TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Oleg Kartašov 155248IAPB
Andres Pajuste 155219IAPB

Web Application for Graduation Project Management

Bachelor's thesis

Supervisor: Ago Luberg
MSc

Tallinn 2020

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.

Author: Oleg Kartašov, Andres Pajuste

25.05.2021

Abstract

The purpose of the current bachelor's thesis is to create a service for automation and simplification of proposing, searching, choosing and confirming projects. The service is partially public and as a public service provides a list of schools and departments with a list of proposed projects. The service supports different types of users with different functionality. For the supervisors, the service allows to quickly create, modify or delete projects to propose, allows to see a list of applications to the project created by students and allows to confirm or reject it. For the students, the service allows to check a list of proposed projects, find a proper project and apply to it. For the admin, the service allows to see a list of all existing users and track whether a student has a confirmed project or not. The service provides one scenario of project confirmation with two-level confirmation by supervisor and student side.

This project is built following a Client-Server architecture and divided into two parts: back-end as a REST API and front-end as a Web application.

This work consists of analysis with requirements to the service, development with project design, development process and project functionality and next steps of development with existing issues.

This thesis is written in English and is 62 pages long, including 7 chapters, 12 figures.

Annotatsioon

Lõputöö projektide haldamise veebirakendus

Käesoleva bakalaureusetöö eesmärgiks on luua veebirakendus, mis automatiseerib ja lihtsustab projektide esitamise, otsimise, valimise ja kinnitamise protsesse. Veebirakendus on osaliselt avalik ja kuvab teaduskonnad koos kättesaadavate projektidega. Veebirakendus toetab erinevat tüüpi kasutajaid erineva funktsionaalsusega. Õppejõu jaoks võimaldab veebirakendus projekti lisada, moodustada, eemaldada ja esitada tudengitele, võimaldab vaadata projektidele esitatud avaldusi tudengitelt ja võimaldab neid kinnitada või tagasi lükata. Tudengi jaoks võimaldab veebirakendus vaadata pakutatud projekte, leida sobivat projekti ja kandideerida projektis osalemiseks. Administraatori jaoks võimaldab veebirakendus vaadata kasutajate nimekirja ja jälgida, kas tudengitel on kinnitatud projekt või mitte. Veebirakendus toetab projekti kahetasemelise kinnitamise stsenaariumi õppejõu ja tudengi poolt.

Veebirakendus on loodud klient-server arhitektuuri baasil ja jagatud kaheks osaks: serverirakendus kui REST API teenus ja kasutajaliides kui veebirakendus.

Lõputöö sisaldab analüüsi, kus kirjeldatakse ära nõuded, arendusprotsessi koos projekti disainiga, veebirakenduse funktsionaalsust ja edasise arenduse samme koos kirjeldatud probleemidega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 62 leheküljel, 7 peatükki, 12 joonist.

List of abbreviations and terms

API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CRUD	Create, Read, Update and Delete
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JSON	JavaScript object Notation, data-interchange format
JWT	JSON Web Token
REST	Representational State Transfer, software architectural style
SASS	Syntactically Awesome Stylesheets
SPA	Single Page Application
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience

Table of contents

List of figures

1 Introduction

1.1 Background

At that moment, finding a graduation topic with a supervisor is a very inconvenient and time-consuming process for a student at the School of IT of the Tallinn University of Technology. A list of potential supervisors with their graduation topics proposals are posted on the TalTech webpage as static data. Each student can go to a webpage and look at a list of graduation topics proposals, but to choose a topic, students need to contact each potential supervisor separately. This is not a very convenient process for both the student and the member of academic staff. On the one hand students need to spend a lot of time dealing with each potential supervisor separately. On the other hand, the supervisor receives proposals from students by letter or verbal agreement. When it comes to a large number of applications, it becomes difficult to manipulate requests from students, because each student waits for an answer, but there is a chance that the supervisor will accidentally miss the letter, due to the high amount of them. In addition, it is necessary each time to manually delete the topic from the page to let other students know that the topic is already not available. In addition, the supervisor does not know exactly how much student attention proposed topics could draw. Also, for the supervisor there are not many ways to manipulate own project, adding new ones, changing existing ones. To do this, the supervisor needs to change the data on the TalTech web page which is not very convenient and also a time-consuming process. As a result, the current graduation topic selection process requires a lot of manual work and consumes a lot of time as well.

1.2 Purpose

The main purpose of the current project is to create separate applications for automating and simplifying the selection, confirmation and managing of projects with subsequent adding of finished graduation thesis documents to Digikogu library. Service must support at least two scenarios: scenario with bachelor and master graduation level.

Service must allow supervisors to create and manage projects for students with different scenarios by graduation levels. A student is given the opportunity to conveniently search for and choose a project to apply and wait for confirmation from the potential supervisor.

A supervisor may confirm or refuse an application of a student or a group of students if it is a group application.

The student is also given the opportunity to create teams and apply as a group if the graduation topic proposal is allowed to do so.

Service can be fully managed by an administrator. Administrators must have ability for flexible configuration of service: manage groups, departments, users, roles and projects.

1.3 Requirements

1.3.1 User story

The following functional requirements are in user story format and reflect expected functionality for specific types of users. Three types of authenticated users can be identified in this project: Admin, Supervisor, Student.

The following functions are required for each type of user

- each user can see supervisor profile with list of available projects
- each user can be able to search projects by author, tags or title under specified department

Authenticated user

- As an Authenticated user, I want to be able to see and edit own profile information

Admin

- As an Admin, I want to be able to see the list of users sorted by roles
    - list of supervisors
    - list of students with selected project if it selected
- As an Admin, I want to be able to manage users
    - add a new user

        - change existed user information
        - change user status
        - change user roles
    - As an Admin, I want to be able to change project status
    - As an Admin, I want to be able to create groups for grouping projects and users by requirement

Supervisor

- As a Supervisor, I want to be able to manage each of my projects
    - add a new project
    - delete a project
    - edit existing project
    - open detailed view of a specified project
    - see the list of my projects sorted by status
- As a Supervisor, I want to be able to manage student applications to choose projects
    - see the list of applications to my projects sorted by status
    - accept an application
    - deny an application

Student

- As a Student, I want to be able to see Supervisor profile with his/her available projects list
- As a Student, I want to be able to create a new team, for doing the group project

- As a Student, I want to be able to manage the created team
    - delete a team member
    - add a new team member
    - delete the team
- As a Student, I want to accept or decline the team participation
- As a Student, I want to be able to see the list of available projects with authors
- As a Student, I want to be able to see detailed view of a specific project
- As a Student, I want to be able to apply to a project
    - create a single application
    - create a group application
- As a Student, I want to see a list of confirmed/declined applications from a supervisor
- As a Student, I want to be able to see the detailed view of a confirmed project

1.3.2 Support requirements

- The service must support multiple roles for one user. Scenario, when the supervisor can be also admin or student.
- Projects can have multiple supervisors and are proposed for multiple departments.
- Service must have Markdown support for more comfortable project description
- Authentication and authorization process must be available via uniId for supervisors and students.

1.3.3 Architecture requirements

- Service must support different types of scenarios such as bachelor and master graduation levels.
- Service must allow to group projects and users

1.3.4 UX/UI requirements

- The service must be partially accessible for viewing. That means that a non-authenticated user has the opportunity to visit a web application to view proposed projects according to department or view supervisor profile with available projects.
- UI must support multiple languages. UI must be available on Estonian and English languages.
- UI must be a multi-platform web application. That means that service must have a responsive UI and be available on different types of devices.

2 Project Design

This project is built with as Client-Server architecture, which allowed splitting the service into three independent parts: database, REST service as a business logic and SPA web application as a user interface. Separation of Business logic and Web application allows for greater flexibility and independence of one part from another.

2.1 Database

Database table diagram. (Figure 12)

Database tables description

- `user` - table for user entities
    - `id` - database unique identifier
    - `user_id` - randomly generated unique identifier of numbers and letters
    - `first_name` - first name of the user
    - `last_name` - last name of the user
    - `email` - email of the user
    - `encrypted_password` - encrypted password by bCryptPasswordEncoder [12]
    - `graduation_level` - graduation level of user (master, bachelor)
    - `status` - status of user (active, not active)
    - `starterTeamId` - this field is set by `team_id` of the team that is automatically generated with user creation to apply for solo projects
    - `confirmedProjectId` - when the application is confirmed by student and supervisor, this field is set by confirmed `project_id`
    - `role_id` - role of the user (admin, student, supervisor)

- `role` - table for role entities
    - `id` - database unique identifier
    - `role_name` - name of the role
- `authority` - table for authority entities
    - `id` - database unique identifier
    - `authority_name` - name of the authority
- `roles_authorities` - table for many-to-many relationship between role table and authority table
    - `role_id` - database id of the role
    - `authority_id` - database id of the authority
- `group` - table for group entities
    - `id` - database unique identifier
    - `group_id` - randomly generated unique identifier of numbers and letters
    - `group_class` - class of the group (for example School, Department, etc)
    - `group_name` - name of the group (for example IAIB, IAPB, etc)
    - `parent_group` - parent of this group
- `user_group_role` - table for relationships between users, groups and roles to understand what role each user in each group has.
    - `id` - database unique identifier
    - `user_id` - database id of the user
    - `role_id` - database id of the role
    - `group_id` - database id of the group

- `user_group` - table for many-to-many relationship between user and group tables (redundant table which logic must be fully transferred to `user_group_role` table)
    - `user_id` - database id of the user
    - `group_id` - database id of the group
- `project` - table for project entities
    - `id` - database unique identifier
    - `project_id` - randomly generated unique identifier of numbers and letters
    - `status` - status of the project (available, not available)
    - `creating_time` - date and time when project is created
    - `accpeting_time` - date and time when project is accepted by team
    - `language` - project language
    - `title` - project title
    - `description` - project description
    - `student amount` - how many students should do this project
    - `degree` - project degree (bachelor, master)
    - `user_id` - project author
    - `team_id` - when project is accepted by team and supervisor this field is set by `team_id` of the team
- `groups_projects` - table for many-to-many relationship between group and project tables
    - `group_id` - database id of the group

    - `project_id` - database id of the project

- `projects_cosupervisors` - table for many-to-many relationship between user and project table
    - `project_id` - database id of the project
    - `user_id` - database id of the user
- `tag` - table for tag entities
    - `id` - database unique identifier
    - `tag_id` - randomly generated unique identifier of numbers and letters
    - `tag_name` - tag name
- `projects_tags` - table for many-to-many relationship between project and tag tables
    - `project_id` - database project id
    - `tag_id` - database tag id
- `team` - table for team entities
    - `id` - database unique identifier
    - `team_id` - randomly generated unique identifier of numbers and letters
    - `team_name` - name of the team
    - `status` - status of the team
    - `author_id` - id of the team creator
    - `project_id` - this field is set when the project is accepted by team and supervisor
- `team_member` - table for team member entities

        - `id` - database unique identifier
        - `team_member_id` - randomly generated unique identifier of numbers and letters
        - `team_id` - team, where this team member is working
        - `user_id` - what user belongs to this team member
        - `role` - role in the team (student, supervisor, co supervisor)
        - `status` - team member status (active, not active)
    - application - table for application entities
        - `id` - database unique identifier
        - `application_id` - randomly generated unique identifier of numbers and letters
        - `status` - application status
        - `work_type` - type of the work (team, solo)
        - `creating_time` - date and time when application is created
        - `message` - application message
        - `team_id` - what team create this application
        - `project_id` - to what project this application was sent

2.1.1 PostgreSQL

For our service as a database, we are using PostgreSQL. PostgreSQL is a free and open-source object-relational database system. [17]

PostgreSQL is running in a docker container on the development server. Service is connected to the database using JDBC.

