagement.

**Table 1.** Risk Analysis

Risk	Risk management
Human:	
Illness	We accept the risk and manage it - In the event of this happening, the team member has to notify the team when possible, in the case of severe sickness, another member will replace their work, if instead it is nothing serious, they can work from home.
Injury	We accept the risk and manage it - If the injury of the team member is not severe, they can work along the team from their

	home. However it is serious, another member will replace the lost team members work.
Team communication	We accept the risk and try to limit it - We limit it with the use of daily scrum meetings, and also by behaving in a democratic way, meaning that we hear both sides of the argument and decide on which is a better choice together as a team.
Technical:	
Loss of project data	We accept the risk and try to avoid it - In order to avoid such a big risk we take on to using control systems such as git and Github.
Reputational:	
Loss of client	We accept the risk but have no control over it - In the case of something like this happening, we have no means of preventing it as it is possible for the client to cancel our partnership for any reason. However we try to avoid this, by having a clear and well rounded communication with the project owner and provide the best possible quality of code and product we can.
Project:	
Taking too much time doing a task	We accept the risk and manage it - By planning ahead and knowing fully the capabilities of each team member. As well as being fully aware of each team members capabilities we are able to assign the work as best as possible to prevent such issue from occurring.
Code quality	We accept the risk and manage it - To avoid this, we follow basic programming conventions in order to make our code as cohesive as possible. Furthermore we also include some of the structures and patterns we learned from our studies in order to meet the standards of what is expected nowadays from software.
QA quality	We accept the risk and manage it - Even though we do not always have access to the product owner to gather feedback for our project, we use other students and teachers in order to test out our project and get some user feedback.
Politics:	
Changes to government laws	We accept the risk but have no control over it - While it is unlikely to occur we are careful to be informed in case something that could impact our work happens.

Green: low risk, Yellow: intermediate risk, Red: high risk.

## 2. Scrum Process

In order to manage the development of the project, the agile development framework “scrum” was chosen as it is the most widely used software development method in the industry and is the method we learned to use during the second semester’s lectures. It enables us to provide a quick and efficient feedback loop in case any changes need to be made, while it also addresses any complex adaptive problems we might encounter which in turn allows us to achieve the most effective team collaboration possible.

During this project the team was given 5 weeks to build the application. This provided the team with a total of two sprint reviews and an extra week for project report writing and bug fixing and any minor development changes.

In the following section we will discuss in detail how the pre game was strategized and planned shortly after the kick-off meeting during the first week of the project. In addition to this there are drafts and sketches for the initial GUI design, initial backlog and epic stories, the development schedule and the project's general architecture.

### 2.1 Pre Game

One of the key tasks of any project is the establishment and definition of the scope of the product or project. This means the creation of a list of backlog items and epic stories, decisions regarding the architecture of the system while understanding that there could be changes in the future which could impact the backlogs and the architecture , and in turn modify the scope of the product or project.

These pre game decisions were made during week 17 when all functional and nonfunctional requirements for the project became clear. Having them in mind an initial backlog was established and project roles were assigned. This and the architecture of the project will be discussed below.

#### 2.1.1 Project Organization

##### **Roles**

The development team consisted of three members which not only exhibited highly flexible and adaptive skill sets which were proven to be perfect for agile methodology but also had advanced skill sets in database, GUI and structure design which allowed most of the compulsory project tasks to be completed fairly easily.

With this in mind no specific roles were assigned in the project, instead the team relied more on trust and communication in tackling specific tasks. Each team member could choose freely which kind of task or tasks they will work on and most of the tasks would be chosen during the morning scrum meetings. However if any conflict arose in the selection of tasks - the scrum master had the right to decide who would do this specific task.

The table below shows (**Table 1**) all the roles and their attached responsibilities.

**Table 1.** Scrum Team Roles and Responsibilities

Role	Responsibilities
<b>Product Owner</b> Morten Kristensen (Representative of Belman) Jeppe Moritz Led (Due to the nature of the project. The representative was not always available to be contacted. Whenever a clarification of the task was necessary during the project. The team would contact the teacher instead)	<ul style="list-style-type: none"><li>• Create product vision</li><li>• Convey vision to the team</li><li>• Be available to their team</li><li>• Communicate well with the team</li><li>• Give correct feedback</li></ul>
<b>Scrum Master</b> David Kalatzis	<ul style="list-style-type: none"><li>• Establish a good work environment</li><li>• Ensure a positive relationship amongst the team</li><li>• Provide an in between communication for the team and product owner</li><li>• Protect the team from outside distractions and interruptions</li><li>• Confirm that the team and project adhere to the scrum methodology and daily meetings</li><li>• Correctly prioritize the backlog items</li></ul>
<b>Team Members</b> Grzegorz Charyszczak David Kalatzis Nedas Surkus	<ul style="list-style-type: none"><li>• Developing and delivering the software</li><li>• Pick prioritized backlog items from the list in order to work on them</li><li>• Providing precise estimations of time needed on a task</li><li>• Perform the daily scrum meetings</li></ul>

In addition to this, a working agreement was made with the approval of the whole group so each team member understood the rules of communication and what is required from them. This can be found in **Appendix A**.

### Software Used

During the initial planning of the project it was agreed upon to use software that the team would be familiar with and have a level of comfort working with. This was done to ensure a smooth overall development process. The software chosen for this project is shown in the list below.

- Discord: For project communication.
- Messenger: For general discussions on problems.



- Trello: For creating an easy to reach and understandable backlog and to keep more detailed information about our project.
- Git: For storing the source code.
- Google Docs and Scrumwise: For administrative processes.
- Lucid Charts: For creating the UML diagrams.
- Microsoft SQL server manager : For creating an SQL database.
- Netbeans : For developing Java code.

### 2.1.2 Overall Project Schedule

The schedule can be found in Appendix B.

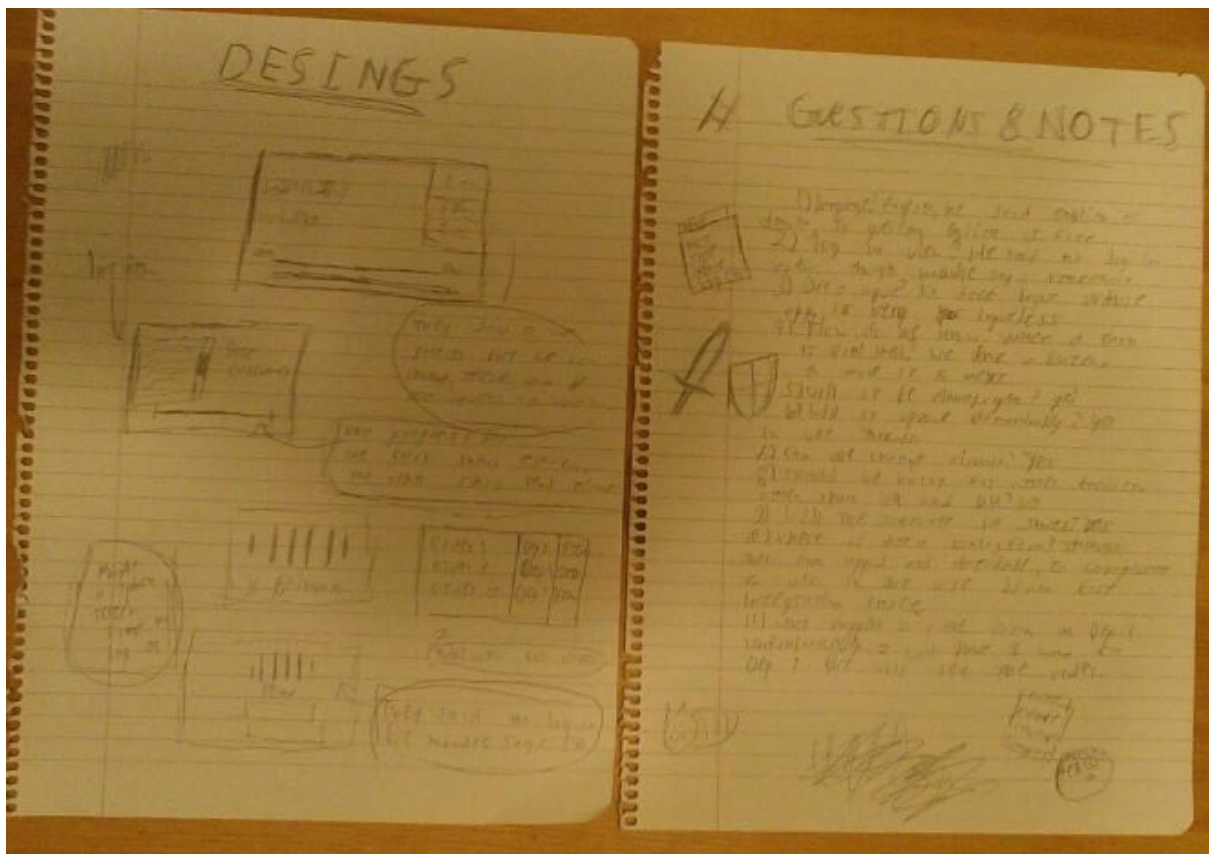
### 2.1.3 Initial Product Backlog

During the kick-off meeting on 23th of April the team created an initial backlog of things that needed to be implemented and prioritised during the project. This prioritisation was done by what the client emphasised most during said meeting.

In addition to this the team members took notes and wrote down the answers to any questions that came up during the meeting. During this time a couple of simple design ideas were done with the information on hand. This was done in order to populate scrumwise backlog in the future.

The sketches and notes can be seen in **Figure 1**.

