KWaSA

Kulla Warehouse Storage Application

TCS Design Project
Unbrickable - warehouse

Angela van Sprang
Anissa Donkers
Kevin Böhmer
Pieter de Regt

Supervisor: Nacir Bouali
University of Twente
April 2021

ACKNOWLEDGEMENTS

This design project is performed for a graduation module of the Bachelor Technical Computer Science (University of Twente).

Everyone in our group would like to thank our supervisor Nacir Bouali, who has helped us a lot during the project. As lecturer of computer science (affiliated with the DMB research group), you asked critical questions about our design and often pointed to things we overlooked. Furthermore, you often provided us with helpful information. Thank you for all this effort and time, we surely appreciate it.

In addition, we have worked closely with two other project groups (DigiBrick and BrickSync). This was necessary for combining all three applications into a large product for our shared client. Thank you for the close collaboration during the project.

Finally, thanks Peter en Roel Westenberg of Unbrickable for the weekly meetings and close contact during the project. We enjoyed having you as our client.

Angela van Sprang, Anissa Donkers, Kevin Böhmer and Pieter de Regt.

ABSTRACT

KWaSA (Kulla Warehouse Storage Application) is a software system that helps stores trading LEGO bricks to manage and organise their warehouse. It contains three components, namely warehouse management, order picking management and refill management. The system is designed for Unbrickable, a social non-profit organization that buys and sells LEGO. Unbrickable helps teenagers with challenges, such as autism or ADHD, by providing them with a part-time job in a multidisciplinary team within the organization. This report elaborates on the design process of the developed system. All phases of the project are discussed, like requirements engineering, planning and the design of the software architecture. Also, all the design choices are explained and evaluated.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to present a detailed description of the KWaSA (Kulla Warehouse Storage Application) system. It will explain the features and the purpose of the system, the interface of the system, what the system will do and the constraints under which it must operate.
Furthermore, this document will discuss the process of this project. It will give insight into the development of KWaSA and explain the major design choices.
This document is intended for the stakeholders and the developers. But also for others who are directly connected with the development of the project, such as the project supervisor.

1.2 Scope

KWaSA is an application for companies trading in LEGO items, specifically for the client Unbrickable. Its goal is to simplify the filling and management of the storage warehouse and the process of order picking, i.e. preparing the orders for delivery. The system builds on a previous project of the Design Module delivered in April 2020. This system was named Kulla. KWaSA is a new version that will be a functional web application, which will also function as a user interface for the already-existing program components.

The project will be one of the three final components for the client Unbrickable. The other components are developed by two other project groups: DigiBrick and BrickSync. A simple overview of the collaboration between the three components is presented in Figure 1.1. DigiBrick will provide KWaSA digitized items to be put in the warehouse. Additionally, KWaSA will present the current inventory in the warehouse to BrickSync. These components together digitize the entire flow of LEGO items for Unbrickable.

Figure 1.1: A preview of the connections between the 3 systems

1.3 Glossary

Table 1.3 presents an overview of the jargon used in the report. Most terms are also explained when they are used, but this table presents a complete overview.

Term	Definition
Batch	Collection of one or more LEGO items to be put in the warehouse (consisting of one or multiple trays)
Client	Unbrickable
DevOps	It creates a work environment, culture and set of practices in which software developers and IT operations constantly communicate and collaborate to rapidly bring products and services to market [1]
(LEGO) Item	A LEGO item is a mold combined with a colour e.g. a red 2x2 brick, but also an instruction booklet
KWaSA	Kulla Warehouse Storage Application
Lots	The number of unique items in an order
Modal	A web page element that displays in front of and deactivates all other page content
Mold	Blank representation of a LEGO item
(Order) picker	Someone whose job it is to collect each item of an order from the warehouse. Also described as user
Picking	the process of collecting ordered items from the warehouse
Progressive web app	A progressive web app uses the same website for mobile and desktop. It is not required to install the application from the store, it is delivered in the browser.
Project	Several batches
Refiller	Someone whose job it is to restock the warehouse. Also described as user
Team	Referring to the developer's team of KWaSA. The members of this team are Angela, Anissa, Kevin and Pieter
Tray	Collection of one or more LEGO items to be put in the warehouse and content of a batch
User	This person could either be an order picker or refiller

Table 1.1: Glossary

1.4 Overview

The succeeding chapter, Background, gives more in-depth information on the organisation Unbrickabe, their current workflow and the system Kulla. It also presents the findings of some research on warehouse management software. Next, the proposed system will be discussed. This is based on the project proposal that has been agreed upon by the company during the first phase of this project. In this proposal, the features of the system were noted. These features were later on translated into initial user and system requirements. During the development of this project, the requirements were constantly slightly altered. The process of this can be found in the chapter Evaluation. The final requirements can be found in the chapter of Requirements Specification. Where and how these requirements were fulfilled, can be found in the High Level and Detailed Design chapters. The introduction and the requirements specification were inspired by IEEE Recommended Practice for Software Requirements Specifications [2]. Furthermore, the methods and results of testing are described in the chapter Testing and the work on documentation is described in the chapter Documentation. The results of the tests can be used for future work, which is discussed in the chapter Future Work. Lastly, this document will give a detailed evaluation and conclusion of the project. Not only an analysis of the project organization, planning and requirements is given, but also a discussion on the contribution of each team member.

2 BACKGROUND

This chapter will discuss the background of this project. The topics that are highlighted are: the background behind Unbrickable, their current workflow of warehouse storage, Kulla and warehouse storage software in general.

2.1 Unbrickable

The system is designed for Unbrickable, a social enterprise that tries to help teenagers with challenges, aged 13 to 21, to get more work experience and to let them develop themselves [3]. By doing this, they offer them an opportunity to prepare themselves to work in a real workspace.

Besides teenagers with a challenge, they also work with teenagers without a challenge. They are put in a group where they need to work together. They learn how to collaborate and how to work together towards a common goal: keep Unbrickable healthy so that they can help more teenagers [4].

2.2 Warehouse organisation

At the moment, the warehouse is only stored in human memory. This means that the inventory is not well digitized. The warehouse has around 300.000 items in their inventory of which 10.000 are unique. When an order picker picks an item from the warehouse, they have to know the location of the item from memory or have to search for the item. The location of an item is not documented. This causes an inefficient workflow and for a new employee, it results in a long period of time before settlement.

Also, the restocking process contains this inefficiency. When the warehouse receives new items, the stock gets updated. But after that, it needs to be put in the warehouse. The refiller takes the box with all new items and searches for the right location for the items.

2.3 Kulla

Kulla was the result of the Design Project of Joost Loohuis, Jelle Maas, Nils Van Noort and Frank Stapel [5]. They have created a lot of functionalities that were required by Unbrickable. However, after a few months of deployment, the system was not operable as there was no development environment for Unbrickable. They were not able to modify the code easily, which made it difficult to maintain.

Although it seems that the requirements for Kulla are similar to the ones for KWaSA, there were major changes. But thanks to the first version, this project was able to quickly make a lot of progress.

2.4 Research into warehouse management software

Warehouse management software is not uncommon to help managers control warehouse operations. It is helpful to look into the available literature on this to retrieve the features that determine whether the software is actually helpful.

First, it is important to define what the software should do and what benefits it should bring. Assuming that the software is designed to manage all processes, including intake and storage of inventory and order picking, it would be good if the software removes paperwork from these processes and reduces human errors. This reduces the time needed to pick items and organize the warehouse, which could make a difference in customer satisfaction.

There are several features in good warehouse management software that should bring these benefits [6]. The first is inventory control. Monitoring the inventory levels in real-time, to prevent under- or overstocking. Secondly, product movement is important, which is the ability to track and manage the placements of the products within the warehouse. Thirdly, software integration with all parties within the organization is important to keep everyone up-to-date with the product movements, especially when the other parties are also part of the automation process.
Furthermore, multiple device support is encouraged so that no employee is tied to their desk. It is good to note that these features have also been described by Unbrickable as their wishes for the application.

3 PROJECT ORGANIZATION

At the start of the project, a meeting was held to determine the scope of the project and its requirements. During this first meeting, an introduction of Unbrickable was given to show where the application would be used and how it should be working with already existing programs and the products of the other project groups. For the rest of the project, weekly meetings with the client were held and meetings with the other project groups were scheduled when necessary. One team member with the role of contact-person managed the contact between the team and the other project groups and the client.

The project made use of Agile development, which is a project management methodology [7]. It breaks the project into multiple sprints and actively involves the client in the development process, enabling the continuous implementation of their feedback.

Within this project, sprints of one week were defined. All tasks involved in the development of the product were collected in the backlog and each sprint started with the planning of tasks from the backlog. The tasks for the corresponding sprint were divided amongst the team members. During the sprint, daily meetings took place to discuss the progress of the previous day, the activities of the current day and whether team members needed or could provide help. At the end of the sprint, a sprint retrospective was organised in which all members evaluated the sprint and the progress made.

A visual overview of the tasks and their progress was presented on the scrum board, for which the application Trello was used [8]. All team members were required to update the progress of their tasks on the scrum board. The scrum board was also shared with the supervisor.

Furthermore, one team member was assigned the role of scrum-master. The scrum master was responsible for organising the daily stand-ups, sprint planning and sprint retrospective. They will also keep the scrum board up-to-date.

Most of the developing tasks were assigned to a pair. This was done to work more efficiently, with the intention to deliver work with fewer little bugs and within a smaller period.

Additionally, the client and supervisor were kept in the loop during each sprint. There was at least one meeting per sprint with the client and one with the supervisor. The progress was discussed and a demo was given when applicable. The client was free to - and even encouraged to - provide feedback on the product, which was taken into account in the following sprint. Next to this meeting, there was constant communication with the client and supervisor via email or Microsoft Teams. The team had contacted the other project groups via Discord and email.

Lastly, it is good to note that the team tried to have good communication. This could have been difficult since all communication was done online (considering the lockdown due to the global COVID-19 pandemic), but the team tried to make the most of online work. First, all team members tried to stay online and accessible during workdays from 9 AM until 5 PM. Everyone tried to be accessible on the Discord server where communication with all or individual team members was done via text, speech or video. Even when a team member was working individually, we considered it good practice to be in a voice channel alone, so that any team member could join any time and ask or provide help. This made it easier to ask for help and work together.

Furthermore, the daily stand-up at 9 AM was also important, as we all started our working day together and kept speaking to each other regularly. Lastly, we asked for feedback regularly, whether to the client or ourselves and tried to keep an open atmosphere where people gave and took feedback nicely.

4 PROPOSED SYSTEM

In this chapter, the final version of the system proposal for the client is explained. This proposal was conducted after four reviews of the document and contains the vision of the overall project. In Chapter 11, there is a reflection upon this vision. The topics that will be included in this section are the features that can be implemented, their priorities, the collaboration with the other project groups (DigiBrick and BrickSync), the enabling technologies and a risk analysis.

4.1 Features

In this section, an overview is given of the discussed features. These features were established during the first two meetings with the organization Unbrickable. There are a lot of features that could be added to the system. It was difficult to decide which are acceptable since the development team was not yet familiar with the Kulla platform and its framework. In the Evaluation section of this report the overall process is discussed. Here, the proposed features are mentioned.

The final deliverable for this project is a progressive web application that can be used on a computer and on a smartphone. The web application has different features to manage the warehouse. Kulla started on an implementation of the database for the warehouse management and a simple web application to interact with the database.

Even though the web application can be used on a computer and smartphone, the order picking and filling processes are meant to be executed on the smartphone. So these views are supported for the smartphone, while the other views are supported for the computer.

Due to time constraints and continuity, we will continue to build on the implementation of Kulla.

4.1.1 Warehouse inventory

The webapp should have a screen which displays all the LEGO items currently in stock in a list view. Each item should have a name, image, amount in stock, location, private note, public note, etc. It should be possible to click on the image to enlarge it. It should also be possible to go to the catalog page of the item, possibly in a new window. This list should also be sortable and searchable. There should also be a possibility to go from an item in the inventory to the item in the catalogue.

4.1.2 Warehouse tree view

One of the features of the webapp is a tree view of the database. The user should be able to see a simple overview of the database. The client suggested a tree view for this. We also believe this is the most convenient way to interact with the database and it also conforms to the guidelines for an helpful warehouse storage application (see Section 2.4). A tree view makes it easy to show the nested locations of a LEGO piece. For instance, a warehouse can have multiple racks. Each rack can have multiple shelves, each shelf can have multiple boxes. A

Figure 4.1: A preview of the tree view

box can hold multiple bags. See Figure 4.1 for an example visual representation. For the final product, there should be more information added to each section.

Drag and drop

The user must be able to move items from one subgroup to another, this can be implemented with a drag and drop mechanism. The user can select one item and drag it to a different location, all the child locations are also moved. Another method is also allowed, as long as the system is fairly easy and quickly adaptable.