2.1.2 H2

For local testing we are using H2 in-memory database. H2 is very comfortable for developing stage, because this database is built-in in spring and only one dependency needed to start working with it. H2 has ddl-auto: create-drop, so every time when the application is stopping the database deletes all the data.

2.2 REST service

REST service that provides API for the web application from client side. RESTful API service is built using Java 11 with Spring Boot framework. All HTTP requests are sent to Spring REST controllers. After that, data from requests go to the service layer where Spring Data JPA repositories are `autowired`. Depending on the request and operation type, Hibernate generates SQL queries to the database to retrieve data. This data is converted from entity to DTO (data transfer object) type and is sent back as a response to client side. Security is provided by Spring Security. (Figure 1)

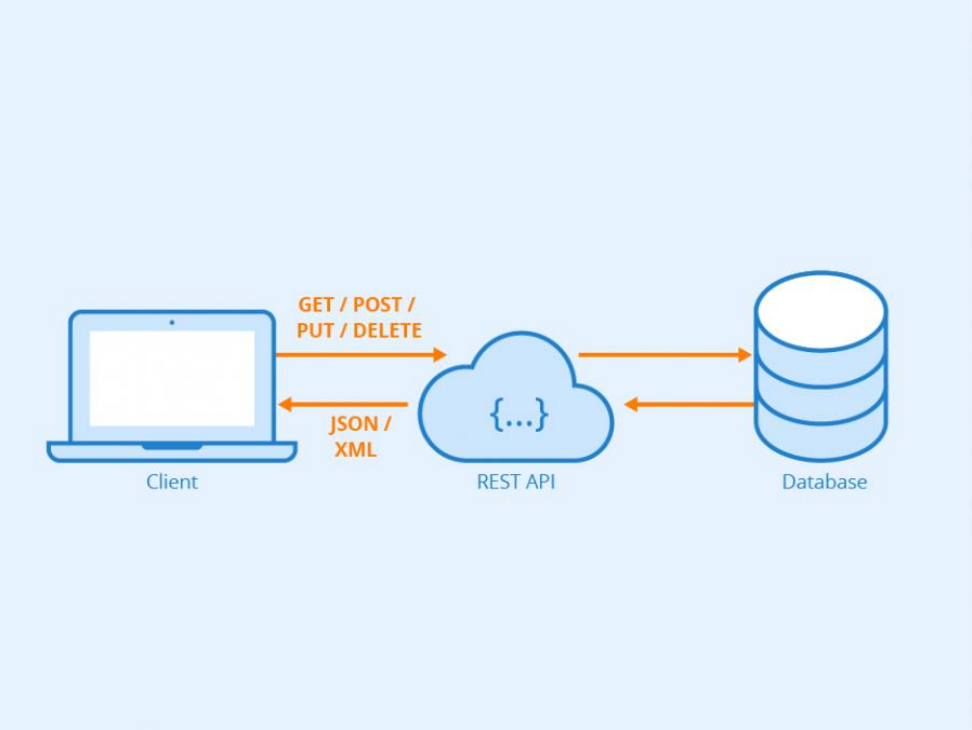

Figure 1. Rest API service. [15]

2.2.1 Spring

Spring is the most popular application framework for Java platform [9][10]. Because of the built-in frameworks like dependency injection and inversion control, Spring has a wide range of functionality and features. Spring is an easy, fast and secure Java framework with high performance, which is perfectly suitable for developing web applications like our service.

2.2.2 Maven

For building application and lifecycle control our REST API is using Maven. Maven is one of the available and popular tools for building Java projects. The POM file contains all the dependencies, which are needed for proper work of REST API. Maven allows users to put in the project root script files, which can be run by application or by CI/CD tools.

2.2.3 JWT Token

For the authorization process we decided to choose JWT Token. JWT token is necessary so that REST API understands whether it is the user who pretends to be. JWT token consists of header, payload and signature and every time when an authenticated user attempts to send the HTTP request to the REST API, the system takes the token from the request header and checks if it is signed with the right token secret value and if this token has not expired yet.

2.2.4 BCryptPasswordEncoder

For user password encoding and decoding our REST API is using bCryptPasswordEncoder. BCryptPasswordEncoder is a very secure password encoder. [12] With the help of this encoder, all users' passwords could be safely presented in the database without the possibility of being revealed. When the admin creates a new account and puts some value in the password parameter field, this value is automatically encoded and only after that is put in the database.

2.2.5 Log4j2

Our REST API service API supporting logging by Log4j2 framework. [14] With the help of that framework all errors and warnings are written into a separate file. The configuration file log4j2.xml is located in the root directory of the project. After the

project starting, the folder "logs" with "application.log" file are created and all logs are going into that file. It helps to track down bugs and errors while application is running.

2.2.6 Authentication

At this stage of development authentication is realized with a built-in spring filter UsernamePasswordAuthenticationFilter. When an authentication attempt is made, the service reads the data from the request body of login input and first of all tries to find email in the database. If the email is existing, the service take password from request body and compares it with the decoded database password value. If the attempt is successful, the service generates a JWT token and adds it to the response header with `userId` of authorized user.

2.2.7 Authorization

When the HTTP request is sent to the REST API, the service checks if authorization is needed to process this type of request. If the authorization is obligatory, service verifies the JWT token and checks that this token is not expired and if it is signed with the proper token secret value, which is located in `application.properties` file. After that Spring Security `@PreAuthorize` annotation checks if the user has the proper role to make this HTTP request. (Figure 2)

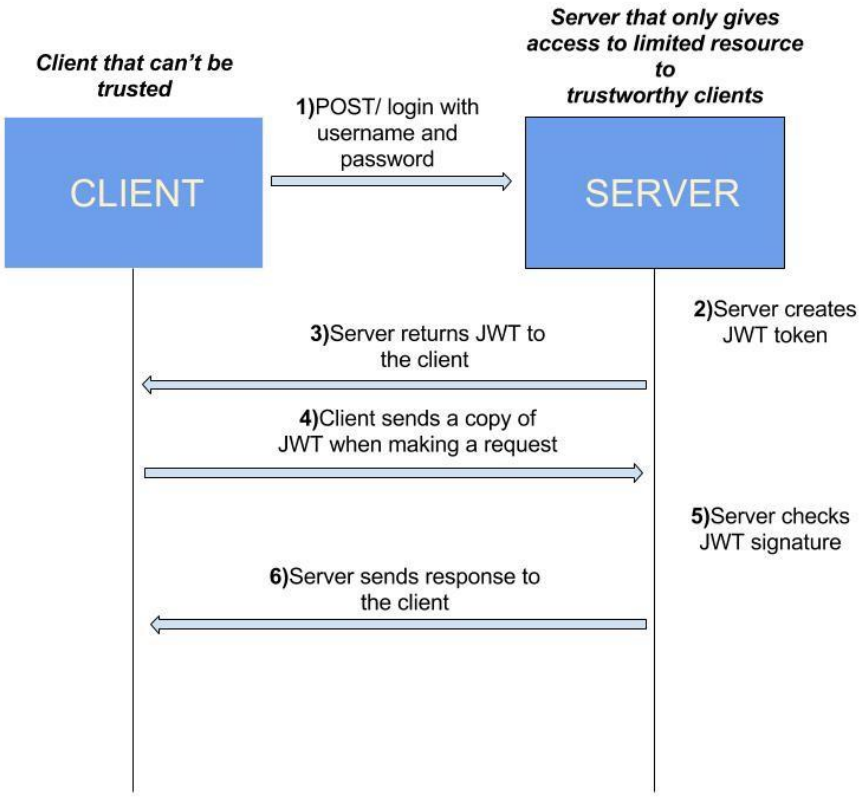

Figure 2. Authentication based on JWT token. [16]

2.2.8 Swagger

Our REST API is supported with Swagger configuration. Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON [11]. Swagger has the UI for sending HTTP requests to REST controllers and showing the full response. It can be used for testing API service without client side or for interactive documentation.

2.3 Web application - SPA

The User interface was realized as a SPA using Vue CLI. SPA is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages [1].

2.3.1 Vue CLI

CLI is a standard tooling for Vue.js. Vue CLI is a full system for rapid Vue.js development based on Node.js provides pre-made development configurations to create comfortable frontend development ecosystem. [2]

This Web application includes some dependencies for more comfortable development such as ESLint [19] for JavaScript code analyzing and finding problems and SASS [20] as a CSS extension for better organization of styles. The whole list of included dependencies can be found in the `package.json` file.

2.3.2 Vuetify

The Vuetify UI framework was included for an easy and more rapid development. The framework provides a rich list of components, tools and configurations. Vuetify takes a mobile first approach to design exactly according to Material Design specification [3].

2.3.3 Markdown

Markdown is a language for creating formatted text. In the current project Markdown is used for decoration of description of graduation topic proposals. Project description data is held in the database in Markdown format and rendered to the HTML for displaying on the client side in web applications [8].

2.4 REST API and Web application communication

REST service communicates with the SPA web application via HTTP protocol. REST service provides API for SPA web application. Data exchange in requests and responses is done in JSON format. For consuming data from REST API a HTTP client called Axios [5] was added.

