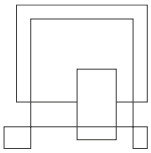


Digitalization of Quality Assurance for KonfAir

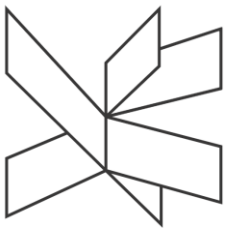
Final Project

Simon Emmanuel Chukwukodinaka 285152

Rafał Pierścieniak 279471

Rokas Barasa 285047

Supervisor: Poul Væggemose



Bring ideas to life
VIA University College

75,126

Software Technology Engineering

7th semester

01.06.2022

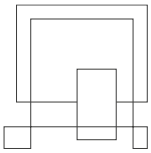
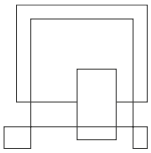
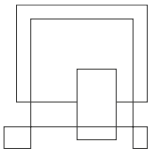
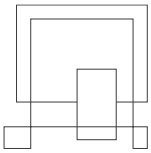


Table of content







1 Introduction

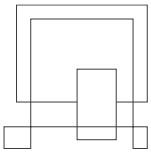
Computers have advanced a lot in the past few decades making them very viable for use in everyday activities. Companies are gradually switching to more digitalized processes to make employees work more efficient and organized. Despite there being clear benefits to digitalization, there are still some companies that have not taken the step. (Pratt, 2021)

KonfAir is a company that manufactures air filters. KonfAir is an old company, it manages a lot of its production process on paper. Orders are received, compiled into steps by the management, which lets the worker know how to make the product, they are manufactured and then are quality checked.

A production paper is a problem for the employee as there is only one needed per product batch. The quality check step creates a lot of paperwork for the employee, a product can be assembled from multiple pieces each of which may need its own page of quality checking. It is also a lot of work for the management as they need to make a new item control form using graphical software. Because the management can’t create many new forms constantly, some forms only have a small number of control points relevant to the item being checked.

At the end of the quality checks, the employee then must scan each of these papers and put them in a folder on a hard drive. KonfAir must provide quality assurance papers to the client to validate that the product was not broken in the manufacturing process. Because the forms are scanned there is no way for the Quality assurance team to quickly learn anything from the forms.

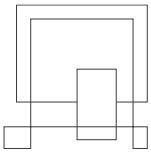
Konfair’s current way of supporting the production process through paper documents is inefficient. Making the employee handle a lot of papers wastes time and may contribute to errors or even missing documents. It also produces a lot of paper waste after the



documents are scanned and thrown out. KonfAir also does not have a way to learn anything from the data while completed orders are only stored as PDFs.

They are using Microsoft Dynamics Nav for handling orders outside the production area and managing a database which is used inside in-house systems. This software is not a good fit for production employees to use as it would increase the complexity of filling in data. What is more, it brings high risk due to the ability to modify any data in the company database which according to KonfAir should be possible only by specifically selected administrators (Appendix 9 – Timetable; Meeting with customer KonfAir).

KonfAir needs a piece of software that would be simple for factory employees to understand, simple for managers to create forms on and recreate the existing process visually and help fill the needed forms quickly



2 Analysis

2.1 Requirements

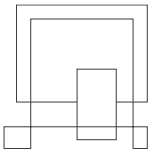
2.1.1 Functional requirements

Using the project description and meetings with the customer several requirements were created to represent customers' needs from the system. The requirements describe the surface level functionality of the system and detailed contents of what is needed from a requirement is contained in the use case descriptions.

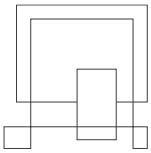
The functional requirements made for this project have been classified using SMART principles to ensure better requirements. The prioritization of the requirements has been made using the MoSCoW principle and can be seen as shown below.

Prioritization of requirements was done in collaboration with the customer and has been accepted by the customer.

Index	Req No	User stories	MoSCoW
1	[REQ1]	As an **QA worker** and **Administrator**, I should be able to **login so that I can be identified.**	**Must have**
2	[REQ2]	As an **Administrator**, I should be able to **see list of control points, so that I can choose one.**	**Must have**
3	[REQ26]	**As an Administrator**, I should be able **to specify a connection for the category items while creating control point, so that it has correct relation**.	**Must have**
4	[REQ27]	**As an Administrator**, I should be able to **specify connection for attributes while creating control point, so that it has correct relation.**	**Must have**
5	[REQ28]	**As an Administrator**, I should be able to **specify description while creating control point, so that**	**Must have**

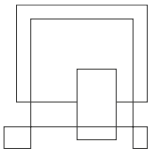


		it provides the QA worker with correct information.	
6	[REQ29]	**As an Administrator**, I should be able to **specify the input type while creating control point, so that the QA worker knows what to input**.	**Must have**
7	[REQ30]	**As an Administrator**, I should be able to **specify a tolerance while creating control point, so that I can specify how accurate the item has to be to the one the customer requested.**	**Must have**
8	[REQ31]	**As an Administrator**, I should be able to **specify the frequency while creating control point, so that I can change how often the check for the specified point is made.**	**Must have**
9	[REQ32]	**As an Administrator**, I should be able to **upload a picture while creating control point, so that it helps QA worker to perform measurements**.	**Must have**
10	[REQ34]	**As an Administrator,** I should be able to **specify measurement type while creating control point, so that it can be define if control point is one time or multiple times measurement.**	**Must have**
11	[REQ33]	**As an Administrator**, I should be able **to edit chosen control point, so I can correct what the control point does.**	**Must have**
12	[REQ11]	As an **Administrator**, I should be able to **add a user, so that I can add him to the system.**	**Must have**
13	[REQ12]	As a **QA worker**, I should be able to **see released orders with uncompleted QA form so that I can choose one.**	**Must have**
14	[REQ13]	As a **QA worker**, I should be able to see the QA form of a chosen order released**, so that it shows all control points and instructions that are applied to this order.**	**Must have**



15	[REQ14]	As a **QA worker**, I should be able to fill in the measurements on QA form **so that I can perform quality assurance check.**	**Must have**
16	[REQ15]	As a **QA worker**, I should be able to submit QA form **so that I save the changes.**	**Must have**
17	[REQ16]	As a **QA worker and Administrator**, I should be able to **choose department so that I can see data relevant to my location.**	**Should have**
18	[REQ17]	As an **Administrator**, I should be able to **see item categories, so that I can choose one.**	**Should have**
19	[REQ18]	As an **Administrator**, I should be able to **edit the frequency of chosen item category, so that I can set how often checks are done**.	**Should have**
20	[REQ19]	As an **Administrator**, I should be able to see orders that have completed QA form **so that I can choose one.**	**Should have**
21	[REQ20]	As an **Administrator**, I should be able to **see QA of chosen completed order, so that I can validate it.**	**Should have**
22	[REQ21]	As an **Administrator**, I should be able to **generate QA form PDF of chosen order with completed QA form so I can present it to a customer.**	**Could have**
23	[REQ22]	As an **Administrator**, I should be able to **delete a chosen control point, so I can remove it from the system.**	**Could have**
24	[REQ23]	As an **Administrator**, I should be able to **delete a user, so I can remove them from the system.**	**Could have**
25	[REQ24]	As an **QA worker and Administrator**, I should be able to **change language to Danish, English or Lithuanian so that I can see the system in a language I understand.**	**Could have**

Table 1 Functional requirements



2.1.2 Non-functional requirements

1. [NREQ-1] The System should make use of already existing Database in the Company
2. [NREQ-2] The System should create Database of QA forms in Microsoft SQL Server
3. [NREQ-3] The System should ensure that all functionalities have a loading time of less than five seconds.
4. [NREQ-4] The System will not accept inputs that are negative in any input area
5. [NREQ-5] The System will not accept inputs that are a number bigger than 2147483647
6. [NREQ-6] The System will not accept inputs that are text above length of 50 characters unless stated otherwise
7. [NREQ-7] The System will not accept specific description inputs that are text above length of 200 characters

2.2 Use Case Diagram

All the functional requirements have been taken into consideration while creating the Use Case Diagram. Use case diagram can be seen in figure 1 and found in appendix 2 – UML Diagrams.

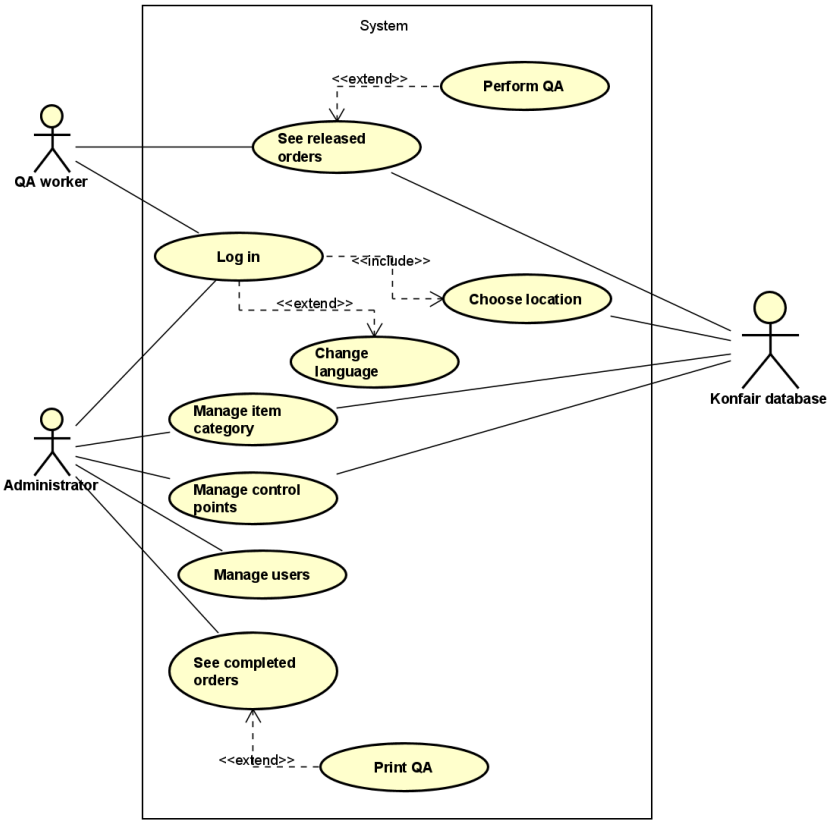


Figure 1 - Use Case Diagram

There were two primary actors derived from the requirements:

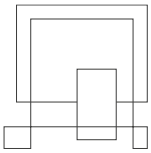
- QA worker
- administrator.

There is also a support actor in the system (Larman, 2004):

- KonfAir database. This actor provides relevant information from the Konfair database for all use cases it associated with it.

These are the use cases that were derived:

- See released order
- Perform QA
- Login



- Change language
- Change location
- Manage item categories
- Manage users
- Manage control points
- See completed orders
- Print QA

The administrator actor manages the critical information of the system like control points, item categories, users and completed order overview with pdf printing. This is a big contrast to the QA worker who only performs QA for released orders.

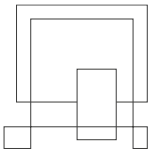
The login use case is supplemented by choosing language and location due to it being required to be specified before the user is logged in to the system.

2.3 Use Case Description

Each of the use cases in the use case diagram has a related description.

The use case descriptions provide a detailed description of the functionality, how the different actors interact with the system, the different outcomes of the use case and its failure scenarios. The use case descriptions are also made to reproducible by steps which can be tested. An example of a use case description for item category management can be seen in figure 2.

The summary section of the description briefly explains the functionality that can be done, the exceptions that can happen and the alternative things (branch sequences) the user can trigger in this use case. This gives an outline of a feature of what it should have and what it should accomplish. The use case descriptions are also where the customer's specific needs are considered, and the functionality of the use case that is too small to be its own requirement. Much like completion criteria for the use case.



ITEM	VALUE
UseCase	Manage item category
Summary	Administrator is able to see item categories in the system as well as edit settings of them. Th e user can change the freuquency of chosen item category. The information of item categories is obtained from the Konfair database
Actor	Administrator, Konfair database
Precondition	User must be authorized as administrator
Postcondition	item categories are shown, item category frequency is edited
Base Sequence	Edit item category frequency 1) User goes to manage item categories page 2) System obtains item category information from konfair database 3) System displays list of item categories 3) User selects an item category form list which he wants to edit 4) System takes user to selected item category info page 5) User edits the frequencies of the item category 6) User saves changes 7) System saves changes and takes user back to item categories page.
Branch Sequence	A) User exits item category info page in step 6 ----System discards changes and takes user to step 3
Exception Sequence	A) User saves with frequencies that are not positive integers. ----System notifies the user of non positive numbers and takes user to step 5 B) User saves with frequencies that are too long. ----System notifies the user of values that are too long and takes user to step 5 C) User saves with frequencies that are text. ----System notifies the user of non and takes user to step 5
Sub UseCase	
Note	

Figure 2 - Use case description of item category management

All use case descriptions can be found in appendix 3 – Use case descriptions.

2.4 Activity Diagram

Activity diagrams were created to clear up complex use case description scenarios. Some examples of complex use cases would be released orders, perform QA and control point management. All activity diagrams can be found in appendix 2 – UML diagrams. The overview of the activity diagram to see a released order can be seen in figure 3.

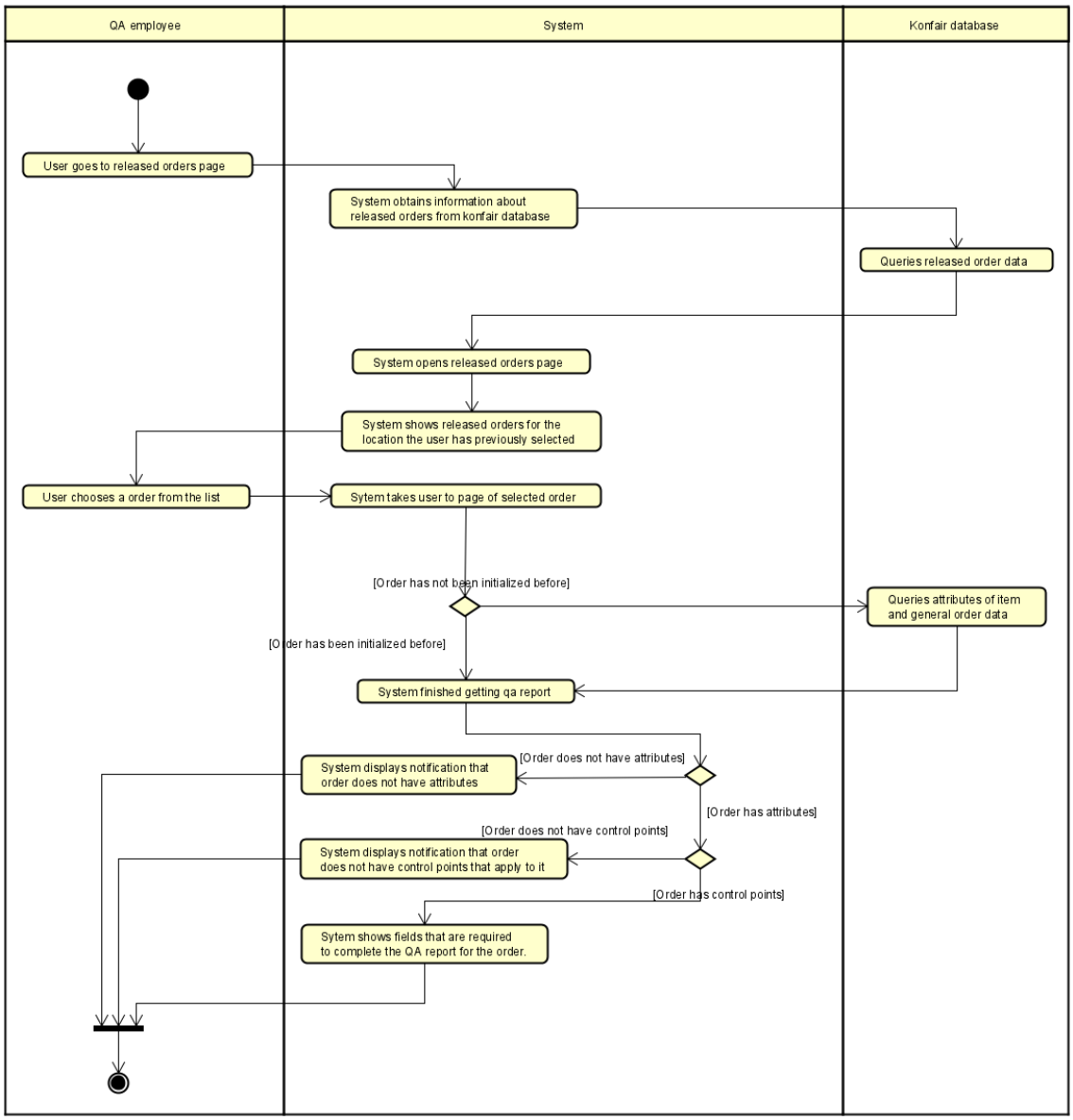


Figure 3 - See a released order activity diagram overview

The activity diagrams demonstrate all the sequences from the chosen use case description.

2.5 System Sequence Diagram

To understand in deep the Use Case Description scenarios the System Sequence Diagrams have been created. The Diagram below explains massages between two timelines, needed to carry out the scenario. All System Sequence Diagrams can be found in appendix 2 – UML Diagrams.

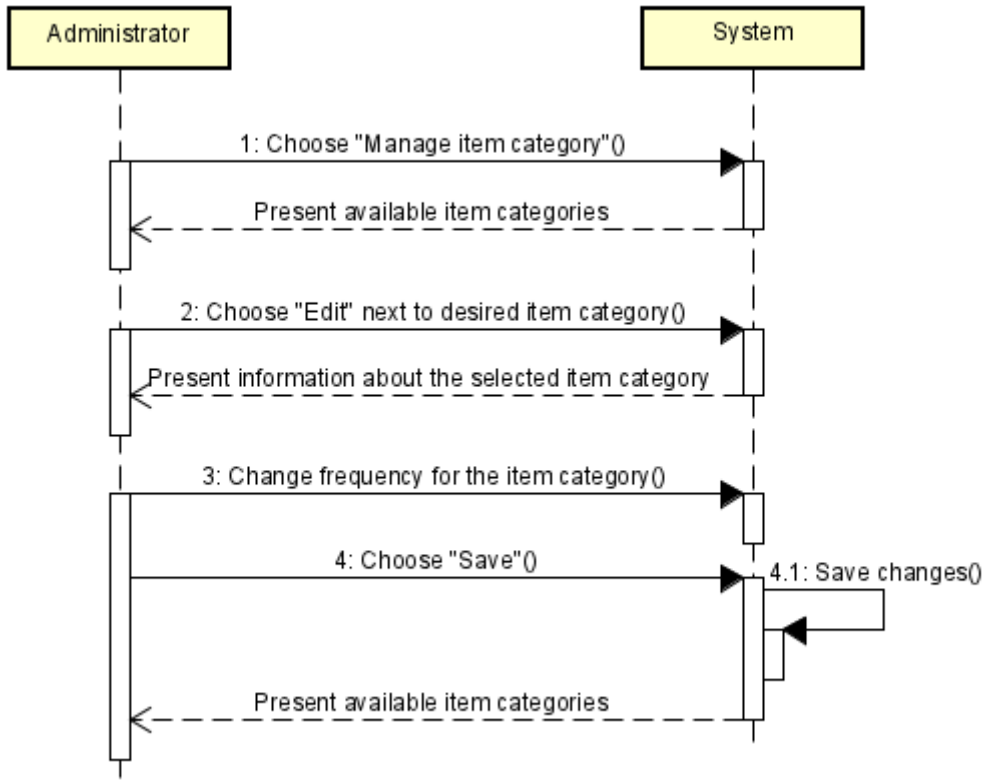


Figure 4 - System Sequence Diagram of "Manage item category" use case for edit frequency

The System Sequence for the same use case but with timeout can be seen on figure 5.

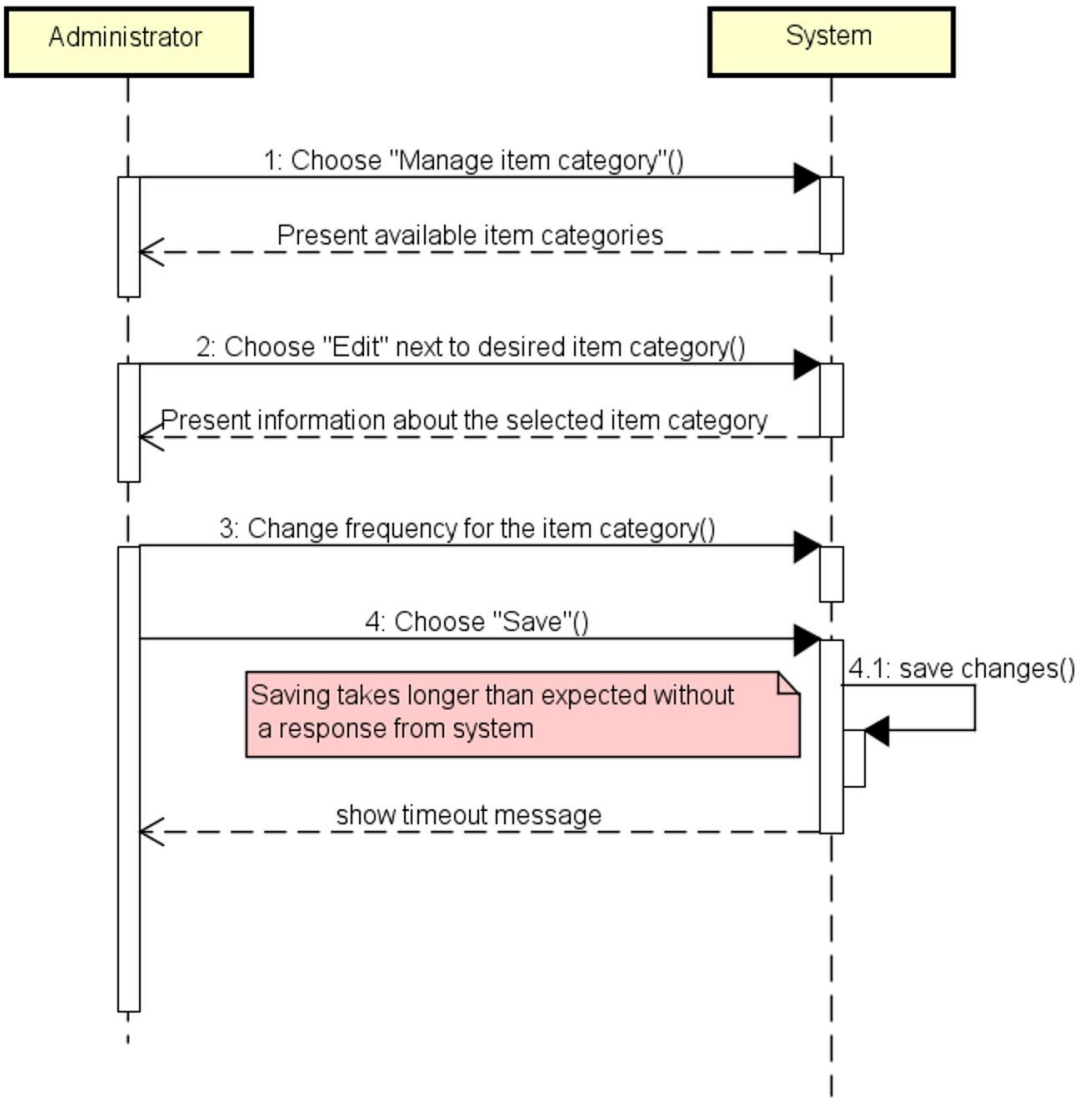


Figure 5 - System Sequence Diagram of "Manage item category" use case for edit frequency with timeout

2.6 Domain model

The result of the analysis of all of the requirements, use case diagram, use case descriptions and activity diagrams is the domain model. The domain model represents conceptual classes in the real world. Conceptual classes are things, objects, or ideas (Larman, 2004). The domain model diagram can be seen in figure 6. An example of a conceptual class is the QA form, it is something that exists in the real world.

The classes are connected by associations that show which classes are related to each other and in what way.

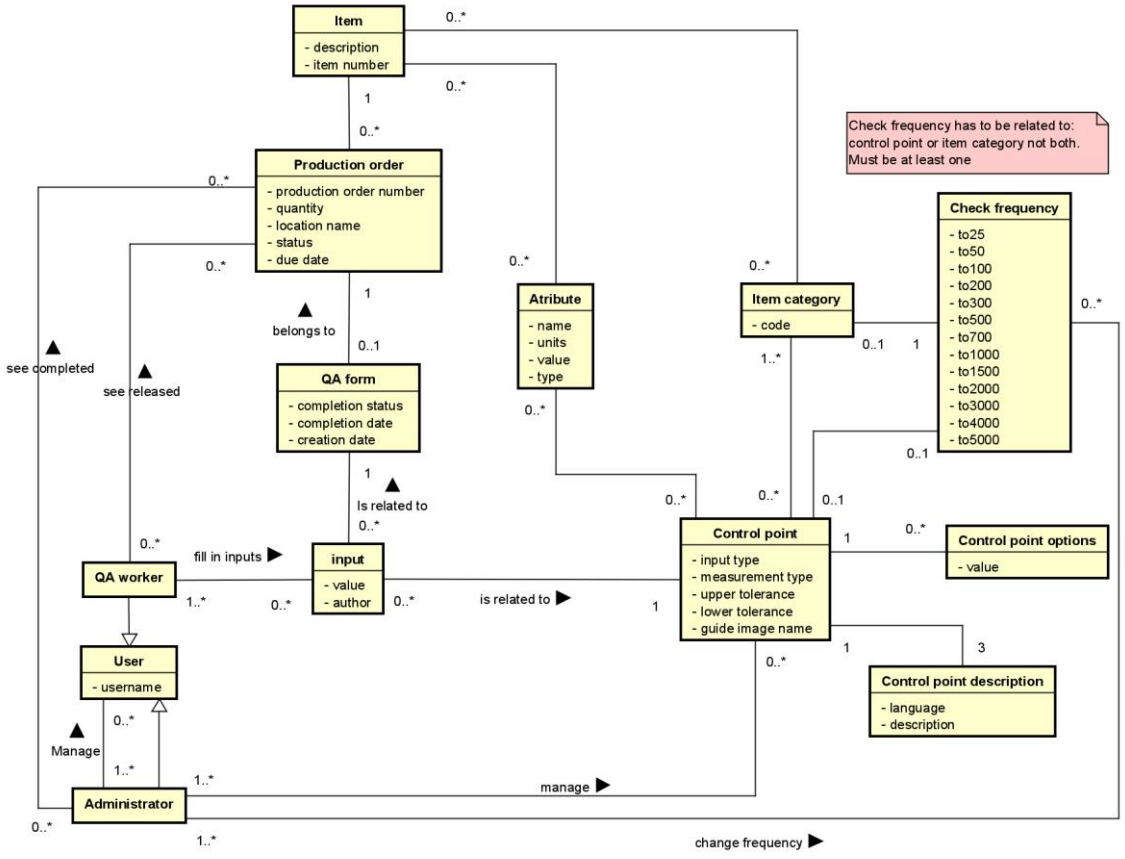
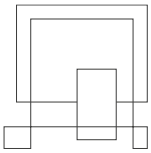


Figure 6 - Domain Model

To understand the domain model a good place to start is the production order.
When KonfAir gets an order from their customer, they make a new production order for each item in their customer's request (KonfAir stated one item per production order).



