Assignment of master's thesis

Title: Workflow: Web Application Implementing Company Processes using State Machines
Student: Bc. Petr Svoboda
Supervisor: Ing. Marek Suchánek
Study program: Informatics
Branch / specialization: Web and Software Engineering, specialization Web Engineering
Department: Department of Software Engineering
Validity: until the end of summer semester 2021/2022

Instructions
Optimizing business processes is a crucial activity for every company that wants to increase profit. Software tools are helping significantly in this effort in both controlling the workflows and also providing reports for further optimization. The goal of this thesis is to design and implement a web application as a workflow management system for Sécheron Hasler CZ spol. s r.o.

- Analyze current company processes and structure, use conceptual models if suitable.
- Briefly research existing workflow management systems.
- Design a system that is going to support all current processes (currently managed using email, excel sheets, etc.). Consider integration with other systems.
- Implement the application, including permission system, reporting, and ability to define and adjust processes. The UI should be intuitive for non-technical users.
- Test and document the resulting application.
- Discuss the benefits for the company, gather user feedback, and propose future development.

Master’s thesis

Workflow: Web Application Implementing Company Processes using State Machines

Bc. Petr Svoboda

Department of Software Engineering
Supervisor: Ing. Marek Suchánek

May 5, 2021
I would like to thank all my colleagues who assisted me in the process of creating the application. Especially I would like to thank Martin Křižka, for guiding me through the gathering of the requirements from the various departments, and for taking care of the company politics side of things. Also, I would like to thank Ing. Jan Beran, for always being there for me when I needed someone to consult about technical solutions, and for saying no when I could not do so myself. And last but not least, I would like to thank my thesis supervisor Ing. Marek Suchánek, for his guidance and active approach.
I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No.121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 5, 2021                      .................
Czech Technical University in Prague
Faculty of Information Technology
© 2021 Petr Svoboda. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author’s permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Podnikové procesy velkých firem jsou často komplikované. Složitost těchto procesů může mít více příčin: interakce více externích systémů v jejich rámci, neznalost účastníků celého jejich průběhu nebo setkání procesu s událostmi, které mohou způsobit nedefinované chování. Tato práce řeší zmíněné problémy vymodelováním podnikových procesů jako stavových automatů, konkrétně procesů tvorby žádostí různých typů (jako je například nákup nebo nepřítomnost). Dále tyto procesy digitalizuje a vylepšuje formou webové aplikace, která využívá vymodelovaných stavových automatů k jejich simulaci. Výsledné řešení je schopné výrazně urychlit čas, za který se žádost dokončí, a navíc poskytuje kompletní dohledatelnost akcí, které účastníci procesu provedli.

Klíčová slova firemní proces, podnikový proces, stavový automat, webová aplikace, digitalizace, React, Node.js, PostgreSQL
Abstract

Business processes of large companies can be often complex. The complexity of the processes can originate from multiple sources: multiple external systems are interacting within them, actors are not familiar with the whole scope of them, or they might encounter events which lead to undefined behaviour. This thesis solves mentioned problems by modelling the company processes as state machines, specifically the processes of creating requests of various types (for example purchase or leave of absence). Then it digitalizes and improves on the processes in a form of web application, which uses the modelled state machines for process simulation. The resulting solution is able to greatly reduce the time it takes for requests to complete and additionally it provides full traceability of actions made by the actors within the processes.

Keywords  company process, business process, state machine, web application, digitalization, React, Node.js, PostgreSQL
Contents

Introduction

1 State-of-the-art

1.1 State machines ........................................... 3
1.2 Alternative approaches .................................. 8

2 Analysis

2.1 Requirements .............................................. 9
2.1.1 Collecting information ................................. 9
2.1.2 Requirement specification .............................. 13
2.2 Use cases .................................................. 16
2.3 Requirement coverage ................................... 16
2.4 Competition analysis .................................... 33
2.4.1 Competing solutions ................................. 34
2.4.2 Summary .............................................. 39

3 Design

3.1 Technologies .............................................. 41
3.1.1 PostgreSQL ............................................ 41
3.1.2 TypeScript ............................................ 42
3.1.3 Node .................................................. 42
3.1.4 React ................................................ 44
3.2 Architecture .............................................. 45
3.2.1 Database .............................................. 46
3.2.2 Server ................................................. 47
3.2.3 Client ................................................. 48
3.2.4 Micro-services ....................................... 48
3.3 Item types ................................................. 49
3.3.1 Prototype ............................................. 49
3.3.2 Request ............................................... 53
## List of Figures

1.1 Example: Simple approval process ........................................... 3  
1.2 Example: Expanded approval process ........................................ 4  
1.3 State machine, states .......................................................... 5  
1.4 State machine, event ........................................................... 5  
1.5 Example: Finite state machine ................................................ 6  
1.6 State machine, guards .......................................................... 7  
1.7 State machine, actions .......................................................... 7  

2.1 Actors ................................................................................. 17  

3.1 System architecture ............................................................. 46  
3.2 Prototype item, process ........................................................ 51  
3.3 Request item, main process .................................................... 55  
3.4 Request item, approval process .............................................. 56  
3.5 Request item, revision process ............................................... 57  
3.6 Service item, process ............................................................. 59  
3.7 Home office item, process ....................................................... 61  
3.8 Vacation item, process ........................................................... 63  
3.9 ER diagram, base entities ....................................................... 65  
3.10 ER diagram, Class Table Inheritance of the item types .......... 66  
3.11 ER diagram, prototype item entities ....................................... 66  
3.12 ER diagram, request and service item entities ....................... 67  
3.13 ER diagram, home office and vacation item entities ............... 68  
3.14 ER diagram, user related entities ......................................... 69  

4.1 State machine: code suggestions of possible states ............... 85  
4.2 Mailer, text email ................................................................. 89  
4.3 Mailer, table email ............................................................... 90  
4.4 Authentication, client ........................................................... 91  
4.5 Sequence diagram: authentication, login ................................ 92  
4.6 Sequence diagram: authentication, verify ............................... 92
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Requirements covered by the use cases</td>
<td>33</td>
</tr>
<tr>
<td>2.2</td>
<td>Competition rating</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>Heuristic analysis, found issues</td>
<td>121</td>
</tr>
<tr>
<td>5.2</td>
<td>Usability testing, user interface issues</td>
<td>125</td>
</tr>
<tr>
<td>5.3</td>
<td>Usability testing, bugs and system issues</td>
<td>126</td>
</tr>
</tbody>
</table>
List of Algorithms

4.1 State machine: event handling .................. 84
4.2 Item creation ....................................... 102
4.3 Item editing ........................................ 103
4.4 Item state machine event handling ............... 104
List of Listings

4.1 State machine data format: Base node .................................. 82
4.2 State machine data format: State node ................................. 82
4.3 State machine data format: Root node ................................. 83
4.4 Server module: QueryService ........................................... 86
4.5 Server module: Controller .............................................. 86
4.6 Server module: Router .................................................. 87
4.7 Utility function signatures ................................................ 89
4.8 Remote database, creating a foreign server ......................... 96
4.9 Remote database, creating a user mapping ......................... 96
4.10 Remote database: creating a foreign table ......................... 96
4.11 Remote database: scheduled jobs ................................... 98
4.12 Client permission helpers, hook and component ................. 112
4.13 Deployment, Dockerfile structure .................................. 114
4.14 Deployment, production container definition .................... 114
4.15 Deployment, production CI/CD deploy stage ..................... 115
E.1 API definition: Mailer .................................................. 165
E.2 API definition: Authentication ........................................ 167
Introduction

This thesis focuses on the topic of digitalization, more specifically on digitalization of the internal processes for the company Sécheron Hasler CZ, spol. s.r.o. The mentioned processes are: request for a purchase and request for a leave of absence. In the context of the company policy, responsibilities and permissions of the departments and the actors (employees performing actions tied to the process) the complexity of these processes is high. This resulted in a search for a better solution by the company.

The task of creating the improved solution is made more difficult, because the company is primarily an industrial manufacturer, and it is slow at picking up new digital trends (which has been improving in the recent years). There would be no need for this project if it was possible to extend the current ERP system, which already interacts with parts of the processes. Regretfully, this system is very old and its maintenance is expensive. Implementing something new in it would have much larger costs and time spent on it, than creating a new solution. This is caused by the non-existence of a staff experienced enough with the system’s development.

The other difficult part of the task is the actual users. They form a heterogeneous group. Many of them will have no problem adapting to the new solution. However, a substantial number of them may have some issues with it as they do not have enough experience with interacting with computer systems (this is author’s observation after working as an IT administrator and technical support for the same company). This means that there will have to be an extra attention paid to the application usability and the final digital process has to be understandable.

With the main difficulties defined, what will the advantages be? The main one is a request traceability. The current pain point for the company is that in many instances of the requests, nobody knows what stage of the process the requests are in. This is caused by an absence of records of the process changes. There is also an issue with the request resolve time (meaning how long it takes from the creation of the request to its completion). The final
application can help with time management. And with all the data that will be collected and saved, the process can be then improved.

In the following chapters the whole process of creating the final solution will be described. Firstly, state machines will be described as they will be used in the modelling of the processes. Secondly, the current processes, chosen solution and caveats that could be encountered have to be properly analysed. After analysis, the application can be designed. Next the implementation will be documented. Then there will follow testing of various approaches, so in the final chapters, the proper evaluation and conclusion can be drawn.
Chapter 1

State-of-the-art

This chapter will build a basic foundation of terms associated with state machines which will be referenced in the thesis. This will prove beneficial in later chapters, when designing and implementing the processes. After that, there will be briefly covered other alternatives for modelling processes.

1.1 State machines

In business processes, complexity arrives with every new step and condition. The process of approval can be used as an example. Simple version of it could be as follows: a request has to be approved, then somebody either approves it or rejects it, the end.

![Diagram of state machine

Figure 1.1: Example: Simple approval process

Now to expand on that process, conditions could be added, such as: is item ready to be approved, does the item actually need approval, etc. To further expand it, there could be multiple approvals required based on specific conditions. It is clear that the different possible paths that could be taken in the process are multiplying rapidly.

The different paths are not the only things that are expanding the complexity of the process. There might be a need to perform specific actions after entering a specific state or after fulfilling one of the conditions.
1. STATE-OF-THE-ART

What started as a simple example is now expanded into the complex process even when the expectations from it were reasonable. The question is, how to model this process in a digital form? Naive approach could be to simply have conditions for all the possibilities at places where the application interacts with the process. That would clearly result in having many conditions (for checking what UI elements to display, or what should the next state of the process be, etc.) just for the expanded approval process mentioned above. The same would apply when considering other systems that would have to be tied to state of the process (permissions, notifications, etc.). To further complicate the problem, the process evolution also has to be taken into consideration. The process can change in time and the more systems depend on its status, the more changes have to be made when the process changes.

In this thesis, there will be encountered even more complex processes, where the approval will form only a part of the whole process. So there is a clear need for a solution that will make describing and interacting with the process as easy as possible.

Thankfully for this exact problem exists a perfect solution, state machines as defined in the Introduction to automata theory, languages, and computation [1].
1.1. State machines

1.1.1 Formal definition

Book *Practical UML statecharts in C/C++*  
will be used to define state machine and other related terms. Excluding that, it will be helpful when designing process diagrams and state machine implementation as it contains useful practical examples.

1.1.1.1 State

Describes a specific *state* in which the system can be at given time. For example for a simple electric appliance, its states could be **on** and **off**. Having set of determined states of a system enables efficiently capturing the system’s history. There are two special types of states: *initial* and *final*. *Initial* state indicates where the process starts, there always has to be one (or more for parallel state machines). *Final* state indicates where the process ends, there can be none or multiple.  

![Figure 1.3: States, from left to right: on, off (extended variant), initial state, final state](image)

1.1.1.2 Event

In the most general term, an event is an occurrence in time and space that has significance to the process. For example, in context of the electrical appliance, the events could be **turn_on** and **turn_off**. An event can have associated parameters, allowing the event instance to convey not only the occurrence of some incident, but also quantitative information regarding that occurrence. For example, the **turn_on** event can have a date stamp as a parameter.  

![Figure 1.4: Event turn_on leading from on state to off state](image)

1.1.1.3 Finite state machine

*Finite state machine* (FSM) is a model which can be used for specification of systems or processes. The process starts in initial state (**off** in context of the appliance example). Then based on the event, from a predefined set
of events, that is sent to the FSM, the FSM can transition from one state to another (on turn_on event it transitions to on state). One or more of the states can be labelled as final states which symbolize an end to the process. [5]

This model enables efficient specification of constraints in complex processes. When the process is in some state, there is defined a set of events it is supposed to react to. After one of these events occurs, there is defined a clear transition to the next state. This drastically reduces complexity of logic tied to the state as there is always a deterministic outcome. [5]

Transition Switching from one state to another based on an event is called state transition, and the event that causes it is called the triggering event (trigger). It should be noted that there can be specified an automatic transition, which is fired right when a state is entered. Basically, the FSM receives empty event. [6]

![Figure 1.5: Example: Finite state machine](image)

1.1.1.4 Extended state machine

In FSMs all the distinct states of process should be represented as separate values. However, in some cases, this could lead to FSM with hundreds of different states (the appliance could behave differently based on time it was turned on, for each different behaviour, there would have to be a special state). In some of those cases, it is an unavoidable result, but often the FSM can be simplified. For this reason, the state machine could be extended with variables (appliance states would still be only on and off and the time would be saved in the variable). Instead of having separate states, the condition of process is tracked via qualitative aspects (states) and quantitative aspects (variables and their values). [7]

Having the ability to define variables and change their values allows usage of additional concepts, namely guards and actions.

1.1.1.5 Guard

In FSMs there is a possibility to define guards for transitions. Guard is a function which takes the state variables, current state and event which initiated the transition and evaluates to either true or false. When the guard evaluates to true, the transition is performed as in regular state machine. But if the guard evaluates to false, the transition is not performed. So with
1.1. State machines

the help of guards, there can be defined multiple transitions for one event on a state. After evaluating the transitions, the ESM can perform only a subset of the transitions or none at all. [8]

![State machine diagram]

Figure 1.6: Event turn_on can be sent to the machine periodically, but it turns the appliance on only between 9am and 5pm

1.1.1.6 Action

Actions are side effects that can be performed by the state machine (in the appliance example it could be: tell the hardware to turn on LED indicator). ESMs can handle three kinds of actions: transition actions, state entry actions and state exit actions. Transition actions are performed on successful state transitions. Entry and exit actions are performed when entering and exiting specific state. [6]

![Action diagram]

Figure 1.7: Actions to make sound when being turned on, turn on light when appliance is on

1.1.1.7 Hierarchical state machine

One last useful expansion of the ESM (for this thesis) is an ability to nest states. State machines containing nested states are called Hierarchical (finite) state machines (HSMs). In HSMs there can be defined a set of substates for each state. The parent of the substates (or superstate) then behaves similarly as an instance of a state machine itself. Therefore, it has defined initial substate and can have final substates. Then when handling events (and when
1. State-of-the-art

the machine is in a substate) the state machine first tries to handle the event in the context of the substate. If the substate is not able to handle the event, the machine tries to handle it one level higher in the superstate and so on. This basically means that when the state machine is in a substate it is also automatically in a superstate. The nesting can be applied indefinitely. It should be clear that this is a powerful abstraction, which helps with building more complex state machines as it allows substate to focus only on part of the process. Additionally, this means that for some cases the nested states (if they are generic enough) can be reused for multiple state machines. [9]

1.2 Alternative approaches

One of the possible alternatives to state machines are Petri nets, which are collections of directed arcs connecting places and transitions [10]. However, their benefit seems to be mainly the ability to model concurrent processes, which in this case is not needed. When searching for other benefits of their use, there wasn’t found any important one for this thesis [11] [12]. The limitations of state machine often mentioned in the comparisons are mostly resolved with the usage of the ESM and the HSM instead of a simple FSM. Even the issue of concurrency (which the FSM does not support) is possible to address with the use of parallel states [13].
This chapter describes the process of gathering the company requirements for the resulting solution. The requirements were used to identify key functionalities which should be provided in the form of use cases. The connection between requirements and use cases is then clearly represented. Finally, there was performed an analysis of the competing applications which meet all or part of the requirements.

2.1 Requirements

As a first step in the analysis, it is important to actually understand what kind of solution will have to be designed. So far there is only a vague idea what will be needed in the finished product. To improve the problem’s understanding there have to be performed two steps:

1. Talk to all the process’s stakeholders and gather their input.
2. Aggregate the collected input into software requirements.

2.1.1 Collecting information

Gathering all the required information for implementing the final solution was in this case a very difficult process. At least from the point of designing a solid replacement of the current processes. The solid replacement should have enough functionality required for fully simulating the processes in their digital form. Additionally, it should be done in a form satisfactory to all the stakeholders and employees. To summarize some of the key issues that made the process difficult:

- no one in the Czech branch of the company is familiar enough with software development,
- there are many stakeholders responsible for different parts of the process,
employees interested in improving the process are not fully authorized to make changes to it,

• no one knew the full specification of how the process should behave,

• the actual specification of the process differed from the way it was carried out in reality.

2.1.1.1 Road to full specification

The following section will be written from the author’s point of view as it is easier to describe the process that way. The text tries to describe chronologically how was the full specification collected and all the different pitfalls encountered before it was finalized. This description hopefully helps with picturing the different kinds of problems that were encountered.

I was approached by a senior mechanical design engineer. He asked me if I could help him with a problem. Some of his responsibilities were designing prototypes of mechanical parts. When he completes design for a part, he needs it in a physical form as soon as possible. To achieve that, he followed these steps:

1. Add a row to an Excel table with a drawing of the part and additional information describing it.

2. Wait until someone contacts him that the part has arrived.

These seem as simple steps, but that was the most critical issue with the system. The missing steps in-between weren’t tracked. What actually happened was this (issues he mentioned are in the parentheses):

1. Add a row to an Excel table with a drawing of the part and additional information describing it.

2. An employee in purchasing department responsible for prototypes has to check the table every day (sometimes they forget to check it for multiple days and have to be reminded).

3. The purchaser then has to decide the process of manufacturing the part. The part sometimes has to go through multiple steps completed by multiple manufacturers (sometimes the purchaser isn’t able to correctly define the steps in a way the designer would be satisfied with, but he doesn’t find out about it until the part arrives).

4. For each of those steps:

   • The purchaser has to create a separate order for the selected manufacturer (this could take longer time that it was supposed to do).

   • The manufacturer fulfils the order and sends the part to the company (the item could get lost at this step).
2.1. Requirements

- The part has to be handled by the receipt of goods and be received in the company’s ERP system (the part could get lost between other parts in storage).
- The part is then inspected by a responsible inspector (it may be faulty) and then handed back to the receipt of goods (again it can get lost there).
- Then if there are further manufacturing steps, this segment repeats.

5. If all the steps are completed, somebody has to notify the part’s designer (usually it was the purchaser, but sometimes nobody did).

As can be seen, almost all the steps in there could cause critical delay, and they did frequently. Parts that had to be delivered in two weeks took months to complete, sometimes they were lost completely and when manufactured twice, the first part was found. The main issue was that there was no information about the process status. So no one could track it. In reality the tracking was done partially by frequent calls between all the responsible employees or by email, but that didn’t amount to much as the issues persisted. This whole process wasn’t clear to me at the start and I understood it only later after multiple meetings with all the departments.

The engineer suggested that I could create some kind of simple application that could track the progress. I saw the potential in the digital solution and agreed. What followed were rounds and rounds of meetings with different departments and even the company’s top management as the problem was much more complex than I had thought at first and because of an unintended scope creep.

I took part in a series of meetings where I understood the whole process described above. I also identified further issues with the process as it was clear in the meetings that there was a conflict between the mechanical design and the purchasing departments about who should be responsible for what steps. Another issue was a disagreement about how long it should take for the part to complete all the steps, basically how strict the deadlines should be. During that period I created a simple prototype of the application, so they could have a clear picture of what I was capable of creating.

Then we realized that if we won’t include the other departments responsible for their steps in the process, we cannot achieve a full process traceability. Without them, nobody else could indicate that they currently have the parts. They weren’t interested in a collaboration at first as they had to perform the same steps in the ERP system which would be duplicated in the new system. Regretfully, this couldn’t be solved by accessing the data in the ERP system. It is managed by the parent company, and they wouldn’t allow access to the data in it, and so we couldn’t use it. In the end the other departments complied with our request and joined the further specification meetings. That sadly complicated the discussion even more. There were now included mul-
multiple department heads and there was no one with a clear authority over all of them. In the end, the consensus was achieved, probably because of my time estimations of what is possible in a reasonable amount of time. On some things, consensus was never achieved, and they were left unspecified (some deadlines for example). Those I implemented how I saw was reasonable, based on the arguments provided.

With the prototype process specified, I was approached by the head of the purchasing department. He was satisfied with how the prototype process was improving and asked me if I could make something similar, but for a different type of item. Apparently there was a system for requesting item orders in the company ERP system, but it was available only for items that were required for the manufacturing. Other items (for example office supplies, computers, tools, etc.) had to be requested outside it, as they weren’t supported. This process was similar to the one with the prototype items. An employee had to fill out a form in an Excel sheet and then send it via email to the purchasing department, this time a supervisor also had to approve it beforehand (via email). So there was an extra approval step, but on the other hand, there was no need for an inspection from the dedicated department, the requestor himself could check the item. I also found a potential in solving this as there were similar problems as with the previous process, and it could be solved very similarly. Only this time it was a process that hundreds of employees were involved in instead of tens, so the benefits in saved time were even greater.

I agreed to include the support for multiple item types as the head of purchasing suggested. I didn’t see much extra effort needed in it and the benefits seemed to outweigh the cost in time and complexity of the development. There was also an indication that the benefit of being able to handle more item types could prove useful in the future. Unfortunately, I was again unpleasantly surprised later on that the process for requests was even more complex than that for prototypes. Even the head of purchasing didn’t expect the bigger scope. Because of the email/Excel process being very lax, it resulted in employees interacting with it in unpredictable ways (some had Excel tables for compiling their requests sent in another Excel tables, etc.). After talking about this issue with the selected employees, it turned out that most of the reasons for their different approach to the process were caused by the lack of the request tracking or absence of a central place for the current request list. Therefore, their issues could be solved automatically in the application, if their simple requests were kept in mind.

The bigger issue we encountered with the head of purchasing was the step of approval. We found that there were explicit steps in the company guidelines based on the price of items requested. There could be a mandatory approval required by the controlling department or by the site manager. Those steps were sometimes ignored, because the process lacked intuitiveness in its current form and also because of the lack of knowledge about the process. We had to agree to include the required approval steps in the application. Later
2.1. Requirements

we showed the prototype solution with the feature of multi-step approval to the management. They were satisfied and decided that the use of the completed application will be mandatory for all employees and there will be no more email requests (in the Czech branch).

It was useful to have the full backing of the top management now, but there was another issue that I didn’t anticipate. There are actually multiple different purchasing departments in the company, each with its own head. Now they were required to use my application and of course, they had their own process slightly adjusted. These were thankfully issues that had to be resolved on the management level and I had to only add few more features (the ability to decide the purchasing department when creating a request and features tied to it).

At this point I had more or less the full specification, but still existed an issue with the lack of data. All the item types had some fields that required specific information filled in, and employees often didn’t know exactly what to fill in. In the Excel version of the process, there were separate sheets with all the possible values, but that data had to be kept manually updated, and also it turned out that often it was not complete. I knew that all the required data was in the database of the ERP system so now with the management support I asked for database views, so I could include at least this data in my solution. This process was a lot of back and forth of emails between me and the parent company, but in the end I received most of the requested views.

Then the COVID-19 pandemic happened and there was a sudden need of home office requests. Until then, the company supported home office only in special cases and an employee had to sign a paper form to be allowed work from a home office. With the need to eliminate as many personal contacts as possible, the management asked me if I could somehow add it to the application (same with requests for holidays). With the specification of the application and the previous processes, it allowed me to confirm that the system can manage multiple different item types and I agreed. The specification of home office and holiday processes was straightforward as there was a detailed guideline describing how it is supposed to work. There was only need of a specialized page in the application, where the managers can see the absence status of the employees in their department.

2.1.2 Requirement specification

From all the information specified in the previous section, there were identified requirements that are needed to correctly design a solution which will leave all the stakeholders satisfied. This step is crucial and will help with identifying and describing all key features required to completely digitize the process.

The requirements will be separated into three groups. Functional requirements that dictate the functionality tied to the correct implementation of the company processes. Non-functional requirements which describe the pa-
rameters the system has to fulfil. Finally, user requirements which will be de-
defined, so the user experience is as good as possible when simulating the original
process steps and taking full advantage of the digital version.

2.1.2.1 Functional requirements
Requirements specifying all the main features and functions that the system
should support. They describe specific functionality important for the core of
the system. Without them, the system cannot be complete.

1. Creation and editing of detailed information of different item types.
2. Support multiple item types with different processes.
3. Allow specifying complex item processes, supporting:
   a) branching based on conditions,
   b) nested sub-processes,
   c) multi-level approval steps,
   d) actions triggered while traversing the process.
4. Overview of items which will help when navigating to actions required
   by the employees as quickly as possible.
5. Overview of user information (absence, calendar, permissions, etc.).
6. Overview of absence data for manager’s subordinates or employees in
   the same department.
7. Setting permissions for users which will enable functionalities.
8. Allow users to enable seeing or performing actions on their items to
   other users (for items they have the required permissions themselves).
9. Export of relevant data into files.
10. Running automatic scheduled jobs (breached deadline notifications, ...).
11. Ability to send email notifications.

