TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Ilja Boitšuk 194041IAIB
David Avedis Injarabian 194044IAIB
Janar Keit Jaakson 193703IAIB

Plagiarism System Integration With Moodle's Plugin Charon

Bachelor's thesis

Supervisor: Ago Luberg
PhD

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ilja Boitšuk 1940401IAIB
David Avedis Injarabian 194044IAIB
Janar Keit Jaakson 193703IAIB

Plagiaadisüsteemi liidestamine Moodle'i pistikprogrammi Charoniga

Bakalaureusetöö

Juhendaja: Ago Luberg
PhD

Tallinn 2022

Author’s declaration of originality

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

Author: David Avedis Injarabian, Janar Keit Jaakson, Ilja Boitšuk

30.05.2022

Abstract

The purpose of this thesis is to integrate the existing plagiarism detection application (hereinafter PLAP) with a Moodle plugin called Charon. This will help teachers engage with the detection software directly from Moodle without having to specifically access any other external application.

Charon is mainly used to view and manage student's programming code solutions, while PLAP is to check the solutions for plagiarism. An integration between the two applications allows for better user experience, custom solutions for teachers, visual data representations and scalability across multiple different Moodle instances.

Technologies used for PLAP are mainly Django and React, while Charon development uses Laravel and Vue.js. Both applications run inside docker containers and are managed by their respective GitLab repositories.

Key functionalities that this thesis aims to implement are to run plagiarism checks from Charon, comparing different solutions, viewing a student's plagiarism history and automatically creating courses and assignments in PLAP from Moodle.

This thesis is written in English and is 40 pages long, including 9 chapters, 7 figures and 0 tables.

Annotatsioon
Plagiaadisüsteemi liidestamine Moodle’i pistikprogrammi Charoniga

Antud lõputöö eesmärk on integreerida tudengite lähtekoodide plagiaadikontrolli rakendust (edaspidi PLAP) Moodle’i pistikprogrammi Charoniga. Integratsioon tagab mugavama plagiaadikontrolli käsitlemise otse Moodle’st ilma, et oleks tarvis kasutada ühtegi välisrakendust.

Charon on peamiselt kasutatud TalTech’i professorite ja õppejõudude poolt tudengite lähtekoodide lahenduste vaatluseks ja manageerimiseks. Selle eesmärk on tagada keskne rakendus, kust on võimalik autoriseeritud isikutel ligi pääseda programmeerimisülesannetele ja tööde esitlustele. PLAP’iga integreerimine ei edenda ainult Charonis kasutajakogemust, vaid võimaldab ka vaadelda visuaalseid tulemusi, arvestada õppejõudude personaalsete ettepanekutega ning tagab laiapõhise kasutatavuse üle ülikooli erinevate Moodle’i instantside.

Antud rakenduste integratsioon toetub PLAP’i ja Charoni tehnoloogiatele. Kasutatud tehnoloogiad koosnevad peamiselt PLAP’is Djangost ja React.js’st ning Charonis Laravelist ja Vue.js’st. Mõlemad rakendused töötavad Dockeri konteinerites ning on hallatud vastavatest GitLab’i repositooriumitest.

Primaarse tähtsusega funktsionaalsused mida antud lõputöö üritab saavutada on plagiaadikontrolli jooksutamine Charonist, tulemustest lähtekoodide visuaalne võrdlemine, sarnasuste plagiarismiks või aktsepteeritavaks märgistamine, tudengi plagiaadiajaloost ülevaate kuvamine ning automatiseeritud kursuste ja ülesannete loomine PLAP’is läbi Moodle’i.

Lõputöö on kirjutatud keeles ning sisaldab teksti 40 leheküljel, 9 peatükki, 7 joonist, 0 tabelit.

List of abbreviations and terms

API	Application programming interface
Charon	Custom Moodle plugin for programming exercises
Charon Popup	An environment in Charon that is meant for teachers to get a better overview and manipulate students Charon submissions
Discord	Communication application
DevOps	Software development (Dev) and IT operations (Ops)
DOM	Document Object Model
Git	Version control system
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
Modal	Popup dialog inside a page, usually triggered by a button
Moodle	Free and open-source learning management system
ORM	Object-relation mapping
PLAP	Plagiarism detection application created by Ragnar Rebase
REST	Representational State Transfer
UI/UX	User interface/User experience
Uni-id	Digital identity for a person in TalTech

Table of contents

List of figures

1 Introduction

During the course of human history, people have always copied or taken inspiration from other people, in all fields of life. In most cases, this is a natural occurrence that has enabled civilizations and cultures to prosper, technologies to advance and human knowledge to grow. However, along with benefits also come downsides. The act of plagiarism allows for the theft of another person's original work to occur. It discredits and robs the author, who put in all the effort, of recognition and rewards of contribution. Plagiarisms can also allow unqualified people to bypass the qualifications process of their field of expertise. Such an event could cause major damage to society, especially if the person in question is a politician or minister, who would in such case not possess the necessary knowledge to make adequate decisions.

Plagiarism has possibly existed for as long as mankind, but only with the arrival of computers have people started receiving more justice. This is due to applications and mainly algorithms that are capable of processing documents and checking for keyword similarities in their database. Such applications are mostly used by academics in universities, where research papers are checked for plagiarism. The issue of plagiarism is perhaps most relevant today, due to the internet and all of the information it brings along with it. Students are finding more and more answers to their questions via online articles, videos, forum posts and possibly even research papers, which creates a strong compulsive reaction that drives them to use the same solution as the person who they learned it from.

Programmers are no different from this, in fact they are perhaps one of the biggest copy-pasters of all time. Almost all of them look on the internet for answers and once a solution is found, the initial reaction is to take it for themselves without referencing the author. Such acts are arguably fine and cause no harm in a personal or hobby project, but in an academic context they are frowned upon. However, in programming there are certain solutions that are very similar to one another, which leads to people using the same approach to certain problems. For this reason, the plagiarism algorithms for programming code solutions must be able to take into account, that not all similar solutions are plagiarisms.

As programming has started growing ever more popular, more and more students enrol into programming courses in universities. Such courses contain a lot of different kinds of exercises. The students, naturally, form certain groups among themselves, in order to receive help in case of need. Divide and conquer is a much safer strategy of approach in the pursuit of excellent academic grades. For that reason, schools need to be extra wary and adopt a way of comparing the students' results between themselves, in the hopes of catching someone who is using the work of another.

TalTech is one of the most popular universities in Estonia. It concentrates on technological subjects and therefore attracts many programmers. The school takes a more practical style of teaching, meaning there are a lot of exercises, where students can play around with the subject instead of theorizing everything. This applies heavily to programming, as the first courses always contain tons of problems for newcomers to solve. However, due to the high number of participants each year, everybody gets more or less the same exercise and it becomes very difficult for teachers to manually check everyone's work for plagiarism. The solution is simple: the school needs a way to algorithmically check for programming code plagiarisms, by only pressing a couple of buttons on the screen.

The aim of this thesis is to provide that solution. While the school was already in possession of an application that could check student's codes for plagiarism, it was very difficult to get teachers to use it regularly. Some were not even aware of its existence. Those that were, did not feel too comfortable switching between two applications all the time. It became clear, that the plagiarism application had to be integrated with the school's programming exercises management Moodle plugin called Charon. When it comes to programming exercises, this plugin is what teachers are usually most familiar with. If there was a way to check student solutions for plagiarism from Charon, then the likelihood that teachers start using it will definitely go up.

In addition to the most important functionalities, we also wanted to provide teachers with more visual information about the data they are looking at. This meant new tables, fields, graphs and sections. Some completely new features were added as well, for example one being the ability to view the plagiarism history of a student. While integrating the applications, we were aware of the importance of data synchronicity. We decided to enable teachers to create identical courses and assignments in PLAP, so that they can always be easily set up and reconfigured if needed.

2 Task proposal

The task of this thesis is to provide the ability for teachers to check student's programming code solutions for plagiarism directly from Charon (Custom Moodle plugin for programming exercises). TalTech already has PLAP (Plagiarism detection application created by Ragnar Rebase), which is an application made for checking code submissions for plagiarism. Therefore, this thesis aims to provide Charon with the necessary controls to engage with PLAP remotely. In addition to remote controls, one of the key features to add are also visual data representations. Teachers need to be able to see more data in order to grasp a better understanding of the situation.

2.1 Client

The task of integration was commissioned by our supervisor Ago Luberg, who would also act as our client. The client explained the need for the ability to check students' code submissions for plagiarism directly from Charon, without having to visit PLAP.

The main reason for this was to encourage teachers to engage with plagiarism checks more frequently, as many professors and teachers are only accustomed to using Charon for their work. This was supported by the fact that some of them had no idea what PLAP is and how it works. Being able to do everything from a single place will definitely improve the user experience for teachers.

Another reason for the proposal of integrating PLAP was that it would allow us to display new sorts of information as well as create custom solutions and improve compatibility with Charon. An example of a compatibility issue was the addition of code submissions to Charon which were not stored in GitLab and were therefore inaccessible to PLAP. Our client wanted the ability to test the previously mentioned submissions as well as the ones that have an association with GitLab for plagiarism.

2.2 Objective

Our objectives were mainly divided into two parts: enable the execution of PLAP functionalities through Charon and adding new features in order for teachers to obtain a better overview and understanding of student’s plagiarism results. We were also aware that we would need to establish a CI/CD pipeline in GitLab for running tests regularly and also to provide a way for remote deployments.

Enabling the execution of PLAP functionalities through Charon consisted of running plagiarism checks, fetching the found matches and viewing the results via diagrams and code comparison modals (Popup dialog inside a page, usually triggered by a button). Since we had access to a REST (Representational State Transfer) API (Application programming interface), we were able to easily run checks and retrieve matches. As for visual data representations, we were able to use the pre-existing retrieval functions to set up similar graphs and code comparison sections.

Before we started to work on additional functionalities, we consulted primarily with our supervisor as to what features he would like to have us implement. We learned that in addition to new tables and ways to portray summarized plagiarism information to teachers, there was also a need for the automation of course and assignment creation in PLAP from Charon. This was mostly a synchronicity issue, as courses and assignments had to be synchronized between PLAP and Moodle (Free and open-source learning management system) in order for plagiarism runs to be activated.

3 Project description

3.1 Workflow

While most of the communication between team members happened in Discord (Communication application), we still relied on our coding practices that we learned in school. An example of this would be working as a team and utilizing the Scrum framework by setting up a roadmap, milestones, distributing work and having regular meetings.

3.1.1 Scrum

In order to coordinate work between ourselves and ensure the best possible outcome, we used Scrum to better understand and manage the project. We created a timeline of different major steps and broke them into smaller pieces called sprints. Each sprint would last for two weeks and contained different issues with deadlines set at the end of the sprint. If a deadline was not met, then the issue would be moved onto the next sprint with a technical debt. At the beginning of each new sprint, we would review the work accomplished in the previous sprint and add new issues to the backlog.

The team held meetings at least three times a week, to ensure that all members were up to date with each other's work. During the meetings we would discuss the current situation of our issues and address certain problems that we were facing or could foresee happening in the future. At the end of the meetings, we would plan out our next actions and write up a memo, which contained the details of the meeting for future reference. Once a week we would also discuss with our supervisor how the project had been moving forward and ask questions regarding certain solutions, as our supervisor was also our client.

3.1.2 Work distribution

At the start of the project, we were confronted by many new technologies of which we knew little of, mainly because our project was the integration of PLAP, which was the work of another student. For that reason, we decided at the beginning not to distribute work by certain groups or categories and rather to start learning and understanding the previous application that we were going to be working on.

While PLAP was foreign to us, Charon was not. All of us had been working on Charon for the past year as software developers. This made the integration a lot easier than it would've been, as we had a good base of understanding in regards to how it operates. With Charon out of the way, we were able to concentrate our attention mainly on PLAP, as it had the biggest learning curve.

Once we had acquired the necessary knowledge to understand the architecture and flow that we were going to be working with, we started adding issues to the backlog. Since each member was advancing through the technologies at a different pace, we would all take different tickets that we felt comfortable working with. During meetings we would make sure that the issues we had taken would not end up in conflict with each other's work. Therefore, our work was always distributed dynamically, at times one of us would work on visual components, at other times they would set up backend communication routes between PLAP and Charon. Everyone was flexible and decided to try and be of use all around.

3.1.3 Code quality

When adding new code to either application, we wanted to make sure that we were not affecting the performance of something that was already implemented and correctly set up. Our main goal was to integrate the two applications in their current states, while adding new functionalities on top of the previous ones.

When creating new data models or endpoints for either application, we took inspiration from the layouts and structures that had already been used to create previous forms of functionalities. For future developers, we added comments to our functions and to the code that already existed before the integration, which would help them in understanding what had been done and why. In order to ensure the durability of our solutions, we wrote tests to find out whether they would keep functioning under different kinds of circumstances.

3.2 Potential solutions

During the initial phases of the project, we wanted to explore our options as to finding the right solution to our problem. We were aware that an application for plagiarism detection already existed, but we needed a broader view of options in order to be convinced whether it was our best choice.

3.2.1 Moodle’s plagiarism plugins

Moodle has a collection of different plagiarism plugins, which have been developed by different companies and institutions in order to provide other parties the ability to check for plagiarisms. This approach would have been the most convenient out of all others, as the integration was straightforward, but unfortunately it came with many drawbacks.