Add and remove items

The warehouse manager must be able to remove LEGO pieces from or add LEGO pieces to the warehouse.

Add and remove warehouse components

The warehouse manager must be able to remove or add components to the warehouse, i.e. shelves, racks, boxes, etc.

Change stock

The number of items of a particular LEGO piece must be displayed to the user. The user should be able to change this amount.

Alteration warnings

For every alteration in the warehouse the warehouse manager should be warned about his action. This could be done with a warning message or with a confirmation button. This can prevent database changes that are not intended. These alterations must also be recorded in the logging.

Reserved stock

The warehouse should have support for reserving items. If an order is made, those items are reserved to prevent the sale of sold items. The application made by the group of BrickSync will communicate a list of ordered items. The database should have support for indicating which items are reserved. The application of BrickSync will update the stock of the platforms BrickLink and BrickOwl. The warehouse manager should be able to check the quantity and reserved quantity of a particular item in the warehouse.

Recreate warehouse section

In order to make it easier to build a warehouse quickly the tree view should have an option to copy a part of the warehouse. For example, a particular template consists of two racks with three shelves and two boxes. The warehouse manager must be able to copy this selection and paste it elsewhere in the warehouse.

4.1.3 Warehouse optimizer

Warehouse issue report

There should be a page which displays all the reports of the order pickers and order fillers warnings which are described below, so the warehouse manager can view, solve and delete the reported issues.

Box reallocation suggestion

A warehouse consists of different boxes where LEGO is located. It can happen that one of the boxes becomes too small. In this case it is a good idea to move some or all of the bags to a bigger or different box. The same holds when a big box only contains one item, then it is better to move this item to a smaller box and use the big box for different bags or items. So when the user is going to refill a box or pick an order, the system should give a suggestion to move the other items as well.

Box location optimizer

The company Unbrickable sorts all LEGO pieces of new LEGO sets and puts them in the right bags. Most of the LEGO sets consist of a lot of different LEGO items, which are put manually in the warehouse. In order to optimize this process, it can be very handy to put boxes with items that occur often together close to each other.
This part can be separated into two subfeatures. One subfeature is an optimizer based on an order, where the application optimizes the warehouse looking at the history of orders. The items which are often ordered together, are also put together.
The other subfeature is based on a LEGO set. Whenever a LEGO set is created by the organization, these items are put together.

Balancing

Suggestions should be made to the warehouse manager to check whether the actual stock of certain items corresponds to the administration in the database, which is called ‘balancing’. These items are not chosen randomly, but are known to not have been checked lately as they have not been ordered or balanced lately.

4.1.4 Order picking

Unbrickable sells LEGO items to buyers, but before the LEGO items can be sent, the items first have to be picked from the warehouse. Currently, this process is executed manually. The webapp should have an option to select the job function “order picking”.

View

For the order picking process, a new view is created. The window shows the current tasks, the location of the item, the current stock and an image of the LEGO piece. When there is a problem with an item, the order picker can report this problem in this window. This report will be visible for the warehouse manager in his dashboard. Problems that can be reported are for instance that the current stock does not match with the number of items in the box, or that a LEGO piece is damaged and not usable.

Cooperation

Some orders can be very big. To make it easier and sometimes more enjoyable, two order pickers sometimes work on the same order. Thus, the application should support the option that two order pickers are working on the same order, in such a way that they cannot pick the same item: a possibility is that one will start working through the list of LEGO items to be picked from top to bottom, and the other from bottom to top.

Issue input

Whenever a user is in the warehouse and registers a mistake in the stock (e.g. there are a different number of items than expected, or the box is too full), then they are able to create an issue and formulate the problem, which reaches the warehouse manager.

Shortest path

The order picker feature should give the shortest route to the order picker for collecting the items. This means that the user does not have to go to a location that he already visited.

Time management

The application should have time management for users such as order pickers. When an order picker starts with an order, the time must be tracked. There should be an option to start and end a break. When the order is finished or the user leaves the order picker screen, the time must be stopped.

4.1.5 Restocker

The web application should support adding new items to the stock, i.e. order filling. A list of the items in the order that should be added in the warehouse should be snown in the web application.

Refill

When given the list of items that should be added in the warehouse, the user gets an overview of the location for each item. When an item has multiple locations, the user can choose between the locations. They can then restock the warehouse with the items on the given locations. They can also search through and filter the list on aspects such as colour and number to refill the warehouse efficiently.

Reverse order

Sometimes, orders that have already been picked need to be cancelled, for example due to the payment holding off. Such an order is called a 'reverse order' and all picked items should be put back in the warehouse. It is known what is in these orders since they have been registered in the list of order pickers. They are handled in the same way as other refills. The order manager should have the functionality to change an order into a reverse order.

Reverse order assistance

When an order is cancelled and an order is already picked, the items must be returned to the warehouse. The user must receive assistance on where these items can be put. Some parts of the UI elements for the order picking and filling process can be reused for this.

4.1.6 Roles and rights

To protect the database, not every employee of Unbrickable will have all functionalities in the application. The roles system was already implemented last year, therefore we can use this implementation in our web application. For Unbrickable this will result in three users, namely the administrator, warehouse manager and the general user. The general user can use the order picker feature, filler feature and other basic functionality. The warehouse manager can use the warehouse tree view and the warehouse optimizer. The administrator has all privileges and has full control over the warehouse and its content. The administrator also assigns roles to each account.

4.1.7 Logging

Every change in the warehouse must be recorded. Logging changes in a text file is sufficient. These changes could for example be, counting the items within a box, taking an item from a box for an order or changing the location of an item.

4.1.8 Warehouse download

The application should provide the functionality that at each moment, the current stock in the warehouse can be downloaded as a BSX file, a special file format for LEGO. The system should also be able to import the exported file

4.1.9 Reliability

The system must be reliable. Data races and race conditions should not happen. Because it is possible that multiple users are working simultaneously, it must be assured that they do not overwrite each other's values. Another data race that can occur is when the warehouse manager changes the stocks of some items and the user finishes picking an order.

4.1.10 Internal order

Sometimes, the company needs LEGO items from the warehouse for internal projects. For these occasions, the order manager must be able to import a BSX compatible file that contains the order. This order can be picked by the order picker.

4.2 Priorities of the Features

Each feature is labelled as must, should, could or don't, according to the MoSCoW principle (see table 4.1) [9]. The "must" features have the highest priority. The functionalities are extended with the "should" features. Unbrickable described these features as "nice to have". These were worked out when the time was available, as they did not have the highest priority. Some features are labelled as "could". These features do not have any priority. It would be nice to have, however, these will be implemented only if all the other "should" features are completed. A reflection upon the implementation of the features considering their priority is given in Chapter 11.

Feature	Must	Should	Could
Warehouse inventory	X		
Warehouse tree-view	X		
Warehouse tree-view: Drag and drop	X		
Warehouse tree-view: Add and remove items	X		
Warehouse tree-view: Add and remove warehouse components	X		
Warehouse tree-view: Change stock	X		
Warehouse tree-view: Alteration warnings		X	
Warehouse tree-view: reserved stock		X	
Warehouse tree-view: Recreate warehouse section			X
Warehouse optimizer: Warehouse issue report	X		
Warehouse optimizer: Box reallocation suggestion			X
Warehouse optimizer: Box location optimizer			X
Warehouse optimizer: Balancing			X
Order picking	X		
Order picking: view	X		
Order picking: cooperation	X		
Order picking: issue input	X		
Order picking: Shortest path		X	
Order picking: Time management		X	
Restocker	X		
Restocker: Refill	X		
Restocker: Reverse order			X
Restocker: Reverse order assistance			X
Roles and Rights	X		
Logging	X		
Warehouse download	X		
Internal order			X

Table 4.1: Features prioritized

4.3 Collaboration

As discussed previously, KWaSA is designed to be used with the two applications from DigiBrick and BrickSync. A clear overview of the collaboration is shown in image 4.2, which shows the tasks of DigiBrick in orange, the tasks of KWaSA in turquoise and those of BrickSync in purple. When Unbrickable receives new LEGO items, DigiBrick will digitize them and pass them on to KWaSA to add them to the warehouse. In more detail, whenever a new LEGO set is received, DigiBrick will create a new project with one or more batches (collection of one or multiple LEGO items, see glossary). The batch undergoes the 'digitize items' process. When a whole batch is done, it will be sent to the warehouse so that warehouse employees can add the items to the warehouse inventory. During this process minor mistakes can happen, that is why it is still possible to change the batch in this phase. As soon as the employee starts with 'filling' the batch cannot be changed anymore.

When all new items from a batch are completely put in the warehouse inventory, the Bricksync group will plan the whole batch in for synchronization. When the time has come, the stock will be updated and be synchronized with the several selling platforms. The flow of information between KWaSA and BrickSync is as follows: first, a client places his order via a selling platform. Then, Bricksync receives the order information and puts it into the database. Finally, the information is shown on KWaSA and order pickers will collect the ordered items from the warehouse. This

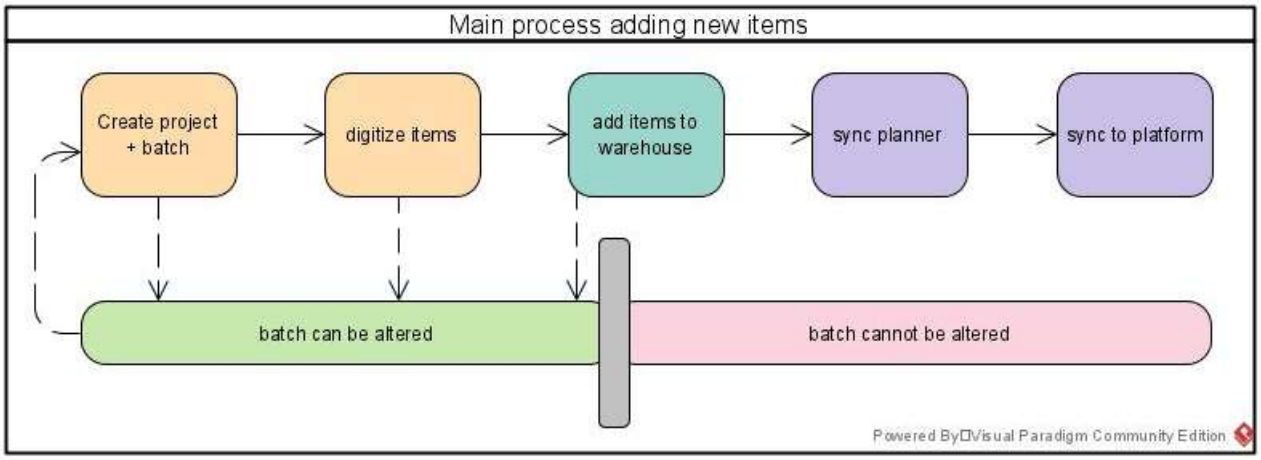

Figure 4.2: A preview of the connections between the 3 systems

completes the process of making an order. The system does not support further steps for the order, i.e. labeling and shipment.

4.4 Enabling Technologies

The application KWaSA is built with a few enabling technologies. The front-end is written with the progressive JavaScript framework Vue.js [10]. As for the framework of the interface, the system works with Materialize, which is a responsive front-end framework based on Material Design [11]. For the back-end KWaSA uses the high-level programming language Python [12] and the necessarily SQL [13] scripts.

These enabling technologies of programming languages and the environment were already used for Kulla. For the sake of continuity these are also used for KWaSA.

There were three different programming environments used namely, Datagrip for the database tier, PyCharm for the back-end and WebStorm for the front-end. For the version control Github was used.

4.5 Risk Analysis

There are many possible threats that may harm the project. Of course, not all can be foreseen. However, some main threats are always good to identify, so they can be taken into account, and the probability of those threats happening, or the impact of these threats, can be minimized. With this risk analysis, exactly that is attempted to be accomplished.

First, a possible threat is sickness of one of the project members. Especially during the current pandemic, this is something to reckon with. It is the individual task of each group member to abide by the active restrictions set by the Dutch government, to minimize the risk of being infected by the Coronavirus. However, there is still a possibility that one of the group members gets sick anyway, whether this is Covid-19, the flu, or something else. Then, this person should inform the group as quickly as possible. In this way, the group can decide how to handle the situation, by dividing the work over the other members, or by finding another suitable solution. The project supervisor and client could also be informed, if deemed appropriate.

A second possible threat is that one of the group members quits with the Design project, for whatever reason. Of course, this should be prevented at any costs, as this would have a big negative impact on the project. By involving all members of the group with the project, keeping

each other up to date and creating a positive influence, this risk can be minimized. If one of the members still quits, this should immediately be communicated to the whole group, which should then inform the project supervisor, the client and the module coordinator. With their help, an appropriate solution can be determined, which could involve lowering the workload by the client.