2.1.2.2 Non-functional requirements
Requirements specifying the technical side of the system. They do not consider
system functionality, but rather an ability of the system to provide the required
functionality. That means: in which environment the system should be able
to function, what it should communicate with or requirements of the system’s
performance.

1. Web interface supporting the browser Google Chrome.
2. Linking data with the ERP database.
3. Optimized data requests (no long loading times in the application).

4. Optimized user interface (handling thousands of items with no input lag).

5. Architecture supporting easy item process refinement (process can be easily changed).

2.1.2.3 User experience requirements

Requirements targeting the user experience of the system. They cover mostly the application side in which the users interact with the system. The application should have specific goals it is supposed to target to provide an easy interaction with it.

1. Item creation process should have simple and clear structure:
   a) it should consist of multiple steps if the item requires lots of information before submitting,
   b) it should be possible to fill out the required fields without looking into the application manual,
   c) it should provide extra information about some of the more abstract fields (to help with choosing the correct option),
   d) if it has multiple steps, there should be a summary step to help with checking the filled out data before submitting,
   e) when there are errors, it should be clear, what fields to fix,
   f) upon submitting the item, it should be clear, what step the user should follow.

2. User should be able to tell easily what actions are required from him at given time.

3. User should be able to navigate to items as easily as possible.

4. Item information should be reasonably structured so the more complex item types are not flooded with a lot of uncategorized data.

5. User should be able to configure default values for fields which they would otherwise fill out with the same data for every item.
2. Analysis

2.2 Use cases

Use cases are derived from the requirements defined by the company as specified in the section above. They describe all core functionality in detail and specify actors involved in processes. The use cases are split into three groups:

- Request item specific functionality:
  - UC-I-01 — Create item
  - UC-I-02 — Edit item
  - UC-I-03 — Initiate event on item
  - UC-I-04 — View items the user is responsible for
  - UC-I-05 — View all items the user has permission for
  - UC-I-06 — View changes to items
  - UC-I-07 — Find specific item
  - UC-I-08 — View item detail

- User-specific functionality
  - UC-U-01 — Login
  - UC-U-02 — Set user permission
  - UC-U-03 — View user detail
  - UC-U-04 — View department calendar
  - UC-U-05 — User substitutes
    - UC-U-05-1 — View substituting user’s items
    - UC-U-05-2 — Manage substituting user’s items
    - UC-U-05-3 — Manage substitutes

- Miscellaneous functionality
  - UC-M-01 — Generate data export
  - UC-M-02 — Send automatic notifications and reports

Actors

Actors provide a way to differentiate between different types of users or external systems. Then there can be defined use cases, which apply only to some actors or which behave differently based on the specific actor.

Employee Every user logged into the system.

Employee’s supervisor Direct supervisor in company structure.

Requestor Employee who requested item (applies only on the specific item).

Purchaser Employee responsible for ordering items.
2.2. Use cases

**Lead purchaser** Purchaser managing purchase items (of specific type).

**Business unit manager** Employee managing BU in company structure.

**Site manager** Employee managing company site.

**Member of controlling department**

**Member of receipt of goods department**

**Member of inspection department**

**Member of methods department**

**Administrator** Employee with all permissions.

**System** Server running automatic jobs.

![Figure 2.1: Actors](image)

### 2.2.1 UC-I-01 — Create item

Allows creating item of selected item type. This functionality should be as accessible as possible as this is a core feature. Furthermore, this process should be as easy as possible. That means providing: autocomplete for input with common options, ability to save state of partially filled item data and a possibility to create item with values copied from a previously created item.

**Actors** Employee
2. Analysis

Pre-conditions
- Application provides an option to create an item of all item types.
- Application provides an option to copy the item.
- Application provides an option to save progress of a partially filled form.
- *Only for Prototype scenario* — Employee has a permission to create prototype item.

Post-conditions
- Application notifies users responsible for the next step in the process of the item creation.
- Application provides a link to the created item.

Main scenario
1. User chooses an option *Create item* and clicks on the desired item type.
2. Application redirects the user to the item creation page.
3. User fills out the required and other desired parameters and submits the form.
4. Application validates the input values. If there are any errors, it shows them to the user. Otherwise, it submits the data to the server.
5. Server creates the item, initiates its process and performs process side effects if there should be any performed.
6. Server notifies key users of the created item (for example lead purchasers for purchase item types).
7. Server sends a response to the application.
8. If the server returns some kind of error, the application shows it to the user. If the server returns a confirmation, the application navigates to the newly created item’s page.

Alternate scenario — drafts

Pre-conditions
- User saved a partially filled form when creating an item and then exited the creation page.

Scenario
1. User chooses an option *Create item* and clicks on the desired item type.
2.2. Use cases

2. Application shows list of partially created items (drafts) next to the item form.

3. User chooses the option to continue item creation with the previously saved values they have previously filled in.

4. Application loads the values into the form.

5. User then continues the Main scenario from the step 3.

Alternate scenario — copy

Pre-conditions
- User previously created an item.

Scenario
1. User chooses an option to copy the item they previously requested.

2. Application opens the creation form with the input fields filled with preloaded values corresponding to the copied item’s parameters.

3. User then continues the Main scenario from the step 3.

2.2.2 UC-I-02 — Edit item

Allows editing all items after they have been created. This action should be allowed only if the item’s process is not finished (it is not in the final state). This action shouldn’t behave as a simple edit of the item information, but it should also consider impact the changes will have on the item’s process and perform required actions based on the item’s state (for example editing item’s price after it is approved should trigger a new approval). Also, the changes made to the item should be saved, so there is a clear trace of the item’s changed values.

Actors Requestor, Lead purchaser, Employee’s supervisor

Pre-conditions
- Application shows an edit option in the item detail page.

- Application shows the edit option to the user who has required permission.

Post-conditions
- Application should allow navigating to the detail page of each item.
2. Analysis

Main scenario

1. User clicks on the edit option.

2. Application redirects the user to the edit page. This page is similar to the create page, but the input fields have preloaded values corresponding to the item’s parameters.

3. User edits desired parameters and submits the changes.

4. Application validates the input values. If there are any errors, it shows them to the user. Otherwise, the application submits the changes to the server.

5. Server applies the changes to the data and saves the item with the new values as well as the changes.

6. Server also indicates an edit action in the item’s process and performs side effects if there should be any performed (this is based on the item’s current state).

7. Server sends response to the application.

8. If the server returns some kind of error, the application shows it to the user. If the server returns a confirmation, the application navigates back to the item detail page.

9. Application shows the performed changes on the item detail page as well as log of the changed values.

2.2.3 UC-I-03 — Initiate event on item (push item to the next status)

Allows pushing an item to the next status in the item type’s process (each item has current status and can be possibly pushed to its next status). This is triggered by a user event. The user event for the UC-I-03 is generic, but there is a possibility of extra constraints or requirements to exist for the special event. The specific constraints and requirements will be defined in the item processes.

**Actors** Employee

**Pre-conditions**

- Application shows interface for triggering a user event based on the current status, item data and user permissions.
2.2. Use cases

Post-conditions

- Application shows the updated item status and changes generated by the event’s side effects.

Main scenario

1. User initiates the event within the application interface.
2. Application sends the event data to the server and the state machine defined for the item type.
3. The state machine checks the event guards, if they do not pass, the application shows an error and stops the event (with no changes made).
4. The state machine transitions to the next status and performs all side effects. If there was any error at some point, the application shows the error and stops the event (with no changes made).
5. Server saves the next status of the item and the side effects.
6. Application shows a confirmation of the completed event transition and shows updated item data.

Alternate scenario — additional data  Allows triggering the user event and passing extra information to the state machine. The extra information is later used in the transition.

Pre-conditions

- Application shows extended interface for entering additional event information (select menu, form, etc.).

Scenario

1. Application shows extended interface after the step 1 in the Main scenario.
2. User fills out the required additional data.
3. Scenario continues at the step 2 of the Main scenario.

2.2.4 UC-I-04 — View items the user is responsible for

Allows viewing items that the user is responsible for. This is one of the most important views in the application. It should be at the most accessible place in the application. On top of that, the application should be clear about the state the items are in and why the user is responsible for the items.

Actors  Employee
2. Analysis

Pre-conditions
- Application shows items on the home page of the application.
- Application can identify what items the user is responsible for.

Post-conditions
- Application should allow navigating to the detail page of each item.

Main scenario
1. Employee opens the application.
2. Application finds the items the user is responsible for.
3. Application shows the items in a clear structure with enough information about the item status.

2.2.5 UC-I-05 — View all items the user has permission for

Allows viewing all items in some data dense format suitable for search and item progress overview. It should be possible to view all the items that the current user has permission to view.

There are three distinct types of the items:
1. items requested by the user,
2. items in the middle of the process,
3. items with finished process.

What is important item information differs for these types of items. The requested items (1) should show important item’s parameters. The active items (2) should show information that identifies the item’s current status in its process. The finished items (3) should show information that helps with quick searching for items, but also that is displayed in a dense enough format, so there could be shown as many items at once as possible.

Actors
Employee

Pre-conditions
- Application shows links to the pages showing all the item types defined above.
- User has a permission to view the items to be shown.

Post-conditions
- Application should allow navigating to the detail page of each item.
Main scenario

1. User clicks on one of the links directing to the items.
2. Application loads the items based on the selected type and user permissions.
3. Application shows the items.

2.2.6 UC-I-06 — View changes to items

Allows viewing the summary of changes made to items the user has permission to view. For each change there should be shown information about: the person who made the change, time of the change and description of the change or comment describing it.

There should be two kinds of view for the changes: changes made to all the items the user has permission to view and changes to one specific item the user has displayed.

Actors Employee

Pre-conditions

- User has permission to view the items or item.
- Application shows all item changes at the home page of the application.

Main scenario — all items changes

1. User opens the home page.
2. Application shows the changes to the items. The changes of one item should be grouped together for easier readability. The changes of one item done by one user should be grouped together. The changes should be ordered by date, so the latest change is placed first.

Main scenario — single item changes

1. User opens the item detail page.
2. Application shows the changes to the item alongside the other elements of the page. The changes to the item done by one user should be grouped together. The changes should be ordered by date, so the latest change is placed first.

2.2.7 UC-I-07 — Find specific item

Allows searching for items when the user knows some of the item’s parameters.

By the term extended search interface used below is meant:
2. **Analysis**

1. interface allowing text search by item parameters,
2. interface allowing sorting the items by parameters,
3. interface allowing filtering the items by selecting common values of the item parameters (for example status name, requesting employee, etc.)

**Actors** Employee

**Pre-conditions**
- Application provides a search interface in the navigation.
- Application provides a search interface in the item lists.

**Post-conditions**
- Search results link to the item detail page.

**Main scenario — navigation search**
1. User opens the search interface.
2. Application shows the input used for entering the search string (it should be able to search at least by the item’s identifier, name and description).
3. User inputs the searched term.
4. Application filters the items the user has permission to view by the supplied parameter and displays the resulting list.

**Main scenario — item list search**
1. User opens a page with the item list.
2. Application shows *extended search interface* alongside the item list.
3. User uses the search interface for supplying the search parameters.
4. Application filters the items in the item list by the supplied parameters and shows only the ones corresponding to the parameters.

**2.2.8 UC-I-08 — View item detail**

Allows viewing detailed information regarding the specific item for employees which have sufficient permissions. The application should provide the information about: the current status and other parameters tied to the item’s process (for example a currently responsible user, available actions), item parameters supplied at the item’s creation time or during the process and then the history data of changes the item went through.
2.2. Use cases

Actors  Employee

Pre-conditions

- User has the permission to view the item.
- Application shows all the specified information.

Main scenario

1. User opens the item’s detail page via the application or external link.
2. Application loads the item’s data.
3. If the user has the permission to view the item, the application shows the data. Otherwise, the application shows a permission error.

2.2.9 UC-U-01 — Login

Allows authentication users accessing the application. By authenticating them it is possible to set the correct permission for each user. Users should be able to authenticate via multiple external methods: site’s Active Directory [14], parent company’s Active Directory or site’s attendance system. By using external methods it is possible to authenticate users without the need to force them to remember additional credentials. It reduces data requirements of managing the user credentials directly. The decision of using three different methods is caused by the incompleteness of the user database in each of them. Not one of the methods has user accounts for all the users which will be accessing the application.

Actors  Employee

Pre-conditions

- Application provides the ability to authenticate the users.
- Application is able to communicate with the authentication services of all the external authentication methods.

Main scenario

1. User opens the application.
2. Application redirects them to the authentication page.
3. User provides his credentials for a chosen authentication method.
4. Application sends the credentials to the method’s provider. If the provided data are correct, it authenticates the user. Otherwise, it displays an error message.
5. Application creates an authentication token for the user and saves it.

6. Application redirects the authenticated user to the page the user has initially opened.

Alternate scenario — verification Allows authenticating the user without the need of logging in. This applies only to the users, who were authenticated in the last 7 days and have their user tokens saved in the application.

1. User opens the application.

2. Application looks for a saved authentication token. If it doesn’t find it, it behaves as in the main scenario from the step 2.

3. Application checks the expiry value of the token, if the token is older than 7 days, it behaves as in the main scenario from the step 2.

4. Application authenticates the user and renews the expiry value of the token.

2.2.10 UC-U-02 — Set user permission

Allows viewing and settings user permissions. There is also a possibility to define a special group of permissions which act as user groups for receiving automatic scheduled reports. The available permissions are:

- administrator,
- member of controlling department,
- member of inspection,
- member of methods,
- member of receipt of goods,
- head of purchasing for prototype items,
- head of purchasing for request items,
- head of purchasing for service items,
- report — receipt items,
- report — active prototype items,
- site manager.

Actors Administrator

Pre-conditions

- Application provides a link to the settings page in the navigation.
2.2. Use cases

Post-conditions

- Application updates the user permission on his next page load.

Main scenario

1. Administrator opens the settings page.
2. Application shows a list of all the application users, it also provides a way to search between the users.
3. Administrator selects a desired user.
4. Application shows the selected user’s permission and options to add non-active permissions and remove active permissions.
5. Administrator adds or removes the desired permissions.
6. Application shows the updated permissions.

2.2.11 UC-U-03 — View user detail

Allows viewing important information regarding a selected user. The information consists of:

- employee’s calendar,
- employee’s absence data for the month,
- employee’s substitution settings,
- employee’s system parameters.

Actors Employee, Employee’s supervisor

Pre-conditions

- Application provides a link to the user’s own detail page in the navigation.
- Application uses links to the user detail page in as many places where the user is mentioned as possible.
- Application shows the user detail page to its user.
- Application shows the user detail page to the user’s supervisor.

Main scenario

1. User opens the user detail page.
2. If the user has the permission to view the page, the application shows the user’s information. Otherwise, the application shows a permission error.

2.2.12 UC-U-04 — View department calendar

Allows employees to view calendar which shows theirs, their department colleagues’ and their subordinates’ absence. The calendar also allows navigating to a selected user’s page or a selected absence item’s page.

**Actors** Employee, Employee’s supervisor

**Pre-conditions**
- Application shows the option to view the department calendar.

**Main scenario**
1. User opens the department calendar.
2. Application shows the calendar for the current month with the options to select different months.
3. Application shows the user’s absence and absence of the user’s colleagues.
4. If the user is a supervisor, the application shows also the absence of all the user’s subordinates. The subordinates are grouped by their department.
5. User can click on a user in the calendar and go to their page, or they can click on specific absence in the calendar and go to its page.

2.2.13 UC-U-05 — User substitutes

Allows viewing and managing items as well as receiving notifications about the items of other users. Users going on a holiday will be able to set themselves a substitute user who will be able to manage their items. Users in the same department can set their colleagues as substitutes and share their workload.

2.2.14 UC-U-05-1 — View substituting user’s items

Allows managing which items will the application show to the user’s substitutes. The substitutes will be able to choose to view their items, items of employee they are substituting or viewing all items at the same time.

**Actors** Employee
2.2. Use cases

Pre-conditions
- User is a substitute of someone.
- Application provides an interface for setting the user to act as a substitute for.

Post-conditions
- Every item list in the application will show all items based on the permissions of the activated substitution settings.
- Item detail permissions will act with accordance to the activated substitution settings.

Main scenario
1. User opens the substitution menu.
2. Application will show a menu with options to turn on substituting for each available user, an option to toggle showing user’s own items and an option to show all previously mentioned items.
3. User will be able to select multiple of these options at once.
4. Application will reload its state based on the selected options.

2.2.15 UC-U-05-2 — Manage substituting user’s items
Allows managing items which only the substituting user has permission to manage. Managing items means for example: pushing the item to the next state, editing the item, etc.

Actors  Employee

Pre-conditions
- User is a substitute of someone.
- Substituted user has turned on the permission, to manage the items on his behalf, to their substitute.

Post-conditions
- Application will allow managing the items based on the permissions of the substituted user.
Main scenario
1. User activates substitution settings as described in UC-U-05-1.
2. Application will calculate permissions according to the substitution settings.

2.2.16 UC-U-05-3 — Manage substitutes

Allows setting users which can act as user’s substitutes. On top of that provides a toggle for an extra permission for each substitute: allowing item management and receiving notifications.

Actors  Employee

Pre-conditions
- Application provides interface to set user’s substitutes.

Post-conditions
- Application will show the option to act as a substitute to substitutes.

Main scenario
1. User opens interface to set their substitutes.
2. Application will show a list of user’s substitutes which were previously set and option to add a new substitute.
3. Then the process follows with one of these options: add a substitute, remove a substitute, set substitute’s permissions.

Main scenario — add a substitute
1. User selects an option to add a substitute.
2. Application shows a list of all users.
3. User selects the desired user from the list.
4. Application registers the selected user as the user’s new substitute and shows them in the list of substitutes.

Main scenario — remove a substitute
1. User finds an unwanted user in his substitute list and clicks on the option to remove him.
2. Application removes the user from the substitutes.
2.2. Use cases

Main scenario — set substitute’s permissions

1. User finds the desired user in his substitute list.
2. Application shows toggle for permissions: allow item management and receive notifications.
3. User clicks on one of the options and toggles it on or off.
4. Application updates the substitution settings.

2.2.17 UC-M-01 — Generate data export

Allows downloading specific data as a CSV file. Available data exports:

- Items of a specific type, which are not finished (not in a final state of their process).
- All items of a specific type.
- Summary of time spent on prototype items for a set period.
- Absence data of subordinates.
- Absence data of employee.

Actors Employee

Pre-conditions

- Application shows the option to export data to the user.
- Application shows the option only to the user with correct permissions.

Post-conditions

- Browser downloads the provided file.

Main scenario

1. Employee chooses an option to export data in the application.
   a) If it is possible to choose between multiple types of the same export, the application will provide options to select them.
   b) If the export requires some parameters, the application will show an interface to supply the information.
2. Application sends the request for the report to the server.
3. Server loads the required data, compiles it and creates a CSV file from the data.
4. Server sends the file back to the application.
2. Analysis

2.2.18 UC-M-02 — Send automatic notifications and reports

Allows automatic sending of various information to the employees on scheduled time periods.

**Actors**  System, Employee

**Pre-conditions**
- Time period is specified (e.g. 03:00 every day, ...)

**Post-conditions**
- Mailer service transforms the information into emails and sends them out to the users.

**Main scenario**
1. System loads the required data (items that have broken deadlines, etc.).
2. System identifies the responsible users or users selected to receive the reports.
3. System compiles the required information for the notification or report type.
4. System sends the information to the mailer service.
2.3 Requirement coverage

In the following table there is shown a visualization of which use case covers which requirement (crossed out cells). This visualization confirms that each requirement is covered.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>F01</th>
<th>F02</th>
<th>F03</th>
<th>F04</th>
<th>F05</th>
<th>F06</th>
<th>F07</th>
<th>F08</th>
<th>F09</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC-I-01</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-I-02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-I-03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-I-04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>UC-I-05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>UC-I-06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-I-07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-I-08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-U-01</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-U-02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-U-03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-U-04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-U-05</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-M-01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UC-M-02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Requirement coverage

2.4 Competition analysis

To further improve the implementation of the application, competition analysis should be performed. For the problems needed to be resolved in this thesis, solutions of the competition should be properly analysed. If appropriate, the solutions should be used as an inspiration. The two main topics of the analysis will be: 

Features For the feature testing, there were extracted the most important features from the requirements. They are the features allowing custom processes for items with custom definitions. Specifically the features are:

- custom workflows,
- possible approval steps,
2. Analysis

- custom forms (for custom item definition),
- user permission definition,
- integration with Active Directory [4],
- email notifications.

UI/UX The key parts of the [UI] and more importantly the [UX] were selected, so the application is usable for the majority of the users, which perform the actions tied to the core functionality. The parts are:

- item creation interface, where it is possible to fill all the required data,
- simple item list,
- simple item detail view,
- permission management,
- item type and item status specific views (so there is possibility to show different UI elements).

2.4.1 Competing solutions

It should be said that for most Workflow Management Systems (WMSs), their complexity rapidly rises with their customizability, as was proved during the competition testing. Therefore, the focus will be targeted primarily on the features mentioned above and the other features will be ignored, so there can be established a common metric. Solutions fulfilling less than half of the requirements will be ignored.

From all the available solutions found, none (with 1 exception) fulfilled all the requirements. There were chosen 3 groups of solutions that share similar features or an architecture:

- JIRA — it is an exception as it is the only solution that could fulfil all the requirements.
- Project management applications (GoodDay) — focus on task management, they support only simple processes and item types, but provide features for the task management.
- Workflow management applications (Quixy) — their focus is more on the processes of items and also providing customizable forms, but they lack more complex features on those items other than the item create, edit and delete.
2.4. Competition analysis

2.4.1.1 JIRA (Service desk)

Jira \cite{15} is one of the most popular tools for project and ticket management \cite{16}. It provides many features outside the scope needed for this thesis, and it can be also used as a Workflow Management System (WMS) \cite{17}.

With some complex customization it would be possible to fulfill most of the requirements. With some more customization it might be even possible to integrate it with the other systems (AD authentication or email notifications for example). However, the high feature availability comes with its price. Jira is too expensive for larger teams \cite{18} (based on the company standards). For the company purposes, with a team size in the hundreds of employees, the price is too high. That is not counting the price for all the customizations.

**Feature analysis**  Jira provides the ability to define custom processes and forms. Within the processes, it allows placing conditions on transitions, and it also allows complex steps like approvals. Complex permission management with custom user groups is also possible, and it is possible to connect to an AD for user synchronization. Jira can also be connected to a SMTP server and send emails.

**Interesting features**

- Advanced filtering of items.
- Custom item list views (based on state, deadline, etc.).
- Custom projects — separation of issue types and processes.
- Custom forms — easy to design forms to get all the required information from users.
- Complex processes — transition guards, actions on transitions, approval steps.

**UI/UX**  One word might describe Jira’s UI — bloated. It has too many features. There are buttons everywhere. To be fair, considering the average user, Jira could make sense after some initial training. The basic components and calls for action are highlighted and placed on meaningful spots. But there is a high number of secondary actions placed through the application. It has its structure and meaning, but most of the time it wouldn’t be possible to guess what to do. Not mentioning the settings pages, navigating through them without manual would be impossible.

**Pros**

- Home page allows navigating to most parts of the application.
2. Analysis

- Issues page (C.1):
  - Advanced search and filters are very powerful.
  - Ability to create custom lists based on filters.
  - Ability to edit the issue right on the spot (without the need to use forms).
  - Switch between the issue view and detail view (C.2).

- Create page (C.3):
  - Non-trivial inputs (file, autocomplete, etc.) are easy to use.
  - Form layout can be customized.

Cons

- Home page (C.4):
  - Too many ways to view the same thing.
  - It shows too little information about items.

- Project page:
  - Summary view is confusing (C.5).
  - Board view is very basic (again shows little information).

- Issues page (C.6):
  - Issue list is too small, cannot show too much information.
  - It is very hard to find the specific item.
  - There are buttons everywhere seemingly without any structure.
  - Some advanced functionality is not very discoverable (buttons used as links).

- Without the manual, it is not possible to navigate the settings page.

- Application isn’t responsive (it often takes more than 1 second between clicking on a link and showing the next screen).

2.4.1.2 GoodDay

GoodDay is a platform that connects the project management with execution and collaboration. GoodDay includes powerful yet user-friendly tools for project planning, task management, resource and portfolio management, time and budget tracking, progress tracking and analytics, as well as customizable workflows, templates, integrations, and more. 19

Feature analysis  GoodDay’s main focus is on the project management and agile development. It allows the definition of custom tasks with custom fields.
2.4. Competition analysis

On the other hand, it doesn’t support complex workflows as it only allows
the definition of states. It doesn’t provide configuration of transitions between
them. It also supports limited integration with other systems, regretfully none
of them can be used for the requirements of this thesis.

Interesting features

- Simple yet powerful UI.
- Multiple issue views — list, board, calendar, etc.
- Time tracking (time spent on tasks).

