

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Aleksandr Ivanov 186093IADB

Product Delivery Accounting Solution for Manufacturing Enterprise

Bachelor's thesis

Supervisor: Nadežda Furs
MBA

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleksandr Ivanov 186093IADB

Toote kohalettoimetamise arvestuse lahendus tootmisettevõttele

Bakalaureusetöö

Juhendaja: Nadežda Furs
MBA

Tallinn 2021

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aleksandr Ivanov

27.04.2021

Abstract

The aim of this thesis is to solve the Novotrade Invest AS enterprise problem of accounting of manufactured production by developing an application. The application is designed to simplify the work of the logistics group employees in accounting and save time and money.

The solution is divided into two parts: the server application and the client application. In addition to the basic accounting functionality, the application should allow employees to view the order history and generate and export a report. During development, the author adhered to a modular structure, which will ensure the integrability of the application, as well as facilitate further development.

The developed application has made life easier for employees, and reduced the time, early use for monotonous work. Currently, the application has taken its place as a tool of the logistics group of the Novotrade Invest AS enterprise.

This thesis is written in English and is 41 pages long, including 6 chapters, 21 figures and 8 tables.

Annotatsioon

Toote kohaletoiemtamise arvestuse lahendus tootmisettevõttele

Käesoleva bakalaureusetöö eesmärk on lahendada Novotrade Invest AS ettevõtte probleem toodetud toodangu arvestuses rakenduse väljatöötamise kaudu. Rakendus on loodud selleks, et lihtsustada logistikagrupi töötajate tööd arvestuses ning säästa aega ja raha.

Lahendus on jagatud kaheks osaks: serverirakendus ja kliendirakendus. Lisaks arvestuse põhifunktsioonidele peaks rakendus võimaldama töötajatel vaadata tellimuste ajalugu ning koostada ja eksportida aruannet. Arenduse käigus pidas autor kinni modulaarsest struktuurist, mis tagab rakenduse integreeritavuse ja hõlbustab edasist arendamist.

Väljatöötatud rakendus on muutnud töötajate elu lihtsamaks ja on vähendanud aega, varajast kasutamist monotoonseks tööks. Praegu on rakendus oma koha võtnud Novotrade Invest AS ettevõtte logistikagrupi tööriistana.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 41 leheküljel, 6 peatükki, 21 joonist, 8 tabelit.

List of abbreviations and terms

API	Application Programming Interface
App	Same as “Application”
Application	Same as “Software”
Backend	Part of application that handles logic
BLL	Business logic layer
Browser	Program to browse the Internet
Cached	Something that saved in computer memory
CRUD	Create, Read, Update and Delete actions
DAL	Data access layer
Database context	Moment of database state, that application can use; bridge between application and database
Database migration	Process of syncing the database schema with application structure
DTO	Date Transfer Object
Endpoint	The entry point to a service, a process, or a queue
Entity	Class that corresponds to specific database table
Framework	Abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software
Frontend	Part of application that handles user interface
FURPS+	Method of collecting and classification of software requirements
GraphQL	Graph Query Language
HTML	The standard markup language for documents designed to be displayed in a web browser.
JSON	JavaScript Object Notation
JWT	Json Web Token
Layout	Placement of components in page
Mapping	Converting one object to another; linking two objects

MoSCoW	Method of prioritization of software requirements
MVC	Model-View-Controller
Online	Connected to the Internet, or internet-based
OOP	Object-Oriented Programming
OS	Operating system
Out-of-the-box	Features that came with software
PC	Personal Computer
PWA	Progressive Web Application
REST	Representational State Transfer
Route	URL path
Serialization	Process of converting objects to bytes
SOAP	Simple Object Access Protocol
Soft delete	Process of deleting, with saving data for future purpose
Soft update	Process of updating, with saving the history of versions
Software	Computer program, is a collection of instructions and data that tell a computer how to work
SOLID	Single Responsibility, Open–Closed, Liskov Substitution, Interface Segregation and Dependency Inversion principles
SPA	Single Page Application
SSR	Server-Side Rendering
Syntax	Program language rules
UI	User Interface
VBA	Visual Basic for Applications
XML	eXtensible Markup Language

Table of contents

1 Introduction	13
1.1 Description of the problem	13
1.2 Goal and main tasks	14
1.3 Starting conditions	14
1.4 Scope and role of author	15
2 Requirements and Existing Solutions	16
2.1 Requirements	16
2.2 Current solution	18
2.3 Existing solutions	19
2.3.1 1C: Enterprise	19
2.3.2 Analogues of 1C: Enterprise – Ananas	19
2.3.3 Goramp	20
2.3.4 UTECH	20
2.3.5 Low Code	20
2.3.6 Online boards	21
2.4 Conclusion	21
3 Analysis of technological aspects	22
3.1 Type of application	22
3.2 Program language and framework	23
3.2.1 JavaScript	24
3.2.2 C# and ASP.NET Core	25
3.2.3 Java and Spring	26
3.2.4 Python and Django	26
3.2.5 Framework and conclusion	26
3.3 Architecture	28
3.4 SOAP, REST and GraphQL	29
3.5 Database choice	30
3.6 Conclusion	30
4 Implementation	31

4.1 Database.....	31
4.2 Backend	33
4.2.1 Server app structure	33
4.2.2 Domain entities.....	34
4.2.3 Entity Framework database context	36
4.2.4 Database initialization	37
4.2.5 Data access layer	37
4.2.6 Business logic layer.....	39
4.2.7 Handling soft update entities	39
4.2.8 Querying of an actual data.....	40
4.2.9 API controllers.....	41
4.2.10 Security	41
4.2.11 Swagger	42
4.3 Frontend.....	42
4.3.1 Client application initialization	43
4.3.2 Client app structure.....	43
4.3.3 API calls	45
4.3.4 Vuex storage	46
4.3.5 Attribute types	46
4.3.6 Attributes	47
4.3.7 Order templates	48
4.3.8 Orders	48
4.3.9 Forms validation.....	50
4.3.10 Security.....	50
4.4 Testing	51
5 Further development.....	52
6 Summary.....	53
References	54
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis	58
Appendix 2 – ER Diagram (ERD).....	59
Appendix 3 – Data initialization.....	60
Appendix 4 – DAL Mapper.....	61
Appendix 5 – Units and Values add dialogs	62

Appendix 6 – CustomValueField component	64
Appendix 7 – Orders filtration dialog	68
Appendix 8 – Date picker component	72
Appendix 9 – Attribute and its value select components	77
Appendix 10 – Attribute type views.....	82
Appendix 11 – Attributes views	84
Appendix 12 – Templates views	85
Appendix 13 – Orders views	86
Appendix 14 – Users views	90
Appendix 15 – Example of generated report.....	91

List of figures

Figure 1. Most popular program languages by TIOBE index in March 2021 [24].....	23
Figure 2. Solution architecture	28
Figure 3. Order and attribute tables relationship	31
Figure 4. Complicated attribute type table relationship	32
Figure 5. Simplified attribute type table relationship	32
Figure 6. Template and attribute tables relationship	32
Figure 7. Server application structure.....	33
Figure 8. AttributeType entity example	35
Figure 9. Example of handling of deleted entities.....	36
Figure 10. Repository factory method.....	37
Figure 11. Repository creation using factory method	38
Figure 12. Entities update method with soft update handling	40
Figure 13. Example of actual data query	41
Figure 14. Default authentication and authorization scheme	42
Figure 15. Example of restricting access to controllers.....	42
Figure 16. Nuxt project initialization	43
Figure 17. Client application structure	44
Figure 18. Axios GET method wrapper with error handling	45
Figure 19. Nuxt unit of work plugin.....	45
Figure 20. Attribute type format display filter	47
Figure 21. Custom validation function example	50

List of tables

Table 1. FURPS+ requirements classification.....	16
Table 2. MoSCoW requirements prioritization	17
Table 3. Program languages comparison.....	24
Table 4. Frameworks comparison	27
Table 5. Frameworks and their purpose	27
Table 6. Domain entities.....	34
Table 7. Domain abstract entities	35
Table 8. API controllers.....	41

1 Introduction

This thesis describes the process of analyzing and developing an application for product delivery accounting as part of a practical task for the Novotrade Invest AS enterprise.

Novotrade Invest AS (abbreviated VNK) is an Estonian industrial organization engaged in the refining of petroleum products. The enterprise production process covers the unloading of raw materials, processing, as well as loading products into railway wagons and trucks. The products then are used in various industrial sectors, from solvents and fuels to metallurgy [1].

The enterprise recently set a goal to digitalize the work environment and started hiring developers to fulfill the goal.

1.1 Description of the problem

The company keeps records of shipments of manufactured products daily. Each order contains a certain, but not fixed, the number of fields (attributes), some of which are displayed in the table representing the days. Completed orders are marked with a specific color. Currently, Microsoft Excel software is used for this. Despite its versatility, at a certain stage, the use of this solution began to bring discomfort to employees.

Among the discomforts are the following: the need to install software for work; the large file size, which takes time and effort; the human factor, which cause consequences in the form of irrelevant data and wasted time for searching and fixing, etc. Also, employees needed additional functionality, for example, displaying data in a calendar view, which is achievable using Excel tools, but not so intuitively and automatic, takes additional time and requires advanced knowledge of use, which the staff does not have.

To solve these problems, the enterprise has decided to develop a highly focused application that will intuitively perform the assigned tasks and possess the required functionality.

1.2 Goal and main tasks

The main purpose of the thesis is to solve the problem of the enterprise by developing an application for the accounting of product delivery, focusing on achieving the required functionality. Since the development of a full-fledged application is a rather scrupulous process that requires the involvement of several developers, the author's goal is to develop the first version of the application – a prototype that will solve the enterprise problem and have the conditions for further development and improvement.

The first task is to determine the functionality of the application, as well as other requirements that the prototype must meet.

The second task to achieve this goal is to analyze and determine the optimal approach for developing an application, considering the possibility of further development and integration with other systems.

The next task is to develop an application prototype based on a requirements analysis and technical approaches. An accompanying task is the design and creation of a database for data storage and operation.

1.3 Starting conditions

The first condition is an enterprise server that runs on Windows Server 2012 R2; therefore, the determination of the approach should consider the compatibility of the application and server components.

The second condition is that the employees of the enterprise mainly use corporate PCs (Personal Computer) for their work.

The third condition is that the enterprise does not want to change the data structure, which means that the application must be able to operate the data in form that is currently in use.

The fourth condition is that the enterprise does not want additional costs, including the cost of purchasing software and support.

1.4 Scope and role of author

The author takes the place of an intern at the Novotrade Invest AS Enterprise and, as a practical assignment, develops an application for logistics group employees. The head of logistics has provided certain requirements that the application must meet. A detailed description and analysis of the requirements will be discussed in the analysis section (see 2.1 Requirements).

To achieve this goal, it is necessary to analyze the current solution and existing solutions on the market to determine the strengths that can be implemented and weaknesses that can be avoided in the developed application.

The author is going to analyze and prioritize application requirements to determine what requirements will be implemented in the application prototype and analyze technical approaches, including various software development practices and programming languages to determine the optimal approach. Then the author will develop a prototype of the application, test the basic functionality, and describe the further step of improvements. For better understanding, the text contains figures, most of which are made by the author himself.

The scope of the thesis does not include the development of the final version of the application, the development of the general system for integration, detailed documentation of the application, thorough testing, and deployment.

2 Requirements and Existing Solutions

In this chapter, the author classifies and prioritizes the requirements for the application being developed and analyzes existing solutions that can possibly solve the enterprise problem to determine the functionality that will be implemented in the prototype.

2.1 Requirements

The requirements were discussed with the head of logistics of Novotrade enterprise. Since different requirements cover various aspects of applications to organize these requirements, the author uses the FURPS+ classification method, because it provides both the classification of those aspects and separation of the functional and non-functional requirements (see Table 1) [2].

Table 1. FURPS+ requirements classification

Category	Requirements list
F – Functionality	F1. Completed orders can be marked. F2. Orders should be displayed in a calendar view. F3. The featured order attributes should appear on the calendar. F4. The rest of the order attributes should be displayed in an additional window. F5. Some orders have a non-fixed date. F6. Orders must have a markup field. F7. It is possible to generate reports for the selected period. F8. Some attribute values must be selected from the list.
U – Usability	U1. Main interface colors should be calm and not distracting. U2. The order information window should be next to the calendar. U3. The calendar should display orders for the month, for the week.
R – Reliability	R1. The system must have a user with the highest access level (root), which cannot be changed or deleted. R2. The system should be available from 8 am to 5 pm, Monday through Friday.
P – Performance	P1. Applications must support at least 3 simultaneous users.
S – Supportability	S1. The interface must be in Russian.

	<p>S2. The interface must support English and Estonian.</p> <p>S3. There should be a button in the interface to get help.</p> <p>S4. Ability to change database connection settings.</p>
“+”	<p>X1. Access is differentiated by role.</p> <p>X2. Only authorized users can access the application.</p> <p>X3. Users with the "User" access level can only view placed orders.</p>

The table does not indicate some requirements, for example CRUD (creation, deletion, etc.), since their presence is fundamental for solving the problem. The author has also removed the requirements “the interface must be intuitive” and similar due to their unmeasurable realizability and objectivity.

To determine the requirements that will be implemented in the prototype application, the author uses the MoSCoW prioritization method (see Table 2) [3].

Table 2. MoSCoW requirements prioritization

Category	Requirements list
MUST	<p>F1. Completed orders can be marked.</p> <p>F2. Orders should be displayed in a calendar view.</p> <p>F3. The featured order attributes should appear on the calendar.</p> <p>F4. The rest of the order attributes should be displayed in an additional window.</p> <p>F8. Some attribute values must be selected from the list.</p> <p>U1. Main interface colors should be calm and not distracting.</p> <p>U2. The order information window should be next to the calendar.</p> <p>U3.1. The calendar should display orders for the month.</p> <p>R1. The system must have a user with the highest access level (root), which cannot be changed or deleted.</p> <p>R2. The system should be available from 8 am to 5 pm, Monday through Friday.</p> <p>S1. The interface must be in Russian.</p> <p>S4. Ability to change database connection settings.</p> <p>X1. Access is differentiated by role.</p> <p>X2. Only authorized users can access the application.</p> <p>X3. Users with the "User" access level can only view placed orders.</p>
SHOULD	<p>F5. Some orders have a non-fixed date.</p> <p>F6. Orders must have a markup field.</p> <p>F7. It is possible to generate reports for the selected period.</p>

	P1. Applications must support at least 3 simultaneous users. S3. There should be a button in the interface to get help.
COULD	U3.2. The calendar should display orders for the week.
WONT	S2. The interface must support English and Estonian.

The prioritization result was discussed with the head of the Novotrade logistics group, and it was decided that only "MUST" and "SHOULD" categories will be implemented in the prototype.

2.2 Current solution

Currently, the employees of logistics use Microsoft Excel software (hereinafter simply Excel) for accounting. Despite its versatility, it has cons that cause discomfort.

All data is stored in one large file with several backup copies. Large cell count is with orientation, due to storage of pasts orders. As a temporal measure, the data is hidden thanks to the grid system.

Several people might work with the file at once so everyone must have access to the current version of the file. Excel provides co-working options, but it has its limitations, for example, users cannot create pages, merge cells, create charts, and others. To perform these actions, the user must first turn off sharing, perform the action, and then turn it back on [4]. Also, the enterprise has difficulties in transferring the current version of the file between computers. Microsoft supplies along with other office programs (Microsoft Word, Microsoft Excel, Microsoft PowerPoint, etc.) application for cloud file storage and sharing – OneDrive, however, it is a paid one [5] and is not included in the package currently used at the enterprise.

Additional functionality, like, calendar view (see 2.1 Requirements) is quite realizable using Excel, however, it is comparable in the development of a separate application, and to support it, it is need to observe the same formatting. Completed orders should also be marked, currently, the corresponding cells are painted in a certain color, which is not very convenient. The developer mode capabilities (interface elements: buttons, forms, etc.) as well as scripts (macros) VBA (Visual Basic for Applications) for automation [6] can be used to solve the problem, however, it requires both knowledges of the opportunities and time for implementation.

The big advantage of this solution is its flexibility, since any data can be changed, the file structure can be changed, as well as the ability to complete the task in a short time. The author will base the prototype on flexibility options, so employees could manipulate the data structure without changing the application.

2.3 Existing solutions

To solve the problem, it was decided to analyze the existing applications that could meet the requirements of the enterprise. Since this problem is typical for many enterprises, there are other solutions available on the market that could solve the problem.

