Design and Setup of a Demo Facility for Wearables in Logistics

Final Thesis

Submitted by Marius Freyer

In fulfilment of the requirements for the degree
Bachelor of Science in Informatics
To be awarded by the
Fontys Hogeschool Techniek en Logistiek

Venlo, January 7, 2019

Information page

Fontys Hogeschool Techniek en Logistiek
Postbus 141, 5900 AC Venlo

Final Thesis

Name of student	Marius Freyer
Student number	2447800
Course	Business Informatics
Period	September 2018 - February 2019
Company	Fontys Hogeschool Techniek and Logistiek
Address	Tegelseweg 255
Post code / City	5912 BG Venlo
State	The Netherlands
Company coach	Stefan Sobek
Email	s.sobek@fontys.nl
University coach	Thijs Dorssers
Email	t.dorssers@fontys.nl
Examinator:	Pieter van den Hombergh
Non-disclosure agreement:	No
Number of words:	8403

Abstract

Wearables are a rather new technology that can be used to enhance logistics processes. Especially small- and medium-sized companies (SMEs) in the logistics industry often refuse to adopt new technologies due to uncertainty and costs. To overcome this issue the LOGwear project was founded by the Euregio and different other partners. LOGwear aims for providing SMEs with knowledge and first-hand experience with wearables in logistics processes without investing a huge amount of money upfront.

While the LOGwear project already has a so-called ”Knowledge Base” where SMEs can find a lot of information about different wearables and how they can improve logistics processes, first-hand experience is only provided for a small number of companies that were chosen for pilots. Do also give other companies a chance to test wearables in a realistic environment or even their own company, a demo facility is created and described in this report.

This report systematically analyzes the requirements for building this Demo Facility. Based on these requirements a hardware platform for the demo facility is created and research regarding suitable existing software is done. As no suitable software could be found during the research, the design and implementation process of a custom-built software solution is described in the report. This process includes designing an architecture of a NodeJS-based backend that provides a REST API and a frontend based on Angular. The implementation of this architecture using Typescript is then described in the report with focus on the backend.

As the result a hardware and software platform is available for simulating one logistics process (order picking) with wearables. The platform is ready to be extended for future use with more processes and wearables.

Statement of Authorship

I, the undersigned, hereby certify that I have compiled and written the attached document and the underlying work without assistance from anyone except the specifically assigned academic supervisors and examinors. This work is solely my own, and I am solely responsible for the content, organization, and making of this document.

I hereby acknowledge that I have read the instructions for preparation and submission of documents provided by my course/my academic institution, and I understand that this document will not be accepted for evaluation or for the award of academic credits if it is determined that it has not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document and the underlying work will not be accepted for evaluation or for the award of academic credits if it is determined that it embodies plagiarism.

Name:	Marius Freyer
Student number:	2447800
Place / Date:	Venlo, January 7, 2019

Signature: _____

Contents

List of Figures

List of Tables

Abbreviations

CRUD Create, Read, Delete, Remove operations.

MVP Minimum Viable Product.

SME Small and medium-sized enterprises.

SRS Software Requirements Specification.

WMS Warehouse Management System.

Glossary

Angular Angular is a Typescript-based frontend framework.

Container A self-contained unit of an application containing all dependencies in a predefined environment.

Docker A technology for container virtualization.

Express A framework for NodeJS for creating REST APIs.

iMac Pro An all-in-one PC manufactured by Apple featuring workstation-grade hardware.

JSON Is a compact an easy to read data format for exchanging data between applications.

MacBook Pro A Notebook manufactured by Apple.

Minimum Viable Product A minimum working version of a product that is developed.

MongoDB MongoDB is a document-based NoSQL database.

NodeJS A technology based on Google Chrome's JavaScript engine that allows running JavaScript on serverside.

Order Picking A logistics process for making products available for shipping.

Putaway A logistics process of stowing good after they are received.

REST API A REST API is a communication interface for client and server based on HTTP.

Software Requirements Specification A software requirements specification (SRS) describes a (software) system to be developed.

Typescript A programming language that is a super set of JavaScript. It extends JavaScript with classes, static types and other functionality.

Warehouse Management System A software system that allows organizing and controlling operations inside a warehouse.

Wearable An electronic device that can be comfortably worn on the body.

1 Introduction

This section introduces the company the project is done with and describes the assignment.

1.1 The LOGwear project

The project *LOGwear - Using wearables to optimize logistic processes* examines how logistic processes, especially in small- and medium-sized enterprises (SMEs), can be optimized by using wearables and which wearables are suitable for this purpose. The aim is to provide companies with an individual online tool for an initial assessment of the extent to which their own processes can be improved with the help of wearables and to offer support in the implementation of a wearable solution. The LOGwear project is part of the INTERREG programme Germany-Netherlands and is co-financed with approx. 1.1 million Euro by the European Union, the Ministerium für Wirtschaft, Innovation, Digitalisierung und Energie des Landes Nordrhein-Westfalen and the Provincie Limburg.

The overriding goals of LOGwear are to increase cross-border innovative strength in the euregio rhein-maas-nord and to increase product and process innovations in the logistics sector, which is important for this region. The main objective of LOGwear is to enable SMEs to use new technologies (in this case wearables) for the innovative development of their processes. In three work packages different measures are taken for this purpose. These include the development of a knowledge database to support the decision-making process of companies, the definition of a reference architecture to facilitate the implementation of software and hardware, and the execution of pilot projects at companies in the region.

The use of wearables also offers the possibility of overcoming language barriers through the use of multilingual software and thus employing staff across borders. (*Das Project*, 2018)

1.2 Problem Description

The LOGwear project aims for providing SMEs with knowledge about wearables by consulting them via a knowledge base but also giving them real-life experience with wearables in logistics via a demo facility.

Currently only customer-specific demo environments for testing the wearables in different logistics processes exist. A general demo environment for testing different wearables with realistic example processes is not yet available. Such a demo environment is needed as the SMEs shall also be able to experience the advantages and disadvantages of different wearables next to being consulted via the knowledge base the LOGwear project also provides.

1.3 Document structure

The document is divided into seven chapters. Chapter 1 and 2 provide a general overview of the environment and background of the project and define the assignment and methods used for fulfilling the projects goals. Chapter 3 describes the analysis phase of the project where the requirements of the demo facility project are defined. Chapter 4 sums up the research results regarding potentially existing software solutions that could be used for the demo facility instead of building a custom software. The design of the demo facility hardware- and software-wise is discussed in chapter 5. Chapter 6 covers the implementation of the design discussed in chapter 5. Chapter 7 finally concludes the report and reflects the results of the project. It also gives an outlook to the reader on how the project can be improved and extended.

2 Project Description

2.1 Assignment

The assignment of the project is building a demo facility for trying out wearables in different logistics processes. This includes selecting and purchasing the required hardware as well as implementing the software for the interaction between the wearables and the demo facility.

2.2 Project scope

In scope of the project is ordering and building the hardware side of the demo facility and designing and implementing the software side of the demo facility. For the hardware side this includes the workstation, network setup, etc. On software side in scope is the development of a minimum viable product either by using existing software solutions or by building a custom one. This depends on the outcome of the research part of the project. The minimum is to have one process working with one wearable in a predefined environment. If time is left more wearables are preferred to processes.

Not in scope is anything that exceeds this minimum viable product (MVP), like definition of different environments, multiple processes, company-specific environments, etc.

2.3 Project phases and deliverables

The project in general consists of four different parts that need to be worked on. The project begins with a Phase of requirements engineering where requirements for the components and scenarios of the demo facility are specified. The second phase of the project is a research regarding a suitable (Warehouse Management) System that can be used to mock different scenarios and processes for testing the wearables using the demo facility. The third part is implementing or configuring an WMS-like system based on the results of the research. The last phase is the actual implementation of different processes in the system and connecting different wearables.

Identifier	Activity	Deliverable
A	**Requirements Engineering**	**Software Requirements Specification**
A.1	Scenarios	Specification of supported scenarios
A.2	Components	Specification of required components (hardware)
A.3	System	Functional requirements of the software system
B	**WMS selection**	**Make or buy WMS**
B.1	Specifying Criteria	
B.2	Candidate selection	
B.3	Assessing candidates	
C	**WMS Implementation/Configuration**	**Running WMS**
D	**Process/Scenario implementation**	**Interaction between wearables and WMS**
D.1	Defining process in system	
D.2	Connecting wearables	

Table 1: Activities and Deliverables

2.4 Project management

For creating the demo facility, a hybrid approach of waterfall and SCRUM are used. For the first part of the project that is mainly focused on analysis, requirements engineering and research, the waterfall approach is used. This is necessary, as selecting and buying the hardware needed for the demo facility is time critical as it relies on budget provided by the Euregio and other project sponsors.

For the second part of the project the development approach is switched to an agile SCRUM-like approach. This part is mainly focused on designing and implementing the software part of the demo facility.

For planning purposes, for creating the backlog and the sprints, Jira is used as the project management tool. Jira is chosen as the LOGwear project already uses Jira for other projects that are carried out in a SCRUM approach and Jira does support SCRUM out of the box.

Meetings with the product owner are held every Wednesday. These meetings are used for

sprint reviews, retrospectives and further sprint planning.

Backlog items are estimated using story points featuring the Fibonacci sequence which is a common practise using SCRUM.

2.5 Quality assurance

The quality measures used for the Demo Facility project range from reviews of (sub-)deliverables to automatic testing during the development phase.

During the first part of the project that is done waterfall-based, meetings are scheduled bi-weekly with the customer. These meetings are used to review the deliverables like the Software Requirements Specification (SRS), the use cases and the research findings. This is done to find quality issues in an early stage so they can be fixed soon after they occurred.

For the second part of the project that is SCRUM-based, progress is reviewed in each weekly sprint meeting. Apart from that automatic measures are taken during the development process. To ensure quality, the implementation is done test-driven. As a unit testing framework, Jest is used. Jest is a Javascript testing framework developed by Facebook that was originally intended for usage with React. However Jest nowadays provides Typescript support and built-in code coverage reports. (*Jest*, n.d.) All services and models are fully covered with unit tests.

To keep the code clean and uniform, Linting is used with Typescript. Linting is a technique that analyzes code for stylistic and programmatic errors before even compiling.(*Code Linting in JavaScript*, n.d.)

2.6 Stakeholder

The following parties are involved in the project:

- Company Coach: Stefan Sobek
- University Coach: Thijs Dorssers
- Customers Demo Facility: Gregor Schwake, Danny Jonker
- Fontys ILEC

2.7 Time planning

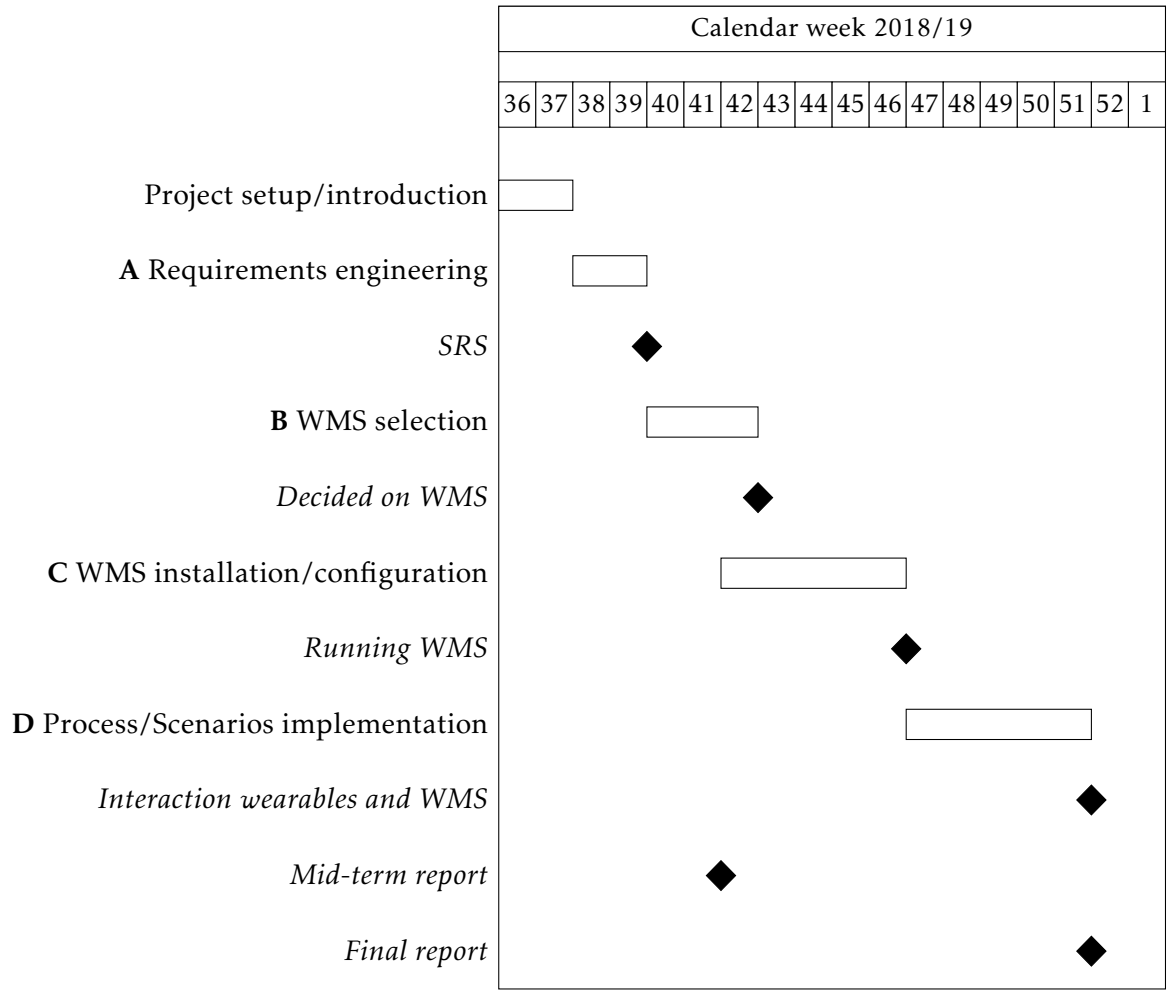

Figure 1: Time planning (Gantt)

Figure 1 shows the initial time planning of the project. This planning mostly worked out during the project. Only activity C could not start before the activity B was ended. This was because the research part showed that there was no suitable system for usage with the Demo Facility. So the start of the design and implementation of the software part was slightly delayed.

2.8 Risk analysis

For the risk analysis three different types of risks are taken into account. These are external risks (**E**) related to external stakeholders, internal risks (**I**) related to the project work and technical risks (**T**).

Identifier	Description	Mitigation
T-1	Connecting a wearable to the WMS/-Mock WMS is not possible	If it is not possible to connect a wearable to the WMS, other wearables are getting a higher prioritization
I-1	Too broad project scope	The project scope is limited at project start and only widened if enough time is left
E-1	No available WMS fulfills the set requirements	If no suitable WMS is available on the market, a fake WMS will be created
E-2	Customer changes requirements of demo facility in later project stages	Regular feedback sessions with customer (bi-weekly)
E-3	Long delivery time of required hardware (Workstation)	Ordering hardware very early in project
E-4	Long delivery time of required hardware (Wearables)	Ordering hardware very early in project

Table 2: Risk analysis

Each risk scores a specific value calculated by impact multiplied with probability. Impact and probability are measured on a scale from 1 to 5 where 5 is the highest impact or probability. Values from 1-5 are considered low risk, values from 6-10 a medium risk, values from 12-16 a high risk and values from 20-25 a very high risk.

T-1: As one requirement for the WMS is an interface the probability of this risk is rather low. However if this happens with multiple wearables the impact is very high.

I-1: As the scope of the project is very limited from the beginning, the probability of a too wide scope is rather low. The impact of not satisfying the project scope is really high as the scope is already rather limited.

E-1: The probability of not finding a suitable WMS for the purpose of the demo facility is quite high as it is a very special environment. The impact is medium as an own implementation of an WMS-like system should be possible.

E-2: As the requirements were set and agreed on in early project stages, significant changes in requirements have a low probability. The impact although would rather high.

E-3: A long delivery time of the workstation components of the demo facility is not very probable as only items in stock will be ordered. If no suitable workstation components can be delivery this has a high impact.

E-4: A long delivery time of wearables is not very probable as only items in stock will be

ordered. As there are already some wearables available for usage with the demo facility, the impact is quite low.

5		T-1/E-3			
4		I-1			
3	E-2		E-1		
2		E-4			
1					
Impact/ Probability	1	2	3	4	5

Table 3: Risk Visualisation

3 Analysis

This section describes the analysis phase of the project. The aim of this phase is to gather the requirements of the demo facility. This phase is split into several parts. First a high-level overview about the required functionality of the demo facility is defined by different usage scenarios of the demo facility and what it is supposed to do at each of them. Another one is a short introduction to wearables in terms of what they are, how they connect and what they are capable of. This is followed by a classical software engineering approach of creating software requirements specifications, derived use cases from those requirements and verifying them by creating mockups showing these use cases and workflow visually. Last but not least the logistics processes are analyzed in order to get a better understanding on how they have to be integrated in the demo facility.

3.1 Usage scenarios

As mentioned above, the starting point for the requirements engineering phase are the usage scenarios of the demo facility which were defined by the customer upfront.