Time consumption issues can also be a big threat. Next to this design project, group members have other time consuming activities during the week, such as other courses and the reflection component. While these are taken into account in our planning, they may turn out to be more time consuming than seemed at the beginning. This can negatively affect the planning. To prevent this, each group member should be aware of this and manage their time to the best of their ability. If issues arise, the member should never hesitate to inform the group, so a solution can be discussed together.

Other time consumption issues that can arise are the incorrect estimation of time needed for a certain part of the project. This is very likely to happen, as estimating the amount of time needed for tasks is very difficult. While preventing it is hard, these risks can be managed by acting on them early. With the daily standup, these risks can be identified early on, and the planning can be adjusted accordingly.

Furthermore, this project has an extra difficulty, as it is in collaboration with two other project groups, which have the same client. Technical problems could occur during development and/or operation due to their dependence. For example, all three groups require access to the same database with Lego items and for operation it is helpful to have all code accessible in the same (GitHub) repository. To minimize the risk of any (technical) problems occurring, communication between the groups is key. That is why meeting with the other groups regularly is advised, and a platform should be created on which the teams can communicate freely with each other. To prevent chaos, one member of each team should be assigned that takes care of the communication with the others groups.

Finally, a good constraint to acknowledge here is a certain limitation in communication channels. Due to the Covid-19 pandemic, we are forced to work online. This can make communication more difficult and lead to miscommunications, since no other communication is possible than voice or video. If miscommunications occur too much, this could lead to unnecessary work, annoyances, and even an unusable product. That is why the risk of communication should be minimized. This is done by creating an open, friendly atmosphere in the group, so all members feel comfortable asking questions and speaking up. Also, daily standups will keep all team members informed about what the others are working on. Finally, the weekly meetings with the client, and the project proposal, should minimize risk of miscommunication between the group and the client.

5 REQUIREMENTS SPECIFICATION

As mentioned in the Project Organisation, Agile practices are used to manage the project. One of the principles behind the Agile Manifesto is to always welcome changing requirements, even in late development [7]. This was a wish from the client. For the agile approach, SCRUM was used. This resulted in short sprints, as mentioned before. In each sprint some use cases were tackled. These were prioritized with the use of the MoSCoW method. In this section an explanation is given on how the use cases were conceived with their given prioritization.

5.1 Product Perspective

The result of this project will be, together with other components of the two other design project groups, a totally self-contained system. The group of DigiBrick will be in charge of the process of incoming LEGO items. The Bricksync group will process orders from LEGO sell platforms.

This project will focus on the warehouse part of the system. The aim is to simplify warehouse management. The system will give a digital overview of the structure of the warehouse together with the inventory. Furthermore, this product will make the picking process easier and more efficient. It will show all the order information on the same platform and will show the user how and where to pick it from the warehouse.

5.2 User Interfaces

In this segment the system will be presented from a system user perspective. First, all the system users are discussed followed by the requirements for given components of the system.

5.2.1 Users

For this project, we have four main users of the system, namely an administrator, a warehouse manager, an order manager and a warehouse worker which would be referred to as ‘user’.

In table 5.1 an overview of the privileges per user is given.

Users	System permissions
Administrator	Has all the functionalities that are possible Can create new accounts and assign roles to those
Warehouse manager	Can change the warehouse structure (add / delete / change location) Can change the location of an item (add / delete / change location) Can solve a warehouse issue
Order manager	Can indicate whether an order can be picked Can solve order issues
User	Select an order to pick from the warehouse (max two users per order) Select items to restock the warehouse Can report an issue

Table 5.1: Overview users and their system permissions

5.2.2 User Stories

The user stories given below, are formulated as use cases. These use cases and their descriptions, together with the diagram can be found in section 5.2.3.

Problem Report

1. As a user, I want to report an issue via a text message in the application on my mobile device when the actual stock in the warehouse does not correspond to the data given by the application.
2. As a warehouse and order manager, I want to see all the warehouse (respectively item) issues reported by the user on one page in the application.
3. As a warehouse or order manager, I want to be able to indicate via a button that I have handled the issue.

Build a Warehouse

1. As a warehouse manager, I want to create a warehouse in the system, by adding components of the real-life warehouse to the system.
2. As a warehouse manager, I want to easily change the set-up of the warehouse without having to perform many actions. Preferably with a drag and drop action for alterations and an add and delete button for adding and deleting warehouse components.
3. As a warehouse manager, I want to see an overview of all the items within the warehouse on one page.
4. As a warehouse manager, I want to be able to search through and filter all items of the warehouse by the use of typing characteristics of an item in the application.
5. As a warehouse manager, I want to be able to change the quantity of an item in the warehouse by selecting the old value and type in the new correct value.
6. As a warehouse manager, I want to see an overview of the warehouse, with all locations and items that can be found there in a tree view.

Order Picking

1. As an order manager, I want to be able to determine which order can be picked by changing the status.
2. As an order manager, I want to see which orders have already been claimed by users.
3. As an order manager, I want to see the progress of the orders.
4. As a user, I want to see the process of each order and know which order I can pick.
5. As a user, I want to decide which order I pick.
6. As a user, I want to know the location in the warehouse of an item that I have to pick.

Refill

1. As a user, I want to see which items need to be restocked and where in the warehouse by the application.
2. As a user, I want to be able to claim a box with items in the application to store in the warehouse.
3. As an order manager, I want to be able to cancel orders and have the items put back in the warehouse by the users.

Roles and Rights

1. As an administrator, I want to be able to create new accounts for employees and set their roles in the system.

Logging

1. As a warehouse manager, I want to see a log of the changes to the warehouse in the application.
2. As a warehouse manager, I want to be able to download the current stock of the warehouse as a BSX file via a button and have the possibility to import it manually again in the future.

5.2.3 Use Cases

Below the use case diagram is given (Figure 5.1) with underneath a detailed description of each use case (table 5.2).

5.3 System Interface

As mentioned before, this project is based on Kulla. Since there were time constraints on the project, the team could not redo their work and had to build on Kulla, although this team did not agree with some of Kulla's design choices. The use of this structure and its alterations are discussed in Chapter 6, High Level Design.
Furthermore, the system requirements for the application are given below. They are written according to the MoSCoW principle.

Must

- The system must have an inventory overview with all the items within the warehouse.
- The warehouse must be represented as a tree view.
- The tree-view must support drag and drop functionalities.
- The system must adhere to the KISS-principle [14].
- The system must be able to convert the warehouse inventory to a BSX file.

Should

- The system should warn the warehouse manager whenever his action leads to an alteration in the warehouse, to prevent changing the databases unintended.
- The system should save and indicate to the warehouse manager which items are reserved, to prevent the sale of sold items.
- The system should give the shortest route through the warehouse to the user that is order picking for collecting the items, to make order picking more efficient.
- The system should track the time whenever a user is order picking or restocking.

Could

- The system could have an option to copy a part of the warehouse in the tree view, to make it easier to build a warehouse.
- The system could give a suggestion to the warehouse manager to move specific items to another box whenever the original box becomes too full.
- The system could give a suggestion to the warehouse manager to move specific boxes to another location in the warehouse, so that items that occur often together in orders are close to each other.
- The system could give a suggestion to the warehouse manager to check whether the actual stock of certain items corresponds to the administration in the database (called balancing), when they are known to not have been checked lately.

6 HIGH LEVEL DESIGN

In this chapter, the overall design of the system will be discussed. In other words, the system's architecture pattern together with a more specific explanation of the structure of Kulla with a comparison to KWaSA. Additionally, an overview of the system's pages is given with the overall functionality.

6.1 Software Architecture Pattern

With regard to the software architecture, the decision for a 3-tier architecture pattern has been made. This type of architecture is a specific type of a client-server system and is composed into three tiers of logical computing, which are often a presentation tier, application tier and data tier. By introducing an additional tier between the interface and data storage, possible time-outs of the software are less of a problem, because the additional tier can be deployed over multiple systems [15]. Furthermore, such a division within the logic is more flexible for a development team, because a specific part of the application can be updated independently of the rest. This is also part of the reason why such an architecture is often used in applications that automate business processes made by undergraduate software engineering students [16].

The three tiers are separated in code and they are described one by one in the following sections. See Figure 6.1 below for an overview of the tiers, their enabling technologies and the corresponding protocols.

6.1.1 Front-end: Presentation Tier

The presentation tier is the front-end which consists of the user interface. This is graphical content that is accessible through a web browser. It is built with the web development framework Vue.js, which uses HTML, JavaScript and CSS. This tier communicates with the application tier with API calls using HTTP and RESTful services (Flask) in particular. This protocol was already in Kulla, so for the sake of continuation and consistency KWaSA adopted this protocol.

6.1.2 Back-end: Application Tier

The application tier contains the functional logic that drives the application's core. It can communicate with the database by SQL connections and with the front-end by API calls. This is written in Python.

6.1.3 Database: Data Tier

The data tier contains the functionality to access the data of the database. This code is written in SQL and this tier can communicate with the back-end using SQL connections. See below for the database schema. As for the database management system, MySQL is used.

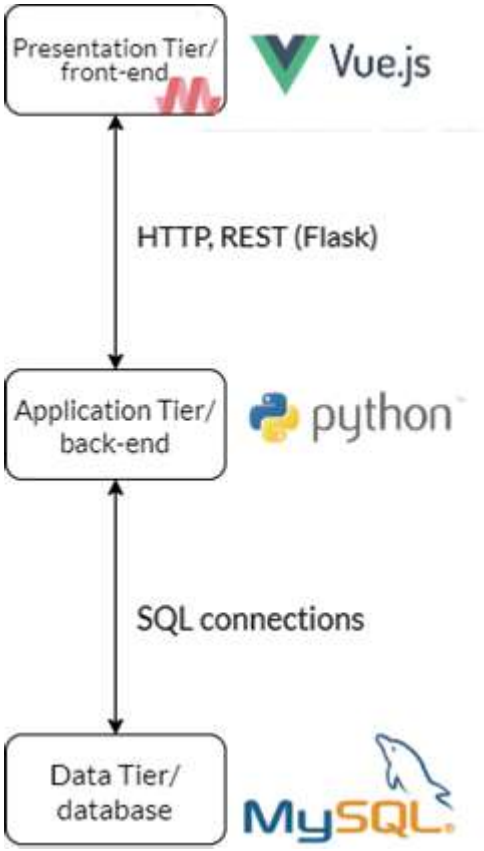

Figure 6.1: The architecture pattern of KWaSA

6.2 Global Design Choices

In this section, an elaboration is given on the major global design choices and how these choices were reflected in the final product. Since the wish of Unbrickable is to build further on the software that will be provided at the end, the system is designed with regard to future enhancements. This is accomplished by good and clear documentation together with automated testing.

6.2.1 Icons

As mentioned before, Unbrickable works with teenagers with a challenge. An example could be autism. In an evaluation of human-computer interaction with people on the autism spectrum a recommendation was made to use icons in the user interface [6]. In this project a combination of icons and text is used since the application is used by employees with and without challenges. Also by doing it this way, if an icon is not clear for the user, the text will give a clarification. An example of this is shown in image 6.2.

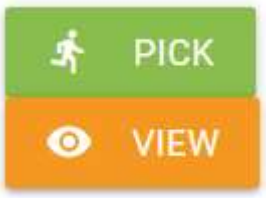

Figure 6.2: Example buttons with icons

6.2.2 Navigation

The website navigation was mostly determined by Kulla. Given that these design choices are compliant with some guidelines, this version is built further onto this with multiple extensions [17].
For the sake of accessibility, users can navigate to a certain page of the website via different paths. This is important to help the system user orient themselves and let them navigate effectively [18].
Furthermore, in some cases where more information is shown, this is done via modals. This will limit the navigation and potential confusion for the user. This confusion can be caused when navigating to different pages when only wanting to see a little part of extra information or when wanting to give data input.

6.2.3 Security

The application is supposed to be used on a daily basis in the real world and it will include a large database, so it is important that the application is secure. It is good to note the application will be used by employees, who are not malicious. Considering this, the main threats are invalid data or behaviour by the users and interaction with the system before logging in. Undoubtedly, there are other security aspects to take into account, considering security is such a broad topic. However, due to time constraints, there was no possibility of diving into more.

With regards to invalid data or behaviour of the users, the security risk is that the application will break. To prevent this, the application is tested (especially usability tested) and such events should not occur. With regards to interaction with the system before logging in, this is the part of the application that is accessible to people outside of the organisation. Therefore, it is important to make this part secure to prevent exploitation of the software and giving information to other people. Therefore, every newly implemented input field has been sanitized to make it more secure and to prevent for example SQL injection.

6.3 System Overview

This section presents an overview of the system based on the system requirements as discussed in Chapter 5. The different components with their functionalities are identified and described. Together, they form the application and serve to aid the user experience. A detailed design of each component will be given in Chapter 7.

6.3.1 Warehouse

In this section of the system there is a warehouse overview page. Here, the structure of the entire warehouse is shown in a tree structured way, as well as the items present at each storage location. The manager can construct a digital version of the warehouse together with adding items to specific locations. When the warehouse has a physical change, this can also be done within the digital warehouse.