UI/UX  What is GoodDay lacking in system features it replaces with UI
features. It features the best UI out of all the applications tested while doing
the competition research.

Pros

- Calendar view on the home page (C.7).
- My time — issue time tracking (C.8).
- Issue list (C.9):
  - Well structured issue list.
  - Powerful filters and sorting, but still easy to understand.
  - Multiple list views (assigned to me, action required, all involved in,
    etc.).
  - Ability to perform action (change status, move deadline, etc.) right
    in the list.
  - Can create and edit the item right in the list.
  - Full-screen list view.
- Table view — good for managing numerous items at once.
- Board view — good for moving the items between the statuses.
- Workload view — managing the items over larger time periods within
  team.
- Project info view — overview and tracking of active issues.
- Well-structured issue view (C.10).

Cons

- Not much information in the issue list on the home page (just name and
  project) (C.11).
2. Analysis

- Custom item fields do not have enough dedicated space for them. It would not be possible to define many of the custom fields, which is required for the purposes of this thesis.

2.4.1.3 Quixy

At its home page Quixy claims to be: “a cloud-based user-friendly business application platform that empowers business users with no coding skills to automate workflows and processes, and build enterprise-grade applications, using simple drag and drop design, ten times faster compared to the traditional approach.” [20]

After few hours of testing, it is clear that even though it allows automating workflows and processes, even complex ones, it is the opposite of user-friendly. Question is, why was this application chosen to represent whole group of applications? It is because there were many other applications with the required features (or at least portion of them), but with a bad user experience. This was proven by the author’s inability to correctly navigate through them even after searching through their documentation (if there was any). Quixy was better than the others in this group in some ways, worse in others, but it provided the most features.

Feature analysis Feature-wise, it provides similar functionality to Jira, except it is mostly less refined. It may differ in some details, like the more advanced form builder or workflow builder. However, as Jira required a manual for its settings, Quixy requires a manual for its everyday use.

Interesting features

- Drag and drop form builder.
- Advanced workflow editor.

UI/UX From a user experience standpoint, Quixy is terrible. It is very hard to navigate the application. Even with the template project it provides. After a few hours of trying to test the application properly, there seemed to be no point in continuing.

Cons

- Home page [C.12]:
  - It is very confusing. There is no indication what pages workspace or app mean and after clicking on the options, there are only tables over the whole screen.
  - From a screenshot it would seem that there are lots of buttons that navigate to other screens that might be more useful. However,
surprisingly most buttons do not do anything, at least they do not indicate that any action has happened.

- Project page (C.13):
  - Again, confusing. There seems to be no option to submit the form.
  - There is no option to see the created items.

### 2.4.2 Summary

To summarize the experience of testing the competition:

- Having many features can be a bad thing from a point of casual user, who doesn’t work with the application every day.

- The pages shouldn’t be overcrowded. Most crucial information should be identifiable right after entering the page.

- Advanced features can improve the user experience, but they shouldn’t be grouped with primary actions.

- If there isn’t a clear outcome what the actions do, the application should show some sort of hint.

- Advanced form of search, sort and filter should be as intuitive as possible.

Tested applications were rated based on their features and UI/UX with 1-5 point scale (1 representing the best rating and 5 representing the worst).

<table>
<thead>
<tr>
<th>Feature</th>
<th>JIRA</th>
<th>GoodDay</th>
<th>Quixy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple workflows</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Advanced workflows</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Custom form fields</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Email notifications</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Project level permissions</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Integration with external systems</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Overall</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2.2: Competition rating
In this chapter the system architecture will be explained. It will be made clear, how the architecture has been designed, what are its key parts and how they communicate with each other. Then there will follow the definition of the data models and components as well as the API endpoints. After that the item processes for each item type will be described. Finally, there will be a description of the process of creating the client application interface design.

3.1 Technologies

Because of the author’s familiarity with the PostgreSQL database engine and TypeScript ecosystem, these technologies were chosen for this thesis.

3.1.1 PostgreSQL

PostgreSQL (or Postgres) is an open-source relational database \[\text{21}\]. It has a big community behind it as it is one of the most used relational databases in the world \[\text{22}\]. One of its advantages is that it provides an easy extendability with third party extensions \[\text{23}\].

3.1.1.1 Foreign Data Wrappers (FDW)

Postgres’s community offers very useful extensions that implement Foreign data wrappers \[\text{24}\]. FDW extensions are used to handle the access to remote objects from an SQL database. For the purpose of this thesis, the foreign object is going to be an SQL database, specifically an Oracle database \[\text{25}\]. Therefore, the extension oracle_fdw \[\text{26}\] is used. This extension has to be installed on the Postgres database server. The installation of the extension is not always needed as it might be already included with the database engine (for example postgres_fdw \[\text{27}\] for Postgres remote access).
3. Design

3.1.1.2 Cron and pg\_cron

Cron \([28]\) is a Linux \([29]\) utility that allows scheduling jobs which have to run at a specific time (for example every 15 minutes or every Sunday at 03:00). With it, it is possible to schedule a job that runs an SQL query (initiated by a Linux command). Postgres expands this possibility with an extension pg\_cron \([30]\). This extension allows scheduling PostgreSQL commands directly from the database.

3.1.2 TypeScript

TypeScript is an open-source language, developed by Microsoft. It is build on top of JavaScript \([31]\). Its main benefit is the ability to define types, which is otherwise not possible with JavaScript. Otherwise, the TypeScript code base behaves similarly to a JavaScript one, and it is possible to choose what TypeScript features will be used. The only drawback of using TypeScript is the need to compile the TypeScript code into a JavaScript code. \([32]\)

The usage of TypeScript will certainly prove beneficial as the code base of the application will be most probably extensive. This is due to the amount of the requirements. With the ability to describe the shape of the data models which will be queried from the database, it will be possible to avoid many type errors. Furthermore, if there have to be decoupled systems implemented, the TypeScript annotations will ease the use of the systems interfaces.

3.1.3 Node

Node.js is a JavaScript runtime, which is used to run application servers (among other things) \([33]\). It provides native interfaces for handling HTTP communication, but they are unnecessarily low-level for the purpose of this application. For that reason a web framework running on top of Node was chosen. Express.js as the most popular Node web framework was the clear choice. It is one of the oldest web frameworks for Node and there exist many libraries which integrate with it \([34]\). Namely, the most important libraries for this application are:

- knex.js \(\text{SQL}\) query builder \([35]\),
- multer multipart FormData \([36]\) handling (used for file upload) \([37]\),
- node-cron automatic job scheduling \([38]\),
- objection.js relational query builder (extending knex functionality) \([39]\),
- passport user authentication \([40]\),
- sheetjs writer of xlsx \([41]\) data format (used for reports) \([42]\).
3.1.3.1 Query builder vs ORM

When mentioning the usage of a query builder, there might arise a question, why not use an ORM. Before comparing the two, it would be appropriate to explain the difference between them.

Query builder could be explained as an abstraction above the native SQL language. Usually it supports the most of the functionality as the SQL, but it also provides input sanitation (which helps with security). Another benefit is that instead of defining the query as a string, the query builder provides an interface for chaining methods on a query object. This approach, with the combination of the higher level language, allows to use some features of the higher language, which makes creating a complex logic easier. One important drawback of the query builder is that it doesn’t support all the functionality the SQL does. To deal with this, knex provides raw methods which can be used to define full SQL queries.

Object–Relational Mappings (ORMs) are higher in the abstraction layers than the query builders. The main idea behind the ORM is that the database tables are mapped to objects. Then instead of using the SQL, object methods are used to read and modify the data. This of course simplifies the process when the data is used in other parts of the system. The main disadvantage of this approach is that not everything can be mapped to an object or an object method, and in some more complex cases it might be impossible to achieve something that could be easily done with the SQL.

For the prototype version of the application, the ORM approach was chosen at first (specifically the library TypeORM). For most of the time the library was usable and the advantages of using objects for the database tables simplified the development significantly as it was easy to connect the data with the business logic. Regretfully later on, the disadvantages of the ORM abstraction were encountered. It wasn’t possible to work easily with a Class Table Inheritance required in the design of multiple item types. On top of that, the library itself was slowly losing the focus of its developer and there were encountered bugs which could not be easily patched.

In previous work of the author, another JavaScript/TypeScript ORM library was chosen — Sequelize.js. In its case, there were encountered different, but fundamentally similar disadvantages. For those reasons, the query builder objection.js was chosen for managing the data. The library objection.js is specifically called a relational query builder on its website. Objection.js uses knex for the communication with the database, but it provides an extra functionality on top of it. It supports mapping relationship tables to objects and specifying relationships between them like the ORM would, but it still exposes the query builder-like interface. Most importantly, it supports falling back to the pure knex queries and even further to the raw knex queries (which basically means raw SQL). In the end this offered advantages of traditional ORM, and it made the development faster. While at the same time, when
there were obstacles encountered, it was possible to resolve them on a lower level.

### 3.1.4 React

At the current time, React is the most used library for building web applications [48]. One of its advantages is its simplicity. The actual size of the library is just over 100 kb [49], and it focuses only on the core functionality used for creating composable web applications. React applications are built using React components, which can be nested in a tree structure. With this, the application can be split into simple components, where each of them can handle its specific part. [50]

The most important libraries extending React for this thesis are:

- **react-async** hooks for data fetching [51],
- **react-hook-form** components and hooks for handling form input [52],
- **react-virtualized** components for efficient rendering of large lists [53],
- **tailwindcss** utility CSS framework [54].

#### 3.1.4.1 State management

To handle the application state there can be chosen multiple approaches: native React Context API [55] or third-party libraries like Redux [56], MobX [57], etc.

In the author’s bachelor thesis, there were explained reasons for choosing Redux rather than the native Context API. The main reasons could be listed as:

1. state data stored only in one global state,
2. complex state data structure and operations on it,
3. frequent state changes (tens of actions per second).

When encountering those cases, the third-party solutions can offer much better performance or easier state manipulation. [58]

Usually, rather than using some form of state management, it is better to keep the data inside the React component or abstract it with the help of hooks. That way, only the components directly manipulating or using the data will have to keep track of it. The more components manipulating the state data, the more cases of unforeseen side effects appear and with them, the complexity of the application. Only when the data is truly needed by more components, there should be used some form of state management. The native Context API provides enough tools for that purpose. [59]

This theory was tested in the prototype version of the application, where global Redux state was used. The reasoning behind that was that keeping
all the application state together enables the reuse of the state data, mainly the fetched data. That would result in decreased load times. Regretfully, as the code base kept growing, this approach was found lacking. For the reasons mentioned above, the application state was rewritten and the state is managed locally on the component level, in some parts on a view component level (shared state for an application route) and minimally on a global level (authentication data, application theme, etc.).

3.1.4.2 Styling solution

For the graphical side of the application Material-UI [60] was initially chosen. There seemed to be clear advantages for its use: it is one of the most popular component libraries for React [61], the documentation is well maintained, and it provides the most used components used in web applications. But there were encountered issues while developing the application which stemmed from the way the library was opinionated about styling its components. This was worsened by the fact that the newest version of the library has chosen to drop the support of CSS-in-JS [62], which was used for this application, in favour of the emotion library [63].

Therefore, there were reasons to test other solutions before migration to the new styling solution. Tailwindcss [54] was one of the tested alternatives and in the end it was chosen for the final implementation. It provided clear benefits as it offers much more low-level approach to the styling, which results in more control over the design. For simple components like buttons and input fields, its benefits do not seem clear, but when there is a need for more complex components, the more granular control improves the development. Compared to pure CSS, it supports consistency with its predefined values (for example colours or margin values) and at the same time, it keeps the bundle size of the application low. [64]

3.2 Architecture

The chosen architecture for the system is a classic client-server model [65], with a database used for the data storage. This approach enables to clearly separate the system’s responsibilities. The server is in charge of querying the data from the database and performing most of the business logic with the addition of running scheduled jobs. The server provides REST API [66], which the client uses for fetching and manipulating the data. Client is responsible for aggregating the data into appropriate format for the user. Aside that, there are two separate services: one used for a user authentication and the other one used for sending emails to the employees.
3. Design

3.2.1 Database

For the purpose of the data storage three different databases are used, two of those are external databases managed by the IT department:

1. primary database,
2. external ERP database,
3. external employee attendance database.

The primary PostgreSQL database stores all the application data. Extending that, to fulfil the company requirements, there is a need to use the already
existing data from the other databases. One of the external databases is used by an attendance system. It stores data describing the company department structure and the employee’s supervisors. The external ERP database stores various data collections describing specific parts manufactured in the company, suppliers registered in the company, etc. What complicates the situation is that the ERP database is physically in a different country and also in a different network (the networks are connected via VPN). Both of the external databases are Oracle databases and their data is used by caching it in the primary database as will be explained in the implementation.

There are multiple ways to obtain the company’s data and at the same time not have to access three databases at the same time. The data could be automatically and regularly duplicated, or someone can be put in charge to update the data when there are important changes to it. But to achieve that the data will be always up to date, the best solution is to use the actual data from those databases. This comes with its pitfalls, the attendance database is notoriously unreliable. What is even worse is that the ERP database is often unreachable due to the connectivity issues. Meaning, that the application could be made unusable by the fault of those issues if there would be no precautions made. Thankfully, there is a better solution to using three separated databases at once, and that is the PostgreSQL’s foreign data wrappers. They would enable connecting the primary database to the other databases, cache the data and supply it to the application as if it were stored there. How is that set up and configured will be explained in the next chapter.

### 3.2.2 Server

The Node server handles the business logic of the application. For that, it uses the data stored in the database using the previously mentioned query builder (combination of knex and objection), which also manages the database connections.

It is possible to use the query builder as a simple interface for interacting with the database data, and making it tightly coupled with the business logic. There could be made a reasonable argument against this approach as the tight coupling makes the possible later changes in the data layer much more time-consuming. On the other hand, designing loosely coupled data layer would mean that there has to be time spent on the interfaces needed for the separation of those two layers. With the author’s time requirements made by the company, it was decided to make this compromise to favour the immediate time saved against the time that might or might not be required, if it would be needed to modify the query builder. 67

For a view layer, which is used to access the data from the business logic, the server provides REST API 29. Using the express web server, there are specified API routes, which are handled by controllers 68. The controllers are in charge of aggregating the HTTP request parameters of the API calls,
passing them to the business logic and then formatting the output before sending appropriate HTTP response. The controllers are properly loosely coupled with the business logic as it is important for the business logic to provide stable interface, which can be used by other parts of the system than the controllers.

One such part of the system is a job scheduler, which uses the node-cron library to perform scheduled jobs. The job scheduler can act as a link between multiple processes of the business logic and perform regular changes (aggregating data), or it can use the data from the business logic and output them to the mail service as notifications for the employees.

3.2.3 Client

The client side shouldn’t perform any business logic as it doesn’t have direct access to the data. Its main task is to present the data accessible from the REST API on the server and send back the data generated by user interactions.

The HTTP communication is handled by a separate layer which is used to transform the data from or into a format required by the API. The data is then used directly by the React components in simple cases (showing directly the data from the API). Or it is handled in React hooks, which make the data accessible using the React Context.

3.2.4 Micro-services

From the company requirements, there can be identified two functionalities which resolve similar problems as other company applications already do or encountered and at the same time can be designed with a simple interface as to be easily reusable. The two functionalities are the user authentication and the email sending system. This can be achieved with a usage of the microservice architecture. Both of those services can be developed as simple systems developed for a single purpose, which can provide REST API for communication with them. This approach will result in an overall reduced development time if the services are reused in the other applications.

3.2.4.1 Authentication

The authentication is a common requirement of the applications developed inside the company. Authenticating users enables giving access to only trustworthy users in the company network. At the same time it enables giving permissions to the authenticated employees.

It is important to allow users to log in with credentials from different systems as not all the users have accounts in every system. Usage of the different systems results in complete coverage of all the user accounts, meaning there is no need to implement separate credentials for the sake of this thesis or other
applications in the development. The different mentioned systems used are: attendance database, Active Directories of multiple company sites.

3.2.4.2 Mailer

Sending emails to the users as notifications or data reports is another common requirement by the company for its applications. The functionality of sending the emails from the node web server is not hard to develop as suitable libraries exist for this use case. However, if there is a need for sending formatted HTML emails, it becomes a complicated issue. This is caused by the fact that the email clients use their own rendering engine, which supports only a fraction of the HTML and CSS specification, and sometimes it is not even implemented correctly [70, 71].

For this reason, the usage of one service fully designed to only handle the HTML email formatting seems appropriate. The issue of the formatting can be resolved only once with the usage of email templates. Then, whatever application is in a need to send a table in an email or an email notification, it only has to send a request to the REST API of the mailer service, which it provides and the task of mail formatting and communication with the server will be handled by it.

3.3 Item types

Based on the requirements, there were identified five request item types, which can be split into two groups. First of the groups is a purchase group, consisting of items: prototype, request and service. The second, absence group consists of items: home office and vacation.

The following sections will describe the process, required data and specifics of each item.

3.3.1 Prototype

The prototype item is used to request the purchase of prototype of a mechanical part by its designers. In this process, the most important aspect is delivering the requested part to the designer as quickly as possible, even when the price of the purchase is higher than the average market price (this scenario is addressed retrospectively to avoid future over-priced purchases).

3.3.1.1 Required parameters

When creating the item, the requestor supplies data required by the process and the data describing the item’s manufacturing parameters.
Process data:

- request name,
- request description (optional),
- priority,
- deadline,
- destination (indicates where to drop off the delivered part).

Manufacture data:

- part revision (optional),
- type of the manufacture (can be created on site, purchased from outside, can be refitted, etc.),
- amount,
- charge account (decides which department and which project will be charged for the part),
- project name (from company approved project list),
- project number (from company approved project list),
- comment (optional, for supplying more data about the item),
- drawing file (optional, manufacturing steps will be decided by this file),
- model file (optional, manufacturing steps will be decided by this file),
- specification file (optional, manufacturing steps will be decided by this file),
- additional attachment file (optional).

3.3.1.2 Actors

**Requestor** Employee who requested the item.

**Purchaser** Employee responsible for ordering the item and keeping the status of the item’s whereabouts.

**Head of purchasing** Purchaser responsible for management of this item type (usually supervisor of the purchasing department).

**Member of methods department**

**Member of receipt of goods department**

**Member of inspection department**
3.3.1.3 Process

Figure 3.2: Prototype item, process
States

Important states of the process with a listed person or group responsible for the item in the defined state. The states with auto-transition are used for a more traceable item history.

Created — head of purchasing
The responsible employee has to define item’s operations. For each operation they supply: responsible purchaser and deadline.

Assistance requested — member of methods
The purchaser needs help with the definition of item’s operations.

To order — purchaser
Item is waiting to be ordered by its assigned purchaser.

Ordered — purchaser
Item is ordered, and it will arrive at the specified date.

Dispatched — member of receipt of goods
The arrival time of the item specified in the order was confirmed.

Received — requestor
Item was received at the company. This state exists mostly so the responsibility of keeping track of the item is transferred to the requestor.

Revision — requestor or purchaser
Item parameters are being edited.
For further information refer to section 3.3.2.5.

Actions

update_bu Edits the item’s BU parameter and notifies the appropriate head of purchasing of the updated BU.

set_purchaser Sets the item’s purchaser.

create_order Creates a record of the item’s order with the information: supplier, order number, delivery date and total price.

create_inspection Creates a record of the item’s inspection with the information: inspecting employee, result and optional note.

 Guards

purchaser_assigned Check whether a purchaser was assigned to the item.

price_defined Checks whether the price of all items is supplied.

req_approval Checks whether the item requires approval (based price).
3.3. Item types

3.3.2 Request

The request item is used to request an overhead purchase (expense not directly associated with creating or producing a product or service). This action can be theoretically performed by any employee.

Item delivery speed is not as a crucial factor here as it is with prototypes, but the volume of requests is an order of magnitude larger (confirmed by the purchasing department). Meaning that this item type will be probably the most requested one. On the other hand, there is no need to involve so many departments (specifically the receipt of goods and inspection) as the employee is responsible for picking up the item and checking if it fits his requirements.

As this request can be done by any employee, there is a need for complex approval logic. Company policy dictates that the supervisor of the employee raising the request has to approve it. Further approvals are tied to the price of requested items.

3.3.2.1 Required parameters

When creating the item, the requestor supplies data required by the process and the data describing the item’s parameters.

Process data:

- request name,
- request description (optional),
- priority,
- deadline,
- direct requestor’s supervisor,
- business unit (of purchasing department responsible which will be responsible for the item),
- order method (the process differs for each option),
- supplier.

Item data:

- supplier,
- attachment file,
- comment (optional, for supplying more data about the item),
- items:
  - item number,
3. Design

- name,
- amount and unit,
- price and currency (optional at item creation time),
- project number, account and department (so the correct company account is charged).

3.3.2.2 Actors

**Requestor** Employee who created the item.

**Supervisor** Employee’s direct supervisor.

**Purchaser** Employee responsible for ordering the item and keeping the status of the item’s whereabouts.

**Head of purchasing** Purchaser responsible for management of this item type (usually supervisor of the purchasing department).

**Member of controlling** Employee from the controlling department.

**BU manager** Employee of specified BU of the company.

**Site manager** Employee on top of the approval hierarchy.

3.3.2.3 Process — main

**States** Important states of the process with a listed person or group responsible for the item in the defined state. The states with auto-transition are used for a more traceable item history.

**To assign** — **head of purchasing** Item is waiting to be assigned to a purchaser.

**Approval** Item is waiting to be approved. This state contains nested states described in the section 3.3.2.4.

**To order** — **purchaser** Item is waiting to be ordered by its assigned purchaser. If the item has the order method value *e-shop*, then it requires no further information. In other cases, the purchaser has to supply additional information: order number, delivery date and total price.

**Ordered** — **purchaser** Item is ordered, and it will arrive at the specified date.

**Received** — **requestor** Item was received at the company. This state exists mostly so the responsibility of keeping track of the item is transferred to the requestor.

**Revision** — **requestor or purchaser** Item parameters are being edited. For further information refer to the section 3.3.2.6.
3.3. Item types

Figure 3.3: Request item, main process
3. Design

Actions

update_bu Edits the item’s BU parameter and notifies the appropriate head of purchasing of the updated BU value.

set_purchaser Sets the item’s purchaser.

create_order Creates a record of the item’s order with the information: supplier, order number, delivery date and total price.

create_inspection Creates a record of the item’s inspection with the information: inspecting employee, result and optional note.

Guards

purchaser_assigned Check whether a purchaser was assigned to the item.

price_defined Checks whether the price of all the items is supplied.

req_approval Checks whether the item requires approval (based on price).

3.3.2.4 Process — approval

![Approval Process Diagram]

Figure 3.4: Request item, approval process
3.3. Item types

**States**  For each of the states, the item is waiting for an approval by the actor with the same name as the state.

**Guards**

- **req_controlling** Total price is over 100 000 Czech crowns or the price of one of the items is over 20 000 Czech crowns.

- **req_bu** The BU of the item is the value Traction Power Systems (TPS) and the total price is over 5 000 Czech crowns.

- **req_site** Total price is over 10 000 Czech crowns.

- **sameApprover** The item was approved by this employee already (in this approval chain). Enables approving multiple steps at once.

3.3.2.5 Process — revision

![Revision Diagram](image)

Figure 3.5: Request item, revision process

**States**  Important states of the process with a listed person or group responsible for the item in the defined state. The states with auto-transition are used for a more traceable item history.

**To revise — requestor or purchaser**  The item’s parameters can or must be edited. The responsible employee is decided by the action initiating the revision.

**To confirm — requestor**  Item revision changes require confirmation.

57
3. Design

Guards

req_confirmation Edits the item’s BU parameter and notifies an appropriate head of purchasing of the updated BU.

reapproval Checks whether the item’s total price was changed. Re-approval is needed when the total price surpasses a price boundary defined in the approval guards 3.3.2.4 or if the price was changed by more than 10% of the total or by more than 5 000 Czech crowns.

3.3.3 Service

The service item can be described as a specific type of the request item type. When designing the process of the request item type, one important point came up regarding the time between the item creation and completion. It probably wouldn’t be possible for the item to arrive the day or the day after the item’s creation. However, there exist crucial types of items that could fit those parameters, namely: machine maintenance, machine repair or critical supplies. For that reason there was designed a streamlined version of the request item type, which doesn’t involve the approval, requestor check and complex revision logic. A need for supervision still remains, so nobody buys unneeded or overpriced items. But, instead of a supervisor that approves the item, there are defined multiple supervisor groups (one for each production department). Members of the supervisor groups should receive a notification after the item’s creation, so they can check it and, if needed, cancel it.

3.3.3.1 Required parameters

When creating the item, the requestor supplies the data required by the process and the data describing the item’s parameters.

All parameters, except for one, are the same as for the request item type 3.3.2.1. The different parameter is the supervisor group instead of the supervisor. The supervisor group consists of employees responsible for checking the item, but they do not approve it.

3.3.3.2 Actors

Requestor Employee who created the item.

Member of supervisor group Employee responsible for a specific service type (this usually corresponds to specific department supervisors).

Purchaser Employee responsible for ordering the item and keeping the status of the item’s whereabouts.

Head of purchasing Purchaser responsible for management of this item type (usually supervisor of the purchasing department).
3.3.3.3 Process

States  Important states of the process with a listed person or group responsible for the item in the defined state. The states with auto-transition are used for a more traceable item history.