Axios is a Node library that provides a promise-based HTTP client. Axios provides a simple way for configuration of requests to the API [5].

3 Development process

All development is done in the GitLab environment [13]. Due to the fact that the project consists of independent parts, it was decided to divide the development into two repositories. One repository includes the back-end (or REST service), another repo includes the web application. Development process takes place in two environments: locally and on a remote test server. Development was organized through the Continuous Integration and Continuous Delivery for both repositories. For organization of CI and CD processes a GitLab tool called GitLab runner was used.

3.1 GitLab CI/CD

GitLab runner is a tool built into GitLab for development through Continuous Integration and Continuous Delivery. In our project this tool helps us to update the project version to the final stage of development in our test environment. Needed configuration file was added to organize CI/CD processes according to GitLab documentation. This configuration file is called `.gitlab-ci.yml` and located in both repositories. These files create pipelines with running processes when code in the master branch was changed. Files contain predefined jobs that run in parallel, and perform all necessary actions to build, test and update applications on the remote production server (Appendix 2) (Appendix 3).

3.2 DB setup

Our database is running in a docker container on our production server and the database properties are located in the `application.properties` file in the root folder of the service. Now, there are three different database settings. In the first commented section there are properties for PostgreSQL database, which is used on the production server. In the second commented section there are properties for PostgreSQL database too, but which are used for local testing.

The third section is the section for H2 database, which is used for local testing as well.

3.3 REST service setup

The REST API service is a Java application, which is using Spring framework. To run this application locally the Java 11 must be installed on the computer. When the project is opened in the IDE, the service's main class should be executed to start the application.

3.4 Web Application Setup

Web application is based on Node.js, as previously stated, so it needs Node installed on the development environment. Installation process is described in the `README.md` file that holds the project repositorium.

3.4.1 Connection with REST API service

For connection with REST API the correct path to this service needs to be specified. Path to the API service is specified as a variable `axios.defaults.baseURL` in `main.js` file, located in the Web application repository under the `src` folder.

3.4.2 Remote Server

To run the web application in the remove test server, a simple web server based on Node.js [6] and pm2 daemon process manager was added to keep the application online [7].

3.5 Testing

3.5.1 REST service testing

At this stage of the development integration and unit tests are not configured and all functionality is tested manually. However, our service supported by REST Assured framework [15] and this framework will be used for integration tests.

3.5.2 Web application testing

At this stage of the development automated testing is not configured for the web application. All functionality is tested manually.

4 Project description

The basic functionality of this service is a simple and easy way for a student to find, choose and confirm projects offered by potential supervisors. This Service is a partially public web application. That means that each guest user can go to the web application page. For non-authenticated users, the application looks only as a directory with a proposed projects list. A similar structure has been created in the service with the TalTech web site, when the supervisors with their offers are sorted by schools and departments. Absolutely any non-authenticated guest user can open any school and department and get acquainted with the proposed projects, choose one of them, open it to get more detailed information about the expected project. Also, non-authenticated users are able to open the supervisor profile and see the list of his projects. The main functionality of the service such as project management, application creation and confirmation are only available for authenticated users.

At this stage of development, the service supports only one scenario regardless of graduation level. Short description of the process: supervisor adds a project, student chooses and applies to it, supervisor and students confirm it. More detailed functionality divided by roles described are below.

4.1 Roles

The service supports three different types of roles: Administrator, Supervisor, Student. The main functions available for authorized users according to the role is listed below. Service also supports multiple roles for one user. Scenario, when the supervisor can be admin or student.

4.1.1 Supervisor

Available main features for supervisors, at this stage of development, are:

- ability to create and manage projects for students with different graduation levels.
- ability to receive requests on a particular project from students. A supervisor may confirm or refuse an application from a student or a group of students, if it is a group application.

- ability to see detailed information of the confirmed project such as project description, project team members.
- ability to open his own profile.

Also, the supervisor can specify co-authors for each project and specify in what groups the projects could be presented. In this case only the authors can manage projects.

4.1.2 Student

Available main features for students, at this stage of development, are:

- to conveniently search for and choose a project to apply and wait for confirmation from the potential supervisor.
- to create teams and apply as a team if the proposed project is suitable for teamwork.
- ability to see detailed information of the confirmed project such as project description, project team members.
- ability to open his own profile.

4.1.3 Administrator

As intended, users with administrator roles are able to perform any actions available to users with other roles, manage projects, tags, teams, users, schools and departments. At the time of development, the focus was on basic functionality for supervisor and students, so intended functionality was not fully implemented. The only available feature for administrators, at this stage of development, is to view user lists.

4.2 UX/UI

This service has a responsive user interface and is available on different types of devices. Also, service has a Markdown support that makes it possible to write simple styled descriptions for projects.

5 Service functionality

5.1 Navigation

On the homepage service presents a list of schools with departments. Under each department there is a list of graduation topic proposals. These pages are public and available for each guest user. For authenticated user two menus appear:

- a sidebar with navigation menu. Display a list of available functions for each type of authenticated users
- a navigation bar user menu. Located on the top of the page, and gives functionality related with the authenticated user

5.2 Roles switching

This service supports multiple roles for a single user. Suggested combinations of roles within the system is:

- Supervisor with an admin role
- Supervisor with a student role

The system does not support access to all roles for the authenticated user at the same time. If the user has multiple roles, he must choose one of them at the login process. After that, if an authenticated user requires functionality with another role, he can switch roles in the navigation bar menu. Role switching can be seen in Figure 3.

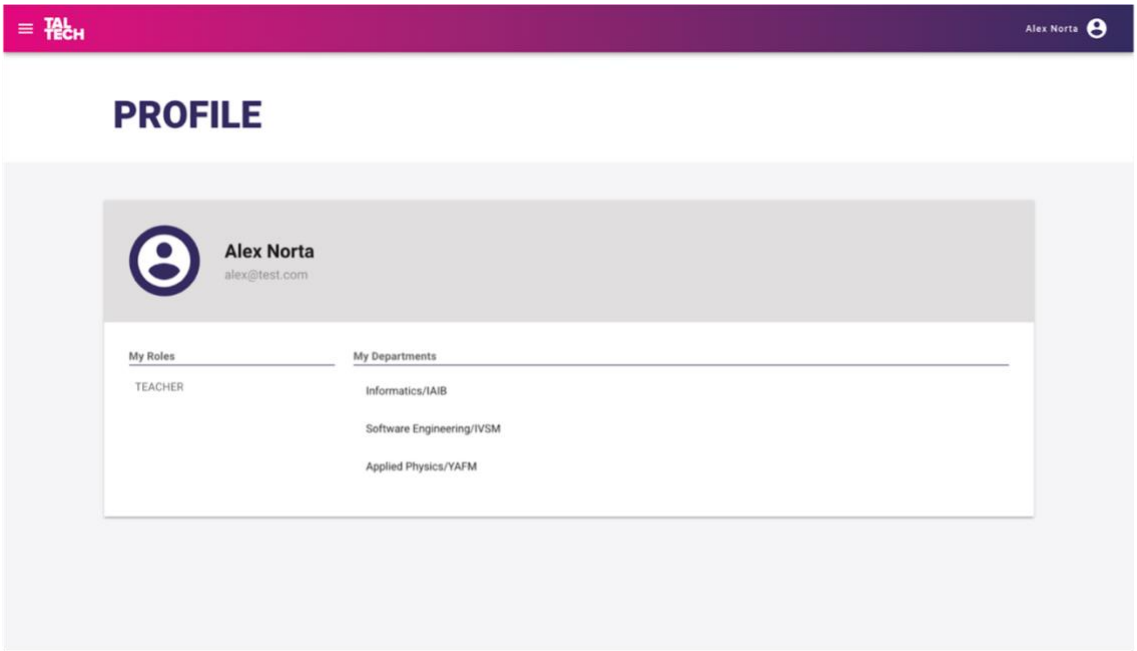

Figure 3. Navigation menu with role switching.

5.3 User management

As intended, only users with administrator roles are able to manage users. This stage of development only supports displaying users divided by roles: a list of students and a list of supervisors separately.

As an administrator, user can see lists of:

- all existing supervisors in the service. The administrator can select one of the supervisors and open a profile with an additional information about the supervisor and a list of all his projects.
- all existing students in the service. In this student list, the administrator is able to see which student has a confirmed graduation topic and which one does not. If the student has a confirmed project, the administrator has the opportunity to open the project and look at the details of this project.

5.4 Profile

The profile display is divided into two types:

- authenticated user's own profile. In their own profile, an authenticated user can see his own name and email and also his own roles and departments in which he is. Authenticated user profile can be seen in Figure 4.

- profile of supervisor. Each user, authenticated and non-authenticated, is able to open a supervisor profile. At this moment, the only way to open a supervisor profile is via department and available projects. Supervisor profile for other users contains the name and email address of the supervisor, and also a list of his own projects.

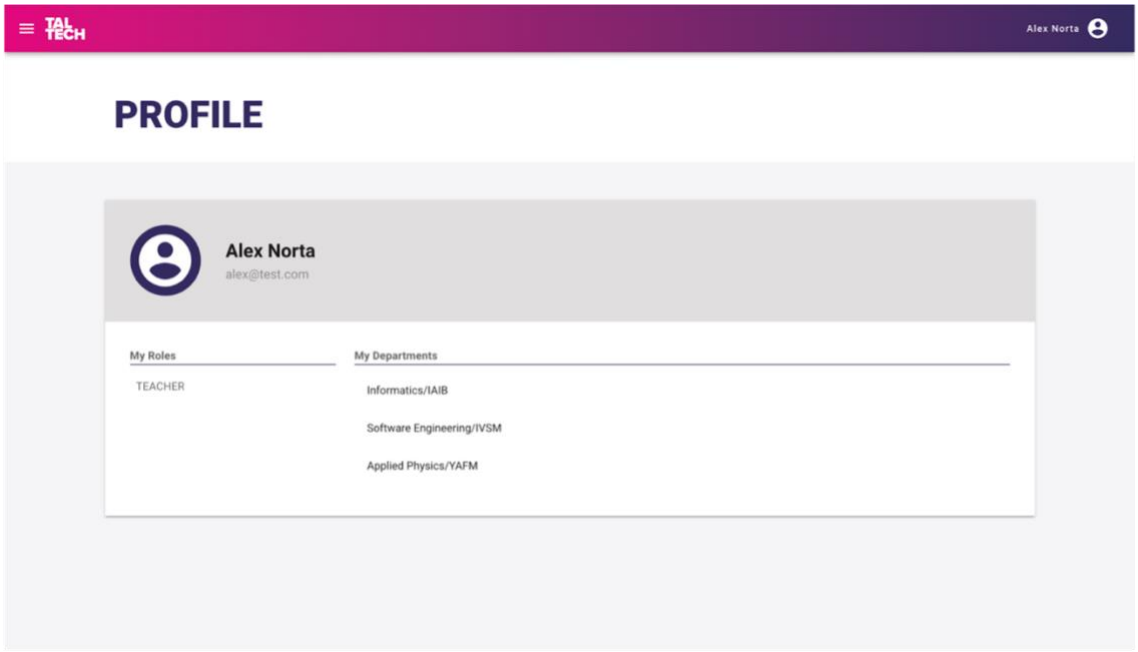

Figure 4. Authenticated user profile.