2.3.1 1C: Enterprise

1C: Enterprise is the most popular solution for enterprises keeping records, especially in Russian-speaking countries. This software comes in the form of a configurator program and a presentation program. Since the program interface is initially empty, the enterprise can purchase one of the many configurations or make its own. This platform is very flexible, allowing users to create both entities with a different set of attributes, lists of preset values, subsystems for ordering the interface, including the ability to generate reports and export data, including in Excel format. The system is configured by the administrator who creates structures, sets possible values, data format, etc [7].

Despite the extensive functionality, the platform has a bad reputation, mainly due to the countless bugs, as well as frequent raw updates, which often disrupt the work of an already configured system, and introduce new bugs [8]. This software is paid on a pay-per-module basis [9].

2.3.2 Analogues of 1C: Enterprise – Ananas

There are also free analogs of 1C: Enterprise, as one of them, could be considered the open-source software – Ananas. This program is a clone of 1C: Enterprise with slightly reduced functionality and a modest interface. The capabilities of this program include accounting, generating reports, entities with custom attributes and lists of preset values, etc. [10].

The disadvantages of this software are support for only MySQL databases [11], as well as the fact that the last version of the program was released in 2007 [12], which indicates

that software is no longer actual. Moreover, it is not clear if this platform supports co-working with the application.

2.3.3 Goramp

GoRamp is an online real-time trucking order management platform. It provides both planning and accounting of orders, as well as their display in a calendar view with the display of certain data in each cell. The disadvantage of this platform is a fixed data structure, which may not be suitable for this solution [13].

Unfortunately, this company does not provide any information or demo versions, and the program's capabilities can be found out only from the main page of the site, which is not a very reliable source of information. The program is paid, but the detailed price can be found only upon request.

2.3.4 UTECH

UTECH provides an online service for accounting and planning logistics, with an emphasis on the delivery of goods. Among the possibilities is a database of trucks, locations, orders, accounts, etc. The scheduling capabilities include both scheduling orders and departures of trucks, with a schedule for the coming week. The entire interface is built in the form of a table, which limits the ability to present data [14].

The disadvantages are the fixed structure, as well as unnecessary functionality, which is not necessary in this case. As in the case of GoRamp, the company does not provide a demo version, however, it is possible to download the program, but it cannot work without a license.

2.3.5 Low Code

LCAP (Low Code Application Platforms) applications are a revision of the traditional development system in response to constant changes in requirements and a variety of development tools. These applications are well suited for testing ideas and developing software in the shortest possible time due to the use of ready-made modules and a graphical interface [15]. Flexibility is a plus since changes can be made at any time without going through all the development stages.

However, such solutions have their problems, and they are mainly aimed at further support and modernization of the system. So, if, when problems arise, the user cannot always go into the source code or find documentation, because such applications are often close sourced [16]. They also do not eliminate the need to think over the logic of the application, and the use of a graphical interface can affect in a bad way, taking minutes to implement a simple, but the unique thing, compared to the moments of a couple of lines of code.

2.3.6 Online boards

Another option is online boards like Trello, Miro, Notion, etc. The advantage of such applications is the relative freedom of action. For example, in Trello, users can create both Kanban boards and something like TODOs, there are import capabilities, distribution of roles, deadlines, etc. [17]. Miro allows users to create custom projects using many modules, ranging from tables to interactive documents and pictures [18]. Notion is an excellent tool for keeping a personal diary, blog, or entire wiki [19].

However, these solutions are paid, or have a free version with reduced functionality.

2.4 Conclusion

To define the functionality of the application being developed, the author has classified and prioritized the requirements. Further, the author made a comparative analysis of the current and other solutions that could solve the problem.

During the analysis, it turned out that the main disadvantages of the solutions are non-flexibility on the one hand and too much functionality that misleads users and provides a drop-in software quality due to numerous bugs. Also, ready-made solutions were either paid or were no longer supported, as in the case of Ananas. A big advantage of such applications is the flexibility of the structure, which is good in a constantly changing environment.

Following these conclusions, the developed application must be flexible so that users can change the structure themselves, without the participation of a programmer, and have only the functionality that users need.

3 Analysis of technological aspects

In this section, the author describes the selected technologies, such as program language, database, and others for the optimal implementation of the application prototype.

3.1 Type of application

First, it is necessary to determine the type of application, since the capabilities of the application and the development tools will depend on this choice.

At the present there are applications of the following types [20]:

- Web applications
- Desktop applications
- Mobile applications
- Cross-platform applications

Based on the initial conditions (see 1.3 Starting conditions), the author will not consider the type "Mobile applications", and also due to the complete lack of experience, the type "Cross-platform applications".

Native applications are written for a specific platform and limit their functionality to the capabilities of the platform. These applications are usually very efficient and can run without the Internet [21]. However, if it is needed to change or add platform support, these applications commonly require re-developing. Also, there should be a solution to provide updates for the application.

Web applications, unlike native applications, work on any platform where a browser can run them. These applications do not need to be installed or downloaded from the online store and typically provide a unified view of the interface on any device, as well as the ability to work from anywhere with Internet access. Unlike native apps, these applications always have actual version. The downside is the dependence on the Internet connection and the aspects of protection, which must be well thought out [20]. Despite the dependence on the Internet, some web applications can also be used without the Internet (for example, PWA (Progressive Web Application) [22]).

To summarize, both the native and web approaches to the implementation of the application are suitable in this situation. However, web applications are easier to update and platform independence makes it possible for one application to support several OS at once (for example, Windows, MacOS, android, and others). Considering the prospects of the web implementation, as well as author experience in developing web applications, the web application type will be used in developing.

3.2 Program language and framework

The choice of a programming language is important, not only because it determines what the final solution will be written on, but also determines the tools and structure of the application, for example, will it be a SPA (Single Page Application), a client with API (Application Programming Interface) server or a traditional server that operates data and renders pages.

At the present, there are many languages used to write web applications, among which, based on a selection of the most popular (see Figure 1), the author will analyze the following [23]: JavaScript, Python, Ruby, C#, Go, PHP

Mar 2021	Mar 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	15.33%	-1.00%
2	1	▼	Java	10.45%	-7.33%
3	3		Python	10.31%	+0.20%
4	4		C++	6.52%	-0.27%
5	5		C#	4.97%	-0.35%
6	6		Visual Basic	4.85%	-0.40%
7	7		JavaScript	2.11%	+0.06%
8	8		PHP	2.07%	+0.05%
9	12	▲	Assembly language	1.97%	+0.72%
10	9	▼	SQL	1.87%	+0.03%

Figure 1. Most popular program languages by TIOBE index in March 2021 [24]

It should be noted that each of the languages mentioned is suitable for running web applications that can be executed on a Windows Server (see 1.3 Starting conditions). To determine the language for development, the author relies on personal experience of use and development using a specific language and the approximate time required author to learn it (see Table 3).

Table 3. Program languages comparison

Language	Author's personal experience	Time required for mastering the language by the author (in weeks) ¹
JavaScript	3 clients using REST ² API, many small personal projects	0
Python	Assignments during the course, a couple of small personal projects	1 – 2
Java	2 courses, 2 personal projects, 2 – 3 mobile applications	1
Ruby	No experience	4+
C#	2 courses, full REST API server, several games on Unity	0
Go	No experience	4+
PHP	Basic knowledge	4+

Based on this comparison, given the lack of time to learn a new program language, further analysis will be between JavaScript, C #, Java, and Python languages.

3.2.1 JavaScript

JavaScript is a scripting program language used mainly to give dynamics and logic to pure HTML (HyperText Markup Language) web pages. By this day it is the most popular language to make a representative part, and it also got powerful tools and frameworks that turn him into a strong competitor to other program languages [25].

The most popular frameworks for writing web applications: React, Vue, Angular; Node.js runtime framework, Express.js pure server framework, and more. Also, must be considered the TypeScript add-on [26], which brings strong typing of variables, allowing JavaScript to compete with languages such as Java and C#. Among frameworks, it makes sense for the author to consider only Vue, Nuxt and React, due to lack of experience with Angular, and it is also quite difficult to learn [27].

¹ This time is relative, which the author suggested based on his own experience and the complexity of the language

² REST (Representational State Transfer) – see 3.4 SOAP, REST and GraphQL chapter

Vue was born in response to the relative complexity of React. The downside of React is that state updates are relatively slow, which can delay information on pages and take time to optimize [28]. Also, the "Store" state store – Redux, which is used in React, is much more difficult to write and typify than the Vuex store for Vue, as the author knows from writing both applications using React+Redux and Vue+Vuex.

Nuxt is a Vue-based framework that preserves the syntax and basic functionality and adds auto route detection and SSR (Server-Side Rendering). Nuxt also benefits from the modularity of its architecture, such as plugins, vuex store and layouts¹ that Vue does not have. [29]. The ability to use TypeScript significantly simplifies development, getting rid of dynamic errors, but it takes time to type objects and plug-in settings that are not sharpened for TypeScript.

3.2.2 C# and ASP.NET Core

C# is a strongly typed OOP (Object-Oriented Programming) language developed by Microsoft that has a wide range of uses from native and web applications to video game development. Programs developed in this language use the .NET platform, which for a long time was only for Windows OS, but with the release of .NET Core, applications are now cross-platform and can be installed on any system that supports .NET Core [30].

The framework for developing cross-platform web applications is ASP.NET Core, which allows to develop both server applications using the MVC (Model-View-Controller) pattern and the REST API servers to work with client applications written, for example, in JavaScript [31]. This framework has very powerful functionality and development tools, among which can be mentioned: Entity Framework, which is responsible for linking database entities with classes in C# and working with the database [32]; NuGet package manager system [33], that allows to easily add the necessary third-party functionality to the application, for example – JSON mapping or serialization; ASP.NET Core Identity API for a smooth and reliable user and role experience [34]; etc.

¹ In Vue, there is only one template by default - *App.vue*, the entry point into the application [44]

3.2.3 Java and Spring

Java is a strongly typed OOP programming language used for writing web applications (especially REST API servers), mobile applications, desktop applications, and others. Programs written in Java run using the Java Virtual Machine (JVM), which makes the applications cross-platform [35].

Spring is the most popular Java framework for writing web applications [36]. Spring provides the functionality for building complete, extensible applications with a clean architecture. Just like C#, Java has various extension repositories such as Maven, Ant, and a Gradle tool to automatically build an application. There is also the possibility of working with the database using entities [37]. Spring and ASP.NET Core are functionally very similar, and regardless of the choice, the application based on these languages will have a solid foundation and great opportunities for further development.

3.2.4 Python and Django

Python is a very popular programming language that has been at the top of the charts for the most used languages for the last couple of years, used mainly for developing artificial intelligence, but also for writing modern extensible server-only and server-client web applications based on the Django framework [38]. Django has a lot of out-of-the-box features like the admin panel that Spring and ASP.NET don't have by default. There are also many modules for Django that extend or add new functionality.

Among the minuses, dynamic typing, rather weak template validation, which do not check whether a model is suitable for a form or not, and no multiple requests processing at the same time. [39].

3.2.5 Framework and conclusion

To choose the framework, the author also relies on personal development experience using a specific technology, as well as the approximate time that the author needs to get a sufficient amount of experience (see Table 4).

Table 4. Frameworks comparison

Framework	Author's personal experience	Additional time required for mastering the language by the author (in weeks)
Express	No experience	2
Vue	REST API client	0
Nuxt.js	REST API client	1
React	REST API client	2
Angular	No experience	4+
Django	No experience	3+
Spring	Basic knowledge	2
ASP.NET Core	REST API server, client	0

The purpose of the framework also plays an important role in the definition, since, for example, ASP.NET Core can be for the frontend, but JavaScript can better handle it, due to smooth rendering and direct use of the browser API (see Table 5).

Table 5. Frameworks and their purpose

Language	Framework	Main purpose
JavaScript	Express	Server
JavaScript	Vue	Client
JavaScript	Nuxt.js	Client
JavaScript	React	Client
JavaScript	Angular	Client
Python	Django	Both
Java	Spring	Server
C#	ASP.NET Core	Server

Java Spring is a good choice for implementing the server-side of the application, and it is also more commonly used in Estonia than ASP.NET Core, however, with the last one, the author has more experience with the development of a full-fledged REST API server, so the author has chosen ASP.NET Core. The author decided not to choose Django due to a complete lack of experience, the large number of lines used as arguments in Django, for example in layouts.

As a client, the author has chosen Nuxt because it is built on top of Vue, and with Vue, the author has a more pleasant experience than with React, which is however by far more popular. Between Vue and Nuxt, the author chose the last one, because Nuxt provides more out-of-the-box capabilities than pure Vue, although they have the same syntax, which should not be reflected in the application.

3.3 Architecture

In development, the author will rely on the "ideals" of the architecture described in the book "Clean Architecture" by Robert Martin [40], also using the SOLID principles, also described in this book [40, pp. 72-74]. Following these recommendations, the final application will not grow exponentially in costs, it will be simply to support and, if errors occur, painlessly fixed.

The application should be logically divided into several parts – layers. The one of the most popular is the three-layer architecture model (see Figure 2):

- View layer
- Business logic layer (BLL)
- Data access layer (DAL)

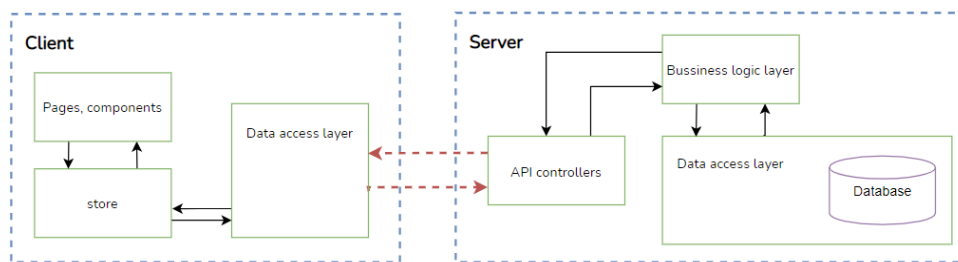


Figure 2. Solution architecture

Each of the layers encapsulates part of the functionality without knowing and caring about what is happening outside. The presentation (view) layer is the mediator between the user and the application, the entry/exit point. The business logic layer is responsible for all calculations. The data access layer encapsulates communication with the database [41]. On the server-side, the presentation layer is the API controllers since the client accesses them directly. For the client, access to the server API is also a kind of data access layer.

3.4 SOAP, REST and GraphQL

For the client and server to exchange information, it is the most common practice to use API requests. Since there is no single approach to building the access point API, the author compared the 3 most popular methods:

- SOAP (Simple Object Access Protocol)
- REST (Representational State Transfer)
- GraphQL (Graph Query Language)

SOAP is a relatively mature API protocol that uses a strongly typed XML (eXtensible Markup Language) structure. The data approach allows to accurately control what data and in what form is transmitted, but, increases the size of the request, and therefore the processing time. SOAP is mainly used to call procedures (like Add Order, Register User, etc) [42].

REST is a convention of building API requests compared to SOAP. Unlike SOAP, REST supports a variety of data formats, JSON (JavaScript Object Notation) is the most popular. A typical request to get a list of orders will be made through a GET request to */api/orders*. Each request has an address and a handler [43]. A significant drawback is the many endpoints in each application.

GraphQL is a query language that allows clients to query only the data they are needed as if they were working directly with the database. Unlike REST, GraphQL has only one API access point. The ability to query only the data that is needed would facilitate the development of the front-end part. However, GraphQL requires more server-side development effort, as well as proper configuration, otherwise, the server may crash under the large and voluminous queries with multiple levels of nesting [44].

SOAP is good for large, tightly architected applications; however, it requires additional effort and expertise to implement compared to REST. GraphQL provides a convenient way to develop a client application but also requires additional effort on server-side. Since the author does not have enough experience to effectively implement GraphQL along with a clean architecture, the author chose REST to implement the API.

3.5 Database choice

Since the enterprise system for the app to integrate with has not yet been developed, there is no point in carefully selecting a database. According to the enterprise requirements not to spend money the choice will be made from free options. Also, since the application will use a specific data structure that will not change, only Relational databases will be considered. The author has three options, among the most common: MySQL, PostgreSQL, and SQLite.

MySQL is a free open-source database for non-commercial projects. Since the application is being developed for employees only, there is no need for open source licensing for it. This database, in addition to the relative flexibility of customization, can offer a convenient program for managing [45].

PostgreSQL is an open-source web application database focused on scalability. This database provides customization flexibility, as well as the ability to work with JSON [45].

SQLite is a compact local solution. Among the above, this database has the least functionality. This database only supports *INTEGER*, *REAL*, *TEXT*, *BLOB*, and *NULL* data types, other types stored using these types [46], which may affect query performance.

To summarize, MySQL and PostgreSQL have a lot in common and both are better suited to the application than SQLite. Due to extensive experience of use and a convenient management application for the author, the author has chosen the MySQL.

