TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Nikita Birjukovs 193953 IAIB
Aleksandr Rudoi 193564 IAIB
Dmitri Voronoi 202032 IAIB

USABILITY IMPROVEMENTS OF THESES MANAGEMENT SYSTEM PROTSESSOR

Bachelor's thesis

Supervisors: Ago Luberg
PhD

Tallinn 2023

TALLINN UNIVERSITY OF TECHNOLOGY

Infotehnoloogia teaduskond

Nikita Birjukovs 193953 IAIB
Aleksandr Rudoi 193564 IAIB
Dmitri Voronoi 202032 IAIB

LÕPUTÖÖDE KESKKONNA PROTSESSOR KASUTATAVUSE TÄIENDUSED

Bakalaurusetöö

Juhendaja: Ago Luberg
PhD

Tallinn 2023

Author's Declaration of Originality

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

Authors: Nikita Birjukovs, Aleksandr Rudoi, Dmitri Voronoi

01.06.2023

Abstract

The purpose of this thesis is to address current design issues present in the *Protsessor* application, to then further develop existing and new features like an overview of study plans for users along with changes to user rights. Our goals would include working on the implementation and visualization of statistics for users as well as working on the addition of a summary of projects assigned to students and teachers, along with attempting to provide them with notifications on the application. Adding to that, we would need to analyze the necessity of *Camunda* in the project, to then decide if it needs to be replaced, or built upon. In case we decide to keep using *Camunda*, we will need to transfer statistics from the *Camunda* admin interface to our *Protsessor* app backend for the opportunity of further implementing and visualizing the statistics in the frontend. Then, we would also need to finish the development of some of the most prominent features like the creation of student topics by students themselves.

The thesis is written in English and is 41 pages long, including 10 chapters and 6 figures.

Annotatsioon

Lõputööde keskkonna Protsessor kasutatavuse täiendused

Lõputöö eesmärk on edasi arendada lõputööde teemade rakendust *Protsessor*. Lõputöö käigus loodi uus õigustepõhine rollisüsteem, mis võimaldab administraatoril mugavamalt määrata erinevaid õigusi erinevatele projektidele. Süsteemi kasutajaliides kirjutati suuresti ümber, et see vastaks *TalTechi* stiilinõuetele ning seeläbi sai see ka vastavaks juurdepääsetavuse standardile *WCAG 2*. Protsessi käigus täiendati statistikat tudengite ja juhendajate kohta ning lisati parem teavituste süsteem. Lõputöö käigus on parandatud ka evituse protsessi, eriti on rõhku pööratud andmebaasi varundamise ja muudatuste jälgimisega. Olemasolev rakendus kasutab *Camunda* töövoo automatiseerimise tööriista, mis lisab päris palju keerukust rakenduse arendamise etapis. Lõputöö käigus analüüsisid autorid, kas *Camunda* kasutamine on õigustatud ning kas selle kasutamist peaks jätkama. Antud analüüs sai tehtud päris töö alguses. Analüüsi tulemusena otsustati *Camundaga* jätkata.

Autorid on töös välja toonud erinevad teemad, mida peaks edasi uurima või arendama, et rakendus paremini toimima saada. Kogu lõputöö vältel on parandatud dokumentatsiooni ja parandatud koodibaasi, mis peaks järgmistel arendajatel töö alustamise lihtsamaks tegema.

Töö on kirjutatud inglise keeles, selles on 41 lehekülge, 10 peatükki ja 6 joonist.

List of Abbreviations and Terms

AD	(Azure) Active Directory
API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Development
CSS	Cascading Style Sheets
DevOps	Development and operations
DMN	Decision Model and Notation
dockerized	Application that is packed, deployed, and ran with the use of Docker containers
HTML	HyperText Markup Language
I18n	Internationalization
IT	Information Technology
JAAS	Java Authentication and Authorization Service
JSON	JavaScript Object Notation
OACC	Object Access Control
ORDBMS	Object-Relational Database Management System
RDBMS	Relational Database Management System
SMTP	Simple Mail Transfer Protocol
SPA	Single-Page Application
SSH	Secure Shell
SQL	Structured Query Language
UI	User Interface
UX	User Experience
WCAG	Web Content Accessibility Guidelines
XML	Extensible Markup Language

Table of Contents

List of Figures

1. Introduction

In many different universities, theses topic selection process is accompanied by a lot of stress and struggles, due to its importance of it. Not only is this a struggle for the students, but also teachers, who do not know if the topic they are suggesting presents any interest to others, or if a chosen student was a proper choice for this topic. That was the case because there was no easy way to go in-depth about many different theses topics to make a proper decision.

Until recently, in *TalTech*, there existed no such system that would have allowed students and teachers to manage theses topic selections in an easy and concise way. Although such a system has been implemented, it currently only is utilized by the *IT* department. The currently existing solution is a web application *Protsessor*, however, due to it being a new and in-development feature made by *TalTech* students, it requires improvement work to be done, before it would become a more useful and superior method of theses topic selection, than the method that existed before.

Our goal is to improve upon the existing project and hopefully make it superior to the pre-established old methods of theses topic selection. We hope to provide *Protsessor* with a better visual design, that would be better understood by students, as well as simplify some aspects for teachers. In addition to that, we would work on adding and improving features that would let users add and apply to different theses topics easier. Beyond that, we are hoping to improve *Protsessor* in a way that would make the changes and implementations of future developers come with fewer issues.

2. Task Proposal

The task of the given thesis is to develop further the existing system for managing theses and projects, called *Protsessor*. Currently, the *Protsessor* is only used by teachers and students of the *IT* department of *TalTech*. This thesis aims to overhaul important aspects of *Protsessor*, such as user interface (*UI*) with user experience (*UX*), as well as add new features and rework older implementations of roles and permissions.

One of the main priorities was the redesign of the *UI* which needed to be in compliance with the *TalTech* style guide [1, 2]. The current implementation of the *UI* had noticeable color combination issues that did not meet the *AA* level of *WCAG 2.0* standards [3], which made it harder to navigate through *Protsessor* for visually impaired people. In addition to *UI* changes, in order to scale the usability of *Protsessor*, we needed to rework the existing permission system, which did not allow the creation of custom global permissions for users, along with missing permissions for certain user types. Due to those issues being present in the permission system, students were not capable of proposing their own topics, along with teachers not being capable of creating topics for other students, without the admin having to grant each permission to each group of people manually.

Another big part of this thesis was related to an open workflow and decision automation platform *Camunda* which was used as a base for process management, addition and/or removal of certain process changes, and gathering of statistics regarding the related processes. Our goal was to determine whether or not, there was merit in using *Camunda* for the future development of *Protsessor*. In case we would have decided to keep *Camunda* in the project, we would have had to learn how to work and build new features around it, otherwise, it would have been required to find a way to efficiently remove and replace it.

After the changes are implemented, the finished product is given to the client. Then, the client tests the product to see whether it is in need of additional changes. These suggestions are therefore forwarded to the authors of this thesis.

3. Project Description

Protsessor is a theses management information system that was initially developed by the students of the IT department. *Protsessor* application allows students to participate in projects by applying or potentially proposing topics, with most of them being theses for Bachelor's or Masters's degrees. Teachers on the other hand could create topics and manage which students to accept in those topics, while also having an overview of statistics if the teacher is given access to those.

The necessity of the development of an application like *Protsessor* comes from the fact of how the currently implemented thesis selection process works in *TalTech*. Although usage of things like *Excel* provides you with a working solution, improving the process of thesis selection is crucial, to allow for more efficient time management of students and teachers, while also providing more information to students to make better and more informed decisions on what thesis topic to select.

The steps that our team has decided to take in order to improve the project are as follows:

- UI improvements
- Overhaul of existing permission system implementation
- Implementation of roles
- Implementation of *Liquibase* to track, manage and apply database schema changes
- Implementation of smaller features
- Changes to the project technology stack
- Analysis of *Camunda*
- Gathering of feedback
- Bug fixes

3.1 Workflow

Due to the fact that our initial plans related to the timeframe that we would be working on this project had to undergo changes, our workflow has undergone some changes as well. Initially, we were planning on finishing the project in a single semester, with that in mind in our first semester, we had established and went through a lot of communication between the team members as well as our client and supervisor to ensure the productive work of each member of the team as well as a clear goal of what needed to be achieved

in the project. In order to manage what work should have been done, our team chose Agile software development methodology, or to be more precise, Scrum methodology, which relies on simultaneous and incremental work of all team members, and is divided into two-week sprints. Most of the existing sprint issues were documented in milestones. During the sprints, team members could take any unfinished issues and work on them, or add new ones to an existing list of issues that needed to be worked on. The decision-making process of which issues had to take priority was based on how important was the issue (if it was a very needed feature or a bug), how hard was it to implement or fix it, and how knowledgeable the specific team member was in regards to the specific issue. In our second semester of work, the amount of client and supervisor communication was significantly lower, due to us having done a major portion of the discussed features and changes. Along with the decrease in client and supervisor communication we as a team changed our approach to work priority. The foreground of our work this semester was bug fixing with the addition of small changes based on provided feedback, with only a few major features being developed only after fixing some of the core problems.

3.2 Communication Channels

One of the most important aspects of successful teamwork is communication, which provides an understanding of the current state of things in the project. Based on our previous experiences in other projects, as well as the work experience of some of the team members, we knew that we needed to establish proper team communication channels, as well as client and supervisor communication.

3.2.1 Team Communication

For team-related communication, our team has decided to use *Discord* as a communication platform, due to the familiarity with it and the ease of use that it provided, as well as the fact that it was free. While going through different stages of the project, the way we handled our team communication definitely went through some changes. Initially, we had team meetings that were essentially lacking in clarity and were missing sets of goals and objectives, however with time we managed to resolve those issues. One of the first changes in communication was related to the removal of an excessive amount of team meetings. In addition to that, with time, we managed to establish different areas of expertise in regard to the project, which then lead to much more efficient communication, due to each one of our team members knowing who he should communicate with within our team in case he is having a problem in the certain part of the project instead of attempting to resolve the issue yourself, leading to inefficient time waste.

3.2.2 Supervisor Communication

Our supervisor meetings were done in Microsoft Teams because it was one of the default communication platforms that was used by teachers in *TalTech*. The goal of the meetings with our supervisor was to establish the main direction of our Bachelor's thesis, as well as discuss changes, their implementation, and other important things such as gaining knowledge of the project based on the information that was known to our supervisor from the work that was done by the previous development team.

3.2.3 Client Communication

Along with our supervisor meetings, our client meetings were also done in Microsoft Teams. Among all of the meetings, the most important meetings were with our client. In those meetings, we were establishing potential goals and major features that our application was in need of, while also discussing if that is achievable in the time frame of our work and why it is or is not.