5.5 Projects

At this stage of development, the only person who can manage the projects is the supervisor. Supervisor can add, delete, edit his own projects. Supervisor sees a list of own projects sorted by status. Each existing project has two main statuses: `project_available` and `project_not_available` but, at this moment, only `project_available` projects are displayed. In addition to individual projects, a supervisor may have projects where he is a co-supervisor. These projects are displayed in a separate list. Also, supervisors have additional sorting features, such as:

- searching a project through available projects by project title.

- sorting available projects list by title, date, graduation level and language.
- searching a project through co-supervising projects by project title or author.

Supervisor project list displaying can be seen in Figure 5.

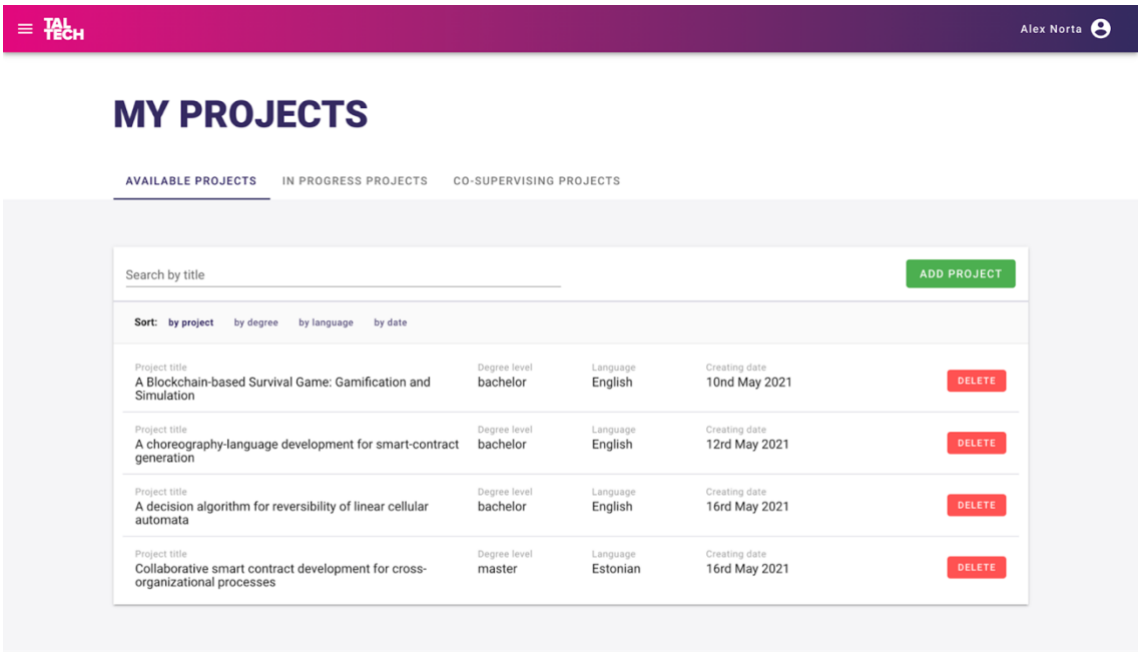

Figure 5. Supervisor projects list page.

5.5.1 Project addition

Each supervisor has access to his own project creation. Project contains such parameters as:

- project title
- description. Added Markdown support for a more beautiful and convenient display of the project description
- list of tags. There is an option to find and add existing tags or create a new one.
- English or Estonian language
- bachelor or master graduation level
- students amount. At that moment students limit for one project is 3 persons.

- list of departments for which the project will be available.
- co-authors list

All parameters listed above are mandatory except for co-authors and tags. Each newly created project gets available project status and becomes visible under the specified departments automatically. Also, this project is added to the specified co-authors.

5.5.2 Project editing and deleting

Project edition includes the ability to change all project values except for departments and co-authors values. At this stage of development, there is no necessary logic on the back-end to change departments and co-authors list. Project edition and deletion is only available for projects with a status `project_available`. Also, only project authors and admins can manipulate the project, co-authors are denied this opportunity at this stage of development.

5.5.3 Project displaying

A list of available projects is displayed separately under each department. Each user can open a selected department, and see a list of available projects, then choose an available one and open it in a separate view with detailed description. If an authenticated user is a student, who has the department which project includes, he is able to see only a sorted list of graduation topic proposals by graduation level. To see all available projects with any graduation level, a supervisor profile has to be opened. Students can also check the list of available graduation projects regardless of the graduation level for other departments. Department page with a list of offered projects for applying can be seen in Figure 6.

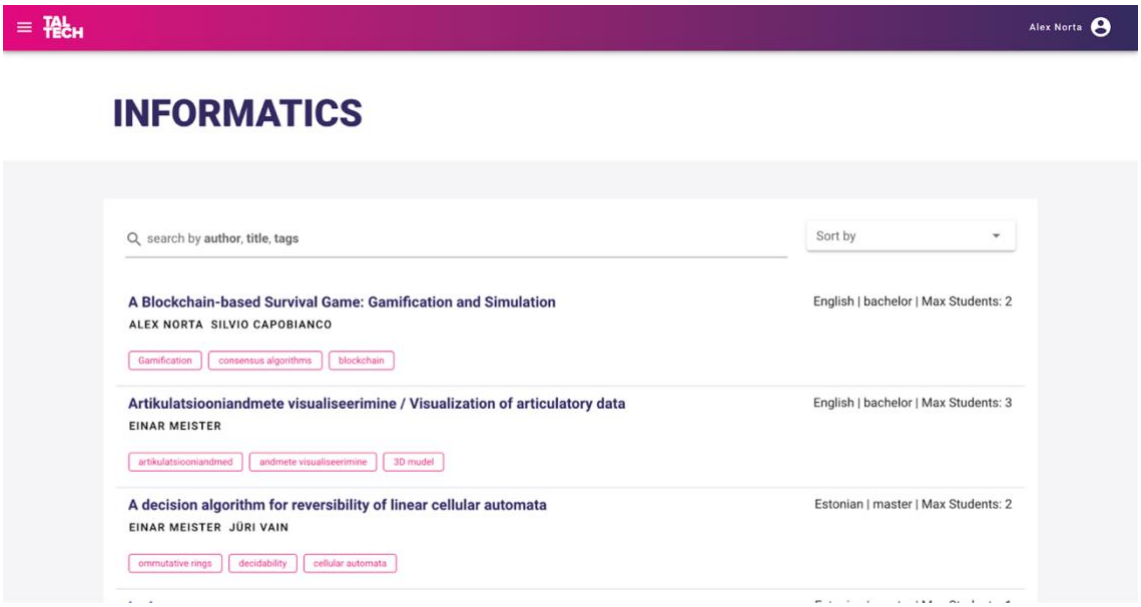

Figure 6. Department with list of offered projects.

Students with confirmed projects have a separate link to the project in the sidebar menu.

5.6 Teams

In this service, the team is a list of users with different roles for each confirmed project. It includes a list of students, supervisors and co-supervisors related to the project. We can mark two scopes for the teams:

- team for application creation
- team after project confirmation

In the scope of the project choosing and application creation, the team is a group of students which would like to find and apply to the group project. Each student can create a team or be invited to the team. The list of teams is sorted by authorship and displayed separately for each student:

- a list of teams, where author is authenticated user
- a list of teams where user was invited

In the scope of the confirmed project, which is in progress, the team contains a list of users related to this project. Each confirmed project has its own team, whether a group of students applied to the project or whether it is an application from a single user. From the

point of user interface, an application with a single user does not contain a team, but from the point of the system, even a single application has a team with one student. When a project is confirmed, all users associated with the current project are added to the team automatically. In summary, this team contains project authors and co-authors, students or groups of students, all with correct team roles.

5.6.1 Team management

This feature is only available for students. Each student can create a team. To create a team, the student must specify a team name and invite another student or students. List of available students for invitation sorted by graduation level, so the team can only consist of students with the same graduation level.

At this stage of development, only team owners can manage teams. Team owner can:

- delete team
- add new team members
- delete team members
- create a group application to the project. Only team owners can apply to the project, but each team member sees a newly created application and can track it.

Team owner view can be seen in Figure 7.

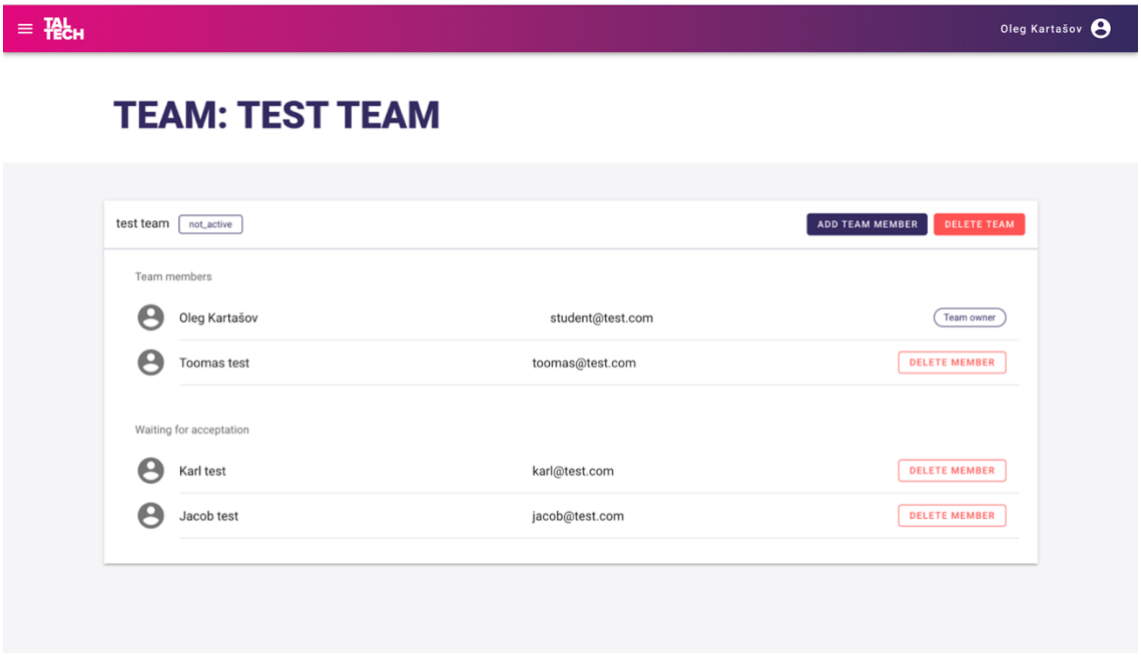

Figure 7. Team view by team owner.