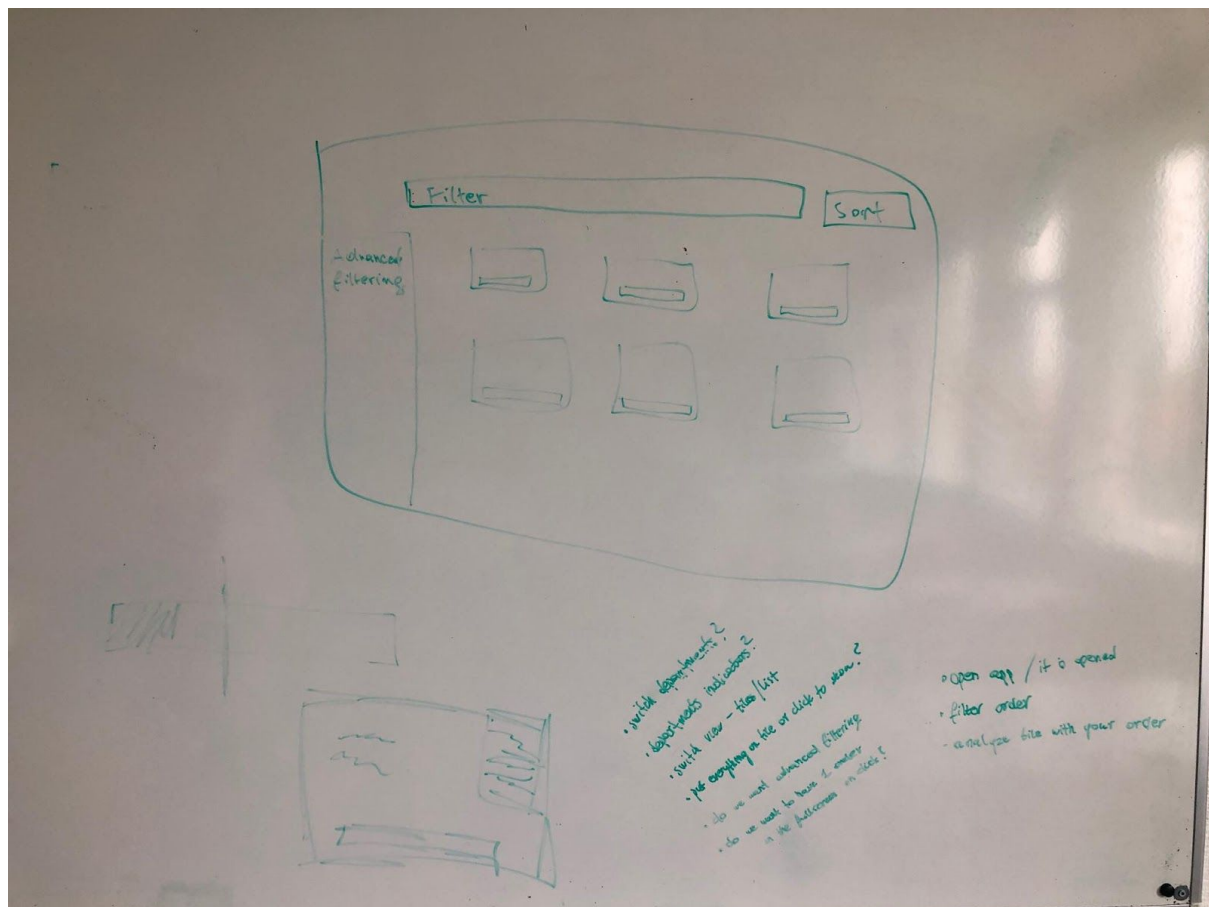
**Figure 1.** Quick sketches and notes taken during first meeting.



The initial backlog included what the team members were thinking at the time, some after meeting ideas were written on the board and a sketch was made to visualise the project better and the tasks that will be associated with it (**Figure 2**). The tasks included the compulsory functions as well as optional ideas and tasks that could be done in order to improve the overall quality of the system.

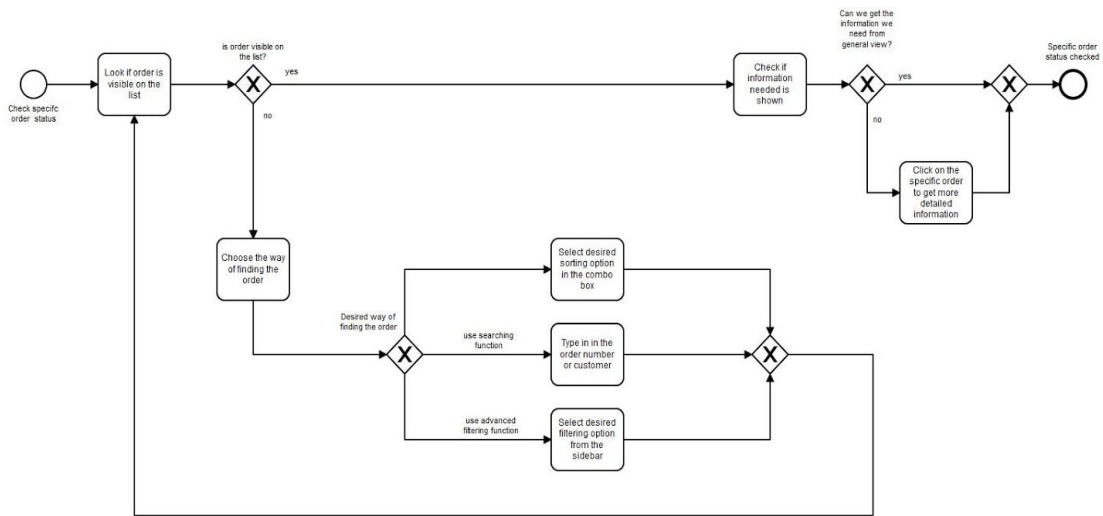
Upon completion of this visualisation the team considered all the ideas in great detail and decided on the following initial backlog of epic stories (**Figure 3**) and user stories (**Figure 4**).

**Figure 2.** Initial Backlog on the Board

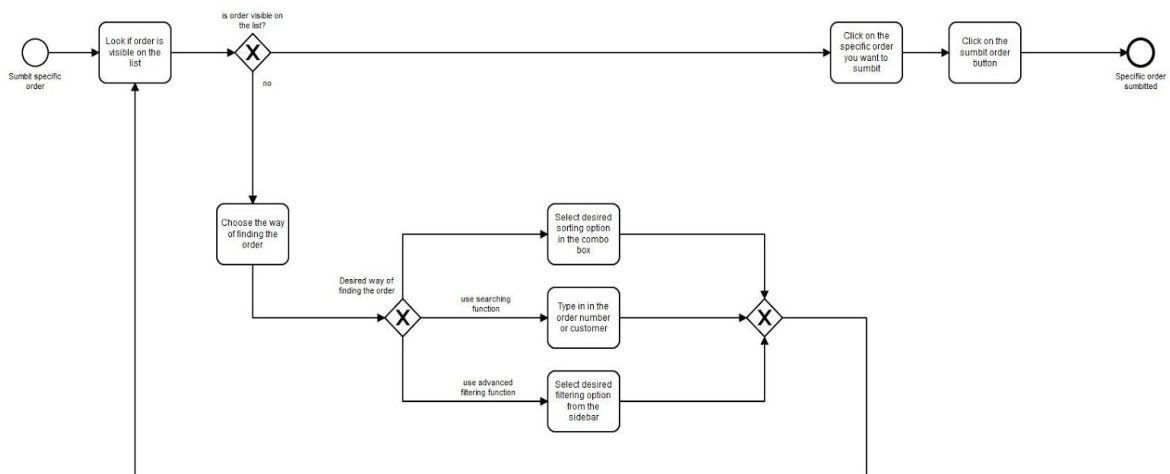


**Figure 3. Epic Stories In The Initial Backlog**

**-Check specific order status**



**-Submit specific order**



**Figure 4.** User Stories In The Initial ScrumWise Backlog

🔴 As a manager I want to see a list of all the orders that my department should work on in order to know which orders my department is working on right now.	48 hours	Assigned to sprint	🟡🟡🟡🟡🟡🟡🟡🟡
🔴 As a manager I want to click on a specific order, in order to see it's detailed information.	New		🟡🟡🟡🟡🟡🟡🟡🟡
🔴 As a manager I want to submit the task of the order for which am responsible in as finished, in order to pass it to the next department	New		🟡🟡🟡🟡🟡🟡🟡🟡
➡ As a manager I want to search a specific order, in order to check its status	New		🟡🟡🟡🟡🟡🟡🟡🟡
➡ As a manager I want to sort the orders in order to prioritize work for my department	New		🟡🟡🟡🟡🟡🟡🟡🟡
⬇ As a manager I want to filter the orders in order to see on the screen only orders that are relevant	New		🟡🟡🟡🟡🟡🟡🟡🟡

### 2.1.4 Architecture

After the creation of the backlog, an initial UML diagram was drawn (**Figure 5**) in order to assess any potential problems that will be encountered in the development of the application. This was done in order to ensure a clearer vision of the overall architecture and design of the application. The team decided on multiple patterns that would be necessary to use in order to ensure the quality of the product.

Firstly, the team decided to use a three layered architecture. It was picked by the team due to the amount of experience the team had with the pattern as it was well as the benefits of using it suited the projects. The architecture itself granted the team to have great flexibility between the three layers (GUI , Business logic, Data Storage) thus allowing the team to have lower coupling between layers and making them easier to change independently.

Secondly, the team has picked MVC (Model, View , Controller) architecture - as it is commonly used for creating user interfaces and it is supported by software platform we would be using (JavaFX). This pattern allowed the team to ensure less coupling between classes and higher cohesion in the application once combined with the three layer architecture.

Thirdly, the team chose to use dependency injection in the main classes of the program as it allowed the team to adhere to the SOLID principles and receive all the benefits that come along side it such as : reducing the frequency in which the team had to change a class, improved reusability and maintainability of the code, decreased the complexity of the code and reduced frequency of the need to change a class once the dependencies between particular classes were reduced.