3.3 Work Distribution

Initially, even though it is recommended to distribute different parts of work to different people, we decided not to, due to the fact that before trying to specialize our team members in different areas of the project we needed to broaden our fundamental understanding of the project. At first, we encountered many issues that required the full attention of every team member. Those initial struggles were related to poorly made documentation that did not clearly state how to set up a locally running project environment. Most of the setup issues in the documentation were related to it not having any descriptive examples of how certain things needed to be set up in *Microsoft Azure AD* for this specific project. In addition to that, there were no proper examples showing us how an unknown platform like *Camunda* is integrated with the existing project database. After overcoming the initial struggles that our team encountered with the project while familiarizing ourselves with it, we attempted to divide different work responsibilities among each team member, the general distribution of responsibilities was as follows:

- Aleksandr Rudoi was our team leader, being responsible for the management of certain project-related aspects, along with him being the main architect of the project, in regards to the implementation and realization of most of the new backend and database-related features, while also working on some *DevOps* aspects such as *CI/CD*.

- Nikita Birjukovs was doing the work of a full stack developer, working on implementing changes and adding new features both in the backend and frontend, while also working on redesigning the frontend.
- Dmitri Voronoi was mostly working on frontend, developing new pages and integrating new functionalities from the backend to frontend, while also doing the work of a *QA* engineer, in regards to finding and resolving existing bugs.

Despite the distribution of work in our team, there were no strict rules in place that would discourage our team members from taking tasks outside the scope of their responsibilities inside the project. Due to such factors, in case of need, we were always ready to help our team members with an issue of any kind that might occur or work on implementing a feature that we personally might have found to have a better understanding of, than other team members. Further down the line, such decisions were of great benefit to us, because not only did they make us improve in most of the aspects of the project such as project architecture understanding, but also improve our future problem-solving.

4. Project Design

Protsessor project consists of three major parts: backend application which is built using *Spring Boot* and *Camunda*, frontend part which is made using *Vue.js* with the addition of *Bootstrap* and *Element Plus* for styling, and the *DevOps* part which includes *PostgreSQL* database management system, Docker and CI/CD. Communication between the frontend and backend happens with the help of the *OpenAPI* interface. Most of the architectural and technological decisions were made by the previous team, however, it also includes some of the choices made by our team during further development.

4.1 Project Architecture

This project has three environments - production environment, development environment, and local environment. All of these environments consist of three docker containers - one container runs the frontend application, the second container runs the backend application, and the third container is running the database. These containers communicate with each other via the docker network bridge. Production and development environments are running on a single server, the local environment can be set up on the developer's local machine. Despite the fact that production and development environments are located on the same server, they act independently from each other. The endpoints of these environments can be reached only via the *cs.ttu.ee* proxy or via the *SSH* connection to this server. The purpose of the local environment is to further develop the application, it has some different settings in order to simplify the process of developing and validating made changes. The purpose of the development environment is to validate that the newly introduced changes are working in the environment that is close to production. The production environment is being served to the customers [4].

4.2 Backend

The backend is a part of the application where most of the data and operating syntax are stored. The backend part is not meant to be accessed by the user.

4.2.1 Gradle

Gradle is an open-source tool that is used for software build automation. With the help of the build script in the form of a file with predefined tasks, dependencies, plugins, and

other configurations *Gradle* is capable of building almost any project [5]. This tool was introduced to the project by the previous team. The alternative for *Gradle* is *Maven*. It is very hard to determine, which tool as each has its pros and cons. However, *Gradle* is a more customizable and powerful tool than *Maven*, and thus our team is positive that this tool is a better choice for the project.

4.2.2 Camunda

Camunda - the universal process orchestrator is an open-source platform that uses a workflow engine and decision engine to automate business processes. Organizations can use it to automate workflow and decision processes, freeing up time for employees to focus on more thoughtful projects [6]. This tool was introduced to the project by the previous team and its analysis will be addressed further in the document.

4.2.3 Spring Data JPA

JPA provides a way to match entries from the database against the object-oriented model. It can not be used alone and requires an implementation like *Hibernate*. *Spring Data JPA* is a level of abstraction that simplifies the usage of the *JPA* implementations [7]. This tool was introduced to the project by the previous team. There are two alternatives - *Spring Data JDBC* also known as *JDBC Template* and *Spring Data R2DBC*. *Spring Data R2DBC* is very promising but is still under development with lots of features missing so it is not a good alternative yet. *JDBC Template* on the other hand has more control over the *SQL* queries and is more flexible overall. It is considered a common practice to have both *JDBC Template* and *Spring Data JPA* for usages where required. Our team is positive about the previous team's choice of using the *Spring Data JPA*, however, we also introduced the *JDBC Template*.

4.2.4 JDBC Template

JDBC is an interface that defines the way of connecting to the database as well as provides means to query and update entries from the database. *JDBC Template* is the central class in *JDBC API* which simplifies its usage [8]. This tool was introduced to the project by our team and the reasoning will be addressed further in the document.

4.2.5 Spring Security

Spring Security is a highly customizable authentication and access-control framework. It is considered to be the standard for securing *Spring-based* applications. Its main focus is on providing authorization and authentication to *Java* applications [9]. This framework was introduced by the previous team. There are multiple alternatives, among which are *Apache Shiro*, *JAAS*, and *OACC*. *OACC* is overall not that popular and seems to be not supported anymore as the last commit was made in 2018. *JAAS* can be well integrated with the *Spring Security*, however, it is more complex and does not have that massive support, so our team does not see it as a good replacement. *Apache Shiro* is the closest alternative to the *Spring Security* being easier to understand and configure. Both *Apache Shiro* and *Spring Security* is considered great choice by our team while favoring the *Spring Security* a little bit more because it has a bigger community and is considered to be a standard.

4.2.6 Project Lombok

Lombok is a *Java* library that provides means to reduce the amount of boilerplate code by using its annotations. It offers various annotations that would automatically generate code that is tedious to write keeping the code clean and simple while also allowing to focus on the main functionality. This library was introduced by the previous team. Two alternatives exist - *AutoValue* and *Immutables* with their own advantages and disadvantages, however, the base difference is that *Lombok* has a lot more support than other libraries. Our team is positive about the decision of choosing *Lombok* over the others.

4.2.7 Mockito

Mockito is a mocking library, that is used for testing *Java* applications. To make testing simpler, *Mockito* allows the creation of mocked interfaces with dummy functionality to reduce the amount of complexity required to test some parts of the application. This allows focusing on testing the required parts of the application without the need to set up the complex infrastructure before and after the testing part. This library was introduced by the previous team. There are two alternatives for the *Mockito* - *EasyMock* and *JMockit*. *EasyMock* has less verbose error messages than Mockito which makes debugging overcomplicated. Other than that, all three libraries have their own advantages and disadvantages, thus making it hard to determine whether one is better than another. Due to the fact, that Mockito is a lot more popular than the alternatives, our team agrees on the decision to use this library for testing purposes, however, *JMockit* is a very good alternative.

4.2.8 Azure Active Directory

Azure Active Directory (*Azure AD*) is a cloud-based identity and access management service. It allows one to access multiple apps, websites, etc. belonging to an organization by using a single set of login credentials [10]. This service was introduced by a previous team and there is no point in addressing the alternatives because *TalTech* already has the integration with *Azure AD* thus using any other services is costly and impractical. This project uses *Microsoft Azure AD* login credentials for authentication of users in the web application, while also providing them with a set of necessary permissions.

4.2.9 Spring Boot Starter Mail

Spring Boot Starter Mail is a dependency, that contains the *JavaMail* library. This library allows sending emails through the *SMTP* straight from the *Spring Boot* application's *Java* code [11]. This tool was introduced to the project by the previous team. Our team knows no analogs to this tool in *Sping*.

4.3 Frontend

Frontend serves as a layer between the user and backend. Its purpose is to serve the data in a user-friendly way.

4.3.1 Vue.js

Vue.js also known as *Vue* is a *JavaScript* framework that is used to build modern *UIs* that follow the *SPA* principle. The best feature of *Vue* is its directives which are the *HTML* attributes that make it possible to extend *HTML* [12]. This framework was used by the previous team and there is a huge amount of analogs to it. The most standing out ones are *Angular* and *React*. All of the mentioned technologies have their own advantages and disadvantages and it is very hard to objectively decide which one is better to use as there is a lot of space to interpret the facts differently. Overall, our team sees no problem with the selected framework.

4.3.2 OpenAPI

The *OpenAPI Specification* defines a standard, language-agnostic interface to *RESTful APIs* which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic

inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. This tool was introduced by the previous team and is used by the *Camunda*, thus making it required for the project.

4.3.3 I18n

Internationalization or *i18n* for short, is the process of making the product delivered with multilanguage support. This project uses *Vue's* version of the *i18n* library which provides these means by creating a *JSON* file that has to be manually populated with text versions in different languages and is available for use throughout the project via the global variable. This tool was introduced by the previous team and has some analogs - *FormatJS* and *Polyglot*, although *i18n* stands out due to its massive collection of plugins, where one probably will find a plugin suited for his needs. Our team is positive about the decision to use *i18n* for introducing the multilanguage support.

4.3.4 Bootstrap

Bootstrap is a very powerful and extensible frontend development toolkit. It contains a huge amount of predefined *HTML* and *CSS* patterns as well as *JavaScript* extensions out of the box which allows one to use not only complex elements without the need to develop them from scratch but also utilize predefined grid and styling. This toolkit was introduced by our team and its analysis will be addressed further in the document.

4.3.5 Element Plus

Element Plus is quite a new frontend development toolkit like *Bootstrap*. It contains many different styled elements out of the box without the need to implement them from scratch. This toolkit was introduced to the project by the previous team and our team does not consider it a good decision. The analysis of *Element Plus* and the reasons why our team came to this conclusion will be addressed further in the document.

4.4 Database

The database is an organized collection of data with provided means to manipulate it easily.

4.4.1 H2

H2 is written in Java language in-memory database management system. One of its main features is that it is very fast, lightweight, open-source, and comes with a built-in browser-based console. In this project, *H2* database management system is used for testing purposes as a testing database management system to separate testing data from the actual, used in the development and live servers. A tremendous amount of other database management systems with their respective pros and cons exist which makes our team believe that usage of the *H2* database management system was a bad decision made by the previous team which led to some issues addressed further in the document. *Protsessor* project could use a more modern and powerful database management system for testing purposes and the best decision for that would be to use *PostgreSQL* due to it being used as a main database management system on the development and live servers in order to have consistency and make sure that during the execution of the tests, the application will work the same way as in any other environment on the database connectivity level which will exclude the issues that might possibly occur due to that difference.

4.4.2 PostgreSQL

PostgreSQL is a very powerful open-source *ORDBMS* that is regarded for its reliability, scalability, stability, and security. One of the best features of *PostgreSQL* is that it doesn't limit itself to the boundaries of the traditional *RDBMS* - in addition to the *RDBMS* functionality, it follows the object-related model. This allows *PostgreSQL* to have objects, classes, and inheritance in its schemas and queries [13]. This feature is not being used in the database yet, however, if such a necessity occurs, there will be no need to migrate to another database management system. This database management system was introduced by the previous team. There are a tremendous amount of other database management system alternatives present with their own pros and cons, however, *PostgreSQL* is a very powerful database management system with a huge support community, active development, free of charge, and many more advantages. Since a relational database fits well for the needs of this project, our team is positive about using *PostgreSQL* as the database management system.