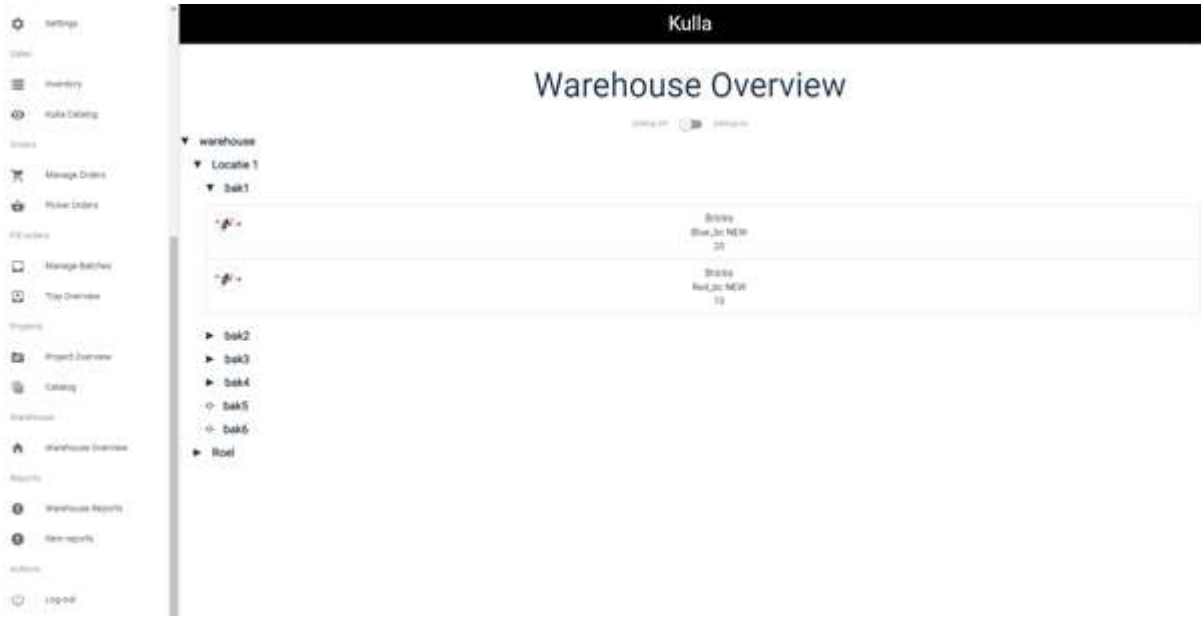

Figure 6.3: Warehouse page

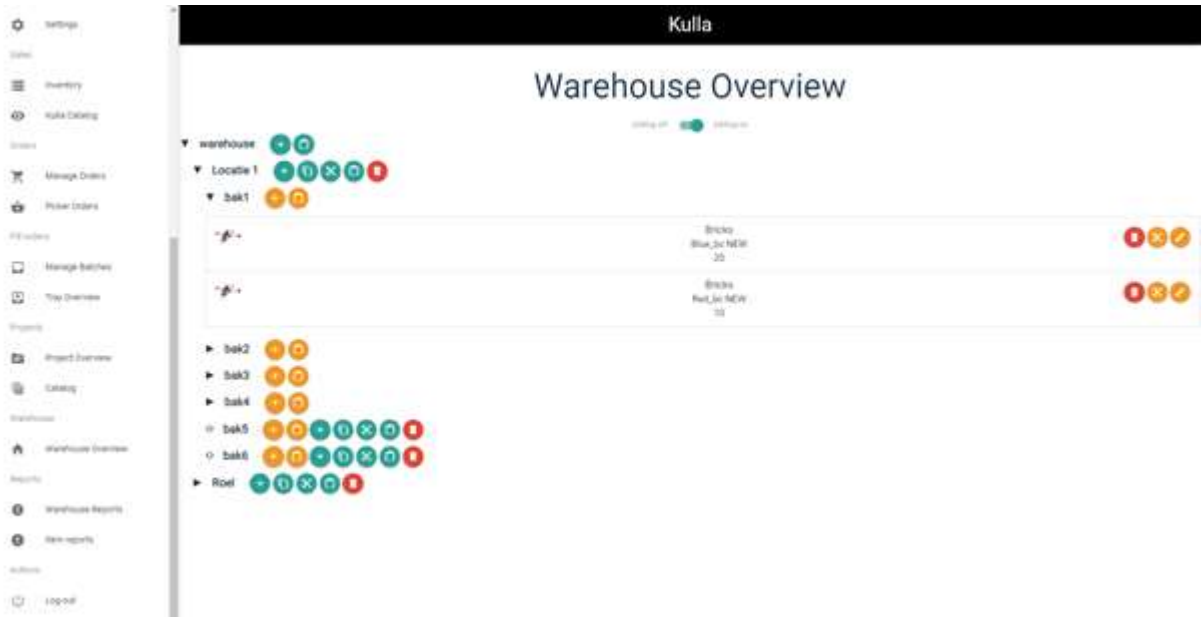

Figure 6.4: Warehouse page with edit mode activated

6.3.2 Orders

In this section of the website there are two pages, namely Orders and Picker Orders. The page Orders is only visible for the order manager. Here they can see all incoming orders with their status and any other important information. They also have the option to see the overview of an order. When there is a problem with an order, the manager can remove it temporarily from the Picker Orders.

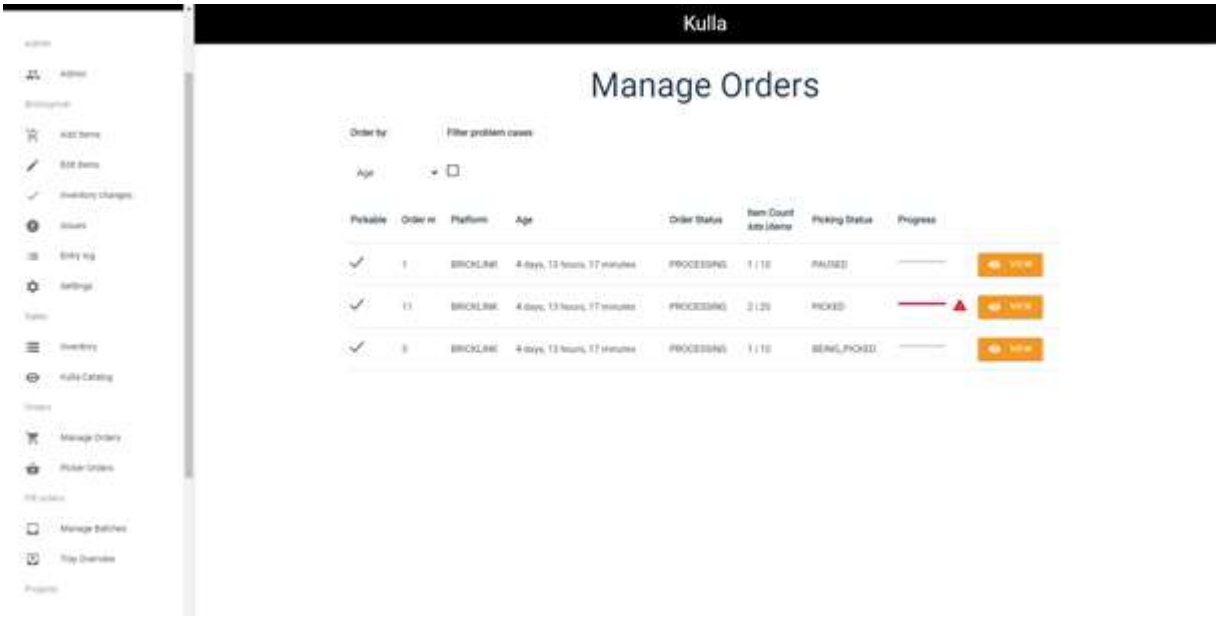

Figure 6.5: Page with an overview of all orders

On the page of Picker Orders the users can see the orders that need to be picked with the necessary information. For each order the user has the option to pick an order or to see the overview of the order.

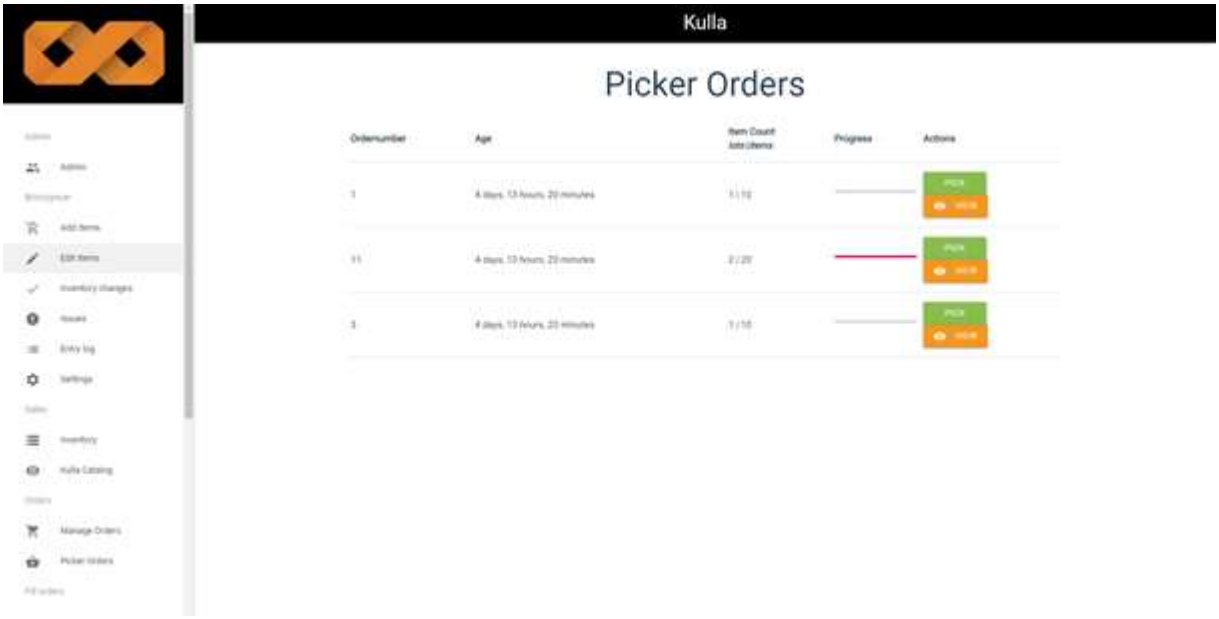

Figure 6.6: Page with an overview of orders for the pickers

Picking

When a user chooses to pick an order, they are redirected to the picker page. This is designed for mobile users, but could also be used on a computer. The page shows the user where to go in the warehouse and which item should be picked. Furthermore, there is a possibility to report a problem for the warehouse manager, order manager or both. Since order picking is an activity where movement is involved, a progressive web app is desirable.

Figure 6.7: Mobile view of picking process

Order Overview

In the case that an employee wants more information about an order, they can visit the order overview page. This can be found when selecting the 'view' button of an order. Here, all the information is given together with information about each ordered item. The employee could also start picking from this page, given that they have the right permissions.

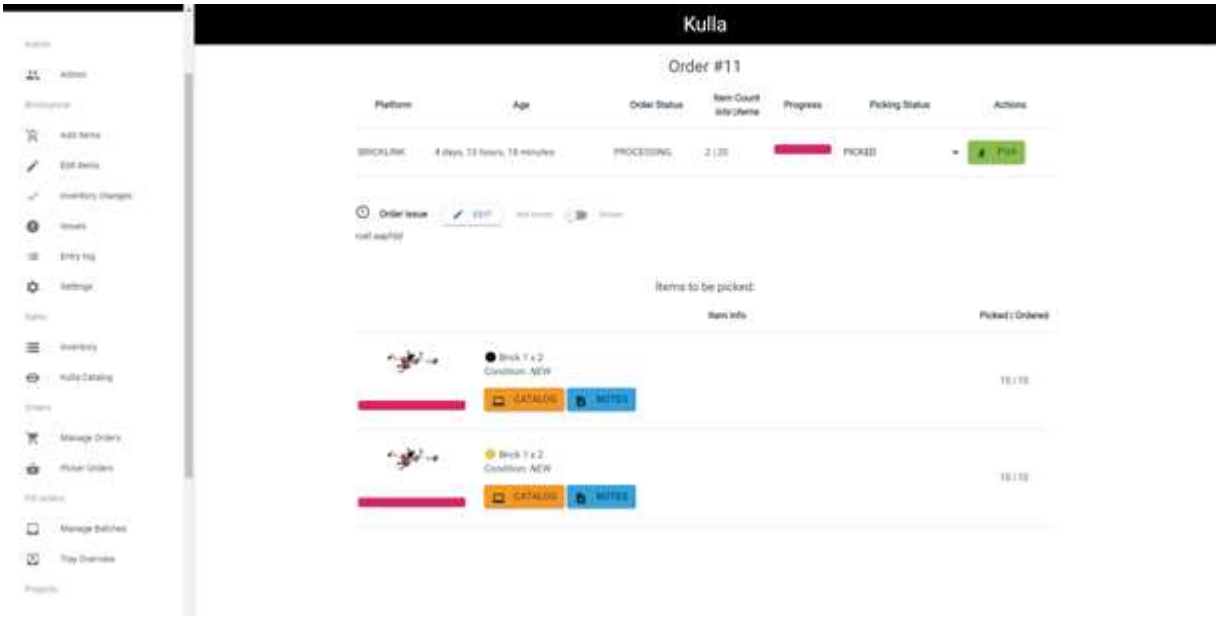

Figure 6.8: Overview of a single order

6.3.3 Fill Orders

This part of the web application is used for storing all LEGO items in the warehouse that are received from DigiBrick. DigiBrick uses a system with projects, consisting of several batches, which in turn consist of several trays. Trays are numbered and only contain one kind of item. For KWaSA, mainly the trays are important, as these can be stored directly into the warehouse, but also an overview of the batches is needed.

