TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

Mart Kaasik 193711IAIB
Kristjan Kõiv 193386IAIB

# GITLAB PROJECT METADATA ANALYSIS ECOSYSTEM

Bachelor Thesis

**Supervisor**
Ago Luberg
Ph D

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mart Kaasik 193711IAIB
Kristjan Kõiv 193386IAIB

# GITLABI PROJEKTIDE METAANDMETE ANALÜÜSI ÖKOSÜSTEEM

Bakalaureusetöö

**Juhendaja**
Ago Luberg
Ph D

Tallinn 2022

# Author’s declaration of originality

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

| | | |
|---|---|---|
| Author: | Mart Kaasik, Kristjan Kõiv | ..................................... |
| | | (signatures) |
| Date: | May 30th, 2022 | |

# Abstract

As part of this thesis, a system for managing team projects was built for Tallinn University of Technology. The purpose of the system is to give an overview of different teams functioning in a course. The application aims to track, analyse and give an overview of changes happening in GitLab projects, to alert the project manager of early warning signs for early project failure prevention.

It gives visualizations for different levels of detail to drill down on the real bottlenecks and issues. It also functions as a platform for Teaching Assistants (TA) to enter their assessments, based on the data visualizations provided by the app. These assessments along with the visualizations can then be audited by the project manager.

The system was built around periodically fetching team project group data from GitLab servers. After curating and storing the data in a logical way, the dataset is explorable from a web applications user interface.

The software consist of the backend and frontend layer. The backend was written in Python using the Django REST framework. The frontend was written in JavaScript using Vue progressive frontend framework and D3js graphics library.

The thesis is in English and contains 49 pages of content, 9 chapters, 17 figures and 9 tables.

# Annotatsioon

Käesoleva bakalaureusetöö raames on tehtud Tallinna Tehnikaülikooli jaoks tiimiprojektide haldamise süsteem. Süsteemi eesmärk on anda kursuse raames ülevaade erinevate tiimide edenemisest. Rakendus püüab jälgida, analüüsida ning anda ülevaate muutustest, mis toimuvad GitLabis olevates projektides, et projektide haldajale teada anda varajastest murekohtadest ning vältida projektide ebaõnnestumist.

Rakendus pakub erineva detailsuse tasemega visualisatsioone, et võimalikult kergesti leida üles probleemid. See funktsioneerib ka platvormina, kus abiõppejõud saavad anda visualisatsioonide põhjal tiimidele hinnanguid. Õppejõud saab seejärel vaadata üle nii abiõppejõudude hinnangud, kui ka andmed, mille põhjal need pandi.

Süsteem päribperioodiliselt andmeid tiimiprojektide kohta GitLabi serveritest. Peale andmete vastu võtmist ning loogilisel viisil hoiustamist, saavad kasutajad neid vaadata läbi veebirakenduse.

Tarkvara koosneb tagarakendusest ning esirakendusest. Tagarakendus kirjutati Pythonis kasutades Django raamistikku. Esirakendus kirjutati JavaScriptis kasutades Vue raamistikku ning D3js graafikateeki.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 49 leheküljel, 9 peatükki, 17 joonist, 9 tabelit.

# List of abbreviations and terms

| | |
|---|---|
| API | Application Programming Interface |
| CI/CD | Continuous Integration and Continuous Development |
| DB | Database |
| CSS | Cascading Style sheets |
| CSV | Comma Separated Values |
| D3 | Data-Driven Documents |
| ERD | Entity Relationship Diagram |
| GMT | Git Time Metric |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| IBM | International Business Machines Corporation |
| JSON | JavaScript Object Notation |
| JS | JavaScript |
| JWT | JSON Web Token |
| NIST | National Institute of Standards and Technology |
| ORM | Object-Relational Mapping |
| PBKDF | Password-Based Key Derivation Function |
| REST | Representational State Transfer |
| SHA | Secure Hash Algorithm |
| SQL | Structured Query Language |
| TA | Teaching Assistant |
| XSS | Cross-Site Scripting |

# Table of Contents

# List of Figures

# List of Tables

# 1. Introduction

It is no surprise that getting a team working on a hard problem is a non-trivial problem in and of itself. The IT degree students learn to work almost every semester in a team environment. Each course has its own grading and assessing criteria. But what use is a criteria if no one knows if anybody adheres to them? There has been no system to illuminate and differentiate the causes from the effects of different mentoring strategies. Presenting reliable data to the course mangers about the performance of the mentors and students is a first step in that direction.

Up until now many team project courses in the IT degree programs were managed in long and unwieldy spreadsheets. This solution works as a basic version of getting a indication of how well the course is performing. But much of the available data is not accounted for and thus the quality of education suffers. Since a spreadsheet is not very reliable or scaleable, a decision was made to build a custom solution with extra features that satisfies the needs of the client and enhances the quality of education provided.

Having an overview of how programming projects are performing is essential knowledge for the projects manager. It is a useful indicator if there is something wrong with the project management and if something needs to change. Furthermore, if the course is cyclical and happening every year at the same time (like many of the TalTech IT courses), then it gives a great perspective about whether the course is evolving or de-evolving in each revolution. The layering of different versions of the same system allows project designers to make informed decisions to recursively improve upon their own mentoring style and the course pathway as a whole.

In this bachelor thesis, a web application was developed that gets project management metadata from a GitLab group and enables teachers and mentors to explore the data and evaluate students.

## 1.1 Iteration zero of the application

The beginning of development for this system can be considered to be a Python script written by Kristjan Kõiv during the autumn semester of 2020.

The script was supposed to fetch data about team project from GitLab and generate a CSV from it. The script was found to be useful even then and further work was put into it in the spring and autumn semesters of 2021. By that time the script had quite a number of features, but at the same time had begun to grow in complexity.

In the end, the script had a text file that functioned as a configuration file, containing info about what GitLab group to fetch, what repositories are in what teams, what members are in what teams as well as what email addresses are associated with that members. The script then fetched info about spent time as well as number of issues and commits. It used this info to generate a CSV file, summarizing the efforts of teams during a manually entered time period. The script was used in almost every team project course in the informatics programme.

It was clear that the idea behind the script had potential, but the current version had become too unwieldy. A decision was then made to dedicate further resources to it, in the form of this bachelor's thesis. While none of the original script made it into the final product, it was a useful prototype, because it gave an idea of what to do, what not to do and how it could be improved.

## 1.2 Project scope

Since the primary client of the application is the informatics department at TalTech, most of the developed features are biased to reflect that. Notwithstanding, with little modifications and configurations, it should be possible to set up the software for personal and company use cases. Analyzing the past repository data where GitLab was used to track time and other project metrics, should be possible also.

The project scope encompasses four main parts. The objective of the first phase is to have a reliable endpoint with which to communicate with GitLab periodically. The next phase is to collect and store the data in an accessible format in the servers. The goal of the third phase is to visualize and display the data in an explorable way. Once the data can be seen and interacted with, the fourth phase aims to give the project managers, mentors and students objective data, which can be used to plan a way forward.

It was discussed with the client if the solution should be data-layer agnostic and support fetching data from sources other than GitLab. But since it would mean more work and the university uses only GitLab as their project management version control system supporting other Git servers like GitHub [1], Gitea [2] or BitBucket [3] was left out of the scope of this project.

[1]GitHub
[2]Gitea
[3]BitBucket

# 2. Development Methodologies

Methodologies are a set of working methods by those who work in a discipline [1]. The project development has adapted some features from the DevOps and agile practices.

Prior to starting work on actual development, responsibilities were divided and rules set for the development process. It was decided early on that Mart Kaasik would primarily work on the frontend and Kristjan Kõiv would primarily work on the backend. Both parties dealt with DevOps.

It was decided that TalTech GitLab servers would be used for project management and DevOps pipelines. TalTech servers were also used for hosting the live and development environments. Work was done in sprints and progress documented on a dedicated wiki page with the reports of the milestone retrospectives [1].

Discord [2] was used for all communication among developers as well as with the mentor and the client. This choice was made because all parties were already familiar with it and used it on a daily basis.

## 2.1 Core Values

Values add the reasoning behind the actions dictated by the methodology. The development of the application follows these values:

- Automation
- Feedback
- Sustainability

### 2.1.1 Automation with the DevOps lifecycle

DevOps engineering is vital for engineering teams, especially when the product matures. The core idea of DevOps is to automate as much of the development lifecycle as possible,

[1]Cognate wiki
[2]Discord

by integrating the work of the developer and operations manager into a closed loop of software development phases [2]. The phases are illustrated on figure 1

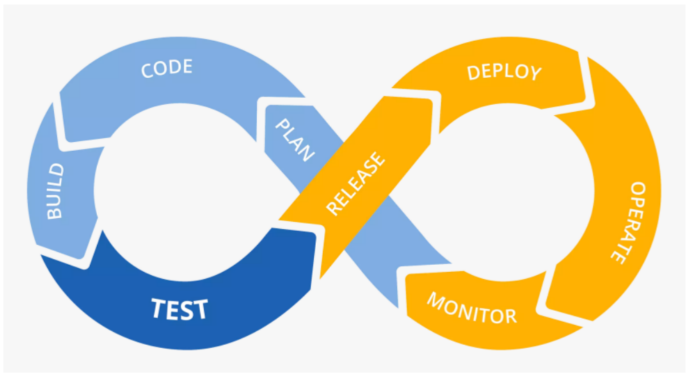


Figure 1. The DevOps lifecycle of development

Measures were taken to ensure that a revolution of the DevOps cycle would be completed each week. Starting with planning and ending with testing. The operations aspect of the cycle was mostly automated by GitLab pipelines [3].

### 2.1.2 User feedback

Since the app is primarily focused on the user and their experience, the technical backend design is not as complex as the frontend. Much more focus was put on testing the user interface and getting constant feedback from the end user. The end user can send feedback on the new features they think would enhance their and other users experience. They can also send bug reports if any bugs are found. The feedback that was submitted was fed back into the planning phase of the next sprint.

### 2.1.3 Sustainability

Developing software is not a straightforward process, it is a complex one, during which some functionalities clarify themselves as the developers accumulate domain knowledge and technical abilities. Usually the best strategy to develop the application, and what could have been done better, reveals itself to the software engineers at the end of the development. The inefficiencies baked in the source code out of developer negligence or laziness is known as *technical dept* that has to be payed with the time and effort of the future developers.[3]

Since starting over from scratch is not oftentimes feasible, source code refactorization has

[3]GitLab pipelines

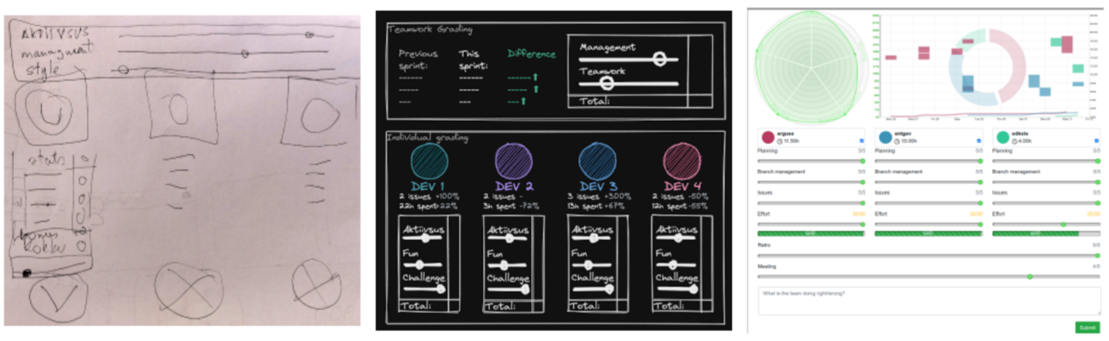
to be relied on. The purpose of that is to increase the software's quality and assure that the code reflects the current experts' knowledge as closely as possible. The long term effect of it is internal consistency and increased software stability. Refactorization left undone can have devastating effects on the software development cycle on the whole. It can slow down delivery of new features and it requires more resource-draining maintenance. [3]

Bearing these risks in mind, a conscious effort was put in on heavily documenting the development process and the source code as well as providing tutorials and official guides used by the developers when building the software. The list of most useful and relevant development resources is posted in the project wiki [4]. Significant work was put towards making the project more robust, so that it could continue to thrive even after the original developers have left the project.