4.4.3 Liquibase

Liquibase is a very handy tool in terms of dealing with databases. It allows the deployment of changes to the database, tracks those changes, informs when the database version doesn't match the supposed version, and provides means to easily roll back the changes to the

database in case anything goes wrong. The main purpose of using this tool is to ensure that the application is released with the matching database. This tool was introduced by our team and its analysis will be addressed further in the document.

4.5 DevOps

DevOps combines development and operations to increase the efficiency, speed, and security of software development and delivery compared to traditional processes [14].

4.5.1 Continuous Integration and Continuous Development

Continuous Integration and Continuous Development or *CI/CD* for short, is a method to easily and frequently deliver new versions of the applications to customers by automating the stages of the application development process. The main concepts of the *CI/CD* are continuous integration, continuous deployment, and continuous delivery. The automated stages of the app development along with continuous monitoring throughout the lifecycle of applications from integration and testing phases to delivery and deployment are usually referred to as "CI/CD pipelines" [15]. For this project, the *Gitlab CI* is used as a *CI/CD* tool. The decision to use the *Gitlab CI* tool was made by the previous team. There are a tremendous amount of different tools and clouds for the *CI/CD* management, which would require the same amount of time to evaluate each of them and make a decision on which one is better. Overall, *Gitlab CI* is an excellent decision due to it being part of the *TalTech* version of *Gitlab* thus staying in the scope of the university.

4.5.2 Docker

Docker is a software platform that allows one to build, test and deploy applications. It packages applications into standardized units named containers which contain everything the application requires to run while keeping it minimal. This makes *Docker* containers lightweight and secure. The best feature of *Docker* is that it guarantees that the application will run in any environment on any operating system [16]. This tool was introduced to the project by the previous team. Despite the fact, that *Docker* has a tremendous amount of alternatives each of which has something done better, than in the *Docker* itself, our team still thinks, that this was a good decision to use *Docker* due to the reason that *Docker* is more than enough for this project while being very popular with a huge support community.

5. Work Results and Analysis

In the process of development of *Protsessor*, our team went through multiple stages of changes in different parts of the project, that occurred in both the backend and frontend, as well as touched upon some important miscellaneous things like project documentation, analysis of *Camunda Platform* along with some changes in the structure of *CI/CD*.

5.1 Backend

There were some challenges that our team had to overcome and some decisions to make during the work on the backend. The changes we made include the addition of the new functionality, the addition of the new endpoints to every new feature, and changes to some of the already existing endpoints as well as the addition of the new technology.

5.1.1 Role implementation

The first challenge that our team faced during the addition of the role system was to look through the current implementation of the permissions in order to decide whether the current solution can be expanded to work with the roles or a new solution has to be introduced. Even though the current implementation was able to expand into a role system, it was still decided by our team to implement our own solution due to the current one being not as flexible as it should be. Previous permissions were hardcoded and assigned to a user from the code. Due to that decision, it would be almost impossible to create new roles on demand and assign new permissions to them as it would require making changes to the existing codebase. Keeping that in mind, our team decided to implement new database tables that would allow us to create and delete permissions and roles on demand, assign them to certain groups either globally or in terms of a certain project, dynamically change them if needed, etc.

Our team implemented the next tables:

- *permissions* table - stores the name of the permissions along with the automatically assigned id to it. It is getting automatically filled with all currently existing permissions with the help of the *Liquibase*.
- *roles* table - stores the name of the roles along with the automatically assigned id to it. It is automatically getting filled with these roles: *ADMIN* which has all the

existing permissions, *TEACHER* which has some permissions related to teachers, and *STUDENT* which has no permissions.

- *role_to_permission* - serves as the middle table to implement the many-to-many relation between roles and permissions tables which means that there can be many roles with the same permission list and many permissions assigned to the same role.
- *group_to_role* table - maps the roles with the groups in a many-to-many relation way which means that there can be one group with multiple roles as well as a role that is assigned to many groups.
- *local_access* table - required in order to distinguish the local role from the global role. This table stores the combination of role, group, and project id which means that the specific role was assigned to a specific group in the specific project.

For example, suppose, we have two groups *SUPERVISOR* and *IAIB_SUPERVISOR*. As the names suggest, members of these groups should be able to supervise topics, which can be achieved by giving them the *VIEW* and *SUBMIT*, however, *IAIB_SUPERVISOR* group should be able to do that only for one specific project. To make this happen, an admin should create the role, e.g. *SUPERVISE* with the previously mentioned permissions. Since the permissions are already added to the database with the help of the *Liquibase*, they will be already available to be used from the permissions table. Once, the admin will create that role, the backend will create a new entry in the role table with the newly generated id and *SUPERVISE* in the name column. After that, the new role will be mapped with the corresponding permissions via the *role_to_permission* database table which will contain a combination of the role and permission ids on each row. The next step is for the admin to assign this role to the groups. When the admin will assign the newly created role to the *SUPERVISOR* group globally, it will add a new entry in the *group_to_role* table mapping the id of the group to the id of the role. On the other hand, when the admin will assign this role to the *IAIB_SUPERVISOR* group to a specific project, it will create a new entry in the *local_access* table, which will include ids of the role, group, and project. Figure 1 shows newly added tables along with the established connections between them.

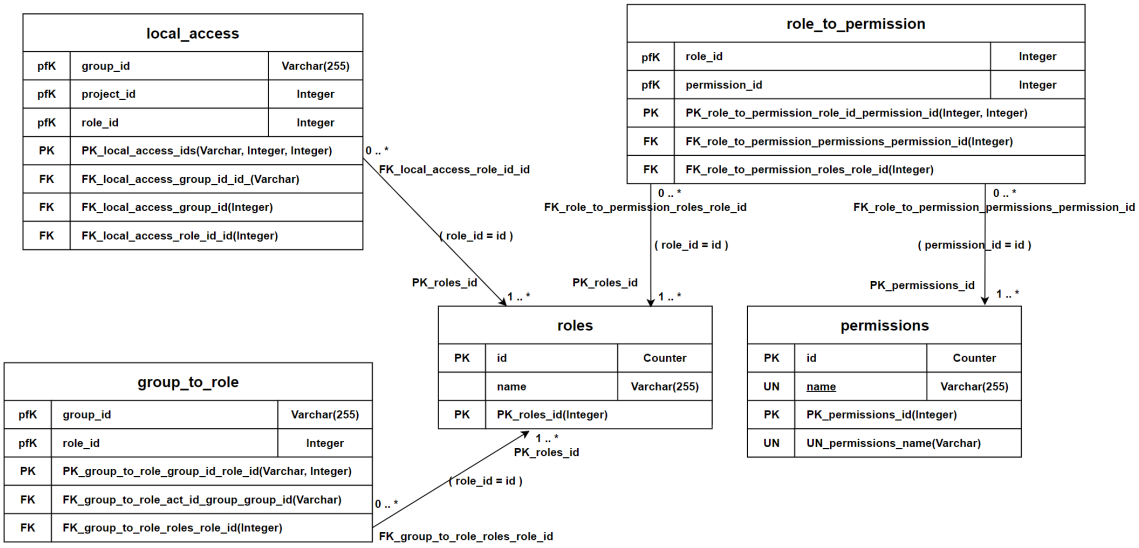


Figure 1. New database tables.

The second challenge occurred due to our decision to rework the current realization of the permissions. Because of the new database tables being added, our team faced a need of implementing new backend endpoints in order to be able to use the new functionality. Some of the functionality could have been inserted into the currently existing controller, however, it was still required to create an additional controller with its own endpoints and business flow. While implementing the endpoints, our team kept in mind all the required actions that could be performed with the newly introduced functionality in order to create everything that is required while not adding the actions that would be obsolete and thus never used by the client and users.

The solutions our team came up with have their own upsides and downsides which we were aware of while making our decisions. The biggest downside of our solution is the fact, that the created pool of roles can be applied globally and locally to many groups, but making any change to a role will affect local and global usages altogether. Due to this fact, admins should be extra cautious when modifying the existing roles as it may lead to unexpected behavior. As one of the possible solutions to that, role names can have *global* or *local* prepended to their names which will allow admins to know whether they will affect any other groups while modifying the role or not. However, the advantage of this solution is the smaller amount of tables added, because the database itself is very big due to the *Camunda* tables which makes it particularly hard to understand and make any changes to it. In addition, it avoids duplicated logic, as in that case there should be two tables with identical information for storing the different types of roles.

5.1.2 JDBC Template

One of the decisions our team made is to introduce *JDBC* usage via the *JDBC Template* as a means to query the data from the database. Currently, the only means to query the data from the database was the *Spring Data JPA*. Although both of these technologies offer the same functionality, there are some differences between them and our team decided to introduce the *JDBC Template* for a couple of reasons.

Spring Data JPA is a high level of abstraction that allows querying data from the database via the usage of annotations and *JpaRepository* which keeps the code clean, elegant, and easy to understand. However, being a high level of abstraction is also one of the downsides of the *Spring Data JPA* because it becomes quite hard to achieve something harder than the standard ways of its usage. *JDBC Template*, on the other hand, is a lower level of abstraction and as such, it requires a bigger setup and more code in order to achieve the same goals, but despite that, it gives you all the required flexibility and, in addition, is a very efficient tool. These two technologies do not conflict with each other and, as a result, may be used together depending on the situation and requirements.

One of the main reasons for introducing the *JDBC Template* for our team was the fact that one of the main features of the *Spring Data JPA* is database schema creation. Currently, this project has too many technologies that include the functionality of making changes to the database schemas: *Spring Data JPA*, *Camunda*, and *Liquibase*. Additionally, due to the poor decisions made during the creation of the tests, they became dependent on *Camunda's* and *Spring Data JPA's* creation of the database schema and, as a result, any attempt to query the data from one of the *Camunda* tables using *Spring Data JPA* will require, to annotate it with the *Entity* annotation which will in its turn, tell the *Spring Data JPA* to create that table in the database. As a result, due to the tests being dependent on *Camunda's* and *Spring Data JPA's* creation of the database schema, it will produce the exceptions, because *Spring Data JPA* and *Camunda* will both attempt to create the same table. On the other hand, *JDBC Template* is able to query complex data from multiple tables without any attempts to create those tables which was exactly what our team required.