Batches

The batches page gives an overview of all batches that are ready to be stored or already (partially) stored in the warehouse. From here, the trays contained in a batch can be accessed, or the batch can immediately be stored in a location, instead of storing each tray one by one. This is especially useful during the initialization of the warehouse.

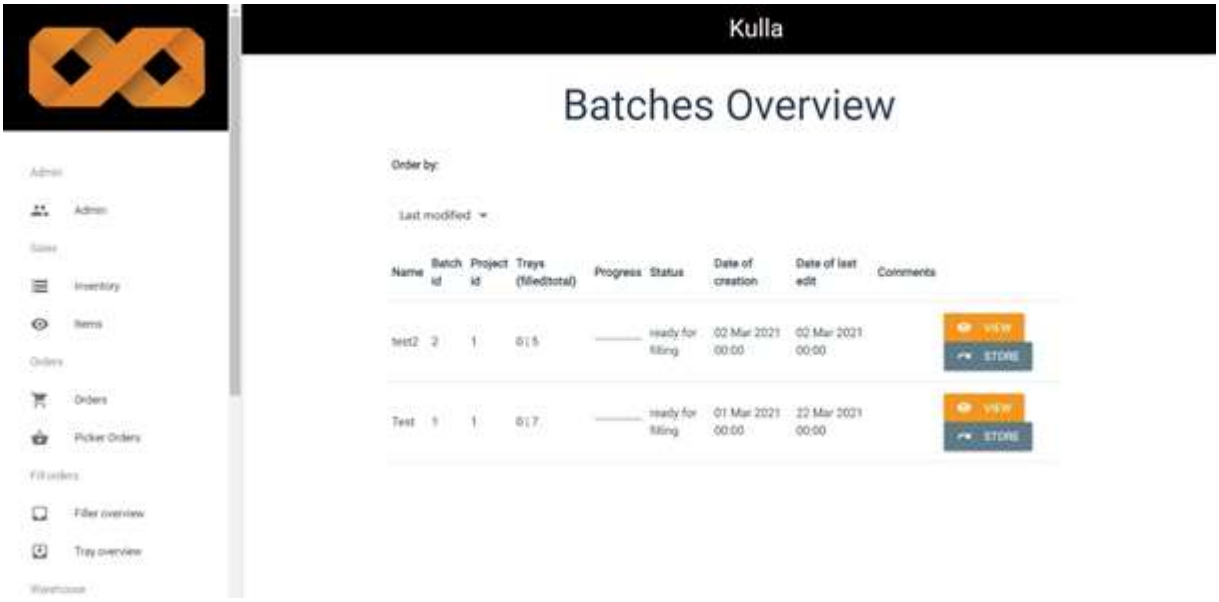

Figure 6.9: Overview of all the batches for the warehouse

Trays

The trays page shows all trays that are ready to be stored in the warehouse. A user can use this page to pick a tray to store, and navigate to the relevant filler page.

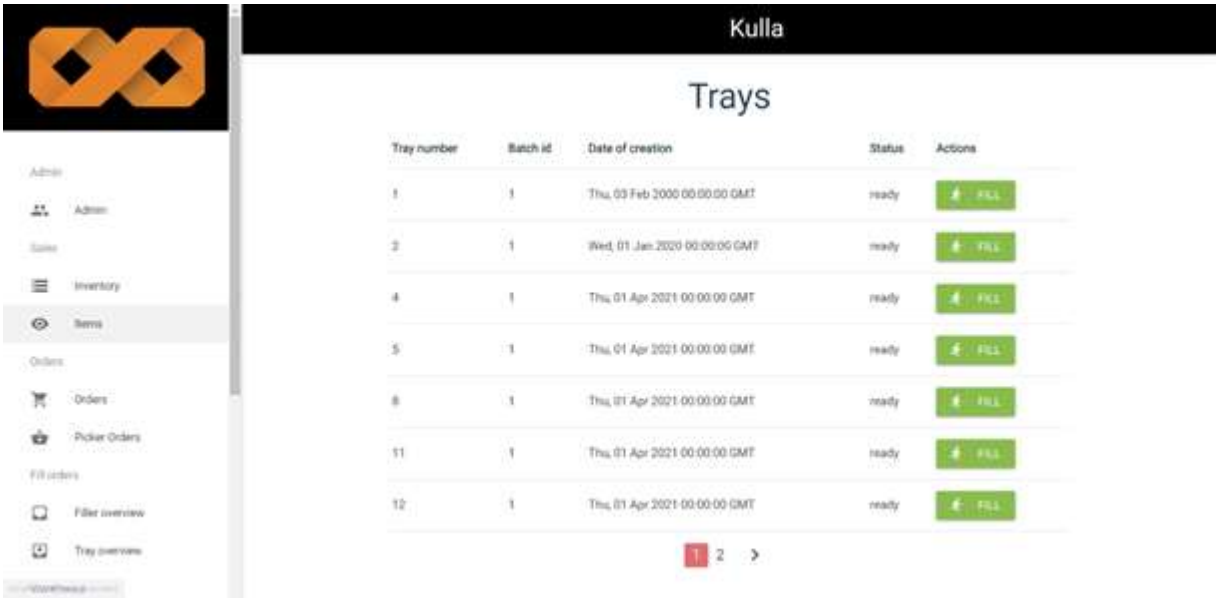

Figure 6.10: Overview of all the to-be-filled trays

Filler

The filler page, accessible from the Trays page, shows information about a single tray, and provides the user with the ability to choose a location from the provided options and store the items contained in the tray in the selected location.

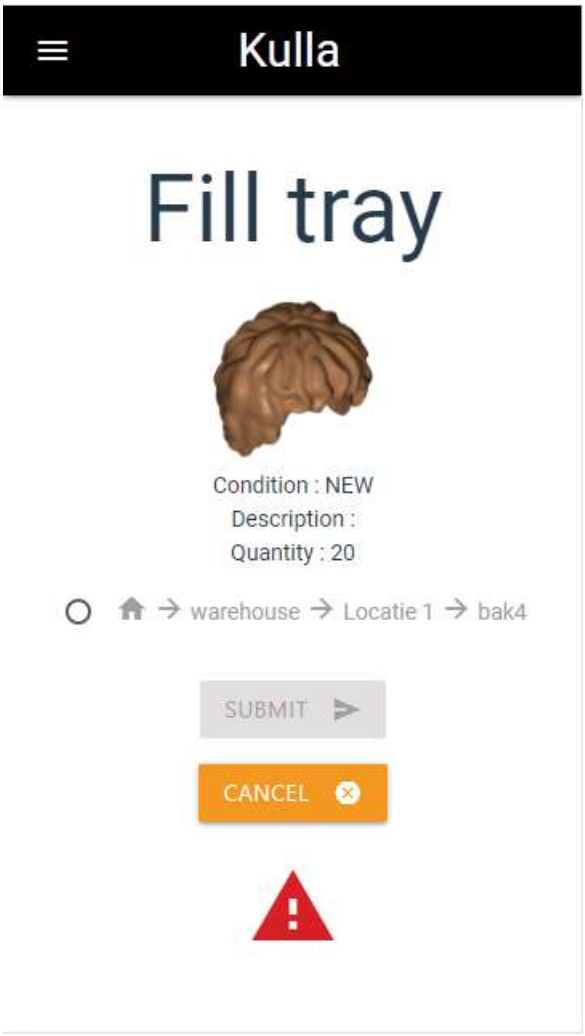

Figure 6.11: Mobile view of refill process

6.3.4 Reports

When an employee runs into a problem, they can report that. The manager can view an overview of all the reports on the Reports page. The manager can filter the problems on whether they are solved or not. Solved problems can be deleted.

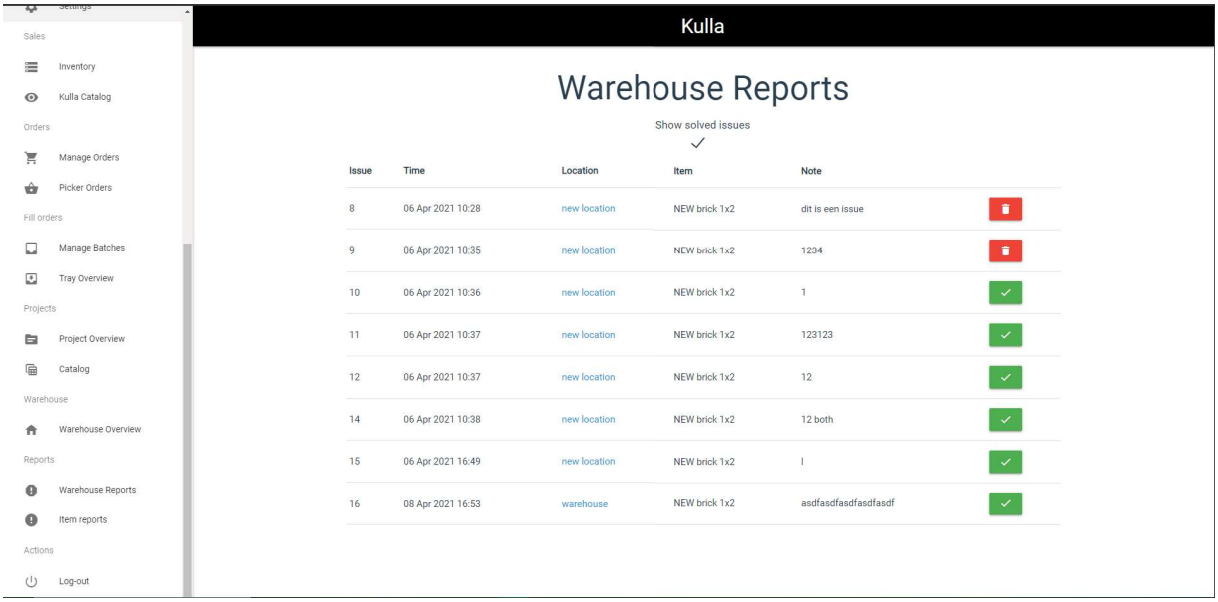

Figure 6.12: Report overview of the warehouse manager

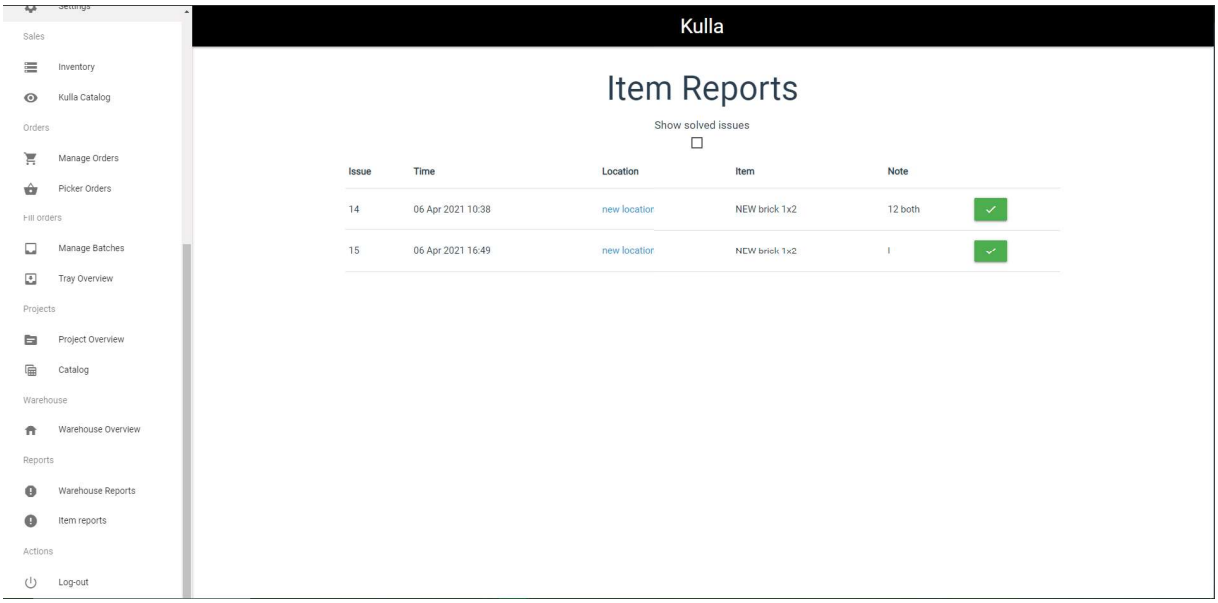

Figure 6.13: Report overview of the order manager

6.3.5 Other

Furthermore, the design of the application takes the roles and rights of employees into account. The different employees were already implemented in Kulla, namely: administrator, order manager, order picker and warehouse manager. When implementing the functionalities, we assigned the rights to use the functionality according to the principle of least privilege: each user only has the minimum privileges necessary to perform their tasks.
Finally, all changes to the warehouse are logged in the back-end in a text file. This makes it possible for the warehouse manager to review the changes and look at the history of the warehouse. This can be really valuable for the manager when mistakes occur.