The production order has some basic information about the order. KonfAir then looks for an existing item that matches what their customer wants or alternatively creates a new item that matches their customer's needs. The item has attributes with parameters that the customer specified and an item category code.
This is what was found during analysis of the Konfair's existing system.

The general unique item attributes and general unique item categories are then attached to a control point and used as a way of predicting what checks the items that have these combinations of attributes and item categories should have in the QA process.

The control point describes what the QA worker should check for. Input type defines if the control point is about the option selection, text validation or numeric measurement. Measurement type defines if the control point is one time or multiple time evaluation.

The control point can have a frequency related to it which determines how often it is checked based on the quantity of the item (for multiple time measurement control points only). The item category can also have a frequency. The item category frequency is the base frequency of checks, if the control point has a frequency related to it then it overrides the item category frequency for that control point checks.

The control points and item categories are managed by the administrator. Once Konfair changes the status of a production order to release the QA worker can see the production order and is able to fill in inputs that are related to the control points that are applied to the released order. The user's inputs are finally related to the QA form for that production order and item combination.

There are two users both having usernames. The administrator can create other users with the role QA worker or administrator.

The domain model can be found in appendix 2 – UML Diagrams.

2.7 Test cases

Test cases were derived from the use case descriptions. The main sequence, branch sequences and exception sequences are all scenarios of the use case that need to be tested. A small example of the test case descriptions can be seen in figure 7.

ID	Test specification						Result	
Test Case Id	Test Scenario	Test Steps	Test Data	Precondition	Postcondition	Expected Results	Actual Results (Only for faild)	Pass / Fail
U01T01	QA worker wants to log into the system "sunny scenario"	Enter username, Choose "language", Choose "location" Choose "Sign in"	Username of existing user in database with QA worker role.	User has selected a location, Entered username exists in the database, There is connection to Konfair's network	System validates the login credentials	User can see the page for QA worker role		

Figure 7 - Test case example

There are multiple test cases per use case. The test cases follow the steps in the use case descriptions to test the sunny scenario, branch sequences and exception sequences of the descriptions. There are other scenarios tested as well, that are not specified in the use case description due to it being abstract and not focused on the design of the application. These other tests focus on some features of the design choices when testing due to the outcome variability of the design.

The whole test case table can be found in appendix 5 – Use Case Testing.

3 Design

3.1 User interface design

All UI designs can be found in appendix 6 – UI layouts.

3.1.1 Gestalt principles

The user interface was designed with the aid of the customer. In addition, to working with the customer on the UI, The GESTALT principles were implored in providing a qualitative, intuitive, and pleasurable user experience to the customers. (Douglas, 2017)

Closure: In the project the team make use of closure by making use of negative and positive spacing, using icons. For the project, mdi icons are available to form closure in the project. They are not readily present in Figma but will be used in the implementation of the system

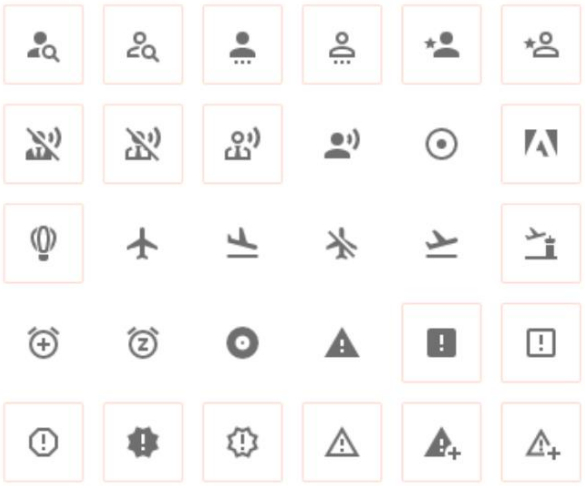


Figure 8 - mdi icons for closure in project

Continuity: In the project, the hierarchy is defined on the UI by making use of continuity. Continuity helps the user of the system understand the different demarcations between layers. This can be easily deduced on several tables, bars, and headers to form separation of concern on the UI. It aids the perception of grouped information.

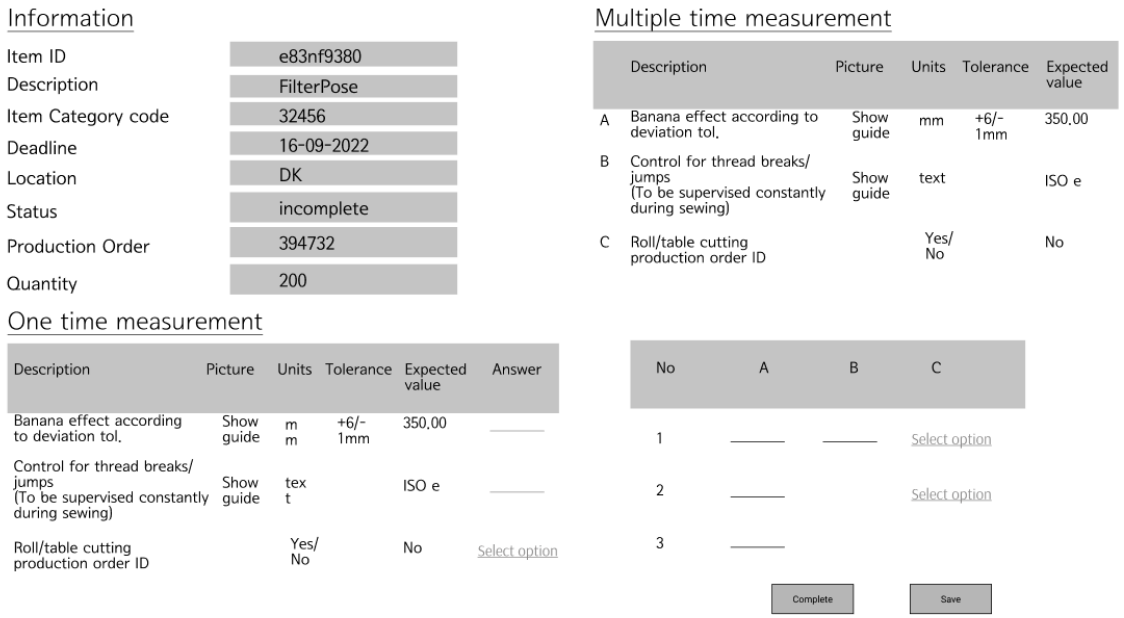


Figure 6 - continuity on the table of the QA form

Similarity in the system is used greatly to form several of the pages as it helps the user to perceive things as like each other. In the project, colour, size, shape, texture, and orientation are used to group similar objects. An example is the several items in the frequency form

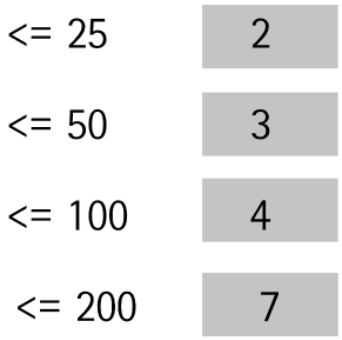


Figure 7 - similarity between two fields of frequency

Common fate: Is useful in the system in achieving information grouping and content organization. It is perceived that items which appear at the same time share the same fate as opposed to those which are stationary. An example of this is the disparity between the navigation bar and the differing pages in the system.

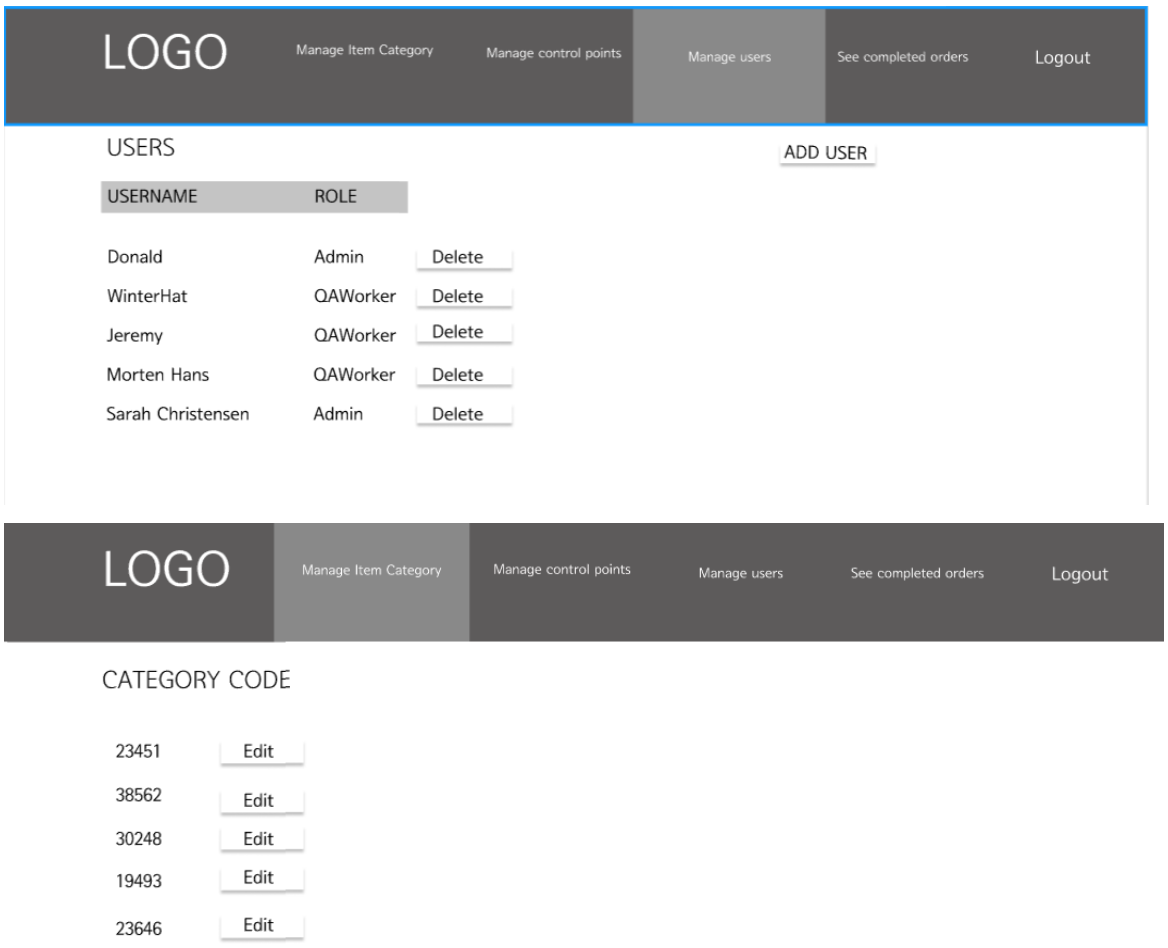
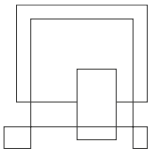


Figure 8 - Common fate on the system via user page and item category page

Figure ground: Gives context in the system to what is a figure (what the user should see) and ground (the background information). This can be seen throughout the whole UI design. An example of this is the login page UI.



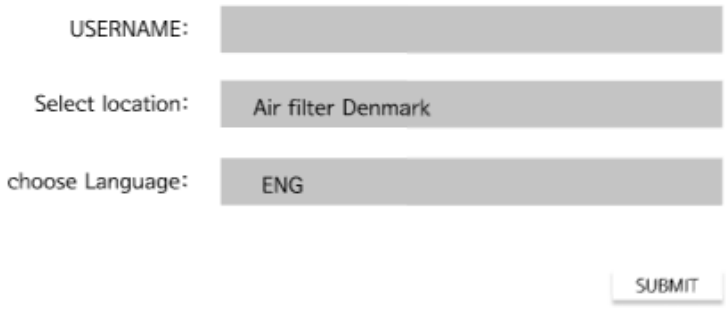


Figure 9 - Figure ground login card on plain white background

Proximity: things that are close together are related together. In the system white space is used to further enhance this principle.

COMPLETED ORDERS

Production Order	Item Number	Item Category code	Quantity	Completion date	Deadline	
234564	123456	32456	100	25-05-2022	15-05-2022	print
942213	849302	32656	6	25-05-2022	15-05-2022	print
098492	284048	12456	900	25-05-2022	15-05-2022	print
276384	440482	26456	24	25-05-2022	15-05-2022	print
648392	348840	20456	654	25-05-2022	15-05-2022	print
938729	129304	32722	123	25-05-2022	15-05-2022	print

Figure 10 - proximity between each column

3.1.2 System platform

KonfAir wanted a system that they could use on either tablet that the employees would carry around or a stationary computer, the latter being more common. Due to this, the system was designed to fit onto a 1080p screen as it is the most common screen resolution. (statcounter, 2022)

3.1.3 Figma

Figma was used to make sketches of what the user interface of the system should look like.

Figma offers the capability to create a UI with pages that have basic hand-created elements. An element can be made clickable and leads to a different page of the application. This can be used to demonstrate the understanding of the team of the needs of the customer. The customer then can comment on and request adjustments to better capture what the customer has in mind. Figma UI can be seen in figure 11

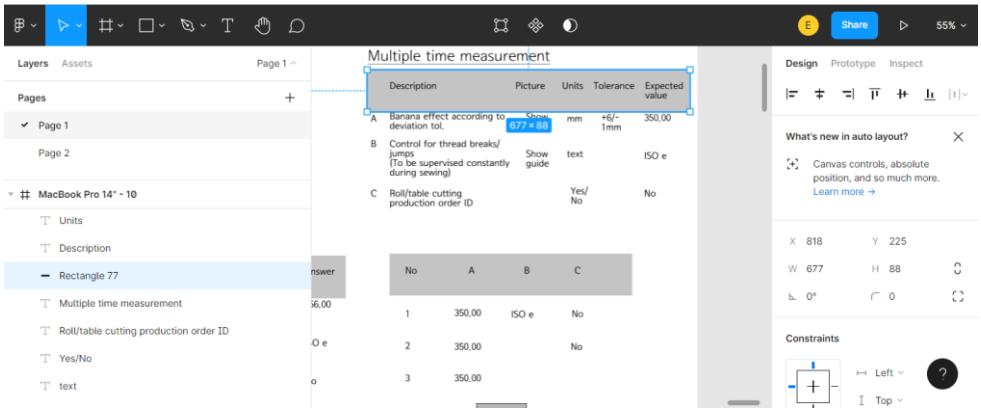


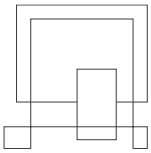
Figure 11 - Figma UI

This was used to understand the requirements and the scope of the project as well as to make a loose visualization of what is expected to be present in the UI.

Another sketching tool was considered, Adobe XD. The team had previous experience with Adobe XD however, it did recently become pay to use service that the team could not invest in. It was dropped for Figma due to this. Figma matched the functionality of Adobe XD while being free to use for one project.

3.1.4 Pages

This section will show some design choices of the individual pages. When reading keep in mind that all lists are designed to be infinitely scrollable. No pages are needed.



3.1.4.1 Navigation bar

The navigation bar helps the user navigate through pages. Only the pages that the user is authorized to visit are shown. Admin view mockup can be seen in figure 12 and the QA worker view mockup can be seen in figure 13.

The navigation bar will highlight the page that the user is on currently.

Figure 12 - Figma admin navigation bar

Figure 13 - Figma QA employee navigation bar

3.1.4.2 Login

The login allows the user to authenticate using a username and at the same time choose location and the language. No password is needed for authentication as was specified by the customer.

The location and language were discussed with the customer for availability on all pages, but a decision was made to only have it in login page as it will be very uncommon for users to change language and location, once they have already picked them.

The language selection translates every piece of static text in the application to the specified language. This was done because the system is used by Lithuanian and Danish employees of the system. English was also kept as according to Konfair 70% of their employees use English documents in the production process already.

A design choice was made to redirect the user to this login page whenever the user visits an authentication requiring page without being authenticated.

The login page elements can be seen in figure 14. The username field is meant to be a text field. Both, “select location” and “chose language” are meant to be dropdowns.

Figure 14 - Figma login page

3.1.4.3 Control point page

The control points page contains a list of all the control points in the system. An id of the control point is shown to make it easier to identify the control point as well as the first few words of the description of the control point. The list items are clickable and lead to the control point details page.

Figure 15 - Figma control point list

3.1.4.4 Control point creation and editing

The control point creation page and the editing page look similar. The page allows a user to specify the description of the control point. A picture can be chosen as well as input type and control point measurement type. The measurement type is used to explain what type of measurement the control point should be and can be one time or multiple time measurement. There is also the input type. Which specifies what type of input should the control point input be to the QA worker. This can be number, text or options. In the options case, there can be multiple options specified for the control point.

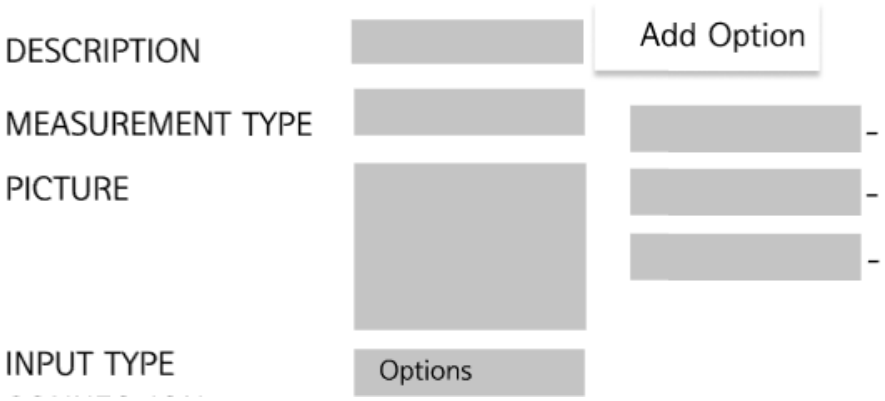
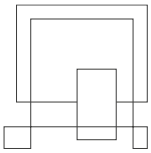


Figure 16 - measurement type, input type and options in control point

Attributes and item category codes can also be chosen between available values in the database. Lastly, the frequency of the measurement can be added by the user. Although the frequency is dependent on the measurement type as if it is one-time



measurement, then the frequency cannot be added. The range of the attributes can also be specified for each attribute that is being connected to the system.

This page is reusable for the editing of the control point.

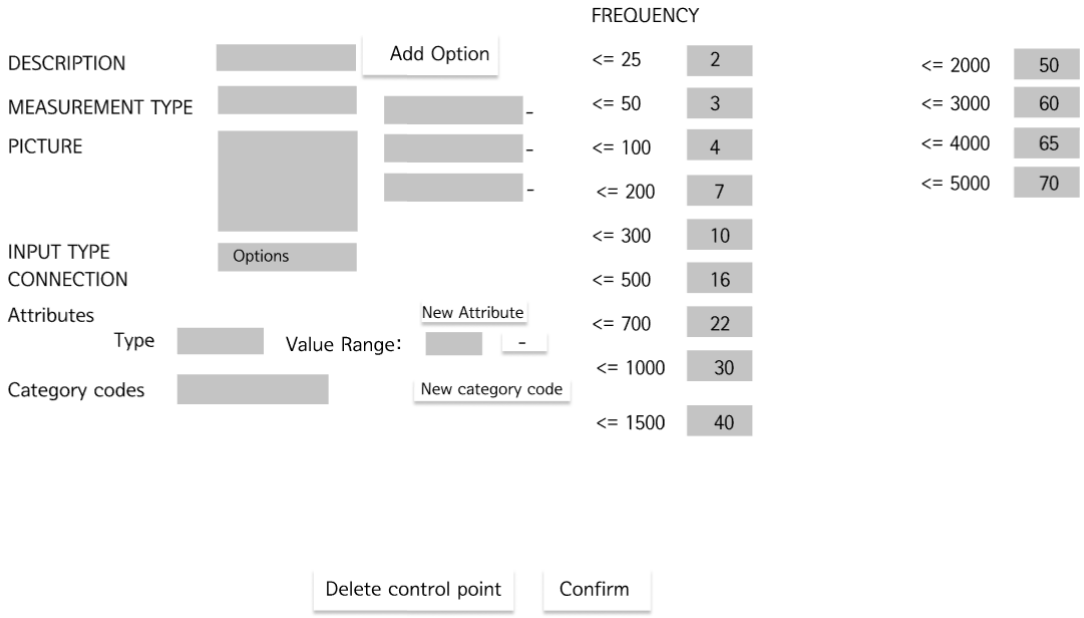
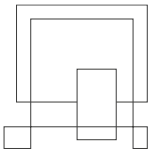


Figure 17 – edit/create control point

3.1.4.5 Released orders page

The released order page contains a list of all released orders with QA forms not yet completed. The orders are shown based on the location that the user has specified when logging in. The table contains columns of production order, item number, item category, quantity, and deadline of the order. The user can click on an order and be led to another page with the QA form of the order. The released order page can be seen in figure 18.



Item Number	Item Cat. Code	Quantity	Deadline	Production Order	
123456	138844	200	14-02-2022	234567	view
233445	377320	45	19-03-2022	233445	view
948503	353829	1000	13-05-2022	948503	view
374655	102930	234	23-05-2022	374655	view
038485	294024	12	09-06-2022	038485	view
874637	103859	457	11-07-2022	874637	view

Figure 18 - Figma released orders list

3.1.4.6 Released order QA form page

This page displays the information about the order and the required inputs to complete the QA form for that order. The initial mockup can be seen in figure 20.

The QA employee can input data in 3 categories:

- Text input
- A number input with tolerances
- Pre-specified options

The inputs are validated as the user enters values. This is done for the QA worker to quickly notice if he did something wrong or something that he should pay attention to. Input is marked red for incorrect and dark orange for out of tolerance.

An expected value based on the attributes connected to the item is shown to guide the QA employee to know if the values he is measuring are okay. The expected value is the actual value of the attribute of the item, possibly the one that Konfair's customer specified.

A button is available for control points that have an image linked to them. The input area once hovered over is to display a tooltip of the author of the input and the data and time when the input was saved by that user.
A design choice was made to also include a button to see a picture guide of the chosen control point. This is shown next to the description of the control point.

The data presented in the figure 19 is just dummy data which is shown to illustrate elements on the page in a realistic way.

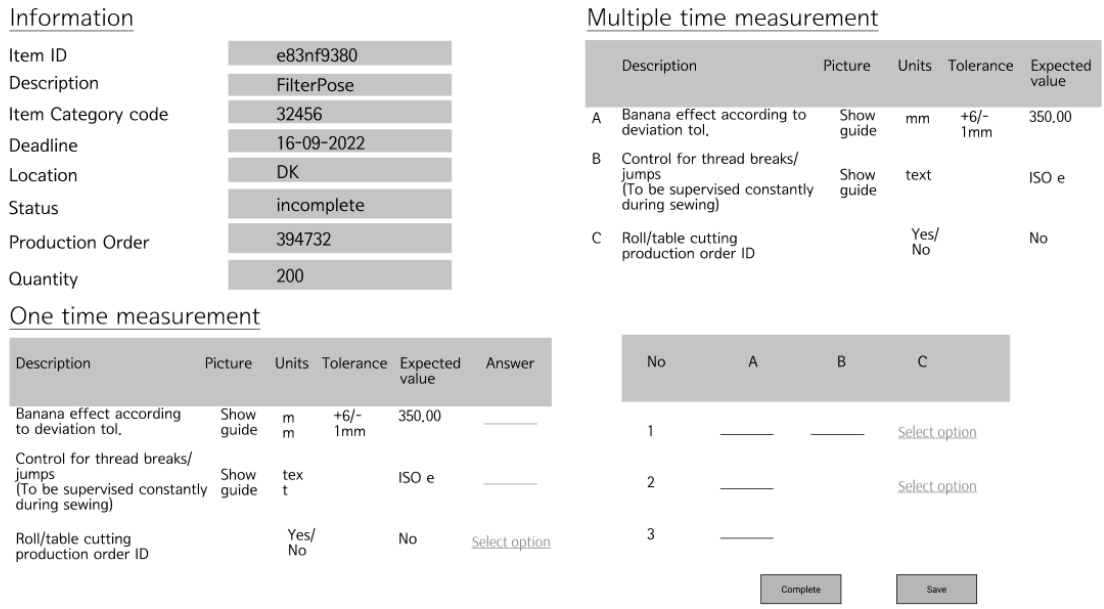
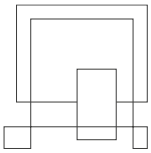


Figure 19 - Figma released order page

Each “show guide” entry in the Picture column for both measurement type tables has a button functionality. It is designed to present an actual picture on the screen that is related to the control point after it is clicked.

3.1.4.7 Completed orders page

The page contains the orders that the QA employee has marked completed. It is almost the same as the released orders list, it has an extra column of the completion date for when the QA worker marked the QA form completed. This is to know if the QA



form was completed on time. The list of orders with completed QA form can be seen in figure 20.

COMPLETED ORDERS

Production Order	Item Number	Item Category code	Quantity	Completion date	Deadline	
234564	123456	32456	100	25-05-2022	15-05-2022	print
942213	849302	32656	6	25-05-2022	15-05-2022	print
098492	284048	12456	900	25-05-2022	15-05-2022	print
276384	440482	26456	24	25-05-2022	15-05-2022	print
648392	348840	20456	654	25-05-2022	15-05-2022	print
938729	129304	32722	123	25-05-2022	15-05-2022	print

Figure 20 - Figma completed orders page

3.1.4.8 Order with completed QA form page

The page is the same as the one for the released orders page, except that it has no way for the user to edit the set values or save them. The administrator has a button to print the page with the information on it as a pdf. The pdf is intended to have the pictures of the control points attached to the end of it.

Information

Item ID	e83nf9380
Description	FilterPose
Item Category code	32456
Deadline	16-09-2022
Location	DK
Status	Completed
Quantity	200
Production Order	384591

One time measurement

Description	Picture	Units	Tolerance	Expected value	Answer
Banana effect according to deviation tol.	Show guide	mm	+6/-1mm	350,00	356,00
Control for thread breaks/ jumps (To be supervised constantly during sewing)	Show guide	text		ISO e	ISO e
Roll/table cutting production order ID		Yes/ No		No	No

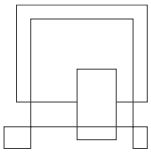
Multiple time measurement

	Description	Picture	Units	Tolerance	Expected value
A	Banana effect according to deviation tol.	Show guide	mm	+6/-1mm	350,00
B	Control for thread breaks/ jumps (To be supervised constantly during sewing)	Show guide	text		ISO e
C	Roll/table cutting production order ID		Yes/ No		No

No	A	B	C
1	350,00	ISO e	No
2	350,00		No
3	350,00		

Print

Figure 21 - completed QA form page



3.1.4.9 Item category list

The item Category page is the page which displays all item category codes in the system. The user can click on an item category code which will take them to a page to edit the item Category code frequency. The list can be seen in figure 22.