5.2 Frontend

While working on frontend, even though the challenges our team has been met with were not as severe as in backend, we still had to go through many problems. A few of the major issues that we have gone through, existed due to poor implementation of some of the core features made by the previous team, in addition to that the frameworks selected for

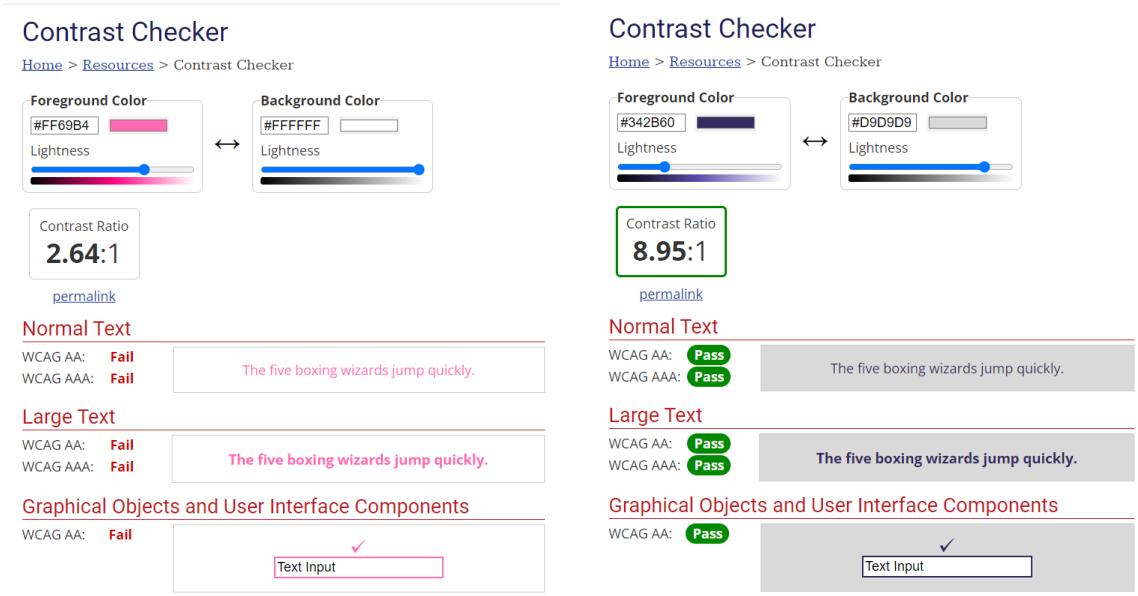
this project also had some very noticeable problems. One of the main goals of our work on frontend was to achieve a fully functional website redesign, that would also comply with certain standards such as *TalTech* style guide and "Contrast Ratio" regulations of *Web Content Accessibility Guidelines* (*WCAG*) [3]. In addition to the redesign, we also worked on adding new functionality to both old and new pages of the website, while also reworking and improving crucial features most websites need like a system of roles and permissions.

5.2.1 User Interface and User Experience

Based on the personal experience of using the web application before taking over as its developers, as well as comments of previous developers stating that *Protsessor* needs design overhaul and/or improvements we have concluded that *UI* and *UX* of the web application needs to be changed and modernized. A modern UI and UX design is a design that is meant to conform to the existing perspective of the user base of the application on how the visual experience should feel and look, which arose in response to the complexity and clutter of past designs styles, with the goal of making the user experience more intuitive and efficient. [17] There are many ways to achieve such design, with the main one being based on user feedback, however, that is subsequently one of the most expensive ways, because of the need of having the finances, time, analysts, and developers, that would be capable of going through the user feedback and deciphering how it should be implemented, due to users not always having the capability of providing feedback in a way that would make sense to the development team.

After reviewing what we had to work with, we arrived at the conclusion that the previous *UI* did not follow the *TalTech* style guide, in addition to not complying with *WCAG*. Improving on those aspects meant that we needed to look through guidelines of *TalTech* style guide as well as read through *WCAG* documentation so that we could properly address issues on existing pages and avoid said issues on future new pages, in order to improve the navigation process for normal or visually-impaired users that could be using *Protsessor*. One of the biggest issues was related to the color choices and contrast ratios of the older design. By definition, the colors are in contrast when you can observe the difference between the brightness of two colors. As an example, we will be using 2 sets of color combinations which are shown in Figure 2. Figure 2 (a) shows the contrast ratio of a supervisor's name on some of the pages, while Figure 2 (b) shows the colors of a text with a background color that is used in the newly remade version of accordion elements. In this case, the ratio of 2.64:1 shown in Figure 2 (a) is not allowed, because it is lower than the minimum acceptance criteria for *WCAG*, with the minimum value of 4.5:1, which every *TalTech* web page has to comply with, while the new color combination of Figure 2 (b) has a value of

8.95:1.



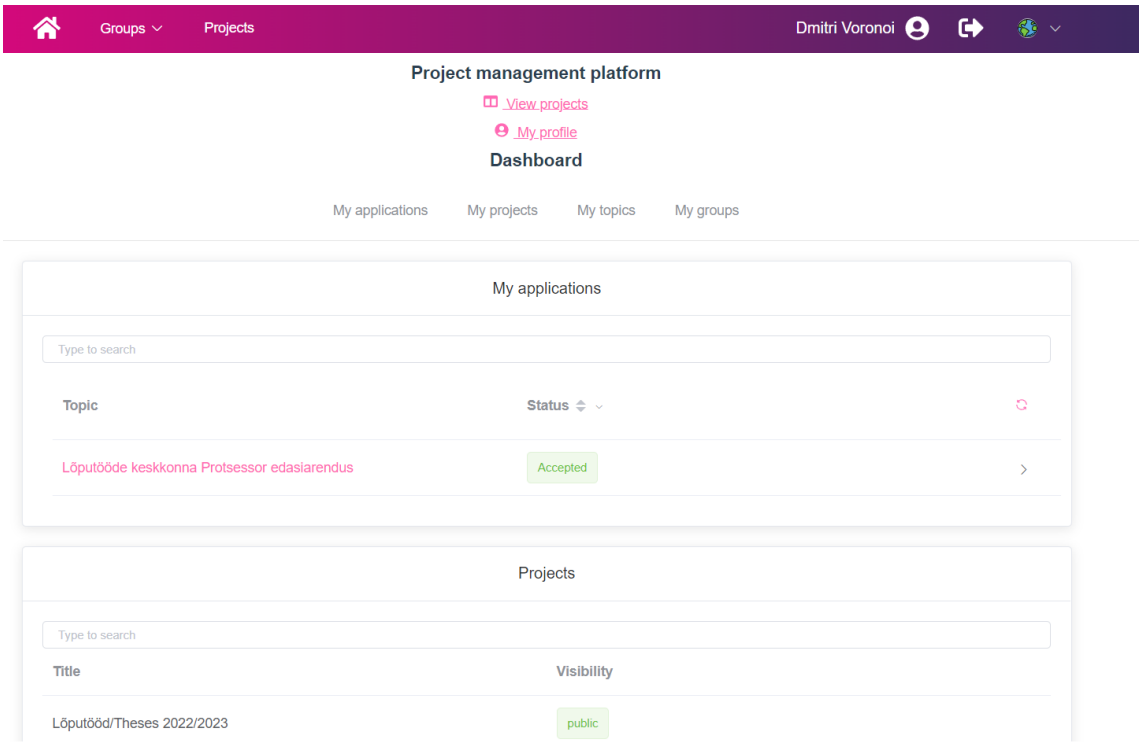
a) Contrast coefficient before.

b) Contrast coefficient after.

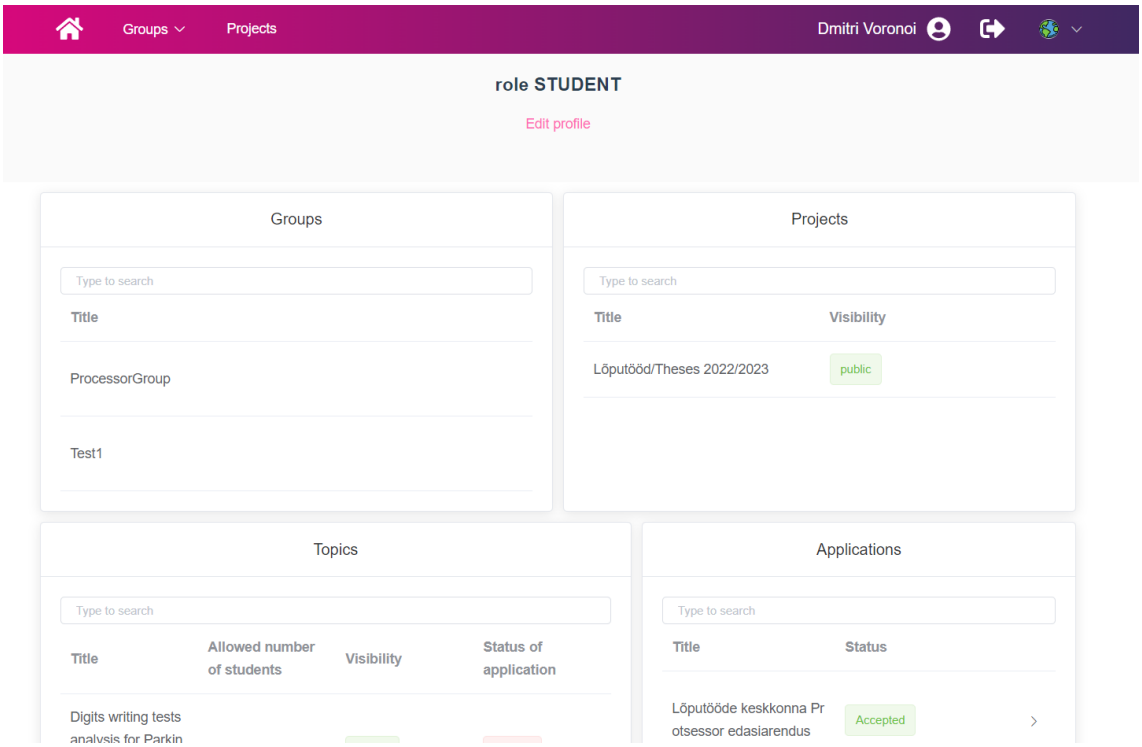
Figure 2. Contrast comparison of different designs.

Before working on implementing any of the changes in the existing design, we had to work on making a cohesive design for most if not all pages of the application. By deciding to make most of our initial designs with the use of *Figma*, we managed to massively increase our development efficiency, by avoiding the risks of having to redo a design that was already implemented in code, because a big portion of those designs was thought out and made by us in advance before implementing them.

One of the first improvements in our design was the minimization and removal of repetitive information that was very obviously present in the web application before. Previously there existed different pages that served the same purposes in two or more places, with the only difference being slight design changes, which could lead, and personally lead us to confusion back when we were normal users and not the developers of the application. The solution to this was to merge pages with repetitive information into one, in addition to that, in places where merging pages or some features was not possible, modal windows were used in order to improve the *UX*. As shown in Figure 3, Figure 3 (a) and Figure 3 (b) show 2 different pages, that have identical information, that is barely any different from one another, meanwhile, Figure 4 shows our vision of both of these pages combined, which leads to better *UX*, due to the removal of unnecessary pages.



(a) Page with duplicate information 1.



(b) Page with duplicate information 2.

Figure 3. Pages with duplicate information.