## 2.2 Planning and the Design Process

The design process begun by extensive interviews with the client and their needs. Then it was possible to brainstorm the user stories and use cases. After imagining all the cases the product could be used for, the list of use cases was sieved through to generate feature sets for each important use case description.

Following the aforementioned feature requirements, the first pencil and paper sketches were drafted, which are illustrated in figure 2a. After consulting with the client, higher fidelity wire-frames were designed to consummate the idea of what will be built. The wire-frames are illustrated in figure 2b. All the wire-frames were made with online sketching tool Excalidraw [5].The wire-frames give the general idea of the final website views that will be built. The final views are showed in figure 2c.



(a) Initial pencil sketch (b) Refined wireframe (c) Finalized view

Figure 2. The design process of generating views from feature requests

[4]Cognate wiki
[5]Excalidraw

The wire-frames were used to build a data model and the user interface design. After the backend developer and frontend designer have agreed on the REST endpoint design, the implementation phase could start. The backend engineer begun work on building and testing the endpoints. The frontend engineer started to construct the frontend views and components. The outcome of the week long sprint was assessed, feedback from the client and the users was processed, the operations pipelines were deployed and the next sprint was planned, renewing the cycle again.

# 3. Project description

This section will cover the main use case and secondary use cases the software provides as well as analysis of existing solutions and their shortcomings.

## 3.1 Use cases

The main intended use case for the application is universities that want to get an overview of students progress as a course progresses. There are other potential categories of users as well who have slightly different requirements. These categories of users are:

- Project metadata analysis for personal usage
- Project metadata analysis for companies - measuring the efficiency of developers in a company team
- Project metadata analysis for universities - measuring students' contribution on team programming subjects

### 3.1.1 Personal usage

The most basic use case is that of people using the application to analyze project metadata for their personal projects.

Personal usage has the following requirements:

- Setup should be as simple as possible
- It should be possible to run the application without a dedicated server
- The interface should be intuitive and the learning curve should be as small as possible

### 3.1.2 Companies

Companies might want to use this application to get an overview of various teams in the company.

Usage by a company has the following requirements:

- Project source code needs to stay secure
- The application needs to be able to handle large projects
- The application needs to allow for a flexible set of metrics to be tracked

### 3.1.3 Universities

The primary use case for the application is for managing the teams in university programming courses.

Usage by a university has the following requirements:

- The application needs to be able to handle large quantities of people in separate projects
- The application needs to allow for a flexible set of metrics to be tracked with custom thresholds for each metric
- The application needs to be able to combine data from automatic metrics and TAs effectively
- The application needs to be convenient to use

## 3.2 Existing solutions

Before starting work on the application, various already existing solutions were researched. The purpose of this was to get an overview of the limitations of current solutions.

### 3.2.1 Kristjan's Python Script

The solution that was used before this application, or the first iteration of this application depending on ones point of view, was a Python script that took the data from GitLab and generated a CSV file from that. The analysis part was roughly the same, but it was not very scalable and manually running it each milestone took considerable effort.

Many values in that script were hard-coded, so it only worked for a single course and only one person had the expertise to reconfigure it for a different course. Also, students and their teams had to be written in a text file at the start of a course and this was a tedious affair.

The entire workflow that had to be completed each milestone was as follows. First, the script had to be run with the correct dates, which were entered manually. After this it was necessary to manually specify teams that did not have exactly matching milestones and then the script ran again. Then a CSV file was created, which would have to be copied over to Google Sheets [1], where TAs would only get numerical summaries of issues, time spent etc. The TAs would then enter their grades there.

In conclusion, the Python script worked and did what it had to, but it was still tedious and difficult to scale. Despite that, the potential was apparent and a decision was made to further develop this idea, in the form of this application.

### 3.2.2 GitLab Ultimate

GitLab Ultimate [2] is the payed version of GitLab that comes with extra features for bigger teams of developers. In this subsection the authors analyse three components of that payed version that most resemble the functionality of the project discussed in this thesis.

#### GitLab Roadmap

GitLab Roadmap [3] is a view in that displays a timeline of a project as a Gantt diagram. This allows for epic issues to be viewed with completion percentages and start and end times, enabling users to get an overview of what state larger features are in.

#### GitLab Insights

Insights [4] is a feature that allows users to specify numerical parameters to track in their projects and then gives an overview of those parameters.

#### GitLab Value Stream Analytics

Value Stream Analytics [5] is a feature that analyses the number of deployments and time that issues stay in various stages of their lifecycle. It aims to identify bottlenecks in development and give and idea of how long it takes for a feature to go from being and idea to being implemented.

[1]Google sheets
[2]GitLab Ultimate
[3]GitLab Roadmap
[4]GitLab Insights
[5]GitLab Value Stream Analytics

### 3.2.3 Current workflow in universities

Based on the developers personal experiences, the student assessment requirements in universities can be divided into two broad categories. The first are courses where there is an automated tester and students' solutions are tested against this. These courses usually do not have teams and each students has his or her own solution. The second are courses with no automated tester and a less defined success criteria. These courses usually have teams of students and the aim of the course is to develop a single application during the course.

For the first type of courses, the usual process is such that the students push their code to GitLab, from there an automated tester fetches the code, evaluates it and based on the results enters the grades to Moodle.

For the second type of courses, the usual process is such that the students use GitLab for project management and time tracking and teachers look through the students repositories or application and give an assessment based on some predetermined functional requirements and / or the quality of project management. Different aspects of these grading criteria are then entered into a spreadsheet and then the spreadsheet shared with the students or grades entered to Moodle or ÕIS.

### 3.2.4 Current workflow in companies

Based on interviews with individuals who are currently working as developers in Estonian startups, the workflow in tech companies is usually as follows. Gitlab or GitHub is used for code storage. Project management and issue tracking can be in the same place as code storage, but more often it is in a separate application such as Jira [6] or Trello [7]. New code is then reviewed by several other developers before merging and closing the issue.

## 3.3 Analysis of requirements and existing solutions

The python script was a good starting point, but as discussed previously, it had significant limitations, which necessitated the development of a more scalable and convenient solution.

A large problem with GitLab Ultimate, is the fact that it is a paid feature. This is especially a problem for universities, because the pricing for GitLab Ultimate works on a per user

[6] Jira
[7] Trello

basis, making buying it for the entire university an inconsiderable option. This is an area, where this application would be significantly better, by aiming to give at least the same functionality, but completely free of charge. Another problem with GitLab Ultimate is the fact that it is more focused on analysing a single project and attempts to find ways of developing software better, but universities are interested in comparing several similar project with one another and finding ways of teaching students better.

While the application is not focused on the university courses with automated testing, it could still be used for those if a few improvements were made. After integrating the application with the automated tester and allowing students access to the application as well, the application could replicate all grading functionality currently present in Moodle.

For the university courses with team projects however, the improvements offered by this application are rather significant. It gives the teachers a more comfortable overview of project management than simply looking at GitLab, going as far as to give suggestions for assessments for some grading criteria. It offers an intuitive user interface for giving out final assessments. In the future it could give students direct access to these assessments as well. The authors hope that these improvements will allow the application to completely replace and improve upon spreadsheets when it comes to grading students in team projects as well as improve the quality, fairness and efficiency of the grading process.

There are several similarities between software development in university team project courses and software development in actual companies. Both methodologies use the same technologies for storing the code and both have some set of rules that need to be followed when managing projects. However there are significant differences as well. Universities are much more focused on validating the performance of the students, but companies usually vet the quality of their workers upon hiring and during the working process, focus is put on ensuring that the quality of the code is as good as possible.

Despite these differences the application could to some extent be used by companies as well, although not as well as by universities due to the previously discussed difference in goals. The first use case comes from the fact that when companies hire interns or junior developers, then there could also be a need for comparing the performances of different employees and ensuring that everyone are at roughly the same level of productivity. In that case this application can be expected to outperform the software whose main goal is to organize project management thanks to mostly the same reason why it can be expected to be useful in universities for evaluating students. In addition to that, even when dealing with experience developers, some companies might want to get a bit better overview of their employees performance, especially when the time spent on setup and maintenance of the application are close to zero. So while the application might need some additional features, such as automatic analysis of code reviews or functionality for issue management, the authors still believe that the groundwork has been set for an application that has the potential to see widespread use among companies as well.

# 4. Project design

The application is split into two primary components: the frontend and backend. The backend is responsible for syncing data with GitLab API and communicating with the frontend through endpoints. The frontend is responsible for displaying data to users and allowing user to interact with the backend.

To properly manage the components, it was decided to have the source codes for backend and frontend code on separate Git repositories. The separation of concerns makes the development time faster and easier, reduces the number and frequency of merge conflicts and makes building CI/CD pipelines effortless.

## 4.1 Backend

Backend is code that runs on a server and responds to requests from the client side. In this case the backend is also responsible for communicating with the GitLab API. The ways in which the backend is connected with other components is illustrated in figure 3.

Backend was built in Python, using the Django web framework [1]. The main reason for this, was the fact that both developers were already familiar with both Python and Django. Django also offered the Django REST framework library [2] , which was used to build the web API.

Other languages like Java and Rust were also considered, but Python was chosen because it allows for fast prototyping and very agile development, especially with the Django framework. Another reason for Python was its low barrier to entry, making it easier to onboard new developers to the project.

[1]Django

[2]Django REST

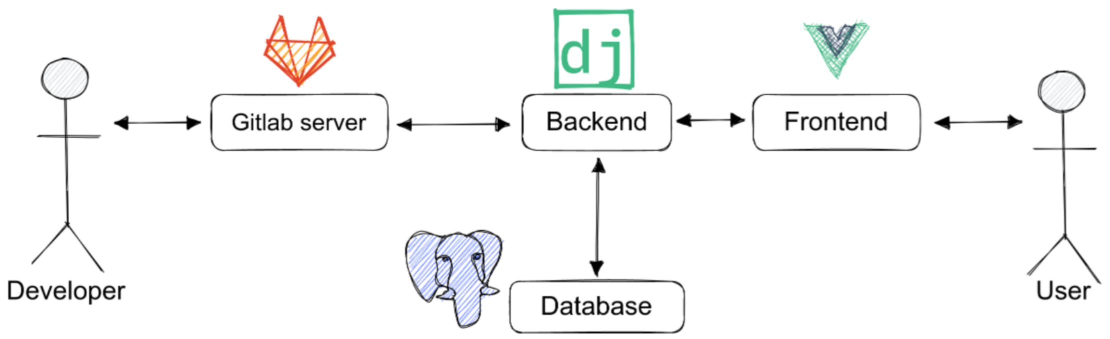


Figure 3. Data flow diagram

### 4.1.1 GitLab API

The application uses the GitLab API in order to get data about projects. Users provide their GitLab API keys and bind them to either their account or a specific group of project in the application. The application then uses these keys to periodically fetch data from GitLab servers.

The same general system could be used to fetch data from GitHub as well, but for this project only support for GitLab API was added, because it is used for university projects. Integration with other git server providers is discussed in chapter 9.

### 4.1.2 Database

The first thing to consider, when choosing which database to use, is the compatibility with the Django framework. This meant that in order to make the most of Django's powerful Object-Relational Mapper (ORM), it would be best to choose a database that is fully supported by Django.

Django officially supports five databases: PostgreSQL, MariaDB, MySQL, Oracle and SQLite. The official Django documentation states that PostgreSQL is the preferred database for Django web apps, so this was the first database that was further investigated. Because PostgreSQL is the database, that is most supported by Django, it provides support for a number of data types that only work with PostgreSQL, for example the JSONField, which was later used in the application.

From this it seemed that PostgreSQL was the correct choice and further investigation confirmed this. Oracle costs money and is closed source, so it was neither financially affordable nor compatible with the values of the developers. SQLite is a very basic database that only has five data types and this simply wasn't enough for this application. This left PostgreSQL, MariaDB and MySQL.

While there weren't significant limitations for MariaDB and MySQL over PostgreSQL, there were small reasons. For example, MariaDB is slightly slower than PostgreSQL [4]. In the end, there wasn't anything that MariaDB or MySQL would offer that PostgreSQL didn't, but PostgreSQL had support for a wider range of data types and was more supported by Django.