The first scenario is the usage by companies visiting Fontys for the purpose of making first experience with wearables in logistics processes. For this purpose the user shall be able to try out various predefined processes with sample data utilizing various wearables. The same applies for the second scenario of taking a portable slimmed-down version of the demo facility to a company for testing wearables in logistics processes. In addition in this scenario the visited company shall be able to alter the data in the system to represent their actual data of their warehouse.

The third scenario is an ”Open Day” at Fontys Hogeschool Techniek en Logistiek. For this scenario the demo facility should be used to represent Fontys’ project work and practical students’ projects.

The fourth scenario is using the demo facility for student projects of the information technology department of Fontys. For this scenario students should be able to modify the demo facility software development wise and add their own components to the demo facility for learning purposes and experiencing the combination of logistics and IT.

3.2 Wearables Requirements

Wearables are electronic devices that can be worn as clothing or accessories. They can have similar capabilities as computers or other technical devices such as barcode scanners or displays while usually keeping the hands of the wearing person free. (Büyüközkan et al., 2016)

For the demo facility only wearables already owned by the LOGwear project are considered. In addition only wearables from 2017 and newer are taken as the development in terms of speed and support are fast in this field of technology.

The following table shows the wearables that are considered for usage with the demo facility. It shows the exact model, a short description and how they can be connected to the demo facility.

Model	Description	Connection
ProGlove	The ProGlove is a glove equipped with a barcode scanner.	USB
ZEBRA RS6000	The Zebra RS6000 is a ring barcode scanner. It can be worn on two fingers.	Bluetooth
ZEBRA WT6000	The Zebra WT6000 is an arm terminal running Android.	Bluetooth, WIFI, USB, NFC
Google Glass 2	The Google glass is a smart glass running a slimmed-down version of Android.	Bluetooth, WIFI, USB

Table 4: Wearables of the LOGwear project

3.3 Software Requirements Specification

The Software Requirements Specification document describes the overall goals of the demo facility and furthermore lists specific requirements of the demo facility. This document is based upon the defined usage scenarios (refer to section 3.1), an initial customer interview and two following customer meetings.

Before writing the SRS, the customer specified that the final software part of the demo facility should follow a backend/frontend structure so these terms are already mentioned in the SRS.

The SRS specifies different types of requirements which can be divided roughly into functional and non-functional requirements. The functional requirements describe functionality of the software part of the demo facility and are mostly derived from the usage scenarios. The non-functional requirements partly describe requirements of the software architecture as well as requirements of the hardware part of the demo facility. These requirements are far more wide-spread and are mostly derived from various customer meetings and are more often subject to change than the functional requirements.

3.3.1 Examples of specific requirements

To give an impression of the different types of requirements, a few of most types are mentioned below. All further analysis results like Use Cases will refer to these requirements when possible to give an impression on how the requirements influence design decisions and the implementation of the application. The entire SRS document can be found in Appendix B.

3.3.1.1 Notation of specific requirements

All specific requirements contain of three parts. The first part is the identifier that is unique and provides an indication of the type of the requirements. Possible types are *Functional Requirements* (Identifier: FR), *External Interface Requirements* (Identifier: IR), *Maintenance Requirements* (Identifier: MR) and *Design Contraints* (Identifier: DC). The second part is the priority of the requirements. This is done according to RFC 2119 which defines the following:

> **MUST** This word, or the terms ”REQUIRED” or ”SHALL”, mean that the definition is an absolute requirement of the specification. **MUST NOT** This phrase, or the phrase ”SHALL NOT”, mean that the definition is an absolute prohibition of the specification.
>
> **SHOULD** This word, or the adjective ”RECOMMENDED”, mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
>
> **SHOULD NOT** This phrase, or the phrase ”NOT RECOMMENDED” mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
>
> **MAY** This word, or the adjective ”OPTIONAL”, mean that an item is truly optional. [...] (Bradner, 1997)

Instead of MUST and MUST NOT the terms SHALL and SHALL NOT are used.

The third part of each requirement describes the requirement and repeats the flag word used for prioritization.

The following section will provide an example for each of the four requirement types.

3.3.1.2 Examples of functional requirements

The functional requirements describe what the system shall or should do and how it shall or should behave. So these requirements can usually be directly transformed into several use cases. An example of a functional requirements is the following:

Identifier	Priority	Description
FR-1	<Shall>	The frontend shall provide a way to add processes to the system and modify processes of the system.

3.3.1.3 Examples of non-functional requirements

The non-functional are further categorized in different types of requirement which can be differentiated in this section by their identifier.

Identifier	Priority	Description
IR-1	<Shall>	The client shall be able to communicate via bluetooth in version 4.0+.
MR-4	<Shall>	The demo facility shall be start-able/runnable by users with administrator role without an IT background.
DC-1	<Shall>	The demo facility shall be fully functional without requiring an internet connection/offline.

3.4 Use Cases

Use Cases describe outwardly visible requirements of the software system. They are used to create and valide the software design and ensure meeting all requirements. (Schneider et al., 2001)

The Use Cases are created on top of the functional requirements of the SRS and give detailed scenarios of how the user will interact with the system. Furthermore these Use Cases will also be used during the agile development of the software. Section 3.4.1 provides an example for a use case. All use cases can be found in Appendix A.

3.4.1 Examples

As mentioned above the Use Cases are based upon the functional requirements of the SRS. So the following Use Cases are examples of how the functional requirements are transformed into Use Cases. To make the whole requirements engineering process traceable the Use Cases include a back reference to the functional requirements if given.

Use Case	**Create Process Activity**
Code	UC-4
Requirement	FR-1
Actor	Administrator
Description	The administrator creates activities for a process.
Precondition(s)	• The user is authenticated and has the administrator role • At least one process exists.
Scenario	1. User opens the administration interface in the frontend. 2. User navigates to the process administration page. 3. User opens list of existing processes. 4. User opens one of the processes. 5. User clicks on ”Activities” tab. 6. User clicks ”Create activity”. 7. User chooses type of activity. 8. User enters data input/output. 9. User saves activity. 10. System checks if provided data is valid.
Extension	
Result	Process activity is created.

Table 5: Use Case: Create process activity

3.5 Mockups

For verification of the gathered requirements and to give a visual impression of the use cases, mockups are created. These mockups show how the product will behave and enables the customer to make corrections before the actual product is built. As the mockups are built on top of the use cases, below the mockups for the process creation, process activity creation and simulation of a process are shown as examples. All other mockups can be found in Appendix C.

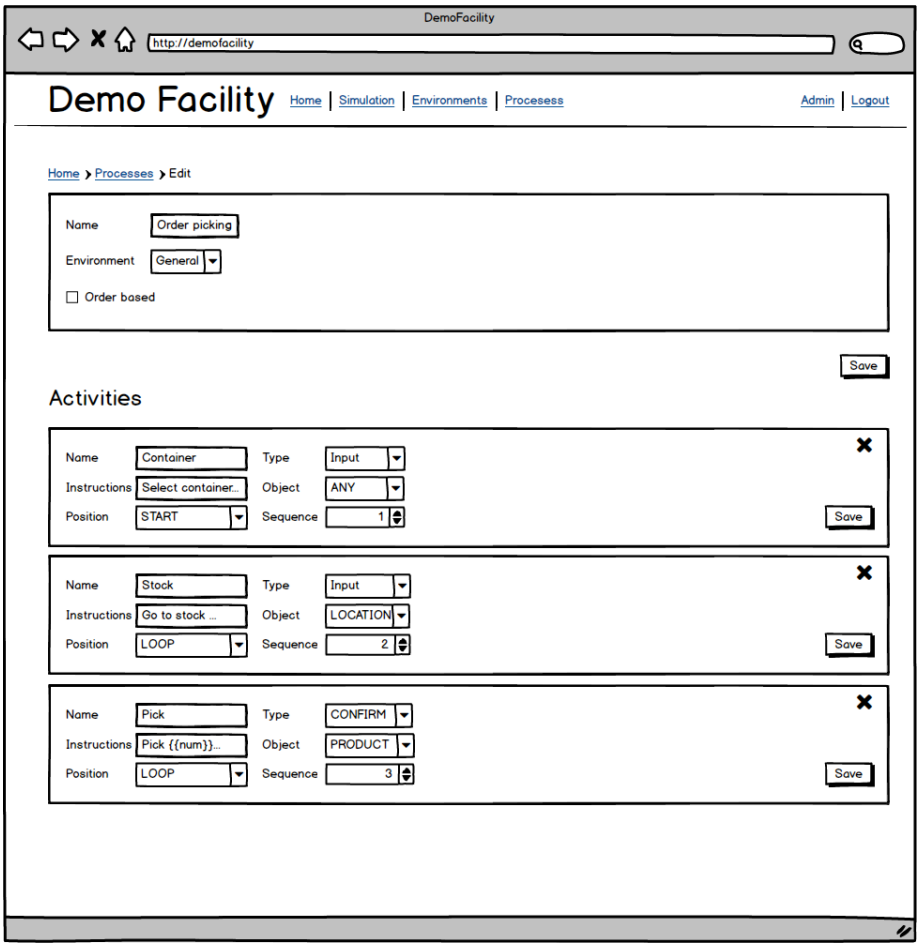

Figure 2: Mockup: Create process activities

In Figure 2 the visual representation of the use case ”Create Process Activity” is shown. It already gives an impression how the user interface for the creation of process activities will look like in later project stages.

3.6 Logistics processes

As the demo facility is aimed to allow simulation of various different logistics processes, these processes need to be analyzed and modelled before creating the demo facility. According to the SRS, the most important process for the first version of the demo facility is the order picking process followed by the putaway process.

There are plenty of variants of this process and this process also tends to be company-specific. For the demo facility the order picking process is modelled according to the research results already available in the LOGwear project. As the LOGwear project does not

yet provide modelled versions of the processes and only lists activities, these processes are modelled for the demo facility project.

Both processes can be found in Appendix D.

In figure 3 the order picking process is shown.

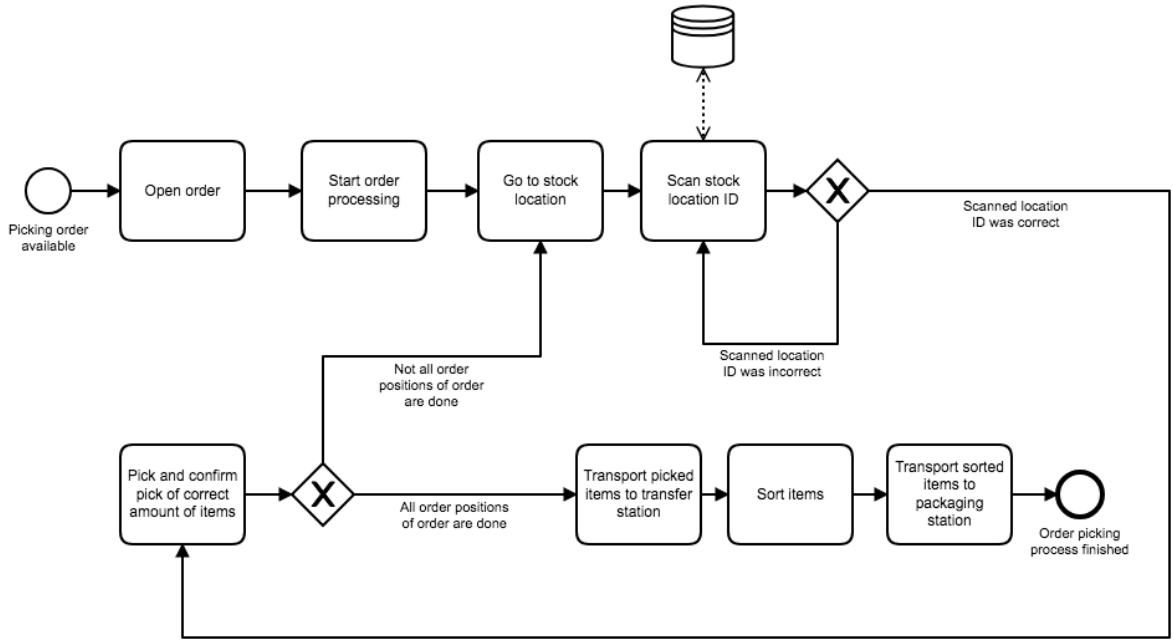

Figure 3: Process: Order Picking

As the order picking process is the main process for the MVP of the demo facility, this process was analyzed first. Key findings are that this process has several activities at the start and several activities at the end that only need to be done once per sequence. Between these activities there are activities that are repeated based on how many items need to be picked for fulfilling the order. As this could also be found in other processes without further analysis, this logic is used for designing an approach that could be reused for other order-based processes with further extension/modification.

4 Research

This section sums up the research results of the Warehouse Management System selection. The entire research can be found in Appendix F.

The purpose of the research is finding an existing software solution that is capable of fulfilling the gathered requirements regarding the software part of the demo facility instead of building a WMS-like system specifically for the demo facility.

4.1 Research setup

The research criteria were setup according to the Software Requirements Specification Document (refer to B) and each criteria was assigned with a score model between 0 and 3 points. Each score was defined prior to further investigation. After defining the criteria, every criteria was assigned with a weight between 1 and 9, where 1 is a low priority criterion and 9 is a very, very high priority criterion. Certain criteria (e.g. price) were taken as KO criteria to filter out candidates before assessing them in detail.

The candidates were selected based on a book and recommendation lists of certain websites. Additionally the already existing WMS-like implementation of a LOGwear pilot was also considered as a candidate.

4.2 Research result

Candidate/ Criteria	**Weight**	**Max. Score**	**OFBiz**	**OpenBoxes**	**Metasfresh**	**Nuclos**
Processes	9	3	9	9	18	18
I/O	9	3	0	9	0	0
Extendable	6	3	0	6	6	12
Simplicity	6	3	0	12	6	0
Setup	3	3	3	6	9	9
Platforms	3	3	3	3	9	9
Clients	1	3	3	3	3	3
Overall			**30**	**48**	**51**	**51**

Table 6: Research result

The research result (refer to table 6) shows that three of the candidates score either equally or almost equally and Apache OFBiz follows with a big margin. So OFBiz is no taken into consideration for further discussion of the results.

However, if you look closely, the three remaining candidates also score quite low points as they not even reach half of the possible 111 points. Especially taking into account the most important criteria Process/activity support and Input/Output matching none of the WMS in the comparison can score more than 9 of 27 points in average with Metasfresh and Nuclos even scoring 0 points in Input/Output matching.

Comparing the WMS with a self-made solution that is directly designed for the purpose of the Demo Facility, the required effort for a self-made solution seems to be lower than customizing one of the existing solutions. Especially in terms of the Process/Activity support and the Input/Output matching a custom-made software solution would be better as it would be designed to define own processes with custom data. Also in terms of being extendible a custom made solution is better as all design-related documentation including all source codes would be available. Also an external API can be fully documented and especially for the purpose of the demo facility which enhances extendability.

So the choice between using an existing system and building a custom one fell upon building a custom one.

5 Design

This section describes the hardware and software design of the demo facility. Most of the design work is based upon the requirements specification in section 3 and the entire SRS document in Appendix B. Focus lays on the software design of the backend.

A full overview over the software design can be found in Appendix E.

5.1 Hardware Selection

The hardware selection of the demo facility is mostly based on the non-functional requirements of the Software Requirements Specification as found in Appendix B.

As according to DC-9 and DC-10, the demo facility will consists of two independent environments, one stationary and one portable environment.

5.1.1 Computers

Due to requirements DC-3 and DC-4 a Mac must be used as the computers as only Macs can run Windows and macOS. DC-11 indicates that for the stationary environment one of the two all-in-one solutions of Apple shall be used, namely an iMac or iMac Pro. For the portable demo facility a Macbook Pro with 15 inch screen is chosen to fulfill requirement DC-10.2.

As the computers shall also be used for development purposes (DC-9.3) also including development using the Unity 3D game engine, a powerful computer is needed. To reduce compiling times and for multi-tasking CPU and RAM resources are upgraded.

5.1.2 Networking

As requirements DC-1 and DC-2 indicate building a local network for the demo facility, therefore a router is required. As DC-11 also applies to the networking part of the demo facility the router should have wifi to reduce the usage of cables. To support the capabilities of the computers, the router should support the latest wifi standard 802.11ac.

While the portable demo facility consists of only one workstation, a local wifi network is still required to connect wearables with wifi connection to the demo facility backend. Therefore the same router model is also used for the demo facility.

5.1.3 Presentation

For presentation, especially at companies or on Open Days at Fontys, a beamer and screen is needed. These will be used for presentation, for showing the graphical interface of the demo faciltity or for showing what the user sees through the wearable.

5.1.4 Full Hardware List

The following hardware is chosen for the demo facility. The workstations and the laptop were upgraded as the budget allowed.

Amount	Type	Name	Extra configuration
2	Workstation	iMac Pro	64 GB RAM
1	Notebook	Macbook Pro 15 inch	32 GB RAM, 512 GB SSD
2	Router	TP-Link Archer C7	-
1	Projector	Optoma GT1070Xe Wi	-
1	Projection screen	Mobile projection screen (160 cm x 90 cm)	-

Table 7: Wearables of the LOGwear project

Figure 4 gives an impression how the stationary Demo Facility looks like. Both workstations provide bluetooth and WIFI connectivity and are connected via the network provided by the WIFI router. Wearables can either be connected to the workstations or to the WIFI network the router provides.