The main problem with Moodle’s plagiarism plugins is that they are all developed specifically to be compatible with another Moodle’s built-in plugin. This would normally have been fine for most popular plugins accessible in Moodle, but Charon is not a built-in plugin. Charon is designed specifically for programming exercises and solutions by TalTech programmers, making it a custom plugin which is not part of Moodle’s standard set of plugins. For this reason, none of the plagiarism plugins made for Moodle would be suitable for integrating with Charon.

3.2.2 PLAP

While searching for external services that would suit our needs, we were unable to find any. This was mainly due to our lack of knowledge of plagiarism detection services, as well as Moodle being a rather unique platform with not sufficient adoption in order to find compatible solutions.

Fortunately, there was a plagiarism application which was the bachelor’s thesis of Ragnar Rebase. His application was developed specifically for students whose code submissions were stored in GitLab. It also had already implemented a plagiarism detection service developed by Stanford University called MOSS. The application used a REST API and modern technologies, which would have allowed us to expand our own implementations and establish communications with Charon.

3.2.3 Final decision

When searching potential solutions, we quickly realized that we needed some form of external service where we could send the code submissions and receive plagiarism results. Moodle plagiarism plugins were in that case not an option since they provided plagiarism checks for built-in plugins only.

Plagiarism detection services made for programming code submissions were difficult to come by. We realized, that even if we did find a suitable service, we would have needed to set up an external application of our own, with a backend configured to communicate with the plagiarism detection service. We did not want to integrate the external service with Charon directly, as that would have added more complexity to Charon as well as limited the usage of plagiarism checks to only Charon users.

Upon analyzing the PLAP created by Ragnar Rebase, we were quickly convinced that it made the most sense to use it for our integration. It was already a working application, which was configured and connected to both GitLab and an external plagiarism detection service. It also enabled the usage of different kinds of authentication and a REST API, which meant custom endpoints accessible by different Moodle instances.

3.3 Working with two projects

Both applications play a critical role in the integration process. PLAP is mainly used to fetch information and send it to Charon, while Charon is used to display that information in different types of ways. From that perspective, our work in PLAP was mainly backend oriented, while in Charon we concentrated mostly on the user experience.

3.3.1 Multiple Moodle instances

Soon after the initial development of the integration, we were faced with a concern regarding multiple instances of Moodle. TalTech wishes to utilize the integration of PLAP and Charon with all of its Moodle instances. This posed a problem regarding courses with the same name. PLAP was initially developed for a single instance, meaning course names were meant to be unique.

Our integration had to address this problem by altering the previous logic of PLAP. Our solution was to remove the *unique* constraint set on the *Course* model in PLAP and add an extra field for the user who created the course. We decided that this would be the most optimal solution, as we also needed an authenticated user through which both applications would be able to communicate with each other. Courses would therefore always contain a reference to the authenticated user who created them, making it possible to identify courses of a certain Moodle instance.

4 Project design

PLAP and Charon are both comprised of various technologies used in the development of their frontend and backend counterparts. This chapter displays the portion of these technologies that were used in the integration specifically. In order to read about all of the technologies used in the development and usage of PLAP and Charon, it is recommended to read their respective documentations.

4.1 PLAP

PLAP is an application built upon the Django framework, which is based on Python. It follows the Model-View-Template pattern, encourages faster and more effective development with good security and an *out-of-the-box* administration interface. It is mainly created to work as a *Single-page application*, meaning that most information arrives to the user via the API [1].

PLAP also uses WebSockets in parallel with the Django REST framework, to display real-time changes without having to make HTTP (Hypertext Transfer Protocol) requests. WebSockets are mainly being used to display real-time logs and when updating match statuses. They are also used for interacting with Moss, as the files are sent via streams instead of HTTP requests [1].

Since PLAP can sometimes contain some time-consuming tasks, Celery and Redis are used to create background tasks in Django. Celery is a task queue implementation in Python and Redis is an in-memory database. Because of this, the user can keep performing new tasks even when there are very slow tasks running in the background [1].

PostgreSQL is used to host the main database. This works really well with Django, because its tables are described as Django models and the data is being processed using the Django ORM (Object-relation mapping), which all in all gives the programmer a lot of flexibility and reduces the complexity of certain procedures automatically [1].

The application's *front end* is built using React, which is known for its development ease and speed. Babel is used to compile the JavaScript code; this allows to make new generation JavaScript compatible with older browsers. Webpack is being used to pack modules together so that they would free up space, which is mostly used in non-development environments as to allow clearer debugging [1].

4.1.1 Moss

Moss is a system that can detect similarities between different kinds of programming codes. Since the system has no way of knowing *why* codes are similar, it cannot detect if the code contains plagiarism. Deciding which code is plagiarism or not is up to the person who reviews the results. Moss only highlights similar parts and makes it easier for reviewers to analyze the code [2].

Moss may seem unsafe at first, because when files are sent to Moss, it doesn’t respond with new files per say. Instead, it responds with an URL, that contains the results. The problem is that the URL can be accessed by anyone during the time that it’s active. Regardless, some precautions are being taken:

- Only the initiator of the request receives the URL and it contains a random integer, so it’s not easy to guess specific results;
- The directory with the results cannot be browsed or indexed by robots;
- Submissions are not retained indefinitely on the server; they are typically deleted after 14 days [2].

4.2 Charon

Charon is a plugin application created to act as a programming assignment in the learning platform Moodle. It is built using the Laravel framework, which is used to develop *model-view-controller* based applications. Using Laravel, the developer only needs to develop the application and not have to worry about redirecting requests, forwarding HTML (Hypertext Markup Language) files and dependency injections. Laravel also has many comfortable services that help the developer, e.g., Blade template engine, Eloquent ORM, automatic dependency injection etc [3].

Vue.js is being used for front-end development, primarily to assemble user interfaces and single page applications. Vue is similar to React, using a virtual DOM (Document Object Model), development based on components, compact and minimal code core with many different libraries that contain extra functionalities, e.g., forwarding requests and global variable management. Vue's advantage over React is that Vue is more beginner-friendly, faster and can contain only html in many components [3].

Using Vue and its component system, the front-end is divided into multiple parts. In general, every component mainly does one thing and one thing only. Due to this, a lot of files are usually created. This however makes it easier to develop, since implementing a small change will likely only affect a single component, while the rest remain unaffected. This type of pattern means that every component should be created as loosely-coupled as possible, meaning that it should depend on other components as less as possible. This way components can be reused in different situations, as they are not heavily dependent on a certain other component [3].

4.3 Implemented technologies

Both projects had a lot of technologies already in place that we simply had to work with, for example Django and React or Laravel and Vue. However, there were a few libraries that we used specifically when setting up new visual components.

ApexCharts.js is a JavaScript library that developers can use to create interactive visualizations for web pages. It supports different JavaScript frameworks and provides multiple types of highly customizable and responsive charts [4]. ApexCharts has proven very reliable for us whenever there has been a need to set up data quickly with good looking reactive graphs. Since its so rich in features, meaning that it has a lot of different graphs to offer, then it really does come in handy in almost all cases when a graph is needed.

The Ace editor is an embeddable code editor that can be embedded in any web page and JavaScript application. It offers a variety of features, including syntax highlighting for over 110 languages, over 20 themes of text and automatic indent and outdent [5]. One of Ace's features is the ability to highlight certain lines of text with color, which helps us to highlight the similar parts inside students' codes.

Network is a visualization to display networks and networks consisting of nodes and edges. The visualization is easy to use and supports custom shapes, styles, colors, sizes, images, and more [6]. Vis-network is used as it gives us similar functionality to visualize students and relationships between them in Charon as it is in PLAP itself. Thanks to its wide range of functionalities, vis-network allowed us to expand colour representation of plagiarism and open code comparison in a separate window.

4.4 External services

GitLab acted as our DevOps (Software development (Dev) and IT operations (Ops)) platform. It enabled us to create issues and milestones, review commits and merge requests, leave feedback on solutions and most importantly, it provided us with file management and source control capabilities. By using a GitLab environment, we made sure that future programmers, who will work on this integration, will have notes and comments for issues that they can always look up when needed.

GitLab isn't just simply our DevOps platform. It plays a crucial role in our integration. TalTech's programming teachers enjoy storing student submissions in GitLab. Thanks to this, PLAP is able to clone the corresponding repositories, as long as it has the right Git (Version control system) Access Token. While GitLab is mainly a DevOps environment for developers, its scope seems to be much larger. By providing different applications various materials through its inner API, it proves itself as a very important part in out integration.

4.5 Development utilities

GitLab CI/CD is the part of GitLab that you use for all of the continuous methods (Continuous Integration, Delivery, and Deployment). With GitLab CI/CD, you can test, build, and publish your software with no third-party application or integration needed [7].

Continuous Integration ensures us that each change submitted to an application, even to development branches, is built and tested automatically and continuously. These tests ensure the changes pass all tests, guidelines, and code compliance standards you established for your application [7].

Continuous Delivery checks the code automatically, but it requires human intervention to manually and strategically trigger the deployment of the changes. The deployments triggering should be made manually [7].

Continuous Deployment means that instead of Continuous Delivery, where you need to deploy your application manually, you instead set it to be deployed automatically. Human intervention is not required [7].

Docker is a software to create and manage environments inside another environment. It uses virtualization technology to create containers, which can host another operating system. This kind of cycle of environments can keep on going further and further. We use docker mainly to isolate certain applications, so that they're dependencies do not end up in conflict with each other. Containers are a great way to isolate new applications on servers where multiple applications could be running at the same time. Containers are also really handy since they provide everyone the same conditions when setting up a dockerized application.

5 Plagiarism system integration with Charon

5.1 Integration architecture

Charon is an application that is connected to multiple different applications. While planning the best course of action towards integrating PLAP with Charon, we decided not to build PLAP inside Charon, but to rather keep both applications separated. Since this project consists of both PLAP and Charon, it can be divided into the following components:

- Charon front-end
- Charon back-end
- PLAP front-end
- PLAP back-end

While the integration architecture can be mostly described through the communications between the aforementioned four components, the complete architectural overview contains some additional external services (Figure 1).

As can be observed from the figure, different connections are differentiated by different colors. Yellow lines represent a HTTP connection, while blue ones denote a socket connection. The gray arrow depicts a GitLab hook, which is called when a student submits a submission to a repository, which is then saved to a database through Charon's backend.

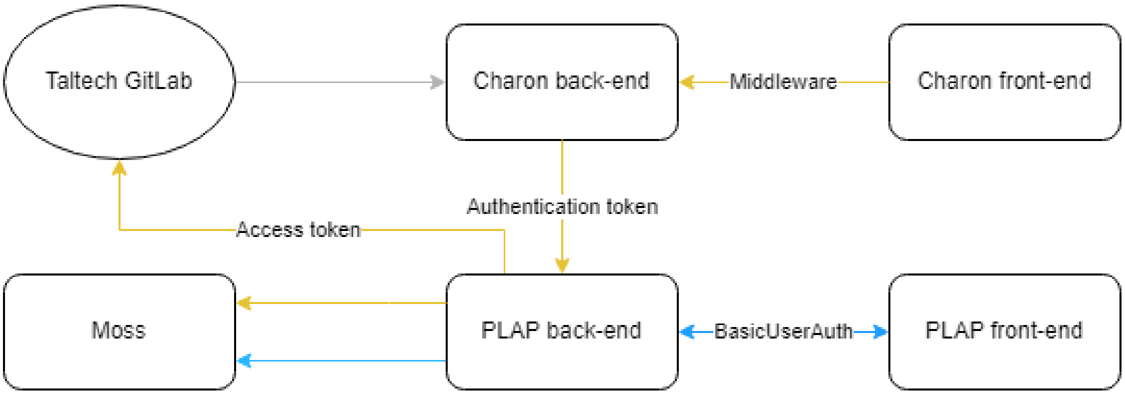


Figure 1. System architecture

Charon and PLAP both use an authentication layer for security measures. For this, Charon mainly uses different kinds of middleware, provided by Laravel, to authenticate Moodle users. Only teachers and other authorized users should have access to sensitive information about students. PLAP on the other hand uses Django’s built-in authentication system in order to handle permissions and access to resources.

Charon and PLAP communicate via an authentication token. The token must be generated from a user in PLAP. Every Moodle instance should have at least one user set up in PLAP, from which a token can be generated. The user acts as a sort of representative figure for its Moodle instance. Using another instance’s token can lead to accidental changes to another Moodle instance’s courses or exercises.

While PLAP uses Django’s built-in authentication system, it relies mainly on WebSocket for communications between React and Django. WebSocket is also used for communicating with Moss. In order to send submissions to Moss, the submissions must first be pulled from GitLab. This requires a PLAP user to have added its GitLab Access token to the system. The token is used to clone the projects and send the submissions to Moss through WebSocket. Upon a successful plagiarism check, Moss responds with an URL, which contains the results of the check. In order to save the results in PLAP, the data is fetched via a HTTP request from the URL and parsed in Django afterwards.

5.2 Data models

Since Charon is a Moodle plugin, it uses data models that are defined in Moodle’s database. PLAP is specifically designed to support the Charon workflow, which means its data models are designed to accommodate the data in Moodle’s models. While both applications contain a large variety of models, this chapter aims to describe the ones that were most important for the integration. The entire PLAP models visualisation can be seen in appendix 2.