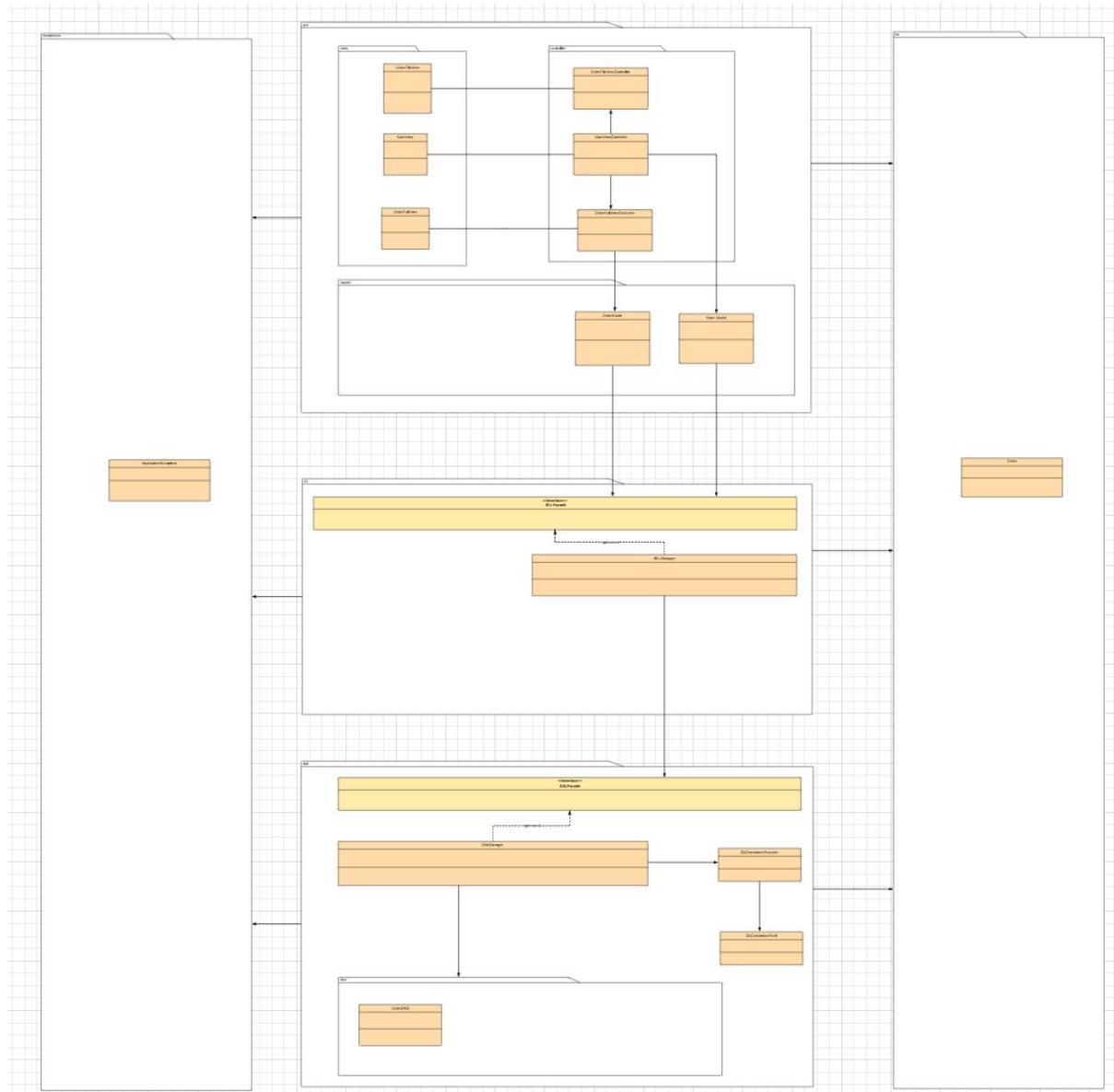
Fourthly, the facade pattern was utilized in every layer to decouple layers from each other and to increase the unification and abstraction of the every layer in the application. This is achieved by defining an entry point to every layer and allowing communication only through these facades. If done in this specific way , the Facade pattern is able to hide the complexities of the system from the client.

Factory pattern was also chosen to be used for the benefit of easily switching the components inside the layer with minimal impact to any other layers. It also combines well

with dependency inversion we adapted to the facades and an added benefit is that factory pattern helps encapsulate object creation.

Lastly, Object pool pattern was chosen to be used for database connections in the application. This was done as each connection to the database is expensive in terms of time and resource expenditure, thus considering the client specification requires a large amount of connections to be made in a short period of time, this pattern allows the application to be faster and more efficient in regards to the expenditure of connections.

**Figure 5.** Initial UML diagram



## 2.2 Sprint 1

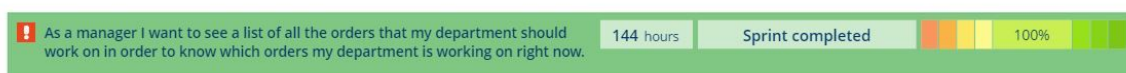
### 2.2.1 Sprint Planning

After taking into consideration everything from initial backlog the team decided they should use a single backlog item as it included the main core functionality of the entire program (**Figure 1**). The core functionality was so extensive that it was essential to all of the other systems in the program. Keeping this in mind each task was estimated and each task was timed after a discussion with a team member appointed to it. Time estimation was not difficult as the group had a good general idea on how much time each task would take to be completed. The main goal of this sprint was to create the base of the system structure on which the second sprint will seek to build most of the applications functionality.

The prototyping in this phase of the sprint was not made difficult. A GUI mockup was created (**Figure 2**) and team members were assigned to work on specific parts of the system. The team members would often try to envision problems down the road and after a brief discussion with fellow team members the team would find the best solution in order to avoid the problems.

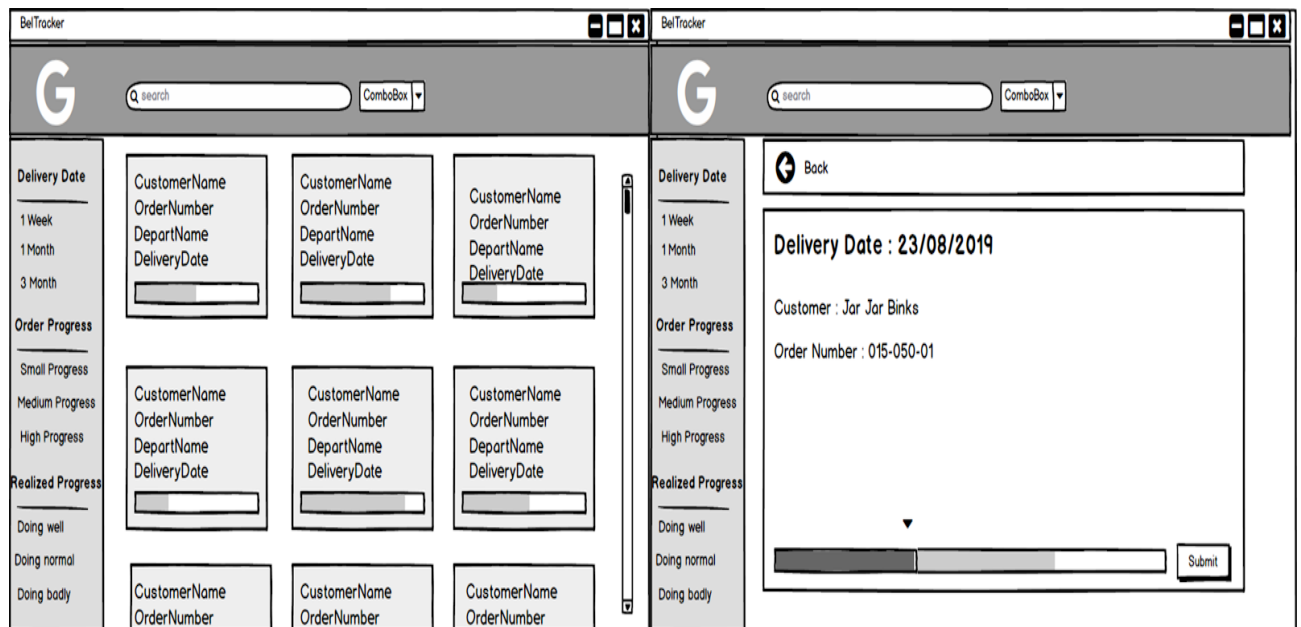
The GUI mockup included only two views : the main view on which the tasks will be displayed and a secondary view for more detailed informations for a specific task. The team decided to focus on polishing these views as the system should not only look professional and modern to be used in a corporate environment each day, but also fast and responsive to decrease frustration for the workers who would use this app.

**Figure 1.** First sprint user story





**Figure 2.** Screenshots of the GUI Prototype



### 2.2.2 Daily Meetings

At the start of each day the team would stand up for roughly 15 minutes and would take turns discussing what task each team member had completed during the previous day, if any problems had arose, the team would try to respect each order by not talking over another team member or rambling as that would take too much time from the daily meeting. This was a clear way of knowing where the team was in comparison with their initial planning. Another benefit for this was the ease of assigning of tasks, and if needed help a team member on any given task.

The daily meetings allowed for the team to be flexible and easily adaptable to any new changes that could arise from day to day, and with everyone having a chance to discuss any major or minor changes that needed to be made, this lead to a more cooperative and understanding atmosphere.

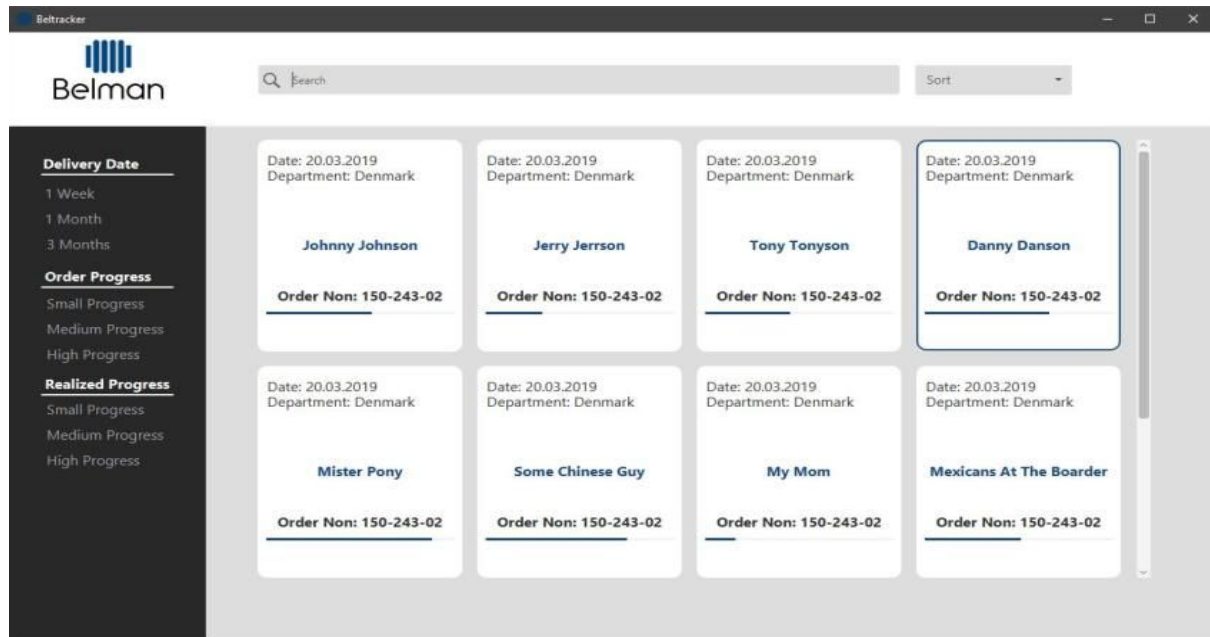
### 2.2.3 GUI

Early on the design of the application was intended to be inspired from Belman's own website. This was done intentionally to match any software the company would have and to allow the companies software structure to have some coherence with an extra benefit of creating a familiar design which workers could adapt easier to. In order to achieve such concept we used a similar color palette, font and placement of elements in the structure. The initial GUI design of the main view can be seen below (**Figure 3**) .

Another core aspect was the ability to easily telegraph key information to the user without extra unnecessary or unimportant information clouding the view. This was done with the intention of simplifying the application in order to promote quicker phase for the worker as the application would be used on a daily basis. The reasoning behind this stems from the

idea that the user would get frustrated and stressed for having to navigate a complicated GUI each time they want to do simple tasks.