To assign — head of purchasing  Item is waiting to be assigned to a purchaser.

To order — purchaser  Item is waiting to be ordered by its assigned purchaser.

Ordered — purchaser  Item is ordered, and it will arrive at the specified date.

Received — requestor  Item was received at the company. This state exists mostly so the responsibility of keeping track of the item is transferred to the requestor.

Revision — requestor or purchaser  Item parameters are being edited. For further information refer to the section 3.3.2.5.

Figure 3.6: Service item, process
3. **Design**

**Actions**

*update_bu* Edits the item’s BU parameter and notifies the appropriate head of purchasing of the updated BU.

*create_order* Creates a record of the item’s order with the information: supplier, order number, delivery date and total price.

3.3.4 **Home office**

The home office item is one of the two absence item types. It acts as an employee request to work from home. Such request has to be always approved by the employee’s supervisor.

Normally the attendance of an employee is managed by the company’s attendance system. That system regretfully doesn’t support home office attendance. Because of the sudden increased need for employees to work from a home office (before the COVID-19 pandemic, it was a rare case, which was allowed by a paper form), it was decided to add this item type and import the attendance data to the attendance system.

3.3.4.1 **Required parameters**

When creating the item, a requestor specifies the date range of the home office. After the item’s approval, the employee fills out the time data based on their work from the home office.

**Item data:**

- employee’s supervisor,
- date range of the home office period,
- comment (optional).

**Time data:**

- date (of being at home office),
- time range (start of work, end of work),
- lunch break time,
- comment (optional).

3.3.4.2 **Actors**

**Requestor** Employee who created the item.

**Supervisor** Requestor’s direct supervisor. This information is stored in the attendance system.
3.3.4.3 Process

Figure 3.7: Home office item, process

**States**  Important states of the process with a listed person or group responsible for the item in the defined state. The states with auto-transition are used for a more traceable item history.

**Open — requestor**  Item is created and waiting for an approval request.

**Approval — supervisor**  Waiting to be approved, rejected or returned to a requestor (to change item’s parameters) by their supervisor.
3. Design

**Time allocation — requestor** At this stage, the employee fills out the time data of the days they spend at the home office.

**Time approval — supervisor** Waiting for defined times to be approved, rejected or returned for a revision to the requestor by their supervisor.

Closed, Rejected Can be reopened to edit the item data.

Actions

create_approval Save the approval data: approver, result and note.

Guards

date_changed Checks if the defined date range of the item was changed.

date_approved Checks if the defined date range of the item was approved.

3.3.5 Vacation

The vacation item is one of the two absence item types. It acts as an employee request for a vacation. Such request has to be always approved by the employee’s supervisor.

3.3.5.1 Required parameters

Item data:

- employee’s supervisor,
- date range of the vacation period,
- location (roughly where the employee is staying at during the vacation period),
- comment (optional).

3.3.5.2 Actors

Requestor Employee who created the item.

Supervisor Requestor’s direct supervisor. This information is stored in the attendance system.

3.3.5.3 Process

States Important states of the process with a listed person or group responsible for the item in the defined state. The states with auto-transition are used for a more traceable item history.

Open — requestor Item is created and waiting for approval request.
Approval — supervisor Waiting to be approved, rejected or returned to the requestor (to change item parameters) by their supervisor.

Closed, Rejected Can be reopened to edit the item data.

Actions
create_approval Save the approval data: approver, result, note.

3.4 Database models

The requirements and the item processes were used to design a database model. As there are multiple item types, each with a different process and different requirements for its data, more than 50 different database entities are used. Some of those are used mainly as junction tables in M to N relationships, but the number of the remaining entities is still high, so it is not possible to show the database model in one piece. Instead of that, the model is split into logical chunks: base item entities, entities for each item type and user related entities. Most entities contain timestamp columns, but as it isn’t a crucial information, it will be omitted in the diagrams.
3. Design

3.4.1 Base item entities

When defining the database entities, the item entities should be the first to consider as they will hold the most important data (user requests and progress in the processes). After analysing what data will be required, it is clear that all the item types will have to keep the track of data with a similar nature (who requested the item, changes to the item status, comments and messages, etc.). It is obvious that there should be defined a base item entity, which would hold or reference this common data and the data specific to the different item types should be held in separate entities using Class Table Inheritance [45]. This approach will prove helpful when implementing the functionality for transitioning items between states, as the logic will be tied to the base item entity, and it won’t have to take the item type into consideration.

Entities org__bu and user are defined in the section 3.4.5. A list of all the other entities related to the base item entity, with a brief description:

item_base Base item entity contains information which all the item types have in common: what item type it is, who requested the item, from what BU the item is requested and what was the original item (if this item was copied).

item_type Data describing the item types.

approval Approvals tied to the item, keeps track of: what type of the approval it is (by supervisor, controlling department, etc.), what was the result (approved, rejected, etc.) and what approval sequence it is a part of (for multi-level approvals).

delay Delays tied to the item, to when, by whom and why was the item delayed.

item_watcher Employees who want to be notified of all item activity.

log_entry Contains information about all activity of the item which is later used for item history log in the application.

message Messages left on the item by users.

notification User notifications with information about changes to the item.

revision Revisions the item went through: what data was added, deleted or changed (from, to) in the item’s parameters, why was it changed and by whom.

status_change All statuses the item went through. The latest status is the current item status.

status Status type description.
Figure 3.9: ER diagram, base entities
3. Design

3.4.2 Prototype item entities

Entity **item_base** is defined in the section 3.4.1. Entity **user** is defined in the section 3.4.5. A list of all the other entities related to the prototype item entity, with a brief description:

- **item_prototype**: Data specific to the prototype item type.

![Figure 3.10: ER diagram, Class Table Inheritance of the item types](image1)

![Figure 3.11: ER diagram, prototype item entities](image2)
3.4. Database models

**project_number** Project number from the list of active project numbers used in the company.

**project_name** Project name from the list of active project names used in the company.

**inspection** Data regarding the process of inspection performed by the inspection department or the responsible employee (passed or failed inspection).

**operation** Manufacturing operations the item has to go through with the employees responsible for it.

**order** Data regarding the order made by a purchaser from a chosen supplier.

**file** Entity for storing attached data files.

**supplier** Supplier information from the list of suppliers registered in the company.

### 3.4.3 Request and service item entities

Entity **item_base** is defined in the section 3.4.1. Entities **file**, **project_number**, **inspection**, **order** and **supplier** are defined in the section 3.4.2. Entities **account**, **department** and **user** are defined in the section 3.4.5. A list of all the other entities related to the request and the service item entities,
3. Design

with a brief description:

**item_request** Data specific to the request item type.

**item_request_subitem** Requested items attached to the request.

**item_number** Item number from the list of item numbers used in the company.

**group_service** Groups responsible for the oversight of service items.

**item_service** Data specific to the service item type.

**item_service_subitem** Requested items attached to the request. There is a separate entity for services in case that in the future, the item data will differ from the request type.

### 3.4.4 Home office and vacation item entities

![ER diagram, home office and vacation item entities](image)

**item_base** is defined in the section 3.4.1. Entity **user** is defined in the section 3.4.5. A list of all the other entities related to the home office and the vacation item entities, with a brief description:

**item_home_office** Data specific to the home office item type.

**item_home_office_time** Time data attached to the home office item.

**item_vacation** Data specific to the vacation item type.

### 3.4.5 User related entities

**user_substitute** Who can substitute the user and what permission they have.

**user** Basic user data and references to default values used when creating the items (account, department, etc.).
3.4. Database models

Figure 3.14: ER diagram, user related entities
3. Design

group All user groups (permissions).

user_group What group is the user part of (can be applied only in the scope of some BU).

org_bu Description of a BU and who is its responsible supervisor.

org_site Description of a company site and who is its responsible supervisor.

account Charge account information.

department Charge department information.

cominfo.department Department data from the attendance system.

cominfo.person Employee data from the attendance system.

3.5 User interface

Among the requirements for the application was a group of requirements which put emphasis on the user experience (defined in the section 2.1.2.3). The application is going to be used only occasionally by the majority of the users and those users might not be experienced enough with performing tasks in web applications. The main use cases for such users will mainly be: the item creation, editing, search or management by sending it to its next state. There was put an extra effort into the interface design of those functionalities.

This section covers the scenarios the occasional users might find themselves in, if they wanted to use the application. The scenarios are used to define main tasks of the users and subsequently in the design of the application wireframe.

3.5.1 User scenarios

The user scenarios extend the defined use cases with a background information of the user’s needs in the use case. The information is then used in the definition of the application interface, which will help the user with achieving his goal. The scenarios are grouped as follows:

- navigation panel (3.5.1.1),
- home page (3.5.1.2),
- creation and editing page (3.5.1.3),
- active items page (3.5.1.4),
- archived items page (3.5.1.5),
- item detail page (3.5.1.6).
3.5. User interface

3.5.1.1 Navigation panel

The navigation panel should be used for helping the users find most of the application’s pages. The scenarios concerning it are:

1. global item search,
2. navigation to other pages,
3. view notifications.

3.5.1.1.1 Global item search

**Background**  The user needs to find a specific item. They expect that when they open the application, it will provide an easy way to find the item as quickly as possible.

**Scenario**  The application includes a search button in the navigation. Pressing the button opens a simple text search interface, which is able to search through all the items the user has permissions to view. The search parameters should be at least the item’s identifier, name and description. The search results should navigate to the item detail page.

3.5.1.1.2 Navigation to other pages

**Background**  The user needs to find data listed in the application or perform some action in the application. They expect the application to provide an easy navigation, which helps them identify where they need to navigate.

**Scenario**  The application shows the navigation panel at all times (on all the pages). The navigation panel contains links to all the application pages. The links in the navigation should be logically separated into sections of common concerns. The application should also indicate which page is currently opened.

3.5.1.1.3 View notifications

**Background**  The user needs to quickly preview changes that happened to the items in the application from the last time they used it or if there were changes while they had it opened.

**Scenario**  The application shows a button which can show a list of the latest notifications of the current user. The notifications can navigate to the item the change is related to. The notification disappears after the user clicks on it and there should be an option to hide all notifications.
3. Design

3.5.1.2 Home page
The home page should contain the most important data relevant to the user at the current moment. The scenarios concerning it are:

1. preview the user’s items,
2. search, sort and filter of the user’s items,
3. preview the latest changes (concerning the user’s items),
4. perform action on a chosen item.

3.5.1.2.1 Preview the user’s items

**Background**  *The user needs to see on which items they have to perform some action. They expect that there will be some sort of a list of the items with enough information, which will help them determine what they should do.*

**Scenario**  The application shows the list of all the items the user is responsible for at the moment. The list should be sorted by the date of the item’s latest change. The information of the item in the list should contain: identifier, name, description, current status, responsible user or department, button for previewing the item’s attachment. The list items should navigate to the item’s detail page.

3.5.1.2.2 Search, sort and filter of the user’s items

**Background**  *The user expects that they will be able to search for a specific item in the list of their items. They need to be able to not only search by various variables, but also sort the items or filter them by specific parameters.*

**Scenario**  The application shows an interface above the item list. The interface contains a search input, which can be set to search by multiple item parameters. The interface also contains a button which can change the item list sort order and parameter. Lastly the interface contains a filter button, which can open the filtering menu. The menu contains the list of common item parameters, for each parameter there is a submenu, which contains all the possible values of the items in the list. Selecting the value filters out all the items in the list, which do not have the parameter set to the chosen value. The application enables choosing multiple filtering options at the same time. Overall it should be always clear which filtering options are active at the moment.
3.5. User interface

3.5.1.2.3 Preview the latest changes

**Background** The user expects that when they open the application that it will show them the latest changes on items which they are, at the moment, responsible for or which they requested.

**Scenario** The application shows a list of the latest changes concerning those items. The changes are grouped based on the day and person, who performed the changes.

3.5.1.2.4 Perform action on a chosen item

**Background** The user expects to be able to perform at least some basic actions on the items in the list without a need to navigate to its detail. They also expect to be able to perform the same action on multiple items at once.

**Scenario** The application enables to select items in the list. If the selected items are in the same state and if there is a possibility to perform action in the state (for example approval), the application shows a button, which will trigger the action.

3.5.1.3 Creation and editing page

The creation and editing page should contain a form for filling the item data and other related components. The scenarios concerning it are:

1. create item,
2. copy item or create item from a draft,
3. edit item,
4. autosave.

3.5.1.3.1 Create item

**Background** The user needs to create a new item. They expect the application to provide a form for the data input. They expect the form to be clear about what data has to be supplied and what the form fields represent. If the fields should contain a value from a common value list, they expect the application to help them select the right value. They also expect the application to provide information about the steps following the item creation.
3. Design

Scenario  The application shows the item form of a chosen type after navigating to the create page. If there are too many form fields to fill in, the application splits the form into multiple steps with a summary as a last step. The application will always show the interface for navigating between the steps. If the form fields require values from a list of known values, the list will be preloaded and used for helping the user decide what value to choose. On the last step of the form, the application shows information about the next steps following the item creation.

3.5.1.3.2 Copy item or create item from a draft

Background  The user expects to use previously created items or item drafts for the creation of new items which will share some values. They expect the creation form to be prefilled with the previous data, so they can submit the item with changing only the different values.

Scenario  In the case of copying items, the application loads the creation form with the data from the copied item (the data will be automatically entered into the form fields). In the case of item drafts, the application will enable the user to save the currently filled in form as a draft. The saved drafts are always shown next to the item form, and it is possible to load the data from them or preview the information stored in them. The drafts can be also deleted.

3.5.1.3.3 Autosave

Background  The user expects that the application will automatically save the form data they filled out in case the application is unexpectedly terminated (because of an application error, losing connection, etc.). They also expect that if they accidentally close the item creation page, their filled in data will be saved.

Scenario  The application periodically saves the filled in data. It also saves the data in case the user navigates to another page. When the user returns to the create page, the application loads the saved data and notifies the user about the action.

3.5.1.4 Active items page

The active items page should contain data regarding the currently active items (not in the final state of the process). The scenarios concerning it are:

1. preview all active items of one type,
2. search, sort and filter of the items (same as in the scenario 2 of the home page)

3.5.1.4.1 Preview all active items of one type

**Background** The user expects the application to show active items in an information dense format, so they are able to quickly navigate through the items. They expect that items of one type are grouped together.

**Scenario** The application shows all items of the selected type in a list. The application shows the same data as on the home page. But it also shows some item type specific data. The items in the list also navigate to the item detail page.

3.5.1.5 Archived items page

The archived items page should contain data regarding the currently archived items (in the final state of the process). The scenarios concerning it are:

1. preview all archived items of one type,
2. search, sort and filter of the items (same as in the scenario 2 of the home page)

3.5.1.5.1 Preview all archived items of one type

**Background** The user expects the application to show archived items in an information dense format, so they are able to quickly navigate through the items. They expect that items of one type are grouped together. The user needs the application to show as much of the item data as possible as the number of archived items will grow in time.

**Scenario** The application shows all items of the selected type in a table. The application shows the same data as on the home page. But it also shows some item type specific data. The items in the table also navigate to the item detail page. Compared to the active items page, the data is shown in a table, and it is possible to show more data in the table columns.

3.5.1.6 Item detail page

The item detail page should contain all the item data as well as all actions that can be performed on the item. The scenarios concerning it are:

1. preview of item data,
2. comment on item,
3. Design

3. perform action on item.

3.5.1.6.1 Preview of item data

**Background**  The user expects that the page will show all data about the item. They also expect that the data will be grouped as to show data of similar nature grouped together. They need the application to highlight the most important data, so they do not have to search for it.

**Scenario**  The application shows the item data in multiple panels: item process information panel (current status, responsible employee or department, etc.), item specification data and miscellaneous item data (for example item history). The item specification data can be separated into multiple tabs as to hide less important information and show the most important information on the main tab.

3.5.1.6.2 Comment on item

**Background**  The user needs to be able to send a message concerning the item to all involved employees. They expect to be able to see other user’s messages.

**Scenario**  The application shows a comment panel, which can be used to leave comments on the item. The panel then shows all the item messages.

3.5.1.6.3 Perform action on item

**Background**  The user expects the application to show all the possible actions they can perform on the item. They need the application to be clear about what actions are needed by them. The actions have to be self-explanatory or provide some information about what they are used for.

**Scenario**  The application shows all the actions the user can perform (tied to the user permissions). The application highlights the action triggering the next step in the process. Other most used actions are displayed next to it in a more subtle design. The less used actions are grouped together in the menu. All the non-obvious actions have simple tooltips explaining their use.
3.5. User interface

3.5.2 Task list and task groups

From the defined user scenarios there were identified application interaction tasks. The tasks are used for the design of the application structure. There are two groups of tasks: data display and actions (changes to the application state).

3.5.2.1 Categories

Tasks were grouped into logical categories:

1. home (initial page),
2. active items,
3. archived items,
4. item detail,
5. item creation and editing.

3.5.2.2 Data display

- Item type data
  - Main parameters (relevant in most occasions: identifier, name, status, etc.)
  - Other parameters (relevant only in specific statuses or small group of users)
  - Items
  - History
  - Comments
- Form for item creation and editing
  - Main parameters
  - List of items
  - Summary of filled data
- List of items the user is responsible for
- List of actions concerning the items the user is responsible for
- List of active items
- List of archived items
- Preview item data
3. Design

3.5.2.3 Actions

- Search, sort and filter items. [1, 2, 3]
- View item’s attached file [1, 2, 3]
- Perform action on item [1]
- Toggle between showing item’s history or comments [4]
- Show item type’s process information [4]
- Leave a comment on item [4]
- Edit item data [4]
- Save or delete item draft [5]
- Cancel item creation or editing [5]
- Add or remove item [5]
- Submit filled in item data [5]

3.5.3 Task graph

The tasks defined above and the links between them are visualized in the task graph which can be found in the attachment D.2.

3.5.4 Wireframe

With the user scenarios and tasks defined, design of the application interface was created in a form of a wireframe. In the wireframe, there is designed the structure of the application elements and their hierarchy. The main focus was to transfer the tasks into a form of a multi-page application and design the elements which fulfil all the user needs described in the user scenarios. The main pages of the wireframe are included in the attachment D.2.
This chapter covers the implementation of the state machine, services, server and client. It also describes the application from an UI/UX standpoint and explains the reasoning behind the decisions made when creating it. Lastly, it describes the folder structure of the codebase, the application deployment and the codebase documentation.

4.1 Modules and services

Some requirements of this application were not unique in the scope of all the software projects developed in the company. Examples of the common functionalities on the server side are: classes for REST API implementation or configuration of the web server. Examples on the client side are: low-level React components (buttons, information panels, etc.), React hooks handling the data fetching or local data persistence. Other shared functionalities are utility functions which can be used on both the server and client.

For this reason there were created modules as separate npm packages for the server, client and utilities. There was created one additional module, which is used for managing the state machines. This way the state machine logic is clearly separated from the application logic. The application then defines the state machines in the module’s format and operates on the machines using the module’s functions.

This approach might seem as it just introduces unneeded complexities. However, as of the time writing this thesis, the benefits of separating the application logic into modular parts are already visible. The modules and microservices are already used in four other applications or services. The existence of these modules brought a significant time reduction in the development of those applications.
4. Realization

4.1.1 State machine

The state machine module is the backbone of this application. It is used to model the business processes, and it is crucial for the implementation to be deterministic, efficient and most importantly without bugs.

The module supports all the constructs mentioned in the first chapter \[1.1\] namely:

- states,
- events (regular and auto),
- hierarchical states,
- actions (enter, exit and transition),
- guards.

On top of that there is a need for a few more constructs, arising from the nature of the application. The item processes will not be performed fully at once. Rather, the process will be started, the changes will be saved to the database, and then it will be stopped. Only when the user or the system performs some actions (send events), the process will be loaded again, the event will be handled and then saved. For that functionality, those additional constructs will be helpful, such as the start and end actions, which will run only at the start or end of the process. Other useful functionality is the ability to transition to previous states. With that, instead of the transition targeting specific state of the state machine, the target would be the first previously visited state. This enables, for example, cancelling the last action triggered by an event.

To make use of the state machines (previously discussed as Extended (finite) state machines) for the purpose of modelling the business processes, the state machine will have to be able to manage context data, such as the item information, to be truly useful. So there is a need for the state machine to read and write the database data. For that, two approaches can be used. With the first approach, all possibly required data would be loaded before the handling of the event. Only after the event is processed by the state machine and the data changed, it would be saved back to the database. With the second approach, to the state machine would be provided just the minimal required data used for the database references (database connection, identifiers, etc.). Then the state machine would read or write the database data as needed while processing the event.

The first approach is most suited when the context data has a simple structure and when it is known exactly what data will be needed at the time of the event. Where it falls short is in the opposite case, and mainly when it is not known what data will be needed. In that case, every piece of data, the process could require, has to be loaded and then saved, which could result
in many unneeded database operations.

The second approach is able to resolve this issue by reading and writing the data at the time when it is truly needed. To be able to do this, the event handling has to be able to perform asynchronous database operations, which complicates the implementation slightly. The benefits outweigh this issue as not only will it reduce the complexity caused by preloading all the data, but it also will allow defining actions and guards which are truly independent of each other and partially also of the context (if designed properly, when implementing them). They will have basically no side effects in the scope of the state machine logic.

To achieve truly deterministic behaviour the module is implemented in a way as not to introduce side effects anywhere. It also performs operations on the state machine instances as if they were immutable structures, and it utilizes other principles from functional programming.

There may arise a question if there exists a JavaScript library for handling state machines. In that case, there wouldn’t be a need to create a new solution. Indeed, some exist, for example a library xstate [73]. The issue is that most of the libraries found, which could be used in this application, do not support the asynchronous approach for handling the actions and guards. For this reason, a new solution was created, which is inspired by some data structures and interfaces of the xstate library.

### 4.1.1.1 Data format

To operate with the state machines a suitable representation of it has to exist in the code. There has to be a specification of the state machine data format that can handle all the state machine constructs.

The state machine is defined with a nested state node structure. Every state is an instance of a **base node**. **Base node** can respond to all kinds of actions (entry, exit, start and end) and it can have defined transitions responding to events (even auto transitions). The transitions can have defined actions which will be performed if the transition is performed as well as guards which will allow performing transitions only when the guards are passed. There can be defined multiple transitions for one event. In that case, the transition that is performed is the first one in the definition. If the transitions have guards, the first performed transition is the one which first passes all the guards (starting from the first transition, continuing on to the next one, etc.).

```javascript
{
  // actions performed after entering the-node
  entry: 'some_action', // action as string
  // actions performed after exiting the-node
  exit: ['some_action'], // array of actions
  // actions performed after starting the-machine in this node
  start: ['some_action', 'some_action2'], // multiple actions
  // actions performed after ending the-machine in this node
}`
```
4. Realization

```plaintext
end: ['some_action'],
on: {
  ': 'state0', // auto transition to 'state0'
  next1: 'state1', // transition on event 'next1' to 'state1'
  // transition on event 'next2' to 'state2' (verbose)
  next2: { target: 'state2' },
  // transition on event 'next3' to 'state3', with actions
  next3: { target: 'state3', actions: ['some_action'] },
  // transition on event 'next4' to 'state4'
  // transitions only when guard 'can_transition' is true
  next4: { target: 'state4', cond: 'can_transition' },
  // transition on event 'next5'
  next5: {
    // transitions to 'state5' only if guards
    // 'can1' and 'can2' are true
    target: 'state5', cond: ['can1', 'can2'],
    // otherwise transitions to 'state6' only if
    // guard 'can1' is true
    target: 'state6', cond: ['can1'],
    // otherwise transitions to 'state7'
    target: 'state7',
  },
  // transition on event 'next6' to 'history' state
  // 'history' state is the last previous state
  next6: '$history'
}
```

Listing 4.1: State machine data format: Base node

Extending the base node is a state node, which can have its own nested states (other base nodes). The state node has defined an initial substate, which is entered when the state node is entered. When handling an event, the implementation first looks for a defined transition in the leaf state node. If it finds a transition, it performs that. Otherwise, it looks to its immediate parent state node and searches for the transition there. This repeats to the root state node. If no transition is found, no transition is performed.

```plaintext
{
  initial: 'state1', // initial state of the-node
  // ...actions
  on: {
    cancel: 'state1',
  },
  states: {
    state1: { // ...actions and transitions
    },
    state2: {
      type: 'final' // final state of the-machine
      // ...actions and transitions
    } // ...other states
  }
}
```

Listing 4.2: State machine data format: State node
Extending the state node is a root node. Which acts as the root of the state machine. It has only one more parameter compared to the state node and that is the machine identifier.

```json
{
  id: 'machine-id',
  // ...state node definition
}
```

Listing 4.3: State machine data format: Root node

### 4.1.1.2 Functions

There are exposed 4 main functions used for interaction with the state machines:

- **init** Initialization of the state machine instance from a state machine definition. The instance is initialized in the initial state of the machine or, if provided, a specific state of the machine. The state machine can be initialized with context data and with the records of previously visited states if the state machine was initialized before.

- **sendEvent** Allows sending an event to the initialized state machine. It can perform transitions and actions (based on passed guards). After the event is resolved, it returns an updated state machine instance. The event doesn’t have to be a simple string, it can also contain payload data. The algorithm of the function is listed below (Algorithm 4.1).