Figure 4: Hardware setup [2]

[2] Icons made by Smashicons, Pixel Buddha, Gregor Cresnar, mavadee from www.flaticon.com

5.2 Architecture and Technology

The software part of the Demo Facility follows a server client architecture with a backend holding the data and implementing the business logic and the frontend for visualization and user interaction. This general architecture was specified during the requirements engineering phase. Figure 5 below shows the high-level architecture which is explained below.

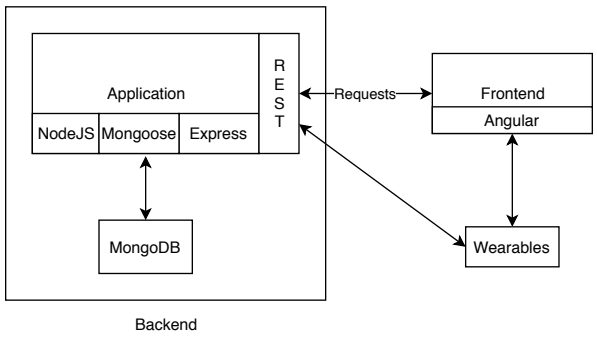

Figure 5: High-level Software Architecture

The backend application is developed with NodeJS and the Express framework for building a RESTful API. NodeJS is built on top of Google Chrome's JavsScript engine and allows to run JavaScript-based applications on server side. (Lei et al., 2014) Instead of Javascript, Typescript is used as the programming language. Typescript is a superset of Javascript and enriches Javascript with classes, interfaces, modules and types. (Bierman et al., 2014) The intention behind choosing Typescript over Javascript is to make the software easier to maintain as Typescript is already used in other LOGwear projects like in the Angular frontend of the Knowledge Base and will also be used for the Angular frontend of the Demo Facility. So the whole software part of the demo facility is developed in one programming language.

For persistence MongoDB with Mongoose is used as database and database driver combination. MongoDB is commonly used with NodeJS applications and can interact natively with Javascript objects and JSON. (Dayley, 2014) Being schemaless, MongoDB also allows for big changes in database structure during development (Aghi et al., 2015) and so supports the agile development process of the demo facility. For connecting the application to the MongoDB Mongoose is used as the database driver.

While a frontend is developed in scope of the project, it is not explained in detail in this report. This is because the frontend is a basic Angular application consuming the REST API of the backend. As there are no special business logic or design decisions involved, focus in this report lays on the backend.

5.3 RESTful API design

The RESTful API provided by the backend application is the main communication interface for the Angular-based frontend and is also intended for connecting more capable wearables that can directly interact with the backend. Having a good API design that follows conventions and best practises is therefore crucial to prepare the Demo Facility for further extensions in the future. The API design is based on the scientific paper "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices" by Roriguez C. et al (2016) with minor tweaks.

5.3.1 Operations

The operations are represented by the standard HTTP methods GET, POST, PATCH and DELETE. The following list shows which HTTP method is responsible for which kind of operation.

HTTP Method	**Operation**
GET	Retrieves a single resource or multiple resources as an array.
POST	Creates a single resource or multiple resources.
PATCH	Updates a single resource or multiple resources.
DELETE	Deletes a single resource or multiple resources.

This clear separation based on HTTP methods makes it possible to perform multiple operations on one API endpoint. This advantage is used for the endpoint modelling described in the next section.

5.3.2 Endpoint design

For the endpoint design nouns are used for representing the resources and the different operations of these resources are represented by the different HTTP methods as explained in section 5.3.1. Verbs for different actions are avoided in the API endpoints. The following table gives an example for the endpoint design for the "Process" resource and is also applicable for all other resources and the simulation endpoints.

Endpoints	Endpoints	Description
/api/processes	GET	Retrieves an array of multiple processes.
/api/processes	POST	Creates a new processes resource or set of process resources.
/api/processes	PATCH	Updates a processes resource or set of process resources.
/api/processes	DELETE	Updates a processes resource or set of process resources.
/api/processes/:id	GET	Retrieves a specific process.
/api/processes/:id	PATCH	Updates a specific process.
/api/simulation/ process/:processId/ order/:orderId	POST	Creates a simulation or retrieves simulation data if simulation already exists
/api/simulation/ process/:processId/ order/:orderId/reset	POST	Resets simulation and pulls new data from process and orders
/api/simulation/ process/:processId/ order/:orderId/delete	POST	Deletes a simulation
/api/simulationActivity/ :id	POST	Checks input data against expected value of simulation activity

5.3.3 Request and Response Payload

The Payload of POST and PATCH requests and responses and the payload of all GET responses is in JSON format. Depending on the response type, a specific HTTP status code is sent back indicating if the operation was successful (status 200 or 201) or not. If the operation is successful, the status code also indicates the reason for the failure (e.g. 401 for not being authorized or 500 for a server error). The format of the response is also standardized.

For successful operations delivering back data the ”success” indicator is true und all data is send back after the ”data” property of the JSON response. The status code is 200 or 201 if a resource was created for the request.

```
{
    "success": true,
    "data": {
        "_id": "5bfd387ef43ba91b21bfcf44",
        "name": "Process name",
        "createdAt": "2018-11-27T12:28:46.574Z"
    }
}
```

In case the operations goes wrong and cannot be completed as requested, an error is sent back in the response. The ”success” indicator is set to false and explanations of the error are sent back in the ”error” property of the response. Furthermore a matching status code is sent back, in the example below status code 500 as the requests caused an error on the server.

```
{
    "success": false,
    "error": {
        "message": "Error message"
    }
}
```

5.4 Basic software design

The following section describes the basic design principles of the demo facility backend. It describes the routing and the interaction between the controllers, services and the database. Also the design of the different resources/models and their relations are described.

5.4.1 Separation of Controllers and Services

The controllers of the software are just responsible for forwarding data from incoming requests to services and sending responses with the data returned by those services. This means that all business logic and database interaction is handled by the services. So the business logic can be easily changed without adjusting the RESTful API logic in the controller and routing. It also allows (unit) testing without mocking HTTP requests by directly testing the services that include all business logic.

5.4.2 CRUD operations utilizing the Repository Pattern

To improve the extendability of the backend, basic ”Create”, ”Read”, ”Update” and ”Delete” (CRUD) operations of resources are implemented according to the repository pattern. While

not being one of the design patterns introduced by the Gang of Four, it is a common design pattern for CRUD operations. The idea of this pattern is having a generic abstraction of data operations. (Bergman, 2017) In case of the demo facility that means that there is a generic implementation of a resource controller and resource service that can be applied to every model that is newly introduced to the system. In case more operations are needed or a generic implementation does not match with the needs of the model that is going to be added, the controllers and services can be extended for this specific model or functions can be overwritten to follow a different logic for a model.

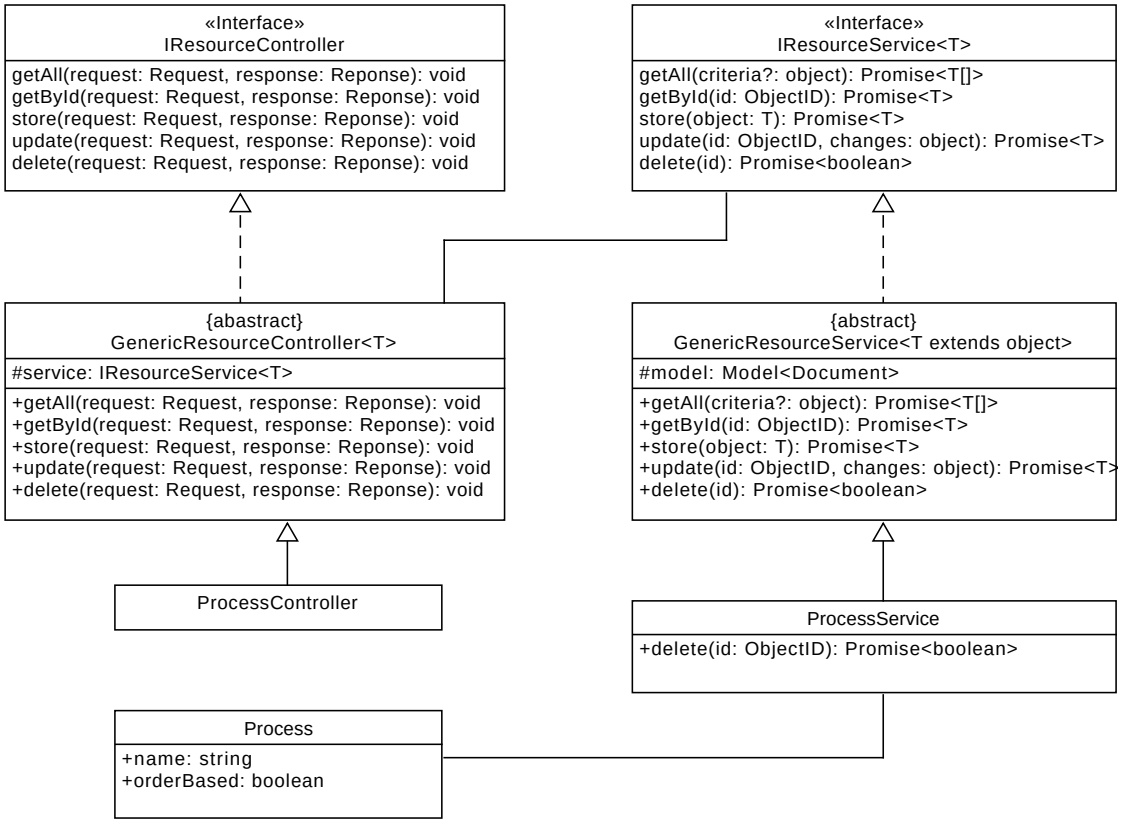

Figure 6: Class Diagram: Repository pattern

Figure 6 shows this design pattern per example for processes. For the service side an interface ”IResourceService” for generic type T with the basic CRUD operations is created. This interface is extended by a generic implementation of that interface that also takes the generic type T (that extends object). This ”GenericResourceService” has a model of a Mongoose specific type that is responsible for database interaction in the service and defined while instantiation of the service. For each resource (also referenced to as model) a specific service is implemented that extends the ”GenericResourceService” with the type of the resource (in this case of type ”Process”).

On the controller side also an interface is implemented called ”IResourceController”. This defines the CRUD operations. This interface is implemented by a generic ”GenericResourceController” of type T, that basically forwards the incoming requests to the responsible service. This service is available as an instance in the resource controller. For each resource a specific controller implementation is created that extends the ”GenericResourceController” with the specific type of the resource.

As shown in the example this pattern allows introducing new resources/models to the system with little work as only special operation for a resource have to be implemented. In the example in figure 6 only the delete operation of the process resource in ”ProcessService” has a special implementation that overrides the generic one from the ”GenericResourceService”. This is the case as a process also has process activities that also need to be deleted when deleting a process.

5.4.3 Sequence from Request to Response

Figure 7 shows a sequence diagram of a get request of a specific process. After the request got routed to the corresponding ”ProcessController”, the controller calls the corresponding method on the server with the data from the requests. The ProcessService then handles the interaction with the process model that gets the process via a Mongoose method. The process is then returned as a response.

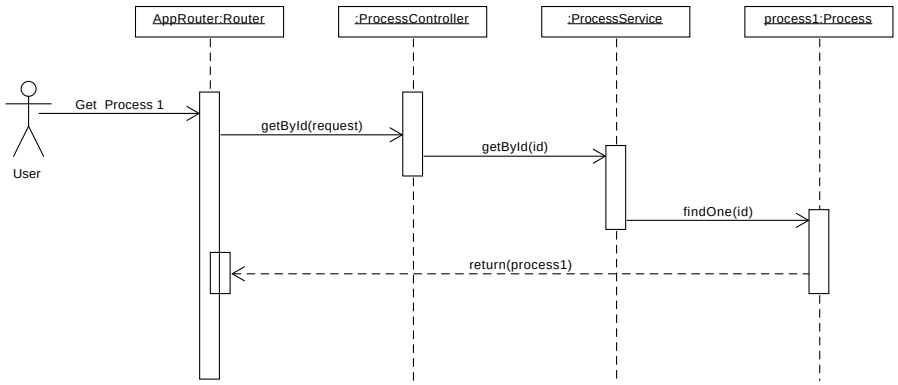

Figure 7: Sequence Diagram: Request sequence with controllers/services separation

5.4.4 Models

All models are defined as a Mongoose schema, Mongoose model and additionally as a Typescript class. While definition as a Mongoose schema and model enables the database interaction and activates schema and validation support for the MongoDB, defining each model

as a class supports the development process by allowing checks and easier interaction with object properties.

In Figure 8 the model classes and their relations among each other are shown. Each model that is currently in the application extends the PersistentModel class as they get assigned with the properties "_id" and "createdAt" once stored and retrieved from the database. If non-persistent models were required in the future, they would not extend this class.

Every model class, that is not in the child part of another composition, belongs to an environment. This enables the possibility, to specify multiple environments in the application that have their own processes, products, storage location etc. so that the demo facility can be used in the various different scenarios as defined in section 3.1. So for example different products and stock locations can be defined for different companies to have data that is similar to their real WMS.

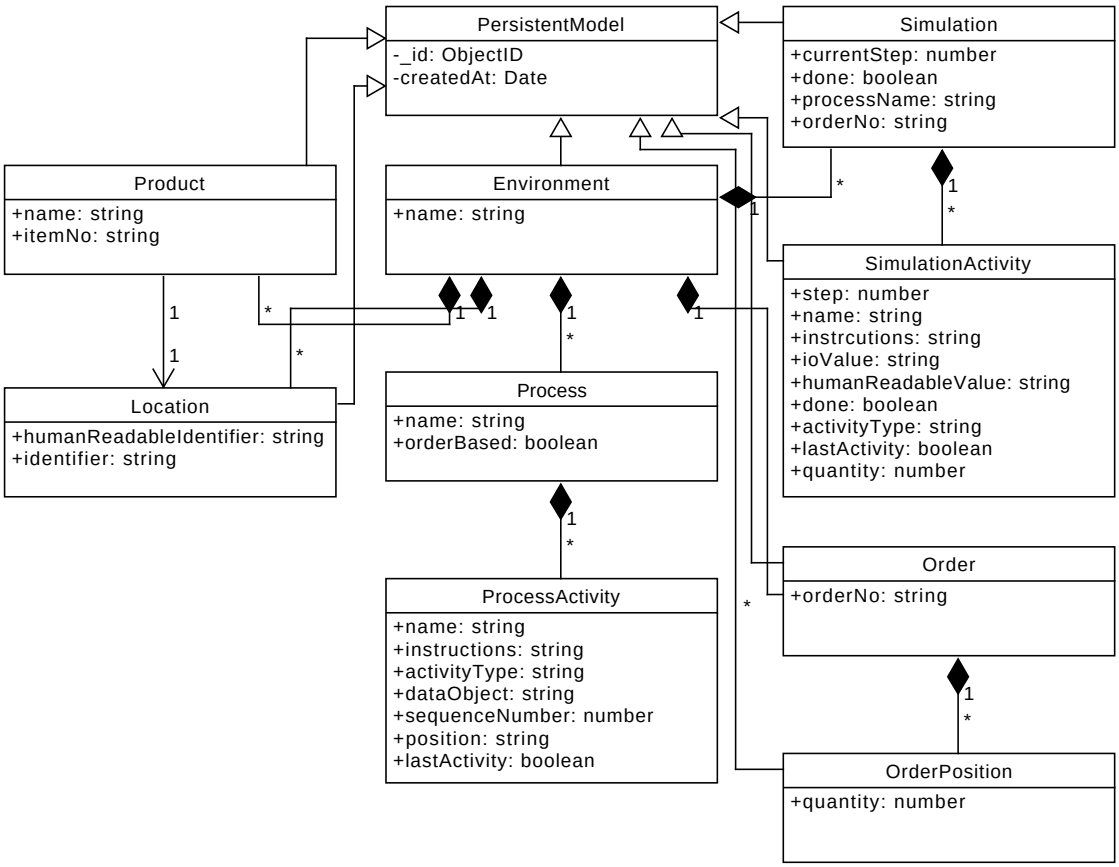

Figure 8: Class Diagram: Model relations

5.5 Simulation logic

The following section describes how the data for a simulation is gathered from the predefined environments. It also explains how a simulation is done.

5.5.1 Starting a simulation

When an environment is defined, available processes in this environment can be used for simulation. Right now this only works with processes that are based on an order (e.g. order picking). When the simulation is started, the data is collected according to the specifications of the process activities. The most important properties of the process activity are ”activityType”, ”dataObject”, ”sequenceNumber” and ”position”. The ”activityType” defines what is done during the simulation and which checks are necessary. Possible values are ”INPUT”, ”OUTPUT” and ”CONFIRM”. While the first two require a ”dataObject” that is gathered from another model like Product or Location, ”CONFIRM” just requires a confirmation without any specific data. The ”dataObject” property can be either ”ORDER”, ”PRODUCT”, ”LOCATION” or ”ANY”, while the last one can be used in combination with INPUT if the input does not matter for the further simulation of a process. The ”position” property together with the ”sequenceNumber” specify when the activity takes place during the simulation and if it only takes place once or multiple times. Possible values for the ”position” are ”BEFORE_LOOP”, ”AFTER_LOOP” and ”IN_LOOP”. While the first two define that an activity will only take place once, the ”IN_LOOP” value specifies that the activity will be repeated for every order position. The ”sequenceNumber” then defines the detailed position of each activity in their respective position scope.