5.6.2 Team participation

Newly created teams automatically get status `not_active`, and all invited students of the team can see a list of teams, where they were invited, under a separate tab. Each invited student can accept or deny their own participation in a specified team. When all students, who are invited to the team, answer about the participation, the team automatically changes status to active and becomes available for application to the projects. If one of the team participants leaves the team, for some reason, the team automatically changes the status to not active.

5.7 Project confirmation process

The main functionality of this service is submission of a project application followed by confirmation. At this stage of development, application confirmation is realized with two steps. Applications need supervisor confirmation and students need to confirm the same application right after. After successful two-sided confirmation, the project is confirmed and become unavailable for other students.

Application has different statuses such as:

- Wait for an answer from the supervisor. Application created by student and sent to supervisor.

- application confirmed by supervisor and sent back to student
- application declined by supervisor and not active any more
- application confirmed by the student and project is confirmed.
- application declined by student and not active any more

5.7.1 Application creation

The student has rights to apply only for a specific project. Students and projects must conform the department and graduation level. If a student planned to create a group application, the student's amount should not exceed the specified max amount in the project.

Students can choose between a single or group application. If a student has available teams with status `active`, that means all invited students of the team accept or decline participation, he can create a group application, or choose a single application. For a group application, any member of the team may apply to the project. Also, in application form, students can add some notes for the supervisor. Application creation example will be seen in Figure 8.

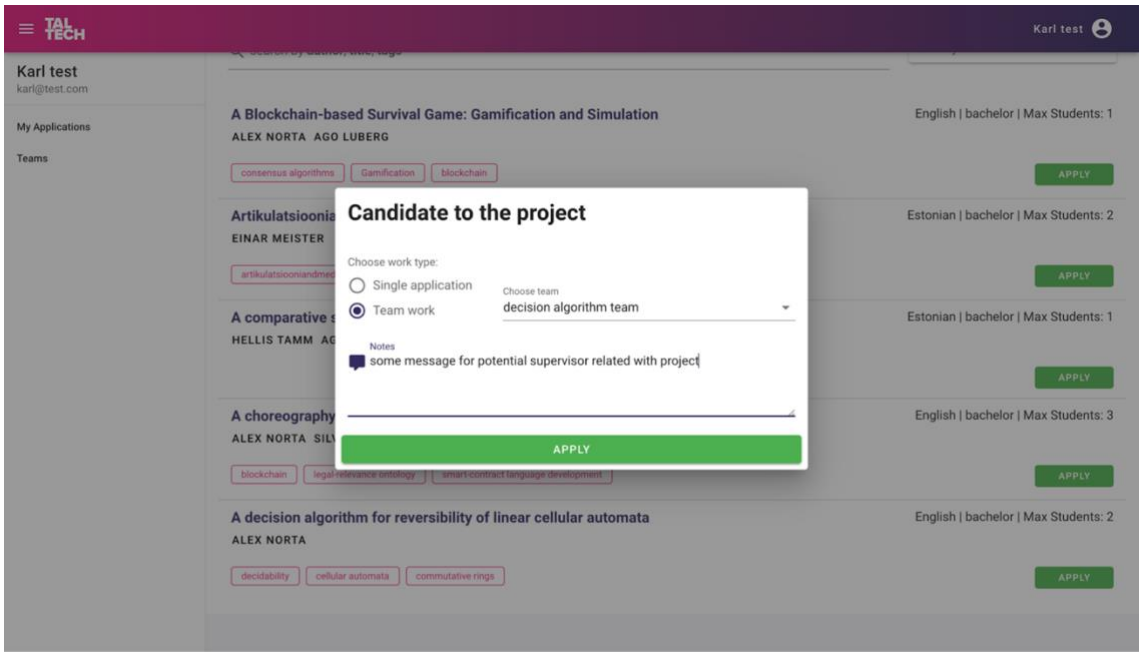

Figure 8. Application creation.

Application to the project can be created only once. At this stage of development students could not create single and group applications to the one project. Need to

decide and choose only one type. Each newly created application gets automatically status `application_sent`.

5.7.2 Application deleting

Students can call off newly created applications until the supervisor reacts to it and the status of the application changes. In this case the application will be remove and the supervisor does not get an application. After the supervisor 's reaction, the application cannot be deleted and can only change status according to student or supervisor manipulation.

5.7.3 Application displaying

Both supervisor and student have a separate page where they see the whole list of applications. Students see their own single or group applications and supervisors see a list of applications to their own projects. Sorting and displaying the application list for supervisors and students is the same and divided into relevant tabs, such as:

- not accepted - list of newly created applications
- accepted by supervisor
- declined by supervisor
- declined by student

For the student the list of applications is also sorted by type. Single applications and group applications displayed separate under each tab of applications. Sorting of application lists is identical but display of each application related information is different. For student view each application contain such parameters, as:

- project name with projects supervisors
- team name if it is a group application
- creating date

For supervisor view each application contain such parameters, as:

- project name

- type of application. This value tells supervisors does it is single application or group application
- student name. If it is group application, in this field displayed name of team leader, who created this team
- creation date
- message from student
- information about the team, if it is a group application. Display team name, list of team members with their names and emails and also mark who is team leader.

Supervisor view with student applications to the projects can be seen in Figure 9.

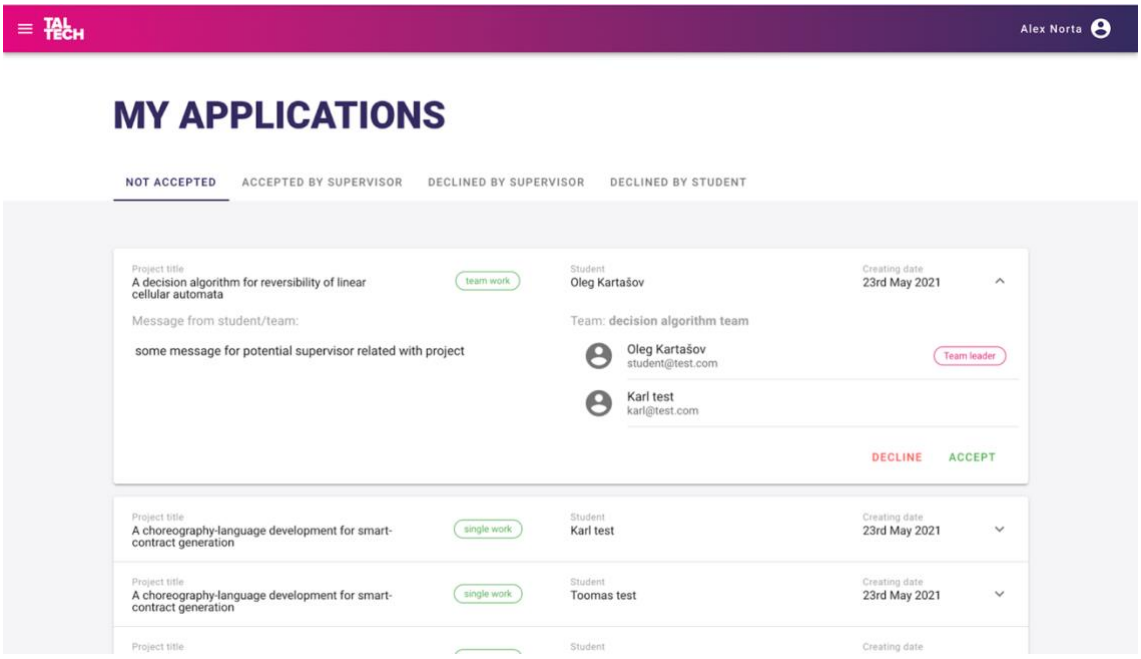

Figure 9. List of student applications to the projects.

5.7.4 Application tracking and confirmation

Absolutely each user related to the application can track its status. The steps of project confirmation will be listed below.

1. Students create single or group applications. After application creation, students can see it in the list of applications. If a student created a group application, the team member will also see it in the list of applications. Students can call off

applications in this stage and therefore delete it. In this stage an application is available and was sent to the supervisor.

2. Supervisor gets applications from students. He decided to accept or decline the application. After acceptance, the application changes status and returns back to receive confirmation from the student. If a supervisor declines an application, it changes status and becomes inactive for both student and supervisor. Regardless of the choice of the supervisor, the student sees the changed status of the project under the relevant tab. If a project contains multiple supervisors, only the author can manage applications related to this project.

3. Student gets a confirmed application from the supervisor. He decided to accept or decline a confirmed application. If a student declines an application, it changes status and becomes inactive for both student and supervisor. If student accept confirmed by supervisor application, next steps will take place:

    - application change status to `application_confirmed`
    - project change status from `project_available` to `project_not_available`. That means that the project becomes unavailable for other student's applications. For supervisor, project transfer from the main list of available projects to another list of projects in progress.
    - the team consisting of students and supervisors is automatically created.
    - The link to the confirmed project for student appears in the sidebar menu. By opening a confirmed project, both supervisor and student can see detailed project description and also project team members, separately supervisors and students.
    - automatically deleted other student applications and teams.

In progress project displaying can be seen in Figure 10.