- **start** Performs start actions in the current state of the state machine instance (if there are any defined).

- **end** Performs end actions in the current state of the state machine instance (if there are any defined).

### 4.1.1.3 Actions and guards

The action and guard functions are not directly attached to the state machine, only the action or guards function name is specified there. The definition of the functions is tied to the state machine instance. This way the state machine definition can be serialized and saved to the database.

The functions are used by the state machine instance handlers (**init**, **sendEvent**, etc.). The instance handlers take the context data, current state and event as arguments and return the updated context (in case of the actions) or boolean value (in case of the guards).
4. Realization

Algorithm 4.1 State machine: event handling

1: function PossibleTransitions(instance, node, event)
2: transitions ← []
3: current ← node
4: while current ≠ nil do
5: transitions.pushBack(GETTRANSITIONS(instance, event))
6: current ← current.parent
7: return transitions
8: function FirstPassedTransition(transitions, instance, event)
9: for trx in transitions do
10: if PassesGuard(trx, instance, event) then
11: return trx
12: return nil
13: function GetActions(instance, transition, node)
14: actions ← []
15: current ← node
16: while current ≠ nil do
17: actions.pushFront(GETEXITACTIONS(instance, current))
18: current ← current.parent
19: actions.pushBack(GETTRANSITIONACTIONS(transition))
20: current ← node
21: while current ≠ nil do
22: actions.pushBack(GETEXITACTIONS(instance, current))
23: current ← current.parent
24: return actions
25: function SendEvent(instance, event)
26: trxs ← PossibleTransitions(instance, instance.node, event)
27: trx ← FirstPassedTransition(trxs, instance, event)
28: if trx = nil then ▷ return if no possible transitions found
29: return instance
30: if trx.target = 'history' then
31: targetNode ← GETTARGETNODEFROMHISTORY(trx, history)
32: else
33: targetNode ← GETTARGETNODE(trx, instance)
34: actions ← GetActions(instance, trx, targetNode)
35: context ← PERFORMACTIONS(instance, actions, event)
36: instance.context ← context ▷ update instance
37: instance.node ← targetNode
38: instance ← SendEvent(instance, 'auto') ▷ perform auto transition
39: instance.history ← UPDATEHISTORY(targetNode)
40: return instance
4.1. Modules and services

4.1.1.4 TypeScript type definitions

One benefit of designing a fully deterministic state machine implementation is the ability to predict what types of states, events, actions and guards the definition contains. The predetermined types can be then extracted from the state machine definition and used in the functions for specifying allowed values. Furthermore, the predetermined values can be automatically used for code suggestions as seen in the picture.

![Figure 4.1: State machine: code suggestions of possible states](image)

It should be noted that the code suggestions of extracted types are not easy to achieve due to the limitations of Typescript, and it was not fully possible until the version 4.1. This applies mostly to the nested objects of recursive types, which the state machine type is as it supports in a theory infinitely nested states. For this reason there has to be set a recursive boundary for the type extraction. [74, 75]

4.1.2 Server

The server module provides functions for reducing a boilerplate code. In case of the REST API server this includes: classes for querying the database, handling HTTP requests and managing routes.

The basic functionality of the server is providing the access to the database entities. Meaning, for each entity provide CRUD operations. To achieve that, there needs to be an interface retrieving the data from the database, controller handling the HTTP request and response, and a configured API route.

This obviously applies only to the simple cases of what could be asked of the API server, so when designing a generic helper classes, the ability to extend their functionality should be taken into consideration.

For this reason the server module provides three generic classes: QueryService, Controller and Router.

4.1.2.1 QueryService

Class QueryService enables an easy configuration of the CRUD operations on the database entities defined as objection.js models. It is only required to create an instance of the generic class with the database model and database connection as arguments, and the class instance automatically provides all
4. Realization

the CRUD functions for the database model. Obviously, it is easy to extend this class to provide more functionality.

```typescript
export interface DataService<M extends Model> {
    findAll(): Promise<M[]>
    findById(id: Id): Promise<M>
    create(data: DeepPartial<M>): Promise<M>
    patch(data: DeepPartial<M>): Promise<M>
    patchById(id: Id, data: DeepPartial<M>): Promise<M>
    deleteById(id: Id): Promise<number>
}

class QueryService<M extends Model, DB extends Knex> implements DataService<M>
```

Listing 4.4: Server module: QueryService

4.1.2.2 Controller

Class Controller provides express.js request handlers for the CRUD operations. It is instantiated with the database model and a QueryService as arguments. Again, the Controller class can be easily extended.

```typescript
export interface CRUDController<M extends Model, DS extends DataService<M> > {
    getAll(req: Request, res: Response): Promise<Response>
    get(req: Request, res: Response): Promise<Response>
    create(req: Request, res: Response): Promise<Response>
    patch(req: Request, res: Response): Promise<Response>
    delete(req: Request, res: Response): Promise<Response>
}

class Controller<M extends Model, DS extends DataService> implements CRUDController<M, DS>
```

Listing 4.5: Server module: Controller

4.1.2.3 Router

Class Routes configures the express.js routes. It can be instantiated with a path specific for the resource corresponding to the database entity. It can also be configured to automatically handle the Controller instances, specific paths or other routers for nested paths.

```typescript
type HandlerOptions = {
    method: 'GET' | 'POST' | 'PATCH' | 'DELETE',
    handler: ExpressRequestHandler,
    params?: string | string[]
}
```

86
4.1. Modules and services

Listing 4.6: Server module: Router

```
action?: string
actionParams?: string | string[]
}
export interface Router {
  path: string
  handle(options: HandlerOptions | HandlerOptions[]): Router
  handleController<C extends Controller>(controller: C): Router
  handleRouter(router: Router): Router
```

4.1.3 Application

The application module consists of common React components and hooks that can be used in multiple applications. Its target is to provide low-level components (such as buttons, information panels, etc.). With them, it will be easy to create applications which will share the same design language and employees can easily identify that the application was developed by the company.

The other use cases of the module is to reduce duplicate code when solving common problems for all the company web applications. Such problem can be as simple as a React hook for managing a text input state and showing a dialog window after completing some action or more complex such as an authentication and a notification management, fetching data from the server or providing an interface for displaying a large list or tabular data with the ability to search, sort and filter it.

4.1.3.1 Interface components

Common components used thorough the application. They mainly abstract common functionality or simplify the maintenance of the design.

**Button** Stylized button.

**ButtonGroup** Merges multiple buttons into one block.

**Card** Basic panel for displaying information.

**Dialog** Interactive dialog window, which can be opened or closed by another component.

**Form** Stylized form element. Provides also stylized input elements (text, checkbox, select, date, time and file).

**Menu** Interactive menu element, which can be opened or closed by another component.

**Stepper** Shows step menu (used in multi-step forms for example).
4. Realization

Tabs Allows viewing content based on the selected tab.

Table Stylized table (and table elements like row, cell, etc.).

WithValue Table Component for showing key-value styled data.

4.1.3.2 Common functionality

More complex components which usually provide access to its React context data or React hooks for interacting with them.

Authentication Provides components for authentication interaction and authentication context, which can be used to determine what to show to the user based on the login state and the user’s permissions.

Data fetching Hooks used for data fetching from the API server.

Notifications Provides hooks for showing notifications to the user after performing some actions (success after editing an item for example).

Virtualized list and table Components for displaying a list or a table with a large amount of items or rows. It renders only a limited number of the items (rows) based on the scroll state and the others are rendered only when the user scrolls to them. This greatly improves performance and allows showing tens of thousands of items (rows) or possibly even more without the need to use pagination.

Data list and table Complements the functionality of the virtualized components with the ability to search, sort and filter data.

4.1.4 Utilities

The utility module consists of very low-level functions that are commonly used in the developed web applications. It could be compared to a simple version of a library like the lodash. The benefits of using the custom approach is that the types can be greatly improved (the lodash types do not use all the available functionalities of TypeScript) or that the functions do not have to be designed so generic, and instead they can be optimized for the use cases of the company needs.

There are multiple categories of the helper functions:

array Sorting based on keys of the array object elements, grouping by a key of the array object elements.

compare Extraction of different parameters of nested objects.

date Common conversion of date objects to string.

number Rounding numbers, random number generation.
4.1. Modules and services

**object**  Reading, setting and updating values of nested objects, merging nested objects, map and reduce for objects.

**string**  String capitalization, string comparison for search.

These functions prove themselves helpful when handling and transforming nested objects as their type definitions check for the presence of the object node defined by the provided path as can be seen below (and also above with the showcase of the code suggestion 4.1).

```plaintext
1 function sort<T>(arr: T[], path: Path<T>, opts: CompareOptions): T[]
2 function groupBy<T>(
3     arr: T[], path: Path<T>, opts: GroupByOptions
4 ): { [key: string]: T[] }
5 function get<T, P extends Path<T>>(
6     obj: T, path: P
7 ): TypeAt<P>
8 function set<T, P extends Path<T>>(
9     obj: T, path: P, value: TypeAt<P>
10 ): T
11 function update<T, P extends Path<T>>(
12     obj: T, path: P, cb: (val: TypeAt<P>) => TypeAt<P>
13 ): T
```

Listing 4.7: Utility function signatures

### 4.1.5 Mailer

The mailer service runs on its own separate server. It provides a simple [API](#) for sending emails to the company employees. There are two types of emails that it can send: a simple text email and an email containing tabular data. The text email can be used to send simple notifications to the users. Such notifications can have a configurable email subject, an email body header and a text body. On top of that the email can contain a button which links to the provided address.

![Workflow](Image)

*Figure 4.2: Mailer, text email*
An email with the tabular data can also have a custom email subject and on top of that it can display a table (in the HTML format, which most email clients can display without an issue). To display such tables, there has to be provided a header row for the columns and then an undetermined number of table rows. The table can also have a specified link column, which contains links to the application.

![Figure 4.3: Mailer, table email](image)

The API is very simple to use, there has be made a HTTP POST request to the service with the correct data (as specified in the API specification E.1). For this, the service uses a configured connection to the company SMTP server via the nodemailer library.

### 4.1.5.1 Emails with tabular data

Before picking the solution of formatting the tabular data in emails using HTML, there was a period when there were used XLSX files as attachments. The reason for this is that many email clients support only subset of the HTML and CSS specification, or they do not respect it fully, and it is very hard to create well-designed emails. A proof can be the existence of many HTML creators for email clients or paid services dealing with this issue. 

Sending the XLSX files in the attachment seemed like a good solution as it is a native format for tables. In the emails a plain text was displayed: “see attached file for the content”. This approach worked and nobody was complaining, but as was later found out, only a very small portion of the users was actually opening the attachments. The reason being that, because of the various phishing scams, the users were wary of the attached files and didn’t open them. This of course meant that there wasn’t much use to
4.1. Modules and services

the emails in that form and the approach with showing the data directly in the email was chosen.

4.1.6 Authentication

The authentication service also runs on its own separate server. It provides a simple API for the user authentication. Its functionality is inspired by the OAuth 2.0 protocol, but the process of authentication is simplified compared to it. When the client application requires an authentication, it redirects the user to the client of the authentication service. When redirecting, there is stored the URL of the page where the redirect was initiated and also the application name, so it can be shown on the login page. The client consists of a simple form requiring the user’s username, password and then a method of authentication. There are three methods of authentication: the first two are via the company Active Directories and the third is via the company attendance system. The methods are configured on the server with the help of the library passport, the AD uses passport-ldap strategy and the attendance uses custom passport-local strategy.

![Authentication, client](image)

Figure 4.4: Authentication, client

If the user supplies the correct credentials for the method, the service will generate a JSON token for them with basic information about the user account (employee number, full name and username if they have an AD account) and
4. Realization

redirects them back to the page of the application client. The token is then stored in the application’s local storage and used for further user verification. The token is valid for seven days.

Figure 4.5: Sequence diagram: authentication, login

After a successful authentication, the application doesn’t require to use the token to access the application’s resources, it is only needed when the user closes and opens the application again. In that case, the application uses the token and verifies it by using the authentication service. If the token is not expired, the service prolongs its expiration and returns it. The application then doesn’t require the user to login.

Figure 4.6: Sequence diagram: authentication, verify

The authentication uses [HTTP POST requests](https://en.wikipedia.org/wiki/HTTP) for this functionality. The API specification can be found in the attachment [E.2](#).
4.2 Database

This section introduces the database views and functions used by the application. Furthermore, it showcases how is resolved the need of accessing data from the remote databases.

4.2.1 Views

There are used database views for the most common joins used in the application or when there is a need to use complex SQL queries. The views are used instead of fetching related tables on the level of the query builder. With this approach it is possible to reduce the time spent selecting the data from the database, especially when there are required multiple joins. [81]

- **log_entry_full**
  Extending the log_entry with user’s name and item’s uid and name.

- **message_full**
  Extending the message with user’s name and item’s uid and name.

- **operation_latest_order**
  Mapping the latest order (if it exists) to every operation.

- **wi_prototype_curr_operation**
  Mapping the current operation (if it exists) to every prototype item.

- **wi_request_latest_order**
  Mapping the latest order (if it exists) to every request item.

- **wi_service_latest_order**
  Mapping the latest order (if it exists) to every service item.

Alongside the common joins, there are used views for every item type, which finalizes the Class Table Inheritance [45] implementation mentioned in the previous chapter.

- **wi_base**
  Extending the item_base with the data about the item’s current status (which is based on the last status_change related to the item) and requestor name.

- **wi_prototype**
  Extending the item_base with the data from the item_prototype table and including information from: project_name, project_number, wi_prototype_curr_operation, operation_latest_order and user (as assigned purchaser and inspector).

- **wi_request**
  Extending the item_base with the data from the item_request table
4. Realization

and including information from: \texttt{wi}\_\texttt{request}_\texttt{latest}\_\texttt{order}, \texttt{supplier}, \texttt{org}\_\texttt{bu} and \texttt{user} (as assigned purchaser and supervisor).

\texttt{wi}_\texttt{service}

Extending the \texttt{item}\_\texttt{base} with the data from the \texttt{item}\_\texttt{service} table and including information from: \texttt{wi}\_\texttt{service}_\texttt{latest}\_\texttt{order}, \texttt{supplier}, \texttt{org}\_\texttt{bu}, \texttt{group}\_\texttt{service} and \texttt{user} (as assigned purchaser).

\texttt{wi}_\texttt{home}\_\texttt{office}

Extending the \texttt{item}\_\texttt{base} with the data from the \texttt{item}\_\texttt{home}\_\texttt{office} table and including information from: \texttt{user} (as supervisor).

\texttt{wi}_\texttt{vacation}

Extending the \texttt{item}\_\texttt{base} with the data from the \texttt{item}\_\texttt{vacation} table and including information from: \texttt{user} (as supervisor).

4.2.2 Functions

Complementing the item views, there are used functions for selecting the item data as a specific user or users. When the functions are called (with the user identifiers as parameters), the rows from the item views are filtered out and the final returned items are decided by the user’s permissions and the current status of the item (as defined in the previous chapter). The functions are: \texttt{user}\_\texttt{item} (items of all types), \texttt{user}\_\texttt{prototype}, \texttt{user}\_\texttt{request}, \texttt{user}\_\texttt{service}, \texttt{user}\_\texttt{home}\_\texttt{office} and \texttt{user}\_\texttt{vacation}.

Aside those functions, there is used one other function for generating the item’s uid — \texttt{get}\_\texttt{uid}. This function dynamically generates a unique identifier for each item which can be used to quickly recognize the item. The uid consists of the item type’s abbreviated name and the value from the id column (of the item type not the base item). The resulting item uids have names in the following format: \texttt{P1} (prototype item, 1st created), \texttt{R102} (request item, 102nd created), \texttt{HO12} (home office item, 12th created), etc.

4.2.3 Querying data from remote databases

In the previous chapters there was mentioned a need to access the data saved on remote databases, specifically on two Oracle databases. The attendance database is managed by an external company and the [ERP] database is managed by another department. In neither case it is possible to receive full access to the required data in the databases. However, there are provided user accounts for which there are created views with the data required for this application.
4.2. Database

4.2.3.1 The Problem

The local application database runs in the local network. In the local database, there are entities which reference data on the other remote databases (the reference can be for example some simple identifier used as a foreign key). To make use of the references between the data, it has to be possible to call a query on the remote database from the local database.

To complicate the situation, even if the remote database could be queried, it may not be accessible at all times due to connection errors. The errors could be caused by multiple issues: the remote database is offline, there were made some changes to the remote database, network connectivity issues, etc.

On top of that, the remote ERP database is not on the same network as the local database, it is actually in a different country, which slows the response times.

To resolve these issues, there could be used some sort of local cache of the data, but that brings other issues. When to cache the data? Will it be done regularly as a scheduled job? How often to update the cached data? What system will be responsible for running the job on time?

4.2.3.2 Querying the remote database

As mentioned previously, remote databases can be accessed with the help of foreign data wrappers. In this case with the extension oracle_fdw \[26\]. With the extension, it is possible to define a foreign server, which is used for the connection with the remote database \[82\].
4. Realization

Listing 4.8: Remote database, creating a foreign server

```sql
CREATE SERVER remote_db
    FOREIGN DATA WRAPPER oracle_fdw
    OPTIONS (dbserver 'server_url');
```

If the foreign server is defined, a user mapping has to be created, which will enable the selected user to run queries for the foreign server. Usage permission has to be also granted for the foreign server to the selected user. 

Listing 4.9: Remote database, creating a user mapping

```sql
GRANT USAGE ON FOREIGN SERVER remote_db TO 'user';
CREATE USER MAPPING FOR 'user' SERVER remote_db
    OPTIONS(user 'username', password 'password');
```

With the mapping defined, it is possible to query the remote database as the selected local user right in the local database. This obviously applies only to the objects that the mapped user has access to. For the mapping there can be created a foreign table representing the table on the remote database.

Listing 4.10: Remote database: creating a foreign table

```sql
CREATE FOREIGN TABLE EXAMPLE(
    ID INT NOT NULL,
    NAME VARCHAR(255) NOT NULL,
    DESCRIPTION VARCHAR(255)
) SERVER remote_db OPTIONS (table_name 'example');
```

To avoid the requirement of specifying the individual foreign tables or if the table column's data type is not known, there can be instead imported the whole database schema.

### 4.2.3.3 Caching the remote data

As explained above, the foreign tables can be used to access the data on remote databases. But every time the remote database is queried, there is a delay caused by the time the response data travels from the remote database to the local database. Also, once the network connection between the databases fails, the remote data cannot be accessed any more.

To resolve that issue, it is possible to use materialized views. The materialized views are created the same way as regular views, but they have one key difference. When the materialized view is created, it will call the query it represents and saves the result. After that, every time a query is called on the materialized view, the result will be computed from the saved data,
4.2. Database

not the current data (the query of the materialized view will not be called). To refresh the data stored in the materialized view, the view itself has to be refreshed. [86]

With the use of materialized views, there will be no issue of waiting for the data to arrive from the remote database. Furthermore, it can be even used if the connection to the remote database gets interrupted. To demonstrate the advantages of this approach, there was performed a test of selecting all the data in the supplier table of the remote [ERP] database. When selecting from the foreign table, the query execution time was 3 seconds. When selecting from the materialized table (cached data of the foreign table), the query execution time was 0.007 seconds. The difference in execution time is clear.

4.2.3.4 Automatic synchronization

So far there was shown a method of querying the remote database almost as if the data was stored on the local database (same availability and same response times). The last issue is that the data might not be always up to date. Materialized views have to be refreshed to update the data they return. Otherwise, they return the same result as when they were created. The view can be refreshed manually with a command, but that approach cannot be used if the remote data is changing regularly.

To fully make use of the remote data, it has to be properly analysed, in this case mainly from the point of how often it is updated. If the remote data is not updated often or not at all, there might not be a need for an automatic refresh. If the remote data is updated often, and there is no issue with the extra time spent waiting for it, there might be no need for the caching. Lastly, if the remote data often changes, and there is a need for a fast and reliable access to it, there has to be configured the automatic refresh.

There are two main options for the automatic refresh. One option is a trigger that would refresh the view after some action in the database. That is usually done after a write action on the tables that the view uses. But, for the use case of this application, this is not needed. The second option is an automatic refresh of the view via a scheduled job. [87]

Another way to achieve the automatic refresh might be a cron job on the server the database is running on, but Postgres allows for a different approach. There can be set up a cron job right in the database with the help of the pg_cron extension [30]. Advantages that this approach brings are that the scheduled job resides right next to the data which will be refreshed, and also that the cron logs are also saved to the database.

When choosing this option, there has to be considered one more thing, how often does the data have to be refreshed. With a longer period between each refresh, the data could get stale quickly, but also it won’t be using many resources of the remote server. Good practice could be to run the refresh once every night if the query is complex. But, if up to date data is required, and
it is changing quickly, it might have to be refreshed every 5 minutes or even quicker. Once the refresh rate reaches such pace, caution should be applied. If a complex query is running or if the query returns a large amount of data, it could overload the remote database, network or even the local database. There could be even a case where the query takes longer time than is the refresh rate.

Once the extension pg_cron is installed and configured on the Postgres database, it can be used to configure scheduled jobs by running the cron.schedule function. In this case, the scheduled jobs are going to be the refresh queries of the materialized views. The configured scheduled jobs can be found in the table cron.job.

Every time the scheduled job is run, the result and the information about the job execution is saved in the table cron.job_run_details. To remove a scheduled job, function cron.unschedule is used.

```sql
-- Schedule a refresh of the view EXAMPLE every 15 minutes
SELECT cron.schedule(
    'refresh_example',
    '*/15 * * * *',
    'REFRESH MATERIALIZED VIEW EXAMPLE'
);

-- Delete the scheduled job
SELECT cron.unschedule('refresh_example');
```

Listing 4.11: Remote database: scheduled jobs

### 4.2.3.5 Summary

The combination of the foreign data wrappers, materialized views and scheduled jobs is used for access to the data stored on the remote databases. As described above, this approach allows reading the data without having to deal with a network delay, and it is even possible to access it if the remote database is unreachable. The refresh of the data by the scheduled jobs is configured based on the frequency the remote data changes. For the most part, the data is refreshed once every night and in some cases once every 6 hours.

### 4.3 Server

This section documents the general API information of the REST endpoints. Then, it describes in detail the core business processes of the application. After that follows a brief explanation of the scheduled job and report systems.

### 4.3.1 API

The server provides generic REST API (using the server module described in the previous section) for any entity that is interacted with on the client side.
Those entities can have its data read or written using the CRUD operations. Namely, the entities are: account, approval, department, file, item_base, item_prototype, item_request, item_request_subitem, item_service, item_service_subitem, item_home_office, item_vacation, log_entry, message, notification, operation, order, org_bu, project_number, supplier, user, user_group. For every entity there is implemented a CRUD controller and there are exposed following API routes:

/api/entity - GET Return array of all entity rows.
/api/entity/:id - GET Return entity with supplied id.
/api/entity/:id - PATCH Edit entity with supplied id.
/api/entity/:id - DELETE Delete entity with supplied id.

Some of those entity controllers are extended, so they can include the data of their relations if required (for example item’s log entries). Or more parameters can be supplied to the controllers to filter or aggregate the results. Additionally, the operations in the controllers can have additional side effects.

On top of that the server supports REST action resources as defined in the Restful Objects Spec [88]. Meaning the resources can have action endpoints. Good example is the action /api/item/:id/action/send_event, which allows sending state machine events to specific items. This approach (instead of strictly adhering to the REST API specification) was chosen, so each API endpoint route is used specifically for one functionality instead of multiple functionalities differentiated by the request data. Usually the decision to use an action instead of extending the PATCH operation of the entity is chosen, when the action would perform some side effects not directly tied to the regular entity parameter editing.

The extended controllers with a brief specification of the additional functionalities are:

- group
  /api/group/:id/action/add_user Add a user to a group.
  /api/group/:id/action/remove_user Remove a user from a group.

- item_base
  /api/item/:id/action/transition
  Sending state machine events.
  /api/item/:id/action/status_set
  Manually setting the item’s status.
  /api/item/action/suggestion
  Data of all the items the user has a permission to view (used for a global search).
4. Realization

/api/item/action/summary
Data of all the items the user is currently responsible for and log entries tied to them.

/api/item/:id/action/file/:type
Getting the file data of an item’s attachment.

/api/item/action/suggestion
Getting the create and edit form parameters used in autofill.

- item_prototype
/api/item/prototype/:id/action/delay
Delay the item.

- item_request
/api/item/request/:id/action/delay
Delay the item.
/api/item/request/:id/action/reassign
Reassign the item.

- item_service
/api/item/service/:id/action/delay
Delay the item.
/api/item/service/:id/action/reassign
Reassign the item.

- item_home_office
/api/item/home_office/:id/times - POST
Add a time record to the item.
/api/item/home_office/:id/times - PATCH
Edit a time record on the item.
/api/item/home_office/:id/times - DELETE
Delete a time record from the item.

- operation
/api/operation/:id/action/set_deadline
Set operation’s deadline.
/api/operation/:id/action/set_name
Set operation’s name.
/api/operation/:id/action/set_workflow_position
Set operation’s position in sequence.
/api/operation/:id/action/assign
Set operation’s purchaser.