**Figure 3.** Initial Main View Design

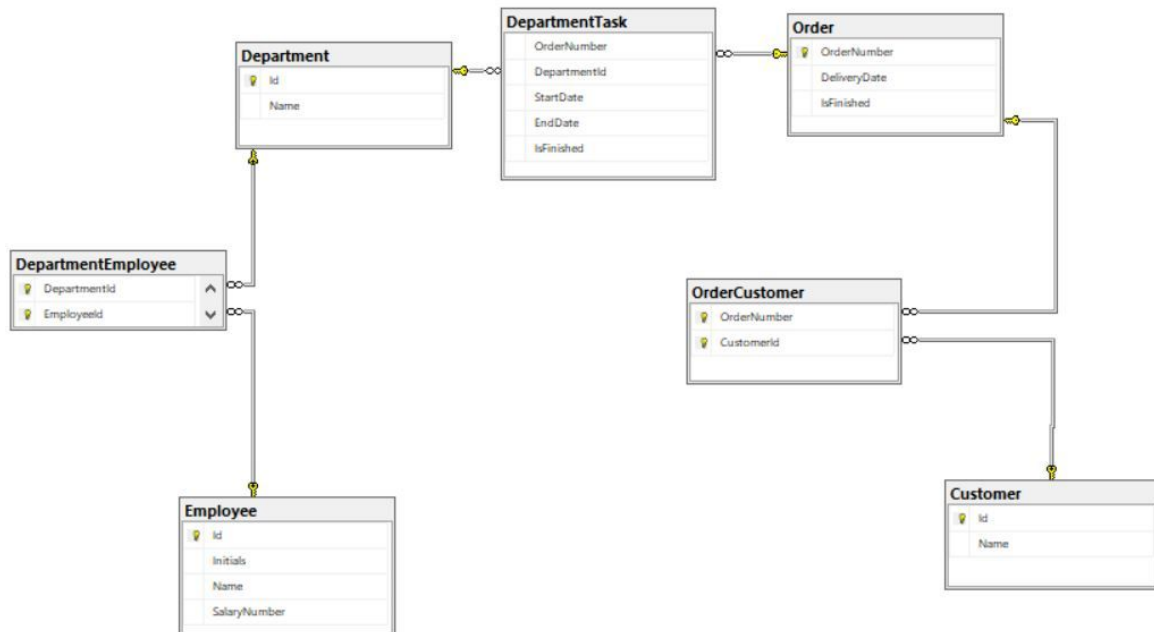


## 2.2.4 Data Model

The first version of the database was built to store all the information gathered from a JSON text file. The structure of the database itself was made in such a way that changes could be easily implemented and would impact other tables on a very minimal level.



**Figure 4.** Initial Database Table Design



The employee table contains all the information on the employee and is connected to the department by DepartmentEmployee table. Department table stores all the department information and is connected to the employee table via a joint table called DepartmentEmployee which contains the employee id and the department id. The order table encompasses all information about the orders and connects to the customer table in the same method like department to employee.

Taking into consideration that orders can have multiple tasks inside of them which will be used by specific departments, a DepartmentTask table is created in order to store all the department task data. To keep consistency for which task belongs to which department and order, DepartmentTask table relies on foreign keys from Order and Department.

## 2.2.5 Implementation

After the initial planning phase was completed, the team began their work on the structure of the program. This meant the creation of the three layer architecture, implementation of MVC pattern, making and populating database with mock data, establishing database connection and beginning work on implementing what was planned.

During this sprint, the team's main goal was the implementation of the main functionality of the program. This would mean that the team focused on selecting an individual department and getting list of tasks for this department and displaying it on the main view of the application.

Another feature the team had focus on was the implementation of the JSON reader to populate the database with real data. However due to time constraints, the JSON reader was not merged into the program before the end of the first sprint review.

Logging system was implemented during this sprint. This logging system is based on log4j library. When an error is detected in our program it is written in the BelTracker.log file which is responsible for keeping track of all errors made in the program. The program itself can store up to 5x10 MB worth of logs. Once this amount is surpassed, the program starts overriding existing logs starting from the oldest inserted data.

The team also focused on implementing planned design patterns during this sprint. More details on implemented design patterns can be found in **2.2.5.2** section.

### 2.2.5.1 Code Examples

By reading the companies presentation on the project the team deducted that a department selection system was necessary to implement. This would allow an easy set up on many computers in the same or different departments. The team implemented a method used for selecting every department from the department database table and using this method a combo box is populated which allows the combo box to contain any number of departments.

**Figure 5.** Department loading code example

```
public void loadDepartmentInformations()
{
    Department selectedDepartment = model.loadDepartment();
    if(selectedDepartment != null)
    {
        displayLoadingView(selectedDepartment);
    }
    else
    {
        showStartConfigurationButton();
    }
}
```

**Figure 6.** Loading all departments from database into combo box

```
private void showStartConfigurationButton()
{
    cmbDepartments.setItems(FXCollections.observableArrayList(model.getAllDepartments()));
    btnStartConfirm.setVisible(true);
    lblStartConfirm.setVisible(true);
}
```

When selecting a department from the combobox, the information of the department which was selected is stored in a property file.

**Figure 7.** Department property file design

```
DepartmentName=Halvfab
DepartmentID=2
```

**Figure 8.** Saving selected department into properties file

```
@Override
public void saveDepartment(Department department) {
    try(OutputStream output = new FileOutputStream(DEPARTMENT_PROPERTIES_FILE))
    {
        Properties departmentProperties = new Properties();
        departmentProperties.setProperty("DepartmentID", Integer.toString(department.getId()));
        departmentProperties.setProperty("DepartmentName", department.getName());
        departmentProperties.store(output, null);
    }
    catch(IOException ex)
    {
        //TO DO
    }
}
```

The configuration view for selecting the department will be shown only once. Once a department has been chosen and the application is set up, application will automatically load the department which was chosen during the initial department configuration.

**Figure 9.** Loading previously selected department and setting up the scene.

```
@Override
public Department loadDepartment() {
    try(InputStream input = new FileInputStream(DEPARTMENT_PROPERTIES_FILE))
    {
        Properties departmentProperties = new Properties();
        departmentProperties.load(input);
        String departmentIdAsString = departmentProperties.getProperty("DepartmentID", null);
        String departmentName = departmentProperties.getProperty("DepartmentName", null);
        if(departmentIdAsString != null && departmentName != null)
        {
            int departmentId = Integer.parseInt(departmentIdAsString);
            return new Department(departmentId, departmentName);
        }
        else
        {
            return null;
        }
    }
    catch(NumberFormatException ex)
    {
        //TO DO
        return null;
    }
    catch(IOException ex)
    {
        //TO DO
        return null;
    }
}
```

#### 2.2.5.2 Design Patterns/Principles

First pattern to be implemented was the facade pattern working as an entry points for our layers. It was done to decouple layers from each other and make them change - independent.

Next pattern we implemented was factory method pattern combined with the dependency inversion which main benefit is the ease of switching particular parts of the program without affecting other parts - low coupling. This combined with the previously mentioned facade pattern helped the team a lot during development of all of the layers - this allowed the team to develop DAL and JSON reader without urgency as data needed for GUI development could be generated in MockDALManager. To reduce coupling between the parts of the program as much as possible the team also used dependency injection on all facades and main components of the application.

Lastly, the final pattern to be developed during the sprint was connection pooling. This was done to give us the ability during the next sprint to implement efficient version of observer pattern in order to refresh the database each 3 seconds as needed in the mandatory requirements.

### 2.2.6 Sprint Review

The first sprint review provided some valuable feedback for the development team. This feedback was then discussed and implemented in the second sprint.

This included:

- Minor changes to the GUI loadout.
- Clarifications on filtering and sorting orders
- Changes to the color palette.
- Inclusion of CSV and Excel readers to read different type of data files.

Despite this feedback the team still prioritised the core functionality more in the second sprint and once the main functionality was completely implemented and tested only then was this feedback applied.

### 2.2.7 Sprint Retrospective

Once the sprint review was finished the team conducted a sprint retrospective (**Table 1**). In the retrospective the team included what went wrong, what went well and which parts could be improved in the next sprint.

**Table 1.** Sprint Retrospective

What went well in this sprint?
Having a sprint plan which allowed us to be more efficient with our time while presenting.
Screenshots were used for examples and questions.
We have a good schedule which allows our priorities to be accurately listed.
The presentation had a more formal look to it.
What went wrong in this sprint?
The user story was too big and was not fully finished .
Time estimation as not every member of the team was on the same page and fully prepared.
Showcasing user stories which were not completed.
The presentation itself was slightly unorganised.
What can be improved?
Having a paper with notes instead of having them on another computer.
Positioning in the room during the presentation.
Scrums clarity which would allow for less misinterpretation by the product owner.

## 2.3 Sprint 2

### 2.3.1 Sprint Planning

In the first sprint the structure was established, thus the team decided to shift their focus from structure to functionality during the second sprint. Suggestions made during the first sprint review were taken into consideration and were implemented at the end of the second sprint when most of the core functionality was completed. The criticism about the user story being too big was also taken into account, with this in mind the team created smaller user stories with more precise descriptions (**Figure 10**).

**Figure 10.** Second sprint user stories

As a manager I want my application to always show me the updated state of all the tasks in order to be instantly notified about all the changes that happen	37 hours	Sprint completed	100%
As a manager I want to click on a specific task, in order to see it's detailed information.	55 hours	Sprint completed	100%
As a manager I want to submit the task for which am responsible in as finished, in order to pass it to the next department	26 hours	Sprint completed	100%
As a manager I want to search a specific task, in order to check its status	9 hours	Sprint completed	100%
As a manager I want to sort the tasks in order to prioritize work for my department	15 hours	Sprint completed	100%

### 2.3.2 Daily Meetings

The daily meeting structure did not change during the second sprint as the team found it very helpful in dealing with problems. Despite this, at the end of the second sprint, conflicts would arise due to stress and miss communication. However they were quickly solved after any miscommunication was cleared up.

The biggest conflict was about merging the database work with the rest of the project due to conflicting merging methodics of both team members. Nonetheless this was cleared out by allowing the first team member to merge first and the second team member to refactor it when he has the time.