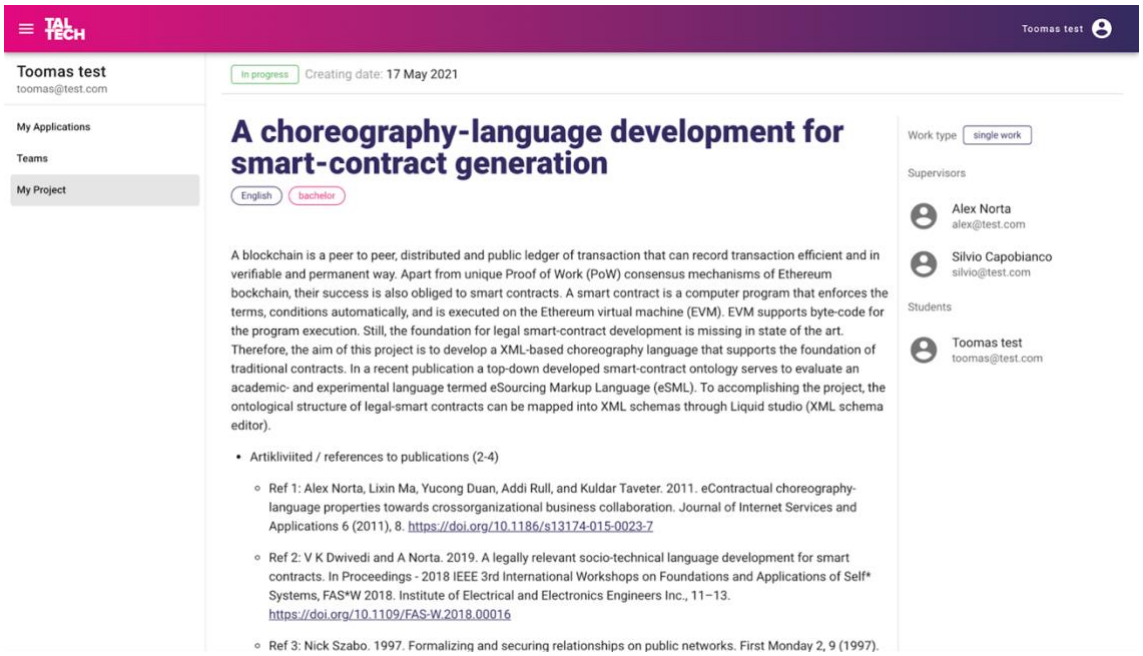

Figure 10. In progress project detailed view.

As was said above, service provides two-level confirmation for application. It was released to avoid situations when students, for example, create a lot of applications, and more than one supervisor confirms it. In the end, students choose and confirm one of the projects, confirmed by the supervisor. Also, two-level confirmation makes application tracking more comfortable for the supervisor. Supervisor can see that the student declined his confirmed application.

5.7.5 Workflow diagram

The process described above is presented as a workflow diagram in Figure 11.

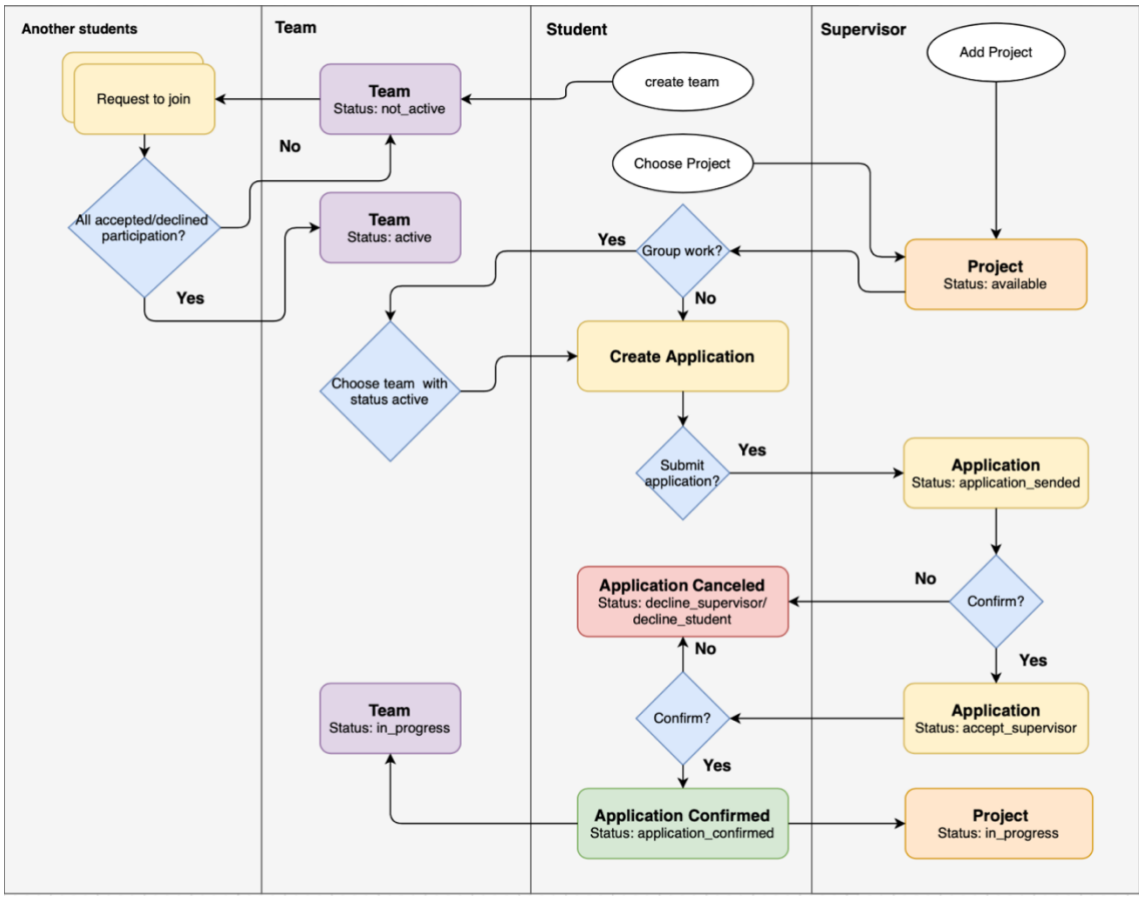

Figure 11. Project confirmation process.

5.8 Searching

At this stage of development searching was realized in several places such as:

- project creation form. Supervisor can search tags, co-authors and departments during project creation.
- team creation form. Student can search students during team creation or team members addition.
- list of projects for supervisor view. Supervisor can search projects by title in the available projects list or by title and author in co-supervising projects list.

- list of projects under the department. Each user can go to one of the departments pages and start searching projects by title, tags or supervisor name. Searching is limited by data. That means that search only takes place inside the department based on the data that was uploaded from the back-end.

6 Validation

The server uses multi-level validation for different tasks due to separate REST and client services.

6.1 REST service validation

REST service has many different layers of validation which are supported by different tools and dependencies. The global and method-level validation is backed by Spring Security, so most of the controller endpoints are secured and authorization is needed to send HTTP requests to them. There are some temporary endpoints which can be proceeded without authorization. It is H2 console endpoint for accessing in-memory database for testing and swagger endpoint to send HTTP requests to the service through user interface. In the production stage GET HTTP method for `/group`, `/project`, `/user` endpoints will be open and will not need authorization, because in our service all types of users, even without logging in, can search for groups with available projects and can watch supervisor profiles with suggested projects. All other endpoints need valid JWT token with correct token secret signature and expiration date. Some of the endpoints in addition have method-level security that only users with the proper role can have access to them. This is supported by Spring Security annotation `@PreAuthorize` above the controller method. All input data which are coming in the request body have their own validation supported by spring-boot-starter-validation dependency. At the data model service layer all methods have their own validation and if this validation fails, it throws service exception with user friendly description.

Each controller validation is described below.

6.1.1 Login(Authentication) validation

Now for authentication our service is using Spring built-in login endpoint and validation (in the issue module is described how we will change it in future). Login can be proceeded without authorization. Client sends HTTP request with request body containing username and password to REST service. Using `@NotNull` annotation in `UserLoginRequestModel` class, spring checks that all input fields are not null and `@Email` annotation above email parameter checks that email input field meets standards. After that, authentication filter

make authentication attempt, where service try to find input email in database and if there is a match, service take password from request body and compare the value with database decoded password value. Then if the attempt is successful, the service generates a JWT token and adds it to the response header with user id value.

6.1.2 User controller and service validation

Without authentication and authorization, clients can access only GET HTTP `/user` endpoints. Service does not have validation for these endpoints because every user, even without logging in could get information about other registered users. At the user service layer methods `getUserByUserId` and `getUserByEmail` have validation that if a user is not found in the database, service throws a service exception. For sending HTTP POST requests to `/user` endpoint (user creation attempt), users must be authorized (present valid JWT token with correct signature and expiration date) and have admin role. Because now, only admins can create new users. This means that beside global security, this endpoint has method-level security annotation `@PreAutorize`(“`hasRole('ADMIN')`”). This annotation is executed before method execution and controls that authorized user have admin role. Then all request body fields are controlled by `@NotNull` annotations and input email parameter field have `@Email` annotation as well. At the user service layer there is a verification that this email is new and has not already been put in the database and verification that all groups id which are sent in the request body are existing in the database.

For sending HTTP PUT requests to `/user` endpoint (update user), user must be authorized and besides that have admin rights or this authorized user must be a profile owner. This means that with a help of `@PreAuthorize`(“`hasRole('ADMIN')` or `#userId == prinicipal.userId`”) annotation spring checks that user id of authorized user is the same as id that was put in the request. Request body is validated by `@NotNull` annotations and at the user service layer there is a verification that this user profile is existing in the database.

HTTP DELETE requests to `/user` endpoint can be executed only by authorized users with admin rights. At the user service layer there is a verification that this user profile is existing in the database.

6.1.3 Project controller and service validation

The same as with users, all GET HTTP requests to `/project` endpoints have access without authentication and authorization, so that every user could get acquainted with the suggested projects and their descriptions. Project service layer methods `getProjectByProjectId` and `getProjectsByUserId` are protected by "not found" service exceptions.

To send HTTP POST requests (create project), users must be authorized and have an admin and/or a teacher role. `@PreAuthorize` annotation supports multiple roles. Only students have no access to create projects. In the request body list of groups, supervisors and tags are sent as well, so at the project service layer there are verifications that all groups and co supervisors which are sent in the request are presented in the database. For tags, create project method checks if the name of the tag is existing in the database, if not the project service creates a new tag and adds it to the project. If the name exists, the existing tag is added to the project

For HTTP PUT requests (update project), spring checks that the user is authorized and has an admin role or/and checks that the authorized user is the project author. At the project service layer there is verification that a project which must be updated is presented in the database.