- user
/api/user/:id/action/add_substitute/:substituteId
Add user’s substitute.
/api/user/:id/action/change_substitute/:substituteId
Change user substitute’s permissions.
4.3. Server

/\api/user/:id/action/remove_substitute/:substituteId
  Remove user’s substitute.

/\api/user/:id/action/absence
  User’s absence data.

/\api/user/:id/action/timeline
  User’s calendar data.

There is implemented one additional controller which is responsible for data exports. Its purpose is to gather and aggregate data from the defined reports and then to provide an API access to it. The export endpoints then return the response data in the CSV or XLSX data format. The exposed endpoints of the controller are:

/\api/report/active_items
  Information about all the not completed items.

/\api/report/summary
  Information about the time it takes the items between changing their statuses grouped by the responsible departments and other related data.

/\api/report/absence
  Information about the absence data for a selected time period of all the user’s subordinates.

/\api/report/absence_user
  Information about the absence data for a selected time period and user.

/\api/report/metrics_changes
  Metric data about the time it takes to change an item’s status, grouped by the item type.

/\api/report/metrics_completion
  Metric data about the time it takes to complete an item, grouped by the item type.

4.3.2 Business processes

The most complex processes that are tied to the core system’s functionality are described below. These processes were chosen, because they manage a functionality which most of the users will encounter. Therefore, it is important that this functionality is well implemented and described. The processes are: item creation, editing and item state machine event handling.

Every process uses a database transaction when performing queries. This allows persisting the data to the database only when the full transaction is committed. If the server encounters error at any point in the middle of the process, it can cancel the whole transaction and inform the user about it. This means that there are never committed partial data to the database which
could cause errors.

### 4.3.2.1 Create item

The item creation on the server side starts with receiving the data in the item controller. At that point, data from the HTTP request are extracted. The extracted data are slightly different for each of the item types (this is tied to the relevant additional tables the items are related to). The extracted data are then transformed into the format of the database entities (again, this step is slightly different for each item type). After the transformation, the data is inserted into the database. The insert operation returns the newly created item’s identifier.

With the item created and saved in the database, its process can be started. This is done by initializing the state machine instance of the item type with the context containing the new item’s identifier. The state machine automatically performs defined transitions and actions tied to them if there should be any performed. Additionally, there are initiated the start and end actions of the state machine. After the state machine instance is in the end state, the database transaction is committed and the item’s identifier is returned in the HTTP response.

#### Algorithm 4.2 Item creation

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP request instance (body, params, ...)</td>
<td>Created item’s id</td>
</tr>
</tbody>
</table>

1. \( data \leftarrow \text{GetDataFromRequest}(request) \)
2. \( data \leftarrow \text{TransformData}(data) \)
3. \( trx \leftarrow \text{InitDBTransaction()} \)
4. \( id \leftarrow \text{InsertDataToDB}(data, trx) \)
5. \( smInstance \leftarrow \text{SMInit}(id, smConfig, trx) \)
6. \( smInstance \leftarrow \text{SMStart}(smInstance, trx) \)
7. \( smInstance \leftarrow \text{SMEnd}(smInstance, trx) \)
8. \( \text{CommitDBTransaction}(trx) \)

### 4.3.2.2 Edit item

The item editing starts similarly to the item creation by receiving the data in the item controller, followed by the data extraction from the request data and then the data transformation. Both steps are tied to the item type.

After that begins the database transaction and the current item data is loaded from the database. The following steps are dependent on the item type. If the item type has revision statuses and if the item itself is not already in the revision status, the automatic revision process is initiated. The automatic revision process enables resolving the item’s process changes in a predetermined state. As an example can be chosen item edit, which changes the item
price. Based on the changed price there could be required some changes to the item’s process (which would not be tied to the item’s current status). For example, the item would have to be approved again before proceeding further. The automatic transition to the revision status then allows having the changes triggering the item’s status transition defined in one place instead of defining them on every status, where they could happen. Then, after processing the item’s changed data, the item can either transition to some other status (previously mentioned re-approval) or to its previous state.

Following that, the current item data is compared with the data in the incoming changes. The changes to the item are saved to the revision data, or the revision data is updated if the item is already in the revision status. If the revision was automatically triggered, there is sent an event, which tries to finalize the revision. In the mentioned example it would try to return to the previous state if the item’s price hasn’t changed, or it would trigger re-approval if it did.

Finally, the item edit is logged and committed.

Algorithm 4.3 Item editing

Input HTTP request instance (body, params, ...)

1: data ← GetDataFromRequest(request)
2: data ← TransformData(data)
3: trx ← InitDBTransaction()
4: item ← GetItemById(data.id, trx)
5: if HasRevisions(data.type) and NotInRevision(data.status) then
6:     autoRevision ← true
7:     SendEvent(revisionRequest, data, trx)
8: changes ← CompareData(data, item)
9: rev ← GetLatestRevision(data.ud, trx)
10: if rev ≠ nil then
11:     CreateRevision(data.id, changes, trx)
12: else
13:     UpdateRevision(data.id, changes, trx)
14: if autoRevision = true then
15:     SendEvent(revisionDone, data, trx)
16: AddLogEntry(data, changes, trx)
17: CommitDBTransaction(trx)
4. Realization

4.3.2.3 Event handling

When the item events are handled, the process also starts with extracting and transforming the data from the HTTP request. However, this time only the minimal item data is extracted (item’s identifier supplied in the request parameters) alongside the event data (event type including possible payload). For the item data all the item’s previous states are also loaded which can be used for transitioning to the item’s history.

After that, the database transaction is initiated. Every further interaction with the database is performed in the scope of this transaction (even the state machine actions and guards). The state machine instance is initiated with the context data and the event data is sent to it. After the event is handled, the end actions are initiated and the transaction is committed.

**Algorithm 4.4** Item state machine event handling

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>function <strong>SendEvent</strong>(event, data, trx)</td>
</tr>
<tr>
<td>2.</td>
<td><code>smConfig.history ← data.prevStates</code></td>
</tr>
<tr>
<td>3.</td>
<td><code>smInstance ← SMInit(data.id, smConfig, trx)</code></td>
</tr>
<tr>
<td>4.</td>
<td><code>smInstance ← SMSendEvent(smInstance, event, trx)</code></td>
</tr>
<tr>
<td>5.</td>
<td><code>smInstance ← SMEnd(smInstance, trx)</code></td>
</tr>
<tr>
<td>6.</td>
<td><code>item ← GetItemDataFromRequest(request)</code></td>
</tr>
<tr>
<td>7.</td>
<td><code>event ← GetEventDataFromRequest(request)</code></td>
</tr>
<tr>
<td>8.</td>
<td><code>trx ← InitDBTransaction()</code></td>
</tr>
<tr>
<td>9.</td>
<td><code>SendEvent(event, item, trx)</code></td>
</tr>
<tr>
<td>10.</td>
<td><code>CommitDBTransaction(trx)</code></td>
</tr>
</tbody>
</table>

4.3.3 Scheduled jobs

The scheduled jobs are handled by the node-cron library. The scheduled jobs are used for aggregating data from the database and then sending it to the users as emails via the mailer service. There are specified multiple job groups based on the period of the jobs:

- Every work day at 6am and 12pm:
  - for the employees of the receipt of goods department, prototype items which they are currently responsible for and prototype items which should arrive in the next 2 days,
  - for the employees of the inspection department, prototype items which they are currently responsible for,
4.4 Client

- any employee, items which should have been pushed to the next state by the employee in the deadline specified by the company guidelines.

- Every work day at 6am:
  - members of the active items report group, all active prototype items.

- Every day, week, month, quarter and year at 2am:
  - members of the summary report group, summary of the prototype items created in this period (the summary is specified in the API endpoints section).

4.3.4 Reports

The application’s data reports are generated on the server side and can be accessed using the API endpoints. For the reports there is used the XLSX format as it is the most common table format used by the employees. The format is proprietary, so there is not an easy way to generate the files manually and instead there is used the sheetjs library. For the defined API endpoints the server directly sends the XLSX files as a response, so they can be easily downloaded.

4.4 Client

This section describes the parts of the web client interface which the users interact the most with. Following that, there is explained the method of fetching the data from the server, how the user permissions and their influence on the displayed content are handled and finally there is described the functionality of user substitutes.

4.4.1 Interface

In the previous chapters, there was described the design of the user interface which aims to be as accessible as possible. The implementation of that design is described below.

4.4.1.1 Navigation

The application was designed with an easy navigation between its pages in mind. Easy navigation in practice means that the number of the steps the user has to perform to get to their desired page is as few as possible. The other implication of the easy navigation is that it is clear that the user has the option navigating to other pages, and it is understandable from the interface what the page will contain.
4. Realization

The application provides the navigation via the navigation panel. The navigation panel contains links to all the pages the application contains (navigation can be seen in the attached screenshots F.1). For a better structure the navigation panel is split into two levels. The first level contains mostly application wide settings and elements influencing the application state:

- user login state (with logout option and link to the user’s detail),
- notification menu,
- substitution settings,
- global item search,
- link to the settings page (for admins),
- light/dark application theme toggle (showcase of the dark theme is in the attachment F.20),
- links to the help and the changelog pages.

The second level of the navigation panel contains links to the application pages:

- home page,
- create item page,
- user’s requested items page,
- user’s department timeline,
- app statistics page,
- item type (project) specific pages.

The item type specific pages include links to active and archived items of the corresponding item type as well as links used for downloading the item type specific reports.

4.4.1.2 Item creation forms

The forms for filling out the item information are a crucial part of the main application functionality. The reason being that most of the employees will encounter the application, when they need to submit (or edit) an item request. The application has to provide a clear and easy to use interface for these tasks, but there are few issues that complicate achieving that goal:

- some item types require a large amount of information to be filled by the employees right at the beginning,
- the process of the form filling can be interrupted (due to the length of the time required to fill out the information),
• employees do not have to know what the required fields represent,
• employees do not have to know what to fill out in the fields even if they know what they mean,
• employees do not have to know what will follow the submitting of the form data.

For each of those issues there was implemented a solution (as shown in F.5):
• the item form is split into multiple steps if the form requires a large amount of information,
• there is a robust draft and save system, which allows employees to save their progress alongside automatic saves done by the application,
• for fields which might not have a clear name, there is supplied a hint information,
• for fields which can have various or complicated values selected (multiple similar values, thousands of possible values), there is provided a hint text for each of those values and extended interface helping with the value selection,
• at the end of the creation form there is a summary of the process which will proceed after creating the item.

4.4.1.3 Item overview

The main purpose of the application is to manage items of multiple item types with various business processes. For that reason, the users will usually have to work with multiple items at the same time, and it is important that the application will display the relevant items in the format most suitable for the user.

As designed, there have to be at least 4 types of items lists:
1. items the user is currently responsible for,
2. items the user requested,
3. items that are currently active and the user has a permission to view,
4. items that are in their final state and the user has a permission to view.

For each of those item lists there is a separate page in the application. There could have been used one list component for each list type, but designing a specific components based on the list requirements leads to better user experience. In the end, three separate components were created:
1. grouped list (for the item list 1 F.1),
2. dense list (for the item lists 2 F.2 and 3 F.3),
3. dense table (for the item lists 4.4).

The separate components share two features. The first feature is the ability to show all items at once without the need to use pagination (loading and showing only some amount of the items at once split between multiple pages). The pagination can be used to reduce the load time of the resources when displaying large lists or tables. Besides saving the resources on loading items, there is no need to use it as the user has no benefit seeing just part of the data. It actually even harms the user experience as often the user cannot see all the information they need at once. If it was possible to load all the items without harming the speed or without requiring too many hardware resources, it would benefit the user experience. There can be multiple levels of the pagination (corresponding to the levels of the system the data flows through before they reach the user): at the level of the database, the server or the application. For each of those levels the performance has to be optimized:

1. On the database levels there were created views (mentioned in the server section 4.2.1), this improves speed as the database engine is best suited for data operations (for data already in the database).

2. On the server level there is queried only the information that will be used in the item lists (no extra data is loaded) and no complex operations are made on the data, this improves the amount of the server resources (memory, process time) needed.

3. On the application level there is used the virtualization of the DOM elements (practically there are shown only list items currently which can be seen in the window and the others are hidden, drastically improving render times).

When testing the optimizations for the list of all the archived items of the request item type (thousands of items), the response time (level 1 and 2) for querying the data was reduced significantly (from roughly 7s to 300-500ms) and the render time (level 3) was reduced even more (from more than 30s to hundreds of ms). These results are impressive, and this approach can be probably used until the item count reaches approximately 10,000 items. After that, further optimizations will be probably needed, but that shouldn’t be an issue as there wasn’t much effort used reaching the current results.

The second shared feature is the search, sort and filtering functionality (later referred to only as filtering). The previous feature (loading all the items) allows using the item data right from the items to be displayed (contrary to having to query the data from the server for showing all the possible item attributes with the pagination approach). The number of items the user can interact with at one time can be high (hundreds or even thousands of items with the current data), so the filtering can be applied on as many attributes as possible to help the user with the item count reduction when searching. The filtering can also be applied on multiple attributes at once. And finally,
the application remembers the state in which the filtering was configured when redirecting to another page and then returning to the list page, or when reloading the application. All of this allows the user to apply complex filtering logic which can stay saved. At the same time when the users return to the application after some time, they could be confused that they do not see all the items they should see, which is caused by the saved filtering which they do not notice. For that reason the filtering clearly indicates its state as shown on the active items page F.3.

4.4.1.4 Item detail

The item detail page, similarly to the item creation page, has to be able to display all the item information in a clear and structured format. The difference between them is that there cannot be used the steps for separating the information as that would require the users to perform multiple actions to be able to see the information they need. For that reason, a different approach is used for displaying the item information. The information is separated into categories:

Process information Item’s identifier and name, current status, responsible employee, requestor and item type process data. As this information is important for every step of the process, it is displayed as the main element. There are also included the actions the user can perform on the item: send process event, edit item, copy item, cancel item download item data, set item status, etc.

Item’s attributes This section contains most of the item’s attributes, where there are prioritized the attributes the users require the most regarding the item process: order number, supplier, price on the purchase items and calendar data on the absence items. The less prioritized item data can be displayed in different tabs in this section (revision data, detailed approval data, etc.).

Item’s history Describes the status changes which happened on the item and other changes made to the item. There is also a possibility to leave a comment on the item in a separate tab.

Item type specific data For some item types it was beneficial for the data structure to display a part of the data in a separate section (operations for the prototype type, purchase items for the request and service types). Displaying the data in a separate section which was designed specifically for the data allowed to show it in a more distinct format.

Examples of the item detail page can be found in the attached screenshots F.10.
4. Realization

4.4.1.5 Department timeline

One of the important requirements of the absence item types was the overview of the employee absence data. This is especially important for supervisors who have many subordinates. The overview of the absence data which can be displayed on the department timeline page is able to display information about:

1. all the subordinates grouped by their department,
2. all requested absence item types of the subordinates,
3. the absence item type’s status,
4. time spent on home office,
5. information if the subordinate was physically present at the company.

At the same time there should be displayed information about as many days as possible, so the user can see as much information at one time as possible.

For this reason the page displays a monthly overview of the data listed above. The information is shown in a compact way to show all the month’s days and also as many user rows on one screen. At the same time it shows the data in a structure, so it is clear what the data means. One example of that would be that the difference between a home office and vacation is not only shown by a colour, but also by using a different shape of the item element. Furthermore, all the items and the users shown on the timeline contain links which navigate to their respective pages. Examples of the timeline page can be found in the attached screenshots F.17.

4.4.1.6 User detail

The last important page of the application is the user detail page. It is used to display data about the user, namely: the user’s calendar (information about the absence item data), the absence report for the month (with the ability to download it), substitution settings and then the general user data with an option to set default values for some item creation form fields. Examples of the user detail page can be found in the attached screenshots F.13.

4.4.2 Data fetching

The fetching of the data from the server is handled with the help of React hooks. There are used the utility hooks from the client module mentioned in the previous section. The hooks are supplied with the API endpoints, and they return the data context (the data itself, function to reload the data, loading status, etc.).
4.4. Client

The client pages are implemented as to always fetch only the currently required data. The data required by the other pages is not fetched until the user navigates to them. At the same time the data required by some hidden components (for example the global search) is also fetched only at the time the user opens or interacts with the component. This results in the client not making any unnecessary data fetching and putting unneeded load on the server or spending time on handling the unneeded data which could prolong the render time.

For the data revalidation the refresh period parameter is used, which can be supplied to the data fetching hooks. When active, the revalidation is periodically triggered and the data are fetched in the configured interval. The interval can be set differently for each use case, and it depends on the requirement of having the most recent data. For example when the item detail page is opened, the data is revalidated every 5 minutes, but the department timeline is revalidated only every hour.

For specific fetched data (an item detail, item lists) the client saves the result in the client’s memory. The saved data is used when the user leaves the page and then returns to it again for the brief period it takes the data to fetch from the server. This way the client can render the information even when the data is not yet fetched and when it is finally fetched, the data is only updated with the most recent values. This of course could result in displaying data in an out of date state, but in most cases there are not that many changes to the fetched data for it to be a problem and even if the newer data are different, the saved data are updated with the most recent version in a short amount of time (the data fetching usually lasts less than a second).

4.4.3 Permission handling

The application has to offer different functionalities to the users based on their permissions. The client receives the user’s permissions after the user is authenticated. Then, it has to allow the user to perform the operations specified by their permissions. One way of implementing that would be to use specialized pages which could be accessed only to the users with the appropriate permissions. However, this would require to define each of those pages separately and could result in dramatical increase in the page count. This increase would then cause more difficult maintenance of the client pages in the long term, and also it would unnecessarily interrupt the flow of the user’s interaction with the application.

Better approach which the client uses is using a dynamic content. Each element or functionality that requires a user permission is encapsulated in a component subscribed to the user permission data. The component can have specified the user permission group, current item’s process status (if the component is defined in the context of some item) or a custom condition.
4. Realization

This component data is then evaluated and the element or functionality is hidden if the specified parameters aren’t the required ones.

```typescript
// Apply only when the-condition is true
type ConditionOn<T> = { id: T; condition: boolean }
type Options = {
  group?: {
    // Show only when member of the-specified groups
    on: GroupId | GroupId[] | ConditionOn<GroupId>[]
    // When true, show only when not member of the-specified groups
    complement?: boolean
    // When true, substitute has to have action permission
    action?: boolean
  }
  status?: {
    // Show only when item in the-specified statuses
    on: StatusId | StatusId[] | ConditionOn<StatusId>[]
    complement?: boolean
    // Show when the-condition is true
    condition?: { on: boolean }
  }
}

// Hook evaluating the-options in the-context
// of the-current user and the-displayed item
function usePermission(options?: Options): boolean

// Encapsulating components with permission options
const Permission: React.FC<Options> = ({ options, children }) => {
  const show = usePermission(options)
  if (!show) return null

  return children
}
```

Listing 4.12: Client permission helpers, hook and component

4.4.4 Substitutes

The last important requirement for the application was to allow users to see or to perform operations on items of other users. This functionality is implemented via the substitution system.

Each user can specify his substitutes in the user detail page. Furthermore, for each substitute, they can toggle a permission to allow performing actions on his behalf to the substitute and also an option for the substitute to receive the same item notifications (as shown in the attachment [F.15]).

The substitutes can set the substitution context in the client. For each user they can act as a substitute for, there is an option to turn on the user in the substitution context. The substitution context dynamically displays the content in the client corresponding to the chosen options. Furthermore, the content is automatically refreshed with the substitution context settings, so the substitute doesn’t have to perform any other actions. When the option
4.5 Deployment

Docker \[89\] is used for the deployment of the whole application. The server and the client code are versioned with the use of git \[90\], both parts are in one repository. In the root folder of the repository is placed a Dockerfile \[91\], which is used to build the image configured to run the application.

The image is derived from the official node alpine image \[92\]. For building the application image multi-stage building is used, which reduces the final size of the image and the required time used for building the application \[93\]. The build process was inspired by the image used for next.js applications \[94\]. The main build steps of the image can be summarized as:

1. Stage 1 (dependencies) — performed only on dependency changes.
   - Copy the package.json and yarn.lock files of the server and client.
   - Run yarn install for both parts to retrieve required node_modules.

2. Stage 2 (builder) — performed on code changes.
   - Copy all project files.
   - Copy node_modules from the dependency stage.
   - Build the application via yarn build for both parts.

3. Stage 3 (runner) — contains only build data.
   - Set time zone to Prague (for correct date related functionality).
   - Copy build data from the builder. The built client will be placed in the server folder, so it can be served by the server.
   - Expose the server port.
   - Run the server.

```
# Install dependencies only when needed
FROM node:15-alpine AS deps
# ...
```
4. Realization

```
COPY api/package.json api/yarn.lock api/
COPY app/package.json app/yarn.lock app/
RUN cd api \
  && yarn install --frozen-lockfile \
  && cd .. /app \
  && yarn install --frozen-lockfile

# Rebuild the source code only when needed
FROM node:15-alpine AS builder
# ...
COPY . .
COPY --from=deps /workflow/api/node_modules ./api/node_modules
COPY --from=deps /workflow/app/node_modules ./app/node_modules
RUN cd api \
  && yarn build \
  && cd .. /app \
  && yarn build \
  && mv .. /build .. /api/dist/build

# Production image, copy all the files and run next
FROM node:15-alpine AS runner
# ...
COPY --from=builder /workflow/api/dist ./dist
COPY --from=builder /workflow/api/node_modules ./node_modules
# ...
EXPOSE 8080
CMD ["node", "dist/index.js"]
```

Listing 4.13: Deployment, Dockerfile structure

In the end, the application can be run from the Docker container. For running it with correct parameters the `docker-compose.yml` file is used, which configures port mapping and passes environment variables to the container. There are defined three containers in the `docker-compose.yml` file configured to run different instances of the application:

- Test instance — for development application testing.
- Beta instance — for user application testing.
- Production instance — for use by the employees.

```
services:
  workflow_prod:
    image: workflow:stable
    container_name: workflow_prod
    restart: always
    ports:
      - "8083:8080"
    environment: # ...
# ...
```

Listing 4.14: Deployment, production container definition
4.5. Deployment

The application repository is hosted on a self-hosted GitLab server, which is configured, so it can communicate with the Docker platform on the company application server. In the application repository there is `.gitlab-ci.yml`, which is used to configure the GitLab CI/CD pipeline for the application. The pipeline is configured to build the test instance of the application on every commit of the next or master branches. The test pipeline has 2 stages: build (building the docker image) and deploy (deploying the image to the application server). The pipeline is also configured to build and deploy the beta instance for commits on the beta branch as well as to build and deploy the production instance for commits on the master branch. There is one exception for the beta and production pipelines and that is the need to manually trigger the deployment stage.

```yaml
image: docker/compose:latest
variables:
  IMAGE_BASE_NAME: workflow
  IMAGE: $IMAGE_BASE_NAME:$CI_COMMIT_REF_NAME
stages:
  - build
  - stage
  - deploy
# ...
deploy_prod:
  stage: deploy
  environment:
    name: production
  variables:
    RELEASE_IMAGE: $IMAGE_BASE_NAME:stable
  script:
    - docker build -t $IMAGE .
    - docker tag $IMAGE $RELEASE_IMAGE
    - docker-compose up -d workflow_prod
  when: manual
  only:
    - master
# ...
```

Listing 4.15: Deployment, production CI/CD deploy stage

Each of the instances managed by the pipelines is configured to run on the application server, they are connected to the database and the application clients can be accessed on the company local network. One distinction is that the test instance is connected to the test database and the other instances are connected to the production database. The test database is recreated every day with the data of the production database. Using the testing instance enables checking if the build and deployment processes work properly and that they do not encounter any errors. This minimalizes the chance of deploying
the production instance, encountering error and then having to debug it while
the instance is unreachable for the users for instance. The beta instance on
the other hand allows testing new application features in production with only
a few chosen users.

Figure 4.8: Deployment diagram

Each instance runs on the application server on its own configured port.
The instances are then accessible outside the server using the configuration
of the Nginx proxy server on the application server that are exposed on
the addresses:
4.6 Documentation


SSL is not used as the certificate authority of the company is currently being set up. Therefore, the unsecured HTTP protocol is used for the time being. Security is not an issue as the application is for internal use only. Furthermore, the users are not warned of unsecure site as the domain secheron.cze is in the list of secure exceptions. However, after the certificate authority will be set up during the summer, the application will start using the SSL.

The application data on the production database is backed up every day to a remote server. The application repository (all source files and configuration required to deploy the application) is also backed up.

4.6 Documentation

The code documentation of the whole application codebase is automatically generated. In the codebase doc comments are used as standardized by the TS-Doc [98]. When appropriate (for example for core or complex functionality), the doc comments are used to describe the relevant variable, function, class, etc.

For the documentation generation the tool TypeDoc is used, which scans the code and outputs HTML files with the documentation. TypeDoc documentation automatically generates basic documentation for all code (input and output parameters and its types, etc.) and when there are present doc comments, it enhances the documentation with their data. [99]

There is generated a separated documentation for the API server and the web client. For each of those, the documentation is built by running the command yarn run docs (in the appropriate subfolder) and the outputted HTML documentation is saved in the docs subfolder.