3.6 Conclusion

During the analysis, the author determined that the best approach is to develop a web application based on ASP.NET Core as a server-side framework and Nuxt.js with TypeScript as a client-side framework using three-layer architecture, SOLID principles, MySQL and REST API.

Based on the analysis, the author can now start to implement the application prototype to solve the defined enterprise problem.

4 Implementation

In this section, the author describes the steps for implementing the application. The application is divided into three parts: database, server, and client. The development of a prototype assumes that all three parts will be developed at a level that allows the application to perform the assigned tasks but does not eliminate possible drawbacks.

4.1 Database

The main purpose of the application is to handle orders. Each order has a deadline (however, in the case of unfixed orders, this field must be null), as well as a list of attributes. For orders, the attributes rarely change; so, it was decided to move them into a separate entity and reuse them. Since orders can have multiple attributes, and attributes can have multiple orders, another entity was created to make this many-to-many relationship possible (see Figure 3).



Figure 3. Order and attribute tables relationship

Attributes differ not only by name and value but also by other features, for example, the values of some attributes may be predefined, for example for the "Product" attribute. Also, some attributes can have units of measurement, for example, the attribute "Quantity of production" can have units of measurement: "kg", "gram", "tons", "bags", etc. For that purpose, the entities "Units" and "Values" were created. They could be simply attached to an attribute, but there is a possibility that two attributes with the same units or defined values would be needed. To implement this feature, the author has created an additional entity – "Attribute Type". In this case, units and values are defined by the attribute type, and the attribute is created with a specific type. It can be assumed that values and units can also be reused by defining intermediate entities (see Figure 4).

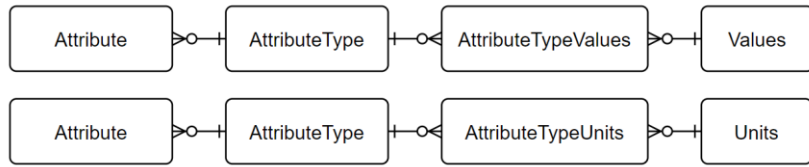


Figure 4. Complicated attribute type table relationship

However, the author considered this as an unnecessary complication of the database and application structure, so the relationship was simplified (see Figure 5).

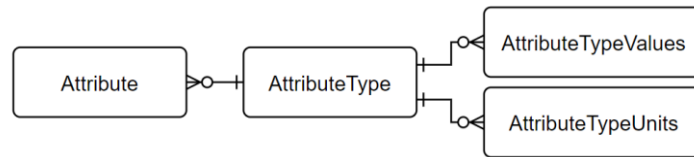


Figure 5. Simplified attribute type table relationship

To make it convenient for users to create new orders, the entity "Template" was created, which will save a list of certain attributes, and when creating an order, it will simply substitute them into the creation form. Again, since the template has several attributes, and the attributes have several templates, an intermediate entity is needed – "Template Attributes" (see Figure 6).



Figure 6. Template and attribute tables relationship

Orders and templates are not physically connected in any way, so there is no need for an intermediate entity. Also, it can be assumed that there is no point in storing templates in the database, since requesting a template may take extra time, however, since attributes can be changed or deleted, as well as the fact that several users can use the same template, the author decided it was appropriate.

The ERD diagram can be found in **Appendix 2**.

4.2 Backend

The application server acts as a data handler and bridge between the client interface and the database. In addition to working with data, the server is also involved in user authorization to certain API endpoints.

4.2.1 Server app structure

The server is designed following a three-layer architecture (see 3.3 Architecture) and is subdivided into several projects responsible both for a specific layer and for the task performed within this layer (see Figure 7).

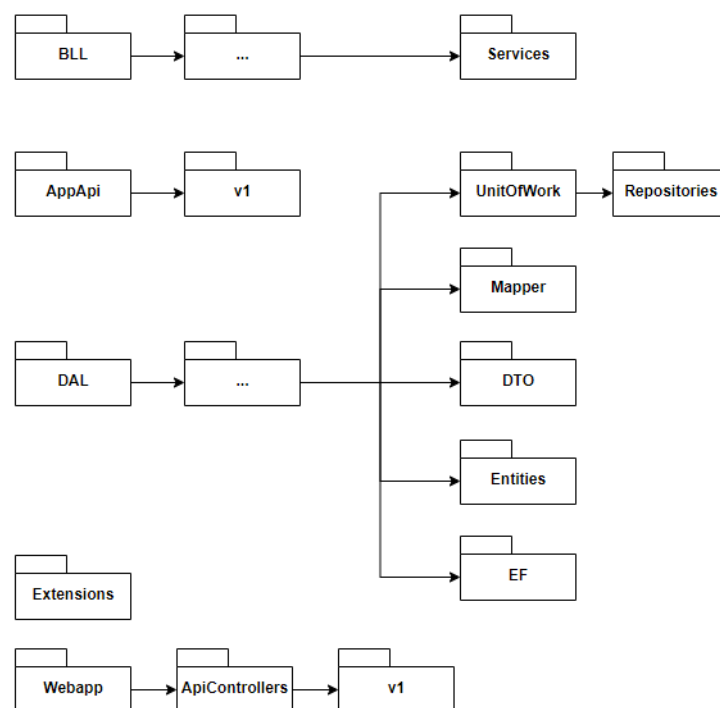


Figure 7. Server application structure

To name the projects, the author used combinations of the following words:

- *DAL* – everything that relates to *Data Access Layer*
- *BLL* – everything that relates to *Business Logic Layer*
- *App* – everything that is specific to the application
- *Base* – general principles from which *App* classes inherited
- *Contracts* – these projects contain interfaces

Also, the author used some names for certain things:

- *EF* – implementation of working with a database using the Entity Framework
- *Entities* – Entity Framework entities
- *Extensions* – classes containing extension functions
- *UnitOfWork* – everything related to the "Unit of Work" pattern implementation
- *DTO* – classes needed to transfer data between layers
- *AppAPI* – classes needed to implement the API
- *Webapp* – main and executable project

Further, the author describes the project implementation.

4.2.2 Domain entities

To work with the described database structure, the author created 8 problem-specific classes as well as 2 additional classes for working with users and roles (see Table 6).

Table 6. Domain entities

Entity	Database table	Description
Attribute	attribute	used to store all attributes
AttributeType	attributetype	used to store all attribute's types
AttributeTypeUnit	typeunit	a unit of measure that an attribute with a suitable type can use
AttributeTypeValue	typevalue	a defined value that an attribute with a suitable type can use
Order	order	used to store all orders
OrderAttribute	orderattribute	used to implement a many-to-many relationship between orders and attributes
Template	template	used to store all templates for quick order creation
TemplateAttribute	templateattribute	used to implement a many-to-many relationship between templates and attributes
AppUser	appuser	Used to store application users
AppRole	approle	Used to store application user's roles

Since entities are similar to each other, at least by identifiers, the author has created abstract classes to give entities certain properties (see Table 7).

Table 7. Domain abstract entities

Abstract entity	Description
DomainEntityId	entities with unary identifier
DomainEntityIdMetadata	entities with unary identifier and <i>createdAt</i> , <i>createdBy</i> , <i>changedAt</i> and <i>changesBy</i> fields to track state of entities
DomainEntityIdSoftDelete	<i>DomainEntityIdMetadata</i> + <i>deletedAt</i> and <i>deletedBy</i> fields, to track the deletion of an entity
DomainEntityIdSoftUpdate	<i>DomainEntityIdSoftDelete</i> + <i>masterId</i> field to keep track of the original version of the entity
DomainEntityMetadata	entities with <i>createdAt</i> , <i>createdBy</i> , <i>changedAt</i> and <i>changesBy</i> fields to track state of entities

Abstract classes have an overload with generic type for identifier, if necessary, the type can be changed to something else. The author has chosen the *long* data type as the default identifier since *long* can be generated in the database and it provides enough values for all possible cases. Data types *UUID (GUID)*, the author considered impractical, since they complicate the data query, and the application uses only one database, which means that there will be no conflict with the same identifiers. For an example of the implementation of the entity, the author gives the *AttributeType* class (see Figure 8).

```
public class AttributeType : DomainEntityIdSoftDelete
{
    public string Name
    public string? DefaultCustomValue

    public bool SystemicType
    public bool UsesDefinedValues
    public bool UsesDefinedUnits

    public AttributeDataType DataType

    public long DefaultValueId
    public long DefaultUnitId

    public ICollection<Attribute>? Attributes
    public ICollection<AttributeTypeValue>? TypeValues
    public ICollection<AttributeTypeUnit>? TypeUnits
}
```

Figure 8. AttributeType entity example

AttributeType inherits from the abstract class *DomainEntityIdSoftDelete*, which gives the entity a unary identifier as well as fields to track changes. When deleted, the *deletedAt* field will contain the date of deletion, which will not disrupt the operation of orders using this attribute, because the entity will not be physically deleted. The other fields are used to store data, like *SystemicType* – to determine if the type is should be protected, *DefaultCustomValue*, which used to store the default value, etc.

4.2.3 Entity Framework database context

To work with the Entity Framework, the author has created a class that inherits from *DbContext* and defined entities in it. Since this procedure is typical for all applications, the author will not focus on implementation. However, since the application must support soft delete and soft update actions, it is necessary to handle entities whose state changes. For this, the author has created a method that will be called when *SaveChanges* is called on the Entity Framework context. As an example, the author gives an implementation of entity validation on deletion (see Figure 9).

```
...
ChangeTracker.DetectChanges();
var now = DateTime.UtcNow;
...
var deletedEntities = ChangeTracker.Entries().Where(x => x.State ==
EntityState.Deleted);

foreach(var entityEntry in deletedEntities) {
if (entityEntry.Entity is not IDomainEntitySoftDelete softDeleteEntity)
continue;

    softDeleteEntity.DeletedAt = now;
    softDeleteEntity.DeletedBy = _userProvider.CurrentName;
    entityEntry.State = EntityState.Modified;
}
...

```

Figure 9. Example of handling of deleted entities

When deleting an entity, it is checked whether the entity should be deleted "softly" with the date of deletion, or if it should be physically deleted, in this case, its state does not change to *EntityState.Modified* and it will be deleted.

4.2.4 Database initialization

For the application to be able to start working with a clean database, the author created a class responsible for migrating the database schema and data, users, and roles seeding. The need for individual actions is configured from the application settings file, where it can be also specified how to translate user roles, as well as login information for the Super Administrator account. As a security measure, the super user is seeded only once.

Since, often, attributes do not require a specific type, to immediately create attributes, during initialization, 7 basic system types (*string*, *boolean*, *integer*, *float*, *date*, *time*, and *datetime*) are seeded. The application does not support types that accept multiple values at the same time; however, users can create multiple attributes with different values. The example of implementation can be found in **Appendix 3**.

4.2.5 Data access layer

To work with DAL the author has created the *AppUnitOfWork* class, which is an implementation of the “Unit of Work” pattern [47, pp. 184-189]. This class has a list of repositories, as well as a factory method [40, p. 100] to get the instance of the repository (see Figure 10).

```
protected TRepository GetRepository <TRepository> (Func<TRepository>
repoCreationMethod) {
    if (_repoInstances.TryGetValue(typeof (TRepository), out
        var obj1))
        return (TRepository) obj1!;

    object obj2 = repoCreationMethod!;

    _repoInstances.Add(typeof (TRepository), obj2);

    return (TRepository) obj2;
}
```

Figure 10. Repository factory method

When called, the method checks if the repository instance exists, and if not, it creates the repository and returns it to the dictionary. On subsequent requests, the cached repository will be returned. Next, the author gives an example of a method call (see Figure 11).

```
public IAttributeRepo Attributes => GetRepository<IAttributeRepo>(  
    () => new AttributeRepo(DbContext, Mapper));
```

Figure 11. Repository creation using factory method

AppUnitOfWork does not contain a list of specific classes, but of their interfaces, which allows, if necessary, to create a test repository inherited from the interface and connect it instead of the main one for test purposes.

The author has created a separate repository for working with each entity. To improve the quality of the code, a similar set of methods were moved to the base class *BaseRepo*. To ensure the modularity of the repositories, as well as the observance of the one of the SOLID principles – "Dependency Inversion" [40, pp. 98-101], the author has created interfaces from which the repositories are inherited, including the base repository. The base repository is generic, accepting two types, an entity type and a DTO type that will pass the data of that entity.

The author has created special classes for transferring information – DTO [40, p. 195] for each entity, because using the entities directly may create a bad situation – if changing happens, it will affect not only the operation of the repository, but also the layers above, and, in addition to the strong dependence on the implementation, which violates modularity, if an error occurs, it will need to be corrected for all layers.

For entities to be converted into DTOs at the layer boundary, it is necessary to map them. This can be done manually by creating new classes and assigning values to all fields, however, since DTOs are identical to entities, at this stage, the author has decided to use the most popular automatic conversion library – *AutoMapper*, since it is easy to configure and allows to quite accurately specify how objects should be converted. The author has created a *UniversalMapper* wrapper class that provides a conversion for all entities. The implementation can be found in **Appendix 4**.

This approach provides modularity, which, as in the case of repositories, allows to test the unit of work module separately.

4.2.6 Business logic layer

To separate API controllers and client implementation from DAL, the author has used an intermediate layer – BLL [48], the purpose of which is both data manipulation and validation, so the controller will only access the BLL and handle errors that occurred during model validation. Another advantage is the ability to reuse the logic of working with individual entities. To implement BLL, the *AppBLL* class was created, which implements the *IAppBLL* interface, as well as service classes that, like repositories, wrap the logic for working with a specific entity. To be able to access the service through the BLL, a factory method was implemented, the principle of which is identical to the factory method of the repositories, so the author will not describe its implementation.

For the controllers to be able to handle erroneous requests, two exception classes *NotFoundException* and *ValidationException* were created, which throws an exception when a validation error occurs on the BLL side, so the controller does not know anything about whether the content of a request is correct or not, but can answer to the client in case of an error, for example by sending a "404" response code.

Since the BLL implementation is similar to the DAL, the author does not see the point in describing almost the same thing again, so the accent will be made only on the transfer of information between the BLL and the controllers. As in the case of DAL entities, it was possible to use the *AutoMapper* package, but since the entity API and DAL DTO are very different, the author has decided to convert them manually on the BLL side. The advantage of this way is that the controllers do not know anything about the form in which the data comes from the DAL, as well as what happens to it. There is a downside – it can be said that controllers depend on the DTO APIs. However, the author considered this insignificant, because, when the app logic changes and if the clients are already using the API, it is better to create a new service and controller that will use different logic and return new DTOs, so that the old versions will not be broken.

4.2.7 Handling soft update entities

To implement order history tracking, it is necessary to save the previous versions. Entities with soft update capability must inherit from *DomainEntityIdSoftUpdate* (see 4.2.2 Domain entities). When the *DomainEntityIdSoftUpdate* entities are updated, a copy is created to save the previous version, the date of the update is set in the *deletedAt* field and

the identifier of the original record is set in the *masterId* field. Thus, there is no need to update all dependent entities, and the entire history of the entity can be queried if necessary (see Figure 12).

```
public virtual async Task UpdateAsync(TDTO dto)
{
    TEntity trackedEntity = await DbSet.FindAsync(dto.Id);
    TEntity entityToTrack = MapToEntity(dto);

    foreach (var entity in DbSet.Local)
    {
        if (entity.Id.Equals(dto.Id))
        {
            DbContext.Entry(entity).State = EntityState.Detached;
        }
    }

    if (trackedEntity is IDomainEntitySoftUpdate softUpdate)
    {
        softUpdate.MasterId = trackedEntity.Id;
        trackedEntity.Id = 0;
        await DbSet.AddAsync(trackedEntity);
    }

    DbSet.Update(entityToTrack);
}
```

Figure 12. Entities update method with soft update handling

When updating, the current version must be obtained, then it is "detached" from the context so that there are no problems if the entity is already tracked somewhere in the application. Further, if the entity requires a soft update, a copy is created and added to the context as a newly added entity. At this stage, the date is not added, because when updating a set of entities, it may take a couple of seconds, which may affect the query for history, if entities also have child entities with history, such as orders that have attributes. The date is applied to DbContext to all changed entities (see 4.2.3 Entity Framework database context).

4.2.8 Querying of an actual data

Since the application supports soft update and soft delete, this must be considered when requesting data from the context, since in both cases, the data of deleted entities and old versions are no longer actual. To get valid data, the author used the filtering capabilities of the Entity Framework [49] (see Figure 13).


```
await _context.AttributeTypes.CountAsync(at => at.DeletedAt == null)
```

Figure 13. Example of actual data query

Entity Framework will automatically consider filtering when creating a database query, without taking up resources and memory to load irrelevant data. The author has implemented soft update in the way that both when soft deleting and when soft updating an entity, the date of deletion is set. Thus, to filter out such entities, it is enough to check for *deletedAt*.