7 DETAILED DESIGN

In this chapter, an overview of the functionalities is given with an elaboration of the technical aspects of the project. Also, an explanation is given on how the system satisfies the user requirements.

7.1 Representation of Locations

A traversal tree is used for representing the locations of the warehouse. Each location has an unique location id and a foreign key to the parent. This design choice was made by Kulla. The content table indicates which LEGO item is linked to which location, with a particular quantity. A location can have zero or multiple Lego items and each LEGO item can have multiple locations. It is possible to create an issue if something is wrong with the location or item. The issue is stored in the issue table. Figure 7.1 shows the relevant database schema.

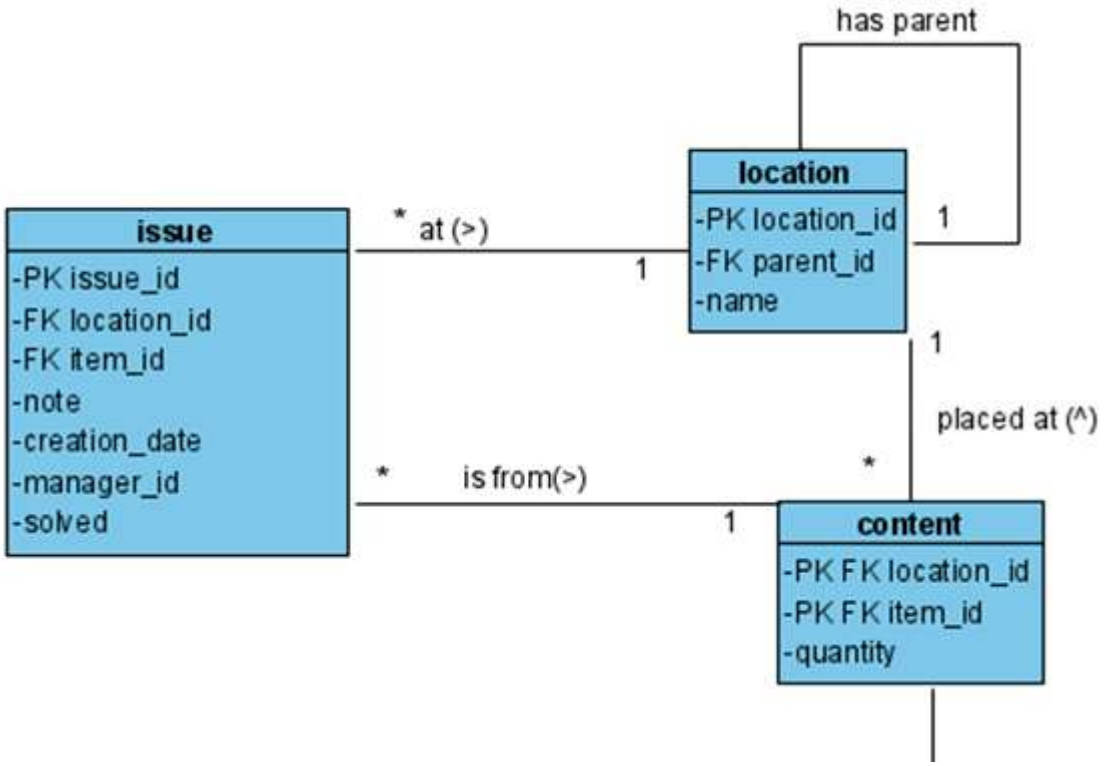

Figure 7.1: Location data storage

8 TESTING

This chapter describes all tests. First, the Test Plan discusses all tests and their objectives. Then, the results of these tests are provided. The results have been used to adapt the corresponding functions such that they work properly.

8.1 Test Plan

The three main goals for our tests were testing the functionality and correctness of KWaSA, the usability of KWaSA (as experienced by the client) and the collaboration of KWaSA with the other applications. After all, any interaction with KWaSA must be handled accurately. Considering these three goals, the test approach consisted of performing unit tests, integration tests, functional tests and usability tests.

8.1.1 Unit Testing

Unit testing is a type of software testing where individual components are tested. The test checks whether the output of each component is as expected.

8.1.2 API Testing

The API calls of the back-end were tested with black-box testing using Postman. For each request the test checks whether the actual response resembles the expected response. This will be done using Postman and the database will be reset after every test.

8.1.3 Integration Testing

Integration testing is testing individual components of the software as a group and is meant to expose mistakes in the interaction between the components. These tests have been executed whenever a new page or functionality was added and after the unit tests of the individual components were passed. Mainly, the connection between the components was tested, such as whether the database is actually updated after a certain event at the front-end. In addition, the tests were also used to check for concurrency problems. After all, the application will be used by multiple people at a time and any new functionality must support this. The front-end will simply be tested by trying out all implemented features and check whether they work as intended. The latter has also been verified by Unbrickable in the weekly meetings during the demo.

8.1.4 System Testing

System testing is testing a complete integrated system to evaluate the systems compliance with the requirements. For this project the whole system consists of the following projects: KwaSa, DigiBrick and BrickSync.

8.1.5 Usability Testing

Usability testing is testing the system by letting users interact with the application. This form of testing is very useful for testing the design of the application, as it shows how intuitive and easy to use the application is, which is obviously an important quality the system should have. A prototype of KWaSA was given to Unbrickable so that they could try the application already and give feedback. Due to time constraints and COVID-19, it was not possible to do a more elaborate usability test unfortunately.

8.2 Results

8.2.1 Unit Testing

Some research was done to investigate the possibilities of automatically unit testing. Unfortunately, all the interesting and complex logic code of the back-end consisted of various sql queries. In order to test these Python functions, a working test database was necessary. It is possible to provide the Flask server a memory database, which can be filled with SQL scripts. A disadvantage of this approach is that it takes a lot of time to implement. Also the client wanted to focus more on system and usability tests than on small parts of the system.
Due to time constraints and the vision of the client, the decision was made to not implement automatic unit testing.
Testing the Javascript functions in the front-end is possible with Unit testing. Due to time constraints and the fact that the functions in the front-end were not complicated enough, the decision was made to not add Unit tests in the front-end. Because many Javascript functions in the front-end consist of simple if, else statements. The idea was to provide the front-end with as much data as possible, so that all the values of the views could be set directly with the received JSON. Although automatically unit testing was not implemented. Each function created was extensively tested. For developing, every team member had their own local database, which could be adjusted to test the outcome of a particular function.

8.2.2 API Testing

Before a new API call is added in the front-end, this API is tested with Postman. Various scenarios were tested. For instance, testing the Api call to fetch all the children of a particular location is tested with the following scenarios:

- a situation where a location has no sub-locations
- a situation where a location has multiple sub-locations
- situation where a location has one or more sub-locations with multiple sub-locations
- etc.

When an API call satisfies the desired behaviour it is added to the front-end. At the time when a function showed strange behaviour, the testing was done for the used API calls in this function. This was to confirm the correctness of the underlying API calls of this function.

8.2.3 Integration Testing

The integration tests were helpful and helped find technical issues that could be solved before the functional or usability tests. Because the tree browser provides a simple visualization of the warehouse, it was helpful to test this component with for instance the filling process. The filling screen has an option to select multiple locations if an item is not coupled to a location

already. When a user has selected a location and pressed the fill button, the tree browser was referenced in order to verify if this process has the right outcome. The tree browser was also tested with the order picking process. The order picking and issues were tested together.

8.2.4 System Testing

The system tests were performed after merging with the other two Unbrickable groups. System testing was mostly performed by the client. They provided a valuable summary of the bugs they have found in the system. When these bugs were fixed, it was communicated to the client. Following this, the client performed a new system test to confirm that the issue is solved. One important issue that appeared is that finished batches were not visible in the screens of Bricksyncer. Due to miscommunication between the project groups, different statuses were used for indicating if a batch is completed and filled. The changes that were made in the database affected the KWaSA project more than the other two systems. Some parts of the system needed to be reimplemented.One other issue that occurred was that original in Kulla, items are integers., BrickStock converted this item id to a string. Therefore many existing and new API calls needed to be (re)implemented.

8.2.5 Usability Testing

The client also tested the software with their employees in order to test the usability of the product, they gave valuable feedback of the system. Including that the employees found the most screens self-explanatory, except for the tree browser. This outcome confirms one of the requirements of the client, namely KISS. To improve the user experience in the tree browser, a tool-tip section can be added to that page. This tool-tip section will describe what the actions of the buttons are and the meaning of the colours. Due to time constraints and the vision of the client, this improvement will be done in further development.

9 DOCUMENTATION

To keep the code maintainable for the client, the code should be understandable. To ensure this, documentation was added to the code where it was deemed necessary.

9.1 Front-end

The front-end features minimal documentation. The reason for this is that most code is already quite clear and straightforward. Variables and functions are named appropriately, and this already causes the code to be much more readable. Where deemed necessary, some comments were provided, but overall this was kept to a minimum to keep the code clean.

9.2 Back-end

The most vital part to document was the API: the communication between the front-end and the back-end. Every HTTP request that the application uses is fully documented, including the purpose of the request, the arguments and the body the request needed (if relevant), and what the server returns, including the status code and the response body in JSON format. This documentation is included in the project as separate files of Markdown Documentation format. In this way, future developers can more easily understand the different API requests and their functions.
Also, some documentation is provided for functions that interact with the database. For these functions, documentation is provided in the code itself. This includes a short description of what the function does, information about the arguments the function needs, and what the function returns.

11 EVALUATION

This chapter will evaluate the process of the development of the system together with the final result. There is also a revision of the proposed system. Here, an explanation is given on how the system fulfills features with their requirements. And if they are not implemented, there is an explanation on how the project does not meet the requirements and why it is not realized.

11.1 Project Organization

The decision to work Agile was very fitting for this project. During the development process, the client often changed their requirements. This was a result of the weekly meetings (see Appendix A for the summaries of the weekly meetings with the client). Thanks to SCRUM, the development team was flexible enough to cope with the changes. Furthermore, the sprint retrospectives worked well as there were many opportunities to give feedback on each other and the workflow. We had a retrospective after each sprint and we used Trello for this. We had three lists on the board: things we should start, stop and continue doing. Everyone could add cards to the lists and then we discussed them afterwards. See Appendix B for the summaries of the retrospectives.

It was also nice to have a lot of structure during this project. Assigning roles to each member and setting up an agenda for each meeting contributed to this structured project. Every team member had the same mindset about the workload. The team decided to work each workday from 9 AM until 5 PM with as maximum 6 PM. Team members were also flexible towards each other since Angela and Pieter had an extra course they were working on and Kevin was occasionally busy with his part-time job.

11.2 Planning

To distribute the workload well and make sure all important features were done first, a planning was made and incorporated in the project proposal. This had been approved by our supervisor. At the time it was difficult to foresee how much time certain parts of the implementation would take. As a result, we made some changes to the planning halfway through the project (see Appendix C for a short version of the final planning). The tasks and the deadlines were incorporated into the SCRUM-board, which was really helpful in following the planning.

With respect to the deadlines of the deliverables, we put internal deadlines into place that ensured the deliverables were done on time and there was still time to ask for feedback from the client or supervisor. This turned out to be really helpful and contributed to the quality of the deliverables.