All gathered data from the definitions in the process activities is then copied into ”SimulationActivity” objects and some more data about the process and orders is put into ”Simulation” objects.

This logic is shown in the example in Figure 9 below.

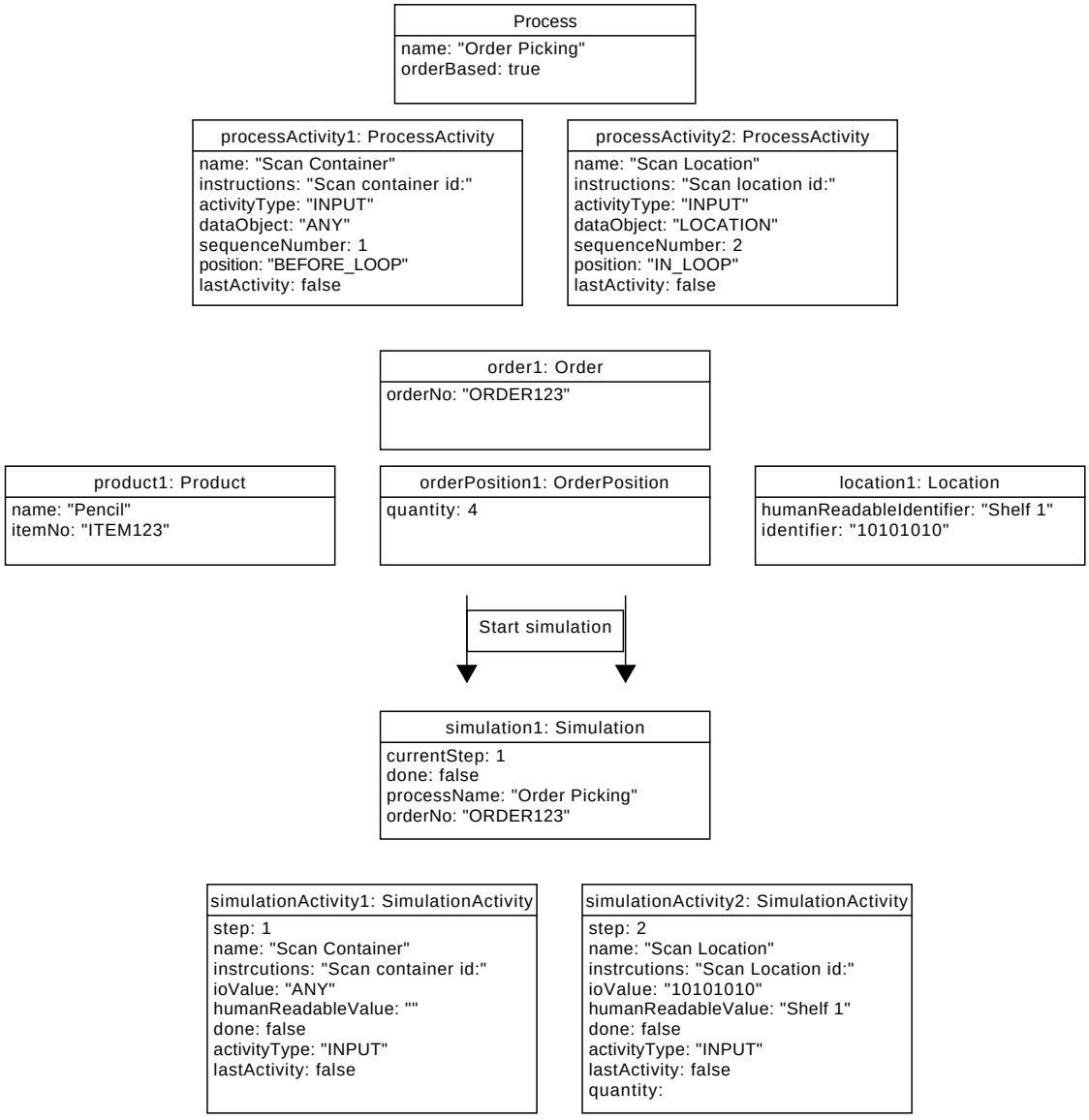

Figure 9: Simulation logic

In the ”SimulationActivity” objects the data is stored according to the specification in the ”ProcessActivity” objects. The Simulation object has information about the process and order and the current status of the simulation.

The data is copied to the simulation objects to avoid any interference during a simulation when someone changes data in an environment.

5.5.2 Doing a simulation

After a simulation and its respective simulation activities were created, the system is ready for interaction with information coming from the frontend (and attached wearables) or from wearables directly to actually simulate a process.

As figure 9 shows, each simulation activity has specific type and may have a specific ”ioValue”. Depending on the type, the system will await either just a confirmation which will set an activity to done or if the activity expects an ”ioValue” the system will check this value against the incoming data and then set the activity to done. Apart from the value, the system also checks if the correct activity in the correct order is executed. This repeats for every simulation activity. The last simulation activity has the property ”lastActivity” set to true which tells the system to also set the simulation to done.

5.6 Deployment

As the SRS (refer to B) states that the demo facility shall run on macOS and Windows and shall be set up and start-able by users who do not have an IT background, the application will be deployed via Docker containers. Containers allow to pack all dependencies and the execution environment of an application into a self-contained unit so that the application can be run on any system. While there are multiple container solutions beside Docker, Docker became the de-facto standard. (Cito et al., 2017).

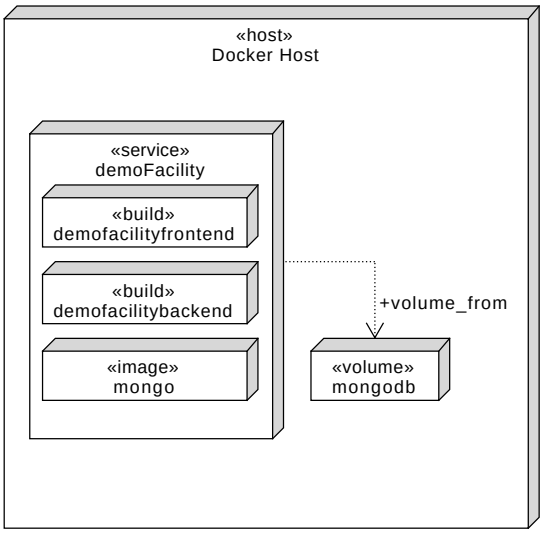

Figure 10: Container structure

Figure 10 shows the deployment diagram of the demo facility application. The host is in

case of the stationary environment (refer to 5.1) one of the iMac Pros and in case of the portable environment the MacBook Pro. On this host the demo facility application is run as one service containing of three containers and a volume for persisting data across recreating containers. While the containers for the frontend and backend are built before deployment, for the MongoDB database a prebuilt image is used.

More details can be found in section 6.3.

6 Implementation

This section describes the implementation of the software design.

6.1 Basic Software Design

This section describes the implementation of the repository patterns and model classes.

6.1.1 Controllers and Services

As explained in section 5.4.1 controllers only handle incoming requests and send back responses while all the business logic is handled by services. The example below in figure 11 shows the controller method called when all instances of a model matching certain criteria are requested from the backend. The ”getAll”-method gets passed the request and response parameters which are of specific types introduced by the Express framework. The request parameter contains all data from the incoming request like query parameters which can be used as filter criteria (line 7) and the response parameter is used for sending back the response. These filter criteria are passed to the ”getAll”-method on the service which then handles the whole business logic (line 8). The only responsibility of the controller is checking the response of the service for a result (line 9) or an error (line 12) and send back a matching response for each case.

The service methods are asynchronous sending back Promises so the ”.then and .catch” syntax can be used on the controller side after the Promise is resolved.

```
/**
  * Returns a list of entity
  * @param req Request for specific set of entities
  * @param res Response of specific set of entities
  */
public getAll(req: Request, res: Response): void {
  const searchCriteria: object = req.query;
  this.service.getAll(searchCriteria)
    .then((results) => {
      new StandardResponse(res, true, 200, results).send();
     })
    .catch((err) => {
      console.error(err);
      new ErrorResponse(res, false, 500, 'An error occured.').send();
     });
}
```

Figure 11: Example resource controller method

The business logic for querying the instances of the resource from the database is found in the services. As the example in figure 11 calls the ”getAll”-method of the service, this example is shown below.

```
public async getAll(criteria?: object): Promise<T[]> {
  const documents: Document[] = await this.model.find(criteria).exec();
  const results: object[] = [];

  for (const document of documents) {
    results.push(document.toObject());
  }

  return results as T[];
}
```

Figure 12: Example service method

The ”getAll”-method is declared as asynchronous and takes the search criteria as parameter und returns a Promise containing an array of a generic type T. First the method queries all matching Mongoose documents from the database. This is done via a method provided by the Mongoose model. As this method also returns a Promise the ”await” syntax can be used to handle asynchronous operations like synchronous operations. To avoid working with Mongoose or MongoDB specific objects through out the backend, the Mongoose documents are transformed into objects and then an array of plain Typescript objects is returned from

the method.

6.1.2 Repository pattern

As mentioned in section 5.4.2 the repository pattern is used for the CRUD functionality of the backend. This pattern can already be partly seen in figures 11 and 12 as a generic service is called and the service returns a generic array of objects of type T.

The repository pattern is implemented using the interfaces ”IResourceController” and ”IResourceService” which specify which methods are needed for all basic CRUD operations and what parameters and return types are needed. These two interfaces are then implemented by the ”GenericResourceController” on Controller side and the ”GenericResourceService” on service side. Figure 13 shows an excerpt from the ”GenericResourceController”.

```
1  export abstract class GenericResourceController<T> implements
       IResourceController {
2
3    public constructor(protected service: IResourceService<T>) { }
4
5    public getAll(req: Request, res: Response): void {
6    this.service.getAll(searchCriteria) [...]
7
8    [...]
9
10 }
```

Figure 13: Code sample: GenericResourceController

The generic controller has an instance of a generic service implementation used for querying the database that is set during instantiation of the controller. So it can be reused for different models. This generic service is shown in figure 14 below.

```
export abstract class GenericResourceService<T extends object> implements
    IResourceService<T> {

  public constructor(protected model: Model<Document>) { }

  public async getAll(criteria?: object): Promise<T[]> {

    [...]

  return results as T[];
}
```

Figure 14: Code sample: GenericResourceService

For example purposes this approach is shown for adding the resource of process activities to the system. Therefore the ”ProcessActivity” model, the ”ProcessActivityController” and ”ProcessActivityService” have to be created. As the process activities do not need any further methods than those defined in the generic implementations, adding this resource does not require much work.

```
export class ProcessActivity extends PersistentModel {

  public constructor(
    public name: string,
    [...]
  ) {
      super();
    }

  export const ProcessActivitySchema = new Schema({
    name: {
      type: String,
      required: true,
      index: { unique: true },
    },
    [...]
  });
}

export const ProcessActivities = model('ProcessActivity',
   ProcessActivitySchema);
```

Figure 15: ProcessActivity model

Figure 15 shows the model class for the process activity. It defines a regular Typescript class (lines 3-8) and a Schema for the Mongoose database driver (lines 10-18). In line 20 a Mongoose model is exported that is used in the service class to query from the database.

```
1 import { ProcessActivity } from '../../models/ProcessActivity';
2 import { processActivityService } from '../../services/resources/
    ProcessActivityService';
3 import { GenericResourceController } from './ResourceController';
4
5 export class ProcessActivityController extends GenericResourceController<
    ProcessActivity> {}
6
7 export const processActivityController: ProcessActivityController = new
    ProcessActivityController(processActivityService);
```

Figure 16: ProcessActivityController

The required controller for process activities is shown in figure 16. This class extends the "GenericResourceController" as type "ProcessActivity" so that the generic implementation can be used with process activities (line 5). An instance of this controller is exported and the matching service is set as the parameter (line 7).

```
1 import { ProcessActivities, ProcessActivity } from '../../models/
    ProcessActivity';
2 import { GenericResourceService } from './GenericResourceService';
3
4 export class ProcessActivityService extends GenericResourceService<
    ProcessActivity> {}
5
6 export const processActivityService = new ProcessActivityService(
    ProcessActivities);
```

Figure 17: ProcessActivityService

Figure 17 shows the service used for process activities that is set as the parameter in the controller class. As in the controller class the "ProcessActivityService" just extends the "GenericResourceService" of type "ProcessActivity" which enables the usage of the generic implementations (line 4). An instance of this service is exported and the Mongoose model is set as parameter (line 6).

Adding a resource that does not require any additional methods so only takes about 13 lines of code not considering the model itself.

6.2 Simulation

This section describes the implementation of the simulation creation and how a simulation is done.

6.2.1 Starting a Simulation

As described in section 5.5.1 a Simulation consists of a ”Simulation” object that has several ”SimulationActivity” objects. These ”SimulationActivity” objects have ”activityType”, ”dataObject”, ”sequenceNumber” and ”position” properties that define what is done in an activity and when and where in the sequence the activities occur.

To achieve that the code of the ”startSimulation”-method is called upon request in the ”RunActivityService”.

In lines 3-6 it checks if there is already a running simulation for exactly that process and order. If this is the case instead of creating a new simulation the existing one is returned.

If there is no simulation running it will retrieve all necessary data about the process and order (line 9) and create a simulation object by copying the needed data over (lines 12-19). After the simulation is stored in the database which assigns the simulation with a unique ID the simulation activities are created in a separate method and assigned to the simulation (line 30). After the simulation and simulation activities are created they are returned in line 32.

```
1  public async startSimulation(processId: ObjectID, orderId: ObjectID):
       Promise<object> {
2
3    const existingSimulation: Simulation[] = await simulationService.getAll
       ({ process: processId, order: orderId });
4    if (existingSimulation.length > 0) {
5      return await this.getSimulationData(processId, orderId);
6    }
7
8    // Set global variables
9    await this.retrieveData(processId, orderId);
10
11   // Copy data from process and order to simulation and store simulation
12   const simulation: Simulation = {
13   process: this.process._id,
14   order: this.order._id,
15   currentStep: 1,
16   done: false,
17   processName: this.process.name,
18   orderNo: this.order.orderNo,
19   };
20
21   let storedSimulation: Simulation;
22
23   try {
24     storedSimulation = await simulationService.store(simulation);
25   } catch (err) {
26     console.error('Could not create simulation: ' + err);
27     return null;
28   }
29
30   await this.createSimulationActivities(storedSimulation._id);
31
32   return await this.getSimulationData(processId, orderId);
33 }
```

Figure 18: "startSimulation"-method

The "createSimulationActivities"-method in line 30 contains most of the business logic. As shown in figure 19 it mostly contains of multiple for-loops that store the activities in the right order to the database. This right order is achieved by using switch cases and if statements that respond to the "activityType", "dataObject" and "position" properties of the process activities as mentioned above.

An excerpt of this method is shown in figure 19 below.

```
1  let stepCounter: number = 1;
2
3  // Creates and stores all simulation activities in right order that
       belong before the loop
4  for (const activity of this.activities) {
5    if (activity.position === 'BEFORE_LOOP') {
6      const simulationActivity: SimulationActivity = {
7        simulation: simulationId,
8        step: stepCounter,
9        name: activity.name,
10       instructions: activity.instructions,
11       [...]
12     };
13
14     switch(activity.dataObject) {
15       case 'ORDER': {
16         simulationActivity.ioValue = this.order.orderNo;
17         simulationActivity.humanReadableValue = this.order.orderNo;
18         break;
19       }
20       default: {
21       [...]
22       break;
23     }
24     stepCounter++;
25     await simulationActivityService.store(simulationActivity);
26 }
```

Figure 19: "createSimulationActivities"-method

This excerpt shows the first for loop that is responsible for the simulation activities that are positioned in front of the loop (compare to section 6.2.1). It first checks if the activity is one of those that belong before the loop (line 5) and then creates each of the simulation activities by copying over information from the process activities (lines 6-12). Depending on the "dataObject" type the expected input value of the activity is added to the simulation activity when applicable (lines 14-23) and finally the activity is saved to the database.

6.2.2 Doing a Simulation

Depending on the "activityType" property of the simulation activity, either a specific value, any value or just a confirmation is expected for completing a simulation activity. For ex-

ample the code in figure X checks if the activity the request is for is the correct step in the process at that time and then checks if the provided ”ioValue” of the request matches with the ”ioValue” of the simulation activity.

```
1  if (simulationActivity.step === simulation.currentStep &&
       simulationActivity.ioValue === ioValue) {
2    simulationActivityService.update(simulationActivity._id, { done: true }
       );
3    simulationService.update(simulation._id, { currentStep: simulation.
       currentStep + 1 });
4    const updatedSimulationActivity: SimulationActivity =
5      await simulationActivityService.update(simulationActivity._id, { done
         : true });
6    const updatedSimulation: Simulation =
7      await simulationService.update(simulation._id, { currentStep:
         simulation.currentStep + 1 });
8
9    const result: object = {
10     simulation: updatedSimulation,
11     simulationActivity: updatedSimulationActivity,
12   };
13
14   return result;
15 }
```

Figure 20: ”doActivity”-method

6.3 Deployment

As described in section 5.6 the demo facility is deployed via docker containers. As multiple containers are needed for the backend, frontend and database, Docker Compose is used. Docker Compose allows to define services consisting of multiple containers in one file and to spin them up via one single command.