The Course model in PLAP represents a course in Moodle, with additional information needed for plagiarism operations. Alongside the name and language, it contains references to the GitLab project that it’s associated with as well as certain Moss parameters that can be used when running plagiarism checks. It is also associated with different assignments. An additional field that was added for the integration is the *created_by* field, which points to the user who created the course. This enables PLAP to identify the Moodle instance to which the course belongs to. Moodle’s course contains various types of information regarding procedures that are not associated with PLAP. The name of the course is the only field that both models need to match.

A *Project* model in PLAP used to be called *GitLabProject.* Due to new changes regarding the integration, we decided to divide it into two tables: *Project* and *GitLabProject,* since it no longer always represented a GitLab project specifically, as submissions without GitLab relation have no GitLab data. A GitLabProject is still a model, but it only contains GitLab related information. A Project model can hold a reference to a GitLab project, but it's not mandatory. Projects are associated with students. When sending files and receiving results from Moss, the project id is used in the file paths. That way, the project is associated with the file path, and later the student can be associated with the file through the project. The column *group_id* was removed from the Project table, due to it being unnecessary.

In PLAP, the assignment is meant to represent a Charon assignment, which in layman's terms is the programming exercise for which submissions are submitted. The name of the assignment must be identical to the Charon assignment. It contains a reference to the course it belongs to and additional information for Moss configurations. It is also referenced by submissions that belong to it. A Moodle assignment is incidentally called *Charon*. It represents a programming exercise and therefore contains information related to the internal operations of the plugin. Additionally, it contains a field called *plagiarism_assignment_id,* which points to an assignment in PLAP. The field is mainly used in Charon to find assignments for which a counterpart exists in PLAP.

Submissions in both applications are similar to one another. They both have a reference to the assignment that they belong to as well as the content of the submission itself. In addition to that, a Moodle submission has information related to its performance, while PLAP's submission can contain the commit hash as well as a reference to the Project, where is stored GitLab links to submission, if those exists.

The Match model is used in PLAP to describe the results received from Moss. It contains details about two students and their code similarities. The model's fields have references to both student's submissions, the similarity percentages, the number of lines matched as well as an overall status rating of the match.

In some cases, when changing match status, you need to add an accompanying comment that will contain information about the change. For this, model *MatchStatusUpdateComment* was created, which has reference to match and contains fields like author, previous status (*old_status*), current status (*new_status*), comment and creation time.

We created a *MatchSimilarity* object to describe the structure of similar parts of the code found in a Match. The model references both students' submissions found in a Match it contains the lines of both students' code, where the similarity was found and their size in lines counts. It is used to display the similarities in students' submissions code inside Match comparison window.

The *PlagiarismRun* model stores information about each plagiarism check. It contains the assignment, start time and references to all matches created by the run. Creating this model allows us to track the plagiarism history for assignments, providing us with unique matches that were found only during the run.

Running plagiarism checks can take up a lot of time. The *PlagiarismRunStatus* model comes very handy in these situations. It's connected with a specific *PlagiarismRun* object and it contains the status and other information about the run. This enables Moodle to keep logs on who triggered the check and how long it took. The object also contains a message, which can convey if anything went wrong when running the check.

No changes were made in *Student* and *StudentDefenseCommit* models. They are connected with submissions through the *Project* model to represent student information. *Student* models store the name and uniid of students, which are later used to connect submissions with students in Moodle. *StudentDefenseCommit* has all the necessary data to clone changes by a user commit.

Due to no need, *rest_framework_api_key* along with the *APIKey* model were removed from the installed applications. They were initially added by Ragnar Rebase to later connect PLAP to Charon. Instead, *Token* models from *rest_framework.authtoken* were now added and are used in token-based communication between PLAP and Charon.

5.3 Authentication

In order to provide a secure connection between Charon and PLAP, we decided to use a user-based authentication solution. Every Charon instance should have a single representative user in PLAP, that has the sufficient permissions to perform all required plagiarism operations. This user is then used to generate an authorization token, which can be used to resolve the user in different requests. Once the authorization token is generated, it needs to be saved in the Moodle instance.

The token will be sent as part of the header of every plagiarism-related request from Charon to PLAP. This enables PLAP to recognize the user, which in turn determines the Moodle instance that initiated the request. This authentication layer is important, because course names in PLAP are not unique and must always belong to the user that created them. This way PLAP can always provide the right resources to the correct user, without having to know anything about Moodle instances.

Following with the single representative user from the previous paragraph, in order to acquire correct information about student repositories and GitLab groups, a user is created in GitLab. A GitLab Access token will be generated for this user and the token will be associated with the user account in PLAP. If the GitLab user has access to certain groups and projects, then the access token can be used in PLAP to clone repositories.

5.4 Automated course and assignment creation

One of the main reasons for the integration was so that there would be no need for an external application to check for plagiarism. Even if all the existing functionalities are integrated, there would still remain the issue of creating the correct courses and assignments. Courses and assignments need to match in Charon and PLAP. For this reason, it makes sense to be able to create a course and assignment in both applications directly from Moodle.

In order to create a course in PLAP, the course must already exist in Moodle. Every course in Moodle, that has been configured with our Charon plugin, has an extra settings page called “Charon settings”. That page contains a section specifically made for creating a course in PLAP. The section consists of multiple fields with descriptions. If the plagiarism system is not responding, the fields are disabled. If a course with the same name already exists, the fields will be filled automatically beforehand and they can only be updated. To create or update a course, all fields must be filled.

Assignment creation does not require the Charon instance to exist beforehand. Assignments can be created at the same time as creating a new Charon in Moodle. The assignment settings have a section specifically for setting PLAP settings. The section contains a portion of the same fields as the course creation. It also has a checkbox, which can be ticked to indicate a creation or update is to take place upon saving the settings. The process is exactly the same as with the course creation. If the plagiarism service is not responding, the fields will be disabled. If the assignment already exists, the fields will be filled with its values. Whenever creating or updating an assignment all fields must be filled.

Fields for the course creation and fields for the assignment creation are more or less the same. The assignment creation contains the same fields as course creation. The extra fields for the course creation are GitLab specifics, which indicate which groups to use to clone projects for the course. In both cases, the following fields are used:

- Programming language type
- GitLab group
- GitLab project location
- File extensions
- Number of Moss passes
- Number of Moss matches shown

5.5 Running checks

Running checks is the most important aspect of this project, since it is what finds the similarities between the files. This has many complicated steps to make it work correctly (Figure 2).

The following figure gives an overview of the steps that are being taken before sending submission files to Moss for plagiarism testing. The dotted line indicates that the next step is a background operation.

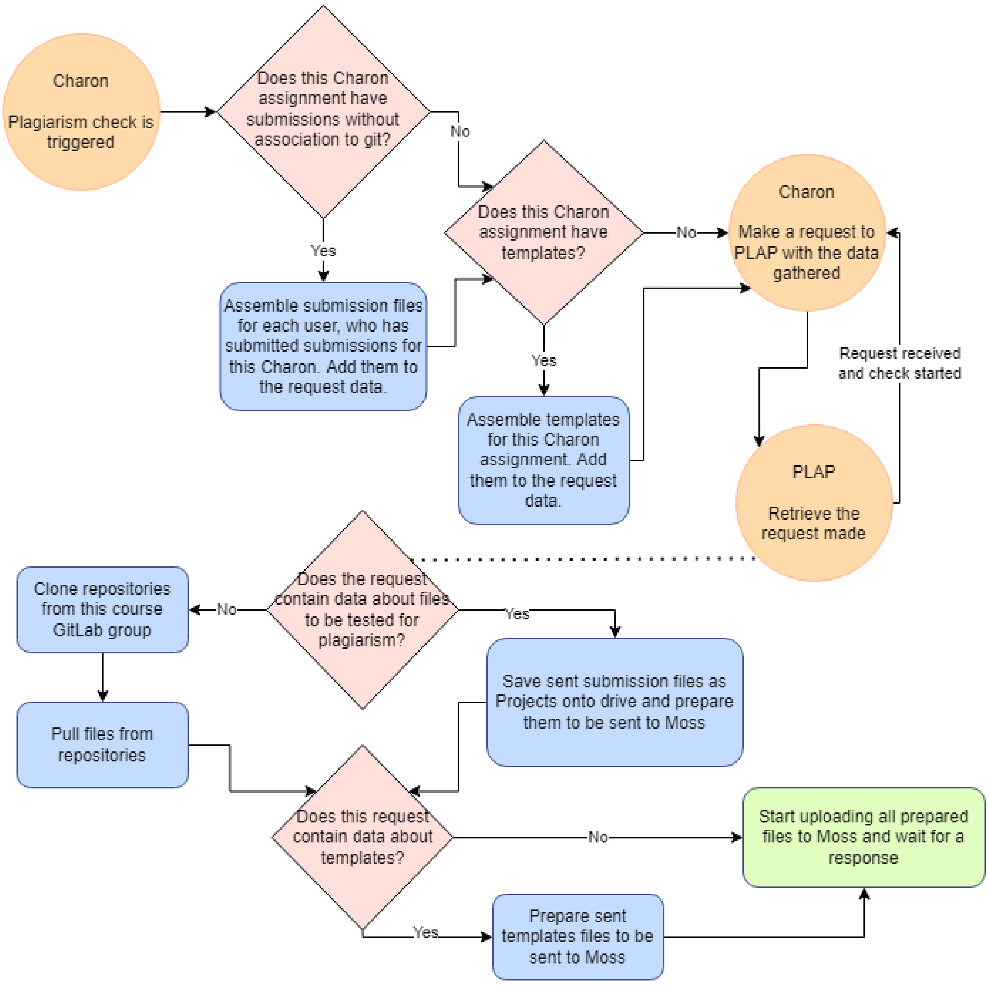

Charon
Plagiarism check is triggered
Does this Charon assignment have submissions without association to git?
No
Yes
Assemble submission files for each user, who has submitted submissions for this Charon. Add them to the request data.
Does this Charon assignment have templates?
No
Yes
Assemble templates for this Charon assignment. Add them to the request data.
Charon
Make a request to PLAP with the data gathered
Request received and check started
PLAP
Retrieve the request made
Does the request contain data about files to be tested for plagiarism?
No
Yes
Clone repositories from this course GitLab group
Pull files from repositories
Save sent submission files as Projects onto drive and prepare them to be sent to Moss
Does this request contain data about templates?
No
Yes
Start uploading all prepared files to Moss and wait for a response
Prepare sent templates files to be sent to Moss

Figure 2. Main flow of uploading files to moss

Running a check fetches submissions from GitLab and sends them to Moss. After some time, Moss returns results to the application, which then are parsed and saved as matches. This entire process requires a course and assignment that have been correctly configured. The user, when running the check, needs to have a valid GitLab access token set. The token allows the user to give the course the correct GitLab group from which student repositories are cloned. Charon requests must contain data regarding the course and assignment which need to be identified in PLAP. In case of submissions that are not associated with git, the code files should also be included into requests.

5.5.1 File management

Files that are sent to Moss need to be stored somewhere, since moss results response is a HTML file and filtering submitted code is time consuming. Instead, files get cloned from students' repositories or passed along with the request that started the plagiarism check. The files are saved on the disk and their paths get saved in a table *Project* before forwarding them to moss. Later, when retrieving the results, the file contents are being read from their respective folders on disk.

Cloning repositories' latest commits is not the correct approach, because the student can create new empty commits after getting a grade for an assignment and bypass plagiarism checks this way. To fix this problem, whenever a teacher assigns a certain submission as *confirmed* in Moodle's Charon popup (an environment in Charon that is meant for teachers to get a better overview and manipulate students Charon submissions), the submissions git commit hash gets sent to PLAP and saved as a *StudentDefenseCommit*. Now whenever a check is being run, instead of cloning latest commit from students' repositories, the commit that had been saved for this student will be cloned instead.

Charon allows to submit submissions that are not related with git, therefore when we want to test these submissions and their files for plagiarism, it is not possible to clone them from Gitlab. Whenever a Charon has this feature turned on, when plagiarism check is being run from Charon, for each user in that course and only for users who have a submission made in that Charon, their latest or defended submissions files are being put together and then sent to PLAP. PLAP creates new instances of projects and saves these files to the drive and continues its usual steps except the step where files are being cloned, because everything necessary has already been saved.

5.5.2 Parsing results from Moss

Matches returned by Moss show exactly what parts of the code between two submissions were found similar. There is made a control to not to save duplicated matches for same students, as Moss sends 2 matches for each pair of students, but we need only one of them.

From the previous version of the PLAP project there was a function that could filter out similar parts of the match, but they were never saved or shown to the end-user. This solution took the code from similar parts and just added them to a list. The problem with this solution was that end-users would be able to only see the parts of the code where it was found similar to another code, but for better user experience this needed to be changed. User should be able to scroll through the entire code and navigate with more ease between these similar parts. Further analysis of the Moss HTML response revealed a better solution, where there was no need to filter out the similar parts.

Moss includes a table of the gap of lines for each file, where there is a similarity found. Filtering out the exact lines where the similar block was found would be a much better approach. This would mean that less data would be needed to save to the database and with this solution the similar blocks could be differentiated from the whole code with various colours for easier recognition. For a match's similar parts, a new database table was implemented – *MatchSimilarity,* which saved both of the files similar parts lines and their section sizes.

Initially PLAP did not support saving nor displaying the history of its matches. In order to implement this functionality, we created a new table called *PlagiarismRun*, which would store each run's start time and the matches that were connected to it. Previously, PLAP would reuse older matches or even rewrite them in case of new submissions. The newer solution does not rewrite older matches. If the same match is found, nothing will happen, however if a new match appears, it gets saved.