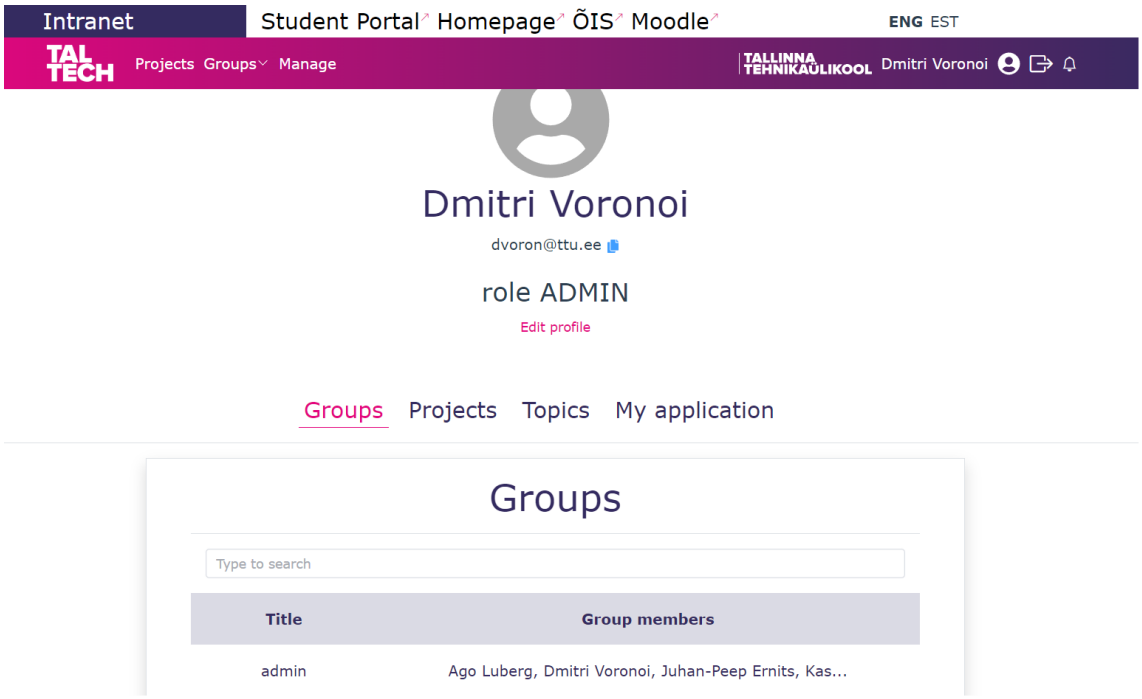


Figure 4. Page without duplicate information.

Besides mentioned changes to the user profile page, there were other pages that went through drastic changes in design, which are:

- Project selection page
- Project page
- Topic page
- Statistics page

Secondly, we improved some of the existing navigation elements while also adding new ones, so that users could have access to certain features easier and faster, without needing to go through too many pages to access those said features. Currently, when going through the project selection page, if the user only has permission to apply to topics within the project, he will not need to go through a page that would otherwise provide him with only a single button that would send him to the topic page. Additionally, users with sufficient permissions can propose topics without having to go to a specific project page and click the button there. Along with these *UI* and *UX* changes, we added multiple new pages to the application. A new group page that allows the user to have an overview of all groups that he is a part of, as well as accept or decline group invites in a better way. Global statistics page where users with certain permissions can view applications of the students divided by their curriculum across all projects. Manage page, which allows users with admin privileges to create new roles or delete existing ones, manage what permissions those roles

have, as well as add or delete groups of people that have those roles. Realization of said feature, lead to major improvements for admin users, who previously had to manually add every single permission to each separate group of users, but currently need to only add roles, which can be added multiple at a time, which subsequently improves the speed and amount of admin work that needs to be done to maintain the projects of *Protsessor* application.

5.2.2 Element Plus to Bootstrap

During the time that we have been working on this project, there were goals that needed to be achieved. While working on improving and reworking the *UI* and *UX* of the project, we encountered issues in changing the design and style of most of the previously implemented buttons, tables, colors, etc due to them being a part of the *Element Plus*. As it currently stands, *Element Plus* is a relatively new framework, that is not yet refined properly, which leads to it having some problems in addition to lacking functionality that is present in bigger frameworks like *Bootstrap*. Initially, after finding out that the project was running on an early beta version of a framework, it was decided that updating to a newer version would be better, due to potentially many major improvements that could have happened from version "1.1.0-beta.12" to version "2.2.21". In addition, most of the documentation for the earlier version of the framework was not accessible anymore through the web page of the framework and had to be manually downloaded from other places. After migrating the versions, we found out that the changes in the libraries of the newer version were not compatible with the older one, that issue lead to most of the styling being broken on the application. Most of the issues that have occurred to the web application were related to, the footer and navigation bar losing all of the stylings they previously had, along with the appearance of some visual "artifacts" or in other words glitches, additionally, text font, text size, element positioning and stylizing of tables was lost. In addition, most of the help on forums that we could find was in the Chinese language which further added difficulty level to using this framework. Following the fact that there were so many problems with *Element Plus* and our attempts at resolving them were unsuccessful, it was decided that we would be switching from *Element Plus* to *Bootstrap*. This decision made us take a step back and make us start removing and replacing the framework, due to our inability to work with it further.

5.3 Statistics

In the course of our work, one of the major aspects was improving statistics that are present in the web application. Initially, the project only had project-specific statistics.

These statistics could provide users that had sufficient permissions with information about supervisors and their topics as well as the status of students in the project. However, based on the perception of our team as well as feedback provided by the client and mentor, it needed changes, improvements, and multiple additions in terms of functionality and information that it provided.

At first, our team worked on improving the UI and UX design of the project statistics. After successfully implementing new web page designs, our focus shifted towards changing the existing information that is shown to the users. In the original version, users could not see in-depth information about supervisors in the supervisors tab, due to it only containing numerical information like the number of submitted or supervised topics. The only way to see that information was to go through the member's tab that contained information about both the students and supervisors, which subsequently limited the amount of information that could have been shown, about one or the other, due to the need for them to have identical information fields being present. In our solution, we have added an extra tab that shows what permissions users have in this specific project, renamed the existing tabs, as well as added information so that those tabs provided different and independent information about supervisors and their topics, as well as the status of students in the project and what topics they have applied to. Additionally, we added sorting and filtering functionality, and search functionality by curriculum and user permissions.

Afterward, it was deemed that improvements made in project statistics were not sufficient enough for users, thus why we also worked on designing and adding global statistics. Global statistics provide users with permissions with an informative overview of all students of a specific curriculum that are present in the web application. Information contained about each student of the curriculum contains things such as the statuses of students in all existing projects, in addition to providing information about what topic was selected for those specific projects.

5.4 Documentation

Another huge challenge that our team faced during this project was documentation. Although the project had existing documentation, after further review of the contents in the documentation, it has been established that the documentation is not sufficient and is missing very important aspects, due to which it would not be possible to work on the project without resolving the initial problems with the documentation. For a project of such scale, in a perfect scenario, there would be a need of having everything documented. However, that is a very unlikely task, that is often neglected even by some big businesses. Documentation for this project should provide all the necessary information about the

database and its tables, some of the less popular technologies and how they are used in the project, as well as all the required steps in setting up the environments for local development.

As it was initially established by the existing bits and pieces of information in the documentation, all we needed was to make a basic *Microsoft Azure AD* account and also create a basic project in it to make everything work. However, soon after creating the account and going through the necessary setup, it has been found out that this was not enough. By following the provided steps, we could not initially run the backend application, due to there being no notice of needing to add additional things like "environmental variables" in the project run configurations, additionally, there were issues with receiving information from *Microsoft Azure AD* to the database, due to the lack of documentation about the database and its tables. This caused a lot of issues that could have been avoided through the necessary documentation, the issues include things such as the need to manually add some data from *Azure AD* to the database tables. Moreover, there were problems with versions of frameworks being automatically updated, and there is no documentation about the versions which were initially used, causing many issues and visual problems in frontend and its development.

In the end, after researching and resolving the issues with the initial local work environment setup, existing documentation underwent thorough changes, so that future development would not be halted for dozens of work hours, due to missing crucial information. Changes in the documentation include things such as:

- Addition of required *SQL* statements for local setup
- Addition of some code examples
- Addition of a *Microsoft Azure AD* account creation and setup guide, due to requiring non-default settings in there
- Documentation of database tables that were created by the previous team, as well as our team
- Moving all the existing documentation from different web applications to *GitLab*

In order to improve and keep on improving the project's health, our team highly recommends for future developers to keep updating the existing documentation that is present in *GitLab*, in order to keep all the crucial project-related information in the same place, instead of spreading it out across multiple different web applications. Furthermore, it is recommended to turn more focus towards documenting *Camunda* related project implementations, due to how difficult it was for our team, and what issues there may be if more functionality would be added.

An updated version of the documentation can be found on the *GitLab* wiki page of the project, which has all of the changes mentioned, along with other smaller changes that have not been brought up here. [18]

5.5 Database

As was already mentioned before, we had to make a lot of changes to the existing database in order to implement the client's requirements. The database is growing bigger for this project and it becomes harder to maintain it, understand and make any changes to it. Overall, there currently are 74 tables in total out of which 47[19] are made and used by *Camunda Platform*, 2 are being created by the *Liquibase* for its needs, 1 is generated by *Spring Data JPA* and 24 were made by the developers[20]. Out of these 24 tables, 5 were made by our team and 19 tables were introduced by the previous team. Due to that reason, our team had to take advantage of several technologies and tools as well as make a few decisions.

5.5.1 Liquibase

The usage of the *Liquibase* allows raising the database schema from scratch and validating the current database schema to make sure it is intact with the expected one. The previous team has already mentioned the requirement for *Liquibase* usage in their document before. Our team considered other technologies that could be used for the same purposes, like *Flyway*, but decided to use *Liquibase* for two main reasons: *Liquibase* dependency was already introduced into the project by the previous team and *Liquibase* allows the use of scripts in *XML* format for schema creation. *XML-type* scripts are able to be executed for any database management system which makes them particularly good in case the client would feel the requirement of migrating to any other database management system. Also, there are many more advantages that *Liquibase* has over the *Flyway*, but they are not as important for the current project. While implementing the *Liquibase* and making it work with the current database our team faced quite some challenges.

The first challenge that our team immediately faced was due to the already existing database schema with a lot of valuable data inside. *Liquibase* is a very easy-to-use tool when it is being used from the very beginning of the project, however, later on, it gets a lot harder to introduce the *Liquibase* into the project with some database setup. The main cause of this is that there is no *SQL* or any other types of scripts that could be utilized in the *Liquibase* changelog. However, during the research of the *Liquibase* capabilities, our team found one of the functions that allow making a copy of the existing database in the desired format in

order to be used for the *Liquibase* changelog. This allowed us to generate over 3000 lines of code for database creation which then was optimized and reduced to a little over 2300 lines by hand. Even though it took some time to rework the generated file, this saved a lot of time that would have been required to manually write down a script that would create tables including those that are being created by the *Camunda*.

The second challenge that our team had to solve was in the *Spring Data JPA*. The problem is that *Liquibase* assumes that the tables it is required to create are definitely not present in the database schema and thus results in exceptions when those tables are already created by any other means. In order to solve that, we had to change the configuration for the *Spring Data JPA*, so that it would not create the database, but validate the schema instead. At this point, it was unknown to our team that this was possible to achieve using the preconditions which our team will introduce later in the project. In addition, due to that decision, our team also had to change the configuration file for the tests, because tests rely on database creation, but *Liquibase* is not suitable to be used there due to the *H2* database management system specifics. The *H2* database management system is an in-memory database due to which it forgets the schema *Liquibase* creates for it as soon, as its finishes.