While the MongoDB is provided as an image from the Docker HUB, the containers for the backend and frontend are custom-built and defined in two Dockerfiles which are briefly explained below.

```
1 FROM node:11
2 WORKDIR /usr/src/app
3 COPY package*.json ./
4 RUN npm install
5 COPY . .
6 EXPOSE 3000
7 CMD [ "npm", "start" ]
```

Figure 21: Dockerfile of the backend

For the backend the (shortened) Dockerfile is shown in figure 21.

Line 1 specifies that the container is derived from the node image in version 11. From line 2 to 5 the directory of the application inside the container is specified, the package.json containing all required node dependencies is copied to the container and the dependencies are installed inside the container. Then the source code of the backend application is copied to the container. In line 6 the port where the application is listening on is opened to the outside and in line 7 the application is transpiled and started.

For the frontend application based on Angular 7 a more complex Dockerfile is needed. This Dockerfile (shortened) is shown in figure 22.

```
1  FROM node:11 as build-stage
2  RUN apt-get update && apt-get install -yq [...]
3  WORKDIR /usr/src/app
4  [..]
5  COPY package*.json /app/
6  RUN npm install
7  COPY ./ /usr/src/app/
8  ARG configuration=production
9  RUN npm run build -- --output-path=./dist/out --configuration
       $configuration
10
11 FROM nginx:1.15
12 COPY --from=build-stage /usr/src/app/dist/out/ /usr/share/nginx/html
13 COPY ./docker/nginx-vhost.conf /etc/nginx/conf.d/default.conf
```

Figure 22: Dockerfile of the frontend

In this Dockerfile a two-stage approach is used. This enables to build the Angular application in an intermediate container that is deleted afterwards and copy the built version of the Angular application to a container that is running an nginx webserver serving the Angular application. For this approach in line 1 the base image for the ”build-stage” of the container

creation is defined. From line 2 to 9 the required dependencies are installed on image and node base and later on the Angular application is built using the production configuration.

As a built version of an Angular application can be served like a static HTML page, the final container is built on top of an nginx image with custom configuration to serve the Angular application (lines 11 to 13).

As mentioned before Docker Compose is used to create and start all containers and the volume needed for the demo facility. For this a file called ”docker-compose.yml” is needed, that defines everything needed.

Figure 23 shows this file. Line 1 defines which version of Docker Compose is used and lines 3-19 then define the services (containers) needed for the demo facility. While the frontend and backend containers need to build (lines 5 and 9), the database service uses a prebuilt ”mongo” image. The files also defines which ports are open and to which port of the host they are bound (lines 7, 11 and 17). In lines 12-13 is defined that the backend starts after the database container is started as the backend needs a running database to work properly. In lines 18-21 the volume of the database container is defined as data shall not be lost after recreation of the database container.

```
1  version: '3'
2
3  services:
4    frontend:
5      build: demofacilityfrontend
6      ports:
7        - "80:80"
8    backend:
9      build: demofacilitybackend
10     ports:
11       - "3000:3000"
12     depends_on:
13       - db
14   db:
15     image: mongo
16     ports:
17       - "127.0.0.1:27017:27017"
18     volumes:
19       - mongodb:/data/db
20 volumes:
21   mongodb:
```

Figure 23: Docker Compose file

6.4 Connecting the Demo Facility and Wearables

After having implemented the software part of the demo facility one important next step is connecting the wearables to the demo facility. There are multiple ways to do so. As described in section 5.2 the Demo Facility is designed to connect to wearables either via the frontend or directly via the REST API. For the time of writing this report, only the first option is considered.

Connecting a wearable to the Demo Facility via the frontend means, that the frontend is opened in the web browser of one of the clients and the wearable is connected to this client. According to table 4 this can either be done via Bluetooth or USB.

Another way of connecting a wearable to the frontend is by running the frontend on the Zebra arm terminal and connecting the wearable via bluetooth to the terminal. As the Zebra arm terminal runs a version of Android it is possible to open the frontend in the web browser without further configuration. As the frontend is responsive (refer to figure 24) this provides a usable experience to the user.

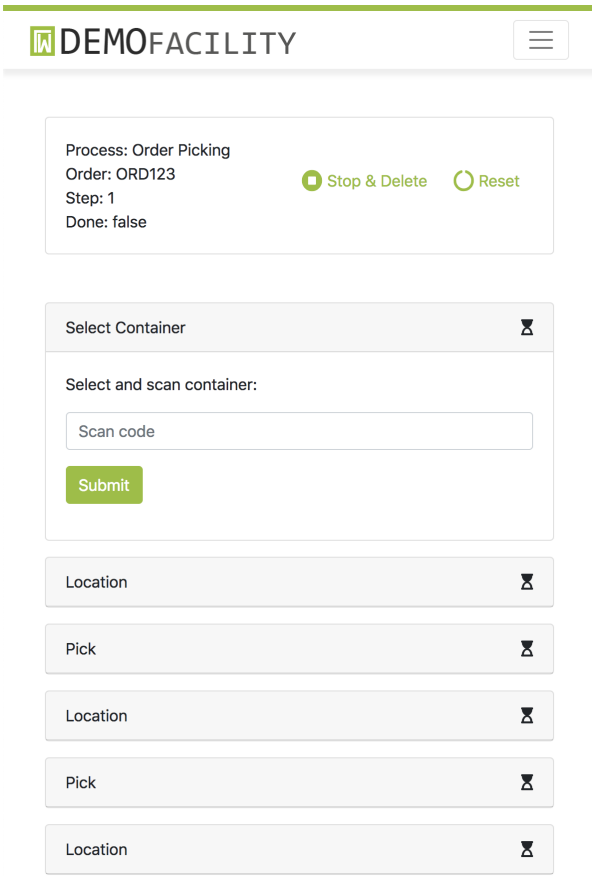

Figure 24: Screenshot: Frontend running on mobile device

7 Conclusion

This report presented the results of the bachelor thesis project of creating and designing a demo facility for wearables in logistics for the LOGwear project. The goal of this project is to build a hardware and software environment that can be used for simulating different logistics processes using wearables. For this a hardware and software platform needed to be created. In scope of the project in terms of the bachelor thesis was creating a minimum viable product that included the hardware as well as a software solution that allows simulating the order picking process with at least one wearable. Furthermore the software solution shall have extendibility in mind for future use with other processes and wearables.

This goal has been reached as hardware was chosen and ordered and a ”fake”-WMS was developed. This ”fake”-WMS allows creation of order-based processes and at least different variations of the order picking process can be simulated. Moreover the wearables can be be connected to the client which allows simulation using three wearables of the LOGwear project. The system is also prepared for connecting other wearables to the system by providing different interfaces like WIFI and Bluetooth via the clients and network equipment as well as a REST API to directly connect wearables to the backend.

Although the goal was reached during the execution of the project there was a lot of room for improvement. Originally the intention was to use an existing WMS for the software side of the demo facility instead of creating a custom ”fake”-WMS. For this particular reason a lot of time was used for research to find a suitable WMS for the demo facility. The research showed that the WMS market is very complex and diverse. Also assessing the candidates against the criteria was difficult as most WMS vendors do not provide a lot of information about their WMS without trying them out. So the research part of the project turned out to be very time consuming without finding a proper candidate. If less time had been invested for research, a more advanced custom implementation would have been possible in time of the project.

Another issue that cost a lot of time in the beginning was choosing and ordering the hardware for the demo facility. This had to be done during the analysis phase and hardware needed to be ordered before the analysis was completed. That had to do with time constraints set by the customer LOGwear, as the budget of the project needed to be used before a certain date.

For future use the project can be extended in a number of ways. First of all, more wearables can be added to the current implementation. This can either be done by connecting them directly to the backend via the REST API or by using other key-board emulating wearables to the client running the frontend. Furthermore, the current logic for order-based processes can be extended to make adding other processes than the order picking to the system as they may require more/different data than the order picking process. Lastly another logic

for processes that are not order-based can be added to the system. This enables simulation of processes like the putaway process already modelled for this project.

References

About. (N.d.). OpenBoxes. [online] Available at: https://openboxes.com/about/ [Accessed Nov. 8, 2018].

Aghi, R., S. Mehta, R. Chauhan, S. Chaudhary, and N. Bohra (2015). "A comprehensive comparison of SQL and MongoDB databases". In: *International Journal of Scientific and Research Publications* 5.2, pp. 228–229.

Bergman, P.-E. (2017). *Repository Design Pattern.* [online] Available at: https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30 [Accessed Dec. 9, 2018].

Bierman, G., M. Abadi, and M. Torgersen (2014). "Understanding TypeScript". In: *ECOOP 2014 – Object-Oriented Programming*. Ed. by R. Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 257–281. ISBN: 978-3-662-44202-9.

Bradner, S. (Mar. 1997). *Key words for use in RFCs to Indicate Requirement Levels.* RFC 2119. Heise Netze, p. 1.

Businessobjekt. (N.d.). Nuclos Wiki. [online] Available at: https://wiki.nuclos.de/display/Konfiguration/Businessobjekt [Accessed Nov. 8, 2018].

Büyüközkan, G., M. Güler, and D. Uztürk (2016). "Selection of wearable glasses in the logistics sector". In: *Proceedings from the 14th international logistics and supply chain congress*, p. 377.

Cito, J., G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall (2017). "An empirical analysis of the Docker container ecosystem on GitHub". In: *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*. IEEE, pp. 323–333.

Code Linting in JavaScript. (N.d.). freeCodeCamp. [online] Available at: https://guide.freecodecamp.org/javascript/code-linting-in-javascript/.

Das Project. (2018). LOGwear. [online] Available at: https://logwear.eu/project [Accessed Sept. 8, 2018].

Dayley, B. (2014). "Node. js, MongoDB, and AngularJS web development". In: Addison-Wesley Professional. Chap. Features of MongoDB, p. 9.

Features. (N.d.). OpenBoxes. [online] Available at: https://openboxes.com/features/ [Accessed Nov. 8, 2018].

Getting Started - Business Users. (2018). The Apache OFBiz Project. [online] Available at: https://ofbiz.apache.org/business-users.html [Accessed Nov. 8, 2018].

Hompel, M. and T. Schmidt (2007). *Warehouse Management: Automation and Organisation of Warehouse and Order Picking Systems.* Springer Berlin Heidelberg. ISBN: 9783540352204.

How to set up the metasfresh stack using Docker?. (N.d.). Metasfresh Docs. [online] Available at: http://docs.metasfresh.org/installation_collection/EN/How_do_I_setup_the_metasfresh_stack_using_Docker.html [Accessed Nov. 8, 2018].

Installation. (N.d.). OpenBoxes Docs. [online] Available at: http://docs.openboxes.com/en/develop/installation/ [Accessed Nov. 8, 2018].

Jest. (N.d.). Jest. [online] Available at: https://jestjs.io/.

Jones, D. E. and B. Jugl (2018). *Demo and Test Setup Guide.* OFBiz Project Open Wiki. [online] Available at: https://cwiki.apache.org/confluence/display/OFBIZ/Demo+and+Test+Setup+Guide [Accessed Nov. 8, 2018].

Lei, K., Y. Ma, and Z. Tan (2014). "Performance comparison and evaluation of web development technologies in php, python, and node. js". In: *2014 IEEE 17th International Conference on Computational Science and Engineering (CSE)*. IEEE, pp. 661–668.

OFBiz Features. (2016). OFBiz Project Open Wiki. [online] Available at: https://cwiki.apache.org/confluence/display/OFBIZ/OFBiz+Features [Accessed Nov. 8, 2018].

Produkt Highlights. (N.d.). Metasfresh. [online] Available at: https://metasfresh.com/en/erp-product-highlights/ [Accessed Nov. 8, 2018].

REST API Guide. (N.d.). OpenBoxes Docs. [online] Available at: http://docs.openboxes.com/en/develop/api-guide/ [Accessed Nov. 8, 2018].

Rodriguez, C., M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, and G. Percannella (2016). "REST APIs: a large-scale analysis of compliance with principles and best practices". In: *International Conference on Web Engineering*. Springer, pp. 21–39.

Schneider, G. and J. P. Winters (2001). "Applying use cases: a practical guide". In: Pearson Education. Chap. Getting started, p. 31.

Setup Authorization Token for accessing REST API. (N.d.). Metasfresh Docs. [online] Available at: http://docs.metasfresh.org/setup/Setup_REST_API_Token_Authorization.html [Accessed Nov. 8, 2018].

Statusmodell. (N.d.). Nuclos Wiki. [online] Available at: https://wiki.nuclos.de/display/Konfiguration/Statusmodell [Accessed Nov. 8, 2018].

WebUI Howtos and Tutorials. (N.d.). Metasfresh Docs. [online] Available at: http://docs.metasfresh.org/pages/webui/index_en [Accessed Nov. 8, 2018].

A Use Cases

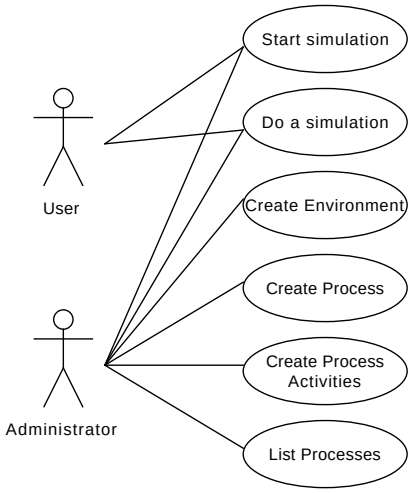

Figure 25: Use Case Diagram

Use Case	**Create environment**
Code	UC-1
Requirement	FR-12
Actor	Administrator
Description	The administrator creates an environment.
Precondition(s)	• The user is authenticated and has the administrator role.
Scenario	1. User opens the administration interface in the frontend. 2. User navigates to the environment administration page. 3. User clicks on add environment 4. User fills in name of environment 5. User saves environment
Extension	
Result	Environment is added to the system

Table 8: Use Case: Create environment

Use Case	**List processes**
Code	UC-2
Requirement	FR-1
Actor	Administrator
Description	The administrator lists existing processes.
Precondition(s)	• The user is authenticated and has the administrator role.
Scenario	1. User opens the administration interface in the frontend. 2. User navigates to the process administration page.
Extension	
Result	System lists existing processes.

Table 9: Use Case: List processes

Use Case	**Create process**
Code	UC-3
Requirement	FR-1
Actor	Administrator
Description	The administrator creates a demo process in the demo environment.
Precondition(s)	• The user is authenticated and has the administrator role.
Scenario	1. User opens the administration interface in the frontend. 2. User navigates to the process administration page. 3. User clicks on ”Add process” 4. User enters information in the provided form. 5. User saves process. 6. System checks if data is valid.
Extension	4. User imports data via xls/csv file.
Result	Process is added to the system.

Table 10: Use Case: Create process

Use Case	**Create Process Activity**
Code	UC-4
Requirement	FR-1
Actor	Administrator
Description	The administrator creates activities for a process.
Precondition(s)	• The user is authenticated and has the administrator role • At least one process exists.
Scenario	1. User opens the administration interface in the frontend. 2. User navigates to the process administration page. 3. User opens list of existing processes. 4. User opens one of the processes. 5. User clicks ”Create activity”. 6. User chooses type of activity. 7. User enters data. 8. User saves activity. 9. System checks if provided data is valid.
Extension	
Result	Process activity is created.

Table 11: Use Case: Create process activity

Use Case	**Start simulation (order-based**
Code	UC-5
Requirement	FR-7, FR-13, FR-14
Actor	User
Description	The user starts an order-based simulation.
Precondition(s)	• At least one process exists. • At least one order exists.
Scenario	1. User opens the simulation interface in the frontend. 2. User navigates to the simulation page. 3. User chooses process and order to work on. 4. User clicks ”start simulation”.
Extension	
Result	Simulation based on order and process is created.

Table 12: Use Case: Start simulation

Use Case	**Do a simulation**
Code	UC-6
Requirement	FR-7, FR-13, FR-14
Actor	User
Description	User does a simulation.
Precondition(s)	• Simulation for process (and order) is started
Scenario	1. System prompts for action by the user. 2. User does action (e.g. confirming step, providing input) 3. System prompts for next action by the user. 4. Steps 1 to 3 are repeated till all actions are done.
Extension	
Result	Process simulation is finished.

Table 13: Use Case: Do a Simulation

B Software Requirements Specifications

Software Requirements Specifications

LOGWear

Author: Marius Freyer

Version: 0.7

Venlo, November 20, 2018

Contents

Version control

Version	Date	Changes
0.1	26/09/2018	Initial version
0.2	27/09/2018	Implemented more details after second requirements meeting with customer, added several requirements
0.3	03/10/2018	Added requirements
0.4	04/10/2018	Added requirements, modifíed structure
0.5	12/10/2018	Added functional requirements
0.6	14/11/2018	Improved structure of specific requirements
0.7	20/11/2018	Deleted obsolete functional requirements

1 Introduction

1.1 Purpose

This software requirements specification document describes all requirements of the demo facility of the LOGwear project. It is intended to be read and agreed on by all involved project members and the customer of the demo facility.

1.2 Project scope

This document describes the requirements for the whole demo facility. This includes requirements on the software system mimicking a warehouse management system, required software for connecting different wearables to the system as well as the presentation layer of the demo facility (visualization, frontend, hardware, etc.).

1.3 Overview

For the structure this document follows IEEE Std. 830-1998 for writing software requirement specification.