For HTTP DELETE requests (delete project) the same strategy is used. User must have an admin role or/and this user must be a project author. At the project service layer there is verification that a project which must be deleted is presented in the database.

6.1.4 Group controller and service validation

All types of users can send HTTP GET requests to `/group` endpoints, without authentication and authorization. At the group service layer `getGroupById` is checked if the group is in the database and in methods `getGroupWithStudents` and `getGroupWithSupervisor` besides group existing verification, is present role verification of users in group.

To send HTTP POST and HTTP DELETE requests to `/group` (create/delete group) endpoints, users must be authorized and must have admin rights, because in our service only admin could create and delete groups.

6.1.5 Tag controller and service validation

HTTP GET requests to `/tag` endpoints could be sent only by authorized users.

HTTP POST and DELETE requests (create and delete tag) could be sent only by a user with an admin role. If the admin would like to create a tag, there is a verification that this tag name has not already been presented in the database.

6.1.6 Team controller and service validation

For teams, only authorized users could send HTTP GET requests to `/team` endpoints for retrieving team data.

In our application only authorized users with admin and/or student roles could send HTTP POST requests to `/team` endpoint (create team). In request body users must pass a list of users id (team members). At the service layer there is a verification that all users in the list and team creator are presented in the database

Only authorized users with admin rights or a team creator could delete the team if the team is presented in the database.

6.1.7 Team member controller and service validation

To get a team member by team member id user must be authorized and this team member which user is looking for must be in the database.

There are three HTTP POST requests endpoints in the team controller.

- `addMemberToTeam` - authorized user with admin or/and students rights. It can be done if the team and user (which must be added to the team) are presented in the database and if this user has not already been in the team.

- `acceptMembership` - authorized user with admin rights or/and an owner of user account, which is added to the team. It can be done if the team and user (which must accept membership) are presented in the database and if membership has not already been accepted.

- `declineMembership` - authorized users with admin or student role rights. It can be done if the team and user (which must decline membership) are presented in the database and if membership has not already been accepted.

6.1.8 Application controller and service validation

All authorized users could send HTTP GET requests to `/application` endpoints and get the application data, if this data is presented in the database.

There are five HTTP POST request endpoints in the application controller.

- `createApplication` - authorized user with admin or/student role. The application could be created only if the project (to which the application was sent) and the team were presented in the database. Besides that, this team must have correct status(`active`), the project must have correct status(`project_available`) and this application is sent to this project for the first time (to escape duplicates).

- `acceptApplicationBySupervisor/declineApplicationBySupervisor` - authorized user with admin rights or an owner of the project, to which the application was sent. The application could be accepted/declined by the supervisor if the application and the supervisor are presented in the database and if this application has not already been accepted or declined.

- `acceptApplicationByStudent/declineApplicationByStudent` - authorized user with admin rights or an owner of the team, which has sent the application. The application could be accepted/declined by the student if the application and the team are presented in the database and if this application has not already been accepted or declined.

Only authorized users with admin rights or a user, which team create the application could send HTTP DELETE requests to the `/application` endpoint (delete application), if this application is presented in the database.

6.2 Web application validation

The main part of validation in web applications is needed to verify access to one or the other function. After successful authentication, records some authenticated user parameters to the local storage for subsequent access verification such as:

- JWT token that needs for securely requesting to the REST service

- Role of authenticated user for displaying relevant functionality

- Department ID for student to specify available department for application creation

6.2.1 Access validation

Each user has access to only certain pages and functionality, according to the role. At the page render stage, a role check is performed and only then the navigation menu is displayed. It can be said, for example, that a student does not have the ability to open a page intended for an administrator or supervisor using a user interface. If, for some reason, a student tries to open a project page for a supervisor directly via URL, he will not be accessed to the page and will be redirected to the web application homepage. In general, when a user tries to enter directly with a URL to a page that is not available to them, a redirect is happening. This control applies to all users regardless of the role. If a guest user tries to open some page intended for an authenticated user, redirect to the login page.

The service also limits a student's ability to apply to the project. The student has the right to apply only for a suitable project. As has been said before, students and projects must conform in department and graduation level. If a student planned to create a group application, the student's amount should be no more than the specified max amount in the project.

6.2.2 Forms validation

For all forms on the pages of the web application organized a simple field validation. Login form has simple validation to check valid format for email and password length and could not be empty. The same applies to the form of the project. When creating a project, the supervisor must specify absolutely all values, otherwise the project will not be created. Also validate team creation form and application creation forms.

7 Summary

The main global purpose of this project was to create applications for automating and simplifying the selection, confirmation and managing of projects. The main task was accomplished, each supervisor has the ability to create a project and offer it to students. Students can view the list of proposed projects and choose one of them and apply for the project. The application is then validated and if the confirmation is successful the project changes status to `application_confirmed` and is recorded for the student and the teacher. Not all was completed as required, as there were difficulties in the development process which will be discussed further.

7.1 Issues

7.1.1 Functional issues

1. At this moment completely missing the intended functionality for the Administrator. At this stage of development we only show a list of users. At least need to add the functions described in the requirements to the project. The administrator needs to be able at least modify user data such as roles and departments and create new users. Also, an important function is changing the status of inactive users or removing it from the system. It would also be nice to show and be able to manage schools and departments as an administrator. However all back-end REST API endpoints have been already configure for admin usage.

2. Need to add ability to specify project type. At this moment all existing projects have one type.

3. Support list of available languages in project description. At that moment, the supervisor can specify only one language for the project. Also, at that moment the project description displayed only in English language. At the future planning user

interface on Estonian. Main difficulty is how to record and hold project descriptions in both languages.

4. Support functionality that project could have both bachelor and master's degree.

5. Need to disable or delete functions for students such as application creation and team creation after project confirmation.

6. Ability to edit profile information is missing at that moment. Authenticated users can only open it and see their own name, email, list of roles and departments.

7. At this stage of development, searching works as a sorting of results by some parameter. Searching is realized on the front-end level and makes it possible to search only through loaded and displayed data from the back-end. It is necessary to develop searching with a query to the back-end. There is also a lack of search for project applications.

8. Need to display a list of co-supervising projects in the supervisor profile. At this stage of development, only supervisors' own projects are displayed.

9. Need to prevent the ability to invite a student to a team that has a confirmed project.

10. Need to cover up all functionality with tests and configure a test plan for each pipeline.

11. While project editing, need to give supervisor opportunity to change groups and co-supervisors.

7.1.2 Architectural issues

- At this stage of development, a lot of logic responsible for the correct display of data to the other user is implemented at the front-end level by JavaScript. It is not the best solution. For example, we display a list of graduation topics proposals sorted by graduation level for the student. Sorting is organized during page rendering, but in fact students get a whole list of graduation topic proposals from the back-end regardless of graduation level. As a result, we can sort it on the

service side and return a sorted list to web-application. It is not a critical issue, but nice to fix it.

- Need to finalize logic of what happens when the team confirms the project. Now after confirmation all other teams of confirmed team author are deleted with all applications. Perfectly, when a project is confirmed all teams of confirmed team members are deleted but in applications are changed only statuses.

- Now `declineMebership` method can decline membership of the team member and delete team member as well. Is needed to separate this functionality to provide additional validation for both methods.

- At the beginning of development, service have many to many relationship between users and groups, that users can have many groups and vice versa. However, later we understand that this is not enough, because besides that service needs to understand what role the user in every group has to provide additional functionality and sorting. So, for this needs `UserRoleGroup` entity was built. This entity must be created with user by admin who have rights to give roles. This makes `users_groups` database table redundant, but there is needed more time to transfer business logic to `UserGroupRole` entity.

7.1.3 UX/UI issues

1. Update date if project or application was changed. At this moment we display the creation date for the project and application. It would be nice to display the date of change status for application or display date of last project modifying.

2. Add notification support. Currently, the supervisor does not receive any notification if an application has been sent to one of his projects. He should manually check in the appropriate tab whether there are any changes. This applies to the whole process of changing the application status. The same goes for an invitation to the team, when one student, who creates a team, invites other users. Other users need to check manually all team invitations. Need to add a notification system for both situations.

3. Show loaders after request to the back-end and before data rendering on the page.

7.2 Comments

The main difficulty in the development process was to understand the entire application design and scope with required functionalities to make the application more flexible for the future development and functionality modernization. In some functionality development processes we got some new conditions and needed requirements that needed to integrate to the system. For this purpose, it was necessary to continuously modify and change application architecture. As a result, it took a lot of time not so much to build the architecture, but to update code for new needs and architecture.

Also, during development, we put emphasis on creating as much functionality as possible, to make architecture as wide as possible. As a result of the ever-changing architecture, we neglected testing. So, for the most part the functionality was tested only by manual.

7.3 Future development steps

For the next development iteration must be included all issues, described above. In addition, the following are described as more important development steps.

7.3.1 Project completion

Currently no scenario available after confirmation of project. Needs ability to change project status after completion. The easiest solution is to add a button into a project with status `in_progress`, that changes project status. Function must be available to both project owner and administrator. After project status changes it displays in a separate list of completed projects for the supervisor and for a student as a finished project.

Need to develop a scenario where the project has been confirmed, but not finished for some reason. The project should be available again for other students and all dependencies associated with the previous student needs to remove. It would be better to have an action history as well. For example, if a project was confirmed at once, but for some reason not finished and reopened for other students, it would be convenient to know the history of the project for the admin and project author.

7.3.2 Finished student

Necessary to decide what to do with students after successful completion. Students who successfully get graduation need to be either removed from the database or change status

to disable. One of the decisions is to disable all functionality for the finished student besides being able to view the project that he did.

7.3.3 Multiple scenario

Need to develop different scenarios of projects depending on graduation level. At this stage of development, the supervisor can choose the graduation level for the project, but it only affects sorting for students. Regardless of graduation level, the project confirmation process is similar. Overall, the system supports only one scenario, when a supervisor adds a project, students apply to the project and follow the submission process.