### 4.1.3 Database versioning

As with all other parts of software development, version control is an important aspect when it comes to versioning source code as well as databases. For normal code, the problems mostly arise when multiple developers are working on a project together. For databases that is also a problem, but there is also the problem of there being several databases in different environments. [5]

When a developer makes changes to a database schema that is necessary for a feature, the database on the developers computer will no longer be compatible with the database in a development or live server. A solution would be to copy the new database from the developers computer to the server, but this is a problem when there are several environments with the need to store different data in them. [5]

To fix this problem, a better solution would be to define all changes to a database programmatically. This means that the developer writes code that generates changes to a database schema without changing the underlying data any more than is necessary. [5]

At it's most basic version the code is a set of SQL statements, but Django offers a better solution in the form of Django migrations [3]. Because Django provides an ORM that is fully compatible with PostgreSQL, it is possible to generate these changes completely automatically. The developer changes the data model that is defined in the form of a Python file and from this Django generates another Python file called a migration. These migrations define the changes that need to be made to a database to transfer it from one version to another.

[3]Django migrations

### 4.1.4 ERD schema

A useful tool for planning the development of databases is an *Entity Relationship Diagram* (ERD). An ERD is a flowchart that illustrates what tables there are in a database, what attributes they have and what connections they possess with other tables. In this context "entity" and "table" are interchangeable terms, because a table defines a set of attributes and a row in a table usually defines the attributes for some "entity" in the real world. [6]

Database modeling is an important part of the software design process. Before implementing a new feature, it is important to look at how the data model needs to change, in order to accommodate this feature. To quote Frederick Brooks, a software architect from IBM: "Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious." [7] This is a philosophy that the authors very much agree with as well.

There are several tools for making ERDs. Initially, Rational Rose [4] was used, because it was the one that the developers were most familiar with. During development a decision was made to start using Visual Paradigm [5] instead because Rational Rose was a bit outdated. In order to get a good overview of all the tables that the application uses, the tables were all separated into different subsystems and ERD schemas made for each subsystem.

## 4.2 Frontend

Frontend is the interface our user interacts with. In our case it is a website that displays the GitLab management metadata for the user according to their privileges. It also has the components to define the configuration of how to divide the incoming data into different sprints or phases and what metrics to track. Along with fetching and configuring the course, the frontend enables to analyse all sprints for assessment, evaluation and comparison. This layering of concerns gives unique perspective for all the course participants.

- It gives the course management insights on the dynamic effectiveness of the team mentors. And gives them concrete data to learn upon their past courses unproductive configurations.
- It gives the mentors of the teams insights about the dynamic effectiveness of the developers and how their own pedagogy is helping or hindering their teams cogency.
- It gives the developers insights on how and in what aspects the course management want to see them improve.

[4]Rational Rose
[5]Visual Paradigm

VueJS [6] was used as a JavaScript framework to build our interface around. The progressive static SPA framework was chosen for the fact that we had prior experience with the it and thus spending great time on learning a new frontend framework was unnecessary.

### 4.2.1 Code design

The frontend code is written following the standards and style principles of VueJS community. For example the code is divided into multiple files/components to prevent repetition and to make each component easily replaceable.

All the visualization graphs and components are generic and parametric. Folder structure is given in Table 1

| Folder | Purpose |
|---|---|
| src/views | Views that are used only one time |
| src/components | Components that are reusable |
| src/components/visualizations | D3js visualization components |
| src/<files> | Vuex store, routes definition, main, axios configuration |
| <root> | GitLab CI/CD, Docker and nginx configuration, package.json, README.md |

Table 1. Cognate/front folder structure

### 4.2.2 User interface design

The user interface has to keep the needs and expectations of the end user in mind. It is the most important part of any web application. Users judge the design quickly and care more about usability and likeability. If the interface fails to deliver on expectation the user might not use the application again [8].

Large part of the interface consist of interactive data visualizations. The purpose of visualization is insight, not pictures. Graphics are a powerful interactive strategy for exploring data, especially when combined with interactive statistical methods. As the father of direct manipulation interfaces [9], Ben Shneiderman has said, "Like a telescope or microscope that increases your perceptual abilities, information visualization amplifies ones cognitive abilities to understand complex processes so as to support better decisions" [10]. The visuals aim to give:

[6] VueJS

- more effective detection of faulty data, missing data, unusual distributions, and anomalies;
- deeper and more thorough data analyses that produce profounder insights; and
- richer understandings that enable researchers to ask bolder questions.

All the graphs were made using the D3.js [7] data driven design library. Which is the most popular and well documented JavaScript library to build responsive and robust visualizations from scratch [11]. Other graphing and charting libraries were considered like Chart.js [8] and NVD3 [9], but these libraries would not give us much customizability. D3js gives us control over each pixel on the screen which is needed for displaying the data in a way that satisfies the requirements of the client.

A lot of emphasis was put into making the user interface elements as responsive and intuitive as possible. That means that graphs elements should react to mouse events organically and bring out the data that the user is interested in in a seamless immediate way. A lot of inspiration came from direct manipulation notion in information design of which the core principles are: [10]

- Continuous representations of the objects and actions of interest with meaningful visual metaphors
- Physical actions or presses of labeled interface objects (e.g., buttons, sliders, etc.) instead of complex syntax
- Rapid, incremental, reversible actions whose effects on the objects of interest are visible immediately.

Early in the development cycle a decision was made to use dark theme version [10] of the Bootstrap CSS framework [11]. This was purely an aesthetic choice made by the user interface designers tastes. Subsequent anecdotal evidence was later noted of users really liking the darkened color scheme. For accessibility and readability reasons an option to use lighter color scheme was also added. The lighter theme uses colours that are more easy to see even for people with colour blindness.

[7]D3.js
[8]Chart.js
[9]NVD3
[10]Bootstrap dark
[11]Bootstrap

### 4.2.3 State management

As the application grew in complexity and multiple pieces of state scattered across many components the need for a state management system grew. Vue offeres its own state management library vuex [12].

Pinia and Redux was also considered. Altough Redux is view-layer agnostic and can easily be integrated with Vue, it is more popular and better supported with ReactJS communitiy. The differences between vuex and Pinia were negligible and virtually offer the same functionality. The developers could have picked either of those, but vuex includes VueDevtools [13] integration which made debugging considerably more efficient.

## 4.3 Security

The application holds sensitive information about clients projects and therefore measures had to be taken to make sure, that this data stays secure. This meant ensuring that in the case of a database leak, users passwords and gitlab tokens would stay secret.

### 4.3.1 Communication

The first step in ensuring that a web application is safe to use, is preventing man-in-the-middle attacks. For this Hypertext Transfer Protocol Secure (HTTPS) was used, which encrypts all HTTP requests [12].

### 4.3.2 Managing passwords and authentication

Passwords are stored using the default standards for Django. This means using the Password-Based Key Derivation Function 2 (PBKDF) [13] algorithm with a SHA256 (Secure Hash Algorithm) hash. PBKDF involves salting and repeatedly hashing the password, thus making the resulting hashes resistant against rainbow attacks and dictionary attacks respectively [14]. This approach is compatible with the recommendations of the National Institute of Standards and Technology (NIST) [15]. After a user has logged in, a JSON Web Token (JWT), which is an industry standard, was used for managing claims about users [16].

[12]VueX
[13]VueDevTools

### 4.3.3 Managing gitlab tokens

For most applications the password is the most important piece of data, but for this application the gitlab tokens tied to users or groups are perhaps even more important, because they allow for access to a users gitlab account. The secure storage of gitlab tokens is a difficult problem due to having several different requirements:

1. the token should be unrecoverable in case of a database leak
2. the application needs to have access to the token when syncing data with gitlab
3. the application should not bother the user by repeatedly asking for his or her password or token
4. a users gitlab token should remain relatively uncompromised in the case of the users password being compromised

A multilayered approach was taken to satisfy all of these requirements in an acceptable manner.

The application has two types of GitLab tokens. The first kind is tied to a user and the second to a group. For encrypting a token tied to a user, that users password can be used. The same approach can not be used for tokens tied to a group however, because a group does not have a password that all users would know. Due to this, the tokens tied to a group are not encrypted unfortunately. However they are not vital for using the application, but simply make it more comfortable, so the option to use them is left in, but an appropriate warning is displayed to the users.

In order to solve the first requirement, the token is stored in the database in an encrypted format. Fernet [14], an implementation of symmetric authenticated cryptography, is used for encryption. First a secret key is generated from the users plain text password and a salt. The secret key is then used to encrypt the token and the key itself is not stored in the database, meaning that even after getting access to the database, it is impossible to gain access to the gitlab token.

The previous strategy immediately generates the second requirement, meaning that when the database only has the encrypted token, then it can not use it to fetch data from gitlab. In order to gain access to the plain text token, the users password is required. This password is provided by the frontend application when the user makes a request to sync data with gitlab. Using the users plain text password, the backend application then regenerates the secret key (using the same salt, which was stored in the database in the meantime), decrypts

[14]Fernet

the token and makes the request to the GitLab API.

The previous strategy then in turn generates the second requirement, meaning that the frontend needs to be able to send the password when a user wants to sync data with GitLab, but it would be bothersome for the user to enter the password each time he or she wants to make that request. For this reason the frontend stores the users password in local storage and sends it to the backend when the user wants to sync data with GitLab. Storing sensitive data in local storage is relatively safe as long as the site in question is protected against cross-site scripting (XSS) attacks, which to the best of the authors knowledge, this site is [17]. However because this could still be considered a small security risk, then it was decided that storing passwords in this way would be an opt-in feature and the default behaviour is asking the user for the password each time that he or she makes a request to sync data with GitLab.

To avoid a single point of failure situation, it would be good if the gitlab token stayed secure even when the users account was compromised. For this reason the application never sends the token back to the user, even when given the correct password. This means that a malicious actor who gains access to a users account will only be able to refresh the data in the application with data from gitlab, because this is the only operation that the backend does with the gitlab token.

As a final mitigatory measure, GitLab access tokens can be configured to have read-only rights. This way even if the access tokens get into the hands of malicious actors, they will not be able to perform harmful actions to the repository. Instructions about how to configure the access tokens properly for the applications needs are given to the users.

Finally a second option is offered to the user which does not involve sharing his or her access token at all. A separate account for the application is created in the organizations GitLab and that account is given access to whatever projects need to be tracked. An access token (with read-only rights) for that account is then saved in an environment variable and used for fetching data, if the account has access to a project. The upside of this approach is the fact that even when malicious actors get access to the token, then only the projects that users specified are compromised, instead of all projects owned by a user.

# 5. Backend contents

The backend is an important component of the project, as it communicates with all other components. The following section describes the inner workings of the backend and what endpoints can be used to communicate with it.

## 5.1 Subsystems

The tables in the backend were divided into conceptually distinct subsystems. The following gives an overview of each subsystem, along with an ERD. Each subsystem has it's own ERD, containing all of the info about tables in that ERD and connections with other tables from other subsystems.

Boxes on the ERD translate to tables in a database. There are 2 different colours of boxes. Light blue means that it is a table from the current subsystem. Light gray means that it is a table from another subsystem.

For the tables that are from the current subsystem, there is info about the name of the entity that it describes, the attributes that is has and the connections that is has to tables from other subsystems. For the tables that are not from the current subsystem, there is only info about the name of the entity that it describes and it's connections to tables from the current subsystems.

### 5.1.1 Classificators

Classificators are used whenever an entity has a field, that can have as a value, only one of a concrete set of possibilities. These sets of possibilities can then be formed into tables of their own and an entity that needs to have a property set, can then have a link to this table.

An ERD describing all classificators can be seen in figure 4. Each classificator has a key and name. A key is a short (1 or 2 letters) string, that is the value that is used for referencing a classificator. A name is a longer string that describes what this option means for the given classificator.