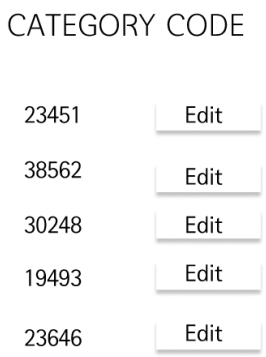


Figure 22 - Item category code list

3.1.4.10 Edit item category page

The edit item category page allows a user to modify the frequency of the chosen item category. The user can apply the changes The item category is then updated for the chosen when the user clicks the save button. The page can be seen in figure 23

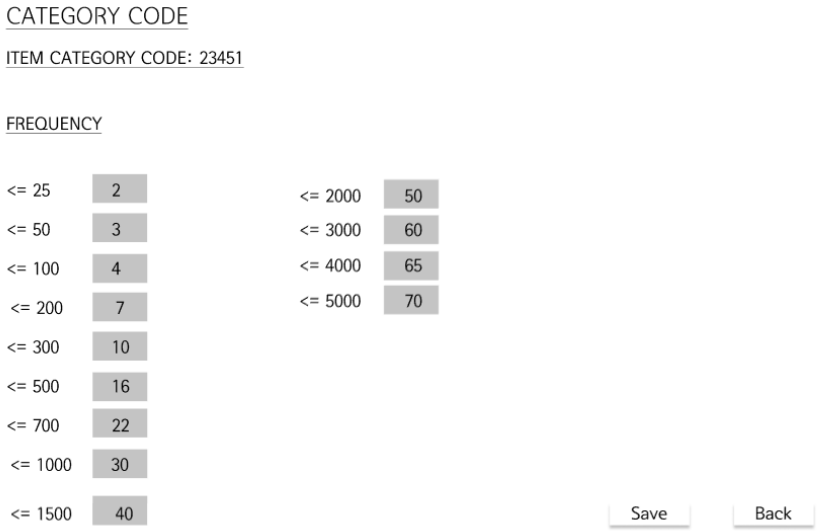
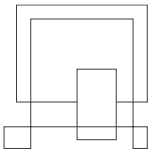


Figure 23 - edit item category page



3.1.4.11 User list and create user

The manage users page contains a list of the users as well as their roles in the system. The usernames and roles are enumerated as well as a delete icon which can delete the specified the selected user if the user is not the currently logged in user
The page can be seen in figure 24.

Figure 24 - Manage users page

There is also an option to add user. A username and role can be simply given to the system (if the username does not already exist). There is a “create user” button which can be used to create the new user and a back button which can be used to cancel the creation of the user. Create user menu can be seen in figure 25.

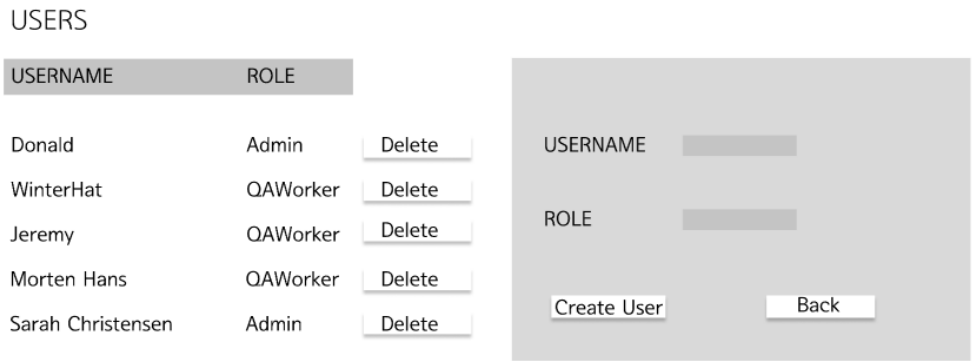


Figure 25 - Manage users page with create section

3.2 System Architecture

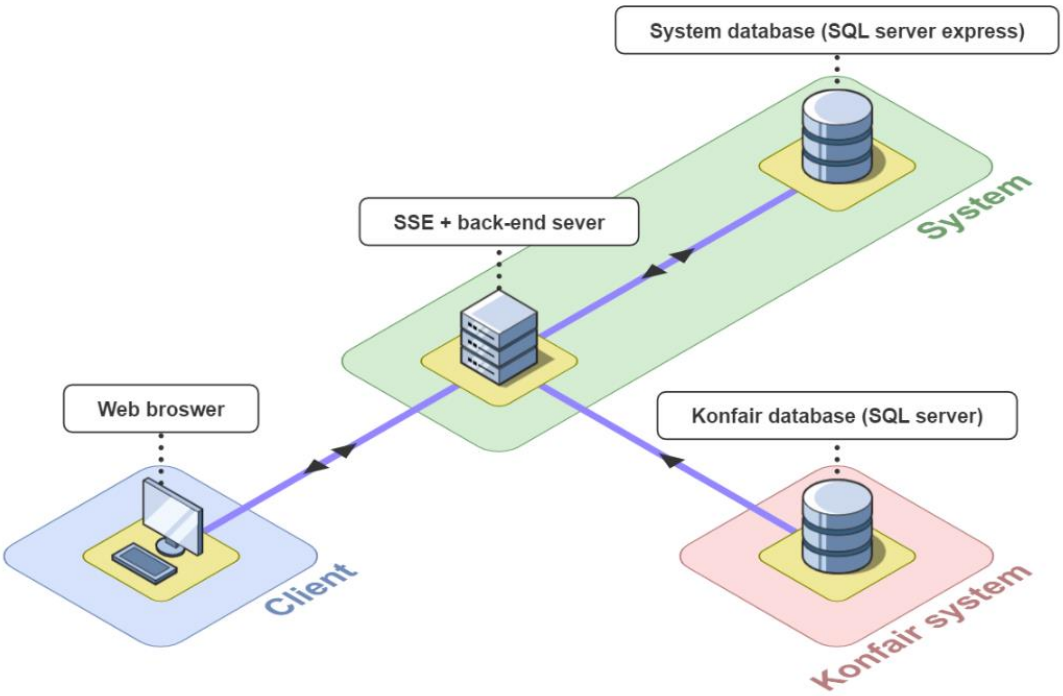
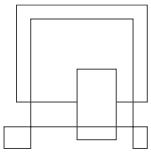


Figure 26 - Architecture diagram

The system provides a web browser UI to the user. The user then uses the UI to make requests for data or make changes to the data in the system. The architecture of the system can be seen in figure 26. Note that this is not a UML diagram.

The server has an API application that plugs into the server which handles the communication to the two databases. This works as the back end of the server while running on the same server.

Konfair needed to make sure that their original database will not be affected by the system. Due to this, the team were given read-only access to their database. Content generated in the system is stored in the system database.



The system can scale in separate pieces. There may be a need for more SSR servers or more back-end performance. The SSR application and back-end can be decoupled from each other. KonfAir can provide more databases or upgrade the servers of the databases when more performance is needed.

The whole architecture can be seen in Class Diagram shown in Appendix 2 – UML Diagrams.

3.2.1 Front-end Architecture

Frontend architecture follows the architecture from the Vue framework (chapter - 3.3.1 Vue.js). For that reason, this part of the system is component-based. In the system, each component is built to perform a specific task. Making use of the SOLID principles facilitated the use of the component-based structure. In the system, they serve as building blocks and provide reusability, extensibility, replaceability, encapsulation, and independence. (Gillin, 2022)

An example of a reusable component is the Navigation bar as it will be used on every page. However, it is important to state that some of the components have the role of pages themselves and those components should not be reused.

On top of that, it follows the technologies which bring the state management pattern (Chapter - 3.3.5 Vuex). The concept of the pattern can be seen in figure 27.

It is important to mention that system does not have a way of handling timeout.

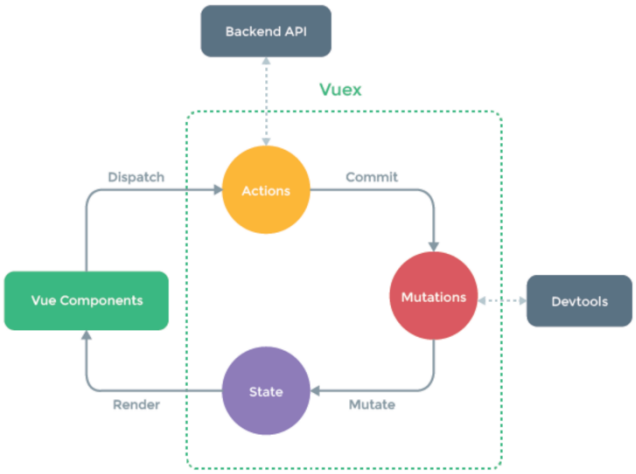


Figure 27 - Vuex state management pattern

3.2.2 Back-end Architecture

The back-end architecture comprises four layers:

- Middleware – handles logic/functionality before getting to the endpoint.
- Route – handles endpoints, and specifies what middleware to use.
- Service – handles business logic of the back-end.
- Model – handles requests to the databases.

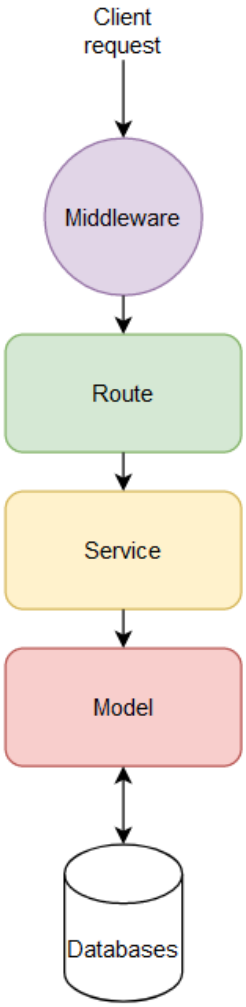


Figure 28 - Back-end architecture overview

This architecture is used to separate concerns between the layers, reduce coupling and improve testability.

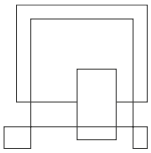
The routes contain requests organized by the pages they serve. For example, route of orders would provide order lists and order information to the order management pages.

The services are not organized in what they return but rather what route they give information back to. For example, the get QA form function in order service should primarily provide the orders route. It is important to note that the services can call each other to use functionality that was already implemented somewhere else.

This way of organizing the back end was chosen based on previous experience with work in the back end. Organizing models based on entity may sound good in theory but in practice, it degrades the structure.

3.3 Front-end Technologies

The front-end is the client browser and the server that generates the pages for the client.



As mentioned before in the UI section, KonfAir wanted a desktop system with the ability to use it on a tablet if needed. Due to this a web browser-based system was chosen. This brings the feature of cross-platform operation which gives the flexibility to use the system on other platforms.

The web application does not need to be installed on the machine; it only needs internet access. A big benefit is that due to the website always being up to date any changes or fixes to the application are applied on every machine instantly instead of updating it on each one. The web application is however sometimes slower than the desktop-based application. (parkersoftware, n.d.)

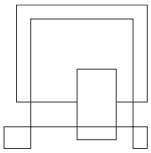
3.3.1 Vue.js

Vue.js is a JavaScript framework that helps build web interfaces. It is built on top of standard HTML, CSS, and JavaScript. (Vue, n.d.)

Comparing Vue.js to other JavaScript frameworks like React and Angular. Vue.js is easier to learn for a developer familiar with standard ways of making websites with HTML, CSS, and JavaScript. (Nowak, 2022)

While all the JavaScript frameworks can get the job done, not all of them have the tools the team needs. Vue does not have all the needed libraries included when you install the framework, missing pieces like routing and global state management must be included with external libraries. Angular is the only framework that has routing and state management included on installation.
The team has the most experience working with Vue.js and slightly less experience with React.

What helped to decide between these frameworks was the availability of widely adopted frameworks that support server-side rendering.



3.3.2 Server-side rendering

The team wanted to go with a server-side rendering. Client-side rendering seemed to be overkill for this project as it is more like a form application rather than a big user experience. Server-side rendering renders the page on the website and then returns the already rendered page to the client. This gives faster time to first render as the client does not request the entire client code on first use of the site, but rather only one page that is already rendered.

Due to the system needing authentication, the server renders the basic components of the page that the data will sit on, and then the client requests the actual data from the back end.

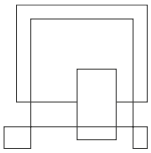
3.3.3 Nuxt.js

Nuxt.js is a server-side rendering framework for Vue.js. It is built on top of Vue.js, it has library compatibility with standard client rendered Vue.js. Nuxt.js has a medium-sized user base of 400 thousand downloads per week meaning there is no lack of already solved problems on the internet that use the framework. (Nuxtjs, n.d.)

Nuxt.js matches a lot of features that a React server-side rendering framework called Next.js has. Next.js has a massive user base of approximately 10 times that of Nuxt.js (Nextjs, n.d.). Both frameworks are massive compared to what Angular has to offer in the server-side rendering domain. Angular's Universal has a user base that is a lot smaller (nguniversal, n.d.).

Due to the similarity of Nuxt.js to Next.js in features and the easier learning curve when going with Nuxt.js, the choice was made to continue with Nuxt.js.

An important thing is that Nuxt in its current stable version is using Vue.js 2. Meaning that functionality from Vue.js 3 will not work



3.3.4 Vuetify

Vuetify is a UI framework. It helps developers build engaging UI by providing working UI components. Additionally, it provides solutions for styling and animations. (vuetifyjs, n.d.)

This library provides reusable components and can replace the need of creating a new component. In this way, it aids component-based structure. It is not used in every place as sometimes it is cleaner and more organized to manually create a component that has functionality that is only needed from this system.

3.3.5 Vuex

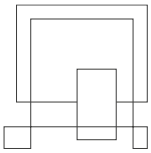
Vuex is a state management pattern for Vue.js, it is used in the system as a centralized storage location for all state that is intended to be shared. Vuex is handled in the store files, there is one for all pages or important components. Each store is divided into three separate categories. The state, mutation functions for the state, and actions that either mutate the state or return something. (VUEX, n.d.)

The actions are called from the pages/components using a dispatch method that specifies what action to call and in what store. It is also used to make calls from it to the back end as it helps handle asynchronous activities in the font-end.

The data is like the data stored in the vue.js components/pages but the difference is that it is available globally throughout the application.
The only way the state of the store can be changed is if it is changed in the mutation functions, otherwise, an error is thrown.

3.3.6 Persist storage

To properly retain data after a page is closed, persistence storage is utilized. By doing so, the data storage outlives the process that was created. This plugin uses local



storage on the browser side. This is used to store vuex data between page changes. This feature is also compatible with Nuxt js.

3.3.7 Observe visibility

The front-end uses a library that observes what the user sees on screen and makes the client logic react when the user reaches a certain part of the page visibly. This is used to make each of the lists of the system infinitely scrollable. Infinite scroll lets the client load only part of the data of a list and in that way improves the performance of the server and the client interface responsiveness.

3.4 Back-end Technologies

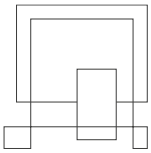
The back end is the API that is integrated into the Nuxt.js server and is run in parallel. A key motivation for the technologies in the following sections is that the team had experience with them before.

3.4.1 Node.js

Both the front-end and the back end use Node.js as their foundation to run JavaScript code. Node.js is a cross-platform JavaScript runtime environment. (Nodejs, n.d.)

Having both the front-end and the back end running on the save runtime environment of node.js gives benefits of code reuse between them. The front-end can use the functions that were meant for the back end, and the back-end can use validation that was done on the front-end to validate the data coming in. Code can be copied over to the other side, and it will work (not the Vue.js structure code and HTML tags). This means that the application can be optimized, to perform best by making decisions about where to put the code. Testing that decision is quick because the code can just be copied over. Decisions are made based on this perception:

- The client-side is where the performance does not cost Konfair or on



- The server-side is where it costs Konfair performance to run but gives a better end-user experience.

Having the same programming language also means the developer does not need to switch to a different syntax when going from the front-end to the back-end.

Node.js only runs a single thread. That single thread is non-blocking to make it very efficient. Non-blocking means that when a request is made from node.js, the single thread does not keep waiting for the response but instead does other tasks while the response is waiting to be received. (Nodejs, n.d.)

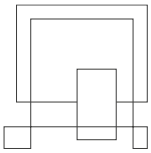
Node.js is good at scalability, the single-core is utilized to its max before it gets to its limit. Konfair does not have many workers at the moment. As the company grows bigger the application will be able to scale by deploying more instances of the application.

Node.js also comes with a very powerful library management system – NPM. Npm provides a lot of publicly available libraries that can be used. A lot of these libraries are utilized in building the application.

Node.js uses JavaScript and that means that Konfair will not have trouble finding developers to help support this system in the future. JavaScript is the most popular programing language at the moment. (StackOveflow, n.d.)

3.4.2 Express.js

Express.js is a minimalistic API framework for node.js. The framework contains the bare essentials to make an API, this lets the developer build only what is needed and add features with additional libraries. The benefit of using express is the possible use of middleware that can be called before the endpoint is reached. The middleware lets the API do things like logging, input validation, or other functionality before the endpoint



is reached. Each endpoint can have specific middleware specified for it. Global middleware that is applied for each endpoint is also possible to specify.

3.4.3 MSSQL Driver

The databases that the API connects to are running SQL Server (T-SQL). One of them is SQL server express which also uses T-SQL but has more limited functionality. (Emma, 2021). Both databases use the same methods of connecting to them.

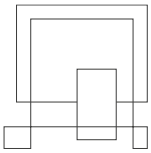
There were two ways of connecting to these databases from the back end. The first was to use an ORM, create local entities of the tables for each of the databases, and then use these entities to manage the connection. For this, the JavaScript library of Prisma was chosen.

Prisma has its own declarative schema definition language which can be written by a developer or scanned from an existing database (Prisma, n.d.). The queries are organized in a similar way as SQL so a developer that knows SQL can instantly know what is going on. It also makes up for the drawbacks of regular ORM by providing more freedom to write queries with complex joins which are essential for an application that needs fast response times to the client.

The team tested this Library with the system database and the Konfair database and found that it struggled to fully recreate the Konfair database tables but worked very well with the system database.

Ultimately a choice was made to not use Prisma and prioritize writing queries using the same library for both databases. It would have not been good to use two different libraries to query one database each just to fix this issue.

To overcome this problem a database connection driver library mssql was used which did not depend on it knowing the exact definition of a table and relied on the developer



knowing the queried database instead. This worked well because each member of the team had a lot of experience with writing raw SQL queries.

The MSSQL driver also had support for windows authentication login, which is what Konfair uses primarily to connect to their databases on their premises.

3.4.4 Express-validator

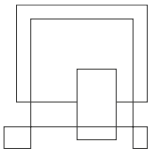
The express-validator is a library that provides a set of middleware that is used in the express API to validate and sanitize data coming in. Although security is not a major concern of Konfair, it is important to care about sanitized data coming into the back end for stability.

Validation does happen on the front-end to make sure users do not unintentionally put bad data into the system. If the user has malicious intent, the front-end validation is easy to overcome, the user can just see what requests are made to the back end with correct information and edit them to have bad information. This is when validation on the back-end is needed.

The express-validator is applied to all endpoints in the back-end that have data passed through the body or passed as a parameter in the URL. Through the express-validator middleware, it is possible to check several inputs such as the username of the user containing a minimum or maximum length. It can also check if integers are in a certain range to not go beyond what the back-end can handle. This way, bogus data is rejected and not passed into the server.

3.4.5 Authentication middleware

Using Express JS allows the creation of middleware to authenticate backend calls. In the system, this is added as an argument in the definition of the endpoints to the back end. For instance, only certain user types should be allowed to perform specific tasks. The middleware validates the authenticity using the passed username in the endpoint.



A query is made to find the user in the database and then see if the user exists and has the needed role for the functionality. If this check passes the request can access the endpoint.

3.4.6 Picture storage

The system has the functionality to store pictures in the system and present them to the client. Pictures are inserted into the system when a control point with a picture is created. The picture is stored as a file in the pictures folder in the API. The picture file name is stored in the database to relate the control point to the picture. The client calls a specific endpoint with the picture name to get the picture from the back end.

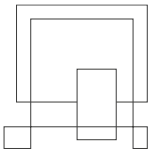
For scalability purposes, the folder where the images are stored can be changed to a specific location so multiple servers can access the same location for the images. This is done through the environment file.

3.4.7 Testability

To help test the functionality of the back end the API is designed to be testable.

The API server is made to be obtained as a function, this lets Nuxt access the API server and integrate it. Additionally, it lets the API server be tested using a library supertest that takes over the API server and helps make requests to it from tests. This is used for route testing (integration testing). (Supertest, 2022).

For unit testing of functions with mocking, the functions in the service and model layers must be made available to the testing libraries in a specific way. Functions must be exported using “module.export.foo” syntax to make them available for mocking. The functions also must be imported to files without using the destructuring functionality available in JavaScript. Destructuring makes the functions unmockable in the testing target. (Sinon, 2022)



3.4.8 Integration with Nuxt.js

This back end is integrated into the Nuxt.js server on startup. This makes the API an extension of Nuxt.js whose paths can be accessed through this syntax when making requests:

- “{base URL}/api/{express endpoint URL}”

Nuxt.js also has its API which can handle the requests in the same way as Express.js would, but it is more restricted and more primitive in functionality as compared to the Express API. Having a separate API integrated into Nuxt.js also means that the API can be decoupled without any problems from the Nuxt.js server and run as a separate service. This helps in a case where SSR and back-end parts scale at different rates and there is a need to scale one side of the server faster than the other. Like having more back-end servers.

3.5 Database

There are two databases that the system connects to. The EER diagram of the databases can be found in appendix 2 – UML diagrams.

3.5.1 Microsoft SQL Server

Microsoft SQL Server is a relational database management system. It has been picked as a technology for storing data as Konfair already uses it when using Microsoft Dynamics 365 NAV. This software uses a SQL Server instance to store its data.

The Nuxt.js API, which uses Express.js, is the only part of the server that can access the databases due to the better control that the back end provides for keeping credentials to the database secure.

3.5.2 System database

This database has read and write access and is the main database that the system uses. It uses SQL Server Express instead of the regular version which is just SQL Server with more limited functionality (Emma, 2021). This is where control point data, QA forms, users, and item category settings are stored. The tables can be seen in figure 29.

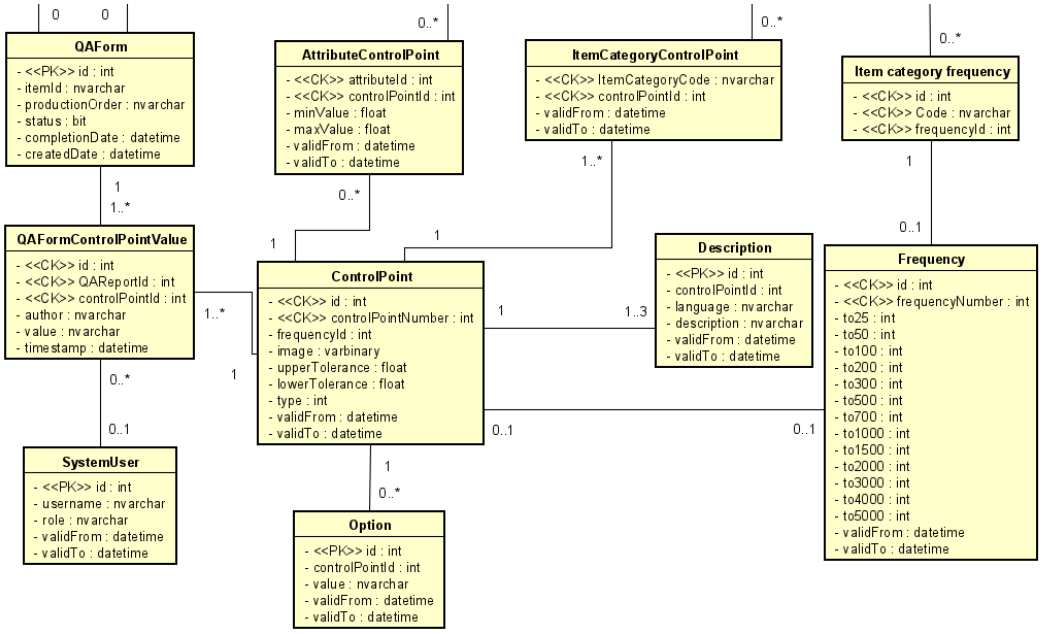
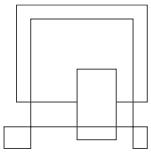


Figure 29 - EER diagram system database

The control point is the biggest table without which the QA forms for the user cannot be generated. When the QA form is generated, connections are made from the control point to the QA form using a value table that connects the two entities. A row is made for each input the user can make into the QA form. The user is linked to a value as an author after he inputs a value into the QA form timestamp is also created for Konfair to know when this input was made.

Most of the system's tables have a "validFrom" and "validTo" column which lets the system store historical data. Historical data is essential to the QA form as once the QA



form is generated it is expected not to change. An example of a problem is a control point description or other value of the control point changing after the QA form has been completed. If there is no historical logic the completed QA form would have its control point data change, this would make the data unreliable.

The QA form uses a “createdDate” column that is made the first time it is created to find the appropriate historical values of control points and their related data.

All the strings in the entities are stored as nvarchar. Nvarchar stores data as 2 bytes per character and can represent up to 4000 characters. Also, a nvarchar column can store up to an 8-bit codepage and is dependent on the database collation for comparisons. Due to the use of Lithuanian and Danish languages in the system, it is important to use nvarchar over varchar as these languages have special characters in them that can be stored by the nvarchar type.

There are no foreign key relationships demonstrated in the EER diagram. As this project is a proof of concept the need for strict relationships were not necessary due to it slowing down development. Each entity still has definitions of composite keys and primary keys.

3.5.3 Customer database

This KonfAir database is read-only because KonfAir cannot risk the system being corrupted or losing data. This database stores tables for Konfair’s Microsoft Dynamics Navision system that the customer uses in parallel to this system. The tables that are being used by the system are shown in the EER diagram in figure 30.