7.3.4 Projects grouping

At this stage of development, all projects are grouped only by department with a simple notation of graduation level. This is sufficient to display projects for students but need more complex project grouping and access configuration. As was described above, service supports only one scenario, regardless of graduation level, but in fact, that bachelor and master graduation levels have different scenarios. To solve this problem, we need to add the ability to group projects by some parameter. In this case we need to group projects by different graduation levels to provide different scenarios. For example, administrators create two groups: bachelor graduation topics and master graduation topics. In addition, projects can be grouped by semester, for example. That means that the project can be contained in several groups at the same time. From an architectural point of view, in such a case, the project should contain a list of the groups in which it is located. Or the second variant is a sublist of groups, when one group is contained in another and one of them contains projects. For example, semester groups that contain department groups that contain groups by graduation levels with included projects. Also, can be developed grouping by tags or topic. Ideally, the administrator should be able to create new groups and adjust the group inheritance.

7.3.5 Users grouping and projects access

Currently, each student has access to projects according to the department. It would be better to be able to group users. It would be easier to set permissions and available functions for users. For example, group all bachelors students connect with a group of bachelor graduation topics. From an architectural point of view, the administrator could

create an abstract group that contains groups of projects and groups of students. It would make it easier to manage permissions and available areas for students.

7.3.6 UniId authentication

Authentication via UniId. This development needs a lot of time for investigation and consist of several problems:

- Login process itself. Complexity of integration Microsoft login is that the developed service consists of two independent systems and for both need to hold Microsoft account sessions. It is a solvable issue, but will take a lot of time, so we decided to postpone it to the end.

- User registration into the service. The problem is that within the service each user has certain parameters necessary for proper operation. For example, graduation level or department id for students. We see two solutions to this problem.

    1. User login to the service for the first time and after successful login, display form requesting necessary parameters. So, each new user sets its parameters itself. But in this case the user can set wrong parameters, which will lead to further errors.

    2. Administrators add new users and set necessary parameters for each user. More flexible solution, that is able to change user parameters in future, if it is needed.

    3. It is possible to use the Active Directory groups to check if the user should have a student or staff role or both.

7.4 Potential functionality

Possible functions to complement the existing service are described below.

7.4.1 Project export

Option of export completed graduation projects for administrator or supervisor. Need to configure the connection between the service and TalTech library archive for automatic export. When the project is completed, it is possible to export it as a separate file with all the necessary data into Digikogu.

7.4.2 Confirmed project management

Working with a confirmed project. One potential option to allow the authenticated user related with the project to input new information related to the project during development. For example, ability to add final documentation, ability for supervisor to add some general notes for students or mention links to repositorium.

7.4.3 Third-party companies

Develop an analogical existing functionality for third-party companies. Enable customers to also add and offer graduation topic proposals to students as well. Or enable administrators to add graduation topic proposals from third-party companies.

7.4.4 Student graduation topic proposals for supervisors

Give the student an opportunity to propose the topic to the supervisor. Students choose a supervisor and offer their own graduation topic proposal. Another scenario, when students offer some topic under department and all supervisors can see it and take it.

References

[1] "Single-page application", 19.05.2021 [Online] Available: https://en.wikipedia.org/wiki/Single-page_application [Used 10.05.2021]

[2] "Vue CLI", 10.07.2019 [Online] Available: https://cli.vuejs.org/guide/ [Used 10.05.2021]

[3] "Vuetify", 08.5.2021 [Online] Available: https://vuetifyjs.com/en/introduction/why-vuetify/ [Used 10.05.2021]

[4] "Introduction to JSON Web Tokens", [Online] Available: https://jwt.io/introduction [Used 10.05.2021]

[5] "Axios", [Online] Available: https://axios-http.com [Used 10.05.2021]

[6] "Simple Web Server in Node.js", [Online] Available: https://gist.github.com/aolde/8104861 [Used 20.03.2021]

[7] "PM2", [Online] Available: https://pm2.keymetrics.io/docs/usage/quick-start/ [Used 10.05.2021]

[8] "Markdown Cheatsheet", 29.05.2017 [Online] Available: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet [Used 10.05.2021]

[9] "Spring", [Online] Available: https://spring.io [Used 10.05.2021]

[10] "10 Best Java Frameworks to Use in 2021", 23.02.2021 [Online] Available: https://hackr.io/blog/java-frameworks [Used 10.05.2021]

[11] "Swagger", [Online] Available: https://swagger.io [Used 10.05.2021]

[12] "Class BCryptPasswordEncoder", [Online] Available: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html [Used 10.05.2021]

[13] "Gitlab", [Online] Available: https://about.gitlab.com [Used 10.05.2021]

[14] "Log4j2", [Online] Available: https://logging.apache.org/log4j/2.x/ [Used 14.05.2021]

[15] "Rest assured", [Online] Available: https://rest-assured.io [Used 12.05.2021]

[16] "Authentication based on JWT token", [Online] Available: https://laptrinhx.com/json-web-token-based-authentication-in-django-2407621164/ [Used 20.05.2021]

[17] "PostgreSQL", [Online] Available: https://www.postgresql.org [Used 20.05.2021]

[18] "RESTful API", [Online] Available: https://www.seobility.net/en/wiki/REST_API [Used 20.05.2021]

[19] "ESLint", [Online] Available: https://eslint.org [Used 20.05.2021]

[20] "SASS", [Online] Available: https://sass-lang.com [Used 20.05.2021]

Appendix 1 – Database diagram

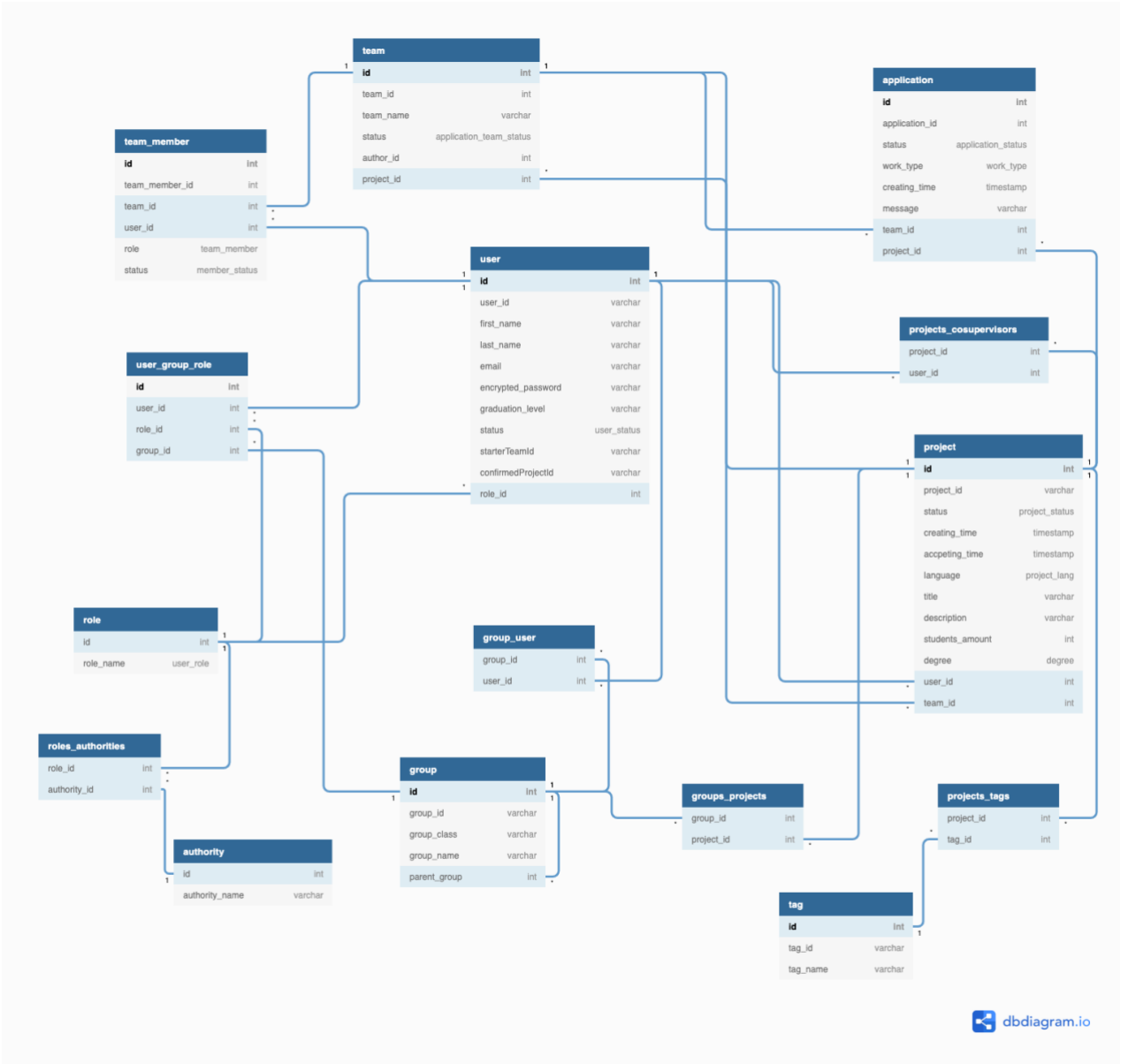

Figure 12. Database table diagram.

Appendix 2 – Gitlab CI/CD Pipeline config for front-end

```
stages:
  - build
  - deploy

build graduation-theses-management-service-front-end:
  stage: build
  image: node:latest
  cache:
    paths:
      - node_modules/
  artifacts:
    paths:
      - dist
  variables:
    api: localhost/api
  script:
    - npm ci
    - npm run build

deploy graduation-theses-management-service-front-end:
  stage: deploy
  only:
    refs:
      - master
  script:
    - mkdir -p ~/front-deployment/services/graduation
    - rm -rf ~/front-deployment/services/graduation/*
    - cp -r dist/. ~/front-deployment/services/graduation
```

Appendix 3 – Gitlab CI/CD Pipeline config for back-end

```
stages:
  - build
  - deploy

build ProjectManagementServiceBackEnd:
  image: maven:maven:3.6.3-jdk-11
  stage: build
  only:
      refs:
        - master
  script: "mvn install -B"
  artifacts:
    paths:
      - target/*.jar


deploy ProjectManagementServiceBackEnd:
  stage: deploy
  only:
    refs:
      - master
  script:
    - mkdir -p ~/api-deployment
    - rm -rf ~/api-deployment/*
    - cp -r target/*.jar ~/api-deployment
    - sudo service project_management restart
```

Appendix 4 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

We Oleg Kartašov, Andres Pajuste

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for our thesis “Web Application for Graduation Project Management”, supervised by Ago Luberg
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. We are aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. We confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

25.05.2021

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