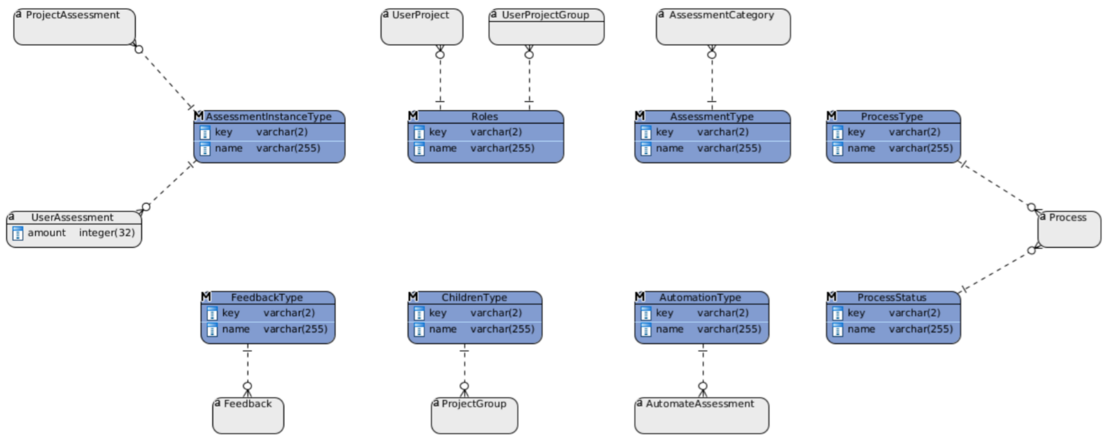


Figure 4. Classificator subsystem ERD

### 5.1.2 User accounts

Django has a built-in support for managing users in the form of a table that contains a username, name, password, email and other things. This built-in functionality was used, along with some custom modifications. On the ERD, shown in figure 5, the table named `User` references this built-in table.

It was necessary to save additional info about users, such as a users GitLab token and whether or not it is an account that is actually used by someone, or generated by the system. For this purpose a profile table was made, that has a one-to-one relationship with the user table.

Because the metadata that users attach to their commits might not match the data associated with their user, it was necessary to add a `Committer` table. This table contains a name and email field. When reading commits from an API, each commit is connected with a committer based on the name and email given in the commit. These committers can then in turn be tied to accounts on the system, allowing for several commit profiles to be tied with one account.

The `Profile` table shown on figure 5 contains an identifier, which is a unique randomly generated string that is used when inviting new `User` s to join a `ProjectGroup`. The reason that a randomly generated string is used instead of the `User` objects id is to avoid `ProjectGroup` owners spamming random users with invitations. The `Profile` table also contains information needed to encrypt and decrypt the GitLab token for that `Profile`.

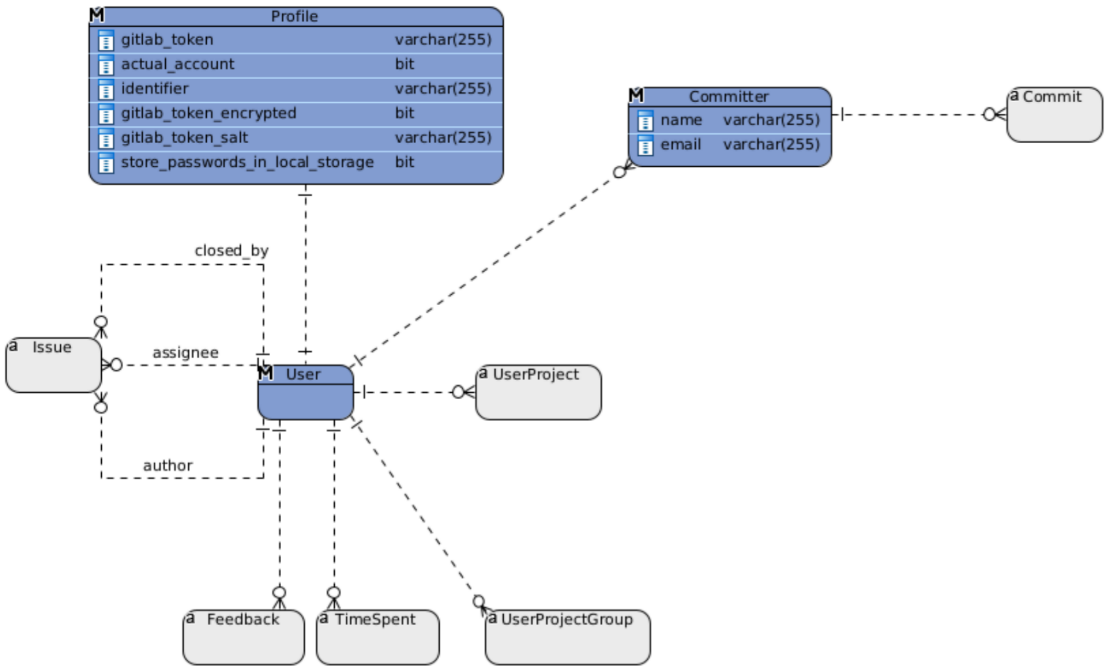


Figure 5. User subsystem ERD

### 5.1.3 Group structure

The group structure subsystem, along with the user subsystem, define the highest organizational layer of data in the application. This means that all data in the system can eventually be tied to a specific group or project. An ERD describing the group structure subsystem can be found in figure 6.

A project is exactly what it commonly means in the context of software development. It has a number of repositories and users, which in this case are connected with projects using `UserProject` objects.

A `Project` can belong to a `ProjectGroup`, which in turn can belong to other `ProjectGroup`s. This allows for a flexible, hierarchical structure to be defined for managing a large number of projects.

In the case of a university, a `ProjectGroup` could be all of the projects in a single course or all of the projects under a single mentor. In the case of company use, it could be all of the projects under one team or department.

A `ProjectGroup` can have a Gitlab access token attached to it. If such a token is attached, then this token is used when making requests to the Gitlab API, not the token attached to the user making the request. The `group_id` attribute of ProjectGroup describes the Gitlab group id, that a given `ProjectGroup` is based on.

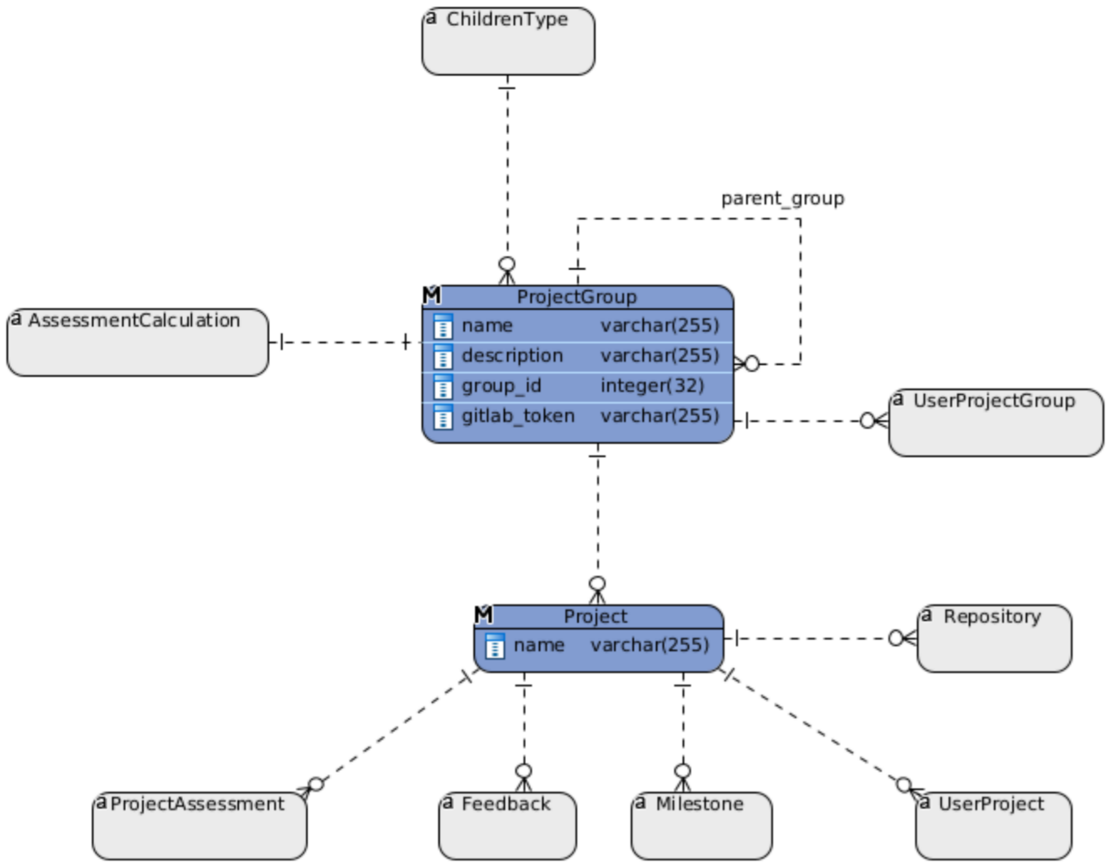


Figure 6. Group structure subsystem ERD

### 5.1.4 Roles

All roles in the application are tied to a group or project in the form of either a `UserProjectGroup` or `UserProject`. `UserProjectGroup`s connect `User`s with `ProjectGroup`s and `UserProject`s connect `User`s with `Project`s. An ERD with `UserProjectGroup` and `UserProject` tables can be found in figure 7.

A user that creates a group is by default the `OWNER` of that group. The `OWNER` can then add other users to that group as either `OWNER`, `ADMIN` or `VIEWER`. A single user can have multiple roles in a single group.

When a new project is added to a group and the application takes project metadata through the GitLab API, then repository members from GitLab are automatically tied to application users and are given `VIEWER` roles. Users with a `OWNER` or `ADMIN` role in a group can assign `OWNER`, `ADMIN`, `MENTOR`, `TEACHER` and `VIEWER` roles to users in a project in that group.

In order to create a `UserProjectGroup` that is tied to a specific `User`, that `User` must give his or her identifier to a `OWNER` or `ADMIN` in a `ProjectGroup`. The `OWNER` or `ADMIN` must then create a new `ProjectGroupInvitation` object and once the `User` accepts, he or she will have a `UserProjectGroup` connecting him or her to the `ProjectGroup`.

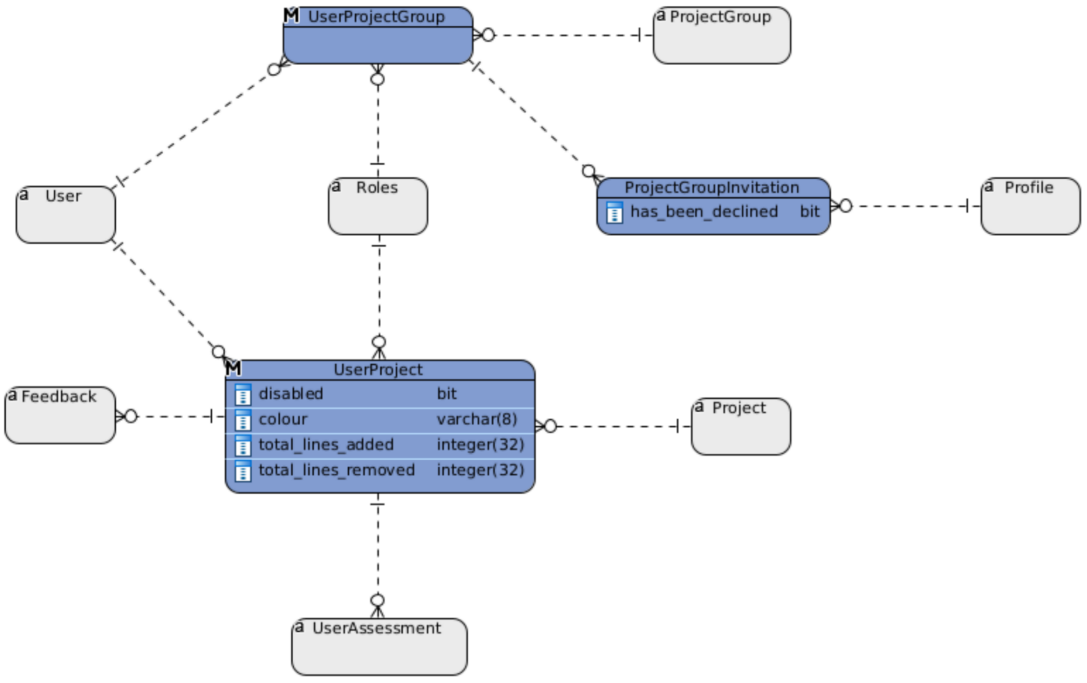


Figure 7. Roles subsystem ERD

### 5.1.5 GitLab API

An important part of the application is fetching project metadata from GitLab. For this there is a separate modular component, that defines logic for fetching info from a specific GitLab repository using an API token.