5.6 Displaying results

5.6.1 Main matches table

When a plagiarism check is finished, the matches found during the process are saved into the database. Matches are submission comparisons performed by the external application Moss. They show us similarities found between two submissions, which helps determine whether someone has committed plagiarism. Fetching the matches of an assignment is necessary to find plagiarism between students' submissions. To show all matches for a run, Charon first fetches all runs for it and then fetches the latest runs matches and shows them inside the main matches' table (Appendix 3).

The table consists of following columns:

- Lines matched – the number of similar lines found in the match.
- Uni-ID (Digital identity for a person in TalTech) – the uni-id of the student in question.
- Percentage – the percentage of similarities for the student in question.
- Other Uni-ID – the uni-id of the other student in the match.
- Other percentage – the percentage of similarities for the other student.
- Status – the general status of the match.

- Actions – a group of actions that can be performed on the match.

This table also has a few possibilities to filter between the data and they are:

- Search – makes it possible to search in the first five columns (Uni-ID, Percentage, Other Uni-ID, Other percentage).
- Status – makes it possible to filter between the three statuses (New, Plagiarism, Acceptable).
- Runs – makes it possible to fetch matches for a certain run by its time.
- Percentage – using slider, there is ability to set the interval by which the filtering will be performed. Using toggle on left, user can choose to filter by both *Percentage* and *Other percentage* fields or one of them.

A match's general status describes the state of the match in a plagiarism context. There are three possible states in which a match can be: New, Acceptable and Plagiarism. Teachers can change the status of a match via one of the actions presented.

Actions are activities that can be performed on the match. They are used to show additional information or to change the status of a match. Currently there are three actions that can be performed: viewing the comments, viewing the code comparison and marking results.

There is a possibility that during the check something goes wrong and *PlagiarismRun* has no matches connected to it. These runs would not be displayed in the run selection box in order to not confuse teachers, as the runs would have no matches to show. Although these runs do not have any matches to show, they are still shown in the history of all runs section.

5.6.2 Student overview matches table

For every student in Charon there exists an overview page in the Popup. It gives an outlook on the performance of the student for the given course. Among grades, submissions and other metrics, it also displays information about the student's plagiarism progress. When opening up the plagiarism progress section, a table of matches is shown (Appendix 4).

The plagiarism history table is an extension of the main matches table, with the main difference being that it contains three additional columns and the matches displayed in the table are only for the specific student in question. It also doesn't have a scroll bar for filtering the matches by percentages. The three new columns are as follows:

- Created at – the date and time when the match was created.
- Activity status – a status indicating if the match is new or not.
- Charon – the name of the Charon for which the match was found.

The activity status is a special kind of status, it is meant to indicate whether the match is new or old. By default, we want to display only new matches, since fetching older matches takes longer than newer ones and teachers are generally more interested in the latest results. However, in order to fetch all older matches as well, a toggle can be switched and the results are automatically fetched.

5.6.3 Student assignment statistics table

The student's overview page's plagiarism section contains an additional table (Appendix 5), which provides a summarized overview of the student's progress for each assignment. It is meant to give teachers the ability to quickly check whether a student has a plagiarism for a certain assignment, as well as other metrics regarding the assignment. The data for the table is calculated based on all of the student's matches that have been currently fetched.

The table contains the following columns:

- Charon - the name of the assignment.
- Assignment status – the plagiarism status of the assignment.
- Max lines matched – the highest number of matched lines for the assignment.
- Max percentage - the highest similarity percentage for the student.
- Max other percentage – the highest similarity percentage for the other student.
- New amount – the number of *new* matches.
- Acceptable amount – the number of *acceptable* matches.
- Plagiarism amount – the number of *plagiarism* matches.

The assignment status shows whether a match with the status *plagiarism* exists for the assignment. If a single match like that is found, a red chip with text “Plagiarism” is shown, otherwise the chip is green with text “Acceptable”.

Based on the amount of different statuses each assignment possesses, a third table was added as well (Appendix 6). It aims to give an easy oversight of every assignment and the ratio between different statuses it contains. This way teachers have multiple options, whether to examine the table with different metrics or to view the second table, which shows only the ratios of the statuses.

5.6.4 Code comparison

Comparing different solutions is a task teachers need to perform manually most of the time. Algorithms can be mistaken if given the opportunity to determine the outcome of a found match. There needs to be a comparison window, where both codes can be examined, and the outcome determined by a teacher. For this reason, every row in the found matches table contains a modal view button. Opening this modal will allow teachers to view the two submissions side-by-side.

This modal is divided into 5 sections (Appendix 7). For each of the students there is a section that describes the student and their overall match information:

- Username, which is in most cases the student's uni-id.
- Percentage of matches lines inside the file.
- Commit hash, if it exists.
- Button “Student overview” that directs to student overview page, where teacher can search for matches by this student.
- Button “Submission” that directs to the view of that submission, where teacher can assign grades and add comments to that submission.
- And a button “GitLab” if a commit hash exists to this student’s commit in GitLab.

All the previous buttons open a new page in a new tab.

Also, for both of the students there are their corresponding code sections for viewing the submitted files, which have their similar blocks differentiated and 1 section for navigating between these similar blocks. Both of the code sections have their similar parts shown with a background color on those lines that they matched at. Both students’ similarities have the same color if they are interconnected.

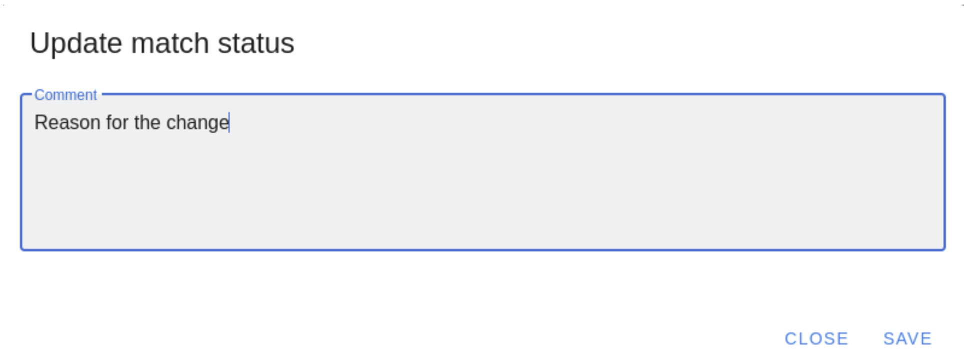
The navigation section is a table that consists of 3 columns:

- The first and third column show which lines are similar in each of the students’ code and when clicked they open the section in their corresponding code section.
- The second column shows the overall size of that section lines and if it is clicked will open both of the code sections at their corresponding lines.

This table’s rows have the same color as the similar parts in the code sections in order for it to be easier to notice them.

5.6.5 Match comments

Every match can contain comments. Teachers are required to add comments when they are marking results (Figure 3).



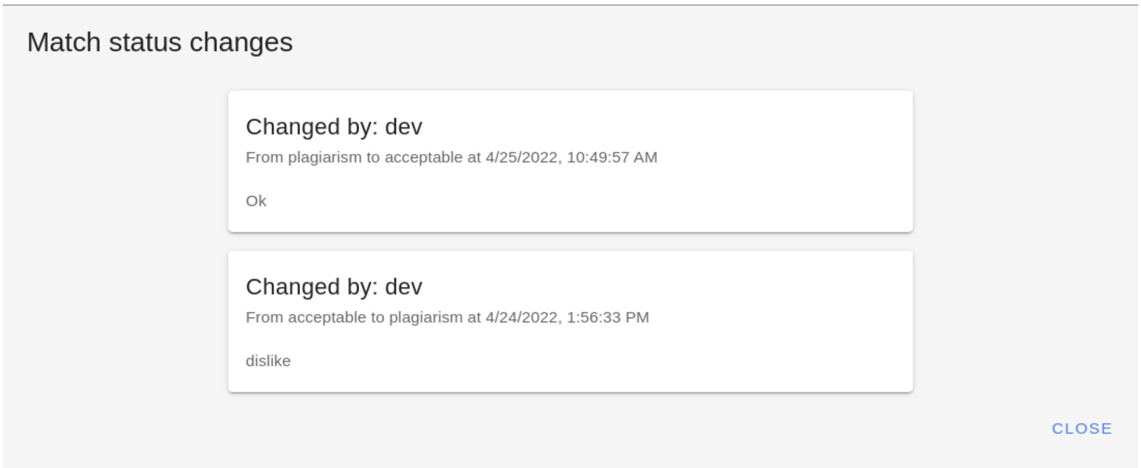
Update match status

Comment

Reason for the change

Figure 3. Adding a comment for a match status change

In both matches tables, one of the available actions is to open the comments section, which is represented by a button with a comment box icon (Appendix 4). When the icon is crossed out, then no comments have been added, and when it’s not crossed out, then comments exist. The comments are displayed in a scrollable list (Figure 4) and every comment has additional details, for example who wrote the comment, what kind of status change was applied and when this occurred.



Match status changes

Changed by: dev

From plagiarism to acceptable at 4/25/2022, 10:49:57 AM

Ok

Changed by: dev

From acceptable to plagiarism at 4/24/2022, 1:56:33 PM

dislike

Figure 4. Match status change comment section

5.6.6 Graph components

When adding graphical components, we wanted to recreate the same graphs as PLAP had: a bar chart, a donut chart and a network chart. All three graphs display different types of information about the fetched matches. As these graphs are made from matches displayed in main matches table, they display graphs based on currently showing matches in table. Even if user is fetching matches for a certain run by its time, it is possible to see graphical visualization of them.

Graph components have their own filtration by allowing or disallowing including matches by status. For this is used toggle for each status, by default all statuses are on. So, there is a possibility to disallow some statuses from graphical display, which may be useful to teacher for analyzing results and makes matches status marking more meaningful.

The bar chart (Figure 5) is divided into multiple different columns. Each column represents a percentage range of similarities found when comparing two different submissions. All matches with a similarity percentage, that fall in between the given range, are grouped under the column. The graph allows teachers to easily find the number of matches with very high similarities.

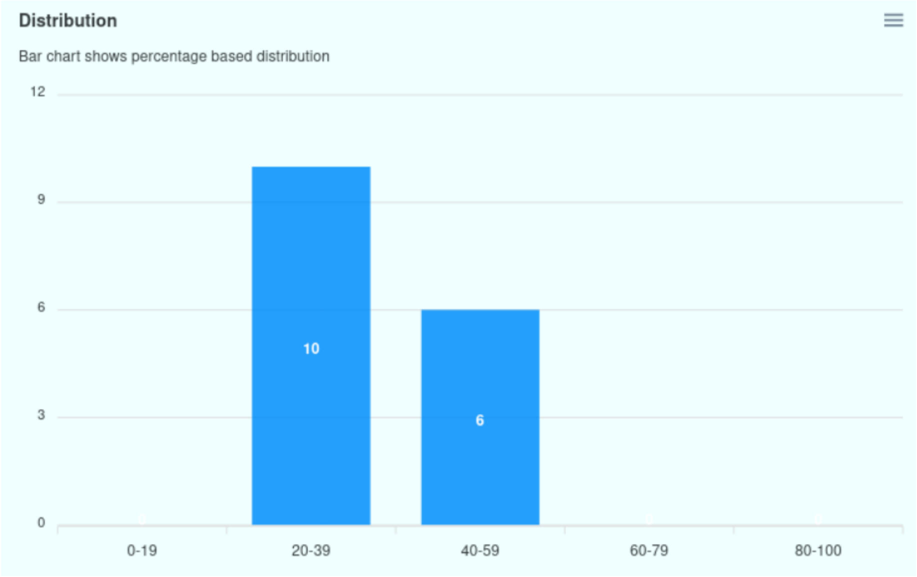


Figure 5. Distribution bar chart

The donut chart (Figure 6) shows us the distribution of the matches statuses. There are three statuses in total: new, acceptable and plagiarism. The text in the upper left corner indicates match count.

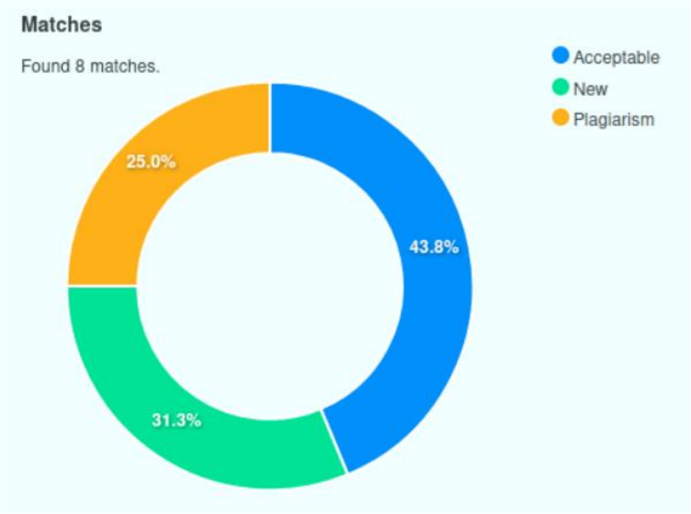


Figure 6. Distribution donut chart

The network chart (Figure 7) is by far the most interesting graph. It displays all of the students found in the matches as nodes and connects each one via edges to other students that they share a match with. This approach can therefore give a unique visual overview of how one student is connected to others and which students make more allied groups.