The third and hardest challenge that our team faced was introducing the changes to the development and production servers. Due to the fact that they already have created schemas along with valuable information that should not be lost, it became a tough and tedious challenge to overcome. Due to the information in the table, our team couldn't just drop the schema and recreate it from scratch using the *Liquibase*, and due to the already existing schema, *Liquibase* would not execute successfully. In order to migrate the database with the usage of the *Liquibase*, our team had to go through several trials and errors until we reached the final solution. The first and most logical solution that we came up with was to back up the stored data in the database, then drop the schema with all the tables, recreate them using the *Liquibase*, and then restore the data saved from the previous step. However, it appeared to be not possible due to the table restrictions that *Camunda* automatically creates. After generating the data backup, the backup file contained warnings about some of the *Camunda* tables stating that they have some circular dependencies which may lead to errors. This appeared to be the case and as a result, any attempt to restore data from that file not only caused some errors and missing data in the database but also was breaking the *Camunda* causing it to stop working and produce exceptions about the missing resources. After some more attempts, it became clear that this is not possible to achieve without huge investments into fixing the backup script with over 50000 lines. After some research, our team found out that *Liquibase* documentation contains multiple ways how to introduce it to the already existing schema [21]. Taking that in mind, our team chose the so-called "We are going to use Liquibase starting...NOW!" approach which essentially means that we left

only the code that would introduce changes in addition to the existing database schema.

The advantage of our decisions about the way of introducing the *Liquibase* into the project is that this way is easier and cheaper in terms of spent time. However, it also comes with a disadvantage. In case the database schema will be lost or becomes malformed and will require a redeployment, it will be first required to restore the database at the point before the *Liquibase* introduction, and only after that should be completed with the help of the *Liquibase*, which is achieved by running the backend application. This is due to the fact, that part of the schema existed before and thus could not be used during the usage of the *Liquibase* as any attempt to create schema parts that already exist would result in errors. Keeping that issue in mind, our team decided to address it by enhancing the *Liquibase* with the preconditions. Preconditions are a set of checks that could be defined by the developer. They are very useful to further validate that database contains necessary entries, tables, restrictions, etc but the best part about these preconditions is that developers can decide how *Liquibase* should act in the case, these checks fail or there will occur an exception during the execution. Using those preconditions along with the ability to control *Liquibase* behavior on different results allows to mitigate the previously mentioned issues by marking the code as executed in case part of the schema already exists while the other part of the schema is still missing.

5.5.2 Database Backups

It quickly became clear that our team will have to do many manipulations with the database and introduce some changes that were capable of causing harm to the existing database and data stored inside. Any mistake could cause the loss of valuable data which is not acceptable. The knowledge of our team about database backups was quite limited. After some research, it appeared, that in order to introduce an automated database backup system, it is required to have knowledge about the operating system in order to schedule the running of the database backup scripts in a certain interval of time. Due to our limited knowledge of *Linux*-based operating systems and the fact that for a long time, we had no access to the server that was hosting the *Protsessor*, we had to come up with another solution. During further research, our team found one dockerized library whose purpose is in backing up the dockerized *PostgreSQL* database management systems [22]. This solution is very convenient because it works independently from the database, it is dockerized and thus is able to run in any environment on any operating system and it is easily customizable without any need to have access to the server.

5.6 Analysis of Camunda

As one of the important aspects of our work, we had to analyze the open workflow and decision automation platform *Camunda*. We had to learn the pros and cons of it, how beneficial it would be in our project, determine our use of it in regards to this project as well as understand how it is currently implemented.

Based on the conducted research of the *Camunda* Platform and the wishes of our client and supervisor as well as the information that we managed to gather about this project, it was determined in the early stages of our work, that *Camunda* as a platform indeed brings a lot of value to this project. As we have learned in our research, *Camunda*, is best used in large-scale enterprise-level business projects [23]. Of all the many different possibilities that it provides, we could divide these features into 3 large categories:

- Design and Connect
- Automate and Monitor
- Improve and Optimize

Design and Connection could provide developers with ways of modeling your business process diagrams with the use of *BPMN* and *DMN*, giving developers access to integration frameworks, that would provide them with easier integration of communication with enterprise applications and protocols. Automation and Monitoring would give access to features like workflow and decision engines, which allows for faster, easier, and better performance when automating processes on big scales, while also providing a visual framework with necessary tools to make or manage changes in those processes. All of this comes with some other things like task lists, that provide even more control over tasks that require some sort of human control or verification. Improvement and Optimization give us information and the ability to gauge the health of certain business processes, with things like data diagrams, while also allowing to work on continuously improving those said processes by keeping track of version changes in processes that are made throughout the lifetime of a project [24].

However, as we went deeper into the existing implementation of *Camunda*, we have been met with multiple problems, that were not initially visible without having immense familiarity with all parts of the project. Most of those issues were related to misuse and improper implementation of core aspects of *Camunda* in the project, due to the lack of knowledge and experience in working and using such a big framework. From what is known to us, *Camunda* was supposed to be used as a task automation platform to satisfy the wishes of the client and supervisor to automate some tasks and processes. Unfortunately, it

was not done well and brought more issues and difficulties to future project development, than if the project would have been developed without *Camunda* in the project.

Regarding the issues in the current implementation of *Camunda* in our project, currently understanding the basic code connections between classes that use any *Camunda* features is too hard, with the reason behind it being that it makes it so there does not need to be any connection between classes for it to work and use functions and methods of those different classes. This lack of connection between classes leads to the existence of hard-to-trace bugs, which persisted through very long periods of time, due to the lack of clarity on how to test and fix such things. Moreover, the processes that were made with the help of *Camunda* were made incorrectly. *Camunda* is meant to allow developers to track changes on each and every step of multiple big multi-step processes, however, most of the currently implemented processes are one-step processes that automatically start and finish at the same time, which defeats the purpose of using *Camunda* in the first place. Along with that, some of the older existing features of the application use the same multi-step process, even though there should be different multi-step processes done for those features. Additionally, the current implementation of the database together with *Azure AD* and *Camunda* has future scaling issues. Developers or people with access to the production server of the web application would need to manually access the database and add information to some of the *Camunda* related database tables, otherwise, students from different departments and curriculums would not be able to access our web application.

With all of the information about *Camunda* being available to us, even though there are many issues that came with it, our team concluded that *Camunda* needs to stay in the project. The reasoning behind this decision was based on multiple factors, first of all, our analysis concluded that *Camunda* was necessary for the future of the project and the development of some of its future features, secondly, even though the current implementation of it had a lot of issues, we believed that we can make better use of our work time by working on improving other aspects of the project, instead of starting the project from scratch without *Camunda*.

In conclusion, after the full analysis of *Camunda* as well as its state in our project, our team still agrees that *Camunda* needs to stay, due to the future benefits of being able to scale the project in the future. However, our team wants to highlight an immense need for knowledge and skill in developing architecturally stable applications as well as the need for knowledge in regards to *Camunda* that will very likely be required for future developers who would wish to tackle on the currently existing implementation problems of *Camunda*.

5.7 Bug Fixing

When most of our core features were implemented, our final degree of work went towards implementing smaller features that were missing from the initial version of the application for no apparent reason. Even though those features were not as hard to implement as others, they meant a lot to the users of the *Protsessor* web application, due to resolving commonly existing issues users had to go through when using the application. Adding an improved notification tab in the navigation bar, added the ability to delete students from topics, making it possible for additional supervisors and not just the initial one to accept or decline applications to their topic, remade the task proposal page, existing topics can now be deleted, users with certain permissions can now view archived projects, added a new client field on the topic page that shows who is the client of the topic. Moreover, there was a multitude of bug fixes that ranged from simple spelling errors, leading up to potential security and data leaks that were previously present in the application. Regarding the common fixes, on the project page we changed the names of accordions to be more reflective of what information is stored in them, we fixed an issue that stopped users from seeing all topics that were made or supervised by them, or they were a part of, fixed users being able to see the information of other users profile pages, fixed users being capable of applying to the same topic multiple times, while still awaiting to be accepted or declined, or while already being accepted and many more bugs. Finally, we fixed one of the biggest potential security threats that were previously present in the web application. In the past, after the user has logged out, most if not all of the user's permissions and other data that is normally stored in the web browser were still present, issue of this kind could lead to making changes in projects or topics or deleting or removing users from said projects or topics by unauthorized users while also allowing them to receive some of the potentially sensitive data that belongs to the original user. In order to fix that issue, we had to change the original logic of information storing on all pages and components of our frontend application that previously existed. Figure 5 shows us a logged-in user with certain permissions that allows him to add a project in *Protsessor*, however as shown in Figure 6 when the user is logged out, if the browser is not closed and reopened, a logged-out user could still click on a "Add project" button and successfully add a project.

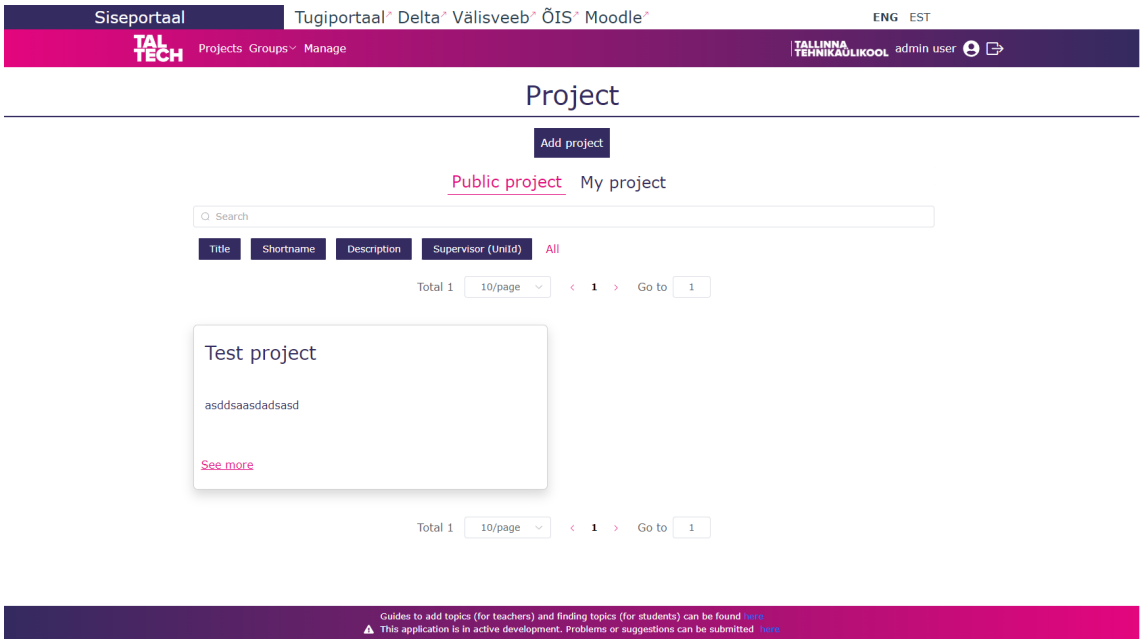


Figure 5. "Add project" button visibility for admin user.