Entities in this subsystem are essentially copies of entities in GitLab's data system. However only a small part of all entities from GitLab's data system were chosen, because these were the ones, that were most critical for analysing developer performance. The chosen entities and fields, along with their connections to other subsystems can be found on figure 8.

From this ERD, it is apparent how data from GitLab is connected with this application's high level data structures. The `Repository` and `Milestone` objects are connected with the group structure subsystem through projects. The `Commit`, `Issue` and `TimeSpent` objects are connected with the user subsystem through the `User` and `Committer` objects.

When data in the applications database is synced with data from the Gitlab API, it is important to be efficient about it. Fetching commits, issues and milestones in itself was not a problem, because there were endpoints for asking for them in bulk. Fetching data about users spent time was an issue however, because this involved sending a separate request for each issue. To avoid having to do this for every issue, an attribute called `last_issue_sync` was added to the `Repository` object. This attribute holds the time of the last sync and the Gitlab API allows for fetching only issues that have been modified after a certain time. This significantly sped up subsequent sync queries, because it meant that a request for spent time had to be sent for only the issues that had been modified since last time.

A similar optimization was made for commits. Commits have the attributes `lines_added` and `lines_removed` and there are endpoints that query for total amount of lines added or removed for each user. To avoid having to recalculate this each time, there were attributes added to `UserProject` that held a cache of total lines added and removed and a boolean attribute called `counter_in_user_project_total` added to the `Commit` table, that shows if the commit has already been counted in that cache.

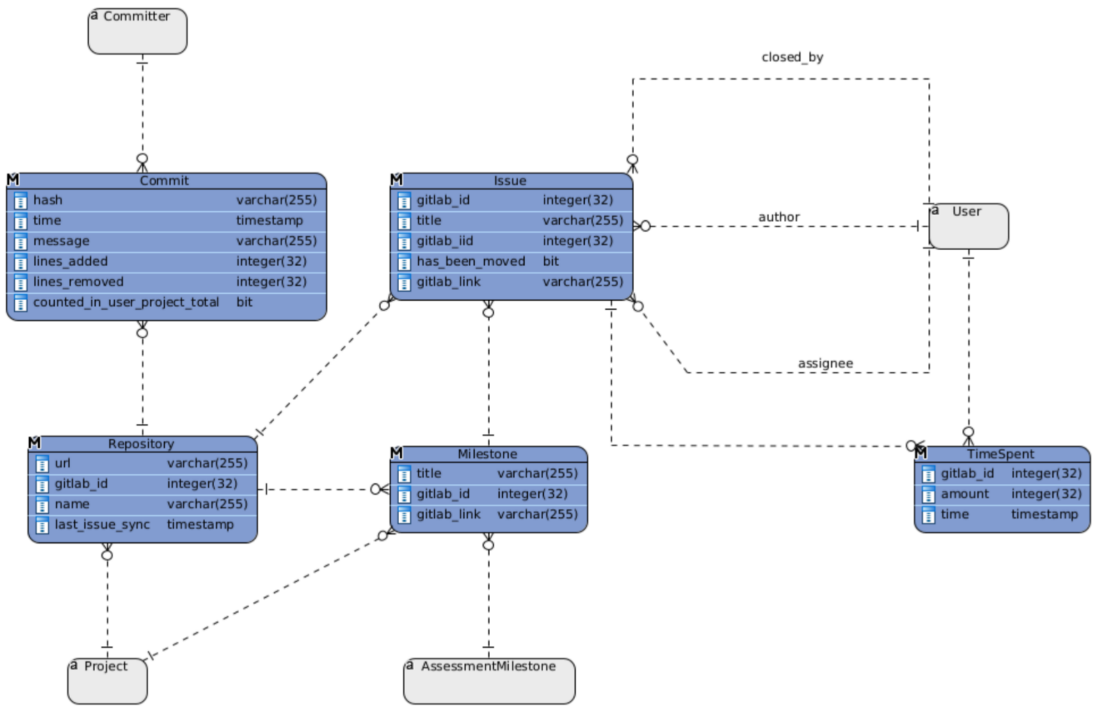


Figure 8. GitLab subsystem ERD

### 5.1.6 Assessment system

For universities and companies it is important that they be able to set measurable metrics for projects and target values for those metrics. Companies and universities would use the result of this analysis for getting an overview of employee or student performance which could then be used as part of the decision process when assigning salaries or grades.

In order to fulfill this purpose the application has a subsystem that allows group owners and administrators to define these metrics, get an overview of them and then allow for supervisors or mentors to give their input.

This subsystem is in itself large enough, that it warrants a further subdivision. Firstly, there is the so called assessment system subsystem, which enables group administrators to define rules for assessing developers in that group. Secondly, there is the so called assessment subsystem, which handles the assignment of assessments to developers and projects. The ERD for the assessment system subsystem is in figure 9 and the ERD for the assessment subsystem is in figure 10.

The most important table of the assessment system subsystem is the `AssessmentCategory`. This has a connection to itself, allowing for a tree like structure to be generated from several `AssessmentCategory` objects. Each `AssessmentCategory` has a numerical attribute "total", that describes the maximum number of points, that it is possible to get for that `AssessmentCategory`.

Each `AssessmentCategory` allows for a different source of assessment to be defined through the `AssessmentType` classificator. The possible values for that classificator are `CUSTOM`, `SUM`, `MIN`, `MAX` and `AUTOMATIC`. For an `AssessmentCategory` of type `CUSTOM`, the source of the assessment is another user, usually a mentor of some kind. For the `SUM`, `MIN` and `MAX` types, the source is child assessments and the specific numerical value is obtained through the mathematical function that shares a name with the types. For an `AssessmentCategory` of type `AUTOMATIC`, the source of the assessment is the application itself and in that case the `AssessmentCategory` has an `AutomateAssessment` entity connected to it, defining the way that the numerical value for the assessment should be calculated.

Some `AssessmentCategory` objects have `AssessmentMilestone` entities tied to them. These entities have a start and end time, as well as a order number. They also allow for a set of milestones from repositories to be connected with them. When an `AssessmentMilestone` is tied to an `AssessmentCategory` in this way, then it means that only data from this time period and / or these milestones should be taken into account when assessing this category or it's children.

The `AutomateAssessment` is connected with an `AutomationType`, which defines which algorithm to use when evaluating the `AssessmentCategory` that the `AutomateAssessment` is connected with. It also has a single attribute, that defines some numerical value that the developers must have reached for the algorithm to give the developers a good result. For example one algorithm looks at the amount of lines added, and in that case the amount needed attribute would describe how many lines are needed for the algorithm to give the maximum result. The algorithms usually require a specific time period or subset of milestones to look at, when evaluating developers. This is given to the algorithm through the `AssessmentMilestone` object. Specifically it starts from the `AssessmentCategory` that is connected with the `AutomateAssessment` and moves toward the root of the tree that is formed from `AssessmentCategory` objects, until it finds an `AssessmentCategory` that is connected with an `AssessmentMilestone`. If there is no such `AssessmentMilestone`, it takes the entire project as the time period.

`AssessmentCategory` also has a boolean attribute called `project_grade`, which specifies, if assessments in this category should be tied to individual developers or to projects. Being able to have an assessment that is tied directly to a project is useful whenever there is a need to compare projects to each other, not only developers in a project and as such, it is needed for both universities and companies.

Because the assessment system is defined per `ProjectGroup`, then it needs some connection to it. This is what `AssessmentCalculation` is for. It is a link between an `AssessmentCategory` and a `ProjectGroup`. The `AssessmentCategory` that it links to, can be considered as the root of the assessment tree for that `ProjectGroup`.

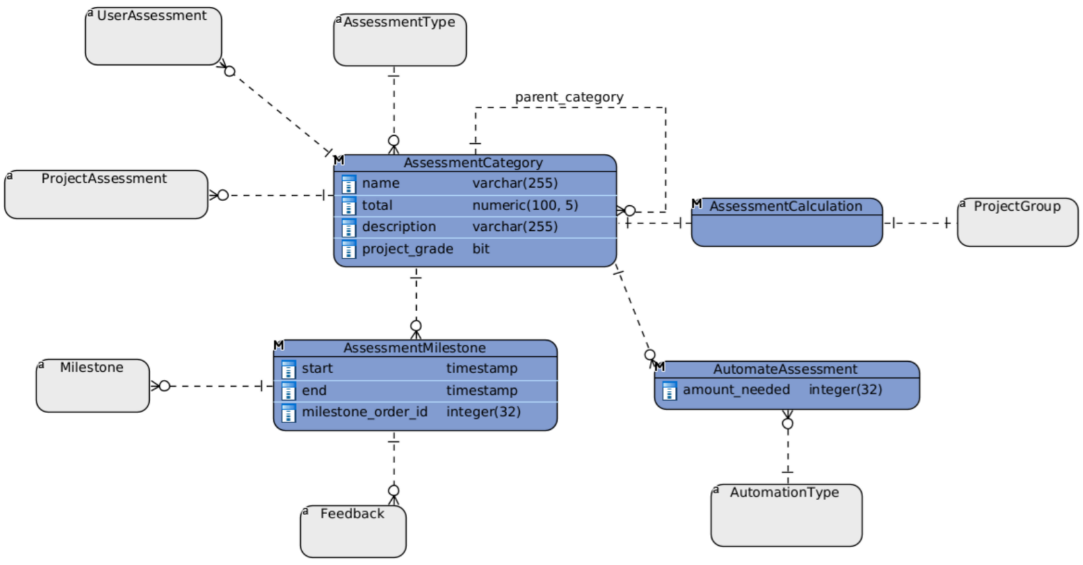


Figure 9. Assessment system subsystem ERD

The second subdivision of assessment system, named the assessment subsystem, manages the assigning of assessments to users and projects.

The main entities in this subsystem are quite similar. They both have a single numerical attribute called `amount` that is used to describe how many points the user or project got for it. The value for `amount` should be in the range from 0 to `total`, where `total` is the attribute on the associated `AssessmentCategory`. Both are connected to an `AssessmentCategory` and then to either a `UserProject` or a `Project`, depending on whether the assessment in question is a `UserAssessment` or a `ProjectAssessment`.

Both also have a connection with a classifier that defines whether the assessment is calculated automatically by the system, given by a mentor or simply a placeholder. This is a distinct concept from the `AutomateAssessment` entity from the assessment system subsystem. The assessment system subsystem defines which `AssessmentCategory` objects have the capability to be calculated automatically, but mentors always have the ability to manually overwrite those assessments, in which case the assessment will be marked as manually assessed. Being a placeholder assessment means that it has not been graded by neither a mentor or the system. It is still useful to generate entities in this case, because other parts of the system might assume that assessments always exist for every valid combination of user and `AssessmentCategory`.

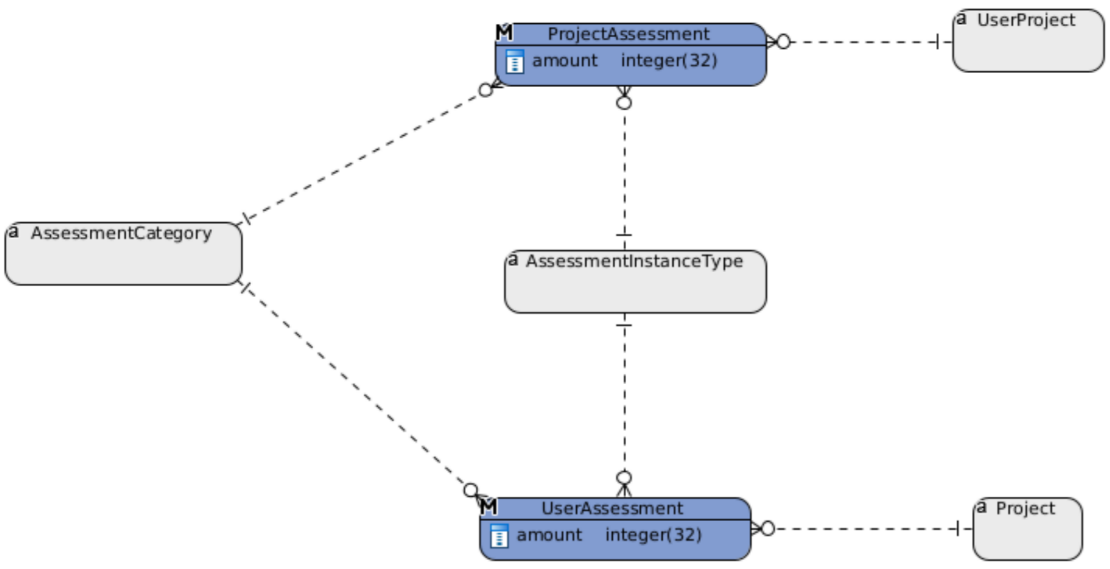


Figure 10. Assessment subsystem ERD

### 5.1.7 Other components

There are two other components, which are not complete subsystems in their own right, but in terms of database design, they do not fit into any existing subsystem either.