The second chapter gives a more general overview over the demo facility while the third chapter lists the specific functional and non-functional requirements of the demo facility.

2 General Description

2.1 Product Perspective

The demo facility is part of the LOGwear project which also consists of a knowledge base and pilot projects with wearables at various companies. The purpose of the demo facility is giving companies an environment where they can test different wearables in different logistics processes.

The product consists of three parts: The software system which holds the logic of the processes and is responsible for the visual presentation of the demo processes as well as holding the data required for these processes (in an offline available database), the physical hardware workstation (PC, mouse, keyboard, etc.) and the wearables. Each of the three system parts must interact/communicate with each other.

2.2 Product Functions

The demo facility will be used to give first hand experience to companies that are interested in usage of wearables in their logistics processes.

The demo facility will provide a software based system that represents a basic warehouse management system supporting various common processes in the logistics department of small and medium sized companies. Moreover the demo facility consists of a physical environment featuring two stationary workstation, one mobile workstation and a small demo warehouse in form of boxes and shelves supporting the processes.

2.3 User Characteristics

There are three different kinds of users. The administrators of the demo facility. They will be responsible for configuring and implementing the software system, selecting and setting up the hardware and developing and maintaining the interfaces of the wearables.

The second group are the users who use the demo facility for giving presentations and workshops. These users are able to modify the data in the system, start the demo facility and reset it to the original state after a workshop or presentation is finished.

The third group are the users following workshops using the demo facility. These people are only allowed to use the presentation layer of the demo facility and are not permitted to

modify data except by following predefined processes.

2.4 General Constraints

The demo facility will mostly be used at Fontys in Venlo but should also be partly portable for demo purposes at companies. In both cases the demo facility will have no fixed place, it should not be dependent on external resources except power. Therefore the demo facility shall be fully usable without an internet connection and the software element of the demo facility shall be easily deployable on other computers.

2.5 Assumptions & Dependencies

2.5.1 Assumptions

- The latest available stable version of the operation system is used
- Administrators and users are used to computers

2.5.2 Dependencies

- Access to electricity (230 V/50 Hz AC)

3 Specific Requirements

3.1 Functional Requirements

Identifier	Priority	Description
FR-1	‹Shall›	The frontend shall provide a way to add processes to the system and modify processes of the system.
FR-1.1	‹Shall›	The frontend shall provide a way to import processes via xls/xlsx files to the system.
FR-1.2	‹Shall›	The frontend shall provide a way to import processes via csv files to the system.
FR-2	‹Shall›	The system shall provide a view of what the users see through the wearables (e.g. Hololens)
FR-3	‹Shall›	The backend shall provide an interface for external access (e.g. API).
FR-4	‹Shall›	The frontend shall provide an overview about the current process and its state visually
FR-5	‹Shall›	The frontend shall provide a way to choose between multiple wearables.
FR-6	‹Should›	The frontend should be able to communicate with multiple wearables at the same time.
FR-7	‹Shall›	The frontend shall provide an option to select the process going to work on.
FR-8	‹Shall›	The frontend shall provide an option to select wearable(s) for the process going to work on.
FR-11	‹Shall›	The system shall provide a way to sign in users.
FR-12	‹Shall›	The system shall provide a way use company specific data in the implemented processes.
FR-13	‹Shall›	The system shall implement the order picking process.
FR-14	‹Shall›	The system shall implement the warehousing process.

3.2 Non-functional Requirements

3.2.1 External Interface Requirements

Identifier	Priority	Description
IR-1	‹Shall›	The client shall be able to communicate via bluetooth in version 4.0+.
IR-1.1	‹Shall›	Communication via bluetooth shall be possible over a range of at least 5 meters.
IR-2	‹Shall›	The client shall communicate via WIFI (IEEE 802.11ac).
IR-2.1	‹Shall›	Communication via WIFI shall be possible over a range of at least 20 meters.
IR-3	‹Shall›	The client shall have USB interfaces (USB 3.1)

3.2.2 Performance Requirements

Identifier	Priority	Description
PR-1	‹Shall›	Requests to the backend shall be answered within maximum one second.
PR-2	‹Shall›	The demo facility shall be usable by up to two users at the same time.

3.2.3 Security Requirements

Identifier	Priority	Description
SR-1	‹Shall›	The demo facility shall define different user roles.
SR-1.1	‹Shall›	The demo facility shall provide a user role administrator that is allowed to change processes and data.
SR-1.2	‹Shall›	The demo facility shall provide a user role user that is allowed to use the predefined processes.

3.2.4 Maintenance Requirements

Identifier	Priority	Description
MR-1	‹Shall›	The data in the system shall be configurable by users with the administrator role without an IT background.
MR-2	‹Should›	The processes in the system should be configurable by users with the administrator role without an IT background.
MR-3	‹Shall›	The data in the system shall be resettable to the origin state by all users without an IT background.
MR-4	‹Shall›	The demo facility shall be start-able/runnable by users with administrator role without an IT background.
MR-5	‹Shall›	There shall always be a production-ready version of the backend and frontend which can be deployed on any machine as a backup solution.
MR-5.1	‹Shall›	The production-ready version shall be available securely online (e.g. in a repository).

3.2.5 Design Constraints

Identifier	Priority	Description
DC-1	‹Shall›	The demo facility shall be fully functional without requiring an internet connection/offline.
DC-2	‹Shall›	The components of the demo facility shall be connected via an own local network.
DC-3	‹Shall›	The frontend shall run on macOS 10.14 and newer.
DC-4	‹Shall›	The frontend shall run on Windows 10 version 1803 and newer.
DC-5	‹Should›	The frontend should run on Ubuntu 18.04 and newer.
DC-6	‹Shall›	The backend shall run on macOS 10.14 and newer.
DC-7	‹Shall›	The backend shall run on Windows 10 version 1803 and newer.
DC-8	‹Should›	The backend should run on Ubuntu 18.04 and newer.
DC-9	‹Shall›	The demo facility shall consist of a stationary environment.
DC-9.1	‹Shall›	The stationary demo facility shall consists of two independent clients.
DC-9.2	‹Shall›	The two clients shall have big high-quality and high-resolution screens as they shall be used to display a live stream of what users see through wearables in real-time.
DC-9.3	‹Shall›	The two clients shall be usable for (software) development for the demo facility (this includes also development with the Unity Engine for the Microsoft Hololens)
DC-9.4	‹Shall›	The two clients should connect to the same software system.
DC-10	‹Shall›	The demo facility shall consist of a portable environment.
DC-10.1	‹Shall›	The portable environment shall consist of a notebook running both backend and client parts of the application.
DC-10.2	‹Shall›	The notebook of the portable environment shall also be usable for development (this includes also development with the Unity Engine for the Microsoft Hololens)
DC-10.3	‹Shall›	The portable environment shall consist of a low range beamer for visual presentation.
DC-10.4	‹Should›	The portable environment should consist of miniaturized items (easily portable small shelves/boxes, etc.)
DC-11	‹Shall›	The demo facility shall have as less wired connections as possible

3.2.6 Software System Attributes

Identifier	Priority	Description
AR-1	‹Shall›	The demo facility shall be ready to be started at any time without requiring setup and configuration.

3.2.7 Other Requirements

Identifier	Priority	Description
OR-1	‹Shall›	The demo facility shall be technically documented.
OR-1.1	‹Shall›	The demo facility shall have a configuration management plan including all external dependencies.
OR-1.2	‹Shall›	The demo facility shall have a handover document describing how further development can to be done.
OR-4	‹Shall›	The demo facility shall have an enduser documentation.
OR-5	‹Should›	The mobile demo facility should be transportable in one bag.
OR-6	‹Shall›	The maximum price of the demo facility solution shall not exceed 20 000 €.