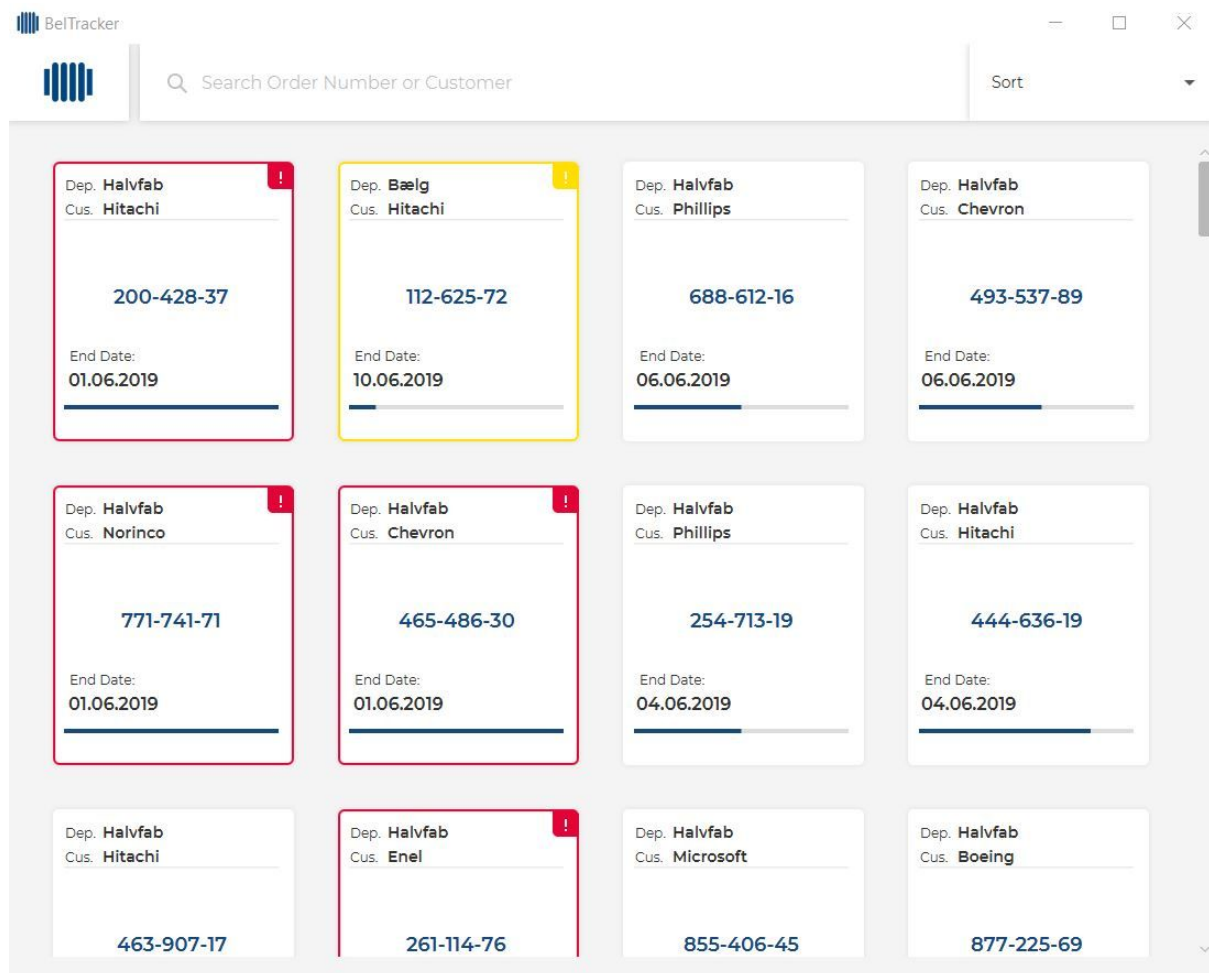
### 2.3.3 GUI

During the second sprint the project GUI changed a lot in comparison with first sprint. As with client request minor changes to the color palette were done. With that change more functionality was added. Functionality such as tasks now showing in case that they are delayed in a previous department, tasks showing If the current department is late on an task or is the task progressing correctly.

Another secondary view was added which overlays the main view when an order is selected. It shows more detailed information of the order along side a warning message if the order is not progressing correctly. This new iteration of the design can be found in **Figure 11**.

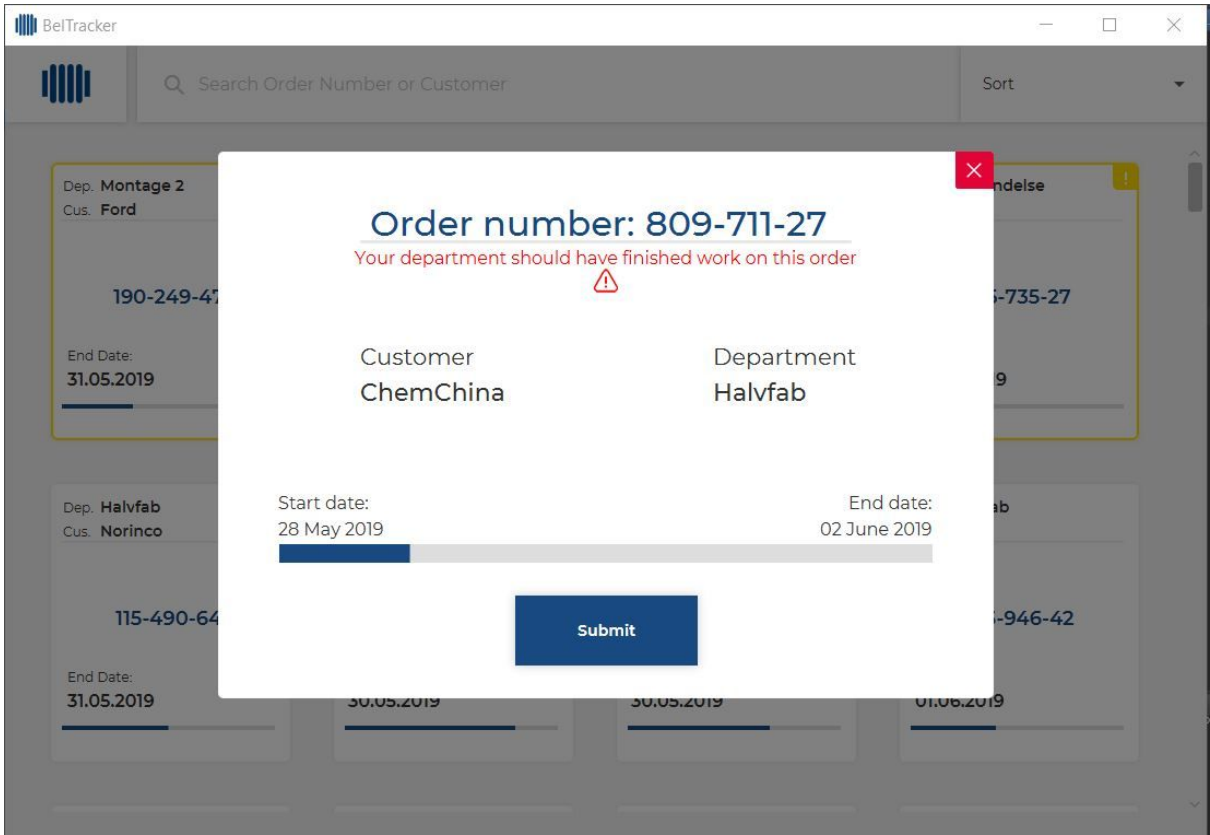
The color palette was changed so that the red color signifies that the task is delayed in the current department (**Figure 12**), yellow color signifies that the task is delayed in a previous department (**Figure 13**), while a task without a border color signifies that the task is waiting to be processed and is on time (**Figure 14**).

**Figure 11.** Second iteration of the GUI design

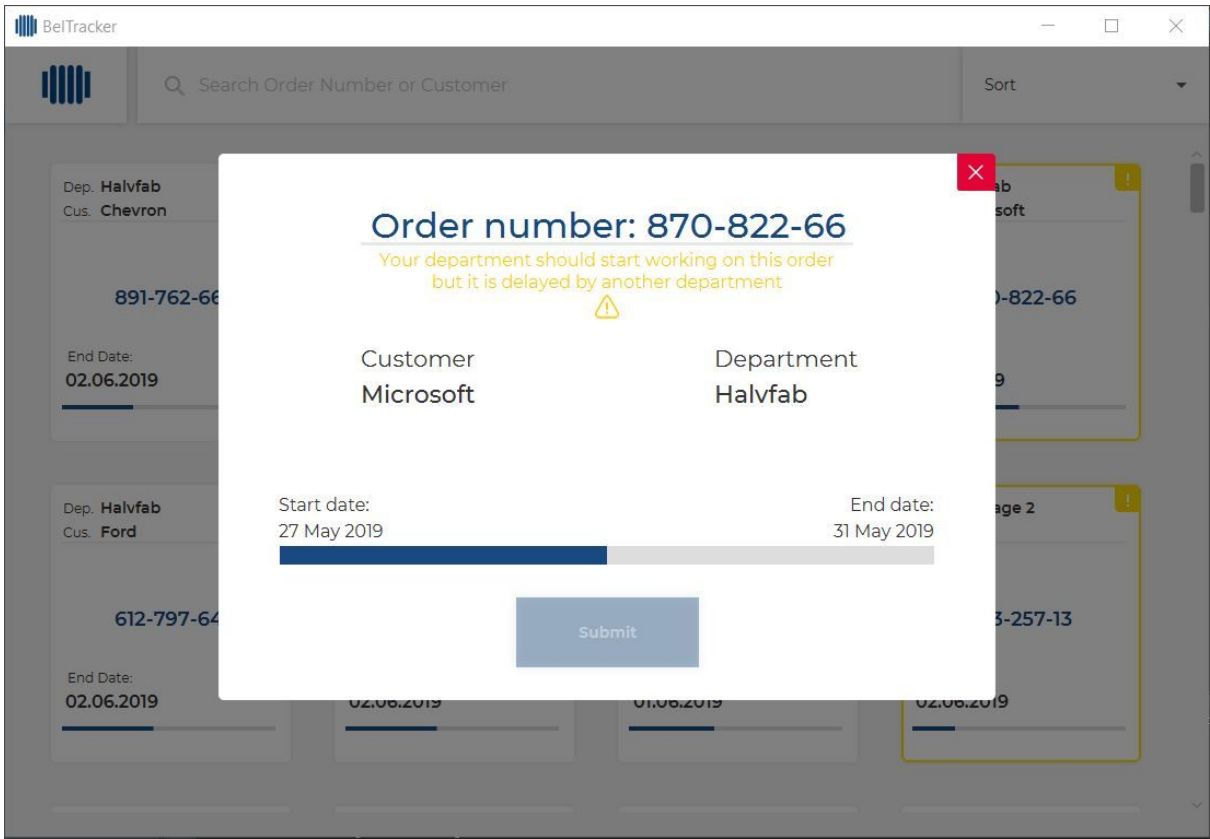




**Figure 12.** Detailed information on an order delayed in a current department



**Figure 13.** Detailed information on a delayed order in a previous department



**Figure 14.** Detailed information on a non delayed order

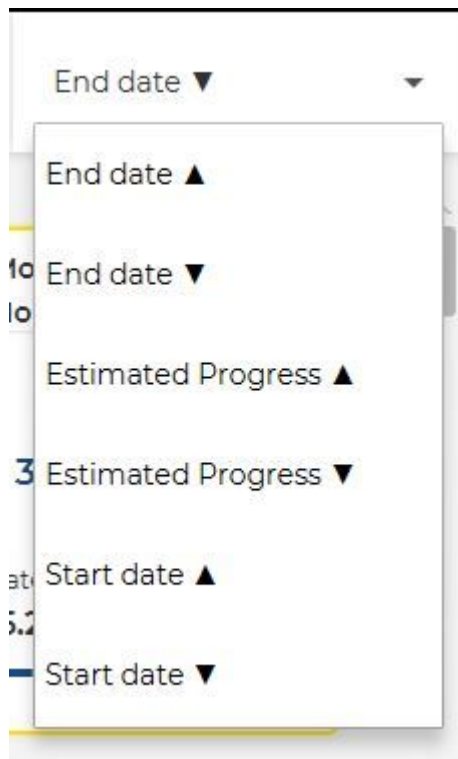
The screenshot shows the BelTracker application interface. A modal window is open, displaying the following information:

- Order number:** 498-609-85
- Customer:** Ford
- Department:** Halvfab
- Start date:** 22 May 2019
- End date:** 28 May 2019

A blue **Submit** button is located at the bottom of the modal. The background shows a list of orders with columns for Department (Dep.), Customer (Cus.), Order Number, and End Date. The first visible order is for Halvfab, Ford, with order number 498-609-85 and end date 28.05.2019.