4.7 Folder structure

The codebase is split into two projects, one for the API server and the other for the web client. Each project resides in its own subfolder. The most important parts of the codebase folder structure are in the figure 4.9.
4. Realization

```
api ........................................................ API server
__src ........................................................ Source files
  __controllers ............................................ Express Controllers
  __data ................................................. Data definitions (status responsibilities, etc.)
  __jobs .................................................. Cron job definitions
  __machines ............................................. State machine definitions
  __models ............................................... Database models
  __reports ............................................... Data reports in XLSX format
  __routes ................................................ API route definitions
  __services ............................................. Business logic and db queries
  __utils ................................................ Utility functions
    __config.ts ......................................... Configuration file
    __db.ts .............................................. Database connection definition
    __server.ts ........................................ Node entrypoint (server start-up)
    __env ................................................ Environment variables
  __package.json ........................................ Project dependencies
  __tsconfig.json ....................................... TypeScript settings
__app ........................................................ Web Application
__public .................................................. Static files (index.html, etc.)
__src ........................................................ Source files
  __components .......................................... React components
    __common ............................................. General components (used in other components)
    __forms ............................................... Form components
    __views .............................................. Components representing pages
      __App.tsx .......................................... Application root
    __hooks ............................................... React hooks
  __types ................................................ Type definitions
  __utils ................................................ Utility functions
    __config.ts ......................................... Configuration file
    __index.tsx ......................................... Application root
    __env ................................................ Environment variables
  __package.json ........................................ Project dependencies
  __tailwind.config.json ............................... Tailwind settings
  __tsconfig.json ....................................... TypeScript settings
__sql ....................................................... Helper SQL definitions
  .gitlab-ci.yml ........................................ GitLab pipeline definitions
  __docker-compose.yml ................................. Docker container definitions
  __Dockerfile ......................................... Docker image definition
```

Figure 4.9: Project folder structure
For the verification of the application’s functionalities, various kinds of testing were performed: heuristic analysis, usability testing, automatic testing and manual testing.

As the overall scope of the application is expansive and the company’s main priority was the delivery of the features, it was not possible to test everything thoroughly. Instead, the emphasis was put on the core functionalities of the state machine implementation and functionalities used by most of the users.

5.1 Heuristic evaluation

“Heuristic evaluation is a usability engineering method for finding the usability problems in a user interface design so that they can be attended to as part of an iterative design process. Heuristic evaluation involves having a small set of evaluators (in this case the thesis’s author) examine the interface and judge its compliance with recognized usability principles.” [100]

The general usability principles as defined by Nielsen [101] are:

1. visibility of system status,
2. match between system and the real world,
3. user control and freedom,
4. consistency and standards,
5. error prevention,
6. recognition rather than recall,
7. flexibility and efficiency of use,
8. aesthetic and minimalist design,
9. help users recognize, diagnose, and recover from errors,
5. Testing

10. help and documentation.

Based on the principles, the main pages of the application were evaluated. If there was encountered an issue in violation of the principles, it is listed in the lists below.

Home page, item list pages
1. OK — highlighted navigation link and notifications.
2. Some navigation icons could be clearer.

Create and edit page
1. OK — highlighted navigation link and notifications.
2. Some fields might be hard to understand for casual users.
3. Cannot undo the draft removal and item creation.
5. Some errors are shown only after submitting the item, it could be earlier.
6. Some of the main form information (item name) could be shown on all the steps.
10. Hint component of the autocomplete fields is too hidden.

Item detail page
1. OK — current status component.
2. Item fields might be hard to understand for casual users.
3. Cannot undo the primary process action.
6. Breadcrumbs could be used. Some information is hidden in the tabs.

Department timeline
1. Cannot see the item process status.
2. The small icon indicators might not make sense to the users.
6. OK — almost all item information is displayed.
7. OK — can navigate to items and users.

User detail page
1. No identification of the opened page.
6. Information hidden in tabs. Substitute permissions might not be clear.
5.2 Usability testing

5.1.1 Results

The identified issues are rated on a scale from 1 to 5 (1 being the lowest priority and 5 the highest priority).

<table>
<thead>
<tr>
<th>Issue</th>
<th>Rule</th>
<th>Priority</th>
<th>Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some navigation icons could be clearer.</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Item fields hard to understand for users.</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Cannot undo draft deletion and item creation.</td>
<td>3</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Errors shown only after submitting the item.</td>
<td>5</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Form information shown on all steps.</td>
<td>6</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>Autocomplete hint hidden.</td>
<td>10</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Item fields hard to understand for users.</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Cannot undo primary process action.</td>
<td>3</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Breadcrumbs could be used.</td>
<td>6</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Information hidden in the tabs.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cannot see the item process status.</td>
<td>1</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>Small icon indicators not clear.</td>
<td>2</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>No identification of the opened page.</td>
<td>1</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>Information hidden in tabs.</td>
<td>6</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>Substitute permissions might not be clear.</td>
<td>6</td>
<td>2</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.1: Heuristic analysis, found issues

Most of the issues identified were fixed with a simple addition of a hint text or tooltips on elements. Some unresolved issues as the undo actions might not be resolved at all as it would require a significant effort to implement the functionality to resolve them, and they do not have a high priority for the company. The rest of the unresolved issues tied to the better presentation of the information will be resolved in the near future.

5.2 Usability testing

“In a usability-testing session, a researcher (called a facilitator or a moderator) asks a participant to perform tasks, usually using one or more specific user interfaces. While the participant completes each task, the researcher observes the participant’s behaviour and listens for feedback.” [102]

The usability testing was in this case used for gathering a feedback about the functionality of the application which most of the users will use.

5.2.1 Process

The testing was concluded remotely with the company employees and one external tester. The users tested the application on their computers in the Chrome web browser. Their screen and voices were recorded during the testing. The tests were
performed in the Czech language and the instructions and results were then translated to English.

The testing was mainly targeted at the basic functionality of the application:
- Item creation, using drafts.
- Search for an item based on various criteria.
- Managing an item, looking up data.

5.2.2 Briefing
Before the start of the questionnaire and the testing, the users were briefed:
- Do you agree to the recording of the testing process?
- The testing will be focused on the item creation, searching and working with items.
- The testing will be focused only on the Request item type.
- Do not be afraid to speak up your mind during the whole testing process.
- Do not be afraid to voice your opinion and be critical.

5.2.3 Pre-test
The briefing was followed by a questionnaire used for gathering the user’s background regarding their previous encounters with the prototype of the application:
1. Have you already encounter the workflow application?
2. How often do you use the application?
3. What actions do you usually do in the application?
4. Summarize your feelings about the application.

5.2.4 Scenarios
The testing was split into multiple scenarios. The scenarios were designed to simulate work of a casual user of the application. The users were instructed to perform steps of the scenarios listed below one by one. Every step was read to them only when they completed the previous step or when it was decided that they will not probably be able to finish the step by themselves.

5.2.4.1 Item creation, using drafts
This scenario was targeted at the item creation page, mainly from a standpoint of a casual user. It was analysed if the users could understand what they were supposed to do to have their request received. Furthermore, it was tested if the autosave and draft functionality was intuitive.
5.2. Usability testing

5.2.4.1.1 Item creation
1. You want to create a new request. Find and open the creation form.
2. You want to request some pencils, you can choose to specify them further. You need the pencils in the next 14 days. The pencils are supplied by the company COPIA.
3. Do you understand the field Business Unit?
4. Attach a file to the request.
5. You have just realized that you chose the wrong supplier, choose another one.
6. Continue filling up the form. Some additional information for your request: you need 2 pencils and one costs 5 Czech crowns.
7. Save the draft of the request.
8. Continue filling up the form. Can you tell what is going to happen next after you submit the item?
9. Submit your form.

5.2.4.1.2 Using drafts
1. Open the form for a request item creation.
2. This time you want 3 pencils and one costs 7 crowns.
3. Your colleague asked you for some information about another item. Navigate to the Home page.
4. Return to the form.
5. This type of request is now handled only by the purchasing department of the TPS business unit, change the form data.
6. Submit your form.

5.2.4.2 Searching for items
This scenario is focused on testing how the users prefer to search for the items and if the application provides the users with enough tools to be able to complete the search as soon as possible.
1. Find an item based on these criteria: you have just received an email about the need of approving an item, its item number is R2596.
2. Find an item based on these criteria: you know that the item is not completed yet, its status is Ordered or Ready to order and its requestor is Petr Svoboda.
3. Find an item based on these criteria: you know that the item is completed and its supplier is SABTECH, S.R.O.
4. Open this item.
5. Find an item based on these criteria: you know that the item is not completed yet and its supplier is GAROMA PLUS, spol. s.r.o.
5. Testing

5.2.4.3 Managing item, looking up data
This scenario tests the structure of the item detail page and if the information is displayed in a format the users expect.
1. Find and open the item R2259. Imagine you were its requestor.
2. Can you tell in which state it currently is, who is responsible for it, and what will happen with the item next (based on the information in the application)?
3. Find out more about the item’s approval information.
4. Describe the process of this item since its creation.
5. Add comment to the item in which you mention the item’s supplier.
6. You picked up the item, indicate it in the application.

5.2.5 Debriefing
After the users completed the testing scenarios, they were asked a series of questions evaluating their overall feelings about the application:
• Summarize your feelings about the application compared to the prototype.
• How do you generally like the work with the application?
• What is your biggest issue with the application?
• Do you have any other suggestions that would help you with your work?

5.3 Results
During the user testing, notes were made, which are used for the identification of the main issues encountered. The notes can be found in the attachment G.1

5.3.1 Users
Five and a half tests were conducted in total (connection was lost with one of the testers in the middle of the test).

Testers
• Nela Koliášová, a student, never worked with the application before.
• Jakub Sláma, an employee, works regularly with the application.
• Vladimír Jirásek, an employee, works regularly with the application.
• Eva Májová, an employee, works occasionally with the application.
• Václav Suchý, an employee, works regularly with the application.
• Kristýna Váňová, an employee, works regularly with the application, test interrupted.
5.3. Results

5.3.2 Summary

The issues found during the testing were summarized and put into the following table. The issues were categorized into the following categories: UI, bugs and system.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Fix</th>
<th>Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create form title not clear, subitems step not clear</td>
<td>Renamed form title from Item to Request, Subitem name changed to Item, added description to the Subitem specification step</td>
<td>Yes</td>
</tr>
<tr>
<td>Not clear what to do on the subitem step in create form</td>
<td>Added primary button to the middle of the form for adding items</td>
<td>Yes</td>
</tr>
<tr>
<td>Create form fields not clear enough — item name, business unit, order method, supervisor, subitem details</td>
<td>Added descriptions</td>
<td>Yes</td>
</tr>
<tr>
<td>Hint button on autocomplete fields is hard to find or users do not understand it might help them</td>
<td>Did not find a better solution without taking up too much space of the form</td>
<td>No</td>
</tr>
<tr>
<td>Cannot click on items in the table showed after clicking the hint button</td>
<td>Clicking on the row fills the field with value</td>
<td>Yes</td>
</tr>
<tr>
<td>Global search (in navigation drawer) — not clear what is searching by</td>
<td>Added better description</td>
<td>Yes</td>
</tr>
<tr>
<td>No search in filter menu</td>
<td>Complex functionality, but useful</td>
<td>No (yet)</td>
</tr>
<tr>
<td>More and clearer search parameters in item lists</td>
<td>Added supplier, improved parameter labels</td>
<td>Yes</td>
</tr>
<tr>
<td>Users do not notice tabs in item detail</td>
<td>Tab headers were made clearer</td>
<td>Yes</td>
</tr>
<tr>
<td>Clearer notifications after loading draft or previous form state</td>
<td>Added panel with information</td>
<td>Yes</td>
</tr>
<tr>
<td>Users want the most important information more highlighted in the item detail</td>
<td>Highlighted supplier name, improved comment position</td>
<td>Yes</td>
</tr>
<tr>
<td>Users do not understand how to load data from drafts in creation form or do not even know there is such option</td>
<td>Improved arrow icon button, added description</td>
<td>Yes</td>
</tr>
<tr>
<td>Option to click on “x items requested” on dashboard and be taken to appropriate screen</td>
<td>Option is now clickable</td>
<td>Yes</td>
</tr>
<tr>
<td>When in item list, after clicking on item and then going back, return to the scroll position</td>
<td>Requires complex rewrite</td>
<td>No (yet)</td>
</tr>
</tbody>
</table>

Table 5.2: Usability testing, user interface issues
5. Testing

<table>
<thead>
<tr>
<th>Issue</th>
<th>Fix</th>
<th>Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid date instead of actual date in history log</td>
<td>Caused by the user’s system setting, is not easily replicable, needs thorough testing over longer time period</td>
<td>No</td>
</tr>
<tr>
<td>File input is not correctly displayed</td>
<td>Caused by the user’s system setting, is not easily replicable, needs thorough testing over longer time period</td>
<td>No</td>
</tr>
<tr>
<td>No data when opening draft details on create screen</td>
<td>Data is shown, format could be better, open item button was deleted as it didn’t make sense for draft</td>
<td>Yes</td>
</tr>
<tr>
<td>Form shows error when user clicks on next</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Creation form doesn’t show user’s previous items</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Using back button breaks some functionality tied to route change</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>(saving form, filter options)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Better history log messages</td>
<td>Needs complex rewrite on the back-end side</td>
<td>No</td>
</tr>
<tr>
<td>New users didn’t understand active and archived lists</td>
<td>Better design or merging the two views will be investigated</td>
<td>No</td>
</tr>
<tr>
<td>Filter out allowed accounts or departments based on chosen value in creation form</td>
<td>Complex rules, need more information</td>
<td>No</td>
</tr>
<tr>
<td>Better mail notifications after adding comment</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>More refined draft management, allow rewriting drafts</td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.3: Usability testing, bugs and system issues

5.4 Automatic testing

For a complex application which is critical for the company it is important that it will function correctly as was defined in the requirements. The usability testing does not cover this requirement as it targets mainly the user interaction with the application rather than the correctness of the application.

As it is not possible to cover all the application functionalities with the automatic tests due to the time constraints, critical parts of the system were identified, and are covered by the following types of testing.

5.4.1 Unit testing

The state machine library as the core of the application is covered by unit tests with a 100% coverage. Similarly, the utility module is almost fully covered as the functions it provides are used both in the server and the client in the critical parts of the application’s functionality.
5.4.2 Integration testing

The business logic of managing the state machine instances of the item types is closely linked with the state machine library. It is also important for this part of the application to function correctly as it simulates the company processes which could decide about approving a purchase of an item with a very high price or allowing employees to take a long leave of absence. Therefore, the functionality of sending the state machine events is covered by the integration tests.

The integration tests are used to test the communication between the server’s business logic of managing the state machine instances and the database. At the beginning of the tests, testing items are initialized in the testing database. Then every possible event transition of the item types is tested. The results of the performed or not performed transitions are checked by analysing the database data (mainly the status change table and the item tables).

The integration tests are also defined using the ts-jest library.

5.4.3 End-to-end testing

Finally, end-to-end tests are used to test the application’s correctness as a whole. They are used to test the application stack from the user interaction in the client, through the server’s business logic, to the data persistence in the database. The tests to simulate the basic functionalities of the application are the:

- rendering of all the application pages,
- item creation of all types (with various combinations of parameters),
- item editing,
- sending item events (single events on the item detail page and multiple events on the home page to the selected items).

The library cypress is used to perform the tests.

5.5 Manual testing

On top of all the already mentioned testing, the application is regularly tested manually when a new version is pushed to the production environment. The manual testing is performed by a part-time employee. Their job is to test all the basic functionality of the application.

This step is important as the application isn’t fully covered with the automatic tests. Some actions are easier to perform manually than simulating them in the automatic tests and the tester can quickly identify the probable cause of the errors if they encounter them as they are thoroughly trained in the application’s functionalities. However, this means that the testing depends on the absence of a human error during the manual testing and unnoticed errors could get into the production environment. For this reason, the ratio of the manual to the automatic tests will be reduced in the future and the tester will hopefully define automatic tests for the tasks functionalities tested manually.
Chapter 6

Evaluation

This chapter describes the gradual process of adopting the application in the company. Additionally, it shows aggregate data regarding the usage of the application. And finally, it comments on the future development of the application.

6.1 Application adoption

The alpha prototype of the application was released in November 2017 (examples can be seen in the attachment F.1). It was used mainly for testing the various approaches of managing the process of the prototype items. Around 30 users were involved in the testing of the application. The users were the members of these following departments: mechanical design, purchasing of the machining department and then the methods, receipt of goods and inspection.

In November 2019 the beta prototype of the application was released (examples can be seen in the attachment F.2). The beta implemented the process management of the item types request and service using the prototype version of the state machine library documented in the thesis. During this time the Prague branch management decided to adopt the application prototype by the whole company and abolish the original company processes. At this time most of the users with the access to the company computers have encountered the application (there are currently 384 registered users).

With the feedback gathered from the prototypes, the application was rethought and redesigned as described in the thesis. The focus was mainly targeted at the user experience and improvement of the company processes rather than just the previous reimplementation. Furthermore, the absence item types were added. The application was officially released on the 25th of February 2021. Currently, the absence item types are used just by the [TPS] business unit, but they will be probably adopted by the whole company before or during the summer of 2021.

6.2 Statistics

As was mentioned in the section above, the prototype version of the application was in use since the end of the year 2017. During that time roughly 50 prototype
items were requested per month on average. After the release of the beta prototype, the total request count per month was usually above 250 items. The official release was followed by a reduction in the total requests which was caused by the Covid-19 pandemic, but in the year 2021 the request count is increasing to around 350 items per month and after the decision to use the absence items by every business unit, it will probably climb to around 450-500 items per month.

To present some metric of the process improvement, the time it takes to complete a request can be used (the amount of time between the request creation and the request’s transition to a final state). When looking at the prototype items, there can be seen a significant initial reduction of time during the first year. The completion time of the first items could be considered similar or slightly better than the completion time of the prototype items before the use of the application. The subsequent improvement is probably caused by the gradual adoption of the application and the revision of the background processes, which the application allowed by providing the necessary data. When looking at the request and service items, the trend of their completion time almost stagnates. This is completely fine, as the average time (15 to 20 days for the requests and 15 days for the services) is within the company guidelines. For example, the process for requests should take around 7 business days (1 day for a purchaser assignment, 2 days for an approval, 3 days for placing an order and 1 day for
the item pickup) and additionally the average time it takes the supplier to deliver an item is 15 days. On the application side, not much can be improved about that. The absence of the initial improvement in the application is probably due to the fact that the purchasers were already familiar with the application. However, the improvement contrary to the situation before the usage of the application was confirmed by the head of purchasing.
6. Evaluation

![Graph showing active users per month]

The interaction of the users with the application can be shown in the amount of the active users (users who performed some action in the application during the month). As can be seen in the graph, the interaction roughly corresponds to the item request amount.

6.3 Future development

The requirements defined for the application were fulfilled, but there are already appearing more requests for an additional development. The requests can be split into the categories: new item types, application features and integration with other systems.

As the application can handle complex processes of different item types, there are currently 2 requests for an item type addition. The first and more connected to the application is the purchase offer type. Currently, when the requestor is in a need of something, but they are not entirely sure what exactly they need, how much it will cost or where to buy it, they use the purchase request item for it. The purchasing managers are asking for a separate item type which will be used in these cases. The new item would allow specifying multiple offers from the suppliers based on the requestor’s parameters and the requestor could choose one of them. Then it would be possible to automatically create and link a purchase request with the offer data. This would improve the decision-making in the approval process as all approvers could see that there was performed a proper process of evaluating the multiple purchase
offers. The second item type is for the process of liquidation of the company property, which is currently performed in the XLSX sheets with a combination of sending data via emails. The benefits of implementing this item type are therefore similar to the already implemented item types. Furthermore, the approval process is almost the same as for the purchase request item.

The requests for application features contain: ease of use improvements requested by the users, more group action on the items, more data export possibilities, etc. They are usually simple to implement, but there wasn’t enough time to do so already. Their implementation is ongoing.

The most complex of the categories is the integration with the other systems. There are currently two integrations on which the development will hopefully soon begin. Firstly, there is the integration with the attendance system, which would allow to import the absence item data into it. This step was supposed to happen during the development of the application when writing this thesis. Unfortunately, the assistance of the company developing the attendance system is required, and they are not very responsive. Secondly, the integration with the ERP database will be extended. Currently, the purchaser has to supply the order data of the purchase items into both of the systems as each of them is used for a different purpose. As the ERP database is supposed to be the primary data store (as per the company guidelines), the workflow application could just link the order data from the ERP database. After that, it would automatically synchronize the data with it. Furthermore, there could be a step of uploading the purchase item data into the ERP database automatically as the data definition is performed in the application with the combined effort of the requestors and the purchasers. This cannot be done in the ERP system, but the finalized data has to be entered into it in the end. Regrettfully, for this integration, the assistance of the colleagues in charge of the ERP database is needed and the process of consulting with them can take years.
Conclusion

The goal of this thesis was to implement an application which could replace the company’s partially digital processes of requesting purchase items and leaves of absence. In addition, there was an opportunity not only to replace the old processes, but to also enhance them with the usage of the web technologies and with the ability to integrate with the existing systems of the company.

The resulting application achieved all those goals. The company’s leadership was satisfied with the implemented solution and decided to fully abolish the original company’s processes in favour of the fully digital versions. Not only was the leadership satisfied, but after consulting with them, the application might be extended with other not yet digitalized processes. Furthermore, other branches of the company showed interest in adoption of the application for their purposes.

One part of the success of the solution can be attributed to the use of the state machines for simulation of the company’s processes. The deterministic nature of the processes in the form of the state machines allowed their perfect simulation. On top of that, any change to the process is a trivial task of only tweaking the process definition file. The other part of the success can be attributed to the user interface. A significant effort was placed on the user experience during the development of the web client. Most of the users seem to appreciate it based on the conversations the author had with them at work or during testing (they are used to an outdated software which often leaves them frustrated).

Although the task was successfully achieved, the development of the application will still continue, probably for years. As the application is much more flexible compared to the previous situation, requests for changes to the processes were made with the target of reducing the administrative tasks or the item delivery time.


19. CREATECH GROUP, INC. *Good Day - Inspiring work management* [online] [visited on 2021-04-04]. Available from: https://www.goodday.work/

20. QUIXY. *Platform Overview / Quixy* [online] [visited on 2021-04-04]. Available from: https://quixy.com/platform-overview/


22. SOLID IT. *DB-Engines Ranking* [online] [visited on 2021-04-06]. Available from: https://db-engines.com/en/ranking


24. POSTGRESQL GLOBAL DEVELOPMENT GROUP. *Foreign data wrappers* [PostgreSQL wiki] [online] [visited on 2021-01-02]. Available from: https://wiki.postgresql.org/wiki/Foreign_data_wrappers

25. ORACLE CORPORATION. *Cost-optimized and High-Performance Database* [online] [visited on 2021-04-06]. Available from: https://www.oracle.com/database/

Bibliography

27. POSTGRESQL GLOBAL DEVELOPMENT GROUP. 
   `postgres_fdw` [PostgreSQL Documentation] [online]. 
   2020 [visited on 2021-01-02]. Available from:
   https://www.postgresql.org/docs/13/postgres-fdw.html

   Available from: https://www.hostinger.com/tutorials/cron-job

29. THE LINUX FOUNDATION. *What is Linux?* [online] 
   [visited on 2021-04-06]. 
   Available from: https://www.linux.com/what-is-linux/

30. CITUS DATA. `pg_cron` [online]. 
   Citus Data, 2021 [visited on 2021-01-02]. 
   Available from: https://github.com/citusdata/pg_cron

31. MOZILLA FOUNDATION. *JavaScript* [online] 
   [visited on 2021-04-06]. Available from: 

32. MICROSOFT. *Typed JavaScript at Any Scale.* [online] 
   [visited on 2021-04-06]. 
   Available from: https://www.typescriptlang.org/

33. OPENJS FOUNDATION. *About Node.js* [online] 
   [visited on 2021-04-06]. 
   Available from: https://nodejs.org/en/about/

34. GREIF, Sacha; BENITTE, Raphaël. 
   *State of JS 2020: Back-end Frameworks* [online] 
   [visited on 2021-04-06]. 

35. SAVIN, Igor. *A SQL Query Builder for Javascript* [online] 
   [visited on 2021-04-06]. Available from: http://knexjs.org/

36. MOZILLA FOUNDATION. 
   *Using FormData Objects - Web APIs* [online]. 
   2021 [visited on 2021-04-06]. 
   Available from: https://developer.mozilla.org/en-US/docs/Web/API/FormData/Using_FormData_Objects

37. OPENJS FOUNDATION. *multer* [online]. 
   expressjs, 2021 [visited on 2021-04-06]. 
   Available from: https://github.com/expressjs/multer
   original-date: 2014-01-30T18:16:04Z.

38. KELEKTIV. `node-cron` [online]. 2021 [visited on 2021-05-03]. 
   Available from: https://github.com/kelektiv/node-cron
   original-date: 2010-06-23T01:25:18Z.


41. OPENSPECS-OFFICE. [MS-XLSX]: Excel (.xlsx) Extensions to the Office Open XML SpreadsheetML File Format [online] [visited on 2021-04-06]. Available from: [https://docs.microsoft.com/en-us/openspecs/office_standards/ms-xlsx/2c5dee00-eff2-4b22-92b6-0738acd4475e](https://docs.microsoft.com/en-us/openspecs/office_standards/ms-xlsx/2c5dee00-eff2-4b22-92b6-0738acd4475e).