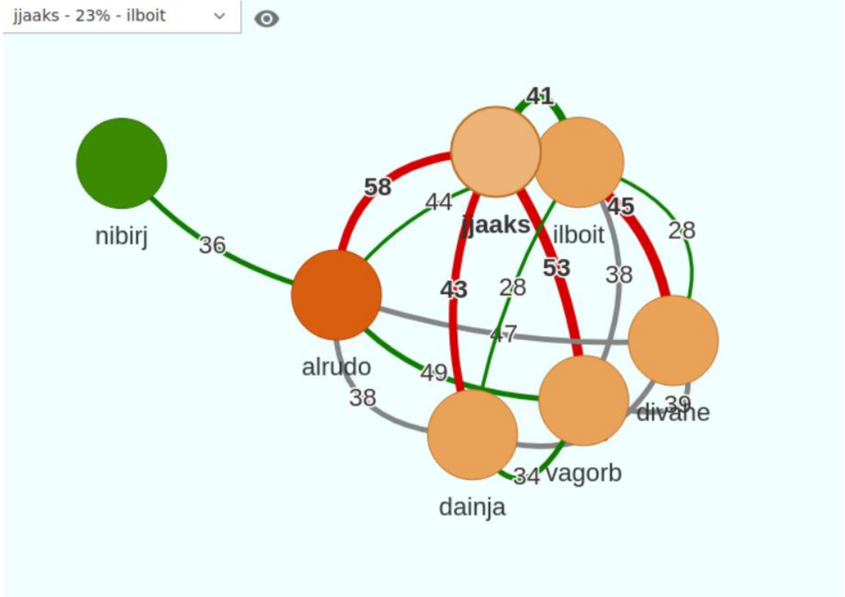


Figure 7. Matches network graph

Main part of this chart is node and edges stylization. Edges are colored in accordance with match status between two students as nodes, where acceptable is green, plagiarism is red and new is grey. Match average percentage is shown in center of the edge and edge thickness corresponds this number, so the bigger percentage the thicker edge. This brings to the fore matches with bigger percentage, while edge with lower number will be less noticeable. Nodes are colored by number of connected to them edges from green, which means just some connections, yellow, when there is couple connections, and red, when there is 5 or more connections.

Edges are nodes are also clickable. When clicking one of them, in the upper left corner there will appear select bow, where you can select match and eye button. Select box shows all matches connected to node, but if to click only for edge, there would be the match, which is behind particular edge. Then, by clicking on eye, code comparison popup will be opened.

Full view of graphical components window can be seen in appendix 8.

5.6.7 Checks status

At the bottom of the plagiarism page in Charon there is a check running section with latest check status (Appendix 9). By choosing Charon and then running check, there would be shown a status of current latest check. Every 5 seconds the status would be updated.

The table consists of following columns:

- Charon – the name of the assignment.
- Author – teacher full name, who run check
- Created at – run check start time
- Updated at – time of the last status change
- Status – state of check’s latest status

There is toggle button in the right corner. By clicking it, history of all checks for the whole course assignments would be shown. It is sorted by date with the same columns as for the latest check but to the status column an added arrow button to open full status change logs for the particular check. There would be shown each added status and its’ time when the status was added. History of all checks can be seen in appendix 10.

5.7 User flow

This project creates a new page and sections in Charon for teachers to use. Since the integration of PLAP with Charon was meant to let teachers make plagiarism checks and view results without having to open and log in to PLAP, all newly added pages and sections are implemented in Charon.

5.7.1 Charon Plagiarism page

This page is part of the Charon Popup and it consists of:

- Plagiarism matches section
- Plagiarism overview section
- Latest check/history of checks section

For teachers this is the main page they will be using. On this page teachers can:

1. Run plagiarism checks for Charons, that have their counter parts created in PLAP. See status of current/latest run check.
2. View history of checks and their information (Charon name, Author name, Created at, Updated at, Status/Logs)
3. Fetch matches for Charons, that have their counterparts created in PLAP. If the Charon does not have any runs/matches then no data is shown. Filter between fetched matches, using Search box for all columns except Status and Actions and using Status select box for the column Status. Fetch matches between different runs to see their history. (Lines matched, Uni-id, Percentage, Other Uni-id, Other percentage, Status, Actions)
4. Open up a page popup called MatchModal, to see more information on that specific match.
5. Inside MatchModal open new tabs for pages Student overview, Submission and GitLab associated with that matches student.
6. Inside MatchModal navigate through both student's code.
7. Assign a new Status for a match. (Plagiarism, Acceptable)
8. Load and view graphs for the fetched matches.
9. Open up MatchModal inside Network Graph for a selected node or a selected edge.

5.7.2 Charon student plagiarism section

The student plagiarism section is part of the Charon Popup’s Student overview page. The page contains multiple Charon-related sections, however there is only a single section for displaying the student’s plagiarism results. The section is meant to give an overview of the latest as well as historic matches for the student.

On this page, teachers can:

10. Fetch the latest matches of a student.
11. Fetch all historic matches of a student.
12. View the code comparisons of matches.
13. View the comments of matches.
14. Mark statuses.
15. View the statistics for all assignments.
16. View the student’s progress for each assignment in regards to match statuses.

5.7.3 Marking results

Marking results means to essentially change the status of a match. Teachers need the ability to mark a match as *Plagiarism* or perhaps as *Acceptable*. For that they need to use the appropriate action listed at the end of the match's row in one of the matches tables. The action is a button with a green thumbs-up or red thumbs-down icon (Appendix 3). Whenever the status of a match changes, teachers are prompted with a comment box, which they have to fill, in order for there to always be a reasoning for the change. The only exception to this is when the status of a *New* match is being changed to *Acceptable*, in which case the comment is not required.

5.8 CI/CD

5.8.1 Deploy and Tests

Both applications are deployable onto a server. Since Charon has been continuously developed and deployed quite often, it has working deployment. Ragnar's project has been deployed on to a working server and had a working script to deploy it to any server. But since it uses dependencies and packages that are outdated, to deploy it we are looking for another way to get it on to a working server using GitLab CI/CD. So far, the deployment itself is working but we have not gotten the server setup just right.

For both applications unit tests are made to ensure that everything would still work if some functions overlap with multiple functionalities and when changes are being implemented. Charon already has a dedicated working pipeline job to run tests with every commit, using GitLab CI/CD. For PLAP tests are created but again the pipeline job is outdated and needs to be changed to work and test correctly. PLAP also has the ability to test code quality.

6 Validation

Validating our work is an important part of analysis, because at the end of the day, the teachers are the ones who are going to be using these features. Therefore, they need to know how to properly use them and the front-end UI/UX (User interface/User experience) needs to be understandable with the flow being simple yet effective. We have gotten validation for the integration from demonstrating it to teachers who are using Charon, the committee and from our supervisor, who also uses Charon.

The main validation aspects that we were looking for when demonstrating our project were as follows:

- Is it understandable how our projects/applications work?
- Are they satisfied with the front-end design?
- Does our solution suit their needs?
- Do they have any suggestions regarding the flow or design?
- What kind of additional features would they like to see/use?

6.1 Validation from supervisor

Our supervisor Ago Luberg is the one who requested the integration between Charon and PLAP, therefore he was also our client. Thanks to him, we were able to obtain a better understanding of PLAP, since it was a little outdated and we were initially having difficulties setting it up locally as well as understanding how it worked under the hood.

Since our supervisor had previous experience working with PLAP, we turned to him once we got PLAP set up locally and started understanding how it operated. We were mainly concerned with what kinds of features he was interested in, as well as what different sorts of requirements needed to be met when integrating the applications.

We had daily meetings with the team and weekly meetings with our supervisor, where we discussed the work that had been done, problems that had occurred and which next steps to take. During every meeting, we would explain and demonstrate our solutions, in order for them to be validated and to keep our client in the loop of how things were progressing. We put the most emphasis on our client’s feedback, as he was the one who would likely be using the integration the most.

Thanks to his feedback we were able to implement a better flow for running plagiarism checks. One of the more problematic parts of the integration was how we would send submissions from Charon to PLAP. Initially, since we wanted to distinguish Git submissions from non-git submissions and defended submissions from non-defended submissions, we contemplated getting rid of Git altogether and sending all files straight from Charon. However, git was a useful utility resource and keeping it was beneficial. Thanks to our supervisor, we learned that it was possible to clone submissions from commits specifically, and therefore we were able to keep the Git and regular submissions separate while distinguishing defended and non-defended submissions.

Another thing that was unfriendly was our initial section, where we showed the similarities between two students' submissions. Our initial solution to this that we would show only the similar parts, but this turned out to be more of a bad thing, because some similar parts have no context and it would be hard for the teacher to understand what is actually done there. So, a suggestion was to show the full code and differentiate the similar parts with a color background on those lines, which makes comparing two files a lot of easier.

We also received feedback about what should happen when a match's status is changed. If a match's status changes, it's something that could possibly end with a student getting exmatriculated, so it needs to be taken seriously. For this reason, whenever a status was to be changed, a comment needed to be left behind as a way of reasoning for the change. Teachers needed the ability to look back and find some information as to who changed the status and what did that person find that made him make such a decision.

6.2 Validation from teachers

During the development after we reached points where bigger functionalities got done, we asked for feedback from some teachers who have had experience with Charon and could better answer on how the flow of the entire application was. With this we got feedback that the network graph that had been implemented could do more. For example, there would be the possibility to select a node and the match comparison view would popup or the same action would mark a match or multiple of them as *Plagiarism*. The first suggestion we took heed of and implemented that if a node or an edge is selected the comparison view will be able to be selected. Another idea was to show only nodes that have statuses *New*, so the teacher would be encouraged to minimize the shown nodes.

7 Results

We started working on this project at the start of this semester, from February. Since Charon is being kept up to date and we have been developing it for over half a year, we had no big problems there. As for PLAP, this was a few years old project that had not been updated and tested regularly. It took a few weeks to get PLAP fully working for our development and a few weeks to fully understand how it worked, since we had not worked on a Django project and the idea of how it check files for plagiarism was difficult to understand at first. After a few weeks into the semester, we were finally able to start developing and making changes to integrate both applications.

7.1 Comparing results to Task proposal

Comparing for what tasks we set for ourselves in *Task proposal*, we got more done than we first anticipated. We got to integrate all that PLAP had to offer to Charon and even a few additional changes in PLAP check flow and few more added functionalities for the view components that were being done similar in PLAP.

The following will list functionalities and important tasks that got done:

- Automation of course and assignment creation from Charon
- Running checks from Charon
- Fetching matches from Charon
- Parse exact lines where similarities were found and display these lines with different color as to notice them
- See overall tables and statistics from Charon
- Finish Ragnar’s initial integration functionality, on how the student files were cloned from a given commit
- Implement history for runs and matches
- Updated graphs visualization
- Student view page to see overall information about student
- Current and previous checks status displaying

Because this was a new part for Charon there are still many possibilities and things to do to make the flow and overall aesthetics better.

7.2 Teamwork and what we learned

We have known each other for years and worked on Charon together since the start of 2021 summer, we had a good sync and knew how working in a team should go. We had meetings 3 times a week, where only a few were skipped, because of personal issues. Because of these meetings we were always on the same line and knew the things we did. Since we all wanted to know how some functionalities worked and what could be done to better them, we also did pair programming a few times a week.

Things that we learned while working on this thesis and both of the projects were how Django framework works, how it connects to Moss and how to implement security measures to make everything safe. We also learned how to discuss and bring out ideas while having meetings with the client for a project that we were fairly new to. We were happy with it, because at every meeting with the client we had something to show and important to discuss about.

8 Conclusions and next steps on development

Overall, the development went well, but there are some things that still need some attention and time to get them to a finished state. Since we as a team had our tasks and everything planned out for the entire semester, then these things were what we focused on and some additional things we decided to not include everything in to the end product as to make sure that what we planned out would get done and the correct way.

8.1 Conclusions

Most important part that did not get implemented fully was the deployment of PLAP on to a server. This is because Ragnar used packages that currently are not supported and outdated to deploy the project on a server. Hence, we could not deploy it the way Ragnar initially did it, so we had to find other solutions, but this problem turned out a little bit too late in the development. To consider that we ourselves do not have good experience in building the project and deploying it, it took us quite a lot of time to even understand how it should work. We are somewhat familiar with GitLab CI/CD and configuring jobs, so we tried to get the deployment working this way. What we could've done better would have been to try to deploy some of our earlier versions, but at that time we were still getting familiar with PLAP and how it works.

Another thing we should have done was to plan out more time for validation, to demonstrate the project, our solutions and get feedback. Analyse feedback and find a consensus with the teachers and implement their wants and needs. Even though we had some demonstration and got some validation we were not able to implement everything what they wished for. An example would be that one teacher wanted to assign status straight from the comparison window, but since we had to change how the status was assigned to a match fairly late in the development, we simply did not have time to implement it.

On a good note, we got more things done if we compare the result with the task proposal. We were not quite optimistic about how much we could do as a team, because everything associated with PLAP was new to us, including some technologies and the overall idea behind PLAP. Everything needed to be discussed and that took time. But in the end, we got everything we set out to do and even more done.

8.2 Next steps on development

This paragraph describes next possible steps on improvement and development of overall thesis.