C Mockups

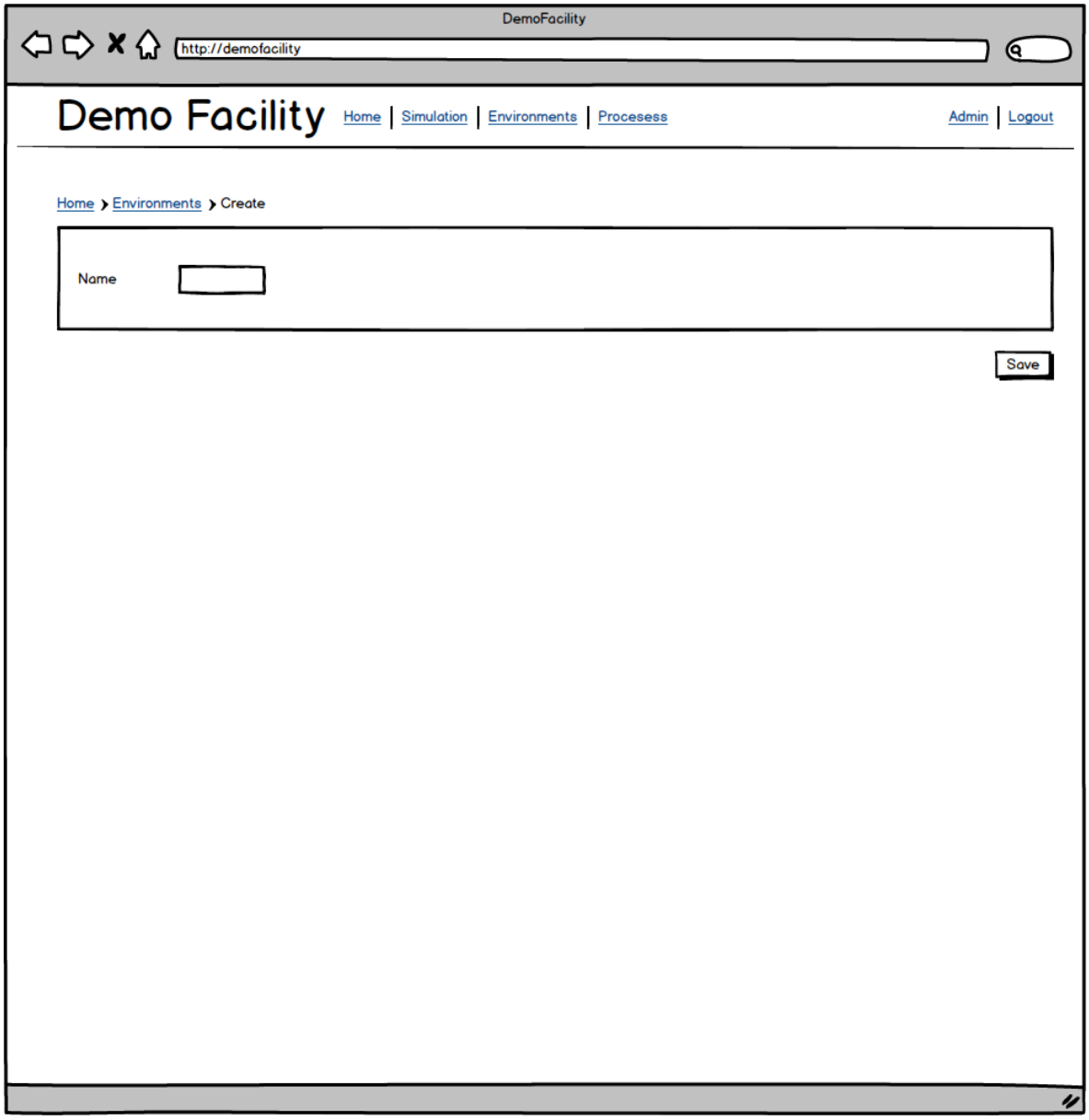

Figure 26: Mockup: Create Environment

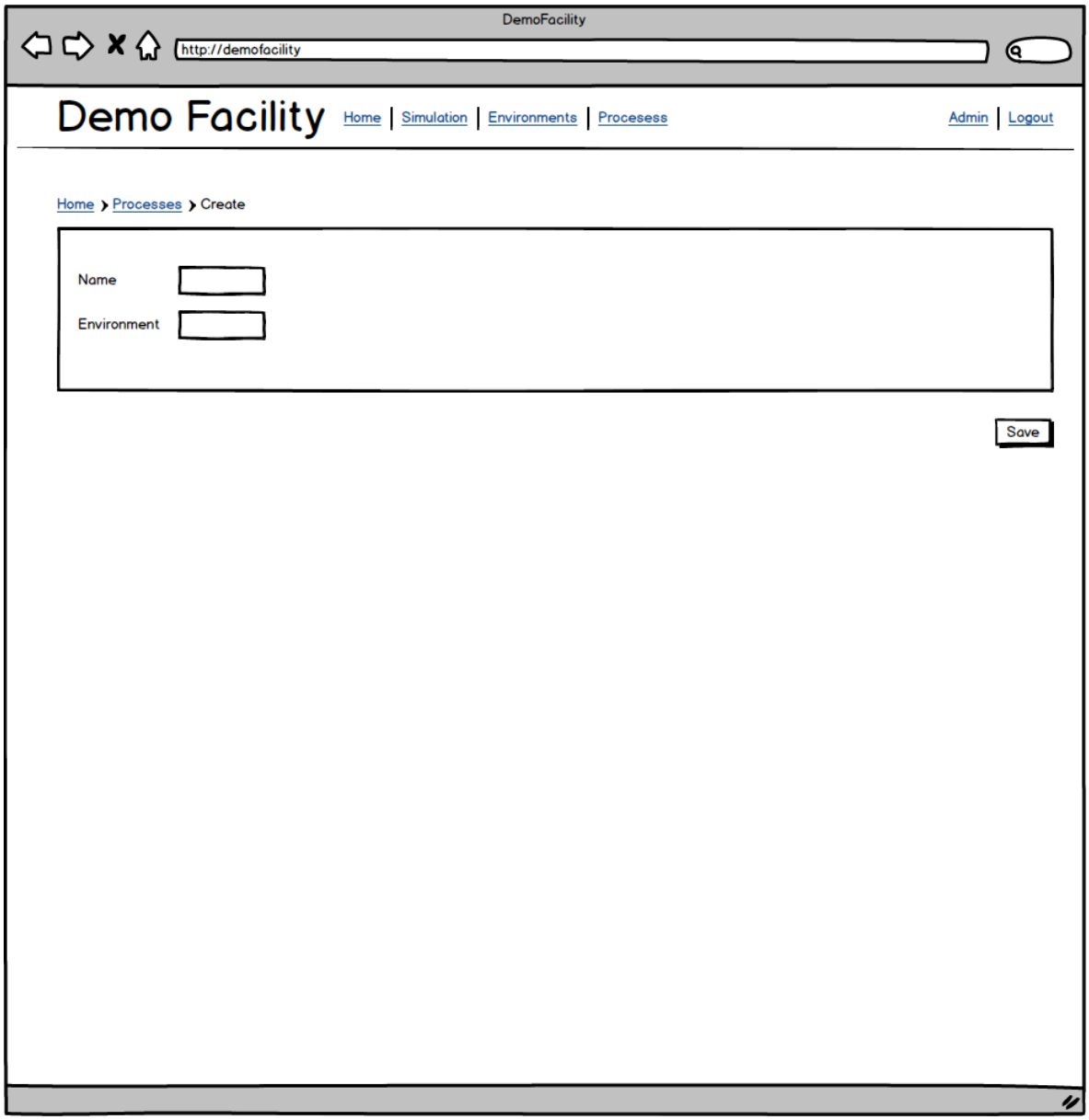

Figure 27: Mockup: Create Process

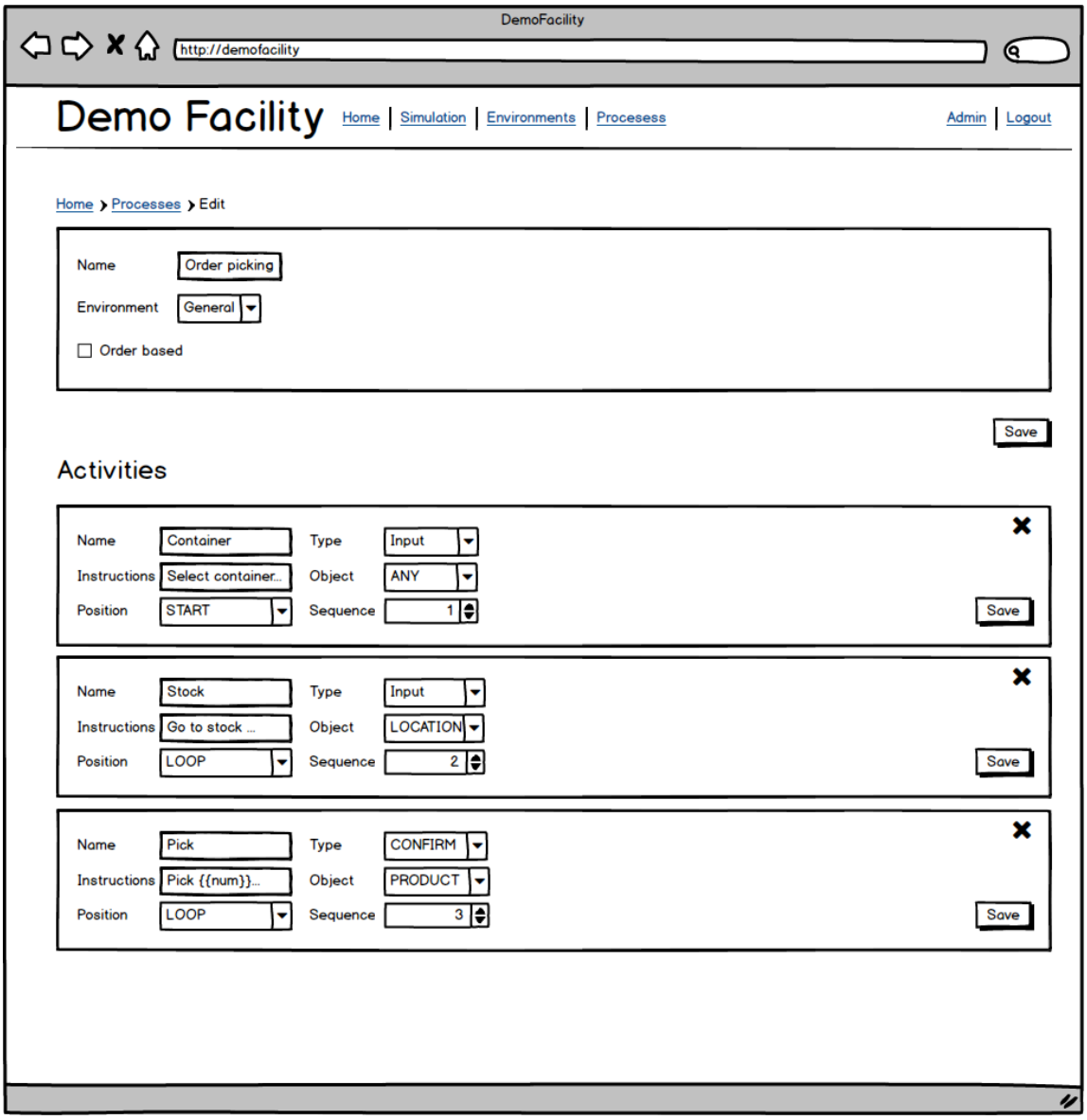

Figure 28: Mockup: Edit Process/Create Process Activities

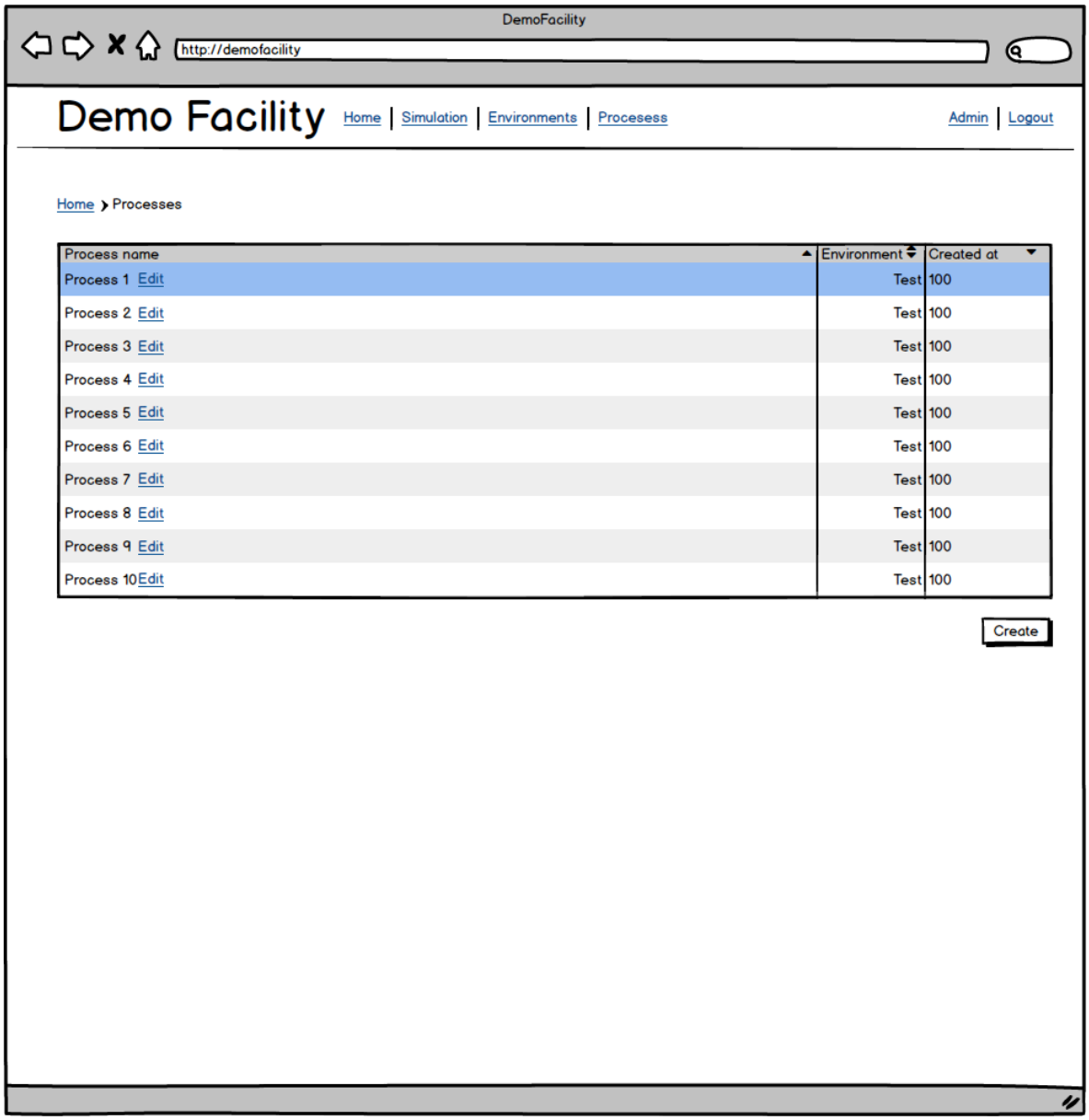

Figure 29: Mockup: List Processes

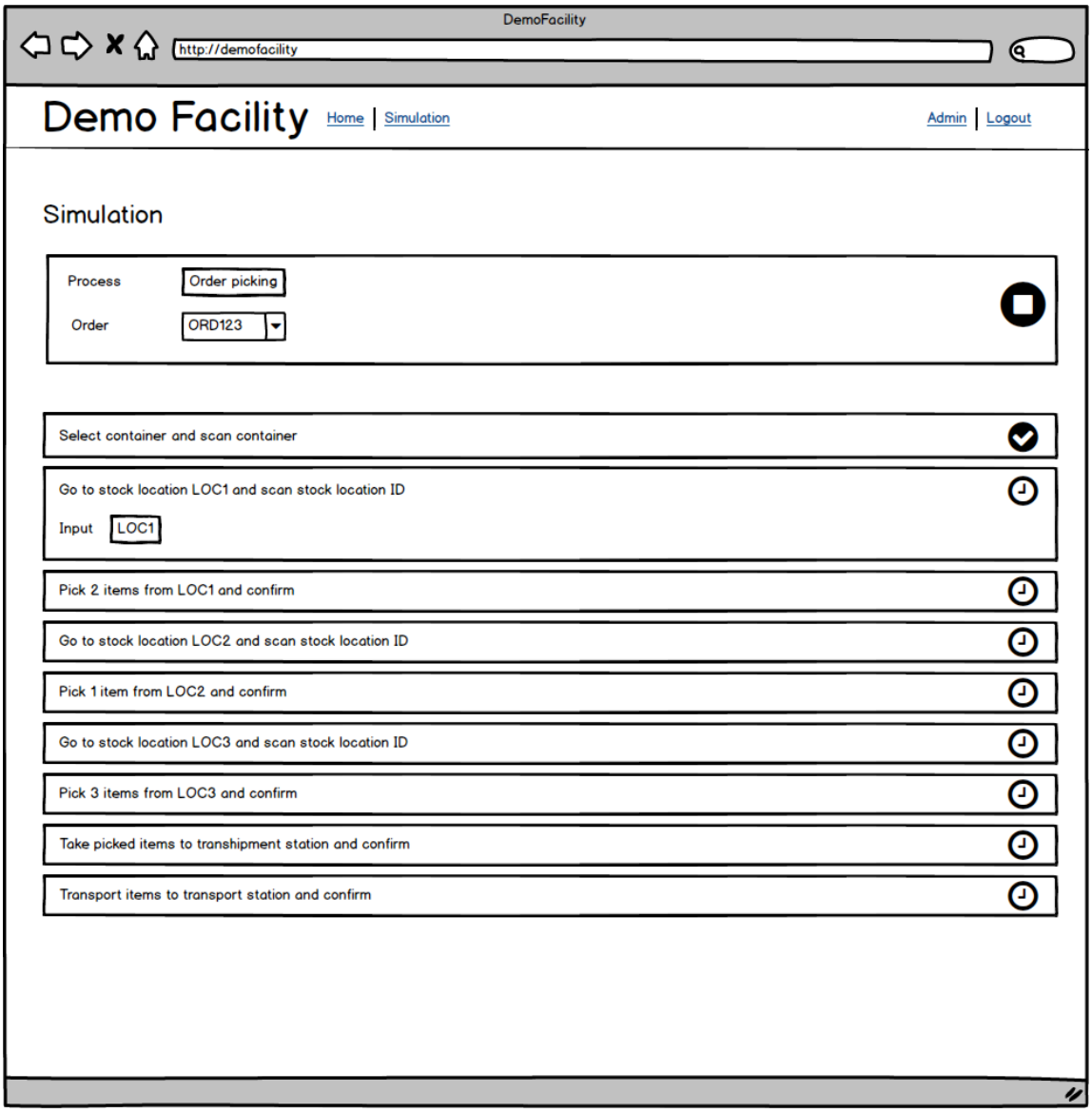

Figure 30: Mockup: Simulation View

D Logistics Processes

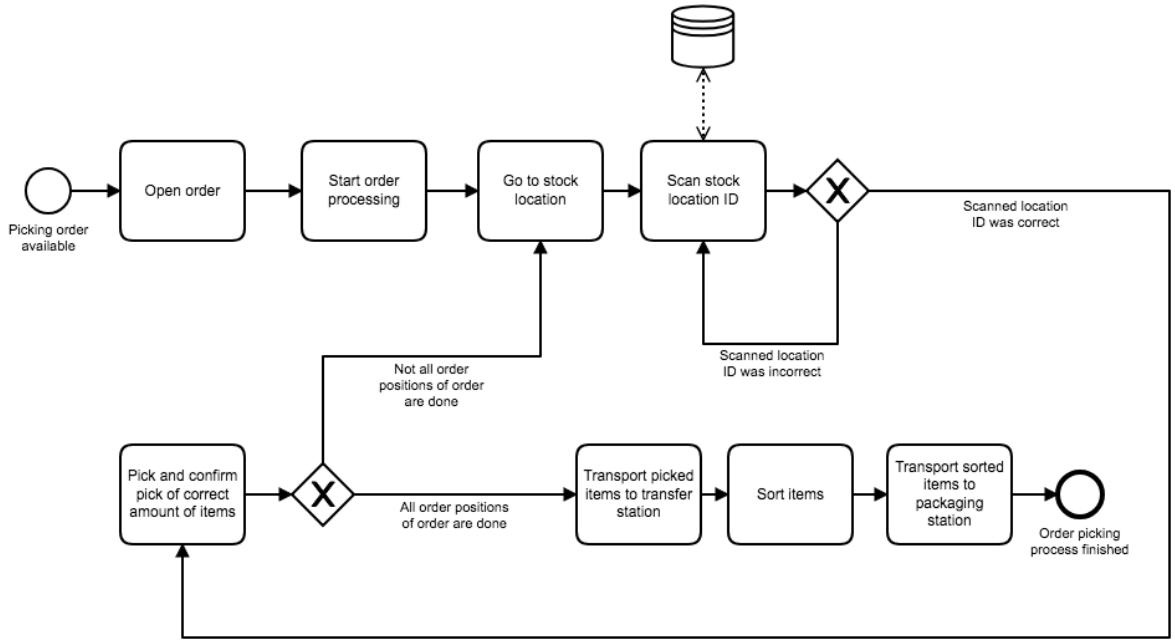

Figure 31: Process: Order Picking

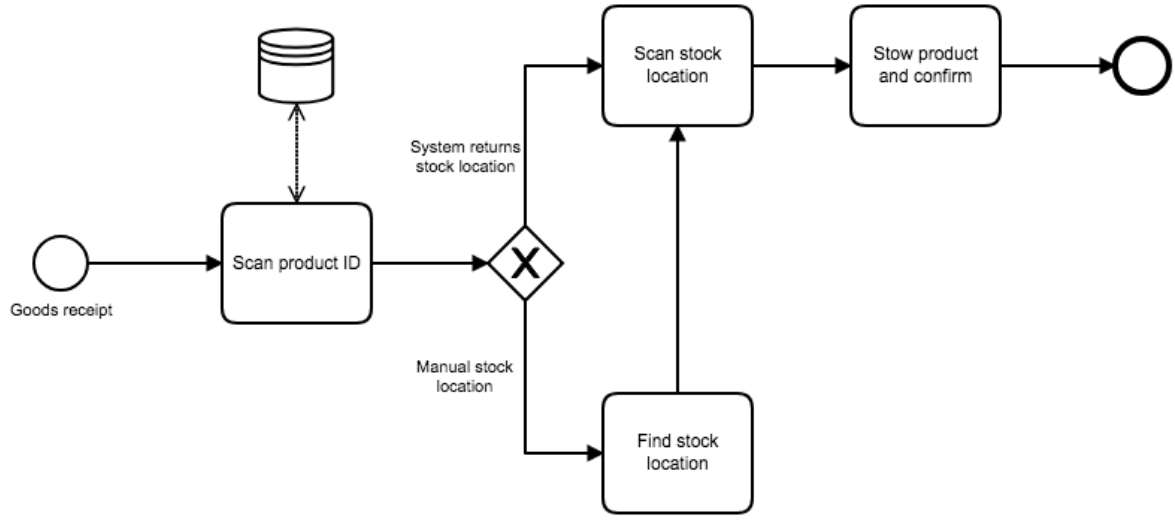

Figure 32: Process: Putaway

E Software Design

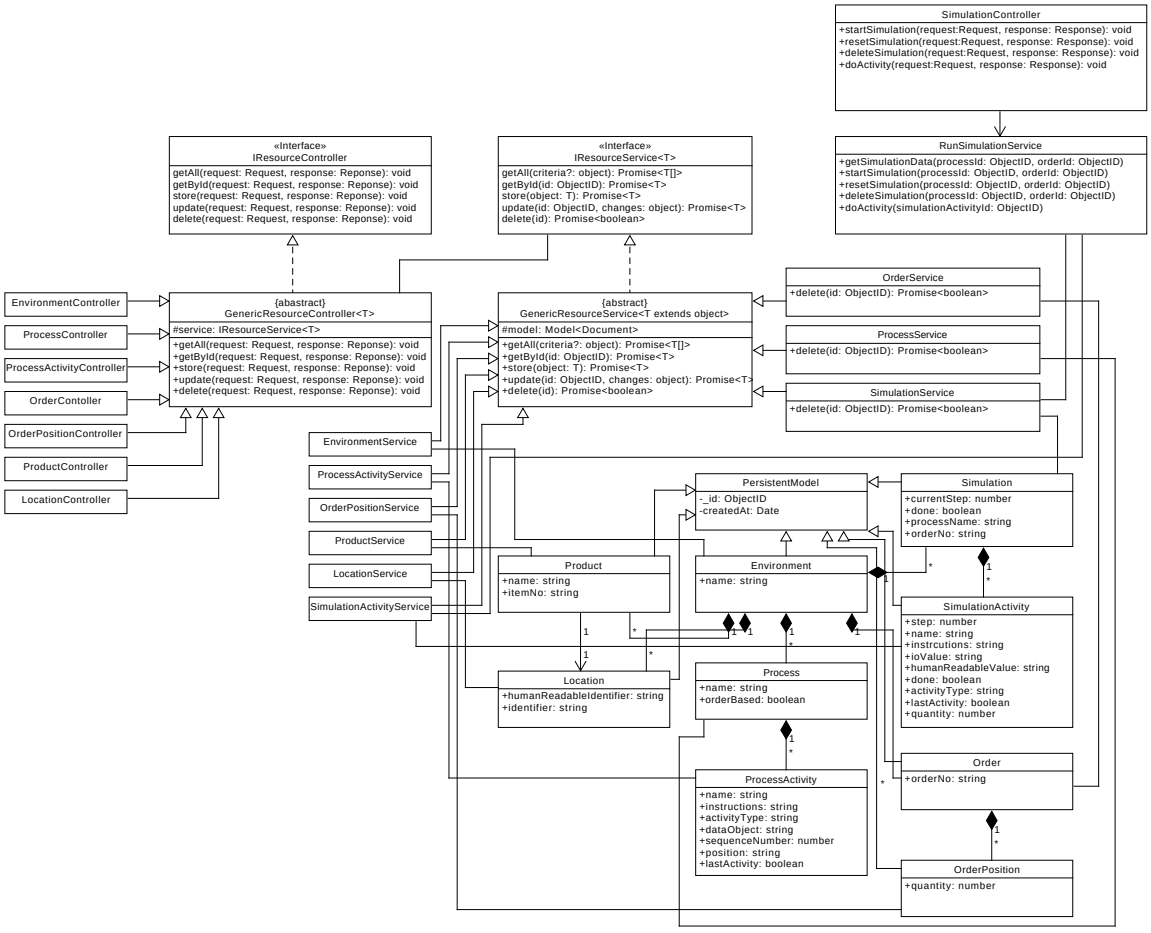

Figure 33: Class Diagram: Software design

F Research on Warehouse Management Systems

F.1 Purpose of Research

The demo facility will consist of a hardware and software part. For the software part a solution has to be found that is capable of fulfilling the specified requirements und use cases and in its nature acting similarly to a warehouse management system.