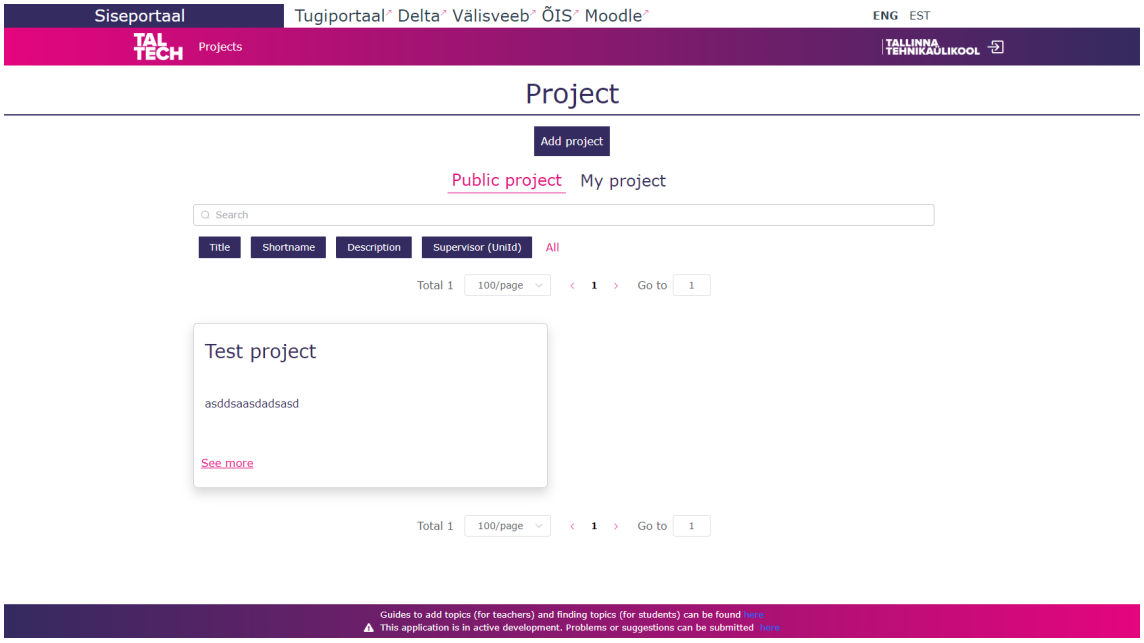


Figure 6. "Add project" button still visible after log out.

6. Validation

During this project, our team had several ways of acquiring feedback and thus validating our work. Initially, when we managed to set up the project and make ourselves familiar with it the first source of feedback that we took a look into was the feedback in Teams that our mentor made visible to us. It consists of two forms: one for the students and one for the teachers with questions about their overall user experience regarding the usage of the *Protsessor* web application. Among these questions, respondents had an opportunity to express their complaints as well as the features that in their opinion would be beneficial for the web application. Our team addressed some of the responses and one particular response shaped the goals that we have set for this thesis - the complaint about the *UI* that needed to be reworked in order to become more understandable and easier to follow. Another example of a complaint from that source that was addressed is that the status of the project was not clear enough which ultimately led to a missing opportunity to find the candidate for the topic due to it being in a "hidden" state which made it not visible for the students.

The next source of the feedback was our mentor and client to whom we presented our work. Not only did we have constant meetings to present what was done and receive further instructions and feedback on the current progress, but we also would communicate between those meetings in case something was unclear to the team or there already was something done that could be tested in order to receive the feedback as soon as possible.

Additionally, we received feedback from Erki Eessaar during one of the Demo days. After the Demo day, we contacted him and asked for further clarification which he kindly provided to our team. His feedback not only contained information regarding some of the bugs that we did not manage to find ourselves before due to having different goals that we worked towards but also challenged some of the use case scenarios that the previous team implemented. Our team summarized the feedback that Erki Eessaar provided to us and split it based on the issue's severity and the difficulty keeping in mind that we were limited by the time. One of the examples would be the lack of the possibility for the topic creator to remove the already assigned person from the topic which would cause a necessity to create a duplicate of that topic. Our team managed to address and fix the most severe issues which should drastically improve the user experience on the *Protsessor* web application.

7. Work Accomplishments

During the further development of the *Protsessor* web application, our team had to overcome many challenges and make some very important decisions along the way. In the scope of this Bachelor's thesis, our team not only managed to accomplish most of the tasks from our task proposal but even went beyond that scope to make the *Protsessor* web application a lot more valuable and helpful for its users in the upcoming years. Our work touches different aspects of the web application varying from the user interface making it drastically more pleasant and easy to follow to different bug fixes that potentially caused huge frustration and additional work required to find a way around it. Our team managed to come up with the initial setup documentation guide for future teams that would work on this web application which will make their first steps drastically easier and, as such, will make future development much more pleasant and faster when compared to dozens of hours and many work days that have been spent by us while struggling through and trying to make the project work locally.

7.1 What Was Done

The following list will provide insight into what tasks were achieved during our work on the *Protsessor* web application:

- Redesign of the *UI* in order to make it comply with the *TalTech* style guide which in turn makes the style of the web application more uniform and in line with the other web applications that belong to the university. Additionally, our team followed the *WCAG 2.0* recommendations which based on the background and font color combination produce a constant that describes how hard it is for visually impaired users to distinguish the text on the specified background. *TalTech* style guide also references the *WCAG 2.0* guidelines setting a minimum required bar for the constant it produces which all university websites should follow.
- *UX* improvement which includes repositioning the web elements like buttons to a different place for the users to find them more easily, as well as reducing the number of clicks that user has to make in order to achieve some goals on the web application making that experience more smooth and pleasant.
- Implementation of the role system which reduces the number of actions that administrators of the web application have to do in order to change the set of permissions of certain users and groups. Those changes should reduce the burden administrators

have to go through during their administrative routine.

- Division of the roles to global and local making it possible to give any type of permission to a user that would apply to all existing projects and the further ones to come.
- Full analysis of the *Camunda* framework which includes such topics as what is *Camunda*, how it should be used, how it is being used currently in the project, whether should it be removed or built upon, etc.
- Addition of the local and global statistics divided by the student's curriculum for the teachers to have a better insight into how their group is doing. This statistic includes the number of applications, the current state of each application, the mentor and topic's name, the date when the user applied to this topic, etc.
- Addition of the initial installation guide including prerequisites and required steps to set up the local environment correctly so the project could be run locally which would definitely save a lot of time for the next team at the beginning of the project.
- Possibility to add clients to the topics.
- Possibility to archive topics.
- Renaming some of the elements on the web application for a clearer understanding of what they are about.
- Removed the possibility for the user to be able to view data and execute some actions on the web application in a logged-out state which could have led to potential security breaches.
- Multitude of bug fixes which include: inability to remove students from the topics, ability to see personal information of another user from a profile page, ability to apply multiple times to the same topic, limitation on the amount of the visible topics on the project page, inability to see archived projects under the "My projects" tab, etc.
- Introduction of the new technologies that our team felt to be required in this project: *Liquibase* and *JDBC Template*.
- Replacement of *Element Plus* to *Bootstrap* technology.
- Decreased technical debt of the project, in regards to documentation, *DevOps*, and in addition to that removed some of the existing code smells.
- Successfully integrated usage of curriculum information from *TalTech* provided by *Microsoft Azure AD*. This functionality is considered to be new, due to being initially added for usage for the previous development team of the project. This integration can serve as a proof of concept, by showing that curriculum information is usable and could provide a far greater integration level with the *TalTech* infrastructure to other *TalTech* applications that are yet to use this feature.

7.2 Task proposal

The following list will provide you with tasks accomplished within our Task proposal:

- *UI* improvements and/or redesign on all pages in compliance with the *TalTelch* style guide
- *UX* improvements, with the rework of user flow on most pages
- Implementation of role functionality
- Addition of global permissions in addition to existing local permissions
- Simplification of the amount of work that an admin is required to do
- Overall analysis of *Camunda*, with all of the underlying positive and negative possibilities it provides, along with the current implementation of it in the project.
- Improvement of the shown information on the statistics page for projects

8. Comments

Not everything could go perfectly during this process of development - some solutions have drawbacks, and sometimes reaching perfection is possible, but not considered a good decision due to the huge amount of time/effort/resources that are required to be invested while the final result is not worth it and a minor solution is more preferable. The same could be said about the work that was done by our team.

8.1 Liquibase Drawbacks

As was already mentioned previously - *Liquibase* is one of the technologies that our team had introduced to this project based on our personal feeling of necessity to make this action as well as the previous team's thoughts. One of the major drawbacks of this solution is that *Liquibase* is now bound to set up the database schema and due to its nature and the solutions that *SpringBoot* offers to work with it - *Liquibase* is also required to set up the *Camundas* database even though *Camunda* also manages this on its own. The reason for this is that *Liquibase* is being executed during the runtime before any other technology that manipulates the database and, as a result, will run into errors trying to create tables, that are dependent on the *Camundas* tables which are not yet existing. Our team at this point is not yet fully aware of what *Camunda* does in that case, but at least it is working without any errors.

8.2 JDBC Template Drawbacks

The introduction of the *JDBC Template* also has some drawbacks that we had to consider when introducing it. The main problem with this decision is that technology with similar functionality already exists - *Spring Data JPA*. Additionally, the introduction of this technology leaves the future of *Spring Data JPA* in an undecided state as the introduction of the *Liquibase* took away a good portion of the *JPA* functionality - database schema creation and now *JDBC Template* takes away a database manipulation functionality from it. A good decision would be to remove *Spring Data JPA* in the future to lessen the amount of the technologies used by the project as well as due to other technologies overtaking its functionality. That, however, will require a good portion of work and effort to make it happen as well as will require changing the database management system used for testing purposes as the in-memory *H2* database schema is currently being raised by the *Spring Data JPA* and it cannot be achieved by the *Liquibase*.

8.3 Bootstrap Drawbacks

Bootstrap is a very great technology that helps to save a lot of time by introducing lots of premade components so developers will not need to create them themselves, thus the major drawbacks of using it are not in *Bootstrap* itself, but, rather, in the situation of the project it was introduced to. Due to the necessity of removing the previous technology used for the same purpose - *Element Plus*, we had to spend quite some time looking for each place where it is used and replacing it with *Bootstraps* alternatives for a work which is hardly distinguished before and after it is done. In addition to that, we did not manage to fully get rid of *Element Plus* due to it having some of the very complex components that *Bootstrap* had no alternatives for and which were crucial for the web application. An example of such components would be the filtering implementation which *Element Plus* handles perfectly while Bootstrap would require implementing it from scratch. Furthermore, since the project has grown quite big, it is a very likely outcome, that there are still some places where *Element Plus* remained due to team members simply missing it while going through the code. Only the removal of the *Element Plus* as a dependency and traversal through the web pages of the *Protsessor* web application to find broken style and functionality at this point would guarantee the complete removal of it.