The first of these two, is the `Process`, shown in figure 11. There are a few endpoints that trigger processes that take a significant amount of time. For example the endpoint that syncs an entire `ProjectGroup` with GitLab. For these cases the endpoint triggers a separate thread that starts to carry out the process, creates a `Process` object and returns the hash of the `Process` object immediately. The thread periodically updates the `Process` object and the frontend can use a different endpoint to query the status of the `Process`.

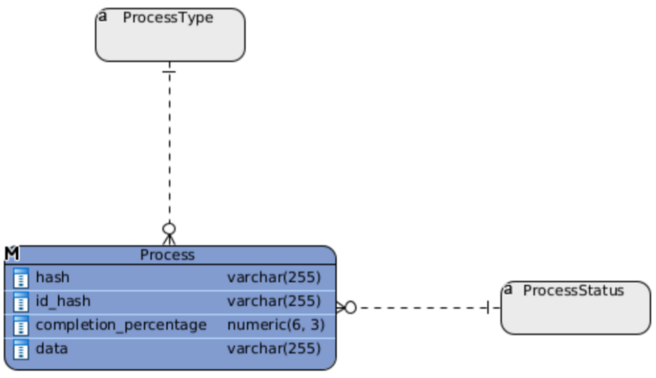


Figure 11. Process subsystem ERD

The second is the `Feedback` object, shown on figure 12. This is a multipurpose object used for freeform textual data. It has 5 different types: `APPLICATION`, `PROJECT`, `PROJECT_MILESTONE`, `USER` and `USER_MILESTONE`. When the type is `APPLICATION`, then it is feedback, that is given to the application itself and is anonymous. When the type is either `PROJECT` or `USER`, then it is feedback given by the mentor, about a project or a user. When the type is `PROJECT_MILESTONE` or `USER_MILESTONE`, then it is feedback given in regards to a single milestone and regarding either the entire team or a single user in it.

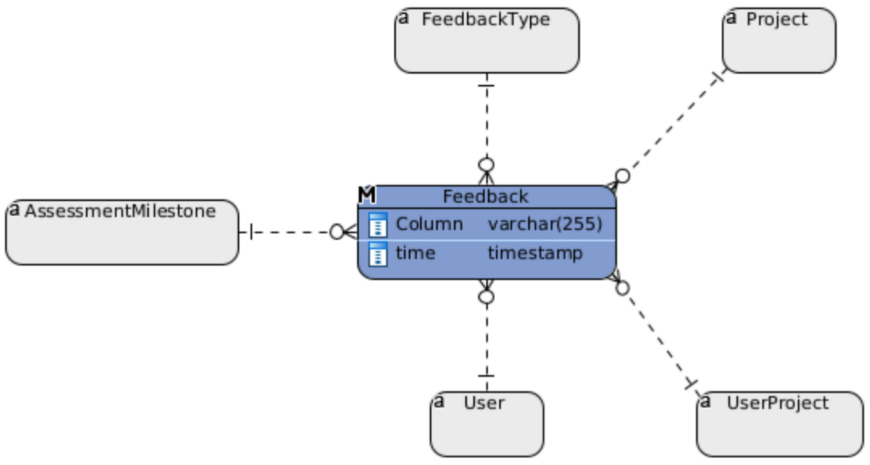


Figure 12. Feedback subsystem ERD

## 5.2 Endpoints

The frontend communicates with the backend using REST API requests.

For the group and project level endpoints, the roles column refers to what role the user needs to have in that group or project to be able to access that endpoint. The roles are abbreviated as `O`, `A`, `T`, `M` and `V` meaning `OWNER`, `ADMIN`, `TEACHER`, `MENTOR` and `VIEWER` respectively.

### 5.2.1 Group level endpoints

A large part of resources are tied to a specific group. The following section contains endpoints that manipulate the data, that is connected to a group in some way. All slugs in table 2 begin with `/groups/<int:id>`, so those have been omitted from the beginnings, to make the slugs fit better. The id in the slugs refers to the id of the group.

| Slug | Type | Roles | Purpose |
|---|---|---|---|
| / | GET | OATMV | Get info for all projects in group |
| / | PUT | OA | Modify group name, token and description |
| /users/ | GET | OA | Get all users in group |
| /users/ | POST | OA | Assign a new group role to a user in a group |
| /users/ | DELETE | OA | Remove a role from a user in a group |
| /project/ | POST | OATM | Create new project in group |
| /project_repo_connections/ | GET | OA | Get project and repo connections in group |
| /assessment/ | GET | OATM | Get assessment system for group |
| /milestone/<int:milestone_id>/ | GET | OATMV | Get summary table for group |
| /invitations/ | GET | OA | Get all pending invitations for group |
| /invitations/ | POST | OA | Create new invitation for group |
| /invitations/ | DELETE | OA | Delete pending invitation for group |
| /accept_invitation/ | POST | - | Accept or decline an invitation (accessible by users who have a pending invitation) |
| /assessment$_csv/$ | POST | OA | Get summary of all assessments in group in CSV format |

Table 2. Group level endpoints

### Assessment system endpoints

The endpoints that manipulate the data in the assessment system do not have a direct reference to a group. Instead the id parameters here refer to the id of the assessment category object, that they interact with. This can still be considered a part of the group level endpoints, because the assessment category objects make up a tree, whose root is connected with a specific group. All slugs in table 3 begin with `/assessment_category/<int:id>`, so those have been omitted from the beginnings, to make the slugs fit better.

| Slug | Type | Roles | Purpose |
|---|---|---|---|
| / | POST | OA | Add new node to assessment tree |
| / | DELETE | OA | Delete node from assessment tree |
| / | PUT | OA | Modify existing node in assessment tree |
| /copy/ | POST | OA | Create a copy of an existing node in assessment tree |
| /recalculate/ | GET | OATM | Recalculate all assessments from this node down |

Table 3. Assessment system endpoints

**Gitlab endpoints**

The endpoints in table 4 are also a part of the group level endpoints and the id parameters here refer to the group id, as usual. But it is worth it to bring these out in a separate table, because they are special in the sense, that they trigger interactions with the Gitlab API.

| Slug | Type | Roles | Purpose |
|---|---|---|---|
| /groups/<int:id>/projects/ | GET | OA | Load new projects from gitlab |
| /groups/<int:id>/update/ | POST | OATM | Update all projects in a group with data from gitlab |

Table 4. Gitlab system endpoints

## 5.2.2 Project level endpoints

A second large set of endpoints are those, that manipulate data, that is tied in some way to a specific project.

For the roles in project level endpoints, it is important to note, that having a role in a `ProjectGroup` automatically means that the user has that role in every project in that group as well.

**Overview endpoints**

The endpoints in table 5 are used to get an overview of the performance of a single project. All slugs in table 5 begin with `/projects/<int:id>`, so those have been omitted from the beginnings, to make the slugs fit better. The id parameter here always refers to the id of the project object.

| Slug | Type | Roles | Purpose |
|---|---|---|---|
| / | GET | OATMV | Get basic overview of project |
| /milestone/<int:milestone_id>/ | GET | OATMV | Get single milestone info for project |
| /milestone/<int:milestone_id>/time_spent/ | GET | OATMV | Get time spent data for single milestone for project |
| /time_spent/ | GET | OATMV | Parametric time spent for project |

Table 5. Project overview endpoints

### Project manage endpoints

The endpoints in table 6 are used to manage various aspects of a project. The id parameters in this section refer to the id of the object that is before it, so either project, repository or milestone.

| Slug | Type | Roled | Purpose |
|---|---|---|---|
| /projects/<int:id>/change_dev_colour/ | POST | OATM | Change dev colour for UserProject |
| /projects/<int:id>/users/ | GET | OATM | Get all users in project |
| /projects/<int:id>/users/ | POST | OATM | Add user to project |
| /projects/<int:id>/users/ | DELETE | OA | Remove user from project |
| /projects/<int:id>/repo/ | POST | OATM | Add new repo to project |
| /repos/<int:id>/project/ | PUT | OATM | Set a repository as belonging to a specific project |
| /projects/<int:id>/milestone_connections/ | GET | OATM | Get gitlab milestone and assessment milestone connections |
| /milestones/<int:id>/assessment_milestone/ | PUT | OATM | Set a milestone as belonging to a specific AssessmentMilestone |
| /repositories/<int:id>/update/ | GET | OATM | Update a single repository |

Table 6. Project manage endpoints

### Assessment endpoints

Group level endpoints had a subset of endpoints, that were connected with the assessment system and in that case they dealt with defining the assessment system for that group. The subset of project level endpoints, that are connected with assessments, then deal with the assessments assigned to that project. Such endpoints are given in table 7. The first fetches info about all assessments given to a project. The second gives a single assessment and the user_id actually refers to a UserProject object. The third allows for several assessments to be sent at once and is simply a list of the type of data that the endpoint before this one sends.

| Slug | Type | Roles | Purpose |
|---|---|---|---|
| /projects/<int:id>/assessments/ | GET | OATM | Get assessments for project |
| /users/<int:user_id>/assess/<int:assessment_id>/ | POST | OATM | Give assessment to user |
| /bulk_assess/ | POST | OATM | Give several assessments at once |

Table 7. Assessment endpoints

### 5.2.3 User level endpoints

Some endpoints are tied to neither project nor group and instead refer to a specific user's data. Such endpoints are given in table 8.

| Slug | Type | Purpose |
|---|---|---|
| /groups/ | GET | Get all groups for user |
| /groups/ | POST | Create new group |
| /profile/ | GET | Get profile for user |
| /profile/ | PUT | Change profile settings for user |
| /invitations/ | GET | Get all invitations for user |

Table 8. User level endpoints

### 5.2.4 Other endpoints

Finally there are a few endpoints that fit into none of the above categories. Those endpoints are given in table 9.

| Slug | Type | Purpose |
|---|---|---|
| /process/<int:id>/<str:hash>/ | GET | Get information about a process |
| /feedback/ | GET | Get feedback matching criteria |
| /feedback/ | POST | Give feedback |
| / | GET | Tests if login credentials are valid |

Table 9. Other endpoints

# 6. Frontend contents

Once the user is authenticated the frontend has 3 main use cases.

- It enables to configure a group of projects to track and compare in the settings view.
- When the course is setup to use the software, some basic visualizations can be drawn for a primitive understanding of teams effort.
- Finally mentors can assess the teams performance using the visual aids to give informed feedback and suggestions for next sprint.

The visualizations are divided into 3 distinct levels of detail.

- On the projects level the graphics display the aggregate data to compare different teams with each other and to bring out the teams that need more care and attention.
- On the level of one project, a detailed graph containing the time-spent data of the team members along with other project metadata is presented.
- And finally, on the level of a single sprint, all the issues are displayed as well as the effort each team member contributed to a given issue. The numerical data of the mentors assessments can also be visualized.

## 6.1 Authentication

To only expose the data the user has been given access to, a JWT (JSON Web Token) [1] access and refresh tokens are generated when the user logs in and they are stored in the browser cookies of the user. This enables to stay authenticated as the logged in user even after the browsing session has ended. If the user tries to authenticate their request with a expired access token, then a new one is generated with the refresh token. The refresh token has a longer lifetime than the access token. If both tokens have expired then the user interface will automatically direct the browser to a log in page.