8.2.1 Deployment and more validation

Because we could not get the project deployed in time, the next step to improve both applications would be to get it deployed correctly and get more validation from teachers. Until now we have gotten validation from others by demonstrating our application by sharing screen in a video call or by creating videos and sharing them. This way they might not know at that moment what they would like to change or what they would want to get implemented. And to add to that interacting with the applications itself would be a lot more intuitive to get others using the application and getting better feedback. This way they can try it from their own chosen time and helps us find some other workarounds and new possible features.

Their feedback would also make the UI more intuitive, as the teacher would get a clearer overview of the changes that should be implemented. The most needed step is to deploy changes continuously and let teachers try the changes themselves to guide further development better.

8.2.2 Moss alternatives

Similar to what Ragnar said in his thesis, Moss is not a reliable service to send files for testing. We found out that the system was quite often offline or reaching response timeout in several hours. This truly might be an issue when teachers are actually going to use it, as well as it makes development slower if we need to upgrade things dependent on Moss. To fix this it might be possible to implement a few other services that could test files for plagiarism or implement our own service for checking code for plagiarism, which can be customized to the needs of Charon and Plagiarism.

8.2.3 Testing plagiarism with previous years courses

The current solution does not take into account data from other courses and the results of previous years. In order for the plagiarism system to give better results, the use of already existing submissions and matches from other courses will identify students who use student reports from past years as a basis. Since course materials are repeated year after year, it is more likely that some students will pass on their done work to future generations.

8.2.4 Plagiarism visualization

Since the detection of plagiarism is a rather complicated process and not always found similar matches is cheating. For better understanding of the existence of plagiarism, graphs have been created, both under the results of the check and on the student's page. Further improvement in data visualization could greatly help the teacher in identifying plagiarism, so based on them the teacher determines the actual result.

The current solution is informative, but it does not include a visual representation of all the Charons in the course and comparison of results between different groups of students. Also, if the submission is a group submission, then all its authors have no connection with it and the uselessness of comparing messages within the same group is not taken into account.

9 Summary

Plagiarisms are highly frowned upon, especially in academic environments. It’s basically the theft of another person’s intellectual property. TalTech is no different in that regard, it values authenticity and tries to minimize plagiarisms as much as possible. A large portion of its IT department students are programmers. These students face a lot of programming exercises during their first bachelor’s year. Naturally, this is an area where many plagiarisms could happen, as students tend to help each other by sharing their solutions with others. Programming code plagiarisms are more difficult to detect as well, as solutions can be similar yet authentic. This poses an adequate problem to solve.

Luckily, PLAP had been already developed for TalTech. Our solution to the problem was to therefore integrate PLAP with our programming exercises management plugin Charon. Teachers needed to be able to access all of PLAP’s functionalities from within Moodle. The main idea was to improve the user experience in order to get more teachers to use the plagiarism checking system. The integration presented the opportunity to add new features as well, which we used to ask teachers about their opinion on our work and what they would like to see.

All of the previous PLAP functionality were successfully implemented. Some of it had to be changed, due to new conditions that the system had to meet. The new PLAP integration functionality supports submissions without GitLab commits as well now. Teachers received a brand-new dashboard, where they have a lot more control over plagiarism checks than before. In addition to that, they have been granted a lot of data visualization. For the most part, tables and graphs, which are used mainly for code comparison and statistical metrics. Matches can receive ratings, rating them as acceptable or not. If a single user is suspicious, then that student’s entire plagiarism history can be reviewed in the Student Overview page. Teachers now finally have the ability to interact with their plagiarism detection software in a comfortable way.

References

[1] R. Rebase, “Lähtekoodi sarnasust tuvastava süsteemi arendamine,” TalTech, Tallinn, 2019.

[2] A. Aiken, “A System for Detecting Software Similarity,” Stanford University, 2022. [Online]. Available: http://theory.stanford.edu/~aiken/moss/. [Accessed 03 05 2022].

[3] J. F. M. Alviste, “Programmeerimisülesannete automaattestimissüsteemi liidestus moodle'i keskkonnaga ja mugav kasutajaliides tudengite haldamiseks,” TalTech, Tallinn, 2017.

[4] “Apexcharts.js,” ApexCharts, 2022. [Online]. Available: https://apexcharts.com/. [Accessed 2 5 2022].

[5] “Build for Code,” 2022. [Online]. Available: https://ace.c9.io/. [Accessed 2 5 2022].

[6] “Network,” [Online]. Available: https://visjs.github.io/vis-network/docs/network/. [Accessed 09 05 2022].

[7] “CI/CD concepts,” GitLab, 2022. [Online]. Available: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration. [Accessed 02 05 2022].

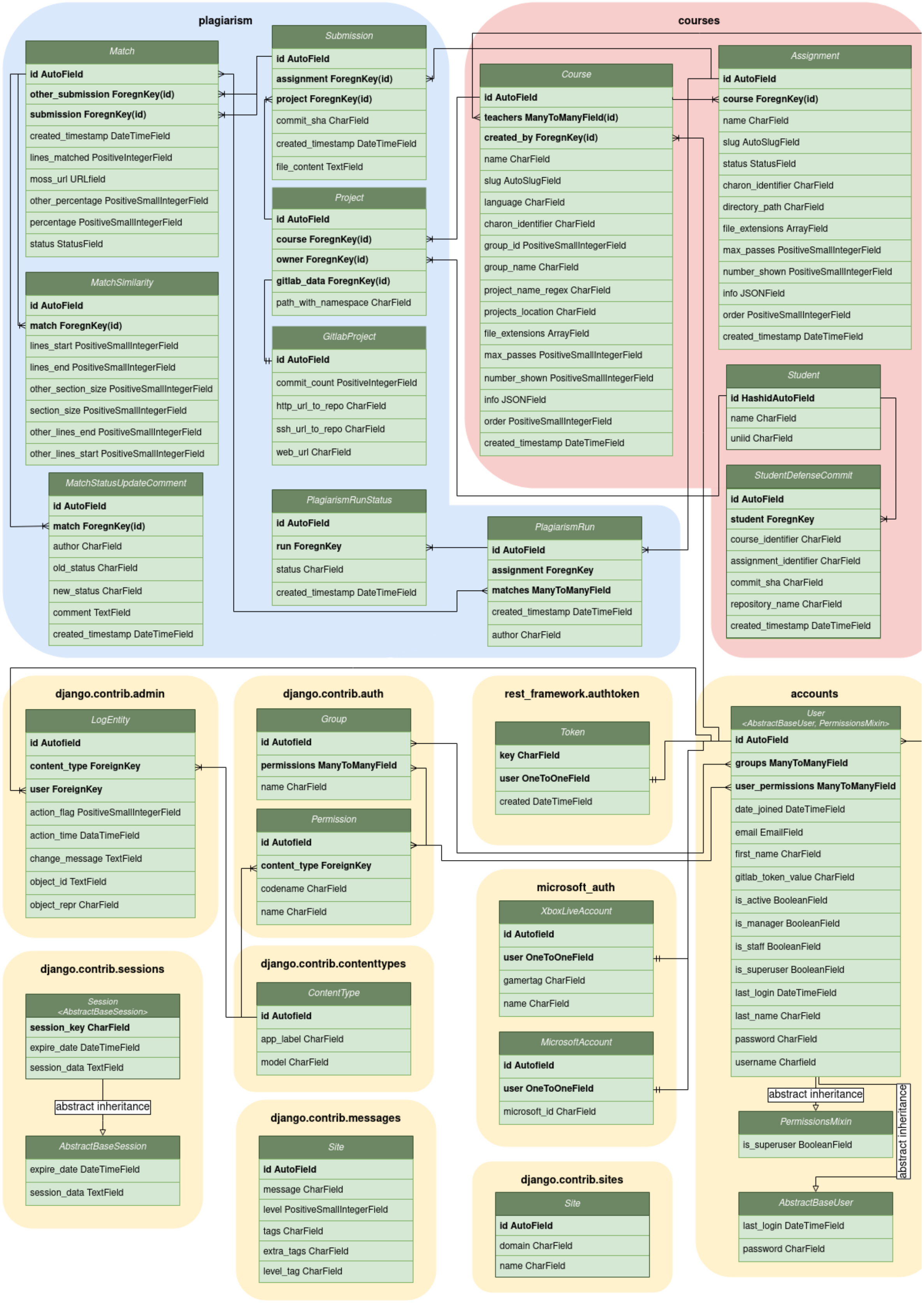
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

We, David Avedis Injarabian, Janar Keit Jaakson and Ilja Boitšuk

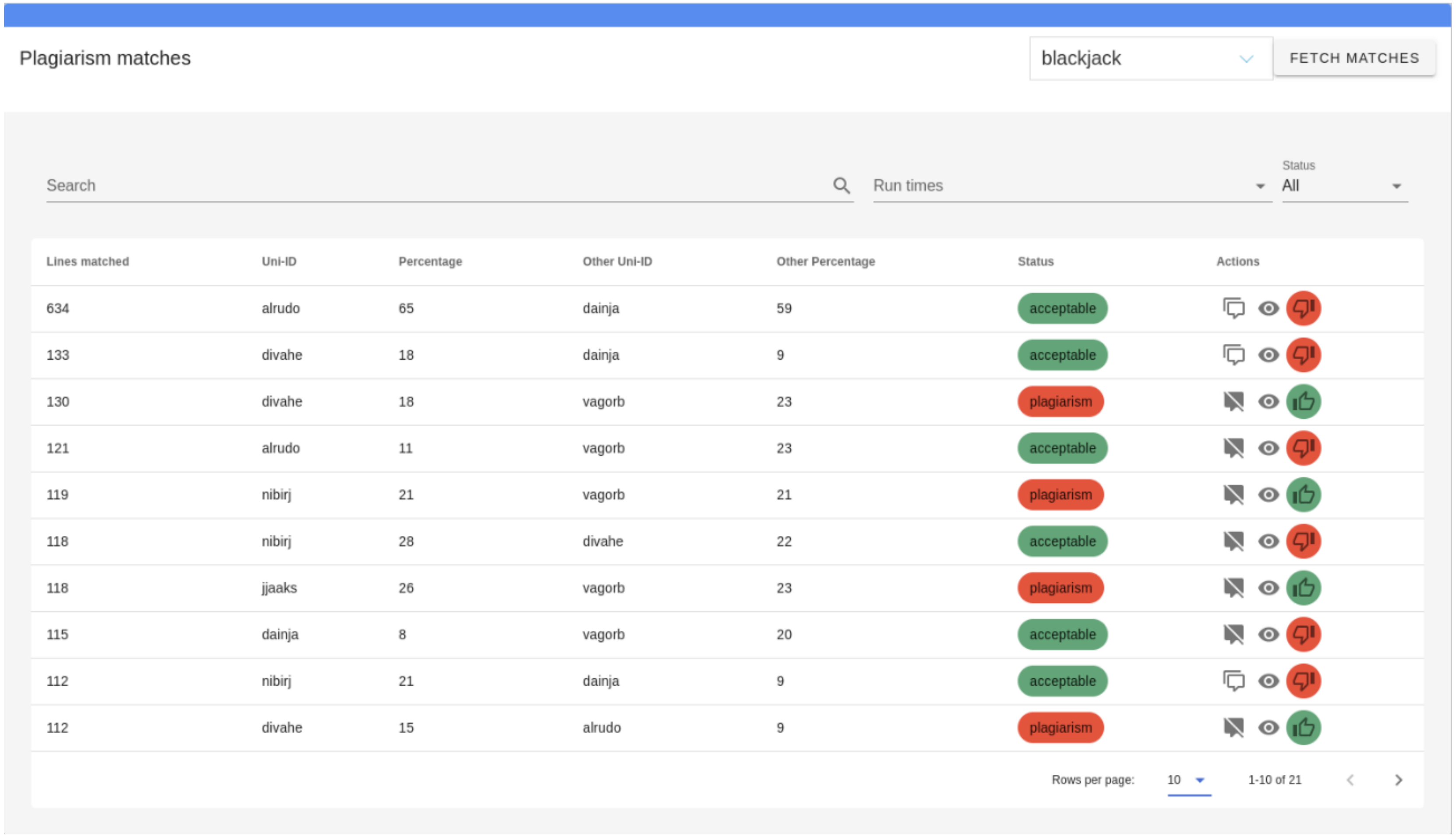
1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Plagiarism system integration with Moodle’s plugin Charon, supervised by Ago Luberg
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