This research is aimed to find such an existing software solution that can be used alongside the demo facility.

F.2 Research Method

The research is based on the weighted scoring method. For this method multiple criteria for the software solution will be derived from the SRS. Each of these criteria is then assigned with a certain weight that represents the priority of the criterion. Then for each candidate the different criteria is evaluated and assigned with a score between 0 and a maximum of 3 points.

F.2.1 Weight

In the beginning each criterion is assigned with a weight between 1 and 3, where 3 represents a high importance. To differentiate even more between the highly important criteria, each of these criteria is again assigned with a weight of 1 to 3. For the final weight the two scores of the criteria with initially high weights are multiplied. With this approach each criterion can have a weight between 1 and 9, where 1, 2, 3, 6 and 9 are possible values.

Weight 1: Low priority

Weight 2: Medium priority

Weight 3: High priority

Weight 6: Very high priority

Weight 9: Very, very high priority

F.2.2 Scoring

To calculate the final score of each candidate, the weight has to be multiplied with a specific value the candidate scores within a specific criterion. Therefore, the candidate can score 0 to 3 points for each criterion. In case the criterion is ”yes/no”, no scores 0 points and yes 3 points to make the end result more comparable.

F.3 Criteria

Based on the requirements (see xyz) and the use cases (see xyz) certain criteria are derived against the different software solutions are assessed. Each criterion has a specific weight that represents each criteria's importance and a score from 0-3 which represents how well the software solution performs in the criterion. Below each criterion is described, the source from where the criteria was derived is given and the weight and possible scores are defined.

F.3.1 Offline usage

The software solution must be fully useable for the purpose of the demo facility without requiring an internet connection while use. It does not matter whether the software needs internet connection for initial setup. This criterion is specified in the SRS as DC-1.

As this criterion is very important it is considered as a KO criterion and therefore only used to filter candidates of the research. In the final assessment among the candidates, this criterion is considered as given as other candidates were filtered out regarding this criterion.

F.3.2 Price

The software solution must be available for free. No initial costs or recurring costs are allowed. This criterion was defined in consultation with the customer as budget is only available for the hardware part of the demo facility.

This criterion is as highly important as the offline usage capability and is therefore considered as a KO criterion for filtering suitable candidates.

F.3.3 Process activity support

The software solution must at least support the order picking and warehousing process either by already implementing these processes out of the box as defined by the LOGwear project or by providing a way to implement those processes or modifying those processes. This criterion was derived from FR-1 and in consultation with the customer who defined the importance of various processes.

The weight of this criterion is very high as it represents the basic functionality of the system and has a weight of 9.

F.3.3.1 Scoring

0 Points: No process is implemented out of the box and no way to implement these processes.

1 Point: One process or both processes are (partly) implemented but there is no way to implement the other process or improve/modify the existing processes.

2 Points:
1. Both processes are partly implemented, and processes can be modified/implemented.
2. The system provides a way to implement processes.
3. Both processes are fully implemented but the software does not provide a way to modify/implement processes.

3 Points: Both processes are fully implemented, and the software provides a way to implement/modify processes.

F.3.4 Input/Output matching

The software solution is going to be used at warehouses of potential customers. Therefore, the software solution must provide a way to adjust the data in the system to represent the data of the customer. As this is one of the main purposes of the demo facility, this criterion has a weight of 9. This criterion is derived from FR-12 of the SRS.

F.3.4.1 Scoring

0 Points: Data cannot be adjusted to represent the customer's data or the duration of the task to do so is expected to take longer than 8 hours.

1 Point: Data can be adjusted to represent the customer's data. The duration of this task is expected to take shorter than 8 hours.

2 Point: Data can be adjusted to represent the customer's data. The duration of this task is expected to take shorter than 4 hours.

3 Points: Data can be adjusted to represent the customer's data. The duration of this task is expected to take shorter than 1 hour.

F.3.5 Extendable

The software solution must provide a way to connect external applications on the wearables to the system. Therefore, it should either provide an API which allows interaction with the

implemented processes and data and/or some kind of SDK which allows direct modifications to the system to make the connection possible.

This criterion is crucial for connecting a wide range of different wearables to the system and so has a high weight of 6. This criterion is derived from FR-3 of the SRS.

F.3.5.1 Scoring

0 Points: The software neither provides an API nor an SDK.

1 Point: The software either provides an API or an SDK.

2 Points: The software provides an API and SDK.

3 Points: The software provides an API and SDK and documentation is considerably good.

F.3.6 Simplicity of usage

The system is going to be used in session of a maximum length of a few hours and so needs to be simple to use. As simplicity is crucial for a good presentation of the wearables, this criterion has a high weight of 6. This criterion is derived from MR-3 and MR-4 of the SRS.

F.3.6.1 Scoring

0 Points: The system needs extensive training (1 day and more) until an end user can do basics.

1 Point: The system needs some training (1 hour and more) until an end user can do basics.

2 Points: The system needs little training (30 minutes and more) until an end user can do basics.

3 Points: The system is self-explanatory or requires very little training (0-30 minutes) until an end user can do basics.

F.3.7 Setup

The setup criterion splits into the effort required for the initial setup of the system and the effort of the recurring startup setup for regular use of the demo facility. While a moderate effort for the initial setup is acceptable, the usual startup setup for the demo facility needs to be short. This criterion has a weight of 3. This criterion is derived from MR-1 and MR-2 of the SRS.

F.3.7.1 Scoring

0 Points: The initial setup takes more than one week, and/or the recurring setup takes more than 30 minutes.

1 Point: The initial setup takes less than one week, and the recurring setup takes less than 30 minutes.

2 Points: The initial setup takes less than one week, and the recurring setup takes less than 15 minutes.

3 Points: The initial setup takes less than two days and the recurring setup takes less than 15 minutes.

F.3.8 Platform support

The software solution has to support one of the operating systems that can run natively on an iMac Pro or MacBook Pro, namely macOS and Windows. Support for Linux is considered as a plus. This criterion is derived from DC-3 to DC-8 of the SRS. This criterion has a weight of 3.

F.3.8.1 Scoring

0 Points: Runs neither on macOS nor Windows.

1 Point: Runs either on macOS or Windows.

2 Points: Runs on macOS and Windows.

3 Points: Runs on macOS, Windows and Linux.

F.3.9 Multiple clients

The software solution should be able to serve multiple clients, so it is possible to use the same data and process implementation running on a backend at multiple workstations.

This feature is nice-to-have but not required and has a weight of 1. This criterion is derived from DC-2 and DC-9.4 of the SRS.

0 Points: Does not support multiple clients.

3 Points: Supports multiple clients.

F.4 Candidates

F.4.1 Selection of candidates

Due to time constraints the period for selection of candidates only lasted two days on October 8 and 9. Sources for the candidates were the book "Warehouse Management: Automation and Organisation of Warehouse and Order Picking Systems" by M. Hompel and T. Schmidt (2007), the Warehouse Management System list of Software Advice [3] and of ExplorWMS [4].

Apart from these candidates also the KLG Pilot was taken into consideration as it already provides a WMS-like system.

F.4.2 Long list

The following list of candidates were selected. Each candidate is assessed against the two KO criteria price and offline-capability as described in the upper section. If a KO criterion is applied to the candidate, the right column shows which one applied.

Name	Website	KO
Apache OFBiz	https://ofbiz.apache.org	-
OpenBoxes	https://openboxes.com	-
Odoo	https://odoo.com	Price[5]
xTuple	https://xtuple.com	Price[6]
myWMS LOS	https://mywms.org	Price[7]
Delivrd	https://delivrd.com	Price[8]
Metasfresh	https://metasfresh.com	-
Finale Inventory	https://finaleinventory.com	Price
Flowtrac	https://flowtrac.com	Price
Sweet	https://getsweet.co	Price
VIN eRetail	http://vinculumgroup.com	Price
ZOHO Inventory	https://zoho.eu	Price
Clarus WMS	https://www.claruswms.co.uk	Price

Continued on next page

[3] https://www.softwareadvice.com/scm/warehouse-management-system-comparison/
[4] https://www.explorewms.com/open-source-wms-buyers-guide.html
[5] WMS functionality requires non-free addons
[6] WMS functionality requires non-free license
[7] Commercial use requires a non-free license
[8] Free version too limited

Name	Website	KO
WAMA	https://www.wama.cloud	Offline usuable
Latitude WMS	https://pathguide.com	Price
Logiwa WMS	https://www.logiwa.com	Price
Carton Cloud	https://cartoncloud.com	Price, Offine ususable
ProVision	http://provisionwms.com	Price
Optimiser WMS	http://www.optima-ws.co.uk	Price
Inventory Pro	http://cissltd.com	Price
Logimax	http://e-logimax.com	Price
ShipEdge	https://shipedge.com	Price, Offline usuable
Raptool	https://raptool.com	Price, Offline usuable
StockOne	https://stockone.in	Price
Veeqo	https://veeqo.com	Price
Nuclos	http://nuclos.de	-
KLG Demo	https://logwear.eu	Offline usuable

Table 14: Research candidates (longlist)

F.4.3 Short list

After applying the KO criteria to each candidate (also refer to table 14) the following candidates will be further assessed.

Name	Website	KO
Apache OFBiz	https://ofbiz.apache.org	-
OpenBoxes	https://openboxes.com	-
Metasfresh	https://metasfresh.com	-
Nuclos	http://nuclos.de	-

Table 15: Research candidates (shortlist)

F.5 Assessment

In the following section the remaining candidates (refer to table 15) are assessed against the criteria gathered in section F.3. The KO criteria are left out in this section as they were already applied to come up with the shortlist of candidates.

F.5.1 Apache OFBiz

F.5.1.1 Process activity support

Apache OFBiz has basic support for the picking process in terms of providing a pick list and support receiving of items. (*OFBiz Features*, 2016) Modifying these processes is intended by the project. So OFBiz scores **1 point** for this criterion.

F.5.1.2 Input/Output matching

OFBiz supports stock management with custom items and defining a custom warehouse-/inventory structure. So the data of the system can be adjusted to match the environment of the customer. (*OFBiz Features*, 2016). According to the demo environment provided by OFBiz this tasks seems to be complex and time consuming. So OFBiz scores **0 points** for this criterion.

F.5.1.3 Extendable

Apache OFBiz does not provide an SDK nor an API. So it scores **0 points** for this criterion.

F.5.1.4 Simplicity of usage

Apache OFBiz offers a lot of features and is not just intended as a Warehouse Management System. (*OFBiz Features*, 2016) According to the demo environment the system seems to be very complex. So Apache OFBiz scores **0 points** for this criterion.

F.5.1.5 Setup

OFBiz provides prebuild releases that can be installed on every computer running the Java Runtime Engine. (Jones et al., 2018) So the initial setup is done in a few hours. Once installed the software can be used without further setup. So OFBiz scores **3 points** for this criteria.

F.5.1.6 Platform support

OFBiz is Java-based and runs on Windows, macOS and Linux. (Jones et al., 2018) So it scores **3 points**.

F.5.1.7 Multiple clients

OFBiz uses a web frontend sending requests to a backend.(Jones et al., 2018) So multiple clients are supported. OFBiz scores **3 points**.

F.5.2 OpenBoxes

F.5.2.1 Process activity support

OpenBoxes provides basic support for order picking in terms of providing a pick list and the warehousing process. (*Features*, n.d.) There is no information available regarding the possibility of modifying or adding processes to the system. So OpenBoxes scores **1 point** for this criterion.

F.5.2.2 Input/Output matching

Stock items and warehouse locations can be specified (*Features*, n.d.) but as OpenBoxes is designed for electronic items (*About*, n.d.) there might be some limitations or difficulties that an be time consuming. So OpenBoxes scores **1 points** for this criterion.

F.5.2.3 Extendable

OpenBoxes provides an RESTful API. (*REST API Guide*, n.d.). So it scores **1 point**.

F.5.2.4 Simplicity of usage

OpenBoxes is focused on Inventory Management (*OFBiz Features*, 2016) and is limited in terms of features. So the expected training duration is less than 30 minutes for basic handling. So OpenBoxes scores 2 **points** for this criterion.

F.5.2.5 Setup

OpenBoxes only provides an installation guide for an outdated version of Ubuntu (*Installation*, n.d.). So the initial setup is expected to take more than two days. The recurring

setup is expected to take less than 15 minutes as once installed no further setup seems to be required. So OpenBoxes scores **2 points** for this criterion.

F.5.2.6 Platform support

OpenBoxes only provides an installation guide for Ubuntu. (*Installation*, n.d.) macOS and Windows seem to be supported but there is not official guide. So OpenBoxes is limited to scoring **1 point** for this criterion.

F.5.2.7 Multiple clients

OpenBoxes uses a web frontend that sends requests to a backend. So multiple clients are supported and it scores **3 points** for this criterion.

F.5.3 Metasfresh

F.5.3.1 Process activity support

Metasfresh supports the order picking and warehousing process. The processes are modifiable inside limited boundaries. (*Produkt Highlights*, n.d.) So Metasfresh scores **2 points** for this criterion.

F.5.3.2 Input/Output matching

Items, warehouse location etc. can be changed to represent customer's data. (*WebUI Howtos and Tutorials*, n.d.) As the system offers a lot of features and seems to be rather complex, the duration of this task is expected to take more than 8 hours. So Metasfresh scores **0 points** for this criterion.

F.5.3.3 Extendable

Metasfresh provides a RESTful API (*Setup Authorization Token for accessing REST API*, n.d.) and so scores **1 point** for this criterion.

F.5.3.4 Simplicity of usage

Due to the number of features and the broad scope of the software, Metasfresh is expected to be rather complex and getting used to it will probably take some hours of training. So Metasfresh scores **1 point** for this criterion.

F.5.3.5 Setup

Metasfresh provides docker images for the various modules of the software and a setup guide. (*How to set up the metasfresh stack using Docker?*, n.d.) Initial setup so probably takes only a few minutes and the recurring setup only a few seconds. So Metasfresh scores **3 points** for this criterion.

F.5.3.6 Platform support

As Metasfresh provides a docker setup it can run on Windows, macOS and Linux. So it scores **3 points** for this criterion.

F.5.3.7 Multiple clients

Metasfresh provides a web frontend that connects to a backend. So it can serve multiple clients and scores **3 points** for this criterion.

F.5.4 Nuclos

F.5.4.1 Process activity support

Nuclos has no processes implemented out of the box, but provides tools to implement them. (*Statusmodell*, n.d.) So Nuclos scores **2 points** for this criterion.

F.5.4.2 Input/Output matching

As Nuclos is a toolbox for developing an ERP-like system there are no restrictions regarding data structure and data. However, implementing these data structures seems to be a time-consuming task. (*Businessobjekt*, n.d.; *Statusmodell*, n.d.) So Nuclos scores **0 points** for this criterion.

F.5.4.3 Extendable

Nuclos provides an SDK and RESTful API. However, documentation seems to be very limited. So it scores **2 points** for this criterion.

F.5.4.4 Simplicity of usage

As Nuclos is only a toolbox for creating a system, the simplicity of usage depends on how the system is developed on top of Nuclos. So no reliable score can be given. So Nuclos scores **0 points** for this criterion.

F.5.4.5 Setup

Nuclos provides a Java-based installer for Windows, macOS and Linux. So initial setup probably takes only a few minutes. Once installed, no recurring setup seems to be required. So Nuclos scores **3 Points** for this criterion.

F.5.4.6 Platform support

Nuclos is Java-based and runs on Windows, macOS and Linux. So it scores **3 Points** for this criterion.

F.5.4.7 Multiple clients

Nuclos has a backend client archtitectures and so supports multiple clients. So Nuclos scores **3 points**.

F.6 Result

F.6.1 Overview

Candidate/ Criteria	Weight	Max. Score	OFBiz	OpenBoxes	Metasfresh	Nuclos
Processes	9	3	9	9	18	18
I/O	9	3	0	9	0	0
Extendable	6	3	0	6	6	12
Simplicity	6	3	0	12	6	0
Setup	3	3	3	6	9	9
Platforms	3	3	3	3	9	9
Clients	1	3	3	3	3	3
Overall			**30**	**48**	**51**	**51**

Table 16: Research result

F.6.2 Conclusion

The research result (refer to table 16) shows that three of the candidates score either equally or almost equally and Apache OFBiz follows with a big margin. So OFBiz is no taken into consideration for further discussion of the results.

However, if you look closely, the three remaining candidates also score quite low points as they not even reach half of the possible 111 points. Especially taking into account the most important criteria Process/activity support and Input/Output matching none of the WMS in the comparison can score more than 9 of 27 points in average with Metasfresh and Nuclos even scoring 0 points in Input/Output matching.

Comparing the WMS with a self-made solution that is directly designed for the purpose of the Demo Facility, the required effort for a self-made solution seems to be lower than customizing one of the existing solutions. Especially in terms of the Process/Activity support and the Input/Output matching a custom-made software solution would be better as it would be designed to define own processes with custom data. Also in terms of being extendible a custom made solution is better as all design-related documentation including all source codes would be available. Also an external API can be fully documented to make it easy to use for other applications.