[1] SimpleJWT

## 6.2 Configuration

There are many ways of configuring a project management. The client strongly preferred to use the application as soon as possible for the Java Games course. Thus all the first developed configuration options reflect the requirements of the client.

### 6.2.1 Assessment tree

The project master must give the mentors a set of parameters to measure in their projects each sprint. This is done with the assessment tree. The assessment system is structured like a tree and the configuration view is built to reflect that in the figure 13. The tree nodes are selectable and modifiable. Each node can be deleted, edited, cloned or be made to spawn new sub-nodes. Each nodes' behaviour relating to how its evaluated points affect the value of the parent can be configured. A assessment node can represent a metric that is evaluated for each team member separately or for the team as a whole.

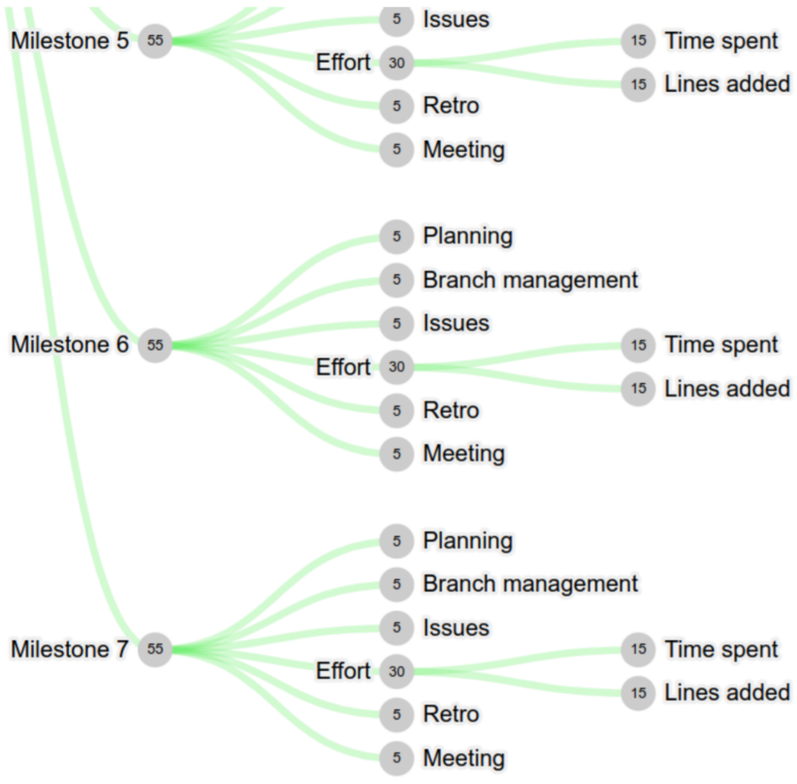


Figure 13. Part of a assessment tree

### 6.2.2 Connection management

There are two distinct places where the categorizing of the incoming data might be obscure and needs to be amendable by the project manager or mentor. This happens when the the right connections were not made automatically.

The first such case is when trying to download a whole GitLab group from the git server. A GitLab group can have many repositories and differentiating them into projects is not a trivial problem. For example in the Java Games course GitLab group there were many projects with multiple repositories for client side and server side code. To determine which repository belonged to which project could not be easily resolved. This problem was settled with a drag-and-drop interface of projects and a bank of un-categorized repositories, where the user can drag the repository under the right project.

A similar situation can occur when the teacher has configured specific start and end dates for the planed sprints which to assess and the GitLab milestones dates of a repository are not aligned. The software tires its best to put the right GitLab milestones in the right sprint, but this still can result in errors which the project mentor should be able to fix easily. The rectification is again done by using a drag-and-drop view shown in the figure 14.

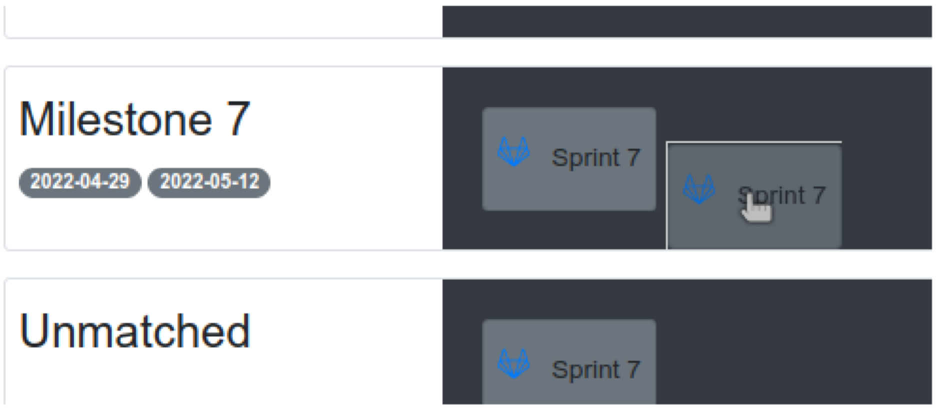


Figure 14. Example of dragging a GitLab milestone in a configured sprint

## 6.3 Visualizations

To enable the assessors to make better judgements and increase the quality of education, helpful data visualization techniques are used across the application.

### 6.3.1 Projects

The projects view is primarily meant for projects managers and project mentors. The mentors will see the groups they are responsible for, while the projects manager sees them all. The view has 2 tiny visualizations. One is a pie chart to highlight the imbalances of the efforts in the team. The other is a graphic showing all the sprints and if they are graded or not. Both graphics are depicted in figure 15.

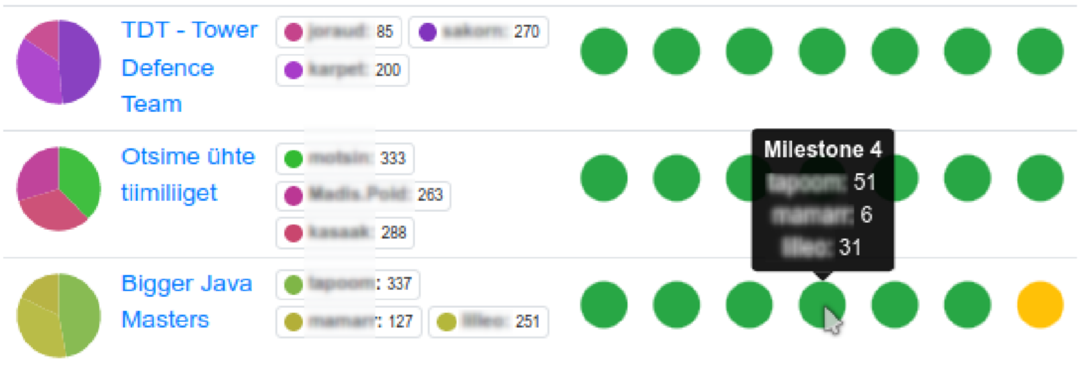


Figure 15. Example two projects in projects view (the usernames are blurred to protect their privacy)

### 6.3.2 A single project

On the level of one project, a detailed graph containing the time-spent data that the team members inserted in their GitLab issues along with other project metadata. The graphic is called GitTime and is presented in the figure 16 where each line of inserted time that the developer supposedly spent, is linked to the issue in which the metadata was inputted. If the data was fetched from multiple repositories then, each one is switchable to hide or show the times from only that repository. The accumulated time each developer spent on the project is presented also in the background.

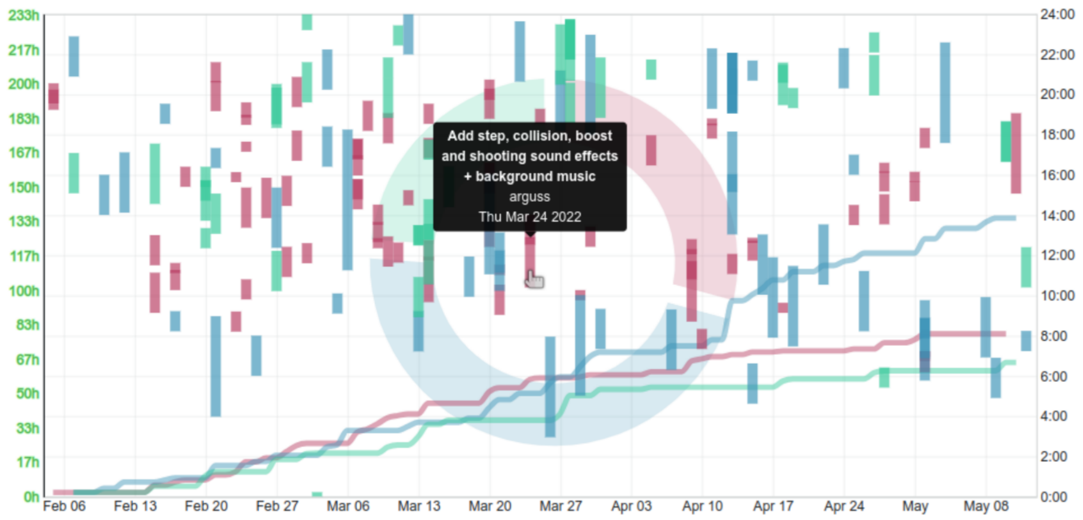


Figure 16. An example of a GitTime graph displaying exactly when each team member worked on the project

### 6.3.3 Sprint

A single sprint displays the same GitTime graphic in figure 16 but with the configured sprint time range. A radar chart like in the figure 17 with the grading axis that were set up in the assessment tree is displayed. If the sprint is being assessed then the radar changes in a reactive fashion. The shadow of the previous sprints radar is also displayed behind the current one, to give a sense of if the project had improved or declined this sprint. The axis range from zero to the configured max points for that specific configured assessment metric.

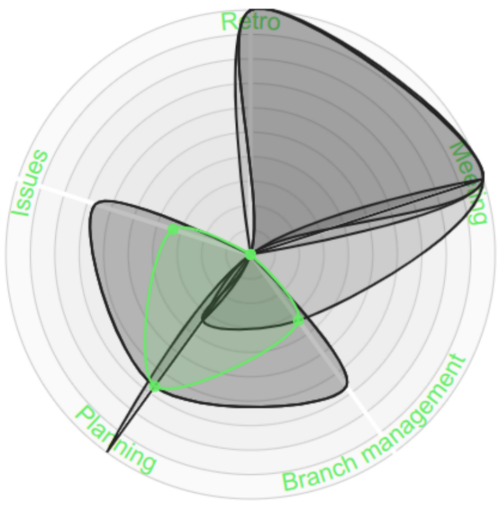


Figure 17. Example radar displaying the assessment on different axis with all the previous sprints radars in the background

# 7. Results and validation

As a result of this project a GitLab group project management system was built. The system is easy to setup, as each component is dockerized and can be simply cloned and run.

The validity of the success of the project depends on how many mentors and teachers are using it and how satisfied are they with their experience. In order to assess this, the project was continously tested using a java game development course in TalTech. At the end of development, feedback was also requested from users, who were mostly TAs, as well as the client, who was also responsible for organizing the Java Games course (ITI0301).

## 7.1 Testing on java games course

Starting from early March, the application was used for assessing students in the Java Games course. The application generated overviews of students progress and TAs used these overviews, to guide their grading. The course manager, could then use the application, to look at the generated overviews, as well as the grades given by the TAs.

## 7.2 Feedback from TAs

At the end of the course, feedback was asked from the TAs via a google forms questionnaire. The questions were chosen in such a way, that they would give both an overview of how good the project is currently, as well as give a direction, where the project should go next. All the questions were in Estonian and they can be seen in appendix 1.

They were asked to rate, on a scale of 1 to 7, how happy they were with the user experience at the moment, how happy they think they would be with the user experience when 3 to 5 most important small updates were added and how happy they think they would be with the user experience when the application would receive further development, roughly in the scope of another bachelor's thesis. It was also specified, that on this scale 1 would mean, that using the application was worse than assessing the teams manually and 7 would mean, that it would be hard to imagine a more comfortable solution.

For the first question, the average rating was 5.25, for the second 5.75 and for the third 6.75. From this it can be seen, that the TAs already found the application useful. Furthermore there is still a considerable amount of low-hanging fruit in terms of development time versus improvements to user experience and the authors estimate, that they will be able to implement the features necessary to reach the 6.0 level, before the final deadline of this thesis. Finally from the last question it is apparent that the TAs also see significant potential in the project and it would be good to continue development on it in the long term as well.