Another feature that has been added was sorting. On click in the sort bar it outputs a drop down menu where you can select different types of sorting (**Figure 15**). This feature is implemented in such a way to allow expansion later on as custom sorting options can be added provided the methods for sorting them are created.

**Figure 15.** List of sorting options



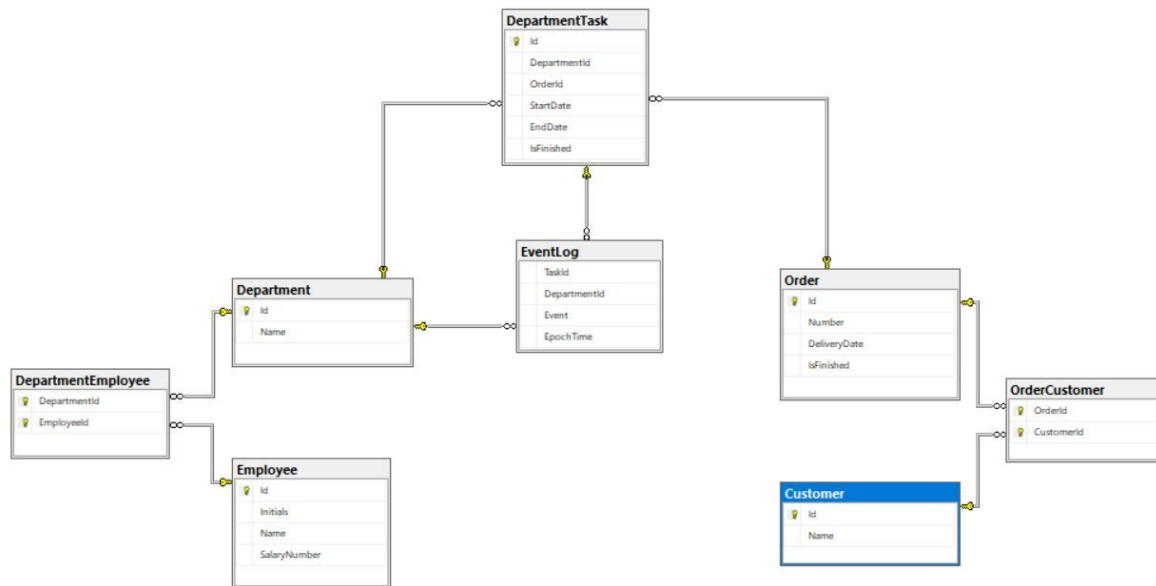
#### 2.3.4 Data Model

During the second sprint review another version of the database design was created. This version can be seen in **Figure 16**. In this second iteration a new table was added, called EventLog which would contain every event that happens to an order. For example an order with an order number “1521-2156” gets completed. The event of the order’s completion is then is inserted into the EventLog database table.

Another change that was implemented was the inclusion of ID in the Order table as it is faster when joining a table, and allows the ID to become the primary key instead of order number. Having this in mind the DepartmentTask and also OrderCustomer tables were changed to now be jointed on ID instead of the order number.

Some minor changes were also implemented, relating to general clarity of the database. These changes were mostly renaming certain columns and table names in order to be more specific and clear.

**Figure 16.** Second iteration of the database design



### 2.3.5 Implementation

During this sprint, the team's main goal was to build upon the functionality on the already established structure from the first sprint. With this in mind the team not only implemented an additional pattern in order to complete the compulsory requirements (See **2.3.5.2**) but also implemented several features that were critical for the client.

One of such feature was the submission of tasks, this allowed for the worker to complete a task in a given department, once completed the task could start to be worked on in the following department.

Another feature developed was made so that when an user clicked a task, it would display another view which includes more detailed information of the task. This feature is talked more about in the GUI section (See **2.3.4**)

Multiple readers were also added in this sprint. This was done in case any different data formats would be inserted so the program will be able to not only read JSON format files but also CSV and Excel data formats.

Multi-threading was implemented into the system as well during this sprint. It was used to observe the database using the observer pattern (You can read more about this pattern in **2.3.5.2**). Alongside the usage in the observing database, multi-threading was also used for detecting new files and mapping the data to the database. These two observers combined with each other allowed the team to have constantly updated data in the application whenever there are changes in the database.

During the second sprint the data structure was redeveloped and improved upon by adding more functionality that consists of a DataTransfer class, data converters and the class for watching the folder for new files. All of these classes are managed and combined by the DALManager. When a file with new data is added to the folder called newData, the

FolderWatcher detects it and notifies the DALManager about this specific file using the observer pattern. Then DALManager depending on the type of the file starts converting its data to the DataTransfer class using the corresponding converter. After converting the data from the file to the DataTransfer class, the DataTransfer is passed to the data access object which adds new data to the database.

After new data is fully mapped, the file is moved to the InsertedData folder. In case something went wrong during the process, the file would be moved to the InvalidData folder.

The last features to be developed were the searching and sorting options. By typing on the search bar, the client could effectively search by either order number or customer name. As for the sorting functionality, by clicking on the dropdown menu, the client could sort the program in many different ways. For example by delivery date in ascending order, or by department name in descending order.

### 2.3.5.1 Code Examples

One of the features that were implemented during the second sprint was the sorting functionality. This was done by selecting a sorting option from a dropdown menu which then is passed from the GUI to BLL. In the BLL there is a switch statement which controls which method is used for sorting (**Figure 17**).

**Figure 17.** Selecting methods in BLL.

```
@Override
public List<Task> sortTasks(List<Task> tasks, SortingType type) {
    switch(type)
    {
        case END_DATE_ASC:
            tasks.sort(new TaskEndDateComparator(TaskEndDateComparator.Type.ASC));
            break;

        case END_DATE_DESC:
            tasks.sort(new TaskEndDateComparator(TaskEndDateComparator.Type.DESC));
            break;

        case ESTIMATED_PROGRESS_ASC:
            tasks.sort(new TaskEstimatedProgressComparator(TaskEstimatedProgressComparator.Type.ASC));
            break;

        case ESTIMATED_PROGRESS_DESC:
            tasks.sort(new TaskEstimatedProgressComparator(TaskEstimatedProgressComparator.Type.DESC));
            break;

        case START_DATE_ASC:
            tasks.sort(new TaskStartDateComparator(TaskStartDateComparator.Type.ASC));
            break;

        case START_DATE_DESC:
            tasks.sort(new TaskStartDateComparator(TaskStartDateComparator.Type.DESC));
            break;

    }
    return tasks;
}
```

Once the method is selected, the class then decides in which kind of order to display the tasks, it then overrides the compare method and sets up the list. An example class can be found in **Figure 18**, Such class can be copied and fitted to with custom comparing.

**Figure 18.** Selecting by end date method.

```
public enum Type {
    ASC, DESC
}

private Type type;

public TaskEndDateComparator(Type type)
{
    this.type = type;
}

@Override
public int compare(Task t1, Task t2) {
    if(type == Type.ASC)
    {
        return t1.getEndDate().compareTo(t2.getEndDate());
    }
    else
    {
        return t2.getEndDate().compareTo(t1.getEndDate());
    }
}
```

#### 2.3.5.2 Design Patterns/Principles

In the duration of this sprint the only pattern to be implemented was the observer pattern. We used observer pattern in two places. Firstly for observing database for changes and secondly for watching the folder and detecting new data files. It was useful in allowing the application to refresh data every 3 seconds to correspond to database changes. This granted the team the ability to have constantly updated data in the GUI as the client requested. This design pattern was not completed in the first sprint due to time constraints coupled with database design being heavily worked on during the first sprint.

#### 2.3.6 Sprint Review

The second sprint review was a lot more positive and uplifting for the team than the first one. It allowed for the team members to gain a more positive outlook on the project. The client gave very valuable feedback which then will be used in the finalisation of the project during the last week. The client expressed their likeness to the changes we made after the first sprint review and that the color palette was improved upon. The product owner then offered some suggestions on how to further improve the application. The first suggestion stated that the team should change date formatting from normal Year-Month-Day formatting to week formatting (For example, instead of 2019-01-07 the information should state that it is

Week 1). The second suggestion was to include how many days the project is behind in the detailed project view.

### 2.3.7 Sprint Retrospective

Despite this sprint review being more positive than the previous one. The team still took the sprint retrospective seriously. The retrospective can be found in **Table 2**. In the retrospective the team included what went wrong, what went well and which parts could be improved in the next sprint.

**Table 2.** Sprint Retrospective

What went well in this sprint?
Presentation was formal.
Effective schedule to show an accurate time estimation.
Using a printed out sprint plan to guide the presentation and to make sure that none of the key points would be missed so that we would focus on what is important.
Smaller and more organised user stories.
What went wrong in this sprint?
Several team members had health issues, which meant that on some days they had to work from home.
Conflicts in the team due to merging issues.
What can be improved?
Team communication. New methods should be developed to combat conflicts.
Bug Testing.

## 3. Post-Game

Despite there being no third sprint after the second sprint. The team after the sprint retrospective decided that the last week will be dedicated to mostly refactoring and writing the report as the changes the client asked for could be implemented very easily and quickly. Another reason for this kind of decision was the teams definition of done. By introducing more code and more methods the team is running the risk of encountering problems down the line that could take time and effort to patch out which a single week could be too little time.



Considering the last week is meant for mostly refactoring, bug fixing and merging things. No user stories were created as no new features will be added that could benefit the client in any way, shape or form.