44. TypeORM - Amazing ORM for TypeScript and JavaScript (ES7, ES6, ES5). Supports MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL databases. Works in NodeJS, Browser, Ionic, Cordova and Electron platforms. [online] [visited on 2021-04-06]. Available from: [https://typeorm.io/](https://typeorm.io/).


46. KHUDOIBERDIEV, Umed. *TypeORM contributors* [online] [visited on 2021-04-06]. Available from: [https://github.com/typeorm/typeorm/graphs/contributors](https://github.com/typeorm/typeorm/graphs/contributors).

47. DEPOLD, Sascha. *Manual | Sequelize* [online] [visited on 2021-04-06]. Available from: [https://sequelize.org/master/](https://sequelize.org/master/).


50. FACEBOOK INC. *React – A JavaScript library for building user interfaces* [online] [visited on 2021-04-06]. Available from: [https://reactjs.org/](https://reactjs.org/).
51. HENGEVELD, Gert. *react-async* [online].
Async Library, 2021 [visited on 2021-04-06].
Available from: https://github.com/async-library/react-async
original-date: 2018-08-12T11:53:05Z.

52. LUO, Bill. *Home* [online] [visited on 2021-04-06].

53. VAUGHN, Brian. *react-virtualized* [online].
2021 [visited on 2021-04-06].
Available from: https://github.com/bvaughn/react-virtualized
original-date: 2015-11-03T00:48:07Z.

54. WATHAN, Adam. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.* [online] [visited on 2021-04-06].

55. FACEBOOK INC. *Context – React* [online] [visited on 2021-04-06].
Available from: https://reactjs.org/docs/context.html

56. ABRAMOV, Dan. *Redux - A predictable state container for JavaScript apps. / Redux* [online] [visited on 2021-04-06].
Available from: https://redux.js.org/.

57. WESTSTRATE, Michel. *MobX* [online] [visited on 2021-04-06].
Available from: https://mobx.js.org/index.html

58. SVOBODA, Petr. *Webový editor konečných automatů* [online].
2019 [visited on 2021-04-06].
Available from: https://dspace.cvut.cz/handle/10467/83509

59. DODDS, Kent C. *Application State Management with React* [online].
2020 [visited on 2021-04-06].

60. TASSINARI, Olivier; NGUYEN, Hai.
*Material-UI: A popular React UI framework* [online]
[visited on 2021-04-06]. Available from: https://material-ui.com/

61. GREIF, Sacha; BENITTE, Raphaël.
*State of JS 2020: Other Tools* [online] [visited on 2021-04-06].

62. ISONEN, Oleg. *JSS* [online] [visited on 2021-04-06].
Available from: http://cssinjs.org/

63. TASSINARI, Olivier. [*RFC* Material-UI v5] [online].
2020 [visited on 2021-04-06].
Available from: https://github.com/mui-org/material-ui/issues/20012\#issue-577205880

65. HOSCH, William L. *Client-server architecture / computer science* [online] [visited on 2021-04-06]. Available from: https://www.britannica.com/technology/client-server-architecture

66. What is a REST API? [online] [visited on 2021-04-06]. Available from: https://www.redhat.com/en/topics/api/what-is-a-rest-api


68. MOZILLA FOUNDATION. *Routes and controllers* [online] [visited on 2021-04-08]. Available from: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes


71. THE ROCKET SCIENCE GROUP. *Email Client CSS Support* [online] [visited on 2021-04-08]. Available from: https://templates.mailchimp.com/resources/email-client-css-support/

72. NPM, INC. *About packages and modules / npm Docs* [online] [visited on 2021-04-15]. Available from: https://docs.npmjs.com/about-packages-and-modules


Bibliography


86. POSTGRESQL GLOBAL DEVELOPMENT GROUP. 
   Materialized Views [PostgreSQL Documentation] [online].

87. MATHEUSOL. postgresql - How can I ensure that a materialized view is always up to date? [Stack Overflow] [online] [visited on 2021-01-03].
   Available from: https://stackoverflow.com/questions/29437650/how-can-i-ensure-that-a-materialized-view-is-always-up-to-date

88. THE APACHE SOFTWARE FOUNDATION. 
   18 ACTION RESOURCE & REPRESENTATION [online] [visited on 2021-04-20]. Available from: https://www.restfulobjects.org/spec/1.0/section-c/chapter-18.html

89. DOCKER, INC. Get Started with Docker [online] [visited on 2021-01-03].
   Available from: https://www.docker.com/get-started

90. MOZILLA FOUNDATION. Git [online] [visited on 2021-04-21].

91. DOCKER, INC. Dockerfile reference [online].

92. DOCKER, INC. Use multi-stage builds [online].

93. VERCHEL. Deployment [online] [visited on 2021-04-21].
   Available from: https://nextjs.org/docs/deployment

94. GITLAB. Iterate faster, innovate together [online] [visited on 2021-04-21]. Available from: https://about.gitlab.com/

95. GITLAB. GitLab CI/CD [online] [visited on 2021-04-21].
   Available from: https://docs.gitlab.com/ee/ci/

96. EDWARD S. 
   The Easiest Way to Set Up an Nginx Reverse Proxy [online].
   2019 [visited on 2021-04-21].
Bibliography

98. MICROSOFT. *TSDoc* [online] [visited on 2021-04-13]. Available from: https://tsdoc.org/.


original-date: 2016-08-30T13:47:17Z.

original-date: 2015-03-04T00:46:28Z.

Appendix A

Acronyms

AD  Active Directory.
API  Application Programming Interface.
BU  Business Unit.
CRUD  Create, read, update and delete.
CSS  Cascading Style Sheets.
CSV  Comma-Separated Values.
DOM  Document Object Model.
ERP  Enterprise Resource Planning.
ESM  Extended (finite) state machine.
FDW  Foreign Data Wrappers.
FSM  Finite state machine.
HSM  Hierarchical (finite) state machine.
HTML  HyperText Markup Language.
HTTP  Hypertext Transfer Protocol.
JSON  JavaScript Object Notation.
ORM  Object–Relational Mapping.
SMTP  Simple Mail Transfer Protocol.
SQL  Structured Query Language.
SSL  Secure Sockets Layer.
TPS  Traction Power Systems.
UI  User Interface.
ACRONYMS

URL Uniform Resource Locator.
UX User Experience.
VPN Virtual Private Network.
WMS Workflow Management System.
Appendix B

Contents of enclosed CD

- screenshots.......................... application screenshots
  - alpha.......................... application screenshots of the alpha prototype
  - beta............................ application screenshots of the beta prototype
- src...................................................... source codes
  - api.................................. OpenApi specification of the micro-services
    - authentication.yml........ authentication service specification
    - mailer.yml..................... mailer service specification
  - thesis.................................. \LaTeX{} source codes of the thesis
- readme.txt.................................. CD contents description
- showcase.mp4.......................... video showcasing the application
- task.pdf................................. thesis task in the PDF format
- thesis.pdf.............................. thesis text in the PDF format
Appendix C

Competition screenshots

C.1 Jira

Figure C.1: JIRA, pro — Issues page

Figure C.2: JIRA, pro — issue list

151
C. Competition screenshots

Figure C.3: JIRA, pro — Create page
Figure C.4: JIRA, con — Home page
C. Competition screenshots

Figure C.5: JIRA, con — Summary

Figure C.6: JIRA, con — Issues page
C.2 GoodDay

Figure C.7: GoodDay, pro — Calendar

Figure C.8: GoodDay, pro — Time tracking
C. Competition screenshots

Figure C.9: GoodDay, pro — Issue list

Figure C.10: GoodDay, pro — Issue view
C.3. Quixy

Figure C.11: GoodDay, con — Issue list

Figure C.12: Quixy, con — Home page
C. Competition screenshots

Figure C.13: Quixy, con — Project page
Appendix D

Wireframe and task graph

Figure D.1: User interface, task graph
D. Wireframe and Task Graph

Figure D.2: Homepage

Figure D.3: Create page, first step
Figure D.4: Create page, second step — no items

Figure D.5: Create page, second step — with items
D. WIREFRAME AND TASK GRAPH

Figure D.6: Create page

Figure D.7: Active items page
Figure D.8: Archived items page

Figure D.9: Help page
API definition

API definition of services and the API server described using OpenAPI 3.0.

E.1 Mailer

Listing E.1: API definition: Mailer

```json
openapi: 3.0.1
info:
  title: Mailer service
  description: >
    Service used sending email to employee's work emails.
    It is able to send simple notification style emails or
    emails with compact tabular data.
  version: 1.0.0
servers:
  - url: http://mailer.secheron.cze/
paths:
  /mail:
    post:
      summary: Send email
      description: >
        Sends email based on provided configuration
        to defined employees. It can either send
        simple text email or email with tabular data.
      requestBody:
        content:
          application/json:
            schema:
              oneOf:
                - $ref: '#/components/schemas/Text'
                - $ref: '#/components/schemas/Table'
            required: true
      responses:
        200:
          description: Mails successfully sent.
          content: {}
        404:
          description: Error.
```
E. API definition

```json
content: {}
components:
schemas:
  Base:
    type: object
    required:
    - _type
    - to
    - subject
    - title
    properties:
      _type:
        type: string
        enum: [text, table]
        description: Type of the message content: 'text' for simple text email, 'table' for email with tabular data
      to:
        oneOf:
        - type: string
        - type: array
          items:
            type: string
          description: Email recipient or recipients.
      subject:
        type: string
        description: Email subject.
      title:
        type: string
        description: Application title.

Text:
  allOf:
  - $ref: '#/components/schemas/Base'
  - type: object
    required:
    - content
    properties:
      _type:
        type: string
        enum: [text]
      header:
        type: string
        description: Text header in the email body.
      content:
        type: string
        description: Text content in the email body.
      link:
        type: string
        description: Shows link button to the specified string.

Table:
  allOf:
  - $ref: '#/components/schemas/Base'
  - type: object
    required:
    - headerRow
    - rows
    properties:
      _type:
        type: string
        enum: [table]
```

166
E.2 Authentication

Listing E.2: API definition: Authentication

```json
openapi: 3.0.1
info:
  title: Authentication service
  description:>
    Service used for company employee authentication.
    Service provides multiple authentication methods, namely:
    Prague site's Active Directory, Geneva site's Active Directory and
    Prague site's attendance system.
  version: 1.0.0
servers:
  - url: http://auth.secheron.cze/
paths:
  /login:
    post:
      summary: Authenticate a user
      requestBody:
        description:>
          User is authenticated with his username and password
          for the chosen authentication method.
        content:
          application/json:
            schema:
              type: object
              properties:
                username:
                  type: string
                password:
                  type: string
```

E.2 Authentication

Listing E.2: API definition: Authentication

```json
openapi: 3.0.1
info:
  title: Authentication service
  description:>
    Service used for company employee authentication.
    Service provides multiple authentication methods, namely:
    Prague site's Active Directory, Geneva site's Active Directory and
    Prague site's attendance system.
  version: 1.0.0
servers:
  - url: http://auth.secheron.cze/
paths:
  /login:
    post:
      summary: Authenticate a user
      requestBody:
        description:>
          User is authenticated with his username and password
          for the chosen authentication method.
        content:
          application/json:
            schema:
              type: object
              properties:
                username:
                  type: string
                password:
                  type: string
```
E. API definition

```
method:
  type: string
enum:
  - SECHERON.CZE
  - SECHERON.NET
  - watt
  required: true
responses:
  200:
    description: Successful login attempt.
    content: {}  
  401:
    description: Error while authenticating.
    content: {}  
  404:
    description: User not found.
    content: {}

/verify:
  post:
    summary: Verify user token
    description: Checks the token's authenticity and expiration date.
    If the verification is successful,
    prolongs the token's expiration date.
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              token:
                type: string
                required: true
    responses:
      200:
        description: Successfully verified token.
        content: {}
      401:
        description: Invalid or expired token.
        content: {}
components: {}
Application screenshots
Figure F.1: Home page
### Requested Items

<table>
<thead>
<tr>
<th>ID</th>
<th>Item Details</th>
<th>Status</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3441</td>
<td>IT - Sophos RED: 2x Sophos RED, 1x Wi-Fi module</td>
<td>To assign</td>
<td></td>
</tr>
<tr>
<td>K3996</td>
<td>IT - Zebron TC21, Brother PT-P900BW, Operon RA8000, 3 skener Zebra pro...</td>
<td>Ordered</td>
<td></td>
</tr>
<tr>
<td>H060</td>
<td>28.02.2021 - 30.02.2021</td>
<td>Closed</td>
<td></td>
</tr>
<tr>
<td>H011</td>
<td>14.03.2021 - 29.03.2021</td>
<td>Closed</td>
<td></td>
</tr>
<tr>
<td>E3260</td>
<td>IT - Dotyky Monitory na TPS, Levé notebooky, kameru, drobnosti.</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>E3184</td>
<td>IT - Rozšíření Centra na Hanavu</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R094</td>
<td>IT - HP Store Ever SAS Fasadač LTO7?</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R0367</td>
<td>IT - Panský LTO 6 a 7 pro žalatky</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R3318</td>
<td>LEMO konektor FAA 05-534-CL/32, Komponenty pro kompresor kabelí...</td>
<td>Ordered</td>
<td></td>
</tr>
<tr>
<td>W026</td>
<td>18.03.2021 - 28.03.2021</td>
<td>Closed</td>
<td></td>
</tr>
<tr>
<td>R3318</td>
<td>Kabel AWG 2548 - 25AWG, Komponenty pro kompletní kabelí k Yoko...</td>
<td>Ordered</td>
<td></td>
</tr>
<tr>
<td>R3277</td>
<td>Komponenty Ochochy, Komponenty pro kompletní kabelí k Yoko...</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R3032</td>
<td>LEMO konektor LEMO konектор k zdrojí Yoko...</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>H026</td>
<td>18.03.2021 - 30.03.2021</td>
<td>Time Allocation</td>
<td></td>
</tr>
<tr>
<td>R3057</td>
<td>IT - Sistémka pro TPS - Virtual FAT, Foodly</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R3026</td>
<td>IT - Nákup komponent pro logistiku, Zvíří pro malé aplikace, ribonery...</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R0353</td>
<td>IT - 2x PC pro Haler, drobnosti 1x Raps PC by Tektronix, 1x normal...</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R2466</td>
<td>zvětšení jízdy - modifikace 2 motorů a jelzbřízných držák + 2 koly St...</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R2100</td>
<td>Srovnání RACO</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R3541</td>
<td>Komponenty A22 pro projekt 010107</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>P0803</td>
<td>Papír do plotu - obv / Canon</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>P0536</td>
<td>Václavík RD, komponenty vedení Blichance</td>
<td>Completed</td>
<td></td>
</tr>
<tr>
<td>R0324</td>
<td>Ochranný název přírody 254mm, Komponenty vedení Blichance</td>
<td>Completed</td>
<td></td>
</tr>
</tbody>
</table>
F. Application screenshots

Figure F.3: Active items page, request type
Figure F.4: Archived items page, request type
Figure F.5: Request item creation, step 1
Figure F.6: Request item creation, step 2
Figure F.7: Request item creation, step 2, value hint
Figure F.8: Request item creation, step 2, value hint interface
Figure F.9: Request item creation, step 3
Figure F.11: Item detail page, more actions
Figure F.12: Item detail page, revisions tab
Figure F.13: User detail page, calendar
### Absence Data

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DATE</th>
<th>TIME FROM</th>
<th>TIME TO</th>
<th>LUNCH</th>
<th>DURATION</th>
<th>NOTE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD</td>
<td>2. 3. 2021</td>
<td>08:00</td>
<td>20:00</td>
<td>60</td>
<td>10</td>
<td>covid testovani</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>3. 3. 2021</td>
<td>08:00</td>
<td>20:30</td>
<td>60</td>
<td>10.5</td>
<td>covid testovani</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>4. 3. 2021</td>
<td>08:00</td>
<td>20:00</td>
<td>60</td>
<td>10</td>
<td>covid testovani</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>5. 3. 2021</td>
<td>08:00</td>
<td>16:45</td>
<td>30</td>
<td>9.3</td>
<td>covid testovani</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>6. 3. 2021</td>
<td>09:00</td>
<td>21:45</td>
<td>120</td>
<td>10.8</td>
<td>covid testy</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>9. 3. 2021</td>
<td>09:00</td>
<td>17:45</td>
<td>120</td>
<td>10.3</td>
<td>covid testy</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>11. 3. 2021</td>
<td>09:00</td>
<td>17:30</td>
<td>30</td>
<td>8</td>
<td>covid testy</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>12. 3. 2021</td>
<td>07:30</td>
<td>13:30</td>
<td>60</td>
<td>5</td>
<td></td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>15. 3. 2021</td>
<td>08:30</td>
<td>17:00</td>
<td>30</td>
<td>8</td>
<td>covid testy, workflow reset za posl. 2 typing</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>16. 3. 2021</td>
<td>10:30</td>
<td>16:00</td>
<td>60</td>
<td>4</td>
<td>covid testy, workflow</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>18. 3. 2021</td>
<td>11:00</td>
<td>19:30</td>
<td>30</td>
<td>5</td>
<td>covid testy</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>23. 3. 2021</td>
<td>11:00</td>
<td>19:30</td>
<td>30</td>
<td>3</td>
<td>určeno serveru</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>24. 3. 2021</td>
<td>09:00</td>
<td>12:00</td>
<td>0</td>
<td>3</td>
<td>human mape</td>
<td>time liberation</td>
</tr>
<tr>
<td>HD</td>
<td>25. 3. 2021</td>
<td>12:00</td>
<td>16:30</td>
<td>30</td>
<td>4</td>
<td>human, covid testy</td>
<td>time liberation</td>
</tr>
</tbody>
</table>
Figure F.15: User detail page, substitutes
Figure F.16: User detail page, user data
F. Application screenshots

Figure F.17: Timeline page
Figure F.18: Substitutes settings
Figure F.19: Settings page
Figure F.20: Home page, dark theme
Figure F.21: Item detail page, dark theme
F.1 Alpha prototype

Figure F.22: Alpha prototype, active items page
Figure F.23: Alpha prototype, item detail page
Figure F.24: Alpha prototype, create modal
F.2 Beta prototype

Figure F.25: Beta prototype, home page
Figure F.26: Beta prototype, active items page
Figure F.27: Beta prototype, item detail page
Figure F.28: Beta prototype, create modal
User testing results

G.1 Nela Koliášová

Scenarios

• 1.1.3 — Doesn’t know what Business Unit means.
• 1.1.4 — Error in file input — doesn’t show data.
• 1.1.6 — Doesn’t understand term subitem. Doesn’t use hint with explained details.
• 1.1.7 — Form shows error even when there is none.
• 1.2.1 — Uses draft.
• 1.2.4 — Doesn’t look at the draft attributes.
• 2.1 — Uses search in navigation drawer.
• 2.2 — Wants to use search in navigation drawer, but doesn’t know what is it searching by. Application doesn’t indicate parameters of search. After that she cannot find other screens where to look for item.
• 2.3 — Expects search in filter menu.
• 3.1 — Again doesn’t know what to search by in navigation drawer.
• 3.2 — Recognizes next state based on the primary action button of the item. Doesn’t notice the status description.
• 3.3 — Looks for information in history log instead of approvals tab, but finds it after a moment.
• 3.4 — Confused about some states in history. History panel also overflows outside the screen.

Post-test questions

• Asks about the distinction of active and archived items. Suggest combine those views and provide more filters instead.
• Mentions her struggle with the navigation drawer search.
G. User testing results

G.2 Jakub Sláma

Pre-test questions
- Uses workflow every day. He mostly assigns requests to purchasers.
- Has some issues with item filtering.

Scenarios
- 1.1.2 — Suggest deadline to be initially set based on priority. Currently, it is set automatically 14 days from today.
- 1.1.3 — Some issues with Business Unit names, suggests better labels.
- 1.1.5 — Autocomplete field filters based on full-text search, he expects to see first items begin with the same letter as supplied to search.
- 1.1.6 — Doesn’t immediately notice option to add subitem. Uses information in pop-ups on account and department fields, but doesn’t notice the info icon. When in info view (after clicking on info icon next to the field), he expects to be able to select form value after clicking row in the table. Suggests filtering out departments based on the allowed value of account.
- 1.2.1 — Uses draft. Application error — it doesn’t show his requested items. He doesn’t like the arrows in drafts (which fill out form with the draft values), he expects something more noticeable.
- 1.2.3 — Saves draft before exiting. When he reopens form, he isn’t sure how to load the draft, but manages to do it. Suggest for application to notify in clearer way about a way to continue in draft.
- 2.1 — Tries searching in active items. At first doesn’t notice search by option.
- 2.2 — Doesn’t use filters only search by.
- 2.3 — He is lost as the supplier name is not in search by options. Cannot find filter button. Expects search in filter options.
- 3.2 — Application doesn’t save search by options.
- 3.3 — Application shows Invalid date when there should be Today or Tomorrow.
- 3.5 — Suggests that application should send a mail with comment’s content that is clearer than the current ones. Maybe have the mail sender be the same user as the one who submitted the comment.

Post-test questions
- Says that the application seems monotone to him (compared to the older version). Wants more of the important information highlighted — supplier name, comment, price, …
- Using back button in browser deletes filters.

G.3 Vladimír Jirásek

Pre-test questions
• Uses workflow every other day.
• Assigns request to purchasers, edits requests with wrong values.
• Has good experience with previous version.

Scenarios
• 1.1.6 — Doesn’t like the term “subitem”. Suggest it should use item in place for subitems and request in place of item. When editing account fields, finds hint button which opens table with all options. Expects when he clicks on row, it fills the form field.
• 1.1.8 — Finds description about the process after item submission.
• 1.2.2 — Wants to copy item to create request, but cannot find it. Cannot find drafts. Then doesn’t realize how to apply draft. Suggests better way of notifying users about the possibility of applying drafts.
• 1.2.3 — Saves draft before exiting.
• 2.1 — Searches in requested items view.
• 2.5 — Deletes filters using clear filters button.
• 3.3 — Cannot find approvals tab, looks for information in history.

Post-test questions
• Suggests for better implementation of the revision process.

G.4 Eva Májová

Pre-test questions
• Uses workflow very occasionally.
• Had issues with slow application of filters, unintuitive filter setting in older version. Wants to have a way to manually close item after months of inactivity from requestor.

Scenarios
• 1.1.1 — At first wasn’t sure what supervisor field is supposed to be filled with. Suggest for more information about the order method field (when to use which option) and also an option to submit an item when user doesn’t know the supplier.
• 1.1.6 — Doesn’t understand the term subitem. Price should have per unit in label. Cannot find hint button without help. Wants to click on rows in hint table. Also thinks corp number column in account hint is confusing.
• 1.1.8 — Finds information about next step in item process after submitting. Wants more detailed diagram of the whole process (show who is responsible for each step, when each transition is used).
G. User testing results

- 1.2.1 — Clicks on draft (not apply draft), doesn’t know to apply it. Suggest that it should be cleared. Also draft information isn’t shown properly by the application.
- 1.2.3 — Saves draft multiple times and complains that there are now 2 drafts with same name. Expects draft to be able to rewrite themselves or have the option to do that. Doesn’t understand the button apply draft and what it does. Also, wants to be shown the name of the item through the whole form steps.
- 2.1 — Wants to be able to click on “x items requested” and be taken to the relevant screen.
- 2.5 — Clears filters by clicking on the x on the pill. Search seems much better compared to the older version.
- 3.3 — Confused by the information in history log (why is something happening). Cannot find approvals tab. History is not important to her, she would prefer the approvals to be in its place.

Post-test questions
- Application seems clearer and better organized.
- Mostly just doesn’t like the history.

G.5 Václav Suchý

Pre-test questions
- Uses application every day. Mainly focuses on prototypes, doesn’t work much with requests.
- He needed time to get used to the previous version, but now he doesn’t have many complains.

Scenarios
- 1.1.2 — Bit confused with the prefilled deadline value.
- 1.1.3 — Doesn’t like the name of MA Business Unit name. Suggest use ESS instead.
- 1.1.6 — Tries to find the right account in the field input. Cannot find hint table without help.
- 1.2.1 — Didn’t think to use draft to fill out the forms values, needs a hint.
- 1.2.3 — Doesn’t save item progress before leaving. After returning doesn’t notice form being automatically filled and is confused about applying drafts.
- 2.1 — Uses search and finds item by id right away.
- 2.2 — Again uses search.
- 2.3 — At first doesn’t think to use archived items screen. Doesn’t think to use filters. Wants to be able to use search whenever possible.
- 2.5 — Wants more noticeable filters. Is confused about the naming of some search by options.
• 3.1 — Uses clear filters button.
• 3.2 — Cannot find approvals tab at first.

Post-test questions
• Likes new version: clearer, more detailed.
• When in long lists of items and going to some of them. When he goes back, he wants the application to automatically scroll to the row where he was before.

G.6 Kristýna Váňová

Pre-test questions
• Uses application daily. Fills out orders based on the requests.
• Has only problem with item revisions (system setting).

Scenarios
• 1.1.1 — Uses her name for the item name field, then uses description for item specification.
• Cannot find hint table.