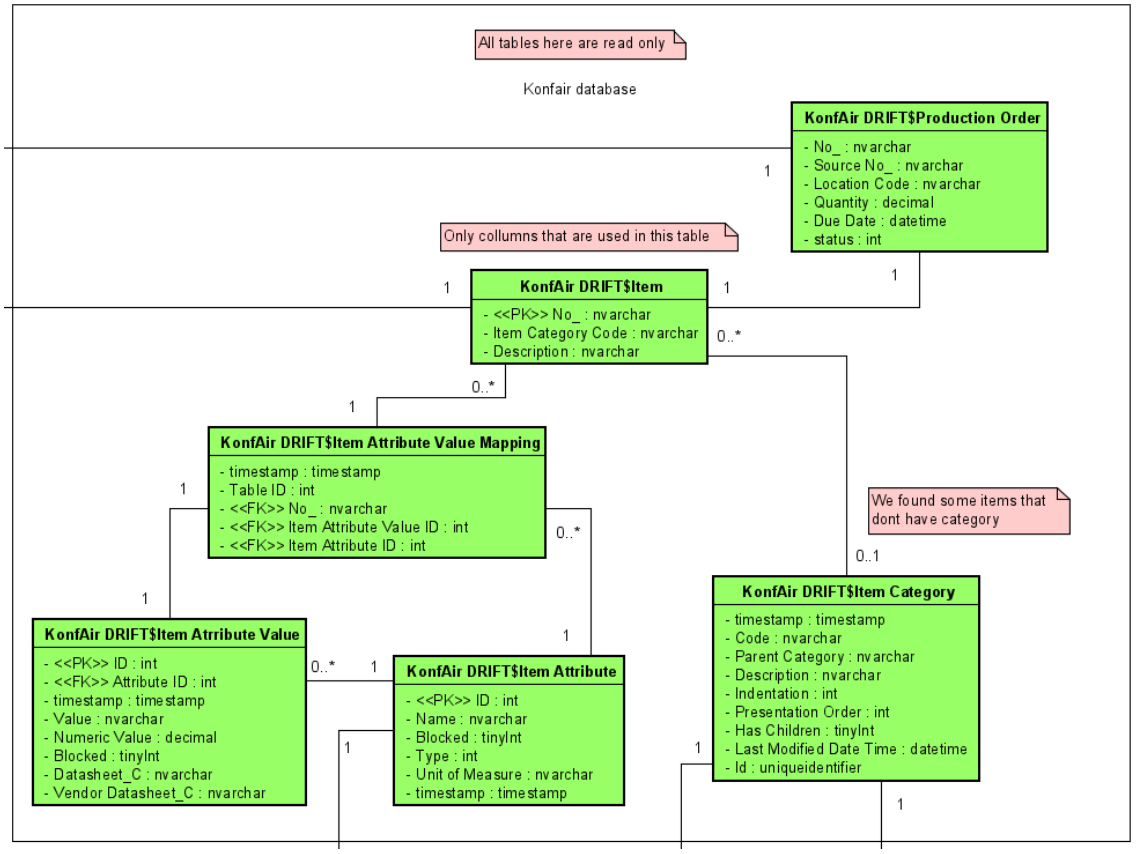


Figure 30 - EER diagram Customer database

The main entity in the customer database is the production order which is related to a single item, the status of the production order can be used to determine if there is an order that needs a QA form created for it. The production order generally holds various information about the order.

The item has attributes like Height, width, material type etc. related to it which also have a value associated with them that is specified by the Konfair customer when they make an order for an item. An item also has an item category.

This is all that is needed from the customer database to generate QA forms. The fields that are not used from the KonfAir database are not shown.

4 Implementation

As an example of implementation – display QA unfinished form of order for QA worker.

The process starts at the front end when the user gets redirected or goes to the QA form page. The page file can be seen in figure 31.

The source code of the application can be found in appendix 4 – Source code.

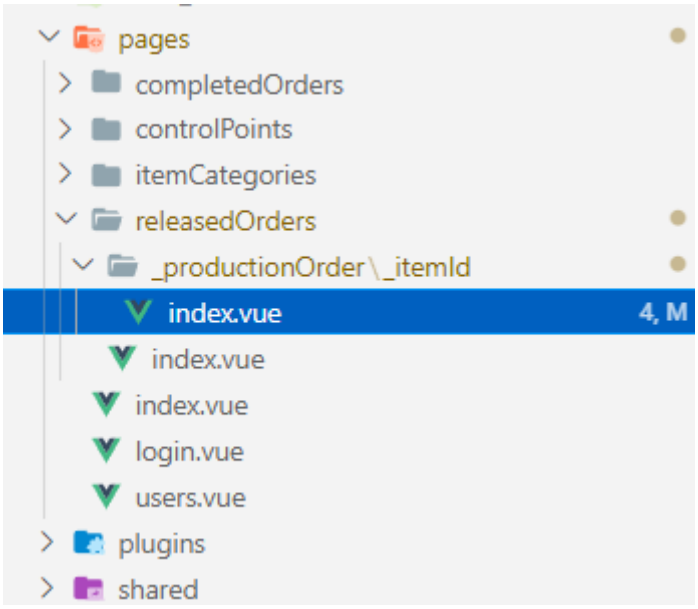
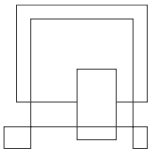


Figure 31 - unfinished QA form page file

The page first checks if the user is validated with the correct role, before rendering the page. Then to render the rest of the page a check is made to see if the order has been loaded from the back end or if a notification with a response is available. This can be seen in figure 32. If there is none of these available, the page does not show anything until the state changes.

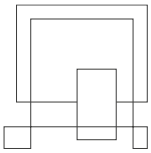


```
<template>
    <div
        :v-if="
            this.$store.state.login.user &&
            this.$store.state.login.user == 'qa employee'
        "
    >
        <AlertModal
            :id="1"
            :message="currentOrder && currentOrder.message"
            :show="modalAlertShowError"
            :status="errorStatus"
            :closeCallback="closeAlertModal"
        />
        <div
            v-if="currentOrder && currentOrder.response == null"
            class="releasedOrder"
        >
            <ImageModal
```

Figure 32 - UI state management in HTML elements

The page calls a created function after it renders at least part of the UI. This function is used to fetch the order and QA form information from the back end. This function can be seen in figure 33. The data request is made to the application store which handles the global state of the application. The request is asynchronous and has to be awaited to get a response, to do this the JavaScript syntax of .then() is used which is activated when the promise returns something. The result of this is set to the state of the page, if the result is a notification, it sets the alert modal of the page to be shown.

The parameters of the production order number and item id are obtained from the URL of the page.



```
created() {
    this.$store
        .dispatch(
            "releasedOrder/loadReleasedOrderFull",
            {itemId: this.$route.params.itemId, productionOrder: this.$route.params.productionOrder}
        )
        .then((result) => {
            if(result && result.response != null){
                this.modalAlertShowError = true;
            }

            this.currentOrder = result;
        });
},
```

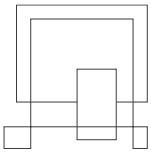
Figure 33 - Created handler function

The store receives the request with the specified parameters. The store gets the state of another store that contains the details of the currently logged-in user and validates that the user exists and is of the correct role. It then creates a request with the parameters to the back-end. The request is made as a promise that resolves when the request finishes, this way the data can be returned to the page from where it was called when it is obtained. The store function can be seen in figure 34.

```
export const actions = {
    loadReleasedOrderFull({ rootState }, parameters) {
        const user = rootState.login.user
        if (user && user.role == "qa employee") {
            const language = rootState.login.chosenLanguage.name

            return new Promise((resolve, reject) => {
                fetch(`/api/orders/released/full/${user.username}/${parameters.itemId}/${parameters.
                    .then(res => res.json())
                    .then(result => {
                        if (result && result.errors == null) {
                            resolve(result)
                        } else {
                            resolve(null)
                        }
                    })
            })

        }
    },
```

Figure 34 - Store function for getting order information and QA form

The request first goes through the middleware that is applied to each of the endpoints in the back-end. These global middleware declarations can be seen in figure 35.



```
// Functions that are called before the actual endpoint is reached
function initializeMiddleware(app) {
    app.use(helmet())
    app.use(express.json({limit: '50mb'}));
    app.use(express.urlencoded({limit: '50mb'}));
    app.use(express.json())
    if(process.env.LOGGING == "true"){
        app.use(require("./middleware/loggingMiddleware"))
    }
}
```

Figure 35 - Global middleware functions

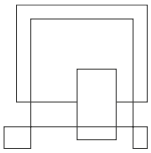
Then the request goes through the validation middleware that is specified in the endpoint. This is some of the functionality that the express-validator handles, these are middleware functions provided by it. In this case, they check the length of the inputs. The endpoint with the middleware and jsdoc can be seen in figure 36.

The endpoint has information describing the endpoint in the jsdoc comment text, seen in green.

After validating all inputs, a middleware call is made to check if all the inputs were valid or if not, the client gets an error response with information that was wrong.

After the express-validate checks are done for the parameters another check is made now using the validated data to check if the user is of the correct role. This is the reason why the username is passed to every single request, to check if the user can access the functionality.

Finally, if everything passes the actual endpoint is accessed which gets the data from services using the parameters passed and then sends it back to the client.

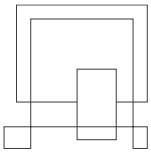


```
/**
 * @description - Get the released order by id
 * @param username - name of the user for validation
 * @param id - id of the selected order
 *
 * @example - GET {BaseURL}/api/orders/released/full/worker/47827/gb
 */
router.get("/released/full/:username/:id/:productionOrder/:language",
    param("username").isLength({ min: 4, max: 50 }),
    param("id").isLength({ min: 1, max: 35 }),
    param("productionOrder").isLength({ min: 1, max: 35 }),
    param("language").isLength({ min: 2, max: 20 }),
    validate,
    validateUserQA,
    async (req, res) => {
        const data = await service.getQAReport(req.params.id, req.params.product
        res.send(data)
    }
)
```

Figure 36 - Endpoint with jsdoc and middleware

This is a deeper look into the validateUserQA middleware. The function can be seen in figure 37. This middleware takes the username parameter that was passed to the endpoint and uses it to get the user from the database. The function then checks if the user is of the correct role. This is done because the user passing the role would make it very easy to break the system. This is why such active checks are made.

If the user is authenticated, the middleware calls the next middleware in line, in this case, it will go straight to the actual endpoint code as this was the last middleware. If the user does not exist or is not of the correct role, a status code is returned to the client and no further middleware or endpoint is accessed.



```
module.exports.validateUserQA = async (req, res, next) => {
    const user = await userModel.getUserByUsername(req.params.username)

    if (user[0] && user[0].role == "qa employee") {
        next()
    } else {
        res.sendStatus(403)
    }
}
```

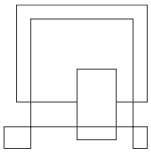
Figure 37 - validateUserQA middleware

This is a deeper look into the service function called by the route. The function can be seen in figure 38. This function has extra parameters that specify if the output is to show the author names of those users who made the inputs and special settings for orders that should be with completed status.

The function is exported using the module exports syntax to help mock the function in tests.

The function first gets the order information which is a combination of the item and production order information. If nothing is returned from the database, the service returns a notification with response 0 which means failure and a message on why it failed. This will be displayed to the client in an alert modal that was talked about earlier.

An example of the model function which accesses the database will be shown later.

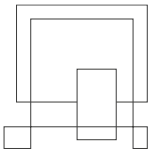


```
/**
 * get a qa report object of specified order with all of the measurements and information included in
 * @param {string} id id of the order
 * @param {string} language language that will be preferred for the information retrieved
 * @param {boolean} showAuthors should the order contain the real author names in the answers
 * @param {boolean} getCompleted should the order be completed or released
 * @returns {Promise} Order, its one time measurement and multiple time measurement control points, ar
 */
module.exports.getQAReport = async (id, productionOrder, language, showAuthors, getCompleted) => {
    // Get a detailed item object from konfair database. Returns array of orders that match this id
    let itemData = await model.getOrderInformation(id, productionOrder)

    // If it exists continue process
    if (itemData && itemData.length != 0) {…
    } else {
        return { response: 0, message: "The order does not exist in the database" }
    }
}
```

Figure 38 - getQAReport function part 1

If the data exists, the function continues. The function gets the QA form from the system database and goes through some checks that depend if the function is getting an order with a QA form that is completed or not. It also checks if the order was released or not.
Then it checks if the QA form for the order has been created in the system database. If it has not been created the function creates relations from the control point to a new QA form and continues with getting the QA form. The code for this functionality can be seen in figure 39.

If the QA form already exists in the system database, the attributes for the item are gotten instead.

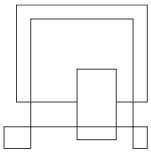


```
// If it exists continue process
if (itemData && itemData.length != 0) {

    // get the first one of the orders. Others dont matter as there should not be more than one o
    itemData = itemData[0]

    // Check if there exists a QA report for this order
    let qaReport = await model.getReleasedOrderReport(id, productionOrder)

    // Check if the exists a qa report for this or not and its status
    if (!getCompleted && qaReport.length != 0 && qaReport[0].status == 1)
        return { response: 0, message: "Order is completed" }
    else if (getCompleted && qaReport.length != 0 && qaReport[0].status == 0)
        return { response: 0, message: "Order is not completed" }
    else if (getCompleted && qaReport.length == 0)
        return { response: 0, message: "Order has not been initialized and completed" }

    // We cannot expect their database state of released or whatever item to change when the orde
    // finished on our part of the system. So a released order can only be released and completed
    // can be whatever status
    if (!getCompleted && itemData.status != 3) {
        return { response: 0, message: "Order has not been released" }
    }

    let controlPoints = null
    let attributes = null

    // If the qa report doesn't exist and this is not a completed order operation
    // Generate the connections for the control points
    if (qaReport.length == 0 && !getCompleted) {…
    } else {

        qaReport = qaReport[0]

        // Get all the attributes that belong to this order
        attributes = await model.getReleasedOrderAttributes(id)
    }
```

Figure 39 - getQAReport function part 2

In the case that the QA form does not exist in the system database the function fetches the attributes associated with the item id. If the item does not have attributes a message is returned. This can be seen in figure 40.

The function then gets all control points that have connections to such attribute ids and this item category. A helper function is used that just generates a string from the specified object parameters. They are separated by a comma. This is done because the SQL query uses the IN syntax which checks that the value of the specified column matches one of the specified values separated by a comma. This function is for



numbers. There is another that does the same but also puts quotes on each value in the string. This is used when the column being checked is a string.

Another check is made for control points that only have the same item category as the item and don't have any attribute connections.

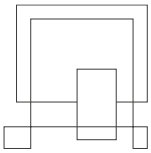
The control points obtained are merged and the attributes for all of them are fetched. Then they are iterated over for checks of applicability to the item attached to the production order.

```
// If the qa report doesn't exist and this is not a completed order operation
// Generate the connections for the control points
if (qaReport.length == 0 && !getCompleted) {

    // Get all the attributes that belong to this order
    attributes = await model.getReleasedOrderAttributes(id)

    // If the order doesn't have any attributes it is bad.
    if (attributes.length == 0)
        return { response: 0, message: "This order does not have any attributes in it" }


    // Get control points that connect to these attributes and are for this categoryCode
    controlPoints = await model.getSpecificControlPoints(module.exports.listToCommaString(attributes, 'id'), itemData.c

    let controlPointsWithCategory = await model.getControlPointsCategoryNoAtrributes(itemData.categoryCode)

    controlPoints.push(...controlPointsWithCategory)
    // Get all the attributes and item categories of these control points
    // Will later validate which one connects to which one
    for (let i = 0; i < controlPoints.length; i++) {
        controlPoints[i].attributes = await model.getControlPointAttributesLatest(controlPoints[i].id)
    }

    let added = []
    // Determine if the control point is applied to tje
    for (let i = 0; i < controlPoints.length; i++) {
```

Figure 40 - getQAReport function part 3

The loop does the following checks and adds the control point when the following conditions are met:

- The item category is the same as the item category code and the control point does not have any attributes. (This passes for all control points not having attributes because only ones with the same category code were retrieved.)
- The item category is the same and the range of connection is null (minValue and maxValue)



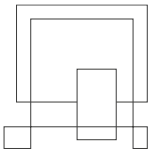
- The item category is the same and the value of the attribute of the item is the range (the number is between minValue and maxValue)

The code can be seen in figure 41.

```
for (let i = 0; i < controlPoints.length; i++) {

    // If no attributes then it is a general control point for this category
    if (controlPoints[i].attributes.length == 0) {
        added.push(controlPoints[i])
        continue;
    }

    attributes:
    for (let j = 0; j < controlPoints[i].attributes.length; j++) {
        for (let k = 0; k < attributes.length; k++) {
            // The control point must be linked to at least one attribute
            if (attributes[k].id == controlPoints[i].attributes[j].id) {
                // Control point without a range

                if (controlPoints[i].attributes[j].maxValue == null && controlPoints[i].attributes[j]
                    added.push(controlPoints[i])
                    // Break out of both loops using the label
                    break attributes;
                }

                // Control point with range which attribute value has to meet
                if (controlPoints[i].attributes[j].maxValue >= parseFloat(attributes[k].value) &&
                    controlPoints[i].attributes[j].minValue <= parseFloat(attributes[k].value)) {
                    added.push(controlPoints[i])
                    break attributes;
                }
            }
        }
    }
}
```

Figure 41 - getQAReport function part 4

The function then continues and checks if there were control points that met the criteria. This can be seen in fig. 41. If there were at least one control point that met the criteria a QA form is created in the system database along with all the connections to



control points assigned to the QA form. If none applied a notification with information useful to the user is sent back.

```
// Add all of the control point connections to this order
if (added.length != 0) {
    // Add qa report
    qaReport = await model.createQAReport(id, productionOrder)
    qaReport = qaReport[0]

    // Add the connections between control point and qa report
    for (let i = 0; i < added.length; i++) {
        await model.insertControlPointConnection(added[i].id, qaReport.id)
    }

} else {
    // The order does not have control points that apply to it
    return { response: 0, message: "This order does not have any control points that apply to it" }
}
```

Figure 42 - getQAReport function part 5

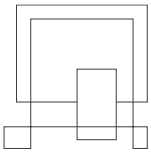
Once the order has a QA form created for it the function continues gathering data for the UI as before. The code for this can be seen in fig. 43. The item category is used to get the frequency. If the category does not have a frequency assigned to it, a default frequency is used. The default frequency is available throughout the whole application.

The control point connections that relate to the QA form id are fetched. The control points are retrieved with author names or anonymous author names. The description of the control point is also gotten, based on the language specified.

Only control points that were valid on the date when the QA form was created are retrieved, this is to prevent the control point updates from changing the data that the QA form is connected to. This is possible because a lot of entities in the database have the validFrom and validFrom columns that are used when the entity changes. Without this feature, the information in the completed QA forms would update every time the administrator makes changes to the control point information.

The attributes of the control points are obtained as well, also using the creation date of the QA form.

The control points are then iterated over to get the other parameters.

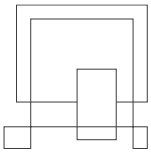


```
    // Get all the attributes that belong to this order
    attributes = await model.getReleasedOrderAttributes(id)
}


// Get the frequency of the category and apply it to the
itemData.frequency = await model.getFrequenciesForCategory(parseInt(itemData.categoryCode), qaReport.cre
if (itemData.frequency && itemData.frequency.length != 0)
    itemData.frequency = itemData.frequency[0]
else
    itemData.frequency = itemCategoryService.defaultFrequency


// Fetch all the control points that relate to this qa report. With author anonymity or not
if (showAuthors) {
    controlPoints = await model.getReleasedOrderControlPointsAuthors(qaReport.id, language, qaReport.cre
} else {
    controlPoints = await model.getReleasedOrderControlPoints(qaReport.id, language, qaReport.createdDat
}

// Get all the attributes and item categories of these control points
for (let i = 0; i < controlPoints.length; i++) {
    controlPoints[i].attributes = await model.getControlPointAttributes(controlPoints[i].id, qaReport.cr
}

// Get all the data for the control point: descriptions, frequency, options
for (let i = 0; i < controlPoints.length; i++) {
```

Figure 43 - getQAReport function part 6

First, the timestamp of when the input in the control point was updated is translated to a human-readable date and time. The frequency of the control point is gotten as well as the options of the control point if it has an input type of option (zero).

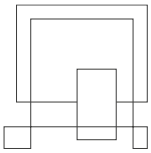
After the control points are updated, some data about the order is applied to the object that will be returned. The status of the QA form is translated into something human-readable form of completed or incomplete. This can be seen in figure 44.



```
// Get all the data for the control point: descriptions, frequency, options
for (let i = 0; i < controlPoints.length; i++) {
    if(controlPoints[i].timestamp != null){
        const date = new Date(controlPoints[i].timestamp)
        controlPoints[i].timestamp = moment(date).format('YYYY-MM-DD HH:mm')
    }

    // Get frequency of the control point
    controlPoints[i].frequency = await model.getFrequencies(controlPoints[i].frequencyId, qaReport.c
    if (controlPoints[i].frequency && controlPoints[i].frequency.length != 0)
        controlPoints[i].frequency = controlPoints[i].frequency[0]
    else
        controlPoints[i].frequency = null

    // Get options if the control poi
    if (controlPoints[i].inputType == 0) {
        controlPoints[i].options = await model.getReleasedOrderControlPointsOptions(controlPoints[i]
    }
}

// Add correct status based on order being retrieved
if (getCompleted) {
    itemData.status = "completed"
} else {
    itemData.status = "incomplete"
}

itemData.qaReportId = qaReport.id
itemData.completionDate = qaReport.completionDate
itemData.oneTimeControlPoints = []
itemData.multipleTimeControlPoints = []

// Categorize the control points into one time or multiple time
```

Figure 44 - getQAReport function part 7

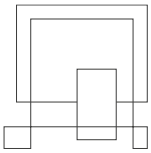
A loop then iterates over the control points to provide them with information from the item attributes. Values like actual item attribute value and the units of that attribute are assigned in this step. The control point is assigned to one-time or multiple-time measurement inputs on the UI, based on the measurement type that they have. The human-readable tolerance text is also computed for the control point if it has tolerances. The code can be seen in figure 45.



```
// Categorize the control points into one time or multiple time
for (let i = 0; i < controlPoints.length; i++) {

    // Assigns the attribute parameters to the control point
    if (controlPoints[i].attributes.length != 0) {
        loop:
        for (let j = 0; j < controlPoints[i].attributes.length; j++) {
            for (let k = 0; k < attributes.length; k++) {
                if (controlPoints[i].attributes[j].id == attributes[k].id) {
                    controlPoints[i].expectedValue = attributes[k].value
                    controlPoints[i].units = attributes[k].units
                    break loop
                }
            }
        }
    }

    // Attributes no longer needed after the assigning
    delete controlPoints[i].attributes

    // One time measurement
    if (controlPoints[i].measurementType == 1)
        itemData.oneTimeControlPoints.push(controlPoints[i])
    else
        itemData.multipleTimeControlPoints.push(controlPoints[i])

    // Compute the text shown for the tolerance
    controlPoints[i].toleranceText = ""
    if (
        controlPoints[i].upperTolerance != null &&
        controlPoints[i].lowerTolerance != null &&
        controlPoints[i].upperTolerance != 0 &&
        controlPoints[i].lowerTolerance != 0 &&
        controlPoints[i].upperTolerance == controlPoints[i].lowerTolerance
    )
        controlPoints[i].toleranceText = "+/-" + controlPoints[i].upperTolerance + controlPoin
    else if (
        controlPoints[i].upperTolerance != null &&
        controlPoints[i].lowerTolerance != null &&
        controlPoints[i].upperTolerance != 0 &&
        controlPoints[i].lowerTolerance != 0 &&
        controlPoints[i].upperTolerance != controlPoints[i].lowerTolerance
    )
        controlPoints[i].toleranceText = "+" + controlPoints[i].upperTolerance + "/-" + contro

    // Compute the options if the control point input type is option
    if (controlPoints[i].inputType == 0) {
        let allUnits = ""
```

Figure 45 - getQAReport function part 8

The unit for the control point is assigned. The unit text for the option is computed by concatenating each of the options together separated by a slash. The unit for text input type is text and the number uses the units that it got from the attribute. The code can be seen in figure 46.



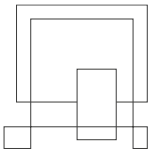
```
    controlPoints[i].upperTolerance != controlPoints[i].lowerTolerance
)
    controlPoints[i].toleranceText = "+" + controlPoints[i].upperTolerance + "/-" + contr

// Compute the options if the control point input type is option
if (controlPoints[i].inputType == 0) {
    let allUnits = ""

    for (let j = 0; j < controlPoints[i].options.length; j++) {
        if (j == 0) {
            allUnits = controlPoints[i].options[j].value
        } else {
            allUnits = allUnits + "/" + controlPoints[i].options[j].value
        }
    }

    controlPoints[i].units = allUnits
    // Assign units of text if input type text
} else if (controlPoints[i].inputType == 1) {
    controlPoints[i].units = "Text"
}
```

Figure 46 - getQAReport function part 9

After gathering all the data for the base control points and separating them into multiple time measurements the function goes into the phase of collecting the multiple time answers of the QA form.

To start this the frequency is calculated based on the quantity parameter of the production order. This is done by finding the correct key for the frequency based on the size of the quantity. This key will be later used to get the correct frequency from the frequency objects.

The available multiple time control point ids are then used to get the rest of the answers for these control point connections. There can be multiple answers for each of the multiple time control points, and all of them are retrieved. Once again the helper function is used to get a string of numbers separated by a comma in string form. The code can be seen in figure 47.

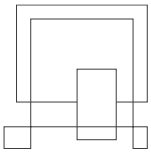


```
// Character to mark each multiple time control time
// Will be incremented for other letters in alphabet
let currentChar = 'a'
let answersMulti = []
let frequencyCategoryKey = ""


// FInd the correct frequency based on the quantity
if (itemData.quantity <= 25) {
    frequencyCategoryKey = "to25"
} else if (itemData.quantity <= 50) {
    frequencyCategoryKey = "to50"
} else if (itemData.quantity <= 100) {
    frequencyCategoryKey = "to100"
} else if (itemData.quantity <= 200) {
    frequencyCategoryKey = "to200"
} else if (itemData.quantity <= 300) {
    frequencyCategoryKey = "to300"
} else if (itemData.quantity <= 500) {
    frequencyCategoryKey = "to500"
} else if (itemData.quantity <= 700) {
    frequencyCategoryKey = "to700"
} else if (itemData.quantity <= 1000) {
    frequencyCategoryKey = "to1000"
} else if (itemData.quantity <= 1500) {
    frequencyCategoryKey = "to1500"
} else if (itemData.quantity <= 2000) {
    frequencyCategoryKey = "to2000"
} else if (itemData.quantity <= 4000) {
    frequencyCategoryKey = "to4000"
} else if (itemData.quantity <= 5000) {
    frequencyCategoryKey = "to5000"
}


// Get the list of connections for the multiple time control points
// These have the answers and the authors to those answers
let mIdList = listToCommaString(itemData.multipleTimeControlPoints, 'id')
let mResults = []
if (mIdList != '') {
    if (showAuthors) {
        mResults = await model.qaReportControlPointResultsAuthors(itemData.qaReportId, mIdLis
    } else {
        mResults = await model.qaReportControlPointResults(itemData.qaReportId, mIdList)
    }
}
```

Figure 47 - getQAReport function part 10

Each of the multiple time control points is assigned a letter that will connect the control point row to its answers column in the UI.

The correct answers are picked off the answer list and stored. The actual frequency value is obtained using the key that was gotten in the previous section. The default category frequency is used if the control point has no frequency of its own.



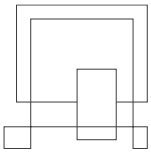
The value of the frequency that is gotten from the frequency object is used as the number of iterations in a for loop to define how many answers that control point should have. The code can be seen in figure 48.

```
for (let i = 0; i < itemData.multipleTimeControlPoints.length; i++) {
    // TUrn the character to uppercase
    itemData.multipleTimeControlPoints[i].letter = currentChar.toUpperCase()

    let arrayOfAnswers = []

    // Get the connections that belong to this control point
    let mResultsForControlPoint = []
    for (let j = 0; j < mResults.length; j++) {
        if (mResults[j].controlPointId == itemData.multipleTimeControlPoints[i].id)
            mResultsForControlPoint.push(mResults[j])
    }

    // Get the correct frequency for this control point
    let assignedFrequency = null
    // If control point freq is not existent then assign the category frequency
    if (itemData.multipleTimeControlPoints[i].frequency == null && itemData.frequency != null
        assignedFrequency = itemData.frequency[frequencyCategoryKey]
    // If the control point has a frequency
    else if (itemData.multipleTimeControlPoints[i].frequency != null)
        assignedFrequency = itemData.multipleTimeControlPoints[i].frequency[frequencyCategory

    // Generate enough rows for this control point as the frequency dictates
    for (let j = 0; j < assignedFrequency; j++) {…
    }
```

Figure 48 - getQAReport function part 11

The loop then picks off answers that belong to the control point one by one from the start of the array, in this process the first element of the array is also deleted. It formats them into answer objects that contain the basic information about the connection as well as the author and timestamp of the input of the connection.