### Implementation

During this post-game sprint the team has made unit testing on the program. This was done to check for unwanted bugs that the team did not know about and fully polish our application. The unit testing in this case tests only the BLL classes inside the task util folder since these classes are directly responsible for the logic of our program.

These unit test classes were done on the last week as this week the team was fully committed on bug fixing, merging and refactoring.

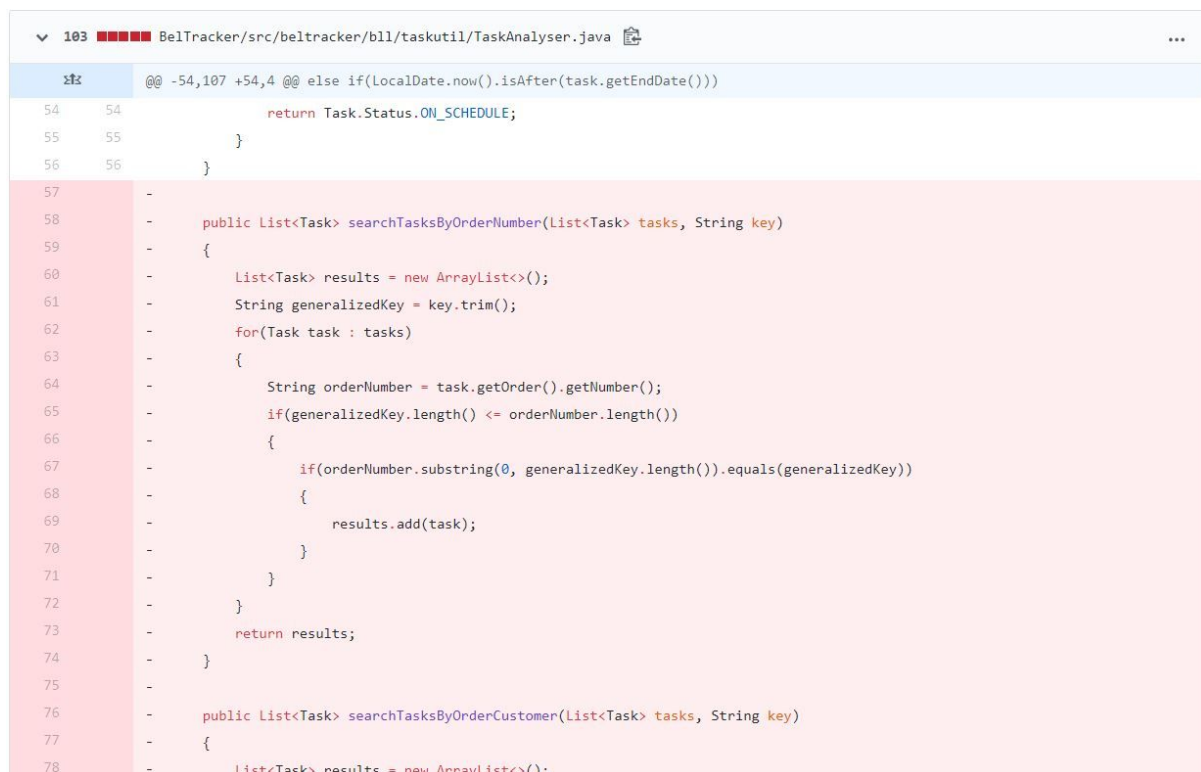
As a last task for this project, the team focused a lot of refactoring and restructuring the code.

### Code examples

As mentioned, the main goal of this week was restructuring and refactoring of methods, one of many application design changes we did was the specialization of classes. This can be seen in **Figure 19** and **Figure 20**.

In these figures you can see that the java class called TaskAnalyser was split up into 3 different classes called TaskAnalyser, TaskDetector and TaskSearcher. This was done to avoid code being bloated into one god class and allow quicker access of specific methods when changes need to be made.

**Figure 19.** Old class called TaskAnalyser. It contained code that needed refactoring and specialising into a different class.



```
103 BelTracker/src/beltracker/bll/taskutil/TaskAnalyser.java
@@ -54,107 +54,4 @@ else if(LocalDate.now().isAfter(task.getEndDate()))
54 54 return Task.Status.ON_SCHEDULE;
55 55 }
56 56 }
57 -
58 - public List<Task> searchTasksByOrderNumber(List<Task> tasks, String key)
59 - {
60 - List<Task> results = new ArrayList<>();
61 - String generalizedKey = key.trim();
62 - for(Task task : tasks)
63 - {
64 - String orderNumber = task.getOrder().getNumber();
65 - if(generalizedKey.length() <= orderNumber.length())
66 - {
67 - if(orderNumber.substring(0, generalizedKey.length()).equals(generalizedKey))
68 - {
69 - results.add(task);
70 - }
71 - }
72 - }
73 - return results;
74 - }
75 -
76 - public List<Task> searchTasksByOrderCustomer(List<Task> tasks, String key)
77 - {
78 - List<Task> results = new ArrayList<>();
```

**Figure 20.** Newly created class called TaskSearcher. It contains mainly task searching methods.

```

54 BelTracker/src/beltracker/bll/taskutil/TaskSearcher.java
...  ...  @@ -0,0 +1,54 @@
1  + /*
2  +  * To change this license header, choose License Headers in Project Properties.
3  +  * To change this template file, choose Tools | Templates
4  +  * and open the template in the editor.
5  +  */
6  + package beltracker.bll.taskutil;
7  +
8  + import beltracker.be.Task;
9  + import java.util.ArrayList;
10 + import java.util.List;
11 +
12 + /**
13 +  *
14 +  * @author Acer
15 +  */
16 + public class TaskSearcher {
17 +
18 +     public List<Task> searchTasksByOrderNumber(List<Task> tasks, String key)
19 +     {
20 +         List<Task> results = new ArrayList<>();
21 +         String generalizedKey = key.trim();
22 +         for(Task task : tasks)
23 +         {
24 +             String orderNumber = task.getOrder().getNumber();
25 +             if(generalizedKey.length() <= orderNumber.length())
26 +             {
27 +                 if(orderNumber.substring(0, generalizedKey.length()).equals(generalizedKey))
28 +                 {
29 +                     results.add(task);
30 +                 }
31 +             }
32 +         }
33 +     }
34 + }

```

Another key task done this week is the implementation of unit testing, while there was not a lot of logic to be tested, a series of tests were done to check for bugs. An example of unit testing can be found in **Figure 21**.

**Figure 21.** Unit testing for checking if task is overdue.

```

/**
 * Test of analyseStatus method, of class TaskAnalyser.
 */
@Test
public void testAnalyseStatusOverdue()
{
    LocalDate currentDate = LocalDate.of(2018, 1, 1);
    LocalDate startDate = LocalDate.of(2018, 1, 1);
    LocalDate endDate = LocalDate.of(2020, 1, 1);
    LocalDate deliveryDate = LocalDate.of(2019, 1, 1);
    Department department = new Department(1, "DepartmentOne");
    Department currentTaskDepartment = new Department(2, "DepartmentTwo");
    Order order = new Order(1, "1111-1111", "Customer", deliveryDate, currentTaskDepartment);
    Task task = new Task(1, department, startDate, endDate);
    task.setOrder(order);

    Task.Status expected = Task.Status.OVERDUE;
    Task.Status actual = taskAnalyser.analyseStatus(task, department, currentDate);
    assertEquals(expected, actual);
}

```

Future perspectives

In the future more features could be added, one such feature is allowing the workers to log into the system and submit how many hours they worked on a project in a given department. The SQL database design already contains the employee table and the employee department table which could allow for an easy implementation for this kind of feature.

Another thing to implement would be that the employee could assign itself to an existing department using some kind of menu. This would allow the employees to work on different tasks in newly created departments.

## 4. Conclusion

This project was a very big learning experience for the team. Not only did the team learn a lot more about software development in terms of coding but also about team communication and human problem solving.

Thanks to this work experience there are some definite improvements which could be made for the later projects.

### **Team improvements:**

Throughout the whole sprint a lot of conflict arose due to incorrect stress management and negligence of some roles. These issues have caused a great deal of division and conflict in the team. As a result, before the start of the next project the team should revise their social interactions and refine their development approach to cause less stress, be more respecting of other team members work and ensure a more open minded work environment.

### **Code improvements:**

During the development of the project, the team tried to use the most efficient and non deprecated methods in order to develop the program. However some methods were inefficient and could be potentially improved, but due to time constraints the improvements were not possible.

In the next project, the team will try to look ahead in order to ensure the most efficient methods will be used.

Another key thing to mention was the speed at which the team developed the code. It was not as fast as the team wanted due to minor and major team issues and contradicting opinions. The work speed and process will be improved in the next project as the team already figured out a new method for handling such issues.

The team also recognises that they were unable to implement transactions during the duration of the sprint. While the transactions would have allowed for an easier time verifying incoming data to the readers due to lack of time, the team was unable to implement this change.

### **Future prospects:**

While the team recognises the application's shortcomings and that they could be improved if given more time. The team decided against it, so that the team could focus more on preparing for the July 6th presentation instead of implementing code that could potentially

be untested or incomplete. Despite this, the team has prepared the structure to be easy to improve and built upon if the application would be chosen as the application Belman A/S would use.

## 5. References

The definitive guide for scrum: the rules of the game.

Link: <https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf>

The principles of UI design.

Link: <http://bokardo.com/principles-of-user-interface-design/>

A simple guide to connection pooling.

Link: [https://sourcemaking.com/design\\_patterns/object\\_pool/java](https://sourcemaking.com/design_patterns/object_pool/java)

Unit testing with JUnit.

Link: <https://www.vogella.com/tutorials/JUnit/article.html>

Database Design- Normalization

Link: <https://opentextbc.ca/dbdesign01/chapter/chapter-12-normalization/>

## 6. Appendices

### **Appendix A Working Agreement**

#### **Working hours and time scheduling**

- 9am - 3:30pm
- Daily scrum meetings for 15 minutes.
- Team members are free to take short breaks whenever needed.
- Two breaks scheduled, first from 11:30 - 11:45, second from 1:15-1:30.

#### **General behaviour**

- Be positive.
- In case of an argument try to be open minded.
- Do not be afraid to say you don't know something.
- Bring your concerns to the table as early as possible.
- Help others in case of problems.

#### **Daily scrum meetings**

- Always stand up.
- Take notes.
- Do not introduce new ideas or ramble for long.

#### **Definition of done**

- Is it fully functional?
- Can you demo it?
- If required, is it commented?
- Is it well structured and refactored?
- If required, is it unit tested?
- Is it compatible with the current version?
- If required, has it been tested on multiple devices?
- Has scrumwise been updated?
- Has it been committed and merged?