4.2.9 API controllers

To implement API requests handling, the author has created 6 API controllers (see Table 8).

Table 8. API controllers

Name	API base url	Description
IdentityController	<i>/api/v1/identity</i>	Handles authorization and user management
AttributesController	<i>/api/v1/attributes</i>	Handles attributes management
AttributeTypesController	<i>/api/v1/attributetypes</i>	Handles attribute types, type values and type units management
HomeController	<i>/api/v1</i>	For the client to check if server is online
OrdersController	<i>/api/v1/orders</i>	Handles orders and order attributes management
TemplatesController	<i>/api/v1/templates</i>	Handles templates and template attributes management

Not all controllers are responsible for only one entity, for example, *AttributeTypesController* is responsible also for the type's defined values and units of measurement.

4.2.10 Security

To authorize users into applications, the author used jwt (Json Web Token) as an enough secure and simple authorization method [50]. In order not to indicate to each controller that authorization goes through jwt, the default authorization and authentication schemes were written in the configuration of the *Startup.cs* executable file (see Figure 14).

```

.AddAuthentication(options =>
{
    options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultAuthenticateScheme = JwtBearerDefaults...;
    options.DefaultChallengeScheme = JwtBearerDefaults...;
})

```

Figure 14. Default authentication and authorization scheme

To differentiate access to individual API endpoints, a special attribute was added to each of the API controllers (see Figure 15).

```
[Authorize(Roles = "User, Administrator, Root")]
```

Figure 15. Example of restricting access to controllers

This attribute tells the controller that this method (or the whole controller) can be accessed by users with the “User”, “Administrator”, or “Root” (Super Administrator) roles.

The author has restricted the access to the application by roles the following way: only authorized users can interact with the API, except for the login method; users with the “User” role can view orders in the calendar, but cannot add orders or change the structure: templates, attributes, types, etc; "Administrator" and "Root" users can change the all. Users with the “Administrator” role differ from “Root” (Super Admin) only in that “Root” is needed to add users and administration and cannot be changed from the application.

4.2.11 Swagger

For more convenient API development, as well as manual testing of the server logic, the author installed the *Swashbuckle.AspNetCore.SwaggerGen* – Swagger package. Also, in the future, the availability of this tool will make further development easier.

4.3 Frontend

The client application is an important part of the solution since it wraps the work with the server in the form of an interface that an ordinary person can work with without delving into the specifics of the API and the server logic.

4.3.1 Client application initialization

When creating a project through the console, it is possible to select an application configuration. The author has chosen the following configuration (see Figure 16).

```
? Programming language: TypeScript
? Package manager: Npm
? UI framework: Vuetify.js
? Nuxt.js modules: Axios - Promise based HTTP client
? Linting tools: ESLint
? Rendering mode: Single Page App
? Deployment target: Server (Node.js hosting)
? Continuous integration: None
? Version control system: None
```

Figure 16. Nuxt project initialization

The TypeScript was chosen as the program language for the possibility of typing components, which will make development easier. NPM was chosen as a package manager, since the author has experience with it, unlike Yarn package manager.

For the interface of the app, the author decided to use frameworks to save time. The author has chosen Vuetify as a framework, even though the author has more experience with Bootstrap. This framework provides special Vue components, which is certainly more convenient than writing style classes for each component, but it is possible to use style classes too. An additional feature is the presence of a building fully customizable calendar component, which will facilitate the creation of a calendar view for orders.

Among other things, Axios library, because it is a light and easy way to communicate between client and APIs; ESLint is a useful plugin that ensures code cleanliness, especially when the project is developed by more than one person; and SPA application mode.

4.3.2 Client app structure

By default, when creating a Nuxt project, several empty folders are created, which are needed for the Nuxt engine to automatically recognize and include content in the application. The following is the structure of the finished application (see Figure 17).

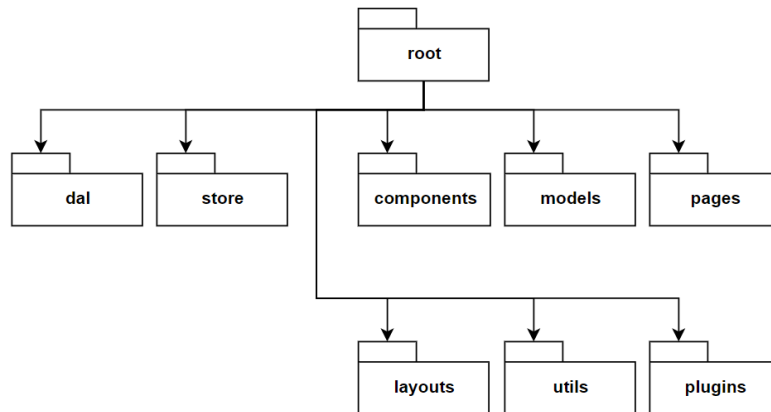


Figure 17. Client application structure

The entire project is contained in the *root* folder, where in addition to folders, the stored configuration also, for example, *nuxt.config.js*, *tsconfig.json*, *package.json*, and others.

All application pages are stored in the *pages* folder. The structure of this folder is the router configuration of the application [51]. So, for example, the *index.vue* file in this folder will be available at *https://<app_url>/index*. To specify page parameters, for example, an entity id, an underscore, and the name of the parameter is added to the title, for example – *_id*. The *index.vue* file in the root folder is the main page of the app – calendar view. The rest of the pages are arranged in subfolders of the corresponding entity, for example: */templates*, */orders*, etc.

Reusable parts are moved into a *components* folder. Unlike pages, other folders have no structure restrictions. The common features of each page are moved into separate layout files, such as a navigation bar, etc. Layouts are stored in the *layouts* folder.

The *models* folder contains DTOs for working with server APIs. Files providing requesting functionality the author named “repos” – repositories; and placed in the *dal* folder. From there, they can be used in both pages and store components, which are in the *store* folder.

Files that provide additional functionality are stored in *utils* and *plugins* folders. The *utils* folder contains utility functions such as form validation, etc. Plugins, stored in the *plugins* folder, can be used to add functionality to the nuxt context. A common example of a plugin would be Vue plugin initialization that connects Vue plugin to Nuxt instance.

4.3.3 API calls

The author has chosen axios for sending and receiving requests as it is very simple and useful enough. However, by default, when an error response is received from the server, axios interprets it as an own error and does not pass the result on. To avoid this, errors handling must be in every request. Since this is relevant for every request, the author has created a base class from which other repositories inherit and which contains wrapper methods for axios methods, including error handling.

```
protected async _get<TKey>(url: string, onError?: ErrorCallback, config?:
AxiosRequestConfig | undefined): Promise<TKey> {
    const response = await this.axios.$get<TKey>(url, config)
        .catch((err: AxiosError) => responseCatch(err, onError))
    return response
}
```

Figure 18. Axios GET method wrapper with error handling

When an error response is received, it is passed on because the error contains a message that can be displayed in forms when they are submitted. In addition, this method can take a callback function as a parameter, which will be executed when an error response is received. The *responseCatch* function checks if it was an error response, if it was, it executes the callback function if provided and returns the content of the response.

Since in this case, axios is a nuxt plugin, axios can only be used from nuxt pages or components. To solve the problem of direct usage of API, the author has created the unit of work plugin, that will provide axios instance to all components that require it. To create a unit of work must be considered two things, it must be accessible from the nuxt context, and, it must use the axios instance supplied from the nuxt context. The implementation was inspired by Alexander Lichter [52] (see Figure 19).

```
const AppUnitOfWork: Plugin = (ctx: Context, inject: Inject) => {
    const repositories: IAppUnitofWork = {
        attributes: new AttributesRepo(ctx.$axios),
        ...
        orders: new OrdersRepo(ctx.$axios)
    }

    inject('uow', repositories)
}
```

Figure 19. Nuxt unit of work plugin

This plugin takes a nuxt context as a parameter, which is supplied through dependency injection, takes an axios instance from it and creates repositories. Repositories are stored in a *repositories* array. Next, the plugin inserts an array with repositories into the nuxt context, which makes them available in pages and components.

4.3.4 Vuex storage

To save and share data between pages, the author used the Vuex store. The additional advantage of this approach is the ability to place the logic for working with the created previously unit of work also in vuex store, which will ensure the independence of the interface from the implementation of requests to the API.

The problem is that even though the store is autodetected by the nuxt engine and accessible from the context, the store components themselves do not have access to the context. To solve this, the author has created two files – *vue-context.ts* and *context-accessor.ts*. The *vue-context.ts* file provides a function that saves the context to a variable, which can then be imported. The *context-accessor.ts* file is a nuxt plugin, its job is to get the context and save it by calling a function from *vue-context.ts*. Thus, by importing the variable, the context can be used in any file.

By default, vuex was designed for JavaScript, which means there is no typing in it. This can be fixed by using the *vuex-module-decorators* module, which provides decorator methods as well as typing for vuex [53].

Further, the author describes the author implementation of the unique cases of individual components.

4.3.5 Attribute types

For attribute types, the author has implemented the ability to view all types, with the ability to search and sort by name, as well as display page by page, 12 pieces¹ on each page. The author also has included attribute type category in the list: “regular”, “system”, “with defined values”, and “with defined units of measurement”. Categories can overlap.

¹ 12 – the number of elements that fit on the screen

The type details display the available values and units, which ones are assigned by default, and the format of stored data. Since the data format is Enum with the numeric value, it needs to be converted to a string. To do this, the author used the capabilities of Vue filters [54].

```
Vue.filter('formatDataType', function (value: any) {  
  if (typeof value !== "string" && isNaN(Number(value))) return value;  
  return localize(value as DataType)  
})
```

Figure 20. Attribute type format display filter

Here the *localize* function takes a number as input and returns a string with the name of the data format.

When creating types, there must a default value, or predefined values, in which case at least one value must be specified. To add values and units of measurement, the author created two dialog components (see **Appendix 5**).

When creating, the data format is required, which affects only the data display, which means that the form for entering values must also be displayed correctly. For this, the author has created a component *CustomValueField.vue* that takes a type as a parameter and substitutes the correct fields for entering values (see **Appendix 6**). The implementation of the change page is a little more complicated, as some values and units may already exist, but new ones may also be added, and this must be considered when updating and displaying in form.

When deleting, a confirmation window is shown to prevent the type from being deleted in the event of an accidental click.

The attribute types implementation result can be found in **Appendix 10**.

4.3.6 Attributes

The difference of the attributes index page is the presence of types and the ability to sort not only by name but also by type.

When creating an attribute, the user can specify the name and select the attribute type in the drop-down list. The list of attributes is dynamic, because if there are many types, it may take time to fully load them, so this field assumes that the user must enter part of the

type name to select it. To get types, the same logic is used for displaying all types on a page, only the first page is always requested, and the server returns only 12 types, which, however, is enough if there are not many of them.

Attribute details show the name, type of attribute, format, default value, and amount of use, which is important because the user cannot change the type of an attribute that is already in use, as this will lead to data loss, since different types differ not only by format, but also by defined values and units. For a user-friendly design, when clicking on an attribute type, the user will be transferred to the type details page. Deletion is also available only if the attribute is not used.

The attributes implementation result can be found in **Appendix 11**.

4.3.7 Order templates

In this case, the author considered creating a page for details inappropriate, since templates have only a name and attributes that can be displayed on a page with all templates. Since there can be a lot of attributes in a template, the author designed the templates on the page in the form of an expansion panel, upon expanding which attributes appear, which in turn can also be opened to see the type of attribute, the format, and also whether the attribute uses defined values and units of measurement.

The order templates implementation result can be found in **Appendix 12**.

4.3.8 Orders

To display all orders, the author made two separate views – a view with all orders, and a calendar view, since the first allows to get more detailed information on all orders. Since orders can be without a date, the author has subdivided the page with all orders into two pages using different server API endpoints. To conveniently filter orders, the author implements a dialog component that allows to filter orders by completion, overdue, specify a date range and a check date to see what the status of orders was in a certain period (only for orders with a deadline date). To be able to select a date, the author also implemented a date selection dialog. Filtering and date picker dialog components implementation can be found in (filtering in **Appendix 7**, picker in **Appendix 8**).

To implement the calendar view, the author used the Vuetify `<v-calendar>` component, which can be customized by overriding the implementation of individual parts [55]. To

display orders in the calendar, the author has overridden the implementation of the calendar day, displaying the number of orders, as well as the selected attributes in the order cell. The obvious problem is that only a small number of orders can fit into a cell, however, programmatically limiting the number of orders is not the best solution, so the author has also overridden the label of the calendar date. Initially, the date in the calendar is displayed as a day of the month, to replace it, the author created the *CalendarMenu.vue* component, which displays the date in the “DD-MM-YYYY” format using the *moment.js* library, and also upon clicking on the date, displays a menu, with orders for a given date. For convenience, if some orders do not fit into the cell of the day, a “v” symbol appears next to the date, indicating that the user can view all orders when clicked.

The window next to the calendar displays information about the selected order: order number, deadline, attributes, and a note. For convenience, there are three buttons downward: a button for quickly marking an order as completed or vice-versa; a button for going to the details page, and a button for changing an order.

When creating an order, the user need to not only select an attribute but also assign values to it. To select attributes, the author created the *AttributeSellect.vue* component, which is a dynamic list, and the *AttributeValueSellect.vue* component, which takes an attribute type as a parameter and shows a form for entering a value (or selecting from the list of available ones) and selecting units of measurement if they are used by the given attribute type. The implementation of these components can be found in **Appendix 9**. Changes to orders are also implemented using these components.

For report generation, the author has created a separate modal window component. The jsPDF library was used as a pdf generator, since it allows to use HTML for generation, which makes the task easier, and also, the text remains selectable in a pdf file, which is certainly important because employees need to be able to work with the report further.

Upon creating a report, the user can select a date range (for orders with a date), and after generating, the pdf file opens in a new browser window, which allows to immediately start printing. An example report can be found in **Appendix 15**.

The order implementation result can be found in **Appendix 13**.

4.3.9 Forms validation

Since the author chose Vuetify as the UI framework, the form component and input field components have built-in validation. For validation, an array of validation functions that check the condition and return either true or a string with an error message must be passed to form components. Initially, the author wanted to use third-party validation libraries - *Vee-validate* and *Vuelidate*, however, the author found them too complicated to use, so it was decided to create a file with pre-installed validation functions. One of these is the *required* function for validating simple fields (see Figure 21).

```
export const required = () => (value: any) => {
  if (typeof(value) === "string" && value.length > 0) return true
  if (typeof(value) === "number" && !isNaN(value)) return true
  return !!value || `Данное поле обязательно`
}
```

Figure 21. Custom validation function example

This function checks the value for a string or number and applies the corresponding checks for the existence of the value, otherwise, it casts the value to *boolean* and checks it. Thus, for each component or page, the same functions can be imported and passed in an array as a parameter. Then, when submitting the form, the form validates all fields, and if they are valid, a request is made to the server through the vuex store and unit of work.

4.3.10 Security

First, the author has implemented the user login capabilities by creating a login page with a form. After the jwt token is received, it is saved in the vuex store, as well as in the local storage of the browser, so that the token is shared not only between components but also when the page is reloaded. However, in this case, it is necessary to check if the token is correct. At this stage, the author decided that it is sufficient to check the token's expiration date and delete it if it is out of date. To work with the token, the user's vuex store sets it to the axios instance obtained through *context-acessor.ts* (see 4.3.4 Vuex storage), so that all requests made after could have an authorization token.

To manage users, the author has created a page where users can change personal data and administrators can manage other users. The access logic in this component is the same as on the server, administrators can change other users, but not administrators, super admin

can change everyone except himself, including setting the “User” and “Admin” roles. The result of the user page implementation can be found in **Appendix 14**.

To restrict user access to certain parts of the application, the author used the capabilities of the nuxt middlewares and created a middleware that checks whether the user is logged in and what role he belongs to. If users are not in the system, it will be redirected to a page with a login form, and if an ordinary user tries to open pages intended for administrators only, he will be redirected to the main page – with a calendar view, that is available for all authorized users.

4.4 Testing

To check the functionality of the application, the author tested each endpoint manually with different input parameters. Among the checks, the author checked both application validation, fault tolerance when entering a large amount of data, as well as stability when changing and deleting entities on which other entities depend. Also, the author has tested the client application for declared functionality.

During development, the application was not tested through unit tests, since writing them requires additional time, and the author does not have enough experience to write high-quality tests. However, during the development of the application, the prerequisites for full testing were created due to the modular structure of the application.

5 Further development

The next steps after the development of the application are to install it in the workplace and test it in real work conditions. It is necessary to explain to employees how to use the application. The application must be user-friendly and free from critical errors in working with data.