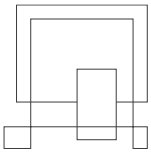
If there are too few answers available to satisfy the required amount, the function creates more of them in the system database. This is a scenario that only happens when the QA form is recently created. The loop can be seen in figure 49.



```
// Generate enough rows for this control point as the frequency dictates
for (let j = 0; j < assignedFrequency; j++) {

    let answer = ""
    let author = ""
    let connectionId = ""
    let timestamp = null

    // Get the first value from answers
    if (mResultsForControlPoint.length != 0) {
        answer = mResultsForControlPoint[0].answer
        author = mResultsForControlPoint[0].author
        connectionId = mResultsForControlPoint[0].connectionId
        timestamp = mResultsForControlPoint[0].timestamp
        if(timestamp != null){
            const date = new Date(timestamp)
            timestamp = moment(date).format('YYYY-MM-DD HH:mm')
        }
        mResultsForControlPoint.shift() // Remove the first element
        // If there are no more values then generate some empty ones to fill it up
    } else {
        // If order doesn't have the needed answers then it is not completed but broken
        if (getCompleted) {
            return { response: 0, message: "Order has not been fully initialized and completed" }
        } else {
            // Add the thing to the database and give the connectionID to it
            let insertResponse = await model.insertMultipleTimeMeasurement(itemData.multipleTimeCon
            connectionId = insertResponse[0].id
        }
    }

    // Add the row to the list
    arrayOfAnswers.push(
        {
            connectionId: connectionId,
            id: itemData.multipleTimeControlPoints[i].id,
            answer: answer,
            inputType: itemData.multipleTimeControlPoints[i].inputType,
            author: author,
            timestamp: timestamp,
        })
}
```

Figure 49 - getQAReport function part 12

The gathered answer objects are then sorted based on the connectionId (this is the id of the connection entity, all are unique) to make them appear in the same order on the UI every time. This can be seen in figure 50. The character that connects the control point to the answers column is then incremented. This gets the ASCII number of the character and then increments it, turning it back to character gives a character that is incremented. This process is repeated for all multiple time control points.



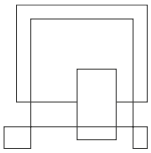
```
                timestamp: timestamp,
            })
        }

        // Sort these so that the order stays the same in the ui every time
        arrayOfAnswers.sort((a, b) => (a.connectionId > b.connectionId) ? 1 : -1)

        answersMulti.push(arrayOfAnswers)

        // Get the next character a -> b -> c ...
        currentChar = String.fromCharCode(currentChar.charCodeAt(0) + 1)
    }
```

Figure 50 - getQAReport function part 13

The final steps of the function then clean up the times and dates of the returned data as well as deletes the parameters that are not used by the client to reduce the size of the data that needs to be transferred. This can be seen in figure 51.

```
    itemData.multipleTimeAnswers = answersMulti

    // Convert the deadline to only have date
    const date = new Date(itemData.deadline)
    itemData.deadline = moment(date).format('YYYY-MM-DD')

    if (itemData.completionDate) {
        const completionDate = new Date(itemData.completionDate)
        itemData.completionDate = moment(completionDate).format('YYYY-MM-DD')
    }

    // Clean up the data before sending to reduce size of payload
    delete itemData.frequency
    for (let i = 0; i < itemData.oneTimeControlPoints.length; i++) {
        delete itemData.oneTimeControlPoints[i].frequency
    }
    for (let i = 0; i < itemData.multipleTimeControlPoints.length; i++) {
        delete itemData.multipleTimeControlPoints[i].frequency
    }

    return itemData
} else {
    return { response: 0, message: "The order does not exist in the database" }
}
```

Figure 51 - getQAReport function part 14

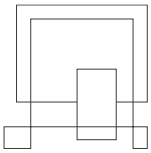


This is an example of the of one of the modal functions that query the database. The models use two different database connections. One for the system database and one for the KonfAir database. The local (system database) query can be seen in figure 52.

The query is asynchronous and the getting of the connection object for a database is also asynchronous because of the singleton.

The parameters passed to the function are passed into the query using inputs to prevent SQL injection

In this specific query, the data being obtained is unique control points that connect to the QA form entity. Distinct and max functionality is used to help the query get only the unique control points and not all the multiple time answers. The validity of the control point entity is also checked to be either between the date or be valid from before the date.



```
module.exports.getReleasedOrderControlPoints = async (id, language, date) => {
    // The max statements are to help group by
    const result = await (await localDB())
        .request()
        .input("id", mssql.Int, id)
        .input("language", mssql.NVarChar(40), language)
        .input("date", mssql.DateTime, date)
        .query(`
            SELECT
            DISTINCT
            point.id,
            MAX(point.image) as image,
            MAX(point.frequencyId) as frequencyId,
            MAX(point.inputType) as inputType,
            MAX(point.lowerTolerance) as lowerTolerance,
            MAX(point.upperTolerance) as upperTolerance,
            MAX(point.measurementType) as measurementType,
            MAX(CASE WHEN connection.author IS NULL THEN '' WHEN connection.author = '' THEN '' ELSE 'taken' END) as author,
            MAX(connection.id) as connectionId,
            MAX(connection.value) as answer,
            MAX(connection.timestamp) as timestamp,
            MAX(description.description) as description
            FROM [QAReportControlPointValue] connection
            INNER JOIN [ControlPoint] point ON connection.[controlPointId] = point.[id]
            INNER JOIN [Description] description ON connection.[controlPointId] = description.[controlPointId]
            WHERE
                connection.[qaReportId] = @id AND
                description.language = @language AND
                point.validFrom < @date AND
                ( point.validTo > @date OR point.validTo IS NULL) AND
                description.validFrom < @date AND
                ( description.validTo > @date OR description.validTo IS NULL)
            Group by point.id
        `)
    return result.recordset
}
```

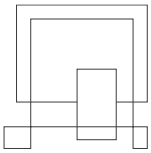
Figure 52 - getReleasedOrderControlPoints function

The connection to the databases is initialized only when they are needed for the first time. After the first time the connection is reused. This can be seen in figure 53.

```
module.exports.localDB = async () => {
    if (localDB == null) {
        try {
            await module.exports.getConnectionsOwn()
            return localDB

        } catch (err) {
            console.log("[" + new Date().toLocaleString() + "] - [ERROR] - Failed to connect to database - Own")
            return null
        }
    }else{
        return localDB
    }
}
```

Figure 53 - get database connection method

There are multiple options for connecting to a database. Two of these are for testing and the application running on Konfair's server uses the KonfAir production connection. The KonfAir production connection uses a different version of mssql library to connect using windows authentication of the machine. When not on Konfair's server the



application uses the basic connection string to connect to a database. This an be seen in figure 54.

```
module.exports.getConnectionsOwn = async () => {
    if (process.env.environment != "testing") {
        if (process.env.DATABASE == "konfairProduction" || process.env.DATABASE == "konfairTesting") {
            localDB = await get("Own", {
                database: "Own",
                server: "SRVAPP3\\SQLEXPRESS",
                driver: "msnodesqlv8",
                options: {
                    trustedConnection: true
                }}, false)
        } else if (process.env.DATABASE == "local") {
            localDB = await get("Own", "Server=localhost,1433;Database=own;User Id=sa;Password=konf123!proj;Enc
        } else {
            localDB = await get("Own", "Server=bpr2.database.windows.net,1433;Database=own;User Id=rafal;Passwo
        }
    }
}
```

Figure 54 - method that triggers connection to database

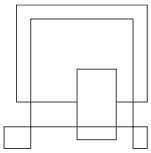
The connections are made as a connection pool, this was needed as the mssql library doesn't support multiple different connections otherwise. This an be seen in figure 55.

```
let konfairDB;
let localDB;

const get = (name, config, useNormalMssql) => {
    if (!pools.has(name)) {
        if (!config) {
            throw new Error('Pool does not exist');
        }
        let pool
        if(useNormalMssql){
            pool = new mssql.ConnectionPool(config);
        }else{
            pool = new mssqlV8.ConnectionPool(config);
        }
        // automatically remove the pool from the cache if `pool.close()` is called
        const close = pool.close.bind(pool);
        pool.close = (...args) => {
            pools.delete(name);
            return close(...args);
        }
        pools.set(name, pool.connect());
    }
    return pools.get(name);
}
```

Figure 55 - connection method

After the data is received from the back end if the result contained an error. Then an alert is shown to the user.

The alert modal state is changed to true which sets the parameter that is passed as props to the component as true. This makes the component render on the page. The component can be seen in figure 56.



```
<AlertModal
    :id="1"
    :message="currentOrder && currentOrder.message"
    :show="modalAlertShowError"
    :status="errorStatus"
    :closeCallback="closeAlertModal"
/>
```

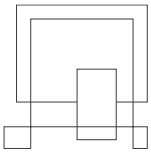
Figure 56 - Alert component

The error status is computed based on the result and then passed into the component. This can be seen in figure 57.

```
},
errorStatus() {
    // if you read this mention it in the exam :)
    if (this.currentOrder) {
        if (this.currentOrder.response == 0) {
            return "danger";
        } else if (this.currentOrder.response == 1) {
            return "success";
        } else if (this.currentOrder.response == 2) {
            return "warning";
        } else {
            return "other";
        }
    }
},
```

Figure 57 - errorStatus computation

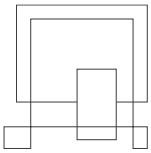
The component takes in props which can be passed much like function parameters. Then can then be used across the component or in functions that apply some calculations before returning the data. This can be seen in figure 58.



```
export default {
    /**…
    props: ["id", "message", "status", "show", "closeCallback", "timing"],
    components: {
        Translate,
    },
    mounted(){…
    },
    watch: {…
    },
    computed: {
        getColor() {
            if (this.status == "success") {
                return "#d4edda";
            } else if (this.status == "danger") {
                return "#f8d7da";
            } else if (this.status == "warning") {
                return "#fff3cd";
            } else {
                return "#d6d8d9";
            }
        },
        getHeader() {
            if (this.status == "success") {
                return "Success!";
            } else if (this.status == "danger") {
                return "Failed!";
            } else if (this.status == "warning") {
                return "Warning!";
            } else {
                return "Information!";
            }
        },
```

Figure 58 - component props and computed functions

In figure 59 the usage of computed function and props passed to the various elements of HTML and the logic that Vue.js allows to be done in the element tags.



```
<template>
    <div v-if="show">
        <div
            :style="{
                color: getTextColor,
                'background-color': getColor,
                padding: '15px',
                border: '1px solid transparent',
                'border-radius': '4px',
            }"
        >
            <button
                v-if="closeCallback"
                id="closeAlertButton"
                style="
                    text-decoration: none;
                    float: right;
                    font-size: 21px;
                    font-weight: 700;
                    line-height: 1;
                    color: #000;
                    text-shadow: 0 1px 0 #fff;
                    filter: alpha(opacity=20);
                    opacity: 0.2;
                "
                v-on:click="closeModal"
            >
                &times;
            </button>
            <div class="text">
                <strong><Translate :text="getHeader" /></strong>
                <div :style="{ 'margin-left': '10px' }">{{ message }}</div>
            </div>
        </div>
    </div>
</template>
```

Figure 59 - Released order page elements

Here is the result of the bad scenario where a bad message gets returned to the client. This can be seen in figure 60.

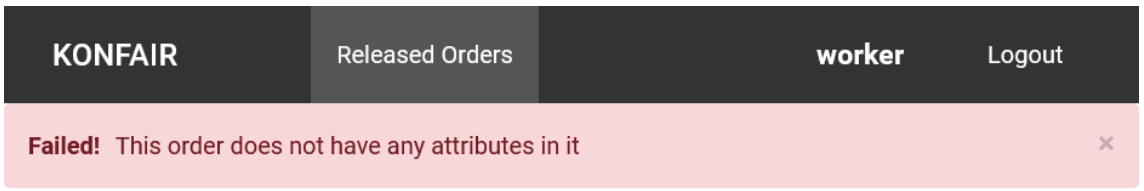
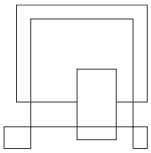


Figure 60 - Bad notification shown to the user

If the result is good the client goes past the if statement in the HTML elements and continues rendering the retrieved data. The data is then divided between the components that generate the tables and display the information. The components shown will not be investigated as they contain some of the functionality of the alert component but with more logic for creating tables. This can be seen in figure 61.



```
<div
    v-if="currentOrder && currentOrder.response == null"
    class="releasedOrder"
>
    <ImageModal
        :image="modalImage"
        :show="modalImageShow"
        :closeCallback="closeImageModal"
    />

    <div class="information">
        <h2><Translate :text="'Order information'"/></h2>
        <DataDisplay :name="'Production order'" :data="currentOrder.productionOrder" />
        <DataDisplay :name="'Item number'" :data="currentOrder.id" />
        <DataDisplay
            :name="'Description'"
            :data="currentOrder.description"
        />
        <DataDisplay
            :name="'Item category code'"
            :data="currentOrder.categoryCode"
        />
        <DataDisplay :name="'Deadline'" :data="currentOrder.deadline" />
        <DataDisplay :name="'Location'" :data="currentOrder.location" />
        <DataDisplay :name="'Status'" :data="currentOrder.status" />
        <DataDisplay :name="'Quantity'" :data="currentOrder.quantity" />

    </div>

    <div class="oneTimeMeasurements">
        <h2><Translate :text="'One time measurements'"/></h2>

        <CustomTableInput
            id="oneTimeMeasurements"
            :allowedHeaders="oAllowedHeaders"
            :rows="currentOrder.oneTimeControlPoints"
            :tableHeaders="oHeaders"
            :imageCallback="showImageModal"
            :valueUpdateCallback="editOValue"
        />
    </div>
```

Figure 61 - part of the released order page

The result of all of this is the data shown on the UI as seen in figure 62.

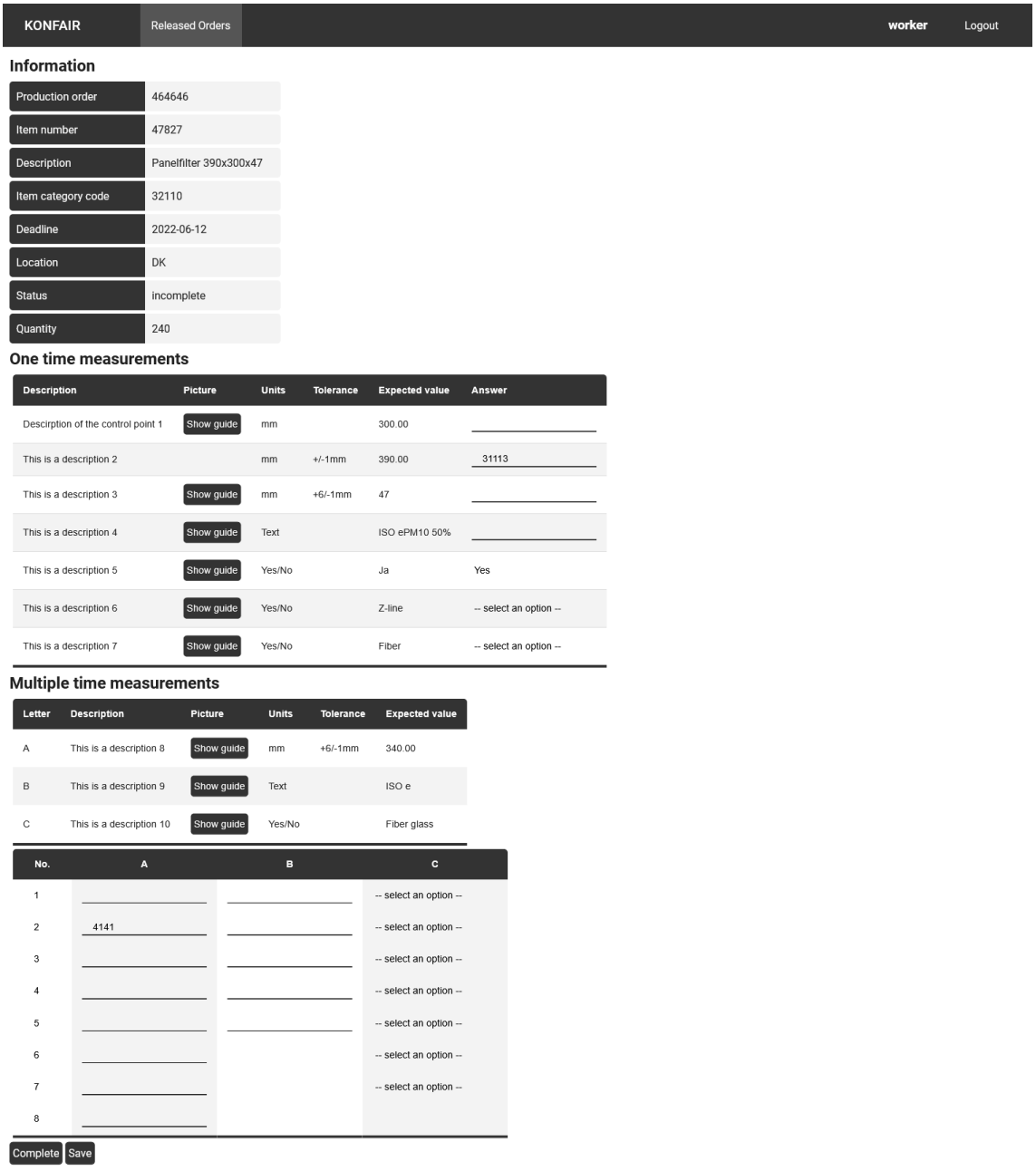


Figure 62 - released order QA from UI

5 Testing

5.1 Testing methodologies

Different kinds of testing are used to help developers ensure that all the parts of the system work as expected and have the least bugs possible.

An illustration of fundaments for testing can be seen in figure 63 of the testing pyramid.

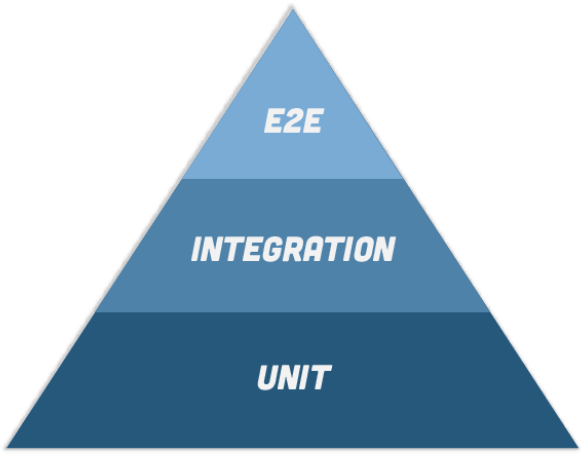
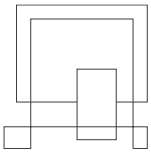


Figure 63 - Testing pyramid

5.1.1 Unit testing

Unit testing was done on the back end. The tests cover the business logic which should be limited only to the service layer, middleware, and custom validators for Express.js. For detailed information about technologies and methodology see chapter 5.2 Back-end testing.



5.1.2 Integration testing

Integration testing was done on the back end. These tests cover the request journey from the client making the request to the route calling the service. This way the middleware, passed parameters and the endpoint location is tested at the same time. For detailed information see chapter 5.2 Back-end testing.

5.1.3 UI testing

Ui testing is done on the front end. These tests check the validity of the UI elements and how they are supposed to change as the user uses the system. This way the team knows if a certain button is visible or if an alert has the needed response or colour. This type of testing covers all dynamic elements on the UI. For detailed information see chapter 5.3 Front-end testing.

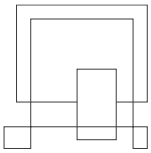
5.1.4 E2E testing

End to end testing covers the scenarios that the user can go through in the system. It does not check that everything that needs to be shown in the UI is shown like UI tests. It tests performed functionality, for example, tests that creating the user works and the user is now in the list. This type of test does go through the test cases that were defined in analysis except that this is not tested by a human.

For detailed information about the technologies and methodologies used for this kind of testing see chapter 5.3 Front-end testing.

5.1.5 System tests/test cases

System tests are defined as a list of test cases. All of them are based on use cases. The result of those tests is defined by the team members. A complete table of system tests can be found in Appendix 9 – Use Case Testing.



5.2 Back-end testing

5.2.1 Jest

Jest is a testing framework for Node.js it provides a lot of functionality for testing. Jest was chosen over the “mocha” testing library because the testing syntax in the command like looks more readable than mocha’s making it easier to find exactly where the error occurred.

Not all of Jest’s features are used in the testing. The team made their own test evaluation functions to show what went wrong in the test as it was more informative.

5.2.2 Sinon

The “Sinon” is a mocking framework that provides stumps, spies and simulations of functions. This was used over Jest’s mocking capabilities due to it being more predictable in syntax and a lower learning curve to do the mocking. Sinon has also been used by the team extensively in the past.

5.2.3 Supertest

Supertest is a high-level HTTP testing framework for Express.js. It gives a reference to make requests to the server and in this way can do end to end testing focused on the back end.

What the team uses this library for is testing that the requests can get through to the endpoint itself and past all the middleware that checks the inputs. Only sunny scenarios are tested this way as the team found that testing the boundary values of the validation middleware was a waste of time from previous experience with the same technologies.

5.2.4 Back-end testing methodology

The back end testing methodology is different from the front-end. The back-end testing focuses not only on the functionality to be working for the client but also on making sure the back end itself validated what the user is sending and that scenarios that do not normally occur when the client makes a request are tested. Testing also needs to be done for scenarios where a user goes around the client interface and calls the back end directly, possibly with incorrect values or types.

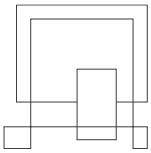
5.2.5 Examples of back-end testing

This example shows a unit test with no dependencies, seen in figure 64. The test checks the target by passing parameters to the function. The result is then checked to be as expected.

```
describe("listToCommaString", () => {
    it("OK", async () => {

        const listOfObjects = [
            {someId: 34},
            {someId: 44},
            {someId: 3},
            {someId: 2},
            {someId: 75},
            {someId: 6},
            {someId: 23},
            {someId: 9},
            {someId: 31},

        ]
        const response = await ordersService.listToCommaString(listOfObjects, "someId")
        assertEquals(response, "34,44,3,2,75,6,23,9,31")
    })
})
```

Figure 64 - test without dependencies

This example shows a unit test being performed on a function that has dependencies, seen in figure 65. The test first mocks out the dependencies of the function using the “Sinon” library and returns specific data to make certain scenarios in the test happen. The aim is to test each branch that can happen in the function. The function is called with parameters and the result of it is checked.



```
describe("FrequencyOfItem", () => {
    it("get Frequency Of Item OK", async () => {
        const defaultFrequencyValue = [{"id":0,"to25":2,"to50":3,"to100":4,"to200":7,"to300":10,

        sinon.stub(itemCategoryModel, "getFrequenciesOfItem").returns(defaultFrequencyValue)

        const data = await itemCategoryService.getFrequenciesOfItem(193345)

        assertEquals(data, defaultFrequencyValue)
        assertEquals(data[0]["to25"],2)
    })
```

Figure 65 - test with dependencies

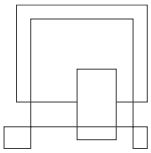
This example shows the integration test, seen in figure 66. An important step in this type of test is to disable any functionality that makes requests to the database on start-up of the API or any initialization of middleware that may make the testing harder or cannot be mocked out. First the middleware that has requests made in it during the execution of the code is mocked out. In this example it is the user validation middleware that makes requests made to the database. Then the service that the route calls is mocked out to return something easily testable. The request is made using the “Supertest” library.

```
describe("Released orders", () => {
    it("Released orders OK", async () => {
        sinon.stub(userModel, "getUserByUsername").returns([{ "role": "qa employee" }])

        sinon.stub(ordersService, "releasedOrders").returns("Test worked")

        const response = await request.get("/orders/releasedList/minimal/rokas/denmark")

        assertEquals(response.text, "Test worked")
    })
})
```

Figure 66 - Released orders

The results of all of the back-end tests can be seen in figure 67

Figure 67 - backend test result



5.3 Front-end testing

5.3.1 Cypress

Cypress is an end-to-end testing library based on JavaScript that works with all web frameworks due to its reliance on HTML elements for testing. It simulates a user clicking through the website. It can be used for both integration and end-to-end testing. This technology has been chosen as opposed to “selenium” for front end testing, because all developers in the team have previously used it.

Cypress was used to do end to end tests that went through the basic scenarios the end user goes through. It can be called use case testing but with a robot running through all the scenarios. During these use case tests UI testing was also performed at the same time. UI testing checked if certain elements behaved correctly on the UI or that they were shown in the first place.

Performance tests were also run to test how long it takes to load the page until the user can see everything that is supposed to be seen on the page.

5.3.2 Front-end testing examples

Here is an example from an end-to-end series of tests that the UI is reacting in the correct way. This can be seen in figure 68. The system must behave in a certain way, it should not let the user create another user without a username or role specified. The inputs in the UI must turn red in this scenario.

```
it( title: 'should check cannot create user without specifying username', config: ()=>{
    cy.get('#createUser').click()
    cy.get('.v-text-field__slot > label').should( chainer: 'have.css', value: 'color', match: 'rgba(0, 0, 0, 0.6)')
    cy.get('.v-btn:nth-child(3) > .v-btn__content').click();
    cy.get('.v-text-field__slot > label').should( chainer: 'have.css', value: 'color', match: 'rgb(255, 82, 82)')
    cy.get('.v-btn:nth-child(4) > .v-btn__content').click();
    cy.get('.v-btn:nth-child(3) > .v-btn__content').click();

})
```

Figure 68 - UI test

Here is an example from the same series of tests as the previous one. This shows the consecutive tests working together to validate that a user was indeed created. The sunny scenario of the test is done here and then the user is found in the table of users in the system.

```
it('should create user',()=>{
    cy.get('#createUser').click()
    cy.get('#usernameInput').click();
    cy.get('#usernameInput').clear();
    cy.get('#usernameInput').type(val);
    cy.get('#roles').parent().click()
    cy.get('.v-menu__content').contains("admin").click();
    cy.get('.v-btn:nth-child(4) > .v-btn__content').click();
})

it('should check created user',()=>{
    cy.get('tr').should('contain.text',val)
})
```

Figure 69 - end to end test p

Here is an example of the cypress UI where the robot can be seen after it finished testing the page of user management. This can be seen in figure 70. A good feature of the cypress UI is that it records everything it does. If an error occurs in the test an easy way to find the problem is just to look at what the robot was looking for and see if it is there in the snapshot of the UI. This functionality can be seen in figure 71.