The TAs were also asked what features they would need to see, for their user experience to reach the level stated in the second question. These features, roughly in order of importance, were adding a wiki or guide about how to use project behind a "Learn more" button, having the students be able to see their points, generating a view of issues in a milestone similar to what Gitlab has, improving the speed that it takes for some views to load, showing content from Gitlab project wiki, being able to exclude some developers from project points if they are not contributing at all and being able to customize what info and how is shown on graphs.

The TAs were also asked about the understandability of different views, on a scale of 1 to 7, and what they would change about the views.

For the view that gave an overview of an entire group, the average rating was 5.5. Among the suggested improvements was some metric, that would show if there has been recent activity from the team. It was also suggested that members of a single team should have more different colours and that it should be explained what some colours in the view mean. There was also a request for a feature to be able to filter only teams, where they are a mentor.

For the view that gave an overview of a single project, the average rating was 5.0. Every TA agreed that the graph showing issues and time spent had too much information and should be split into several graphs. It was also said that it should be clearer what some of the information on that page means.

For the view that was used to assess a milestone, the average rating was 5.75. For this view, there was also a similar graph, that showed too much information. Here it was also brought out that it would be good to show time estimates for issues as well. There was also a request for being able to enter points numerically, instead of with sliders and for having a link from spent time to gitlab issue.

## 7.3 Feedback from client

Prior to writing the thesis, feedback was also asked from the client. This time in a more freeform manner.

The client was extremely happy and said that the visualizations really add a lot. He has experience doing the same thing with spreadsheets and even with the previous iteration of Kristjan's script, it simply was not the same and it was not possible to really get an overview of how the teams were doing. He mentioned that the time spent graphs, cumulative graph of team progress and milestone info together gives a very good overview.

He also mentioned that he likes that the system is flexible. For example having the ability to put groups inside of other groups. He also said that he was hopeful, that the code base is clean enough, that the development could be continued next year.

Of course he also had a list of possible further improvements, such as being able to compare suggested assessments with mentor assessments, exporting the assessments to a CSV file and hosting the site at an official TalTech url. The developers are confident that these features will also make it to the application in a few weeks time.

Overall the clients assessment was that it is a very useful application with further room for improvements, but that it has already been useful and has made the lives of TAs in the Java Games course easier.

# 8. Comments

In this section, an overview is provided about the most noteworthy technological and managerial decisions.

## 8.1 Backend

One of the most important design decisions regarding the backend, was the choice to use Python. We hoped that it would be easy to use and allow for great freedom regarding design decisions. The initial fears were that it would be too slow, but with the right programming architecture it proved to be quite preferment.

In the end the fears about Python being too slow, proved to be unfounded. While at some times we struggled with slow endpoints, the reasons were always problems at the levels of the algorithms, that would have been present regardless of language choice.

The fact that Python is not the strictest of languages was both a positive and a negative factor. At times this enabled the writing of rather ugly code, but this was always refactored at later iterations, to match industry standards.

The ability to use Django was probably the greatest reason why using Python was the correct decision. After having used it to develop this application, it became apparent why Django is completely dominant in the Python web framework landscape. It was easy to learn and modify to fit the requirements of this project.

Ultimately the decision to use Python was a success and we hope that using this simple of a language will make it easier to onboard new developers.

We went to great lengths to find the right tool to build out our ERD schemas. Many online tools were discovered and tested, but we found each one of them lacking. Whether it was a confusing user interface or lack of features, none of them met our standards. In the end it was decided, to use Rational Rose [1]. A tool used in our databases course that had a interface few generations older than us, seemed to turn out to offer exactly the feature set

[1]Rational Rose

we needed. Later this was replaced with Visual Paradigm [2]. It would have been good to use Visual Paradigm from the beginning, because Rational Rose was simply too outdated and it took quite a bit of time to redo the ERD schemas in Visual Paradigm.

## 8.2 Frontend

Before the development had begun the designer had some prior experience with the Vue frontend JavaScript library. But the little experience they had did not compare with building and architecting a whole client side interface. Much was learned about authenticating, routing and state management. But in the end the choice of using Vue turned out brilliantly. The framework is built beautifully and working with it was a joy.

What took us by surprise was the enormous learning curve of the D3js web graphics library. Although it was anticipated that some time would go into familiarizing with the library, no one expected that it would take almost a month before something would draw on the screen without breaking the browser. After countless hours of learning from the extensive documentation, the visualizations would begin appearing in our frontend. If the designer would have to do it again they would start by getting the hang of the basics of D3js first and only after that would they begin to design their own elaborate graphs.

## 8.3 Development methodologies

Most of the development methodologies that were decided upon in the beginning, ended up being rather useful.

One of the best decision made at the beginning, was to make different versions of the project easily be deployable. That means dockerizing both the frontend and the backend and setting up GitLab pipelines for automatic deployment. These steps need to be made at some point of the development arc anyway and if they are done later, then they might be harder to set up. What is more, the time spent manually deploying before setting up pipelines would simply be wasted.

It was also decided early that we would work in one week sprints and have a retro at the end of each. This proved to be a good idea as well, as it meant that there was never a long pause in development where no one did anything. The authors had noticed issues like this happen in other group project courses where they did two week sprints.

[2]Visual Paradigm

What was not planned from the beginning, but proved to be fruitful, were the 1-2 day "hackathons". These "hackathons" happened, when the developers noticed, that the development had been going slower than preferred. they would both clear a day or two from their calendars to get together and develop the application. This turned out to be an effective way to develop large features quickly and also to get the developers out of a rut, when they found themselves in one.

What helped most to motivate us, was a sense of a strong vision of what we wanted to build. As we got closer to our goal the vision got progressively more precise. After all, one can have the most advanced project management software in the world, but that would still not matter if the vision is not strong enough to mobilize oneself to change ones behaviour, skills and ultimately, ones identity to reach the goals. That might be the most important insight we got from working on this project.

# 9. Conclusion and future developments

In the end, we are delighted with the result as we have managed to make substantial advances to the status quo of software developer team management in TalTech. We even managed to test our product out in the Java Games course and collect a reasonable amount of data.

To keep this application in constant development, the plan is to give this project over to the next team doing a bachelors thesis next spring. That way the cost of development will stay negligible. We have already found people interested in growing this application further and they have started familiarizing themselves with the codebase.

An avenue for future development is definitely integrating the application with other TalTech systems. It would make life easier for a course organizer if the app was integrated with Moodle so that assessments from the app could easily be transformed to grades on Moodle. Right now they have to be inserted manually.

Some programming courses use GTM [1] to track the actual programming time in students git commits. It would be neat to see how the legitimate time of writing code differs from the time the developers themselves inserted. That way the data can be further discerned between the time writing code and the time spent on supportive activities.

The TAs from the java games course were asked what features they would like to see, after another bachelor's thesis was written that further developed this application. In addition to the things already mentioned, they also suggested that the system for suggesting grades should be more comprehensive and that it would be good if the students applications could be launched before and after each issue, to see what that issue did. A common theme among these requests was that the end goal should be that opening Gitlab is no longer necessary for grading and all data is in this application.

What is perhaps most intriguing for the architects of this application, is the fact that the app can be used to improve itself recursively. As the developer develops the tool that helps them and others like them develop better, the developer gets wiser about his development

[1]GTM

habits and improves them, which in turn generates higher quality software, making the tool even more useful. If nothing else, we hope to have made major gains to bootstrap the process of building higher quality software in universities and beyond. It will be to our total pleasure to see the ideas presented in this thesis inspire the next team that has a chance to work on it.

We are planning to continue using the app personally and are also planning to put more work into developing it further in the coming months.

# Bibliography

[1] Synopsys Editorial Team. *Top 4 software development methodologies*. URL: https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/.

[2] Andrea Crawford. *What is DevOps?* URL: https://www.ibm.com/cloud/learn/devops-a-complete-guide.

[3] Andrzej Maleszewski. *Technical Debt in Software Development*. URL: https://www.miquido.com/blog/technical-debt-in-software-development/.

[4] *MariaDB vs PostgreSQL: Know the difference between the databases*. URL: https://optimizdba.com/mariadb-vs-postgresql-know-the-difference-between-the-databases%5C%EF%5C%BB%5C%BF/. (accessed: 07.05.2022).

[5] Vladimir Khorikov. *Database versioning best practices*. URL: https://enterprisecraftsmanship.com/posts/database-versioning-best-practices/. (accessed: 07.05.2022).

[6] *What is an Entity Relationship Diagram (ERD)?* URL: https://www.lucidchart.com/pages/er-diagrams. (accessed: 07.05.2022).

[7] Frederick Brooks. *The Mythical Man-Month. Essays on Software Engineering*. 1975.

[8] Interaction Design Foundation. *User Interface Design*. URL: https://www.interaction-design.org/literature/topics/ui-design.

[9] Ben Shneiderman. “Direct manipulation: a step beyond programming languages,” in: *IEEE Computer* 16.8 (1983), pp. 57–69.

[10] Jessica Hullman. *The Purpose of Visualization is Insight, not Pictures: an Interview with Ben Shneiderman*. URL: https://interactions.acm.org/blog/view/the-purpose-of-visualization-is-insight-not-pictures-an-interview-with-ben. (accessed: 01.05.2022).

[11] Jakub Majorek. *19 Best JavaScript Data Visualization Libraries [UPDATED 2022]*. URL: https://www.monterail.com/blog/javascript-libraries-data-visualization.

[12] *Introduction to HTTPS*. URL: https://https.cio.gov/faq/. (accessed: 30.05.2022).

[13] B. Kaliski. *PKCS 5: Password-Based Cryptography Specification. Version 2.0*. URL: https://www.ietf.org/rfc/rfc2898.txt. (accessed: 30.05.2022).

[14] *Rainbow Table Attack*. URL: https://www.beyondidentity.com/glossary/rainbow-table-attack. (accessed: 30.05.2022).

[15] Meltem Sönmez Turan et al. *Recommendation for Password-Based Key Derivation*. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf. (accessed: 30.05.2022).

[16] M. Jones et al. *JSON Web Token (JWT)*. URL: https://datatracker.ietf.org/doc/html/rfc7519. (accessed: 08.05.2022).

[17] Eva Sarafianou. *Secure Browser Storage: The Facts*. URL: https://auth0.com/blog/secure-browser-storage-the-facts/. (accessed: 30.05.2022).

# Appendices

# Appendix 1 - Cognate Feedback form

All the questions are in Estonian.

- Kui rahul oled praeguse Cognate kasutajakogemusega? (1-7p)
- Mis on Sinu jaoks kõige olulisem osa Cognate funktsionaalsusest, mis kindlasti peaks igal juhul alles jääma?
- Mis on Sinu jaoks kõige vähem oluline osa Cognate funktsionaalsusest, mille võib vabalt ära võtta?
- Kuidas sa hindad, milline oleks Cognate kasutajakogemus kui teha 3-5 Sinu jaoks kõige olulisemat väikest parandust / edasiarendust? (1-7p)
- Mis oleks kõige tähtsam nendest parandustest / edasiarendustest?
- Mis oleks teine kõige tähtsam nendest parandustest / edasiarendustest?
- Mis oleksid ülejäänud väikesed parandused / edasiarendused, mida tahaksid näha?
- Kuidas sa hindad, milline oleks Cognate kasutajakogemus kui selle kallal teha veel märgatavalt tööd (näiteks veel ühe bakatöö raames), arendades Sinu jaoks kõige olulisemaid funktsionaalsuseid? (1-7p)
- Kirjelda lühidalt, mis oleksid need funktsionaalsused või mis muudatused oleksid toimunud, kui saaksid selle bakatöö vältel suunata Cognate arendust nii, nagu Sina soovid?
- Kui arusaadav on grupi kokkuvõtte vaade? (1-7p)
- Mida sooviksid muuta grupi kokkuvõtte vaate juures?
- Kui arusaadav on projekti kokkuvõtte vaade? (1-7p)
- Mida sooviksid muuta projekti kokkuvõtte vaate juures?
- Kui arusaadav on milestone hindamise vaade? (1-7p)
- Mida sooviksid muuta milestone hindamise vaate juures?
- Kas soovid veel midagi öelda?