Further development of the application will be aimed at realizing the wishes of employees, increasing additional functionality, identifying, and fixing potential errors, improving the interface, as well as integrating with the general system when it will be developed. The author plans to change the solution according to the changes of business requirements.

When developing the next versions, the author plans to write unit tests, since now the application can be tested manually, however, as the application grows, manual testing will become impossible. Also, for the application to work correctly in real conditions, the author plans to log the application's actions to detect a potential error as early as possible. The further development will not be described in this document.

6 Summary

The enterprise delivers manufactured products; for this, a logistic group keeps records of what product, in what quantity, when, where, and by whom it is delivered. The problem for the enterprise was to use standard accounting tools, which, although they coped with the task at hand, the staff needed to spend time on unnecessary work every time. To solve this, the enterprise set the task of developing a more highly specialized application that would allow employees to concentrate on completing their tasks. The purpose of the thesis was to solve the problem of the enterprise by creating that application.

During the development, the author solved all the tasks. The required functionality of the application was determined based on the collection of employees requirements and the analysis of third parties applications that offer their own solution to the problem. Further, having determined the optimal path, the author developed the application for manufactured products accounting. The application allows to create orders, specify their attributes, and customize the attributes, specifying both the valid values and units of measurement and the data format. At the request of the enterprise was also implemented the ability to view the history of changes in orders, display in a calendar form, and export orders for a certain period.

The application has a modular architecture that provides the ability to test business logic, replace certain parts and integrate with other systems. At the moment, the application has taken its place as a tool of the logistics group of the Novotrade Invest AS enterprise. The solution saves employees time approximately 6-12 hours per week, depending on the number and complexity of orders. In case of continuing work, the author is ready to continue working on further improving of application.

The created solution can also be used for solving similar problems in other manufacturing enterprises and not only.

References

- [1] "VNK - Home," [Online]. Available: <http://www.vnk.ee>. [Accessed 23.02.2021].
- [2] P. E. IBM, "Capturing Architectural Requirements," 15.11.2005. [Online]. Available: <https://www.ibm.com/developerworks/rational/library/4706-pdf.pdf>. [Accessed 11.04.2021].
- [3] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," March 1997. [Online]. Available: <https://www.ietf.org/rfc/rfc2119.txt>. [Accessed 11.04.2021].
- [4] "Share excel file at the same time," ExelTABLE, [Online]. Available: <https://exceltable.com/vozmojnosti-excel/sovместnyi-dostup-k-failu-excel>. [Accessed 29.03.2021].
- [5] Microsoft, "Compare OneDrive cloud storage pricing and plans," Microsoft, [Online]. Available: <https://www.microsoft.com/en-us/microsoft-365/onedrive/compare-onedrive-plans?activetab=tab:primaryr2>. [Accessed 29.03.2021].
- [6] Microsoft, "Getting started with VBA in Office," Microsoft, 14.08.2019. [Online]. Available: <https://docs.microsoft.com/en-us/office/vba/library-reference/concepts/getting-started-with-vba-in-office>. [Accessed 11.04.2021].
- [7] 1C, "Overview of the "1C: Enterprise 8" system," [Online]. Available: <https://v8.1c.ru/tekhnologii/overview/>. [Accessed 30.03.2021].
- [8] K. Ramil, "Why 1C is bad and why 1C programmers are so disliked," 02.12.2014. [Online]. Available: <https://habr.com/ru/company/trinion/blog/244727/>. [Accessed 11.04.2021].
- [9] 1C, "1C: Enterprise 8 - Prices and delivery procedure," [Online]. Available: <https://v8.1c.ru/price/>. [Accessed 11.04.2021].
- [10] A. Pascal, ""Operational accounting" on the "Ananas" platform - User's Guide," 2007. [Online]. Available: <https://ananas.su/docs/ananas-inventory-user-manual.pdf>. [Accessed 11.04.2021].
- [11] Ananas, "Installing Ananas for Windows," 27.07.2009. [Online]. Available: https://ananas.su/wiki/Установка_Ананаса_для_Windows. [Accessed 11.04.2021].
- [12] "FREE ANANAS PROGRAM," Freeanalogs team, 04.02.2016. [Online]. Available: <https://freeanalogs.ru/Ananas>. [Accessed 11.04.2021].
- [13] "GoRamp TMS," GoRamp, [Online]. Available: <https://goramp.eu/>. [Accessed 30.03.2021].
- [14] Utech Corp, "UTECH TMS | Transportation Management Software - Demo," [Online]. Available: <https://www.youtube.com/watch?v=JIsLCuIDEwQ>. [Accessed 30.03.2021].

- [15] C. Richardson and J. R. Rymer, "New Development Platforms Emerge For Customer-Facing Applications," *Forrester*, June 9, 2014.
- [16] C. Richardson and J. R. Rymer, "Low-Code Platforms Deliver Customer-Facing Apps Fast, But Will They Scale Up?," *Forrester*, August 11, 2015 | Updated: August 13, 2015.
- [17] "Trello," Atlassian, [Online]. Available: <https://trello.com/>. [Accessed 11.04.2021].
- [18] "The online collaborative whiteboard platform to bring teams together, anytime, anywhere.," Miro, [Online]. Available: <https://miro.com/index/>. [Accessed 11.04.2021].
- [19] "One tool for your whole team. Write, plan, and get organized.," Notion Labs, Inc., [Online]. Available: <https://www.notion.so/>. [Accessed 05.04.2021].
- [20] G. Perlman, "Web, Desktop, Mobile, or Cross-Platform: Options for App Developers," 12.01.2017. [Online]. Available: <https://learntocodewith.me/posts/cross-platform-apps/#web-applications>. [Accessed 04.04.2021].
- [21] A. S. Gillis, "What is native app?," August 2020. [Online]. Available: <https://searchsoftwarequality.techtarget.com/definition/native-application-native-app>. [Accessed 04.04.2021].
- [22] Sam Richard and Pete LePage, "What are Progressive Web Apps?," Google, 24.02.2020. [Online]. Available: <https://web.dev/what-are-pwas/>. [Accessed 17.03.2021].
- [23] J. Paul, "Top 5 Programming languages for Web development in 2021," 13.02.2021. [Online]. Available: <https://medium.com/javarevisited/top-5-programming-languages-for-web-development-in-2021-f6fd4f564eb6>. [Accessed 22.03.2021].
- [24] "TIOBE Index for March 2021," TIOBE, 03 2021. [Online]. Available: <https://www.tiobe.com/tiobe-index/>. [Accessed 22.03.2021].
- [25] J. Toledo, "Why Millions of Developers use JavaScript for Web Application Development," 07.06.2018. [Online]. Available: <https://torquemag.io/2018/06/why-millions-of-developers-use-javascript-for-web-application-development/>. [Accessed 04.04.2021].
- [26] Microsoft, "Typed JavaScript at Any Scale.," Microsoft, [Online]. Available: <https://www.typescriptlang.org/>. [Accessed 04.04.2021].
- [27] T. Merkle, "Why Angular Made Me Quit Web Dev," 05.11.2018. [Online]. Available: <https://hackernoon.com/why-angular-made-me-quit-web-dev-f63b83a157af>. [Accessed 12.04.2021].
- [28] D. Han, "My React App is Slow. What Should I do?," Nov 8, 2019, 08.11.2019. [Online]. Available: <https://medium.com/in-the-weeds/my-react-app-is-slow-what-should-i-do-e1fd020e69ec>. [Accessed 12.04.2021].
- [29] O. Omole, "Nuxt.js: a Minimalist Framework for Creating Universal Vue.js Apps," 18.03.2019. [Online]. Available: <https://www.sitepoint.com/nuxt-js-universal-vue-js/>. [Accessed 12.04.2021].
- [30] Microsoft, "What is .NET?," [Online]. Available: <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet>. [Accessed 04.04.2021].

- [31] A. Lock, "Getting started with ASP.NET Core," 23.06.2020. [Online]. Available: <https://andrewlock.net/aspnetcore-in-action-2e-getting-started-with-asp-net-core/>. [Accessed 04.04.2021].
- [32] Microsoft, "Getting Started with EF Core," 17.09.2019. [Online]. Available: <https://docs.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>. [Accessed 07.04.2021].
- [33] Microsoft, "An introduction to NuGet," 24.05.2019. [Online]. Available: <https://docs.microsoft.com/en-us/nuget/what-is-nuget>. [Accessed 12.04.2021].
- [34] Microsoft, "Introduction to Identity on ASP.NET Core," 15.07.2020. [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio>. [Accessed 12.04.2021].
- [35] METANIT.COM, "Introduction to Java," [Online]. Available: <https://metanit.com/java/tutorial/1.1.php>. [Accessed 12.04.2021].
- [36] P. Banerjee, "Top 10 Most Popular Java Frameworks for Web Development," 03.10.2020. [Online]. Available: <https://www.geeksforgeeks.org/top-10-most-popular-java-frameworks-for-web-development/>. [Accessed 04.04.2021].
- [37] Oracle, "A quick tour of Java EE," [Online]. Available: <https://www.oracle.com/java/technologies/java-ee-glance.html>. [Accessed 12.04.2021].
- [38] MDN contributors, "Django introduction," [Online]. Available: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>. [Accessed 12.04.2021].
- [39] S. Bhatt, "Pros and Cons of Django Framework for App Development," Aug. 31, 20. [Online]. Available: <https://dzone.com/articles/pros-and-cons-of-django-framework-for-app-developm>. [Accessed 12.04.2021].
- [40] R. C. Martin, Clean Architecture, USA: Pearson Education, 2018.
- [41] "Common web application architectures," 12.01.2020. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>. [Accessed 20.03.2021].
- [42] W3C, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," 27 April 2007. [Online]. Available: <https://www.w3.org/TR/soap12/>. [Accessed 12.04.2021].
- [43] R. T. Fielding, "Representational State Transfer (REST)," 2000. [Online]. Available: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. [Accessed 12.04.2021].
- [44] GraphQL community, "Security," [Online]. Available: <https://www.howtographql.com/advanced/4-security/>. [Accessed 12.04.2021].
- [45] "Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch and others," altexsoft, 20.07.2019. [Online]. Available: <https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/>. [Accessed 26.04.2021].
- [46] "Datatypes In SQLite Version 3," [Online]. Available: <https://www.sqlite.org/datatype3.html>. [Accessed 07.04.2021].
- [47] M. Fowler, Patterns of Enterprise Application Architecture, Indiana: Addison-Wesley, 2002.

- [48] L. Esposito, "The Mythical Business Layer," *CODE Magazine*, 2014 - November/December.
- [49] Microsoft, "Entity Framework Core Docs," Microsoft, 20.09.2020. [Online]. Available: <https://docs.microsoft.com/en-us/ef/core>. [Accessed 17.04.2021].
- [50] Auth0, "JSON Web Token," Auth0, [Online]. Available: <https://jwt.io/>. [Accessed 17.04.2021].
- [51] NuxtJS, "Nuxt Docs," NuxtJS, [Online]. Available: <https://nuxtjs.org/docs/2.x/get-started/>. [Accessed 17.04.2021].
- [52] A. Lichter, "Organize and decouple your API calls in Nuxt.js," 18.04.2020. [Online]. Available: <https://blog.lichter.io/posts/nuxt-api-call-organization-and-decoupling/>. [Accessed 17.04.2021].
- [53] A. Gupta, "vuex-module-decorators," [Online]. Available: <https://github.com/championswimmer/vuex-module-decorators>. [Accessed 29.04.2021].
- [54] "Vue docs," [Online]. Available: <https://ru.vuejs.org/v2/guide/>. [Accessed 17.04.2021].
- [55] Vuetify, "Vuetify docs," [Online]. Available: <https://vuetifyjs.com/en/components/calendars/>. [Accessed 17.04.2021].
- [56] S. Ivanenko, "Vue: how to use multiple templates in spa," 31.12.2018. [Online]. Available: <https://si-dev.com/ru/blog/vue-multiple-layouts>. [Accessed 12.04.2021].
- [57] "The best development tool for agile teams," Atlassian, [Online]. Available: <https://www.atlassian.com/ru/software/jira>. [Accessed 11.04.2021].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis¹

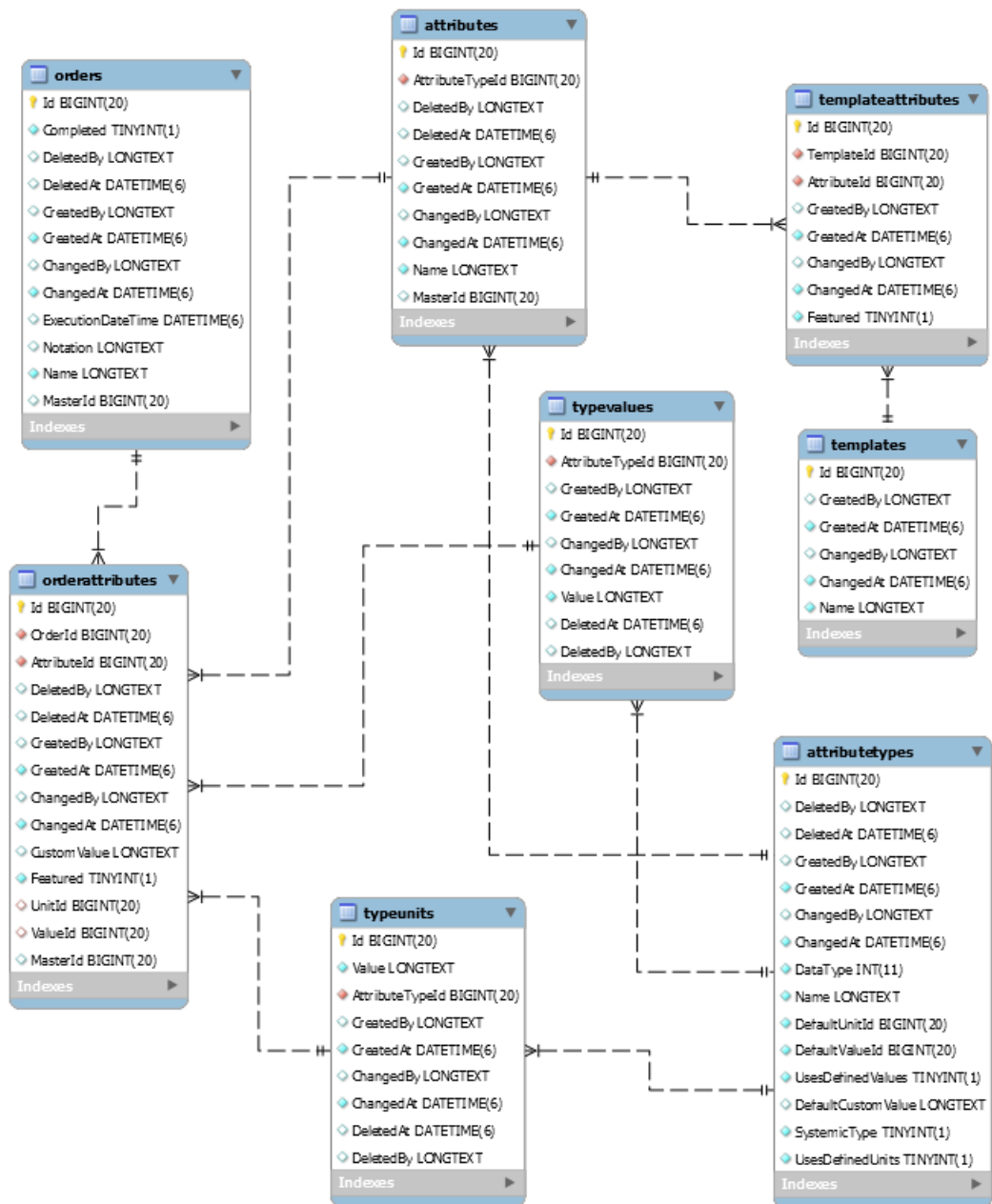
I Aleksandr Ivanov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Product Delivery Accounting Solution for Manufacturing Enterprise”, supervised by Nadežda Furs
 - 1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
 - 1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

27.04.2021

¹ The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – ER Diagram (ERD)



Appendix 3 – Data initialization

```
public static void SeedData(AppDbContext context, ILogger? logger)
{
    logger.LogInformation("SeedData");

    var types = new List<AttributeType>()
    {
        new()
        {
            Name = "Строка",
            DataType = AttributeDataType.String,
            SystemicType = true,
            DefaultCustomValue = ""
        },
        new()
        {
            Name = "Тождество",
            DataType = AttributeDataType.Boolean,
            SystemicType = true,
            DefaultCustomValue = "false"
        },
        new()
        {
            Name = "Целое число",
            DataType = AttributeDataType.Integer,
            SystemicType = true,
            DefaultCustomValue = "0"
        },
        new()
        {
            Name = "Число с плавающей точкой",
            DataType = AttributeDataType.Float,
            SystemicType = true,
            DefaultCustomValue = "0.00"
        },
        new()
        {
            Name = "Дата",
            DataType = AttributeDataType.Date,
            SystemicType = true,
            DefaultCustomValue = ""
        },
        new()
        {
            Name = "Время",
            DataType = AttributeDataType.Time,
            SystemicType = true,
            DefaultCustomValue = "12:00"
        },
        new()
        {
            Name = "Дата со временем",
            DataType = AttributeDataType.DateTime,
            SystemicType = true,
            DefaultCustomValue = "false"
        }
    };

    foreach (var type in types)
    {
        context.AttributeTypes.Add(type);
    }

    context.SaveChanges();
}
```