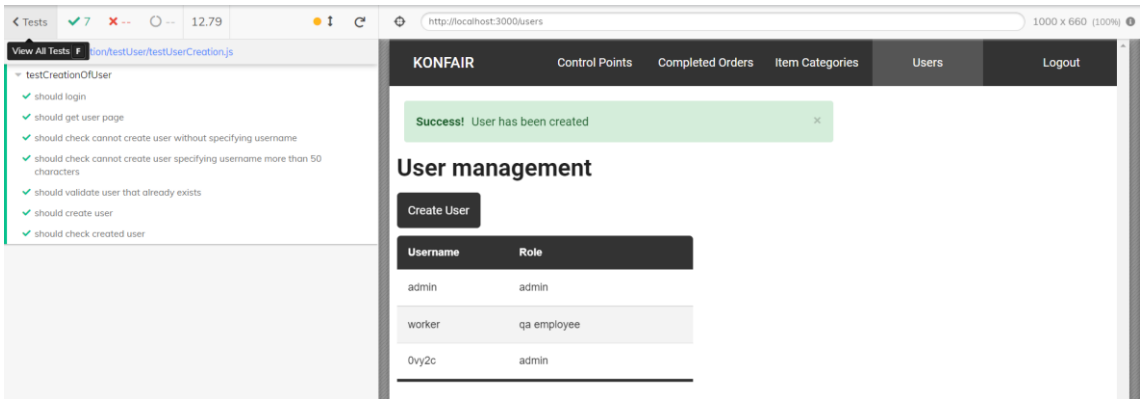


Figure 70 - completed tests

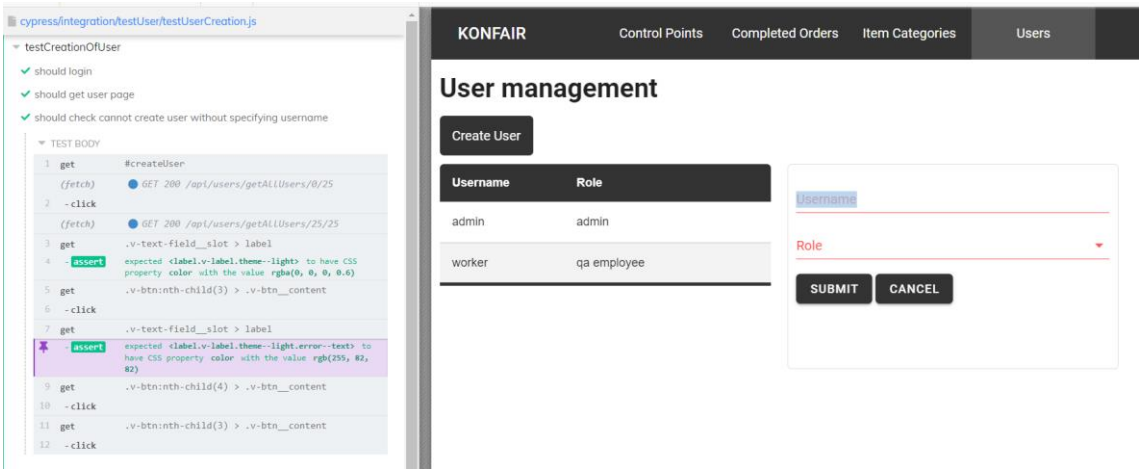


Figure 71 - test recording inspection

5.4 Testing results

The results of the back end and the front-end testing that was run can be found in appendix 13 – Front end and backend test results

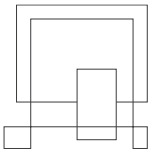
6 Results and Discussion

6.1 Performance testing

The performance test conducted for this project only aims to address the loading time as described in the non-functional requirement.

- NREQ-3 - The System should ensure that all Functionalities has performance loading time of less than five second

For the sake of the project, soak test, spike test and stress test have been disregarded since the number of users of the system is of a limited quantity and it is expected that the system will not be exposed to strenuous activities as such. A load test has been made to ensure that the response load of a page in the system is under the acceptable



level for the customer and fulfils the requirement. Due to its recording prowess, cypress has been used to for this performance test.

According to dotcom-tools, the ideal load time for a webpage should be between 2 to 5 seconds as 40% of polled users abandon a site if it takes longer than 3 seconds to load. Therefore, for the purpose of this project, below 3 seconds will be seen as a good response time and from 3 seconds to 4.9 seconds will be the average time for the project. The table below shows the response time for each load in the system.

Tests are conducted in milliseconds: 1000 MS = 1s

0.1– 2.99 = Good

3 – 4.99 = Average

5 > = Failed

ID	Load test	Test file	1st test	2nd test	3rd test	Conclusion
PT1	Load a completed Order	completedOrder.spec.js	2103	1977	2410	Passed
PT2	Load the completed Orders	completedOrders.spec.js	1435	1555	1649	Passed
PT3	Load control point list page	controlPoints.spec.js	1815	1697	1783	Passed
PT4	Loads create control point page	createControlPoint.spec.js	2726	3303	2417	Passed
PT5	Load edit Control point page	editControlPoint.spec.js	2695	2646	3041	Passed
PT6	Load item Category page	itemCategory.spec.js	1469	1779	1518	Passed
PT7	Load edit the frequency of item category page	itemCategoryEdit.spec.js	1578	1812	1902	Passed

PT8	Load login page	login.spec.js	1597	1538	1432	Passed
PT9	Load a released order	releasedOrder.spec.js	2996	2976	3000	Passed
PT10	Load released orders page	releasedOrders.spec.js	1517	1918	1760	Passed
PT11	Load users page	Users.spec.js	1562	1645	1650	Passed

Table 2 - Performance test result

A graph could therefore be plotted to further visualize the performance of the system at each scenario.

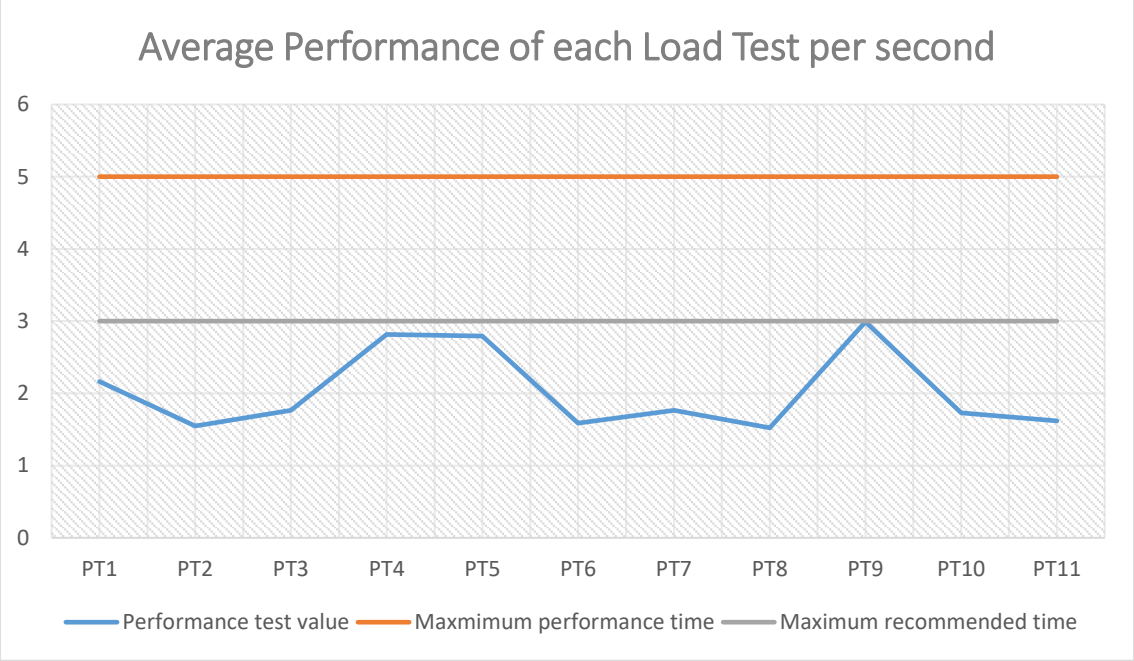
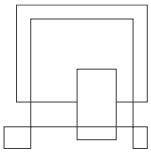


Figure 72 - Performance test result

6.2 Test cases

The test cases were performed to check if all the scenarios that the team has specified in the analysis section are being fulfilled by the implemented system.

A table can be seen with the status of each user story with relevant test cases linked to it. A user story is partially implemented if a test case is not passing.



The test cases and results for them can be seen in Appendix 5 – Use Case Testing

6.3 Acceptance testing

An acceptance test has been made with the customer and is the final stage of this software evaluation. The purpose of this acceptance test is to help increase customer satisfaction and reduce software maintenance costs. For the sake of this project, user acceptance testing has been specifically chosen to determine whether the system is working for the user with the given specific requirements. These tests have been defined into two main sections:

1. Final Deliverables and Acceptance Criteria Validation
2. Outstanding Issues and Resolution Plan

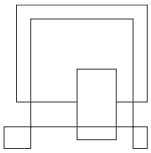
These tests will be described in-depth below.

6.3.1 Final Deliverables and Acceptance Criteria Validation

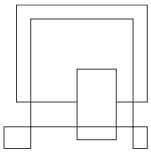
The table below describes the result of the acceptance testing, which includes the result of user evaluation as well the description or condition related to the acceptance of the deliverables.

The customer not having any comments was considered an accepted evaluation. This is because the customer explicitly informed the team of this.

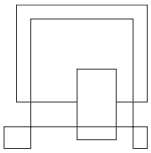
The customer was very pleased with the application and noted that the application could be put into production in its current state.



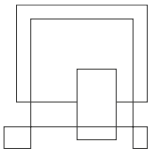
Deliverables	Result of User Evaluation	Contingencies or Conditions	Date of Acceptance
User should be able to login to System	Accepted	N/A	2022-05-25
Administrator should see list of control points	Accepted	N/A	2022-05-25
Administrator should specify a connection for the category items while creating control point	Accepted	N/A	2022-05-25
Administrator should specify a connection for the attributes while creating control point	Accepted	N/A	2022-05-25
Administrator should specify description while creating control point	Accepted	N/A	2022-05-25
Administrator should specify the type while creating control point	Accepted	N/A	2022-05-25
Administrator should specify the frequency while creating control point	Accepted	N/A	2022-05-25



Administrator should be able to upload a picture while creating control point	Accepted	Should also be able to specify video in the future	2022-05-25
Administrator should be able to specify measurement category while creating control point	Accepted	N/A	2022-05-25
Administrator should be able to edit chosen control point	Accepted	N/A	2022-05-25
Administrator should be able to add a user	Accepted	super-user in the future	2022-05-25
QA worker should be able to see released orders	Accepted	Should have quantity specified. Time stamp should also be provided for released time	2022-05-25
QA worker should be able to see the QA form	Accepted	Should show that qa is in progress	2022-05-25
QA worker should be able to fill in the measurements on QA form	Accepted	Good to know user who filled out in incomplete qa.	2022-05-25
QA worker, should be able to submit QA form	Accepted	N/A	2022-05-25



QA worker and Administrator should be able to choose department	Accepted	Advance department management in the future.	2022-05-25
Administrator should be able to see item categories	Accepted	N/A	2022-05-25
Administrator should be able to edit the frequency of chosen item category	Accepted	N/A	2022-05-25
Administrator should be able to see the list of completed QAs of an order	Accepted	List of completed orders should have production order	2022-05-25
Administrator should be able to see QAs of chosen completed order	Accepted	N/A	2022-05-25
Administrator should be able to generate PDF of QA for chosen completed order	Accepted	Not particularly printer friendly and could be improved on in the future. Could be exported to excel in the future.	2022-05-25
Administrator should be able to delete a chose control point	Accepted	N/A	2022-05-25
Administrator should be able to delete a user	Accepted	N/A	2022-05-25



QA worker and Administrator should be able to change language to Danish, English or Lithuanian.	Accepted	Help with Translation is required.	2022-05-25

Table 3 - Acceptance tests

6.3.2 Outstanding Issues and Resolution Plan

This section lists out the issues identified during acceptance testing and the corresponding resolution plans

Issue	Resolution Plan
The QA worker cannot see who entered the input, due to anonymity.	Remove the anonymity and trust the QA worker with seeing the details
UI design has a lot of unused space	Addition to version two of system
Concern on the next step of the Project	Provision on documentation, diagrams, and source code to aid future developers of the system.
User not seeing who is logged in	Fix in next sprint
Header of the tab has default application name and username	Add current logged in user to tab name (header)

Table 4 - Outstanding issues and resolution plan

A user guide has also been created which consists of a video of each scenario covering all the use cases in the system and how to perform the scenarios. This user guide can be found in appendix 11.

A reference from the company was also given to the team to show their satisfaction at the work that had been created for them by the team. This can be seen in the figure below and can be found in appendix 12

konfAir
Flexible Filter Technology

Herning, d. 31-05-22

Reference letter

We have at konfAir had the pleasure of working with the group of students in the spring semester of 2022. The group consisted of the students:

- Rokas Barasa
- Emmanuel Chukwukodinaka Simon
- Rafał Pierścieniak

The project was to find a way to make digital system for our quality assurance. Today all our quality control is done on paper and we had a wish to make this system digital and paperless.

The group worked verry analytical to figure out what the requirements was for the project and what our specific needs was to make the system fit in our production. They worked well to listen to our suggestions and to fit in the features that we needed for the system.

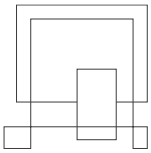
We are very pleased with the end result of the project. It is a working solution that we can use to demonstrate what features we need for a finished system.

It has been very a pleasure working with the group. Communication has been friendly and to the point, meetings has been relevant, and we have in general only had a nice experience.

Kind Regards
Allan Kildevang Ladekjær
Business Developer
konfAir a/s

konfAir a/s - Orebygårdvej 13 - DK-7400 Herning - Tlf: +45 97137133 – konfAir@konfAir.dk - konfAir.dk

Figure 73 - Reference letter from KonfAir a/s



7 Conclusions

In this project, the team has analyzed the requirements of the digitalization of the KonfAir project. Firstly, by analyzing the problems that were faced in the company while working together with KonfAir, requirements were drafted through a thorough and rigorous process. Hence, functional, and non-functional requirements were drafted for the system. By analyzing the requirements, a domain model was created.

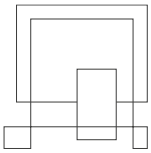
The process of creating a domain model included creating and identifying a use case diagram for the system and a description for each use case. Hence it was able to create the activity diagram which shows a series of the flow of control for the system and the sequence diagram which shows the interaction between actors and the system. Test cases were also created to show how to test the systems use cases.

In the design, the UI choices were made using gestalt principles. Figma was the technology implored to create it. The system architecture was also defined. Technologies were chosen according to project needs while considering development team experience. The database was designed by creating an EER diagram. Microsoft SQL Server was chosen because the customer already used it and it was being integrated into the customer's database. The implementation of the system was made by using the appropriate methodologies and the application was well tested using testing frameworks. Jest was used to test the backend in collaboration with Sinon and Supertest. And for the frontend testing, Cypress was used. Both testing methodologies were deployed to ensure proper and thorough testing in the system.

Looking back at the original purpose of the project. The point of the project was to digitalize the existing production process of quality assurance in KonfAir and speed up the process for the employee doing the quality assurance.

The project was completed with a successful digitalization of the project. All aspects of the QA assurance process can now be done on this system.

The speed improvement of inputting information was improved, it can be argued that inputting data using the keyboard is faster than writing. The system does significantly



reduce the work needed to of creating a QA form for the administrators of the KonfAir company.

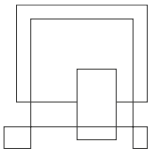
Overall, the result of this project is very good. The customer (KonfAir) is happy by what the team delivered.

8 Project future

The second version of the application has been discussed with KonfAir, the plan is to build on top of the proof of concept as it already has a lot of features that KonfAir considers production ready, implemented. This has been affirmed after the success of the acceptance test. KonfAir said that they will need a team to support the system further, it did seem like KonfAir was offering us the job of support, but the team refused as it is already employed elsewhere.

Most of the additions that have been discussed and then moved to version two are user experience improvements and some medium sized features. Here are the ones that the team has written down when going through the project:

- New admin user who doesn’t have permissions to create other users.
- Remembering the logged in user and order which he was doing. (This can still be done by opening different tabs in the browser, as long as no user is logged in at the time)
- Highlighting the last order, the QA employee did.
- Item category description in item category list.
- Search by order number, item number, category code. In that order. For administrator and QA employee.
- Add ability for administrator to manage departments in the company. This would be creating the department with country specified as well as the category codes that apply to this location. The user login page would show these locations.

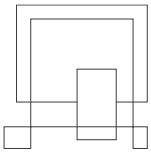


Although the features are the future of the project, it is important to address the features which were implemented but could be improved upon. These are suggestions from KonfAir that could provide better user experience.

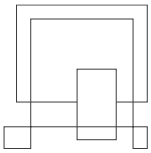
- UI designing white space – it was suggested by the customer to have a better UI as it contained too much white space.

9 Sources of information

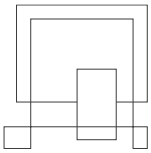
Anon., n.d. *Project Management Lifecycle Navigator.* [Online]
Available at: https://www.visual-paradigm.com/features/project-management-lifecycle-guide-through/
[Accessed 31 5 2022].
Anon., n.d. *vuex-persistedstate.* [Online]
Available at: https://www.npmjs.com/package/vuex-persistedstate
[Accessed 31 05 2022].
Anon., n.d. *What is a state managemnt pattern.* [Online]
Available at: https://vuex.vuejs.org/#what-is-a-state-management-pattern
[Accessed 1 6 2022].
Anon., n.d. *What is acceptance testing? Types and Examples.* [Online]
Available at: https://w3softech.com/blog/acceptance-testing-types-and-examples/
[Accessed 31 05 2022].
Dotcom Tools, 2018. *How Fast Should My Website Load?.* [Online]
Available at: https://www.dotcom-tools.com/web-performance/blog/how-fast-should-my-website-load/
[Accessed 31 05 2022].
Douglas, S., 2017. *How to improve UX using gestalt principles.* [Online]
Available at: https://www.justinmind.com/blog/how-to-improve-ux-using-gestalt-principles/
[Accessed 29 05 2022].
Emma, 2021. *The Difference Between SQL Express and SQL Standard Explained.* [Online]



Available at: https://www.yourofficeanywhere.co.uk/the-difference-between-sql-express-and-sql-standard-explained/
[Accessed 31 05 2022].
Gillin, P., 2022. *What is Component-Based Architecture?.* [Online]
Available at: https://www.mendix.com/blog/what-is-component-based-architecture/#:~:text=Component%20architecture%20is%20a%20framework,requiring%20modification%20of%20other%20components
[Accessed 29 05 2022].
Larman, C., 2004. In: *Applying UML and Patterns.* s.l.:s.n., p. 66.
Nextjs, n.d. *Next.* [Online]
Available at: https://www.npmjs.com/package/next
[Accessed 10 05 2022].
nguniversal, n.d. *Angular Express Engine.* [Online]
Available at: https://www.npmjs.com/package/@nguniversal/express-engine
[Accessed 10 05 2022].
NISO, 2010. *Scientific and Technical Reports -,* Baltimore: National Information Standards Oganization.
Nodejs, n.d. *Introduction to Node.js.* [Online]
Available at: https://nodejs.dev/learn
[Accessed 31 05 2022].
Nowak, M., 2022. *Vue vs React in 2022 - Which Framework to Choose and When?.* [Online]
Available at: https://www.monterail.com/blog/vue-vs-react
[Accessed 31 05 2022].
Nuxtjs, n.d. *Nuxt.* [Online]
Available at: https://www.npmjs.com/package/nuxt
[Accessed 10 05 2022].
parkersoftware, n.d. *Web vs desktop apps: a weigh-up.* [Online]
Available at: https://www.parkersoftware.com/blog/web-vs-desktop-apps-a-weigh-up/
[Accessed 29 05 2022].
Pratt, M. K., 2021. *SearchCIO.* [Online]
Available at: https://searchcio.techtarget.com/tip/Top-10-digital-transformation-benefits-



for-business
[Accessed 8 11 2021].
Prisma, n.d. *Prisma schema.* [Online]
Available at: https://www.prisma.io/docs/concepts/components/prisma-schema
[Accessed 31 05 2022].
Sinon, 2022. *Sinonjs.* [Online]
Available at: https://sinonjs.org/how-to/stub-dependency/
[Accessed 23 05 2022].
StackOveflow, n.d. *Most popular technologies.* [Online]
Available at: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof
[Accessed 31 05 2022].
statcounter, 2022. *Desktop Screen Resolution Stats Worldwide.* [Online]
Available at: https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide
[Accessed 29 05 2022].
Supertest, 2022. *Supertest library.* [Online]
Available at: https://www.npmjs.com/package/supertest
[Accessed 23 05 2022].
VIA Engineering, in preparation. *Confidential Student Reports,* s.l.: s.n.
Vue, n.d. *Introduction.* [Online]
Available at: https://vuejs.org/guide/introduction.html
[Accessed 31 05 2022].
vuetifyjs, n.d. *Why Vuetify?.* [Online]
Available at: https://vuetifyjs.com/en/introduction/why-vuetify/
[Accessed 30 05 2022].
VUEX, n.d. *What Is Vuex?.* [Online]
Available at: https://vuex.vuejs.org/
[Accessed 31 05 2022].



10 Appendices

Appendix 1 – Group Contract

Appendix 2 – UML Diagrams

Appendix 3 – Use Case Descriptions

Appendix 4 – Source Code

Appendix 5 – Use Case Testing

Appendix 6 – UI Layouts

Appendix 7 – Project Description

Appendix 8 – Sprints Documentation

Appendix 9 – Timetable

Appendix 10 – Requirement Change Over Time

Appendix 11 – User Guide

Appendix 12 – Reference from Konfair

Appendix 13 – Frontend and backend test result

Digitalization of Quality Assurance for KonfAir

Final Project

Simon Emmanuel Chukwukodinaka 285152

Rafał Pierścieniak 279471

Rokas Barasa 285047

Supervisor: Poul Væggemose

Bring ideas to life
VIA University College

konfAir
Flexible Filter Technology

38,188

Software Technology Engineering

7th semester

01.06.2022

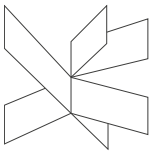
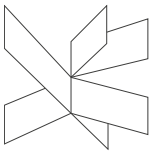
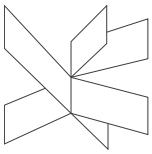


Table of content





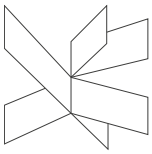
1 Introduction

We formed this group based on our collective main goal and expertise that had been shared in the previous semester. We made this group in the 6th semester and gained experience working with each other on a project. On the 27th of October 2021, we created this group. The purpose of this group was to focus on the semester project as well as the bachelor project. By signing a binding group contract and working with supervisors, we were able to choose a project from a pool of potential clients and customers who were offering to make a project for them.

To fully understand the strengths and weaknesses of the team, we took multiple personality tests including the Hofstede insight and Estimate profile. This was very important as using the information we gathered from these tests we were able to draft a swot analysis to identify the group more.

The group was very focused through all phases of the project as we continually worked closely with the supervisor to aid in questions and help solve outstanding issues that we came across throughout the project. The client we worked with was a company Konfair A/S an air filter manufacturing company. They were always readily available to cooperate with the group throughout all the phases of the process. As this was very important, it helped us produce a qualitative and useful project for them. Although we identified with them that this project is only a proof of concept, they were understanding.

The project that we worked on was a quality assurance system that ensured that the process of the user quality assurance was made easier. And although the project aimed exactly at this, there are more directions the project could go including the use of business intelligence and machine learning.



2 Group Description

The group was created in the 6th semester. The group members Rafal and Simon have had a lot of experience working together before. Rokas never has worked with Simon or Rafal.
Rokas chose to be on this team after reviewing previous projects done by Rafal and Simon. The goal was to have teammates who are who have good problem-solving skills and good technical knowledge.

2.1 Group members

Rokas – Lithuanian, 22 years old, student of ICT – Software engineering at VIA with no specialization area.
Simon – Nigerian, 18 years old, student of ICT – Software engineering at VIA- Data Engineering Specialization
Rafal – Polish, 22 years old, student of ICT – Software engineering at VIA with no specialization area.

2.2 Hofstede's Culture Insight

The Hofstede Insight's CEO (Cultural Executive Ownership) enables us to solve Intercultural and Organizational Culture challenges by utilizing their effective and proven frameworks. We have utilized this tool to fully understand the cultural domains within our group. We visualize the 6 dimensions as shown below.