## Appendix B

### Overall Schedule

#### **Week 17** (23-26/04/2019)

- Trello board
- Planning
- Problem definition
- Product vision
- Strategic analysis
- Project organization
- Team collaboration policies
- Overall schedule
- Main epic stories
- Initial product backlog
- UML diagram & architectural design
- Mockup prototype
- Friday 26/4
  - First sprint start

#### **Week 18** (29-3/05/2019)

- Coding

#### **Week 19** (6-10/05/2019)

- Coding

#### **Week 20** (13-17/05/2019)

- Monday 13/5
  - 1st sprint review
- Sprint retrospective
- Tuesday 14/5
  - Second sprint start
- Coding

#### **Week 21** (20-24/05/2019)

- Refactoring
- Testing

#### **Week 22** (27-31/05/2019)

- Monday 27/5
  - 2nd sprint review
- Sprint retrospective
- Refactoring
- Testing

#### **Week 23** (3/05/2019)

- 12:00 Hand-in the report

## **Appendix C**

### **Standup Meeting Minutes**

#### **23/04/2019**

What was done today:

- Created some basics to prepare for the project (set up Discord and Trello, questions for product owner etc)
- Had our initial kick-off meeting with the product owner.
  - Some of our questions were answered and pinned on discord, but we still had need for some clarification
- Did a rough sketches for the project
- Started the base skeleton for the report
- Created our git repository
- Set our problem definition

#### **24/04/2019**

What was done today:

- Had feedback given to our previous project, notes taken are on Trello
- Came up with some design ideas
- Cleared up our goal for the project
- Created our epic stories
- Set up our project organization and schedule
- Updated our report

#### **Weekend(25/04/2019 - 26/04/2019)**

- Created the user stories.
- Set up scrum.
- Created the mockup prototype of our design.
- Finished our initial UML.
- Created the first sprint.
- Finished our initial Main View.

#### **26/04/2019**

What was done today:

- Updated the report.
- Started our first sprint.

#### **27/04/2019**

What was done today:

- Created initial database design.
- Started working on the JSON reader.

#### **29/04/2019**

What was done today:

- Updated and polished main view
- Finished reading properties files
- Updated the UML

#### **30/04/2019**

What was done today:

- Finished the main view.
- Made main view scalable/resizable.

- Created a loading view.
- Updated the database

#### **01/05/2019**

What was done today:

- Made a loading animation.
- Started on the tile view.
- Started on the factory pattern.

#### **02/05/2019**

What was done today:

- Created mock data for the orders.
- Finished the tile view.

#### **03/05/2019**

What was done today:

- Fixed minor visual bugs.
- Added two more tile views for delayed and overdue orders.
- Implement initial functionality for integrating JSON to database.

#### **Weekend(04/05/2019 - 05/05/2019)**

- Fully finished the delayed,overdue and on schedule orders views.
- Finished error logging.
- Refactored packages by putting resources outside the main package(Images,css,properties file).

#### **06/05/2019**

What was done today:

- Fixed visuals.
- Finished implementing functionality for checking task status.
- Finished the get order function in the database.

#### **07/05/2019**

What was done today:

- Refactored CSS.
- Updated UML.
- Started connection pooling.

#### **08/05/2019**

What was done today:

- Working on the connection pooling.
- Worked on some exceptions.
- Started working on the choose department view.

#### **09/05/2019**

What was done today:

- Finished connection pooling.
- Finished department choice view.
- Changed GUI structure.

#### **10/05/2019**

What was done today:

- Finished merging new GUI.

#### **Weekend(11/05/2019 - 12/05/2019)**

- Fixed minor bugs and refactored BLL.



**13/05/2019**

What was done today:

- Sprint review.
- Sprint retrospective.

**14/05/2019**

What was done today:

- Finished merging DAL with BLL.
- Refactored get orders.

**15/05/2019**

What was done today:

- Finished on schedule full view.
- Created delay and overdue full view.
- Updated observer pattern.

**16/05/2019**

What was done today:

- Finished animations.
- Finished observer pattern.

**17/05/2019**

What was done today:

- Fixed bugs in the observer pattern.
- Updated and refactored CSS.

**Weekend(18/05/2019 - 19/05/2019)**

- Updated UML.

**20/05/2019**

What was done today:

- Added a confirmation window.
- Added CSS for the confirmation window.

**21/05/2019**

What was done today:

- Finished the submission of the tasks in the database.
- Started working on an animation for the confirmation window.

**22/05/2019**

What was done today:

- Finished the function responsible for submitting the task.
- Finished the animation for the confirmation window.

**23/05/2019**

What was done today:

- Started working on unit tests.
- Started working on the searching function.

**24/05/2019**

What was done today:

- Finished the searching function.

**Weekend(25/05/2019 - 26/05/2019)**

- Finished the sorting function.

**27/05/2019**

What was done today:

- Sprint review.
- Sprint retrospective.

**28/05/2019**

What was done today:

- Finished merging the data readers.
- Finished unit testing.

**29/05/2019**

What was done today:

- Started refactoring the design of the application.
- Updated UML.

**30/05/2019**

What was done today:

- Finished fully refactoring the design of the application.
- Finished UML.

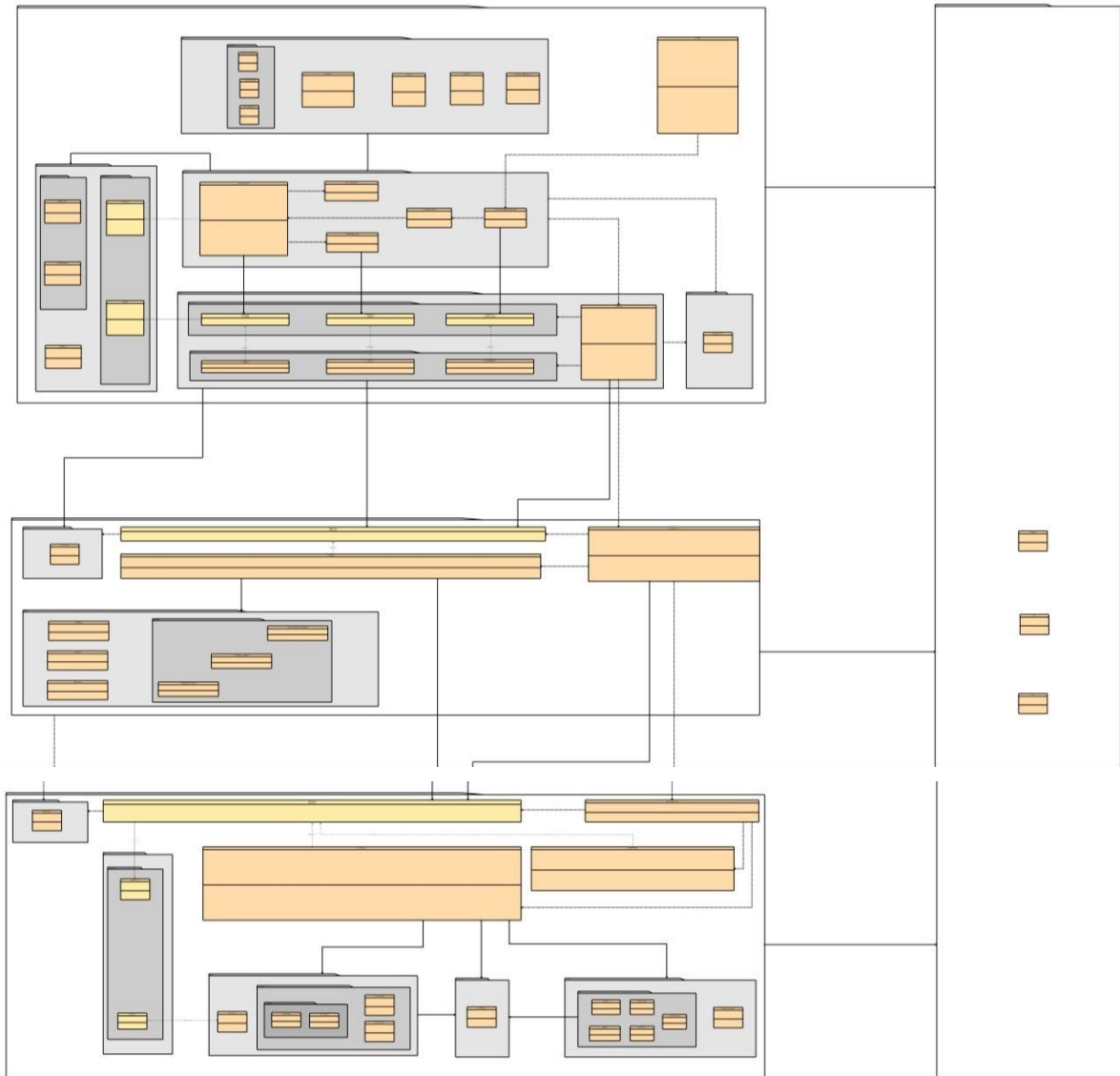
## Appendix D

### Full Vision Board of Our Program

<div style="display: flex; align-items: center;"> <div> <p><b>VISION</b></p> <p>What is your purpose for creating the product? Which positive change should it bring about?</p> </div> <div style="background-color: #e6f2ff; padding: 5px; margin-left: 10px;"> <p>The purpose of the product is improving efficiency of classes at EASV. It should make it easy and fast for students to mark their attendance and take this responsibility from teachers.</p> </div> </div>			
<div style="display: flex; align-items: center;"> <p><b>TARGET GROUP</b></p> </div> <p>Which market or market segment does the product address? Who are the target customers and users?</p> <div style="background-color: #e6f2ff; padding: 5px;"> <p>- Education institutions. - Target customers and users are teachers and students at EASV.</p> </div>	<div style="display: flex; align-items: center;"> <p><b>NEEDS</b></p> </div> <p>What problem does the product solve? Which benefit does it provide?</p> <div style="background-color: #e6f2ff; padding: 5px;"> <p>- Product solves the problem of wasting a lot of time during classes on marking attendance for all the students by teacher. - The benefit of the product is having more time spent on teaching during classes.</p> </div>	<div style="display: flex; align-items: center;"> <p><b>PRODUCT</b></p> </div> <p>What product is it? What makes it stand out? Is it feasible to develop the product?</p> <div style="background-color: #e6f2ff; padding: 5px;"> <p>- Desktop application which stores data in a database. - Clean visual that is simple and intuitive in it's use. - Yes we have a team of highly skilled people and the resources required to finish this product.</p> </div>	<div style="display: flex; align-items: center;"> <p><b>BUSINESS GOALS</b></p> </div> <p>How is the product going to benefit the company? What are the business goals?</p> <div style="background-color: #e6f2ff; padding: 5px;"> <p>- More time spent on education during classes what will benefit in better quality of education. - Create a freely distributed attendance system for the university to use.</p> </div>

## Appendix E

## Completed UML Diagram



To see a more detailed version of this diagram. Open [BelmanApplicationsUML.PDF](#) included in the ZIP file.