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

Appendix 2 – All PLAP models



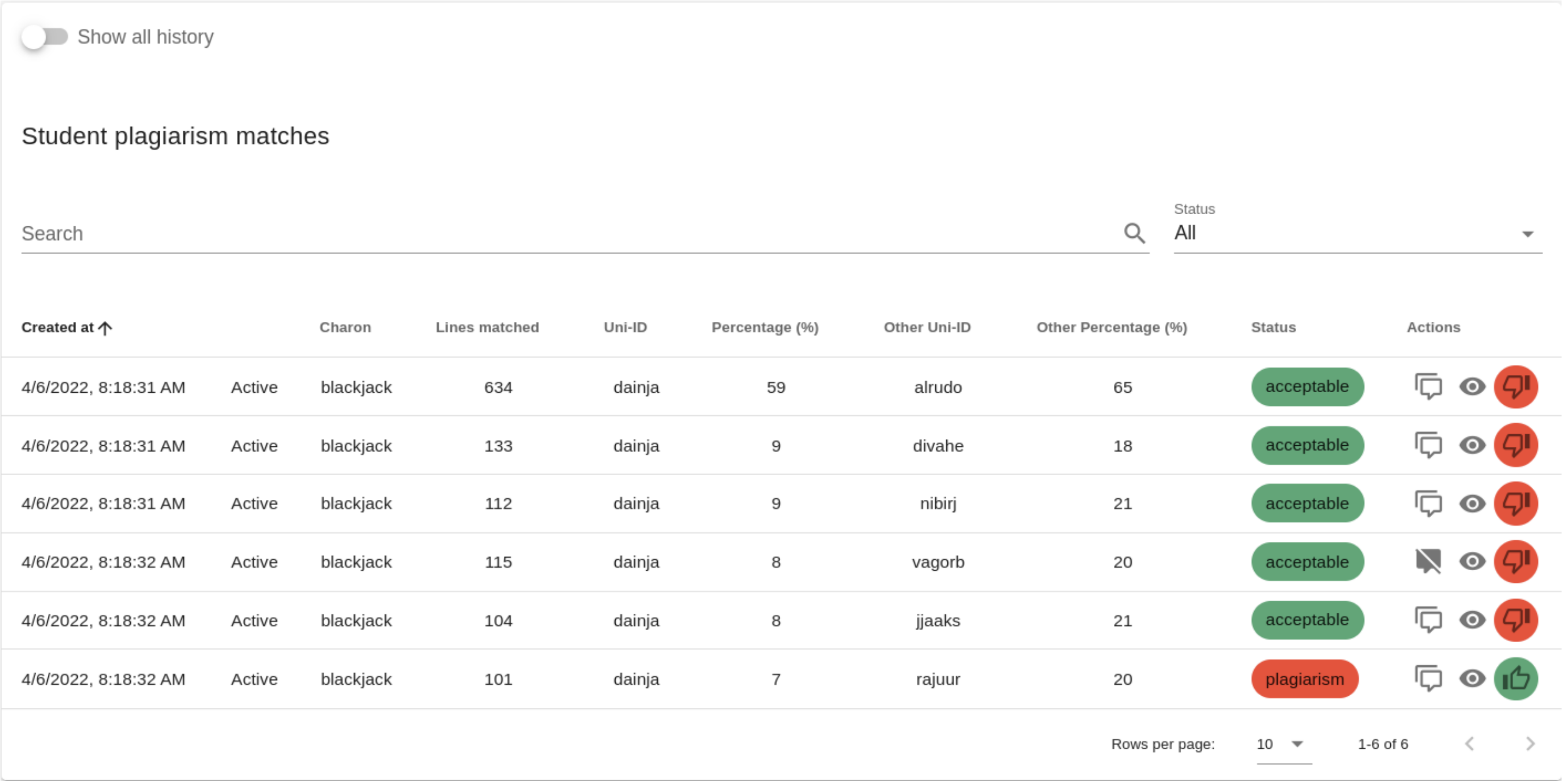
Appendix 3 – Main matches table



Plagiarism matches

Lines matched	Uni-ID	Percentage	Other Uni-ID	Other Percentage	Status	Actions
634	alrudo	65	dainja	59	acceptable	
133	divahe	18	dainja	9	acceptable	
130	divahe	18	vagorb	23	plagiarism	
121	alrudo	11	vagorb	23	acceptable	
119	nibirj	21	vagorb	21	plagiarism	
118	nibirj	28	divahe	22	acceptable	
118	jjaaks	26	vagorb	23	plagiarism	
115	dainja	8	vagorb	20	acceptable	
112	nibirj	21	dainja	9	acceptable	
112	divahe	15	alrudo	9	plagiarism	

Appendix 4 – Student matches table



Show all history

Student plagiarism matches

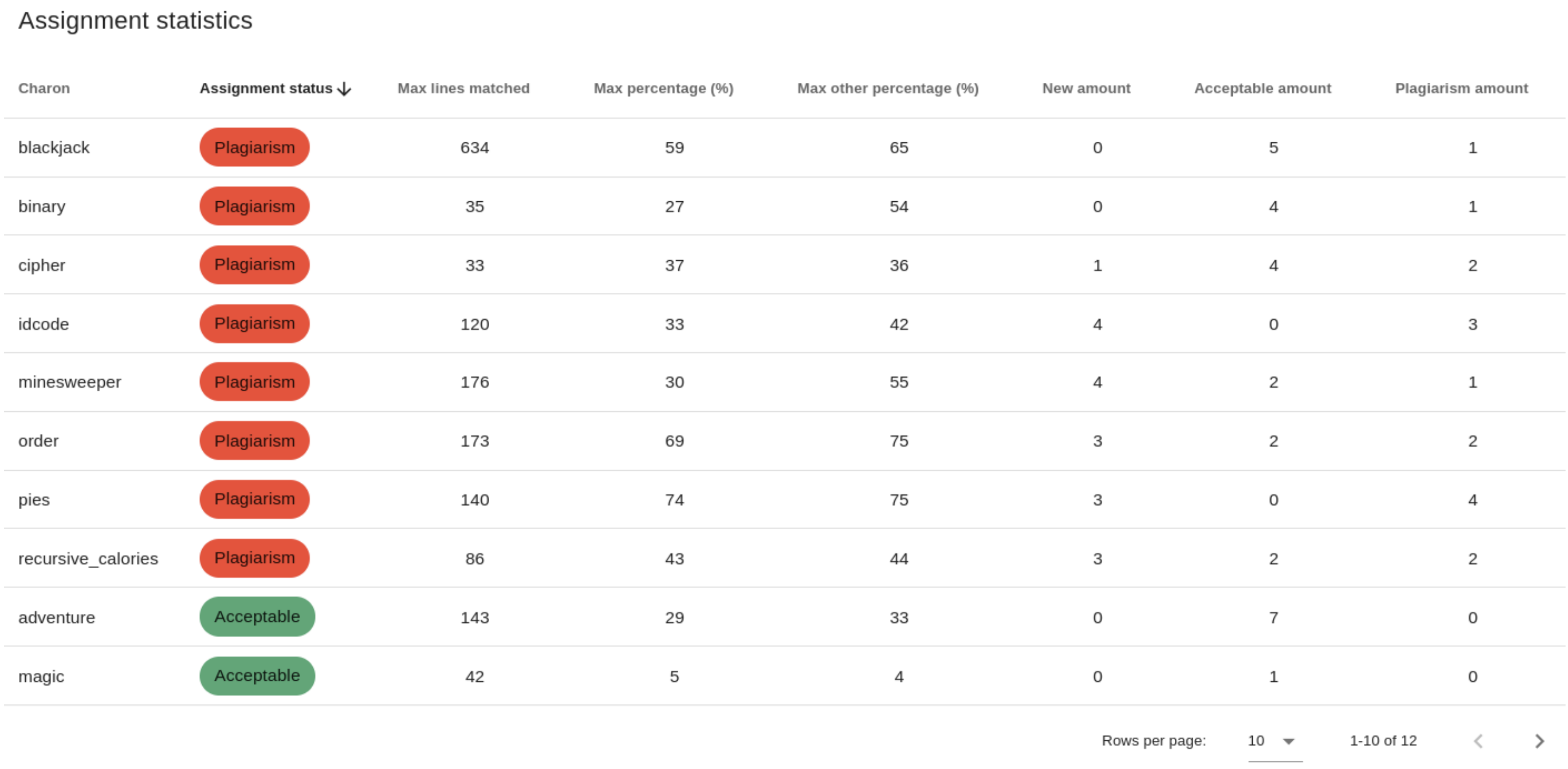
Search

Status
All

Created at ↑		Charon	Lines matched	Uni-ID	Percentage (%)	Other Uni-ID	Other Percentage (%)	Status	Actions
4/6/2022, 8:18:31 AM	Active	blackjack	634	dainja	59	alrudo	65	acceptable	
4/6/2022, 8:18:31 AM	Active	blackjack	133	dainja	9	divahe	18	acceptable	
4/6/2022, 8:18:31 AM	Active	blackjack	112	dainja	9	nibirj	21	acceptable	
4/6/2022, 8:18:32 AM	Active	blackjack	115	dainja	8	vagorb	20	acceptable	
4/6/2022, 8:18:32 AM	Active	blackjack	104	dainja	8	jjaaks	21	acceptable	
4/6/2022, 8:18:32 AM	Active	blackjack	101	dainja	7	rajuur	20	plagiarism	

Rows per page: 10 1-6 of 6

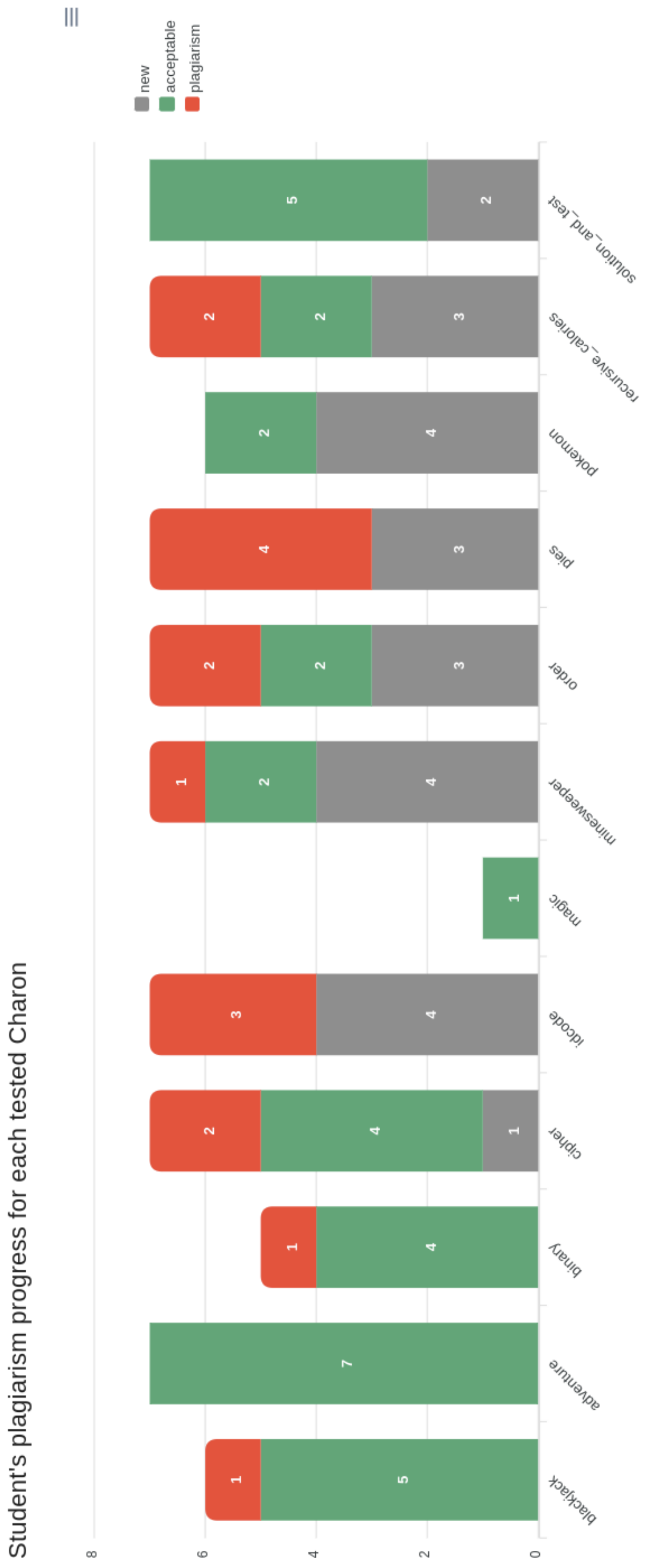
Appendix 5 – Student assignment statistics