11.2.1 Risk Analysis

With respect to the anticipated risks to the project as discussed in Section 4.5, we can conclude that luckily most threats did not occur. No project members had to take a break from the project due to sickness and no one quit the project. With respect to time consumption issues, it is true that many group members had other time consuming activities such as other courses or part-time jobs. However, these did not turn out to take more time than anticipated. Especially considering this project is not supposed to be a full-time job (it is worth 10 EC), this did not affect the project negatively. Additionally, time consumption issues due to wrong estimation of tasks were taken into account by adjusting the planning on time as discussed previously. Furthermore, the communication with the other two project groups did turn out to be cumbersome. Especially in the end, it turned out that both other groups made design choices that impacted the collaboration between the projects and therefore it also had an impact on our project. These choices were not discussed beforehand, so this did require some additional time and work on our side to make our project compliant with the other two. We would have preferred it if they had indicated this earlier. Finally, it is good to mention that the online communication did not cause a lot of issues. Even though we definitely would have preferred it to do this project outside a pandemic, the communication channels we had were more than sufficient to collaborate and communicate well.

Unfortunately, some issues arose that were not taken into account during the risk analysis at the start of the project. At some point during implementation, we tried out Bootstrap for some parts of the interface of the application [24]. However, it did not work as intended. After investigation, it turned out that Bootstrap clashed with Materialize and they could not be used together. This was not anticipated as a risk beforehand, because it was unclear to us that Materialize was used in Kulla. Luckily, we found the cause of the problem and were able to solve all issues.

11.3 Requirements

In Chapter 4 the proposed system is explained and the priorities of all features are given. In Chapter 6 and 7, the design is discussed on both a high level and in detail. Not all possible features are implemented. This section will give an evaluation on the features that are implemented according to the priorities (see Section 4.2).

All features are listed in Table 11.1 with their MoSCoW priority and it is indicated whether they are implemented or not. An explanation for each feature that is not implemented is given below.

There are a few features that have been implemented differently, namely:

1. Warehouse tree-view: drag and drop
   The aim of this feature is that the user is able to move items from one subgroup to another and drag and drop was one way this could be implemented. It turned out to be technically difficult, so this requirement has been fulfilled with a cutting and pasting functionality instead. This also fulfilled the wishes of the client.

2. Order-picking: shortest path
   This feature is described as the feature that should give the shortest route to the user that is order picking. This has been implemented in a way, but it does not give the physically shortest route. So, the user will not be led to a location they have already visited, but the application does not know whether there is another route that is for example a few meters shorter. After all, there are no physical dimensions known to the application. Unbrickable is content with this.

Feature	Must	Should	Could	Implemented
Warehouse inventory	X			Yes
Warehouse tree-view	X			Yes
Warehouse tree-view: Drag and drop	X			Differently
Warehouse tree-view: Add and remove items	X			Yes
Warehouse tree-view: Add and remove warehouse components	X			Yes
Warehouse tree-view: Change stock	X			Yes
Warehouse tree-view: Alteration warnings		X		Differently
Warehouse tree-view: reserved stock		*X*		Yes
Warehouse tree-view: Recreate warehouse section			X	Yes
Warehouse optimizer: Warehouse issue report	X			Yes
Warehouse optimizer: Box reallocation suggestion			X	No
Warehouse optimizer: Box location optimizer			X	No
Warehouse optimizer: Balancing			X	No
Order picking	X			Yes
Order picking: view	X			Yes
Order picking: cooperation	X			Yes
Order picking: issue input	X			Yes
Order picking: Shortest path		X		Differently
Order picking: Time management		X		No
Restocker	X			Yes
Restocker: Refill	X			Yes
Restocker: Reverse order			X	No
Restocker: Reverse order assistance			X	No
Roles and Rights	X			Yes
Logging	X			Yes
Warehouse download	X			No
Internal order			X	No

Table 11.1: Overview features with indication of priority and implementation

3. Warehouse tree-view: alternation warnings
   The aim of this feature is that unintended database changes can be prevented. Initially, the plan was to warn the warehouse manager for every alteration in the warehouse. However, it was decided with Unbrickable to make a switch to toggle the editing mode. This way, unintentional changes to the warehouse are also prevented, as the manager manually has to switch the mode. This is even better than the initial plan of alternation warnings, because it is inconvenient to receive so many warnings when building the warehouse the first time.

Furthermore, there are some features that have not been implemented due to time constraints. All these features had a low priority (should or could) and therefore they were dropped:

1. Warehouse optimizer: Box reallocation suggestion
2. Warehouse optimizer: Box location optimizer
3. Warehouse optimizer: Box reallocation suggestion
4. Warehouse optimizer: Balancing
5. Order picking: Time management
6. Restocker: Reverse order
7. Restocker: Reverse order assistance
8. Internal order

9. Warehouse download
   This was not implemented, because Unbrickable indicated this was not so important after all, and there was no time for this.

It is good to note that it was never the intention to implement all features. It was anticipated there would not be enough time to do everything

11.4 Contributions

For this project, each team member was assigned a role. These roles were mostly effective for meetings with the client and the supervisor. But of course the scrummaster supported the whole process.

Furthermore, the team worked mostly in pairs. These pairs were often switched up in order to have an efficient workflow. It is difficult to determine who was responsible for which component since the group cooperated a lot. Underneath, most of the responsibilities are noted. The component of the project is only noted when the team member contributed at least 30% of the total work. This does not mean that someone with less responsibility, did less work. The team agrees that they all had about the same amount of workload.

Angela
Role: Scrum master
Description: At the start of the sprint Angela led the sprint planning. Leading the sprint is something that she improved each sprint with the help of the retrospective. Also for each meeting she made the agenda with all the points that needed to be discussed and mostly led the meetings.
Responsibilities:

- Project proposal
- Sprint retrospectives
- Order picking process from a picker point of view
- Make the order picking a concurrent process of two pickers
- Picking issue
- Design report
- Poster

Anissa
Role: Contact person
Description: At the start of the project, Anissa contacted the client and searched for a supervisor. Since this project was a collaboration of a total of three groups, communication was an important part. She made sure that meetings were planned and that there were clear regulations on how the components would communicate with each other. During meetings, she made sure that the requirements of the company were clear.
Responsibilities:

- Project proposal
- Requirements engineering
- Order overview for the managers

- Order overview for the pickers
- Order issue
- Database structure
- Design report
- PowerPoint presentation
- User manual

Kevin
Role: Technical expert
Description: Since Kevin has a lot of development experience, he quickly knew what consequences could be for a certain requirement. Also during meetings with the client, he could answer all of the questions from the client on what was possible and what not. At the start of the project he also made sure that Kulla was runnable for each group. He made a step by step description for them on how to let the system run locally.
Responsibilities:

- Make Kulla runnable
- Project proposal
- Warehouse overview
- Merging back-end and front-end with the other Unbrickable groups
- Make the order picking a concurrent process of two pickers
- Implemented the front-end for the trays and batches overview pages
- Bug fixes

Pieter
Role: Secretary
Description: During meetings, Pieter made notes on every important thing that was discussed. Also when things were not clear, he made sure to ask the right questions in order to clear things up. Furthermore, Pieter was very valuable for this project. As soon as someone had a question, he was ready to help. He helped with a lot of components that the other members were responsible for.
Responsibilities:

- Project proposal
- Order issue
- Picking issue
- Issue overview page
- API calls
- Implemented the backend for the trays and batches overview pages
- Documentation

11.5 Kulla and KWaSA

As already indicated in Section 2.3, Unbrickable was not able to work with Kulla. The existing software was built for the development environment. This environment is not the same as the production environment. The client indicated that the problem with Kulla layed there. Another problem that occured, is that the three components had to work together in their operating environment.

The solution to this problem was creating a DevOps environment [1]. This principle was new for this project team. However, with help from Daniël Huisman, a member of the BrickSync project, a DevOps environment was created. This process was very time consuming and not something that other design projects are asked to do. More about this and other extra processes in the next section.

11.6 Project Scope

With regards to the scope of the project, there were circumstances that made the design and implementation of the application more difficult, while they were outside the scope of a regular design project. First, we had to continue on Kulla, which forced us to use certain design choices. This made designing KWaSA more difficult, since there were more arguments to take into account. Secondly, cooperation with the other project groups was very important for the project. After all, all three applications had to be designed to be merged and used together. Therefore, we had to synchronize the designs and it was difficult in the beginning, as it was unclear what everyone would need exactly. The design consisted basically of two parts: the design of KWaSA and the design of the total application including DigiBrick and BrickSync. Finally, we made a DevOps environment to maintain the projects altogether, which was also extra work due to the collaboration with all groups and the wishes of the client. All in all, these made the project more work than anticipated.

12 CONCLUSION

All in all, KWaSA satisfies most wishes of Unbrickable, and so the client is pleased with the result and the project can be seen as successful. Furthermore, we learned a lot during the project, both technical and organizational. None of us were familiar with web development before, so this is a new skill for all team members. In addition, designing and developing a three-tier application was also challenging, considering we had to combine multiple skills. With regard to the organization, we look back on a good collaboration, division of tasks and planning between the team members.

However, there are some things that we would do differently in the future considering the collaboration between the teams. Near the end of the project, it became clear that all three teams had made design choices that would create conflicts at merging, such that we had to change some things such as names and references to the database. These changes did improve the overall design, but it would have been less work if these changes were communicated earlier. Therefore, we would schedule weekly meetings with the other groups as well if we were to do the project again. Furthermore, if we were to have more time for the project, we would implement the features as discussed in Section 10.2.

One thing we could have done differently, is the examination of the current API calls. Since those were not documented in detail, we did not have the overview of all the work that has been done. Resulting in sometimes double and dead code in the project. Dead code is code that is nowhere used in the project. Maybe it was better to start the project to have a better look at the API calls to determine which we could use and which we could regard.

Especially considering the circumstances that made the project more work than anticipated (Kulla, collaboration and DevOps), we look back on a good project that meets its goal, because it is a warehouse storage application that simplifies the filling and management of the storage warehouse and the process of order picking.

REFERENCES

[1] *What is DevOps?* https://www.rackspace.com/library/what-is-devops

[2] IEEE Recommended Practice for Software Requirements Specifications. In: *IEEE Std 830-1998* (1998), S. 1–40. http://dx.doi.org/10.1109/IEEESTD.1998.88286. – DOI 10.1109/IEEESTD.1998.88286

[3] *Unbrickable*. https://unbrickable.info/

[4] *De Missie van Unbrickable*. https://unbrickable.info/sociaal/

[5] Loohuis, Joost ; Maas, Jelle ; Van Noort, Nils ; Stapel, Frank: *Kulla Design Report*. April 2020

[6] *2021 best warehouse software*. https://technologyadvice.com/warehouse-management-software/. Version: March 2021

[7] Kent, Beck: *Manifesto for Agile Software Development*. http://agilemanifesto.org/. Version: 2001

[8] https://trello.com/nl

[9] Ma, Qiao: The effectiveness of requirements prioritization techniques for a medium to large number of requirements: a systematic literature review. (2009), 01

[10] https://vuejs.org/

[11] https://materializecss.com/

[12] https://www.python.org/

[13] https://www.mysql.com/

[14] https://www.interaction-design.org/literature/article/kiss-keep-it-simple-stupid-a-design-principle

[15] Marchetti, C. ; Baldoni, R. ; Tucci-Piergiovanni, S. ; Virgillito, A.: Fully distributed three-tier active software replication. In: *IEEE Transactions on Parallel and Distributed Systems* 17 (2006), Nr. 7, S. 633–645. http://dx.doi.org/10.1109/TPDS.2006.89. – DOI 10.1109/TPDS.2006.89

[16] Mitra, Sandeep: Using UML Modeling to Facilitate Three-Tier Architecture Projects in Software Engineering Courses. In: *ACM Trans. Comput. Educ.* 14 (2014), Oktober, Nr. 3. http://dx.doi.org/10.1145/2635831. – DOI 10.1145/2635831

[17] https://brand-experience.ieee.org/guidelines/digital/style-guide/navigation-linking/

[18] https://www.w3.org/WAI/fundamentals/accessibility-principles/standards

[19] ParadeTo: *ParadeTo/vue-tree-list*. https://github.com/ParadeTo/vue-tree-list

[20] Megafetis: *megafetis/vue3-treeselect*. https://github.com/megafetis/vue3-treeselect

[21] Meijer, Gerben ; Boschma, René ; Koopman, Michael ; Weener, Jeroen ; De Boer, Tristan: Project Proposal Magazijn Module. In: *Design Project Technical Computer Science University of Twente* (2017), April

[22] Frontend-Collective: *frontend-collective/react-sortable-tree*. https://github.com/frontend-collective/react-sortable-tree

[23] *Computed Properties and Watchers - Vue.js*. https://vuejs.org/v2/guide/computed.html

[24] https://getbootstrap.com/

A WEEKLY MEETINGS CLIENT

Meeting 1

The first meeting with our client took place on Friday the 5th of February, 2020, in the afternoon, and took place on the platform Microsoft Teams. The participants were all members of the project group, the main client and another person affiliated with the client. The meeting started with all of the people present shortly introducing themselves. Then, the company was introduced, including the general goals and an introduction to the concerning field, namely LEGO in general, and more specifically LEGO items and the platforms the company uses to sell LEGO. As there already was a project last year, which we can build on and use for our own product, the client shortly explained what already exists and is implemented, including the platforms, tools, and languages used in this previous product, as these are relevant for us and our group will most likely also use these. Next, some organizational aspects were discussed, such as where we could find the source code of the previous project, how our group will handle the organization and planning, how often meetings are planned and how to best contact the client. It was also made clear that we would have to work together with two other project groups participating in the design project. These groups were assigned to the same client. The projects of all groups have some shared parts and issues, so cooperation between project groups was also deemed necessary. Finally, the client elaborated on what we as a project group are expected to develop and deliver. All aspects of the product were covered, and a division of tasks between the three project groups was given, marking the boundaries and highlighting areas on which the products of the three groups overlapped, hence for which collaboration and communication is important. Finally, a new meeting was planned to discuss the project proposal.