8.4 What Could Have Been Done Differently

Despite our team being positive about the made changes and finding them valuable and necessary, one of the major things we would change is the end goals that were set up at the beginning of the project. There are multiple issues with the currently selected goals which make it drastically harder to achieve a good result in the process of a Bachelor's thesis defense. The first reason for that is the fact that a good portion of the work that was done as a part of this thesis is very technical which makes it particularly hard to present those changes and to understand them as they can not be simply put into the picture for the visualization purposes or easily described without going through the code. One of the biggest concerns our team has is that all the complexity and hardship we had to go through will not be embraced to the full extent. Secondly, the problem with the goals is that in the end, we can not simply describe the work that was done with a simple compact sentence that would immediately make it clear what was done in the scope of this work and why those changes are important. Our team had to make a lot of small and medium changes in different areas and aspects of the project, however just naming them "Further development" would have been too broad of a concept, which would only make it harder to see the bigger picture.

9. Next Steps in Development

Despite the fact that our team managed to do a big work on the *Protsessor* web application it is not nearly ready and is still far from perfection. During the work on it, our team came up with a few ideas for future development that could potentially bring *Protsessor* web application to a whole new level making it very easy and pleasant to work on. Even though, there is still some core functionality missing, our team believes, that the next steps should be taken towards improving the technical health of the project to ensure that the project will not become overcomplicated and too expensive in terms of time to work on.

9.1 Health Improvement

There are several reasons why our team thinks that the current project is in a very poor health state even though we have already made some steps towards improving it. The first and major reason for this is that the project's backend contains many places where outdated ways are used like using autowired annotation instead of the constructor injections, some duplicated and unused code, imports, so-called code smells, commented-out code, TODOs, etc. All of this results in a way more confusing code and makes it a lot more expensive in terms of time for the new developers to live into the project and make any further changes as well as make the whole application work slower while also making it harder to spot any additional bugs. One of the things that also makes a huge contribution to the previously mentioned problem is functions that contain too many code lines. Functions have to have a descriptive name and have a single responsibility. If the amount of code grows too big inside a single function, it probably means that this function is responsible for multiple things at the same time which makes code over complicated and hard to understand. In addition, there are a lot of places that retrieve information from the database by making too many requests when the same result can be achieved with a lower amount. Additional requests make the application work drastically slower and make it harder to understand the code for the new developers. All previously mentioned problems make it particularly hard for the new teams to introduce any changes as they would require a lot more than just time to make themselves familiar with the project.

One of the possible solutions to the occurred project health issues that our team sees as a meaningful step is to dedicate one of the topics to overhauling the whole project in order to remove all the duplicated logic, and code smells, address the hanging TODOs, etc. In addition, to the mentioned steps, it might be wise to analyze the necessity as well

as determine the suitable tool whose purpose is to improve code quality to be used in this project. One such tool is the *SonarQube* - a tool which can be integrated into the *CI/CD* [25]. If configured properly, it can send a report for each pull request that contains an analysis of the number of code smells, percent of tests code coverage, potential bugs, etc. This will not only allow to determine most of the technical debt issues but also enforce a certain level of code quality among the other teams that will be further developing the *Protsessor* application at any time.

9.2 Securing the Project

Another huge issue in this project that should be addressed as quickly as possible is the security issue. This project is poorly designed in terms of security which in turn may lead to potential data loss, unexpected data modification, privilege escalation, etc. Currently, the only thing that prevents users from accessing the functionality that is restricted according to their permissions, is *Vue's* conditional rendering system. Additionally, only some of the backend rest API endpoints are protected by *Spring Security*. Also, frontend is using some of *Vue's* attributes that are considered not secure like the "HTML" attribute which can be used to inject malicious *HTML* code. In our opinion, one of the good topics for the next thesis would be to further investigate the current state of security measures in order to address the possible security breach vectors and make the *Protsessor* application as secure as possible.

9.3 Split the Application Into Microservices

Another huge topic that can be part of further development is splitting the currently monolithic application into separate smaller services - microservices. A monolithic approach is very convenient in the early stages of the application development as it provides a complete overview of the whole project in the same environment, it is a lot easier to deploy, and understand as well as is more efficient in terms of performance. However, as the project keeps growing and becomes huge, the previous pros quickly turn into cons. It is a lot harder to understand a huge project and thus more time is required to make any changes to it. Even the slightest change to the project will require redeploying the whole monolith. Furthermore, if any part of the application will have an error, it might affect the whole application, in general, [26]. As a result, our team thinks that this change can lead to the better overall health of the project, as well as make it more realistic for the next teams to introduce any changes, as carefully and well-planned topics might require to make changes to only one or few microservices which in turn might drastically lower the amount of the boilerplate that the students will have to go through as well as might decrease the amount

of the technologies that they would need to know in order to make any meaningful changes.

However, it is very important to note that splitting the monolith into microservices can turn out to be a very hard, time-consuming, and tedious task. If such a decision would be made, it shouldn't be taken lightly and is required to properly plan each step as well as each microservice's responsibility and purpose. Probably it would be impossible to fully migrate to the microservice architecture type as part of a single Bachelor's thesis topic.

9.4 Tests Overhaul

One of the issues that the *Protsessor* web application currently has is an improper way of testing which leads to certain problems. Currently, all tests include the actual flow of the application, as an integration test should have. However, even though the integration tests are quite handy and are widely used, this type of test is not well suited for every component that has to be tested in the *Spring Boot* application [27]. Due to current methods of testing, it can become very expensive and tedious to make any changes to the currently implemented business logic. As such, test methods should be overhauled to test most of the components by using the unit type of tests with the proper use of a *Mockito* framework while leaving the integration tests mostly for testing controllers and repositories.

10. Conclusion

Our team is very positive about this project and feels the necessity for such web application to exist as it can drastically improve the way of going through the thesis experience as well as make this process a lot easier and more pleasant. However, in our opinion, there are several steps that should be taken in order to make this project more appealing and subsequently in bigger demand by the students and teachers, as well as make it a lot easier for future teams to make any further changes.

Regarding some of the project issues, in the scope of our work, we had pointed out some major flaws of the implementation by the previous developers, however, it was not done in an attempt to undermine the work that was done by them or to elevate our accomplishments. The goal was to bring attention to the fact that, such systems with this many requirements need proper project structure and architecture, which most of the students would not be capable of achieving, without majorly halting the progress of future teams, due to core project issues that can not be avoided by junior/undergraduate developers.

Overall, while inefficient in many places due to errors and personal lack of knowledge, this project provided us with a lot of experience, due to requiring us to learn proper ways of solving certain issues, which then lead us to an existing solution that we feel quite pleased with. Our team believes that we managed to make *Protsessor* web application drastically closer to the end goal of becoming an ultimate thesis management web platform for the students.

References

[1] *Taltech styleguide*. [Accessed: 24-12-2022]. Tallinna Tehnika Ülikool. URL: https://www.figma.com/proto/Nngd7mhoajdGO3Nesz58E4/Taltech-styleguide.

[2] *Taltech styleguide*. [Accessed: 24-12-2022]. Tallinna Tehnika Ülikool. URL: https://portal-dev.ttu.ee/styleguide.

[3] *Web Content Accessibility Guidelines 2.0*. [Accessed: 24-12-2022]. W3C. URL: https://www.w3.org/TR/WCAG20/.

[4] Sedrik Suurmets, Mikk Järvis, and Kaspar Ustav. "Lõputööde ja projektide haldamise infosüsteem Protsessor". Tallinna Tehnika Ülikool, 2022.

[5] *What is Gradle?* [Accessed: 24-12-2022]. URL: https://docs.gradle.org/current/userguide/what_is_gradle.html.

[6] *What is Camunda, Features and Use Cases?* [Accessed: 24-12-2022]. URL: https://www.adservio.fr/post/what-is-camunda-features-and-use-cases.

[7] *Spring Boot JPA*. [Accessed: 24-12-2022]. URL: https://www.javatpoint.com/spring-boot-jpa.

[8] *Spring – JDBC Template*. [Accessed: 24-12-2022]. URL: https://www.geeksforgeeks.org/spring-jdbc-template.

[9] *Spring Security*. [Accessed: 24-12-2022]. URL: https://spring.io/projects/spring-security.

[10] *What is Azure Active Directory?* [Accessed: 24-12-2022]. URL: https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis.

[11] *Spring Boot – Sending Email via SMTP*. [Accessed: 24-12-2022]. URL: https://www.geeksforgeeks.org/spring-boot-sending-email-via-smtp.

[12] *What Is Vue.js? The Pros and Cons of Vue.js in 2022*. [Accessed: 24-12-2022]. URL: https://www.trio.dev/blog/why-use-vue-js.

[13] *The benefits of PostgreSQL*. [Accessed: 24-12-2022]. URL: https://www.prisma.io/dataguide/postgresql/benefits-of-postgresql.

[14] *What is DevOps?* [Accessed: 01-01-2023]. URL: https://about.gitlab.com/topics/devops.

[15] *What is CI/CD?* [Accessed: 24-12-2022]. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[16] *What is Docker?* [Accessed: 24-12-2022]. URL: https://aws.amazon.com/docker.

[17] *What is a modern UI design*. [Accessed: 19-05-2023]. URL: https://mockitt.wondershare.com/ui-ux-design/modern-ui.html.

[18] *Project documentation*. [Accessed: 19-05-2023]. URL: https://gitlab.cs.ttu.ee/services/graduation/protsessor-wiki/-/wikis/Introduction.

[19] *Camunda's Database Schema Documentation*. [Accessed: 19-05-2023]. URL: https://docs.camunda.org/manual/7.19/user-guide/process-engine/database/database-schema/.

[20] *Protsessor database documentation*. [Accesed: 21-05-2023]. URL: https://gitlab.cs.ttu.ee/services/graduation/protsessor-wiki/-/wikis/Database.

[21] *Adding Liquibase to an Existing Project*. [Accessed: 24-12-2022]. URL: https://www.liquibase.com/blog/adding-liquibase-on-an-existing-project.

[22] *Github Repository: docker-postgres-backup-local*. [Accessed: 24-12-2022]. URL: https://github.com/prodrigestivill/docker-postgres-backup-local.

[23] *Introduction to Camunda Platform 8*. [Accessed: 24-12-2022]. URL: https://docs.camunda.io/docs/guides.

[24] *Camunda Platform: Reinventing Process Automation for the Digital Enterprise*. [Accessed: 24-12-2022]. URL: https://f.hubspotusercontent10.net/hubfs/4513465/Camunda%5C%20Platform%5C%20Datasheet%5C%20EN-1.pdf.

[25] *clean code for teams and enterprises with SonarQube*. [Accessed: 24-12-2022]. URL: https://www.sonarsource.com/products/sonarqube.

[26] *Microservices vs. monolithic architecture*. [Accessed: 25-12-2022]. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith.

[27] *Unit Test Vs Integration Test*. [Accessed: 25-12-2022]. URL: https://www.practitest.com/qa-learningcenter/resources/unit-test-vs-integration-test.