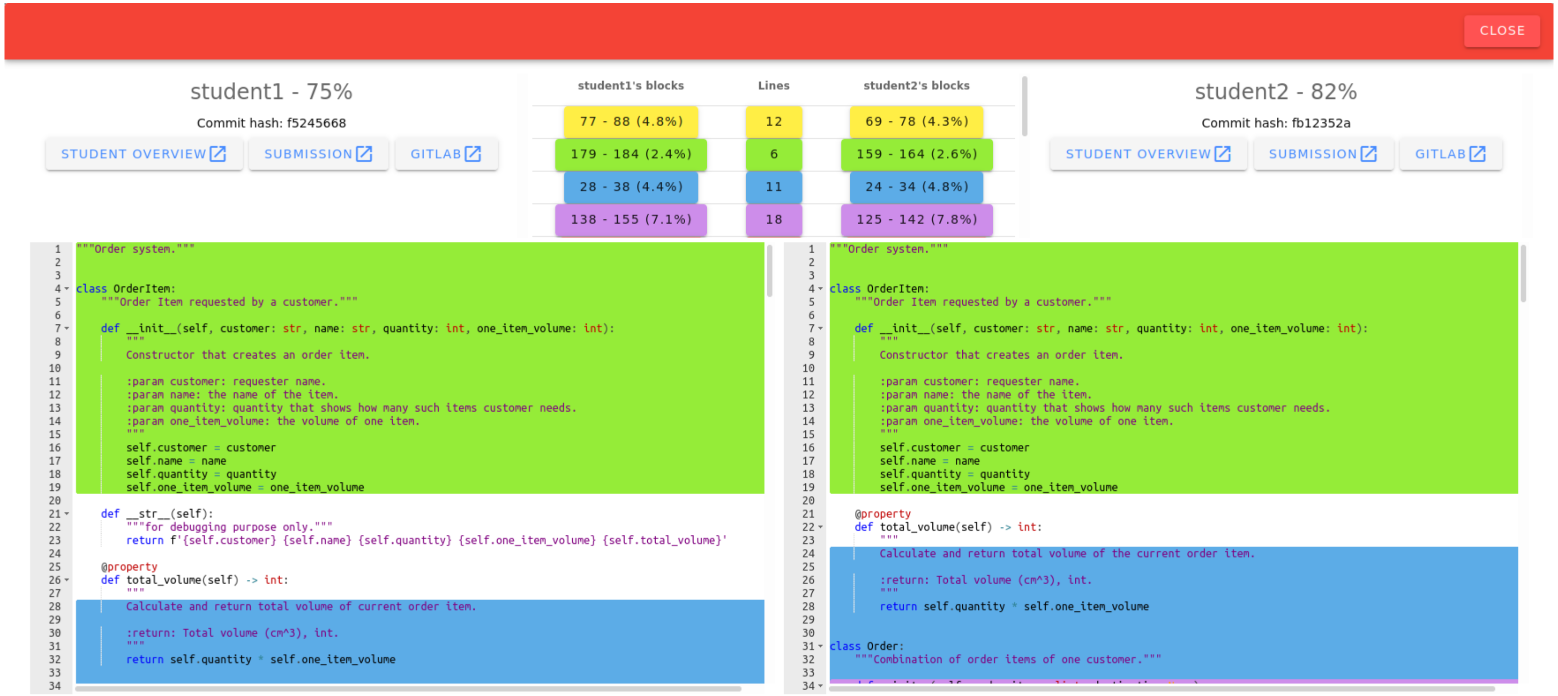
Assignment statistics

Charon	Assignment status ↓	Max lines matched	Max percentage (%)	Max other percentage (%)	New amount	Acceptable amount	Plagiarism amount
blackjack	Plagiarism	634	59	65	0	5	1
binary	Plagiarism	35	27	54	0	4	1
cipher	Plagiarism	33	37	36	1	4	2
idcode	Plagiarism	120	33	42	4	0	3
minesweeper	Plagiarism	176	30	55	4	2	1
order	Plagiarism	173	69	75	3	2	2
pies	Plagiarism	140	74	75	3	0	4
recursive_calories	Plagiarism	86	43	44	3	2	2
adventure	Acceptable	143	29	33	0	7	0
magic	Acceptable	42	5	4	0	1	0

Appendix 6 - Student plagiarism progress for each Charon



Appendix 7 – Match comparison section



student1 - 75%

Commit hash: f5245668

student1's blocks	Lines	student2's blocks
77 - 88 (4.8%)	12	69 - 78 (4.3%)
179 - 184 (2.4%)	6	159 - 164 (2.6%)
28 - 38 (4.4%)	11	24 - 34 (4.8%)
138 - 155 (7.1%)	18	125 - 142 (7.8%)

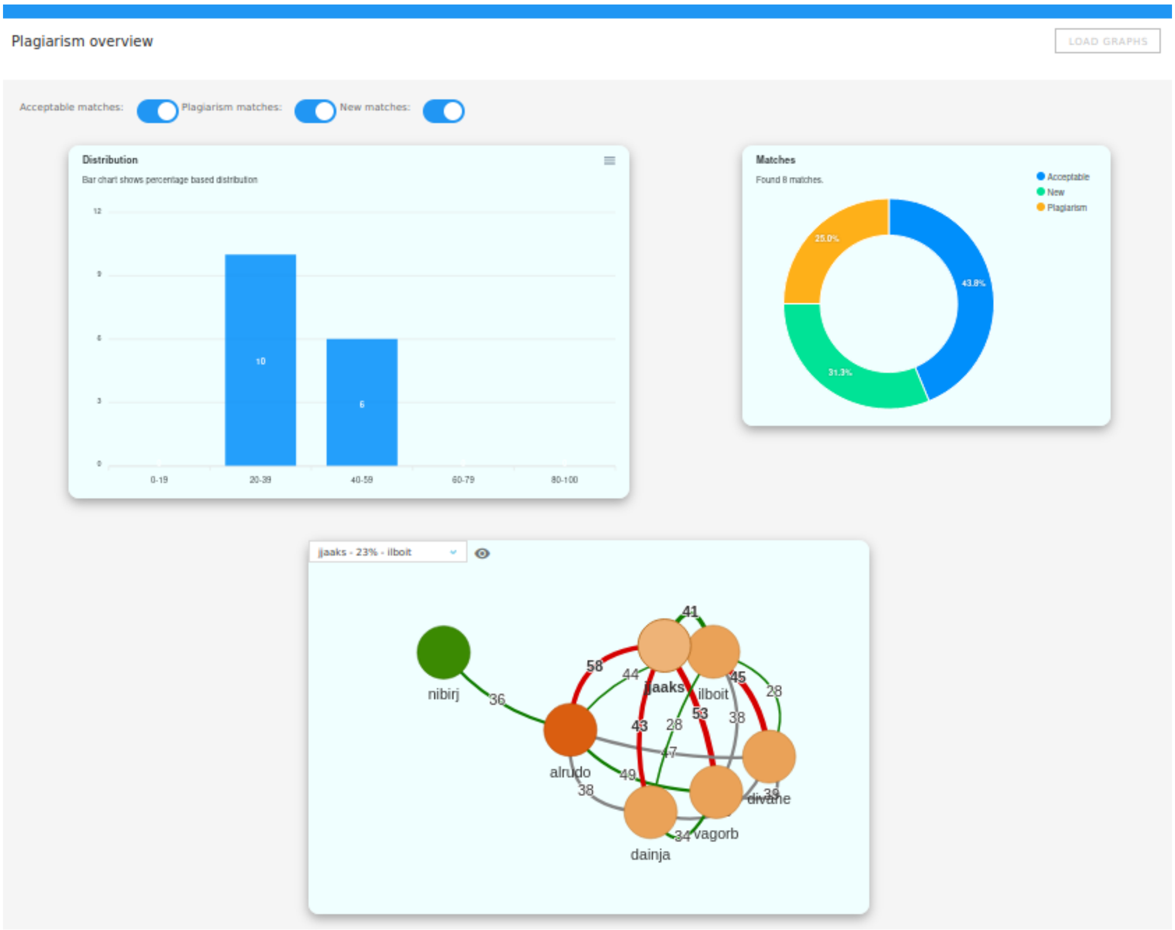
student2 - 82%

Commit hash: fb12352a

```
"""Order system."""


class OrderItem:
    """Order Item requested by a customer."""

    def __init__(self, customer: str, name: str, quantity: int, one_item_volume: int):
        """
        Constructor that creates an order item.

        :param customer: requester name.
        :param name: the name of the item.
        :param quantity: quantity that shows how many such items customer needs.
        :param one_item_volume: the volume of one item.
        """
        self.customer = customer
        self.name = name
        self.quantity = quantity
        self.one_item_volume = one_item_volume

    def __str__(self):
        """For debugging purpose only."""
        return f'{self.customer} {self.name} {self.quantity} {self.one_item_volume} {self.total_volume}'

    @property
    def total_volume(self) -> int:
        """
        Calculate and return total volume of current order item.

        :return: Total volume (cm^3), int.
        """
        return self.quantity * self.one_item_volume

```

```
"""Order system."""


class OrderItem:
    """Order Item requested by a customer."""

    def __init__(self, customer: str, name: str, quantity: int, one_item_volume: int):
        """
        Constructor that creates an order item.

        :param customer: requester name.
        :param name: the name of the item.
        :param quantity: quantity that shows how many such items customer needs.
        :param one_item_volume: the volume of one item.
        """
        self.customer = customer
        self.name = name
        self.quantity = quantity
        self.one_item_volume = one_item_volume

    @property
    def total_volume(self) -> int:
        """
        Calculate and return total volume of the current order item.

        :return: Total volume (cm^3), int.
        """
        return self.quantity * self.one_item_volume


class Order:
    """Combination of order items of one customer."""

```

Appendix 8 – Graphs component section



Plagiarism overview

Acceptable matches: ● Plagiarism matches: ● New matches: ●

Distribution

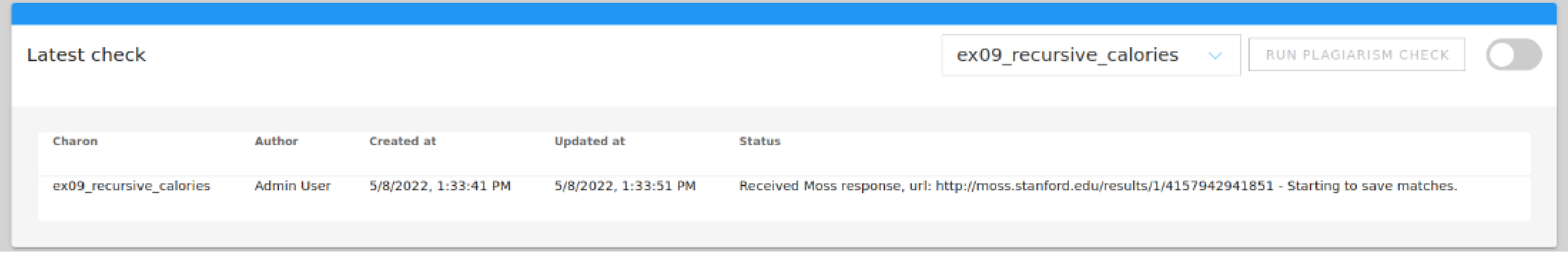
Bar chart shows percentage based distribution

Matches

Found 8 matches.

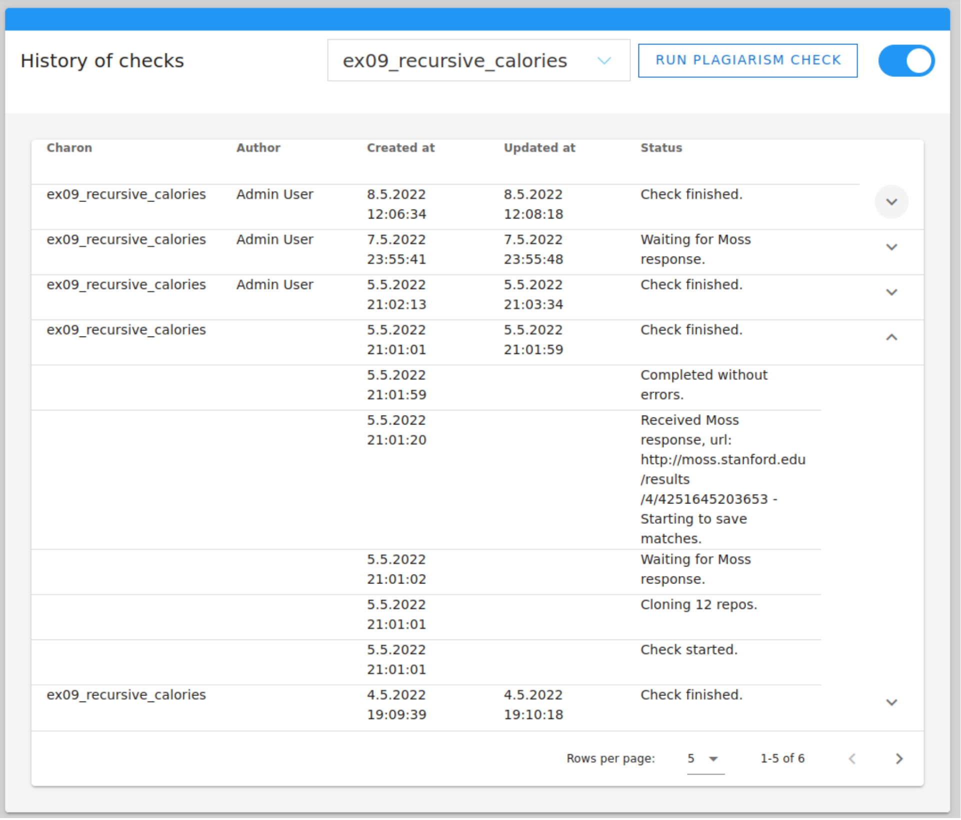
jjaaks - 23% - ilboit

Appendix 9 – Latest check section



Charon	Author	Created at	Updated at	Status
ex09_recursive_calories	Admin User	5/8/2022, 1:33:41 PM	5/8/2022, 1:33:51 PM	Received Moss response, url: http://moss.stanford.edu/results/1/4157942941851 - Starting to save matches.

Appendix 10 – History of checks section



Charon	Author	Created at	Updated at	Status
ex09_recursive_calories	Admin User	8.5.2022 12:06:34	8.5.2022 12:08:18	Check finished.
ex09_recursive_calories	Admin User	7.5.2022 23:55:41	7.5.2022 23:55:48	Waiting for Moss response.
ex09_recursive_calories	Admin User	5.5.2022 21:02:13	5.5.2022 21:03:34	Check finished.
ex09_recursive_calories		5.5.2022 21:01:01	5.5.2022 21:01:59	Check finished.
		5.5.2022 21:01:59		Completed without errors.
		5.5.2022 21:01:20		Received Moss response, url: http://moss.stanford.edu/results/4/4251645203653 - Starting to save matches.
		5.5.2022 21:01:02		Waiting for Moss response.
		5.5.2022 21:01:01		Cloning 12 repos.
		5.5.2022 21:01:01		Check started.
ex09_recursive_calories		4.5.2022 19:09:39	4.5.2022 19:10:18	Check finished.

Rows per page: 5 1-5 of 6