Appendix 4 – DAL Mapper

```
public class UniversalMapper : IUniversalMapper
{
    private readonly IMapper _mapper;
    private readonly MapperConfiguration _configuration;

    public UniversalMapper()
    {
        _configuration = new MapperConfiguration(config =>
        {
            CreateTwoWayMap<Entities.Attribute, DTO.Attribute>(config);
            CreateTwoWayMap<Entities.AttributeType, DTO.AttributeType>(config);
            CreateTwoWayMap<Entities.AttributeTypeUnit, DTO.AttributeTypeUnit>(config);
            CreateTwoWayMap<Entities.AttributeTypeValue, DTO.AttributeTypeValue>(config);
            CreateTwoWayMap<Entities.Order, DTO.Order>(config);
            CreateTwoWayMap<Entities.OrderAttribute, DTO.OrderAttribute>(config);
            CreateTwoWayMap<Entities.Template, DTO.Template>(config);
            CreateTwoWayMap<Entities.TemplateAttribute, DTO.TemplateAttribute>(config);
            CreateTwoWayMap<Entities.Enums.AttributeDataType,
            DTO.Enums.AttributeDataType>(config);
            config.AllowNullDestinationValues = true;
        });

        _mapper = _configuration.CreateMapper();
    }

    public MapperConfiguration Configuration => _configuration;

    public virtual TOutObject Map<TInObject, TOutObject>(TInObject inObject) =>
        _mapper.Map<TInObject, TOutObject>(inObject);

    private static void CreateTwoWayMap<TFirstObject, TSecondObject>(IProfileExpression config)
    {
        config.CreateMap<TFirstObject, TSecondObject>();
        config.CreateMap<TSecondObject, TFirstObject>();
    }
}
```

Appendix 5 – Units and Values add dialogs

```
<template>
  <v-dialog v-model="active" max-width="600px">
    <v-form class="mt-6" @submit.prevent="onSubmit()" ref="form">
      <v-card>
        <v-card-title>
          <span class="headline">Добавить значение</span>
        </v-card-title>
        <v-card-text>
          <v-container>
            <v-text-field
              v-model="newValue"
              label="Значение поля"
              :rules="rules.value"
            />
          </v-container>
        </v-card-text>
        <v-card-actions>
          <v-spacer></v-spacer>
          <v-btn color="blue darken-1" text @click.stop="onClose()"
            >Отмена</v-btn>
          <v-btn color="blue darken-1" text type="submit">Сохранить</v-btn>
        </v-card-actions>
      </v-card>
    </v-form>
  </v-dialog>
</template>

<script lang="ts">
import { Component, Vue, Prop } from "nuxt-property-decorator";
import { required } from "~/utils/form-validation";

@Component({
  components: {},
})
export default class UnitAddDialog extends Vue {
  @Prop()
  value!: boolean;

  @Prop()
  model!: string;

  rules = {
    value: [required()],
  };

  get newValue() {
    return this.model;
  }

  set newValue(value) {
    this.$emit("change", value);
  }
}

```

```

}

get active() {
  return this.value;
}

set active(value) {
  this.$emit("input", value);
}

onClose() {
  (this.$refs.form as any).resetValidation()
  this.active = false;
}

onSubmit() {
  if ((this.$refs.form as any).validate()) {
    this.$emit("submit");
    this.onClose();
  }
}
}
}
</script>

```

Result:

Создать тип атрибута

Название

Тип данных
Строковый

Добавить значение

Значение поля

ОТМЕНА СОХРАНИТЬ

ОТМЕНА СОЗДАТЬ

Appendix 6 – CustomValueField component

```
<template>
  <v-input v-if="isBoolean">
    <v-spacer></v-spacer>
    <v-switch :label="switchLabel" v-model="fieldValue"></v-switch>
    <v-spacer></v-spacer>
  </v-input>
  <v-text-field
    v-else-if="isString"
    :label="label"
    v-model="fieldValue"
    :rules="rules.string"
  ></v-text-field>
  <v-text-field
    v-else-if="isInteger"
    :label="label"
    v-model="fieldValue"
    type="number"
    :rules="rules.integer"
  ></v-text-field>
  <v-text-field
    v-else-if="isFloat"
    :label="label"
    v-model="fieldValue"
    type="number"
    step=".01"
    :rules="rules.float"
  ></v-text-field>
  <div v-else-if="isDate">
    <DateTimePicker
      v-model="fieldValue"
      :label="label"
      :hasTime="false"
      :rules="rules.date"
    />
  </div>
  <div v-else-if="isTime">
    <DateTimePicker
      v-model="fieldValue"
      :label="label"
      :hasDate="false"
      :rules="rules.time"
    />
  </div>
  <div v-else-if="isDateTime">
    <DateTimePicker
      v-model="fieldValue"
      :label="label"
      :rules="rules.dateTime"
    />
  </div>
</template>
```



```

<script lang="ts">
import { Component, Prop, Vue, Watch } from "nuxt-property-decorator";
import { DataType } from "~/models/Enums/DataType";
import DateTimePicker from "~/components/common/DateTimePicker.vue";
import { required, integer, float } from "~/utils/form-validation";

@Component({
  components: {
    DateTimePicker,
  },
})
export default class CustomValueField extends Vue {
  @Prop()
  dataType!: DataType;

  @Prop()
  value!: string;

  dateTimeTab = null;

  rules = {
    string: [required()],
    date: [required()],
    time: [required()],
    dateTime: [required()],
    integer: [required(), integer()],
    float: [required(), float()],
  };

  @Prop()
  label!: string;

  get fieldValue() {
    switch (this.dataType) {
      case DataType.Boolean:
        return this.value === "true" ? true : false;
      default:
        return this.value;
    }
  }

  set fieldValue(value: any) {
    if (value == null) {
      this.$emit("input", "");
    } else {
      this.$emit("input", String(value));
    }
  }

  get switchLabel() {
    let value = this.fieldValue ? "Да" : "Нет";
    return `${this.label}: ${value}`;
  }

  get isBoolean() {
    return this.dataType === DataType.Boolean;
  }
}

```

```

}

get isString() {
    return this.dataType === DataType.String;
}

get isInteger() {
    return this.dataType === DataType.Integer;
}

get isFloat() {
    return this.dataType === DataType.Float;
}

get isDate() {
    return this.dataType === DataType.Date;
}

get isTime() {
    return this.dataType === DataType.Time;
}

get isDateTime() {
    return this.dataType === DataType.DateTime;
}

@Watch("dataType")
onDataTypeChanged(newType: DataType) {
    let newValue = "";

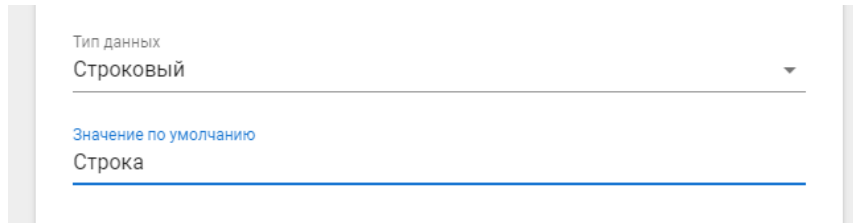
    if (newType === DataType.Integer) {
        newValue = "0";
    } else if (newType === DataType.Float) {
        newValue = "0.00";
    } else if (newType === DataType.Boolean) {
        newValue = "false";
    }

    this.$emit("input", newValue);
}
}
</script>

```

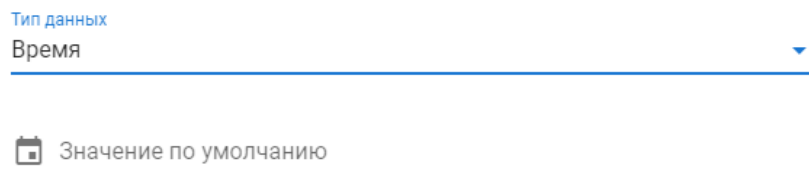
Result:

String datatype field:

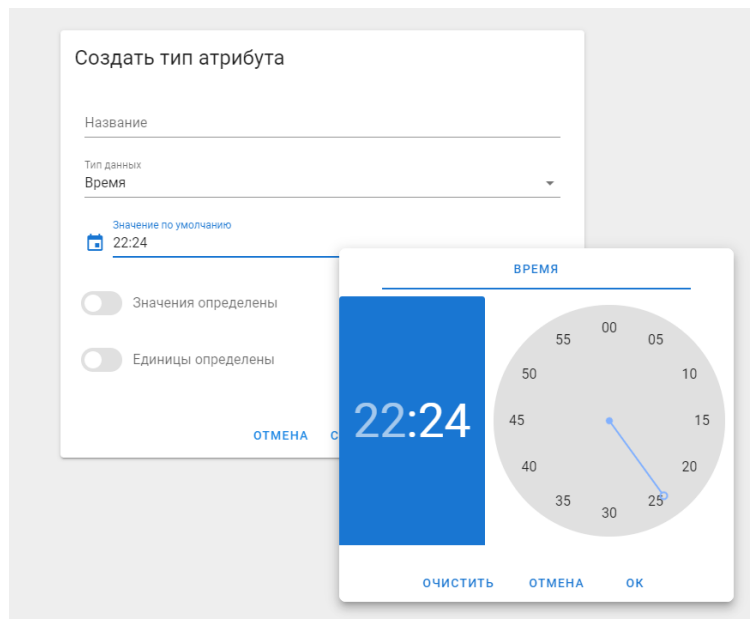


Скриншот интерфейса конфигурации для строкового типа данных. Вверху под заголовком "Тип данных" (Type) выбран вариант "Строковый" (String). Ниже, под заголовком "Значение по умолчанию" (Default value), введено значение "Строка" (String).

Time datatype field:

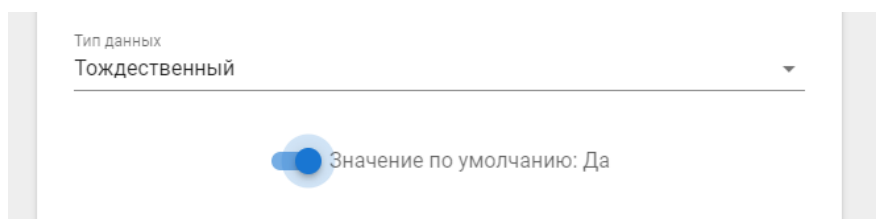


Скриншот интерфейса конфигурации для временного типа данных. Вверху под заголовком "Тип данных" (Type) выбран вариант "Время" (Time). Ниже, под заголовком "Значение по умолчанию" (Default value), введено значение "22:24".



Скриншот диалогового окна выбора времени. В центре отображается цифровое время "22:24" на синем фоне. Справа — аналоговый циферблат с минутными и часовыми стрелками. В нижней части диалогового окна расположены кнопки "ОЧИСТИТЬ" (Clear), "ОТМЕНА" (Cancel) и "ОК" (OK).

Boolean datatype field:



Скриншот интерфейса конфигурации для булевого типа данных. Вверху под заголовком "Тип данных" (Type) выбран вариант "Тожественный" (Boolean). Ниже, под заголовком "Значение по умолчанию" (Default value), включен переключатель, и указано значение "Да" (Yes).

Appendix 7 – Orders filtration dialog

```
<template>
  <v-dialog v-model="active" max-width="600px">
    <v-form class="mt-6" @submit.prevent="onSubmit()" ref="form">
      <v-card>
        <v-card-title>Настроить фильтрацию</v-card-title>
        <v-card-text>
          <v-container>
            <span class="text-body-1">Фильтровать по выполнению</span>
            <v-slider v-if="hasDeadline"
              :tick-labels=["Все', 'Будущие', 'Прошедшие']"
              :max="2"
              step="1"
              tick-size="4"
              v-model.number="overdued"
            >>/v-slider>
            <v-slider
              :tick-labels=["Все', 'Не выполненные', 'Выполненные']"
              :max="2"
              step="1"
              tick-size="4"
              v-model.number="completed"
            >>/v-slider>
            <template v-if="hasDeadline">
              <br />
              <span class="text-body-1">Фильтровать по дате</span>
              <DateTimePicker
                :label="'Начальная дата'"
                v-model="model.startDatetime"
                :forceCentered="true"
              />
              <DateTimePicker
                :label="'Конечная дата'"
                v-model="model.endDatetime"
                :forceCentered="true"
              />
              <br />
              <span class="text-body-1">Указать дату проверки</span>
              <DateTimePicker
                :label="'Дата проверки'"
                v-model="model.checkDatetime"
                :forceCentered="true"
              />
            </template>
          </v-container>
        </v-card-text>
        <v-card-actions>
          <v-spacer></v-spacer>
          <v-btn color="blue darken-1" text @click.stop="onClear()"
            >Очистить</v-btn>
          >
          <v-btn color="blue darken-1" text @click.stop="onClose()"
            >Отмена</v-btn>
        </v-card-actions>
      </v-card>
    </v-form>
  </v-dialog>
</template>
```

```

    >
    <v-btn color="blue darken-1" text type="submit"
      >Применить фильтр</v-btn
    >
    <v-spacer></v-spacer>
  </v-card-actions>
</v-card>
</v-form>
</v-dialog>
</template>

<script lang="ts">
import { Component, Vue, Prop } from "nuxt-property-decorator";
import DateTimePicker from "~/components/common/DateTimePicker.vue";

@Component({
  components: {
    DateTimePicker,
  },
})
export default class FilterDialog extends Vue {
  @Prop()
  value!: boolean;

  @Prop({ default: true })
  hasDeadline!: boolean;

  @Prop()
  filter!: {
    startDatetime?: Date;
    endDatetime?: Date;
    checkDatetime?: Date;
    completed?: boolean;
    overdue?: boolean;
  };

  model: {
    startDatetime?: Date;
    endDatetime?: Date;
    checkDatetime?: Date;
    completed?: boolean;
    overdue?: boolean;
  } = {};

  get active() {
    return this.value;
  }

  set active(value) {
    this.$emit("input", value);
  }

  get completed() {
    return this.model.completed == undefined ? 0 : this.model.completed ? 2 : 1;
  }
}

```

```

set completed(value: number) {
  switch (value) {
    case 1:
      this.model.completed = false;
      break;
    case 2:
      this.model.completed = true;
      break;
    default:
      this.model.completed = undefined;
  }
}

get overdue() {
  return this.model.overdue == undefined ? 0 : this.model.overdue ? 2 : 1;
}

set overdue(value: number) {
  switch (value) {
    case 1:
      this.model.overdue = false;
      break;
    case 2:
      this.model.overdue = true;
      break;
    default:
      this.model.overdue = undefined;
  }
}

onClose() {
  this.active = false;
}

onClear() {
  this.model = {
    startDatetime: undefined,
    endDatetime: undefined,
    completed: undefined,
    checkDatetime: undefined,
  };
}

onSubmit() {
  this.$emit("update:filter", { ...this.model });
  this.onClose();
}

mounted() {
  this.model = { ...this.filter };
}
}
</script>

```

Result:

Настроить фильтрацию

Фильтровать по выполнению

● Все Будущие Прошедшие

● Все Не выполненные Выполненные

Фильтровать по дате

Указать дату проверки

[ОЧИСТИТЬ](#) [ОТМЕНА](#) [ПРИМЕНИТЬ ФИЛЬТР](#)