Meeting 2

The second meeting with our client took place on Wednesday the 10th of February 2020 in the morning, and took place on Microsoft Teams. The participants were all project group members, and the same 2 people representing the client as last time. Next to some small organisational things, the main thing of the meeting was to go over the project proposal, of which a first version was finished and sent to the client the day before. Together, we went over mainly the chapter deliverables, which contained all aspects we believed the client had asked us to implement for this project. The client mostly confirmed these features, and suggested some improvements and changes, and clarified some parts and provided us with more details and their view on it. Finally, they suggested some features that they believed were still missing, so that we could add these to the report. A start was already made during this meeting in prioritising different requirements, but this would have to be continued in a later meeting. The next meeting was planned for Friday the same week.

Meeting 3

The third meeting with our client took place on Friday the 12th of February 2020 in the afternoon, and took place on Microsoft Teams. All members of the project group were present, as well as the two representatives of the client. In the meeting, the client presented some changes they made to how the system would function, which overall made it somewhat easier for us and more clear how to implement the requirements of the client. Before the meeting, we had prepared a list with all the features we believed the client desired, and had asked the client to label these features as a must, a should, a could or a don't. During the meeting, we went over this and discussed the labelling, as well as features the client had added, as they believed they were still missing. This helped us understand what the client found important and less important. Finally, we discussed the use of DevOps, which would ensure the client would be able to more easily continue with the development of the product, after we have finished the project. A new meeting was planned for next week Wednesday.

Meeting 4

The 4th meeting with unbrickable took place on Wednesday the 16th of February, with all group members and the two representatives of the client, on Microsoft Teams. The main goal was to discuss the project proposal again. The client was given the document beforehand, and prepared some comments and feedback, mainly concerning some of the details. We went over these comments, and gave a quick overview of what we think will happen next. We proposed to make the last changes to the proposal based on the feedback, after which we would send it to them before Friday. If the client is then not yet satisfied, they would give feedback again, and discuss this with us on Friday the 18th. If they approve of the proposal, the next meeting would be on the 5th of March, as the week in between is a holiday.

Meeting 5

The 5th meeting with Unbrickable took place on Friday the 18th of February, with 3 group members and the two representatives of the client, on Microsoft Teams. The client had approved the final version of the project proposal and the main goal of this meeting was to discuss it once more, also with relation to the project proposals of the other 2 project groups. We also asked for some information that would help with implementation and the client also gave us relevant information about this. The next meeting was scheduled for the 5th of March, as the week in between is a holiday.

Meeting 6

The 6th meeting with the client was held on Friday the 5th of March in the afternoon, on Microsoft Teams, with all group members and the two representatives of the client. The meeting started by discussing a part of Kulla, namely the statuses. We discussed what should be manually changeable, what should be read-only, and when these statuses should change. Next, we showed our progress with a short demo, which mainly concerned the warehouse tree view, the order overviews for the picker and the manager, and the order picking. Based on the demo, the client provided us with some feedback and suggestions to improve the application. One of the main points was about how issues in the warehouse could be reported and fixed using the application. After some discussion we came to a solution, which classified issues based on who should receive it: the warehouse manager or the order manager. Following the feedback, the communication between our part of the system and the part of a different group was discussed,

and it was clarified how to approach this. Finally, we asked for formal approval of our project proposal, which they would send via email. The next meeting was planned for exactly one week later.

Meeting 7

The 7th meeting with the client Unbrickable took place on the 12th of March, 2021, on the platform Microsoft Teams with all group members and the two client representatives. At the start of the meeting, we received official approval of our project proposal. Next, a design for a part of the application, namely the order picking, was discussed and the client gave some tips and feedback on how to improve the design, such as whether or not and how to display certain properties of items and orders. Not only the design was shown, a short demo of our (partial) implementation was given as well. Also, issues, a topic from the last meeting, was discussed again and elaborated on. After showing the progress on the tree view of the warehouse, a point was discovered which we had not thought about yet: how to deal with new items without a location. After some discussion we decided we would think about the problem in the coming week and write down an appropriate solution. With this, the meeting was finished.

Meeting 8

The 8th meeting with Unbrickable was on the 19th of March, on Microsoft Teams, with all group members and the two representatives of the client. As usual, we started the meeting by showing the progress we made with the product with a short demo, and the client gave us some feedback about the layout and other aspects. Then we came back to the discussion we had last meeting, concerning the initial setup of the warehouse and the storage of new items, that are thus not in the warehouse yet, in locations in the warehouse. The client had prepared a proposal for a solution they believed would work well. We discussed this, and mostly agreed on this solution, although the client gave us the task of writing it down properly, so it was clear for everyone. After this, we discussed shortly how to deal with keeping a history of all items, locations, trays and orders, and the client asked about the deployment of the application through DevOps. After this was taken care of, the time was up, and a new meeting for next week was planned.

Meeting 9

The 9th meeting with Unbrickable was on the 26th of March, on Microsoft Teams, with all group members and the two representatives of the client. As usual, we started the meeting by showing the progress we made on the product with a short demo and the client gave us some feedback about the functionality. They were pleased with the product but mentioned some features that we still need to implement. We also asked them to specify how they would most like a certain feature and with that information we can continue to make it. We then discussed the merging with the other project groups and the testing of our application. The company will do usability testing on the merged version. Finally we shortly discussed the ethics essay, because we wanted to know their view on this. We agreed to send our final essay to them, because they are interested in this. We were done after this and planned a meeting for next week on Thursday April 1st, since Friday is a public holiday.

Meeting 10

The 10th meeting with Unbrickable was on the 1st of April, on Microsoft Teams, with all group members and the two representatives of the client. We did not give a demo as there were

no new functionalities to show or discuss. We did discuss the final implementations that we intended to do and that Unbrickable requested. Then, we discussed the making of DevOps and its future planning. We scheduled that Unbrickable would perform a usability test next week and a next meeting on April 9th.

Meeting 11

The 11th meeting with Unbrickable was on the 9th of April, on Microsoft Teams, with two group members and the two representatives of the client. We showed and discussed the lastly done implementations and spoke about the results of the usability tests. This resulted in some useful feedback and brought little issues to the surface, which are shared with us via GitHub. Furthermore, we discussed the future of KWaSA. Unbrickable is pleased with the application and has the intention to use it. They will appoint some of their own employees to continue on it, which we think is excellent, because the system must be maintained in the future. We scheduled a new meeting next week Thursday, on the 15th of April.

Meeting 12

The 12th and last meeting with Unbrickable was on the 15th of April, on Microsoft Teams, with two group members and the two representatives of the client. We showed and discussed the final implementations. This resulted in the last tasks that will be done to the implementation Furthermore, we discussed their opinion of KWaSA. Unbrickable is very pleased with the application and has the intention to use it. Finally, we explained how the application will be transferred to Unbrickable. We did not schedule a new meeting, since this is the last week of the project.

B SPRINT RETROSPECTIVES

Sprint 1

During the retrospective of the first sprint, everyone agreed it was a good sprint as we achieved what we planned to do and the set-up of communication channels works. However, the communication within the group could be improved by staying focused and direct. Furthermore, communication with other parties such as the other Unbrickable project groups and the client could be improved by making agendas beforehand, then the goal of the meeting is clear to everyone, which makes the meeting more efficient. The goal for the next sprint is therefore to communicate better (inside and outside the group).

Sprint 2

From the retrospective followed that we should start taking our working hours seriously. We have said from the beginning that we would have working days from 9 to 5, but we would regularly work until after 5.30 PM during sprint 2. We want to plan better and more efficiently during the day. This also includes limiting the amount of meetings per day. Some days, we had so many meetings (with the client and the other project groups) that we barely had any time to work on our own. This was annoying and we will stop with this kind of planning. There are also some things we will continue doing, namely: the current workflow with the distribution of the roles and the good communication whenever a group member cannot be present during a part of the day. It happens sometimes that someone has to leave for a certain period of time, but this is always communicated well so that the other team members can anticipate. Everyone agrees this works well and we should continue doing this.

Sprint 3

During the retrospective of sprint 3, we concluded that we had improved from the action points of the last retrospective. We have worked from 9 to 5 almost each day, with no day longer than 5.30 PM. We planned better and worked more efficiently because of this. We also did not plan so many meetings in a day (also because we had less meetings this sprint). We also mentioned in the retrospective that we intend to start making agendas for the weekly meeting with our supervisor. This way our story is more structured and it is likely we forget to mention anything. Furthermore, we intend to start documenting the (major) design choices during implementation. This is something that you can easily forget, but it will be really insightful and useful when writing the design report. We intend to stop getting stuck on one problem, by doing something else or talking about the problem with someone. There is always enough to do and it is beneficial to distract yourself with something else and return to the old problem with a fresh perspective. Furthermore, we intend to continue our way of collaboration and asking for help. We also intend to merge all versions of the program together at the end of the week, to make sure everyone is aware of and works with the latest version. Also, we intend to continue to include a demo in the weekly meeting with Unbrickable. Even when the program is not finished or the feature does

not yet work as intended, it is still useful to show progress and discuss whether everything is alright.

Sprint 4

During the retrospective of sprint 4, we first reflected on the action points from the last sprint. We do make agendas now for the weekly meeting with our supervisor, which is going great and helps the meetings be more structured. Also, we are now documenting the design choices immediately, which is good. As for the new action points, we want to start by making the tasks smaller. Especially in the beginning, the tasks were formulated quite generally and big, because we did not really know the difficulties and subproblems we would run into. This resulted in a Trello board that was not so insightful, so therefore we are subdividing the tasks into smaller tasks from now on. This will make our Trello board more meaningful. Furthermore, we realized we were quite busy with programming and implementing and we should not lose sight of the design theory and ethics part of the project. We want to continue thinking about and working on them, especially now that working on the design report becomes a priority.

Sprint 5

First, we reflected on the action points from the retrospective of sprint 4. In general, the tasks on Trello were smaller and so the board was more insightful. We do want to continue this and keep it as an action point for this sprint so that we remember it. Furthermore, we have looked at the design theory and ethics part of the project, which is good. This sprint will be week 7 of the design project and the end of the project is in sight. We have already had quite a number of retrospectives and feel like we have found a workflow that works for us. More specifically, we want to continue doing things parallel, but work in the same call so you can always ask help from someone. We also want to continue communicating well with the other groups working for Unbrickable, especially now that our code will need to be merged soon.

Sprint 6

During the retrospective of sprint 6, we could acknowledge that the Trello board was full of smaller tasks and the board was more insightful, so we did well at continuing to do this. Furthermore, we have continued to do things in parallel, but communicate well with each other and the other project groups. Again there are not many points we want to stop or start doing. We feel like we are working the way that works best for us. However, we do acknowledge that we sometimes say ‘Yes’ too often when the client asks for something. This is something that we want to do less, especially now that the end of the project is in sight.

Sprint 7

Sprint 7 was the last sprint of the project and during the retrospective we acknowledged that the client still asked for small adjustments and we said yes often. However, we did not see this is as problem, because many of these were the result of the usability tests performed by Unbrickable and would improve our work. In addition, this process is also the consequence of agile working. Furthermore, we were happy with our way of working this sprint.

C OVERVIEW PLANNING

The table below shows an overview of the important events and deadlines for deliverables of the project.

Week	**Sprint**	**Event**
1 (01-02 - 05-02)		First contact client
2 (08-02 - 12-02)	1	Draft project proposal Draft requirement specification
3 (15-02 - 19-02)	2	Draft test plan Project proposal approved Peer review: project proposal and planning
Holiday		
4 (01-03 - 05-03)	3	Started with warehouse tree view, order picking
5 (08-03 - 12-03)	4	*Peer review: requirements and test plan*
6 (15-03 - 19-03)	5	Started with filling
7 (22-03 - 26-03)	6	*Peer review: design and first prototype*
8 (29-03 - 02-04)	7	Finalizing implementing
9 (05-04 - 09-04)		Presentation *Peer review: presentation*
10 (12-04 - 16-04)		Final poster presentation

Table C.1: Overview planning with events and deadlines