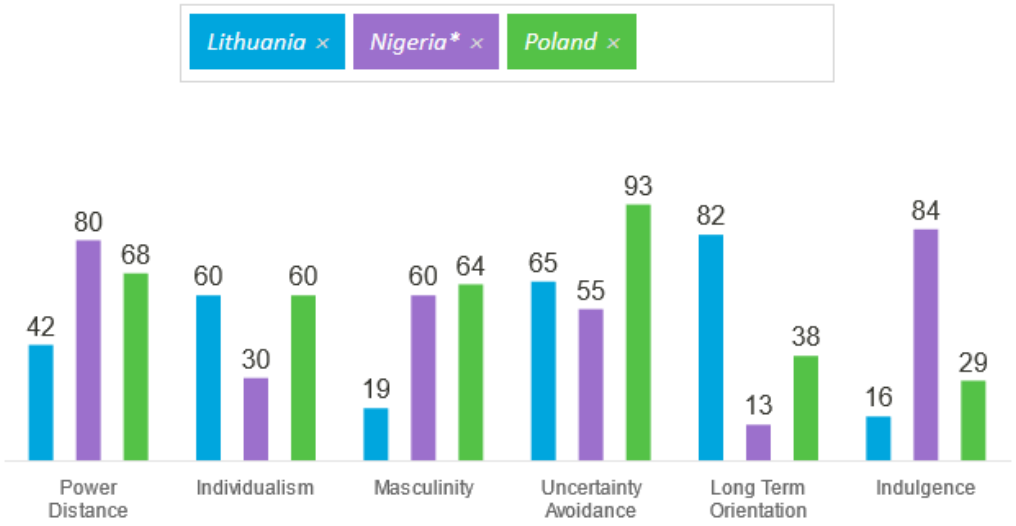


Figure 1 - Hofstede's culture comparison

2.3 E-estimate profiles

The estimate profile below represents the behavioral tendencies in our group

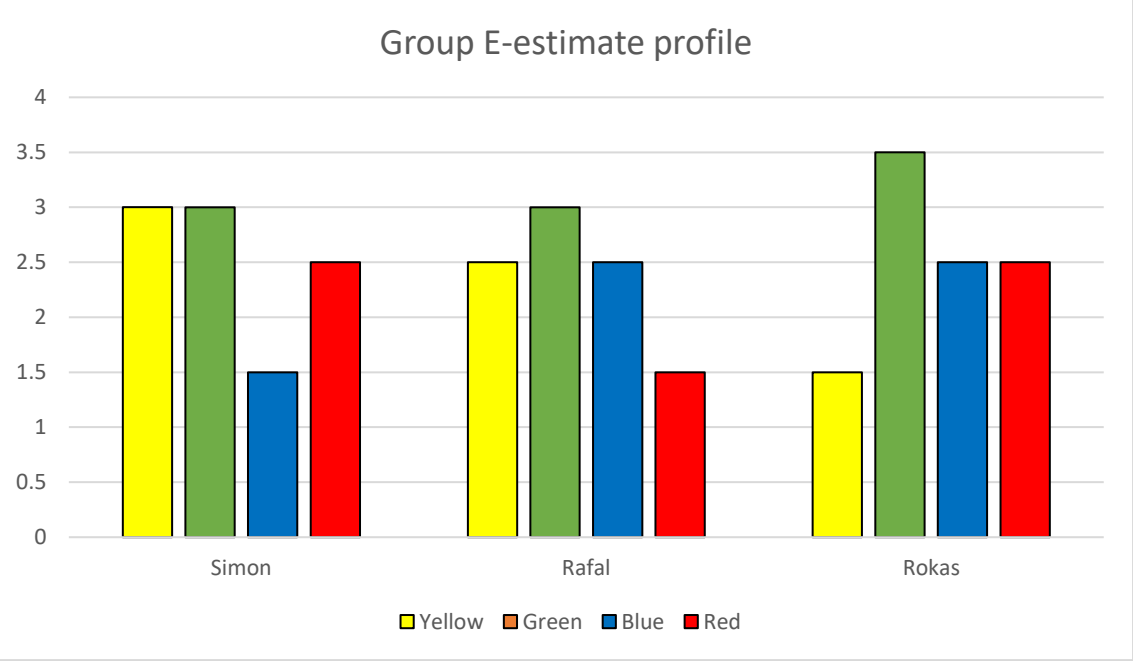


Figure 2 - E-estimate profile of each member

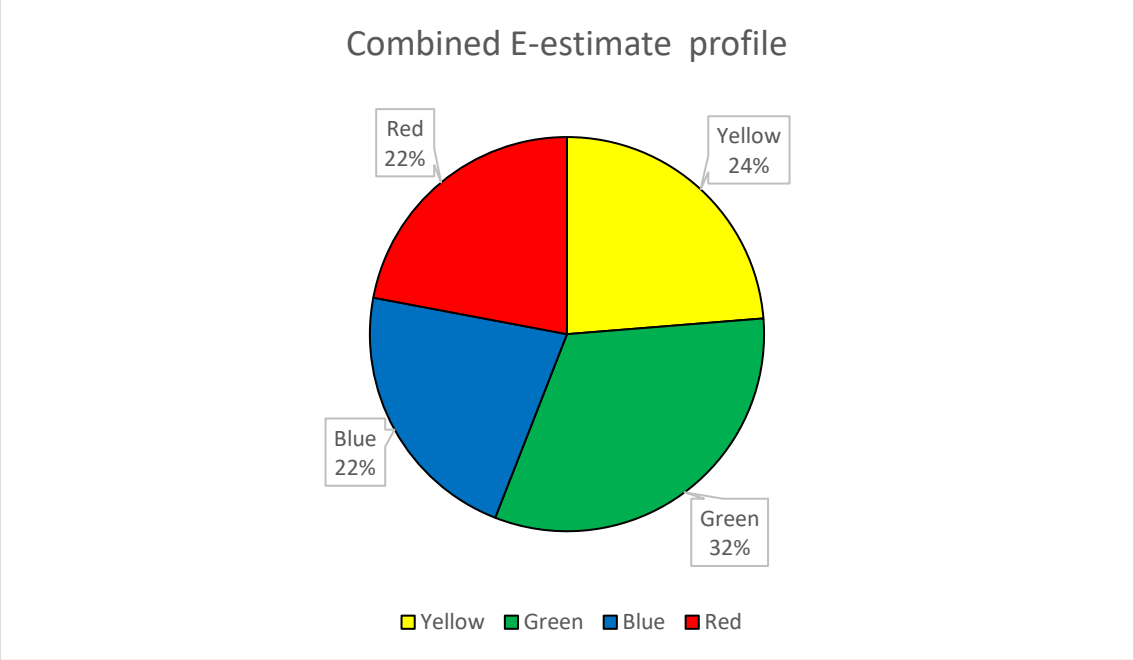


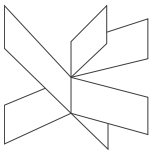
Figure 3 – Combined E-estimate profile of the group

Based on the data gathered from the e-estimate and cultural backgrounds some assumptions about the team can be made. The group is balanced in every color except green which is a very good imbalance to have as it promotes cooperation.

Yellow – The team is moderately yellow. This can mean that the team will be quite independent when doing tasks. The team may be inclined to have more variety in technologies and may refuse to use older proven technologies, this could cause conflicts. This can also mean that members can be unfocused.

Green - The team is leaning heavily on the green, which means the team members should be good at communicating with each other. This also means that the team is less likely to have members who do not contribute as the members will feel responsible for the project.

Blue – The team has a moderate blue presence. This means that the team should be good at organization and keeping structure. The project should have good quality. Heavily blue people can also be too skeptical and shoot down good ideas.



Red - The team has a moderate red presence. This means that the team should be good at acting and having the initiative to do things. The bad side of this is that red people can be too dominant and not let other team members impact decisions.

2.4 Methodology preferences

All the members described SCRUM combined with Unified Process as the preferred methodology for the project. The arguments were given as follows:

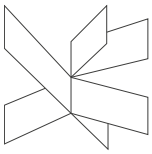
- All of the members have been using scrum for four previous semesters.
- Most of the members have been using the scrum during the internship.
- All of the members agreed that this methodology is the most commonly used in the industry.
- The scrum gives us valuable outcomes on what is the progress within each sprint.
- The unified process fits our plan for the project. Find a project, elaborate on requirements and analysis, design and implement and close off the project.

2.5 Roles

As Scrum methodology is planned to be used, the roles in a group are defined accordingly:

Product owner	**Scrum master**	**Development team**
Rafal, Simon, Rokas	Simon, Rokas, Rafal	Rafal, Simon, Rokas

The development team will include all the members of the group since all of us want to actively participate in the project. We agreed that we will have the scrum master role assigned to three members. As the first candidate, we have chosen Rokas as he is a systematic and detail-oriented person with a strong tendency to be a leader. Simon has been considered a scrum master – following the E-estimates profile he is considered a



person who cares about the people around him. Unlike Rokas and Rafal, Simon is not that tied to traditions and is seen as an innovative person with an easy-going lifestyle. We expect that Simon cares about people's well-being and can bring good results during the sprints. Despite that Simons and Roka's E-estimate profiles differ the most, they will share the role. However, only one will be active at the time. The change of scrum master role is planned for every second sprint.

Similar to the product owner role. All the members are assigned to this role as the team is not that big. The decisions are taken by taking open discussion and with the main rule to avoid any assumptions. For this reason, all of the members are in contact with the project initiators to clarify the business model whenever needed.

The role assigned to the person will bring the meaning of being responsible for the tasks related to the role. However, each of the members may have an impact and initiatives for future actions.

2.6 SWOT Analysis

After experiences working in different groups and analyzing all the available theories, we have conducted a SWOT analysis (Strength, Weakness, Opportunities, Threats) on the group Project and have come to the following conclusions.

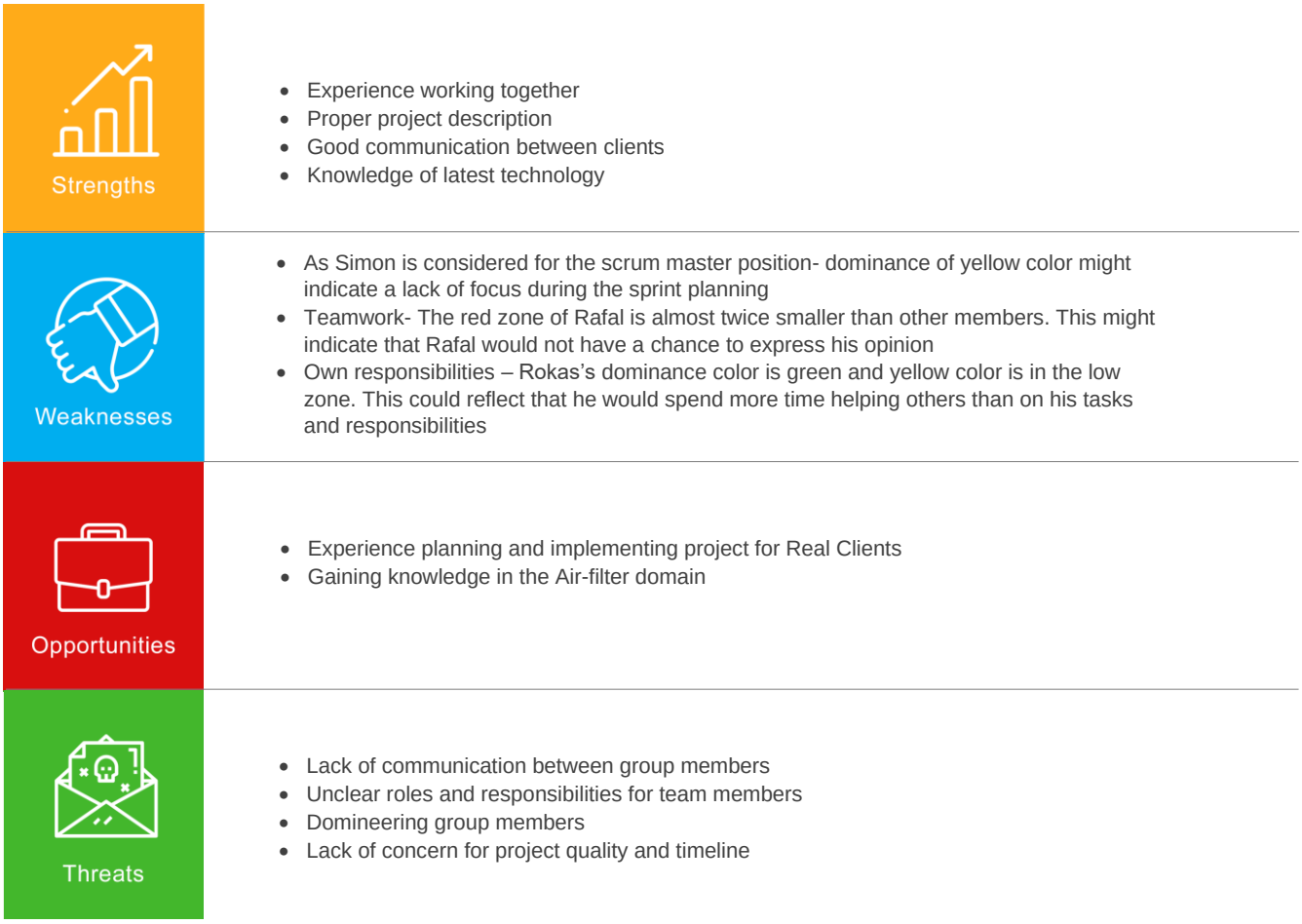


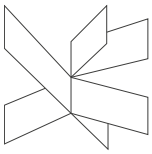
Figure 4 - SWOT analysis of the group

2.7 Group contract

The group contract can be found in appendix 1 – group contract.

3 Project Initiation

During the project initiation phase, we made a group contract and some basic descriptions of each member of the group. We started looking for companies with which we could cooperate and do a project after the first presentation of companies that Michael Viuff organized. We interviewed 3 companies after the presentation. We chose one good candidate from them – KonfAir A/S. Out of all the companies they had



the most concrete problem to solve and didn't require us to come up with a problem to solve.

In parallel, the team also got a contact in Stiesdal A/S to have more choices who are developing a grid-scale energy storage system. The team met with KonfAir before the end of October and got some more information about the problem. The team also met with the Stiesdal A/S representative in mid-November.

KonfAir

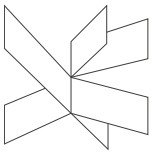
Konfair's problem was a non-digitalized production process. Managers getting an order, need to compile it into steps and forms for the employee which needs to fill them out. The QA forms were reused often as they did not have enough time to make an individual form for each item. It was very hard for them to learn anything from their production process quality because the forms were scanned and kept in a folder on a hard drive.

Stiesdal

Stiesdals problem was more unclear and more abstract as they could not disclose all information. The work would have focused on using their data and employing it to improve the thermodynamic processes in the system as well as analyzing it to give more feedback to their team.

Final choice

We were convinced on going with Stiesdal due to more possible workload, flexibility, and technologies, but we reconsidered. Stiesdal did not have a very clear problem to solve and a half-year delay in their timeline meant that we would not have to concreate data or system to integrate with. This would have made the bachelor project extremely hard to do hence, we chose to go with Konfair.



4 Project Description

In the project inception phase, we turned what we have learned from meeting with Konfair into a project description with specific problems to solve and a definition of the scope by delimiting certain topics in the project. While compiling the description we planned on what methodology to use as well as a timeline for the project.

Initially, since we had a meeting with the company it was very easy to create a background description. They presented to us the process that they used for quality assurance, and this was a crucial step for us. They had used a production paper to ensure their quality assurance and hence could not be used for any purposes such as machine learning and business intelligence and was harder and less effective to document the process.

With this in mind, we were able to create a definition of purpose. The digitalization of their existing production process of quality assurance. To be able to fully solve this problem we had to ask ourselves the most pressing questions like how could we speed up the process of their quality assurance? All these questions were then documented in the problem statement as we decided what should be delimited from the system.

The process of delimitation was a rather hard one as we didn't fully grab the scope of the project. The delimitations would be subject to subsequent changes throughout the process of the project. However, things such as analytics and security were thoroughly examined in the delimitations.

For this project, the team chose Scrum-like methodology and Unified Process as they had already been in use by all members of the project for previous projects. The scrum master, product owner, and development team were also decided during this phase. All members of the group would fulfill all these roles during the project execution, with the scrum master rotating between us. We wanted the product owner to be a role for all members of the team because we felt that one member would not be capable of grasping all of the requirements that the customer is specifying. Part of this was

because the customer had difficulty explaining what they wanted in detail during our first meeting.

A schedule was then plotted as we made a risk assessment covering all the possible risks that would probably be faced during the project.

Overall, the group was able to be set up by going through all these phases. A realistic goal was set and the group was ready to execute the project in a well-structured and scoped definition.

The plan and result until the end of the Project Description were looking as shown in figure 5.

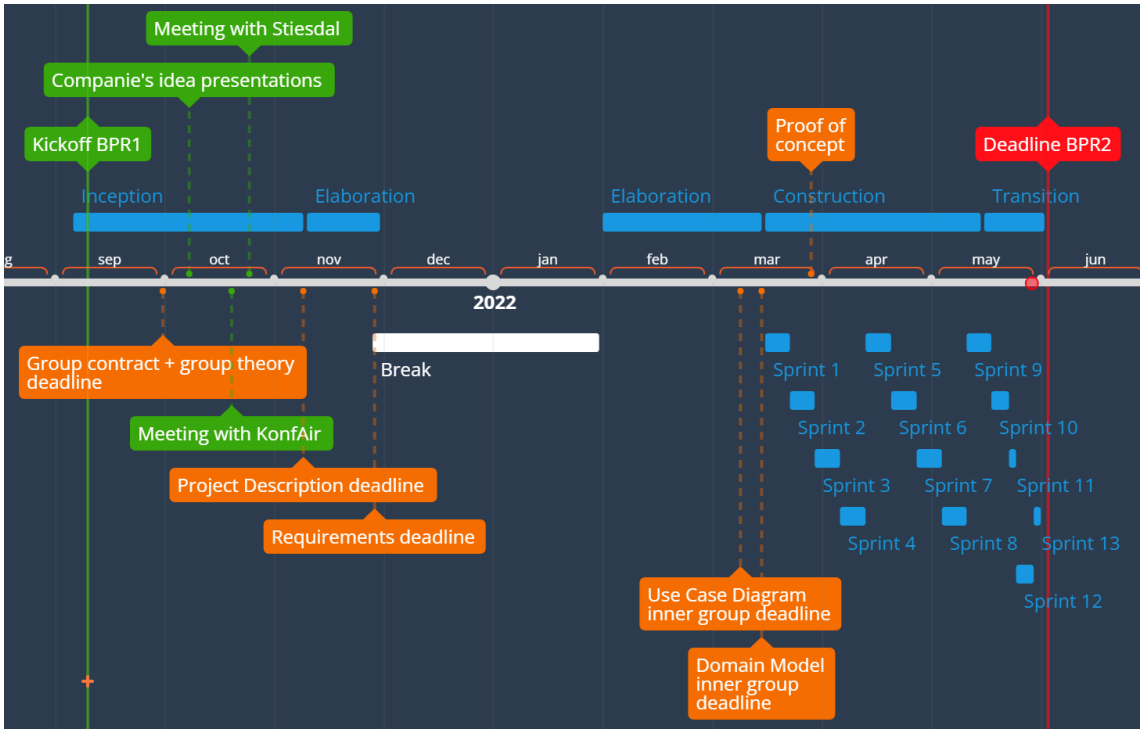
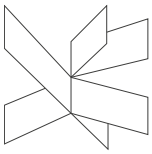


Figure 5 - Project plan timeline



The project description was changed a lot as we gained more understanding of the project, it would be best to read the one in appendix 7 – project description as it is most up to date.

5 Project Execution

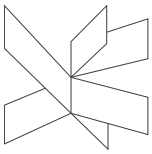
The execution of the project can be compared with a plan from the project description. The deviation between them is explained in this chapter and can be compared visually by analyzing Figures 5 and figure 6.

The requirements were defined at the end of November. These were gathered from the first meeting with KonfAir. As we saw later these requirements had a lot of assumptions made due to us not completely understanding the problem after the meeting. We made the mistake of not sending the requirements after defining them. After we did the first version of the requirements the bachelor project was paused until the end of January due to sixth-semester exams.

Once the pause ended, we started the project elaboration phase. That was the time when we started unofficial consultations with the customer via exchanging emails. This was an efficient way to communicate as this way we were not forced to reply in rush and on the other hand, we could clarify doubts whenever needed.

Using the requirements that were not accepted by the customer a use case diagram was made that was mostly wrong. A Figma UI example of how our understanding of the requirements would look like was made.

After the first half of February, we brought requirements and Figma UI to the customer in an online meeting. The customer had more members participating and giving us feedback than he had In the first meeting with us (the initial meeting in the inception phase). The Figma UI was very useful, it helped the team visualize what we were



thinking, and the customer could quickly understand what was going on. We received a lot of feedback, and changes in requirements were necessary. The customer wanted to add more of what the administrator could do to the system. We got a lot of good information from a QA analyst from Konfair that was participating in the meeting. We recorded the meeting and analyzed it carefully to add the new requirements.

A first-time meeting with our supervisor was made. We were on track in the elaboration phase of the project. He suggested some changes and some more things to clarify with the customer (supervisor meetings can be read upon in chapter 7 - Supervision).

A meeting was arranged in early March in person to show the updated requirements that were changed after the second meeting. The main goal of the meeting was to get acceptance of the requirements from the customer. Initially, requirements were rejected again. During that day the requirements were constantly changing with the supervision of the customer. The end of the day was successful as both sides approved the requirements which end up being prioritized using the MoSCoW method.

After getting acceptance for requirements, the team focused on Use Case Diagram and descriptions for them as this together with requirements was the base for system functionality and UI design. After a supervisor meeting on that, we made necessary changes and approved it inside the group to be able to move forward. We started making the first UI design scatches for the system.

We continued working on other diagrams where the main goal was Domain Model. Once we had the result we made a meeting with the supervisor where he provided us feedback on all documentation we did so far. After discussing the feedback about the Domain Model and including the changes, we approved the first version of this diagram inside the group.

We had a meeting with the customer during which we presented UI design. The group found the feedback useful as the next step was to start the construction phase where we could continue deep analysis, design, implementation, and testing. After this

meeting, we all felt ready and well-motivated to start some basic implementation and start the sprints.

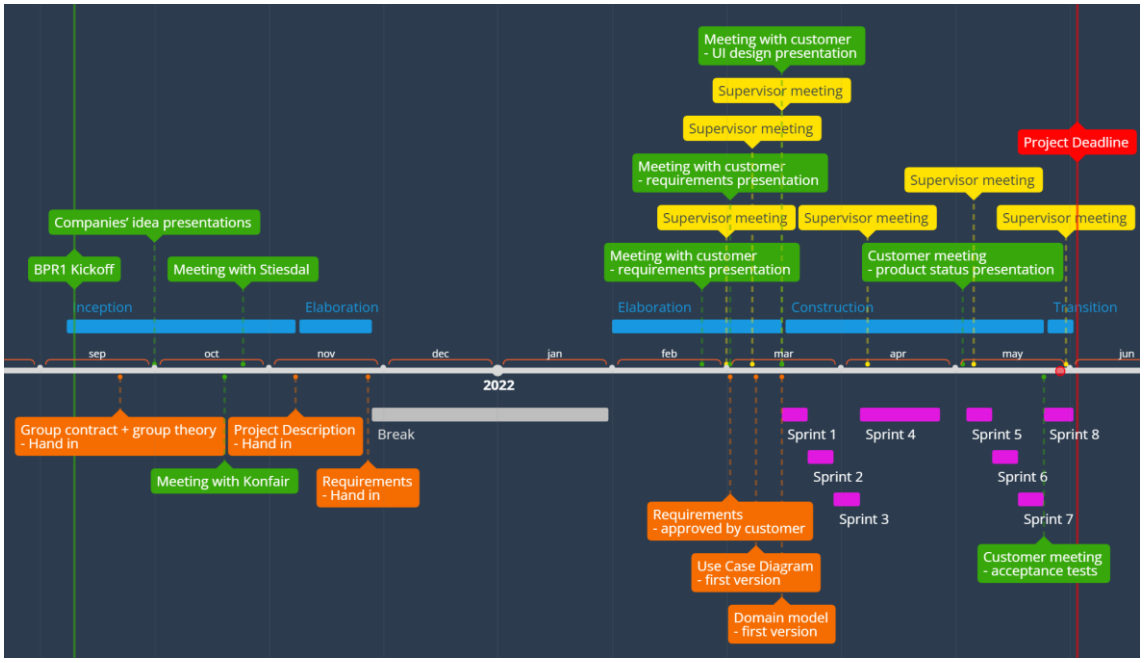
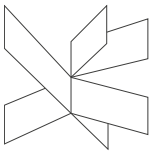


Figure 6 - Project execution timeline

The first three sprints were according to the planned timeline. During those sprints, we realize that we are too ambitious while planning. We were underestimating the workload for the task defined. We also understood that some of the requirements were too big to finish in one or two sprints, especially while the workload during this period was only two days.

The fourth sprint was not set in the same timeline as planned as it has been extended (details on why we did this can be found in sprint 3 – chapter 5.2 Sprints). At this point, we also realized that we would rather have fewer but longer sprints. Therefore we have planned the rest of the sprints as shown in figure 6.

During the sprints, there was a continuous push from the customer to add more features or requirements to the system some of which were important to the system.



Some of them were accepted and added to the requirements and some of them we had to say “for version 2”. We were learning a lot and getting smarter about the problem as the sprints went on.

We had a meeting with the customer while moving from the construction phase to the transition phase (end of sprint 7). During that meeting, we made acceptance tests together with the Konfair employees. We also got access to the company server. This allowed us to test the system also in the Konfair environment, it worked well.

Because the construction phase ended without any requests to change. In the last sprint, we continued focusing on testing, fixing small bugs inside the system, and finishing documentation.

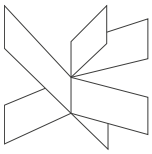
At the end of the elaboration phase, we had done an analysis and figured out and agreed on requirements, made a use case diagram with detailed descriptions of the functionality, made a domain model, and did some sketches of how the customer would like the application to look like. We were ready for the construction phase.

Timetable which can be found in Appendix 9 represents the execution with the short outcome. The outcome can be compared with the stated plan/description of the milestones.

5.1 Tools

5.1.1 Trello

For the first few sprints, the team used Trello for issue tracking. All team members had experience with it before and it did a good job at specifying tasks in swim lanes. We could not specify the time the task takes cleanly.



The alternative to Trello was GitHub projects, which would attach to our project on GitHub. The team had used GitHub projects in the previous semester. The user interface of GitHub projects was not as nice, however, so we went with Trello.

5.1.2 Jira

After using Trello, the team felt like it was lacking information about the sprints and the tasks. We decided to switch to Jira because it had these features and some of the team members were familiar with it.
Jira is a task management software that is made to work with agile methodologies. It provides reports about the ongoing sprints such as a burn-down chart, it also has more information that can be attached to a task, like time or story points of the task or importance level of the task.

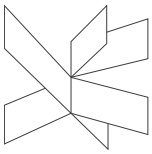
Going with Jira was a very good choice. The quality of our documentation of the sprints immediately improved after switching from Trello.

5.1.3 Messenger

We have used Messenger for both, general chat and video meetings inside the group. The functionality of the messenger was enough to keep the communication efficient and dynamic.

We believe that it was a good choice as all group members were using this platform for personal purposes already. Therefore, everyone was familiar with all the functionality since the first day of the project.

To separate the work from personal usage of this platform we created group conversations where all communication was hosted. Additionally, it came out that besides the communication, this platform was useful for sending quick notes among the members.



5.1.4 Zoom

We made use of zoom in contact with the supervisor and the customer. Since the company of the customer was not easily accessible, some of the meetings were arranged on zoom. Also, meetings with the supervisor were sometimes arranged on zoom. Zoom helped make video calls and also allows the participants to have the ability to share the screen and display presentations to the user. This feature was greatly helpful in the course of the project.

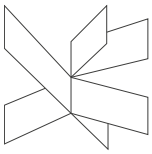
5.1.5 Github

Git was used for source control. It made it possible for the team to work on separate features of the system at the same time and then merge and fix conflicts manually.

At first, the team uses a trunk-based development style. Where we would make changes and merge them into the main branch often. This proved to be unsustainable, as members working on bigger features could not merge so often without breaking their changes, or not being able to compile the application. The team decided to change to git-flow instead as it was more focused on completed feature implementation before merging. This worked very well with our definition of done as everything could be tested in the branch merging and then checked again once it merged, validating the functionality working.

5.1.6 Email

For the sake of this project, we made use of outlook and g-mail for main communication between supervisor and customer. To arrange meetings email was used for communication and arrangement. Also when we wanted to ask questions about the project to the supervisor or customer, email was used as the primary source of communication.



5.2 Sprints

Sprints Overview

The start of the sprints meant the start of the construction phase of the project.

We had a list of ranked user stories, that was our backlog.

A definition of done was agreed upon which meant something is not finished until it is functioning, is tested, and is documented in diagrams and at least noted down in the reports.

More detailed sprint logs can be found in appendix 8 – sprint documentation.

Sprint 1

This sprint was meant as an architecture sprint with a single simple user story of login. There were discussions on what front-end framework the system should use. We knew it was either react or vue.js because we had a lot of experience with both. The team tested two server-side rendering frameworks of Next and Nuxt by implementing similar applications in both of them. It was decided that we will use Nuxt. The team created a login UI, but without the logic in the backend. A plan for the database was made in an EER diagram. A lot of time was spent planning on how to structure the application, as to not cause us trouble down the line. A connection to the database was made from the application but the functionality of querying was still not clean and reusable.