Appendix 8 – Date picker component

```
<template>
  <div>
    <v-menu
      ref="picker"
      :close-on-content-click="false"
      :return-value.sync="fieldValue"
      rounded="lg"
      min-width="290px"
      absolute
      :content-class="forceCentered ? 'modal-center' : ''"
      z-index="999"
    >
      <template v-slot:activator="{ on, attrs }">
        <v-text-field
          :label="label"
          prepend-icon="mdi-calendar"
          readonly
          v-bind="attrs"
          v-on="on"
          v-model="formattedFieldValue"
          :rules="rules"
        ></v-text-field>
      </template>
    </v-menu>
  </div>
</template>
<v-sheet>
  <v-form @submit.prevent="onSubmit()" ref="form">
    <v-tabs fixed-tabs v-model="dateTimeTab" class="mb-2">
      <v-tab v-if="hasDate || !hasTime">Дата</v-tab>
      <v-tab :disabled="!timeTabEnabled" v-if="hasTime">Время</v-tab>
    </v-tabs>
    <v-tabs-items v-model="dateTimeTab">
      <v-tab-item v-if="hasDate || !hasTime">
        <v-card flat>
          <v-date-picker
            locale="ru"
            :first-day-of-week="1"
            v-model="dateValue"
            landscape
            :allowed-dates="allowedDates"
          ></v-date-picker>
        </v-card>
      </v-tab-item>
      <v-tab-item v-if="hasTime">
        <v-card flat>
          <v-time-picker
            format="24hr"
            landscape
            locale="ru"
            :first-day-of-week="1"
            v-model="timeValue"
          ></v-time-picker>
        </v-card>
      </v-tab-item>
    </v-tabs-items>
  </v-form>
</v-sheet>
```



```

</v-tabs-items>
<v-input :messages="error" :error="!!error" class="mx-2"></v-input>
<v-sheet class="d-flex justify-center">
  <v-btn text large color="primary" @click="onClear()">
    Очистить
  </v-btn>
  <v-btn text large color="primary" @click="onClose()">
    Отмена
  </v-btn>
  <v-btn text large color="primary" type="submit">OK</v-btn>
</v-sheet>
</v-form>
</v-sheet>
</v-menu>
</div>
</template>

```

```

<script lang="ts">
import { Component, Prop, Vue } from "nuxt-property-decorator";

```

```

@Component({})
export default class DateTimePicker extends Vue {
  @Prop({})
  rules: any;

  @Prop({ default: () => [0, 1, 2, 3, 4, 5, 6] })
  allowedDays!: number[];

  @Prop({ default: true })
  hasDate!: boolean;

  @Prop({ default: true })
  hasTime!: boolean;

  error = "";

  @Prop()
  label!: string;

  @Prop()
  value!: string;

  @Prop({default: false})
  forceCentered!: boolean;

  timeValue: null | string = null;
  dateValue: null | string = null;

  dateTimeTab = null;

  get timeTabEnabled() {
    return (this.dateIsCorrect && this.hasTime) || !this.hasDate;
  }

  get formattedFieldValue() {
    if (this.hasDate && this.hasTime) {

```

```

        return (this.$options.filters as any).formatDateTime(this.fieldValue);
    } else if (this.hasTime) {
        return this.fieldValue;
    }
    return (this.$options.filters as any).formatDate(this.fieldValue);
}

get fieldValue() {
    return this.value;
}

set fieldValue(value) {
    this.$emit("input", value);
}

get dateIsCorrect() {
    return this.dateValue != null && /\d{4}-\d{2}-\d{2}/.test(this.dateValue);
}

get timeIsCorrect() {
    return this.timeValue != null && /\d{2}:\d{2}/.test(this.timeValue);
}

allowedDates(val: any) {
    return _.includes(this.allowedDays, this.$moment(val).day());
}

onSubmit() {
    let timeValid = !(this.hasTime && !this.timeIsCorrect);
    let dateValid = !(this.hasDate && !this.dateIsCorrect);

    if (!timeValid && !dateValid) {
        this.error = "Дата и время должны быть указаны";
        return;
    } else if (timeValid && !dateValid) {
        this.error = "Дата должна быть указана";
        return;
    } else if (!timeValid && dateValid) {
        this.error = "Время должно быть указано";
        return;
    } else {
        if (this.hasDate && this.hasTime) {
            (this.$refs.picker as any).save(this.dateValue + "T" + this.timeValue);
        } else if (!this.hasDate && this.hasTime) {
            (this.$refs.picker as any).save(this.timeValue);
        } else {
            (this.$refs.picker as any).save(this.dateValue);
        }
    }
}

onClose() {
    (this.$refs.picker as any).isActive = false;
}

onClear() {

```

```
    this.dateValue = null;
    this.timeValue = null;
    this.dateTimeTab = null;
    this.error = "";


    (this.$refs.picker as any).save(null);
  }

  mounted() {
    this.timeValue = "12:00";
  }
}
</script>
```

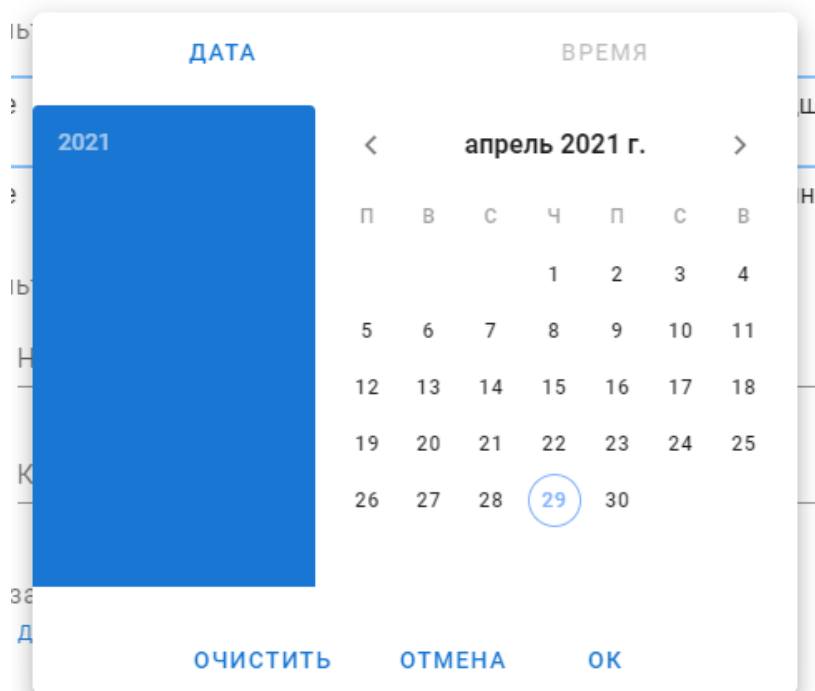
Results:

Field without value:

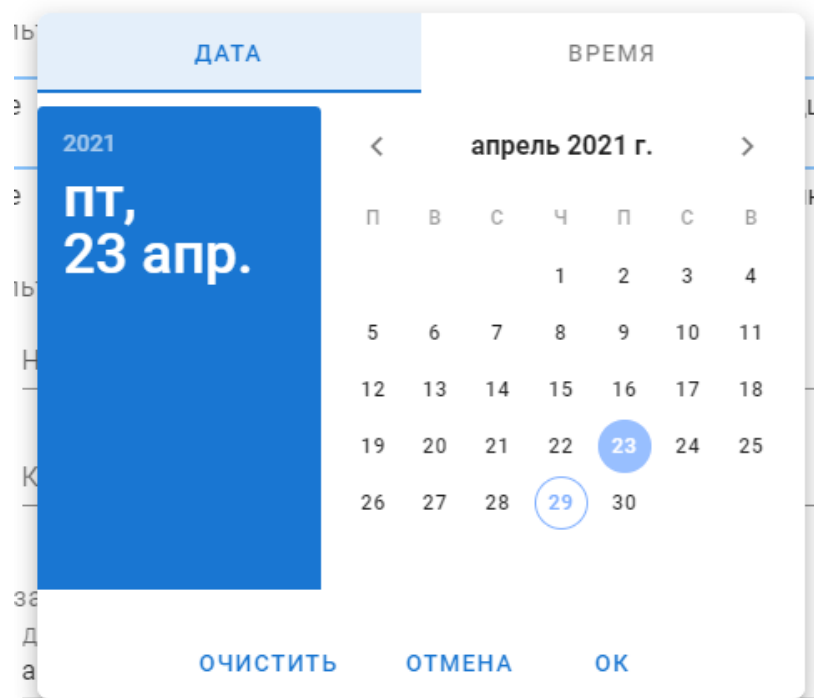
Указать дату проверки

 Дата проверки

Menu appears:




Date selected:



Field with value:

Указать дату проверки

Дата проверки

 апрель 23-го 2021, 17:26

Appendix 9 – Attribute and its value select components

```
<template>
  <v-autocomplete
    name="attribute"
    v-model="attribute"
    :items="availableTypes"
    :loading="isLoading"
    :search-input.sync="searchKey"
    hide-no-data
    item-text="name"
    item-value="id"
    label="Атрибут"
    placeholder="Начните ввод для поиска"
    prepend-icon="mdi-database-search"
    :rules="rules.attribute"
    return-object
  >
</v-autocomplete>
</template>

<script lang="ts">
import { Component, Prop, Vue, Watch } from "nuxt-property-decorator";
import { SortOption } from "~/models/Enums/SortOption";
import { attributesStore } from "~/store";

@Component({})
export default class AttributeSeselect extends Vue {
  @Prop()
  value!: { id: number; name: string };

  searchKey = "";
  isLoading = false;

  rules = {
    attribute: [
      (value?: { id: number; name: string }) =>
        (value != null && value.id > 0) || `Данное поле обязательно`,
    ],
  };

  get attribute() {
    return this.value;
  }

  set attribute(value) {
    if (value != null) {
      this.$emit("input", value);
    }
  }

  get availableTypes() {
    return attributesStore.attributes;
  }
}
```

```

@Watch("searchKey")
onFetchRequired() {
  this.isLoading = true;
  attributesStore
    .getAttributes({
      pageIndex: 0,
      byName: SortOption.False,
      byType: SortOption.False,
      searchKey: this.searchKey,
    })
    .then((_) => {
      this.isLoading = false;
    });
}
}
</script>

```

```

<template>
  <v-row v-if="fetched">
    <v-col>
      <CustomValueField
        :dataType="attributeType.dataType"
        v-model="customValue"
        :label="label"
        v-if="!attributeType.usesDefinedValues"
        class="ma-0"
      />
      <v-select
        v-else
        v-model="valueId"
        :items="attributeType.values"
        item-text="value"
        item-value="id"
        :label="label"
        class="ma-0"
      >>/v-select>
    </v-col>
    <v-col v-if="attributeType.usesDefinedUnits">
      <v-select
        v-model="unitId"
        :items="attributeType.units"
        item-text="value"
        item-value="id"
        label="Ед. измерения"
        class="ma-0"
      >>/v-select>
    </v-col>
  </v-row>
</template>

```

```

<script lang="ts">
import { Component, Prop, Vue, Watch } from "nuxt-property-decorator";
import { attributeTypesStore } from "~/store";
import CustomValueField from "~/components/common/CustomValueField.vue";

```

```

import { AttributeTypeDetailsGetDTO } from "~/models/AttributeTypeDTO";

@Component({
  components: {
    CustomValueField,
  },
})
export default class AttributeValueSellect extends Vue {
  @Prop()
  value!: {
    customValue: string;
    valueId: null | number;
    unitId: null | number;
  };

  @Prop({ default: null })
  typeId!: number | null;

  @Prop({ default: "Значение" })
  label!: string;

  fetched = false;

  attributeType!: AttributeTypeDetailsGetDTO;

  get customValue() {
    return this.value.customValue;
  }

  set customValue(value) {
    this.$emit("input", { ...this.value, customValue: value });
  }

  get valueId() {
    return this.value.valueId;
  }

  set valueId(value) {
    this.$emit("input", { ...this.value, valueId: value });
  }

  get unitId() {
    return this.value.unitId;
  }

  set unitId(value) {
    this.$emit("input", { ...this.value, unitId: value });
  }

  mounted() {
    this.fetchAttributeType();
  }

  validateValue(id: number | null) {
    return (
      id != null &&

```

```

        _.includes(
          _.map(this.attributeType.values, (value) => value.id),
          id
        )
      );
    }

    validateUnit(id: number | null) {
      return (
        id != null &&
        _.includes(
          _.map(this.attributeType.units, (unit) => unit.id),
          id
        )
      );
    }

    @Watch("typeId")
    fetchAttributeType(): void {
      this.fetched = false;
      if (this.typeId) {
        attributeTypesStore.getAttributeType(this.typeId).then((succeeded) => {
          if (succeeded) {
            this.attributeType = attributeTypesStore.selectedAttributeType!;

            let valueId = this.value.valueId;
            let unitId = this.value.unitId;
            let customValue = this.value.customValue;

            if (this.attributeType != null) {
              if (
                this.attributeType.usesDefinedValues &&
                !this.validateValue(valueId)
              ) {
                valueId = this.attributeType.defaultValueId;
              } else if (customValue.length == 0) {
                customValue = this.attributeType.defaultCustomValue;
              }
              if (
                this.attributeType.usesDefinedUnits &&
                !this.validateUnit(unitId)
              ) {
                unitId = this.attributeType.defaultUnitId;
              }
            }

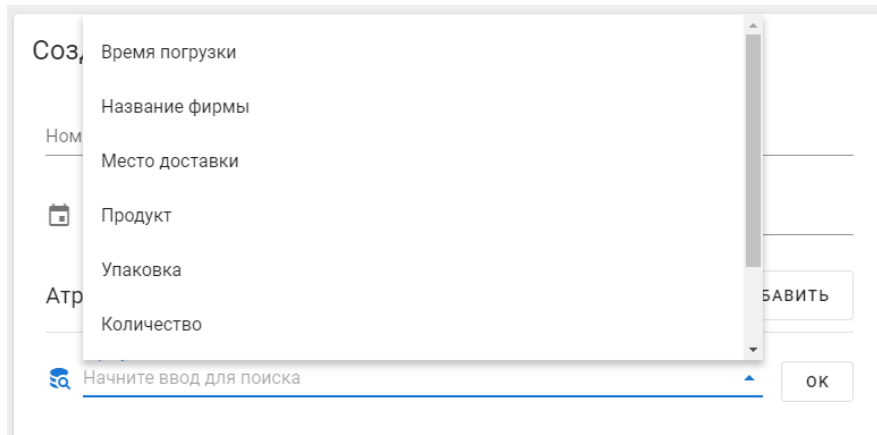
            this.$emit("input", { valueId, unitId, customValue });

            this.fetched = true;
          }
        });
      }
    }
  }
}
</script>

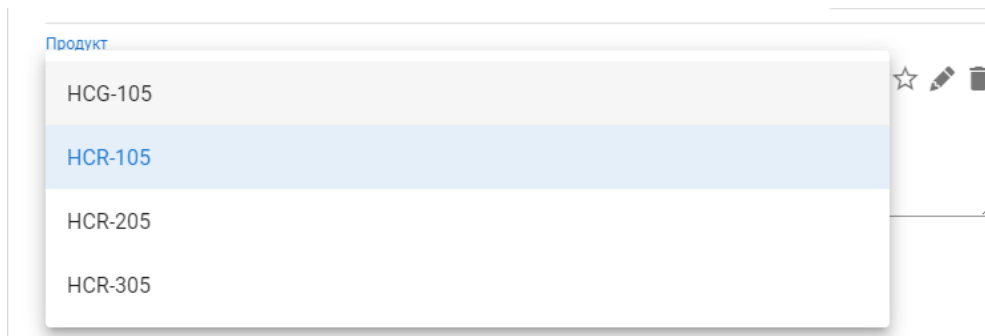
```


Result:

Attribute select:



Attribute selected, now value is select:



Value selected:



Appendix 10 – Attribute type views

Index page with all types on second page:

Название ↑	Тип
тест изменения	с единицами измерения с определенными значениями
Тожество	СИСТЕМНЫЙ
Фирма-клиент	с определенными значениями
Целое число	СИСТЕМНЫЙ
Число с плавающей точкой	СИСТЕМНЫЙ

Attribute type details page:

Название: Фирма-клиент
Формат данных: Строковый
Количество использований: 1

ИЗМЕНИТЬ УДАЛИТЬ

Допустимые значения

- AZAZCompany LTD
- BlablaCompany LTD (по умолчанию)
- Plague Inc

Attribute type edit page:










Изменить тип атрибута

Название
Фирма-клиент

Тип данных
Строковый

[ДОБАВИТЬ](#)


Допустимые значения

AZAZCompany LTD	  
BlablaCompany LTD	  
Plague Inc	  

[ОТМЕНА](#) [СОХРАНИТЬ](#)

Attribute type delete confirmation:

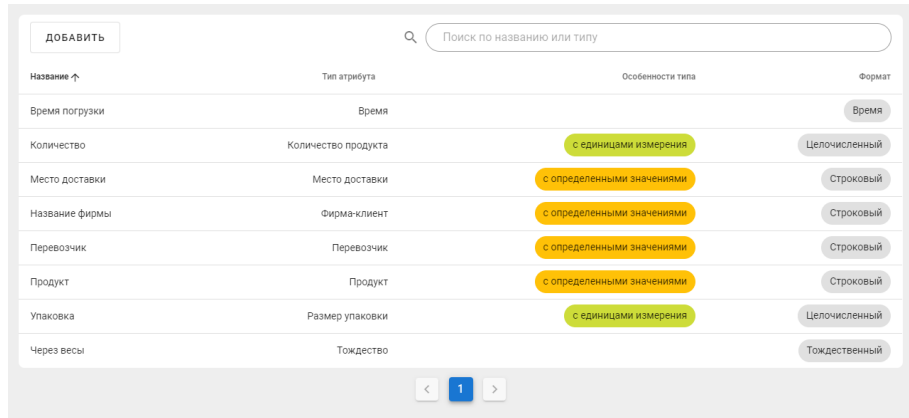
Вы уверены, что хотите удалить этот тип?

 Данное действие не может быть отменено!

[ОТМЕНА](#) [УДАЛИТЬ](#)

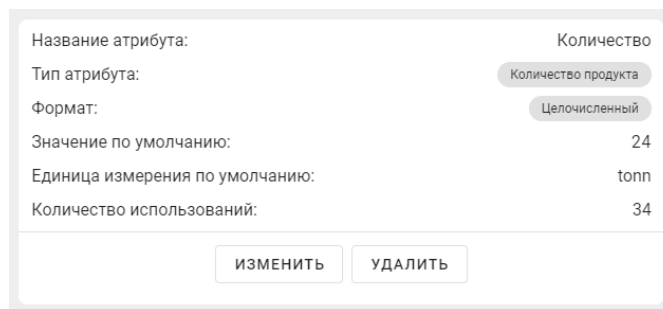
Appendix 11 – Attributes views

Attributes index page:



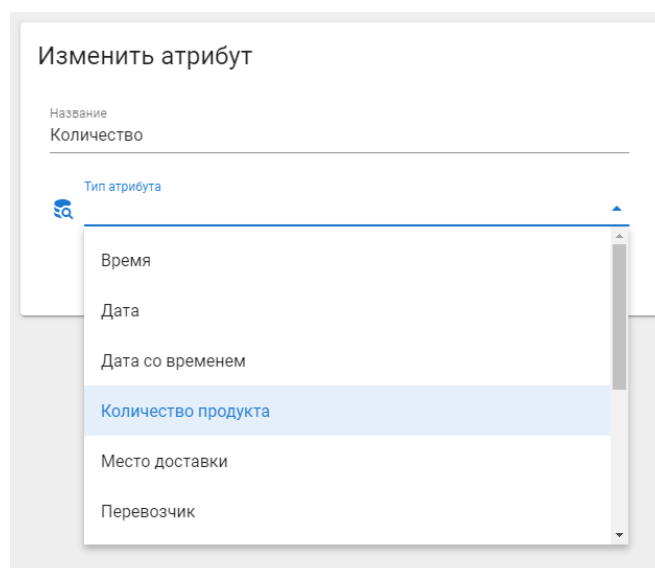
Название ↑	Тип атрибута	Особенности типа	Формат
Время погрузки	Время		Время
Количество	Количество продукта	с единицами измерения	Целочисленный
Место доставки	Место доставки	с определенными значениями	Строковый
Название фирмы	Фирма-клиент	с определенными значениями	Строковый
Перевозчик	Перевозчик	с определенными значениями	Строковый
Продукт	Продукт	с определенными значениями	Строковый
Упаковка	Размер упаковки	с единицами измерения	Целочисленный
Через весы	Товжество		Товжественный

Attribute details page:



Название атрибута:	Количество
Тип атрибута:	Количество продукта
Формат:	Целочисленный
Значение по умолчанию:	24
Единица измерения по умолчанию:	topp
Количество использований:	34

Attribute edit page with type being selected:



Изменить атрибут

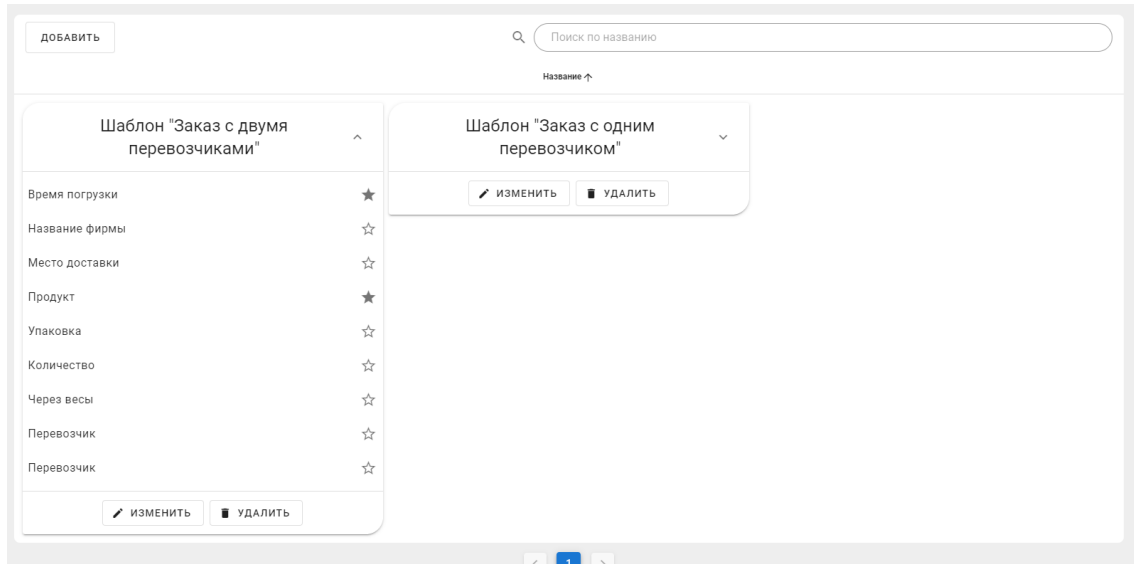
Название
Количество

Тип атрибута

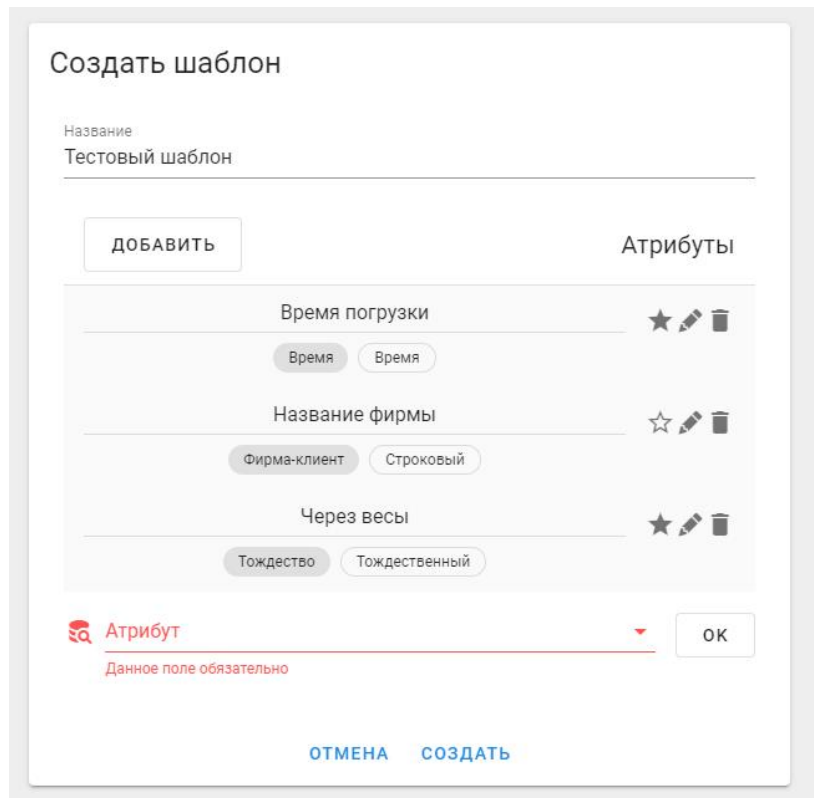
- Время
- Дата
- Дата со временем
- Количество продукта**
- Место доставки
- Перевозчик

Appendix 12 – Templates views

Template index page, with one template being inspected:



Template edit page, with validation error occurs:



Appendix 13 – Orders views

Orders with date index page:

ЗАКАЗЫ С ДАТОЙ		ЗАКАЗЫ БЕЗ ДАТЫ	
ДОБАВИТЬ	ОТЧЕТ	🔍	Поиск по номеру заказа
Номер заказа	Дата	Статус	
23231212312	апрель 1-го 2021, 12:00	Выполнен	
23231212312	апрель 12-го 2021, 12:00	Выполнен	
23231212312	апрель 12-го 2021, 12:00	Просрочен	
23231212312	апрель 13-го 2021, 12:00	Выполнен	
23231212312	апрель 13-го 2021, 12:00	Просрочен	
23231212312	апрель 13-го 2021, 12:00	Просрочен	
23231212312	апрель 13-го 2021, 12:00	Выполнен	
23231212312	апрель 13-го 2021, 12:00	Выполнен	
23231212312	апрель 14-го 2021, 12:00	Выполнен	
23231212312	апрель 14-го 2021, 12:00	Выполнен	
23231212312	апрель 14-го 2021, 12:00	Выполнен	

Orders without date index page:

ДОБАВИТЬ		ОТЧЕТ		🔍	Поиск по номеру заказа
Номер заказа ↑	Статус				
4575343473434	Запланирован				

Order create page, with template already applied:

Применить шаблон

Заказ с одним перевозчиком

ПРИМЕНИТЬ

Создать заказ

Номер заказа

Дата исполнения

Атрибуты заказа

ДОБАВИТЬ

Время погрузки: 12:00

Название фирмы: **BlablaCompany LTD**

Место доставки: **Marseille, France**

Calendar view, with one order being inspected:

ДОБАВИТЬ ЗАКАЗ ОТЧЕТ Поиск по номеру заказа апрель 2021

29.03.2021	30.03.2021	31.03.2021	01.04.2021	02.04.2021
			время погрузки: 08:00, продукт: hcr-105	
05.04.2021	06.04.2021	07.04.2021	08.04.2021	09.04.2021
12.04.2021	13.04.2021	14.04.2021	15.04.2021	16.04.2021
время погрузки: 08:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105	
19.04.2021	20.04.2021	21.04.2021	22.04.2021	23.04.2021
время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105
26.04.2021	27.04.2021	28.04.2021	29.04.2021	30.04.2021
время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105	время погрузки: 12:00, продукт: hcr-105 время погрузки: 12:00, продукт: hcr-105	время погрузки: 12:00, продукт: hcr-105 время погрузки: 08:00, продукт: hcr-105	время погрузки: 08:00, продукт: hcr-105

Номер заказа: 23231212312
Дата заказа: апрель 22-го 2021, 12:00
Состояние: **Выполнен**

Время погрузки: 08:00
Название фирмы: SomeCompany LTD
Место доставки: Tallinn, Estonia
Продукт: HCR-105
Упковка: 1 tonn bags
Количество: 32 tonn
Через весы: Нет
Перевозчик: BESTLOGISTICS LTD

ОТМЕТИТЬ ПОДРОБНЕЕ ИЗМЕНИТЬ

Calendar view, with searching applied and one order being inspected:

ДОБАВИТЬ ЗАКАЗ ОТЧЕТ 🔍 2322 апрель 2021 < >

29.03.2021 30.03.2021 31.03.2021 01.04.2021 02.04.2021

05.04.2021 06.04.2021 07.04.2021 08.04.2021 09.04.2021

12.04.2021 13.04.2021 14.04.2021 15.04.2021 16.04.2021

19.04.2021 20.04.2021 21.04.2021 22.04.2021 23.04.2021

26.04.2021 27.04.2021 28.04.2021 29.04.2021 30.04.2021

Номер заказа: 23231212312
 Дата заказа: апрель 22-го 2021, 12:00
 Состояние: **Выполнен**
 Время погрузки: 08:00
 Название фирмы: SomeCompany LTD
 Место доставки: Tallinn, Estonia
 Продукт: HCR-105
 Упаковка: 1 tonn bags
 Количество: 32 tonn
 Через весы: Нет
 Перевозчик: BESTLOGISTICS LTD

ОТМЕТИТЬ ПОДРОБНЕЕ ИЗМЕНИТЬ

Order with date index page filtrated by “future completed” orders only:

ЗАКАЗЫ С ДАТОЙ ЗАКАЗЫ БЕЗ ДАТЫ

ДОБАВИТЬ

Номер заказа

23231212312

23231212312

Статус

Выполнен

Выполнен

Настроить фильтрацию

Фильтровать по выполнению

Все Будущие Прошедшие

Все Не выполненные Выполненные

Фильтровать по дате

📅 Начальная дата

📅 Конечная дата

Указать дату проверки

📅 Дата проверки

ОЧИСТИТЬ ОТМЕНА ПРИМЕНИТЬ ФИЛЬТР

Order with date index page filtrated by “future” orders only:

ДОБАВИТЬ

Номер заказа

31231312312312

23231212312

23231212312

23231212312

Статус

Запланирован

Выполнен

Выполнен

Запланирован

Настроить фильтрацию

Фильтровать по выполнению

Все Будущие Прошедшие

Все Не выполненные Выполненные

Фильтровать по дате

📅 Начальная дата

📅 Конечная дата

Указать дату проверки

📅 Дата проверки

ОЧИСТИТЬ ОТМЕНА ПРИМЕНИТЬ ФИЛЬТР

Appendix 14 – Users views

Current user details page:

Мои данные

Пользователи

Имя: Супер
Фамилия: Администратор
Эл.адрес: root@root.com
Роль: Супер администратор

Данный пользователь не может быть изменен

Another user being inspected:

← НАЗАД

Мои данные

Пользователи

Имя: Александр
Фамилия: Иванов
Эл.адрес: admin@admin.com
Роль: Администратор

ИЗМЕНИТЬ ДАННЫЕ

ИЗМЕНИТЬ РОЛЬ

ИЗМЕНИТЬ ПАРОЛЬ

УДАЛИТЬ

Another user role is being changes because current user is “Root”

СУПЕР АДМИНИСТРАТОР

КАЛЕНДАРЬ ЗАКАЗЫ АТРИБУТЫ ТИПЫ АТРИБУТОВ ШАБЛОНЫ ПОЛЬЗОВАТЕЛИ ВЫЙТИ

Мои данные

Пользователи

← НАЗАД

Имя: Александр
Фамилия: Иванов
Эл.адрес: admin@admin.com
Роль: Администратор

ИЗМЕНИТЬ ДАННЫЕ

ИЗМЕНИТЬ РОЛЬ

Сменить роль пользователю "Александр Иванов"

Пользователь

Администратор

ОТМЕНА СОХРАНИТЬ

Appendix 15 – Example of generated report

Дата: апрель 1-го 2021	
Номер заказа:	23231212312
Дата заказа:	апрель 1-го 2021, 12:00
Состояние:	Выполнен
Время погрузки:	08:00
Название фирмы:	SomeCompany LTD
Место доставки:	Tallinn, Estonia
Продукт:	HCR-105
Упаковка:	1 tonn bags
Количество:	32 tonn
Через весы:	Нет
Перевозчик:	BESTLOGISTICS LTD
Дата: апрель 12-го 2021	
Номер заказа:	23231212312
Дата заказа:	апрель 12-го 2021, 12:00
Состояние:	Выполнен
Время погрузки:	08:00
Название фирмы:	SomeCompany LTD
Место доставки:	Tallinn, Estonia
Продукт:	HCR-105
Упаковка:	1 tonn bags
Количество:	32 tonn
Через весы:	Нет
Перевозчик:	BESTLOGISTICS LTD
Дата: апрель 13-го 2021	
Номер заказа:	23231212312
Дата заказа:	апрель 13-го 2021, 12:00
Состояние:	Просрочен
Время погрузки:	08:00
Название фирмы:	SomeCompany LTD
Место доставки:	Tallinn, Estonia
Продукт:	HCR-105
Упаковка:	1 tonn bags
Количество:	32 tonn
Через весы:	Нет
Перевозчик:	BESTLOGISTICS LTD

Дата: апрель 12-го 2021

Номер заказа:	23231212312
Дата заказа:	апрель 12-го 2021, 12:00
Состояние:	Выполнен
Время погрузки:	08:00
Название фирмы:	SomeCompany LTD
Место доставки:	Tallinn, Estonia
Продукт:	HCR-105
Упаковка:	1 tonn bags
Количество:	32 tonn
Через весы:	Нет
Перевозчик:	BESTLOGISTICS LTD
Номер заказа:	23231212312
Дата заказа:	апрель 12-го 2021, 12:00
Состояние:	Просрочен
Время погрузки:	08:00
Название фирмы:	SomeCompany LTD
Место доставки:	Tallinn, Estonia
Продукт:	HCR-105
Упаковка:	1 tonn bags
Количество:	32 tonn
Через весы:	Нет
Перевозчик:	BESTLOGISTICS LTD