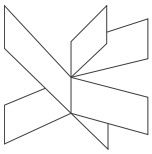
What was completed:

- Part of other tasks

What was not completed:

- Login functionality
- Part of other tasks

Sprint 2



This sprint had a goal of implementing basic login functionality that was not finished in the last sprint and adding some pages that the login redirects to. An emphasis was made to pick simple functionality that could be reusable. We finished implementing the basic login functionality, added a reusable navigation bar, and made a reusable component of a custom table that will be reused often in the application future.

What was completed:

- Unfinished tasks from the last sprint
- Login
- List of control points

What was not completed:

- Create control point
- Edit control point item categories connection

Sprint 3

The sprint goal was to implement some basic functionality for control point creation. This was important to do before starting the implementation of the QA report.

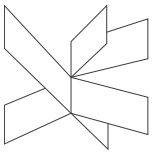
Before the start of the sprint, we added two new requirements to the project:

- **As an Administrator**, I should be able to **specify the input type while creating a control point, so that the QA worker knows what to input**.
- As an **Administrator**, I should be able to **edit the description of the chosen control point, so that it provides the QA worker with correct information**.

What was completed:

- Edit control point attribute connection
- Edit control point description (The one added above)
- Edit control point type (input type) (The one added above)
- Edit control point tolerance
- Add user as administrator

What was not completed:



- (From the previous sprint) Create control point
- (From the previous sprint) Edit control point item categories connection

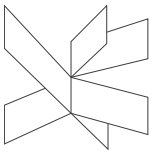
At the end of the sprint, we made a decision. We realized we made a mistake in our sprint planning in the inception phase of the project when we decided to have week-long (37.5 hours) sprints before our elective lessons ended.

We could not possibly do a workload of elective hours + 37.5 hours in a week and the most we could give was two days per week of work for the project. Meaning that the day we had to start the sprint we would have to finish the previous sprint early in the morning and start a new one right after. We would then have a total of one and a half-day to analyze, design, implement and test the user stories that we picked. It was a bad experience.

A change was made, and the sprint time was calculated for what we can give to the project in a week (2 days or 13 h) and extended to as many weeks as 37.5 hours would be. This gives us 6 days in total to work on the sprint. The next sprint would last 2.5 weeks, just enough time to reach the project period.

Due to this change, we could now pick more user stories in the same sprint as we can expect to have enough time to finish them. This also benefits the team members, we will be able to work on separate user stories between each other. Stepping on each other's toes in implementation was a big problem in the previous schedule, not letting the members completely realize their ideas for solving the problem and constantly getting into conflicts on how things should be done. Daily meetings can also be properly utilized.

This was allowed to happen because we are not using strict scrum which dictates the sprints have the same durations. Want to align more to what our scrum process will be later when we transition between phases of our education that do not have the same time allowance.



Sprint 4

We decided to switch to Jira for task management instead of Trello board as it provides burndown chart reports.

In this sprint, we decided to extend the sprint duration due to the issue mentioned in the previous sprint. We decided to extend it to the rest of April as it gives us about a 6 working day sprint which fits well with the sprint duration we are going to use in the May period. Each developer was given some tasks which were chosen specifically so that we do not step on each other's toes.

Each user story was given a unique Id. This Id is used to uniquely identify each user story. Before this sprint, the order of the user story determined its identification, but this wasn't feasible because whenever the order changes, so does the order number.

We noticed a mistake in our requirements. The create control point requirement was prioritized above edit control point requirements and the edit control point requirements had most of the inner control point functionality in them that the creative control point functionality needed. The create control point requirement was left over from the previous sprint because it could not be completed. We changed the edit control point requirements to create control point requirements and instead left a single requirement for editing the control point. This lets us split the create control point into smaller more manageable tasks in the sprint. The change can be seen in the requirements V1 and V2 files in appendix 10.

The burndown chart of the sprint can be seen in figure 7. Note that the sprint ended on April 28, the line extends past that due to us forgetting to mark the sprint as completed.

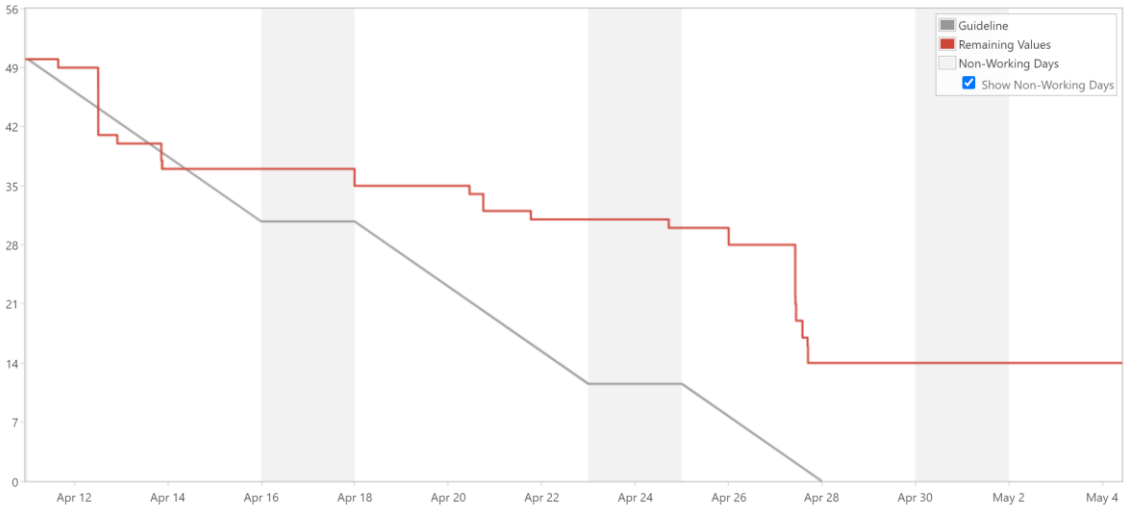


Figure 7 - Sprint 4 burndown chart

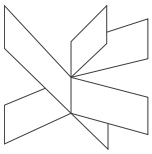
What was completed:

- Uncompleted user stories from the previous sprint
- [REQ-9] Upload a picture when creating control point
- [REQ-12] See released orders list
- [REQ-13] See QA from of chosen order
- [REQ-14] Fill in QA report inputs
- [REQ-17] See item categories list
- [REQ-18] Edit item category frequency

What was not completed:

- [REQ-8] Specify frequency when creating control point
- [REQ-15] Submit QA report with inputs (save and complete)
- Other tasks

We had some uncompleted tasks, but the sprint was a big success. Many important features were completed. The team was also very happy with the change in the sprint length that was done in the previous sprint. The adjusted length gave us time to think and implement features well. Testing was improved a lot. Each member of the team had his own set of user stories to do that other members did not get in the way.



When doing sprint review we presented what we have accomplished to Konfair. Konfair's response to the functionality was positive, they gave us more information about their system which they have not given before, specifically what type of database they intended to use for the proof of concept of the system. They suggested we change the picture functionality in the control point and released order QA report, they wanted the picture to be made storable in a folder instead of a database. They also suggested changing the QA report input table contents.

Konfair suggested newer functionality that they did not think about before. We determined that it is not must-have functionality and with their agreement, we left those features for version 2 of the application. The functionality was for administrators to add custom department locations that have item categories attached to them.

Detailed information about each day of the sprint (daily scrum meetings) and backlog can be found in Appendix 8

Sprint 5

What was completed:

- (From the previous sprint) [REQ-8] Specify frequency when creating control point
- (From the previous sprint) [REQ-15] Submit QA report with inputs (save and complete)
- (From the previous sprint) Other tasks
- [REQ-19] List of completed QA reports
- [REQ-9] Upload a picture while creating control point (Requested adjustments by Konfair)
- [REQ-14] Fill in QA report inputs (Requested adjustments by Konfair)

What was not completed:

- Single deployment diagram task

The burndown chart of the sprint can be seen in figure 8.

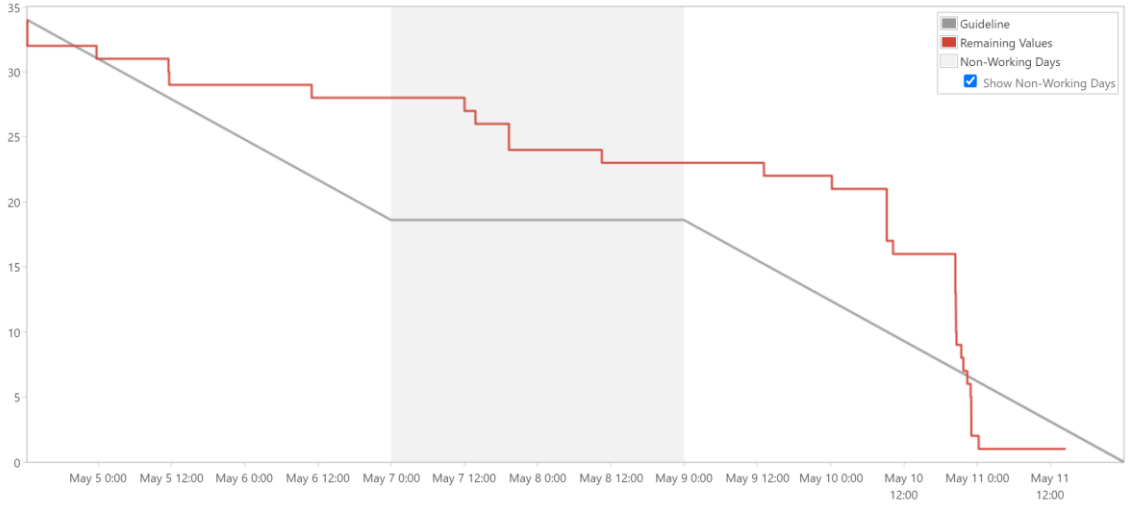


Figure 8 - Sprint 5 burndown chart

Detailed information about each day of the sprint (daily scrum meetings) and backlog can be found in appendix 8

Sprint 6

Before the sprint started, we added a new requirement:

- [REQ34] **As an Administrator,** I should be able to **specify measurement type while creating control point, so that it can be defined if the control point is one time or multiple times measurement.**

the new requirement that was more important we will not be able to complete some requirements due to time constraints. The lowest priority requirement was removed:

- [REQ25] As an **Analyst** I should be able to **use the data for analysis.**

What was completed:

- (From the previous sprint) diagram task
- [REQ34] Specify measurement type for control point (New requirement)
- [REQ33] Edit chosen control point
- [REQ16] Choose location when logging in
- [REQ20] See completed QAs

- [REQ23] Delete user
- Other tasks

What was not completed:

- None

The burndown chart of the sprint can be seen in figure 9

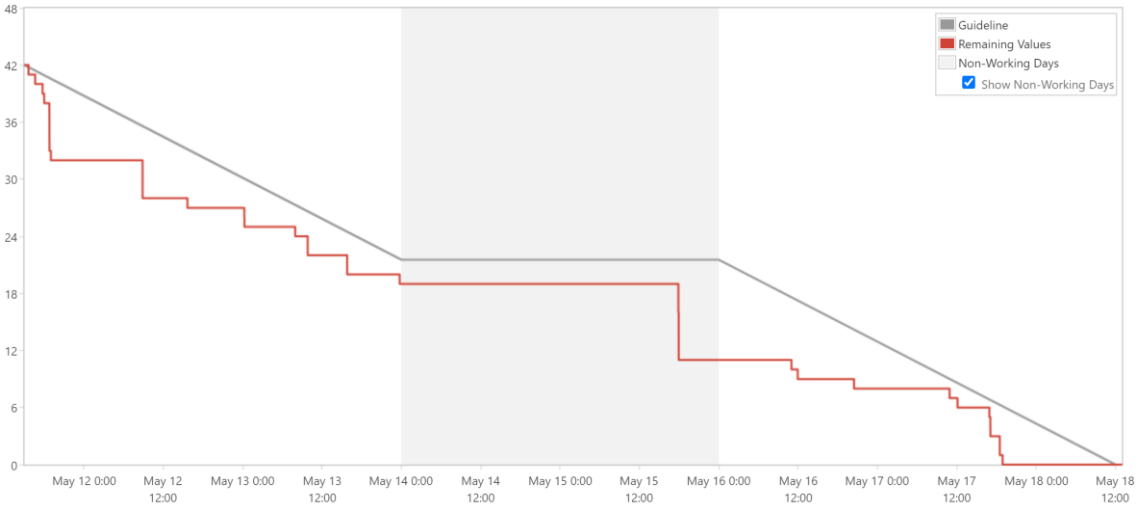


Figure 9 - Sprint 6 burndown chart

Detailed information about each day of the sprint (daily scrum meetings) and backlog can be found in appendix 8

Sprint 7

What was completed:

- [REQ21] Generate pdf for completed order
- [REQ22] Delete chosen control point
- Other tasks
- Acceptance test with the customer

What was not completed:

- None

During this sprint, we were provided with access to the Konfair server and the database. On the last day of the sprint, we performed acceptance tests with the customer which went very well. The customer said we did a good job and that the system worked well enough that they could use it in production as is. We talked about the future of the project as well, they are planning to get some support on the version two features of the system that we did not have enough time to implement. The project was a very big success for us.

The burndown chart of the sprint can be seen in figure 10

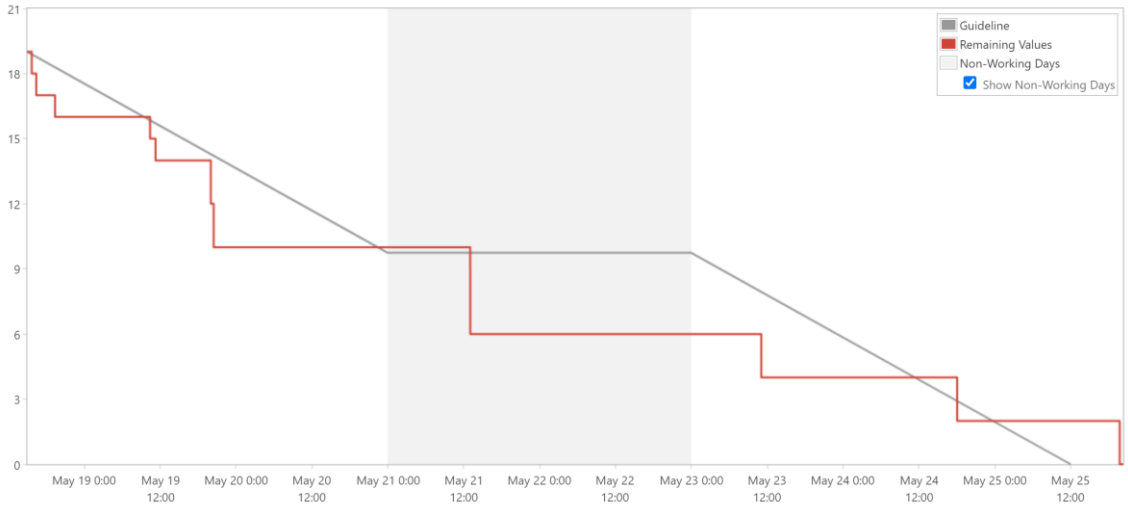


Figure 10 - Sprint 7 burndown chart

Due to a long meeting with the customer, the sprint ended later than planned.

Detailed information about each day of the sprint (daily scrum meetings) and backlog can be found in appendix 8

Sprint 8

This sprint was dedicated to the documentation of the project. We are going to send the project report to the customer so their future team that will upgrade the system will have somewhere to learn about the details of the system.

The burndown chart of the sprint can be seen in figure 11

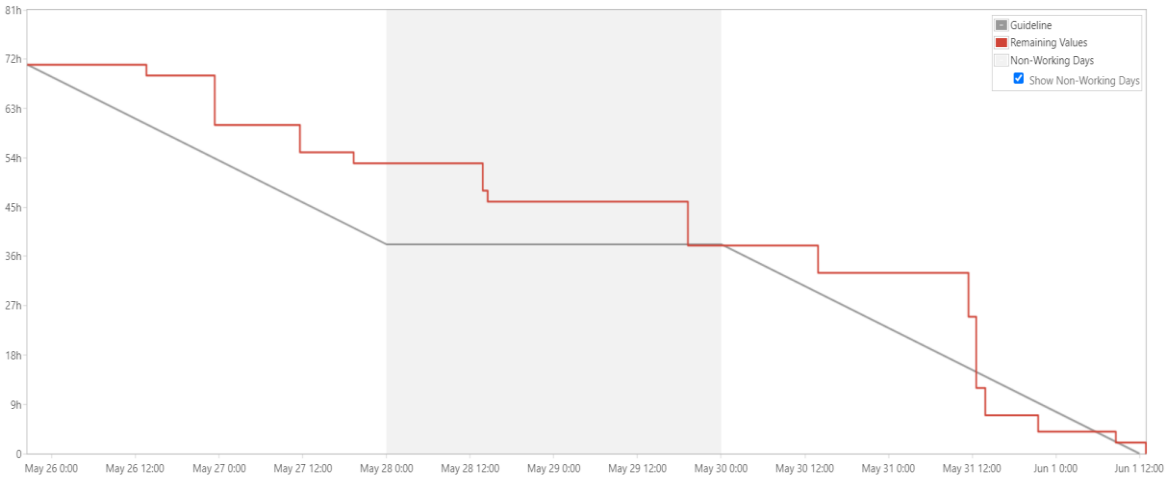


Figure 11 - Sprint 8 burndown chart

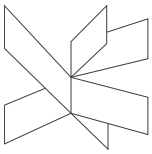
Detailed information about each day of the sprint (daily scrum meetings) and backlog can be found in appendix 8

6 Personal Reflections

Rafal

Choosing the idea for the project was no easy task as I felt that each of the team members has different preferences. Konfair company convinced me to make the project for them at they knew exactly the goal for their project. What is more, they had some preferences and visions on “How”. By the time I pictured how much I can improve my Software Engineering competences by working with them, I knew it is my favorite. As the project felt not big, by this time, I was motivated by having a chance to deliver a production-ready solution.

After we went through some Elaboration and made the first meeting on-site about requirements I did not feel good. I felt like we need to start over. I felt like I don’t understand the process behind Quality Assurance. However, I was happy to spend hours clarifying and picturing the business logic. What is more, we all as a group were active. Due to this, the whole meeting felt professional.



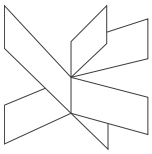
When was the time, I was excited to start the construction phase. I was happy to agree on UI before this phase as I knew this would save some misunderstandings. After a few sprints, I did not feel as energetic as at the beginning. The workload was too big. What is more, some of the user stories I did work on were too big to finish in one sprint. I believe that change in the followed sprints put the project back on track. However, we still had a habit of underestimating tasks. That thought me to be a bit pessimistic when it comes to planning. This approach came out to be good, as our backlogs estimated time increased, stress decreased, and delivered code was just better.

During construction, I had a different point of view on design and implementation than one particular group member. For the short time, I was annoyed by this. However, quickly it turned out that this problem became an advantage. Each time due to disagreements we could analyze the risks of both ideas, learn something from each other and sometimes combine good parts from each of them. I quickly learned that disagreement while having a common goal can factor increase quality.

The cooperation during the project was better than good. I felt that we all care about each other results. I like the rule we had about pointing out assumptions while working with the customer as this way we avoided mistakes and extra work. Due to this, recordings of meetings with the customer were not only useful to review feedback bad also to validate assumptions.

I felt a lot of pressure from the customer as we were constantly getting some new ideas/requests. I learned from my supervisor how to find the balance between the two sides of the project.

Not mentioning the acceptance test, the project was a great success as the customer claimed that this is something he is willing to use in production. I felt good about the job I did.



Finally, I would like to say thank you to my group members, to Konfair for bringing the project, and to the supervisor for support. Last but not least thank you to VIA University College without which gaining such competence in Software Engineering would not be possible.

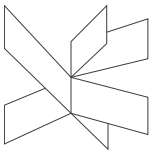
Simon

This project has been a successful one for me. I have worked for over one year with the best and most professional group mates. In the beginning, it was important for me to work in a group that not only wanted to get a good grade but wanted to make a good memorable project that we would look back on and be proud about. I had always worked with Rafal in a group, and I intended to continue working with him. It was also good to work with someone with good experience and Rokas was a good fit with lots of experience working in a company. Choosing a project was an enjoyable and good challenge for me as I always wanted to create a project for a real client. I worked with Rafal and Rokas in a rigorous and thorough process of choosing a project and decided to go with Konfair over Steisdal as they had a well-defined scope.

Working on this project has given me very good experience in organizing a project for a client. I learned how analysis is very important in the process of a project. The company had different ideas as opposed to us many times and many times I thought I understood the analysis of the project for them to backtrack or correct the misunderstanding. The whole process of meeting with customers helped give an in-depth understanding of the disparity between a customer and developers and how it is important to focus deep down on analyzing the project properly when faced with real customers and even when making a personal project. As part of this project, I also learned how to use new technologies such as Nuxt.js and Express

A very big aspect of this project was learning how to give scope and definition to a limitless project. It was really helpful to work with the supervisor on this as he always reminded us to keep in mind what can be achieved to prevent unrealistic goals.

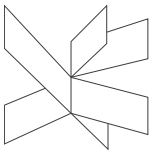
Working with a company that had never had in-house developers meant that they had an endless pool of requests on how the system should work and what features should be implemented. It was up to us, to analyse the most important requirements and weed out the less important ones for the next feature. Even taking into consideration the



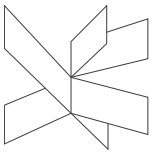
most important requirements there was a necessity to prioritize these requirements using Moscow principles. It was important for me to realize that the company always had something extra or some new feature they always wanted to add and we had to scope down to what we could perform with our time.
This was the last project I will make in my bachelor's education, and it was important for me to see just how far I have come as a software engineer. This project has helped me to realize all the technologies I have learned over the past years. For example, I was able to adapt my technology span just by analysing the requirements of the customer. I decided together with the group to go with a technology that we hadn't used before and felt comfortable using it. This was an experience that I have gained from this project that, there will be a need to adapt to different technologies depending on the necessity of the clients.
In conclusion, I am glad to conclude this project on a high as the customers were satisfied and glad about the proof of concept, we presented to them. Working in this group with quality group members has shaped my knowledge span and has thought me more about the technical aspects of programming such as good coding practices and will be very memorable to me. I also, want to thank my group members, supervisors, VIA University, and client for giving me such a wonderful learning experience.

Rokas

This project has been a very good experience for me. I feel like I had a lot of time to apply new things I have learned in testing, more than I could do in my part-time job. I feel like I learned a lot of things this semester that I can use to make working easier as a consultant I think the best skill I learned it was to say no to the customer and push things for version two of projects. This was thought to us by our supervisor. I have struggled a lot with this at my part-time job until now and I have noticed my mangers also struggle with this. I did get overwhelmed with the details though, there are some things that are a given in web service development, like timeout handling that I never thought about explaining in a report and doing it is exhausting.

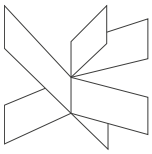


About my team, I feel like my decisions were sufficiently challenged to a fair degree. I had been in projects where my team shoots down ideas out of sheer laziness, but this time was different. I could argue and change my teammates minds on something, and they were able to change my mind on some things. I am very sure that I will use what I learned in this project and that this project will provide me with more opportunities. We got a reference from KonfAir because they really liked it, it will make me find a new job a lot easier if I need to. Overall I feel like the project was very good.



7 Supervision

In general, the cooperation with the supervisor was very satisfactory. The Supervisors were always readily available to help and assist with all problems that we faced during the group work. One of the many scenarios where the supervisor was very helpful was on 09-05-2022 when the supervisor helped resolve a situation in the group where we couldn't move forward. The group had found a problem during the execution and two members of the group came up with an idea on how to solve it. The supervisor understood the problem and allowed us to shed light on the pros and cons of each idea that the group members had. At the end of the day by giving the group a 3rd party to help remain neutral and un-bias the group came to a resolution to its dilemma.
Cooperation and communication with the supervisor were as often as needed. During the analysis and design phase of the project, there were a lot of questions that could be asked to the supervisor and therefore meeting schedules were almost weekly however, during the start of the implementation, the supervisor meeting was less frequent. The meetings held with the supervisor were mostly physically in a class. We would send the report or whatever agenda needed to be discussed at least a day prior and the supervisor would then look at it and would be discussed it in-depth in the upcoming meeting. Sometimes, if the supervisor or any of the group members could not meet physically, they would join via online zoom meeting.
Also, a very important and successful aspect of working with the supervisor is that he helped the group in realizing the scope of the project. He helped us identify a balance between the customer's needs and the group's work. This helped us identify a well-defined scope of what should be done and what could be carried over to version 2. This way proper time and care were devoted to each task that was performed and ensured customer satisfaction.it also helped remove unrealistic goals
One less successful aspect of the supervision is the fact that the business logic of the project was very complex, and it took much time for the group to understand it sometimes, the supervisor couldn't assist with some specific tasks as he did not fully



understand the business logic. However, the supervisor still tried his best to assist with whatever knowledge that could be offered which was more than enough.

8 Conclusions

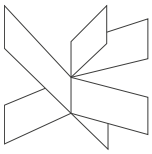
Overall, the team has had a very important and interesting process. Being the last project of the group's studies, we all stayed motivated throughout whole the period.

Due to real-life customers, the attitude to work on the project was professional. The whole process was very dynamic and required consultation with the customer before making decisions. This led to either doing something in assumption or delaying the work until the customer clarify the problem. During this project, we have learned to avoid the assumptions as this had worst consequences than postponing the work. The understanding of the KonfAir processes was the most difficult part of the project.

The communication with the customer was good enough. However, sometimes we had to ask a quick question. During those moments we wish we would have agreed on communication via phone calls to some extent.

When it comes to the process of implementation inside the group. It was really important to present shared functionality to other group members as this way others could make use of it.

Once we got the acceptance test the whole group felt happy to test the product with the aimed users (Administrators from the office and QA workers from production). This was a great idea to ensure that system is easy enough to interact with. Since the result was good and we have tried to replicate the KonfAir environment in the early state, we did not face any problems during the transition phase.



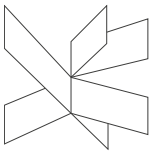
This all was possible due to considering the end environment throughout the whole project period as well as constant communication with the client.

The recommendation given by KonfAir outline also good process through the development of the system (Appendix 12 – Reference from Konfair).

9 Source of Information

Estimate. 2022. *Profiles and testing for recruitment and development | e-stimate*. [online] Available at: <https://www.e-stimate.dk/en/> [Accessed 19 October 2021].

Hofstede Insights. 2022. *Home - Hofstede Insights Organisational Culture Consulting*. [online] Available at: <https://www.hofstede-insights.com/> [Accessed 30 October 2021].



10 Appendices

Appendix 1 – Group Contract

Appendix 2 – UML Diagrams

Appendix 3 – Use Case Descriptions

Appendix 4 – Source Code

Appendix 5 – Use Case Testing

Appendix 6 – UI Layouts

Appendix 7 – Project Description

Appendix 8 – Sprints Documentation

Appendix 9 – Timetable

Appendix 10 – Requirement Change Over Time

Appendix 11 – User Guide

Appendix 12 – Reference from Konfair

Appendix 13 – Frontend and backend test result